# Q_LEARNING BASED ON ACTIVE BACKUP AND MEMORY MECHANISM

**YANG LIU, MAO-ZU GUO, AND HONG-XUN YAO**

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
E-MIAL: liu_yang@hit.edu.cn, maozuguo@hit.edu.cn, yhx@vilab.hit.edu.cn

**Abstract:**

**Exploration is used in Q_learning because the agent will be caught in locally optimal policies due to blind exploitation. However excessive exploration will degrade the performance of Q_learning and it is difficult to meet the trade-off between exploration and exploitation. In this paper, the active backup is introduced into Q_learning and the corresponding algorithm AB_Q_learning based on Dijkstra backup in dynamic programming is proposed. Then, the memory mechanism based MEAB_Q_learning algorithm is given for the agent to learn in completely unknown environment. The experimental results show that these two algorithms not only converge more quickly, but also solve the problem of local optimization.**

**Keywords:**

**Reinforcement learning, Q_learning, Dijkstra backup**

## 1. Introduction

Reinforcement learning is one of the more rapidly developing machine learning methods in recent years [1,2,3,4], and Q_learning [5,6] is the most important one among all reinforcement learning algorithms. Also, it is considered as a special case of temporal difference algorithm [1,7]. Researches have been made on the application of Q_learning, such as time sequence prediction [8], robot soccer competition [9], and many more.

In reinforcement learning, the agent interacts with environment. On each step of interaction, the agent perceives the current state $s$ of the environment, then selects an action $a$ to execute. As a result, the action changes the state of environment and the agent receives a scalar reinforcement signal. The target of Q_learning is to find an optimal policy, i.e. to find the map from state to action, which can maximize the cumulative reward or minimize the punishment. Although Q_learning has successively applied in some systems, it faces the problem of being caught in locally optimal policies. This paper will propose an improved Q_learning to overcome the obstacle.

According to the relation between Q_learning and dynamic programming, backup is introduced into Q_learning, which can dramatically speed up the learning. Experimental results show that the proposed algorithms achieve better performance than traditional method.

In this paper, Q_learning and dynamic planning are briefly introduced. Then active backup based Q_learning and modified algorithm with memory mechanism are proposed. Experimental results are presented and analyzed in section 4.

## 2. Q_learning and dynamic programming

### 2.1. Q_learning algorithm

Q_learning [1,5,6] is one of the algorithms for solving reinforcement learning problem, the advantage of which is that it doesn't need to know the transition relationship between states unlike dynamic programming. Q_learning is a kind of on-line and unsupervised learning, in which learning process and performance process proceed simultaneously and the agent learns the optimal policy in the interaction process.

In Q_learning, the agent updates Q value function table through perceiving state, then performs action and receives reinforcement signal. The entry of the table is composed of state and action, referred as $Q(s,a)$. As proved in [6], if all the state-action pairs are continuously visited, the Q value will converge to the optimal value. Q_learning is described as algorithm 1.

---

Algorithm 1: Q_learning algorithm

1. Initialize $Q(s,a)$ //arbitrary
2. Repeat
3. Initialize an initial state $s$
4. Select one action $a$ from $A(s)$ according to policy $\pi$ ($\varepsilon$ greedy)
5. Perform action $a$, receive the instant reward $r$, and come into next state $s'$

6. $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a} Q(s^{'}, a^{'}) - Q(s,a)]$
7. $s \leftarrow s^{'}$
8. Until $s$ is final state

---

Where $r$, $\gamma$ and $\alpha$ are the instant reward, the discount factor and the learning step respectively. $\gamma$ and $\alpha$ are both in the range of $[0,1]$.

### 2.2. Dynamic programming

Dynamic programming (DP) is a kind of solution for decision process. In 1957, R. E. Bellman et al proposed the famous principle of optimality, when they studied multistep decision process [10]. This principle converts multi-step problem into one-step problem, then solve the problem step by step. The method for this kind of problem is referred as dynamic programming.

In the methods of dynamic programming, backward algorithm is in common use. Dijkstra backup is the key process in the backward algorithm, which uses the optimal solution of successive state to search the optimal solution for its prior state. As DP requires the information about environment, the application of DP in unknown environment is restricted.

## 3. Active backup based Q_learning

From dynamic programming, it can be found that after the value function of a state is updated, it may affect its prior state in all probability. In Q_learning, if the value function of some state-action pair is updated, the value function of the prior state-action pair will be likely affected. So, we propose a new Q_learning integrated with Dijkstra backup used in dynamic programming, which is referred as active backup Q_learning (AB_Q_learning). To further adapt the new algorithm to completely unknown environment, a modified AB_Q_learning is put forward, which has the capacity of remembering the relationship of the visited state-action pairs.

