Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2016, Article ID 4824072, 15 pages
http://dx.doi.org/10.1155/2016/4824072

Research Article

Efficient Actor-Critic Algorithm with Hierarchical Model Learning and Planning

Shan Zhong,[1,2] Quan Liu,[1,3,4] and QiMing Fu[5]

[1]*School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215000, China*
[2]*School of Computer Science and Engineering, Changshu Institute of Technology, Changshu, Jiangsu 215500, China*
[3]*Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu 210000, China*
[4]*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China*
[5]*College of Electronic & Information Engineering, Suzhou University of Science and Technology, Jiangsu, Suzhou 215000, China*

Correspondence should be addressed to Quan Liu; quanliu@suda.edu.cn

Received 29 May 2016; Revised 28 July 2016; Accepted 16 August 2016

Academic Editor: Leonardo Franco

To improve the convergence rate and the sample efficiency, two efficient learning methods AC-HMLP and RAC-HMLP (AC-HMLP with ℓ_2-regularization) are proposed by combining actor-critic algorithm with hierarchical model learning and planning. The hierarchical models consisting of the local and the global models, which are learned at the same time during learning of the value function and the policy, are approximated by local linear regression (LLR) and linear function approximation (LFA), respectively. Both the local model and the global model are applied to generate samples for planning; the former is used only if the state-prediction error does not surpass the threshold at each time step, while the latter is utilized at the end of each episode. The purpose of taking both models is to improve the sample efficiency and accelerate the convergence rate of the whole algorithm through fully utilizing the local and global information. Experimentally, AC-HMLP and RAC-HMLP are compared with three representative algorithms on two Reinforcement Learning (RL) benchmark problems. The results demonstrate that they perform best in terms of convergence rate and sample efficiency.

1. Introduction and Related Work

Reinforcement Learning (RL) [1–4], a framework for solving the Markov Decision Process (MDP) problem, targets generating the optimal policy by maximizing the expected accumulated rewards. The agent interacts with its environment and receives information about the current state at each time step. After the agent chooses an action according to the policy, the environment will transition to a new state while emitting a reward. RL can be divided into two classes, online and offline. Online method learns by interacting with the environment, which easily incurs the inefficient use of data and the stability issue. Offline or batch RL [5] as a subfield of dynamic programming (DP) [6, 7] can avoid the stability issue and achieve high sample efficiency.

DP aims at solving optimal control problems, but it is implemented backward in time, making it offline and computationally expensive for complex or real-time problems. To avoid the curse of dimensionality in DP, approximate dynamic programming (ADP) received much attention to obtain approximate solutions of the Hamilton-Jacobi-Bellman (HJB) equation by combining DP, RL, and function approximation [8]. Werbos [9] introduced an approach for ADP which was also called adaptive critic designs (ACDs). ACDs consist of two neural networks (NNs), one for approximating the critic and the other for approximating the actor, so that DP can be solved approximately forward in time. Several synonyms about ADP and ACDs mainly include approximate dynamic programming, asymptotic dynamic programming, heuristic dynamic programming, and neurodynamic programming [10, 11].

The iterative nature of the ADP formulation makes it natural to design the optimal discrete-time controllers. Al-Tamimi et al. [12] established a heuristic dynamic programming

algorithm based on value iteration, where the convergence is proved in the context of general nonlinear discrete systems. Dierks et al. [13] solved the optimal control of nonlinear discrete-time systems by using two processes, online system identification and offline optimal control training, without the requirement of partial knowledge about the system dynamics. Wang et al. [14] focused on applying iterative ADP algorithm with error boundary to obtain the optimal control law, in which the NNs are adopted to approximate the performance index function, compute the optimal control policy, and model the nonlinear system.

Extensions of ADP for continuous-time systems face the challenges involved in proving stability and convergence meanwhile ensuring the algorithm being online and model-free. To approximate the value function and improve the policy for continuous-time system, Doya [15] derived a temporal difference (TD) error-based algorithm in the framework of HJB. Under a measure of input quadratic performance, Murray et al. [16] developed a stepwise ADP algorithm in the context of HJB. Hanselmann et al. [17] put forward a continuous-time ADP formulation, where Newton's method is used in the second-order actor adaption to achieve the convergence of the critic. Recently, Bhasin et al. [18] built an actor-critic-identifier (ACI), an architecture that represents the actor, critic, and model by taking NNs as nonlinearly parameterized approximators while the parameters of NNs are updated by least-square method.

All the aforementioned ADP variants utilized the NN as the function approximator; however, linear parameterized approximators are usually more preferred in RL, because they make it easier to understand and analyze the theoretical properties of the resulting RL algorithms [19]. Moreover, most of the above works did not learn a model online to accelerate the convergence rate and improve the sample efficiency. Actor-critic (AC) algorithm was introduced in [20] for the first time; many variants which approximated the value function and the policy by linear function approximation have been widely used in continuous-time systems since then [21–23]. By combining model learning and AC, Grondman et al. [24] proposed an improved learning method called Model Learning Actor-Critic (MLAC) which approximates the value function, the policy, and the process model by LLR. In MLAC, the gradient of the next state with respect to the current action is computed for updating the policy gradient, with the goal of improving the convergence rate of the whole algorithm. In their latter work [25], LFA takes the place of LLR as the approximation method for value function, the policy, and the process model. Enormous samples are still required when only using such a process model to update the policy gradient. Afterward, Costa et al. [26] derived an AC algorithm by introducing Dyna structure called Dyna-MLAC which approximated the value function, the policy, and the model by LLR as MLAC did. The difference is that Dyna-MLAC applies the model not only in updating the policy gradient but also in planning [27]. Though planning can improve the sample efficiency to a large extent, the model learned by LLR is just a local model so that the global information of samples is yet neglected.

Though the above works learn a model during learning of the value function and the policy, only the local information of the samples is utilized. If the global information of the samples can be utilized reasonably, the convergence performance will be improved further. Inspired by this idea, we establish two novel AC algorithms called AC-HMLP and RAC-HMLP (AC-HMLP with ℓ_2-regularization). AC-HMLP and RAC-HMLP consist of two models, the global model and the local model. Both models incorporate the state transition function and the reward function for planning. The global model is approximated by LFA while the local model is represented by LLR. The local and the global models are learned simultaneously at each time step. The local model is used for planning only if the error does not surpass the threshold, while the global planning process is started at the end of an episode, so that the local and the global information can be kept and utilized uniformly.

The main contributions of our work on AC-HMLP and RAC-HMLP are as follows:

(1) Develop two novel AC algorithms based on hierarchal models. Distinguishing from the previous works, AC-HMLP and RAC-HMLP learn a global model, where the reward function and the state transition function are approximated by LFA. Meanwhile, unlike the existing model learning methods [28–30] which represent a feature-based model, we directly establish a state-based model to avoid the error brought by inaccurate features.

(2) As MLAC and Dyna-MLAC did, AC-HMLP and RAC-HMLP also learn a local model by LLR. The difference is that we design a useful error threshold to decide whether to start the local planning process. At each time step, the real-next state is computed according to the system dynamics whereas the predicted-next state is obtained from LLR. The error between them is defined as the state-prediction error. If this error does not surpass the error threshold, the local planning process is started.

(3) The local model and the global model are used for planning uniformly. The local and the global models produce local and global samples to update the same value function and the policy; as a result the number of the real samples will decrease dramatically.

(4) Experimentally, the convergence performance and the sample efficiency are thoroughly analyzed. The sample efficiency which is defined as the number of samples for convergence is analyzed. RAC-HMLP and AC-HMLP are also compared with S-AC, MLAC, and Dyna-MLAC in convergence performance and sample efficiency. The results demonstrate that RAC-HMLP performs best and AC-HMLP performs second best, and both of them outperform the other three methods.

This paper is organized as follows: Section 2 reviews some background knowledge concerning MDP and the AC algorithm. Section 3 describes the hierarchical model

learning and planning. Section 4 specifies our algorithms—AC-HMLP and RAC-HMLP. The empirical results of the comparisons with the other three representative algorithms are analyzed in Section 5. Section 6 concludes our work and then presents the possible future work.

2. Preliminaries

2.1. MDP. RL can solve the problem modeled by MDP. MDP can be represented as four-tuple (X, U, ρ, f):

(1) X is the state space. $x_t \in X$ denotes the state of the agent at time step t.

(2) U represents the action space. $u_t \in U$ is the action which the agent takes at the time step t.

(3) $\rho : X \times U \to \mathbb{R}$ denotes the reward function. At the time step t, the agent locates at a state x_t and takes an action u_t resulting in next state x_{t+1} while receiving a reward $r_t = \rho(x_t, u_t)$.

(4) $f : X \times U \to X$ is defined as the transition function. $f(x_t, u_t, x_{t+1})$ is the probability of reaching the next state x_{t+1} after executing u_t at the state x_t.

Policy $h : X \to U$ is the mapping from the state space X to the action space U, where the mathematical set of h depends on specific domains. The goal of the agent is to find the optimal policy h^* that can maximize the cumulative rewards. The cumulative rewards are the sum or discounted sum of the received rewards and here we use the latter case.

Under the policy h, the value function $V^h : X \to \mathbb{R}$ denotes the expected cumulative rewards, which is shown as

$$V^h(x) = E_h\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid x = x_t\right\}, \tag{1}$$

where $\gamma \in [0, 1]$ represents the discount factor. x_t is the current state.

