Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 760459, 14 pages
http://dx.doi.org/10.1155/2015/760459

Research Article

Two-Phase Iteration for Value Function Approximation and Hyperparameter Optimization in Gaussian-Kernel-Based Adaptive Critic Design

Xin Chen,[1] Penghuan Xie,[2] Yonghua Xiong,[1] Yong He,[1] and Min Wu[1]

[1]*School of Automation, China University of Geosciences, Wuhan, Hubei 430074, China*
[2]*School of Information Science and Engineering, Central South University, Changsha, Hunan 410083, China*

Correspondence should be addressed to Yonghua Xiong; xiongyh@cug.edu.cn

Received 7 January 2015; Accepted 26 May 2015

Academic Editor: Simon X. Yang

Adaptive Dynamic Programming (ADP) with critic-actor architecture is an effective way to perform online learning control. To avoid the subjectivity in the design of a neural network that serves as a critic network, kernel-based adaptive critic design (ACD) was developed recently. There are two essential issues for a static kernel-based model: how to determine proper hyperparameters in advance and how to select right samples to describe the value function. They all rely on the assessment of sample values. Based on the theoretical analysis, this paper presents a two-phase simultaneous learning method for a Gaussian-kernel-based critic network. It is able to estimate the values of samples without infinitively revisiting them. And the hyperparameters of the kernel model are optimized simultaneously. Based on the estimated sample values, the sample set can be refined by adding alternatives or deleting redundances. Combining this critic design with actor network, we present a Gaussian-kernel-based Adaptive Dynamic Programming (GK-ADP) approach. Simulations are used to verify its feasibility, particularly the necessity of two-phase learning, the convergence characteristics, and the improvement of the system performance by using a varying sample set.

1. Introduction

Reinforcement learning (RL) is an interactive machine learning method for solving sequential decision problems. It is well known as an important learning method in unknown or dynamic environment. Different from supervised learning and unsupervised learning, RL interacts with the environment through trial mechanism and modifies its action policies to maximize the payoffs [1]. It is strongly connected from a theoretical point of view with direct and indirect adaptive optimal control methods [2].

Traditional RL research focused on discrete state/action systems; state/action only takes on a finite number of prescribed discrete values. The learning space grows exponentially as the number of states and the number of allowed actions increase. This leads to the so-called curse of dimensionality (CoD) [2]. In order to mitigate this CoD problem, function approximations and generalization methods [3] are introduced to store the optimal value and the optimal control as a function of the state vector. Generalization methods based on parametric model such as neural networks [4–6] have become one of popular means to solve RL problem in continuous environments.

Currently, research work on RL in continuous environment to construct learning systems for nonlinear optimal control has attracted attention of researchers and scholars in control domains for the reason that it can modify its policy only based on the value function without knowing the model structure or the parameters in advance. A family of new RL techniques known as Approximate or Adaptive Dynamic Programming (ADP) (also known as Neurodynamic Programming or Adaptive Critic Designs (ACDs)) has received more and more research interest [7, 8]. ADPs are based on the actor-critic structure, in which there is a critic assessing the value of the action or control policy applied by an actor and an actor modifying its action based on the assessment of values. In literatures, ADP approaches are categorized as the following main schemes: heuristic dynamic programming (HDP), dual heuristic programming (DHP), globalized dual heuristic programming (GDHP), and their action-dependent versions [9, 10].

ADP researches always adopt multilayer perceptron neural networks (MLPNNs) as the critic design. Vrabie and Lewis proposed an online approach to continuous-time direct adaptive optimal control which made use of neural networks to parametrically represent the control policy and the performance of the control system [11]. Liu et al. solved the constrained optimal control problem of unknown discrete-time nonlinear systems based on the iterative ADP algorithm via GDHP technique with three neural networks [12]. In fact, different kinds of neural networks (NNs) play the important roles in ADP algorithms, such as radial basis function NNs [3], wavelet basis function NNs [13], and echo state network [14].

Besides the benefits brought by NNs, ADP methods always suffer from some problems concerned in the design of NNs. On one hand, the learning control performance greatly depends on empirical design of critic networks, especially the manual setting of the hidden layer or the basis functions. On the other hand, due to the local minima in neural network training, how to improve the quality of the final policies is still an open problem [15].

As we can see, it is difficult to evaluate the effectiveness of the parametric model when the knowledge on the model's order or nonlinear characteristics of the system is not enough. Compared with parametric modeling methods, nonparametric modeling methods, especially kernel methods [16, 17], do not need to set the model structure in advance. Hence, kernel machines have been popularly studied to realize nonlinear and nonparametric modeling. Engel et al. proposed the kernel recursive least-squares algorithm to construct minimum mean-squared-error solutions to nonlinear least-squares problems [18]. As popular kernel machines, support vector machines (SVMs) also have been applied to nonparametric modeling problems. Dietterich and Wang combined linear programming with SVMs to find value function approximations [19]. The similar research was published in [20], in which the least-squares SVMs were used. Nevertheless, they both focused on discrete state/action space and lacked theoretical results on the policies obtained more or less.

In addition to SVMs, Gaussian processes (GPs) have become an alternative generalization method. GP models are powerful nonparametric tools for approximate Bayesian inference and learning. In comparison with other popular nonlinear architectures, such as multilayer perceptrons, their behavior is conceptually simpler to understand, and model fitting can be achieved without resorting to nonconvex optimization routines [21, 22]. In [23], Engel et al. first applied GPs in temporal-difference (TD) learning for MDPs with stochastic rewards and deterministic transitions. They derived a new GPTD algorithm in [24] that overcame the limitation of deterministic transitions and extended GPTD to the estimation of state-action values. GPTD algorithm just addressed the value approximation, so it should be combined with actor-critic methods or other policy iteration methods to solve learning control problems.

An alternative approach employing GPs in RL is model-based value iteration or policy iteration method, in which GP model is used to model system dynamics and represent the value function [25]. In [26], an approximated value-function based RL algorithm named Gaussian process dynamic programming (GPDP) was presented, which built dynamic transition model, value function model, and action policy model, respectively, using GPs. In this way, the sample set will be adjusted to such a reasonable shape with high sample densities near the equilibriums or the places where value functions change dramatically. Thus it is good at controlling nonlinear systems with complex dynamics. A major shortcoming, even if the relatively high computation cost is endurable, is that the states in sample set need to be revisited again and again in order to update their value functions. Since this condition is unpractical in real implements, it diminishes the appeal of employing this method.

Kernel-based method is also introduced to ADP. In [15], a novel framework of ACDs with sparse kernel machines was presented by integrating kernel methods into critic network. A sparsification method based on the approximately linear dependence (ALD) analysis was used to sparsify the kernel machines. Obviously, this method can overcome the difficulty of presetting model structure in parametric models and realize actor-critic learning online [27]. However, the selection of samples based on the ALD analysis is an offline way without considering the distribution of the value function. Therefore, the data samples cannot be adjusted online, which makes the method more suitable for control systems with smooth dynamics, where value function changes gently.

We think GPDP and ACDs with sparse kernel machines are complementary. As indicated in [28], it is known the prediction of GPs is viewed as a linear combination of the covariance between the new points and the samples. Hence it seems reasonable to introduce kernel machine with GPs to build critic network in ACDs, if the values of the samples are known or at least can be assessed numerically. And then the sample set will be adjusted online during critic-actor learning.

The major problem here is how to realize the value function learning and GP models updating simultaneously, especially under the condition that the samples of state-action space can hardly be revisited infinitely in order to approximate their values. To tackle this problem, a two-phase iteration is developed in order to get optimal control policy for the system whose dynamics are unknown a priori.

2. Description of the Problem

In general, ADP is an actor-critic method which approximates the value functions and policies to encourage the realization of generalization in MDPs with large or continuous spaces. The critic design plays the most important role, because it determines how the actor optimizes its action. Hence, we give a brief introduction on both kernel-based ACD and GPs, in order to derive the clear description of the theoretical problem.

2.1. Kernel-Based ACDs. Kernel-based ACDs mainly consist of a critic network, a kernel-based feature learning module, a reward function, an actor network/controller, and a model of the plant. The critic constructed by kernel machine is used to approximate the value functions or their derivatives. Then the output of the critic is used in the training process of the actor so that policy gradients can be computed. As actor finally converges, the optimal action policy mapping states to actions is described by this actor.

Traditional neural network based on kernel machine and samples serves as the model of value functions, just as the following equation shows, and the recursive algorithm, such as KLSTD [29] serves as value function approximation:

$$Q(x) = \sum_{i=1}^{L} \alpha_i k(x, x_i), \tag{1}$$

where x and x_i represent state-action pairs (s, a) and (s_i, a_i), α_i, $i = 1, 2, \ldots, L$, are the weights, and (s_i, a_i) represents selected state-action pairs in sample set and $k(x, x_i)$ is a kernel function.

The key of critic learning is the update of the weights vector α. The value function modeling in continuous state/action space is a regression problem. From the view of the model, NN structure is a reproducing kernel space spanned by the samples, in which the value function is expressed as a linear regression function. Obviously as the basis of Hilbert space, how to select samples determines the VC dimension of identification, as well as the performance of value function approximation.

If only using ALD-based kernel sparsification, it is only independent of samples that are considered in sample selection. So it is hard to evaluate how good the sample set is, because the sample selection does not consider the distribution of value function, and the performance of ALD analysis is affected seriously by the hyperparameters of kernel function, which are predetermined empirically and fixed during learning.

