Front. Comput. Sci., 2019, 13(1): 106–126
https://doi.org/10.1007/s11704-017-6222-6

RESEARCH ARTICLE

Efficient reinforcement learning in continuous state and action spaces with Dyna and policy approximation

Shan ZHONG[1,2,3,4], **Quan LIU** (✉)[1,4,5], **Zongzhang ZHANG**[1], **Qiming FU**[1,3,4,6]

1 School of Computer Science and Technology, Soochow University, Suzhou 215000, China
2 School of Computer Science and Engineering, Changshu Institute of Technology, Changshu 215500, China
3 Jiangsu Province Key Laboratory of Intelligent Building Energy Efficiency, Suzhou University of Science and Technology, Suzhou 215006, China
4 Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China
5 Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210000, China
6 College of Electronic & Information Engineering, Suzhou University of Science and Technology, Suzhou 215000, China

Abstract Dyna is an effective reinforcement learning (RL) approach that combines value function evaluation with model learning. However, existing works on Dyna mostly discuss only its efficiency in RL problems with discrete action spaces. This paper proposes a novel Dyna variant, called Dyna-LSTD-PA, aiming to handle problems with continuous action spaces. Dyna-LSTD-PA stands for Dyna based on least-squares temporal difference (LSTD) and policy approximation. Dyna-LSTD-PA consists of two simultaneous, interacting processes. The learning process determines the probability distribution over action spaces using the Gaussian distribution; estimates the underlying value function, policy, and model by linear representation; and updates their parameter vectors online by LSTD(λ). The planning process updates the parameter vector of the value function again by using offline LSTD(λ). Dyna-LSTD-PA also uses the Sherman–Morrison formula to improve the efficiency of LSTD(λ), and weights the parameter vector of the value function to bring the two processes together. Theoretically, the global error bound is derived by considering approximation, estimation, and model errors. Experimentally, Dyna-LSTD-PA outperforms two representative methods in terms of convergence rate, success rate, and stability performance on four benchmark RL problems.

Keywords problem solving, control methods, heuristic search methods, dynamic programming

1 Introduction

The reinforcement learning (RL) framework is a considerable abstraction of the problem of a goal-directed learning agent interacting with an uncertain environment [1–3]. At each time step, the agent learns to select the actions yielding the highest return by trying them. The effect of taking an action reflects on not only the immediate reward but also the next state [4]. The agent's goal is to learn optimal mapping from states to actions that maximizes the expected return. Dynamic programming (DP) [5], heuristic search [6], Monte Carlo (MC) methods [7, 8], and temporal difference (TD) learning [9, 10] are important elementary solution methods in RL. DP as a model-based offline method has successfully handled many control problems, but its backward implementation in time makes it still computationally expensive for real-time problems. Approximate dynamic programming (ADP) solved the

Received April 13, 2016; accepted January 16, 2017

E-mail: quanliu@suda.edu.cn, sunshine-620@163.com

Hamilton–Jacobi–Bellman (HJB) equation approximately by combining DP, RL, and function approximation to overcome the curse of the dimensionality of DP. Werbos [11] introduced adaptive critic designs (ACDs) to obtain the solution of ADP, where two neural networks (NNs) are used to approximate the actor network and the critic network.

The iterative nature of the formulation process for ACD makes it naturally suitable to be applied to discrete domains [12–14]. However, when referring to continuous spaces, the variants of ACD have difficulties in satisfying the requirements of stability, convergence, and being online. Murray et al. [15] described an improved ACD by fusing soft computing techniques to learn the optimal cost function for a stabilizable continuous nonlinear system. Hanselmann et al. [16] constructed a continuous-time formulation of an ACD, where a second-order actor adaption using Newton's method is established to accelerate the convergence of the optimal policy. Wei et al. [17] introduced a data-driven zero-sum optimal ACD under the consideration of system disturbance, where three single-layer neural networks are used to approximate the performance index function, the optimal policy, and the disturbance.

All the aforementioned ACD variants use the NN as the nonlinear function approximator while not learning the model of the environment dynamics. Linear parameterized approximators are usually more preferred in RL due to the easy analysis of theoretical properties [18]. Moreover, the model can be approximated during learning and then be used further to accelerate learning. By using model learning, Dyna-Q [19], as a simple, but well-known Dyna architecture to integrate incremental planning methods with acting and model-learning was proposed. Prioritized sweeping [20, 21], as an improved Dyna, focuses on prioritizing the backups according to their priorities. A Dyna algorithm based on linear function approximation and prioritized sweeping, called Dyna-LFA-PS, was put forward in [22] to handle problems with continuous state spaces. Recently, Dyna-2 [23] and Monte Carlo tree search algorithms [24] achieved great success in the computer game Go. Dyna-2's key idea is to use a model-free method TD(λ) to evaluate the action-value function Q and the temporal action-value function $\bar{Q}$ in the learning and planning processes, respectively. Generally speaking, Dyna-2 has better convergence performance and generalization ability than Monte Carlo tree search. Another recent work introduced DynaTSVI [25], applying the value iteration algorithm to the planning process, to improve Dyna-2's convergence rate and accuracy.

In the above works, only Dyna-LFA-PS [22] is able to deal with the problems with continuous state spaces. However, when the action space is continuous, more natural in real-world tasks, these methods are not applied directly. One feasible way is to discretize the continuous action space (e.g., [26–28]), but discretizing the continuous action space is associated with issues such as (1) how to discretize the action space reasonably; and (2) how to generalize the learned optimal policy to the whole continuous action space effectively. Another existing approach for continuous action spaces is to represent the policy via function approximation in the framework of actor-critic, and update the policy according to the policy gradient (e.g., [29–31]). Nevertheless, in real-time systems, such an approach cannot avoid the problem of low sample efficiency.

This paper aims to develop an efficient algorithm for continuous-action Markov decision processes (MDPs), called Dyna-LSTD-PA, which stands for Dyna based on least-squares temporal difference (LSTD) [32] and policy approximation. In Dyna-LSTD-PA, the value function, policy, and model are represented by linear function approximators. Their parameter vectors are learned by LSTD(λ) [33, 34]. Dyna-LSTD-PA combines policy learning with the TD error of the value function to improve the updating efficiency of the policy. It also weights the parameter vectors of the value functions from the learning and planning processes to generate the ultimate parameter vector. Theoretically, we derive a global error bound for Dyna-LSTD-PA. We finally tested Dyna-LSTD-PA on four well-known benchmark problems. Empirical results show that it outperforms two representative methods in terms of convergence rate, success rate, and sample efficiency.

This paper is organized as follows. Section 2 introduces the notations of the MDP and the value function. Section 3 specifies the related work. Section 4 describes our Dyna-LSTD-PA algorithm. Section 5 derives the global error bound of our algorithm. Empirical results are analyzed and discussed in Section 6. We conclude with a discussion of future topics in Section 7.

2 Notation

In RL, the learning task satisfying the Markov property can be modeled by a MDP. It can be defined as a quadruple $M = \langle X, U, r, f \rangle$:

- X is a set of system states;
- U is a set of system actions;

- $r : X \times U \times X \to \mathbb{R}$ denotes the reward function, where $r = r(x, u, x')$ is the reward obtained from the environment when ending up at state x' after executing action u at state x, and $\mathbb{R}$ is the real number space;
- $f : X \times U \times X \to [0, 1]$ defines the state transition function, where $f(x, u, x')$ is the probability of ending up at state x' after executing action u at state x.

The cumulative reward from $k = t + 1$ to n is denoted by

$$R_t = \sum_{k=t+1}^{n} r_k. \tag{1}$$

Policy h is a mapping from the state space X to the action space U. Under policy h, the state value function is the expected return starting from state x, and $V^h(x)$ has the form

$$V^h(x) = \mathbb{E}_h\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | x_t = x\}, \tag{2}$$

where $0 < \gamma \leqslant 1$ is a discounted factor.

The action value function $Q^h(x, u)$ represents the expected return by following policy h after executing action u at state x. That is

$$Q^h(x, u) = \mathbb{E}_h\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | x_t = x, u_t = u\}. \tag{3}$$

The optimal state value function $V^*(x)$ satisfies

$$V^*(x) = \max_h V^h(x), \forall x \in X. \tag{4}$$

The optimal action value function $Q^*(x, u)$ satisfies

$$Q^*(x, u) = \max_h Q^h(x, u), \forall x \in X, \forall u \in U. \tag{5}$$

The optimal policy $h^*(x)$ is the policy that maximizes the value for every state-action pair, i.e.,

$$h^*(x) = \arg\max_u Q^*(x, u). \tag{6}$$

A sample is a quadruple $(x_t, u_t, r_{t+1}, x_{t+1})$, where x_t is the current state, u_t is the action executing at x_t, r_{t+1} is the immediate reward reflected by the environment, and x_{t+1} is the next state.

3 Related work

3.1 Dyna structure

Dyna-Q [19] provides an effective way to achieve incremental planning while maintaining responsiveness. The structure of Dyna-Q is shown in Fig. 1. Dyna-Q represents the value function, policy, and model in tabular form, and it consists of two interacting processes: learning and planning. The learning process is specified as follows: the agent interacts with the environment and obtains the observed samples. The observed samples are used to evaluate the value function and learn the model. However, the planning process is used to evaluate the value function again by using the learned model. The value function is updated by the TD method. At each time step, the policy is updated according to the evaluation of the current value function. When Dyna-Q converges, the optimal policy h^* can be obtained.

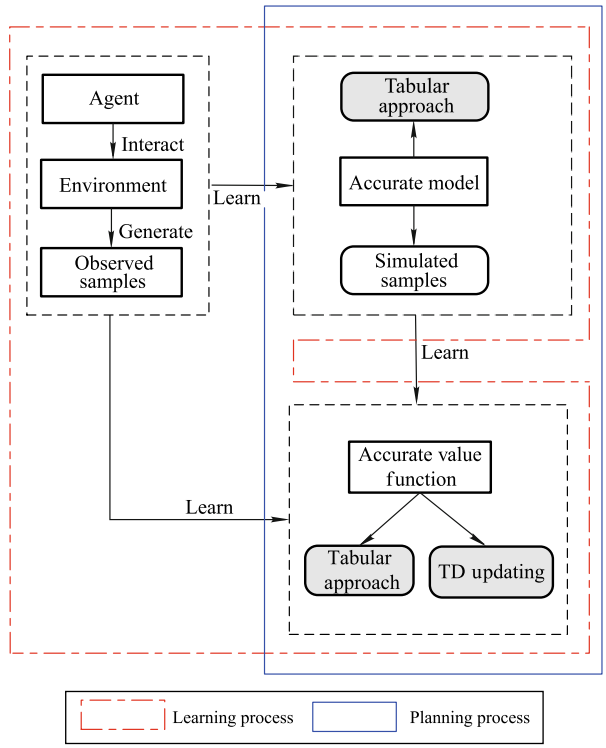


Fig. 1 The structure of Dyna-Q

Most of the subsequent extensions of Dyna-Q (e.g., [20, 21, 25, 35]) also use the tabular representation in their approaches. Thus, they are limited to relative small problems.

Function approximation can be combined with reinforcement learning to cope with most larger problems. The states or state-action pairs are mapped to feature vectors, which are then with the learned parameters, mapped to the value and model estimates. The function approximator can be nonlinear (e.g., [36]) or linear (e.g., [22, 37, 38]). Linear function approximation has the advantages such as simple representation and strong convergence guarantee. Thus, we represent the value function, model, and policy by linear function approximation. Being able or unable to apply to continuous action spaces is the main difference between our work and other

Dyna variants.

3.2 LSTD(λ)

LSTD(0), which stands for the least-squares temporal difference learning algorithm, and its recursive version RLSTD(0) were proposed with convergence guarantees for prediction problems [32]. From the viewpoint of statistics, LSTD(0) and RLSTD(0) are more efficient than the conventional linear TD(λ). Additionally, they eliminate the requirement of designing the step-size schedule. However, LSTD(0) and RLSTD(0) just update the value function based on one-step experience so that the information from historical data is not extracted sufficiently as LSTD(λ) does. LSTD(λ) [33] extends the work of [32] by generalizing λ from 0 to any arbitrary value. Its goal is to estimate the solution of the equation

$$V = \Pi T^{\lambda} V, \tag{7}$$

where T is the Bellman operator with the form

$$(TV)(x) = R(x) + \gamma \int_{y \in X} P(y|x)V(y)\mathrm{d}y, \tag{8}$$

and T^{λ} is a weighted result of the application of the power T^i on T:

$$\forall \lambda \in (0,1), (T^{\lambda}V)(x) = (1-\lambda)\sum_{i=0}^{\infty} \lambda^i (TV)^{i+1}(x). \tag{9}$$

Let S be the subspace spanned by feature vectors $(\phi_j)_{j\in\{1,2,\ldots,d\}}$, and Π be the orthogonal projection with the form

$$\Pi = \Phi(\Phi^{\mathrm{T}} D_{\rho} \Phi)^{-1} \Phi^{\mathrm{T}} D_{\rho}, \tag{10}$$

where Φ is the feature matrix with dimension $|X| \times d$, ρ is the stationary distribution of the Markov chain, and D_{ρ} is the diagonal matrix with the value ρ on the diagonal.

Π is nonexpansive with respect to ρ [39], and ΠT^{λ} is contracting; therefore, Eq. (7) has a unique solution [40]. Let $V_{LSTD(\lambda)}$ be the solution of LSTD(λ); then there always exists a $\theta \in \mathbb{R}^d$ so that

$$V_{LSTD(\lambda)} = \Phi\theta = \Pi T^{\lambda} \Phi\theta. \tag{11}$$

After applying Eqs. (8) and (9), θ can be estimated by solving $(A - \gamma B)\theta = b$, where $A - \gamma B$ is represented as

$$\begin{aligned} A - \gamma B &= \Phi^{\mathrm{T}} D_{\rho}(I - \gamma P)(I - \gamma\lambda P)^{-1}\Phi \\ &= \mathrm{E}_{x_{-\infty}\sim\rho}\left[\sum_{k=-\infty}^{i} (\gamma\lambda)^{i-k}\phi(x_k)(\phi(x_i) - \gamma\phi(x_{i+1}))^{\mathrm{T}}\right], \end{aligned} \tag{12}$$

and b is computed as

$$b = \Phi^{\mathrm{T}} D_{\rho}(I - \gamma\lambda P)^{-1} r = \mathrm{E}_{x_{-\infty}\sim\rho}\left[\sum_{k=-\infty}^{i} (\gamma\lambda)^{i-k}\phi(x_k) r(x_{i+1})\right], \tag{13}$$

where P is the transition kernel.

