Soft Comput
DOI 10.1007/s00500-010-0581-3

FOCUS

Continuous-action reinforcement learning with fast policy search and adaptive basis function selection

Xin Xu · Chunming Liu · Dewen Hu

Abstract As an important approach to solving complex sequential decision problems, reinforcement learning (RL) has been widely studied in the community of artificial intelligence and machine learning. However, the generalization ability of RL is still an open problem and it is difficult for existing RL algorithms to solve Markov decision problems (MDPs) with both continuous state and action spaces. In this paper, a novel RL approach with fast policy search and adaptive basis function selection, which is called Continuous-action Approximate Policy Iteration (CAPI), is proposed for RL in MDPs with both continuous state and action spaces. In CAPI, based on the value functions estimated by temporal-difference learning, a fast policy search technique is suggested to search for optimal actions in continuous spaces, which is computationally efficient and easy to implement. To improve the generalization ability and learning efficiency of CAPI, two adaptive basis function selection methods are developed so that sparse approximation of value functions can be obtained efficiently both for linear function approximators and kernel machines. Simulation results on benchmark learning control tasks with continuous state and action spaces show that the proposed approach not only can converge to a near-optimal policy in a few iterations but also can obtain comparable or even better performance than Sarsa-learning, and previous approximate policy iteration methods such as LSPI and KLSPI.

Keywords Reinforcement learning · Approximate policy iteration · Markov decision processes · Learning control · Generalization

X. Xu (✉) · C. Liu · D. Hu
College of Mechatronics and Automation, Institute of Automation, National University of Defense Technology, ChangSha, Hunan 410073, People's Republic of China
e-mail: xinxu@nudt.edu.cn

C. Liu
e-mail: lcm_nudt@gmail.com

D. Hu
e-mail: dwhu@nudt.edu.cn

1 Introduction

Reinforcement learning (RL) has been considered as an efficient way to solve complex multistage decision problems that fall under the general framework of Markov decision problems (MDPs), with possibly unknown model parameters. Due to its wide applicability, RL has been popularly studied in the literature of artificial intelligence and machine learning (Kaelbling et al. 1996; Sutton and Barto 1998). Unlike the dynamic programming (DP) methods in operations research, which study various solution methods for MDPs with known models, RL mainly focuses on MDPs with little model information. Therefore, RL is more suitable for solving sequential optimization and control problems with uncertain conditions, which are more practical in many real-world applications.

To estimate the optimal policies of MDPs, various value function estimation techniques have been studied in the RL community. Earlier work on value function estimation in RL mainly dealt with MDPs with discrete state and action spaces. In Sutton (1988), the temporal difference (TD) learning algorithm was proposed. A general form of TD learning algorithm, i.e., TD(λ), was proved to converge when the cardinality of tunable parameters is the same as that of the state space (Dayan 1992; Dayan and Sejnowski 1994). Furthermore, several algorithms have been studied for estimating the optimal value functions of finite-state

Published online: 28 March 2010

MDPs, and some theoretical results have been obtained. In Watkins and Dayan (1992) and Tsitsiklis (1994), the Q-learning algorithm was proved to be asymptotically convergent to the optimal value functions if every state of a finite-state MDP has been visited an infinite number of times. In Singh et al. (2000), the convergence results of a variant of Q-learning called Sarsa-learning were also established.

Although much progress has been made in RL algorithms for finite state and action spaces, it is difficult or even impossible to extend tabular RL algorithms to MDPs with large or continuous state/action spaces. This problem, which is called the 'curse of dimensionality', is also an open problem for DP methods in operations research. In recent years, approximation techniques for RL and DP have attracted much research interest (Bertsekas and Tsitsiklis 1996). Among these techniques, value function approximation (VFA) is the most popular one, and many empirical results (Tesauro 1994; Crites and Barto 1998; Zhang and Dietterich 1995), as well as some theoretical analysis, have been given for VFA in RL. However, when nonlinear approximators are used, it is hard for VFA-based RL algorithms to converge to the optimal policy or a near-optimal policy. In contrast to the VFA methods, policy search is another class of approximate RL approaches, where the policies of MDPs are approximated directly. An earlier work on policy search for RL is the REINFORCE algorithm (Williams 1992), which is based on gradient search techniques. The GPOMDP algorithm (Baxter and Bartlett 2001) was proposed for partially observable MDPs with convergence properties. In these policy search algorithms, the computational costs are usually very large for large-scale MDPs and due to the local minima of gradient algorithms, the optimality of ultimate policies is sensitive to initial conditions.

In addition to the convergence problem, previous work on VFA-based RL algorithms as well as policy search methods is mainly dedicated to MDPs with discrete actions. For MDPs with both continuous state and continuous action spaces, VFA techniques in RL will have computational difficulties in searching for the optimal or near-optimal policies. Compared with approximation methods for RL in continuous state spaces, there have been few efforts devoted to RL in continuous action spaces. A direct and simple approach to dealing with continuous action spaces is to make use of interpolation techniques in discretized action spaces, where the action value functions are usually approximated by existing discrete-action RL algorithms such as Q-learning (Millan et al. 2002; Hasselt and Wiering 2007). However, as indicated by Lazaric et al. (2008), to realize efficient searches in continuous action spaces, interpolation-based algorithms need to make (often implicit) assumptions about the shape of the value function.

The actor-critic algorithms are another class of RL methods for MDPs with continuous spaces. Unlike VFA-based methods, actor-critic algorithms approximate the value functions and policies of an MDP separately so that they will be beneficial for realizing generalization in MDPs with continuous spaces. Much of the recent work on actor-critic methods has been focused on adaptive critic designs (ACDs) (Prokhorov and Wunsch 1997), where an approximated model of the plant dynamics is required. In Lazaric et al. (2008), a Sequential Monte Carlo (SMC) method was proposed to approximate the sequence of probability distributions implemented by the actor, thus obtaining a novel actor-critic algorithm called SMC-learning. Nevertheless, discussions on SMC-learning are limited to discrete state spaces and a good discretization of the state space is required for SMC-learning.

Generally speaking, for MDPs with both continuous state and continuous action spaces, there are still two challenging problems for existing RL algorithms. One is the local convergence problem for policy search using gradient-based techniques. The other is the structure or feature selection problem for function approximators, e.g., the neural networks used in ACDs. Aiming at the above difficulties, an approximate policy iteration approach with fast policy search and adaptive basis function selection, which is called Continuous-action Approximate Policy Iteration (CAPI), is presented in this paper. In CAPI, an analytic optimization technique is employed to realize fast policy search in continuous action spaces. Two adaptive basis function selection methods are developed for TD learning in continuous action spaces so that automatic feature selection can be realized by analyzing the relevance of candidate basis functions. In the simulations, it is illustrated that CAPI can converge to near-optimal policies in a few iterations and the generalization ability of CAPI is well guaranteed with sparse features automatically constructed by the proposed basis function selection methods. The performance of CAPI was compared with some recently developed RL algorithms, namely least-squares policy iteration (LSPI) (Lagoudakis and Parr 2003), kernel-based LSPI (KLSPI) (Xu et al. 2007), and Sarsa-learning with VFA, and it was shown that CAPI can obtain comparable or even superior results to these popular RL methods.

In summary, existing work on RL in continuous state and action spaces can be divided into two categories. The first category is to estimate performance gradients of MDPs or POMDPs, which includes various policy gradient algorithms such as GPOMDP and the REINFORCE algorithm. The main disadvantage of policy gradient RL is the large variance and slow convergence in estimation of policy gradients. Therefore, the data efficiency of policy gradient RL algorithms needs to be improved for real-world

applications. The second category of RL for continuous spaces is the actor-critic algorithms, e.g., ACDs and LSPI, which use value function estimation to decrease the variance of policy gradients. However, the performance of actor-critic algorithms relies greatly on the estimation precision of value functions. The proposed CAPI method can be viewed as a novel actor-critic method based on approximate policy iteration, where the policy gradients are replaced by a greedy policy improvement operator. It is different from previous approaches in that a fast policy search mechanism as well as adaptive basis function selection methods is integrated in an API framework so that the estimation precision of value functions can be improved for better convergence and data efficiency.

Therefore, the main contributions of this paper include the following two aspects. The first aspect is that the general framework of CAPI with fast policy search and adaptive basis function selection is presented so that near-optimal policies of MDPs with continuous states and actions can be approximated in a simple and efficient way. In the CAPI framework, the gradient information of differentiable value function approximators is used to perform fast policy search in continuous action spaces and the solution can be obtained in a simple and analytical way, and it will be computationally more efficient than other mathematical programming methods for policy search. The second aspect of contributions is that adaptive basis function selection is incorporated into CAPI to increase the performance of policy evaluation so that good convergence properties can be guaranteed for approximate policy iteration. Specifically, two adaptive basis function methods were studied in detail, for linear and kernel-based TD-learning, respectively.

This paper is organized as follows: In Sect. 2, an introduction on TD learning and approximate policy iteration is given. A general framework of the CAPI algorithm with fast policy search and adaptive basis function selection is presented in Sect. 3. In Sect. 4, two basis function selection methods are presented for CAPI using kernel functions and linear polynomials. In Sect. 5, simulation results on two learning control problems are provided to illustrate the effectiveness of the proposed algorithm. Section 6 draws conclusions and suggests future work.

2 TD learning and approximate policy iteration

In this section, a brief introduction on the popular linear TD(λ) algorithm as well as the LS-TD(λ) algorithm will be given. Then the basic process of API will be discussed.