If the hyperparameters can be optimized online and the value function w.r.t. samples can be evaluated by iteration algorithms, the critic network will be optimized not only by value approximation but also by hyperparameter optimization. Moreover with approximated sample values, there is a direct way to evaluate the validity of sample set, in order to regulate the set online. Thus in this paper we turn to Gaussian processes to construct the criterion for samples and hyperparameters.

2.2. GP-Based Value Function Model. For an MDP, the data samples x_i and the corresponding Q value can be collected by observing the MDP. Here x_i is the state-action pairs (s_i, a_i), $s \in R^N$, $a \in R^M$, and $Q(x)$ is the value function defined as

$$Q^*(x) = R(x) + \beta \int p(s' \mid x) V(s') \, ds', \tag{2}$$

where $V(s) = \max_a Q(x)$.

Given a sample set collected from a continuous dynamic system, $\{X_L, y\}$, where $y = Q(X_L) + \varepsilon$, $\varepsilon \sim N(0, v_0)$, Gaussian regression with covariance function shown in the following equation is a well known model technology to infer the Q function:

$$k(x_p, x_q) = v_1 \exp\left[-\frac{1}{2}(x_p - x_q)^T \Lambda^{-1} (x_p - x_q)\right], \tag{3}$$

where $\Lambda = \operatorname{diag}([w_1, w_2, \ldots, w_{N+M}])$.

Assuming additive independent identically distributed Gaussian noise ε with variance v_0, the prior on the noisy observations becomes

$$K_L = K(X_L, X_L) + v_0 I_L, \tag{4}$$

where

$$K(X_L, X_L) = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_L) \\ k(x_2, x_1) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ k(x_L, x_1) & \cdots & \cdots & k(x_L, x_L) \end{bmatrix}. \tag{5}$$

The parameters w_d, v_0, and v_1 are the hyperparameters of the GR_q and collected within the vector $\theta = [w_1 \ \cdots \ w_{N+M} \ v_1 \ v_0]^T$.

For an arbitrary input x_*, the predictive distribution of the function value is Gaussian distributed with mean and variance given by

$$E[Q(x_*)] = K(x_*, X_L) K_L^{-1} y, \tag{6}$$

$$\operatorname{var}[Q(x)] = k(x_*, x_*) - K(x_*, X_L) K_L^{-1} K(X_L, x_*). \tag{7}$$

Comparing (6) to (1), we find that if we let $\alpha = K_L^{-1} y$, the neural network is also regarded as the Gaussian regression. Or the critic network can be constructed based on Gaussian kernel machine, if the following conditions are satisfied.

Condition 1. The hyperparameters θ of Gaussian kernel are known.

Condition 2. The values y w.r.t. all samples states X_L are known.

With Gaussian-kernel-based critic network, the sample state-action pairs and corresponding values are known. And then the criterion such as the comprehensive utility proposed in [26] can be set up, in order to refine sample set online. At the same time it is convenient to optimize hyperparameters, in order to approximate Q value function more accurately. Thus besides the advantages brought by kernel machine, the critic based on Gaussian-kernel will be better in approximating value functions.

Consider Condition 1. If the values y w.r.t. X_L are known (note that it is indeed Condition 2), the common way to get hyperparameters is by evidence maximization, where the log-evidence is given by

$$L(\theta) = -\frac{1}{2} y^T K_L^{-1} y - \frac{1}{2} \log |K_L| - \frac{n}{2} \log 2\pi. \tag{8}$$

It requires the calculation of the derivative of $L(\theta)$ w.r.t. each θ_i, given by

$$\frac{\partial L(\theta)}{\partial \theta_i} = -\frac{1}{2}\operatorname{tr}\left[K_L^{-1}\frac{\partial K_L}{\partial \theta_i}\right] + \frac{1}{2}\alpha^T \frac{\partial K_L}{\partial \theta_i}\alpha, \tag{9}$$

where $\operatorname{tr}[\cdot]$ denotes the trace, $\alpha = K_L^{-1} y$.

Consider Condition 2. For unknown system, if K_L is known, that is, the hyperparameters are known (note that it is indeed Condition 1), the update of critic network will be transferred to the update of values w.r.t. samples by using value iteration.

According to the analysis, both conditions are interdependent. That means the update of critic depends on known hyperparameters, and the optimization of hyperparameters depends on accurate sample values.

Hence we need a comprehensive iteration method to realize value approximation and optimization of hyperparameters simultaneously. A direct way is to update them alternately. Unfortunately, this way is not reasonable because these two processes are tightly coupled. For example, temporal differential errors drive value approximation, but the change of weights α will cause the change of hyperparameters θ simultaneously; then it is difficult to tell whether this temporal differential error is induced by observation or by Gaussian regression model changing.

To solve this problem, a kind of two-phase value iteration for critic network is presented in the next section, and the conditions of convergence are analyzed.

3. Two-Phase Value Iteration for Critic Network Approximation

First a proposition is given to describe the relationship between hyperparameters and the sample value function.

Proposition 1. *The hyperparameters are optimized by evidence maximization according to the samples and their Q values, and the log-evidence is given by*

$$\frac{\partial L(\theta)}{\partial \theta_i} = -\frac{1}{2}\operatorname{tr}\left[K_L^{-1}\frac{\partial K_L}{\partial \theta_i}\right] + \frac{1}{2}\alpha^T \frac{\partial K_L}{\partial \theta_i}\alpha^T = 0, \quad i = 1, \ldots, D, \tag{10}$$

where $\alpha = K_L^{-1} y$. It can be proved that, for arbitrary hyperparameters θ, if $\alpha \neq 0$, (10) defines an implicit function or a continuously differentiable function as follows:

$$\theta = f(\alpha) = \left[f_1(\alpha) \;\; f_2(\alpha) \;\; \cdots \;\; f_D(\alpha)\right]^T. \tag{11}$$

Then the two-phase value iteration for critic network is described as the following theorem.

Theorem 2. *Given the following conditions*

(1) *the system is boundary input and boundary output (BIBO) stable,*

(2) *the immediate reward r is bounded,*

(3) *for $k = 1, 2$, $\eta_t^k \to 0$, $\sum_{t=1}^{\infty} \eta_t^k = \infty$,*

the following iteration process is convergent:

$$\begin{aligned}
\alpha_{t+1} &= \alpha_t, \\
\theta_{t+1} &= \theta_t + \eta_t^1 \left(f(\alpha_t) - \theta_t\right), \\
\alpha_{t+2} &= \alpha_{t+1} + \eta_t^2 k_{\theta_t}^{\aleph}(x, X_L) \\
&\quad \cdot \left[r + \beta V\left(\theta_t, x'\right) - k_{\theta_t}(x, X_L)\alpha_{t+1}\right], \\
\theta_{t+2} &= \theta_{t+1},
\end{aligned} \tag{12}$$

where $V(\theta_t, x') = \max_a k_{\theta_t}(x', X_L)\alpha(t)$ and $k_{\theta_t}^{\aleph}(x, X_L)$ is a kind of pseudoinversion of $k_{\theta_t}(x, X_L)$; that is, $k_{\theta_t}(x, X_L) k_{\theta_t}^{\aleph}(x, X_L) = 1$.

Proof. From (12), it is clear that the two phases include the update of hyperparameters in phase 1, which is viewed as the update of generalization model, and the update of samples' value in phase 2, which is viewed as the update of critic network.

The convergence of iterative algorithm is proved based on stochastic approximation Lyapunov method.

Define that

$$\begin{aligned}
P_t k_{\theta_t}(x, X_L)\alpha_t &= r + \beta V\left(\theta_t, x'\right) \\
&= r + \max_a k_{\theta_t}\left(x', X_L\right)\alpha(t).
\end{aligned} \tag{13}$$

Equation (12) is rewritten as

$$\begin{aligned}
\alpha_{t+1} &= \alpha_t, \\
\theta_{t+1} &= \theta_t + \eta_t^1 \left(f(\alpha_t) - \theta_t\right), \\
\alpha_{t+2} &= \alpha_{t+1} + \eta_t^2 k_{\theta_t}^{\aleph}(x, X_L) \\
&\quad \cdot \left[P_{t+1} k_{\theta_t}(x, X_L)\alpha_{t+1} - k_{\theta_t}(x, X_L)\alpha_{t+1}\right], \\
\theta_{t+2} &= \theta_{t+1}.
\end{aligned} \tag{14}$$

Define approximation errors as

$$\begin{aligned}
\varphi_{t+2}^1 &= \varphi_{t+1}^1 = \theta_{t+1} - f(\alpha_{t+1}) = \theta_t - f(\alpha_{t+1}) \\
&\quad + \eta_t^1 \left[f(\alpha_t) - \theta_t\right] = \left(1 - \eta_t^1\right)\varphi_t^1,
\end{aligned} \tag{15}$$

$$\begin{aligned}
\varphi_{t+2}^2 &= (\alpha_{t+2} - \alpha^*) = (\alpha_{t+1} - \alpha^*) + \eta_t^2 k_{\theta_t}^{\aleph}(x, X_L) \\
&\quad \cdot \left[P_{t+1} k_{\theta_t}(x, X_L)\alpha_{t+1}\right. \\
&\quad - P_{t+1} k_{\theta_t}(x, X_L)\alpha^* + P_{t+1} k_{\theta_t}(x, X_L)\alpha^* \\
&\quad \left. - k_{\theta_t}(x, X_L)\alpha_{t+1}\right] = \varphi_{t+1}^1 + \eta_t^2 k_{\theta_t}^{\aleph}(x, X_L) \\
&\quad \cdot \left[P_{t+1} k_{\theta_t}(x, X_L)\alpha_{t+1}\right.
\end{aligned}$$

$$\begin{aligned}&- P_{t+1}k_{\theta_t}(x, X_L)\alpha^*\big] + \eta_t^2 k_{\theta_t}^{\aleph}(x, X_L)\\&\cdot\big[P_{t+1}k_{\theta_t}(x, X_L)\alpha^*\\&- k_{\theta_t}(x, X_L)\alpha_{t+1}\big].\end{aligned} \tag{16}$$

