Applied Intelligence
https://doi.org/10.1007/s10489-019-01487-4

A new asynchronous reinforcement learning algorithm based on improved parallel PSO

Shifei Ding[1] · Wei Du[1] · Xingyu Zhao[1] · Lijuan Wang[1,2] · Weikuan Jia[3]

Abstract
As an important machine learning method, reinforcement learning plays a more and more important role in practical application. In recent years, many scholars have studied parallel reinforcement learning algorithm, and achieved remarkable results in many applications. However, when using existing parallel reinforcement learning to solve problems, due to the limited search scope of agents, it often fails to reduce the running episodes of algorithms. At the same time, the traditional model-free reinforcement learning algorithm does not necessarily converge to the optimal solution, which may lead to some waste of resources in practical applications. In view of these problems, we apply Particle swarm optimization (PSO) algorithm to asynchronous reinforcement learning algorithm to search for the optimal solution. First, we propose a new asynchronous variant of PSO algorithm. Then we apply it into asynchronous reinforcement learning algorithm, and proposed a new asynchronous reinforcement learning algorithm named Sarsa algorithm based on backward Q-learning and asynchronous particle swarm optimization (APSO-BQSA). Finally, we verify the effectiveness of the asynchronous PSO and APSO-BQSA algorithm proposed in this paper through experiments.

Keywords Reinforcement learning · Asynchronous · Parallel · Particle swarm optimization · Artificial intelligence

1 Introduction

As an important machine learning methods, reinforcement learning (RL) enables agents to learn the mapping relationship from states to actions by trial and error in the continuous interaction with the environment, so as to maximize the long-term cumulative reward [1]. At present, reinforcement learning methods have been widely used in intelligent control, robotics and other fields [2–4].

In 2016, Mnih et al. proposed an asynchronous architecture for deep reinforcement learning [5]. Asynchronous reinforcement learning adopts several parallel actor-learners to explore the environment, and each actor-learner online updates the global parameters. Using this approach, it can no longer rely on experience reply to store historical experience, and greatly shorten training time. Based on this method, Zhao et al. [6, 7] put forward a general framework of asynchronous reinforcement learning algorithms, which further speeds up the convergence speed of the algorithms. However, when using existing parallel reinforcement learning to solve problems, due to the limited search scope of agents, it often fails to reduce the running episodes of algorithms. At the same time, the traditional model-free reinforcement learning algorithm does not necessarily converge to the optimal solution, which may lead to some waste of resources in practical applications.

Although reinforcement learning has been widely applied, there are still some problems. For example, although Sarsa converges faster, it is easy to fall into local minima. The convergence precision of Q-learning is relatively good, but its convergence is slow. In order to speed up the convergence of the algorithm and improve the convergence accuracy of the algorithm, Wang et al. Proposed a backward Q-learning algorithm, which is named BQSA [8]. BQSA can enhance learning speed and improve final performance. However, when BQSA performs backward update, it still uses the continuous states of an agent, and cannot eliminate the correlation of different states.

✉ Shifei Ding
dingsf@cumt.edu.cn

Xingyu Zhao
757008724@qq.com

[1] School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China

[2] School of Information and Electrical Engineering, Xuzhou College of Industrial Technology, Xuzhou 221000, China

[3] School of Information Science and Engineering, Shandong Normal University, Jinan 265000, China

Published online: 25 May 2019

Particle swarm optimization (PSO) is a general optimization algorithm, which is proposed by Kennedy et al. [9]. It starts from the random solution and evaluates the quality of the solution through fitness, then finds the optimal solution by iteration. It finds the global optimal value through the current optimal value, which has the advantages of easy realization, high precision and fast convergence speed. After being put forward, PSO soon attracted the attention of the academic community and demonstrated its superiority in solving practical problems.

Based on previous research results, we propose a new asynchronous reinforcement learning algorithm based on improved parallel PSO parallel (APSO-BQSA). APSO-BQSA algorithm first introduces asynchronous PSO algorithm into asynchronous reinforcement learning, it uses our improved parallel PSO algorithm, and combines our improved BQSA algorithm to update the parameters of reinforcement learning asynchronously, which greatly improves the convergence speed and convergence precision of the algorithm.

The other parts of this article are as follows: in Section 2, we introduce the basic theory of reinforcement learning and PSO algorithm. Section 3 gives the details of the APSO-BQSA algorithm. In Section 4, we verify the effectiveness of the proposed algorithm through experiments. And the last part is the conclusion.

2 Basic theory

2.1 Reinforcement learning

We consider tasks in which an agent interacts with an environment ε over a number of discrete time steps. At each timestep, the agent receives a state s and selects a legal action a according to its policy π, where $\pi : S \rightarrow A$ is a mapping from states to actions. The agent executes the selected action a, then the state turns into s' and the environment returns a reward r to the agent. The agent then selects the next action based on the reward signal.

Let us take a classic video game Tetris Russia as an example to illustrate the whole process of reinforcement learning. In Tetris, the environment is the whole game, and the state is the game scene displayed on the screen. During the game, the player needs to constantly choose actions a (left, right, or down) according to the states s to eliminate bars and maximize the score. In the meantime, the reward is shown as the score of the player, and the policy that the player chooses action a under the state s is called the action selection policy $\pi(s, a)$. Figure 1 gives the framework of reinforcement learning.

In reinforcement learning, we use Q value to represent the value of different state-action pairs. We usually use standard model-free reinforcement learning methods such as Q-learning and Sarsa learning to solve problems [10–15]. In Q-learning, the Q value is updated as follows:

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

In formula (1), $Q(s, a)$ denotes the Q-value of the state-action pair (s, a), 훼 denotes the learning rate, and $\gamma \in (0, 1)$ denotes the discount factor.

In Sarsa learning, we use the actual Q value for iteration. So Sarsa learning is called the on-policy method. In Sarsa learning, the Q value is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2)$$

In addition to the model-free methods mentioned above, we can also use the model-based methods such as Dyna to implement the reinforcement learning algorithms. Dyna architecture integrates learning and planning, which means agents can use the experience to build an environment model and use the environment model to generate hypothesis experience as a learning resource, which can effectively improve the convergence speed of the value function (Fig. 2), shows the general Dyna framework.

BQSA is a novel reinforcement learning algorithm, which combines Sarsa algorithm and Q learning algorithm. In BQSA algorithm, after an action is executed, the agent uses Sarsa algorithm to update the parameters directly. Its update formula is formula (2). At the end of an episode, the parameters are updated indirectly using backward Q-learning algorithm. The backward update formula is formula (1).

