PAPER

# Deep Reinforcement Learning with Sarsa and Q-Learning: A Hybrid Approach

**Zhi-xiong XU**[†a)], **Lei CAO**[†b)], **Xi-liang CHEN**[†c)], ***Nonmembers***, **Chen-xi LI**[†d)], ***Student Member***, **Yong-liang ZHANG**[†e)], ***and*** **Jun LAI**[†f)], ***Nonmembers***

**SUMMARY** The commonly used Deep Q Networks is known to overestimate action values under certain conditions. It's also proved that overestimations do harm to performance, which might cause instability and divergence of learning. In this paper, we present the Deep Sarsa and Q Networks (DSQN) algorithm, which can considered as an enhancement to the Deep Q Networks algorithm. First, DSQN algorithm takes advantage of the experience replay and target network techniques in Deep Q Networks to improve the stability of neural networks. Second, double estimator is utilized for Q-learning to reduce overestimations. Especially, we introduce Sarsa learning to Deep Q Networks for removing overestimations further. Finally, DSQN algorithm is evaluated on cart-pole balancing, mountain car and lunarlander control task from the OpenAI Gym. The empirical evaluation results show that the proposed method leads to reduced overestimations, more stable learning process and improved performance.

***key words:*** *deep reinforcement learning, Deep Q Network, overestimation, double estimator, Sarsa*

## 1. Introduction

The reinforcement learning [1] mainly aims to learn a good policy for Markov decision problem, by maximizing a cumulative future reward. Q-learning [2] is one of the most popular reinforcement learning algorithms, which is also called off-policy TD learning. However, it sometimes overestimates the action values because Q-learning uses the maximum action value as an approximation for the maximum expected action value, which might lead to the poor performance.

Previously, it was proved by Thrun and Schwartz [3] that in Q-learning if the action values contain random errors uniformly distributed in an interval $[-\varepsilon, \varepsilon]$, then each target will be overestimated up to $\gamma \cdot \varepsilon \cdot \frac{m-1}{m+1}$, where $m$ represents the number of actions, $\gamma$ represents discount factor in reinforcement learning. Besides, they also noted that the overestimation in Q-learning even lead to sub-optimal policies when using function approximation. And Van Hasselt [4] proposed Double Q-learning, which alternately approximates the maximum action value by introducing a double estimator approach to determine the value of the next state, which was shown to have a better performance than Q-learning in a roulette game and a maze problem. Recently, Minh [5], [6] combined Q-learning algorithm with a flexible deep neural network into deep Q-learning algorithm, which we called DQN algorithm, it was tested on many Atari games [7], and the experimental result showed that DQN algorithm has reached human-level performance on almost all games. However, van Hasselt [8] proved that even in deterministic environment like Atari games, which prevents the harmful effects of noise, DQN sometimes substantially overestimates the values of the actions, and proposed a Double DQN algorithm (DDQN) to reduce overestimations of DQN, DDQN indeed obtains more accurate value estimates and lead to higher scores on several Atari games. Besides, Anschel [9] presented the AVERAGED TARGET DQN (ADQN) algorithm, using a weighted average over past learned networks to reduce overestimations, which leads to more stable learning process and improved performance on a toy Gridworld problem, as well as on several of the Atari games from the Arcade Learning Environment.

The reasons why the overestimations is introduced even in deterministic problem lie in that Q-learning is off-policy algorithm, which uses the maximum action value as an approximation for the maximum expected action value. As a consequence, we introduce an on-policy reinforcement learning—Sarsa learning, which successfully avoids overestimations problem. Rummery and Niranjan [10] proposed a model-based algorithm called the improved Q-learning algorithm, later called Sarsa learning algorithm. It is a kind of online strategy Q-learning algorithm. Compared with Q-learning, which uses the maximum action-state value for iteration, whereas Sarsa learning use the actual action-state value for iteration, and strictly updates the value function according to the experience gained by executing a strategy [11], [12]. It is validated that Sarsa learning algorithm is better than Q-learning algorithm in some of the applications of learning control problem.