Further define that

$$\begin{aligned}\lambda_1(x, \alpha_{t+1}, \alpha^*) &= k_{\theta_t}^{\aleph}(x, X_L)\\&\cdot\left[P_{t+1}k_{\theta_t}(x, X_L)\alpha_{t+1} - P_{t+1}k_{\theta_t}(x, X_L)\alpha^*\right],\\\lambda_2(x, \alpha_{t+1}, \alpha^*) &= k_{\theta_t}^{\aleph}(x, X_L)\\&\cdot\left[P_{t+1}k_{\theta_t}(x, X_L)\alpha^* - k_{\theta_t}(x, X_L)\alpha_{t+1}\right].\end{aligned} \tag{17}$$

Thus (16) is reexpressed as

$$\varphi_{t+2}^2 = \varphi_{t+1}^1 + \eta_t^2\lambda_1(x, \alpha_{t+1}, \alpha^*) + \eta_t^2\lambda_2(x, \alpha_{t+1}, \alpha^*). \tag{18}$$

Let $\varphi_t = \left[(\varphi_t^1)^T \;\; (\varphi_t^2)^T\right]^T$, and the two-phase iteration is in the shape of stochastic approximation; that is,

$$\begin{aligned}&\begin{bmatrix}\varphi_{t+1}^1\\\varphi_{t+1}^2\end{bmatrix} = \begin{bmatrix}\varphi_t^1\\\varphi_t^2\end{bmatrix} + \begin{bmatrix}-\eta_t^1\varphi_t^1\\0\end{bmatrix},\\&\begin{bmatrix}\varphi_{t+2}^1\\\varphi_{t+2}^2\end{bmatrix}\\&\quad = \begin{bmatrix}\varphi_{t+1}^1\\\varphi_{t+1}^2\end{bmatrix}\\&\qquad + \begin{bmatrix}0\\\eta_t^2\lambda_1(x, \alpha_{t+1}, \alpha^*) + \eta_t^2\lambda_2(x, \alpha_{t+1}, \alpha^*)\end{bmatrix}.\end{aligned} \tag{19}$$

Define

$$\begin{aligned}&V(\varphi_t)\\&= \varphi_t^T\Lambda\varphi_t\\&= \left[(\varphi_t^1)^T \;\; (\varphi_t^2)^T\right]\begin{bmatrix}I_D & 0\\0 & k_{\theta_t}^T(x^*, X_L)k_{\theta_t}(x^*, X_L)\end{bmatrix}\begin{bmatrix}\varphi_t^1\\\varphi_t^2\end{bmatrix},\end{aligned} \tag{20}$$

where D is the scale of the hyperparameters and x^* will be defined later. Let k_t^* represent $k_{\theta_t}^T(x^*, X_L)k_{\theta_t}(x^*, X_L)$ for short.

Obviously (20) is a positive definite matrix, and $|x^*| < \infty$, $|k_t^*| < \infty$. Hence, $V(\varphi_t)$ is a Lyapunov functional. At moment t, the conditional expectation of the positive definite matrix is

$$\begin{aligned}&E_tV_{t+2}(\varphi_{t+2})\\&\quad = E_t\left[\varphi_{t+2}^T\Lambda\varphi_{t+2}\right]\\&\quad = E_t\left\{\left[(\varphi_{t+2}^1)^T \;\; (\varphi_{t+2}^2)^T\right]\begin{bmatrix}I_D & 0\\0 & k_t^*\end{bmatrix}\begin{bmatrix}\varphi_{t+2}^1\\\varphi_{t+2}^2\end{bmatrix}\right\}.\end{aligned} \tag{21}$$

It is easy to compute the first-order Taylor expansion of (21) as

$$\begin{aligned}&E_tV_{t+2}(\varphi_{t+2})\\&\quad = E_tV_{t+1}(\varphi_{t+1}) + \eta_t^2E_t\left[V_{t+1}'(\varphi_{t+1})(\varphi_{t+2} - \varphi_{t+1})\right]\\&\qquad + \left(\eta_t^2\right)^2C_2E_t\left[(\varphi_{t+2} - \varphi_{t+1})^T(\varphi_{t+2} - \varphi_{t+1})\right]\end{aligned} \tag{22}$$

in which the first item on the right of (22) is

$$\begin{aligned}&E_tV_{t+1}(\varphi_{t+1})\\&\quad = V_t(\varphi_t) + \eta_t^1E_t\left[V'_t(\varphi_t)(\varphi_{t+1} - \varphi_t)\right]\\&\qquad + \left(\eta_t^1\right)^2C_1E_t\left[(\varphi_{t+1} - \varphi_t)^T(\varphi_{t+1} - \varphi_t)\right].\end{aligned} \tag{23}$$

Substituting (23) into (22) yields

$$\begin{aligned}&E_tV_{t+2}(\varphi_{t+2}) - V_t(\varphi_t)\\&\quad = \eta_t^1E_t\left[V'_t(\varphi_t)Y_t^1\right] + \eta_t^2E_t\left[V_{t+1}'(\varphi_{t+1})Y_{t+1}^2\right]\\&\qquad + \left(\eta_t^1\right)^2C_1E_t\left[\left(Y_t^1\right)^2\right] + \left(\eta_t^1\right)^2C_2E_t\left[\left(Y_{t+1}^2\right)^2\right],\end{aligned} \tag{24}$$

where $Y_t^1 = \begin{bmatrix}-\varphi_t^1\\0\end{bmatrix}$,

$$\begin{aligned}Y_{t+1}^2 &= \begin{bmatrix}0\\P_{t+1}k_{\theta_t}(x, X_L)\alpha_{t+1} - k_{\theta_t}(x, X_L)\alpha_{t+1}\end{bmatrix}\\&= \begin{bmatrix}0\\\lambda_1(x, \alpha_{t+1}, \alpha^*) + \lambda_2(x, \alpha_{t+1}, \alpha^*)\end{bmatrix}.\end{aligned} \tag{25}$$

Consider the last two items of (24) firstly. If the immediate reward is bounded and infinite discounted reward is applied, the value function is bounded. Hence, $|\alpha_t| < \infty$.

If the system is BIBO stable, $\exists\Sigma_0, \Sigma_1 \subset R^{n+m}$, n, m are the dimensions of the state space and action space, respectively, $\forall x_0 \in \Sigma_0$, $x_\infty \in \Sigma_1$, the policy space is bounded.

According to Proposition 1, when the policy space is bounded and $|\alpha_t| < \infty$, $|\theta_t| < \infty$, there exists a constant c_1, so that

$$E\left|V_{t+1}'(\varphi_{t+1})Y_{t+1}^2\right| \le c_1. \tag{26}$$

In addition,

$$\begin{aligned}E\left|Y_{t+1}^2\right|^2 &= E\left|\left[\lambda_1(x, \alpha_{t+1}, \alpha^*) + \lambda_2(x, \alpha_{t+1}, \alpha^*)\right]^2\right|\\&\le 2E\left|\lambda_1^2(x, \alpha_{t+1}, \alpha^*)\right| + 2E\left|\lambda_2^2(x, \alpha_{t+1}, \alpha^*)\right|\\&\le c_2\varphi_{t+1}^2 + c_3\varphi_{t+1}^2 \le c_4V(\varphi_{t+1}).\end{aligned} \tag{27}$$

From (26) and (27), we know that

$$E\left|Y_{t+1}^2\right|^2 + E\left|V_{t+1}'(\varphi_{t+1})Y_{t+1}^2\right| \le C_3V(\varphi_{t+1}) + C_3, \tag{28}$$

where $C_3 = \max\{c_1, c_4\}$. Similarly

$$E\left|Y_t^1\right|^2 + E\left|V'_t(\varphi_t)Y_t^1\right| \le C_4V(\varphi_t) + C_4. \tag{29}$$

According to Lemma 5.4.1 in [30], $EV(\varphi_t) < \infty$, and $E|Y_t^1|^2 < \infty$ and $E|Y_{t+1}^2|^2 < \infty$.