2.1 TD learning for policy evaluation

In TD learning, an MDP is assumed to have a stationary policy π and it is assumed that an identical Markov chain with control actions can be defined for the MDP. The objective of TD learning is to compute the value function or action value function of the Markov chain. Let the trajectory generated by the Markov chain be denoted by $\{x_t, a_t | t = 0, 1, 2, \ldots;\ x_t \in X\}$. For each state transition from x_t to x_{t+1}, a scalar reward r_t is defined. The action value function of each state-action pair is defined as follows:

$$Q^{\pi}(x,a) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_t | x_0 = x, a_0 = a\right\} \tag{1}$$

where $0 < \gamma \leq 1$ is a discount factor.

In the TD(λ) algorithm, there are two basic mechanisms, the TD and the eligibility trace, respectively. TDs are defined as the differences between two successive estimations and have the following form:

$$\delta_t = r_t + \gamma \hat{Q}_t^{\pi}(x_{t+1}, a_{t+1}) - \hat{Q}_t^{\pi}(x_t, a_t) \tag{2}$$

where x_{t+1} is the successor state of x_t, $\hat{Q}^{\pi}(x,a)$ denotes the estimate of the action value function $Q^{\pi}(x,a)$, and r_t is the reward received after the state transition from x_t to x_{t+1}.

The eligibility trace can be viewed as an algebraic trick to improve learning efficiency without recording all the data of a multi-step prediction process. This trick is based on the idea of using truncated returns of a Markov chain. To realize incremental or online learning, eligibility traces are defined for each state-action pair (x,a) as follows:

$$z_{t+1}(x,a) = \begin{cases} \gamma\lambda\, z_t(x,a) + 1, & \text{if } x = x_t, a = a_t \\ \gamma\lambda\, z_t(x,a), & \text{else} \end{cases} \tag{3}$$

where $0 \leq \lambda \leq 1$ is a constant.

The online TD(λ) update rule with eligibility traces is

$$\hat{Q}_{t+1}^{\pi}(x,a) = \hat{Q}_t^{\pi}(x,a) + \alpha_t \delta_t z_{t+1}(x,a) \tag{4}$$

where δ_t is the TD at time step t, which is defined in (2) and $z_0(x) = 0$ for all x.

Since the state space of a Markov chain is usually large or infinite in practice, function approximators are commonly used to approximate the value function, where TD(λ) algorithms with linear function approximators are the most popular and well-studied ones.

In linear TD(λ), the following linear function approximator with a fixed basis function vector is considered:

$$\phi(x,a) = (\phi_1(x,a), \phi_2(x,a), \ldots, \phi_n(x,a))^T$$

The estimated action value function can be denoted as

$$\hat{Q}_t^{\pi}(x,a) = \phi^T(x,a) W_t \tag{5}$$

where $W_t = (w_1, w_2, \ldots, w_n)^T$ is the weight vector.

The corresponding incremental weight update rule for linear TD(λ) is

$$W_{t+1} = W_t + \alpha_t\left(r_t + \gamma\phi^T(x_{t+1}, a_{t+1})W_t - \phi^T(x_t, a_t)W_t\right)\vec{z}_{t+1} \quad (6)$$

where the eligibility trace vector $\vec{z}_{t+1} = (z_1, z_2, \ldots, z_n)^T$ is defined as

$$\vec{z}_{t+1} = \gamma\lambda\vec{z}_t + \phi(x_t, a_t) \quad (7)$$

In Tsitsiklis and Roy (1997), the above linear TD(λ) algorithm was proved to converge with probability 1 under certain assumptions. The limit of convergence W^* was also derived, which satisfies the following equation:

$$E_0[A(X_t)]W^* - E_0[b(X_t)] = 0, \quad (8)$$

where $X_t = (x_t, a_t, x_{t+1}, a_{t+1}, z_{t+1})$ $(t = 1,2,\ldots)$ form a Markov process, $E_0[\cdot]$ stands for the expectation with respect to the unique invariant distribution of $\{X_t\}$, and $A(X_t)$ and $b(X_t)$ are defined as

$$A(X_t) = \vec{z}_t\left(\phi^T(x_t) - \gamma\phi^T(x_{t+1})\right) \quad (9)$$

$$b(X_t) = \vec{z}_t r_t \quad (10)$$

To improve the efficiency of linear TD(λ) algorithms, the least-squares temporal-difference algorithm, called LS-TD(λ), was proposed in Boyan (2002) by solving (8) directly and the model-based property of LS-TD(λ) was also analyzed. Kernel methods can also be employed to design kernel-based TD-learning with nonlinear function approximation capability in a kernel-induced feature space, which is called KLS-TD and has been studied in (Xu et al. 2007). In this paper, both LS-TD(λ) and KLS-TD learning algorithms will be applied and evaluated in the policy evaluation process of CAPI. Adaptive basis function selection will be studied for TD-learning in Sect. 4.

2.2 Approximate policy iteration in RL

Unlike the traditional policy iteration process in DP, approximate policy iteration focuses on using function approximators to estimate the value functions and policies in MDPs with large or continuous spaces. The basic structure of approximate policy iteration or an actor-critic RL agent (Sutton and Barto 1998) is depicted in Fig. 1. In Fig. 1, the critic and the actor perform the procedures of policy evaluation and policy improvement, respectively. Policy evaluation usually makes use of TD learning algorithms to estimate the action value functions $Q^{\pi[n]}$ of policy $\pi[n]$ without any model information on the underlying MDP, where $\pi[n]$ is the policy in iteration number n.

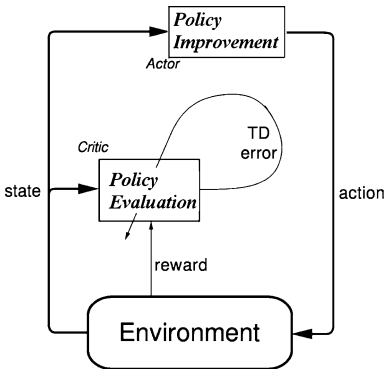

Fig. 1 Basic structure of approximate policy iteration (Sutton and Barto 1998)

Based on the estimated action value function $\hat{Q}^{\pi[n]}$, the policy improvement in the actor produces a greedy policy $\pi[n+1]$ over $\pi[n]$ as

$$\pi[n+1] = \arg\max_a \hat{Q}^{\pi[n]}(x, a) \quad (11)$$

Thus, the greedy policy $\pi[n+1]$ is a deterministic policy and when the estimated action value function $\hat{Q}^{\pi[n]}$ approximates $\pi[n]$ with high precision, $\pi[n+1]$ will be at least as good as $\pi[n]$. This iteration process is repeated until there is no change between the policies $\pi[n]$ and $\pi[n+1]$. After the convergence of approximate policy iteration, a near-optimal policy can be obtained, usually within a very few iterations. However, the convergence of approximate policy iterations relies greatly on the approximation precision of the real value functions. If the value functions are exactly represented, e.g., in cases of tabular state spaces, or the approximation errors are small enough to be neglected, the convergence and performance of approximate policy iteration will be very satisfactory.

3 A general framework of the CAPI algorithm

As discussed in Sect. 1, many research works on RL have been focused on improving the generalization ability of RL algorithms in MDPs with continuous state spaces and discrete action spaces. However, in lots of real-world applications, it will be necessary to develop RL algorithms for MDPs with both continuous state and continuous action spaces. In this paper, a general RL framework for continuous-action API with fast policy search and adaptive basis function selection, called CAPI, is proposed to obtain near-optimal control policies for MDPs with continuous state and action spaces. The basic framework of CAPI, which is depicted in Fig. 2, mainly consists of the following three procedures.

The first procedure is to perform adaptive basis function selection using the data samples of the current policy.

Given: Data samples $D(0)=\{(x_t, a_t, r_t, x_{t+1}, a_{t+1})\}$ (t=1,2,…T), which are generated by an initial policy $\pi(0)$; the maximum iteration number is set to N.

(a) Initialize: Set the parameters of TD learning algorithms; iteration number n=0.

(b) (**Adaptive Basis Function Selection**): For data sample set $D(n)$, employ adaptive basis function selection methods to construct the basis functions for linear approximators or kernel machines in TD-learning;

(c) (**Policy Evaluation**): Use TD-learning to compute the approximated action value function $\hat{Q}^{\pi(n)}(x,a)$ for the current policy.

(d) (**Continuous-action Policy Search**) For every state x_{t+1} in the data samples, perform fast policy search in the continuous-action space based on the gradients of the approximated action value function.

(e) Update the data samples: $D(n+1)=\{(x_t, a_t, r_t, x_{t+1}, a'_{t+1})\}$ (t=1,2,…T), where a'_{t+1} is the greedy action computed in step (d).

(f) If the difference between two successive policies is smaller than a given threshold or $n>N$, output the final weights and approximated action policy, stop.
Else $n=n+1$, return to (b).

Fig. 2 A general framework of the CAPI algorithm

Various basis function selection methods can be employed in this procedure to construct basis functions for TD-learning with function approximation. In this paper, we will focus on two adaptive basis function selection techniques, for linear least-squares TD-learning and kernel-based TD-learning, respectively. Furthermore, the value function approximators can be selected to be differentiable with respect to the actions so that a fast policy search can be performed based on the gradient information of the value function approximators. These two basis function selection methods will be studied in detail in Sect. 4.

The second procedure in the framework is to use TD-learning for policy evaluation. To approximate the value functions in policy evaluation, both LS-TD(λ) using linear function approximators and its kernelized version KLS-TD(λ) can be used in CAPI. In this section, to simplify notations, we will focus on CAPI with linear function approximators, and the extension to CAPI using kernels will be discussed in Sect. 4.