### 3.1. Active backup based Q_learning

---

Algorithm 2: AB_Q_learning
1. Initialize $Q(s,a)$ //arbitrary
2. Repeat
3. Initialize an initial state $s$
4. Select one action $a$ from $A(s)$ according to policy $\pi$ ($\varepsilon$ greedy)
5. Perform action $a$, receive the instant reward $r$, and come into the next state $s^{'}$
6. $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a} Q(s^{'}, a^{'}) - Q(s,a)]$
7. If $Q(s,a) = \max_{a^{'}} Q(s,a^{'})$
8. For (every action $a$)
9. $P \leftarrow a^{-1}(s)$
10. For (every state $s^{''} \in P$)
11. $Q(s^{''},a) \leftarrow Q(s^{''},a) + \alpha[r + \gamma Q(s,a) - Q(s^{''},a)]$
12. $s \leftarrow s^{'}$
13. Until $s$ is final state

---

Algorithm 2 describes the active backup based Q_learning, where $a^{-1}(s) = \{s^{''} \mid a(s^{''}) = s\}$, which is the set of states due to applying $a$ results in state $s$. In Q_learning, when the value function of a state-action pair is updated, the max value of all state-action pairs of the next state must be used. So, the line 7 of algorithm 2 tests whether the updated value function of the current state-action pair is the maximum. If it is true, the backup process will be performed and corresponding value function of the prior pairs will be updated. Otherwise, it is unnecessary to apply backup.

In some cases, the agent can not know what is the effect of performing inverse action in a state, so it is infeasible to acquire the prior states of current state for a given action $a$. Thus the process of backup in AB_Q_learning is infeasible. To overcome the obstacle, a modified AB_Q_learning based on memory mechanism is put forward as follows.

### 3.2. AB_Q_learning with memory mechanism.

Since AB_Q_learning depends upon the degree the environment is known, we expect that the agent can store the relationship of states when it interactives with the environment. Thus, in the case of inverse action computation is infeasible, the upper proposed algorithm, AB_Q_learning, can still work. The AB_Q_learning with memory mechanism (MEAB_Q_learning) is described in algorithm 3.

---

Algorithm 3: MEAB_Q_learning
1. Initialize $Q(s,a)$ //arbitrary

2. Repeat
3. Initialize an initial state $s$
4. Select one action $a$ from $A(s)$ according to policy $\pi$ ($\varepsilon$ greedy)
5. Perform action $a$, receive the instant reward $r$, and come into next state $s'$
6. Put state-action pair $(s,a)$ into the set $PSA$ of prior state-action pairs of state $s'$
7. $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
8. If ( $Q(s,a) = \max_{a'} Q(s,a')$ )
9. For (every $(s'',a'')$ in $PSA$)
10. $Q(s'',a'') \leftarrow Q(s'',a'') + \alpha[r + \gamma Q(s,a) - Q(s'',a'')]$
11. $s \leftarrow s'$
12. Until $s$ is final state

---

In MEAB_Q_learning, a new data structure *PSA* is introduced, which is a chain for storing the state-action pairs. The agent stores the relationship of the states. So that, the modified algorithm gets rid of the shortcoming of AB_Q_learning. Thus the agent can perform backup process in completely unknown environment.

### 3.3. Analysis in theory

Lines 7-11 in AB_Q_learning and lines 8-10 in MEAB_Q_learning are the essential portions, which differentiate the two proposed algorithms from Q_learning. The backup is named as Dijkstra backup in [11]. If the value function of the current state-action pair is updated and the updated value is the maximum of the state, the process of backup is performed. This strategy will speed up the learning process. Assuming that the current state $s$ has $n$ prior states respect to the inverse of action $a$, when the backup process is performed, $n$ values of these state-action pairs will be updated; Otherwise, it will need at least $n$ steps to update these values. In fact, the backup process is much faster than performing an action [4], therefore the backup is valuable.

As to how many backups will be performed and what factor affects it, it can be found that, the backup ratio $B$ (i.e. how often the Dijkstra backup will be performed) will relate to $\varepsilon$, which is probability of randomly selecting an action not according to the $\varepsilon$-greedy policy. For a given $\varepsilon$, the ratio $B$ of backup will converge to a fixed value with the proceeding of learning. When the exploration ratio $\varepsilon$ increases, the backup ratio will decrease. The following proposition depicts the relation between the backup ratio and $\varepsilon$.