Now we focus on the first two items on the right of (24). The first item $E_t[V'_t(\varphi_t)Y_t^1]$ is computed as

$$E_t\left[V'_t(\varphi_t)Y_t^1\right] = E_t\left\{\left[\left(\varphi_t^1\right)^T \; \left(\varphi_t^2\right)^T\right]\begin{bmatrix} I_D & 0 \\ 0 & k_t^* \end{bmatrix}\begin{bmatrix} -\varphi_t^1 \\ 0 \end{bmatrix}\right\} = -\left|\varphi_t^1\right|^2. \tag{30}$$

For the second item $E_t[V'_{t+1}(\varphi_{t+1})Y_{t+1}^2]$, when the state transition function is time invariant, it is true that $P_{t+1}K_{\theta_t}(x,X_L)\alpha_{t+1} = P_tK_{\theta_t}(x,X_L)\alpha_{t+1}$, $\alpha_{t+1} = \alpha_t$. Then we have

$$\begin{aligned} E_t\lambda_1(x,\alpha_{t+1},\alpha^*) &= E_t\left[k_{\theta_t}^{\aleph}(x,X_L)P_tk_{\theta_t}(x,X_L)\alpha_t - k_{\theta_t}^{\aleph}(x,X_L)P_tk_{\theta_t}(x,X_L)\alpha^*\right] = k_{\theta_t}^{\aleph}(x,X_L) \\ &\cdot E\left[P_tk_{\theta_t}(x,X_L)\alpha_t - P_tk_{\theta_t}(x,X_L)\alpha^*\right] \le k_{\theta_t}^{\aleph}(x,X_L) \\ &\cdot \left\|E\left[P_tk_{\theta_t}(x,X_L)\alpha_t - P_tk_{\theta_t}(x,X_L)\alpha^*\right]\right\|. \end{aligned} \tag{31}$$

This inequality holds because of positive $k_{\theta_t}^{\aleph}(x,X)$, $k_{\theta_t,i}^{\aleph} > 0$, $i = 1,\ldots,L$. Define the norm $\|\Delta(\theta,\cdot)\| = \max_x|\Delta(\theta,x)|$, where $|\Delta(\cdot)|$ is the derivative matrix norm of the vector $\mathbf{1}$ and $x^* = \arg\max_x|\Delta(\theta,x)|$. Then

$$\begin{aligned} &\left\|E\left[P_tk_{\theta_t}(x,X_L)\alpha_t - P_tk_{\theta_t}(x,X_L)\alpha^*\right]\right\| \\ &= \max_x E_t\left|\left[P_tk_{\theta_t}(x,X_L)\alpha_t - P_tk_{\theta_t}(x,X_L)\alpha^*\right]\right| \\ &= \max_x\left|E_t\left[\left(r + \beta\max_{a'}k_{\theta_t}(x',X_L)\alpha_t\right) - \left(r + \beta\max_{a'}k_{\theta_t}(x',X_L)\alpha^*\right)\right]\right| = \beta \\ &\cdot \max_x\left|E_t\left[\max_{a'}k_{\theta_t}(x',X_L)\alpha_t - \max_{a'}k_{\theta_t}(x',X_L)\alpha^*\right]\right| = \beta\max_x\left|\int_{s'}p(s'\mid x)\right. \\ &\cdot \left.\left[\max_{a'}k_{\theta_t}(x',X_L)\alpha_t - \max_{a'}k_{\theta_t}(x',X_L)\alpha^*\right]ds'\right| \\ &\le \beta\max_x\int_{s'}p(s'\mid x) \\ &\cdot \left|\left[\max_{a'}k_{\theta_t}(x',X_L)\alpha_t - \max_{a'}k_{\theta_t}(x',X_L)\alpha^*\right]\right|ds' \\ &\le \beta\max_x\int_{s'}p(s'\mid x) \\ &\cdot \max_{a'}\left|k_{\theta_t}(x',X_L)\alpha_t - k_{\theta_t}(x',X_L)\alpha^*\right|ds' = \beta \\ &\cdot \max_x\int_{s'}p(s'\mid x)\max_{a'}\left|k_{\theta_t}(x',X_L)(\alpha_t-\alpha^*)\right|ds' \\ &= \beta\max_x\int_{s'}p(s'\mid x) \\ &\cdot \max_{a'}\left|E_t\left[P_tk_{\theta_t}(x',X_L)\alpha_t - P_tk_{\theta_t}(x',X_L)\alpha^*\right]\right|ds' \\ &\le \beta\max_x\left|k_{\theta_t}(x,X_L)(\alpha_t-\alpha^*)\right| = \beta\left|k_{\theta_t}(x^*,X_L)\right. \\ &\left.\cdot\varphi_t^2\right|. \end{aligned} \tag{32}$$

Hence

$$E_t\lambda_1(x,\alpha_{t+1},\alpha^*) \le \beta k_{\theta_t}^{\aleph}(x^*,X_L)\left|k_{\theta_t}(x^*,X_L)\varphi_t^2\right|. \tag{33}$$

On the other hand,

$$\begin{aligned} E_t\lambda_2(x,\alpha_t,\alpha^*) &= E_t\left[k_{\theta_t}^{\aleph}(x,X_L)P_tk_{\theta_t}(x,X_L)\alpha^* - k_{\theta_t}^{\aleph}(x,X_L)k_{\theta_t}(x,X_L)\alpha_t\right] \\ &= E_t\left[k_{\theta_t}^{\aleph}(x,X_L)P_tk_{\theta_t}(x,X_L)\alpha^* - k_{\theta_t}^{\aleph}(x,X_L)k_{\theta_t}(x,X_L)\alpha^*\right] + k_{\theta_t}^{\aleph}(x,X_L) \\ &\cdot k_{\theta_t}(x,X_L)\alpha^* - k_{\theta_t}^{\aleph}(x,X_L)k_{\theta_t}(x,X_L)\alpha_t \\ &= -k_{\theta_t}^{\aleph}(x,X_L)\left[k_{\theta_t}(x,X_L)\varphi_t^2 - E_t\lambda_3(x,\theta_t,\theta^*)\right], \end{aligned} \tag{34}$$

where $\lambda_3(x,\theta_t,\theta^*) = P_tk_{\theta_t}(x,X_L)\alpha^* - k_{\theta_t}(x,X_L)\alpha^*$ is the value function error caused by the estimated hyperparameters.

Substituting (33) and (34) into $E_t[V'_{t+1}(\varphi_{t+1})Y_{t+1}^2]$ yields

$$\begin{aligned} E_t\left[V'_{t+1}(\varphi_{t+1})Y_{t+1}^2\right] &= E_t\left\{\left[k_{\theta_t}(x^*,X_L)\varphi_t^2\right]^T \cdot k_{\theta_t}(x^*,X_L)\left[\lambda_1(x,\alpha_{t+1},\alpha^*) + \lambda_2(x,\alpha_{t+1},\alpha^*)\right]\right\} \\ &= \left[k_{\theta_t}(x^*,X_L)\varphi_t^2\right]^T k_{\theta_t}(x^*,X_L) \\ &\cdot \left[E_t\lambda_1(x,\alpha_{t+1},\alpha^*) + E_t\lambda_2(x,\alpha_{t+1},\alpha^*)\right] \\ &\le \beta\left[k_{\theta_t}(x^*,X_L)\varphi_t^2\right]^T k_{\theta_t}(x^*,X_L)k_{\theta_t}^{\aleph}(x^*,X_L) \\ &\cdot \left|k_{\theta_t}(x^*,X_L)\varphi_t^2\right| - \left[k_{\theta_t}(x^*,X_L)\varphi_t^2\right]^T k_{\theta_t}(x^*,X_L) \\ &\cdot k_{\theta_t}^{\aleph}(x^*,X_L)\left[k_{\theta_t}(x^*,X_L)\varphi_t^2 - E_t\lambda_3(x^*,\theta_t,\theta^*)\right] \\ &\le -(1-\beta)\left|k_{\theta_t}(x^*,X_L)\varphi_t^2\right|^2 + \left|\left[k_{\theta_t}(x^*,X_L)\varphi_t^2\right]^T\right. \\ &\left.\cdot E_t\lambda_3(x^*,\theta_t,\theta^*)\right|. \end{aligned} \tag{35}$$

Obviously, if the following inequality is satisfied,

$$\eta_t^2 \left|\left[k_{\theta_t}(x^*, X_L)\varphi_t^2\right]^T E_t\lambda_3(x^*, \theta_t, \theta^*)\right| < \eta_t^1 \left|\varphi_t^1\right|^2 + \eta_t^2(1-\beta)\left|k_{\theta_t}(x^*, X_L)\varphi_t^2\right|^2 \quad (36)$$

there exists a positive const δ, such that the first two items on the right of (24) satisfy

$$\eta_t^1 E_t\left[V'_t(\varphi_t) Y_t^1\right] + \eta_t^2 E_t\left[V'_{t+1}(\varphi_{t+1}) Y_{t+1}^2\right] \le -\delta. \quad (37)$$

According to Theorem 5.4.2 in [30], the iterative process is convergent; namely, $\varphi_t \xrightarrow{w.p.1} 0$. □

Remark 3. Let us check the final convergence position. Obviously, $\varphi_t \xrightarrow{w.p.1} 0$, $\left[\begin{smallmatrix}\alpha_t\\ \theta_t\end{smallmatrix}\right] \xrightarrow{w.p.1} \left[\begin{smallmatrix}\alpha^*\\ f(\alpha^*)\end{smallmatrix}\right]$. This means the equilibrium of the critic network meets $E_t[P_t K_{\theta^*}(x, X_L)\alpha^*] = K_{\theta^*}(x, X_L)\alpha^*$, where $\theta^* = f(\alpha^*)$. And the equilibrium of hyperparameters $\theta^* = f(\alpha^*)$ is the solution of evidence maximization that $\max_\varphi(L(\varphi)) = \max_\varphi((1/2)y^T K_{\theta^*}^{-1}(X_L, X_L)y + (1/2)\log|C| + (n/2)\log 2\pi)$, where $y = K_{\theta^*}^{-1}(X_L, X_L)\alpha^*$.

