Information Sciences 261 (2014) 1–31

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Reinforcement learning algorithms with function approximation: Recent advances and applications

Xin Xu *, Lei Zuo, Zhenhua Huang

College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, PR China

ARTICLE INFO

Article history:
Received 19 June 2012
Received in revised form 30 June 2013
Accepted 17 August 2013
Available online 5 September 2013

Keywords:
Reinforcement learning
Function approximation
Approximate dynamic programming
Learning control
Generalization

ABSTRACT

In recent years, the research on reinforcement learning (RL) has focused on function approximation in learning prediction and control of Markov decision processes (MDPs). The usage of function approximation techniques in RL will be essential to deal with MDPs with large or continuous state and action spaces. In this paper, a comprehensive survey is given on recent developments in RL algorithms with function approximation. From a theoretical point of view, the convergence and feature representation of RL algorithms are analyzed. From an empirical aspect, the performance of different RL algorithms was evaluated and compared in several benchmark learning prediction and learning control tasks. The applications of RL with function approximation are also discussed. At last, future works on RL with function approximation are suggested.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

Reinforcement learning (RL) is a machine learning framework for solving sequential decision problems that can be modeled as Markov Decision Processes (MDPs). In recent years, RL has been widely studied not only in the machine learning and neural network community but also in operations research and control theory [28,70,71,93,111,120,134,141]. In reinforcement learning, the learning agent interacts with an initially unknown environment and modifies its action policies to maximize its cumulative payoffs. Thus, RL provides an efficient framework to solve learning control problems which are difficult or even impossible for supervised learning and traditional dynamic programming (DP) methods. The aim of dynamic programming is to compute optimal policies given a perfect model of the environment as an MDP. Classical DP algorithms have some limitations both due to their assumption of a perfect model and due to their great computational costs. In fact, RL methods can be viewed as adaptive DP or approximate DP, with less computation and without assuming a perfect model of the environment [111]. From the perspective of automatic control, the DP/RL framework comprises a nonlinear and stochastic optimal control problem [29]. Moreover, RL is an important way for adaptive optimal control [18,112]. In the research of human brain, RL has been studied as an important mechanism for human learning [113,137–139]. From the viewpoint of artificial intelligence, RL is a basic mechanism for an agent to optimize its behavior in an uncertain environment [28,121,135,140]. Fig. 1 illustrates the main components of the RL problem, where an agent receives the state and reward information at time t, takes action a_t to make the environment's state change to next state s_{t+1} with a reward r_{t+1}. The learning objective is to find an action policy to optimize the long-term expected total or average reward from the environment.

* Corresponding author. Tel.: +86 731 84574980.
E-mail addresses: xuxin_mail@263.net, xinxu@nudt.edu.cn (X. Xu).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.08.037

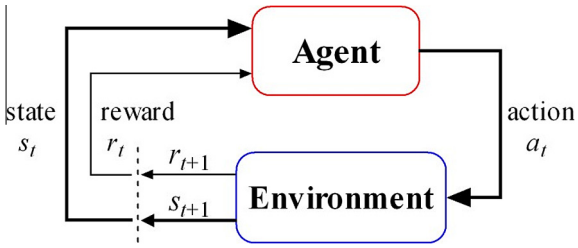


Fig. 1. The RL problem.

Function approximation has been a traditional topic in the research of machine learning. In earlier works, researchers mainly focused on function approximation techniques for supervised learning problems which can be formulated as a regression task [28]. For a regression task, the training samples are in the form of input–output pairs (x_i, y_i), $i = 1, 2, \ldots, n$. The regression problem can be stated as: given a training data set $D = f(y_i; t_i)$, $i = 1, 2, \ldots, n$, of input vectors y_i and associated targets t_i, the goal is to find a function $g(y)$ which approximates the relation inherited between the data set points and it can be used to infer the output t for a new input data point y. Generally, a regression algorithm has a loss function $L(t; g(y))$, which describes how the estimated function deviated from the true one. Many forms for the loss function have been proposed in the literature: e.g. linear, quadratic loss function, exponential, etc. [72].

To compute the optimal or near-optimal policies of MDPs, RL algorithms usually estimate the value functions of MDPs by observing data generated from state transitions. However, since no explicit teacher signals can be obtained in RL, the estimation of value functions is different from the function regression problem in supervised learning. In earlier research of RL, tabular algorithms were popularly studied, such as tabular Q-learning and tabular Sarsa-learning. In tabular RL algorithms, value functions are represented and estimated in tabular forms for each state or state-action pair. But in many real-world applications, a learning controller has to deal with MDPs with large or continuous state and action spaces. In such cases, earlier RL algorithms such as Q-learning and Sarsa-learning usually converge slowly when tabular representations of value functions are used. Since function approximation is essential to realize the generalization ability of learning machines, function approximation and generalization methods for RL have received more and more research interests in recent years [14,31,44,45,114,124,125,133,153]. Currently, there are three main categories of research work on RL methods with function approximation, i.e., policy search [9], value function approximation (VFA) [11], and actor–critic methods [13,16]. Among these three classes of approximate RL methods, the actor–critic algorithms, viewed as a hybrid of VFA and policy search, have been shown to be more effective than VFA or policy search in online learning control tasks with continuous spaces [95]. In an actor–critic learning control architecture, there is an actor for policy learning and a critic for value function approximation or policy evaluation. The policy evaluation process in the critic is also called learning prediction, which can be viewed as a sub-problem of RL.

In the past decade, the research works on RL with function approximation have been brought together with the approximate dynamic programming (ADP) community [93,134], which is to compute near-optimal solutions to MDPs with large or continuous spaces. One common objective of ADP and RL is to solve MDPs with large or continuous state and action spaces. Thus, RL and ADP provide a very promising framework for learning control problems which are difficult or even impossible for supervised learning and mathematical programming methods. Originally, RL methods mainly focused on learning control problems in MDPs without model information while ADP methods usually approximate near-optimal solutions of MDPs with some model information, which can be viewed as planning cases for sequential decision making. In this paper, we will mainly focus on recent developments in RL algorithms with function approximation and related works on ADP algorithms with function approximation will also be surveyed. Firstly, since learning prediction is an important sub-problem of RL, major advances in learning prediction algorithms with function approximation are discussed. Specifically, we put more emphasis on the developments in temporal-difference (TD) learning theory and algorithms, which are basic mechanisms for value function approximation. Secondly, learning control algorithms with function approximation are surveyed, where the main focus is put on highly efficient RL algorithms with function approximation such as fitted-Q iteration, approximate policy iteration, and adaptive critic designs (ACDs). The convergence of RL algorithms with different representations is analyzed from a theoretical point of view. Furthermore, the performance of different RL algorithms was evaluated and compared in several benchmark learning prediction and learning control tasks. The selected RL algorithms for performance comparisons have been popularly studied in the literature and most of them are state-of-the-art RL approaches with function approximation. The applications of RL will also be summarized by analyzing the advantages and disadvantages of different RL algorithms.

The rest of this paper is organized as follows. In Section 2, an overview on RL algorithms with function approximation for learning prediction is given. The RL methods with function approximation for learning control in MDPs are introduced in Section 3. Then, the theoretical results on the convergence of RL algorithms with function approximation are discussed in Section 4. The performance of different RL algorithms with function approximation was evaluated and compared in Section 5. In Section 6, the applications of RL are summarized, where the advantages and disadvantages of different algorithms are highlighted. Section 7 discusses some open problems in function approximation of RL. In the end, Section 8 draws conclusions and suggests future work.

2. RL algorithms with function approximation for learning prediction

In RL, there are two basic tasks. One is called learning prediction and the other is called learning control. The goal of learning control is to estimate the optimal policy or optimal value function of an MDP without knowing its complete model. Learning prediction aims to solve the policy evaluation problem of a stationary-policy MDP, which can also be viewed as a Markov reward process, without prior model information and it can be regarded as a sub-problem of learning control. Furthermore, in RL, learning prediction is different from that in supervised learning [14]. As pointed out by Sutton [111,114], the prediction problems in supervised learning are single-step prediction problems while those in reinforcement learning are multi-step prediction problems. To solve multi-step prediction problems, a learning system must predict outcomes that depend on a future sequence of decisions [142]. Therefore, the theory and algorithms for multi-step learning prediction become an important topic in RL and much research work has been done in the literature [114,124]. In this section, we will focus on learning prediction methods with function approximation. At first, some introduction on the Markov reward process model for learning prediction is given.

2.1. Markov reward process

Markov reward processes are popular stochastic models for sequential modeling and decision making. A Markov reward process can be modeled as a tuple $\{X,R,P\}$, where X is the state space, R is the reward function, P is the state transition probability. Let $\{x_t|t = 0, 1, 2, \ldots; x_t \in X\}$ denote a state sequence generated by a Markov reward process. For each state transition from x_t to x_{t+1}, a scalar reward r_t is defined. The state transition probabilities satisfy the following property:

$$P\{x_{t+1}|x_t, x_{t-1}, \ldots, x_1, x_0\} = P\{x_{t+1}|x_t\} \tag{1}$$

Let the trajectory generated by the Markov chain be denoted by $\{x_t|t = 0, 1, 2, \ldots, x_t \in X\}$.The dynamics of the Markov chain can be described by a transition probability matrix P whose (i,j)th entry, denoted by p_{ij}, is the transition probability for $x_{t+1} = j$ given that $x_t = i$. For each state transition from x_t to x_{t+1}, a scalar reward r_t is defined. The expected average reward or value starting from a state x is defined as:

$$\rho(x) = \lim_{N\to\infty} \frac{1}{N} E\left[\sum_{t=0}^{N} r_{t+1} \middle| x_0 = x\right] \tag{2}$$

For the discounted case, the value function of each state is defined as follows:

$$V(x) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| x_0 = x\right\} \tag{3}$$

where $1 > \gamma \geqslant 0$ is a discount factor, and the expectation is with respect to the state transition probabilities.

Similarly, the state-action value function $Q(x,a)$ is defined as the expected, discounted total rewards when taking action a in state x:

$$Q(x,a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| x_0 = x, a_0 = a\right] \tag{4}$$

2.2. Monte-Carlo methods and tree-based batch RL

Since the state value is defined as the expectation of the stochastic rewards when the process starts from the state, a simple method of value estimation is to average over multiple independent runs of the process. This is one example of the so-called Monte-Carlo method. Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns [111]. To ensure that well-defined returns are available, Monte Carlo methods are usually defined for episodic tasks. It is assumed that experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. The value estimates and policies are only changed after the end of an episode. Monte Carlo methods are thus incremental in an episode-by-episode style, but not in a step-by-step style. Unfortunately, the variance of the returns can be high, which makes convergence slow.

Let R_t denote the total rewards following each state x_t. Since the real state value is the expected total rewards after the state, the learning target $v_t = R_t$ is an unbiased estimate of $V(x_t)$. Thus, the following gradient-descent rule of Monte Carlo state-value prediction will converge to a locally optimal solution [111].

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t(R_t - \widehat{V}(x_t, \vec{\theta}_t))\partial\widehat{V}/\partial\vec{\theta}_t \tag{5}$$

In batch mode, it can be computed by approximating the action value function called Q-function based on a set of four-tuples (x_t, a_t, r_t, x_{t+1}) where x_t denotes the system state at time t, a_t the control action taken, r_t the immediate reward obtained and x_{t+1} the successive state, and by using the control policy from this Q-function. The Q-function approximation can be esti-

mated from the limit of a sequence of (batch mode) supervised learning problems. Based on this idea, the use of several classical tree-based supervised learning methods (CART, Kd-tree, tree bagging) has been studied [45,125].

2.3. TD learning algorithms with linear function approximation

As a class of multi-step learning prediction methods, temporal-difference (TD) learning [114] was studied and applied in the early research of machine learning, including the well-known checkers-playing program [79,100]. In 1988, Sutton presented the first formal description of temporal-difference methods and the TD (λ) algorithm [114]. Convergence results were established for tabular temporal-difference learning algorithms where the cardinality of tunable parameters is the same as that of the state space [35,61,114,136]. Since many real-world applications have large or continuous state space, value function approximation (VFA) methods need to be studied in those cases.

Consider a general linear function approximator with a fixed basis function vector

$$\phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_n(x))^T \tag{6}$$

The approximated value function is given by

$$\widehat{V}_t(x) = \phi^T(x)\vec{\theta}_t \tag{7}$$

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

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t(r_t + \gamma\phi^T(x_{t+1})\vec{\theta}_t - \phi^T(x_t)\vec{\theta}_t)\vec{z}_{t+1} \tag{8}$$

where the eligibility trace vector $\vec{z}_t(x_t) = (z_{1t}(x_t), z_{2t}(x_t), \ldots, z_{nt}(x_t))^T$ is defined as

$$\vec{z}_{t+1} = \gamma\lambda\vec{z}_t + \phi(x_t) \tag{9}$$

In [124], the above linear TD (λ) algorithm is proved to converge with probability 1 under certain assumptions and the limit of convergence $\vec{\theta}^*$ is also derived, which satisfies the following equation.

$$E_0[A(X_t)]\vec{\theta}^* - E_0[b(X_t)] = 0 \tag{10}$$

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

$$A(X_t) = \vec{z}_t(\phi^T(x_t) - \gamma\phi^T(x_{t+1})) \tag{11}$$

$$b(X_t) = \vec{z}_t r_t \tag{12}$$

The solution of Eq. (10) is called the fixed-point solution of linear TD learning. To improve the efficiency of linear TD (λ) algorithms, least-squares methods were used with the linear fixed-point TD (0) algorithm, and the LS-TD (0) and RLS-TD (0) algorithms were proposed in [27]. In LS-TD (0) and RLS-TD (0), the following quadratic objective function was defined.

$$J = \sum_{t=1}^{T-1}\left[r_t - (\phi_t - \gamma\phi_{t+1})^T\vec{\theta}\right]^2 \tag{13}$$

By employing the instrumental variables approach [109], the least-squares fixed-point solution of (13) is given as

$$\vec{\theta}_{LS-TD(0)} = \left(\sum_{t=1}^{T}(\phi_t(\phi_t - \gamma\phi_{t+1})^T)\right)^{-1}\left(\sum_{t=1}^{T}\phi_t r_t\right) \tag{14}$$

where ϕ_t is the instrumental variable chosen to be uncorrelated with the input and output noises.

In RLS-TD (0), recursive least-squares methods are used to decrease the computational burden of LS-TD (0). The convergence (with probability one) of LS-TD (0) and RLS-TD (0) was proved for periodic and absorbing Markov chains under certain assumptions [27]. In [23], LS-TD (λ) was proposed by solving (10) directly and the model-based property of LS-TD (λ) was also analyzed. However, for LS-TD (λ), the computation per time-step is $O(K^3)$, i.e., the cubic order of the state feature number. Therefore the computation required by LS-TD (λ) increases very fast when K increases, which is undesirable for online learning.

In [142], Xu et al. proposed the RLS-TD (λ) algorithm and it was shown that the RLS-TD (λ) algorithm is superior to conventional TD (λ) algorithms in data efficiency and it also eliminates the design problem of the step sizes in linear TD (λ) algorithms. The weight update rules of RLS-TD (λ) are given by

$$K_{t+1} = P_t\vec{z}_t/(\mu + (\phi^T(x_t) - \gamma\phi^T(x_{t+1}))P_t\vec{z}_t) \tag{15}$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + K_{t+1}(r_t - (\phi^T(x_t) - \gamma\phi^T(x_{t+1}))\vec{\theta}_t) \tag{16}$$

$$P_{t+1} = \frac{1}{\mu}[P_t - P_t\vec{z}_t[\mu + (\phi^T(x_t) - \gamma\phi^T(x_{t+1}))P_t\vec{z}_t]^{-1}(\phi^T(x_t) - \gamma\phi^T(x_{t+1}))P_t] \tag{17}$$

where for the standard RLS-TD (λ) algorithm, $\mu = 1$; for the general forgetting factor RLS-TD (λ) case, $0 < \mu \leqslant 1$.

In order to improve robustness and efficiency, Geramifard et al. [54] proposed an incremental version of LSTD, called iLSTD, which computes the matrix $A(x_t)$ and vector $b(X_t)$ with one dimension of the parameter vector being updated in each time step.

As analyzed in [54], the fixed-point (FP) solution for (10) minimizes the projected Bellman residual:

$$J = \min_w \|\Pi T^\pi(\hat{V}) - \hat{V}\|_\rho \tag{18}$$

where $\Pi = \Phi(\Phi^T\Phi)^{-1}\Phi^T$ is the projection operator determined by the feature vector. $\Phi = [\vec{\phi}(x_1), \vec{\phi}(x_2), \ldots, \vec{\phi}(x_n)]^T$, $\hat{V} = \Phi W$ is the vector of approximated value functions for all the finite states, T^π is the Bellman operator defined by $T^\pi(\hat{V}) = R^\pi + P^\pi\hat{V}$, P^π is the state transition matrix.

