Modification of Q-learning to Adapt to the Randomness of Environment

Xiulian Luo
Science and Technology on Electronic Information Control Laboratory
Chengdu, China
xiaoluo6070@126.com

Youbing Gao
Science and Technology on Electronic Information Control Laboratory
Chengdu, China
gaoyoubing@hotmail.com

Shao Huang
Science and Technology on Electronic Information Control Laboratory
Chengdu, China
huangshaolee@163.com

Yaodong Zhao
Science and Technology on Electronic Information Control Laboratory
Chengdu, China
zhyd0921@163.com

Shengmiao Zhang
Science and Technology on Electronic Information Control Laboratory
Chengdu, China
172188131@qq.com

***Abstract*—Q-learning is a typical model-free algorithm in reinforcement learning to achieve a goal by interacting with an uncertain environment. However, conventional Q-learning cannot reach convergence and even learns bad policies when the state transition and the immediate reward of the environment are randomly distributed. This paper gives a modification of the Q-learning algorithm by exploring a Monte Carlo method to settle the above problems. Furthermore, simulation experiments are performed to validate the modified Q-learning algorithm.**

***Keywords*—Reinforcement learning, Q-learning, Convergence, Monte Carlo**

I. Introduction

Recently, reinforcement learning (RL) has been widely investigated since its tremendous success in the application of AlphaGo [1]. Then, a remarkable variety of RL algorithms have been widely re-explored in almost every field involving interaction and sequential policy, such as intelligence drive [2][3], computer games [4], robotics [5], and recommendation systems [6]. These algorithms can be divided into value function based approaches and policy search approaches. The former can then be split mainly into three classes [5]: (i) dynamic programming-based optimal control approaches such as policy iteration or value iteration, (ii) rollout-based Monte Carlo methods and (iii) temporal difference methods such as TD(λ) (Temporal Difference learning), Q-learning, and SARSA (State-Action-Reward-State-Action). For the latter, the most popular one is the gradient-based method. Additionally, to deal with the curse of dimensionality and the problem of sparse reward, the value based and policy search approaches are combined with deep learning in DQN (Deep Q-learning), Actor-Critic, A3C (Asynchronous Advantage Actor-Critic), and DDPG (Deep Deterministic Policy Gradient) algorithms [7][8].

This paper focuses on the Q-learning algorithm [9]. The influence of the environment randomness, mainly reflected in two aspects: the state transition and the immediate reward for the same state transition, is not considered in conventional Q-learning algorithm. In real applications, however, the environment randomness is often encountered. For instance, after being watered by an agent, the flower can become "healthy" or "dead" with probabilities. Moreover, the immediate reward is randomly distributed by taking into account the time that the state transition spends, as the flower can change from "watery" to "healthy" in two days this time but three days next time for the same action. Note that the agent requires no prior knowledge of the environment before the training. Instead, it observes the next state and the immediate reward after an action from the trial. As a result, the observed values change with different visits. The analysis of the above problem is barely found in literatures, as the interacted environment is usually considered to be deterministic. This paper analyzes the influence of the environment randomness on its performance, and then proposes a modification to improve the performance of the Q-learning algorithm.

This paper is organized as follows. In Section II, the classical Q-learning algorithm is reviewed and its shortcomings with respect to the environment randomness are analyzed. In Section III, a modification is proposed. Then, simulation experiments are presented in Section IV to evaluate the modification. Finally, the conclusions are drawn in Section V.

II. The Principle of Q-learning

Q-learning is derived from value iteration method of dynamic programming, which takes full advantage of the Markov decision process (MDP). The action-value function, also called Q-value, is updated after every step of the policy to improve the learning efficiency in comparison with that after a whole episode. The n th update of the action-value function $Q(s,a)$ at current state s with action a is defined by [9]:

$$Q_n(s,a) = Q_{n-1}(s,a) + \alpha\left[R_n(s,a) - Q_{n-1}(s,a)\right], \qquad (1)$$

where constant α is the learning rate with $0 \le \alpha \le 1$. $R_n(s,a)$ can be regarded as the n th sampled value for the state-action pair (s,a). According to the Bellman optimality equation [10], $R_n(s,a)$ can be represented by the immediate reward $r(s' \mid s,a)$ and the Q-value estimated at next state s':

$$R_n(s,a) = r(s' \mid s,a) + \gamma \max_{a'} Q_{n-1}(s',a'), \qquad (2)$$

where r and s' are observed from the environment, and γ denotes the discount factor with $0 \le \gamma \le 1$.