Remark 4. It is clear that the selection of the samples is one of the key issues. Since now all samples have values according to two-phase iteration, according to the information-based criterion, it is convenient to evaluate samples and refine the set by arranging relative more samples near the equilibrium or with great gradient of the value function, so that the sample set is better to describe the distribution of value function.

Remark 5. Since the two-phase iteration belongs to value iteration, the initial policy of the algorithm does not need to be stable. To ensure BIBO in practice, we need a mechanism to clamp the output of system, even though the system will not be smooth any longer.

Theorem 2 gives the iteration principle for critic network learning. Based on the critic network, with proper objective defined, such as minimizing the expected total discounted reward [15], HDP or DHP update is applied to optimize actor network. Since this paper focuses on ACD and more importantly the update process of actor network does not affect the convergence of critic network, though maybe it induces premature or local optimization, the gradient update of actor is not necessary. Hence a simple optimum seeking is applied to get optimal actions w.r.t. sample states, and then an actor network based on Gaussian kernel is generated based on these optimal state-action pairs.

Up to now we have built a critic-actor architecture which is named Gaussian-kernel-based Approximate Dynamic Programming (GK-ADP for short) and shown in Algorithm 1. A span K is introduced, so that hyperparameters are updated every K times of α update. If $K > 1$, two phases are asynchronous. From the view of stochastic approximation, this asynchronous learning does not change the convergence condition (36) but benefit computational cost of hyperparameters learning.

4. Simulation and Discussion

In this section, we propose some numerical simulations about continuous control to illustrate the special properties and the feasibility of the algorithm, including the necessity of two phases learning, the specifical properties comparing with traditional kernel-based ACDs, and the performance enhancement resulting from online refinement of sample set.

Before further discussion, we firstly give common setup in all simulations:

(i) The span $K = 50$.

(ii) To make output bounded, once a state is out of the boundary, the system is reset randomly.

(iii) The exploration-exploitation tradeoff is left out of account here. During learning process, all actions are selected randomly within limited action space. Thus the behavior of the system is totally unordered during learning.

(iv) The same ALD-based sparsification in [15] is used to determine the initial sample set, in which $w_i = 1.8$, $v_0 = 0$, and $v_1 = 0.5$ empirically.

(v) The sampling time and control interval are set to 0.02 s.

4.1. The Necessity of Two-Phase Learning

4.1. The Necessity of Two-Phase Learning. The proof of Theorem 2 shows that properly determined learning rates of phases 1 and 2 guarantee condition (36). Here we propose a simulation to show how the learning rates in phases 1 and 2 affect the performance of the learning.

Consider a simple single homogeneous inverted pendulum system:

$$\dot{s}_1 = s_2$$
$$\dot{s}_2 = \frac{\left(M_p g \sin(s_1) - \tau\cos(s_1)\right)d}{M_p d}, \quad (38)$$

where s_1 and s_2 represent the angle and its speed, $M_p = 0.1\,\text{kg}$, $g = 9.8\,\text{m/s}^2$, $d = 0.2\,\text{m}$, respectively, and τ is a horizontal force acting on the pole.

We test the success rate under different learning rates. Since, during the learning the action is always selected randomly, we have to verify the optimal policy after learning; that is, an independent policy test is carried out to test the actor network. Thus the success of one time of learning means in the independent test the pole can be swung up and maintained within $[-0.02, 0.02]$ rad for more than 200 iterations.

In the simulation, 60 state-action pairs are collected to serve as the sample set, and the learning rates are set to $\eta_t^2 = 0.01/(1+t)^{0.018}$, $\eta_t^1 = a/(1+t^{0.018})$, where a is varying from 0 to 0.12.

The learning is repeated 50 times in order to get the average performance, where the initial states of each run are randomly selected within the bound $[-0.4, 0.4]$ and $[-0.5, 0.5]$ w.r.t. the dimensions of s_1 and s_2. The action space

```
Initialize:
        $\theta$: hyperparameters of Gaussian kernel model
        $X_L = \{x_i \mid x_i = (s_i, a_i)\}_{i=1}^{L}$: sample set
        $\pi(s_0)$: initial policy
        $\eta_t^1, \eta_t^2$: learning step size
Let $t = 0$;
Loop:
    for $k = 1$ to $K$ do
        $t = t + 1$;
        $a_t = \pi(s_t)$
        Get the reward $r_t$
        Observe next state $s_{t+1}$
        Update $\alpha$ according to (12)
        Update the policy $\pi$ according to optimum seeking
    end for
    Update $\theta$ according to (12)
Until the termination criterion is satisfied
```

Algorithm 1: Gaussian-kernel-based Approximate Dynamic Programming.

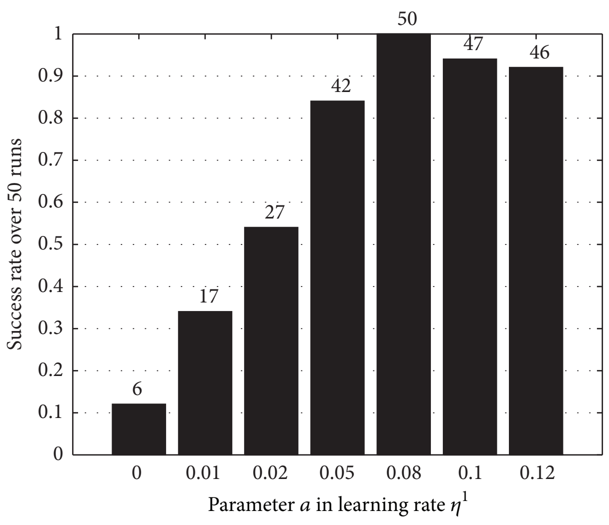


Figure 1: Success times over 50 runs under different learning rate η_t^1.

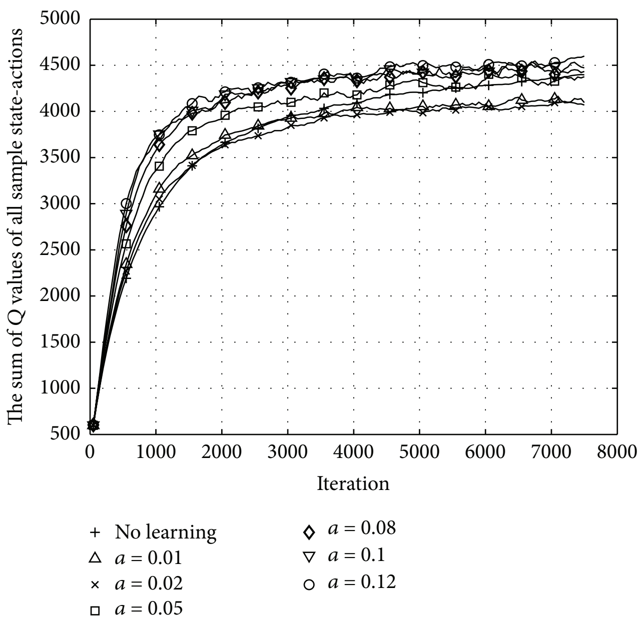


Figure 2: The average evolution processes over 50 runs w.r.t. different η_t^1.

is bounded within $[-0.5, 0.5]$. And, in each run, the initial hyperparameters are set to $[e^{-2.5} \quad e^{0.5} \quad e^{0.9} \quad e^{-1} \quad e^{0.001}]$.

Figure 1 shows the result of success rates. Clearly, with η_t^2 fixed, different η_t^1's affect the performance significantly. In particular, without the learning of hyperparameters, that is, $\eta_t^1 = 0$, there are only 6 successful runs over 50 runs. With the increasing of a the success rate increases till 1 when $a = 0.08$. But as a goes on increasing, the performance becomes worse.

Hence both phases are necessary if the hyperparameters are not initialized properly. It should be noted that the learning rates w.r.t. two phases need to be regulated carefully in order to guarantee condition (36), which leads the learning process to the equilibrium in Remark 3 but not to some kind of boundary where the actor always executed the maximal or minimal actions.

If all samples' values on each iteration are summed up and all cumulative values w.r.t. iterations are depicted in series, then we have Figure 2. The evolution process w.r.t. the best parameter $a = 0.08$ is marked by the darker diamond. It is clear that, even with different η_t^1, the evolution processes of the samples' value learning are similar. That means, due to BIBO property, the samples' values must be finally bounded and convergent. However, as mentioned above, it does not mean that the learning process converges to the proper equilibrium.

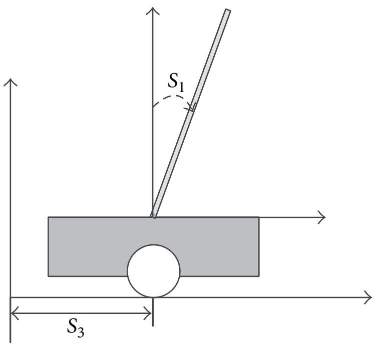


Figure 3: One-dimensional inverted pendulum.

4.2. Value Iteration versus Policy Iteration. As mentioned in Algorithm 1, the value iteration in critic network learning does not depend on the convergence of the actor network, and, compared with HDP or DHP, the direct optimum seeking for actor network seems a little aimless. To test its performance and discuss its special characters of learning, a comparison between GK-ADP and KHDP is carried out.

The control objective in the simulation is a one-dimensional inverted pendulum, where a single-stage inverted pendulum is mounted on a cart which is able to move linearly, just as Figure 3 shows.