The optimal state-value function $V^*(x)$ is computed as

$$V^*(x) = \max_h V(x), \quad \forall x \in X. \tag{2}$$

Therefore, the optimal policy h^* at state x can be obtained by

$$h^*(x) = \arg\max_h V^h(x), \quad \forall x \in X. \tag{3}$$

2.2. AC Algorithm. AC algorithm mainly contains two parts, actor and critic, which are stored separately. Actor and critic are also called the policy and value function, respectively. The actor-only methods approximate the policy and then update its parameter along the direction of performance improving, with the possible drawback being large variance resulting from policy estimation. The critic-only methods estimate the value function by approximating a solution to the Bellman equation; the optimal policy is found by maximizing the value function. Other than the actor-only methods, the critic-only methods do not try to search the optimal policy in policy space. They just estimate the critic for evaluating the performance of the actor; as a result the near-optimality of the resulting policy cannot be guaranteed. By combining the merits of the actor and the critic, AC algorithms were proposed where the value function is approximated to update the policy.

The value function and the policy are parameterized by $V(x, \theta)$ and $h(x, \beta)$, where θ and β are the parameters of the value function and the policy, respectively. At each time step t, the parameter θ is updated as

$$\theta_{t+1} = \theta_t + \alpha_c \delta_t \frac{\partial V(x, \theta)}{\partial \theta} \quad x = x_t,\ \theta = \theta_t, \tag{4}$$

where

$$\delta_t = r_t + \gamma V(x_t, \theta_t) - V(x_t, \theta_t), \tag{5}$$

denoting the TD-error of the value function. $\partial V(x, \theta)/\partial\theta$ represents the feature of the value function. The parameter $\alpha_c \in [0, 1]$ is the learning rate of the value function.

Eligibility is a trick to improve the convergence via assigning the credits to the previously visited states. At each time step t, the eligibility can be represented as

$$e_t(x) = \begin{cases} \dfrac{\partial V(x, \theta)}{\partial \theta} & x_t = x \\ \lambda\gamma e_{t-1}(x) & x_t \neq x, \end{cases} \tag{6}$$

where $\lambda \in [0, 1]$ denotes the trace-decay rate.

By introducing the eligibility, the update for θ in (4) can be transformed as

$$\theta_{t+1} = \theta_t + \alpha_c \delta_t e_t(x). \tag{7}$$

The policy parameter β_t can be updated by

$$\beta_{t+1} = \beta_t + \alpha_a \delta_t \Delta u_t \frac{\partial h(x, \beta)}{\partial \beta}, \tag{8}$$

where $\partial h(x, \beta)/\partial\beta$ is the feature of the policy. Δu_t is a random exploration term conforming to zero-mean normal distribution. $\alpha_a \in [0, 1]$ is the learning rate of the policy.

S-AC (Standard AC algorithm) serves as a baseline to compare with our method, which is shown in Algorithm 1. The value function and the policy are approximated linearly in Algorithm 1, where TD is used as the learning algorithm.

3. Hierarchical Model Learning and Planning

3.1. Why to Use Hierarchical Model Learning and Planning. The model in RL refers to the state transition function and the reward function. When the model is established, we can use any model-based RL method to find the optimal policy, for example, DP. Model-based methods can significantly decrease the number of the required samples and improve the convergence performance. Inspired by this idea, we introduce the hierarchical model learning into AC algorithm so as to make it become more sample-efficient. Establishing a relative accurate model for the continuous state and action spaces is still an open issue.

Input: γ, λ, α_a, α_c, σ^2
(1) Initialize θ, β
(2) **Loop**
(3) $e \leftarrow \vec{1}, \theta \leftarrow \vec{0}, \beta \leftarrow \vec{0}, x \leftarrow x_0, t \leftarrow 1$
(4) **Repeat all episodes**
(5) Choose Δu_t according to $N(0, \sigma^2)$
(6) $u_t \leftarrow h(x, \beta) + \Delta u_t$
(7) Execute u_t and observe r_{t+1} and x_{t+1}
(8) Update the eligibility of the value function: $e_t(x) = \{\partial V(x, \theta)/\partial\theta, x_t = x; \lambda\gamma e_{t-1}(x), x_t \neq x\}$
(9) Compute the TD error: $\delta_t = r_{t+1} + \gamma V(x_{t+1}, \theta_t) - V(x_t, \theta_t)$
(10) Update the parameter of the value function: $\theta_{t+1} = \theta_t + \alpha_c \delta_t e_t(x)$
(11) Update the parameter of the policy: $\beta_{t+1} = \beta_t + \alpha_a \delta_t \Delta u_t (\partial h(x, \beta)/\partial\beta)$
(12) $t \leftarrow t + 1$
(13) **End Repeat**
(14) **End Loop**
Output: θ, β

Algorithm 1: S-AC.

The preexisting works are mainly aimed at the problems with continuous states but discrete actions. They approximated the transition function in the form of probability matrix which specifies the transition probability from the current feature to the next feature. The indirectly observed features result in the inaccurate feature-based model. The convergence rate will be slowed significantly by using such an inaccurate model for planning, especially at each time step in the initial phase.

To solve these problems, we will approximate a state-based model instead of the inaccurate feature-based model. Moreover, we will introduce an additional global model for planning. The global model is applied only at the end of each episode so that the global information can be utilized as much as possible. Using such a global model without others will lead to the loss of valuable local information. Thus, like Dyna-MLAC, we also approximate a local model by LLR and use it for planning at each time step. The difference is that a useful error threshold is designed for the local planning in our method. If the state-prediction error between the real-next state and the predicted one does not surpass the error threshold, the local planning process will be started at the current time step. Therefore the convergence rate and the sample efficiency can be improved dramatically by combining the local and global model learning and planning.

3.2. Learning and Planning of the Global Model. The global model establishes separate equations for the reward function and the state transition function of every state component by linear function approximation. Assume the agent is at state $x_t = \{x_{t,1}, \ldots, x_{t,K}\}$, where K is the dimensionality of the state x_t; the action u_t is selected according to the policy h; then all the components $\{x'_{t+1,1}, x'_{t+1,2}, \ldots, x'_{t+1,K}\}$ of the next state x'_{t+1} can be predicted as

$$\begin{aligned} x'_{t+1,1} &= {\eta_{t,1}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t) \\ x'_{t+1,2} &= {\eta_{t,2}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t) \\ &\vdots \\ x'_{t+1,K} &= {\eta_{t,K}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t), \end{aligned} \tag{9}$$

where $\eta_{t,i} = (\eta_{t,i1}, \eta_{t,i2}, \ldots, \eta_{t,iD})^{\mathrm{T}}$, $1 \le i \le K$, is the parameter of the state transition function corresponding to the ith component of the current state at time step t, with D being the dimensionality of the feature $\phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$.

Likewise, the reward r'_{t+1} can be predicted as

$$r'_{t+1} = {\varsigma_t}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t), \tag{10}$$

where $\varsigma_t = (\varsigma_{t1}, \varsigma_{t2}, \ldots, \varsigma_{tD})$ is the parameter of the reward function at time step t.

After $\eta_{t,1}, \eta_{t,2}, \ldots, \eta_{t,K}$ and ς are updated, the model can be applied to generate samples. Let the current state be $x_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,k})$; after the action u_t is executed, the parameters $\eta_{t+1,i}$ $(1 \le i \le K)$ can be estimated by the gradient descent method, shown as

$$\begin{aligned} \eta_{t+1,1} &= \eta_{t,1} \\ &\quad + \alpha_m (x_{t+1,1} - x'_{t+1,1}) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t) \\ \eta_{t+1,2} &= \eta_{t,2} \\ &\quad + \alpha_m (x_{t+1,2} - x'_{t+1,2}) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t) \\ &\vdots \\ \eta_{t+1,K} &= \eta_{t,K} \\ &\quad + \alpha_m (x_{t+1,K} - x'_{t+1,K}) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t), \end{aligned} \tag{11}$$

where $\alpha_m \in [0, 1]$ is the learning rate of the model. x_{t+1} is the real-next state obtained according to the system dynamics, where $x_{t+1,i}$ $(1 \le i \le K)$ is its ith component. $(x'_{t+1,1}, x'_{t+1,2}, \ldots, x'_{t+1,K})$ is the predicted-next state according to (9).

The parameter ς_{t+1} is estimated as

$$\varsigma_{t+1} = \varsigma_t + \alpha_m (r_{t+1} - r'_{t+1}) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t), \tag{12}$$

where r_{t+1} is the real reward reflected by the system dynamics while r'_{t+1} is the predicted reward obtained according to (10).

3.3. Learning and Planning of the Local Model. Though the local model also approximates the state transition function and the reward function as the global model does, LLR is served as the function approximator instead of LFA. In the local model, a memory storing the samples in the form of $(x_t, u_t, x_{t+1}, r_{t+1})$ is maintained. At each time step, a new sample is generated from the interaction and it will take the place of the oldest one in the memory. Not all but only L-nearest samples in the memory will be selected for computing the parameter matrix $\Gamma \in \mathbb{R}^{(K+1)\times(K+2)}$ of the local model. Before achieving this, the input matrix $X \in \mathbb{R}^{(K+2)\times L}$ and the output matrix $Y \in \mathbb{R}^{(K+1)\times L}$ should be prepared as follows:

$$X = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{L,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{L,2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1,K} & x_{2,K} & \cdots & x_{L,K} \\ u_1 & u_2 & \cdots & u_L \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \quad Y = \begin{bmatrix} x_{2,1} & x_{3,1} & \cdots & x_{L+1,1} \\ x_{2,2} & x_{3,2} & \cdots & x_{L+1,2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{2,K} & x_{3,K} & \cdots & x_{L+1,K} \\ r_2 & r_3 & \cdots & r_{L+1} \end{bmatrix}. \tag{13}$$

The last row of X consisting of ones is to add a bias on the output. Every column in the former $K + 1$ lines of X corresponds to a state-action pair; for example, the ith column is the state-action $(x_i, u_i)^{\mathrm{T}}, 1 \le i \le L$. Y is composed of L next states and rewards corresponding to X.