2.2 Particle swarm optimization algorithm

Particle swarm optimization algorithm mainly simulates the predation behavior of birds. PSO algorithm first initializes a swarm of random particles (random solutions), and then updates the position of each particle through their individual experiences and the swarm experience. In each iteration, the particles are updated by tracking two optimal values: one is the individual optimal value P_{best}, and the other is the global optimum value G_{best}. Formula (3) and (4) gives the update formula of PSO algorithm:

$$v_i \leftarrow w v_i + c_1 r_1 (p_i - x_i) + c_2 r_2 \left(p_g - x_i\right) \quad (3)$$

$$x_i \leftarrow x_i + v_i \quad (4)$$

Where $v_i \in [v_{\min}, v_{\max}]$ and $x_i \in [x_{\min}, x_{\max}]$ denote the velocity and position of the ith particle, $w$denotes the inertia factor, c_1 and c_2 are two hyper parameters of the algorithm, p_i denotes the individual optimal value of the i thparticle, p_g denotes the global optimal value of global, and r_1,r_2 are two random number between 0 and 1.

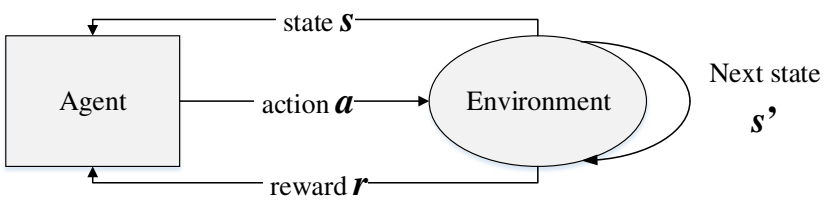


Fig. 1 The framework of reinforcement learning

The iterative termination conditions of particle swarm optimization algorithm are generally selected according to specific problems, such as satisfying the maximum number of iterations or meet the fitness threshold.

The traditional parallel PSO algorithm uses master-slave mode. The master process mainly completes the random initialization of the population, the distribution of tasks and the selection of particles according to the fitness value. The slave process mainly completes the calculation of the fitness value of particles. After the calculation, the slave process returns the fitness value of each particle to the main process. Master process compares the fitness value of each particle with its individual extreme value, and selects a better individual extreme value. Then the optimal solution of the whole population is selected by comparison, and then the position and speed of each pull is updated. This method does not make full use of parallelization technology to improve computing speed. At the same time, this method cannot effectively solve the problem of algorithm getting into local minimum.

3 APSO-BQSA algorithm

3.1 Asynchronous PSO

Traditional particle swarm optimization algorithm only requires a small number of hyper parameters, which is simple and easy to use. However, it has the disadvantage of easily falling into local minimum, and its search accuracy is not high. To overcome these shortcomings, we propose a new multi-threading asynchronous parallel particle swarm optimization algorithm in this paper which is name APSO.

The asynchronous parallel PSO algorithm proposed in this paper transforms particles into multiple populations, and each population corresponds to one thread. Each population explores the environment alone, which avoids the problem of local minima. At the same time, because we adopt the multi-thread method, each thread executes its own exploration, which makes the algorithm can explore a wider range and improves the convergence precision of the algorithm. Each thread uses standard PSO algorithm, which ensures the convergence of the algorithm. We defined f_{best} and x_{best} as the optimal fitness and the optimal position. When calculating the fitness of each particle, if it is found that its fitness f_i is greater than f_{best}, then f_{best} and x_{best}will be updated to f_i and x_i. In the same way, each thread has its own optimal fitness $f_g$and optimal position x_g, they can also be updated when f_i is greater than f_g.

In order to achieve interaction between threads, we define a thread update factor t_{update} to denote the update frequency of threads. When each thread iterated a certain number of times, it will interact with the global optimal value. In order to avoid the algorithm falling into local minimum, we do not copy the global optimum value to the optimal value of the thread directly. In order to make rational use of the global optimal value and the thread optimal value, we define an update factor β to approximate the local optimal value to the global optimal value. The formula (5) gives the updating method.

$$x' \leftarrow \beta \cdot x_{best} + (1-\beta) \cdot p_g \tag{5}$$

In the operation, we first use the method given by the formula (5) to generate a temporary position x'. Next we calculate the fitness of x' and decide whether to update the optimal value and the optimal position according to it. Algorithm 1 gives the pseudocode of the proposed APSO algorithm.

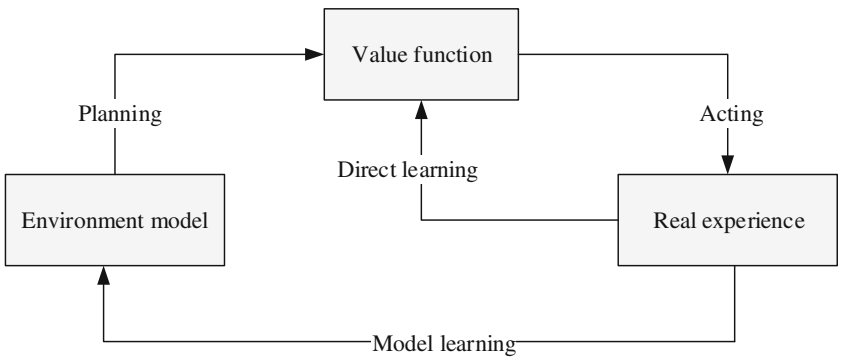


Fig. 2 The general Dyna framework

Algorithm 1 The proposed APSO algorithm - pseudocode for each thread.