It can be shown that an optimal policy $\pi^*(s)$ can be derived by always picking the action in the current state s with highest Q-value :

$$\pi^*(s) = \arg\max_a Q(s,a) \tag{3}$$

During the iteration, (1) can be rewritten as :

$$\begin{aligned} Q_n(s,a) &= \alpha R_n(s,a) + (1-\alpha) Q_{n-1}(s,a) \\ &= \alpha R_n(s,a) + (1-\alpha)\left[\alpha R_{n-1}(s,a) + (1-\alpha) Q_{n-2}(s,a)\right] \\ &= (1-\alpha)^n Q_0(s,a) + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i(s,a) \end{aligned} \tag{4}$$

From (4), we can derive that the weight given to R_i decreases as $n-i$ increases, i.e. the n th sample $R_n(s,a)$ has a greater contribution to $Q_n(s,a)$ than others. However, $R_n(s,a)$ will change in different visit of (s,a) even though the iteration is close to convergence. This is because the state transition is distributed by probability, thus producing different immediate reward $r(s,a)$ and different $Q_{n-1}(s',a')$ for different next state s' .

The above derivation indicates that the convergence is not assured with a constant learning rate because of the weighted average in (4).

Moreover, in literatures $r(s' \mid s,a)$ for the same next state s' is supposed to be invariable throughout the training. Nevertheless, we may run across the situation that $r(s' \mid s,a)$ changes in different visit when the time taken by the state transition is considered to learn more sophisticated policies. For instance, in the news recommendation system, the reward for the same kind of news changes when the experiment person opens the news in 2 seconds this time but 3 seconds next time. In these situations, it is hard to reach convergence to use the observed immediate reward in a single visit to update the Q function, because the reward differs in different iterations.

III. MODIFICATION OF Q-LEARNING

In conventional algorithm, convergence cannot be guaranteed because of two aspects of randomness: (i) state transition, (ii) reward for the same transition. To solve the problem of the randomness, we employ the Monte Carlo method to get the expected Q-value by averaging the observed sample returns. Then, (1) can be rewritten as :

$$\begin{aligned} Q_n(s,a) &= \frac{1}{n}\sum_{i=1}^{n} R_i(s,a) \\ &= Q_{n-1}(s,a) + \frac{1}{n}\left[R_n(s,a) - Q_{n-1}(s,a)\right] \end{aligned} \tag{5}$$

with n the number of visits of (s,a), and the learning rate becomes:

$$\alpha(n) = 1/n . \tag{6}$$

A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1[8]:

$$\sum_{n=1}^{\infty} \alpha(n) = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha^2(n) < \infty \tag{7}$$

The convergence conditions in (7) are met for $\alpha(n) = 1/n$ but not for $\alpha(n) = \alpha$. Therefore, the modification of learning rate by substituting $1/n$ for constant α can deal with the randomness of the state transition.

Furthermore, to handle the problem that $r(s' \mid s,a)$ varies with iterations, the first item in (2) is replaced by the sample average $\overline{r_{n'}}(s' \mid s,a)$:

$$\overline{r_{n'}}(s' \mid s,a) = \frac{1}{n'}\sum_{i=1}^{n'} r_i = \frac{1}{n'}\left[(n'-1)\cdot\overline{r_{n-1}}(s' \mid s,a) + r_{n'}\right] \tag{8}$$

where r_i is the i th observed sample of $r(s' \mid s,a)$, and n' denotes the number of visits of (s,a,s'). When n approaches infinity, the state-transition probability can also be learned from the n trials, which is:

$$p(s' \mid s,a) = \frac{n'}{n} \tag{9}$$

This probability is employed in Dyna [11] to learning the environment model to reduce the cost of interacting with the real environment.

The key of the modification is to employ statistical average to reduce the influence of environment randomness on the algorithm performance. However, the additional cost of the memory is required to store n for every (s,a), n' and $\overline{r_{n'}}$ for every (s,a,s'). This is acceptable when the number of dimensions of state-action space is not very huge.

