Online Selective Kernel-Based Temporal Difference Learning

Xingguo Chen, Yang Gao, *Member, IEEE*, and Ruili Wang

***Abstract*— In this paper, an online selective kernel-based temporal difference (OSKTD) learning algorithm is proposed to deal with large scale and/or continuous reinforcement learning problems. OSKTD includes two online procedures: online sparsification and parameter updating for the selective kernel-based value function. A new sparsification method (i.e., a kernel distance-based online sparsification method) is proposed based on selective ensemble learning, which is computationally less complex compared with other sparsification methods. With the proposed sparsification method, the sparsified dictionary of samples is constructed online by checking if a sample needs to be added to the sparsified dictionary. In addition, based on local validity, a selective kernel-based value function is proposed to select the best samples from the sample dictionary for the selective kernel-based value function approximator. The parameters of the selective kernel-based value function are iteratively updated by using the temporal difference (TD) learning algorithm combined with the gradient descent technique. The complexity of the online sparsification procedure in the OSKTD algorithm is $O(n)$. In addition, two typical experiments (Maze and Mountain Car) are used to compare with both traditional and up-to-date $O(n)$ algorithms (GTD, GTD2, and TDC using the kernel-based value function), and the results demonstrate the effectiveness of our proposed algorithm. In the Maze problem, OSKTD converges to an optimal policy and converges faster than both traditional and up-to-date algorithms. In the Mountain Car problem, OSKTD converges, requires less computation time compared with other sparsification methods, gets a better local optima than the traditional algorithms, and converges much faster than the up-to-date algorithms. In addition, OSKTD can reach a competitive ultimate optima compared with the up-to-date algorithms.**

***Index Terms*— Function approximation, online sparsification, reinforcement learning (RL), selective ensemble learning, selective kernel-based value function.**

I. Introduction

IN RECENT years, there has been an increasing interest in reinforcement learning (RL). It is well known that large scale and/or continuous space RL problems suffer from the curse of dimensionality, which means that the complexity exponentially increases with the dimensionality of a state space (i.e., the number of state variables) [1]. In other words, traditional tabular RL becomes impractical. A possible approach toward coping with this curse is approximate RL or approximate dynamic programming including value function approximation [2], policy search [3], actor-critic approaches [4], hierarchical RL [5], and transfer learning [6]. In this paper, we focus on value function approximation which is the most popular method.

There are several value function approximation techniques, such as linear function approximation [1], [7], regression tree methods [8], neural networks [9]–[13], and kernel methods [14]–[17]. Among them, we focus on the kernel methods in this paper. According to the Representer Theorem [18], a value function in RL can be formed as $V(s) = \sum_{i=1}^{N} \theta_i k(s, s_i)$, where s is a state, θ is the parameter vector, $k(.,.)$ is the kernel function, and s_i is the sample.

A kernel approach for RL needs to consider the following issues.

1) The kernel-based reinforcement learning (KBRL) needs to be able to construct a sample dictionary online, since RL is an online learning method.
2) The online sparsification process in constructing the sample dictionary needs to consider the tradeoff between accuracy and the complexity of the value function.
3) It is difficult to select a proper kernel function for a problem, since different kernel functions may have different performances for different problems.
4) It is difficult to select a proper learning algorithm to learn the parameters of the value function and the trick of the parameters settings [19]. In this paper, we propose an online selective kernel-based temporal difference (OSKTD) learning algorithm to address the above four issues.

Recently, KBRL is intensively studied. Table I gives a comparison between various KBRL algorithms, which are compared in terms of algorithm names, kernel sparsification methods and learning methods. Among them, radial basis function (RBF) networks are applied to approximate the value function, where sample needs to be manually tuned [1]. KBRL is proposed using a kernel method with an instance-based approach to approximate the value function offline [14]. Gaussian process temporal difference (GPTD) consists of using Gaussian process regression to approximate the value function and using an approximate

Manuscript received May 28, 2012; revised January 12, 2013 and March 26, 2013; accepted June 18, 2013. Date of publication July 29, 2013; date of current version November 1, 2013. This work was supported in part by the National Science Foundation of China under Grants 61035003, 61175042, and 60721002, the National 973 Program of China under Grant 2009CB320702, the 973 Program of Jiangsu, China under Grant BK2011005, and the Program for New Century Excellent Talents in University under Grant NCET-10-0476.

X. Chen and Y. Gao are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China (e-mail: chenxgspring@gmail.com; gaoy@nju.edu.cn). *(Corresponding author: Y. Gao.)*

R. Wang is with the School of Engineering of Advanced Technology, Massey University, Palmerston North 4442, New Zealand (e-mail: r.wang@massey.ac.nz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2013.2270561

TABLE I
COMPARISON OF VARIOUS KBRL METHODS

Algorithm	Kernel sparsification	Learning method
RBF network [1]	None	General function approximation
KBRL [14]	None	General reinforcement learning algorithms
GPTD [20]	ALD	Gaussian process temporal difference
KRR [22]	None	Support vector machine
KBPS [23]	None	Prioritized sweeping
Kernel-LSTD [24]	ALD	Least squares temporal difference
KLSPI [15]	ALD	Least squares policy iteration
GPDP [25]	None	Gaussian process dynamic programming
RKRL [26]	None	Gaussian process with sequential Monte Carlo
BRE(GP) [28]	None	Gaussian process regression
Kernel-Sarsa(λ) [31]	Projectron	Sarsa(λ)
NPDP [30]	None	Nadaraya-Watson kernel regression
OSKTD	Selective ensemble learning	TD(λ) using a selective kernel-based value function