// *Assume global shared the best position* x_{best} *and the best fitness* f_{best}
Initialize thread step counter $t \leftarrow 0$ and maximum number of iterations $T_{\max}$
Initialize the number of particles n, inertia factor w and learning factors c_1, c_2
Initialize the position x_i and velocity v_i of each particle (uniform distribution)
Initialize the best position p_g and the best fitness f_g
Initialize the best position p_i and the best fitness f_i' of each particle
repeat
  $i = 1$
  repeat
    Calculate of the fitness of each particle f_i
    Update $f_{best}, f_g, f_i', x_{best}, p_g, p_i$ if f_i is larger
    $v_i \leftarrow w_i \cdot v_i + c_1 \cdot r_1 \cdot (p_i - x_i) + c_2 \cdot r_2 \cdot (p_g - x_i)$
    $x_i \leftarrow x_i + v_i$, $i \leftarrow i + 1$
  until $i > n$
  $t \leftarrow t + 1$
  if t mod $t_{update} == 0$ **then**
    $x' \leftarrow \beta \cdot x_{best} + (1 - \beta) \cdot p_g$
    Calculate of the fitness of x'
    Update $f_{best}, f_g, x_{best}, p_g$ if the fitness of x' is larger
  end if
until $t > T_{\max}$

3.2 APSO-BQSA algorithm

BQSA is an efficient reinforcement learning algorithm. It combines Q-learning with Sarsa algorithm, so that the algorithm has high convergence speed and good convergence precision. The basic idea of BQSA is that agent performs Sarsa algorithm updates every step. At the end of an episode, it backward updates the parameters according to backward Q-learning algorithm. However, when BQSA performs backward update, it still uses a continuous state of agent, and cannot eliminate the correlation of different states. In order to solve this problem, we set up an experience reply for the BQSA algorithm. The improved BQSA algorithm selects the state randomly when executing backward update.

In order to integrate BQSA algorithm into our asynchronous framework, we redefine the formulas used in Sarsa algorithm and backward Q-learning algorithm in formula (6) and formula (7).

$$Q(s_t^i, a_t^i) \leftarrow Q(s_t^i, a_t^i) + \alpha_1 \left[r_{t+1}^i + \gamma_1 Q(s_{t+1}^i, a_{t+1}^i) - Q(s_t^i, a_t^i) \right] \quad (6)$$

$$Q(s_t^j, a_t^j) \leftarrow Q(s_t^j, a_t^j) + \alpha_2 \left[r_{t+1}^j + \gamma_2 \max_a Q(s_{t+1}^j, a) - Q(s_t^j, a_t^j) \right] \quad (7)$$

Where $i = 1, 2, \cdots N$ and $j \in [1, N]$are the number of times for updated the Q-learning and Sarsa algorithm function in current episode. α_1 and α_2 are the learning rates. γ_1 and γ_2 are the discount factor. In this way, we can combine Q-learning with Saras algorithm in asynchronous environment, and learn the knowledge acquired by agent better by backward Q-learning.

Combining APSO algorithm and improved BQSA, we propose the APSO-BQSA algorithm. APSO-BQSA divides threads into the APSO threads and the BQSA threads. In APSO-BQSA algorithm, APSO is executed in parallel with the improved BQSA algorithm. The parameters are updated by the improved BQSA algorithm, and APSO is used to search the optimal parameters. In this way, the algorithm can converge faster and avoid convergence to local optima.

In order to realize the interaction between APSO and improved BQSA algorithm, we redefine the parameter values used in PSO search. We stipulate that when the state of agent is s, the corresponding Q-value of action a' is:

Table 1 Experimental results of APSO algorithm (with 5 parameters)

Function	Algorithm	Mean value	standard deviation
Ackley	PSO	7.5794e-01	2.1271e+00
	Modified PSO	2.8069e-01	7.0361e-01
	DMS-PSO	4.7474e-02	4.8260e-02
	APSO	**4.8038e-04**	**5.8871e-04**
Griewank	PSO	2.4571e-01	1.0088e-01
	Modified PSO	2.0124e-01	9.6534e-02
	DMS-PSO	2.0618e-01	7.0051e-02
	APSO	**5.4116e-02**	**3.0147e-02**
Rastrigin	PSO	5.0165e+00	4.6479e+00
	Modified PSO	6.0078e+00	6.2935e+00
	DMS-PSO	2.9552e+00	1.8523e+00
	APSO	**1.6258e+00**	**8.5373e-01**
Schwefel_1	PSO	5.4250e-02	1.1151e-01
	Modified PSO	8.4560e-03	5.0145e-02
	DMS-PSO	8.0527e-03	3.8067e-02
	APSO	**1.0378e-06**	**2.4470e-06**
Schwefel_2	PSO	7.3810e-02	1.4094e-01
	Modified PSO	5.5811e-03	1.3359e-02
	DMS-PSO	1.4861e-02	2.4568e-02
	APSO	**1.2383e-04**	**1.2251e-04**
Sphere	PSO	4.8261e-01	1.9702e+00
	Modified PSO	5.5965e-03	2.9153e-02
	DMS-PSO	1.3900e-02	6.2389e-02
	APSO	**7.6968e-07**	**1.2740e-06**

$$Q\left(s,a^{'}\right)=\frac{Q_{APSO}\left(s,a^{'}\right)-\min\limits_{a}Q_{APSO}(s,a)}{\max\limits_{a}Q_{APSO}(s,a)-\min\limits_{a}Q_{APSO}(s,a)}+\frac{Q_{BQSA}\left(s,a^{'}\right)-\min\limits_{a}Q_{BQSA}(s,a)}{\max\limits_{a}Q_{BQSA}(s,a)-\min\limits_{a}Q_{BQSA}(s,a)} \tag{8}$$

Where Q_{APSO}, Q_{BQSA} denote the Q-value of APSO algorithm and BQSA algorithm respectively. We normalize the Q values of APSO and BQSA respectively to eliminate the influence of different hyper parameter settings, and make full use of the updating information of BQSA algorithm. The pseudocode of the proposed APSO-BQSA algorithm is given in algorithm 2.

Table 2 Experimental results of APSO algorithm (with 10 parameters)

Function	Algorithm	Mean value	standard deviation
Ackley	PSO	2.3908e+00	2.3982e+00
	Modified PSO	1.3348e+00	1.1167e+00
	DMS-PSO	4.6077e-01	3.9739e-01
	APSO	**4.1282e-03**	**1.2254e-02**
Griewank	PSO	8.0585e-01	1.7952e-01
	Modified PSO	4.0045e-01	2.2366e-01
	DMS-PSO	6.2756e-01	2.2186e-01
	APSO	**9.5393e-02**	**4.9496e-02**
Rastrigin	PSO	1.9115e+01	1.7878e+01
	Modified PSO	1.7204e+01	1.3197e+01
	DMS-PSO	7.3710e+00	3.6550e+00
	APSO	**5.5034e+00**	**2.2764e+00**
Schwefel_1	PSO	7.5930e+00	1.0531e+01
	Modified PSO	4.7122e+00	1.4688e+01
	DMS-PSO	4.1605e-01	7.9101e-01
	APSO	**7.8379e-04**	**1.4234e-03**
Schwefel_2	PSO	5.9976e-01	6.3280e-01
	Modified PSO	2.6935e-01	2.9062e-01
	DMS-PSO	8.8367e-02	5.8919e-02
	APSO	**4.7290e-03**	**5.4422e-03**
Sphere	PSO	2.9234e+00	4.1416e+00
	Modified PSO	2.6240e-01	5.7644e-01
	DMS-PSO	2.7751e-01	5.6818e-01
	APSO	**2.6352e-05**	**3.6078e-05**

Algorithm 2 The proposed APSO-BQSA algorithm - pseudocode for each thread.