We use the architecture of double Q-learning to construct a new algorithm call DSQN (Deep Sarsa and Q Networks), which combines DQN and Sarsa learning as a hybrid approach. First, the proposed DSQN algorithm does not simply add DQN and Sarsa together, but rather a mixture of probabilities. Second, the probability setting is

Manuscript received September 4, 2017.
Manuscript revised March 1, 2018.
Manuscript publicized May 22, 2018.

[†]The authors are with Institute of Command Information System, PLA University of Science and Technology, Nanjing, 100190 China.
a) E-mail: xu.nj@foxmail.com
b) E-mail: caolei.nj@foxmail.com
c) E-mail: 383618393@qq.com
d) E-mail: streamorning@qq.com
e) E-mail: zhangylnj@126.com
f) E-mail: 2568754202@qq.com

DOI: 10.1587/transinf.2017EDP7278

not simply set manually, but automatically set by utilizing the characteristics of the convergence of two algorithms and the actual number of training on different tasks. Finally, this paper is a combination of two different method for approximating Q values in DQN and Sarsa, but applies a common deep neural network, and the error return to update the same network, not two separate systems.

This algorithm not only yields more accurate value estimates, but also accelerate learning process, because in very first episode the agents' knowledge is little, Q-learning is "looking into the future" during Q-value assignment and searching the best possible path, whereas Sarsa learning is assigning Q-values based on much more immediate information, what is the reward from what agent just did. As a consequence, Q-learning agent will find a sub-optimal policy faster, but will slowly changes when it towards the optimal policy, while Sarsa learning will find a better policy after a few steps more, when agent has found a little knowledge about environment.

Section 2 reviews existing deep reinforcement learning algorithms. Section 3 introduces a new algorithm called DSQN to solve overestimations problem, meanwhile, accelerate learning process. Section 4 carries out experiments to prove the correctness and rationality of DSQN algorithm. Section 5 summaries the results and presents conclusions and future directions.

## 2. Deep Reinforcement Learning

Assuming that the task is a Markov Decision Process specified by the tuple $(S, A, T, \gamma, D, R)$. $S$ and $A$ are respectively the sets of states and actions. is a transition function, which returns the probability of state transition. $\gamma$, the discount factor, exponentially decreases the value of a future reward. $D$ is the distribution of start states. $R$ is a reward function, which means the reward given by environment. As shown in Fig. 1.

To solve Markov Decision Process (MDP) problem, reinforcement learning agent estimates for the optimal value of each action in a state, which is defined as the expected sum of future rewards when agent follows the optimal policy and takes a certain action. The policy $\pi$ defined in reinforcement learning outputs a distribution over actions given a state. Under a given policy $\pi$ and time $t$, the Q-value of a certain action $a_t$ in a state $s_t$ is

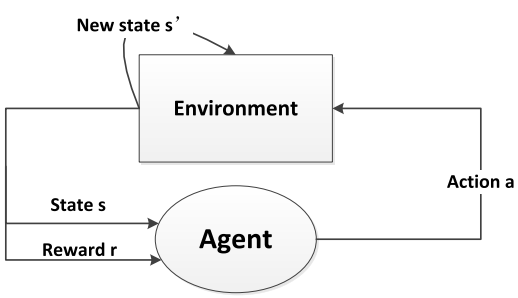


**Fig. 1** The model of reinforcement learning

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

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 $t$. We assume the optimal value is $Q_*(s, a) = \max_\pi Q_\pi(s, a)$, by selecting the highest-valued action in every state, agent will easily obtain an optimal policy.

### 2.1 Deep Q Networks

Estimates for the optimal action values can be learned using Q-learning [2], a form of temporal difference learning [13]. In classical Q-learning [14], the update rule for Q-value is

$$Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha(r(s, a) + \gamma \max_{a'} Q_t(s', a') - Q_t(s, a)) \quad (2)$$

where state $s'$ is the resulting state after the agent applies action $a$ in the state $s$, the function $r(s, a)$ means that the reward given by the environment, $\gamma$ is the discount factor and $\alpha$ is a time varying learning rate.