If the samples (x_0, u_0, x_1, r_1), (x_1, u_1, x_2, r_2), $\ldots$, $(x_{n-1}, u_{n-1}, x_n, r_n)$ are generated from the Markov chain, the expected expressions of $\hat{A} - \gamma\hat{B}$ and $\hat{b}$ can be estimated as

$$\hat{A} - \gamma\hat{B} = \frac{1}{n-1}\sum_{i=0}^{n-1} z_i(\phi(x_i) - \gamma\phi(x_{i+1}))^{\mathrm{T}}, \tag{14}$$

and

$$\hat{b} = \frac{1}{n-1}\sum_{i=0}^{n-1} z_i r(x_i), \tag{15}$$

where

$$z_i = \sum_{k=0}^{i} (\gamma\lambda)^{i-k}\phi(x_k). \tag{16}$$

If the number of samples is infinite, the asymptotic convergence of LSTD(λ) can be guaranteed [40]. Given a finite number of samples n and a β-mixing assumption, a high probability of error bound when λ=0 with a rate $O(\frac{1}{\sqrt{n}}\log^k(\frac{1}{\sqrt{n}}))$ was derived by Lazaric et al. [41]. With the same probability, Tagorti et al. [34] deduced a global error bound that consists of approximation error and estimation error, with $0 \leqslant \lambda < 1$.

Though LSTD (λ) has an advantage over TD (λ) in sample efficiency, it requires more computation per time step. Inspired by reducing the computational cost, Xu et al. [42] put forward a recursive algorithm by combining LSTD, the eligibility trace, and the recursive inversion formula [43], called RLSTD (λ). They also proved RLSTD (λ) converges with probability 1 in the case of ergodic Markov chains. Afterward, Geramifard et al. [44] focused on developing an incremental least-squares TD learning algorithm, iLSTD(0), to reduce the computational cost from O(n^2) to O(n) at each time step. Notice that iLSTD(0) can achieve the computational cost O(n) only if the features are sparse. Other than RLSTD(λ) [42], iLSTD(0) chooses the Sherman–Morrison formula as the recursive inversion equation due to its simpler form. Like LSTD(0) and RLSTD(0), iLSTD(0) also does not use the eligibility trace to incorporate multiple information of the experiences.

The above works only take LSTD or its recursive version to learn the value function, but the convergence of the whole algorithm will also heavily depend on the learning of other parts such as the policy and the model. Consequently, we derive a recursive LSTD(λ) algorithm by combining LSTD(λ)

and the Sherman–Morrison formula so that the value function, the policy, and the model can be learned effectively.

4 Dyna-LSTD-PA: improved Dyna

4.1 Dyna-LSTD-PA

The structure of our new algorithm, Dyna-LSTD-PA, is shown in Fig. 2. Like Dyna-Q, Dyna-LSTD-PA also contains two simultaneous, interacting processes. However, the model, value function, and policy in Dyna-LSTD-PA are represented by linear approximation. The parameter vector in the approximator is updated by LSTD(λ). The approximate model generates the simulated samples, which are used to evaluate the value function.

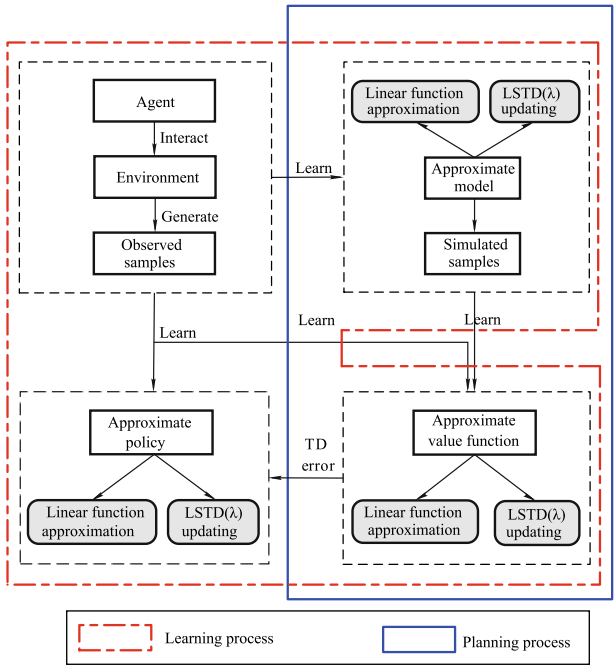


Fig. 2 The structure of Dyna-LSTD-PA

Dyna-LSTD-PA differs from Dyna-Q and the other variants of Dyna mainly in the following aspects:

1) It takes the exploration policy according to the Gaussian distribution. The variance decays gradually to leverage between exploration and exploitation.
2) It represents the policy by linear approximation, making problems with continuous action spaces be solvable. The value function and the model are also linearly approximated. The parameter vectors of the value function, model, and policy are updated by LSTD(λ). The Sherman–Morrison formula is combined with LSTD(λ) to improve Dyna-LSTD-PA's efficiency.
3) It weights the parameter vectors from learning and planning processes. These weights are changeable over time, to adaptively adjust the parameter vectors in the two processes.
4) It derives a global error bound with high probability that consists of approximation error, estimation error, and model error.

4.2 Exploration policy

The exploration policy is used to select actions in the interaction with the environment, with the goal of searching the state space as fully as possible. If the optimal action is represented by u^*, then at state x, action u is selected according to

$$h(x,u)=\frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{1}{2\sigma^2}(u-u^*)^2\}, \tag{17}$$

where σ is the variance. The variance decays as the algorithm runs, to explore more in the preceding phase and exploit more in the latter phase.

4.3 Linear approximation

A function approximator represents a mapping from the parameter space to the goal function space. We use an approximator $F: X \to V$ to represent the value function, where X is the state space and V is the value function space. Thus, $\hat{V}$ is an approximate value function of the parameter vector $\theta \in \mathbb{R}^d$:

$$\hat{V}=F(\theta). \tag{18}$$

Under policy h, at state $x \in X$, the approximate value function $\hat{V}^h(x)$ is represented as

$$\hat{V}^h(x)=[F(\theta)](x). \tag{19}$$

If we denote the feature vector for every state by $\phi_1,\phi_2,\ldots,\phi_d: X \to \mathbb{R}$, and the parameter vector by $\theta_1,\theta_2,\ldots,\theta_d: X \to \mathbb{R}$, then the approximate state value function is given by

$$\hat{V}^h(x)=\phi^{\mathrm{T}}(x)\theta=\sum_{i=1}^{d}\phi_i(x)\theta_i, \tag{20}$$

where $\phi^{\mathrm{T}}(x)=[\phi_1(x),\phi_2(x),\ldots,\phi_d(x)]$.

To be applied in continuous action space, the policy is approximated linearly. Let $\phi=[\phi_1,\phi_2,\ldots,\phi_d]^{\mathrm{T}}$ be the feature vector at the state x, and $\beta=[\beta_1,\beta_2,\ldots,\beta_d]^{\mathrm{T}}$ be the parameter vector; then the approximate optimal policy $\hat{h}$ can be represented as

$$\hat{h}=\phi^{\mathrm{T}}(x)\beta=\sum_{i=1}^{d}\phi_i(x)\beta_i. \tag{21}$$

The model in reinforcement learning problems refers to the state transition function and the reward function. However, it cannot be represented explicitly in continuous state and action spaces. Thus, we use the feature transition function $F : \Phi \to \Phi$ to replace the state transition function, where Φ denotes the feature space. We use $a : \Phi \to R$ to denote the reward function, where R is the reward space. Let $\phi = \phi(x)$ be the feature of the current state x; then the next expected feature ϕ' and the expected reward r can be linearly approximated by

$$\begin{cases} \phi' \leftarrow F\phi, \\ r \ \leftarrow \phi^{\mathrm{T}} a. \end{cases} \tag{22}$$

4.4 Value function learning

Most works of Dyna use TD to learn the value function. However, as pointed out by Bradke and Barto in [32], TD is associated with the following issues: (1) it uses the gradient descent approach to update the parameter vector; (2) it uses the sample once and then drops it. The first one might cause that the settings for the learning rate and the initial value of the parameter vector have a big effect on the algorithm's efficiency. The second one probably results in its poor sample efficiency. However, LSTD has no requirement of setting values for the learning rate or parameter vectors. Moreover, LSTD is sample-efficient since it stores the samples. Thus, we use LSTD(λ) to learn the value function, policy, and model.

The agent interacts with the environment and receives a sequence of samples as (x_0, u_0, x_1, r_1), (x_1, u_1, x_2, r_2), $\ldots$, $(x_{n-1}, u_{n-1}, x_n, r_n)$. Assume that the current policy is h. Let $\hat{V}^h(x)$ be the approximate value function, and $V^h(x)$ be the real value function. Then the approximate error between $\hat{V}^h(x)$ and $V^h(x)$ at state x_t, $\varepsilon^h(x)$, can be expressed as

$$\varepsilon^h(x_t) = V^h(x_t) - \hat{V}^h(x_t), \tag{23}$$

where $\hat{V}^h(x) = \phi^{\mathrm{T}}(x)\theta = \sum_{i=1}^{d} \phi_i(x)\theta_i$.

To minimize $\varepsilon^h(x_t)$, we formulate the goal function $J(\theta)$ as

$$J(\theta) = \frac{1}{2n} \sum_{t=0}^{n} [V^h(x_t) - \phi^{\mathrm{T}}\theta]^2, \tag{24}$$

where $J(\theta)$ is the square function of the approximate error. Since $V^h(x_t)$ is unknown, we use the Bellman equation of the value function to replace it, as below:

$$V^h(x_t) = r_{t+1} + \gamma \hat{V}^h(x_{t+1}). \tag{25}$$

From $\dfrac{\partial J(\theta)}{\partial \theta} = 0$, we have

$$\theta = [\frac{1}{n} \sum_{t=0}^{n-1} \phi(x_t)(\phi(x_t) - \gamma\phi(x_{t+1}))^{\mathrm{T}}]^{-1} [\frac{1}{n} \sum_{t=0}^{n-1} r_{t+1}\phi(x_t)]. \tag{26}$$

To transfer Eq. (26) to the matrix form, we define $\hat{A}$ as $\sum_{t=0}^{n-1} \phi(x_t)\phi^{\mathrm{T}}(x_t)$, $\hat{B}$ as $\sum_{t=0}^{n-1} \phi(x_t)\phi^{\mathrm{T}}(x_{t+1})$, and $\hat{b}$ as $\sum_{t=0}^{n-1} r_{t+1}\phi(x_t)$. Note that both $\hat{A}$ and $\hat{B}$ are $d \times d$ matrices and $\hat{b}$ is a $d \times 1$ vector. Therefore,

$$\theta = (\hat{A} - \gamma\hat{B})^{-1}\hat{b}. \tag{27}$$

LSTD(λ) is an extension to LSTD obtained by combining LSTD with the eligibility trace z_t, where z_t is updated by

$$z_t \leftarrow \gamma\lambda z_{t-1} + \phi(x_t). \tag{28}$$

By introducing z_t, $\hat{A}_{t+1}$, $\hat{B}_{t+1}$ and $\hat{b}_{t+1}$ can be updated as

$$\begin{cases} \hat{A}_{t+1} \leftarrow \hat{A}_t + z_t\phi^{\mathrm{T}}(x_t), \\ \hat{B}_{t+1} \leftarrow \hat{B}_t + z_t\phi^{\mathrm{T}}(x_{t+1}), \\ \hat{b}_{t+1} \leftarrow \hat{b}_t + z_t r_{t+1}. \end{cases} \tag{29}$$

After establishing $\hat{A}_n$, $\hat{B}_n$ and $\hat{b}_n$ for n independent trajectories, θ_λ can be estimated as $\theta_\lambda = (\hat{A}_n - \gamma\hat{B}_n)^{-1}\hat{b}_n$. The computational cost of $(\hat{A}_n - \gamma\hat{B}_n)^{-1}$ is $O(d^3)$.

The Sherman–Morrison formula can transfer the direct inversion of matrix to the following incremental computation:

$$(X + uv^{\mathrm{T}})^{-1} = X^{-1} - \frac{X^{-1}uv^{\mathrm{T}}X^{-1}}{1 + v^{\mathrm{T}}X^{-1}u}, \tag{30}$$

where X is a square matrix, and u and v are column vectors with the same dimension. Eq. (30) holds if satisfying the condition of $1 + v^{\mathrm{T}}X^{-1}u \neq 0$. It can reduce the computational cost of $(X + uv^{\mathrm{T}})^{-1}$ from $O(n^3)$ to $O(n^2)$, where n is the dimensionality of X.

To apply the Sherman–Morrison equation, the inversion of equation $(A_{t+1} - \gamma B_{t+1})$ is transferred as

$$(A_{t+1} - \gamma B_{t+1})^{-1} = (A_t - \gamma B_t + z_t(\phi(x_t) - \gamma\phi(x_{t+1}))^{\mathrm{T}})^{-1}. \tag{31}$$

Let $C_{t+1} = (A_{t+1} - \gamma B_{t+1})^{-1}$. Then, by Eq. (30), we have

$$\begin{aligned} C_{t+1} \leftarrow C_t - \frac{C_t z_t(\phi(x_t) - \gamma\phi(x_{t+1}))^{\mathrm{T}} C_t}{1 + (\phi(x_t) - \gamma\phi(x_{t+1}))^{\mathrm{T}} C_t z_t}, \\ \text{s.t.} \quad 1 + (\phi(x_t) - \gamma\phi(x_{t+1}))^{\mathrm{T}} C_t z_t \neq 0. \end{aligned} \tag{32}$$

Therefore, we have

$$\theta_\lambda = C_{t+1} b_{t+1}. \tag{33}$$

Thus, by combining the Sherman–Morrison formula with the LSTD(λ) method, Dyna-LSTD-PA can learn the parameter vectors of the value function, policy, and model.