The third procedure in CAPI is to perform fast policy search in continuous-action spaces based on the approximated action value functions. In the following, we will present a computationally efficient method to search for the optimal actions by using the gradient information of differentiable value function approximators, so that the solution can be obtained in a simple and analytical way. Although other mathematical programming methods for optimization may be used in this process, the computational costs may prevent them from being used in complex learning control problems which require a large number of data samples for approximate policy iteration.

The initial conditions of CAPI include the initial policy $\pi(0)$ and the data samples generated from the initial policy. The data samples are collected by simulating or observing the underlying MDP controlled by policy $\pi(0)$ for a number of episodes and a state transition trajectory is recorded for each episode. For every state x_{t+1} in the data samples $D(n) = \{(x_t, a_t, r_t, x_{t+1}, a_{t+1})\}$ (t = 1,2,…T), a fast policy search process can produce a new greedy action a'_{t+1} in the continuous-action space based on the approximated action value function. Then, a new set of data samples $D(n+1) = \{(x_t, a_t, r_t, x_{t+1}, a'_{t+1})\}$ (t = 1,2,…T) can be obtained for policy evaluation in the next iteration. The CAPI algorithm is terminated when the difference between two successive policies is smaller than a given threshold or the iteration number reaches the maximum number of iterations.

Compared with previous works on API in RL such as LSPI, the main novelties of the above CAPI framework are the fast policy search process in continuous-action spaces and the adaptive basis function selection procedure for performance improvement in TD-learning. The fast policy search process will be presented in the following discussions and the techniques for adaptive basis function selection will be given in the next section.

In CAPI with linear function approximators, the action value functions are approximated by a weighted sum of linear feature vectors

$$\hat{Q}^{\pi(n)}(x,a) = \phi^T(x,a)W \tag{12}$$

The policy evaluation process of CAPI can make use of the following LS-TD(λ) updates for estimating the weights of action VFA.

$$W_{LS-TD(\lambda)} = A_T^{-1} b_T = \left(\sum_{t=1}^{T} A(X_t)\right)^{-1} \left(\sum_{t=1}^{T} b(X_t)\right) \tag{13}$$

where $A(X_t)$, $b(X_t)$ are defined by (7), (9) and (10).

After computing the weights in (13), the approximated action value function is obtained by

$$\hat{Q}^{\pi(n)}(x,a) = \phi^T(x,a)W_{LS-TD(\lambda)} \tag{14}$$

To realize policy improvement in CAPI, it is necessary to perform policy search in the action value function space so that the best action for each state in the sample trajectories can be found:

$$\pi[n+1]|_x = a^* = \arg\max_a \hat{Q}^{\pi(n)}(x,a) \tag{15}$$

The fast policy search method in this paper employs a differentiable approximation structure for the action value functions, i.e., the basis functions in (14) are selected to be differentiable with respect to the action a. As will be discussed in Sect. 4, by appropriately selecting the basis functions for VFA, the linear function approximators and the kernel-based basis functions in CAPI can both be differentiable with respect to the action a, so that analytic solution techniques can be used for fast policy search in continuous action spaces.

We assume that the action a varies in an interval $[a_{\min}, a_{\max}]$. When the approximated action value function is differentiable with respect to a, the derivatives of the action value function can be computed for $a_{\min} < a < a_{\max}$,and the solution a^0 of the following equation can be obtained as a candidate for the best action

$$\frac{\partial \hat{Q}(x,a,W_{LS-TD})}{\partial a}|_{a=a^0} = 0 \tag{16}$$

As we will discuss in Sect. 4, when polynomials or polynomial kernels are used as the basis functions, the solutions to the above equation can be computed directly. In addition, when a low degree polynomial is selected, the optimal solution to (16) can be obtained by solving a low-degree algebraic equation. This is computationally efficient, and a unique solution may be computed. If multiple solutions are obtained for (16), a^0 will become a set of candidate actions.

Then, for every state x_t in the sample trajectories, the greedy action a^* can be selected among the candidate action set $\{a^0, a_{\min}, a_{\max}\}$, which satisfies the following equation:

$$\hat{Q}^{\pi}(x_t,a^*) = \max\left\{\hat{Q}^{\pi}(x_t,a_{\min}), \hat{Q}^{\pi}\left(x_t,a^0\right), \hat{Q}^{\pi}(x_t,a_{\max})\right\} \tag{17}$$

After computing the greedy action for every state in the sampled trajectories, a new greedy policy can be obtained on the state trajectories to implement the policy improvement process:

$$\pi[n+1](x_t) = \arg\max_a \hat{Q}^{\pi[n]}(x_t,a) = a^* \tag{18}$$

The above process can be used as a fast policy search technique for policy improvement in continuous action space. Although the computation of the best action for policy improvement can be formulated as a mathematical programming problem, and linear or nonlinear programming methods may be used, the above policy search method is computationally simple and effective. Furthermore, the implementation of the fast policy search process will be integrated with the adaptive basis function selection process in Sect. 4 so that low degree linear polynomials or kernels can be selected and the action value function approximators will be differentiable with respect to the actions.

4 Adaptive basis function selection in CAPI

Since the success of approximate policy iteration will mainly depend on the performance of TD learning algorithms for policy evaluation, it is essential to use TD-learning algorithms with good generalization capability and high approximation accuracy. In this section, two adaptive basis function selection methods will be integrated in CAPI with linear polynomials and kernel-based features, respectively. The first method is based on the principle of correlation analysis so that a subset of polynomial basis functions can be selected to improve the generalization performance of LS-TD(λ) learning algorithms. The second method uses the approximately linearly dependent (ALD) analysis approach for selecting kernel-based feature vectors to realize sparse kernel-based CAPI.

4.1 Correlation analysis for CAPI with polynomial basis functions

TD learning with linear basis functions has been widely employed for policy evaluation because of its simplicity and low computational complexity. In CAPI, in order to implement fast policy search using approximated value functions, it is beneficial to make the action value function differentiable with respect to the action variables and the optimization computation in (16) as simple as possible. In the following, linear polynomials will be used as candidate

basis functions for policy evaluation using LS-TD(λ) so that the solutions to (16) can be derived analytically. To improve the generalization ability of linear polynomials, a novel correlation analysis method is presented to select the most appropriate basis functions.

Let $\{\phi_i(x,a)\}$ $(i = 1,2,\ldots,N)$ be the set of candidate polynomials, where N is the number of all possible basis functions, $x = (x_1, x_2, \ldots, x_m)$ is the state vector, a is the action variable and m is the dimension of states. The polynomial features have the following form:

$$\phi_i(x,a) = a^{k_{ai}} x_1^{k_{1i}} x_2^{k_{2i}} \cdots x_m^{k_{mi}} \tag{19}$$

where $0 \le k_{ai} \le n_a, 0 \le k_{1i} \le n_1, 0 \le k_{2i} \le n_2, \ldots 0 \le k_{mi} \le n_m, n_a, n_j (j = 1,2,\ldots m)$ are the maximum power number of the action a and the states, respectively.

Based on the above polynomial basis functions, the approximated action value functions can be expressed as

$$\hat{Q}^{\pi}(x,a) = \sum_{i=1}^{N} a^{k_{ai}} x_1^{k_{1i}} x_2^{k_{2i}} \cdots x_m^{k_{mi}} w_i \tag{20}$$

where w is the weight vector computed by LS-TD learning algorithms in the process of policy evaluation.

The derivative of the action value function with respect to action a is

$$\frac{\partial}{\partial a}\hat{Q}^{\pi}(x,a) = \sum_{i=1}^{N} k_{ai} a^{(k_{ai}-1)} x_1^{k_{1i}} x_2^{k_{2i}} \cdots x_m^{k_{mi}} w_i \tag{21}$$

Let

$$\frac{\partial}{\partial a}\hat{Q}^{\pi}(x,a) = 0 \tag{22}$$

Then, an algebraic equation for variable a is obtained and the solutions can be computed for fixed states and weight vectors. Since all solutions of Eq. 22 are extreme values of the action value function, an optimal value of action a for a given state x can be selected according to (15). In CAPI, the weight vector can be computed by LS-TD(λ) using the data samples collected in each iteration. Using the above optimization technique, it is direct and simple to perform policy improvement in approximate policy iteration with continuous action and state spaces. However, sparsification techniques still need to be developed to reduce the computational costs, since for candidate basis functions in the form of polynomials, the total number of possible features is

$$N = (n_a + 1)\prod_{i=1}^{m}(n_i + 1) \tag{23}$$

If the dimension m of the state space or the maximum power number of each dimension increases, the feature dimension of the above linear polynomials will increase very quickly. This may lead to large computational costs and a possible reduction of generalization abilities. Therefore, it is necessary to develop adaptive feature selection methods for the above polynomial basis functions.

