Proceeding of the IEEE
International Conference on Information and Automation
Wuyi Mountain, China, August 2018

Research on Online Reinforcement Learning Method Based on Experience-Replay

Ning Hu [1] and Zhijun Ge [1] Xuanwen Chen [1]

[1] *China Electronic Product Reliability and Environmental Testing Research Institute*
Guangzhou, China
huning@ceprei.com

Chunguang Ding [1] and Haobin Shi [2*]

[2] *School of Computer Science*
Northwestern Polytechnical University
Xi'an, China
shibaobin@nwpu.edu.cn

***Abstract* - As for standard reinforcement learning, the key is that the agent's next step is directed by the instantaneous and delayed reporting from constant interaction with the environment and trial and error learning. But it makes the convergence rate slower for actual reinforcement learning; at the same time, inconsistency state will occur in the agent learning process. Therefore, it is necessary for the agent to remember what has been learned within the time specified to improve the convergence and robustness of decision making. With regard to the above-mentioned issues, this paper proposes to accelerate the convergence rate of reinforcement learning by using the function approximation ability of neural network and to improve the robustness of reinforcement learning by using the Memory-based Experience-Replay(ER) algorithm. The experimental results show the effectiveness of the proposed method.**

Index Terms - Experience-Replay, Reinforcement Learning, Neural Network.

I. INTRODUCTION

Reinforcement learning algorithm mainly uses the environment state as the core factor with the agent interacting with the environment. The decision-making ability of the agent will be improved through learning regardless of the results of the policy [1] [2]. When the problem is complex, reinforcement learning can make better use of computing resources through value function. This paper studies reinforcement learning method from the perspective of value function estimation [3].

A. SARSA Algorithm

The SARSA algorithm, proposed in 1994 by Rummey [4], is an online Q-learning method, and belongs to the Q value update iteration method. The difference between SARSA and Q-learning is that the Q-learning uses maximum function in iterative update while the SARSA algorithm uses the actual trajectory data (s_t, a_t), (s_{t+1}, a_{t+1}) of the action at the next moment to estimate the update of the value function. The Q-learning is an offline learning mode while the SARSA algorithm is an online learning mode. The value function iteration of SARSA algorithm is calculated as

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_t(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (1)$$

where s_t *and* s_{t+1} donate the state, a_t *and* a_{t+1} are action. α_t is the learning rate and γ is the discount factor.

In this formula, the definition of each term is the same as that of Q-learning method. However, the behavioral policy and value function iteration are calculated as independent processes because Q-learning has to choose the maximum function in the calculation. But, the SARSA algorithm is more concerned with the actual execution of the agent, so the SARSA algorithm uses the iteration of the value function like the TD learning [5], that is to say, behavioral policy selection is not independent of the value function iteration process in the SARSA algorithm. In fact, the SARSA algorithm is proved to be better than Q-learning algorithm in practice [6]. The flow of the SARSA algorithm is algorithm 1.

Algorithm 1 SARSA algorithm

Initialize $Q(s,a)$
Repeat
  $s \leftarrow$ the current state; $a \leftarrow \varepsilon - greedy(s)$
  For each episode
    Execute action a ; observe next continuous state s' and reward r , $a \leftarrow \varepsilon - greedy(s')$
    Update $Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s', a') - Q(s, a_t))$
    $s \leftarrow s'$, $a \leftarrow a'$
  Until s is terminal state
Until the $Q(s,a)$ convergence

The SARSA algorithm can explore the behavioral policy using methods such as gradual greedy infinite exploration to improve the convergence of this algorithm.

B. Q-Neural Network Algorithm

Both Q-learning and the SARSA algorithms present the objective function as a dominant independents storage structure, such as a look-up table. A table item is allocated for each state-action pair, and both algorithms have been proved that the learning process will converge only if the state-action can be accessed indefinitely. However, the interaction between the agent and the environment cannot be made when the state-action space is large or the value of the action is too high. In practice, to solve this difficulty, function approximation method such back-propagation is often incorporated into the reinforcement learning algorithm; the neural network is used to replace the lookup table; and $Q(s,a)$ is updated as a training example [7]. For instance, the s, a is encoded as a network input, and the target as output using the Q updating method mentioned above. Deep Mind company uses CNN model in the deep neural network to do reinforcement learning [8], and the robot is trained by means of Auto-Encoder in other parts of the study. The autonomous navigation robot based on visual image processing is realized,