The Q-learning update rule can be directly implemented in a multi-layered neural network, the table of state action pairs is replaced by parameterized neural networks, as a result, the original TD error is modified as a new loss function [15] (Eq. (3)), and the gradient descent takes the form of Eq. (4):

$$L(\theta_t) = E_{(s,a)\sim D}[(y_t^Q - Q(s, a; \theta_t))^2] \quad (3)$$

$$\theta_{t+1} \leftarrow \theta_t + \eta E_{(s,a)\sim D}[(y_t^Q - Q(s, a; \theta_t))\nabla_\theta Q(s, a; \theta_t)] \quad (4)$$

Where $\eta$ is a scalar learning rate, $y_t^Q$ is the target value with $\theta^- = \theta_t$, and $D$ is an experience buffer which contains one tuple of experience $D = \{s_t, a_t\}$.

The DQN algorithm proposed by Mnih et al introduces two important technologies to improve the learning performance, the one is target network, and its parameters $\theta^-$ will be copied from online networks, $\theta_t^- = \theta_t$, only every $\tau$ steps. Then

$$Y_t^{DQN} \equiv r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t^-) \quad (5)$$

The other one is experience replay [16], agents' experience are 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.

### 2.2 Double Q-Learning

In classical Q-learning and DQN, the algorithm itself always uses the maximum values both to select and to evaluate an action. This might result in overoptimistic value estimates. To avoid overestimations, Hasselt et al decouple the selection from the evaluation, propose Double Q-learning [17].

In Double Q-learning, double estimator is applied to Q-learning to approximate the maximum expected action

value. By assigning experience randomly to train one of two value functions, the algorithm trains two sets of weights, $\theta$ and $\theta'$. When updating value function, the one set of weights is used to choose action, and the other one is used to evaluate the action. The original target in Q-learning is

$$Y_t^Q = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t) \quad (6)$$

The target in Double Q-learning can be rewritten as:

$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t') \quad (7)$$

Double Q-learning stores two value functions: $Q^A$ and $Q^B$, each value function is updated with a value form the other value function for the next state.

### 2.3 Double Deep Q Networks

By combining Double Q-learning and DQN, Hasselt et al proposed a new algorithm called Double DQN, and use online network to evaluate the greedy policy, while using target network to estimate its value, since the target network in the DQN architecture provides a natural candidate for the second value function, without introducing additional networks. With the idea of Double Q-learning, the resulting algorithm's target is replaced by

$$Y_t^{DoubleDQN} \equiv r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t), \theta_t^-) \quad (8)$$

While its update is same as for DQN.

Compare to Double DQN, one of the weights of two network $\theta_t'$ are replaced with the weights of the target network $\theta_t^-$, whose update stays unchanged from DQN, and remains a periodic copy of the online network.

The goal of the structure of Double DQN is to make full use of the benefit of the Double Q-learning, while taking perfectly advantage of the feature of the DQN.

## 3. Deep Sarsa and Q Networks

The idea of Deep Sarsa and Q Network (DSQN) algorithm is to reduce overestimations by combining Q-learning with Sarsa algorithm on the basis of the structure of Double DQN, which helps minimize the impact of overestimations, the reasons why we keep Q-learning lies in that in the initial learning stage, agents know little about environment, and the max operator in Q-learning helps agent to learn a sub-optimal policy quickly. With the progress of the learning process, we use Sarsa algorithm with greater probability, it's advantageous for agent to change the sub-optimal policy towards optimal policy and break the local optimum, which we called it "the plight of learning".

### 3.1 Sarsa Learning

In Sarsa learning, agent starts in state $s_t$, execute action $a_t$, then get reward $r_t$, so agent transfers to the state $s_{t+1}$, before updating $Q(s_t, a_t)$, agent have carried out another action $a_{t+1}$ and got reward $r_{t+1}$. In contrast, in Q-learning, agent starts in state $s_t$, execute action $a_t$, then get reward $r_t$, then check in state $s_{t+1}$ and take action $a$, which $a = \arg\max_a Q(s_{t+1}, a)$, then update $Q(s_t, a_t)$. So the difference between this two algorithm is the way the future rewards are found. In the Q-learning, the future rewards is the maximum expected value of which agent is most likely to take the most advantageous action, whereas in SARSA, it is the actual reward for executing real action.