Another approach is to minimize the Bellman residual (BR). This technique computes a solution by minimizing the magnitude of the Bellman residual, where the errors for each state are weighted according to distribution ρ:

$$\min_w \|T^\pi(\hat{V}) - \hat{V}\|_\rho = \min_w \|R^\pi + \gamma P^\pi \Phi w - \Phi w\|_\rho \tag{19}$$

The least-squares solution is to minimize $\|A_{BR}w - b_{BR}\|_\rho$ where:

$$\begin{aligned} A_{BR} &= \Phi^T(I - \gamma P^\pi)^T D_\rho (I - \gamma P^\pi)\Phi, \\ b_{BR} &= \Phi^T(I - \gamma P^\pi)^T D_\rho R^\pi. \end{aligned} \tag{20}$$

In [64], by including the norm of the Bellman residual in their objective functions, hybrid algorithms were proposed to protect against large residual vectors. They also have the flexibility of finding solutions that are almost fixed points but have more desirable properties (smaller Bellman residuals).

$$\min_w \left[\beta \left\| T^\pi(\hat{V}) - \hat{V} \right\|_\rho^2 + (1-\beta)\left\| \prod_\rho (T^\pi(\hat{V}) - \hat{V}) \right\|_\rho^2\right] \tag{21}$$

Each approximate policy evaluation algorithm uses the Bellman equation in different ways to compute a value function. There is an intuitive geometric perspective to the algorithms when using linear function approximation. As discussed in [64], the Bellman equation with linear function approximation has three components: $\hat{V}$, $T^\pi\hat{V}$ and $\Pi T^\pi\hat{V}$. These components geometrically form a triangle where $\hat{V}$ and $\Pi T^\pi\hat{V}$ reside in the space $[\Phi]$ spanned by Φ while $T^\pi\hat{V}$ is, in general, outside this space. This is illustrated in the leftmost triangle of Fig. 2.

The three-dimensional space in Fig. 2 is the space of exact value functions while the two-dimensional plane represents the space of approximate value functions in $[\Phi]$. The angle between subspace $[\Phi]$ and the vector $\Pi T^\pi\hat{V} - \hat{V}$ is denoted as θ. The BR and FP solutions minimize the length of different sides of the triangle. The second triangle in Fig. 2 shows the BR solution, which minimizes the length of $T^\pi\hat{V} - \hat{V}$. The third (degenerative) triangle shows the FP solution, which minimizes the length of $\Pi T^\pi\hat{V} - \hat{V}$. This length is 0 which means $\theta_{FP} = 90°$. The fourth triangle shows the hybrid solution in [64], which minimizes a combination of the lengths of the two sides. In general, θ_H lies between θ_{BR} and 90°. The hybrid solution allows for controlling the shape of this triangle.

2.4. Kernel-based TD learning

Kernel methods or kernel machines [101] were popularly studied to realize nonlinear and non-parametric versions of conventional supervised or unsupervised machine learning algorithms. The main idea behind kernel machines is that inner products in a high-dimensional feature space can be represented by a Mercer kernel function so that conventional learning algorithms in linear spaces may be transformed to nonlinear algorithms without explicitly computing the inner products in high-dimensional feature spaces. This idea, which is usually called the "kernel trick", has been widely applied in various kernel-based supervised and unsupervised learning problems. In supervised learning, the most popular kernel machines include support vector machines (SVMs) and the Gaussian process model for regression, which have been applied to many classification and regression problems [101,102,128]. In unsupervised learning, kernel principal component analysis (KPCA) and kernel independent component analysis have also been studied by many researchers [8,101].

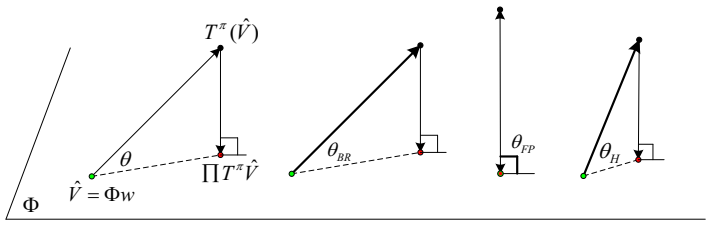


Fig. 2. The triangle on the left shows the general form of the Bellman equation. The other three triangles correspond to the different approximate policy evaluation algorithms where the bold lines indicate what is being optimized [64].

According to the Mercer Theorem [128], there exists a Hilbert space H and a mapping ϕ from X to H such that

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{22}$$

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

Kernel-based reinforcement learning (KBRL) computes a value function offline by generalizing value function updates from a given sample of transitions over an instance-based representation [86]. KBRL is noteworthy for its theoretical guarantee of convergence to the optimal value function as its sample size increases, under appropriate assumptions, but it does not answer the exploration question of how to efficiently gather the data online. To realize efficient and convergent TD learning with kernels, Xu et al. [144] presented a class of kernel-based least-squares TD learning algorithms with eligibilities, which is called KLS-TD (λ). The idea of KLS-TD (λ) is to make use of Mercer kernel functions to implement least-squares TD learning in a high-dimensional nonlinear feature space produced by a kernel-based feature mapping. Thus, compared to conventional linear TD (λ) and LS-TD (λ) algorithms, better performance of approximation accuracy can be obtained for KLS-TD (λ) in nonlinear VFA problems.

Let

$$A_T = \sum_{t=1}^{N} \vec{k}(x_t)[\vec{k}^T(x_t) - \gamma \vec{k}^T(x_{t+1})] \tag{23}$$

$$b_T = \sum_{t=1}^{N} \vec{k}(x_t) r_t \tag{24}$$

where $\vec{k}(x)$ is a kernel-based feature vector, N is the total number of samples.

Then, the kernel-based least-squares fixed-point solution to the TD learning problem is as follows:

$$\vec{\alpha} = A_T^{-1} b_T \tag{25}$$

To improve the generalization ability and reduce the computational complexity of kernel machines, in [144], the ALD-based sparsification procedure was introduced for regularizing the kernel machines. After collecting a set of data samples and initialize a dictionary with the first sample, ALD-based sparsification mainly includes two steps. The first step is to compute the following optimization solutions

$$\Delta_t = \min_c \left\| \sum_{x_j \in D_t} c_j \phi(x_j) - \phi(x_t) \right\|^2 \tag{26}$$

Due to the kernel trick, after substituting [22] into (26), we can obtain

$$\Delta_t = \min_c \{ c^T K_{t-1} c - 2 c^T k_{t-1}(x_t) + k_{tt} \} \tag{27}$$

where $[K_{t-1}]_{i,j} = k(x_i, x_j)$, x_i ($i = 1, 2, \ldots, d(t-1)$) are the elements in the dictionary, $d(t-1)$ is the length of the data dictionary, $k_{t-1}(x_t) = [k(x_1, x_t), k(x_2, x_t), \ldots, k(x_{d(t-1)}, x_t)]^T$, $c = [c_1, c_2, \ldots, c_d]^T$ and $k_{tt} = k(x_t, x_t)$.

The second step of ALD-based sparsification is to update the data dictionary by comparing Δ_t with a predefined threshold μ. If $\Delta_t < \mu$, the dictionary is unchanged, otherwise, x_t is added to the dictionary, i.e., $D_t = D_{t-1} \cup x_t$.

2.5. Gradient TD

The computational complexity of least-squares TD methods will be larger than that of TD (λ). Two recent algorithms were proposed by Sutton et al. [115,116], which overcome the instability issue, converge to the TD (λ) solutions in the on-policy case, and yet the computational costs are almost the same as TD (λ).

Define the temporal difference at time t as $\delta_{t+1}(\theta) = r_{t+1} + \gamma \hat{V}(x_{t+1}) - \hat{V}(x_t)$, where $\hat{V}(x_t) = \phi_t^T(\theta) w_t$. Rewrite J defined in (18) in the following form:

$$J(\theta) = E[\delta_{t+1}(\theta)\phi_t]^T E\left[\phi_t \phi_t^T\right]^{-1} E[\delta_{t+1}(\theta)\phi_t] \tag{28}$$

In GTD2 [115], the update of θ_t is based on the negative stochastic gradient of J defined in (28) assuming that $w_t \approx w(\theta_t)$, while w_t is updated so that for any fixed θ, w_t would converge almost surely to $w(\theta)$:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha_t \left(\phi_t - \gamma \phi'_{t+1}\right) \phi_t^T w_t, \\ w_{t+1} &= w_t + \beta_t \left(\delta_{t+1}(\theta_t) - \phi_t^T w_t\right) \phi_t. \end{aligned} \tag{29}$$

In TDC ("temporal difference learning with corrections"), the gradient is [115]

$$\nabla_\theta J(\theta) = -2\left(E[\delta_{t+1}(\theta)\phi_t] - \gamma E\left[\phi'_{t+1}\phi_t^T\right] w(\theta)\right) \tag{30}$$

Leaving the update w_t unchanged, the update rules in TDC are as follows:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t\big(\delta_{t+1}(\theta_t)\phi_t - \gamma\phi'_{t+1}\phi_t^T w_t\big),\\ w_{t+1} &= w_t + \beta_t\big(\delta_{t+1}(\theta_t) - \phi_t^T w_t\big)\phi_t.\end{aligned} \tag{31}$$

In TDC, the update of w_t should have larger step-sizes than the update of θ_t:$\alpha_t = o(\beta_t)$. Thus, TDC is a class of the so-called two-timescale stochastic approximation algorithms [20,21,120]. It was proved that when the above condition as well as the standard RM conditions are also satisfied by both step-size sequences, $\theta_t \to \theta$ holds almost surely [115]. The algorithms can also be extended to use eligibility traces [73,120].

2.6. Learning prediction with Gaussian Processes [43,96]

Gaussian Processes (GPs) have been widely studied for classification and regression. Based on a probabilistic generative model, GP methods can generate a full posterior distribution rather than a point estimate as in non-Bayesian methods. In [43], the use of GPs for solving the RL problem of value estimation was introduced. Since GPs belong to the family of kernel machines, they bring in RL the high, and quickly growing representational flexibility of kernel based representations. In [96], Rasmussen and Kuss used Gaussian process (GP) models for two distinct purposes: first to model the dynamics of the system and secondly to use GP for representing the value function.

In analogy to GP regression, Engel et al. [43] imposed a Gaussian prior over value function, i.e., $V \sim N(0, k(\cdot,\cdot))$, which means that V is a GP for which, $E(V(x)) = 0$ and $E(V(x)V(x')) = k(x,x')$ for all $x, x' \in X$. The form of the function k should reflect the prior knowledge on the similarity of states in the domain at hand. The following generative model for the sequence of rewards corresponding to the trajectory $\{x_1, x_2, \ldots, x_t\}$ was proposed in [43]:

$$R(x_i, x_{i+1}) = V(x_i) - \gamma V(x_{i+1}) + N(x_i) \tag{32}$$

where N is a white Gaussian noise process, i.e., $N \sim N(0, \Sigma)$ with $\Sigma(x, x') = \sigma_0(x)^2\delta(x - x')$, where δ denotes the Dirac delta function. For a finite state sequence of length t, the (finite dimensional) random processes are defined as follows:

$$\begin{aligned}R_t &= (R(x_1), \ldots, R(x_t))^T,\\ V_t &= (V(x_1), \ldots, V(x_t))^T,\\ N_t &= (N(x_1), \ldots, N(x_t))^T,\end{aligned} \tag{33}$$

and the vector and matrices are (respectively) defined as

$$\begin{aligned}\mathrm{k}_t(x) &= (k(x_1, x), \ldots, k(x_t, x))^T,\\ \mathrm{K}_t &= [\mathrm{k}_t(x_1), \ldots, \mathrm{k}_t(x_t)],\\ \Sigma_t &= \mathrm{diag}(\sigma_0^2, \ldots, \sigma_0^2),\end{aligned} \tag{34}$$

where diag $(\cdot)$ denotes a diagonal matrix whose diagonal is the argument vector. Using these definitions one can obtain

$$\begin{pmatrix} N_t \\ V_t \end{pmatrix} \sim N\left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \Sigma_t & 0 \\ 0 & \mathrm{K}_t \end{bmatrix}\right\}. \tag{35}$$

Defining the $(t-1) \times t$ matrix

$$\mathrm{H}_t = \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & \ldots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \ldots & 1 & -\gamma \end{bmatrix}, \tag{36}$$

Eq. (32) can be states as

$$R_{t-1} = \mathrm{H}_t V_t + N_{t-1}. \tag{37}$$

Using standard results on jointly Gaussian random variables, the following relation can be obtained [43]

$$\begin{pmatrix} R_{t-1} \\ V(x) \end{pmatrix} \sim N\left\{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{bmatrix} \mathrm{H}_t\mathrm{K}_t\mathrm{H}_t^T + \Sigma_{t-1} & \mathrm{H}_t\mathrm{k}_t(x) \\ \mathrm{k}_t(x)^T\mathrm{H}_t^T & k(x,x) \end{bmatrix}\right\},$$

and the posterior distribution of the value at some point x, conditioned on the observed sequence of rewards $\vec{r}_{t-1} = (r_1, \ldots, r_{t-1})^T$ is given by [43]

$$(V(x)|R_{t-1} = \vec{r}_t - 1) \sim N\{\hat{v}_t(x), p_t(x)\}, \tag{38}$$

where

$$\begin{aligned}\hat{v}_t(x) &= \mathrm{k}_t(x)^T \mathrm{H}_t^T \mathrm{Q}_t \vec{r}_t - 1,\\ p_t(x) &= k_{xx} - \mathrm{k}_t(x)^T \mathrm{H}_t^T \mathrm{Q}_t \mathrm{H}_t \mathrm{k}_t(x),\end{aligned} \tag{39}$$

with $\mathrm{Q}_t = \left(\mathrm{H}_t \mathrm{K}_t \mathrm{H}_t^T + \Sigma_{t-1}\right)^{-1}$, and $k_{xx} = k(x,x)$.

Both the probabilistic generative model and the corresponding Gaussian Process Temporal Differences (GPTD) algorithm proposed in [43] have two problems to be considered. First, the original model is strictly correct only if the state transitions of the underlying MDP are deterministic, and if the rewards are corrupted by white Gaussian noise. While the second assumption is relatively innocuous, the first is a serious handicap to the applicability of the GPTD model to general MDPs. Secondly, the GPTD algorithm just addresses the value estimation problem. To solve learning control problems, it should be combined with API methods or other actor–critic methods.

3. RL algorithms with function approximation for learning control in MDPs

The learning control problem in MDPs is to approximate the optimal value function or optimal policy without knowing the MDP model as *a priori*. Until recently, the three main categories of approximate RL methods for learning control have included value function approximation (VFA) [11], policy search [17], and actor–critic methods [69,19,71]. Unlike VFA-based methods, actor–critic algorithms approximate the value functions and policies of an MDP separately to encourage the realization of generalization in MDPs with large or continuous spaces. Many recent studies of actor–critic methods have focused on adaptive critic designs (ACDs), which usually require an approximated model of the plant dynamics. In addition, relational reinforcement learning (RRL) has been studied for learning control in domains that exhibit structural properties and in which different kinds of related objects exist. In this section, we will introduce the MDP model and traditional dynamic programming methods at first. Then, we will put our focus on approximate RL methods for learning control, including VFA methods, policy gradient methods, approximate policy iteration, actor–critic methods, and relational RL. Moreover, some discussions on the relationship between planning and learning will also be given.

3.1. MDP

A Markov decision process is denoted as a tuple $\{X,A,R,P\}$, where X is the state space, A is the action space, P is the state transition probability and R is the reward function. When referring to a policy π, we use $\pi(a|x)$ to denote the probability of selecting action a in state x by π. A deterministic policy directly maps states to actions, denoted as:

$$a_t = \pi(x_t) \quad t \geqslant 0 \tag{40}$$

When the actions a_t ($t \geqslant 0$) satisfy (40), policy π is followed in the MDP M. A stochastic stationary policy π is said to be followed in the MDP M if $a_t \sim \pi(a|x_t), t \geqslant 0$.

The objective of a learning controller is to estimate the optimal policy π^* satisfying:

$$J_{\pi^*} = \max_{\pi} J_\pi = \max_{\pi} E^\pi\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right] \tag{41}$$

or

$$J_{\pi^*} = \max_{\pi} \lim_{N\to\infty} \frac{1}{N} E^\pi\left[\sum_{t=0}^{N} r_{t+1}\right] \tag{42}$$

where $0 < \gamma < 1$ is the discount factor and r_t is the reward at time-step t, $E^\pi[\cdot]$ stands for the expectation with respect to the policy π and the state transition probabilities, and J_π is the averaged or expected total discounted reward along the state trajectories by following policy π. In this paper, J_π is also called the performance value of policy π.

For the case of average rewards, the relative value function is defined as:

$$\widetilde{V}^\pi(x) = \sum_{t=0}^{\infty} E[r_t - \rho^\pi | x_0 = x] \tag{43}$$

where

$$\rho^\pi = \lim_{N\to\infty} \frac{1}{N} E^\pi\left[\sum_{t=0}^{N} r_t\right] \tag{44}$$

For the discounted case, the state value function for a stationary policy π and the optimal state value function for the optimal policy π^* are defined as follows:

$$V^{\pi}(x) = E^{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\middle| x_0 = x\right] \tag{45}$$

$$V^{*}(x) = E^{\pi*}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\middle| x_0 = x\right] \tag{46}$$

According to the theory of dynamic programming [18], the optimal value function satisfies the following Bellman equation

$$V^{*}(x) = \max_{a}[R(x,a) + \gamma E[V^{*}(x')]] \tag{47}$$

where $R(x,a)$ is the expected reward received after taking action a in state x.

The state-action value function under policy π is defined as

$$Q^{\pi}(x,a) = E^{\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\middle| x_0 = x, a_0 = a\right] \tag{48}$$

The optimal state-action value function is

$$Q^{*}(x,a) = \max_{\pi} Q^{\pi}(x,a) \tag{49}$$

When $Q^*(x,a)$ is obtained, the optimal policy is easy to be computed by

$$\pi^{*}(x) = \arg\max_{a} Q^{*}(x,a) \tag{50}$$

3.2. Planning and dynamic programming [120]

Define the Bellman optimality operator, T^*: $R \to R$, by

$$(T^{*}V)(x) = \max_{a\in A}\left\{R(x,a) + \gamma\sum_{y\in X}P(x,a,y)V(y)\right\}, \quad x \in X \tag{51}$$

By making use of the operator T^*, Eq. (47) can be re-written as

$$T^{*}V^{*} = V^{*} \tag{52}$$

If $0 < \gamma < 1$, then T^* is a maximum-norm contraction, and the fixed-point equation $T^*V = V$ has a unique solution [120]. The above property is the basic principle for the value iteration and policy iteration algorithms.