The mathematic model is given as

$$\begin{aligned}
\dot{s}_1 &= s_2 + \varepsilon, \\
\dot{s}_2 &= \frac{g\sin(s_1) - M_p d s_2^2 \cos(s_1)\sin(s_1)/M}{d\left(4/3 - M_p \cos^2(s_1)/M\right)} \\
&\quad + \frac{\cos(s_1)/M}{d\left(4/3 - M_p \cos^2(s_1)/M\right)} u, \\
\dot{s}_3 &= s_4, \\
\dot{s}_4 &= \frac{u + M_p d\left(s_2^2 \sin(s_1) - \dot{s}_2 \cos(s_1)\right)}{M},
\end{aligned} \tag{39}$$

where s_1 to s_4 represent the state of angle, angle speed, linear displacement, and linear speed of the cart, respectively, u represents the force acting on the cart, $M = 0.2$ kg, $d = 0.5$ m, $\varepsilon \sim U(-0.02, 0.02)$, and other denotations are the same as (38).

Thus the state-action space is 5D space, much larger than that in simulation 1. The configurations of the both algorithms are listed as follows:

(i) A small sample set with 50 samples is adopted to build critic network, which is determined by ALD-based sparsification.

(ii) For both algorithms, the state-action pair is limited to $[-0.3, 0.3]$, $[-1, 1]$, $[-0.3, 0.3]$, $[-2, 2]$, $[-3, 3]$ w.r.t. s_1 to s_4 and u.

(iii) In GK-ADP, the learning rates are $\eta_t^1 = 0.02/(1 + t^{0.01})$ and $\eta_t^2 = 0.02/(1 + t)^{0.02}$.

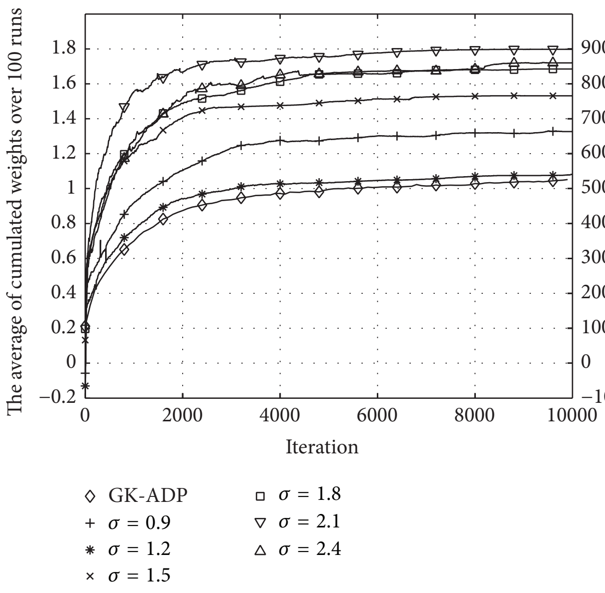


Figure 4: The comparison of the average evolution processes over 100 runs.

(iv) The KHDP algorithm in [15] is chosen as the comparison, where the critic network uses Gaussian kernels. To get the proper Gaussian parameters, a series of σ as 0.9, 1.2, 1.5, 1.8, 2.1, and 2.4 are tested in the simulation. The elements of weights α are initialized randomly in $[-0.5, 0.5]$, the forgetting factor in RLS-TD(0) is set to $\mu = 1$, and the learning rate in KHDP is set to 0.3.

(v) All experiments run 100 times to get the average performance. And in each run there are 10000 iterations to learn critic network.

Before further discussion, it should be noted that it makes no sense to figure out ourselves with which algorithm is better in learning performance, because, besides the debate of policy iteration versus value iteration, there are too many parameters in the simulation configuration affecting learning performance. So the aim of this simulation is to illustrate the learning characters of GK-ADP.

However to make the comparison as fair as possible, we regulate the learning rates of both algorithms to get similar evolution processes. Figure 4 shows the evolution processes of GK-ADP and KHDPs under different σ, where the y-axis in the left represents the cumulated weights, $\sum_i \alpha_i$, of the critic network in kernel ACD, and the other y-axis represents the cumulated values of the samples, $\sum_i Q(x_i)$, $x_i \in X_L$, in GK-ADP. It implies although, with different units, the learning processes under both learning algorithms converge nearly at the same speed.

Then the success rates of all algorithms are depicted in Figure 5. The left six bars represent the success rates of KHDP with different σ. With superiority argument left aside, we find that, such fixed σ in KHDP is very similar to the fixed hyperparameters θ, which needs to be set properly in order to get higher success rate. But unfortunately there is no mechanism in KHDP to regulate σ online. On the

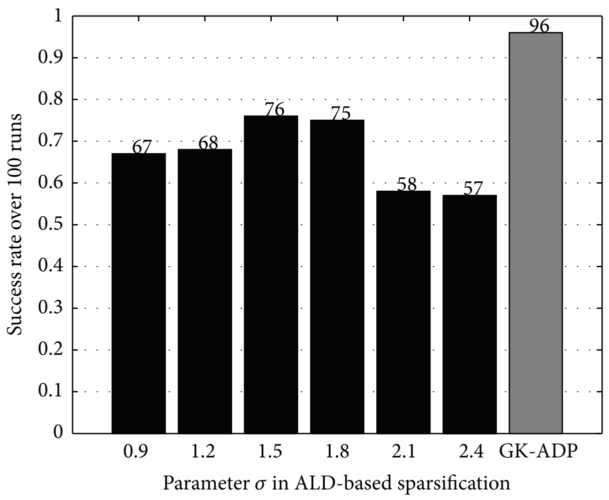


FIGURE 5: The success rates w.r.t. all algorithms over 100 runs.

contrary, the two-phase iteration introduces the update of hyperparameters into critic network, which is able to drive hyperparameters to better values, even with the not so good initial values. In fact, this two-phase update can be viewed as a kind of kernel ACD with dynamic σ, when Gaussian kernel is used.

To discuss the performance in deep, we plot the test trajectories resulting from the actors, which are optimized by GK-ADP and KHDP, in Figures 6(a) and 6(b), respectively, where the start state is set to $[0.2\ \ 0\ \ -0.2\ \ 0]$.

Apparently the transition time of GK-ADP is much smaller than KHDP. We think, besides the possible well regulated parameters, an important reason is nongradient learning for actor network.

The critic learning only depends on exploration-exploitation balance but not the convergence of actor learning. If exploration-exploitation balance is designed without actor network output, the learning processes of actor and critic networks are relatively independent of each other, and then there are alternatives to gradient learning for actor network optimization, for example, the direct optimum seeking Gaussian regression actor network in GK-ADP.

Such direct optimum seeking may result in nearly nonsmooth actor network, just like the force output depicted in the second plot of Figure 6(a). To explain this phenomenon, we can find the clue from Figure 7, which illustrates the best actions w.r.t. all sample states according to the final actor network. It is reasonable that the force direction is always the same to the pole angle and contrary to the cart displacement. And clearly the transition interval from negative limit $-3N$ to positive $3N$ is very short. Thus the output of actor network, resulting from Gaussian kernels, intends to control angle bias as quickly as possible, even though such quick control sometimes makes the output force a little nonsmooth.

It is obvious that GK-ADP is with high efficiency but also with potential risks, such as the impact to actuators. Hence how to design exploration-exploitation balance to satisfy Theorem 2 and how to design actor network learning to balance efficiency and engineering applicability are two important issues in future work.

Finally we check the samples values, which are depicted in Figure 8, where only dimensions of pole angle and linear displacement are depicted. Due to the definition of immediate reward, the closer the states to the equilibrium are, the smaller the Q value is.

If we execute 96 successful policies one by one and record all final cart displacements and linear speed, just as Figure 9 shows, the shortage of this sample set is uncovered that, even with good convergence of pole angle, the optimal policy can hardly drive cart back to zero point.

To solve this problem and improve performance, besides optimizing learning parameters, Remark 4 implies that another and maybe more efficient way is refining sample set based on the samples' values. We will investigate it in the following subsection.

4.3. Online Adjustment of Sample Set. Due to learning sample's value, it is possible to assess whether the samples are chosen reasonable. In this simulation we adopt the following expected utility [26]:

$$U(x) = \rho h\left(E\left[Q(x) \mid X_L\right]\right) + \frac{\beta}{2}\log\left(\operatorname{var}\left[Q(x) \mid X_L\right]\right), \tag{40}$$

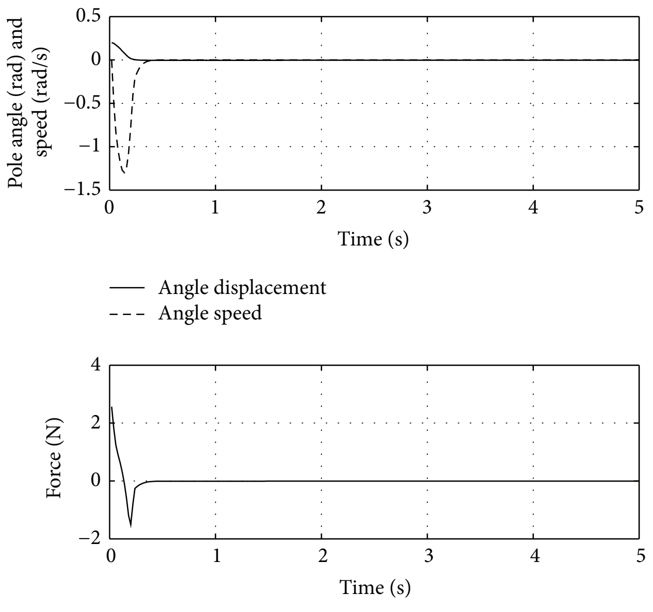
where $h(\cdot)$ is a normalization function over sample set and $\rho = 1$, $\beta = 0.3$ are coefficients.