**Table 1** Sarsa algorithm

| Algorithm 1: Sarsa learning |
|---|
| Initialize $Q(s,a)$ arbitrary |
| Repeat (for each episode): |
| Initialize $s$ |
| Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon-greedy$) |
| Repeat (for each step of episode): |
| Take action $a$, observe $r$, $s'$ |
| Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon-greedy$) |
| $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ |
| $s \leftarrow s'; a \leftarrow a'$ |
| Until $s$ is terminal |
| Until all $Q(s,a)$ is convergent |

As in all on-policy methods, Sarsa continually estimate $Q(s,a)$ for the behavior policy $\pi$, and at the same time change $\pi$ toward greediness with respect to $Q(s,a)$ [18]. The general form of the Sarsa algorithm is given in Table 1.

Besides, The DSN (Deep Sarsa Networks) algorithm [19], [20], which combines on-policy Sarsa and deep neural network, is also proved have better performance in some aspects than DQN.

### 3.2 Deep Sarsa and Q Networks Algorithm

The full algorithm for training DSQN is presented in Table 2. The agent selects and executes actions according to the $\varepsilon$-*greedy* policy based on $Q$ value. The DSQN algorithm modifies standard Q-learning in three ways to improve the stability of our algorithm and reduce overestimations as far as possible.

First, DSQN uses two technologies known as experience replay [21] and target network [22], [23]. Experience replay technique store the tuples of experience at each time-step, in the form of $\{s_t, a_t, s_{t+1}, r_t\}$, in a data set $\Omega$, 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, we use a separate network as target network for generating the target in the Q-learning update.

**Table 2** DSQN algorithm

Algorithm 2: Deep Sarsa and Q Networks

1. Initialize reply memory $\Omega$ to capacity $N$
2. Initialize action-value function $Q$ with random weights $\theta$
3. Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$
4. **For** episode=1, $M$ **do**
4. Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$
5. **For** t=1,T **do**
6. With probability $\varepsilon$ select a random action $a_t$

   Otherwise select $a_t=\arg\max_a Q(\phi(s_t),a;\theta)$
7. Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
8. Set $s_{t+1}=s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$
9. Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in $\Omega$
10. Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from $\Omega$
11. With probability $\delta$ set

$$y_j=\begin{cases} r_j & \text{if episode stop at step j+1} \\ r_j+\gamma \hat{Q}(\phi_{j+1},\arg\max_a Q(\phi_{j+1},a;\theta_t);\theta_t^{-}) & \text{otherwise} \end{cases}$$

   otherwise
12. With probability $\varepsilon$ select a random action $a_{j+1}$

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

$$y_j=\begin{cases} r_j & \text{if episode stop at step j+1} \\ r_j+\gamma \hat{Q}(\phi_{j+1},a_{j+1},\theta_t^{-}) & \text{otherwise} \end{cases}$$

13. Perform a gradient descent step on $(y_j-Q(\phi_j,a_j;\theta_t))^2$ with respect to the network parameters $\theta_t$
14. Every $C$ steps reset, $\hat{Q}=Q$
15. **End For**
16. **End For**

Every $C$ steps, we copy the weights of online network to obtain the weights of target network. This modification helps make the neural network more stable compared to standard online Q-learning.

Second, DSQN utilizes the double estimator approach in Double Q-learning, which determines the value of the next state, to reduce overestimations. Although not fully decoupled, the target network architecture provides a natural candidate for the second value function, without having to introduce additional networks. DSQN evaluate the greedy policy according to the online network, but using the target network to estimate its value.

Finally, DSQN algorithm introduces a novel reinforcement learning, Sarsa learning, to reduce overestimations further, since Sarsa learning finds the future rewards by following certain strategies rather than taking the most advantageous action at that time. Not considering the so-called optimal and follow certain strategy sometimes is a good thing. Besides, we still adopt the Q-learning, because while meeting high-dimensional state input, we need to use Q-learning to help to obtain a sub-optimal policy.

As a consequence, The DSQN algorithm utilizes the probability $\delta$ to mix Q-learning and Sarsa learning together. More precisely, the DSQN algorithm uses Q-learning with greater probability in the initial stage and utilizes Sarsa learning more likely in the late stage.