In value iteration, the value functions are updated as

$$V_{k+1} = T^{*}V_k, \quad k \geqslant 0, \tag{53}$$

where V_0 is arbitrary.

Value iteration can also be applied to action-value functions, where the update rule is

$$Q_{k+1} = T^{*}Q_k, \quad k \geqslant 0, \tag{54}$$

which converges to Q^* at a geometric rate. The idea is that once V_k (or Q_k) is close to V^* (resp., Q^*), a policy that is greedy with respect to V_k (resp., Q_k) will be near-optimal [120]. Fix an action-value function Q and let π be a greedy policy w.r.t Q. The following relation holds [108,120]:

$$V^{\pi}(x) \geqslant V^{*}(x) - \frac{2}{1-\gamma}\left\|Q - \underset{a\in A}{Q^{*}}\right\|_{\infty}, \quad x \in X \tag{55}$$

The main procedures in policy iteration include the following steps [120]. Fix an arbitrary initial policy π_0. At iteration $k > 0$, compute the action-value function for policy π_k (called policy evaluation). Next, define π_{k+1} as a greedy policy with respect to the action-value function (called policy improvement). After k iterations, policy iteration generates a policy not worse than the greedy policy w.r.t. to the value function computed using k iterations of value iteration if the same initial value function is used. Nevertheless, the computational cost of a single step in policy iteration is much higher than that of one update in value iteration [120].

3.3. VFA for learning control

In previous work of value function approximation in RL, a class of approximated gradient descent method is commonly used, which can be called the direct gradient method [11] and has the following form.

$$\Delta\theta = \alpha_t[r(x,a) + \gamma\widehat{Q}(x',\theta,a') - \widehat{Q}(x,\theta,a)]\frac{\partial\widehat{Q}(x,\theta,a)}{\partial\theta} \tag{56}$$

where $\widehat{Q}(x,\theta,a)$ is the estimated action value function, (x,a) and (x',a') are two successive state-action pairs and θ is the weight vector.

Since the above direct gradient learning rule is not based on any objective function of approximation errors, convergence results cannot be obtained by the well-known stochastic gradient descent theory. For direct gradient Sarsa-learning, when linear function approximators are used and the policy is stationary, it is equivalent to linear TD (0) learning algorithm and has been proved to converge [107]. But for general cases with nonstationary policies, both direct gradient Q-learning and Sarsa-learning algorithms have been shown to be unable to converge to any policy for simple MDPs and simple function approximators [11,124].

In [11], the following objective function is chosen to compute the residual gradient for MDPs with stationary policies.

$$J = \frac{1}{2}\sum_{x} E\left[r(x,a) + \gamma \sum_{\pi(x')} Q(x',a') - Q(x,a)\right]^2 \tag{57}$$

where $\pi(x')$ is a stationary policy which does not change over time. Although the residual gradient learning algorithm has been proved to converge to a local minimum of the objective function (57), it cannot guarantee the convergence of value function approximation when the policy changes in order to approximate the optimal policy of an MDP because the above residual gradient rule is only for the case of stationary policies.

Fitted Q-iteration [45] is to approximate the optimal action-value function $Q(x,a)$ and mimics value iteration. Since computing the Bellman operator applied to the last iterate at any point involves evaluating a high-dimensional integral, a Monte-Carlo approximation can be used together with a regression procedure. For this purpose, a set of samples D is generated: $D = \{(x_1,a_1,R_1,x'_1),\ldots,(x_N,x_N,R_N,x'_N)\}$. Here, R_t, x'_t are the reward and the next-state when action a_t is chosen in state $x_t : x'_t \sim P(\cdot|x_t,a_t)$. For the sake of simplicity, it is assumed that the actions are generated by some fixed stochastic stationary policy π: $a_t \sim \pi(\cdot|x_t)$ and $\{x_t\}$ is an i.i.d. sequence. The regularized fitting procedure Fitted-Q studied in [5] is penalized least-squares:

$$Q_{k+1} = \arg\min_{Q\in F^M} \frac{1}{M_k} \sum_{i=N_k}^{N_k+M_k-1} \left[R_i + \gamma \max_{a'\in A} Q_k(x'_i,a') - Q(x_i,a_i)\right]^2 + \lambda Pen(Q) \tag{58}$$

where in the kth iteration, samples are used with index $N_k + M_k = N_{k+1} - 1 > i \geqslant N_k$, $Pen(Q)$ is a penalty term.

3.4. Relational reinforcement learning

Relational reinforcement learning (RRL) is to solve learning control problems that have structural properties and there are multiple related objects. In this kind of environment, most traditional reinforcement learning techniques may have low efficiency.

To estimate an action-value function or Q-function, RRL algorithms usually use incremental relational regression to make use of the relational property of the samples [51]. The estimated Q-function is then used to select subsequent actions. Three regression algorithms [40,41,50] have been developed in a general RRL framework: the TG algorithm, which incrementally builds first order regression trees, an instance based algorithm called RIB and a kernel based algorithm KBR that uses Gaussian processes as the regression technique.

Tree based regression. As discussed in [51], RRL-TG [40] uses an incremental first-order regression tree algorithm to construct the Q-function. However, the performance of the algorithm is greatly influenced by the order of sample representation and more training episodes may be needed to find a good policy [39,50].

Instance based regression. RRL-RIB [41] realizes the generalization ability via regression based on relational instances. The instance based regression has robust performance but a first-order distance needs to be defined [51].

Kernel based regression. RRL-KBR [50] uses Gaussian processes as the regression technique. Gaussian processes require a positive definite covariance function to be defined between the example descriptions. Because of the use of relational representations in the RRL system, kernels for structured data have to be used to fulfill this task. Possible candidates here are the convolution kernel [57] or kernels defined on graphs [51]. Since Gaussian processes are a class of Bayesian techniques, the KBR algorithm offers more than just a basic prediction of the Q-value of a new example. It can also give the expected accuracy of this estimate, which can be used to guide exploration.

3.5. Approximate policy iteration

As a popular method studied in operations research, policy iteration (PI) can also be viewed as a class of actor–critic learning algorithms since in PI, the value functions and the policies are approximated separately, which correspond to the critic and the actor, respectively [144]. In [69], based on the work of least-squares temporal difference learning methods in [27], the least-squares policy iteration (LSPI) algorithm was proposed. In LSPI, the data efficiency of least-squares temporal difference learning, i.e. the LS-TD (λ) algorithm, is employed and it offers an off-policy RL method with better properties in convergence, stability, and sample complexity than previous RL algorithms.

Approximate policy iteration (API) is closely related to the actor–critic learning control architecture of RL, which can be depicted in the following Fig. 3.

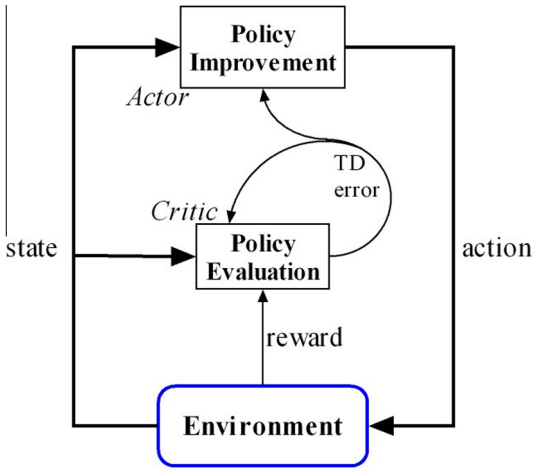


Fig. 3. Approximate policy iteration and actor–critic learning [111].

In Fig. 3, approximate policy iteration is implemented in an actor–critic learning control architecture. 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 value functions $Q^{\pi[t]}$ without any model information of the underlying MDPs. Based on the estimation of $Q^{\pi[t]}$, the policy improvement in the actor produces a greedy policy $\pi[t+1]$ over $Q^{\pi[t]}$ as

$$\pi[t+1] = \arg\max_a Q^{\pi[t]}(x,a) \tag{59}$$

Thus, the greedy policy $Q^{\pi[t+1]}$ is a deterministic policy and when the value function $Q^{\pi[t]}$ approximates $\pi[t]$ very well, $\pi[t+1]$ will be at least as good as $\pi[t]$ if not better. This iteration process is repeated until there is no change between the policies $\pi[t]$ and $\pi[t+1]$. After the convergence of policy iteration, the optimal policy may be obtained, usually within a very few iterations. However, the convergence of policy iteration greatly relies on the approximation precision of the real value functions of policies. 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 the performance of policy iteration will be very satisfactory. Thus the success of approximate policy iteration will mainly depend on the performance of TD learning algorithms for policy evaluation. In [144], Xu et al. presented a new kernel-based TD-learning algorithm, i.e. the KLSTD-Q algorithm, for kernel-based least-squares policy iteration (KLSPI).

In [76]. by modeling the state space topology using a graph, two Laplacian operators are introduced for the basis construction and the representation policy iteration (RPI) algorithm was proposed. For some excellent spectral properties of the Laplacian operators, basis functions which can also be termed as proto-value functions are learned automatically by spectral analysis from the two operators. Let $G = (V,E,W)$ denote an undirected or directed graph with vertices V, edges E and weight matrix W whose entry w_{ij} means the weight on edge $(i,j) \in E$. Samples obtained from a sub-sampling step construct the vertices of the graph and edges combine them by the weights. An undirected or directed graph can be constructed by connecting two temporally successive states with a unit cost edge in discrete state space or using a local distance measure such as k-nearest neighbor (knn) to connect states in continuous spaces. From the constructed graph, the combinatorial or normalized Laplacian operator, is used to form the suite of PVFs by computing the smoothest eigenvectors corresponding to the smallest eigenvalues of the graph Laplacian. In order to obtain basis functions which have good performance, an appropriate Laplacian operator O of the graph G should be selected. In RPI, by using the PVFs to approximate the value function, the least-squared policy iteration algorithm (LSPI) [69] is used to find a near-optimal policy π.

3.6. Policy gradient methods

Policy gradient methods are a type of RL techniques that rely upon optimizing parameterized policies with respect to the expected return (long-term cumulative reward) by gradient descent. They do not suffer from many of the problems in traditional reinforcement learning approaches such as the lack of guarantees of a value function, the intractability problem resulting from uncertain state information and the complexity arising from continuous states and actions.

The literature on policy gradient methods has yielded a variety of estimation methods over the last years. The most prominent approaches are finite-difference and likelihood ratio methods, called REINFORCE in reinforcement learning. In the policy gradient approach, the policy parameters are updated approximately proportional to the gradient:

$$\Delta\theta \approx \alpha \frac{\partial \rho}{\partial \theta} \tag{60}$$

In contrast with the full observability mandated by the MDP model, a partially observable Markov decision process allows for optimal decision making in environments which are only partially observable to the agent [17]. A partially observable Markov decision process can be defined as a tuple $\{S,A,W,T,O,R\}$ in which S is a set of states, A is a set of actions, W is a set of observations, T is a transition function defined as T: $S \times A \times S \to [0,1]$, O is an observation function defined as O:$S \times A \times W \to [0,1]$ and R is a reward function defined as R:$S \times A \times S \to R$. Due to the partial observability, there is a need for memory when considering optimal decision-making in a POMDP. Usually, we can transform the POMDP to a belief-state MDP in which the agent summarizes all information about its past using a belief vector $b(s)$. However, computing value functions over a continuous belief space is computationally expensive. As policy gradient methods do not require the estimation of a belief state, they can be more easily applied in partially observable Markov decision process.

The GPOMDP algorithm was presented in [17] for generating a *biased* estimate of the gradient of the average reward in POMDPs. The main advantages of GPOMDP are that it requires storage of only twice the number of policy parameters, and only uses one free parameter. One recent advance in policy gradient RL is called the natural policy gradients (NPG). The NPG approach was originally proposed in [65], which was inspired by Amari's natural gradient algorithms in supervised learning contexts [4]. In [65], a Riemannian metric is defined to measure the effects of changes on an action probability distribution $\pi(x;a;\theta)$ by a small incremental vector $\Delta\theta$ in the current policy $\pi(x;a;\theta)$ as

$$D_{p(s,a)}[\theta \| \theta + \Delta\theta] \equiv \Delta\theta^T F(\theta)\Delta\theta \tag{61}$$

where

$$F(\theta) \equiv \sum_{x\in S} d^{\pi}(x) \sum_{a\in A} \pi(x,a;\theta)[\nabla_\theta \ln \pi(a,x;\theta) \nabla_\theta \ln \pi(a,x;\theta)^T] \tag{62}$$

The update rule of policy parameters with Kakade's NPG [65] is

$$\theta := \theta + \alpha F(\theta)^{-1} \nabla_\theta \eta(\theta) \tag{63}$$

where α is a sufficiently small step-size parameter.

Recently, many studies have empirically demonstrated that natural policy gradients significantly outperformed ordinary policy gradients in terms of their convergence rates as in [10,55,65,92,99].

3.7. ACDs for learning control

The actor–critic algorithms, viewed as a hybrid of VFA and policy search, have been shown to be more effective than standard VFA or policy search in continuous online learning tasks [95]. In an actor–critic learning control architecture, there is an actor for policy learning and a critic for VFA or policy evaluation. One pioneering work on RL algorithms using the actor–critic architecture was published in [16]. Recently, there are increasing research interests on actor–critic methods for RL, where adaptive critic designs (ACDs) [13,19,71,90,129] were widely studied as an important class of learning control methods for nonlinear dynamical systems. ACDs can be categorized as the following major groups: heuristic dynamic programming (HDP), dual heuristic programming (DHP), globalized dual heuristic programming (GDHP), and their action dependent (AD) versions [95]. Among these ACD architectures, DHP is the most popular one which has been proven to be more efficient than HDP [129].

As discussed in [148], in all ACDs, there are a critic and an actor in the learning control structure. However, there are some variations in the critic for different ACD methods. Different ACD structures can be distinguished from three aspects in the critic training process, i.e., the inputs of the critic, the outputs of the critic, and the requirements for a plant model in the training process of the critic. For the first aspect, the plant states are usually used as the inputs of the critic network; while in action dependent structures, the critic also receives the action outputs from the actor. For the second aspect, the critic outputs can be either the approximated value function or its derivatives. For example, in the HDP structure, the critic approximates the value function $V(x_t)$. In the DHP structure, it approximates $\lambda(t)$, which is defined as the gradient of $V(x_t)$; and in GDHP, it approximates both, $V(x_t)$ and its gradient. For the third aspect, the plant model can be either used or unused during the learning process of the critic. Usually, to approximate the value function $V(x_t)$ alone, the plant model may not be used, e.g., in HDP. It is necessary to use some information in the plant model to approximate the derivative of $V(x_t)$ such as in DHP.

In the critic, TD (λ) learning algorithms with neural networks [95] are popularly employed to approximate the value functions or their derivatives. Based on the outputs of the critic, policy gradient methods can be used to train the actor:

$$\Delta\theta_t = \frac{\partial V(x_t)}{\partial a_t} \frac{\partial a_t}{\partial \theta_t} \tag{64}$$

where a_t is the action output of the actor, and θ_t is the weight vector of the actor.

In [148], a novel framework of ACDs with sparse kernel machines is presented by integrating kernel methods to the critic learning of ACD algorithms. The kernel ACDs' framework is shown in Fig. 4. The sparsification method based on the approximate linear dependence (ALD) analysis is used to sparsify the kernel machines when approximating the action value functions or their derivatives. Using the regularized kernel machines, two kernel ACD algorithms, i.e., Kernel HDP (KHDP) and Kernel DHP (KDHP), are proposed to realize efficient online learning control.

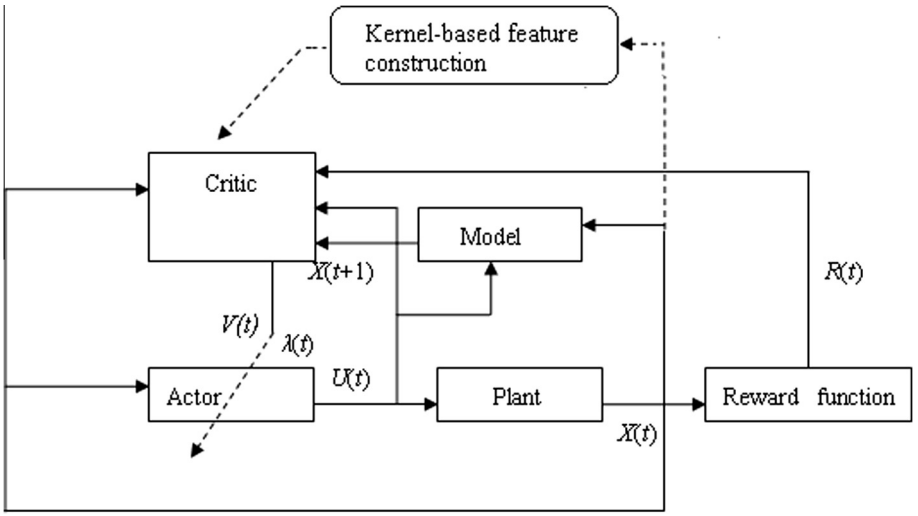


Fig. 4. Learning control structure of kernel ACDs [148].