4.5 Policy learning

At each time step t, the agent selects the action u_t at state x_t; after the action u_t is executed, the current state x_t is transferred to the next state x_{t+1}, and the agent receives an immediate reward r_{t+1}. The TD error of the value function is $\delta_t = r_{t+1} + \gamma\hat{V}(x_{t+1}) - \hat{V}(x_t)$.

At state x, assume that two actions u_1 and u_2 are available. Let the next state, immediate reward, and TD error after executing u_1 be x', r', and δ', respectively. For u_2, the corresponding ones are denoted by x'', r'', and δ'', respectively. If $\delta' > \delta''$, we have

$$\begin{aligned} &r' + \gamma\hat{V}(x') - \hat{V}(x) > r'' + \gamma\hat{V}(x'') - \hat{V}(x) \\ &\Rightarrow r' + \gamma\hat{V}(x') > r'' + \gamma\hat{V}(x''). \end{aligned} \tag{34}$$

Equation (34) shows that the action u_1 is better than the action u_2 at state x; hence, we adjust the parameter vector to increase the selection probability of u_1. Otherwise, the selection probability of the action u_2 will be increased but not that of the action u_1. Therefore, we use the TD error of the value function in defining $J(\beta)$, to improve the updating efficiency of the policy. The goal function $J(\beta)$ is defined as

$$J(\beta) = \frac{1}{2n}\sum_{t=0}^{n}\delta_t[u - \phi^{\mathrm{T}}(x_t)\beta]^2. \tag{35}$$

Our goal is to minimize $J(\beta)$. From $\dfrac{\partial J(\beta)}{\partial\beta} = 0$, we have

$$\beta = [\frac{1}{n}\sum_{t=0}^{n}\delta_t\phi(x_t)\phi^{\mathrm{T}}(x_t)]^{-1}[\frac{1}{n}\sum_{t=0}^{n}\delta_t u\phi(x_t)]. \tag{36}$$

To transfer Eq. (36) to the matrix form, we build the expression of the eligibility trace z'_{t+1} (of dimension $d \times 1$), $D_{t+1} = \sum_{t=0}^{n}\delta_t\phi(x_t)\phi^{\mathrm{T}}(x_t)$ (of dimension $d \times d$) and $e_{t+1} = \sum\limits_{t=0}^{n}\delta_t u\phi(x_t)$ (of dimension $d \times 1$) as

$$\begin{cases} z'_t \leftarrow \gamma\lambda z'_{t-1} + \phi(x_t), \\ D_{t+1} \leftarrow D_t + \delta_t z'_t\phi^{\mathrm{T}}(x_t), \\ e_{t+1} \leftarrow e_t + \delta_t u z'_t. \end{cases} \tag{37}$$

Thus, we have

$$\beta_{t+1} = D_{t+1}^{-1}e_{t+1}. \tag{38}$$

Let $E_{t+1} = D_{t+1}^{-1}$. According to Eq. (30), E_{t+1} can be represented as

$$E_{t+1} = E_t - \frac{E_t\delta_t z'_t\phi^{\mathrm{T}}(x_t)E_t}{1 + \phi^{\mathrm{T}}(x_t)E_t\delta_t z'_t}. \tag{39}$$

Using E_{t+1}, we have

$$\beta_{t+1} = E_{t+1}e_{t+1}. \tag{40}$$

4.6 Model learning

In Dyna-LFA-PS [22], the model is linearly approximated, and learned by TD. Here, we use the improved LSTD(λ) to learn the model. Let $J(F)$ be the goal function of the feature transition function F and $J(a)$ be the goal function of the reward function a; then $J(F)$ and $J(a)$ are expressed as

$$\begin{cases} J(F) = \dfrac{1}{2n}\sum\limits_{t=0}^{n}[\phi(x_{t+1}) - F\phi(x_t)]^2, \\ J(a) = \dfrac{1}{2n}\sum\limits_{t=0}^{n}[r_{t+1} - \phi^{\mathrm{T}}(x_t)a]^2. \end{cases} \tag{41}$$

From $\dfrac{\partial J(F)}{\partial F} = 0$ and $\dfrac{\partial J(a)}{\partial a} = 0$, we have

$$\begin{cases} F = \sum\limits_{t=0}^{n}(\phi(x_{t+1})\phi^{\mathrm{T}}(x_t))(\sum\limits_{t=0}^{n}\phi(x_t)\phi^{\mathrm{T}}(x_t))^{-1}, \\ a = (\sum\limits_{t=0}^{n}\phi(x_t)\phi^{\mathrm{T}}(x_t))^{-1}(\sum\limits_{t=0}^{n}r_{t+1}\phi(x_t)). \end{cases} \tag{42}$$

To transfer Eq. (42) to the matrix form, let M be $\sum\limits_{t=0}^{n}\phi(x_{t+1})\phi^{\mathrm{T}}(x_t)$, N be $\sum\limits_{t=0}^{n}\phi(x_t)\phi^{\mathrm{T}}(x_t)$, and q be $\sum\limits_{t=0}^{n}r_{t+1}\phi(x_t)$. By introducing z_t (see Eq. (28)), M_{t+1} (of dimension $d \times d$), N_{t+1} (of dimension $d \times d$), and q_{t+1} (of dimension $d \times 1$) can be represented as

$$\begin{cases} M_{t+1} \leftarrow M_t + \phi(x_{t+1})(z''_t)^{\mathrm{T}}, \\ N_{t+1} \leftarrow N_t + z''_t\phi^{\mathrm{T}}(x_t), \\ q_{t+1} \leftarrow q_t + r_{t+1}z''_t. \end{cases} \tag{43}$$

Therefore, F_{t+1} and a_{t+1} in the matrix form are represented as

$$\begin{cases} F_{t+1} = M_{t+1}N_{t+1}^{-1}, \\ a_{t+1} = N_{t+1}^{-1}q_{t+1}. \end{cases} \tag{44}$$

Let $S_{t+1} = N_{t+1}^{-1}$; according to Eq. (30), we have

$$S_{t+1} \leftarrow S_t - \frac{S_t z''_t\phi^{\mathrm{T}}(x_t)S_t}{1 + \phi^{\mathrm{T}}(x_t)S_t z''_t}. \tag{45}$$

Therefore, F_{t+1} and a_{t+1} can be computed by

$$\begin{cases} F_{t+1} = M_{t+1}S_{t+1}, \\ a_{t+1} = S_{t+1}q_{t+1}. \end{cases} \tag{46}$$

4.7 Weighting the parameter vectors

The ultimate value θ is computed by weighting θ, θ_λ, and $\overline{\theta}$. Let $\overline{b}$ be the weight for θ_λ and $\overline{c}$ be the weight for $\overline{\theta}$. Then θ can be weighted as

$$\theta \leftarrow (1 - \overline{b} - \overline{c}) \times \theta + \overline{b} \times \theta_\lambda + \overline{c} \times \overline{\theta}, \tag{47}$$

where θ_λ and $\bar{\theta}$ are generated from the learning process and the planning process, respectively.

At the beginning of the algorithm, the approximate model is not accurate. Hence, we assign $\bar{b}$ with a relative large value to make θ_λ have a large effect on θ. As the model becomes increasingly accurate, the solution $\bar{\theta}$ will be assigned with a larger value to have more effect on θ. Therefore, in Dyna-LSTD-PA, $\bar{b}$ decreases and $\bar{c}$ increases over time.

4.8 Algorithm specification

Dyna-LSTD-PA includes two procedures, namely, learning and planning, as shown in Algorithms 1 and 2, respectively. The learning procedure (Algorithm 1) directly interacts with the environment; that is, in state x_t, the agent selects the action u according to the Gaussian distribution (see Line 4 in Algorithm 1), and then executes it, followed by the next state x_{t+1} and the immediate reward r_{t+1} (see line 6 in Algorithm 1). Thus, by using the sample $(x_t, u_t, x_{t+1}, u_{t+1})$, the parameter vectors of the value function, the policy, and the model, that are $\theta_\lambda, \beta_{t+1}$, and F_{t+1}, a_{t+1}, are updated via LSTD(λ) (see lines 9, 11, 14, and 15 in Algorithm 1, respectively).

Algorithm 1 Dyna-LSTD-PA algorithm

1: Initialize all the matrices $\leftarrow \ell I$, where ℓ is a small positive number and I is the identity matrix. $\gamma \leftarrow 0.9, \lambda \leftarrow 0.9, \theta \leftarrow \mathbf{0}, \bar{\sigma} \leftarrow 2, t \leftarrow 0$;
2: **repeat** for every episode
3: Initialize $x \leftarrow x_0, \bar{c}, \bar{b}, K, \sigma$;
4: Compute current feature $\phi \leftarrow \phi(x)$ and current action $u \leftarrow N(\phi^{\mathrm{T}}\beta, \bar{\sigma})$;
5: **repeat** for every time step of episode
6: Execute action u, get next state x_{t+1} and immediate reward r_{t+1};
7: Compute next feature $\phi(x_{t+1})$ and next action $u' \leftarrow N(\phi(x_{t+1})^{\mathrm{T}}\beta, \bar{\sigma})$;
8: Update C_{t+1} by Eq. (32) and $\hat{b}_{t+1}$ by Eq. (29);
9: Update the value function parameter: $\theta_\lambda \leftarrow C_{t+1}b_{t+1}$;
10: Update E_{t+1} by Eq. (39) and e_{t+1} by Eq. (37);
11: Update the policy parameter: $\beta_{t+1} \leftarrow E_{t+1}e_{t+1}$;
12: Update M_{t+1}, q_{t+1} by Eq. (43) and S_{t+1} by Eq. (45);
13: Update the model: $F_{t+1} \leftarrow M_{t+1}S_{t+1}$; $a_{t+1} \leftarrow S_{t+1}q_{t+1}$;
14: Planning(K);
15: $\theta \leftarrow (1 - \bar{b} - \bar{c}) \times \theta + \bar{b} \times \theta_\lambda + \bar{c} \times \bar{\theta}$;
16: $x \leftarrow x_{t+1}, u \leftarrow u', \phi \leftarrow \phi(x_{t+1}), \bar{\sigma} \leftarrow 0.95\bar{\sigma}, \bar{b} \leftarrow 0.95\bar{b}, \bar{c} \leftarrow 1.05\bar{c}, t \leftarrow t + 1$;
17: **until** x' is the terminal state
18: **until** All episodes have been iterated

In the meantime, the planning procedure (Algorithm 2) is called (see line 16 in Algorithm 1). The learned model is used to generate the sample like $(\phi_{\bar{t}}, \phi_{\bar{t}+1}, r_{\bar{t}+1})$, which consists of the current feature $\phi_{\bar{t}}$, the next feature (see line 5 in Algorithm 2), and the immediate reward $r_{\bar{t}+1}$ (see line 7 in Algorithm 2). The simulated sample is used to estimate the parameter vector $\bar{\theta}$ of the value function. After the planning procedure is executed K times, $\bar{\theta}$ is fed back to the learning procedure. Then, via weighting θ, θ_λ, and $\bar{\theta}$, the ultimate parameter vector θ of the value function is obtained (see line 15 in Algorithm 1).

Algorithm 2 Planning(K) algorithm

1: Initialize $\bar{z}_0 \leftarrow \vec{0}, \bar{C}_0 \leftarrow \vec{0}, \bar{t} \leftarrow 0$;
2: **repeat** for K times
3: $\phi_0 \leftarrow \phi(x_0)$, $x_0 \sim \rho$;
4: **repeat** for each step of episode
5: Compute the next feature: $\phi_{\bar{t}+1} \leftarrow F\phi_{\bar{t}}$;
6: Update the eligibility trace: $\bar{z}_{\bar{t}+1} \leftarrow \gamma\lambda\bar{z}_{\bar{t}} + \phi_{\bar{t}+1}$;
7: Compute the immediate reward: $r_{\bar{t}+1} \leftarrow \phi^{\mathrm{T}}a$;
8: $\bar{C}_{\bar{t}+1} \leftarrow \bar{C}_{\bar{t}} - \dfrac{\bar{C}_{\bar{t}}\bar{z}_{\bar{t}}(\phi_{\bar{t}} - \gamma\phi_{\bar{t}+1})^{\mathrm{T}}\bar{C}_{\bar{t}}}{1 + (\phi_{\bar{t}} - \gamma\phi_{\bar{t}+1})^{\mathrm{T}}C_{\bar{t}}\bar{z}_{\bar{t}}}$;
9: $\bar{b}_{\bar{t}+1} \leftarrow \bar{z}_{\bar{t}}r_{\bar{t}+1}$;
10: Update the current feature: $\phi \leftarrow \phi_{\bar{t}+1}$;
11: $\bar{t} \leftarrow \bar{t} + 1$;
12: **until** ϕ is the terminal state feature
13: Update the parameter vector of the value function: $\bar{\theta} \leftarrow \overline{Cb}$;
14: $K \leftarrow K - 1$;
15: **until** $K = 0$

In Algorithms 1 and 2, there are some parameters such as the discounted factor γ, decaying factor λ, variance σ, planning times K, and control parameters $\bar{b}$ and $\bar{c}$. γ is used to discount the future rewards, while λ assigns the TD error to the previous visited states via the eligibility trace. σ exists in the Gaussian distribution of the policy, to control the exploring area. K decides how many times the planning procedure will be executed, and a larger K means more use of the learned model. $\bar{b}$ and $\bar{c}$, that satisfy $\bar{b} + \bar{c} \leqslant 1$, are used to assign the weights in the computation of the ultimate parameter vector of the value function. Generally speaking, the learned model is inaccurate in the former phase of the algorithm, and is hence assigned with a small value. With the running of the algorithm, the model becomes increasingly accurate, and $\bar{c}$ requires to be adjusted to a large value. In these parameters, some are initialized by experience (e.g., γ and λ), while others are set according to the empirical results of specific RL problems.

5 Theoretical analysis

Dyna-LSTD-PA uses LSTD(λ) to learn the value function, policy, and model. Under the circumstance of infinite number of samples, the asymptotic convergence of LSTD(λ) in policy evaluation was proved by Nedic et al. [40]. Therefore, if the number of the samples is large enough, the value function

learned by LSTD(λ) will converge. Lazaric et al. [41] derived a high probability error bound for LSTD(0), on the condition that the number of samples is finite. Recently, Tagorti et al. [34] deduced a high probability of the convergence rate for LSTD(λ), in the general case of $\lambda \in (0, 1)$, with the global error consisting of approximation error and estimation error.