Γ can be obtained via solving $Y = \Gamma X$ as

$$\Gamma = YX^{\mathrm{T}}\left(XX^{\mathrm{T}}\right)^{-1}. \tag{14}$$

Let the current input vector be $[x_{t,1}, \ldots, x_{t,K}, u_t, 1]^{\mathrm{T}}$; the output vector $[x'_{t+1,1}, \ldots, x'_{t+1,K}, r'_{t+1}]^{\mathrm{T}}$ can be predicted by

$$\left[x'_{t+1,1}, \ldots, x'_{t+1,k}, r'_{t+1}\right]^{\mathrm{T}} = \Gamma\left[x_{t,1}, \ldots, x_{t,k}, u_t, 1\right]^{\mathrm{T}}, \tag{15}$$

where $[x_{t,1}, \ldots, x_{t,K}]$ and $[x'_{t+1,1}, \ldots, x'_{t+1,K}]$ are the current state and the predicted-next state, respectively. r'_{t+1} is the predicted reward.

Γ is estimated according to (14) at each time step; thereafter the predicted-next state and the predicted reward can be obtained by (15). Moreover, we design an error threshold to decide whether local planning is required. We compute the state-prediction error between the real-next state and the predicted-next state at each time step. If this error does not surpass the error threshold, the local planning process will be launched. The state-prediction error is formulated as

$$Er_t = \max\left\{\left|\frac{x'_{t+1,1} - x_{t+1,1}}{x_{t+1,1}}\right|, \left|\frac{x'_{t+1,2} - x_{t+1,2}}{x_{t+1,2}}\right|, \ldots, \left|\frac{x'_{t+1,K} - x_{t+1,K}}{x_{t+1,K}}\right|, \left|\frac{r'_{t+1} - r_{t+1}}{r_{t+1}}\right|\right\}. \tag{16}$$

Let the error threshold be ξ; then the local model will be used for planning only if $Er_t \le \xi$ at time step t. In the local planning process, a sequence of locally simulated samples in the form of $(x_{t,1}, \ldots, x_{t,K}, u_t, x'_{t+1,1}, \ldots, x'_{t+1,K}, r')$ are generated to improve the convergence of the same value function and the policy as the global planning process does.

4. Algorithm Specification

4.1. AC-HMLP. AC-HMLP algorithm consists of a main algorithm and two subalgorithms. The main algorithm is the learning algorithm (see Algorithm 2), whereas the two subalgorithms are local model planning procedure (see Algorithm 3) and global model planning procedure (see Algorithm 4), respectively. At each time step, the main algorithm learns the value function (see line (26) in Algorithm 2), the policy (see line (27) in Algorithm 2), the local model (see line (19) in Algorithm 2), and the global model (see lines (10)~(11) in Algorithm 2).

There are several parameters which are required to be determined in the three algorithms. r and λ are discount factor and trace-decay rate, respectively. α_a, α_c, and α_m are the corresponding learning rates of the value function, the policy, and the global model. M_size is the capacity of the memory. ξ denotes the error threshold. L determines the number of selected samples which is used to fit the LLR. σ^2 is the variance that determines the region of the exploration. P_l and P_g are the planning times for local model and global model. Some of these parameters have empirical values, for example, r and λ. The others have to be determined by observing the empirical results.

Notice that Algorithm 3 starts planning at the state x_t which is passed from the current state in Algorithm 2, while Algorithm 4 uses x_0 as the initial state. The reason for using different initializations is that the local model is learned according to the L-nearest samples of the current state x_t, whereas the global model is learned from all the samples. Thus, it is reasonable and natural to start the local and global planning process at the states x_t and x_0, respectively.

4.2. AC-HMLP with ℓ_2-Regularization. Regression approaches in machine learning are generally represented as minimization of a square loss term and a regularization term. The ℓ_2-regularization also called ridge regress is a widely used regularization method in statistics and machine learning, which can effectively prohibit overfitting of learning. Therefore, we introduce ℓ_2-regularization to AC-HMLP in the learning of the value function, the policy, and the model. We term this new algorithm as RAC-HMLP.

Input: γ, λ, α_a, α_c, α_m, ξ, M_size, L, σ^2, K, P_l, P_g

(1) Initialize: $\theta \leftarrow \overrightarrow{0}, \beta \leftarrow \overrightarrow{0}, \Gamma \leftarrow \overrightarrow{0}, \eta_1, \eta_2, \ldots, \eta_K \leftarrow \overrightarrow{0}, \varsigma \leftarrow \overrightarrow{0}$

(2) **Loop**

(3) $e_1 \leftarrow \overrightarrow{1}, x_{1,1}, \ldots, x_{1,k} \leftarrow x_{0,1}, \ldots, x_{0,k}, t \leftarrow 1$, number $\leftarrow 0$

(4) **Repeat all episodes**

(5) Choose Δu_t according to $N(0, \sigma^2)$

(6) Execute the action: $u_t = h(x, \beta) + \Delta u_t$

(7) Observe the reward r_{t+1} and the next state: $x_{t+1} = \{x_{t+1,1}, ..., x_{t+1,k}\}$

% **Update the global model**

(8) Predict the next state:

$$x'_{t+1,1} = {\eta_{t,1}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$
$$x'_{t+1,2} = {\eta_{t,2}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$
$$\vdots$$
$$x'_{t+1,K} = {\eta_{t,K}}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$

(9) Predict the reward: $r'_{t+1} = {\varsigma_t}^{\mathrm{T}} \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$

(10) Update the parameters $\eta_1, \eta_2, \ldots, \eta_K$:

$$\eta_{t+1,1} = \eta_{t,1} + \alpha_m \left(x_{t+1,1} - x'_{t+1,1}\right) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$
$$\eta_{t+1,2} = \eta_{t,2} + \alpha_m \left(x_{t+1,2} - x'_{t+1,2}\right) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$
$$\vdots$$
$$\eta_{t+1,K} = \eta_{t,K} + \alpha_m \left(x_{t+1,K} - x'_{t+1,K}\right) \phi(x_{t,1}, x_{t,2}, \ldots, x_{t,K}, u_t)$$

(11) Update the parameter ς: $\varsigma_{t+1} = \varsigma_t + \alpha_m (r_{t+1} - r'_{t+1}) \phi(x_{t,1}, x_{t,2}, ..., x_{t,K}, u_t)$

% **Update the local model**

(12) **If** $t \leq M$_size

(13) Insert the real sample $(x_t, u_t, x_{t+1}, r_{t+1})$ into the memory M

(14) **Else If**

(15) Replace the oldest one in M with the real sample $(x_t, u_t, x_{t+1}, r_{t+1})$

(16) **End if**

(17) Select L-nearest neighbors of the current state from M to construct X and Y

(18) Predict the next state and the reward: $[x'_{t+1}, r'_{t+1}]^{\mathrm{T}} = \Gamma [x_t, u_t, 1]^{\mathrm{T}}$

(19) Update the parameter Γ: $\Gamma = Y X^{\mathrm{T}} (X X^{\mathrm{T}})^{-1}$

(20) Compute the local error:

$$Er_t = \max\{|(x'_{t+1,1} - x_{t+1,1})/x_{t+1,1}|, |(x'_{t+1,2} - x_{t+1,2})/x_{t+1,2}|, \ldots, |(x'_{t+1,K} - x_{t+1,K})/x_{t+1,K}|, |(r'_{t+1} - r_{t+1})/r_{t+1}|\}$$

(21) **If** $Er_t \leq \xi$

(22) Call Local-model planning $(\theta, \beta, \Gamma, e, \text{number}, P_l)$ (Algorithm 3)

(23) **End If**

% **Update the value function**

(24) Update the eligibility: $e_t(x) = \{\phi(x_t), \ x_t = x; \ \lambda \gamma e_{t-1}(x), \ x_t \neq x\}$

(25) Estimate the TD error: $\delta_t = r_{t+1} + \gamma V(x_{t+1}, \theta_t) - V(x_t, \theta_t)$

(26) Update the value-function parameter: $\theta_{t+1} = \theta_t + \alpha_c \delta_t e_t(x)$

% **Update the policy**

(27) Update the policy parameter: $\beta_{t+1} = \beta_t + \alpha_a \delta_t \Delta u_t \phi(x_t)$

(28) $t = t + 1$

(29) Update the number of samples: number = number + 1

(30) **Until the ending condition is satisfied**

(31) Call Global-model planning $(\theta, \beta, \eta_1, \ldots, \eta_k, \varsigma, e, \text{number}, P_g)$ (Algorithm 4)

(32) **End Loop**

Output: β, θ

Algorithm 2: AC-HMLP algorithm.