To realize online learning in the critic, the following update rules based on the kernel RLS-TD (0) algorithm is used in the critic of KHDP.

$$\begin{aligned}
\beta_{t+1} &= P_t\vec{k}(s_t)/(\mu + (\vec{k}^T(s_t) - \gamma\vec{k}^T(s_{t+1}))P_t\vec{k}(s_t)) \\
\vec{\alpha}_{t+1} &= \vec{\alpha}_t + \beta_{t+1}(r_t - (\vec{k}^T(s_t) - \gamma\vec{k}^T(s_{t+1}))\vec{\alpha}_t) \\
P_{t+1} &= \frac{1}{\mu}\left[P_t - \frac{P_t\vec{z}_t(\vec{k}^T(s_t) - \gamma\vec{k}^T(s_{t+1}))P_t}{\left[\mu + (\vec{k}^T(s_t) - \gamma\vec{k}^T(s_{t+1}))P_t\vec{k}(s_t)\right]}\right] \\
\widetilde{Q}(s) &= \sum_{i=1}^{t}\alpha_i k(s, s_i)
\end{aligned} \tag{29}$$

where s, s_i are the combined features of state-action pairs (x, a), $\vec{k}(s_t) = (k(s_1, s_t), k(s_2, s_t), \ldots, k(s_T, s_t))^T$ is the sparsified kernel feature vector, β_t is the step size in the critic, $\mu(0 < \mu \leqslant 1)$ is the forgetting factor, $P_0 = \delta I$, δ is a positive number, and I is the identity matrix.

Recently, there are also increasing research interests on ACD or ADP for feedback control systems. Bradtke et al. [25] presented a Q-learning policy iteration method to solve the linear quadratic optimal control problem online without knowledge of the system dynamics. Al-Tamimi et al. [2] applied Q-learning to find a control that meets a given L_2 gain without knowing the system dynamics matrices. Note that all these results including Al-Tamimi et al. [2] seek to find an optimal control that minimizes a quadratic cost functional of the I/O data set. Also, they use pseudo-inverse or least-squares techniques to estimate solutions to a set of vector–matrix equations. In [3], adaptive critic designs corresponding to HDP and DHP were proposed to solve online the discrete-time zero-sum game with continuous state and action spaces appearing in H-∞ optimal control.

3.8. Model-based planning and learning

Planning is an off-line method, and it improves the policy without direct interactions with the environment. The DYNA structure [111] combines knowledge from planning on imaginary experience with that from learning over the real instances. The key idea is to maintain a world model from real interactions with the environment, and apply the world model to generate (i.e., simulate) virtual experience.

Planning and learning in DYNA interleave and reinforce each other: at each time step, planning starts from the learned parameters; and the improved parameters after planning are passed back for learning. Planning helps learning in that it provides better parameters for learning and decision making. In turn, we get improved experience that helps refine the world model and thereby also improve planning [111]. In DYNA, we can use various learning algorithms for learning and planning such as linear TD methods [119] and least-squares TD [23].

The overall architecture of Dyna agents is shown in Fig. 5. The left part is the basic interaction between the agent and the environment, which is used to generate real experience. The arrow on the left of the figure is the direct RL update using real experience to improve the value function and the policy [111]. The right part includes model-based simulation and learning. The model is learned from real experience and is used to generate simulated experience.

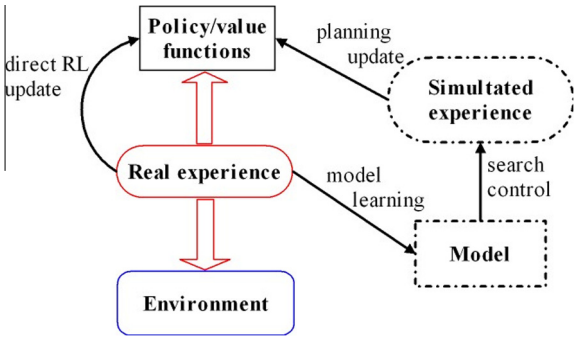


Fig. 5. The DYNA architecture for planning and learning [111].

Typically, the same reinforcement learning method can be used both for learning from real experience and for planning from simulated experience, as in Dyna-Q [111]. As indicated in [111], the reinforcement learning method is thus the common solution method for both learning and planning.

3.9. Approximate RL for averaged reward problems

In the average reward case, algorithms such as R-Learning [103], Relaxed-SMART [52] have been proposed. All these algorithms use some form of value iteration without function approximation. Relaxed-SMART has been proven to converge in [52]. In [56], based on the average reward optimality criterion, a hierarchical reinforcement learning (HRL) framework was proposed, where two formulations of HRL was investigated based on the average reward Semi-MDP model, both for discrete-time and continuous-time. However, most of the above works did not study the function approximation problem in averaged reward RL. In [19], actor–critic reinforcement learning algorithms utilizing linear function approximation were proposed for average reward MDPs and convergence results were established based on two-timescale stochastic approximations.

4. Feature representation and convergence of RL algorithms with function approximation

The convergence of RL/ADP algorithms with different feature representations has been studied in the past decades. Existing theoretical results mainly include three aspects. One aspect is the convergence of linear TD algorithms for learning prediction problems. The second aspect is the convergence and performance error bounds of approximate policy iteration algorithms with linear or kernel feature representations. The third aspect is convergence analysis of ACDs using two-timescale stochastic approximations. We will introduce the main results in the following discussions.

4.1. Convergence of TD learning with different representations

The convergence of TD (λ) learning algorithms is determined by the representations of states and the form of function approximators. Sutton [114] and Dayan [34] proved tabular TD algorithms converge in the mean, and Dayan and Sejnowski [35] proved TD (λ) converges with probability 1 when it was applied to absorbing Markov chains.

According to Watkins and Dayan [136], Jaakkola, Jordan, and Singh [61], and Tsitsiklis [123], the TD (0) learning rule can be viewed as a special case of Q-learning, so the convergence proofs for Q-learning can be extended to the case of the TD (0) learning rule with tabular representation [27]. Bradtke [26] extended Tsitsiklis' proof to show that on-line use of TD (0) with a linear function approximator converges to V with probability 1.

In order to use function approximation with RL in a safe way, some theoretical and empirical results were obtained in [22,122].

In [124], TD learning algorithms with linear function approximation was proved to converge with probability 1 under certain assumptions. In [27], Bradtke and Barto considered the asymptotic performance of LSTD when used on-line to approximate the value functions of absorbing and ergodic Markov chains. Under standard assumptions on the samples, it was proved that LSTD (λ) (and its recursive variants) converges almost surely to the solution of the projected fix-point equation if this solution exists [120]. The above property was proved $\lambda = 0$ by Bradtke and Barto [27], and for $\lambda > 0$ by Xu et al. [130] and Nedic and Bertsekas [83].

Sutton et al. proved that GTD2 and TDC both converge with linear function approximation in a general setting that includes both on-policy and off-policy learning [115]. Both algorithms have time and memory complexity that is linear in the number of features used in the function approximation, and both are significantly faster. Moreover, the TDC algorithm appears to be comparable in speed to conventional linear TD in on-policy problems [115].

4.2. Convergence of approximate policy iteration

Since exact representation and methods are impractical for large state and action space [69]. Approximate Policy Iteration (API) methods are popularly studied to deal with such cases. A convergence result of API can be given in terms of the infinity norm, as follows. If the policy evaluation error $\|\widehat{Q}^{\hat{\pi}_m} - Q^{\hat{\pi}_m}\|_\infty$ is upper-bounded by ε at every iteration $k \geqslant 0$, and if policy improvements are exact, then policy iteration eventually produces policies with a performance error bound as follows [18,28]:

$$\limsup_{m\to\infty} \|\widehat{Q}^{\hat{\pi}_m} - Q^*\|_\infty \leqslant \frac{2\gamma}{(1-\gamma)^2}\cdot\varepsilon \tag{65}$$

Where, Q^* is the optimal Q-function which corresponds to the optimal performance. If approximate policy improvements are performed, a similar bound holds. The result is given by the following theorem.

Theorem 1. [[28,69]]*Let $\hat{\pi}_0, \hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_m$ be the sequence of policies generated by an approximate policy-iteration algorithm and let $\widehat{Q}^{\hat{\pi}_0}, \widehat{Q}^{\hat{\pi}_1}, \widehat{Q}^{\hat{\pi}_2}, \ldots, \widehat{Q}^{\hat{\pi}_m}$ be the corresponding approximate value functions. Let ε and δ be positive scalars that bound the error in all approximations (over all iterations) to value functions and policies respectively. If*

$$\forall m = 0, 1, 2 \ldots, \|\widehat{Q}^{\hat{\pi}_m} - Q^{\hat{\pi}_m}\|_\infty \leqslant \varepsilon,$$

and

$$\forall m = 0, 1, 2 \ldots, \|T_{\hat{\pi}_{m+1}}\widehat{Q}^{\hat{\pi}_m} - T_* Q^{\hat{\pi}_m}\|_\infty \leqslant \delta.$$

where T_* is the Bellman optimality operator defined as

$$(T_*Q)(s,a) = R(s,a) + \gamma\sum_{s'\in S} P(s,a,s')\max_{a'\in A} Q(s',a').$$

Then, this sequence eventually produces policies whose performance is at most a constant multiple of ε and δ away from the optimal performance:

$$\limsup_{m\to\infty} \|\widehat{Q}^{\hat{\pi}_m} - Q^*\|_\infty \leqslant \frac{\delta + 2\gamma\varepsilon}{(1-\gamma)^2}. \tag{66}$$

In [144], it was analyzed that the convergence of KLSPI is determined by three factors. One is the convergence of the ALD-based kernel sparsification process. Second is the approximate error of KLSTD-Q and the third one is the convergence of approximate policy iteration based on approximate policy evaluation and greedy policy improvement. The convergence theorem of KLSPI can be stated as follows:

Theorem 2. [144] *If the initial data samples $\{(x_i, a_i, r_i, x_{i+1}, a_{i+1})\}$ are generated by an MDP using a stationary initial policy, the policies produced by the KLSPI algorithm will at least converge to an area of policy space having suboptimal performance bounds determined by the approximation error of KLSTD-Q. Furthermore, if the approximation error becomes zero, KLSPI will converge to the optimal policy of the MDP.*

These results show that approximate policy iteration is a fundamentally sound algorithm. As discussed in [144], the relationship between LSPI and KLSPI can be summarized in two aspects. On one hand, the KLSPI algorithm can be viewed as a substantial extension of LSPI by introducing kernel-based features that are constructed by an ALD-based kernel sparsification procedure. This extension from manually selected features of LSPI to automatically constructed features eliminates one of the main obstacles to successful applications of LSPI. On the other hand, by using KLSTD-Q, the policy evaluation in KLSPI can efficiently approximate the state-action value functions with high precision; therefore, the convergence as well as the policy optimality of KLPSI is better guaranteed than LSPI.

4.3. Convergence of ACDs [19,148]

According to the recent theoretical results in [68,19], the convergence of ACDs can be ensured based on two-timescale stochastic approximations, where the critic needs to implement a faster recursion than the actor. In [148], a novel framework of ACDs with sparse kernel machines is presented by integrating kernel methods into the critic of ACDs. To regularize kernel machines for approximating the value functions or their derivatives, a sparsification method based on the approximately linear dependence (ALD) analysis is used. Using the sparse kernel machines, two kernel-based ACD algorithms, i.e., Kernel HDP (KHDP) and Kernel DHP (KDHP), are proposed and their performance is analyzed both theoretically and empirically. Because of the representation learning and nonlinear approximation ability of sparse kernel machines, KHDP and KDHP can obtain much better performance than previous HDP and DHP methods with manually designed neural networks. Simulation and experimental results on two nonlinear control problems, i.e., a continuous-action inverted pendulum problem and a ball and plate control problem, demonstrate the effectiveness of the proposed Kernel ACD methods.

In Kernel ACDs, by making use of the kernel-based features, which are in a form of linear basis functions, the RLS-TD algorithm [142] is used to approximate the value functions or their derivatives with improved data efficiency and stability. As shown in [143], the kernel-based LS-TD algorithm is superior to conventional linear or nonlinear TD algorithms in terms of fast convergence rates. Therefore, with faster learning in the critic, Kernel ACDs can have better performance than previous ACDs in terms of convergence rates.

Similar to the analysis in [19], the update rules in ACDs can be modeled as a general setting of two-timescale stochastic approximations:

$$X_{t+1} = X_t + \beta_t\left(f(X_t, Y_t) + N^1_{t+1}\right), \tag{67}$$

$$Y_{t+1} = Y_t + \gamma_t\left(g(X_t, Y_t) + N^2_{t+1}\right), \tag{68}$$

where f, g are *Lipschitz* continuous functions and $\left\{N^1_{t+1}\right\}, \left\{N^2_{t+1}\right\}$ are martingale difference sequences with respect to the field

$$E\left[\left\|N^i_{t+1}\right\|^2 | F_t\right] \leqslant D_1\left(1 + \|X_t\|^2 + \|Y_t\|^2\right), \quad i = 1, 2, t \geqslant 0 \tag{69}$$

for some constant $D_1 < \infty$.

In KHDP and KDHP, the learning rules in the critic use recursive least-squares methods and the step-sizes are adaptively determined by online computation rules. When the update in the critic is a faster recursion than the update in the actor, the weights in the critic have uniformly higher increments as compared to the weights in the actor.

To analyze the convergence of Kernel ACDs based on two-timescale stochastic approximations, the following ODEs (Ordinary Differential Equations) can be considered:

$$\dot{X} = f(X(t), Y) \tag{70}$$

where Assumptions (A1)–(A3) hold:

(A1) $\sup_t\|X_t\|$, $\sup_t\|Y_t\| < \infty$.
(A2) $\dot{X} = f(X(t), Y)$ has a globally asymptotically stable equilibrium $\mu(Y)$, where $\mu(\cdot)$ is a *Lipschitz* continuous function.
(A3) $\dot{Y} = g(\mu(Y(t)), Y(t))$ has a globally asymptotically stable equilibrium Y^*.

In [19], Bhatnagar et al. presented the main convergence result for two-timescale stochastic approximations:

Theorem 3. *Under Assumptions (A1)–(A3), the updates in (67) and (68) converge asymptotically to the equilibrium, i.e., $(X_t, Y_t) \to (\mu(Y^*), Y^*)$ as $t \to \infty$, with probability one.*

In KHDP and KDHP, by appropriately selecting the actor's step sizes, it can be expected that the update in the critic is a faster recursion than the update in the actor and the weights in the critic have uniformly higher increments as compared to the weights in the actor. In [19], when the update in the critic is a faster recursion than the actor, it was proved that a class of actor–critic algorithms with linear function approximators will converge almost surely to a small neighborhood of a local minimum of the averaged reward J. In Kernel ACDs, by making use of kernel-based features and the RLS-TD algorithm in the critic, the updates in the critic can be a faster recursion than the actor. Thus, it will be more beneficial to ensure the convergence of the online learning process. In Section 4, extensive performance tests and comparisons were conducted and it was shown that Kernel ACDs have much better performance than conventional ACDs both in terms of convergence speed and in terms of the quality of the final policies.

5. Performance comparisons of RL algorithms with function approximation

In this section, some empirical results to compare different RL/ADP methods are shown and the performance of these algorithms is analyzed. The performance comparisons include three parts. The first part is to show the performance comparisons among typical learning prediction algorithms with linear function approximation such as GTD, TDC and GTD2. The second part is to compare different approximate policy iteration algorithms, which belong to batch learning algorithms, using some benchmark learning control problems. The third part is to compare the performance of online learning control algorithms with different feature representations. In the performance comparisons, typical RL algorithms with function approximation were tested so that the comparisonal results will be useful to guide practitioners in selecting appropriate RL algorithms for their applications.

5.1. Performance evaluation of different learning prediction algorithms [115,142,143]

In [115], the convergence rate of GTD2 and TDC was compared with that of GTD and conventional TD on four small problems—three random-walk problems and a Boyan-chain problem. All of these problems are episodic, undiscounted, and involved only on-policy training with a fixed policy. In the following, the empirical results obtained in [115] will be shown together with other comparison results of least-squares TD algorithms.

As introduced in [115], the random-walk problems are based on the standard Markov chain [111,114] with a linear arrangement of five states plus two absorbing terminal states at each end. Episodes begin in the center state of the five states, then transit randomly with equal probabilities to a neighboring state until a terminal state is reached. The rewards are zero everywhere except on transition into the right terminal state, upon which the reward is +1. Three versions of this problem were considered, with different feature representations. For detailed settings of the parameters and features, please refer to [115].

In [115], GTD, GTD2, TDC and TD were applied to these problems with a range of constant values for their step-size parameters. The parameter was varied over a wide range of values, in powers of 2. For the GTD, GTD2 and TDC algorithms, the ratio $\eta = \beta/\alpha$ takes values from the set $\{1/4, 1/2, 1, 2\}$ for the random-walk problems; one lower power of two was added for the Boyan-chain problem. The initial parameter vectors, θ_0 and w_0, were set to 0 for all the algorithms. Each algorithm and parameter setting was run for 100–500 episodes for different problems, where the square root of the mean-square projected Bellman error (MSPBE) were computed after each episode, then averaged over 100 independent runs [115]. The results are show in Fig. 6.

As seen in Fig. 6, all the algorithms are similar in terms of their dependence and sensitivity to the step sizes. Overall, GTD learned the slowest, followed after a significant margin by GTD2, followed by TDC and TD. TDC and TD performed similarly, with the extra flexibility provided by β sometimes allowing TDC to perform slightly better [115].

In [142], the performance of TD (λ) and RLS-TD (λ) was compared in a finite-state absorbing Markov chain called the Hop-World problem [23]. As shown in Fig. 7, the Hop-World problem is a 13-state Markov chain with an absorbing state.

In Fig. 7, state 12 is the initial state for each trajectory and state 0 is the absorbing state. Each non-absorbing state has two possible state transitions with transition probability 0.5. Each state transition has reward -3 except the transition from state 1 to state 0 which has a reward of -2. Thus, the true value function for state $i (0 \leqslant i \leqslant 12)$ is $-2i$.