However, these works do not learn a model and use it to evaluate the value function, as our work does. In fact, the model is nearly impossible to be accurate, especially in the preceding running of the algorithm. Nevertheless, even if the model is not fully accurate, it can still be very useful in many cases, by generating additional simulated samples to accelerate the convergence. The reason behind it might be that, in most cases, the model is able to cover the important area of the state spaces, even if not in whole.

As a result, other than previous works, we will introduce an additional error caused by the inaccurate model, and we call this error as model error. In the following part of this section, we will try to bound the model error $\|V_{\mathrm{LSTD}(\lambda)} - \hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}}\|_\rho$, where $V_{\mathrm{LSTD}(\lambda)}$ is the value function estimated through $V_{\mathrm{LSTD}(\lambda)} = \Phi(A - \gamma B)^{-1}b$, and A, B, and b are accumulated from infinite number of samples generated from the interaction with the environment. $\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}}$ is the value function estimated by using model, and it is defined as $\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}} = \Phi(\bar{A} - \gamma\bar{B})^{-1}\bar{b}$, where $\bar{A}$, $\bar{B}$, and $\bar{b}$ are accumulated from the simulated samples.

Assumption 1 Assume the Markov chain M considered here is ergodic.

In Assumption 1, ergodicity holds if and only if M is aperiodic and irreducible, indicating if and only if $\forall(x, y) \in X^2, \exists n_0, \forall n \geqslant n_0, s.t., P^n(x, y) > 0$, where P is the transition kernel. Such an assumption ensures a unique stationary distribution ρ.

Assumption 2 The feature functions $\phi_1, \phi_2, \ldots, \phi_d$ are linearly independent.

Assumption 2 is used to guarantee that Eq. (12), $A - \gamma B$, is invertible, and thus $(A - \gamma B)\theta = b$ can be solved (Nedic and Bertsekas, 2002). Since $\theta = (A - \gamma B)^{-1}b$ is obtained, we can compute $V_{LSTD(\lambda)} = \Phi(A - \gamma B)^{-1}b$.

Assumption 3 The reward function r and the feature functions $\phi_1, \phi_2, \ldots, \phi_d$ are bounded.

Let $B(X, K)$ be the set of measurable functions defined on X and bounded by K. We define the reward function $r \in B(X, R_{\max})$ with $R_{\max} \in \mathbb{R}$, and the feature function $\phi_i \in B(X, L)$, where L is the bound for ϕ_i. Since the reward function is bounded, the value function is also a bounded function $V \in B(X, V_{\max})$ with $V_{\max} = \dfrac{R_{\max}}{1 - \gamma}$.

Theorem 1 Let Assumptions 1–3 hold and let $x_0 \sim \rho$. Let v be the smallest eigenvalue of the Gram matrix $G \in \mathbb{R}^{d\times d}, G_{ij} = \int \phi_i{}^{\mathrm{T}}(x)\phi_j(x)\rho dx$. Let n be the number of the samples and d be the dimensionality of the feature function ϕ_i. Then, the model error of Dyna-LSTD-PA can be defined as

$$\|V_{\mathrm{LSTD}(\lambda)} - \hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}}\|_\rho \leqslant \frac{4\sqrt{d}LV_{\max} + (R_{\max} + 1)\sqrt{d}L}{\sqrt{v}(1-\gamma)(n-1)(1-\gamma\lambda)}. \tag{48}$$

The proof of Theorem 1 can be found in Appendix A.

As can be noticed in Theorem 1, when the number of samples, n, goes to infinity, the model error will be 0. Moreover, we find the relationships between the model error and the parameters, that are, the model error increases with the range of the feature function L, the dimensionality of the feature function d, the bound of the reward function $R_{\max}$, and the bound of the value function $V_{\max}$. However, it decreases with the number of samples n and the smallest eigenvalue of the Gram matrix v. Therefore, we can assign their values along the direction of minimizing the model error.

Theorem 2 With probability$1 - \delta$, $\delta \in (0, 1)$, the global error between the true value V and the solution $\hat{V}_{LSTD(\lambda)}$ obtained from Dyna-LSTD-PA satisfies

$$\begin{aligned}&\|V - \hat{V}_{LSTD(\lambda)}\|_\rho \leqslant \\ &\frac{1-\gamma\lambda}{1-\gamma}\|V - \Pi V\|_\rho + \frac{4\sqrt{d}LV_{\max} + (R_{\max} + 1)\sqrt{d}L}{\sqrt{v}(1-\gamma)(n-1)(1-\gamma\lambda)} + \\ &\frac{4V_{\max}dL^2}{\sqrt{n-1}(1-\gamma)v}\sqrt{\left(1 + \left\lceil \frac{\log(n-1)}{\log(\frac{1}{\gamma\lambda})} \right\rceil\right) I(n-1, \delta) + h(n, \delta)}.\end{aligned} \tag{49}$$

The proof of Theorem 2 can be found in Appendix B.

From Theorem 2, we can see that, like the model error, the global error also increases when d and $R_{\max}$ become larger, and decreases when n becomes larger.

6 Empirical results

This section reports the empirical results on four RL benchmarks with continuous state and action spaces: the pole-balancing problem [45], the Dyna maze problem [46], the inverted pendulum swing-up problem [47], and the cleaning

robot problem [18]. To reduce possible oscillations, we executed all algorithms in the coming comparisons 20 times, and used their average values as the final outputs, respectively. We used radial basis functions (RBFs) to construct the nonlinear features for states and actions, and the parameters of the RBFs such as the center points and the variances are selected manually. That is to say, given a group of selective values of the parameters, the final determination for the values of the parameters depend on the comparisons of the corresponding experimental results. Besides using manual features, the features can be automatically selected [48] from a set of features by using regularization (L_2 regularization [49,50] and L_1 regularization [51–53]), matching pursuit (orthogonal pursuit [54] and order recursive matching pursuit [55]), random project [56, 57], kernel sparsity [58], etc. As automatic feature selection is beyond our main concern, we will not specify them. Dyna-LSTD-PA will be compared with Dyna-LFA-PS [22] and classic actor-critic learning algorithm (CACLA) ([30]). Dyna-LSTD-PA is a Dyna variant for discrete action space, and it approximates the value function and the model via a linear function approximator. The parameter vectors of the value function and the model are learned by TD. CACLA can handle the problems with continuous state and action spaces. Its policy is linearly approximated and the parameter vector is learned by TD.

6.1 Pole-balancing problem

The pole-balancing problem (Fig. 3) requires balancing a pole of unknown length and mass at the upright position by applying forces to the cart it is attached to. The state consists of the vertical angle χ and the angular velocity $\dot{\chi}$ of the pole, denoted as $(\chi, \dot{\chi})$. The action, that is the force exerting on the cart, ranges from –50 N to 50 N. Negative action means a force to the left, and positive action means a force to the right. The transition is governed by the nonlinear dynamics of the system formalized as follows:

$$\ddot{\chi} = \frac{g\sin(\chi) - \dfrac{\eta\bar{m}l(\dot{\chi})^2\sin(2\chi)}{2} - \eta\cos(\chi)u}{\dfrac{4l}{3} - \eta\bar{m}l\cos^2(\chi)}, \qquad (50)$$

where $g = 9.8\bar{m}/s^2$ is the gravity constant, $\bar{m}$ is the mass of the pole ($\bar{m} = 2.0$kg), p is the mass of the cart ($p = 8.0$kg), l is the length of the pole ($l = 0.5$m), and $\eta = 1/(\bar{m} + p)$ is a constant. The time step of system simulation is set to 0.1 s. If the intersection angle between the pole and the horizontal line does not exceed $\pm\pi/4$, the immediate reward will be 1. Otherwise, a reward of -1 will be given. If the angle is greater than $\pi/4$ or smaller than $-\pi/4$, the pole will fall down and the episode will end.

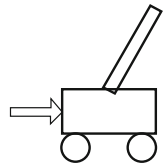


Fig. 3 Illustration of the pole-balancing problem

The state space is $X = \{(\chi, \dot{\chi}) | \chi \in [-\pi/4, \pi/4], \dot{\chi} \in [-2, 2]\}$, and the action space is $U = \{u | u \in [-50, 50]\}$. For any state $(\chi, \dot{\chi})$, the basis functions are

$$[e^{-\frac{\|x-\mu_1\|^2}{2\bar{\sigma}^2}}, e^{-\frac{\|x-\mu_2\|^2}{2\bar{\sigma}^2}}, \cdots, e^{-\frac{\|x-\mu_{25}\|^2}{2\bar{\sigma}^2}}]^{\mathrm{T}}, \qquad (51)$$

where μ_i is the ith center point, and it locates over the grid points $\{-\pi/4, -\pi/8, 0, \pi/8, \pi/4\} \times \{-2, -1, 0, 1, 2\}$; thus, we can obtain 25 basis functions. $\bar{\sigma}$ is the variance with value 2. The number of episodes is set to 1000. Each episode ends when the pole has balanced for 3,000 time steps or the pole has fallen down. The pole is reset to vertical, that is, the initial state $(0, 0)$, after each failure.

We now discuss Dyna-LSTD-PA's results on this problem. Episodes are launched as many as possible until convergence is reached. The optimal policy and the value function learned by Dyna-LSTD-PA are shown in Figs. 4(a) and 4(b). When the angle approaches $\pi/4$ and the angular velocity approaches 2, the policy takes almost the largest force 48.12 N (see green five-pointed star in Fig. 4(a)) and the corresponding value is 9.97 (see green five-pointed star in Fig. 4(b)). This implies that a maximal force will be necessary to make the pole balance when the pole is in an extreme state $(\pi/4, 2)$. It is similar when the pole is in an opposite state $(-\pi/4, -2)$, in which case the force to keep balance is -48.189 N (see yellow triangle in Fig. 4(a)) and the value is 9.869 (see yellow triangle in Fig. 4(b)). When the pole is nearly upright and its angular velocity is much smaller than 2, the policies in these states are to take a mild force to keep the pole balance, and the values are evidently large. For example, when the state of the pole is $(-0.03, 0.08)$, the policy is -0.06 N (see blue square in Fig. 4(a)) and the value is 10.003 (see blue square in Fig. 4(b)). Therefore, we can get a conclusion: the closer the distance between the state and the center point (0, 0), the larger the value. Hence, a smaller force is needed, and the pole is more likely to reach a balance.

To determine the values of the parameters such as $\bar{b}$, $\bar{c}$, and K, we prepare some settings and then compare the experimental performance. The experimental results of different

settings of $\bar{b}$ and $\bar{c}$ are shown in Fig. 5(a). If $\bar{b}$=0.5 and $\bar{c}$=0.3, Dyna-LSTD-PA has the best convergence performance, and it converges after 86 episodes. If $\bar{b}$=0.4 and $\bar{c}$=0.4, Dyna-LSTD-PA converges after 119 episodes. If $\bar{b}$=0.6 and $\bar{c}$=0.2, it learns fast in the former phase but converges until the 129th episode. The worst case is $\bar{b}$=0.3 and $\bar{c}$=0.5, where Dyna-LSTD-PA converges until the 167th episode. Therefore, the settings $\bar{b}$ =0.5 and $\bar{c}$ =0.3 can gain the fast convergence rate. This result might be caused by the fact that the model is learned asymptotically. In the preceding phase of the algorithm, the model is not accurate enough and the simulated samples are not reliable. Thus, $\bar{c}$ has to be assigned with a relatively small value, while $\bar{b}$ is a large value. We also can notice that the settings of $\bar{b}$=0.6 and $\bar{c}$=0.2 are no better than $\bar{b}$=0.5 and $\bar{c}$=0.3. This reveals that $\bar{c}$ should be assigned with a small value, but not too small, to realize a better performance of the algorithm.

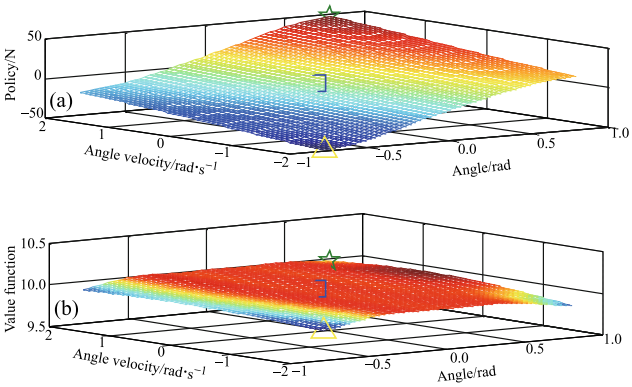


Fig. 4 Approximate policy and value function. (a) Approximate policy; (b) approximate value function

We implement an experiment to make clear the effect of the planning number K on the convergence efficiency. The experimental results are shown in Fig. 5(b). There are three alternative settings of K, namely 20, 50, and 80. K = 80 converges in the 83rd episode, while K = 50 converges until the 86th episode. When K is set to 20, Dyna-LSTD-PA converges after 126 episodes. The best convergence performance is obtained when K = 80. K = 80 behaves much better than K = 20 but only has a slight priority over K = 50. However, K = 80 means that the planning process has to be performed 80 times at each time step. Thus, it needs much more computational cost than K = 50, with only a small improvement on the convergence rate. Therefore, we still set K to 50 in our experiment.