As discussed earlier, the aim of TD(λ) algorithms is to approximate the action value function that satisfies the Bellman equations 24 and 25, where $R(x, a)$ is the reward function, P is the state transition probability, and Π_π is the action selection probability of the policy. Since the approximated action value function is represented by (20), it is desirable to select a subset of basis functions which can approximate the reward function $R(x, a)$ efficiently and compactly.

$$Q^{\pi}(x,a) = R(x,a) + \gamma P \Pi_{\pi} Q^{\pi}(x,a) \tag{24}$$

$$(I - \gamma P \Pi_{\pi}) Q^{\pi}(x,a) = R(x,a) \tag{25}$$

In the proposed approach, the correlation coefficient between each basis function and $R(x, a)$ is adopted to quantify the validity of the basis functions for approximating $R(x, a)$. When data samples of an MDP are collected, the correlation coefficients between the basis functions and the reward function can be computed as follows:

$$\rho_i = \frac{\left|\sum_{j=1}^{M}\left[\phi_i(x_j,a_j) - \bar{\phi}_i\right]\left[r(x_j,a_j) - \bar{r}\right]\right|}{\sqrt{\sum_{j=1}^{M}\left[\phi_i(x_j,a_j) - \bar{\phi}_i\right]^2 \cdot \sum_{j=1}^{M}\left[r(x_j,a_j) - \bar{r}\right]^2}} \tag{26}$$

where M is the total number of samples, $i = 1,2,\ldots,m$, and

$$\bar{r} = \frac{\sum_{j=1}^{M} r(x_j,a_j)}{M} \tag{27}$$

$$\bar{\phi}_i = \frac{\sum_{j=1}^{M} \phi_i(x_j,a_j)}{M} \tag{28}$$

In our implementation, an iterative process was designed to construct a set of polynomial basis functions based on the above correlation analysis to select the most appropriate polynomial basis functions. In the iterative process, the set of selected basis functions is constructed incrementally and the correlation analysis procedure is performed between a candidate basis function and the residual approximation errors of the reward function using selected basis functions. Thus, in the computation of the correlation coefficients in (26), the reward function $r(x, a)$ will be replaced by the approximation error $\Delta r(x_j,a_j) = r(x_j,a_j) - r'(x_j,a_j)$, where $r'(x,a)$ is the least-squares approximation to $r(x, a)$ using the basis functions that have been selected. Figure 3 gives a flow chart of the basis function selection process based on correlation analysis.

The first stage in the iterative process is to initialize the set of basis functions as follows:

$$\Lambda = \left\{\phi_1(x,a) = a^0 x_1^0 x_2^0 \cdots x_m^0 = 1\right\} \tag{29}$$

where Λ is the set of selected basis functions and $\phi_1(x,a) = 1$ can also be viewed as an offset value.

The second stage is to approximate the reward function $R(x, a)$ with the basis function set Λ by making use of the least-squares regression method as follows:

$$w = \left\{\sum_{i=1}^{M}\left[\mathbf{\Phi}^T(x_i,a_i)\mathbf{\Phi}(x_i,a_i)\right]\right\}^{-1}\sum_{i=1}^{M}\left[\mathbf{\Phi}^T(x_i,a_i)r(x_i,a_i)\right] \tag{30}$$

$$r'(x,a) = \mathbf{\Phi}(x,a)w \tag{31}$$

where $r'(x, a)$ is the least-squares approximation of $r(x, a)$, and $\mathbf{\Phi}(x,a)$ is defined as

$$\mathbf{\Phi}(x,a) = (\phi_1(x,a), \phi_2(x,a), \cdots, \phi_n(x,a)), \quad \phi_i(x,a) \in \Lambda \quad i = 1,2,\ldots,n$$

For every sample (x_j, a_j) $(j = 1,2,\ldots,M)$, the residual error of the above least-squares approximation can be expressed as

$$\Delta r(x_j,a_j) = r(x_j,a_j) - r'(x_j,a_j) \tag{32}$$

The third stage is to compute the correlation coefficients between the remaining basis functions and $\Delta r(x,a)$ using (26), where the reward function $r(x, a)$ and its mean value are replaced by the residual error $\Delta r(x,a)$ and its mean value $\Delta\bar{r}(x,a)$. The basis function with the maximal correlation coefficient will be added to Λ.

As illustrated in Fig. 3, the above construction process of the basis function set will be iterated until the maximum of all the correlation coefficients is less than a given threshold. After the end of the basis function selection, a subset of the polynomial basis functions can be obtained to realize sparse approximation of the action value function for better generalization performance and lower computational complexity.

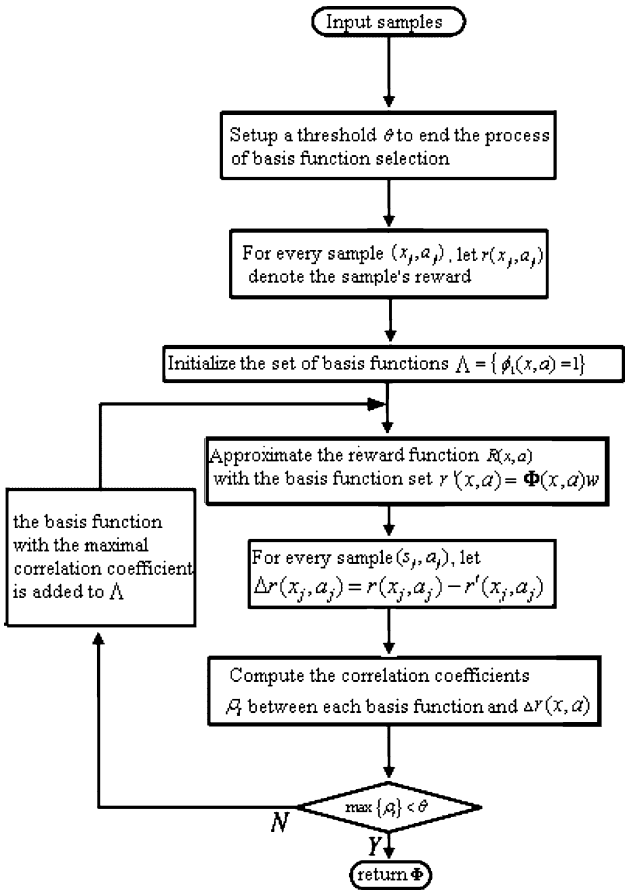

Fig. 3 Adaptive basis function selection based on correlation analysis

4.2 ALD analysis for kernel-based CAPI

Because of the capabilities of implicit nonlinear feature transformations, kernel functions or kernel methods have been popularly studied and employed in various machine learning algorithms such as Kernel PCA (Schölkopf and Smola 2002), Kernel ICA (Bach and Jordan 2002). The application of kernel methods in RL has also received increasing attention in recent years (Rasmussen and Kuss 2004). In the following, CAPI with kernel functions will be studied and the basis function selection method for the sparsification of kernel-based features in CAPI will be discussed.

Let X be the original state space. A kernel function is a mapping from $X \times X$ to R and a Mercer kernel is a kernel function that is positive definite, i.e., for any finite set of points $\{x_1, x_2,\ldots,x_n\}$, the kernel matrix $K = [k(x_i, x_j)]$ is positive definite. According to the Mercer theorem (Schölkopf and Smola 2002), there exist a Hilbert space H and a mapping ϕ from X to H such that

$$k(x_i,x_j) = \langle\phi(x_i), \phi(x_j)\rangle \tag{33}$$

where $\langle \bullet, \bullet \rangle$ is the inner product in H. Although the dimension of H may be infinite and the nonlinear mapping ϕ is usually unknown, all the computations in the feature space can still be performed if it is in the form of inner products.

In CAPI, kernel functions can be used to approximate the action value function by the following KLS-TD(λ) algorithm (Xu et al. 2007):

$$\alpha = A_T^{-1} b_T \tag{34}$$

$$A_T = \sum_{i=1}^{T}\vec{k}(s_i)\left[\vec{k}^T(s_i) - \gamma\vec{k}^T(s_{i+1})\right] \tag{35}$$

$$b_T = \sum_{i=1}^{T}\vec{k}(s_i)r_i \tag{36}$$

where $s_i = (x_i, a_i)(i = 1, 2, \ldots T)$ are the observed state-action pairs at different time steps, and

$$\vec{k}(s_i) = (k(s_1, s_i), k(s_2, s_i), \ldots, k(s_T, s_i))^T \quad (37)$$

Because the original dimension of the kernel-based basis functions is equal to the number of data samples, feature selection and sparsification are also needed to reduce the computational complexity and improve the generalization performance of kernel machines. In this paper, the ALD analysis approach (Engel et al. 2004) will be employed to implement sparsification and feature selection in CAPI using kernel-based basis functions.

The main idea of ALD analysis is to select the approximately linearly independent features to represent the target function in a least-squares sense. In CAPI using kernels, the features to be selected are generated from the implicit feature mapping induced by the kernel function. However, by making use of the kernel trick, the ALD condition discussed below can be computed by the kernel function, which is equal to the inner product of two kernel-induced feature vectors.

Let $S_n = \{s_1, s_2, \ldots, s_n\}$ denote a set of observation data samples. To perform approximate linear dependence analysis on the feature vector set, a data dictionary is defined as a subset of the whole feature vector set. The data dictionary is initially empty and the ALD analysis is implemented by testing every feature vector in the feature set, one at a time. If the feature vector of a data sample cannot be approximated, within a predefined precision, by a linear combination of the feature vectors in the dictionary, it will be added to the dictionary; otherwise it will not be added to the dictionary. Thus, after the ALD analysis process, all the feature vectors of the data samples in S_n can be approximately represented by linear combinations of the feature vectors in the final dictionary within a given precision.

Suppose we have tested $t-1$ $(1 < t \leq n)$ feature vectors of the samples in the original data set S_n and $D_{t-1} = \{\phi(s_j)\}$ $(j = 1,2,\ldots d(t-1))$ is the obtained data dictionary, where $d(t-1)$ is the number of selected features in the data dictionary. The ALD condition of a new feature vector $\phi(s_t)$ can be tested as follows (Engel et al. 2004):

$$\delta_t = \min_c \left\| \sum_j c_j \phi(s_j) - \phi(s_t) \right\|^2 \leq \mu \quad (38)$$

where $c = [c_j]$ $(j = 1,2,\ldots d(t-1))$ and μ is a threshold parameter to determine the approximation accuracy and the sparsity level. When μ is appropriately selected, the sparsity of kernel-based solutions can be guaranteed without sacrificing much in approximation accuracy.