// *Assume global shared the best cumulative reward* R_{best} *and* $Q^*(s,a)$
Initialize thread step counter $t \leftarrow 0$
Initialize $Q(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
repeat
  Initialize S
  With probability ε select a random action A
  Otherwise $A \leftarrow \max_a Q(S,a)$ according to formula (8)
  repeat
    Execute action A, observe reward R and the next state S'
    With probability ε select a random action A'
    Otherwise $A' \leftarrow \max_a Q(S',a)$ according to formula (8)
    if BQSA thread **then**
      Update $Q^*\left(s_t^i, a_t^i\right)$ according to formula (6)
    end if
    $S \leftarrow S'$, $A \leftarrow A', t \leftarrow t+1$
    if S is terminal **and** APSO thread **then**
      Perform PSO algorithm according to algorithm 1
    end if
    if S is terminal **and** BQSA thread **then**
      Backward update $Q\left(s_t^i, a_t^i\right)$ according to formula (7)
    end if
  until S is terminal
until $t > T_{\max}$

4 Experiments

In order to verify the effectiveness of the algorithms proposed in this paper, we carried different experiments on APSO and APSO-BQSA algorithm. Among them, we verified the effectiveness of the APSO algorithm through six standard test functions. And then, we verified the effectiveness of the APSO-BQSA algorithm through two standard reinforcement learning environments, cliff-walking problem and frozen-lake problem. Our experiments ran under the software environment of Windows 10 and Python 3.6.4. The CPU we used is Intel i5-3317 U with 4 cores, and the memory is 10GB.

4.1 Experiments of APSO algorithm

In order to verify the effectiveness of APSO algorithm, we choose six standard intelligent algorithm test functions to test the effectiveness of the algorithm. Their information is given in formula (9) to (14).

(1) Ackley function ($x_i \in [-32.768, 32.768], f(x^*) = 0$)

$$f(x) = -20*\exp\left(-0.2*\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sqrt{\sum_{i=1}^{d}\cos(2\pi x_i)}\right) + 20 + e \tag{9}$$

(2) Griewank function ($x_i \in [-600, 600], f(x^*) = 0$)

$$f(x) = \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{10}$$

(3) Schwefel function ($x_i \in [-500, 500], f(x^*) = 0$)

$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{|x_i|}\right) \tag{11}$$

(4) Schwefel's problem1.2 ($x_i \in [-100, 100], f(x^*) = 0$)

$$f(x) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2 \tag{12}$$

(5) Schwefel's problem2.22 ($x_i \in [-10, 10], f(x^*) = 0$)

$$f(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{13}$$

(6) Sphere function ($x_i \in [-5.12, 5.12], f(x^*) = 0$)

$$f(x) = \sum_{i=1}^{d} x_i^2 \tag{14}$$

In the test experiment of intelligent algorithm, the number of parameters has a great influence on the experimental results. So we tested the performance of the original PSO algorithm, the modified PSO algorithm [16], DMS-PSO algorithm [17] and the APSO algorithm on these six test functions with the number of parameters 5, 10, 20 and 100 respectively. The number of iterations is set to 100 times. The parameters of the experiment are as follows: the inertia factor w is 0.9, the learning factor $c_1 = 2$, $c_2 = 2$. The number of particles is 50. In APSO algorithm, the update rate β is 0.1, the update frequency t_{update} is 5, and the number of threads is 8. Tables 1, 2, 3 and 4 shows mean value and standard deviation obtained from 50 optimization experiments under different conditions. The bold numbers are the best results. Figures 3, 4, 5 and 6 give the graph of fitness that varies with the number of iterations of the algorithm.

Table 3 Experimental results of APSO algorithm (with 20 parameters)

Function	Algorithm	Mean value	standard deviation
Ackley	PSO	3.6826e+00	1.3178e+00
	Modified PSO	3.8137e+00	1.1510e+00
	DMS-PSO	2.0743e+00	**6.0338e-01**
	APSO	**8.9208e-01**	6.1125e-01
Griewank	PSO	1.3996e+00	5.0780e-01
	Modified PSO	1.2146e+00	2.1224e-01
	DMS-PSO	9.8281e-01	1.5742e-01
	APSO	**1.6080e-01**	**9.1931e-02**
Rastrigin	PSO	5.4742e+01	3.6460e+01
	Modified PSO	5.5111e+01	5.0702e+01
	DMS-PSO	**1.5541e+01**	5.6257e+00
	APSO	1.5949e+01	**3.8734e+00**
Schwefel_1	PSO	4.7744e+02	4.3588e+02
	Modified PSO	4.9840e+02	4.6327e+02
	DMS-PSO	9.4944e+01	1.2367e+02
	APSO	**3.3332e+00**	**2.5225e+00**
Schwefel_2	PSO	2.7028e+01	1.6637e+02
	Modified PSO	3.4453e+00	1.7581e+00
	DMS-PSO	1.1532e+00	5.4177e-01
	APSO	**1.6068e-01**	**1.0145e-01**
Sphere	PSO	6.2247e+01	1.0550e+02
	Modified PSO	2.8262e+01	3.8133e+01
	DMS-PSO	4.4149e+00	4.2799e+00
	APSO	**1.6330e-02**	**1.8642e-02**

Table 4 Experimental results of APSO algorithm (with 100 parameters)