## 4. Experiments

We choose several classic control task in OpenAI Gym [24] to evaluate the performance of the DSQN algorithm and some assumptions made in previous section. First, we will compare DSQN algorithm with DQN, DSN and DDQN on cart-pole balancing problem, which is the most commonly used control problem for RL algorithms. Next, we will show experiments on a mountain car problem where exists a locally optimal solution. Finally, we also test our algorithm on a Box2D game—LunarLander, which is a more complex task than the above classic control problems.

As to the experimental setup, four algorithms all employed 4-layer networks with 100, 100 nodes in hidden layer, the number of input neurons is the state dimension of task, the number of output neurons is the action dimension. Networks were trained using the Adaptive Moment Estimation (ADAM) optimizer [25] with momentum 0.95 and a network learning rate of $10^{-5}$. Gradients were clipped at 10 scaled as they approached parameter bounds. We set the discount factor $\gamma = 0.99$, the batch size to 32, updated the target network parameters each $C = 300$, used a replay memory of ten thousands most recent transitions, and in the $\varepsilon$-*greedy* method, we set $\varepsilon = 0.1$.

For DSQN algorithm, the probability $\delta$ in Table 2 decides whether to use Q-learning or Sarsa during learning process. The initial value of probability $\delta$ is $\delta_{ini}$. We set $\delta_{ini} = 0.99$, so the DSQN algorithm would use Q-learning with greater probability in the initial stage. For probability $\delta$, each step is decremented by $\Delta\delta$ until it is equal to $\delta_{fin}$, where $\Delta\delta = \delta_{ini} - \frac{\delta_{ini}-\delta_{fin}}{N}$, $N$ refers to the total training steps. We set $\delta_{fin} = 0.01$, so the DSQN algorithm would utilize Sarsa learning more likely in the late stage.

### 4.1 Cart-Pole Balancing

The cart-pole balancing task in dynamic and control theory has been originally described by Stephenson [26]. The observation of the cart-pole balancing consists the cart position $x$, pole angle $\theta$, the cart velocity, and the pole velocity. Besides, the action consists the horizontal force applied to the cart body. The system is controlled by applying a force of $+1$ or $-1$ to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of $+1$ is provided for every timestep when the pole remains upright. The episode ends when $|x| > 2.4$ or $|\theta| > 0.2$. We set a threshold $\Delta = 300$ for cart-pole balancing, which means if the pendulum keeps not falling over for 300 continuous timesteps, the task will be terminated automatically.

We independently executed each method 10 times

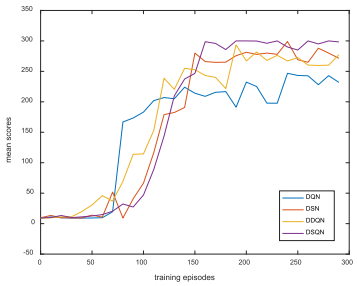


**Fig. 2** The average score of cart-pole balancing task

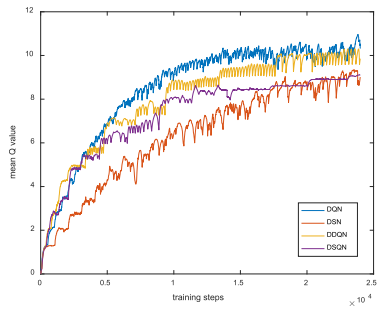


**Fig. 3** The comparison of mean Q value estimation

respectively. The learned policy will be tested 30 episodes respectively without exploration by every 10 training episodes to calculate average score and Q value of multiple runs.

Figure 2 shows that average scores of four algorithms on the cart-pole balancing task.

Figure 3 shows that mean Q value estimation over all state for four algorithms on the cart-pole balancing task, the average Q value of the four algorithm after converged are 10.41, 9.11, 10.0, 9.04 respectively. The standard deviation of Q value are 0.30, 0.21, 0.22, 0.05 respectively. From the standard deviation, we can see that DSQN has the most stable value estimate, and according to the average value, the Q value of DSQN is also the most conservative in four algorithms.

## 4.2 Mountain Car

The well-known classic mountain car control task is described by Moore [27]. A car is on a one-dimensional track, positioned between two "mountains". Our final goal is to drive up the mountain on the right; of course, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. The observation of mountain car is given by the horizontal position $x$ and the horizontal velocity $v$ of the car. In this experiment, the reward is given by $r(s,a) := -1$ for every timestep when the car remains between two "mountains", the episode ends when the car reaches a target height of 0.6.