978-1-5386-8069-8/18//$31.00 ©2018 IEEE

Algorithm 2 Q-network algorithm

Initialize $Q(s,a)$ and ϖ_{ji}
For each episode do:
  For iterations $t>0$
    $s \leftarrow$ the current state; $a \leftarrow \varepsilon - greedy(s)$
    Execute action a; observe next continuous state s' and reward r, execute forward propagation
    Calculate the output value of neural network y_i
    Calculate the δ_k
    Calculate $\delta_h = o_h(1-o_h)\sum_{k\in output}\omega_{kh}\delta_k$
    Update $w_{ji} = w_{ji} + \Delta w_{ji}$, where $\Delta w_{ji} = \eta\delta_j x_{jx}$
Return convergence $Q(s,a)$

which also shows the effectiveness of the function approximation algorithm.

For Q-learning and SARSA algorithms, a corresponding lookup table item is often set up for each state-action. When the state space is large, the Q value look up table will take up a large amount of storage space. At the same time, since the execution cost is large when solving actual problems, in order to increase the generalization ability of the Q-learning algorithm, BP neural network, which is relatively strong in function approximation algorithm, can be used to replace look-up table to fit Q function, and to overcome the curse of dimensionality caused by state space [9].

According to the algorithm flow of BP network, the basic idea of Q-network method is that the state space is quantified as the input vector $s=(s_1,s_2,\cdots,s_n)$ of the neural network; ω_{ij} is weight parameter (the weight value of the layers, from input layer i to hidden layer j) between each network level; Θ_j is the hidden layer j; and the network output layer is a value function $Q(s,a), a\in \mathrm{A}$, which A is an action cluster. Unlike supervised learning, training samples have standard input and output samples, that is, accurate output values are calculated using neural network errors. In the Q-network method, the error value depends on the update formula of Q value in Q-learning or SARSA algorithm:

$$Q(s,a) = E[r+\gamma \max_{a'} Q(s',a') \mid s,a] \tag{2}$$

where s' donates the next state, a' is the action in next state.

The output value is the Q value after the action a is executed under the state s, and the loss function can be expressed as:

$$L(w) = E[(y_i - Q(s,a))^2], y_i = \begin{cases} r & \text{final state} \\ r+\gamma \max_{a'} Q(s',a') & \text{other state} \end{cases} \tag{3}$$

The corresponding error value is:

$$\delta = E[r+\gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4}$$

According to the BP neural network process, the error δ is used to propagate in the direction, and the neuron connection weight w_{ji} of each level is calculated and updated.

To make the Q-network process more concrete, this paper takes the GridWorld problem as an example: the agent expects to learn behavioural decision policies in the rasterized world, so that the agent can reach the target location in

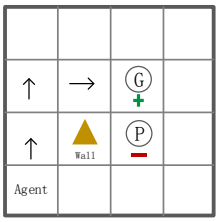


Fig. 1 The structure of Gridworld

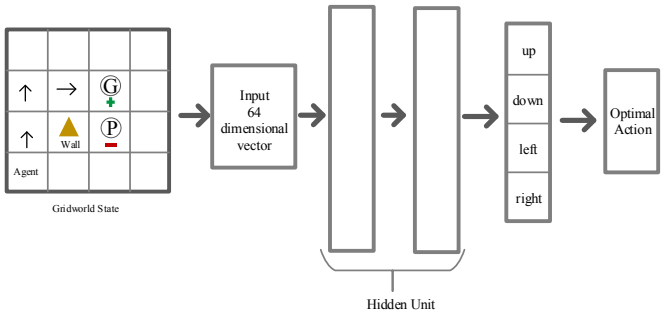


Fig. 2 The network structure of Q-network

GridWorld in an effective way or get the highest reward. As shown in Fig. 1, it is a rasterized scene of 4×4, in which the "+" is the target location, the reward is +10, and the "-" is the trap, punishment is the -10, the other states are 0, and "=" represents the wall, indicating that the grid cannot be marched to. Fig. 2 shows the network structure diagram combining Q-learning and BP neural network.

The process of solving this problem is as follow:

1) Initialize neural network parameters ω.

2) Input state s (64×1 vectors) as x_i.

3) obtain the Q value of the action after executing forward propagation.

$$Q_A = (\text{up、 down、 left、 right}) = (q_1,q_2,q_3,q_4)$$

4) $a \leftarrow \varepsilon - greedy$ and execute a.

$$Q_A = (\text{up、 down、 left、 right}) = (10,q_2,q_3,q_4) = y_i$$

5) (x_i,y_i) is the training sample, calculate the current neural network error and execute back-propagating.