To apply linear temporal-difference algorithms to the value function prediction problem, a set of four-element state features or basis functions is chosen, as shown in Fig. 7. The state features of states 12,8,4 and 0 are, respectively, [1], [0,1,0,0], [0,0,1,0], [0,0,0,1] and the state features of other states are obtained by linearly interpolating between these. From Fig. 8, it can be concluded that by making use of RLS methods, RLS-TD (λ) can obtain much better performance than conventional linear TD (λ) algorithms and eliminates the design problem of the step-size schedules.

5.2. Performance comparisons among API algorithms

The first experiment comparing the performance of LSPI and KLSPI is a 20-state chain problem which is a problematic MDP noted in [69]. The MDP consists of a chain with 20 states (numbered from 1 to 20) and a simplified example with 4 states is shown in Fig. 9. For each state, there are two actions available, i.e., "left" (L) and "right" (R). Each action succeeds with probability 0.9, changing the state in the intended direction, and fails with probability 0.1, changing the state in the opposite direction. The two boundaries of the chain are dead-ends. For the 4-state problem in Fig. 9, the reward vector over states is (0,+1,+1,0) and the discount factor is set to 0.9. It is clear that the optimal policy is RRLL.

In [69], LSPI was tested on the same problem. However, for the 20-state problem, careful selection of linear basis functions is required since the state space is larger than the 4-state problem. The 20-state problem has the same dynamics as the

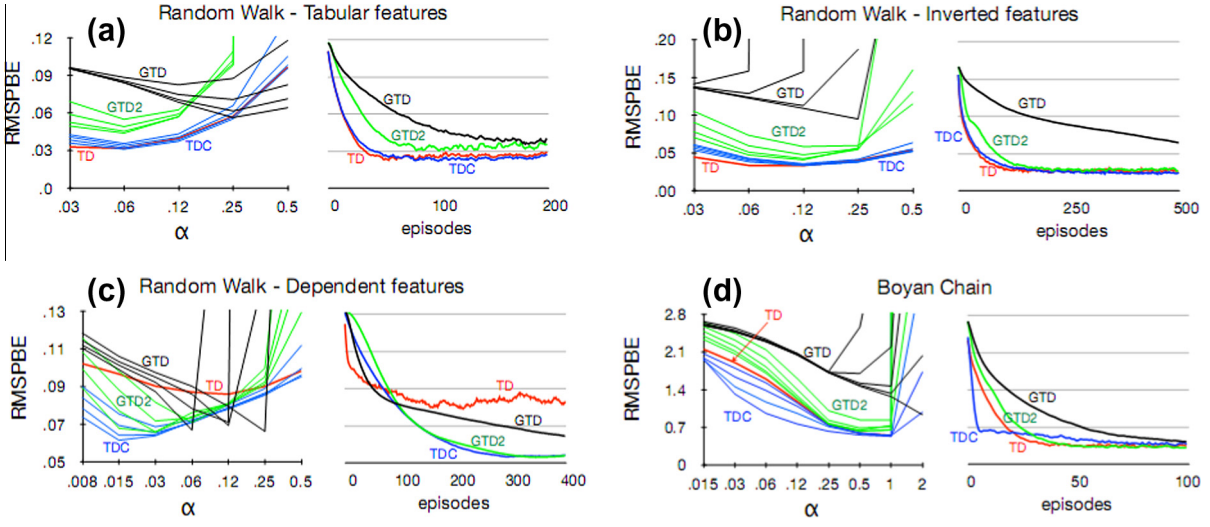


Fig. 6. Empirical results on the four small problems—three versions of the 5-state random walk plus the 14-state Boyan chain. In each of the four panels, the right subpanel shows a learning curve at best parameter values, and the left subpanel shows a parameter study plotting the average height of the learning curve for each algorithm, for various $\eta = \beta/\alpha$, as a function of α [115].

4-state problem except that a reward of +1 given only at the boundaries (states 1 and 20). The optimal policy in this case is to go left in states 1–10 and right in states 11–20.

In the following, the experimental setup is the same as the experiments for LSPI in [69]. For the LSPI algorithm, a polynomial of degree 4 was used to approximate the value function for each of the two actions, giving a block of 5 basis functions per action. The two algorithms use a single set of samples collected from a single episode in which actions were chosen uniformly at random for 5000 steps. The performance is evaluated by the iterations for convergence as well as the ultimate policies after convergence. Figs. 10 and 11 show the improved policy after each iteration of KLSPI and LSPI, respectively. From Figs. 10 and 11, it is illustrated that although both algorithms converge to the optimal policy, KLSPI converges to the optimal policy only after 3 iterations while LSPI converges to the optimal policy after 7 iterations. In fact, KLSPI finds the optimal policy only after 2 iterations and then it stabilize in the optimal policy. Thus, compared with LSPI, KLSPI is much more efficient in convergence rates and little work is required on feature selection. In KLSPI, we use a RBF (Radius Basis Function) kernel function and the unique parameter to be selected is the width σ of RBF. In the experiments, the RBF width is selected as $\sigma = 0.4$, which was simply tuned using a one-dimensional search process. The other parameter for KLSPI is the precision threshold μ of ALD-based kernel sparsification procedure. In all the following experiments, μ is equal to 0.001.

The convergence of KLSPI is mainly due to the powerful nonlinear approximation and generalization ability of the kernel-based policy evaluation using KLSTD-Q. This can be illustrated in the following Figs. 12–15, where the approximated value functions in each iteration as well as the exact values are plotted for both KLSPI and LSPI. In Fig. 12, it is shown that by using KLSTD-Q, KLSPI can approximate the exact state-action value functions with high precision so that it converges to the optimal policy in fewer iterations. In Fig. 13, it can be seen that there are relatively larger approximation errors in LSPI using

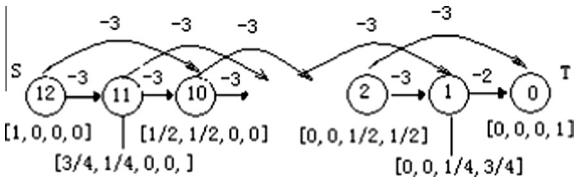


Fig. 7. The hop-world problem.

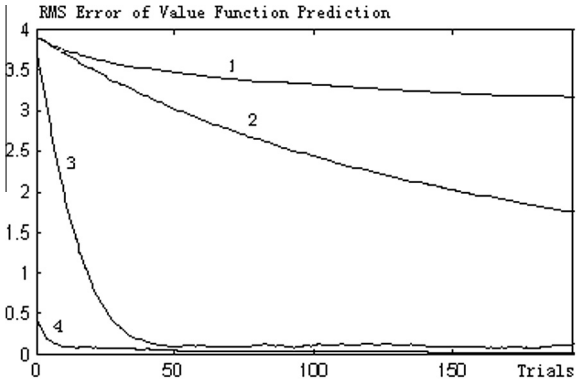


Fig. 8. Performance comparison between RLS-TD (λ) and TD (λ) [142]. 1, 2, 3—TD (0.3) with different step-size parameters; 4—RLS-TD (0.3) with initial variance matrix $P_0 = 500I$.

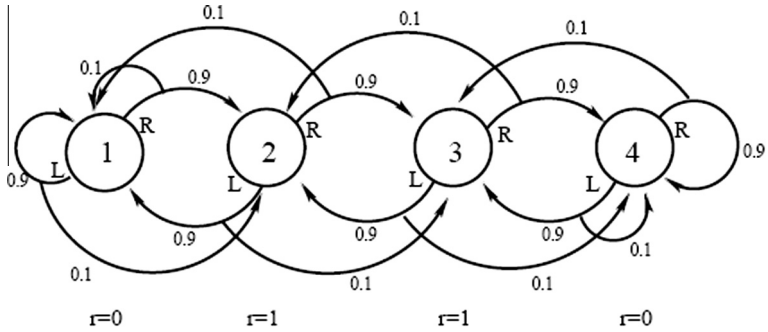


Fig. 9. The 4-state problematic MDP [69].

LSTD-Q with manually selected linear basis functions, especially in the first 4 iterations. As the errors become smaller, LSPI also converges to the optimal policy but more iterations are needed.

In the experiments, the state-action pairs of the 20-state MDP have a total number of 40 since there are two actions available for each state, and every state-action pair was originally represented as a two-dimensional vector $[x_t, a_t]$, where the elements were normalized by a positive constant, i.e., $x_t \in \{1/20, 2/20, \ldots, 20/20\}$ and $a_t \in \{0.1, 0.2\}$. After the kernel sparsification process, a kernel-based feature vector, which has the dimension of 21, was automatically obtained from the 5000 training samples. Although the feature dimension of KLSPI ($d = 17$) is larger than that of LSPI ($d = 10$), they are both compact representations of the original state-action space ($d = 40$) and the increase of computational costs in KLSPI is not significant when compared with the benefits of better approximation accuracy and better convergence behavior. Moreover, the features in KLSPI are automatically produced and optimized by the kernel sparsification procedure. Figs. 14 and 15 make comparisons of the state value functions $V(x)$ approximated by KLSPI and LSPI. It is also clearly shown that KLSPI can approximate the true state value functions with smaller errors and converge within fewer iterations than LSPI.

The performance of three API algorithms, LSPI, KLSPI and RPI was compared in the mountain-car problem [111] in the following experiment. The objective of the mountain-car task is to drive an underpowered car up a steep mountain road as quickly as possible, as suggested by the diagram in Fig. 16. The difficulty is that gravity is stronger than the car's engine, and even at full throttle the car cannot accelerate up the steep slope. The only solution is that the car must oscillate on the slope to build up enough momentum.

The reward is -1 on all time steps until the car moves past its goal position at the top of the mountain, which ends the episode. The state space includes the position and velocity of the car. There are three candidate actions: full throttle forward (+1), full throttle reverse (-1), and zero throttle (0). The car moves according to a simplified physics. Its position, x_t, and velocity, $\dot{x}_t$, are updated by

$$\begin{cases} \dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t + -0.0025\cos(3x_t)] \\ x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}] \end{cases} \tag{71}$$

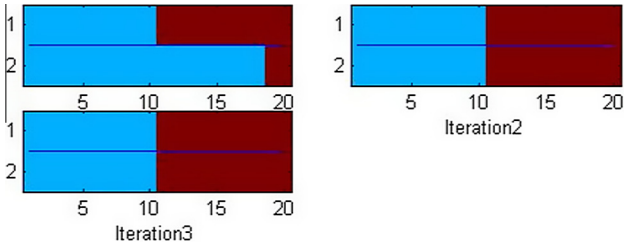


Fig. 10. The improved policy of KLSPI after each iteration (R action – dark/red shade; L action – light/blue shade; KLSPI – top stripe; exact – bottom stripe). The subplots are: Top left–Iteration 1, Down Left–Iteration 3, Top right—Iteration 2 [69]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

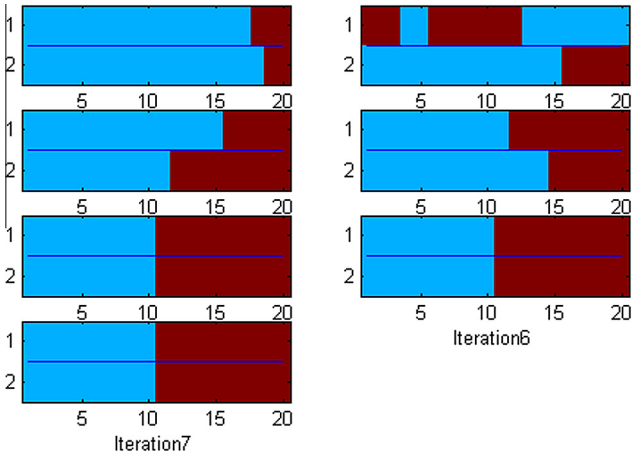


Fig. 11. The improved policy of LSPI after each iteration (R action – dark/red shade; L action – light/blue shade; LSPI – top stripe; exact – bottom stripe). The subplots are: Top left–Iteration 1, Down Left–Iteration 7, Others—Iterations 2–6 [69]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

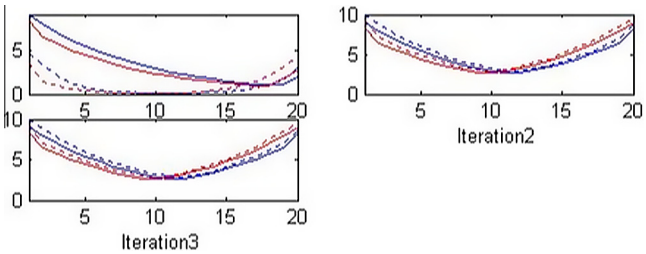


Fig. 12. The state-action value function $Q(x, \pi(x))$ of the policy being evaluated in each iteration (KLSPI approximation – solid line; exact values – dotted line). The subplots are: Top left–Iteration 1, Down Left–Iteration 3, Top right—Iteration 2.

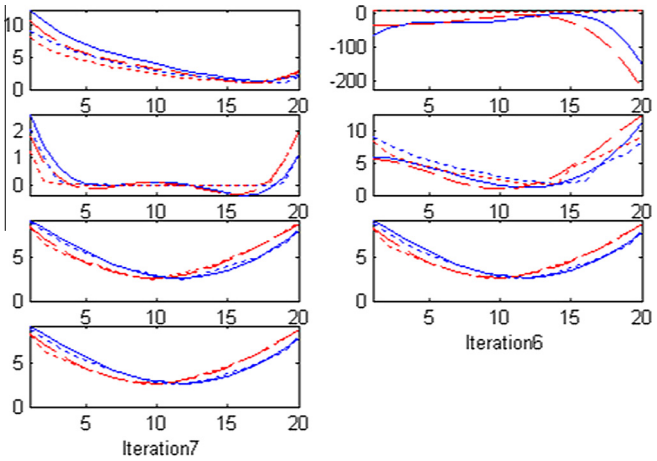


Fig. 13. The state-action value function $Q(x, \pi(x))$ of the policy being evaluated in each iteration (LSPI approximation – solid line; exact values – dotted line). The subplots are: Top left–Iteration 1, Down Left–Iteration 7, Others—Iteration 2–6.

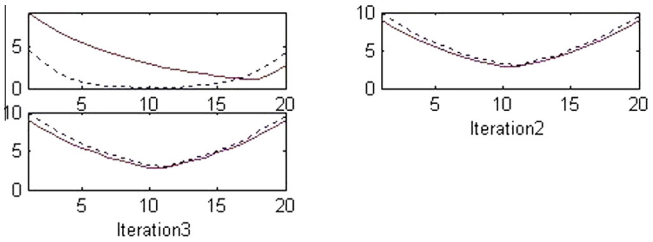


Fig. 14. The state value function $V(x)$ of the policy being evaluated in each iteration (KLSPI approximation – solid line; exact values – dotted line). The subplots are: Top left–Iteration 1, Down Left–Iteration 3, Top right—Iteration 2 [69].

where the bound operation enforces $-1.2 \leqslant x_{t+1} \leqslant 0.5$ and $-0.07 \leqslant \dot{x}_{t+1} \leqslant 0.07$. When x_{t+1} reaches the left bound, $\dot{x}_{t+1}$ is reset to 0. When it reaches the right bound, the goal is reached and the episode is terminated. In our experiments we allow a maximum of 300 steps, after which the task is terminated without success. For performance in the mountain-car domain is measured by the number of steps, lower numbers indicate better performance. Table 1 summarizes the range of parameters over which the algorithms are tested in the mountain-car domain. The results for the following experiments are averaged over 10 runs.

In the implementation of LSPI, RBF basis functions are used and the width is set as 0.5. In the implementation of KLSPI, RBF kernel functions are used and the width parameter for RBF kernel is selected as 0.5. The threshold parameter for ALD-based sparsification is set as 0.1. The simulation experiments were conducted for several independent runs with random initial policies. For every run of KLSPI, the learning control process of KLSPI consists of eight iterations. The results are shown in Fig. 17.

As shown in Fig. 17, LSPI learned the slowest, followed after a significant margin by KLSPI, followed by RPI. KLSPI and RPI performed similarly, despite the different feature representations. The results also indicate that the feature representation

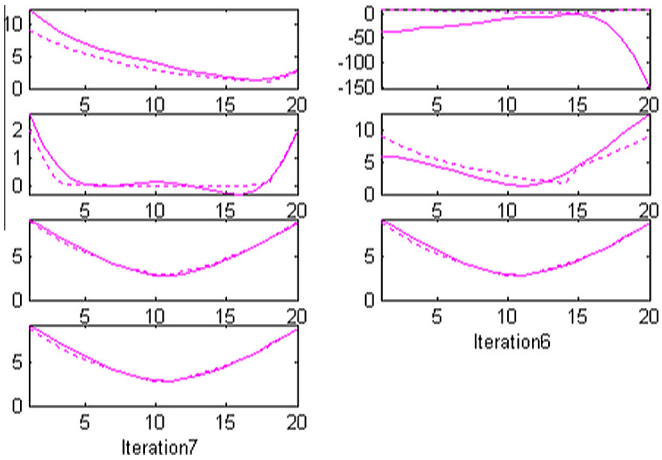


Fig. 15. The state value function $V(x)$ of the policy being evaluated in each iteration (LSPI approximation – solid line; exact values – dotted line). The subplots are: Top left–Iteration 1, Down Left–Iteration 7, Others—Iterations 2–6 [69].

can influence the convergence of the API algorithms. Suitable representation can make the method convergent and converge fast.

5.3. Performance comparisons of ACDs [148]

In the following, simulation studies will be conducted on the inverted pendulum problem to compare the performance of different ACDs algorithms. In simulation, the performance of Kernel ACDs is compared with ACDs under different conditions and parameter settings [148]. The inverted pendulum problem requires balancing a pendulum of unknown mass and length by applying force to the cart to which the pendulum is attached to, as shown in Fig. 18.