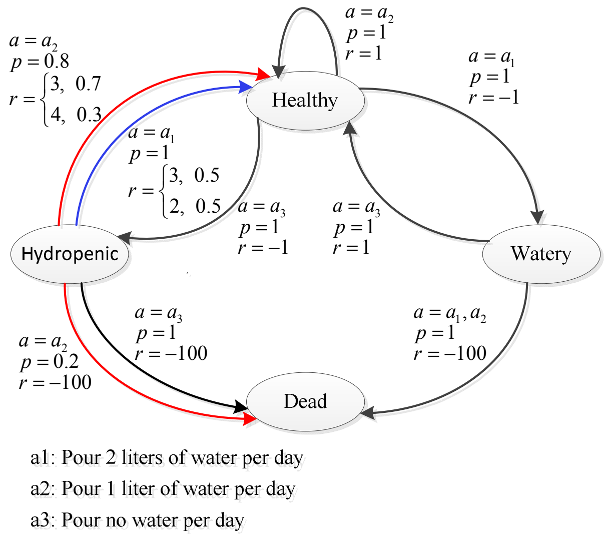
Healthy
Hydropenic
Watery
Dead
$a=a_2$, $p=1$, $r=1$
$a=a_1$, $p=1$, $r=-1$
$a=a_2$, $p=0.8$, $r=\begin{cases}3, & 0.7\\4, & 0.3\end{cases}$
$a=a_1$, $p=1$, $r=\begin{cases}3, & 0.5\\2, & 0.5\end{cases}$
$a=a_3$, $p=1$, $r=-1$
$a=a_3$, $p=1$, $r=1$
$a=a_2$, $p=0.2$, $r=-100$
$a=a_3$, $p=1$, $r=-100$
$a=a_1,a_2$, $p=1$, $r=-100$
a1: Pour 2 liters of water per day
a2: Pour 1 liter of water per day
a3: Pour no water per day

Fig. 1. MDP of watering flowers. The arrows represents the state transition, while a , p and r denotes the action, state-transition probability and immediate reward, respectively. Greater r means the flowers get healthy with less time.

IV. Simulation Experiments

To evaluate the performance of the modified algorithm, the example that an agent learns how to water the flowers is given in this section.

The environment model is illustrated in Fig.1, where the time that the state transition takes is considered for the agent to learn more sophisticated actions. As the time can be randomly distributed, the immediate reward is undetermined. For simplicity, only in "Hydropenic" state, the state-transition probability is not "1" or "0" and the reward is distributed at two values with a certain probability. Note that the agent cannot get any information from this figure. Instead, it observes the next state and the immediate reward in every interaction. "Dead" is a terminal state and the goal is to keep the flower undying for 50 days.

After 1000 episodes, the Q-value tables for conventional and modified algorithms are given in Table 1. We can see that in "Hydropenic" state, a_2 will be chosen for the conventional algorithm but a_1 for the modified one. The action a_2 can achieve greater immediate reward but greater probability to get "Dead", which means a worse policy is learned in the conventional algorithm.

TABLE I. Q-VALUE TABLES

Q values	Conventional algorithm			Modified algorithm		
	a_1	a_2	a_3	a_1	a_2	a_3
Healthy	8	10	10.2	4	6	5.3
Hydropenic	11.4	12.5	-100	7.5	-11.7	-100
Watery	-100	-100	10	-100	-100	5.6

To assess the two algorithms, the curves of the sum square error of the Q-values between two sequential episodes versus episodes are plotted in Fig.2. It indicates that the conventional algorithm cannot reach convergence but the modified one can.

Moreover, to evaluate the stability of the two algorithms, Fig.3 and Fig.4 show the values of $Q(\text{Hydropenic}, a_1)$ and $Q(\text{Hydropenic}, a_2)$ after iterating of 1000 episodes in 100 experiments for the conventional algorithm and the modified one, respectively. As the agent always picks the action with highest Q-value, it is shown in Fig.3 that different actions are chosen in different experiments for conventional algorithm.

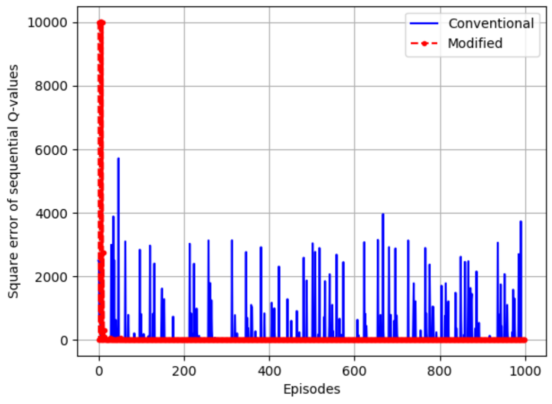

Fig. 2. Sum square error of Q-values in two sequential episodes.

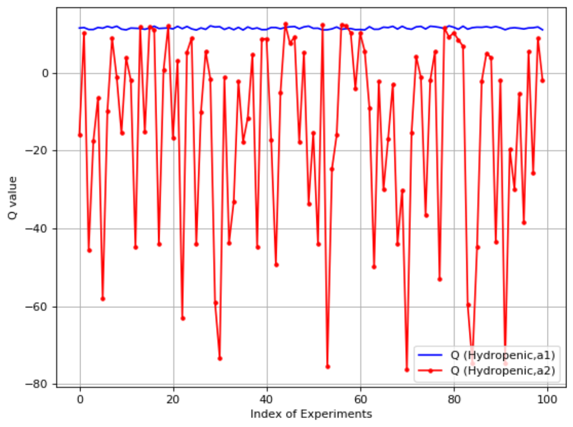

Fig. 3. Q-values after 1000 episodes in different experiments for the conventional algorithm.