To make the comparisons of Dyna-LSTD-PA with CACLA and Dyna-LFA-PS in terms of convergence performance, the three algorithms are implemented and the experimental

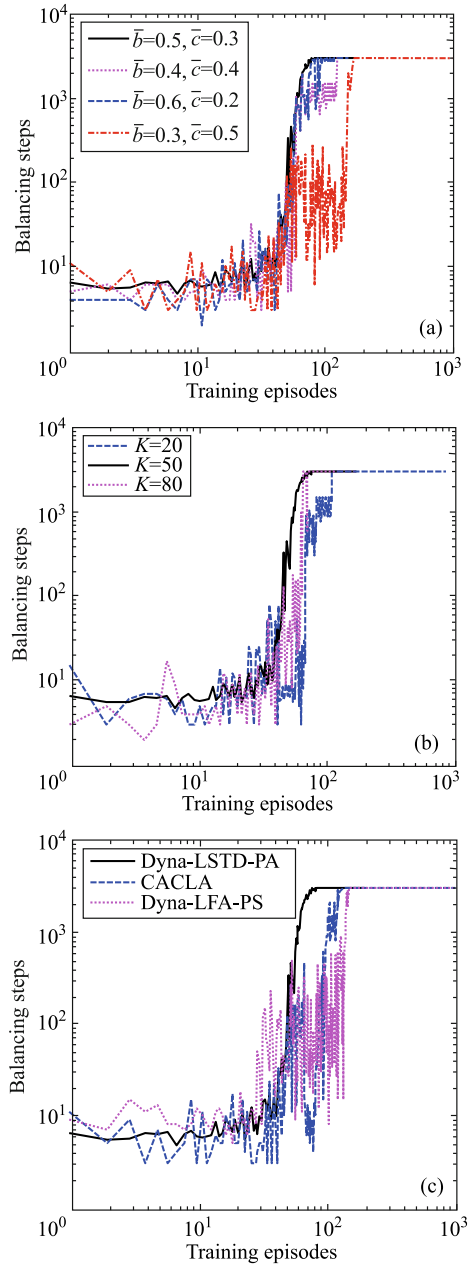


Fig. 5 Convergence rate comparisons for different parameter settings and algorithms. (a) Convergence rate comparisons of different weights; (b) convergence rate comparisons of different planning times; (c) convergence rate comparisons of different algorithms

results are shown in Fig. 5(c). The decaying factor λ and the discounted factor γ are set as 0.9 in the three algorithms. In CACLA and Dyna-LFA-PS, the learning rates are set to 0.8 and the parameter vector is set to 0. Dyna-LFA-PS cannot be directly applied to the continuous action spaces, and hence its continuous actions are discretized into –50, 0, and 50. From Fig. 5(c), we can see that Dyna-LSTD-PA converges after 86 episodes. CACLA converges in the 150th episode, and Dyna-LFA-PS converges in the 175th episode. Evidently, CACLA performs better than Dyna-LFA-PS, while Dyna-LSTD-PA

has the best convergence performance. In CACLA and Dyna-LFA-PS, TD is used to learn the value function, and the samples are dropped after they have been used once. Therefore, if the new coming sample is considerably different from the former samples, the value function may oscillate. That is why the performances of CACLA and Dyna-LFA-PS oscillate sharply in the former 110 episodes. LSTD(λ) is used to learn the value function in Dyna-LSTD-PA and the samples are stored by matrices. Therefore, even if a very different sample is arrived, the matrices will not be affected considerably. Thus, Dyna-LSTD-PA behaves in a more stable manner than CACLA and Dyna-LFA-PS. However, like Dyna-LSTD-PA, Dyna-LFA-PS also uses the learned model to generate additional samples to update the value function, and it still fails to converge faster than CACLA. This result might be caused by the fact that Dyna-LFA-PS plans too much when the model is inaccurate, resulting in poorer convergence performance.

The different settings of the action-noise range and the variance σ (see Eq. (17)) may have a significant effect on the experimental results. Therefore, we prepare the settings for the action-noise range as [0 N, 0 N], [–15 N, 15 N], and [–30 N, 30 N], and the settings for the variance are as 2, 4, and 6. The maximal episode is set to 1,000. The three algorithms take the exploration policy based on the Gaussian distribution here. They are implemented and compared, and their results are shown in Table 1. The success rate is defined as the ratio of the average time steps divided by 3,000.

We can see that Dyna-LSTD-PA outperforms the two other methods in terms of the success rate under different conditions of the action-noise range and variance. Among the three methods, the best performances are obtained when $\sigma = 2$ under the different settings of the action-noise range. The larger value of variance means the agent has to explore more, and hence the variance has to be set to a proper value to balance between exploration and exploitation. The best performance is achieved when $\sigma = 2$. It implies that $\sigma = 2$ is better than $\sigma = 4$ and $\sigma = 6$ in trading off between exploration and exploitation. Generally speaking, the larger the range of the action-noise, the larger is the uncertainty. Thus, action-noise with a smaller range will result in a higher success rate. However, it is not always true. For example, when $\sigma = 4$, the success rate of [–30N,30N] is better than that of [0N,0N]. Similar circumstances can also be found in CACLA and Dyna-LFA-PS. Therefore, according to the experimental result, we cannot find a close relationship between the action-noise range and the success rate.

6.2 Dyna maze problem

The environment of the Dyna maze problem with discrete state and action spaces is shown in Fig. 6(a), while the one with continuous state and action spaces is shown in Fig. 6(b). The goal of the agent is to move from the start point "Start" to the goal point "Goal". The gray grids are obstacles and the deep gray grids are walls, and the agent cannot land on them. When the point "Goal" is reached, the agent will receive an immediate reward of 1. In the other cases, the agent will receive an immediate reward of –1. The state is the position of the agent represented by (x, y). In (x, y), x is the horizontal coordinate, and y is the vertical coordinate. At each time step, the agent moves a constant distance of 0.2. The state space is $X = \{(x, y)|x, y \in [0, 8]\}$ and the action space is $U = \{u|u \in [-\pi, \pi]\}$. Every state corresponds to a feature vector with the dimension 36, and is shown as

$$(e^{-\frac{\|x-\mu_1\|^2}{2\bar{\sigma}^2}}, e^{-\frac{\|x-\mu_2\|^2}{2\bar{\sigma}^2}}, \ldots, e^{-\frac{\|x-\mu_{36}\|^2}{2\bar{\sigma}^2}})^{\mathrm{T}}, \tag{52}$$

where $\mu_i = (\mu_{ix}, \mu_{iy})$ is a center point and it locates over the grid points $\{0.5, 2, 3.5, 5, 6.5, 8\} \times \{0.5, 2, 3.5, 5, 6.5, 8\}$, and $\bar{\sigma}$ is the variance with the value of 2.

Table 1 Comparisons of success rate

Action noise	Variance	Dyna-LSTD-PA		Dyna-LFA-PS		CACLA	
		Average step	Success rate	Average step	Success rate	Average step	Success rate
[0N,0N]	2	2,987	0.996	2,567	0.856	2,678	0.893
[0N,0N]	4	2,874	0.958	2,298	0.766	2,543	0.848
[0N,0N]	6	2,895	0.965	2,434	0.811	2,023	0.674
[–15N,15N]	2	2,945	0.982	2,456	0.819	2,376	0.792
[–15N,15N]	4	2,943	0.981	2,023	0.674	2,237	0.746
[–15N,15N]	6	2,893	0.964	2,045	0.682	1,962	0.654
[–30N,30N]	2	2,937	0.979	2,663	0.888	2,378	0.793
[–30N,30N]	4	2,876	0.959	2,456	0.819	1,763	0.588
[–30N,30N]	6	2,875	0.958	2,145	0.715	2,123	0.708

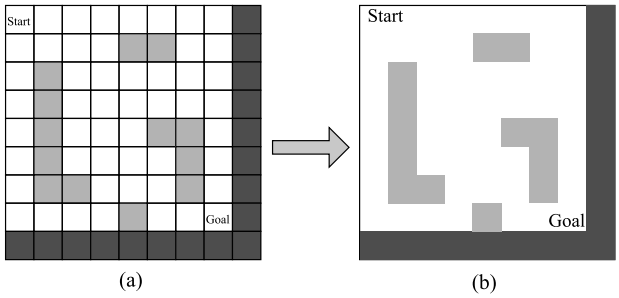


Fig. 6 Illustration of the Dyna maze problem. (a) Discrete state and action spaces; (b) continuous state and action spaces

This problem is an episodic task and the maximal episode is set to 200. An episode ends when the agent has reached the goal or the time steps have exceeded 10,000. Afterward, the initial state of the agent is set to (0.5, 0.5). The optimal policy and the value function learned by Dyna-LSTD-PA are shown in Figs. 7(a) and 7(b), respectively. The obstacle-areas or areas far from the goal area, have small values. The goal state (8, 8) has the largest value 1.127, with its policy being 0.181.

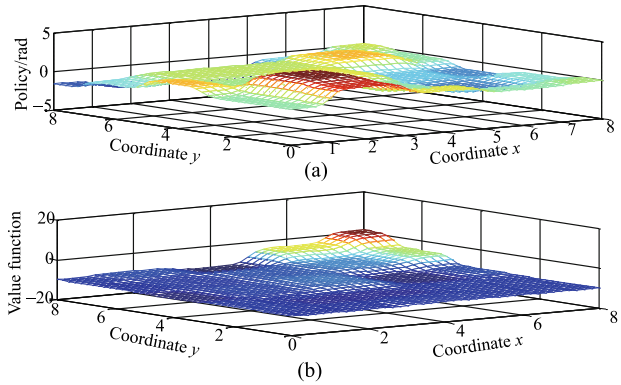


Fig. 7 Optimal value function and policy. (a) Approxiamte policy; (b) approxiamte value function

The area near the goal state has a relative high value. For example, most of the values in the area that satisfies $7 \leqslant x \leqslant 8$ and $7 \leqslant y \leqslant 8$ are larger than 0, and the policies are near 0. This reveals that the agent only has to move a small angle to reach the goal. For the obstacle areas where $4 \leqslant x \leqslant 6$ and $1 \leqslant y \leqslant 2$, the values are near –10 and the policy is near –1. This indicates that the agent has to move a large angle to leave the current state. As for the obstacle area where $5 \leqslant x \leqslant 7$ and $4 \leqslant y \leqslant 5$, the values are near –7 and policies are near 0.7. Unlike the other obstacle areas, they have a comparatively large value, and the agent only has to move with a small angle. Dyna-LSTD-PA is compared with CACLA and Dyna-LFA-PS in terms of convergence rate and sample efficiency, and the experimental result of convergence rate is shown in Fig. 8(a). The parameter settings for the three algorithms are the same as the settings in the former experiment. From Fig. 8(a), we find that Dyna-LSTD-PA converges after 40 episodes. It outperforms the two other methods not only in terms of convergence rate, but also in terms of time steps to the goal. Dyna-LFA-PS discretizes the continuous actions to 0, 1, 2, and 3, and it converges to 81 time steps in the 75th episode. CACLA converges to 85 time steps in the 88th episode. Thus, Dyna-LSTD-PA converges faster than CACLA and Dyna-LFA-PS. More importantly, the curve of Dyna-LSTD-PA fluctuates less and converges more smoothly, indicating that it performs in a more stable manner. As in the pole-balancing problem, Dyna-LFA-PS behaves slightly poorer than CACLA, and this may be caused by the unreasonable discretization of the continuous action space. Dyna-LSTD-PA uses the policy by linear approximation to avoid unreasonable discretization of continuous action spaces. Moreover, it uses the model to plan, leading to the best convergence performance.

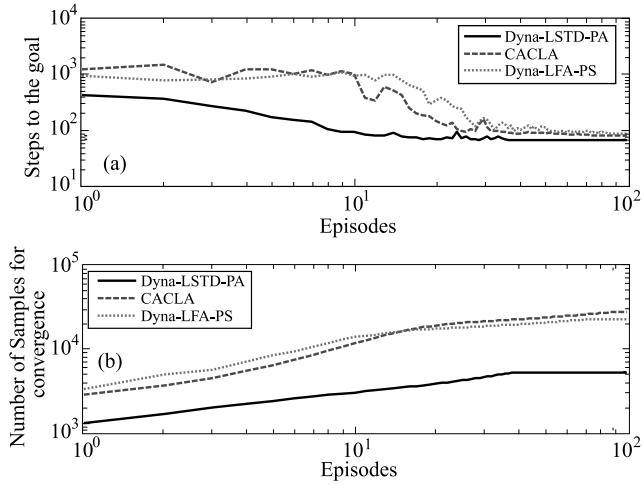


Fig. 8 Comparisons of (a) convergence performance and (b) sample efficiency

The experimental results of sample efficiency comparisons are shown in Fig. 8(b). We can see that Dyna-LSTD-PA requires 4,891 samples to converge, while Dyna-LFA-PS requires 21,582, and CACLA requires 25,984. Thus, Dyna-LSTD-PA significantly outperforms the other two algorithms. Dyna-LSTD-PA performs best in all iterations of the episodes. The reason behind it is that Dyna-LSTD-PA not only uses the model to generate the additional samples but also uses LSTD(λ) to learn the value function. Before the 23th episode, CACLA performs better than Dyna-LFA-PS in that the model is not accurate enough. However, with the processing of learning, Dyna-LFA-PS can generate more useful

samples to accelerate the convergence. Thus, Dyna-LFA-PS requires less samples than CACLA empirically.

Stability plays a particularly significant role in some real-world applications. To emphasize the stabilities of the algorithms, the results of multiple runs are illustrated in Fig. 9. The vertical lines show the maximal step differences among 20 runs of every time step, and the line passing through them indicates the average value of maximal and minimal values. Therefore, the length of the vertical line shows the stability of the algorithm, that is, a shorter line means the algorithm is more stable. The stability performances of Dyna-LSTD-PA, CACLA, and Dyna-LFA-PS are shown in Fig. 9(a)–9(c).

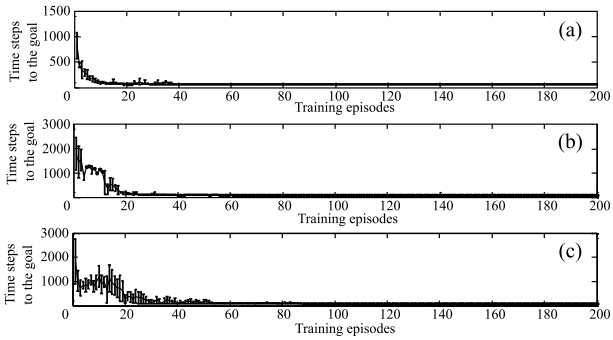


Fig. 9 Stability comparisons. (a) Maximal step difference among 20 runs for Dyna-LSTD-PA; (b) maximal step difference among 20 runs for CACLA; (c) maximal step difference among 20 runs for Dyna-LFA-PS

We have three observations from these figures. (1) Compared with the few sharp oscillations of CACLA in Fig. 9(b) and Dyna-LFA-PS in Fig. 9(c), the area covered by the lines of Dyna-LSTD-PA in Fig. 9(a) is narrower and smoother. (2) Dyna-LFA-PS has the poorest performance of stability in the former 53 episodes. (3) Dyna-LSTD-PA converges in 39 iterations of episodes in all 20 runs, while CACLA and Dyna-LFA-PS still have some oscillations until the 75th episode and 88th episode respectively. Thus, our results show that compared with the two other methods, Dyna-LSTD-PA is more stable.