**Proposition 1:** Supposing that the probability of every state appears in environment is $p(s_i), i = 1,2,\cdots,n$, the backup ratio $B = \sum_{i=1}^{n} p(s_i) \cdot ((1-\varepsilon) + \frac{\varepsilon}{|A(s_i)|})$, where $\varepsilon$ is exploration ratio.

**Proof:** When the algorithm comes into converge state, the probability of selecting optimal action is $(1-\varepsilon) + \frac{\varepsilon}{|A(s_i)|}$, where $|A(s_i)|$ is the number of action in state $s_i$, so the backup ratio in state $s_i$ is $p(s_i) \cdot ((1-\varepsilon) + \frac{\varepsilon}{|A(s_i)|})$. Therefore, the total ratio of backup ratio is $B = \sum_{i=1}^{n} p(s_i) \cdot ((1-\varepsilon) + \frac{\varepsilon}{|A(s_i)|})$.

## 4. Experiments and analysis

### 4.1. Simulation environment

In this paper, we adopt the maze model in [11] as our simulation environment. The model is drawn in Figure 1, and the actions are depicted as follows:

One-step action: North, South, West, East. The four actions cause the agent to move one grid along the corresponding direction. If the wall is met, the agent will remain the original position.

To-wall action: North_to_Wall, South_to_Wall, West_to_Wall, East_to_Wall. These four actions make the agent walk forward until the wall is met along the corresponding direction.

These actions have two kinds of cost. The first kind of action has the cost 1, and the second kind of action has the cost 3 (the instant reward can be regarded as the negative of the costs). When the agent comes into the goal grid, marked with G, it will receive the instant reward 100.

The target of the agent is to find an optimal path from every grid to one of the goal grids, which maximize the accumulated reward along corresponding path.

### 4.2. Experimental results

Figure 2 and Figure 3 illustrate the converge speeds of the two proposed algorithm compared with the standard Q_learning. The vertical axis is the percent of the states with the optimal actions, and the horizontal axis is the number of episodes. As Figure 2 shows, the convergence speed of AB_Q_learning is much faster than the standard Q_learning. In addition, the new algorithm does not plunge

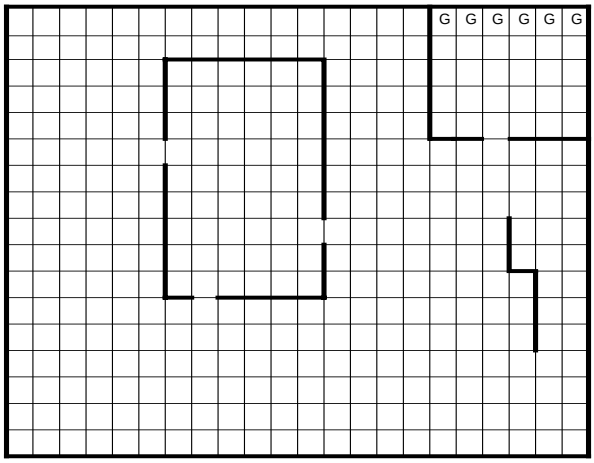


Figure 1. Maze game model

into local optimization like Q_learning. Along with the proceeding of learning, more and more states converge to the optimal action. Figure 3 depicts the convergence speed of MEAB_Q_learning. Similar to AB_Q_learning, the algorithm with memory mechanism also has faster speed of convergence than standard Q_learning.

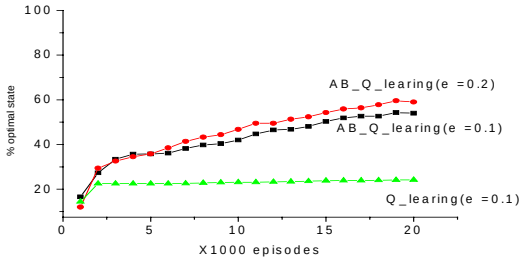


Figure 2. The convergence speed of active backup based Q_learning.

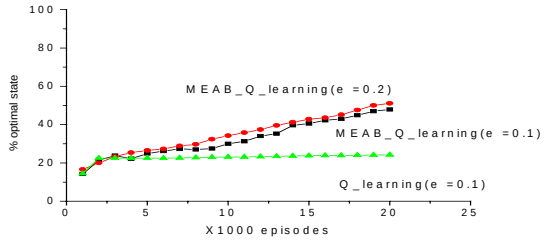


Figure 3. The convergence speed of AB_Q_learning with memory mechanism.