Function	Algorithm	Mean value	standard deviation
Ackley	PSO	9.4062e+00	1.1857e+00
	Modified PSO	9.2800e+00	8.3564e-01
	DMS-PSO	6.7168e+00	1.1895e+00
	APSO	**5.6549e+00**	**5.9160e-01**
Griewank	PSO	4.6448e+01	2.2296e+01
	Modified PSO	3.6634e+01	1.3319e+01
	DMS-PSO	1.4523e+01	7.0111e+00
	APSO	**4.8244e+00**	**1.3231e+00**
Rastrigin	PSO	5.8426e+02	1.6175e+02
	Modified PSO	5.0562e+02	1.1786e+02
	DMS-PSO	3.8765e+02	8.0782e+01
	APSO	**3.7097e+02**	**3.5663e+01**
Schwefel_1	PSO	4.5318e+04	1.5387e+04
	Modified PSO	4.0529e+04	1.3670e+04
	DMS-PSO	2.1274e+04	1.0274e+04
	APSO	**8.1672e+03**	**1.8749e+03**
Schwefel_2	PSO	6.1721e+01	1.1088e+01
	Modified PSO	5.9783e+01	9.4852e+00
	DMS-PSO	3.2068e+01	7.9324e+00
	APSO	**2.3200e+01**	**3.5564e+00**
Sphere	PSO	5.0180e+03	4.1762e+03
	Modified PSO	4.0357e+03	1.6338e+03
	DMS-PSO	1.6395e+03	8.0491e+02
	APSO	**4.3810e+02**	**1.6359e+02**

From Tables 1, 2, 3, 4 and Figs. 3, 4, 5, 6 we can see that the convergence speed and convergence precision of APSO are totally better than those of other improved PSO algorithms. We notice that APSO does not perform as well as DMS-PSO in some special cases (such as Rastrigin function with 20 parameters), this may be due to some unstable factors caused by asynchronous updates. But in other cases, APSO performance is significantly better than other algorithms. This shows that our APSO algorithm is effective and can significantly improve the performance of PSO algorithm, which has practical significance and application value.

4.2 Experiments of APSO-BQSA algorithm

In this section, we test the performance of our APSO-BQSA algorithm in two standard reinforcement learning problems: cliff-walking problem and frozen-lake

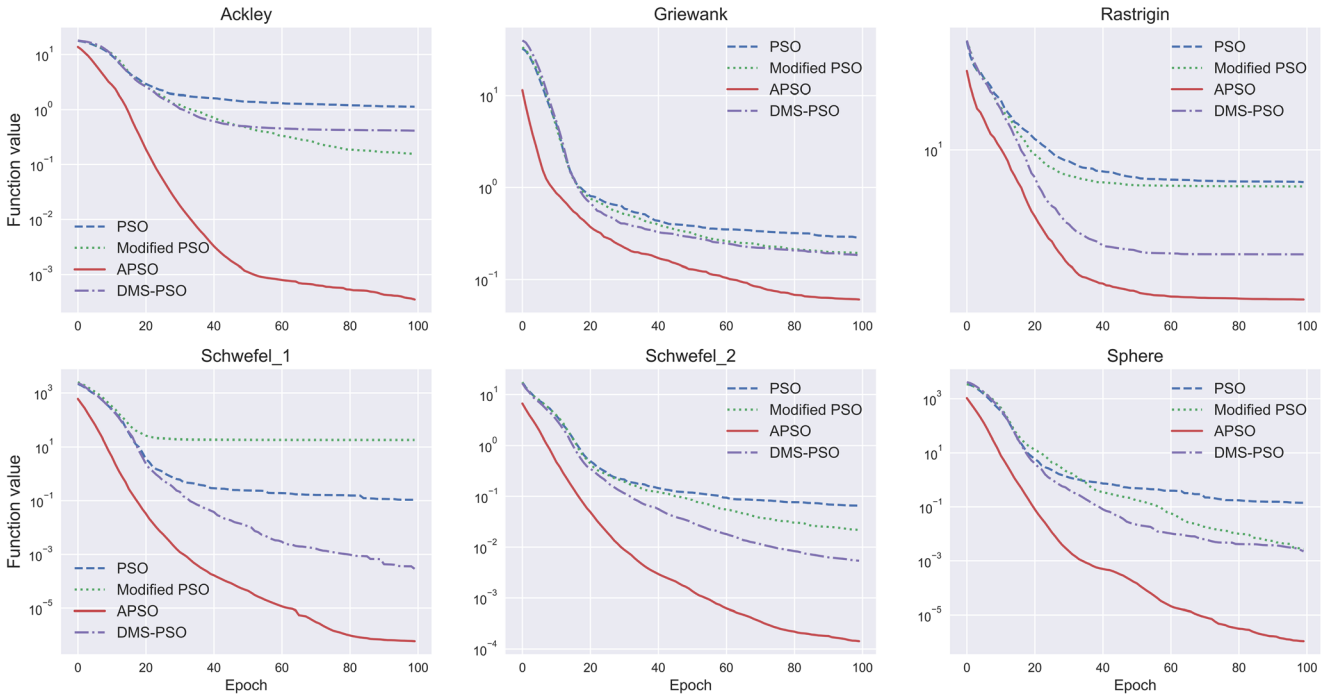


Fig. 3 Result of APSO and PSO (with 5 parameters)

problem. Cliff-walking problem [1] is a standard episodic reinforcement learning task. As show in Fig. 7, agent starts with the transition 'S', and its goal is reach 'G' successfully. The actions that the agent can choose are up, left, down and right. When the agent reaches the region marked "The Cliff", it will get a reward of −100, and be sent to the start instantly. In other transitions, the reward is −1.

We chose six typical existing reinforcement learning algorithms, including Q-learning, Sarsa,

