Computer Science and Information Systems 14(3):789–804 https://doi.org/10.2298/CSIS170107029Z

A kernel based true online Sarsa(λ) for continuous space control problems

Fei Zhu[1,2], Haijun Zhu[1], Yuchen Fu[3], Donghuo Chen[1], and Xiaoke Zhou[4]

[1] School of Computer Science and Technology, Soochow University
Shizi Street No.1 158 box, 215006, Suzhou, Jiangsu, China
zhufei@suda.edu.cn, 1017942265@qq.com, dhchen@suda.edu.cn
[2] Provincial Key Laboratory for Computer Information Processing Technology, Soochow University
Shizi Street No.1 158 box, 215006, Suzhou, Jiangsu, China
[3] School of Computer Science and Engineering, Changshu Institute of Technology
yuchenfu@suda.edu.cn
[4] University of Basque Country, Spanish
xzhou001@ikasle.ehu.eus

Abstract. Reinforcement learning is an efficient learning method for the control problem by interacting with the environment to get an optimal policy. However, it also faces challenges such as low convergence accuracy and slow convergence. Moreover, conventional reinforcement learning algorithms could hardly solve continuous control problems. The kernel-based method can accelerate convergence speed and improve convergence accuracy; and the policy gradient method is a good way to deal with continuous space problems. We proposed a Sarsa(λ) version of true online time difference algorithm, named True Online Sarsa(λ)(TOSarsa(λ)), on the basis of the clustering-based sample specification method and selective kernel-based value function. The TOSarsa(λ) algorithm has a consistent result with both the forward view and the backward view which ensures to get an optimal policy in less time. Afterwards we also combined TOSarsa(λ) with heuristic dynamic programming. The experiments showed our proposed algorithm worked well in dealing with continuous control problem.

Keywords: reinforcement learning, kernel method, true online, policy gradient, Sarsa(λ).

1. Introduction

Reinforcement learning (RL) is an extremely important class of machine learning algorithm [15]. The agent of reinforcement learning keeps continuous interaction with the unknown environment, and receives feedback, usually called reward, from the environment to improve the behavior of agents so as to form an optimal policy [8]. Reinforcement learning maps the state of the environment to the action of the agent: the agent selects an action, the state changes, and the environment gives an immediate reward as an excitation signal. The goal of intensive learning is to get a maximum long-term cumulative reward from the environment, called return. As a kind highly versatile machine learning framework, reinforcement learning has been extensively studied and applied in many domains, especially in control tasks [14][1][19][6].

In many practical applications, the tasks that have to be solved are often with continuous space problems, where both the state space and the action space are continuous. Most common methods of solving continuous space problems include value function methods [13] and policy search methods [2]. The policy gradient method [16] is a typical policy search algorithm which updates policy parameters in the direction of maximal long-term cumulative reward or the average reward and gets optimal policy distribution. The policy gradient method has two parts: policy evaluation and policy improvement. Reinforcement learning has many fundamental algorithms for policy evaluation is concerned, such as value iteration, policy iteration, Monte Carlo and the time difference method (TD) [9] where the time difference method is an efficient strategy evaluation algorithm. Both the value function in the policy evaluation and the policy function in the policy improvement require function approximation [3]. The policy evaluation and policy improvement of the policy gradient method can be further summarized as the value function approximation and the policy function approximation. In reinforcement learning algorithms, the approximation of the function can be divided into parametric function approximation where the approximator and the number of parameters need to be predefined, and nonparametric function approximation where the approximator and the number of parameters are determined by samples. So nonparametric function approximation has high flexibility, and has better generalization performance. Gaussian function approximation and kernel-based method are nonparametric function approximation methods.

Although conventional reinforcement learning algorithms can deal with online learning problems, most of them have low convergence accuracy and slow convergence speed. The kernel based method is nonparametric function approximation method, and its approximation value function or strategy can alleviate the above problem of reinforcement learning. The policy gradient is an efficient way to deal with continuous space problems. In this paper, we propose an online algorithm that is based on kernel-based policy gradient method to solve continuous space problem. In the Section 2, we introduce the related work, including Markov decision process, reinforcement learning, and policy gradient; in the Section 3, we state how forward view matches backward view; in the Section 4, we introduce a true online time difference algorithm, named TOSarsa(λ); in the Section 5, we combine TOSarsa(λ) with heuristic dynamic programming.

2. Related Work

2.1. Markov Decision Process

Markov Decision Process (MDP) [5] is one of the most influential concepts in reinforcement learning. Markovian property refers that the development of a random process has nothing to do with the history of observation and is only determined by the current state. The state transition probability with a Markovian stochastic process [12] is called the Markov process. By Markov process, a decision is made in accordance with the current state and the action set, affecting the next state of the system, and the successive decision will be determined with the new state.