The left side of inequality (38) can be rewritten as

$$\delta_t = \min_c \left\{ \sum_{i,j} c_i c_j \langle \phi(s_i), \phi(s_j) \rangle - 2 \sum_i c_i \langle \phi(s_i), \phi(s_t) \rangle + \langle \phi(s_t), \phi(s_t) \rangle \right\} \quad (39)$$

Due to the kernel trick, after substituting (37) into (39), we can obtain

$$\delta_t = \min_c \{ c^T K_{t-1} c - 2 c^T k_{t-1}(s_t) + k_{tt} \} \quad (40)$$

where $[K_{t-1}]_{i,j} = k(s_i, s_j)$, s_i $(i = 1,2,\ldots,d(t-1))$ are the elements in the dictionary, $k_{t-1}(s_t) = [k(s_1,s_t), k(s_2, s_t),\ldots, k(s_{d(t-1)},s_t)]^T$, $c = [c_1, c_2,\ldots,c_d]^T$ and $k_{tt} = k(s_t, s_t)$.

In the ALD-based sparsification process, the data dictionary is updated by comparing δ_t with a predefined threshold μ. If $\delta_t < \mu$, the dictionary is unchanged, otherwise, s_t is added to the dictionary, i.e., $D_t = D_{t-1} \cup \{s_t\}$. After all the data samples are tested, a subset of data samples can be selected as the dictionary for computing the kernel-based basis functions so that the action value function can be estimated with lower computational complexity.

5 Performance evaluation

To illustrate the effectiveness of the proposed approach, two learning control tasks with continuous state and action spaces were considered to evaluate the performance of different RL algorithms, namely CAPI, LSPI, Sarsa-learning and KLSPI. These two tasks, the mountain-car task and the inverted pendulum problem, have been popularly used as benchmark problems for performance evaluation of RL algorithms (Sutton 1996). However, in previous studies, the control action sets were commonly assumed to be discrete and the control policies obtained by RL algorithms were restricted to discrete action spaces. In CAPI, continuous action spaces will be searched for estimation of near-optimal policies. This is more practical in many real-world applications. In the following simulation, the performance of different RL algorithms will be compared not only in terms of learning efficiency or convergence but also in terms of the quality of final control policies. It will be shown that the CAPI algorithms with polynomial basis functions and kernel functions can converge to near-optimal policies in a few iterations. More importantly, since CAPI can implement fast policy search in continuous action spaces and realize better generalization using adaptive basis function selection, the near-optimal policies of CAPI will have comparable or even better performance than LSPI, KLSPI and Sarsa-learning.

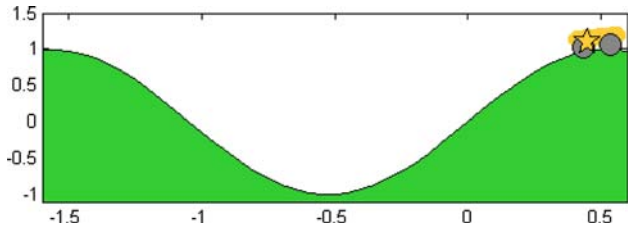

Fig. 4 The mountain-car task

5.1 The mountain-car task

The mountain-car task is to learn a time-optimal control policy to drive an underpowered car up a steep mountain road, as shown in Fig. 4. The difficulty in this task is that gravity is stronger than the car's engine power, and even at full throttle the car cannot accelerate up the steep slope. The feasible solution is to first move back away from the goal and up the opposite slope on the left. Then, the car must apply its maximum throttle to push it up the steep slope even though it is slowing down the whole way. This is a time-optimal control problem with continuous state and control action spaces. It is difficult for many control methodologies unless explicit aid can be provided from a human designer.

For RL algorithms, the problem becomes more difficult, since the dynamics of the car are assumed to be unknown. Furthermore, when no prior expert knowledge is available, it is more challenging to find a near-optimal policy in both continuous state and continuous action spaces. In previous work, by making use of human experience, only discrete actions with maximum values were considered and the main focus was the generalization in continuous state spaces. For example, in Sutton (1996) and Whiteson and Stone (2006), various function approximators were considered to realize generalization in continuous state spaces. In CAPI, the generalization in both continuous state and continuous action spaces will be realized by combining the fast policy search method with LS-TD learning as well as automatic basis function selection. To illustrate the effectiveness of the proposed method, CAPI will be applied in the mountain-car task and its performance will be compared with previous RL algorithms, including Sarsa-learning, LSPI, and KLSPI.

In the learning control experiments, the dynamics of the car can be described as follows:

$$\begin{cases} \dot{x} = v \\ \dot{v} = 0.001u - g\cos 3x \end{cases} \tag{41}$$

where the gravity term $g = 0.0025$, x and v are the position and velocity of the car, and u is the control action which is bounded by the car's maximum throttle. The variation range for the car's states is

$$-1.5 \le x \le 0.5 \tag{42}$$

$$-0.07 \le v \le 0.07 \tag{43}$$

When the car is at the left peak, $x = -1.5$. The initial position of the car is $x = -0.5$, which is at the lowest point of the valley. The goal of the car is the right-most peak, with $x = 0.5$. The learning control task is to find a time-optimal control policy, without any model information, to move the car from its initial position to the goal.

To apply RL algorithms, the problem can be modeled as an MDP, where the states include the car's position and velocity, the action is the control force u, and the reward function can be defined as

$$r_t = \begin{cases} -1, & x < 0.5 \\ 100, & x \ge 0.5 \end{cases} \tag{44}$$

For LSPI, KLSPI, and Sarsa-learning, the control actions are limited in a discrete set $\{-1, 1\}$. In CAPI, policy search is performed in a continuous interval $[-1, 1]$. The maximum iteration number of LSPI, KLSPI, and CAPI is 10. The discount factor of all the algorithms is set to 0.95. In the experiments, one episode for data collection is defined as the time period from an initial state of the car to the goal state or the final state after a maximum number of time steps. For approximate policy iteration algorithms, 20 episodes of data samples were collected where the maximum number of time steps in each episode is 100,000. The initial policy for the API algorithms is a randomized control policy. The initial weights of Sarsa-learning were also generated from a uniform probabilistic distribution.

For the four RL algorithms evaluated in the experiments, since different parameter settings may cause performance variations, the parameter setting with the best average performance among five independent tests was selected for the evaluation of each algorithm. In Sarsa-learning, the state and action spaces of the mountain-car problem were both partitioned into discrete values, and a learning rate $\alpha = 0.5$ was used. For LSPI, the following polynomials were selected as the linear basis functions:

$$\phi(\vec{x}, a) = [\delta(\vec{x}, a_1), \delta(\vec{x}, a_2)] \tag{45}$$

$$\delta(\vec{x}, a_i) = \begin{cases} 0, 0, 0, 0, 0 & a \ne a_i \\ [1,\ x,\ x^2,\ v,\ v^2] & a = a_i \end{cases} \tag{46}$$

where $\vec{x} = [x, v]$, and $a_1 = 1$, $a_2 = -1$, corresponding to the two action values of the car.

To illustrate the effectiveness of the proposed methods for basis function selection, linear polynomials and polynomial kernels with the following forms were used in CAPI for VFA:

$$\phi_i(\vec{x}, a) = a^{k_{ai}} x^{k_{1i}} v^{k_{2i}} \tag{47}$$

$$k((\vec{x}_1, a_1), (\vec{x}_2, a_2)) = (1 + a_1 a_2 + x_1 v_2 + x_2 v_1)^2 \quad (48)$$

where $0 \le k_{ai} \le 2,\ 0 \le k_{1i} \le 2,\ 0 \le k_{2i} \le 2$, and the number of candidate polynomials is 27.

In KLSPI for MDPs with discrete actions, a radius basis function (RBF) kernel was used to approximate the action value functions, where all the state and action variables were normalized into the interval of $[-1, 1]$. The width parameter for RBF kernels was set to 1 and the threshold for ALD analysis is 0.65.

In the experiments, it was observed that the proposed CAPI algorithms, either with linear polynomials or with kernels, can converge to a near-optimal policy in a few iterations. The learning curves of CAPI, averaged in five independent runs, were depicted in the following Fig. 5, where the performance variations (in terms of running steps to the goal) of CAPI with linear polynomials and CAPI with kernels were shown in solid lines and dotted lines, respectively.

Because there are performance variations in different runs of RL algorithms, a best near-optimal policy was selected and compared from five independent runs for each RL algorithm. In Fig. 6, the performance of the final policies obtained by different RL algorithms was evaluated based on the trajectory of the car's position. The horizontal axis is the number of time steps and the vertical axis is the position of the car controlled by the final policies of RL algorithms, where the red dotted line indicates the goal position. Every curve in Fig. 6 illustrates the position variations of the car and a curve ends when the car's position is equal to the goal position. Figure 6 shows that the proposed CAPI algorithms, including CAPI with kernels and with polynomials, can achieve better near-optimal policies than Sarsa-learning, LSPI, or KLSPI.

In the simulation, the basis function selection methods presented in this paper can generate sparse representations for action value functions in continuous spaces. In CAPI with kernel functions, the number of selected elements in the data dictionary is 10. A total of 11 polynomials were selected in CAPI with polynomials.