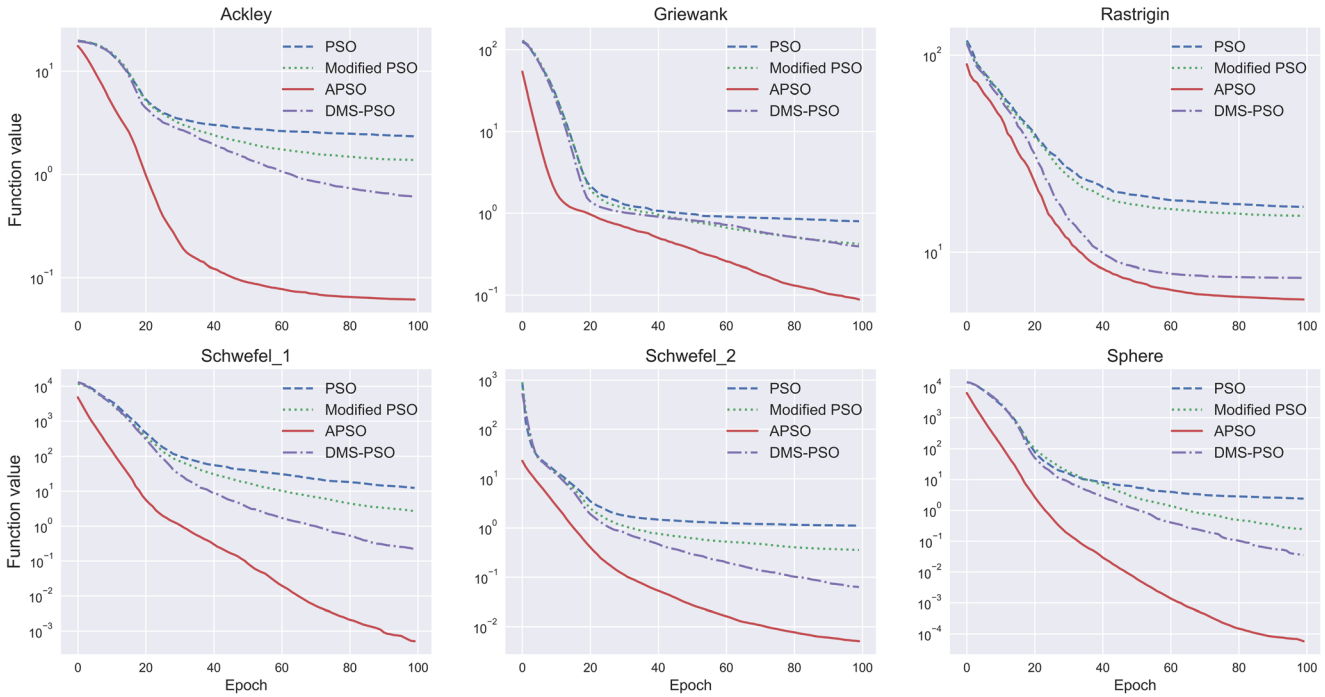


Fig. 4 Result of APSO and PSO (with 10 parameters)

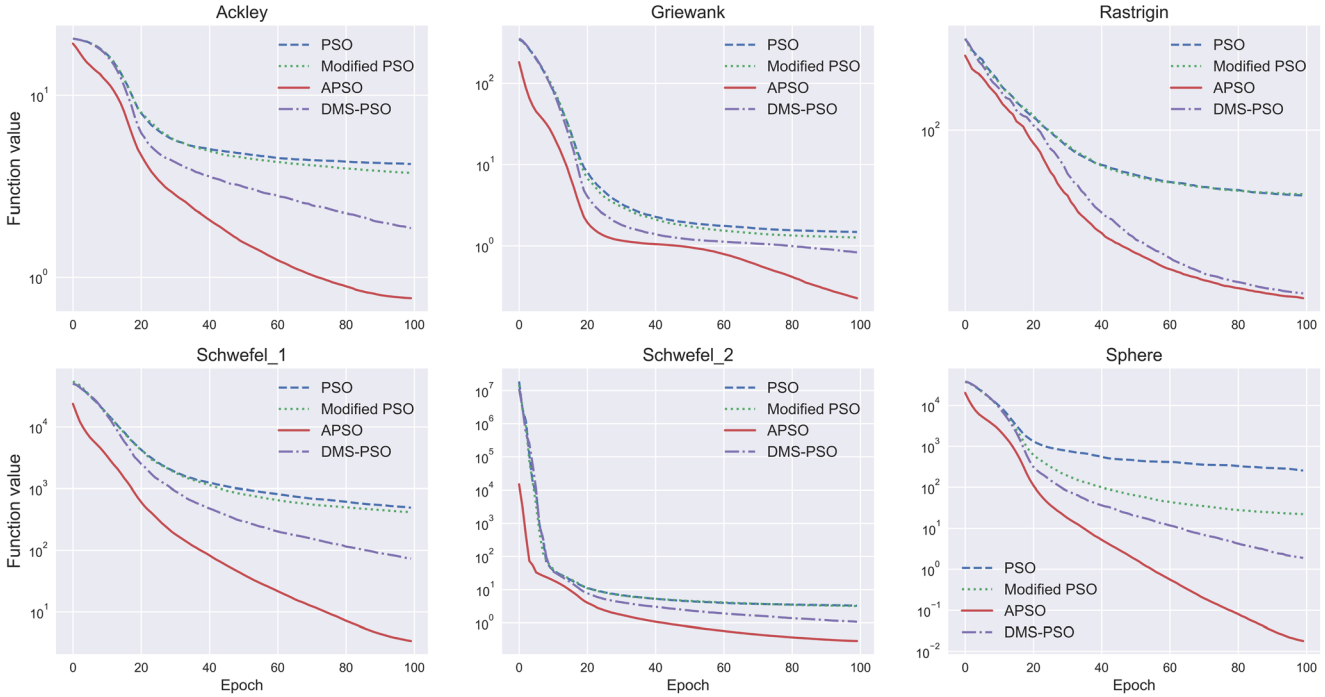


Fig. 5 Result of APSO and PSO (with 20 parameters)

Q_VDBE [18], BQSA, Dyna-Q and Asynchronous Q-learning, to compare with our APSO-BQSA algorithm. We set the learning rate for all algorithms as 0.1 and the discount factor $\lambda = 0.9$. All algorithms used $\varepsilon-$greedy policy with $\varepsilon = 0.01$. The max step for an episodes we set is 200. In APSO-BQSA, we set one thread as the BQSA thread and three threads as APSO threads. Each APSO thread has 10 particles. Table 5 gives the mean episodes that seven algorithms need for finding the best solution and their standard deviation in ten experiments. The bold numbers are the best results. Figure 8 gives the learning curves of seven algorithms. The cumulative rewards are smoothed by smoothing factor of 0.01.