(1) **Loop for P_l time steps**
(2) $\bar{x}_1 \leftarrow x_t, \bar{t} \leftarrow 1$
(3) Choose $\Delta u_{\bar{t}}$ according to $N(0, \sigma^2)$
(4) $u_{\bar{t}} = h(x_{\bar{t}}, \beta_{\bar{t}}) + \Delta u_{\bar{t}}$
(5) Predict the next state and the reward: $[x_{\bar{t}+1,1}, \ldots, x_{\bar{t}+1,K}, r_{\bar{t}+1}] = \Gamma_{\bar{t}}[x_{\bar{t},1}, \ldots, x_{\bar{t},K}, u_{\bar{t}}, 1]^{\mathrm{T}}$
(6) Update the eligibility: $e_{\bar{t}}(x) = \{\phi(x_{\bar{t}}), x_{\bar{t}} = x; \lambda\gamma e_{\bar{t}-1}(x), x_{\bar{t}} \neq x\}$
(7) Compute the TD error: $\delta_{\bar{t}} = r_{\bar{t}+1} + \gamma V(x_{\bar{t}+1}, \theta_{\bar{t}}) - V(x_{\bar{t}}, \theta_{\bar{t}})$
(8) Update the value function parameter: $\theta_{\bar{t}+1} = \theta_{\bar{t}} + \alpha_c \delta_{\bar{t}} e_{\bar{t}}(x)$
(9) Update the policy parameter: $\beta_{\bar{t}+1} = \beta_{\bar{t}} + \alpha_a \delta_{\mathrm{TD}} \Delta u_{\bar{t}} \phi(x_{\bar{t}})$
(10) **If** $\bar{t} \leq S$
(11) $\bar{t} = \bar{t} + 1$
(12) **End If**
(13) Update the number of samples: number = number + 1
(14) **End Loop**
Output: β, θ

Algorithm 3: Local model planning $(\theta, \beta, \Gamma, e, \text{number}, P_l)$.

(1) **Loop for P_g times**
(2) $x_{\tilde{t}} \leftarrow x_0, \tilde{t} \leftarrow 1$
(3) **Repeat all episodes**
(4) Choose $\Delta u_{\tilde{t}}$ according to $N(0, \sigma)$
(5) Compute exploration term: $u_{\tilde{t}} = h(x_{\tilde{t}}, \beta_{\tilde{t}}) + \Delta u_{\tilde{t}}$
(6) Predict the next state:

$$x'_{\tilde{t}+1,1} = {\eta_{\tilde{t},1}}^{T} \phi\left(x_{\tilde{t},1}, x_{\tilde{t},2}, \ldots, x_{\tilde{t},K}, u_{\tilde{t}}\right)$$
$$x'_{\tilde{t}+1,2} = {\eta_{\tilde{t},2}}^{T} \phi\left(x_{\tilde{t},1}, x_{\tilde{t},2}, \ldots, x_{\tilde{t},K}, u_{\tilde{t}}\right)$$
$$\vdots$$
$$x'_{\tilde{t}+1,K} = {\eta_{\tilde{t},K}}^{T} \phi\left(x_{\tilde{t},1}, x_{\tilde{t},2}, \ldots, x_{\tilde{t},K}, u_{\tilde{t}}\right)$$

(7) Predict the reward: $r'_{\tilde{t}+1} = {\varsigma_{\tilde{t}}}^{\mathrm{T}}(x_{\tilde{t},1}, x_{\tilde{t},2}, \ldots, x_{\tilde{t},K}, u_t)$
(8) Update the eligibility: $e_{\tilde{t}}(x) = \{\phi(x_{\tilde{t}}), x_{\tilde{t}} = x; \lambda\gamma e_{\tilde{t}-1}(x), x_{\tilde{t}} \neq x\}$
(9) Compute the TD error: $\delta_{\tilde{t}} = r_{\tilde{t}+1} + \gamma V(x'_{\tilde{t}+1}, \theta_{\tilde{t}}) - V(x_{\tilde{t}}, \theta_{\tilde{t}})$
(10) Update the value function parameter: $\theta_{\tilde{t}+1} = \theta_{\tilde{t}} + \alpha_c \delta_{\tilde{t}} e_{\tilde{t}}(x)$
(11) Update the policy parameter: $\beta_{\tilde{t}+1} = \beta_{\tilde{t}} + \alpha_a \delta_{\tilde{t}} \Delta u_{\tilde{t}} \phi(x_{\tilde{t}})$
(12) **If** $\tilde{t} \leq T$
(13) $\tilde{t} = \tilde{t} + 1$
(14) **End If**
(15) Update the number of samples: number = number + 1
(16) **End Repeat**
(17) **End Loop**
Output: β, θ

Algorithm 4: Global model planning $(\theta, \beta, \eta_1, \ldots, \eta_k, \varsigma, e, \text{number}, P_g)$.

The goal of learning the value function is to minimize the square of the TD-error, which is shown as

$$\min_{\theta} \left\{ \sum_{t=0}^{\text{number}} \left\| r_{t+1} + \gamma V\left(x'_{t+1}, \theta_t\right) - V\left(x_t, \theta_t\right) \right\| + \ell_c \left\| \theta_t \right\|^2 \right\}, \tag{17}$$

where number represents the number of the samples. $\ell_c \geq 0$ is the regularization parameter of the critic. $\|\theta_t\|^2$ is the ℓ_2-regularization which penalizes the growth of the parameter vector θ_t, so that the overfitting to noise samples can be avoided.

The update for the parameter θ of the value function in RAC-HMLP is represented as

$$\theta_{t+1} = \theta_t \left(1 - \frac{\alpha_c}{\text{number}} \ell_c\right) + \frac{\alpha_c}{\text{number}} \sum_{s=0}^{\text{number}} e_s(x) \delta_s. \tag{18}$$

The update for the parameter β of the policy is shown as

$$\beta_{t+1} = \beta_t \left(1 - \frac{\alpha_a}{\text{number}} \ell_a\right) + \frac{\alpha_a}{\text{number}} \sum_{s=0}^{\text{number}} \phi\left(x_s\right) \delta_s \Delta u_s, \tag{19}$$

where $\ell_a \geq 0$ is the regularization parameter of the actor.

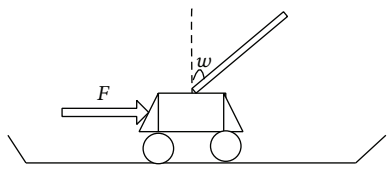


Figure 1: Pole balancing problem.

The update for the global model can be denoted as

$$\begin{aligned}
\eta_{t+1,1} &= \eta_{t,1}\left(1-\frac{\alpha_m}{\text{number}}\ell_m\right)+\frac{\alpha_m}{\text{number}}\\
&\quad\cdot\sum_{s=1}^{\text{number}}\phi\left(x_{s,1},\ldots,x_{s,K},u_s\right)\left(x_{s+1,1}-x'_{s+1,1}\right)\\
\eta_{t+1,2} &= \eta_{t,2}\left(1-\frac{\alpha_m}{\text{number}}\ell_m\right)+\frac{\alpha_m}{\text{number}}\\
&\quad\cdot\sum_{s=1}^{\text{number}}\phi\left(x_{s,1},\ldots,x_{s,K},u_s\right)\left(x_{s+1,2}-x'_{s+1,2}\right)\\
&\vdots\\
\eta_{t+1,K} &= \eta_{t,K}\left(1-\frac{\alpha_m}{\text{number}}\ell_m\right)+\frac{\alpha_m}{\text{number}}\\
&\quad\cdot\sum_{s=1}^{\text{number}}\phi\left(x_{s,1},\ldots,x_{s,K},u_s\right)\left(x_{s+1,K}-x'_{s+1,1}\right)
\end{aligned}\tag{20}$$

$$\begin{aligned}
\varsigma_{t+1} &= \varsigma_t\left(1-\frac{\alpha_m}{\text{number}}\ell_m\right)+\frac{\alpha_m}{\text{number}}\\
&\quad\cdot\sum_{s=1}^{\text{number}}\phi\left(x_{s,1},\ldots,x_{s,K},u_s\right)\left(r_{s+1}-r'_{s+1}\right),
\end{aligned}\tag{21}$$

where $\ell_m \geq 0$ is the regularization parameter for the model, namely, the state transition function and the reward function.

After we replace the update equations of the parameters in Algorithms 2, 3, and 4 with (18), (19), (20), and (21), we will get the resultant algorithm, RAC-HMLP. Except for the above update equations, the other parts of RAC-HMLP are the same with AC-HMLP, so we will not specify here.

5. Empirical Results and Analysis

AC-HMLP and RAC-HMLP are compared with S-AC, MLAC, and Dyna-MLAC on two continuous state and action spaces problems, pole balancing problem [31] and continuous maze problem [32].

5.1. Pole Balancing Problem. Pole balancing problem is a low-dimension but challenging benchmark problem widely used in RL literature, shown in Figure 1.

There is a car moving along the track with a hinged pole on its top. The goal is to find a policy which can guide the force to keep the pole balance. The system dynamics is modeled by the following equation:

$$\ddot{w} = \frac{g\sin(w) - vml\dot{w}\sin(2w)/2 - v\cos(w)F}{4l/3 - vml\cos^2(w)},\tag{22}$$

where w is the angle of the pole with the vertical line. $\dot{w}$ and $\ddot{w}$ are the angular velocity and the angular acceleration of the pole. F is the force exerted on the cart. The negative value means the force to the right and otherwise means to the left. g is the gravity constant with the value $g = 9.81\ \text{m/s}^2$. m and l are the length and the mass of the pole, which are set to $m = 2.0$ kg and $l = 0.5$ m, respectively. v is a constant with the value $1/(m + m_c)$, where $m_c = 8.0$ kg is the mass of the car.