We independently executed each method 10 times respectively. The learned policy will be tested 30 episodes

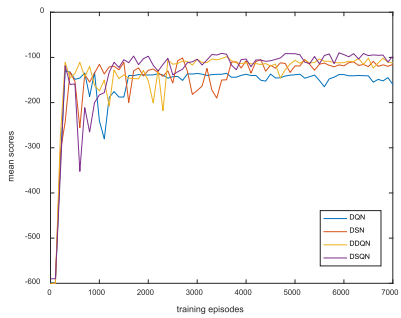


**Fig. 4** The average score of mountain car task

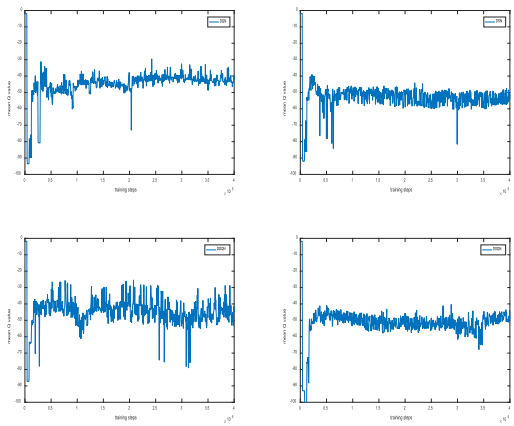

**Fig. 5** The mean Q value estimation of four algorithms

respectively without exploration by every 100 training episodes to calculate average score and Q value of multiple runs.

Figure 4 presents a visual comparison on the mountain car task.

Figure 5 shows the mean Q value estimation of four algorithm respectively. The average Q value of the four algorithm after converged are $-41.9$, $-51.2$, $-46.3$, $-55.8$ respectively. The standard deviation of Q value are 7.2, 6.6, 4.7, 2.6 respectively. The standard deviation means that DSQN has the most stable value estimate, and the Q value is also the most conservative in four algorithms.

## 4.3 LunarLander

Lunar Lander is an arcade game released by Atari, which uses a vector monitor to display vector graphics. The game is a variant on the Lunar Lander concept, which dates back to 1969. In Box2D games, Landing pad is always at coordinates $(0,0)$. Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100 to 140 points. If lander moves away from landing pad, it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional $-100$ or $+100$ points. Each leg ground contact is $+10$. Firing main engine is $-0.3$ points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and

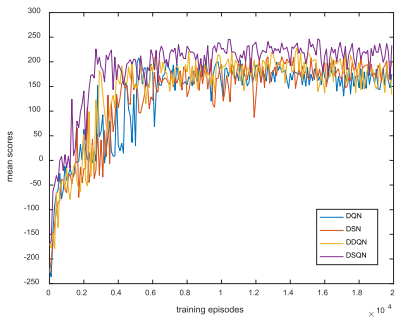


**Fig. 6** The average score of LunarLander task

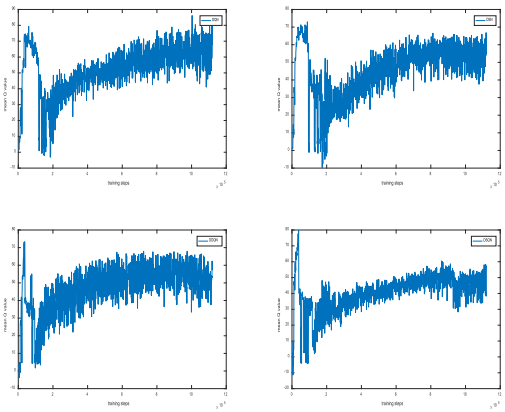

**Fig. 7** The mean Q value estimation of four algorithms

then land on its first attempt. Four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

We independently executed each method 10 times respectively. The learned policy will be tested 30 episodes respectively without exploration by every 100 training episodes to calculate average score and Q value of multiple runs.

Figure 6 presents the average score of four algorithms on LunarLander task,

Figure 7 shows the mean value estimates of different algorithms on LunarLander task respectively. The standard deviation of Q value are 8.0, 6.6, 6.5, 6.2 respectively, DSQN has the most stable value estimate. The average Q value of the four algorithm after converged are 63.8, 54.7, 54.2, 47.5 respectively, we can see that the Q value from DSQN is also the most conservative in four algorithms.