6.3 Inverted pendulum swing-up

The inverted pendulum swing-up problem (shown in Fig. 10) is a challenging and highly nonlinear control problem commonly found in RL literature. We compare our algorithm with CACLA and Dyna-LFA-PS on this task. The objective is to learn how to swing up an inverted pendulum from the pointing-down position to the upright position immediately and then stabilize it in this position. The formula of motion of the system is defined as

$$J\ddot{a} = Mgl\sin(a) - (b + \frac{K^2}{R})\dot{a} + \frac{K}{R}u, \tag{53}$$

where a is the angle between the pendulum and the upright direction, J is the pendulum inertia with the value $1.91 \cdot 10^{-4}$ kgm^2, M is the pendulum mass whose value is set to $5.50 \cdot 10^{-2}$ kg, g is the gravity with the value 9.81 m/s^2, l represents the pendulum length and its value is given as $4.20 \cdot 10^{-2}$ m, b denotes the damping with the value $3 \cdot 10^{-6}$ Nms, K is the torque constant whose value is assigned with $5.36 \cdot 10^{-2}$ Nm/A, and R is the rotor resistance with its value of 9.50 Ω.

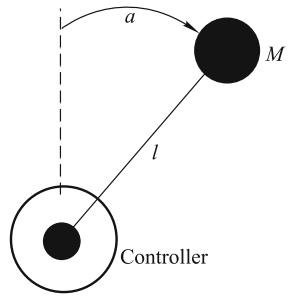


Fig. 10 Inverted pendulum setup

The state $x^{\mathrm{T}} = [a, \dot{a}]$ consists of the angle a and the angular velocity $\dot{a}$ of the pendulum with the range of $|a| \leqslant \pi$ rad and $|\dot{a}| \leqslant 8\pi$ rad $\cdot$s^{-1}. The action u is an actuation signal whose value satisfies $|u| \leqslant 3$ V.

Applying only the actuation signal without the momentum cannot push the pendulum to the upright position. The controller has to swing the pendulum back and forth so as to increase the momentum of the pendulum. The quadratic reward function is defined as

$$r_k(x_{k-1}, u_{k-1}) = -x_{k-1}^{\mathrm{T}} Q x_{k-1} - P u_{k-1}^2, \tag{54}$$

with

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad P = 1.$$

The state of the pendulum is initialized to the upside-down position $x_0^{\mathrm{T}} = [\pi, 0]$ at the start of each episode. To transform the state to the feature, the center points of the RBFs for both states and actions need to be initialized. The center points for every state feature locate over $\{-3, -2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5, 2, 2.5, 3\} \times \{-25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25\}$, with the variances $\sigma_a = 0.24$ and $\sigma_{\dot{a}} = 2.79$. The center points for each action distribute over $\{-2.8, -2.1, -1.4, -0.7, 0, 0.7, 1.4, 2.1, 2.8\}$ with its variance $\sigma_u = 0.33$. The algorithm runs for

54 min in the whole simulation. Every time step lasts for 0.005 s. An episode contains 800 time steps so that it will take 4 s.

We compare our algorithm with CACLA and Dyna-LFA-PS in terms of cumulative discounted rewards and sample efficiency. The action space is discretized into –3, 0, and 3 in Dyna-LFA-PS. The planning times are set to 10 for both Dyna-LFA-PS and Dyna-LSTD-PA. The other parameter settings are the same as in the former experiments. The cumulative discounted rewards for the three algorithms are shown in Fig. 11(a). Clearly, Dyna-LSTD-PA converges fastest during the whole training, with the limit of the cumulative discounted rewards about –320. In the former 11 min, CACLA converges faster than Dyna-LFA-PS, but Dyna-LFA-PS outperforms instead from the 11th minute to the 19th minute. Dyna-LSTD-PA and CACLA converge to the same solution nearly –320, but Dyna-LSTD-PA converges to the solution about –2,300 after 34 min pass. The fast learning of Dyna-LFA-PS may be attributed to its modeling learning and planning. Dyna-LFA-PS also learns and uses the model so that it behaves better than CACLA in the initial phase of the training. As the value function is more and more accurate, the effect of the model learning and planning will be not so evident. At this moment, Dyna-LFA-PS does not have an advantage over CACLA due to its unreasonable discretization of the action space.

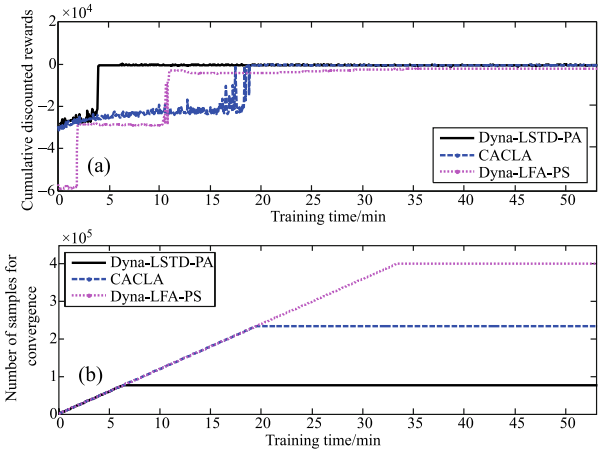


Fig. 11 (a) Cumulative discounted rewards and (b) sample efficiencies of different algorithms

The comparisons of sample efficiencies for different algorithms are shown in Fig. 11(b). The number of required samples to converge for Dyna-LSTD-PA, CACLA, and Dyna-LFA-PS are 76,000, 233,600, and 400,000, respectively. It is evident that our algorithm requires the fewest samples while achieving the fastest convergence rate. This may be caused by the fact that Dyna-LSTD-PA approximates a model and then uses it for planning to accelerate the learning of the value function and the policy. However, Dyna-LFA-PS also uses the model but with a poorer sample efficiency, which may be resulting from its unreasonable manual discretization. Other than manual discretization, Dyna-LSTD-PA directly approximates the policy and adopts LSTD(λ) to learn the value function, the policy, and the model. As a result, Dyna-LSTD-PA outperforms Dyna-LFA-PS in terms of sample efficiency.

6.4 Cleaning robot

The cleaning robot problem is depicted in Fig. 12, and is the extension of the discrete case specified in [18]. A cleaning robot has to collect used cans and then go to charge its batteries. The state $x = (x_h, x_v)$ satisfies $0 \leqslant x_h, x_v \leqslant 1$, where x_h represents the position in the horizontal direction and x_v is the position in the vertical direction. The actions the agent can take are charging, moving, and collecting. Charging and collecting are discrete actions. Moving is a continuous action with the range $[-\pi, \pi]$. The goal of the robot is to collect all the cans as quickly as possible. The state transition function is defined as

$$f(x,u) = \begin{cases} x + 0.1\cos(u), & \text{if } u = \textit{moving}; \\ x, & \text{otherwise.} \end{cases} \tag{55}$$

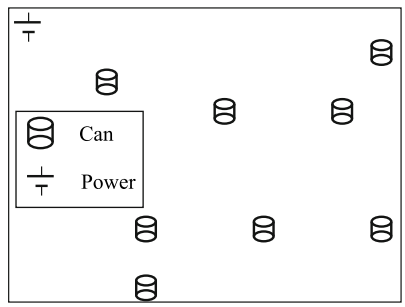


Fig. 12 Illustration of the cleaning robot problem

The agent will be fed back with a reward of -1 for the action moving. When the robot reaches one of the positions with a can, it collects the can while receiving a reward of 5. The consequence is that the can will disappear after it is collected. The robot can go to the recharging station to charge. If the agent is charged for the first time, it will be given a reward of 1; otherwise, –1. The corresponding reward function is defined as

$$r(x,u) = \begin{cases} 5, & \text{if } x = \textit{can} \text{ and } u = \textit{collecting}; \\ 1, & \text{if } x = \textit{power} \text{ and } u = \textit{charging (first time)}; \\ -1, & \text{otherwise.} \end{cases} \tag{56}$$

We will evaluate this experiment in two settings: fixed case and random case. In the former, the initial position of the agent and the cans are fixed, while in the latter, the positions are randomly distributed. The fixed case is shown in Fig. 12. There are eight cans and one recharging station. The center points of the RBFs for the state locate over $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\} \times \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, with the variances $\sigma_h = 0.05$ and $\sigma_v = 0.05$. The center points of the RBFs for the action moving distribute over $\{-3, -2.4, -1.8, -1.2, -0.6, 0, 0.6, 1.2, 1.8, 2.4, 3\}$, with the variance $\sigma_u = 0.31$. All the other parameter settings are the same as those of the Dyna maze. The cumulative discounted rewards for the fixed case and the random case are shown in Figs. 13(a) and 13(b), respectively. Evidently, Dyna-LSTD-PA learns fastest in both cases, and it converges to the solutions 25 and 29 at the 28th episode and the 22nd episode respectively. Dyna-LFA-PS performs slightly better than CACLA in the former 36 episodes for the fixed case. CACLA converges to –4 at the 136th episode while Dyna-LFA-PS converges to –23 at the 75th episodes. As for the random case, Dyna-LSTD-PA, CACLA, and Dyna-LFA-PS converge to 10, –114, –165 at the 22nd, 59th, and 83rd episodes, respectively. It is clear that CACLA and Dyna-LFA-PS cannot find a solution as good as that of the fixed case. Comparatively, the obtained solution of CACLA seems slightly better than that of Dyna-LFA-PS in both cases. This phenomenon further demonstrates that policy approximation can significantly improve the quality of the solution.

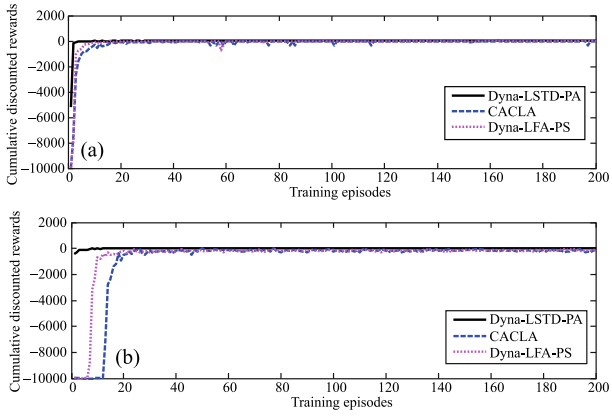


Fig. 13 Comparisons of cumulative discount rewards for (a) fixed and (b) random initial position and barriers

The comparisons of the sample efficiencies for the three algorithms in both cases are also presented in Figs. 14(a) and 14(b). For the fixed case, Dyna-LSTD-PA requires 6,899 samples to converge, while CACLA and Dyna-LFA-PS require 41,762 and 28,018 samples to converge. The number of required samples for the three algorithms in the random case are 2,809, 14,965, and 94,783, respectively. We find that Dyna-LSTD-PA has the best sample efficiency due to its fastest convergence rate; additionally, it has an advantage over the others especially in the random case. Though CACLA converges faster than Dyna-LFA-PS in the random case, its sample efficiency is lower in both cases. Since both CACLA and Dyna-LFA-PS cannot find good solutions, their sample efficiencies cannot reflect the number of required samples to converge in reality.

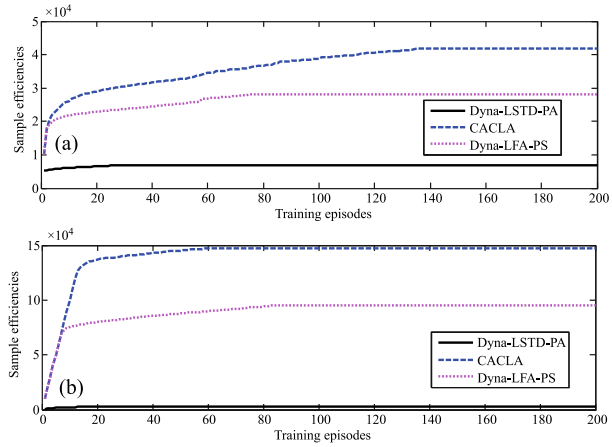


Fig. 14 Comparisons of sample efficiencies for (a) fixed and (b) random initial position and barriers

7 Conclusion

This paper proposes an improved Dyna algorithm based on LSTD and policy approximation, named Dyna-LSTD-PA, to handle RL problems with continuous state and action spaces. The model, policy, and value function in our method are linearly approximated. Their parameter vectors are learned by an improved LSTD(λ) to improve the sample efficiency to a large extent. By using the Dyna structure, the value function is learned in not only the learning process but also the planning process.

In comparison to Dyna-LFA-PS, Dyna-LSTD-PA represents the policy by linear approximation to be applied in the continuous action space. Moreover, Dyna-LSTD-PA does not have to set the learning rate or the initial value of the parameter vector. Distinguishing from CACLA, Dyna-LSTD-PA uses the Dyna structure and learns an approximate model, which is then used to evaluate the value function. Thus, Dyna-LSTD-PA appears to be faster than CACLA in the learning process and has better convergence performance on

our tested RL problems.

The model learned in this study is just a sample-average model, and hence it is not accurate especially in the initial iterations of the episodes. Such a model will probably slow down our algorithm's convergence rate. In the future, we would like to explore methods of learning a more accurate model without adding more computational cost.

Acknowledgements This paper was partially supported by Innovation Center of Novel Software Technology and Industrialization, the National Natural Science Foundation of China (Grant Nos. 61772355, 61702055, 61303108, 61373094, 61472262, 61502323, 61502329), Natural Science Foundation of Jiangsu (BK2012616), Provincial Natural Science Foundation of Jiangsu (BK20151260), High School Natural Foundation of Jiangsu (13KJB520020, 16KJD520001), Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University (93K172014K04, 93K172017K18), and Suzhou Industrial Application of Basic Research Program Part (SYG201422).