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 dynamics equations are assumed to be unknown or only partially known for the learning controller. For HDP and KHDP, the reward r is always 0 before the pole angle or the position of the cart exceeds the boundary conditions, i.e., if $|\theta| \leqslant 12^\circ$, $|x| \leqslant 1.2m$, $r(t) = 0$; else $r(t) = -1$. For DHP and KDHP, a differentiable reward function is defined as $r(t) = 0.5(x^2 + \theta^2)$. The simulation time-step is 0.02 s. A learning controller is regarded to be successful when its final policy can balance the pole for at least 10,000 time steps. A trail starts from an initial state near the equilibrium and ends when the controller balances the pole for 10,000 time steps or the pole angle or the position of the cart exceeds the boundary conditions.

The performance of Kernel ACDs and conventional ACDs was compared under different parameter settings including the variations of actor learning rates, the cart mass, and the pole length. We use two performance measures to evaluate the learning efficiency of different learning control methods. One is the successful rate of a learning controller, which is defined as the percentage of successful learning trials that can learn a policy to balance the pole for at least 10,000 time steps. The other is the averaged number of trials that are needed to learn a successful policy. The averaged number of trials was computed by running the learning control process for 10 independent runs. For each independent run, the maximum number of learning trials is 100. For KHDP and KDHP, 40 trials of samples were collected by a random policy to construct the dictionary of kernel features. The threshold parameter for the ALD analysis is set as $\mu = 0.001$. The results are shown in Fig. 19.

As shown in Fig. 19, the performance of KDHP and KHDP is much better than DHP and HDP, respectively. In Fig. 19(a), the successful rates of KDHP are all 100 percent under different settings of actor learning rates. While the performance of DHP and HDP is greatly influenced by the actor learning rates. It was observed that KHDP has higher successful rates than HDP

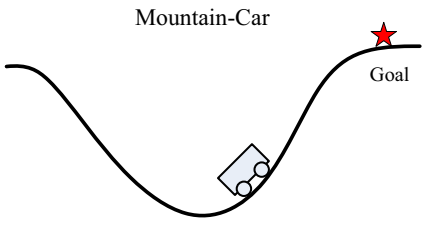


Fig. 16. The mountain-car.

Table 1
Parameter values for different algorithms.

Parameter	LSPI	KLSPI	RPI
Discount factor	0.99	0.99	0.99
Episodes	[0:30:300]	[0:30:300]	[0:30:300]
Max steps	300	300	300
Max iterations	10	10	10
Epsilon	10^{-5}	10^{-5}	10^{-5}
Delta	/	0.1	/
Width of kernel	/	0.5	/
The width of RBF	0.5	/	/
The width for Gauss distance	/	/	0.5

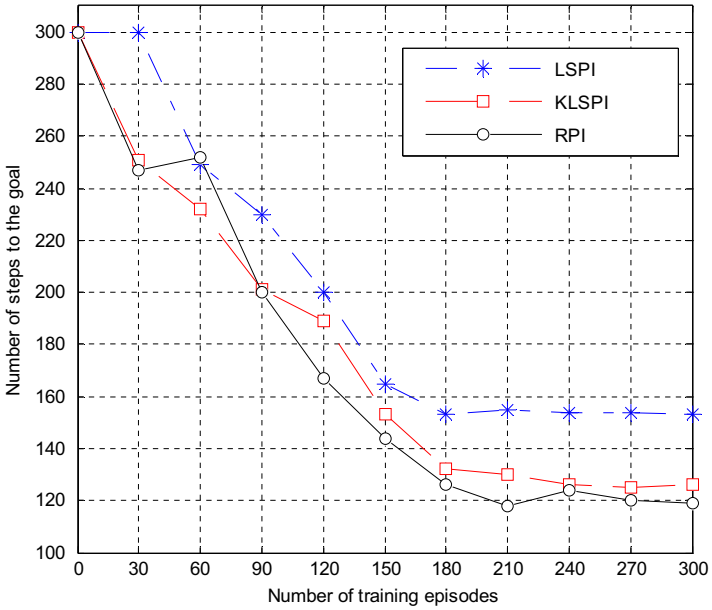


Fig. 17. Performance comparisons between LSPI, KLSPI and RPI in the mountain-car domain.

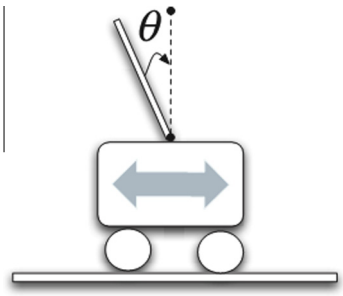


Fig. 18. The inverted pendulum.

and it is also less sensitive to the variations of actor learning rates. In Fig. 19(a), it is illustrated that KDHP has the best performance (100 percent successful rates) under different dynamics changes of the plant including the variations of the cart mass and the pole length. The performance of KHDP is also much more robust than HDP and DHP. In Fig. 19(b), it is shown that KDHP needs the minimum averaged number of trials for learning a successful control policy, which means that KDHP converges faster than other learning control algorithms. Compared with HDP, KHDP converges to a good control policy much

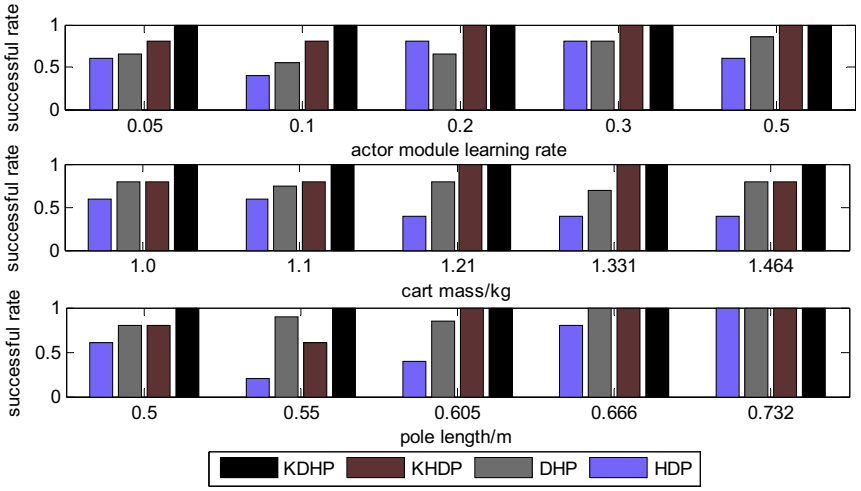
faster. However, compared with KHDP and HDP, DHP needs smaller number of trials to balance the pole successfully. This is mainly due to that DHP makes use of some model information to estimate the policy gradient, which will greatly reduce the variance of policy gradients and increase the convergence speed of ACDs.

6. Applications of RL and ADP with function approximation

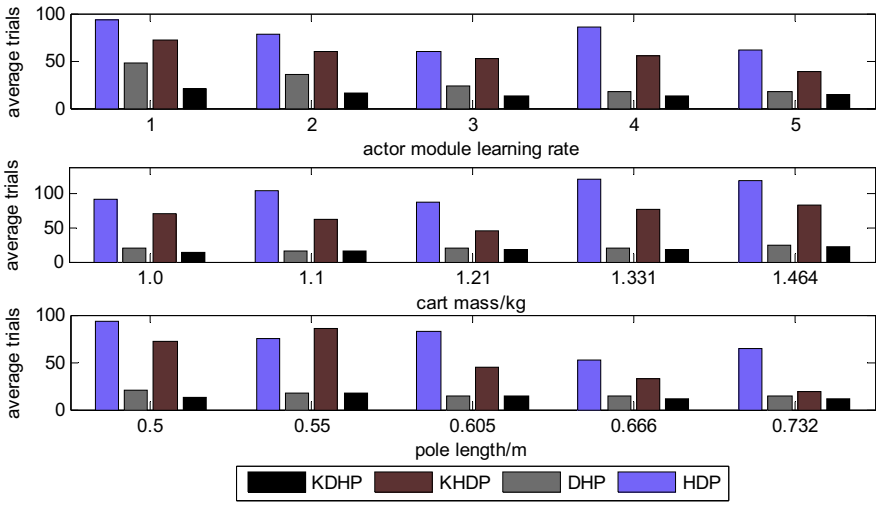
In recent years, RL has been applied in many fields, such as computer games, robotics, industrial systems, power systems and so on. A majority of the applications are listed in Table 2.

6.1. Computer games

RL has been applied in learning how to play games for many years. The first application of RL to a complex non-trivial task was learning the game of Backgammon [121]. Tesauro discussed a number of important practical issues and examined these issues in the case of applying the TD (λ) to learning the game of Backgammon from the outcome of self-play. It is found that, after training with zero knowledge built in, the agent can play the game at a fairly strong intermediate level of performance. Tetris is a popular video game played on a two-dimensional grid. Bertsekas and Tsitsiklis has applied the TD-based policy



(a) Performance comparisons in terms of successful rates



(b) Performance comparisons in terms of average trials

Fig. 19. Performance comparisons between K-ACDs and ACDs under different parameter settings [148].

iteration method to the Tetris and obtained quite good performance [18]. In [105], a reinforcement learning approach based on linear evaluation function and large numbers of binary features was applied to the game of Go. McPartland and Gallagher investigated the tabular Sarsa (λ) RL algorithm applied to a purpose built first person shooter (FPS) game [78]. The experimental results indicated that RL can be used in a generalized way to control a combination of tasks in FPS bots such as navigation, item collection, and combat.

6.2. Industrial and financial systems

A wide range of successful applications in industrial and financial systems can be addressed, such as packet routing [24], job scheduling [153], elevator dispatching [32] and stock trading [85] to name a few. Recently, more applications to industrial and financial systems have appeared. Auctions is a primary pricing mechanism in market components of a deregulated power industry. Nanduri and Das [82] presented a non-zero sum stochastic game theoretic model and a RL-based auction method that allow assessment of market power in DA markets. Shimokawa et al. applied RL to detect the most suitable learning model of the human decision-making process in financial investment tasks [104].

Table 2
Typical applications of RL/ADP algorithms. ("D" stands for "Discrete", "C" stands for "Continuous").

Application domain		State/Action spaces	RL/ADP methods	Ref.
Computer Games	Backgammon	D/D	TD (λ) with neural network	Tesauro [121]
	Tetris	D/D	Optimistic policy iteration	Bertsekas and Tsitsiklis [18]
	Go	D/D	TD (0) with linear features	Silver et al. [105]
	First person shooter	D/D	Sarsa (λ)	McPartland and Gallagher [78]
Industrial and Financial systems	Packet routing	D/D	Q-routing	Boyan and Littman [24]
	Job-shop scheduling	D/D	TD (λ) with neural network	Zhang and Dietterich [153]
	Elevator dispatching	C/D	Q-learning with neural network	Crites and Barto [32]
	Predicting investment behavior	C/C	TD-learning	Shimokawa et al. [104]
Vehicle control and Robotics applications	Autonomous helicopter control	C/C	Policy Search	Bagnell and Schneider [9]
	Autonomous helicopter control	C/D	PEGASUS	Ng [84]
	Fast quadrupedal locomotion	C/D	Parameterized policy	Kohl and Stone [67]
	Robot soccer keepaway	C/D	Sarsa (λ) with tile coding	Stone et al. [110]
	Car driving	C/D	Neural Fitted Q Iteration (NFQ)	Riedmiller et al. [98]
	Motor skills	C/C	Policy gradient	Peters and Schaal [91]
	Robot soccer	C/D	Neural fitted value/policy iteration	Riedmiller et al. [97]
	Robot navigation	D/D	Q-learning with state abstraction	Jaradat et al. [62]
Traffic Systems	Traffic signal control	D/D	Q-learning with neural networks	Abdulhai et al. [1]
	Urban traffic signal control	D/D	Q-learning	Balaji et al. [12]
	Traffic signal control	D/D	Q-learning with linear FA	Prashanth and Bhatnagar [94]
Communication networks and Cognitive Radio	Dynamic channel allocation	D/D	TD (0) with linear FA	Singh and Bertsekas [14]
	Aggregated interference control	D/D	Q-learning with neural networks	Serrano and Giupponi [49]
	Anomaly detection in computers	D/D	TD learning with linear FA	Xu [145]
	Repeated power control game	D/D	Linear Bush-Mosteller reinforcement scheme	Zhou et al. [154]
Power Systems	Power systems stability control	C/D	Model-based Q-learning	Ernst et al. [46]
	Constrained load flow problem	D/D	Q-learning with state abstraction	Vlachogiannis and Hatziargyriou [130]
	Static compensator controller	C/C	Action-dependent HDP	Mohagheghi et al. [81]
	Automatic generation control	D/D	Q (λ) with state abstraction	Yu et al. [151]
Computer Vision	Image retrieval	D/D	Tabular Q-learning	Yin et al. [150]
	Automatic target recognition	D/D	HDP and DHP with neural networks	Iftekharuddin [60]

6.3. Vehicle and robot control

With the advent of increasingly efficient RL methods, one can observe a growing number of successful RL applications in robotics, e.g. helicopter control [9,84], car driving [98], learning of robot behaviors [66,91,155], control of soccer robots [97,110], and mobile robot navigation [62]. Autonomous helicopter control is a challenging problem with high-dimensional, complex, asymmetric, noisy, nonlinear, dynamics. Bagnell and Schneider studied applying policy search methods to the flight control of an autonomous helicopter [9]. In [84], Ng et al. described a successful application of RL to autonomous helicopter flight. Autonomous underwater vehicles (AUVs) control is another successful application of RL. Carreras et al. presented a hybrid behavior-based scheme using RL for high-level control of autonomous AUVs [30]. There have been other applications of RL to vehicle and robot control besides helicopter and AUVs. Riedmiller et al. applied Neural Fitted Q Iteration (NFQ) to learn to drive a real car in 20 min [98]. Jaradat et al. developed a RL approach for solving the problem of mobile robot path planning in an unknown dynamic environment based on Q-learning [62].

6.4. Power systems

There have existed several applications of RL in power systems. One is to improve the stability of the power system. Power system stability is the property of a power system which enables it to remain in a state of equilibrium under normal operating conditions and to regain an acceptable state of equilibrium after a disturbance [46]. One way to increase power system stability margins is to control power systems more efficiently. For this purpose, Ernst introduced a methodology based on RL as a frame work that provided a systematic approach to design power system stability control agents [46]. Another application is to solve the constrained load flow problem by using RL methods. The Constrained Load Flow (CLF) problem deals with the adjustment of the power system control variables in order to satisfy physical and operating constraints. Vlachogiannis and Hatziargyriou formulated the CLF problem as a multistate decision problem and applied the Q-learning algorithm to the IEEE 14-bus system and to the IEEE 136-bus system for constrained reactive power control [130]. Besides, the adaptive critic design method was used for the design of the static compensator neuro-fuzzy controller in a multi-machine power system [81]. In [151], a novel stochastic optimal relaxed automatic generation control in non-Markov environment using Q (λ) learning is proposed and successfully implemented on a small two-area power system and the China Southern Grid power systems.

6.5. Traffic systems

Traffic control in urban areas is becoming increasing complex with the exponential growth in vehicle count. The traffic junctions play a very important role in determining the congestion state of the road network. Traffic signals were introduced to control the traffic flow, thereby improving the safety of road users. Many traffic junctions worldwide currently use fixed signal timings, i.e., they periodically cycle through the sign configurations in a round-robin manner. RL has been studied in the design of intelligent traffic signal timing for the junctions. Arel et al. introduced a multi-agent RL system to obtain an efficient traffic signal control policy, aiming at minimizing the average delay, congestion and likelihood of intersection cross-blocking [7]. In [12], a multi-agent reinforcement learning (RLA) signal control algorithm was proposed to reduce the total travel time and delay experienced by vehicles. Simulation tests conducted on a virtual traffic network of Central Business District in Singapore for four different traffic scenarios showed almost 15% improvement over the benchmark signal controls. Prashanth and Bhatnagar [94] proposed a RL algorithm with function approximation for traffic signal control. The algorithm incorporated state-action features and was easily implementable in high-dimensional settings. They designed and evaluated two Q-learning-based algorithms for road traffic control on a network of junctions.

6.6. Communication networks and cognitive radio

In cellular communication systems, an important techniques is to allocate the communication resource (bandwidth) so as to maximize the service provided to mobile callers whose demand for service varies stochastically. This problem can be easily formulated as a dynamic programming problem. Singh and Bertsekas used the TD(0) algorithm to find dynamic channel allocation policies which have better performance than previous heuristic solutions [106]. The policies were tested with good performance for a broad variety of call traffic patterns.

Cognitive radio (CR) systems can be seen as a new way to implement efficient reuse of the pooled radio spectrum assigned to multiple wireless communication systems, by exploiting a wide variety of intelligent behavior [80]. Yang and Grace presented a random picking distributed channel assignment scheme applied to a cognitive radio system, which exploited reinforcement learning with a user population receiving multicast downlink transmissions [148]. Serrano and Giupponi proposed a decentralized Q-learning algorithm to manage the aggregated interference generated by multiple CRs [49]. The concept of a Cognition Cycle (CC) is the key element of CR to provide context awareness and intelligence so that each unlicensed user is able to observe and carry out an optimal action on its operating environment for performance enhancement. In [149], RL was applied to implement the conceptual of the Cognition Cycle. Jiang, Grace and Mitchell introduced two approaches, pre-partitioning and weight-driven exploration, to enable an efficient learning process in the context of cognitive radio base on RL [63]. The simulation results showed that the exploration of cognitive radio was more efficient by using the proposed

approaches and the system performance was improved accordingly. A robust power control algorithm with low implementation complexity was designed for competitive and autonomous CR networks in [154].