### 4.4 Comparison and Discussion

Table 3 presents the average score and standard deviation of four algorithms after converged. A detailed comparison shows that there are several games in which DSQN greatly improves upon DQN, DSN and DDQN. Noteworthy examples include CartPole-v0 (Performance has been improved by 31.6%, 6.5%, 9.2% and standard deviation has been reduced by 75.9%, 45.6%, 51.9%), MountainCar-v0 (Performance has been improved by 30.2%, 22.4%, 11.0%

**Table 3** The average score and standard deviation of DQN, DSN, DDQN and DSQN

| Task (AVG,STD) | Random | DQN | DSN | DDQN | DSQN |
|---|---|---|---|---|---|
| CartPole-v0 | 9.8 | (225.4, 20.4) | (278.6, 9.2) | (271.2, 10.4) | (296.7, 5.0) |
| Mountain Car-v0 | -600.0 | (-139.6, 7.0) | (-125.7, 6.5) | (-109.7, 6.3) | (-97.8, 5.2) |
| Lunar Lander-v2 | -242.8 | (166.2, 24.3) | (181.2, 18.9) | (179.2, 16.5) | (216.6, 15.7) |

and standard deviation has been reduced by 25.7%, 20.0%, 17.3%), and LunarLander-v2 (Performance has been improved by 30.1%, 19.3%, 20% and standard deviation has been reduced by 35.3%, 16.9%, 4.8%).

## 5. Conclusions and Future Directions

In this work, we propose the DSQN algorithm, an efficient overestimation avoidance and performance improvement technique for deep reinforcement learning. We utilize the experience replay and target network to reinforce the stability of network, besides, double estimator in double Q-learning is used to reduce overestimations, more importantly, we introduce a new reinforcement learning algorithm, Sarsa learning, not just to help DQN to solve overestimations, but also to jump out of local optimum towards global optimum. Using the classic control task in OpenAI Gym, we have shown that DSQN indeed has improvements in the learned policies over DQN, DSN and DDQN, the learning process is also more stable.

About future work, on the one hand, we will study the parameter probability $\delta$ further to see whether it can be learned as a function of the reward, or the function of the rate of change in reward. On the other hand, how to spread DSQN algorithm to continuous action space task is a critical work, which should be explored as soon as possible.

## Acknowledgments

This work was supported by the National Natural Science Fund Projects (61203192), and was also supported by the Natural Science Fund Project in Jiangsu province (BK2011124).

### References

[1] R.S. Sutton and A.G. Barto, Introduction to Reinforcement Learning, Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions, pp.90–127, 2011.
[2] C.H.C.J. Watkins, “Learning from delayed rewards,” Robotics & Autonomous Systems, vol.15, no.4, pp.233–235, 1989.
[3] S. Thrun and A. Schwartz, “Issues in using function approximation for reinforcement learning,” Proc. Fourth Connectionist Models Summer School, vol.14, no.3, pp.65–90, 1993.
[4] H.V. Hasselt, “Double Q-learning,” Advances in Neural Information