In Fig. 7, the approximated action value functions of the near-optimal policy obtained by CAPI with kernel functions are depicted in the form of 3-dimensional surfaces, where the three sub-figures show the action value functions of the control policies when the car's velocities are at -0.07, 0, and 0.07, respectively. The approximated action value functions of CAPI using polynomials, which are almost identical to those of CAPI using kernels, are shown in Fig. 8.

5.2 The inverted pendulum problem

The inverted pendulum problem has been widely studied as a benchmark control problem with nonlinearity and instability. The controller design for inverted pendulums becomes more difficult when there are model uncertainties and unknown disturbances in the plant dynamics. The learning control of inverted pendulums is different from conventional controller design in that no model information is required a priori and the controller is constructed in a data-driven style. The learning control of inverted pendulums has also been considered as a test problem for machine learning methods, especially for reinforcement learning algorithms (Barto et al. 1983). In the following, simulation studies will be conducted on the inverted pendulum problem to compare the performance of different RL algorithms.

Figure 9 shows a typical inverted pendulum system, which consists of a cart moving horizontally and a pole

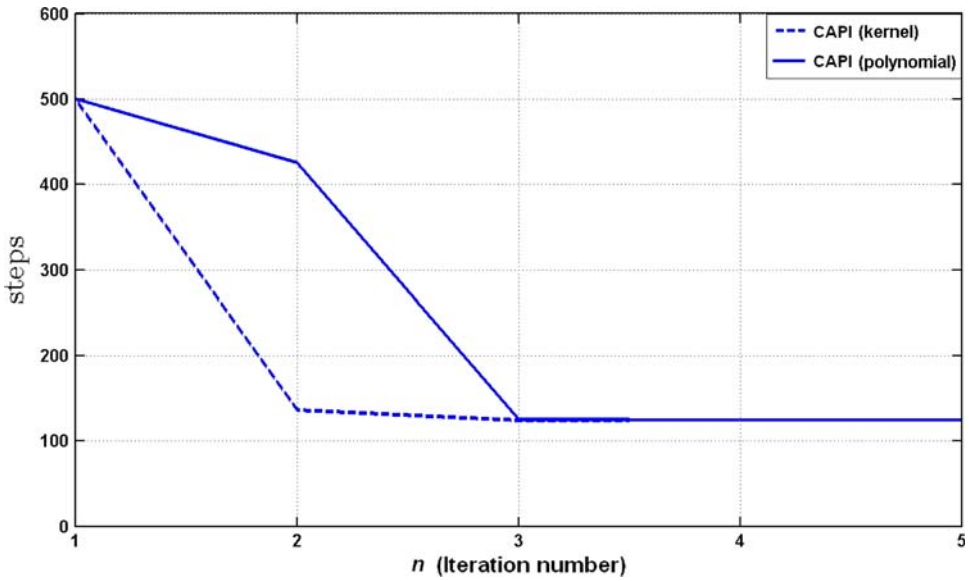

Fig. 5 The learning curves of CAPI averaged in five independent runs (*dotted* CAPI with kernels; *solid* CAPI with polynomials)

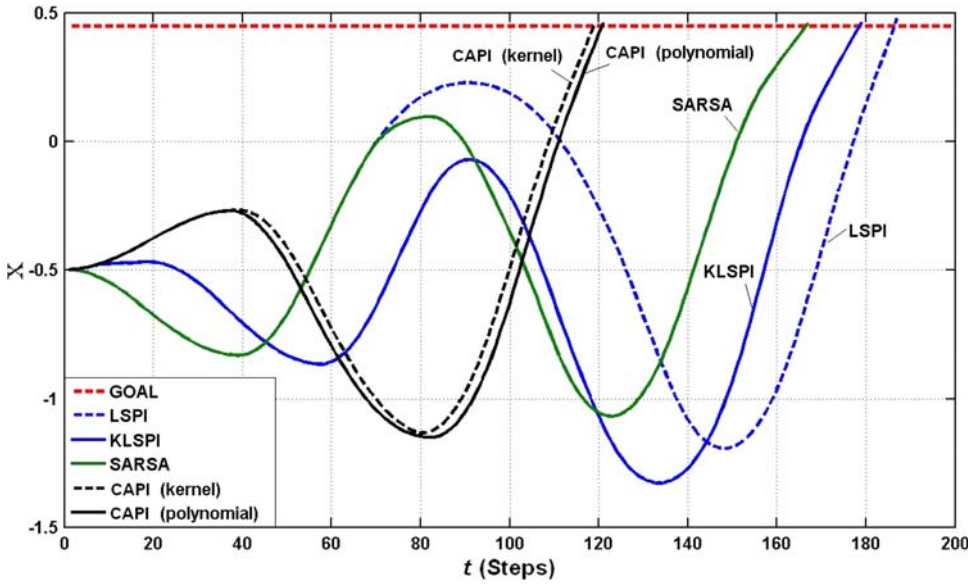

Fig. 6 Performance comparisons of the final policies obtained by different RL algorithms

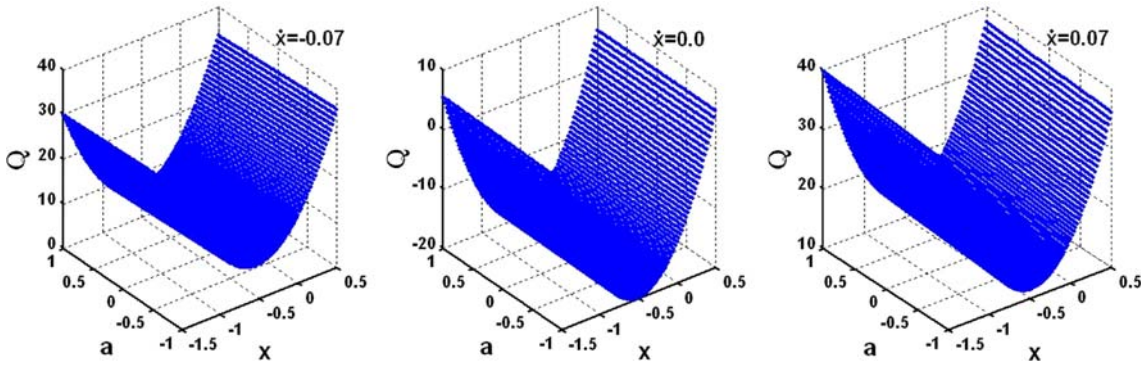

Fig.7 Approximated action value functions of CAPI using kernels

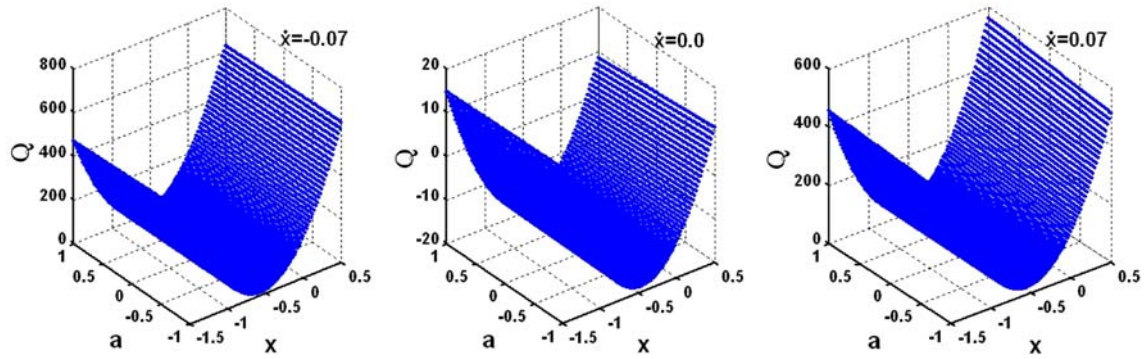

Fig. 8 Approximated action value functions of CAPI using polynomials

with one end fixed at the cart. Let x denote the horizontal distance between the center of the cart and the center of the track, where x is negative when the cart is in the left part of the track. Variable θ denotes the angle of the pole from its upright position (in degrees) and F is the amount of force (N) applied to the cart to move it towards its left or right. So the control system has four state variables $x, \dot{x}, \theta, \dot{\theta}$, where $\dot{x}, \dot{\theta}$ are the derivatives of x and θ, respectively.

The dynamics of the control system can be described by the following equations.

$$\begin{cases} \ddot{\theta} = \dfrac{(m+M)g\sin\theta - \cos\theta\left[F + ml\dot{\theta}^2\sin\theta - \mu_c \mathrm{sgn}(\dot{x})\right] - \frac{\mu_p(m+M)\dot{\theta}}{ml}}{\frac{4}{3}(M+m)l - ml\cos^2\theta} \\ \ddot{x} = \dfrac{F + ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta) - \mu_c \mathrm{sgn}(\dot{x})}{M+m} \end{cases} \tag{49}$$

where g is the acceleration due to the gravity (-9.8 m/s^2). In Fig. 9, the mass of the cart is $M = 8.0$ kg, the mass of

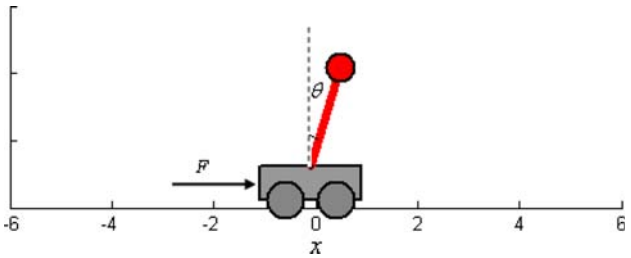

Fig. 9 Control system of an inverted pendulum

the pole is $m = 2.0$ kg, the half-pole length is $l = 0.5$ m, μ_c is the coefficient of friction of the cart on the track and μ_p is the coefficient of friction of the pole on the cart. In the simulation, both of the friction coefficients were neglected. The simulation time-step is 0.1 s.