6) Complete iteration and obtain Q_A

$$Q_A = (\text{up、 down、 left、 right}) = (q_1',q_2',q_3',q_4')$$

7) select the maximum value from (q_1',q_2',q_3',q_4'), the corresponding action is the optimal policy action.

At this point, the Q-network process is completed. The GridWorld process is a common experimental problem in the reinforcement learning, including the mobile robot navigation, games and other fields, which can be reduced to this process. It will be studied in depth later in this paper.

II. The Learning Algorithm Based on Experience-Replay

The Model-Free characteristics in the reinforcement learning algorithm makes it to be applied to the decision-making problem of the unpredictable environment model [10], and also makes it widely used in the dynamic uncertain environment, including nonlinear deterministic or random

processes, and the need to approximate the continuous variables in the dynamic uncertain solution domain problem. However, for practical application in real time, online fitting reinforcement learning algorithm needs to meet two conditions: 1) after the simple interaction between the agent and the decision environment, the information obtained should have strong representability to meet the requirements of the algorithm. For a real-time control system, if a satisfactory decision-making behavior has not been achieved within a certain number of decision-making cycles, it indicates that the interactive return information is invalid, i.e., this series of policies are the best policies [11]; 2) the online reinforcement learning algorithm needs to have high computing efficiency so that the decision process of the algorithm can be applied to the real-time system [12].

The existing online reinforcement learning algorithms are based on the classical Q-learning algorithm and the SARSA algorithm. Although the two are relatively high in computing efficiency, they only use one example to train the agent in each update to improve the decision-making ability of the agent, and the sample is discarded after the training. In the real world, the most effective way is to use the Memory-Based method [13] [14] to improve the robustness of the learning process of the agent, that is to collect and store effective optimization decisions during the decision-making process of the agent and to reuse the decisions stored in the online learning process, so as to improve the efficiency of the data on the premise of ensuring the computing efficiency of the algorithm. Therefore, this paper proposes an empirical evolutionary reinforcement learning method based on Experience Replay method, that is, ER-RL algorithm, and applies this method to real-time decision-making system.

A. Approximate Expression of Reinforcement Learning

In practice, when the state-action space is large, it is difficult to define all the state action pairs accurately, so approximate representation should be used [15]. Application of linear approximation is expressed as follows:

$$\hat{Q}(s,a)=\sum_{i=1}^{n}\phi_i(s,a)\theta_i=\phi^T(s,a)\theta \tag{5}$$

where $\theta\in\Re^n$ is a parameter vector, $\phi: S\times A\to\Re^n$ is the base function of the parameter vector (BFs) $[\phi_1(s,a),\cdots,\phi_n(s,a)]^T$. This design can simplify the subsequent algorithm description, which is also widely used in approximate reinforcement learning algorithms, such as interpolative representation, Fuzzy approximation and so on. The gradient descent algorithm can be used in Q-network, and the formula for iterative updating parameters of the gradient descent method in the Q-learning and SARSA algorithms with linear approximation functions is as follows:

$$\theta_{k+1}=\theta_k+\alpha_k\delta_k\frac{\partial}{\partial\theta}\hat{Q}_k(s_k,a_k)=\theta_k+\alpha_k\delta_k\phi(s_k,a_k) \tag{6}$$

δ_k is a general time difference error. For Q-learning approximation algorithm, δ_k can be replaced by Q error:

$$\delta_k^S=r_{k+1}+\gamma\phi(s_{k+1},a_{k+1})^T\theta_k-\phi(s_k,a_k)^T\theta_k \tag{7}$$

Algorithm 3 E-R-Q-network algorithm

Initialize θ, $D\leftarrow\varnothing, l\leftarrow 1, k\leftarrow 0$
For each episode do:
  For each time interval k
    $s_0\leftarrow$ the current state; $a\leftarrow\varepsilon-greedy(s)$
    Execute action a; observe next continuous state s' and reward r, execute forward propagation
    Calculate the migration sample index corresponding to the action $\tau\leftarrow k-(l-1)\times T$
    migration sample $\{(k,l,\tau,s_k,a_k,s_{k+1},a_{k+1})\}$ stored in the D
    Select migration sample from D
      Calculate output value y_i
      Update θ
Return

The above algorithm [16] only updates with the latest transformation, and the latest trajectory selected according to all running trajectories will enhance the convergence speed of the algorithm. To achieve this goal, the latest state action pairs for algorithm browsing can be updated by using eligibility trace $e\in\Re$. In this paper, the replacement trace is initialized to 0 and updated with (8):

$$e_k=\min\lambda\gamma e_{k-1}+\phi(s_k,a_k) \tag{8}$$

When $\lambda=0$, it is the original algorithm. For the eligibility trace problem, since the Q-learning algorithm is updated using the algorithm, the causality effect of the state action pair is broken when an action is selected for execution. So, it needs to be reset as "0" after each execution. The SARSA algorithm only considers the next step, including the strategy exploration process, therefore the value doesn't need to be reset.