Normally a Markov Decision Process model can be represented by a tuple $M = < S, A, f, r >$, where:

S is the state space, and $s_t \in S$ denotes the state of the agent at time t;

A is the action space, and $a_t \in A$ denotes the action taken by the agent at time t;

f: $S \times A \longrightarrow [0,1]$ is the state transfer function, and f is usually formalized as the probability of the agent taking action $a_t \in A$ and transferring from the current state $s_t \in S$ to the next state $s_{t+1} \in S$;

ρ: $S \times A \longrightarrow$ R is the reword function which is received when the agent takes action $a_t \in A$ at the state $s_t \in S$ and the state transfers to the next state $s_{t+1} \in S$.

A Markov decision process is often used to model the reinforcement learning problem.

2.2. Reinforcement Learning

Reinforcement learning is based on the idea that the system learns directly from the interaction during the process of approaching the goals. The reinforcement learning framework has five fundamental elements: agent, environment, state, reward, and action, showed as Fig. 1. In the reinforcement learning, an agent, which is also known as a controller, keeps interaction with the environment, generates a state $s_t \in S$, and chooses an action $a_t \in A$ in accordance with a predetermined policy π such that at $a_t = \pi(s_t)$. Consequently, the agent receive an immediate reward $r_{t+1}=\rho(s_t, a_t)$ and gets to a new state s_{t+1}. By continuous trails and optimizing, the agent gets the maximal sum of the rewards as well as an optimal action sequence.

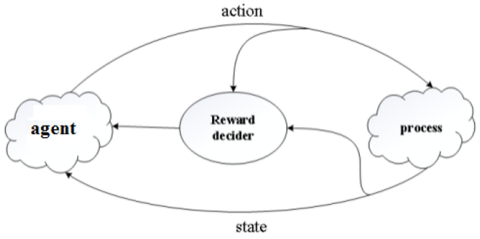

Fig. 1. Framework of reinforcement learning. The agent selects an action; the environment responds to the action, generates new scenes to the agent, and then returns a reward.

The goal of reinforcement learning is to maximize a long-term reward R which is calculated by:

$$\begin{aligned} R &= \mathrm{E}^{\pi}\left\{r_1 + \gamma r_2 + \cdots + \gamma^{T-1} r_T + \cdots\right\} \\ &= \mathrm{E}^{\pi}\left\{\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right\} \end{aligned} \tag{1}$$

where E^{π} is expectation of accumulation of the long term reward, and $\gamma \in (0,1]$ is a discount factor increasing uncertainty on future rewards showing how far sighted the controller is in considering the rewards.

Reinforcement learning algorithms use state value function $V(s)$ to represent the expected rewards of state s under policy π. The value function $V(s)$ is defined as [15]:

$$
\begin{aligned}
V(s) &= \mathrm{E}^{\pi}\left\{\sum_{i=1}^{\infty}\gamma^{i-1}r_{t+i}|s_t = s\right\} \\
&= \mathrm{E}^{\pi}\left\{r_{t+1} + \gamma\sum_{i=1}^{\infty}\gamma^{i-1}r_{t+i+1}|s_t = s\right\} \\
&= \sum_{t=1}^{\infty}\gamma_t r(s_t)
\end{aligned} \tag{2}
$$

Reinforcement learning algorithms also use state state-action function $Q(s,a)$ which represents the accumulated long-term reward from a starting state. State-action function $Q(s,a)$ is defined as[15]:

$$
\begin{aligned}
Q(s,a) &= \mathrm{E}^{\pi}\left\{\sum_{i=1}^{\infty}\gamma^{i-1}r_{t+i}|s_t = s, a_t = a\right\} \\
&= \mathrm{E}^{\pi}\left\{r_{t+1} + \gamma\sum_{i=1}^{\infty}\gamma^{i-1}r_{t+i+1}|s_t = s, a_t = a\right\} \\
&= \sum_{t=1}^{\infty}\gamma_t r(s_t, a_t)
\end{aligned} \tag{3}
$$

Despite that the state value function $V(s)$ and the state action value function $Q(s, a)$ represent long-term returns, they still can be expressed in a form that is relevant to the MDP model and the successive state or state action pair, called one step dynamic. In this way, it is not necessary to wait for the end of the episode to calculate the value of the corresponding value function, but to update the new value function in each step, so that the algorithm has the ability real-time online learning. In addition, the state value function and the state action value function also can be expressed as:

$$
V(s) = \int_{a\in A}\pi(a|s)\,Q(s,a)da \tag{4}
$$

$$
Q(s,a) = \int_{s'\in S} f(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]ds \tag{5}
$$

As we can see that in the case where the environment model is completely known, the state value function and the state action value function can be transferred to each other seamlessly.