In the learning control experiments, the dynamics of the pendulum system is assumed to be unknown. The aim of the learning controller is to balance the pole as long as possible and make the angle variations of the pendulum be as small as possible. The reward function at time step t can be defined as

$$r_t = e^{-|2\cdot\theta(t)|} \tag{50}$$

The objective function to be optimized can be given as

$$V = \sum_{t=1}^{M} \gamma^t r_t \tag{51}$$

where M is the maximum time step and $0 < \gamma < 1$ is a discount factor. When the objective function V is maximized, the balancing ability and control performance of the learning controller will also be enhanced to its maximum.

In the learning control problem of inverted pendulums, the state space is spanned by four continuous state variables. In our implementation, only two state variables, i.e., θ and $\dot{\theta}$, were used for VFA. In traditional RL algorithms for discrete actions, there are only two values of the force F, +50 N and −50 N. When the action space is limited to finite elements, the control policies of RL algorithms may not obtain near-optimal policies in the original continuous action space. For example, it will be necessary to design a continuous control policy to minimize the control objective specified by (50) and (51).

In the simulation, different RL algorithms were tested in the inverted pendulum problem. For LSPI, KLSPI, and CAPI, the maximum iteration number is 10 and 50 episodes of data samples were collected where the maximum number of time steps in each episode is 1,000. In every episode, the initial angle of the pole was set to −0.4 rad. The initial policy in the first iteration of all the API algorithms is a randomized control policy. The initial weights of Sarsa-learning were also generated from a uniform distribution. In Sarsa-learning, an episode has a maximum of 1,000 time steps and the maximum number of episodes is 50,000. The discount factor for all the RL algorithms is 0.9. One representational parameter setting with the best average performance among five independent performance tests was selected for each algorithm. In Sarsa-learning, the state and action spaces of the inverted pendulum problem were both partitioned into discrete values, and a learning rate $\alpha = 0.3$ was selected.

Similar to the simulation results in the previous mountain-car problem, it was observed that the proposed CAPI algorithms can converge to a near-optimal policy with continuous actions in a few iterations. The learning curves of CAPI, averaged over five independent runs are depicted in Fig. 10. The performance variations of CAPI with linear polynomials are shown by solid lines and those of CAPI with kernels by dotted lines, where the performance measures were computed as

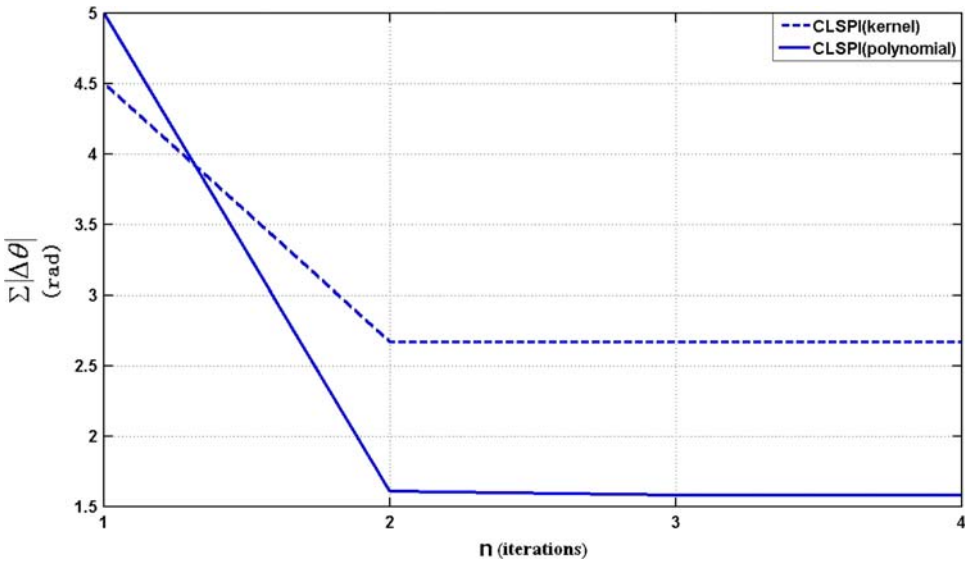

Fig. 10 The learning curves of CAPI averaged in five independent runs of pendulum control

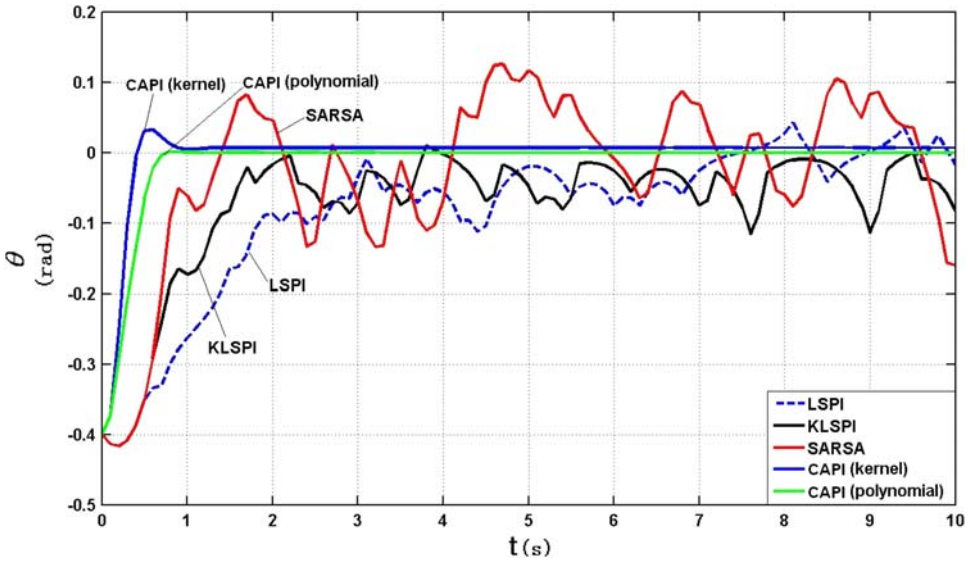

Fig. 11 Performance comparisons of the final policies obtained by different RL algorithms

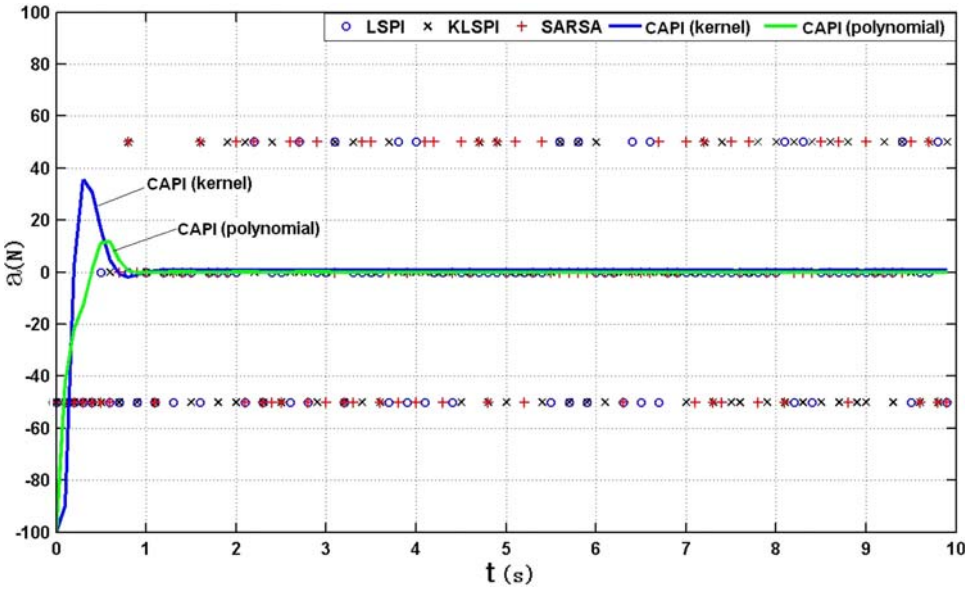

Fig. 12 Control outputs of the final policies obtained by different RL algorithms

$$J = \sum_{t=0}^{T} |\Delta\theta| \tag{52}$$

When the objective function in (51) is maximized, the performance measure in (52) will also be minimized. Thus, the final policies after the convergence of CAPI can balance the pole with minimal variations of the pole angle, as demonstrated in Fig. 11. In Fig. 11, the angle variations of the pole controlled by the final policies obtained from different RL algorithms are depicted. It is shown that using the near-optimal policies of CAPI, the pole angle can be stabilized to a very small region around the equilibrium and the angle variations during the process are minimized. For other RL algorithms with discrete actions, although the pendulum can be balanced around the equilibrium, there are large oscillations of the pole angles. Therefore, the near-optimal policies found by CAPI have much better performance than LSPI, Sarsa-learning and KLSPI.