B. Experience Replay

The E-R reinforcement learning algorithm improves the ability of reinforcement learning algorithm by saving transition samples and repeatedly using the migration samples, improves the use of effective data on the premise of guaranteeing the computing capability, and uses the linear approximation function to integrate the method into the Q-network algorithm.

The length of the track storage area D is set to N, trajectory length is set to T, BFs is set to $\phi_1,\cdots,\phi_n$. Adopting $\varepsilon-greedy$ action selection method, the E-R-Learning algorithm flow framework is algorithm 3.

The above process gives the framework of ER algorithm. For each time slice k, the migration sample $(s_k,a_k,s_{k+1},a_{k+1})$ is obtained and stored in the memory D, then the action a is selected by a certain way. This paper adopts the classical $\varepsilon-greedy$ action selection method to carry out the agent exploration and use of the problem balance. Other methods can also be used here.

$$a=\begin{cases}\max\phi^T(s,a)\theta\\ \text{random choose action a}\in\text{A}\end{cases} \tag{9}$$

For each selection within the length of T, a random sampling method can be used to update the parameter θ in the D. When the iteration is over, T (number) samples are

collected. The process of these samples being combined is called the trajectory. Each trajectory is represented by l, and the behaviour in discrete time can be indicated by $\tau = 0, \cdots, T-1$. Therefore, the algorithm can be applied to time-varying learning rate and exploration probability, etc.

Two different ways are needed to apply the collected samples when applying this method to Q-learning and SARSA methods, namely the Replay process:

1) A single policy sample is randomly extracted from *D*, and the specific process is: sampling samples $\{(k, l', \tau, s, a, s', a')\}$ randomly from memory D, calculating the error δ and updating the parameter $\theta \leftarrow \theta\alpha(\delta^Q / \delta^S)\phi(s, a)$;

2) The policy trajectories are extracted independently in *D*, and each sample in the trajectory is used one by one based on their sequence: trajectory $l' \in \{1, \cdots, 1\}$ is randomly sampled from *D;* the sample $\{(k, l', \tau, s, a, s', a')\}$ corresponding to each $\tau \in \{0, \cdots, T-1\}$ in l' is calculated in an iterative way; and the error δ is calculated, and the parameter $\theta \leftarrow \theta\alpha(\delta^Q / \delta^S)\phi(s, a)$ is updated.

Combining E-R with Q-learning and SARSA methods, we can get: 1) ER-Q-trajectories; 2) ER-Q-separate-samples; 3) ER-SARSA-trajectories; 4) ER- SARSA -separate-samples. The above algorithms can ensure the convergence in the actual decision-making problem because they are in essence reinforcement learning algorithms. Literature [17] has proved that ER Reinforcement Learning method is effective.

C. The Analysis on the Computational Complexity of E-R Algorithm

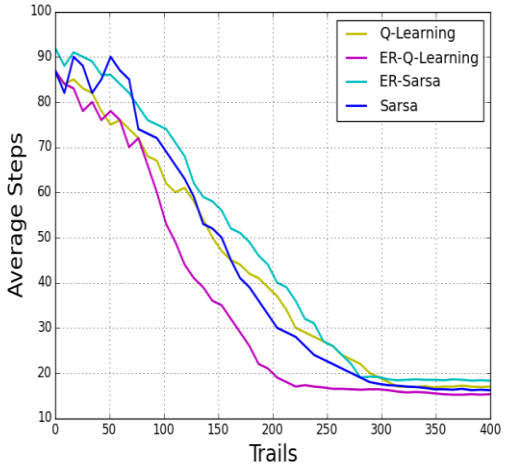
For the E-R algorithm-based reinforcement learning algorithm, its main computational complexity is the parameter updating part. The effort of computing the policy, the preservation of the sample, and the sample calculation is so small that it can be neglected. According to the previous parameter update method: $\theta \leftarrow \theta\alpha(\delta^Q / \delta^S)\phi(s, a)$, it is assumed that the action space of the agent is discrete state space $A=\{a_1, \cdots, a_n\}$, the Q value is updated by enumerating all actions, the complexity is $O(n)$, and the computational complexity of the whole reinforcement learning update is $O(n)$. For any trajectory l, there are NlT (number) behavior samples in memory D and they are used for incremental parameter updating, so the overall complexity is $O(nNlT)$, in which N represents the number of Replay. For the whole process, the maximum value L is reached by repeating the qualified path l, and the overall algorithm complexity is $O(nNL^2T)$.