2.3. Policy Gradient

The reinforcement learning method can be categorized as the value function method and the policy gradient method. The typical value function methods include value iteration,

the policy iteration, the Q learning [10], Sarsa algorithm [11] and LSPI algorithm [7]. The policy iteration algorithm computes the optimal policy by repeating policy evaluating and policy improving. The value function method is a generalized iterative algorithm, focusing on the solution of the state action of the value function, and then the strategy is calculated by the value function, commonly by greedy strategy. Unlike the value function method, the policy gradient method represents the strategy directly through a set of policy parameters, rather than indirectly through the value function. The policy gradient method maximizes the cumulative reward function or the average reward by the gradient method to find out the optimal policy parameters, and each update is along the fastest rising direction of the reward function. The updates of policy parameters can be denoted as:

$$\psi = \psi + \alpha \frac{\partial Q(s, a_\psi)}{\partial u_\psi} \frac{\partial u_\psi}{\partial \psi} \tag{6}$$

$$\psi = \psi + \alpha \frac{\partial R}{\partial \psi} \tag{7}$$

The gradient becomes zero when the reward function reaches the local optimal point. The core of the policy gradient method update is the solution of the gradient.

The updates of policy parameters in the policy gradient method can be categorized as deterministic policy and non-deterministic policy. A deterministic policy is a greedy strategy that can deal with continuous action space problems. Because reinforcement learning requires action exploration, deterministic policy cannot be applied individually to reinforcement learning, often with some other method such as ε-greedy method. The non-deterministic policy gradient can solve both discrete and continuous space problems, just being provided with strategy distribution in advance. The Gibbs distribution is often used for discrete space problems as:

$$\pi\left(a|s\right) = \frac{e^{\kappa(s,a)^{\mathrm{T}}\psi}}{\sum\limits_{a' \in A} e^{\kappa(u,a')^{\mathrm{T}}\psi}} \tag{8}$$

While continuous space problem often takes advantage of Gaussian distribution, as:

$$\pi(a|s) = \frac{1}{\sqrt{2\pi\sigma^2(s)}} \exp\left(-\frac{\left(a - \mu(s)\right)^2}{2\sigma^2(s)}\right) \tag{9}$$

$$\mu(s) = \kappa_\mu^\top(s)\psi_\mu \tag{10}$$

$$\sigma(s) = \kappa_\sigma^\top(s)\kappa_\sigma \tag{11}$$

where κ(*s*,*a*) is the kernel of the state action pair (*s*, *a*), μ (s) is the mean value of the Gaussian distribution, $\sigma(s)$ is the standard deviation of the Gaussian distribution, $\psi = \left(\psi_\mu^\top, \psi_\sigma^\top\right)^\top$ is the parameter vector, and $\kappa(s) = \left(\kappa_\mu^\top(s), \kappa_\sigma^\top(s)\right)^\top$ is the kernel vector.

However, policy gradient algorithms are often suffered from the disadvantage brought by large gradient variance, which will affect the algorithm learning speed and convergence

performance. Therefore, in practice, the natural gradient method is used to replace the gradient method, so as to reduce the variance of the gradient, speed up the convergence rate of the algorithm and improve the convergence performance of the algorithm.

3. Forward View and Backward View

As the most important part of the reinforcement learning method, the time difference (TD) method is an effective method to solve the long-term forecasting problem. However, the traditional TD methods have problems in matching forward view and backward view. In this section, we will state how to make the forward view equivalent to backward view, which is a very important foundation of the proposed algorithms.

3.1. Time Difference (TD)

TD method is one of the core algorithms of reinforcement learning. TD method, which is able to learn directly from the raw experience from an unknown environment and update the value function at any time without determining dynamic model of environment in advance. Temporal difference combines the advantages of Monte Carlo method and dynamic programming. It updates the model by estimation based on part of learning rather than final results of the learning. Temporal difference works very well in dealing with real time prediction problems and control problems. Temporal difference learning updates by [15]:

$$V\left(s_{t+1}\right) \leftarrow V\left(s_t\right) + \alpha\left[R_t - V\left(s_t\right)\right] \tag{12}$$

$$Q\left(s_{t+1}, a\right) \leftarrow Q\left(s_t, a\right) + \alpha\left[R_t - Q\left(s_t, a\right)\right] \tag{13}$$

where R_t is return of step t, α is a step size parameter. Temporal difference learning updates V or Q in step $t+1$ using the observed reward r_{t+1} and estimated $V(s_{t+1})$ $Q(s_{t+1}, a_{t+1})$or .

One simple form of time difference algorithm, TD(0), updates the value function using the estimated deviation of a state s at the two time points, before and after. As TD (0) algorithm updates the value function every step, rather than after all steps, the entire update process does not require environment information as many other algorithms do. This advantage of TD(0) algorithm makes it suitable for the online learning task under the unknown environment. In addition, as the value function updating of TD(0) doesn't need to wait until the end of the episode, TD(0) can actually be used for non-episodic tasks, which sharply widens its application range compared to the Monte Carlo algorithm. The TD (0) algorithm is a method of evaluating the strategy. The Q learning algorithm and Sarsa algorithm are the two forms of TD(0).