The state $x = (w, \dot{w})$ is composed of w and $\dot{w}$ which are bounded by $[-\pi, \pi]$ and $[-2, 2]$, respectively. The action is the force F bounded by [−50 N, 50 N]. The maximal episode is 300, and the maximal time step for each episode is 3000. If the angle of the pole with the vertical line is not exceeding $\pi/4$ at each time step, the agent will receive a reward 1; otherwise the pole will fall down and receive a reward −1. There is also a random noise applying on the force bounded by [−10 N, 10 N]. An episode ends when the pole has kept balance for 3000 time steps or the pole falls down. The pole will fall down if $|w| \geq \pi/4$. To approximate the value function and the reward function, the state-based feature is coded by radial basis functions (RBFs), shown as

$$\phi_i(x) = e^{-(1/2)(x-c_i)^{\mathrm{T}}B^{-1}(x-c_i)},\tag{23}$$

where c_i denotes the ith center point locating over the grid points $\{-0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6\} \times \{-2, 0, 2\}$. B is a diagonal matrix containing the widths of the RBFs with $\sigma_w^2 = 0.2$ and $\sigma_{\dot{w}}^2 = 2$. The dimensionality z of $\phi(x)$ is 21. Notice that the approximations of the value function and the policy only require the state-based feature. However, the approximation of the model requires the state-action-based feature. Let x be $(w, \dot{w}, u)$; we can also compute its feature by (23). In this case, c_i locates over the points $\{-0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6\} \times \{-2, 0, 2\} \times \{-50, -25, 0, 25, 50\}$. The diagonal elements of B are $\sigma_w^2 = 0.2$, $\sigma_{\dot{w}}^2 = 2$, and $\sigma_F^2 = 10$. The dimensionality z of $\phi(w, \dot{w}, u)$ is 105. Either the state-based or the state-action-based feature has to be normalized so as to make its value be smaller than 1, shown as

$$\overline{\phi_i}(x) = \frac{\phi_i(x)}{\sum_{i=1}^{z}\phi_i(x)},\tag{24}$$

where $z > 0$ is the dimensionality of the feature.

RAC-HMLP and AC-HMLP are compared with S-AC, MLAC, and Dyna-MLAC on this experiment. The parameters of S-AC, MLAC, and Dyna-MLAC are set according to the values mentioned in their papers. The parameters of AC-HMLP and RAC-HMLP are set as shown in Table 1.

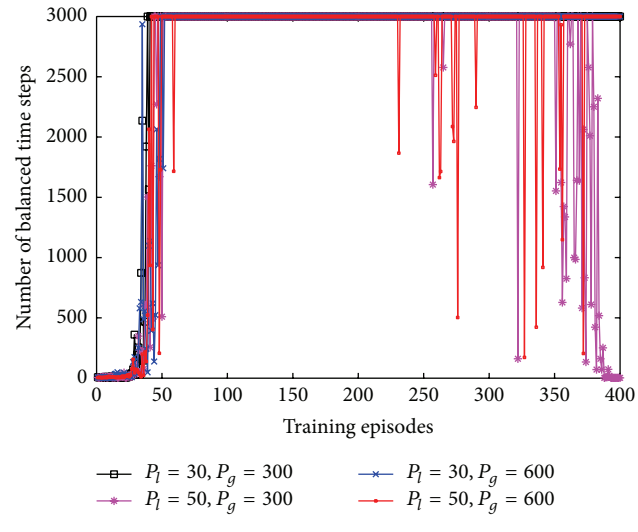
The planning times may have significant effect on the convergence performance. Therefore, we have to determine the local planning times P_l and the global planning times P_g at first. The selective settings for the two parameters are (30, 300), (50, 300), (30, 600), and (50, 600). From Figure 2(a), it

Table 1: Parameters settings of RAC-HMLP and AC-HMLP.

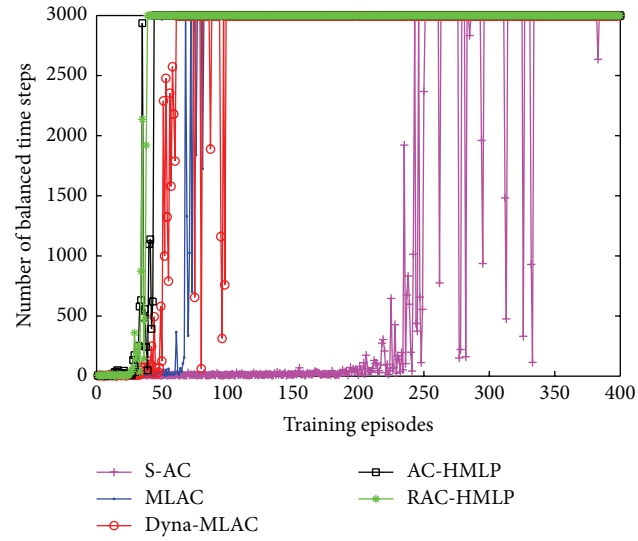
Parameter	Symbol	Value
Time step	T_s	0.1
Discount factor	γ	0.9
Trace-decay rate	λ	0.9
Exploration variance	σ^2	1
Learning rate of the actor	α_a	0.5
Learning rate of the critic	α_c	0.4
Learning rate of the model	α_m	0.5
Error threshold	ξ	0.15
Capacity of the memory	M_size	100
Number of the nearest samples	L	9
Local planning times	P_l	30
Global planning times	P_g	300
Number of components of the state	K	2
Regularization parameter of the model	ℓ_m	0.2
Regularization parameter of the critic	ℓ_c	0.01
Regularization parameter of the actor	ℓ_a	0.001

is easy to find out that ($P_l = 30$, $P_g = 300$) behaves best in these four settings, with 41 episodes for convergence. ($P_l = 30$, $P_g = 600$) learns fastest in the early 29 episodes, but it converges until the 52nd episode. ($P_l = 30$, $P_g = 600$) and ($P_l = 30$, $P_g = 300$) have identical local planning times but different global planning times. Generally, the more the planning times, the faster the convergence rate, but ($P_l = 30$, $P_g = 300$) performs better instead. The global model is not accurate enough at the initial time. Planning through such an inaccurate global model will lead to an unstable performance. Notice that ($P_l = 50$, $P_g = 300$) and ($P_l = 50$, $P_g = 600$) behave poorer than ($P_l = 30$, $P_g = 600$) and ($P_l = 30$, $P_g = 300$), which demonstrates that planning too much via the local model will not perform better. ($P_l = 50$, $P_g = 300$) seems to converge at the 51st episode but its learning curve fluctuates heavily after 365 episodes, not converging any more until the end. Like ($P_l = 50$, $P_g = 300$), ($P_l = 50$, $P_g = 600$) also has heavy fluctuation but converges at the 373rd episode. Evidently, ($P_l = 50$, $P_g = 300$) performs slightly better than ($P_l = 50$, $P_g = 300$). Planning through the global model might solve the nonstability problem caused by planning of the local model.

The convergence performances of the five methods, RAC-HMLP, AC-HMLP, S-AC, MLAC, and Dyna-MLAC, are shown in Figure 2(b). It is evident that our methods RAC-HMLP and AC-HMLP have the best convergence performances. RAC-HMLP and AC-HMLP converge at the 39th and 41st episodes. Both of them learn quickly in the primary phase, but the learning curve of RAC-HMLP seems to be steeper than that of AC-HMLP. Dyna-MLAC learns faster than MLAC and S-AC in the former 74 episodes, but it converges until the 99th episode. Though MLAC behaves poorer than Dyna-MLAC, it requires just 82 episodes to converge. The method with the slowest learning rate is S-AC where the pole can keep balance for 3000 time steps for the first time at the 251st episode. Unfortunately, it converges until the 333rd episode. RAC-HMLP converges fastest



(a) Determination of different planning times



(b) Comparisons of different algorithms in convergence performance

Figure 2: Comparisons of different planning times and different algorithms.

which might be caused by introducing the ℓ_2-regularization. Because the ℓ_2-regularization does not admit the parameter to grow rapidly, the overfitting of the learning process can be avoided effectively. Dyna-MLAC converges faster than MLAC in the early 74 episodes but its performance is not stable enough embodied in the heavy fluctuation from the 75th to 99th episode. If the approximate-local model is distinguished largely from the real-local model, then planning through such an inaccurate local model might lead to an unstable performance. S-AC as the only method without model learning behaves poorest among the five. These results show that the convergence performance can be improved largely by introducing model learning.

The comparisons of the five different algorithms in sample efficiency are shown in Figure 3. It is clear that the numbers of the required samples for S-AC, MLAC, Dyna-MLAC, AC-HMLP, and RAC-HMLP to converge are 56003,

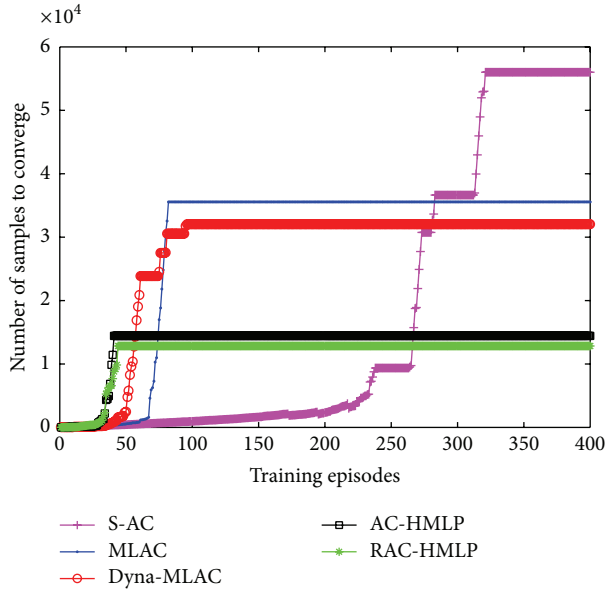


Figure 3: Comparisons of sample efficiency.

35535, 32043, 14426, and 12830, respectively. Evidently, RAC-HMLP converges fastest and behaves stably all the time, thus requiring the least samples. Though AC-HMLP requires more samples to converge than RAC-HMLP, it still requires far fewer samples than the other three methods. Because S-AC does not utilize the model, it requires the most samples among the five. Unlike MLAC, Dyna-MLAC applies the model not only in updating the policy but also in planning, resulting in a fewer requirement for the samples.