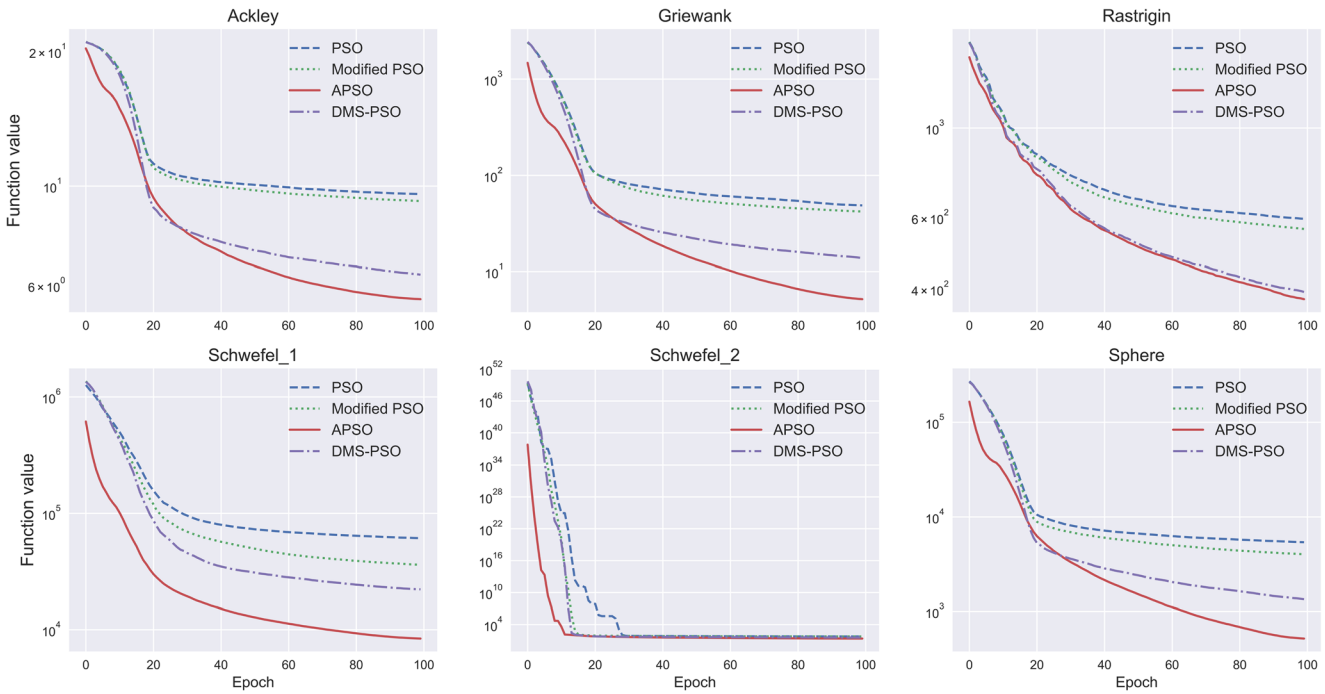


Fig. 6 Result of APSO and PSO (with 100 parameters)

Fig. 7 The cliff-walking problem

R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1
R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1
R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1	R= -1
S	The Cliff (R = -100)								G

From Table 5, we can see that the convergence speed of our APSO-BQSA algorithm is obviously better than that of other algorithms. It shows that the model-based method Dyna-Q can solve the cliff-walking problem faster than existing reinforcement learning algorithms, and the introduction of asynchronous PSO can significantly improve the convergence speed of algorithm, makes the algorithm runs faster even more than the model-based method. Figure 8 shows that APSO-BQSA has obvious advantages over other algorithms in terms of convergence speed. Because of the combination of Sarsa algorithm and backward Q-learning algorithm, BQSA algorithm has a good performance in single-threaded model-free algorithm. However, it uses the continuous states of an agent to update the parameters, and cannot eliminate the correlation of different states, and its performance needs to be further improved. APSO-BQSA algorithm overcomes the shortcoming of BQSA algorithm, and further improves the convergence performance of the algorithm through parameter search. This shows that the APSO-BQSA algorithm is effective and can greatly enhance the performance of the algorithm.

We then tested our algorithm in frozen-lake problem. Frozen-lake problem is a standard discrete space path planning problem. We take 4×4 frozen-lake problem provided by OpenAI Gym [19] as example. There are sixteen grids in the environment. The agent start with the upper left corner, and its goal is successfully reached the lower right corner. There exist four holes where the ice has melted. If stepping into one of those holes, our exploration will fail immediately. The experiment setting is that when we achieve the goal, we receive a reward of 10, and when we fall into the ice hole, we get a reward of −10. Figure 9 shows the frozen-lake environment.

In this experiment, we also use the above five algorithms to compare with our APSO-BQSA algorithm. The setup of experimental parameters is consistent with the previous experiment. Table 6 gives the mean episodes that seven algorithms need for finding the best solution and their standard deviation in ten experiments. The bold numbers are the best results. Figure 10 give the learning curves of seven algorithms.

From Table 6, we can see that in this environment, the asynchronous method has obvious advantages over other methods, and can significantly improve the convergence speed of the algorithm. It shows that the convergence speed of our APSO-BQSA algorithm is obviously better than that of other algorithms. Figure 10 shows that APSO-BQSA can also find the best solution very fast in this environment. It shows that APSO-

Table 5 Episodes required for seven algorithms to solve cliff-walking problem

Algorithm	Episode
Q-learning	191 ± 25
Sarsa	177 ± 26
Q_VDBE	159 ± 21
BQSA	125 ± 21
Asynchronous Q(with 4 threadings)	112 ± 34
Dyna-Q	61 ± 23
APSO-BQSA(with 4 threadings)	**12 ± 3**

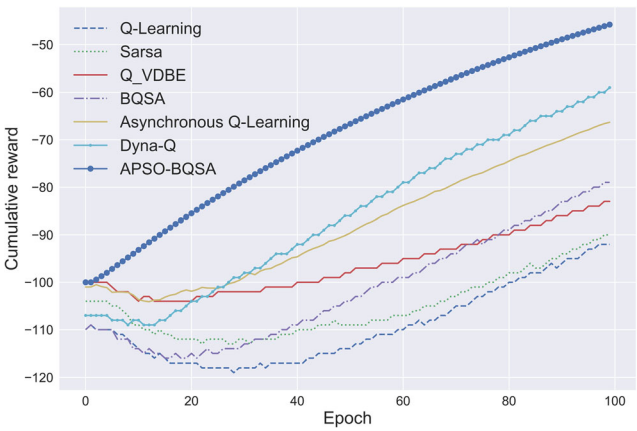


Fig. 8 Learning curves of seven algorithms in cliff-walking problem

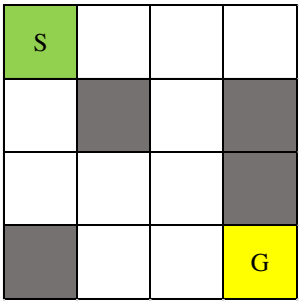


Fig. 9 The frozen-lake problem

BQSA has generality and can solve specific problems in different settings. This indicates that we can use the APSO-BQSA algorithm to solve problems in practical applications.