In Fig. 12, the control outputs of the final policies obtained by different RL algorithms are depicted. Since the actions of Sarsa-learning, LSPI and KLSPI only have discrete values, their control outputs at different time steps are plotted as isolated points. The control outputs of the CAPI algorithms, either with polynomials or with kernels, are continuous, so continuously varying curves can be obtained. It is shown that the CAPI algorithm can approximate a smooth near-optimal control policy for the inverted pendulum problem in a continuous action space while the other RL algorithms can only search for control policies with discrete actions, which will limit the performance of the final policies. Thus, compared with previous RL methods, CAPI provides a very efficient RL approach

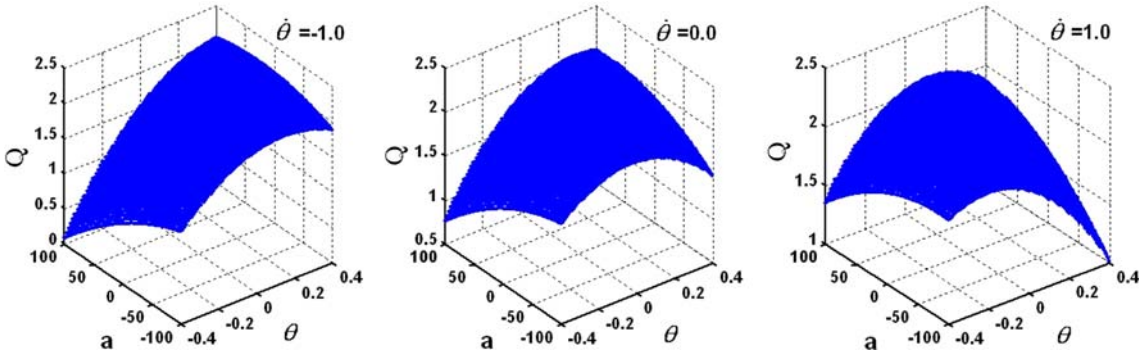

Fig. 13 Approximated action value functions of CAPI using kernels

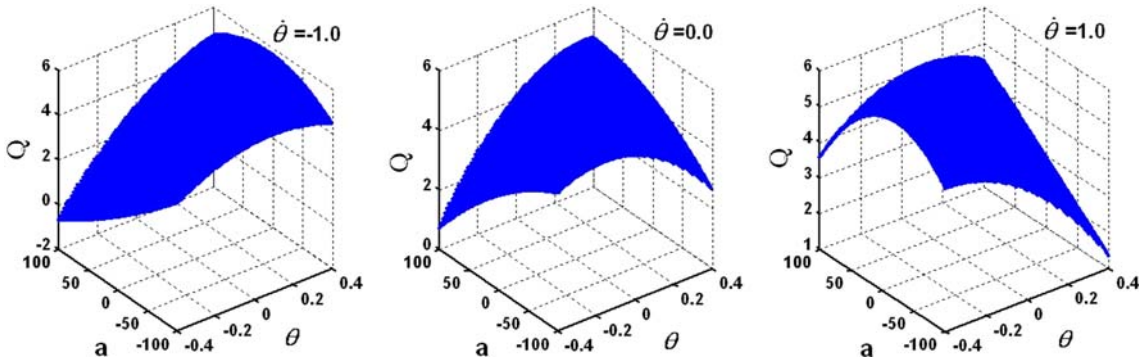

Fig. 14 Approximated action value functions of CAPI using polynomials

to find near-optimal policies for MDPs with continuous state and action spaces.

For policy evaluation in CAPI, linear polynomials and kernel functions are both good candidates for VFA, and sparse approximations with improved generalization ability can be realized by making use of the adaptive basis function selection techniques presented in Sect. 4. In CAPI with kernel functions, the number of selected elements in the data dictionary is 11, and a total number of 18 polynomials were selected in CAPI with polynomials. The following Figs. 13 and 14 show the approximated action value functions of the near-optimal policies obtained by CAPI with polynomial kernels and linear polynomials, respectively.

In Figs. 13 and 14, the approximated action value functions are depicted in the form of 3-dimensional surfaces, where the three sub-figures show the action value functions of the control policies when the pendulum's angular velocities are at -1.0, 0, and 1.0, respectively.

6 Conclusions

Reinforcement learning has been widely studied as an important class of solution techniques for complex sequential decision problems. Although much progress has been made in the last decade, it is still a challenging problem to approximate near-optimal policies for MDPs with continuous state and action spaces, where the generalization ability and learning efficiency in continuous action spaces become critical factors for the success of RL and approximate DP methods.

In this paper, the CAPI method, integrated with fast policy search and adaptive basis function selection, was proposed. Compared with previous RL methods, CAPI makes use of two key mechanisms to solve the difficulties in approximating near-optimal policies for MDPs with continuous spaces. One is that a fast policy search technique is employed to search for optimal actions in continuous action spaces, where differentiable function approximators are employed for policy evaluation. The other is that adaptive basis function selection techniques are developed for two different VFA methods, i.e., TD learning using linear polynomials and kernel functions. Based on the above two factors, the generalization ability and learning efficiency of CAPI can be well guaranteed. Simulation results on two learning control tasks of MDPs with continuous state and action spaces illustrate that CAPI not only has good convergence property but also can obtain comparable or even better performance than Sarsa-learning, LSPI and KLSPI.

The CAPI approach presented in this paper provides a very promising way to solve difficult MDPs with continuous state and action spaces. Nevertheless, there are still some problems to be studied in future work. Since the selection of basis functions is very important to the performance of TD learning algorithms, one challenging problem is to develop adaptive basis function selection methods for high-dimensional state and action spaces. In addition, it would be valuable to study the combination of CAPI and other basis function selection methods such as those in Representation Policy Iteration (Mahadevan and Maggioni 2007). For multi-dimensional action spaces, it is necessary to study fast policy search techniques to perform policy improvement efficiently. Although the superiority of CAPI has been demonstrated in some typical learning control problems, it will be desirable to evaluate CAPI and its future improved versions in more complex MDPs, and apply CAPI in real-world decision and control problems.

Acknowledgments The authors would like to thank the anonymous reviewers for their helpful comments. This research is supported by the National Natural Science Foundation of China (NSFC) under Grant 60774076 and 90820302, the Fork Ying Tung Education Foundation under Grant 114005, National Basic Research Program of China (2007CB311001), Ph.D. Programs Foundation of Ministry of Education of China and the Natural Science Foundation of Hunan Province under Grant 2007JJ3122.

References

Bach FR, Jordan MI (2002) Kernel independent component analysis. J Mach Learn Res 3:1–48

Barto AG, Sutton RS, Anderson CW (1983) Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans Syst Man Cybern 13(5):835–846

Baxter J, Bartlett PL (2001) Infinite-horizon policy-gradient estimation. J Artif Intell Res 15:319–350

Bertsekas DP, Tsitsiklis JN (1996) Neurodynamic programming. Athena Scientific, Belmont

Boyan J (2002) Technical update: least-squares temporal difference learning. Mach Learn 49(2–3):233–246

Crites RH, Barto AG (1998) Elevator group control using multiple reinforcement learning agents. Mach Learn 33(2–3):235–262

Dayan P (1992) The convergence of TD(λ) for general λ. Mach Learn 8:341–362

Dayan P, Sejnowski TJ (1994) TD(λ) converges with probability 1. Mach Learn 14:295–301

Engel Y, Mannor S, Meir R (2004) The kernel recursive least-squares algorithm. IEEE Trans Signal Process 52(8):2275–2285

Hasselt HV, Wiering M (2007) Reinforcement learning in continuous action spaces. In: 2007 IEEE symposium on approximate dynamic programming and reinforcement learning, pp 272–279

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285

Lagoudakis MG, Parr R (2003) Least-squares policy iteration. J Mach Learn Res 4:1107–1149

Lazaric A, Restelli M, Bonarini A (2008) Reinforcement learning in continuous action spaces through sequential Monte Carlo methods. In: Advances in neural information processing systems. MIT Press, Cambridge

Mahadevan S, Maggioni M (2007) Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes. J Mach Learn Res 8:2169–2231

Millan JDR, Posenato D, Dedieu E (2002) Continuous-action q-learning. Mach Learn 49(2/3):247–265

Prokhorov DV, Wunsch DC (1997) Adaptive critic designs. IEEE Trans Neural Netw 8(5):997–1007

Rasmussen CE, Kuss M (2004) Gaussian processes in reinforcement learning. In: Thrun S, Saul LK, Schölkopf B (eds) Advances in neural information processing systems, vol 16. MIT Press, Cambridge, pp 751–759

Schölkopf B, Smola A (2002) Learning with kernels. MIT Press, Cambridge

Singh SP, Jaakkola T, Littman ML, Szepesvari C (2000) Convergence results for single-step on-policy reinforcement-learning algorithms. Mach Learn 38:287–308

Sutton R (1988) Learning to predict by the method of temporal differences. Mach Learn 3(1):9–44

Sutton R (1996) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Advances in neural information processing systems, vol 8. MIT Press, Cambridge, pp 1038–1044

Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge

Tesauro G (1994) TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput 6:215–219

Tsitsiklis JN (1994) Asynchronous stochastic approximation and Q-learning. Mach Learn 16:185–202

Tsitsiklis JN, Roy BV (1997) An analysis of temporal difference learning with function approximation. IEEE Trans Autom Control 42(5):674–690

Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8:279–292

Whiteson S, Stone P (2006) Evolutionary function approximation for reinforcement learning. J Mach Learn Res 7:877–917

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8:229–256

Xu X, Hu DW, Lu XC (2007) Kernel-based least-squares policy iteration for reinforcement learning. IEEE Trans Neural Netw 18(4):973–997

Zhang W, Dietterich T (1995) A reinforcement learning approach to job-shop scheduling. In: Proceedings of the fourteenth international joint conference on artificial intelligence. Morgan Kaufmann, pp 1114–1120