The optimal policy and optimal value function learned by AC-HMLP after the training ends are shown in Figures 4(a) and 4(b), respectively, while the ones learned by RAC-HMLP are shown in Figures 4(c) and 4(d). Evidently, the optimal policy and the optimal value function learned by AC-HMLP and RAC-HMLP are quite similar, but RAC-HMLP seems to have more fine-grained optimal policy and value function.

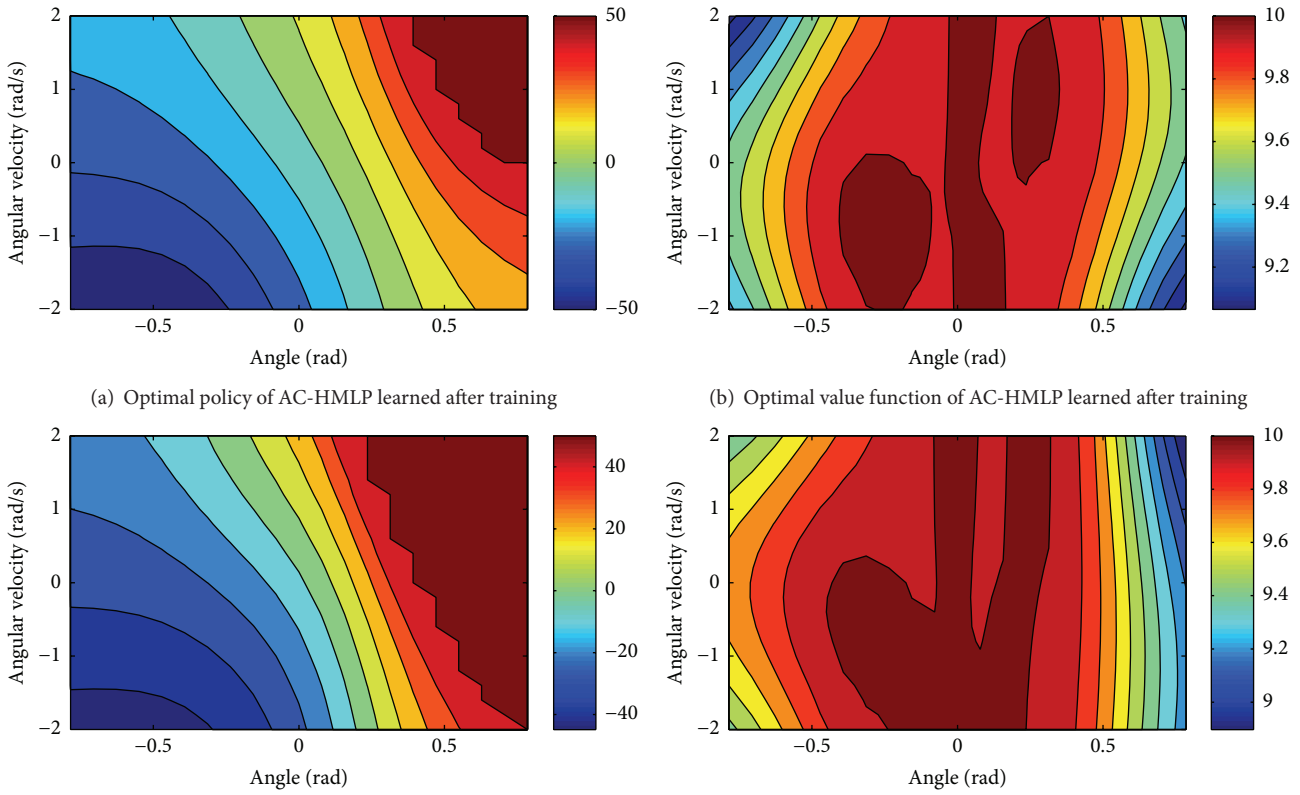
As for the optimal policy (see Figures 4(a) and 4(c)), the force becomes smaller and smaller from the two sides (left side and right side) to the middle. The top-right is the region requiring a force close to 50 N, where the direction of the angle is the same with that of angular velocity. The values of the angle and the angular velocity are nearing the maximum values $\pi/4$ and 2. Therefore, the largest force to the left is required so as to guarantee that the angle between the upright line and the pole is no bigger than $\pi/4$. It is opposite in the bottom-left region where a force attributing to $[-50\,\text{N}, -40\,\text{N}]$ is required to keep the angle from surpassing $-\pi/4$. The pole can keep balance with a gentle force close to 0 in the middle region. The direction of the angle is different from that of angular velocity in the top-left and bottom-right regions; thus a force with the absolute value which is relatively large but smaller than 50 N is required.

In terms of the optimal value function (see Figures 4(b) and 4(d)), the value function reaches a maximum in the region satisfying $-2 \leq w \leq 2$. The pole is more prone to keep balance even without applying any force in this region, resulting in the relatively larger value function. The pole is more and more difficult to keep balance from the middle to the two sides with the value function also decaying gradually. The fine-grained value of the left side compared to the right one might be caused by the more frequent visitation to the left side. More visitation will lead to a more accurate estimation about the value function.

After the training ends, the prediction of the next state and the reward for every state-action pair can be obtained through the learned model. The predictions for the next angle w, the next angular velocity $\dot{w}$, and the reward for any possible state are shown in Figures 5(a), 5(b), and 5(c). It is noticeable that the predicted-next state is always near the current state in Figures 5(a) and 5(b). The received reward is always larger than 0 in Figure 5(c), which illustrates that the pole can always keep balance under the optimal policy.

5.2. Continuous Maze Problem. Continuous maze problem is shown in Figure 6, where the blue lines with coordinates represent the barriers. The state (x, y) consists of the horizontal coordinate $x \in [0, 1]$ and vertical coordinate $y \in [0, 1]$. Starting from the position "Start," the goal of the agent is to reach the position "Goal" that satisfies $x + y > 1.8$. The action is to choose an angle bounded by $[-\pi, \pi]$ as the new direction and then walk along this direction with a step 0.1. The agent will receive a reward -1 at each time step. The agent will be punished with a reward -400 multiplying the distance if the distance between its position and the barrier exceeds 0.1. The parameters are set the same as the former problem, except for the planning times. The local and global planning times are determined as 10 and 50 in the same way of the former experiment. The state-based feature is computed according to (23) and (24) with the dimensionality $z = 16$. The center point c_i locates over the grid points $\{0.2, 0.4, 0.6, 0.8\} \times \{0.2, 0.4, 0.6, 0.8\}$. The diagonal elements of B are $\sigma_x^2 = 0.2$ and $\sigma_y^2 = 0.2$. The state-action-based feature is also computed according to (23) and (24). The center point c_i locates over the grid points $\{0.2, 0.4, 0.6, 0.8\} \times \{0.2, 0.4, 0.6, 0.8\} \times \{-3, -1.5, 0, 1.5, 3\}$ with the dimensionality $z = 80$. The diagonal elements of B are $\sigma_x^2 = 0.2$, $\sigma_y^2 = 0.2$, and $\sigma_u^2 = 1.5$.

RAC-HMLP and AC-HMLP are compared with S-AC, MLAC, and Dyna-MLAC in cumulative rewards. The results are shown in Figure 7(a). RAC-HMLP learns fastest in the early 15 episodes, MLAC behaves second best, and AC-HMLP performs poorest. RAC-HMLP tends to converge at the 24th episode, but it really converges until the 39th episode. AC-HMLP behaves steadily starting from the 23rd episode to the end and it converges at the 43rd episode. Like the former experiment, RAC-HMLP performs slightly better than AC-HMLP embodied in the cumulative rewards -44 compared to -46. At the 53rd episode, the cumulative rewards of S-AC fall to about -545 quickly without any ascending thereafter. Though MLAC and Dyna-MLAC perform well in the primary phase, their curves start to descend at the 88th episode and the 93rd episode, respectively. MLAC and Dyna-MLAC learn the local model to update the policy gradient, resulting in a fast learning rate in the primary phase.



(a) Optimal policy of AC-HMLP learned after training

(b) Optimal value function of AC-HMLP learned after training

(c) Optimal policy of RAC-HMLP learned after training

(d) Optimal value function of RAC-HMLP learned after training

Figure 4: Optimal policy and value function learned by AC-HMLP and RAC-HMLP.

However, they do not behave stably enough near the end of the training. Too much visitation to the barrier region might cause the fast descending of the cumulative rewards.