6.7. Computer vision

Reinforcement learning has been successfully applied to several computer vision problems such as image segmentation, feature extraction, and object recognition [88,89]. In image retrieval system, relevance feedback (RF) is an interactive process which refines the retrievals to a particular query by utilizing user's feedback on previously retrieved results. Yin [150] proposed an image relevance reinforcement learning (IRRL) model for integrating existing RF techniques in a content-based image retrieval system. Recently, Iftekharuddin has applied the ACD methods to automatic target recognition (ATR) [60]. ATR has been an active research area due to its widespread applications in defense, robotics, medical imaging and geographic scene analysis. Iftekharuddin used adaptive critic design (ACD) neural network (NN) models to obtain an on-line ATR system for targets in presence of image transformations, such as rotation, translation, scale and occlusion as well as resolution changes. Implementation of two ACD-based learning designs, such as heuristic dynamic programming (HDP) and dual heuristic dynamic programming (DHP), demonstrate that RL is an effective approach for on-line transformation invariant ART.

6.8. Discussions

As seen above, RL and ADP methods have been applied in many different fields. However, different classes of RL/ADP algorithms may be suitable to different applications since different problems have their own characteristics. In general, we can classify existing RL/ADP algorithms into three main types. One is value-based algorithms which mainly include Q-learning and Sarsa-learning algorithms with different function approximators. The second class of RL/ADP methods makes use of an actor–critic architecture which approximates the value function and the policy simultaneously. The third class belongs to policy search methods which only approximate the policy. Table 3 lists the application domains in which different classes of RL and ADP methods have been popularly applied. As shown in Table 3, learning control methods using value-based RL/ADP algorithms were most widely used in different domains, including computer games, industrial and financial systems, vehicle control and robotics, to name a few. All of these applications usually have discrete and countable action spaces. The popularity of value-based algorithms may be due to their simplicity for online implementation. Besides, when linear approximators are used, value-based algorithms usually have fast convergence speed. However, the performance of value-based algorithms may not be satisfactory for solving problems with continuous state and action spaces. To deal with this difficulty in practical applications, approximate policy iteration and ACD methods have been taken into consideration. So far, API and ACD methods have been applied in computer games, robotics, power systems and computer vision. In particular, ACDs have been shown to be suitable for solving online learning control problems of continuous dynamical systems. Policy search methods have been applied in several domains with large action spaces. The advantage of policy search methods is their stable performance for online learning but the convergence speed of policy search is usually slow when compared with value-based RL/ADP methods and ACD methods.

7. Some open problems

7.1. Feature learning in RL and ADP

Feature representation is a fundamental problem in VFA for RL. Earlier research in RL usually relied on hand-coded neural network structure for VFA in large or continuous spaces. Thus, the performance of RL greatly depends on an empirical pro-

Table 3
Major application domains of different classes of RL/ADP algorithms.

RL/ADP methods	Domains	Algorithms
Value-based algorithms	Computer Games	Tabular Q-learning
	Industrial and Financial systems	Q-learning with neural network
	Vehicle control and Robotics applications	Q-learning with state abstraction
	Traffic Systems	Q-learning with linear FA
	Communication networks and Cognitive Radio	Neural Fitted Q-iteration (NFQ)
	Power Systems	Sarsa (λ)
	Computer Vision	
Policy Iteration/ACDs	Computer Games	Optimistic policy iteration using TD (λ) with neural network
	Robotics	Neural fitted value/policy iteration
	Power Systems	HDP and DHP with neural networks
	Computer Vision	
Policy Search	Industrial and Financial systems	Policy search with TD (λ) learning PEGASUS
	Robotics	Parameterized policy
	Communication networks and Cognitive Radio	Policy gradient

cess of feature representation and selection. Due to the successful applications of kernel machines in supervised learning, the combination of kernel methods with RL and ADP receives increasing interests in recent years. Several attempts have been made to apply GPs or SVMs in reinforcement learning problems, such as Gaussian processes in TD (0) learning [43], SVMs for RL [96] and Gaussian processes in model-based approximate policy iteration [36]. Recently, a framework for VFA called proto-reinforcement learning (PRL) has been proposed [75,76]. Instead of learning task-specific value functions using a hand-coded parametric architecture, agents can automatically learn basis functions by using spectral analysis of the self-adjoint Laplace operator. This approach also yields new learning control algorithms called representation policy iteration (RPI) where both the underlying representations (basis functions) and policies are simultaneously learned. Laplacian eigenfunctions also provide ways of automatically decomposing state spaces since they reflect bottlenecks and other global geometric invariants. However, both KLSPI and RPI are mainly restricted to solve MDPs with discrete actions. In addition, it is still difficult select subsamples from which the graph is derived from, especially in continuous MDPs. ACDs can be used solve MDPs with both continuous state and action spaces. Nevertheless, for ACDs, it was pointed out in [19] that a study on the choice of the basis functions for the critic to obtain a good estimate of the policy gradient needs to be done. Recent works on model selection and regularization in RL [47,48] addressed the problem of regularization of function approximators in order to improve the generalization ability of RL algorithms.

7.2. Rate of convergence and tuning step-sizes in RL and ADP

As indicated in [53], it is often the case in RL that the learning process goes through an initial transient period where the estimate of the value either increases or decreases steadily and then converges to a limit at an approximately geometric rate, which means that the series of observations is no longer stationary. In such situations, either of stepsize rules (constant or declining) would give an inferior rate of convergence. The rigorous theoretical results on the rate of convergence for RL/ADP algorithms with FAs are still open research questions to be addressed. Nonstationarity could arise if the initial estimate is of poor quality, in the sense that it might be far from the true value, but with more iterations, it moves closer to the correct value. Alternatively, the RL problem could be of a nonstationary nature and it might be hard to predict when the estimate has actually approached the true value. It is an open problem to design the optimal stepsize when the observed data is nonstationary. Darken and Moody [33] addresses the problem of having the stepsizes evolve at different rates, where the stepsize chosen is a deterministic function of the iteration counter. These methods are not able to adapt to differential rates of convergence among the parameters. George and Powell [53] addressed the problem of determining optimal stepsizes for estimating parameters in ADP and derived formulas for optimal stepsizes for minimizing estimation errors. However, much work still needs to be done to design optimal or near-optimal stepsizes for general RL and ADP algorithms. In a recent work [77], it was shown that the performance of existing step-size adaptation methods are strongly dependent on the choice of their meta-step-size parameter and that their meta-step-size parameter cannot be set reliably in a problem-independent way. They introduce a series of modifications and normalizations to the IDBD method [117] that together eliminate the need to tune the meta-step-size parameter to the particular problem.

7.3. FA for hierarchical RL [146]

Recent attempts to overcome the obstacles associated with dimensionality have turned to principled ways of problem decomposition or exploiting temporal abstraction, which are called hierarchical approaches to RL [15,146]. As indicated in [15], existing work in hierarchical RL (HRL) has followed three trends: focusing on subsets of the state space in a divide and conquer approach (state space decomposition) [37], grouping sequences or sets of actions together (temporal abstraction) [118], and ignoring differences between states based on the context (state abstraction) [6,38,58]. These three trends have led to three main approaches to hierarchical RL: options formalism [118], the hierarchies of abstract machines (HAMs) approach [87], and the MAXQ framework [37], in which the model of semi-Markov decision processes (SMDPs) is commonly used as a formal basis [59]. Although function approximators can be combined with HRL, few successful applications of existing HRL approaches to MDPs with large or continuous spaces have occurred. In addition, it remains difficult to decompose the state space of MDPs automatically or construct options so that global optimal policies can be well approximated [146].

7.4. Robust ADP for nonlinear systems

As indicated in [152], it is worth mentioning that many existing results based on ADP technique require a knowledge of known nonlinear dynamics. Recent works have attempted to solve the optimal control solution based on ADP technique without an *a priori* system model [126,127,131,132]. For the continuous-time case, an offline neural net policy iteration solution was given in [131] and an online actor–critic algorithm was proposed in [126] to solve the continuous-time infinite horizon optimal control problem. Nevertheless, the requirement of system dynamics is hard to be satisfied, either fully or even partially known. Hence, for the unknown general nonlinear systems, ADP methods mentioned above cannot be applied directly. Recent progresses in this direction also include ADP methods for partially unknown nonlinear systems [132], multiplayer non-zero-sum games [127] and robust approximate optimal tracking control scheme for unknown general nonlinear systems [152].

8. Conclusions and future work

As an interdisciplinary area, RL and ADP algorithms with function approximation have attracted many research interests from different domains such as machine learning, control theory, operations research, and robotics. In this paper, recent developments in RL algorithms with function approximation are comprehensively surveyed. Theoretical results on the convergence and feature representation of RL algorithms are discussed. The performance of different RL algorithms was evaluated and compared in several benchmark learning prediction and learning control tasks. The applications of RL with function approximation are also summarized. Although many progresses have been made in the past decade, there are still some research challenges and open problems. It is desirable to develop new RL/ADP algorithms with the abilities of learning representation, fast convergence in online learning process [74], generalization in high-dimensional continuous spaces [147]. The research of RL/ADP methods as data-driven or learning control techniques for complex dynamical systems is also an important topic for future work. Furthermore, RL/ADP methods for multi-agent systems and multi-objective decision and control tasks still need to be explored for real-world applications.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant 61075072, 91220301, and New-Century Excellent Talent Plan of the Ministry of Education of China (Grant No. NCET-10-0901). The authors would like to thank Prof. Richard S. Sutton for his comments and suggestions.

References

[1] B. Abdulhai, R. Pringle, et al, Reinforcement learning for true adaptive traffic signal control, Journal of Transportation Engineering 129 (3) (2003) 278–285.
[2] A. Al-Tamimi, F.L. Lewis, M. Abu-Khalaf, Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control, Automatica 43 (2007) 473–481.
[3] A. Al-Tamimi, M. Abu-Khalaf, F.L. Lewis, Adaptive critic designs for discrete-time zero-sum games with application to H-Infinity control, IEEE Transactions on Systems Man Cybernetics-Part B (2006).
[4] S. Amari, Natural gradient works efficiently in learning, Neural Computation 10 (2) (1998) 251–276.
[5] A. Antos, R. Munos, C. Szepesvari, Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems, in: 2009 American Control Conference, Hyatt Regency Riverfront, St. Louis, MO, USA, June 10–12, pp. 725–730.
[6] D. Andre, S.J. Russell, State abstraction for programmable reinforcement learning agents, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence, 2002, pp. 119–125.
[7] I. Arel, C. Liu, et al, Reinforcement learning-based multi-agent system for network traffic signal control, IET Intelligent Transport Systems 4 (2) (2010) 128–135.
[8] F.R. Bach, M.I. Jordan, Kernel independent component analysis, Journal of Machine Learning Research 3 (2002) 1–48.
[9] J.A. Bagnell, J.G. Schneider, Autonomous helicopter control using reinforcement learning policy search methods, in: Proceedings of the 2001 IEEE International Conference on Robotics & Automation, Seoul, Korea, 2001, pp. 1615–1620.
[10] J.A. Bagnell, J.G. Schneider, Covariant policy search, in: G. Gottlob, T. Walsh (Eds.), Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Morgan Kaufmann, San Francisco, CA, USA, 2003, pp. 1019–1024.
[11] L.C. Baird, Residual algorithms: reinforcement learning with function approximation, in: Proceedings of the 12th International Conference on Machine Learning (ICML 1995), Morgan Kaufman, San Francisco, CA, USA, 1995, pp. 30–37.
[12] P.G. Balaji, X. German, et al, Urban traffic signal control using reinforcement learning agents, IET Intelligent Transport Systems 4 (3) (2010) 177–188.
[13] S.N. Balakrishnan, V. Biega, Adaptive-critic-based neural networks for aircraft optimal control, Journal of Guidance, Control, Dynamics 19 (4) (1996) 893–898.
[14] A.G. Barto, T.G. Dietterich, Reinforcement learning and its relationship to supervised learning, in: J. Si, A. Barto, W. Powell, D. Wunsch (Eds.), Handbook of Learning and Approximate Dynamic Programming, Wiley-IEEE Press, New York, 2004.
[15] A.G. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, Discrete Event Dynamic Systems-Theory and Applications 13 (1-2) (2003) 41–77.
[16] A.G. Barto, R. Sutton, C.W. Anderson, Neuron-like adaptive elements that can solve difficult learning control problems, IEEE Transactions on Systems, Man, and Cybernetics 13 (5) (1983) 834–846.
[17] J. Baxter, P.L. Bartlett, Infinite-horizon policy-gradient estimation, Journal of Artificial Intelligence Research 15 (2001) 319–350.
[18] D.P. Bertsekas, J. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, 1996.
[19] S. Bhatnagar, R.S. Sutton, M. Ghavamzadeh, M. Lee, Natural actor–critic algorithms, Automatica 45 (11) (2009) 2471–2482.
[20] V.S. Borkar, Stochastic approximation with two time scales, Systems & Control Letters 29 (5) (1997) 291–294.
[21] V.S. Borkar, Stochastic Approximation: A Dynamical Systems Viewpoint, Cambridge University Press, 2008.
[22] J. Boyan, A.W. Moore, Generalization in reinforcement learning: safely approximating the value function, in: Advances in Neural Information Processing Systems, 1995, pp. 369–376.
[23] J. Boyan, Technical update: least-squares temporal difference learning, Machine Learning 49 (2–3) (2002) 233–246.
[24] J. Boyan, M. Littman, Packet routing in dynamically changing networks: a reinforcement learning approach, Advances in neural information processing systems 6 (NIPS 1994) (1994).
[25] S. Bradtke, B. Ydstie, A. Barto, Adaptive linear quadratic control using policy iteration, Univ. Massachusetts, Amherst, MA, Tech. Rep. CMPSCI-94-49, June 1994.
[26] S. Bradtke, Incremental Dynamic Programming for On-Line Adaptive Optimal Control, Ph.D. thesis, University of Massachusetts, Computer Science Dept. Tech. Rep., 1994, pp. 94–62.
[27] S.J. Brartke, A. Barto, Linear least-squares algorithms for temporal difference learning, Machine Learning 22 (1996) 33–57.
[28] L. Busoniu, R. Babuska, B. De Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, NY, 2010.
[29] X. Cao, Stochastic Learning and Optimization, Springer-Verlag, Berlin, 2009.
[30] M. Carreras, J. Yuh, et al, A behavior-based scheme using reinforcement learning for autonomous underwater vehicles, IEEE Journal of Oceanic Engineering 30 (2) (2005) 416–427.
[31] R.H. Crites, A.G. Barto, Elevator group control using multiple reinforcement learning agents, Machine Learning 33 (2–3) (1998) 235–262.