Figure 4 compares the speeds of the two proposed algorithm: AB_Q_learning and MEAB_Q_learning. From the figure, it can be seen that the converge speed of the former is faster than that of the latter. It results from that at the starting point of learning process, the agent in the latter algorithm knows nothing about the environment. As the learning process proceeds, the difference between them becomes trivial. This is because $PSA$ has stored more knowledge about the environment.

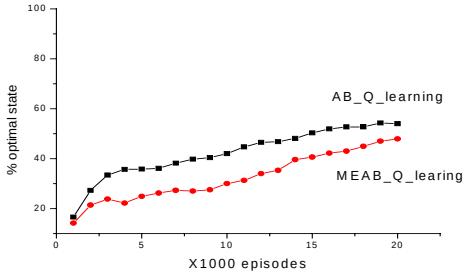


Figure 4. The convergence speed comparison between the two proposed algorithms

Figure 5 and Figure 6 illustrate the backup ratios of the two proposed algorithms with different $\varepsilon$ values. According to proposition 1, the backup ratio in maze game is

$$B^* = \sum_{i=1}^{n} p(s_i)\cdot((1-\varepsilon)+\frac{\varepsilon}{|A(s_i)|})$$
$$= \sum_{i=1}^{n} p(s_i)\cdot((1-\varepsilon)+\frac{\varepsilon}{8}) \quad .$$
$$= (\sum_{i=1}^{n} p(s_i))\cdot((1-\varepsilon)+\frac{\varepsilon}{8})$$
$$= 1-\varepsilon+\frac{\varepsilon}{8}$$

For example, when $\varepsilon=0.1$, the backup ratio is 0.9125, and when $\varepsilon=0.3$, the backup ratio is 0.7375. These are consistent with Figure 5 and Figure 6.

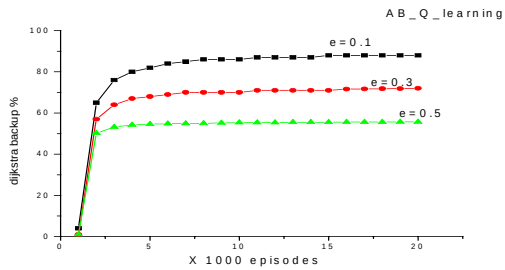


Figure 5. The backup ratio of AB_Q_learning with different $\varepsilon$ values.

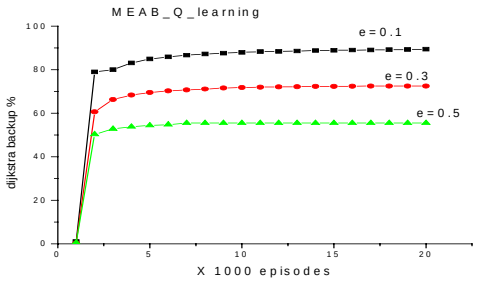


Figure 6. The backup ratio of MEAB_Q_learning with different $\varepsilon$ values.

## 5. Conclusion

An active backup based Q_learning, AB_Q_learning is proposed in this paper, in which backup process is introduced. Since partial environment knowledge is needed, a modified algorithm with memory capacity is designed. Experimental results show that the proposed algorithms have faster convergence speed, mean while the problem of plunging local optimization is well solved.

## References

[1] Richard S. Sutton, Andrew G. Barto. Reinforcement Learning An Introduction. The MIT Press. 1998

[2] L. P. Kaelbling, M. L. Littman, A. W. Moore. Reinforcement Learning. A Survey. Journal of AI Research, 4 (1996), pp. 237-285.

[3] M.Z. Guo, B. Chen, X.L. Wang, J.R. Hong. A summary on reinforcement learning. Computer Science, 25 (3) (1998), pp. 13-15 (in Chinese).

[4] T. Mitchell. Machine Learning. McGraw-Hill, 1997.

[5] Watkins C.J.C.H. Learning from delayed rewards [Ph D dissertation]. Psychology Department, Cambridge University, 1989

[6] Watkins C.J.C.H. Dayan P. Q-learning, Machine Learing,1992,8:279-292

[7] Richard S. Sutton. Learning to predit by the method of temporal differences. Machine Learning, 1988,3,9-44

[8] L. Yang, J.R. Hong, T.Y. Huang. Combining the methods of temporal di_erences with neural network for real-time modeling and prediction of time series. Chinese Journal of Computers, 19 (9) (1996), pp. 695-700 (in Chinese).

[9] M. Asada, E. Uchibe, K. Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. Artificial Intelligence, 110 (1999), pp. 275-292.

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

[11] T.G. Dietterich, N.S. Flann. Explanation-based learning and reinforcement learning: a unied view. Machine Learning, 28 (1997), pp. 169-210.