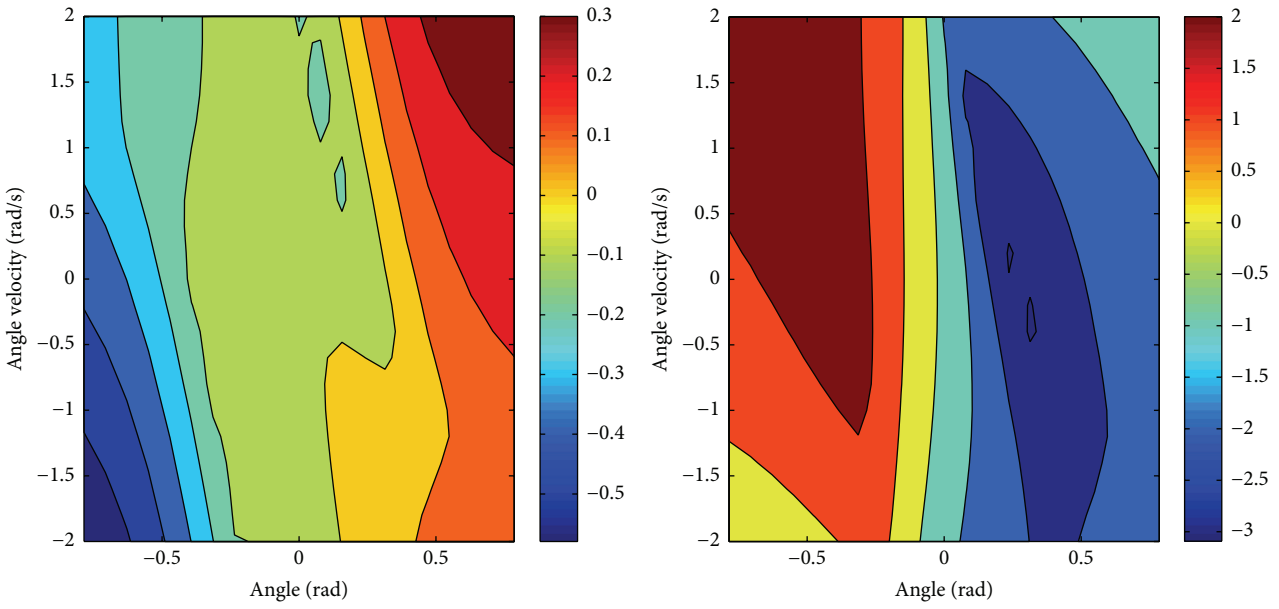
The comparisons of the time steps for reaching goal are simulated, which are shown in Figure 7(b). Obviously, RAC-HMLP and AC-HMLP perform better than the other three methods. RAC-HMLP converges at 39th episode while AC-HMLP converges at 43rd episode, with the time steps for reaching goal being 45 and 47. It is clear that RAC-HMLP still performs slightly better than AC-HMLP. The time steps for reaching goal are 548, 69, and 201 for S-AC, MLAC, and Dyna-MLAC. Among the five methods, RAC-HMLP not only converges fastest but also has the best solution, 45. The poorest performance of S-AC illustrates that model learning can definitely improve the convergence performance. As in the former experiment, Dyna-MLAC behaves poorer than MLAC during training. If the model is inaccurate, planning via such model might influence the estimations of the value function and the policy, thus leading to a poor performance in Dyna-MLAC.

The comparisons of sample efficiency are shown in Figure 8. The required samples for S-AC, MLAC, Dyna-MLAC, AC-HMLP, and RAC-HMLP to converge are 10595, 6588, 7694, 4388, and 4062, respectively. As in pole balancing problem, RAC-HMLP also requires the least samples while S-AC needs the most to converge. The difference is that Dyna-MLAC requires samples slightly more than MLAC. The ascending curve of Dyna-MLAC at the end of the training demonstrates that it has not converged. The frequent visitation to the barrier in Dyna-MLAC leads to the quick descending of the value functions. As a result, enormous samples are required to make these value functions more prone to the true ones.

After the training ends, the approximate optimal policy and value function are obtained, shown in Figures 9(a) and 9(b). It is noticeable that the low part of the figure is explored thoroughly, so that the policy and the value function are very distinctive in this region. For example, in most of the region in Figure 9(a), a larger angle is needed for the agent so as to leave the current state. Clearly, the nearer the current state and the low part of Figure 9(b), the smaller the corresponding value function. The top part of the figures may be not frequently or even not visited by the agent, resulting in the similar value functions. The agent is able to reach the goal only with a gentle angle in these areas.

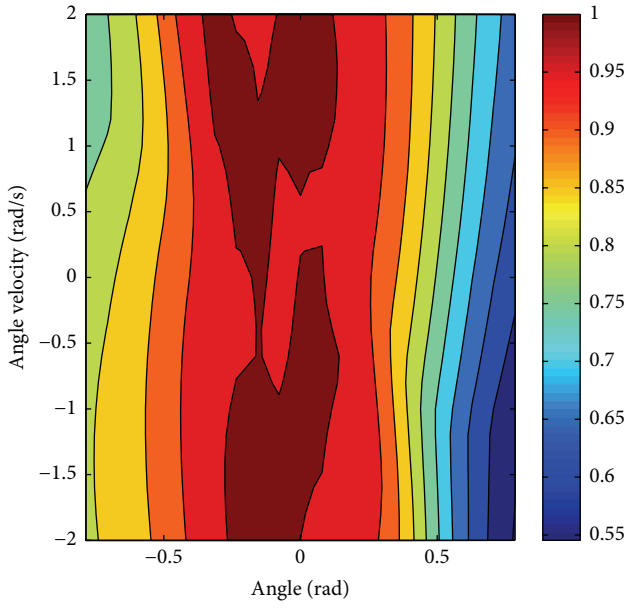
6. Conclusion and Discussion

This paper proposes two novel actor-critic algorithms, AC-HMLP and RAC-HMLP. Both of them take LLR and LFA to represent the local model and global model, respectively. It has been shown that our new methods are able to learn the



(a) Prediction of the angle at next state

(b) Prediction of the angular velocity at next state



(c) Prediction of the reward

Figure 5: Prediction of the next state and reward according to the global model.

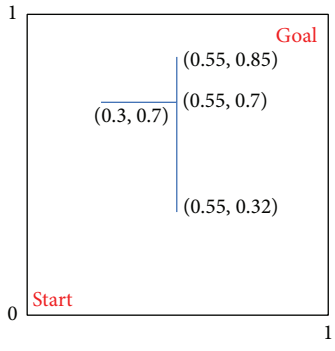
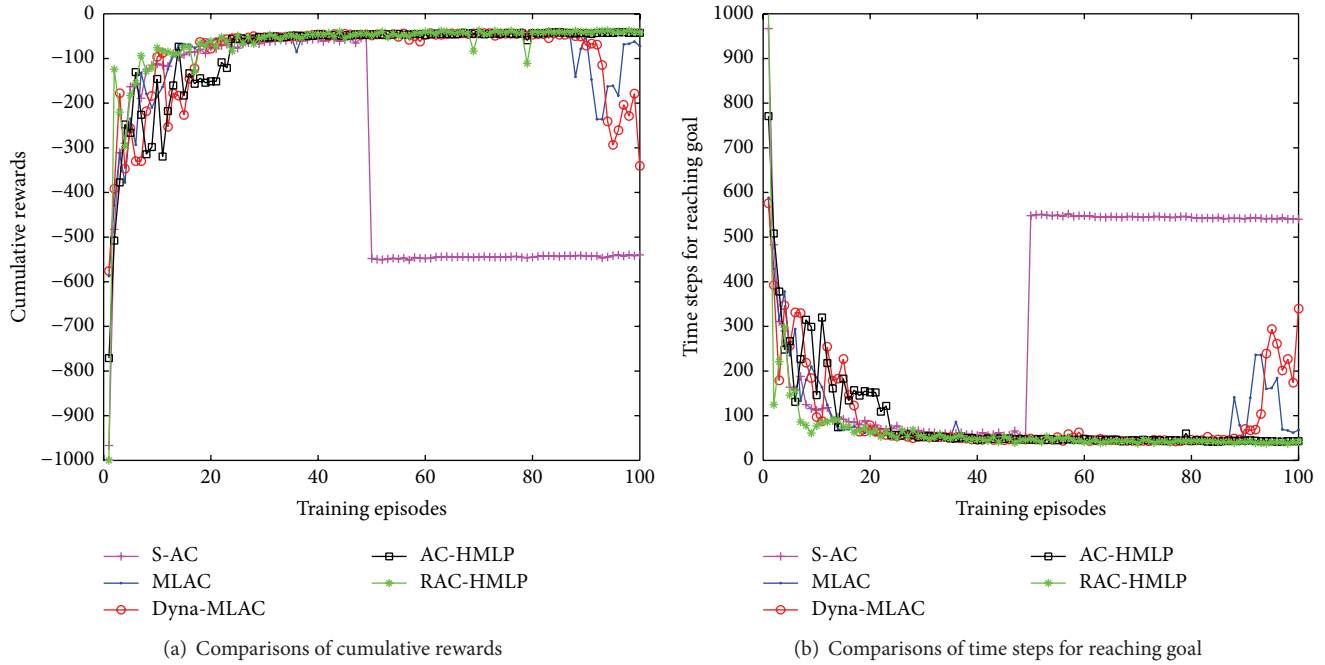


Figure 6: Continuous maze problem.

optimal value function and the optimal policy. In the pole balancing and continuous maze problems, RAC-HMLP and AC-HMLP are compared with three representative methods. The results show that RAC-HMLP and AC-HMLP not only converge fastest but also have the best sample efficiency.

RAC-HMLP performs slightly better than AC-HMLP in convergence rate and sample efficiency. By introducing ℓ_2-regularization, the parameters learned by RAC-HMLP will be smaller and more uniform than those of AC-HMLP, so that the overfitting can be avoided effectively. Though RAC-HMLP behaves better, its improvement over AC-HMLP is not significant. Because AC-HMLP normalizes all the features to



(a) Comparisons of cumulative rewards

(b) Comparisons of time steps for reaching goal

Figure 7: Comparisons of cumulative rewards and time steps for reaching goal.