This is also the main feature of the algorithm. Fully making use of effective data can reduce the complexity of the algorithm, and the whole algorithm can be adjusted by adjusting the number (N) of parameters of the Replay. The lower the value (N) is, the lower the complexity of the algorithm is. The higher the value (N) is, and the higher the accuracy of the algorithm is. In order to maximize the search for effective data, the iterations of the algorithm will be twice as large as the number of samples in memory. However, there is no need to do all iterations in the actual use of the algorithm, only the iteration of the fixed number of times is needed, that is, the same result will be obtained by debugging the parameters that conform to the decision problem.

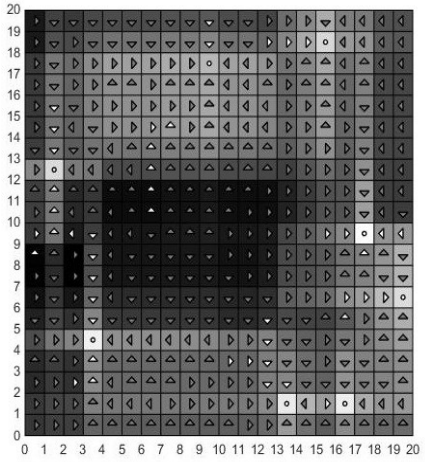
III. SIMULATION EXPERIMENT

A. Gridworld

Taking Gridworld as a decision environment, here is a Grid of 20×20, △▽◁▷ represent the optional actions for the agent, ◯ represents a state of reward with +1. In the eight directions around it, the random cell is a penalty with -1. Unlike the previous Gridworld task, the initial location and location of the target are iteratively calculated. In this paper, the target location and the initial location of the agent are randomly set in each training process. The parameters of each part of this paper are {N=30，T=50，α=0.1，ε=0.2，γ=0.9}.



(a) The average learning steps



(b) GridWorld State

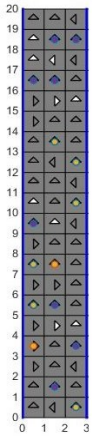
Fig.3 The experimental result of GridWorld

The experimental results, as shown in Fig. 3 (a), show that the number of training Trail and the average step number

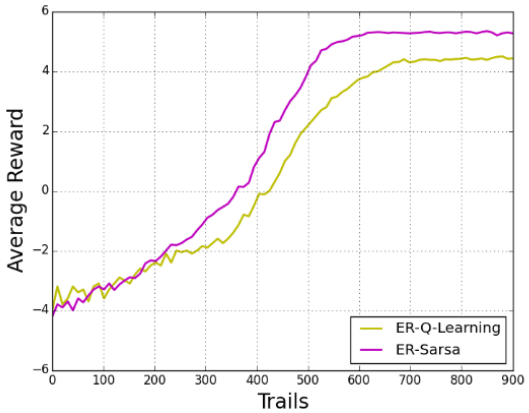
of the agent arrive at the target state, in which the convergence rate of the ER-Q-Learning algorithm and the ER-SARSA algorithm is better than that of the Q-learning algorithm and the SARSA. But in the initial state of the algorithm, the ER algorithm will do the training using the movement trails recorded in the memory. But in the initial state, the storage sequence of the memory is not completely convergent, so when the number of Trail is small, the ER algorithm will greatly fluctuate and become unstable. Fig. 3 (b) shows the final state of the training result.

B. Simulation of Highway Driving

In addition to the Gridworld decision environment, this paper uses the Highway Driving Behavior simulation as an example to verify the effect of the ER-Q-network algorithm. The experimental target is "driving" car, driving on the three-lane highway. The agent can choose the driving lane and can choose the speed of the car ranging from 1-4 level. Other cars on the highway, either ordinary vehicles or police cars, run at a fixed speed. The agent need to advance as fast as possible and will be rewarded for +1 every time when it overtakes one vehicle. But in the process of moving ahead, it should overtake the police car at a speed not more than twice as that of the police car, otherwise it will be regarded as overspeed with a penalty of -1. If the speed of the agent is too slow and collision occurs, the penalty will be -2. Parameters are {N=50，T=30，α=0.2，ε=0.1，γ=0.9}.