3.2. TD(λ)

Inspired by the Monte Carlo algorithm, researchers introduced the idea of n-step updating and applied it to time difference. The update of the current value function that is based

on the next state value function and the immediate reward is called a one-step update. Likewise, it is referred as *n*-step update if the update is based on the next *n* steps. The *n*-step update can be defined as:

$$R_{t,\omega}^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \omega^\top \kappa\left(s_{t+n}\right) \tag{14}$$

As the *V* function value of the current state *s* has a variety of estimates, in the process of algorithm implementation, we often uses weighted average of the different *n* steps, which is called λ-return:

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_{t,\omega}^{(n)} + \lambda^{T-t-1} R_{t,\omega}^{(T-t)} \tag{15}$$

where λ is regarded as recession factor, and *T* is the maximum number of steps. It is called λ-return algorithm when using λ-return to update the current state value function:

$$\omega_{t+1} = \omega_t + \alpha \left[R_t^\lambda - \omega^\top \kappa\left(s_t\right)\right] \kappa(s_t) \tag{16}$$

In the reinforcement learning, the above stated view is called forward view. The λ-return algorithm cannot update value function until the end of the episode. Therefore, λ-return algorithm uses the backward view to update the value function, which employs current TD error to update the value function of all states.

TD(λ) introduced the concept of the eligibility trace in backward view. The eligibility trace is essentially a record of the state or state of action recently visited. The cumulative eligibility trace can be defined as:

$$\mathbf{e}_t = \lambda\gamma \mathbf{e}_{t-1} + \kappa(s_t) \tag{17}$$

In the backward view, the TD error δ_t is updated according to the eligibility trace for all state values, as:

$$\omega_{t+1} = \omega_t + \alpha \delta_t \mathbf{e_t} \tag{18}$$

The conventional forward calculates λ return R_t^λ until the end of episode, while the online forward view method is able to calculate λ return at time *t*. This is called a truncated return, as:

$$R_t^{\lambda|t'} = (1-\lambda) \sum_{n=1}^{t'-t-1} \lambda^{n-1} R_{t,\omega_{t+n-1}}^{(n)} + \lambda^{t'-t-1} R_{t,\omega_{t'-1}}^{(t'-t)} \tag{19}$$

4. TOSarsa(λ) Algorithm

In the previous section, we have introduced how to achieve equivalence between forward view and backward view as well as its benefit of doing so. In this section, we will introduce a true online time difference algorithm which uses a clustering-based sample sparsification method [20] and selective kernel-based value function [4] as value function representation.

4.1. TOSarsa(λ) Algorithm Description

The true online time difference algorithm, named True Online State-action-reward-state-action(λ) (TOSarsa(λ)) is based on the effective Sarsa(λ) algorithm and uses Equation (17) as basic form of update equation to calculate eligibility trace, Equation (18) to calculate TD error and Equation (19) to calculate return.

Algorithm 1 True Online State-action-reward-state-action(λ) (TOSarsa(λ))

Input: policy, threshold
Output: optimal policy

Initialize kernel function $\kappa(\cdot,\cdot)$
Initialize sample set S
Set up data dictionary $\boldsymbol{D}$
repeat
  Initialize starting state s_0
  Initialize eligibility trace $\mathbf{e} \leftarrow 0$
  $V(s) \leftarrow \omega^\top \kappa(s)$
  repeat
    $V(s_{t+1}) \leftarrow \omega^\top \kappa(s_{t+1})$
    a $\leftarrow \pi(a|s)$
    Observe r, s
    $\delta_t \leftarrow r_{t+1} + \gamma \omega_t^\top \kappa(s_{t+1}) - \omega_{t-1}^\top \kappa(s_t)$
    $\mathbf{e}_t \leftarrow \gamma\lambda \mathbf{e}_{t-1} + \alpha_t \kappa(s_t) - \alpha_t \gamma\lambda \left[\mathbf{e}_{t-1}^\top \kappa(s_t)\right] \kappa(s_t)$
    $\omega_{t+1} \leftarrow \omega_t + \delta_t \mathbf{e}_t + \alpha_t \left[\omega_{t-1}^\top \kappa(s_t) - \omega_t^\top \kappa(s_t)\right] \kappa(s_t)$
    $\xi \leftarrow \min_{s_i \in D} (\kappa(s,s) + \kappa(s_i,s_i) - 2\kappa(s,s_i))$
    Update $\boldsymbol{D}$
    if ξ is greater than a predefined threshold **then**
      $V(s_t) \leftarrow \omega^\top \kappa(s_{t+1})$
      Get ω and **e**
    else
      $V(s_t) \leftarrow V(s_{t+1})$
    end if
    $s_t \leftarrow s_{t+1}$
  until all step of the current episode end
until all episodes end
return optimal policy

4.2. Mountain Car Problem

Mountain car problem [18] is a classic problem in strengthening learning, as shown in Fig. 2. The task of the car is to get to the top of the mountain, the right side of the "star" mark position, as soon as possible. However, as the car is short of power, it is unable to drive to the top of the mountain directly. It has to accelerate back and forth many times to reach a higher position, and then accelerated to reach the end.