Let N_{add} represent the number of samples added to the sample set, N_{limit} the size limit of the sample set, and $T(x) = x(0), x(1), \ldots, x(T_e)$ the state-action pair trajectory during GK-ADP learning, where T_e denotes the end iteration of learning. The algorithm to refine sample set is presented as Algorithm 2.

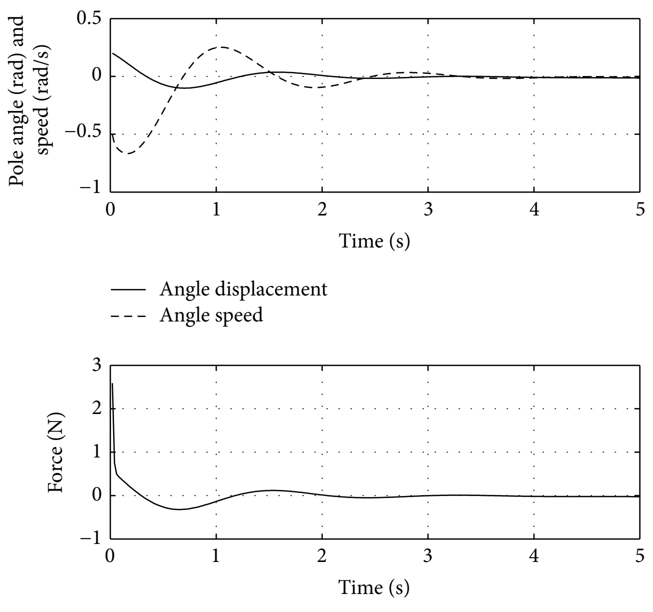
Since in simulation 2 we have carried out 100 runs of experiment for GK-ADP, 100 sets of sample values w.r.t. the same sample states are obtained. Now let us apply Algorithm 2 to these resulted sample sets, where $T(x)$ is picked up from the history trajectory of state-action pairs in each run, and $N_{\text{add}} = 10$, $N_{\text{limit}} = 60$, in order to add 10 samples into sample set.

We repeat all 100 runs of GK-ADP again, with the same learning rates η^1 and η^2. Based on 96 successful runs in simulation 2, 94 runs are successful in obtaining control policies swinging up pole. Figure 10(a) shows the average evolutional process of sample values over 100 runs, where the first 10000 iterations display the learning process in simulation 2 and the second 8000 iterations display the learning process after adding samples. Clearly the cumulative Q value behaves a sudden increase after sample was added and almost keeps steady afterwards. It implies, even with more samples added, there is not too much to learn for sample values.

However if we check the cart movement, we will find the enhancement brought by the change of sample set. For the jth run, we have two actor networks resulting from GK-ADP before and after adding samples. Using these actors, We carry out the control test, respectively, and collect the absolute



(a) The resulted control performance using GK-ADP



(b) The resulted control performance using KHDP

Figure 6: The outputs of the actor networks resulting from GK-ADP and KHDP.

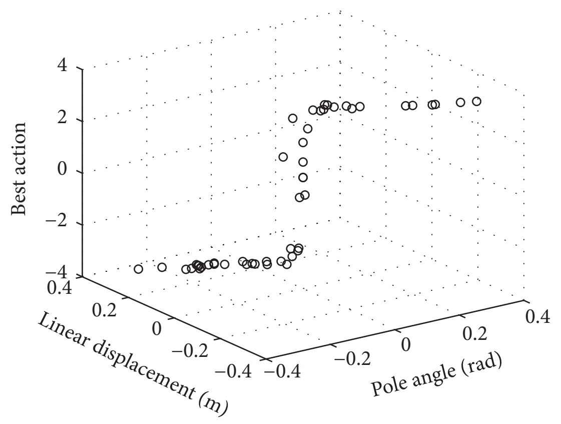


Figure 7: The final best policies w.r.t. all samples.

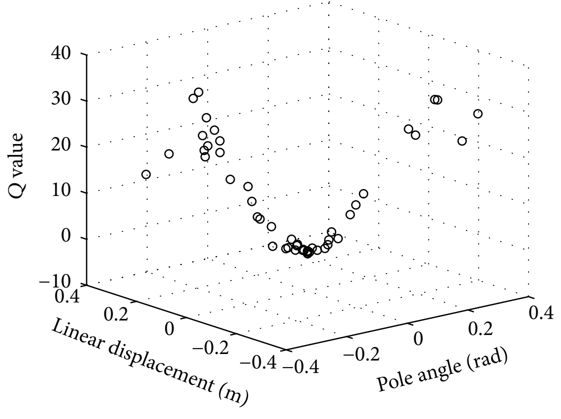


Figure 8: The Q values projected on to dimensions of pole angle (s_1) and linear displacement (s_3) w.r.t. all samples.

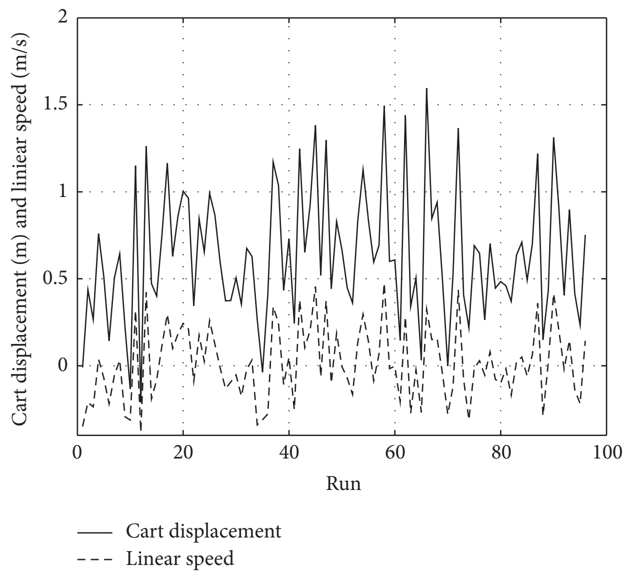


Figure 9: The final cart displacement and linear speed in the test of successful policy.

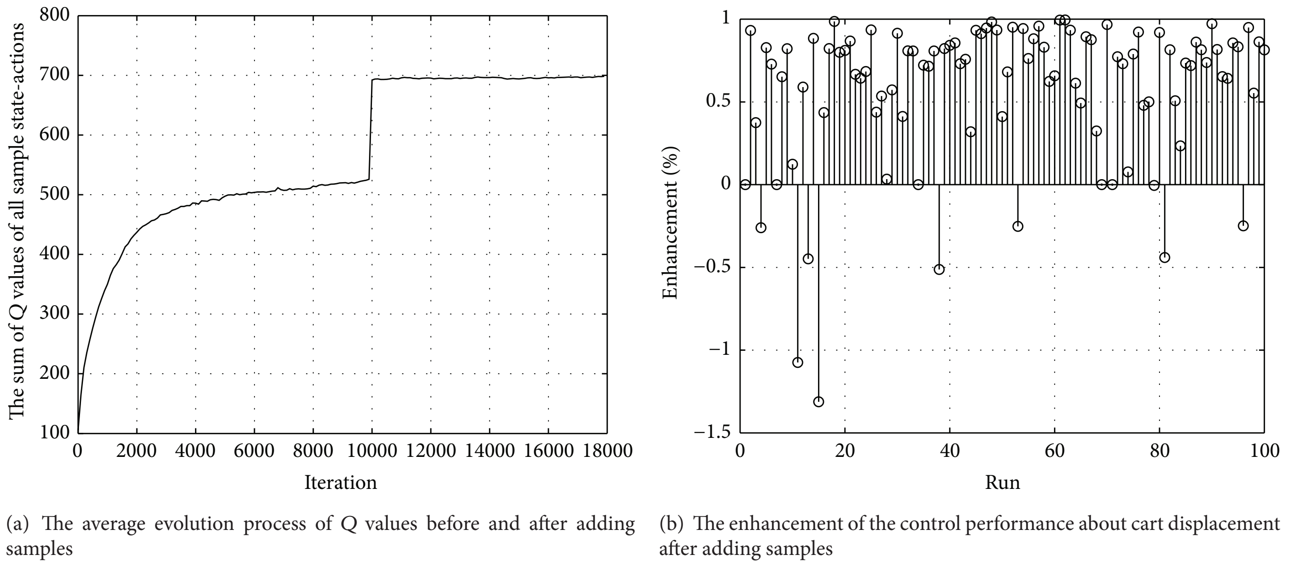
values of cart displacement, denoted by $S_3^b(j)$ and $S_3^a(j)$, at the moment that the pole has been swung up for 100 instants. We define the enhancement as

$$E(j) = \frac{S_3^b(j) - S_3^a(j)}{S_3^b(j)}. \tag{41}$$

Obviously the positive enhancement indicates that the actor network after adding samples behaves better. As all enhancements w.r.t. 100 runs are illustrated together, just

```
for $l = 1$ to the end of $T(x)$
    Calculate $E[Q(x(l)) \mid X_L]$ and $\mathrm{var}[Q(xl) \mid X_L]$ using (6) and (7)
    Calculate $U(x(l))$ using (40)
end for $l$
Sort all $U(x(l)), l = 1, \ldots, T_e$ in ascending order
Add the first $N_{\mathrm{add}}$ state-action pairs to sample set to get extended set $X_{L+N_{\mathrm{add}}}$
if $L + N_{\mathrm{add}} > N_{\mathrm{limit}}$
    Calculate hyperparameters based on $X_{L+N_{\mathrm{add}}}$
    for $l = 1$ to $L + N_{\mathrm{add}}$
        Calculate $E[Q(x(l)) \mid X_{L+N_{\mathrm{add}}}]$ and $\mathrm{var}[Q(xl) \mid X_{L+N_{\mathrm{add}}}]$ using (6) and (7)
        Calculate $U(x(l))$ using (40)
    end for $l$
    Sort all $U(x(l)), l = 1, \ldots, L + N_{\mathrm{add}}$ in descending order
    Delete the first $L + N_{\mathrm{add}} - N_{\mathrm{limit}}$ samples to get refined set $X_{N_{\mathrm{limit}}}$
end if
```

Algorithm 2: Refinement of sample set.