(a) Highway simulation environment



（b）The Average Reward

Fig.4 The experimental result of Highway Driving

the results of the experiment are as shown in Figure 4-10. ▲◀▶▷ represents the movements of agents, while the white arrow means accelerate. 🟠🔵🟡 represents the ordinary vehicles, motorcycles, and police cars respectively. The speed of the motorcycle is higher than that of the ordinary vehicle and the police car, so it is necessary for the agent to move faster than the motorcycle to overtake it. In this paper, the average reward of ER-Q-Learning algorithm and ER-SARSA algorithm is obtained for each running environment 300s, as shown in Fig. 4 (a) for different Trail numbers. The value updating method for Q-learning and SARSA is different and this experiment focuses on verifying that the effect of the two algorithms on this kind of problem. As shown in the diagram, as for the expressway, which is more focused on the current state and the next step, the SARSA algorithm is better than the Q-learning method, the average reward is as shown in Fig. 4 (b).

V. Conclusion

This paper mainly introduces the combination of Q-learning algorithm, SARSA algorithm and neural network, and integrates the effectiveness of the data and the robustness of the algorithm in Experience Replay. In order to verify the effect of the algorithm, a simulation test is carried out with Gridworld and HighWay as examples. After combining with Experience Replay method, the experimental results show that the convergence speed of the ER algorithm is better than that of the reinforcement learning algorithm of the basic Q-network, and the use of data information is more effective. Secondly, comparing the difference between Q-learning algorithm and SARSA algorithm under ER algorithm, experiments show that SARSA algorithm is more suitable for issues with stronger correlation of adjacent state.

References

[1] Littman M L. Reinforcement learning improves behaviour from evaluative feedback[J]. Nature, 2015, 521(7553): 445-451.

[2] Kober J, Bagnell J A, Peters J. Reinforcement learning in robotics: A survey[J]. The International Journal of Robotics Research, 2013: 0278364913495721.

[3] Engle R F, Russell J R. Autoregressive conditional duration: a new model for irregularly spaced transaction data[J]. Econometrica, 1998: 1127-1162.

[4] Rummery G A, Niranjan M. On-Line Q-Learning Using Connectionist Systems[J]. 1994.

[5] Sutton R S. Learning to Predict by the Method of Temporal Differences[J]. Machine Learning, 1988, 3(1):9-44.

[6] Singh S P, Sutton R S. Reinforcement learning with replacing eligibility traces.[M]. Kluwer Academic Publishers, 1996.

[7] Sorokin I, Seleznev A, Pavlov M, et al. Deep Attention Recurrent Q-Network[J]. Computer Science, 2015.

[8] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge.[J]. Nature, 2017, 550(7676):354-359.

[9] Zhao X H, Shi J J, Zhen-Long L I, et al. Traffic Signal Control Based on Q-learning and BP Neural Network[J]. Journal of Highway & Transportation Research & Development, 2007.

[10] Uther W T B, Veloso M M. Tree based discretization for continuous state space reinforcement learning[C]// Fifteenth National/tenth Conference on Artificial Intelligence/innovative Applications of Artificial Intelligence. American Association for Artificial Intelligence, 1998:769-774.

[11] Zhao D, Zhu Y. MEC--a near-optimal online reinforcement learning

algorithm for continuous deterministic systems[J]. IEEE Transactions on Neural Networks & Learning Systems, 2015, 26(2):346-356.
[12] Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization[J]. Journal of Machine Learning Research, 2011, 12(7):257-269.
[13] Sarkar K, Shaw S K. A Memory-Based Learning Approach for Named Entity Recognition in Hindi[J]. Journal of Intelligent Systems, 2017, 26(2):301-321.
[14] Goode A. Efficient Memory-based Learning for Robot Control[J]. Phd Thesis Computer Laboratory University of Cambridge, 1990.
[15] Lewis F L, Liu D. Reinforcement Learning and Approximate Dynamic Programming for Feedback Control[J]. IEEE Circuits & Systems Magazine, 2015, 9(3):32-50.
[16] Kakade S, Langford J. Approximately Optimal Approximate Reinforcement Learning[C]// Nineteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc. 2002:267-274.
[17] D. Ernst, "Near optimal losed-loop control. Application to electric power systems," Ph.D. dissertation, University of Li`ege, Li`ege, Belgium, Mar.2003.