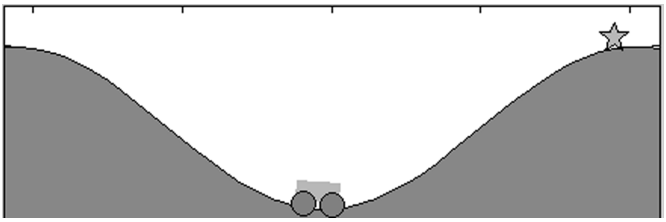

Fig. 2. Diagram of mountain car problem. The task of the car is to get to the top of the mountain, the right side of the "star" mark position, as soon as possible.

We use MDP to model mountain car problem. In the mountain car problem, the state contains two dimensions, the position denoted by p and the speed denoted by *v*. Then state of the car can be represented by a vector $\mathbf{x} = \begin{bmatrix} p \\ v \end{bmatrix}$. The acceleration of the car is in the range of -1 to 1, that is, the action $a \in$ [-1,1]. The curve of the road surface can be expressed by the function

$$h = sin(3p) \tag{20}$$

The state transition function can be expressed as

$$v_{t+1} = \textbf{\textit{bound}}[v_t + 0.001u_t - 0.0025\cos(3p_t)] \tag{21}$$

$$p_{t+1} = \textbf{\textit{bound}}[p_t + 1] \tag{22}$$

where ***bound*** is a function used to limited the value, ***bound***$(v_t) \in$ [-0.07,0.07], ***bound***$(p_t) \in$ [-1.5, 1.5]. The coefficient of gravity acceleration direction is -0.0025.

Sarsa is an effective TD algorithm for control problems. We implemented the Sarsa version of the TOSarsa(λ) algorithm and compared with Sarsa and Sarsa(λ). Fig. 3 shows the control effect of the three algorithms on the initial state's value function.

As it can be seen from Fig 3, in the both initial stage and final stage after convergence, the algorithm TOSarsa(λ) was better than the other two algorithms, Sarsa and Sarsa(λ). The three algorithms are value function methods, and their control policy is directly related to the evaluation of the value function. From the approximation point of view, TOSarsa(λ) got to convergence earlier than the other two. In general, the three algorithms were all effective in dealing with the mountain car problem and TOSarsa(λ)

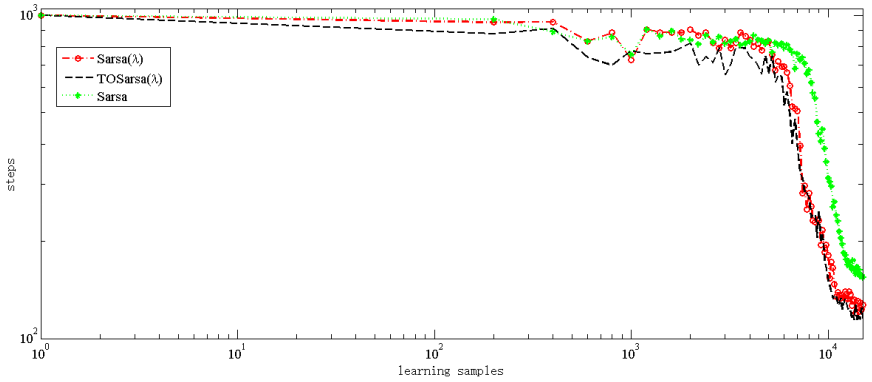

Fig. 3. The approximation effects of the algorithms on the initial state's value function on the initial state's value function.

which had a better strategy to evaluate performance was the best of three. However, all of the three algorithms had fluctuations at the beginning stage because at the initial stage, the data dictionary for the algorithms has not yet been completely established, and the algorithm kept exploration. We used TOSarsa(λ), Sarsa and Sarsa(λ) to solve the mountain car problem for 50 times. The results are shown in Fig. 4, where we can see that TOSarsa(λ) is the fastest in the three algorithms in the process of approximation. Fig. 4 shows the number of episodes required by the three algorithms, TOSarsa(λ), Sarsa and Sarsa(λ), to reach the target in different scenarios. TOSarsa(λ) was superior to the other two in the convergence rate and the convergence result. Moreover, the convergence result of TOSarsa(λ) was more stable.