(a) The average evolution process of Q values before and after adding samples

(b) The enhancement of the control performance about cart displacement after adding samples

Figure 10: Learning performance of GK-ADP if sample set is refined by adding 10 samples.

as Figure 10(b) shows, almost all cart displacements as the pole is swung up are enhanced more or less except for 8 times of failure. Hence with proper principle based on the sample values, the sample set can be refined online in order to enhance performance.

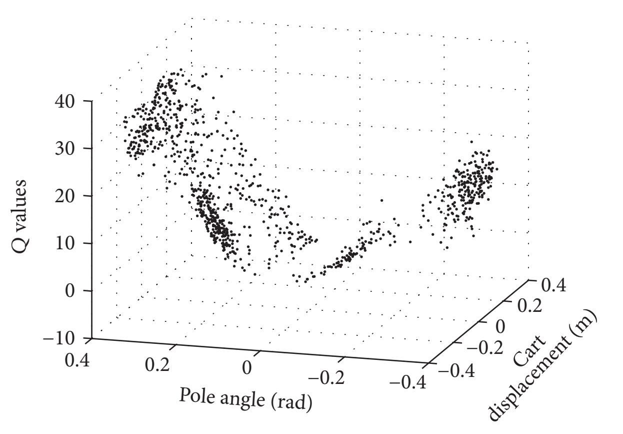
Finally let us check which state-action pairs are added into sample set. We put all added samples over 100 runs together and depict their values in Figure 11(a), where the values are projected onto the dimensions of pole angle and cart displacement. If only concerning the relationship between Q values and pole angle, just as Figure 11(b) shows, we find that the refinement principle intends to select the samples a little away from the equilibrium and near the boundary −0.4, due to the ratio between ρ and β.

5. Conclusions and Future Work

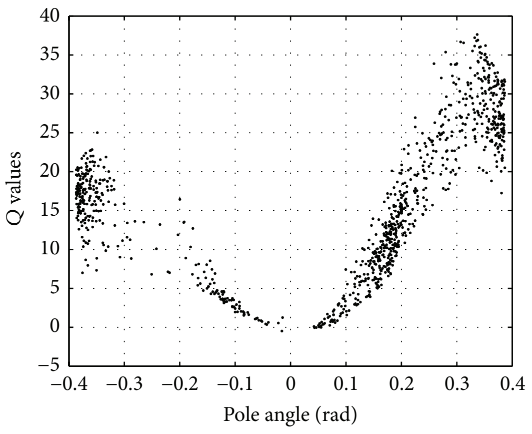
ADP methods are among the most promising research works on RL in continuous environment to construct learning systems for nonlinear optimal control. This paper presents GK-ADP with two-phase value iteration which combines the advantages of kernel ACDs and GP-based value iteration.

The theoretical analysis reveals that, with proper learning rates, two-phase iteration is good at making Gaussian-kernel-based critic network converge to the structure with optimal hyperparameters and approximate all samples' values.

A series of simulations are carried out to verify the necessity of two-phase learning and illustrate properties of GK-ADP. Finally the numerical tests support the viewpoint that the assessment of samples' values provides the way to



(a) Projecting on the dimensions of pole angle and cart displacement



(b) Projecting on the dimension of pole angle

Figure 11: The Q values w.r.t. all added samples over 100 runs.

refine sample set online, in order to enhance the performance of critic-actor architecture during operation.

However there are some issues needed to be concerned in future. The first is how to guarantee condition (36) during learning, which is now determined by empirical ways. The second is the balance between exploration and exploitation, which is always an opening question, but seems more notable here because bad exploration-exploitation principle will lead two-phase iteration to failure not only to slow convergence.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants 61473316 and 61202340 and the International Postdoctoral Exchange Fellowship Program under Grant no. 20140011.

References

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

[2] F. L. Lewis and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE Circuits and Systems Magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[3] F. L. Lewis, D. Vrabie, and K. Vamvoudakis, “Reinforcement learning and feedback control: using natural decision methods to design optimal adaptive controllers,” *IEEE Control Systems Magazine*, vol. 32, no. 6, pp. 76–105, 2012.

[4] H. Zhang, Y. Luo, and D. Liu, “Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints,” *IEEE Transactions on Neural Networks*, vol. 20, no. 9, pp. 1490–1503, 2009.

[5] Y. Huang and D. Liu, “Neural-network-based optimal tracking control scheme for a class of unknown discrete-time nonlinear systems using iterative ADP algorithm,” *Neurocomputing*, vol. 125, pp. 46–56, 2014.

[6] X. Xu, L. Zuo, and Z. Huang, “Reinforcement learning algorithms with function approximation: recent advances and applications,” *Information Sciences*, vol. 261, pp. 1–31, 2014.

[7] 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.

[8] S. Ferrari, J. E. Steck, and R. Chandramohan, “Adaptive feedback control by constrained approximate dynamic programming,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 4, pp. 982–987, 2008.

[9] F.-Y. Wang, H. Zhang, and D. Liu, “Adaptive dynamic programming: an introduction,” *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 39–47, 2009.

[10] D. Wang, D. Liu, Q. Wei, D. Zhao, and N. Jin, “Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming,” *Automatica*, vol. 48, no. 8, pp. 1825–1832, 2012.

[11] D. Vrabie and F. Lewis, “Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems,” *Neural Networks*, vol. 22, no. 3, pp. 237–246, 2009.

[12] D. Liu, D. Wang, and X. Yang, “An iterative adaptive dynamic programming algorithm for optimal control of unknown discrete-time nonlinear systems with constrained inputs,” *Information Sciences*, vol. 220, pp. 331–342, 2013.

[13] N. Jin, D. Liu, T. Huang, and Z. Pang, “Discrete-time adaptive dynamic programming using wavelet basis function neural networks,” in *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 135–142, Honolulu, Hawaii, USA, April 2007.

[14] P. Koprinkova-Hristova, M. Oubbati, and G. Palm, “Adaptive critic design with echo state network,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC ’10)*, pp. 1010–1015, IEEE, Istanbul, Turkey, October 2010.

[15] X. Xu, Z. Hou, C. Lian, and H. He, “Online learning control using adaptive critic designs with sparse kernel machines,” *IEEE*

Transactions on Neural Networks and Learning Systems, vol. 24, no. 5, pp. 762–775, 2013.

[16] V. Vapnik, *Statistical Learning Theory*, Wiley, New York, NY, USA, 1998.

[17] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning, MIT Press, Cambridge, Mass, USA, 2006.

[18] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[19] T. G. Dietterich and X. Wang, "Batch value function approximation via support vectors," in *Advances in Neural Information Processing Systems*, pp. 1491–1498, MIT Press, Cambridge, Mass, USA, 2002.

[20] X. Wang, X. Tian, Y. Cheng, and J. Yi, "Q-learning system based on cooperative least squares support vector machine," *Acta Automatica Sinica*, vol. 35, no. 2, pp. 214–219, 2009.

[21] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[22] M. F. Huber, "Recursive Gaussian process: on-line regression and learning," *Pattern Recognition Letters*, vol. 45, no. 1, pp. 85–91, 2014.

[23] Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: the Gaussian process approach to temporal difference learning," in *Proceedings of the 20th International Conference on Machine Learning*, pp. 154–161, August 2003.

[24] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proceedings of the 22nd International Conference on Machine Learning*, pp. 201–208, ACM, August 2005.

[25] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds., pp. 751–759, MIT Press, Cambridge, Mass, USA, 2004.

[26] M. P. Deisenroth, C. E. Rasmussen, and J. Peters, "Gaussian process dynamic programming," *Neurocomputing*, vol. 72, no. 7–9, pp. 1508–1524, 2009.

[27] X. Xu, C. Lian, L. Zuo, and H. He, "Kernel-based approximate dynamic programming for real-time online learning control: an experimental study," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 1, pp. 146–156, 2014.

[28] A. Girard, C. E. Rasmussen, and R. Murray-Smith, "Gaussian process priors with uncertain inputs-multiple-step-ahead prediction," Tech. Rep., 2002.

[29] X. Xu, T. Xie, D. Hu, and X. Lu, "Kernel least-squares temporal difference learning," *International Journal of Information Technology*, vol. 11, no. 9, pp. 54–63, 2005.

[30] H. J. Kushner and G. G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*, vol. 35 of *Applications of Mathematics*, Springer, Berlin, Germany, 2nd edition, 2003.