Appendixes

Appendix A Proof of Theorem 1

By Assumption 1, $(A-\gamma B)$ is invertible. Then starting from the definitions of $\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}}$ and $V_{\mathrm{LSTD}(\lambda)}$, we have

$$\begin{aligned}\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}} - V_{\mathrm{LSTD}(\lambda)} &= \Phi\bar{\theta} - \Phi\theta \\ &= \Phi(A-\gamma B)^{-1}[(A-\gamma B)\bar{\theta} - b].\end{aligned}$$

Since $(\bar{A}-\gamma\bar{B})\bar{\theta} = \bar{b}$, $(A-\gamma B)\bar{\theta} - b$ can be derived as

$$\begin{aligned}(A-\gamma B)\bar{\theta} - b &= (A-\gamma B)\bar{\theta} - b - (\bar{A}-\gamma\bar{B})\bar{\theta} + \bar{b} \\ &= [A-\gamma B-(\bar{A}-\gamma\bar{B})]\bar{\theta} - (b-\bar{b}) \\ &= \varepsilon_1\bar{\theta} - \varepsilon_2,\end{aligned}$$

where $A-\gamma B-(\bar{A}-\gamma\bar{B}) = \varepsilon_1$ and $b-\bar{b} = \varepsilon_2$. Thus,

$$\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}} - V_{\mathrm{LSTD}(\lambda)} = \Phi(A-\gamma B)^{-1}(\varepsilon_1\bar{\theta} - \varepsilon_2). \quad (57)$$

Before bounding $\hat{V}_{\mathrm{LSTD}(\lambda)\text{–M}} - V_{\mathrm{LSTD}(\lambda)}$, we need to bound $\Phi(A-\gamma B)^{-1}$ and $\varepsilon_1\bar{\theta} - \varepsilon_2$, as stated in Lemmas 1 and 2, respectively.

Lemma 1 Let $\mathrm{Z} = (1-\lambda)\gamma P(1-\lambda\gamma P)^{-1}$ and $\mathrm{Z}_\rho = \Phi^{\mathrm{T}}D_\rho\Phi$; then we have

$$\Phi(A-\gamma B)^{-1} = (I-\Pi\mathrm{Z})^{-1}\Phi\mathrm{Z}_\rho^{-1}.$$

Proof From the definition of $A-\gamma B = \Phi^{\mathrm{T}}D_\rho(I-\gamma P)(I-\gamma\lambda P)^{-1}\Phi$, we can see that

$$\begin{aligned}&\Phi(A-\gamma B)^{-1} \\ &\quad= \Phi[\Phi^{\mathrm{T}}D_\rho(I-\gamma P)(I-\gamma\lambda P)^{-1}\Phi]^{-1} \\ &\quad= \Phi\Big[\Phi^{\mathrm{T}}D_\rho(I-\gamma\lambda P-(1-\lambda)\gamma P)(I-\gamma\lambda P)^{-1}\Phi\Big]^{-1} \\ &\quad= \Phi\Big\{\Big(\Phi^{\mathrm{T}}D_\rho\Phi\Big) - \Big[\Phi^{\mathrm{T}}D_\rho(1-\lambda)\gamma P(I-\gamma\lambda P)^{-1}\Phi\Big]\Big\}^{-1} \\ &\quad= \Phi(\mathrm{Z}_\rho - \Phi^{\mathrm{T}}D_\rho\mathrm{Z}\Phi)^{-1}.\end{aligned}$$

Since Z_ρ and $A-\gamma B$ are invertible, $I-\mathrm{Z}_\rho^{-1}\Phi^{\mathrm{T}}D_\rho\mathrm{Z}\Phi$ is also invertible. Thus, we have

$$\Phi(A-\gamma B)^{-1} = \Phi(I-\mathrm{Z}_\rho^{-1}\Phi^{\mathrm{T}}D_\rho\mathrm{Z}\Phi)^{-1}\mathrm{Z}_\rho^{-1}.$$

Let $X = \Phi$ and $Y = \mathrm{Z}_\rho^{-1}\Phi^{\mathrm{T}}D_\rho\mathrm{Z}$. Using $X(I-YX)^{-1} = (I-XY)^{-1}X$, we have

$$\begin{aligned}\Phi(A-\gamma B)^{-1} &= \Phi\Big(I-\Phi\mathrm{Z}_\rho^{-1}\Phi^{\mathrm{T}}D_\rho\mathrm{Z}\Big)^{-1}\Phi\mathrm{Z}_\rho^{-1} \\ &= \Big(I-\Phi\Big(\Phi^T D_\rho\Phi\Big)^{-1}\Phi^{\mathrm{T}}D_\rho\mathrm{Z}\Big)^{-1}\Phi\mathrm{Z}_\rho^{-1}.\end{aligned}$$

Since $\|\Pi M\|_\rho = \dfrac{(1-\lambda)\gamma}{1-\lambda\gamma}$, $(I-\Pi M)$ is invertible (Tsitsiklis and Roy, 1997). Then according to $\Pi = \Phi\Big(\Phi^T D_\rho\Phi\Big)^{-1}\Phi^{\mathrm{T}}D_\rho$, we obtain

$$\Phi(A-\gamma B)^{-1} = (I-\Pi\mathrm{Z})^{-1}\Phi\mathrm{Z}_\rho^{-1}.$$

Thus, we get the final result. □

Lemma 2 Let $\varepsilon_1 = \bar{A}-\gamma\bar{B}-(A-\gamma B)$, where $\bar{A}-\gamma\bar{B}$ is obtained from the learned model. Let $\varepsilon_2 = \bar{b}-b$, where $\bar{b}$ is obtained from the learned model. Then, we can derive

$$\|\varepsilon_1\bar{\theta} - \varepsilon_2\|_2 \leqslant \frac{4\sqrt{d}LV_{\max} + (R_{\max}+1)\sqrt{d}L}{(n-1)(1-\gamma\lambda)^2}.$$

Proof We start by

$$\begin{aligned}\|\varepsilon_1\bar{\theta} - \varepsilon_2\|_2 &\leqslant \|\varepsilon_1\bar{\theta}\|_2 + \|\varepsilon_2\|_2 \\ &\leqslant \|\varepsilon_1\|_2\|\bar{\theta}\|_2 + \|\varepsilon_2\|_2. \quad (58)\end{aligned}$$

Then, we compute $\|\varepsilon_1\|_2$, $\|\bar{\theta}\|_2$, and $\|\varepsilon_2\|_2$ respectively.

1) Compute $\|\varepsilon_1\|_2$.

Recall that

$$\begin{aligned}&A-\gamma B \\ &\quad= \frac{1}{n-1}\sum_{i=1}^{n-1}\sum_{k=-\infty}^{i}(\gamma\lambda)^{i-k}\phi(x_k)(\phi(x_i)-\gamma\phi(x_{i+1}))^{\mathrm{T}} \\ &\quad= \frac{1}{n-1}\sum_{i=1}^{n-1}(\gamma\lambda)^{i}\sum_{k=-\infty}^{i}(\gamma\lambda)^{-k}\phi(x_k)(\phi(x_i)-\gamma\phi(x_{i+1}))^{\mathrm{T}}.\end{aligned}$$

According to Assumption 3, the range and the dimensionality of the feature function are bounded by L and d, respectively. Thus, $\|\phi(x_k)(\phi(x_i)-\gamma\phi(x_{i+1}))\|_2 \leqslant 2dL^2$, and

$$\begin{aligned}&\|A-\gamma B\|_2 \\ &\quad= \frac{1}{(n-1)(1-\gamma\lambda)}\|\sum_{k=-\infty}^{i}(\gamma\lambda)^{-k}\phi(x_k)(\phi(x_i)-\gamma\phi(x_{i+1}))^{\mathrm{T}}\|_2 \\ &\quad\leqslant \frac{2dL^2}{(n-1)(1-\gamma\lambda)^2},\end{aligned}$$

where

$$\begin{aligned}\bar{A}-\gamma\bar{B}&=\frac{1}{n-1}\sum_{i=1}^{n-1}\sum_{k=1}^{i}(\gamma\lambda)^{i-k}\phi(x_k)(\phi(x_i)-\gamma F\phi(x_i))^{\mathrm{T}}\\&=\frac{1}{n-1}\sum_{i=1}^{n-1}(\gamma\lambda)^{i}\sum_{k=1}^{i}(\gamma\lambda)^{-k}\phi(x_k)(\phi(x_i)-\gamma F\phi(x_i))^{\mathrm{T}}.\end{aligned}$$

By $\|\phi(x_k)(\phi(x_i)-\gamma F\phi(x_i))^{\mathrm{T}}\|_2\leqslant 2dL^2$, we have

$$\begin{aligned}&\|\bar{A}-\gamma\bar{B}\|_2\\&=\frac{1}{(n-1)(1-\gamma\lambda)}\|\sum_{k=1}^{i}(\gamma\lambda)^{-k}\phi(x_k)(\phi(x_i)-\gamma F\phi(x_i))^{\mathrm{T}}\|_2\\&\leqslant\frac{2dL^2}{(n-1)(1-\gamma\lambda)^2}.\end{aligned}$$

Thus,

$$\begin{aligned}\|\varepsilon_1\|_2&=\|A-\gamma B-(\bar{A}-\gamma\bar{B})\|_2\\&\leqslant\|A-\gamma B\|_2+\|\bar{A}-\gamma\bar{B}\|_2\\&\leqslant\frac{4dL^2}{(n-1)(1-\gamma\lambda)^2}.\end{aligned}$$

2) Compute $\|\bar{\theta}\|_2$.

Since $\hat{V}_{LSTD(\lambda)-M}(x)=\phi(x)\bar{\theta}$ and $\sqrt{d}L\|\bar{\theta}\|_2\leqslant\sqrt{\phi^{\mathrm{T}}(x)Z_u\theta}=\|\phi(x)\bar{\theta}\|_u=\|\hat{V}_{LSTD(\lambda)-M}(x)\|_u\leqslant V_{\max}$, we have

$$\|\bar{\theta}\|_2\leqslant\frac{V_{\max}}{\sqrt{d}L}.$$

3) Compute $\|\varepsilon_2\|_2$.

Recall that

$$\begin{aligned}b&=\frac{1}{n-1}\sum_{i=1}^{n-1}\sum_{k=1}^{i}(\gamma\lambda)^{i-k}\phi(x_k)r(x_i)\\&=\frac{1}{n-1}\sum_{i=1}^{n-1}(\gamma\lambda)^{i}\sum_{k=1}^{i}(\gamma\lambda)^{-k}\phi(x_k)r(x_i),\end{aligned}$$

and

$$\begin{aligned}\bar{b}&=\frac{1}{n-1}\sum_{i=1}^{n-1}\sum_{k=1}^{i}(\gamma\lambda)^{i-k}\phi(x_i)^{\mathrm{T}}\alpha\phi(x_k)\\&=\frac{1}{n-1}\sum_{i=1}^{n-1}(\gamma\lambda)^{i}\sum_{k=1}^{i}(\gamma\lambda)^{-k}\phi(x_i)^{\mathrm{T}}\alpha\phi(x_k).\end{aligned}$$

Since $\|\phi(x_k)r(x_i)\|_2\leqslant\sqrt{d}LR_{\max}$ and $\|\phi(x_i)^{\mathrm{T}}\alpha\phi(x_k)\|_2\leqslant\sqrt{d}L$, we can derive

$$\begin{aligned}\|\varepsilon_2\|_2&=\|b-\bar{b}\|_2\\&\leqslant\|b\|_2+\|\bar{b}\|_2\\&\leqslant\frac{1}{n-1}(\frac{1}{1-\gamma\lambda})\left[\frac{R_{\max}\sqrt{d}L}{1-\gamma\lambda}+\frac{\sqrt{d}L}{1-\gamma\lambda}\right]\\&\leqslant\frac{(R_{\max}+1)\sqrt{d}L}{(n-1)(1-\gamma\lambda)^2}.\end{aligned}$$

By combining 1), 2), and 3), we have

$$\begin{aligned}&\|\varepsilon_1\bar{\theta}-\varepsilon_2\|_2\\&\leqslant\frac{4dL^2}{(n-1)(1-\gamma\lambda)^2}\|\bar{\theta}\|_2+\frac{(R_{\max}+1)\sqrt{d}L}{(n-1)(1-\gamma\lambda)^2}\\&\leqslant\frac{4\sqrt{d}LV_{max}+(R_{\max}+1)\sqrt{d}L}{(n-1)(1-\gamma\lambda)^2}.\end{aligned}$$

Thus, Lemma 2 is proved. □

Now, we are ready to prove Theorem 1.

Proof Equation (57) can be further transferred as

$$\begin{aligned}&V_{\mathrm{LSTD}(\lambda)}-\hat{V}_{\mathrm{LSTD}(\lambda)-\mathrm{M}}\\&=(I-\Pi Z)\,\Phi{Z_\rho}^{-1}[(A-\gamma B)\bar{\theta}-b].\end{aligned}$$

From Assumption 1, the Markov chain has the unique distribution ρ, and hence we use ρ to bound the error. $\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)-M}\|_\rho$ can be formulated as

$$\begin{aligned}&\|V_{\mathrm{LSTD}(\lambda)}-\hat{V}_{\mathrm{LSTD}(\lambda)-\mathrm{M}}\|_\rho\\&=\|\,(I-\Pi Z)\,\Phi{Z_\rho}^{-1}\left(\varepsilon_1\bar{\theta}-\varepsilon_2\right)\|_\rho.\end{aligned}$$

Since $\|\Pi Z\|_\rho=\frac{(1-\lambda)\,\gamma}{1-\lambda\gamma}<1$, we have

$$\|(I-\Pi Z)^{-1}\|_\rho=\|\sum_{i=0}^{\infty}(\Pi Z)^i\|_\rho\leqslant\frac{1}{1-\frac{(1-\lambda)\,\gamma}{1-\lambda\gamma}}=\frac{1-\lambda\gamma}{1-\gamma}.$$

Notice for all x,

$$\begin{aligned}\|\Phi{Z_\rho}^{-1}x\|_\rho&=\sqrt{x^{\mathrm{T}}{Z_\rho}^{-1}\Phi^T D_\rho\Phi{Z_\rho}^{-1}x}\\&=\sqrt{x^{\mathrm{T}}{Z_\rho}^{-1}x}\\&\leqslant\frac{1}{\sqrt{v}}\|x\|_2.\end{aligned}$$

By using the above result, we have

$$\begin{aligned}&|V_{\mathrm{LSTD}(\lambda)}-\hat{V}_{\mathrm{LSTD}(\lambda)-\mathrm{M}}\|_\rho\\&\leqslant\frac{1-\lambda\gamma}{1-\gamma}\frac{1}{\sqrt{v}}\|\left(\varepsilon_1\bar{\theta}-\varepsilon_2\right)\|_2\\&\leqslant\frac{4\sqrt{d}LV_{\max}+(R_{\max}+1)\sqrt{d}L}{\sqrt{v}(1-\gamma)(n-1)(1-\gamma\lambda)}.\end{aligned}\tag{59}$$

Therefore, we get the final result. □

Appendix B Proof of Theorem 2

The proof of Theorem 2 requires two conclusions, which are shown in Lemma 3 and Lemma 4.