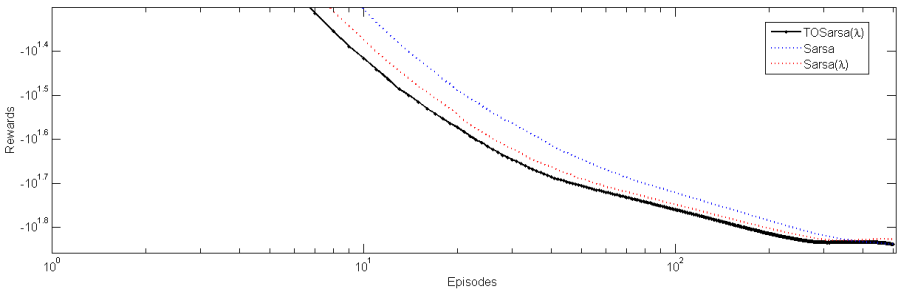

Fig. 4. The number of average steps of three algorithms. The abscissa represents the number of episodes and the ordinate shows the average number of steps.

5. TOSarsa(λ) With Heuristic Dynamic Programming

The dual heuristic dynamic programming (DHDP) algorithm is a method of dealing with continuous action space by neural network. It applies the actor-critics framework, evaluates the strategy in the critics section, and calculates deterministic strategies in the actors section. In this section, we will try to combine TOSarsa(λ) with heuristic dynamic programming.

5.1. TOSHDP Algorithm Description

The TOSarsa(λ) is used to evaluate the derivative of the value function to the state; update policy is updated by using the gradient descent method. The value of the function is:

$$\lambda(s_t) = \frac{\partial V(s_t)}{\partial s_t} = \omega^\top \kappa(s_t) \tag{23}$$

It satisfies the Bellman equation. We take TOSarsa (λ) method to get value of $\lambda(s_t)$, TD error, as:

$$\delta_t = \frac{\partial r_{t+1}}{\partial s_t} + \gamma \left(\frac{\partial s_{t+1}}{\partial s_t} + \frac{\partial s_{t+1}}{\partial a_t} \frac{\partial a_t}{\partial s_t} \right) \omega_t \kappa(s_{t+1}) - \omega_{t-1} \kappa(s_t) \tag{24}$$

As it can be seen from the above equation, the Equation(24) needs to solve $\frac{\partial s_{t+1}}{\partial s_t}$ and $\frac{\partial s_{t+1}}{\partial a_t}$, which requires a complete information of environment or model. The dual heuristic dynamic programming algorithm uses more environment knowledge and has a pretty good performance. In addition, the dual heuristic dynamic programming algorithm algorithm calculates the value of $\frac{\partial a_t}{\partial s_t}$, which is the actor part of the policy function of the derivative. The policy parameters updating as follows:

$$\begin{aligned} \omega_{t+1} &= \omega_t - \beta \Delta \omega_t \\ &= \omega_t - \beta \frac{\partial V(s_{t+1})}{\partial a_t} \frac{\partial a_t}{\partial \omega_t} \\ &= \omega_t - \beta \lambda(s_{t+1}) \frac{\partial s_{t+1}}{\partial u_t} \frac{\partial a_t}{\partial \omega_t} \\ &= \omega_t - \beta \lambda(s_{t+1}) \frac{\partial s_{t+1}}{\partial a_t} \kappa(s_t) \end{aligned} \tag{25}$$

where β is learning step for policy parameters. The following is the algorithm of TOSarsa(λ) with heuristic dynamic programming, where the 8^{th} step of the algorithm is the combination of optimal policy function and ε-greedy.

5.2. Cart Pole Balancing Problem

In this section, we verify the algorithm by cart pole balancing problem [17], which is a very classic continuous problem. There is a car on the horizontal track with a mass of

Algorithm 2 TOSarsa(λ) with heuristic dynamic programming (TOSHDP)))

Input: policy, threshold
Output: optimal policy

Initialize sample set S
Set up data dictionary $\boldsymbol{D}$
repeat
  Initialize starting state s_0
  Initialize eligibility trace $\mathbf{e} \leftarrow 0$
  $\lambda(s_t) \leftarrow \omega^\top \kappa(s_t)$
  repeat
    a $\leftarrow \pi(a|s)$
    Observe r, s
    $\lambda(s_{t+1}) \leftarrow \omega^\top \kappa(s_{t+1})$
    $\delta_t \leftarrow r_{t+1} + \gamma \left(\frac{\partial s_{t+1}}{\partial s_t} + \frac{\partial s_{t+1}}{\partial a_t} \frac{\partial a_t}{\partial s_t} \right) \lambda(s_{t+1}) - \lambda(s_t)$
    $\mathbf{e}_t \leftarrow \gamma\lambda \mathbf{e}_{t-1} + \alpha_t \kappa(s_t) - \alpha_t \gamma\lambda \left[\mathbf{e}_{t-1}^\top \kappa(s_t) \right] \kappa(s_t)$
    $\omega_{t+1} \leftarrow \omega_t - \beta\lambda(s_{t+1}) \frac{\partial s_{t+1}}{\partial a_t} \kappa(s_t)$
    $\xi \leftarrow \min_{s_i \in D} \left(\kappa(s, s) + \kappa(s_i, s_i) - 2\kappa(s, s_i) \right)$
    Update **D**
    if ξ is greater than a predefined threshold **then**
      $V(s_t) \leftarrow \omega^\top \kappa(s_{t+1})$
      Get ω and **e**
    else
      $V(s_t) \leftarrow V(s_{t+1})$
    end if
    $s_t \leftarrow s_{t+1}$
  until all steps of the current episode end
until all episode end
return optimal policy