5 Conclusions

In this paper, we first proposed a new asynchronous multi-threading parallel PSO algorithm: APSO algorithm. APSO absorbs the advantages of simple and easy implementation of PSO algorithm, and overcomes the problem that the traditional PSO algorithm is easy to fall into local minimum. Experiments results show that in almost any case, the performance of APSO is better than traditional PSO algorithm. We then apply APSO in asynchronous reinforcement learning algorithm, and proposed a new asynchronous reinforcement learning named APSO-BQSA. APSO-BQSA uses APSO algorithm to optimize the parameters while using the improved BQSA to update the parameters. The experiments show that APSO-BQSA has faster convergence speed and better convergence performance than the existing reinforcement learning algorithm, and can be applied to different problems. However, we notice that the APSO-BQSA algorithm has some details that can be improved, such as how to make the APSO thread communicate with the BQSA thread better, which is our future research work. Moreover, we will combine our algorithm with some model-based algorithms, to make this architecture more general.

Table 6 Episodes required for seven algorithms to solve frozen-lake problem

Algorithm	Episode
Q-learning	28 ± 5
Q_VDBE	24 ± 8
Sarsa	20 ± 9
BQSA	19 ± 4
Dyna-Q	12 ± 3
Asynchronous Q (with 4 threadings)	11 ± 5
APSO-BQSA (with 4 threadings)	**6 ± 3**

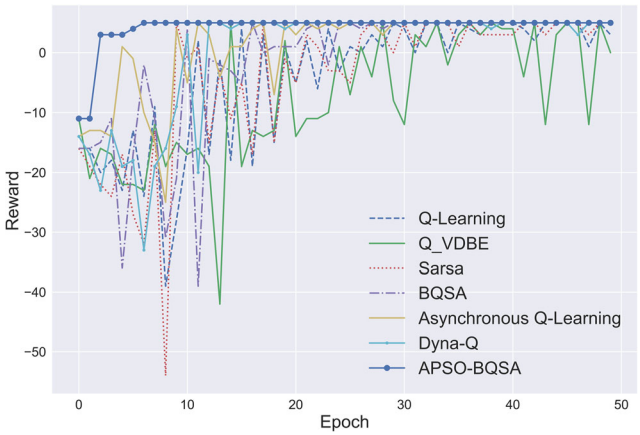


Fig. 10 Learning curves of seven algorithms in frozen-lake problem

Acknowledgments This work is supported by the Fundamental Research Funds for the Central Universities (No.2017XKZD03).

References

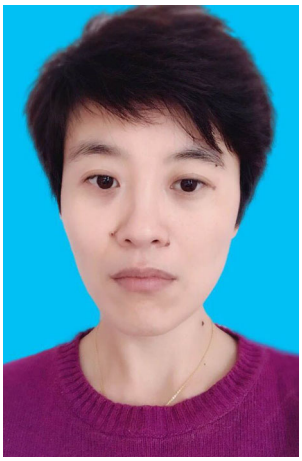
1. Sutton R, Barto A (1998) Reinforcement learning: An introduction. MIT press, Cambridge
2. Mnih V, Kavukcuoglu K, Silver D et al (2013) Playing atari with deep rein-forcement learning. Proceedings of Workshops at the 26th Neural Information Pro-cessing. Systems Lake Tahoe, USA, pp 201–220
3. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
4. Silver D, Huang A, Maddison M et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587): 484–489
5. Mnih V, Badia A, Mirza M et al (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, pp 1928–1937
6. Zhao X, Ding S, An Y, Jia W (2018) Asynchronous reinforcement learning algorithms for solving discrete space path planning problems. Appl Intell 48(12):4889–4904
7. Zhao X, Ding S, An Y (2018) A new asynchronous architecture for tabular re-inforcement learning algorithms. In: Proceedings of the eighth international conference on extreme learning machines, pp 172–180
8. Wang Y, Li T, Lin C (2013) Backward Q-learning: the combination of Sarsa algorithm and Q-learning. Eng Appl Artif Intell 26(9): 2184–2193
9. Kennedy J, Eberhart R (1995) Particle swarm optimization. ICNN 1942–1948
10. Watkins C (1989) Learning from delayed rewards. Robot Auton Syst 15(4):233–235
11. Rummery G, Niranjan M (1994) On-line Q-learning using connectionist systems. University of Cambridge, Department of Engineering

12. Sutton R (1988) Learning to predict by the methods of temporal differences. Mach Learn 3(1):9–44
13. Singh S, Sutton R (1996) Reinforcement learning with replacing eligibility traces. Mach Learn 22(1-3):123-158
14. Silver D, Lever G, Heess N et al (2014) Deterministic policy gradient algorithms. In: Proceedings of the 31st international conference on machine learning, pp 387–395
15. Schulman J, Levine S, Abbeel P et al (2015) Trust region policy optimization. In: Proceedings of the 32nd international conference on machine learning, pp 1889–1897
16. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. IEEE CEC 69–73
17. Liang J, Suganthan PN (2006) Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. 2006 IEEE congress on. Evol Comput:9–16
18. Tokic M, Palm G (2011) Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In: Annual conference on artificial intelligence. Springer, Berlin, Heidelberg, pp 335–346
19. Greg Brockman, Vicki Cheung, Ludwig Pettersson, et al. OpenAI Gym. arXiv preprint arXiv: 1606.01540 (2016)

Shifei Ding received his Ph.D degree from Shandong University of Science and Technology in 2004. He received postdoctoral degree from Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, and Chinese Academy of Sciences in 2006. He is a professor and Ph.D supervisor at China University of Mining and Technology, His research interests include artificial intelligence, pattern recognition, machine learning, data mining, and granular computing et al.

Wei Du received his B.Sc. degree in mathematics from Shandong University in 2016. He is currently a graduate student now studying at School of Computer Science and Technology, China University of Mining and Technology, and his supervisor is Prof. Shifei Ding. His research interests include machine learning, and reinforcement learning et al.

Xingyu Zhao received his B.Sc. degree in computer science from China University of Mining and Technology in 2016. He is currently a graduate student now studying at School of Computer Science and Technology, China University of Mining and Technology, and his supervisor is Prof. Shifei Ding. His research interests include machine learning, and reinforcement learning et al.



Lijuan Wang received her M.Sc. degree in computer science from Dongbei University in 2013. She is currently a Ph.D graduate student now studying at School of Computer Science and Technology, China University of Mining and Technology, and her supervisor is Prof. Shifei Ding. Her research interests include machine learning, and clustering analysis et al.

Weikuan Jia received Ph.D degree from Jiangsu University in 2016. He is currently an associate professor at School of Information Science and Engineering, Shandong Normal University. His research interests include machine learning, and data mining et al.