Lemma 3 (Tsitsiklis and Roy, 1997) The approximation error $\|V-V_{LSTD(\lambda)}\|_\rho$ between the true value function V and the estimated value function $V_{LSTD(\lambda)}$ satisfies

$$\|V-V_{LSTD(\lambda)}\|_\rho=\frac{1-\gamma\lambda}{1-\gamma}\|V-\Pi V\|_\rho.$$

Lemma 4 (Tagorti and Scherrer, 2015) Let n_0 be the smallest number of the sample that satisfies

$$\frac{4dL^2}{\sqrt{n_0-1}(1-\gamma)\nu}\sqrt{\left(1+\left\lfloor\frac{\log(n_0-1)}{\log(\frac{1}{\lambda\gamma})}\right\rfloor\right)I(n_0-1,\delta)}+$$
$$\frac{2dL^2}{(1-\gamma)\nu(n_0-1)(1-\lambda\gamma)}+\frac{4dL^2}{(1-\gamma)\nu(n_0-1)}\left\lfloor\frac{\log(n_0-1)}{\log(\frac{1}{\lambda\gamma})}\right\rfloor<1,$$

where $I(n,\delta)=32\Lambda(n,\delta)\max\left\{\frac{\Lambda(n,\delta)}{b},1\right\}^{\frac{1}{\kappa}}$ and $\delta\in(0,1)$.

Then the estimation error between $V_{LSTD(\lambda)}$ and $\hat{V}_{LSTD(\lambda)}$ satisfies the following inequality with the probability 1-δ:

$$\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)}\|_\rho \leqslant \frac{4V_{\max}dL^2}{\sqrt{n-1}(1-\gamma)\nu}\sqrt{\left(1+\left\lfloor\frac{\log(n-1)}{\log(\frac{1}{\gamma\lambda})}\right\rfloor\right)I(n-1,\delta)}+h(n,\delta), \tag{60}$$

where $\Lambda(n,\delta)=\log\left(\frac{8n^2}{\delta}\right)+\log(\max\{4e^2,n\bar{\beta}\})$, $h(n,\delta)=O(\frac{1}{\sqrt{n}}\log^k(\frac{1}{\sqrt{n}}))$ and ν is the smallest eigenvalue of the Gram matrix $\Phi^{\mathrm{T}}D_\rho\Phi$.

Now, we are ready to prove Theorem 2.

Proof The global error comprises the following three errors:

1) The approximation error $\|V-\Pi V\|_\rho$ (Lemma 3);
2) The model error $\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)-M}\|_\rho$ (Theorem 1);
3) The estimation error based on finite samples $\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)}\|_\rho$ (Lemma 4).

Since the global error contains $\frac{1-\gamma\lambda}{1-\gamma}\|V-\Pi V\|_\rho$, $\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)-M}\|_\rho$, and $\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)}\|_\rho$, the global error $\|V-\hat{V}_{LSTD(\lambda)}\|_\rho$ can be expressed as

$$\|V-\hat{V}_{LSTD(\lambda)}\|_\rho \leqslant \frac{1-\gamma\lambda}{1-\gamma}\|V-\Pi V\|_\rho+$$
$$+\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)-M}\|_\rho+\|V_{LSTD(\lambda)}-\hat{V}_{LSTD(\lambda)}\|_\rho.$$

By Theorems 1, 3 and 4, we get the final result. □

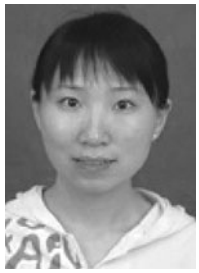
References

1. Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press, 1998
2. Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J, Bellemare M, Graves A, Riedmiller M, Fidjeland A, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human level control through deep reinforcement learning. Nature, 2015, 518(7540): 529–533
3. Littman M L. Reinforcement learning improves behaviour from evaluative feedback. Nature, 2015, 521(7553): 445–451
4. Andersson O, Heintz F, Doherty P. On the undecidability of probabilistic planning and infinite-horizon partially observable. In: Proceedings of the 29th National Conference on Artificial Intelligence. 2015, 2497–2503
5. Bellman R E, Dreyfus S E. Applied Dynamic Programming. Princeton, NJ: Princeton University Press, 2015
6. Barker J K, Korf R E. Limitations of front-to-end bidirectional heuristic search. In: Proceedings of the 29th National Conference on Artificial Intelligence. 2015, 1086–1092
7. Robert C P, Casella G. Monte Carlo Statistical Methods. New York: Springer Science & Business Media, 2013
8. Sutton R, Mahmood A R, Precup D, Hasselt H V. A new Q (λ) with interim forward view and Monte-Carlo equivalence. In: Proceedings of International Conference on Machine Learning. 2014, 568–576
9. Seijen H V, Sutton R. True online TD (λ). In: Proceedings of International Conference on Machine Learning. 2014, 692–700
10. Hasselt H V, Mahmood A R, Sutton R S. Off-policy TD (λ) with a true online equivalence. In: Proceedings of International Conference on Uncertainty in Artificial Intelligence. 2014, 330–339
11. Werbos P J. Advanced forecasting methods for global crisis warning and models of intelligence. General Systems, 1977, 22(6): 25–38
12. Al-Tamimi A, Lewis F L, Abu-Khalaf M. Discrete-time nonlinear HJB solution using approximate dynamic programming: convergence proof. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2008, 38(4): 943–949
13. Wang F Y, Jin N, Liu D E, Wei Q L. Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with ϵ-error bound. IEEE Transactions on Neural Networks, 2011, 22(1): 24–36
14. Liu D, Wei Q L. Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems. IEEE Transactions on Neural Networks and Learning Systems, 2014, 25(3): 621–634
15. Murray J J, Cox C J, Lendaris G G, Saeks R. Adaptive dynamic programming. IEEE Transactions on Systems, Man, and Cybernetics, Part C, 2002, 32(2): 140–153
16. Hanselmann T, Noakes L, Zaknich A. Continuous time adaptive critics. IEEE Transactions on Neural Networks, 2007, 18(3): 631–647
17. Wei Q, Song R, Yan P. Data-driven zero-sum neuro-optimal control for a class of continuous-time unknown nonlinear systems with disturbance using ADP. IEEE Transactions on Neural Networks and Learning Systems, 2016, 27(2): 444–458
18. Busoniu L, Babuška R, De Schutter B, Ernst D. Reinforcement Learning and Dynamic Programming Using Function Approximators. Boca Raton: CRC Press, 2010
19. Sutton R S. Integrated architecture for learning, planning and reacting based on approximating dynamic programming. In: Proceedings of International Conference on Machine Learning. 1990, 216–224

20. Peng J, Williams R J. Efficient learning and planning within the Dyna framework. Adaptive Behavior, 1993, 4(1): 437–454
21. Moore A W, Atkeson C G. Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning, 1993, 13(1): 103–130
22. Sutton R S, Szepesvári C, Geramfard A, Bowling M. Dyna-style planning with linear function approximation and prioritized sweeping. In: Proceedings of International Conference on Uncertainty in Artificial Intelligence. 2008, 528–536
23. Silver D, Sutton R S, Müller M. Temporal-difference search in computer Go. Machine Learning, 2012, 87(2): 183–219
24. Coulom R. Efficient selectivity and backup operators in Monte-Carlo tree search. In: Proceedings of the 5th International Conference on Computers and Games, 2006, 72–83
25. Zhou Y C, Liu Q, Fu Q M, Zhang Z Z. Trajectory sampling value iteration: Improved Dyna search for MDPs. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems. 2015, 1685–1686
26. Martin H, De Lope J. Ex< α>: An effective algorithm for continuous actions reinforcement learning problems. In: Proceedings of the 35th Annual Conference of IEEE on Industrial Electronics. 2009, 2063–2068
27. Weinstein A, Littman M L. Bandit-based planning and learning in continuous-action Markov decision processes. In: Proceedings of International Conference on Automated Planning and Scheduling. 2012, 306–314
28. Busoniu L, Daniels A, Munos R, Babuška R. Optimistic planning for continuous-action deterministic systems. In: Proceedings of IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. 2013, 69–76
29. Grondman I, Vaandrager M, Busoniu L, Babuška R, Schuitema E. Efficient model learning methods for actor-critic control. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 2012, 42(3): 591–602
30. Hasselt H V. Reinforcement learning in continuous state and action spaces. In: Wiering M, van Otterlo M, eds. Reinforcement Learning. Berlin: Springer Heidelberg, 2012, 207–251
31. Degris T, Pilarski P M, Sutton R S. Model-free reinforcement learning with continuous action in practice. In: Proceedings of IEEE American Control Conference. 2012, 2177–2182
32. Bradtke S J, Barto A G. Linear least-squares algorithms for temporal difference learning. Machine Learning, 1996, 22(1–3): 33–57
33. Boyan J A. Technical update: least-square temporal difference learning. Machine learning, 2002, 49(2–3): 233–246
34. Tagorti M, Scherer B. On the rate of the convergence and error bounds for LSTD(λ). In: Proceedings of International Conference on Machine Learning. 2015, 528–536
35. Hwang K S, Jiang W C, Chen Y J. Model learning and knowledge sharing for a multiagent system With Dyna-Q. IEEE Transactions on Cybernetics, 2015, 45(5): 964–976
36. Faulkner R, Precup D. Dyna planning using a feature based generative model. In: Proceedings of Advances in Neural Information Processing Systems. 2010, 1–9
37. Silver D, Sutton R S, Müller M. Reinforcement learning of local shape in the game of Go. In: Proceedings of International Joint Conference on Artificial Intelligence. 2007, 1053–1058
38. Yao H, Szepesvári C. Approximate policy iteration with linear action models. In: Proceedings of the 26th National Conference on Artificial Intelligence. 2012
39. Tsitsiklis J N, Roy B V. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, 1997, 42(5): 674–690
40. Nedic A, Bertsekas D P. Least squares policy evaluation algorithms with linear function approximation. Theory and Applications, 2002, 13(1–2): 79–110
41. Lazaric A, Ghavamzadeh M, Munos R. Finite-sample analysis of least-square policy iteration. Journal of Machine Learning Research, 2012, 13(4): 3041–3074
42. Xu X, He H G, Hu D W. Efficient reinforcement learning using recursive least-squares methods. Journal of Artificial Intelligence Research, 2002, 16(1): 259–292
43. Ljung L, Soderstron T. Theory and practice of recursive identification. Cambridge, MA: MIT Press, 1983
44. Geramifard A, Bowling M, Sutton R S. Incremental least-squares temporal difference learning. In: Proceedings of the National Conference on Artificial Intelligence. 2006, 356–361
45. Berenji H R, Khedkar P. Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks, 1992, 3(5): 724–740
46. Sutton R S. Generalization in reinforcement learning: successful examples using sparse coarse coding. Neural Information Processing Systems, 1996: 1038–1044
47. Bhatnagar S, Sutton R S, Ghavamzadeh M, Lee M. Natural actor-critic algorithms. Automatica, 2009, 45(11): 2471–2482
48. Liu D R, Li H L, Wang D. Feature selection and feature learning for high-dimensional batch reinforcement learning: a survey. International Journal of Automation and Computing, 2015, 12(3): 229–242
49. Farahmand A M, Ghavamzadeh M, Szepesvári C, Mannor S. Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems. In: Proceedings of American Control Conference. 2009, 725–730
50. Farahmand A M, Szepesvári C. Model selection in reinforcement learning. Machine Learning, 2011, 85(3): 299–332
51. Kolter J Z, Ng A Y. Regularization and feature selection in least-squares temporal difference learning. In: Proceedings of the 26th Annual International Conference on Machine Learning. 2009, 521–528
52. Ghavamzadeh M, Lazaric A, Munos R, Hoffman M W. Finite-sample analysis of LASSO-TD. In: Proceedings of the 28th International Conference on Machine Learning. 2011, 1177–1184
53. Mahadevan S, Liu B. Sparse Q-learning with mirror descent. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence. 2012, 564–573
54. Painter-Wakefield C, Parr R. Greedy algorithms for sparse reinforcement learning. In: Proceedings of the 29th International Conference on Machine Learning. 2012, 1391–1398
55. Johns J, Mahadevan S. Sparse approximate policy evaluation using graph-based basis functions. Technical Report UM-CS-2009-041. 2009
56. Ghavamzadeh M, Lazaric A, Maillard O A, Munos R. LSTD with random projections. In: Proceedings of Advances in Neural Information

Processing Systems. 2010, 721–729

57. Liu B, Mahadevan S. Compressive reinforcement learning with oblique random projections. Technical Report UM-CS-2011-024. 2011
58. Xu X, Hu D W, Lu X C. Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactionson Neural Networks, 2007, 18(4): 973–992



Shan Zhong is now a PhD candidate in School of Computer Science and Technology at Soochow University, China. She received her master degree from Jiangsu University, China. She is also a lecturer in Changshu Institute of Technology, China. Her main research interests include machine learning and deep learning.

Quan Liu is now a professor and PhD supervisor in School of Computer Science and Technology at Soochow University, China. He received his PhD degree at Jilin University, China in 2004. He worked as a post-doctor at Nanjing University, China from 2006-2008. He is a senior member of China Computer Federation. His main research interests include reinforcement learning, intelligence information processing, and automated reasoning.

Zongzhang Zhang received his PhD degree in computer science from University of Science and Technology of China, China in 2012. He is currently an associate professor at Soochow University, China. He worked as a research fellow at National University of Singapore, Singapore from 2012 to 2014 and as a visiting scholar at Rutgers University, USA from 2010 to 2011. His research directions include POMDPs, reinforcement learning, and multi-agent systems.

Qiming Fu received his master's and PhD degrees in School of Computer Science and Technology at Soochow University, China in 2011 and 2014, respectively. He works as a lecturer at Suzhou University of Science and Technology, China. His main research interests include reinforcement learning, Bayesian methods, deep learning.