m=1kg, the length l = 1m. The pole and the car are hinged together. The pole and the vertical direction are at an angle. In order to make the angle of the pole and the vertical direction in $[-36^\circ, 36^\circ]$, where the angle is negative if the pole is on the left side of the vertical line, and the angle is positive if the pole is on the right side of the vertical line. After each time interval Δt = 0.1s, a horizontal force F is applied to the cart, where F is within [-50N, 50N] (negative means the force is to the left, and positive is to the right), and there is a random noise disturbance between [-10N, 10N] when F is applied. All frictional forces were not considered. The task of the agent is to learn a policy so that the angle between the pole and the vertical direction is kept as much as possible in the specified range.

We use MDP to model the cart pole balancing problem. The state of the environment is represented by two variables α and β, where α is the angle formed by the pole and the vertical line, and β is the angular acceleration of the rod. The state space is:

$$S = \{(\alpha, \beta) | \alpha \in [-36^\circ, 36^\circ], \beta \in [-36^\circ, 36^\circ]\} \tag{26}$$

the action space is:

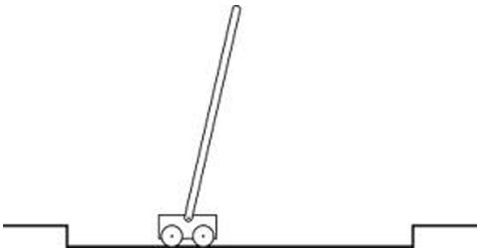

Fig. 5. Cart pole balancing problem diagram.

$$A = \{a|a \in [-50N, 50N]\} \tag{27}$$

Agent exerts force *F* on the cart, and the angular acceleration of the pole is:

$$\xi = \frac{g \sin\alpha + \cos\alpha \left(\frac{-f - ml\beta^2 \sin\theta}{m+M} \right)}{l \left(\frac{4}{3} - \frac{m\cos^2\alpha}{m+M} \right)} \tag{28}$$

where g is the constant of gravitational acceleration, with value 9.81 m/s^2; and f is the value of force *F*. After Δt , the states are $\alpha = \beta + \xi\Delta t$, $\beta = \alpha + \beta\Delta t$, and the reward function is

$$\rho(x,u) = \begin{cases} 1, \ |f(x,u)| < 36^\circ \\ -1, \ |f(x,u)| \geq 36^\circ. \end{cases} \tag{29}$$

The episode ends when the angle between the pole and the vertical line exceeds the given range. If the pole has not fallen and kept standing after 3000 time steps, it is regarded as a successful trial.

We compare TOSHDP with conventional DHP algorithm, where DHP uses two three-layer neural networks for value functions and policy approximation, all of their learning steps are 0.1. The results are shown in Fig. 6.

We can see from Fig. 6 that be seen from the convergence rate of the TOSHDP algorithm is higher than that of the conventional DHP algorithm in the same step size. The TOKDHP algorithm begins to converge at about 200 episodes, while the traditional DHP algorithm requires about 270 episodes to converge. There are mainly three factors caused this. First, kernel method is a more lightweight approximation algorithm than the neural network as the kernel method deals with the nonlinear problem directly by mapping and linear technique, while the neural network deals with the nonlinear problem through the multi-layer nonlinear transformation. Secondly, when the learning step is large, the neural network is easy to fall into the local optimal solution. Thirdly, our approach is more efficient in policy evaluation that was verified in the earlier experiment, resulting in an accelerated effect on the learning of the policy.

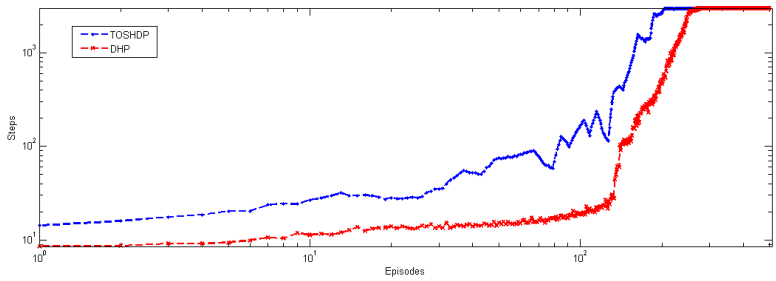

Fig. 6. Results of TOSHDP vs. conventional DHP algorithm where the value of the learning steps were set as 0.1.

6. Conclusion

We propose a true online kernel time difference algorithm, TOSarsa(λ), which employs a clustering-based sample sparsification method and selective kernel-based value function as value function representation. The experiment on mountain car problem showed our algorithm was effective in deal with the typical continuous problems and could speed up strategy search as well.