[32] R.H. Crites, A.G. Barto, Improving elevator performance using reinforcement learning, Advances in Neural Information Processing Systems 8 (NIPS 1995) (1996).
[33] C. Darken, J. Moody, Note on learning rate schedules for stochastic optimization, in: Lippman, et al. (Eds.), Advances in Neural Information Processing Systems, vol. 3, 1991, pp. 1009–1016.
[34] P. Dayan, The convergence of TD(λ) for general λ, Machine Learning 8 (1992) 341–362.
[35] P. Dayan, T.J. Sejnowski, TD(λ) converges with probability 1, Machine Learning 14 (1994) 295–301.
[36] T.G. Dietterich, X. Wang, Batch value function approximation via support vectors, Advances in Neural Information Processing Systems, vol. 14, MIT Press, Cambridge, MA, 2002, pp. 1491–1498.
[37] T.G. Dietterich, Hierarchical reinforcement learning with the Max-Q value function decomposition, Journal of Artificial Intelligence Research 13 (2000) 227–303.
[38] T.G. Dietterich, State abstraction in MAXQ hierarchical reinforcement learning, in: S.A. Solla, T.K. Leen, K.R. Muller (Eds.), Advances in Neural Information Processing Systems, NIPS, 2000, pp. 994–1000.
[39] K. Driessens, S. Dzeroski, Integrating guidance into relational reinforcement learning, Machine Learning 57 (2004) 271–304.
[40] K. Driessens, J. Ramon, H. Blockeel, Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner, in: L. De Raedt, P. Flach (Eds.), Proceedings of the 13th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, vol. 2167, Springer-Verlag, 2001, pp. 97–108.
[41] K. Driessens, J. Ramon, Relational instance based regression for relational reinforcement learning, in: Proceedings of the Twentieth International Conference on Machine Learning, AAAI Press, 2003, pp. 123–130.
[43] Y. Engel, S. Mannor, R. Meir, "Bayes meets bellman: the Gaussian Process approach to temporal difference learning, in: Proceedings of the Twentieth International Conference of Machine Learning, Washington, DC, 2003, pp. 154–161.
[44] R. Enns, J. Si, Helicopter trimming and tracking control using direct neural dynamic programming, IEEE Transactions on Neural Networks 14 (4) (2003) 929–939.
[45] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, Journal of Machine Learning Research 6 (2005) 503–556.
[46] D. Ernst, M. Glavic, et al, Power systems stability control: reinforcement learning framework, IEEE Transactions on Power Systems 19 (1) (2004) 427–435.
[47] A. Farahmand, Cs. Szepesvári, Model selection in reinforcement learning, Machine Learning 85 (3) (2011) 299–332.
[48] A.m. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, S. Mannor, Regularized policy iteration, NIPS (2008) 441–448.
[49] A. Galindo-Serrano, L. Giupponi, Distributed Q-Learning for aggregated interference control in cognitive radio networks, IEEE Transactions on Vehicular Technology 59 (4) (2010) 1823–1834.
[50] K. Driessens, J. Ramon, T. Gärtner, Graph kernels and Gaussian Processes for relational reinforcement learning, Machine Learning 64 (1–3) (2006) 91–119.
[51] T. Gärtner, P. Flach, S. Wrobel, On graph kernels: hardness results and efficient alternatives, in: M.W.B. Scholkopf (Ed.), Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop, 2003, pp. 129–143.
[52] A. Gosavi, Reinforcement learning for long-run average cost, European Journal of Operational Research 155 (2004) 654–674.
[53] A.P. George, W.B. Powell, Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming, Machine Learning 65 (2006) 167–198.
[54] A. Geramifard, M. Bowling, M. Zinkevich, R.S. Sutton, iLSTD: eligibility traces and convergence analysis, in: B. Scholkopf, J. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems, vol. 19, MIT Press, Cambridge, MA, 2007, pp. 441–448.
[55] M. Ghavamzadeh, Y. Engel, Bayesian policy gradient algorithms, in: Advances in Neural Information Processing Systems, 2006, pp. 457–464.
[56] M. Ghavamzadeh, S. Mahadevan, Hierarchical average reward reinforcement learning, Journal of Machine Learning Research 8 (2007) 2629–2669.
[57] D. Haussler, Convolution Kernels on Discrete Structures, Technical Report, Department of Computer Science, University of California at Santa Cruz, 1999.
[58] B. Hengst, Safe state abstraction and reusable continuing subtasks in hierarchical reinforcement learning, in: AI 2007: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 4830, 2007, pp. 58–67.
[59] Q.Y. Hu, W.Y. Yue, Markov Decision Processes with Their Applications, Springer, 2008.
[60] K.M. Iftekharuddin, Transformation invariant on-line target recognition, IEEE Transactions on Neural Networks 22 (6) (2011) 906–918.
[61] T. Jaakkola, M. Jordan, S. Singh, On the convergence of stochastic iterative dynamic programming algorithms, Neural Computation 6 (6) (1994) 185–1201.
[62] M.A.K. Jaradat, M. Al-Rousan, et al, Reinforcement based mobile robot navigation in dynamic environment, Robotics and Computer-Integrated Manufacturing 27 (2011) 135–149.
[63] T. Jiang, D. Grace, et al, Efficient exploration in reinforcement learning-based cognitive radio spectrum sharing, IET Communication 5 (10) (2011) 1309–1317.
[64] J. Johns, M. Petrik, S. Mahadevan, Hybrid least-squares algorithms for approximate policy evaluation, Machine Learning 76 (2009) 243–256.
[65] S. Kakade, A natural policy gradient, Advances in Neural Information Processing Systems (2002) 1531–1538.
[66] J. Kober, J. Peters, Policy search for motor primitives in robotics, Advances in Neural Information Processing Systems (NIPS 2008) (2008).
[67] N. Kohl, P. Stone, Machine learning for fast quadrupedal locomotion, in: D.L. McGuinness, G. Ferguson (Eds.), Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004), AAAI Press, Menlo Park, pp. 611–616.
[68] V.R. Konda, J.N. Tsitsiklis, Actor–critic algorithms, Advances in Neural Information Processing Systems, vol. 12, MIT Press, Cambridge, MA, 2000.
[69] M.G. Lagoudakis, R. Parr, Least-squares policy iteration, Journal of Machine Learning Research 4 (2003) 1107–1149.
[70] F.L. Lewis, G. Lendaris, D. Liu, Special issue on approximate dynamic programming and reinforcement learning for feedback control, IEEE Transactions on Systems, Man, and Cybernetics B 38 (4) (2008).
[71] F.L. Lewis, D. Vrabie, Reinforcement learning and adaptive dynamic programming for feedback control, IEEE Circuits and Systems Magazine 9 (3) (2009) 32–50.
[72] D. Liu, Y. Zhang, H. Zhang, A self-learning call admission control scheme for CDMA cellular networks, IEEE Transactions on Neural Networks 16 (5) (2005) 1219–1228.
[73] H.R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, R.S. Sutton, Convergent temporal-difference learning with arbitrary smooth function approximation, in: J. Laferty, C. Williams (Eds.), Advances in Neural Information Processing Systems, vol. 22, MIT Press, Cambridge, MA, USA, 2010.
[74] H.R. Maei, C. Szepesvári, S. Bhatnagar, R. Sutton, Toward off-policy learning control with function approximation, in: J. Fürnkranz, T. Joachims (Eds.), ICML, Omnipress, 2010, pp. 719–726.
[75] S. Mahadevan, Proto-value functions: developmental reinforcement learning, in: Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 553–560.
[76] S. Mahadevan, M. Maggioni, Proto-value functions: a laplacian framework for learning representation and control in markov decision processes, Journal of Machine Learning Research 8 (2007) 2169–2231.
[77] A.R. Mahmood, R. Sutton, T.Degris, P.M. Pilarski, Tuning-free step-size adaptation, in: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Kyoto, Japan, 2012.
[78] M. McPartl, M. Gallagher, Reinforcement learning in first person shooter games, IEEE Transactions on Computational Intelligence and AI in Games 3 (1) (2011) 43–56.
[79] M.L. Minsky, Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem, Ph.D. Thesis, Princeton University, 1954.

[80] J. Mitola III, G.Q. Maguire, Cognitive radio: making software radios more personal, IEEE Personal Communications 6 (4) (1999) 13–18.
[81] S. Mohagheghi, G.K. Venayagamoorthy, et al, Adaptive critic design based neuro-fuzzy controller for a static compensator in a multimachine power system, IEEE Transactions on Power Systems 21 (4) (2006) 1744–1754.
[82] V. Nanduri, T.K. Das, A reinforcement learning model to assess market power under auction-based energy pricingm, IEEE Transactions on Power Systems 22 (1) (2007) 85–95.
[83] A. Nedic, D.P. Bertsekas, Least squares policy evaluation algorithms with linear function approximation, Discrete Event Dynamic Systems 13 (1) (2003) 79–110.
[84] A.Y. Ng, H.J. Kim, et al, Autonomous helicopter flight via reinforcement learning, Advances in Neural Information Processing Systems 16 (NIPS 2003) (2004).
[85] J. O, J. Lee, J.W. Lee, B.-T. Zhang, Adaptive stock trading with dynamic asset allocation using reinforcement learning, Information Sciences 176 (15) (2006) 2121–2147.
[86] D. Ormoneit, S. Sen, Kernel-based reinforcement learning, Machine Learning 49 (2-3) (2002) 161–178.
[87] R. Parr, S. Russell, Reinforcement learning with hierarchies of machines, in: Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 1998, pp. 1043–1049.
[88] J. Peng, B. Bhanu, Delayed reinforcement learning for adaptive image segmentation and feature extraction, IEEE Transactions on System Man and Cybernetics-Part C 28 (3) (1998) 482–488.
[89] J. Peng, B. Bhaun, Closed-loop object recognition using reinforcement learning, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (2) (1998) 139–154.
[90] J. Peters, S. Schaal, Natural actor–critic, Neurocomputing 71 (2008) 1180–1190.
[91] J. Peters, S. Schaal, Policy gradient methods for robotics, in: Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots ans Systems, Beijing, China, 2006, pp. 2219–2225.
[92] J. Peters, S. Vijayakumar, S. Schaal, Reinforcement learning for humanoid robotics, in: IEEE/RSJ International Conference on Humanoid Robotics, 2003.
[93] W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley, NY, 2007.
[94] L.A. Prashanth, S. Bhatnagar, Reinforcement learning with function approximation for traffic signal control, IEEE Transactions on Intelligence Transportation Systems 12 (2) (2011) 412–421.
[95] D.V. Prokhorov, D.C. Wunsch, Adaptive critic designs, IEEE Transactions Neural Networks 8 (5) (1997) 997–1007.
[96] C.E. Rasmussen, M. Kuss, Gaussian processes in reinforcement learning, in: S. Thrun, L.K. Saul, B. Schölkopf (Eds.), Advances in Neural Information Processing Systems, vol. 16, MIT Press, 2004, pp. 751–759.
[97] M. Riedmiller, T. Gabel, et al, Reinforcement learning for robot soccer, Autonomous Robots 27 (1) (2009) 55–74.
[98] M. Riedmiller, M. Montemerlo, et al., Learning to drive in 20 min, in: Proceedings of the FBIT 2007 Conference, Jeju, Korea, 2007.
[99] S. Richter, D. Aberdeen, J. Yu, Natural actor–critic for road traffic optimisation, in: Advances in Neural Information Processing Systems, 2006, pp. 3522–3529.
[100] A.L. Samuel, Some studies in machine learning using game of checkers, IBM Jounal on Research and Development 3 (1959) 211–229.
[101] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002.
[102] B. Schölkopf, S. Mika, C.J.C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, A.J. Smola, Input space vs feature space in kernel-based algorithms, IEEE Transactions on Neural Networks 10 (3) (1999) 1000–1017.
[103] A. Schwartz, A reinforcement learning method for maximizing undiscounted rewards, in: Proceedings of the Tenth Annual Conference on Machine Learning, Morgan Kaufmann, 1993, pp. 298–305.
[104] T. Shimokawa, K. Suzuki, et al, Predicting investment behavior: an augmented reinforcement learning model, Neurocomputing 72 (2009) 3447–3461.
[105] D. Silver, R.S. Sutton, et al, Reinforcement learning of local shape in the game of Go, Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007) (2007) 1053–1058.
[106] S.P. Singh, D. Bertsekas, Reinforcement learning for dynamic channel allocation in cellular telephone systems, Advances in Neural Information Processsing Systems 9 (NIPS 1996) (1997) 974–980.
[107] S.P. Singh, T. Jaakkola, M.L. Littman, Cs. Szepesvari, Convergence results for single-step on-policy reinforcement-learning algorithms, Machine Learning 38 (2000) 287–308.
[108] S.P. Singh, R.C. Yee, An upper bound on the loss from approximate optimal value functions, Machine Learning 16 (3) (1994) 227–233.
[109] T. Söderström, P. Stoica, Instrumental Variable Methods in System Identification, Springer-Verlag, Berlin, 1983.
[110] P. Stone, R.S. Sutton, et al, Reinforcement learning for RoboCup-soccer keepaway, Adaptive Behavior 13 (3) (2005) 165–188.
[111] R. Sutton, A.G. Barto, Reinforcement Learning. An Introduction, MIT Press, Cambridge MA, 1998.
[112] R. Sutton, A.G. Barto, R.J. Williams, Reinforcement learning is direct adaptive control, IEEE Control Systems 12 (2) (1992) 19–22.
[113] R. Sutton, A.G. Barto, A temporal-difference model of classical conditioning, in: Proceedings of the 9th Annual Conference Cognitive Science Society, 1987, pp. 355–378.
[114] R. Sutton, Learning to predict by the method of temporal differences, Machine Learning 3 (1988) 9–44.
[115] R. Sutton, H.R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvari, E. Wiewiora, Fast gradient-descent methods for temporal-difference learning with linear function approximation, in: Proceedings of the 26th Annual International Conference on Machine Learning (ICML-09), 2009, pp. 993–1000.
[116] R. Sutton, C. Szepesvari, H.R. Maei, A convergent O(n) temporal-difference algorithm for off-policy learning with linear function approximation, Advances in Neural Information Processing Systems, vol. 21, MIT Press, Cambridge, MA, USA, 2009, pp. 1609–1616.
[117] R. Sutton, Adapting bias by gradient descent: an incremental version of delta-bar-delta, in: Proceedings of the 10th National Conference on Artificial Intelligence, 1992, pp. 171–176.
[118] R. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning, Artificial Intelligence 112 (1999) 181–211.
[119] R. Sutton, Cs. Szepesvári, A. Geramifard, M. Bowling, Dyna-style planning with linear function approximation and prioritized sweeping, UAI, 2008, pp. 528–536.
[120] Cs. Szepesvári, Algorithms for Reinforcement Learning, Morgan and Claypool, 2010.
[121] G. Tesauro, TD-Gammon, a self-teaching backgammon program, achieves master-level play, Neural Computation 6 (1994) 215–219.
[122] S.Thrun, A. Schwartz, Issues in using function approximation for reinforcement learning, in: Proceedings of the Fourth Connectionist Models Summer School, 1993, pp. 255–263.
[123] J.N. Tsitsiklis, Asynchronous Stochastic Approximation and Q-learning, Technical Report LIDS-P-2172, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1993.
[124] J.N. Tsitsiklis, B.V. Roy, An analysis of temporal difference learning with function approximation, IEEE Transactions on Automatic Control 42 (5) (1997) 674–690.
[125] W.T.B. Uther, M.M. Veloso, Tree based discretization for continuous state space reinforcement learning, in: Proceedings of AAAI-98, 1998, pp. 769–774.
[126] K.G. Vamvoudakis, Frank L. Lewis, Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem, Automatica 46 (5) (2010) 878–888.
[127] K.G. Vamvoudakis, Frank L. Lewis, Multi-player non-zero-sum games: online adaptive learning solution of coupled Hamilton–Jacobi equations, Automatica 47 (8) (2011) 1556–1569.
[128] V. Vapnik, Statistical Learning Theory, Wiley Interscience, NewYork, 1998.

[129] G.K. Venayagamoorthy, R.G. Harley, D.C. Wunsch, Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator, IEEE Transactions on Neural Networks 13 (3) (2002) 764–773.
[130] J.G. Vlachogiannis, N.D. Hatziargyriou, Reinforcement learning for reactive power control, IEEE Transactions on Power Systems 19 (3) (2004) 1225–1317.
[131] D. Vrabie, F. Lewis, M. Abu-Khalaf, Adaptive optimal control for continuous-time linear systems based on policy iteration, Automatica 45 (2) (2009) 477–484.
[132] D. Vrabie, F. Lewis, Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems, Neural Networks 22 (3) (2009) 237–246.
[133] X. Wang, Y. Cheng, J.-Q. Yi, A fuzzy actor–critic reinforcement learning network, Information Sciences 177 (18) (2007) 3764–3781.
[134] F.Y. Wang, H. Zhang, D. Liu, Adaptive dynamic programming: an introduction, IEEE Computational Intelligence Magazine (2009) 39–47.
[135] C. Watkins, Learning from Delayed Rewards, Ph.D. thesis, Cambridge Univ., Cambridge, England, 1989.
[136] C. Watkins, P. Dayan, Q-Learning, Machine Learning 8 (1992) 279–292.
[137] P.J. Werbos, Intelligence in the brain: a theory of how it works and how to build it, Neural Networks (2009) 200–212.
[138] P.J. Werbos, Using ADP to understand and replicate brain intelligence: the next level design, in: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 209–216.
[139] P.J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavior Sciences, Ph.D. thesis, Committee Appl. Math. Harvard Univ., 1974.
[140] B. Widrow, N. Gupta, S. Maitra, Punish/reward: Learning with a critic in adaptive threshold systems, IEEE Transactions on Systems, Man, and Cybernetics SMC-3 (5) (1973) 455–465.
[141] X. Xu, Reinforcement Learning and Approximate Dynamic Programming, Science Press, Beijing, 2010.
[142] X. Xu, H.G. He, D.W. Hu, Efficient reinforcement learning using recursive least-squares methods, Journal of Artificial Intelligence Research 16 (2002) 259–292.
[143] X. Xu, T. Xie, D.W. Hu, X.C. Lu, Kernel least-squares temporal difference learning, International Journal of Information Technology 11 (9) (2005) 54–63.
[144] X. Xu, D.W. Hu, X.C. Lu, Kernel based least-squares policy iteration for reinforcement learning, IEEE Transactions on Neural Networks 18 (4) (2007) 973–992.
[145] X. Xu, Sequential anomaly detection based on temporal-difference learning: principles, models and case studies, Applied Soft Computing 10 (3) (2010) 859–867.
[146] X. Xu, C. Liu, S. Yang, D. Hu, Hierarchical approximate policy iteration with binary-tree state space decomposition, IEEE Transactions on Neural Networks 22 (12) (2011) 1863–1877.
[147] X. Xu, C. Liu, D. Hu, Continuous-action reinforcement learning with fast policy search and adaptive basis function selection, Soft Computing – A Fusion of Foundations, Methodologies and Applications 15 (6) (2011) 1055–1070.
[148] X. Xu, Z. Hou, C. Lian, H. He, Online learning control using adaptive critic designs with sparse kernel machines, IEEE Transactions on Neural Networks and Learning Systems 24 (5) (2013) 762–775.
[149] K.-L.A. Yau, P. Komisarczuk, et al., Applications of reinforcement learning to cognitive radio networks, 2010 IEEE International Conference on Communication Workshops (ICC), 2010, pp. 1–6.
[150] P. Yin, B. Bhanu, et al, Integrating relevance feedback techniques for image retrieval using reinforcement learning, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (10) (2005) 1536–1551.
[151] T. Yu, B. Zhou, et al, Stochastic optimal relaxed automatic generation control in non-Markov environment based on multi-step Q(λ) learning, IEEE Transactions on Power Systems 26 (3) (2011) 1272–1282.
[152] H. Zhang, L. Cui, X. Zhang, Y. Luo, Data-driven robust approximate optimal tracking control for unknown general nonlinear systems using adaptive dynamic programming method, IEEE Transactions on Neural Networks 22 (12) (2011) 2226–2236.
[153] W. Zhang, T.G. Dietterich. A reinforcement learning approach to job-shop scheduling, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995), 1995, pp. 1114–1120.
[154] P. Zhou, Y. Chang, et al, Reinforcement learning for repeated power control game in cognitive radio networks, IEEE Journal on Selected Areas in Communications 30 (1) (2012) 54–69.
[155] C. Zhou, Robot learning with GA-based fuzzy reinforcement learning agents, Information Sciences 145 (2002) 45–68.

Further reading

[42] S. Dzeroski, L. De Raedt, H. Blockeel, Relational reinforcement Learning, in: Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, 1998, pp. 136–143.