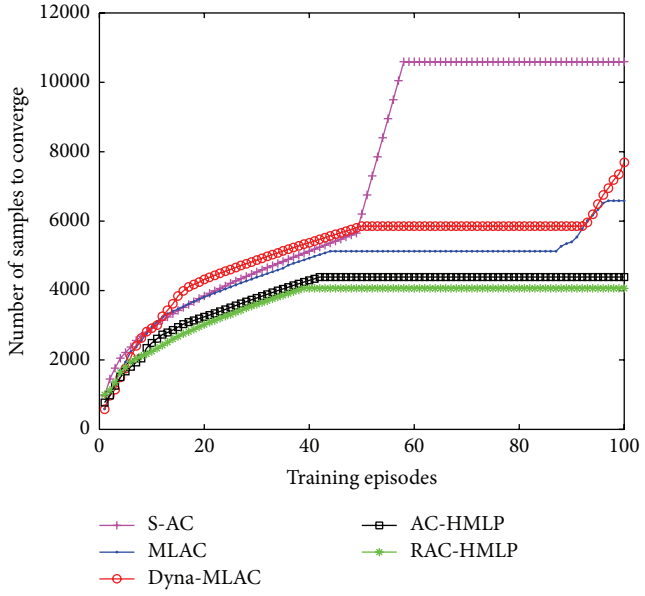


Figure 8: Comparisons of different algorithms in sample efficiency.

[0, 1], the parameters will not change heavily. As a result, the overfitting can also be prohibited to a certain extent.

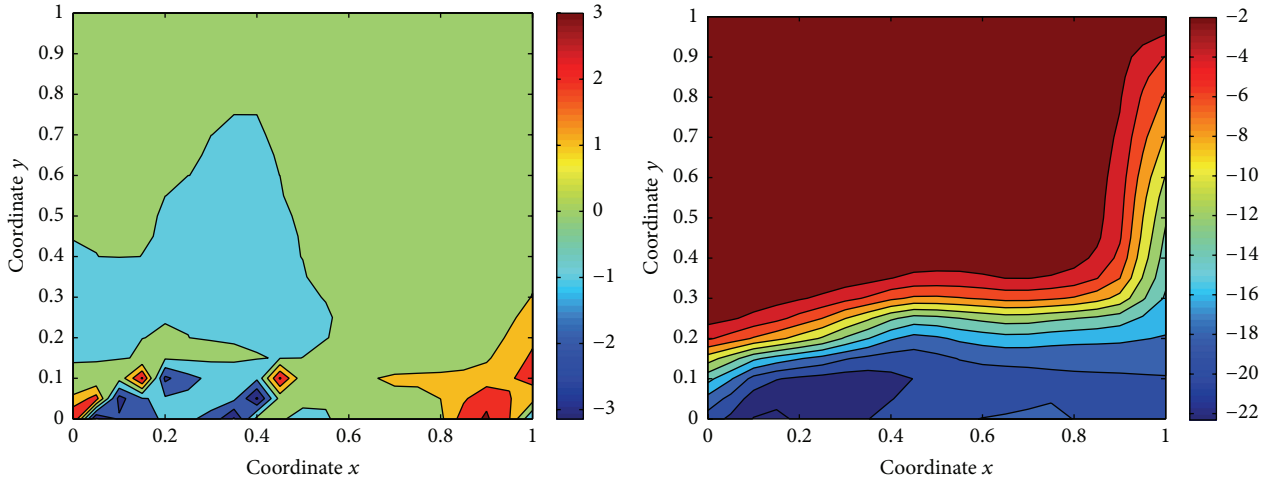
S-AC is the only algorithm without model learning. The poorest performance of S-AC demonstrates that combining model learning and AC can really improve the performance. Dyna-MLAC learns a model via LLR for local planning and policy updating. However, Dyna-MLAC directly utilizes the model before making sure whether it is accurate; additionally, it does not utilize the global information about the samples. Therefore, it behaves poorer than AC-HMLP and RAC-HMLP. MLAC also approximates a local model via LLR as Dyna-MLAC does, but it only takes the model to update the policy gradient, surprisingly with a slightly better performance than Dyna-MLAC.

Dyna-MLAC and MLAC approximate the value function, the policy, and the model through LLR. In the LLR approach, the samples collected in the interaction with the environment have to be stored in the memory. Through KNN or K-d tree, only L-nearest samples in the memory are selected to learn the LLR. Such a learning process takes a lot of computation and memory costs. In AC-HMLP and RAC-HMLP, there is only a parameter vector to be stored and learned for any of the value function, the policy, and the global model. Therefore, AC-HMLP and RAC-HMLP outperform Dyna-MLAC and MLAC also in computation and memory costs.

The planning times for the local model and the global model have to be determined according the experimental performance. Thus, we have to set the planning times according to the different domains. To address this problem, our future work will consider how to determine the planning times adaptively according to the different domains. Moreover, with the development of the deep learning [33, 34], deep RL has succeeded in many applications such as AlphaGo and Atari 2600 by taking the visual images or videos as input. However, how to accelerate the learning process in deep RL through model learning is still an open question. The future work will endeavor to combine deep RL with hierarchical model learning and planning to solve the real-world problems.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this article.



(a) Optimal policy learned by AC-HMLP

(b) Optimal value function learned by AC-HMLP

Figure 9: Final optimal policy and value function after training.

Acknowledgments

This research was partially supported by Innovation Center of Novel Software Technology and Industrialization, National Natural Science Foundation of China (61502323, 61502329, 61272005, 61303108, 61373094, and 61472262), Natural Science Foundation of Jiangsu (BK2012616), High School Natural Foundation of Jiangsu (13KJB520020), Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University (93K172014K04), and Suzhou Industrial Application of Basic Research Program Part (SYG201422).

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass, USA, 1998.

[2] M. L. Littman, "Reinforcement learning improves behaviour from evaluative feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[3] D. Silver, R. S. Sutton, and M. Müller, "Temporal-difference search in computer Go," *Machine Learning*, vol. 87, no. 2, pp. 183–219, 2012.

[4] J. Kober and J. Peters, "Reinforcement learning in robotics: a survey," in *Reinforcement Learning*, pp. 579–610, Springer, Berlin, Germany, 2012.

[5] D.-R. Liu, H.-L. Li, and D. Wang, "Feature selection and feature learning for high-dimensional batch reinforcement learning: a survey," *International Journal of Automation and Computing*, vol. 12, no. 3, pp. 229–242, 2015.

[6] R. E. Bellman, *Dynamic Programming*, Princeton University Press,, Princeton, NJ, USA, 1957.

[7] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, Mass, USA, 1995.

[8] F. L. Lewis and D. R. Liu, *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, John Wiley & Sons, Hoboken, NJ, USA, 2012.

[9] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems*, vol. 22, pp. 25–38, 1977.

[10] D. R. Liu, H. L. Li, and D. Wang, "Data-based self-learning optimal control: research progress and prospects," *Acta Automatica Sinica*, vol. 39, no. 11, pp. 1858–1870, 2013.

[11] H. G. Zhang, D. R. Liu, Y. H. Luo, and D. Wang, *Adaptive Dynamic Programming for Control: Algorithms and Stability*, Communications and Control Engineering Series, Springer, London, UK, 2013.

[12] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: convergence proof," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 4, pp. 943–949, 2008.

[13] T. Dierks, B. T. Thumati, and S. Jagannathan, "Optimal control of unknown affine nonlinear discrete-time systems using offline-trained neural networks with proof of convergence," *Neural Networks*, vol. 22, no. 5-6, pp. 851–860, 2009.

[14] F.-Y. Wang, N. Jin, D. Liu, and Q. Wei, "Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with ε-error bound," *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 24–36, 2011.

[15] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[16] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 32, no. 2, pp. 140–153, 2002.

[17] T. Hanselmann, L. Noakes, and A. Zaknich, "Continuous-time adaptive critics," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 631–647, 2007.

[18] S. Bhasin, R. Kamalapurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, "A novel actor-critic-identifier architecture for approximate optimal control of uncertain nonlinear systems," *Automatica*, vol. 49, no. 1, pp. 82–92, 2013.

[19] L. Busoniu, R. Babuka, B. De Schutter et al., *Reinforcement Learning and Dynamic Programming Using Function Approximators*, CRC Press, New York, NY, USA, 2010.

[20] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[21] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL '07)*, pp. 272–279, Honolulu, Hawaii, USA, April 2007.

[22] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[23] I. Grondman, L. Buşoniu, G. A. D. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1291–1307, 2012.

[24] I. Grondman, L. Busoniu, and R. Babuska, "Model learning actor-critic algorithms: performance evaluation in a motion control task," in *Proceedings of the 51st IEEE Conference on Decision and Control (CDC '12)*, pp. 5272–5277, Maui, Hawaii, USA, December 2012.

[25] I. Grondman, M. Vaandrager, L. Buşoniu, R. Babuška, and E. Schuitema, "Efficient model learning methods for actor-critic control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 3, pp. 591–602, 2012.

[26] B. Costa, W. Caarls, and D. S. Menasche, "Dyna-MLAC: trading computational and sample complexities in actor-critic reinforcement learning," in *Proceedings of the Brazilian Conference on Intelligent Systems (BRACIS '15)*, pp. 37–42, Natal, Brazil, November 2015.

[27] O. Andersson, F. Heintz, and P. Doherty, "Model-based reinforcement learning in continuous environments using real-time constrained optimization," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI '15)*, pp. 644–650, Texas, Tex, USA, January 2015.

[28] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI '08)*, pp. 528–536, July 2008.

[29] R. Faulkner and D. Precup, "Dyna planning using a feature based generative model," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS '10)*, pp. 1–9, Vancouver, Canada, December 2010.

[30] H. Yao and C. Szepesvári, "Approximate policy iteration with linear action models," in *Proceedings of the 26th National Conference on Artificial Intelligence (AAAI '12)*, Toronto, Canada, July 2012.

[31] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 724–740, 1992.

[32] R. S. Sutton, "Generalization in reinforcement learning: successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, pp. 1038–1044, MIT Press, New York, NY, USA, 1996.

[33] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[34] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.