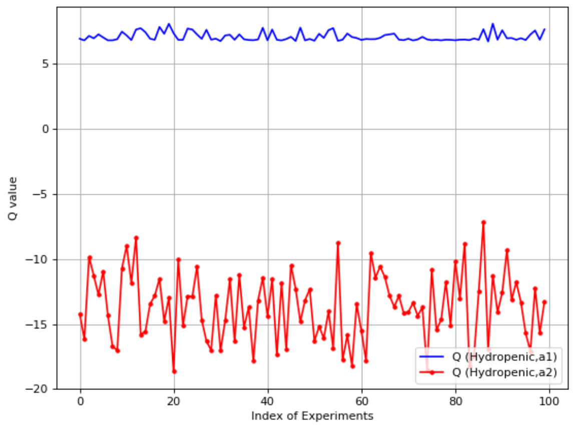

Fig. 4. Q-values after 1000 episodes in different experiments for the modified algorithm.

Fortunately, Fig.4 shows that the agent always chooses action a_1 in different experiments by employing the modified algorithm, thus indicating that the modified algorithm is much more stationary than the conventional one.

V. Conclusions

Q-learning algorithm is a typical RL algorithm to interact with an environment to achieve a goal. When the environment is not deterministic, the observed samples and sequences will affect the convergence and stability. This paper proposes a modified Q-learning algorithm to deal with the above problem. Compared with the conventional one, it can reach more stationary policy as well as assure convergence and validity for the training results at the cost of more storage.

As we know, the Q-learning algorithm is based on a lookup table, thus generally used for discrete and low-dimensional spaces. Therefore, the additional cost of storage is acceptable for most applications. For continuous or high dimensional spaces, deep reinforcement learning (DRL) is introduced, where the environment randomness will also affect the algorithm robustness. The approaches to improve the robustness for DRL algorithms are under intense investigation.

REFERENCES

[1] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, and Laurent Sifre1 et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol.529, pp. 279~292, 2016.

[2] Alex Kendall, Jeffrey Hawke, David Janz, Przemslaw Mazur, and Daniele Reda et al., "Learning to drive in a day," arXiv Preprint, arXiv:1807.00412V2, 2018.

[3] Xia Wei, Li Huiyuan, "Training Method of Automatic Driving Strategy Based on Deep Reinforcement Learning," Journal of Integration Technolology, 6(3), pp. 29-40, 2017.

夏伟, 李慧云, "基于深度强化学习的自动驾驶策略学习方法", 集成技术, 6(3), pp. 29-40, 2017.

[4] D. Silver, T. Huang, J. Schirittwieser, A. A. Rusu, and J. Veness et al., "Mastering chess and shoji by self-play with general reinforcement learning algorithms, " CoRR, vol. abs. 1712.01815, 2017.

[5] Jens Kober, Andrew Bagnell, and Jan Peters, "Reinforcement learning in robotics: a survey," International Journal of Robotics Research, vol. 32, pp. 1238-1274, 2013.

[6] X. Zhao, L. Zhang, Z. Ding, D. Yin, and Y. Zhao et al., "Deep Reinforcement learning for list-wise recommendations," arXiv Preprint, arXiv:1801.00209, 2017

[7] C. Li, L. Cao, Y. Zhang, X. Chen, and Y. zhou et al., "Knowledge-based deep reinforcement learning: a review," System Engineering and Electronics, vol 39, no. 11, pp. 2603-2613, 2017.

李晨溪，曹　雷，张永亮，陈希亮，周宇欢, 段理文, "基于知识的深度强化学习研究综述," 系统工程与电子技术, 39(11), pp. 2603-2613, 2017.

[8] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning:An Introduction, The MIT Press, London, England, 2017.

[9] Watkins C. J. C. H, P. Dayan, "Q-learning," Machine learning, vol.8, pp. 279~292, 1989.

[10] R. E. Bellman, Dynamic Programming, Peinceton University Press, Princeton, NJ, 1957.

[11] Richard S. Sutton, "Dyna, an intergrated architecture for learning, plannning, and reacting," ACM Sigart Bulletin, vol.2, no.4, pp.160~163, 1991.