We combined the proposed TOSarsa(λ) algorithm with the dual heuristic dynamic programming algorithm to improve policy learning speed of policy search algorithms by replacing approximating using neural network method with approximating using kernel method. The experiment on cart pole balancing problem verified that our proposed algorithm really worked. It is a good alternative to deal with continuous action space problems. However, there is still some work to study further, such as how to extend the model to deal with the continuous space problems of unknown environment.

Acknowledgments. This paper is supported by National Natural Science Foundation of China (61303108, 61373094,61772355, 61702055,61602332), Jiangsu Province Natural Science Research University major projects (17KJA520004), Suzhou Industrial application of basic research program part (SYG201422), Provincial Key Laboratory for Computer Information Processing Technology of Soochow University (KJS1524), China Scholarship Council project (201606920013).

References

1. Al-Rawi A, Ng A, Y.A.: Application of reinforcement learning to routing in distributed wireless networks: a review. Artificial Intelligence Review 43(3), 381–416 (2015)
2. Bagnell A, Ng Y, S.J.: Policy search by dynamic programming. In: Advances in Neural Information Processing Systems. pp. 831–838 (2004)
3. Busoniu L, Babuska R, e.a.: Reinforcement learning and dynamic programming using function approximators. CRC Press (2010)
4. Chen X, Gao Y, W.R.: Online selective kernel-based temporal difference learning. IEEE transactions on neural networks and learning systems 24(12), 1944–1956 (2013)

5. E., B.: A markov decision process. Journal of Mathematical Fluid Mechanics 6(1), 65–73 (1957)
6. El I, Feng M, e.a.: Reinforcement learning strategies for decision making in knowledge-based adaptive radiation therapy: application in liver cancer. International Journal of Radiation Oncology Biology Physics 96(2), 38–45 (2016)
7. Ghorbani F, Derhami V, A.M.: Fuzzy least square policy iteration and its mathematical analysis. International Journal of Fuzzy Systems 19(13), 1–14 (2016)
8. H., V.H.: Reinforcement learning, chap. Reinforcement learning in continuous state and action spaces, pp. 207–251. Springer Berlin Heidelberg (2012)
9. K., D.: Reinforcement learning in continuous time and space. Neural Computation 12(1), 210–219 (2000)
10. Kiumarsi B, Lewis L, e.a.: Reinforcement q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. Automatica 50(4), 1167–1175 (2014)
11. Kober J, Bagnell A, P.J.: Reinforcement learning in robotics: a survey. The International Journal of Robotics Research 32(11), 1238–1274 (2013)
12. L, P.: Markov decision processes: discrete stochastic dynamic programming. John Wiley and Sons (2014)
13. M., H.: Value-function approximations for partially observable markov decision processes. Journal of Artificial Intelligence Research 13(1), 33–94 (2011)
14. Scholkopf B, Platt J, H.T.: An application of reinforcement learning to aerobatic helicopter flight. In: Advances in Neural Information Processing Systems. vol. 19, pp. 1–8. Proceedings of the Twentieth Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada (2007)
15. Sutton R, B.G.: Reinforcement learning : an introduction. IEEE Transactions on Neural Networks 16(1), 285–286 (2005)
16. Sutton R, Mcallester D, e.a.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems. vol. 12, pp. 1057–1063 (2000)
17. T., P.: Solving the pole balancing problem by means of assembler encoding. Journal of Intelligent and Fuzzy Systems 26(2), 857–868 (2014)
18. Whiteson S, Tanner B, W.A.: The reinforcement learning competitions. AI Magazine 31(2), 81–94 (2010)
19. Yau A, Goh G, e.a.: Application of reinforcement learning to wireless sensor networks: models and algorithms. Computing 97(11), 1045–1075 (2015)
20. Zhu H, Zhu F, e.a.: A kernel-based sarsa(λ) algorithm with clustering-based sample sparsification. In: International Conference on Neural Information Processing. pp. 211–220. Springer International Publishing (2016)

Fei Zhu is a member of China Computer Federation. He is a PhD and an associate professor. His main research interests include machine learning, reinforcement learning, and bioinformatics.

Haijun Zhu is a postgraduate student in the Soochow University. His main research interest is reinforcement learning. He programmed the algorithms and implemented the experiments.

Yuchen Fu (corresponding author) is a member of China Computer Federation. He is a PhD and professor. His research interest covers reinforcement learning, intelligence information processing, and deep Web. He is the corresponding author of this paper.

Donghuo Chen is a member of China Computer Federation. He is a PhD. His research interest includes reinforcement learning, model checking.

Xiaoke Zhou is now an assistant professor of University of Basque Country UPV/EHU, Faculty of Science and Technology, Campus Bizkaia, Spain. He majors in computer science and technology. His main interests include machine learning, artificial intelligence and bioinformatics.

Received: January 7, 2017; Accepted: May 15, 2017.