Processing Systems 23, Proceedings of A Meeting Held 6-9 Dec. 2010, Conference on Neural Information Processing Systems 2010, Vancouver, British Columbia, Canada, OAI, pp.2613–2621, 2010.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Playing Atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602v1 [cs.LG], 2013.
[6] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," vol.518, no.7540, pp.529–533, 2015.
[7] M.G. Bellemare, Y. Naddaf, J. Veness, et al., "The arcade learning environment: an evaluation platform for general agents," Journal of Artificial Intelligence Research, vol.47, no.1, pp.253–279, 2013.
[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement learning with double Q-learning," arXiv preprint arXiv:1509.06461v1 [cs.LG], 2015.
[9] O. Anschel, N. Baram, and N. Shimkin, "Deep reinforcement learning with averaged target DQN," 30th Conference on Neural Information Processing Systems, pp.78–99, 2016.
[10] G.A. Rummery and M. Niranjan, Online Q-learning using connectionist systems, Cambridge University, pp.23–86, 1994.
[11] R.S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," AAAI Spring Symposium, pp.151–155, 1991.
[12] J. Peng and R.J. Williams, "Efficient learning and planning within the Dyna framework," Adaptive Behaviore, vol.78, no.4, pp.437–549, 1993.
[13] R.S. Sutton, "Learning to predict by the methods of temporal differences[J]," Machine learning, vol.3, no.1, pp.9–44, 1988.
[14] C.J. Watkins and P. Dayan, "Q-learning," Machine Learning, vol.8, no.3-4, pp.279–292, 1992.
[15] M. Riedmiller, "Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method," Machine Learning: Ecml 2005, European Conference on Machine Learning, Porto, Portugal, pp.317–328, 2005.
[16] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," Machine Learning, vol.8, no.3, pp.293–321, 1992.
[17] H. van Hasselt, "Double Q-learning," Advances in Neural Information Processing Systems, vol.23, no.7, pp.2613–2621, 2010.
[18] R.S. Sutton and A.G. Barto, "Reinforcement learning: an introduction[J]," IEEE Trans. Neural Netw., vol.9, no.5, p.1054, 1998.
[19] D. Zhao, H. Wang, K. Shao, et al., "Deep reinforcement learning with experience replay based on SARSA," IEEE Computational Intelligence, 2017.
[20] W. Hu, "Double Sarsa and double expected Sarsa with shallow and deep learning," vol.04, no.4, pp.159–176, 2016.
[21] L.-J. Lin, "Reinforcement learning for robots using neural networks," Technical report, DTIC Document, vol.8, no.4, pp.12–45, 1993.
[22] J.L. McClelland, B.L. Mcnaughton, and R.C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory," Psychological Review, vol.102, no.3, p.419, 1995.
[23] J. O'Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, "Play it again: reactivation of waking experience and memory," Trends in Neurosciences, vol.33, no.5, pp.220–229, 2010.
[24] I. Zamora, N.G. Lopez, V.M. Vilches, et al., "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo," vol.9, no.11, pp.89–115, 2016.
[25] D.P. Kingma and J. Ba, "Adam: A Method for StochasticOptimization," Computer Science, vol.89, no.5, pp.45–145, 2014.
[26] A. Stephenson, "LXXI. On induced stability," Philosophical Magazine, vol.17, no.101, pp.765–766, 1968.
[27] A. Moore, "Efficient memory-based learning for robot control," Technical report, University of Cambridge, Computer Laboratory, vol.63, no.9, pp.62–167, 1990.
[28] G. Dejong and M.W. Spong, "Swinging up the Acrobot: an example of intelligent control," IEEE American Control Conference, pp.2158–2162, 1994.

**Zhixiong Xu** received the B.E. from PLA University of Science and Technology in 2015. He is currently studying in PLA University of Science and Technology for M.S. degree. His research interests include machine learning and intelligent decision making.

**Lei Cao** received the B.S. and M.S. degrees in China University of Science and Technology and PLA University of Science and Technology in 1987 and 1990, respectively. He is currently a professor at the PLA University of Science and Technology. His research interests include machine learning, command information system, and intelligent decision making.

**Xiliang Chen** received the B.S. and M.S. degrees in PLA University of Science and Technology in 2007 and 2009, respectively. He is currently teaching in PLA University of Science and Technology and also studying for Ph.D. degree. His research interests include reinforcement learning and intelligent decision making.

**Chenxi Li** received the B.S. and M.S. degrees in Northwestern Polytechnical University and PLA University of Science and Technology in 2011 and 2014, respectively. He is currently studying in PLA University of Science and Technology for Ph.D. degree. His research interests include reinforcement learning and intelligent decision making.

**Yongliang Zhang** received the B.S. and M.S. degrees in Nanjing artillery in 2003 and 2006, respectively. Besides, he received Ph.D. degree from Nanjing Army Command College in 2011. He is currently teaching at the PLA University of Science and Technology. His research interests include intelligent decision making, command and control.

**Jun Lai** received the B.S. and M.S. degrees in PLA University of Science and Technology in 2001 and 2005, respectively. He is currently an associate professor the PLA University of Science and Technology. His research interests include deep learning, command and control.