linear dependence (ALD) method for kernel sparsification [20], [21]. An off-policy method, kernel rewards regression (KRR) considers a reinforcement learning problem as a regression task [22]. Kernel-based prioritized sweeping (KBPS) combines the strengths of both KBRL and prioritized sweeping, which features model-based exploration [23]. A sparse kernel-based LS-TD learning algorithm combines the least squares TD with the ALD method for kernel sparsification [24]. Kernel-based least squares policy iteration (KLSPI) uses the ALD method for sparsification, and the least-squares policy iteration for updating the parameters of the value function [15]. GPDP uses Gaussian processes to model the value function in the Bellman recursion of the dynamic programming algorithm [25]. replacing-kernel reinforcement learning (RKRL) is an online model selection method for GPTD using a sequential Monte-Carlo method [26]. An approach as Bellman residual elimination (BRE) rather than Bellman residual minimization [27] is introduced to KBRL, which emphasizes the fact that the Bellman error is explicitly forced to zero, and BRE(GP) is proposed based on Gaussian process regression [28]. A unifying view of the different approaches is proposed to kernelized value function approximation for RL and demonstrates that several model-free kernelized value function approximators can be viewed as special cases of a novel, model-based value function approximator [29]. Nonparametric dynamic programming (NPDP) is a nonparametric approach to policy evaluation, which uses kernel density estimation [30]. Kernel-SARSA(λ) [31] is a kernelized version of SARSA(λ) using sparse kernel projectron (a bounded kernel-based perceptron [32], [33]) techniques to permit kernel sparsification for arbitrary λ for $0 \leq \lambda \leq 1$.

From Table I, we can see that: 1) many algorithms do not have the procedure of kernel sparsification, which is crucial for online learning and 2) the ALD method is a common method for kernel sparsification in many RL algorithms. Note that the projectron is an approximation to ALD, while the surprise criterion is a novel method for kernel sparsification. However, the computational complexity of the ALD method, projectron and surprise criterion, is $O(n^2)$, where n is the size of the kernel dictionary [20], [32]–[34]. Obviously, these methods are then unable to response fast enough to real-time applications. This paper aims to solve this problem by trading off between accuracy and complexity.

In this paper, a KBRL algorithm is proposed, named OSKTD. In OSKTD, there are two procedures: online sparsification and parameters updating for the selective kernel-based value function. In the procedure of the online sparsification, a newly developed sparsification approach is proposed based on selective ensemble learning. Based on the proposed sparsification approach, a kernel distance-based online sparsification method, the complexity of which is $O(n)$, is proposed to construct the sparsified sample dictionary online by checking if a sample needs to be added to the sample dictionary.

The selective kernel-based value function is proposed based on local validity. In the procedure of parameters updating for the selective kernel-based value function, the classic framework of TD(λ) is applied to learning the parameters of the selective kernel-based value function. The experimental results compared with both traditional and up-to-date algorithms from Maze and Mountain Car demonstrate the effectiveness of the proposed OSKTD algorithm.

The rest of this paper is organized as follows. An introduction of RL, tabular and linear TD learning family and KBRL is given in Section II. In Section III, an online sparsification approach based on selective ensemble learning is proposed. Based on this approach, the kernel distance-based online sparsification method is proposed. In Section IV, based on local validity, a selective kernel-based value function is proposed and analyzed. The OSKTD algorithm is proposed in Section V. Section VI presents the experimental settings and results, and some analysis is given. Finally, the conclusion is given in Section VII.

II. RL AND KERNEL METHODS

We start with a short introduction to RL, temporal difference learning and KBRL.

A. RL and Markov Decision Process

RL in sequential decision making can typically be modeled as a Markov Decision Process (MDP) [1]. An MDP is a tuple $< S, A, R, P >$, where S is the state space of the environment, A is the action space of the agent, $R : S \times A \times S \to \mathbb{R}$ is a reward function, and $P : S \times A \times S \to [0, 1]$ is a state transition function.

A policy is a mapping $\pi : S \times A \to [0, 1]$. The goal of the agent is to find an optimal policy π^* to maximize the expectation of a discounted accumulative reward in a long period R: $\pi^* = \arg\max_\pi R_\pi = \arg\max_\pi E_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$,

where γ is a discount factor, r_t is the reward at time-step t, and $E_\pi[.]$ is the expectation with respect to the policy π.

In this paper, we focus on the state value function for a stationary policy π, which is defined as $V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$. The optimal value function is defined as $V^*(s) = E_{\pi^*}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$. According to the Bellman optimality equation

$$V^*(s) = \max_a E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^*(s')]. \quad (1)$$

B. Tabular and Linear TD Learning Family

Tabular TD(λ) learning is a classical learning algorithm to predict the value function in RL. In tabular TD(λ), there are two basic mechanisms: the TD and eligibility trace. The TD stands for the difference between the current prediction and its successive prediction $\delta_t \leftarrow r_t + \gamma \widetilde{V}_t(s_{t+1}) - \widetilde{V}_t(s_t)$, where s_{t+1} is the successive state of s_t, $\widetilde{V}(s_t)$ is the prediction of value function $V(s)$ at time-step t. An eligibility trace records the occurrence of an event, such as visiting a state or taking an action. The eligibility traces are defined for each state as follows:

$$e_t(s) \leftarrow \begin{cases} \gamma \lambda e_{t-1}(s), & \text{if } s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1, & \text{if } s = s_t. \end{cases} \quad (2)$$

The update rule for online TD(λ), for all s, is

$$\widetilde{V}_{t+1}(s) \leftarrow \widetilde{V}_t(s) + \alpha_t \delta_t e_t(s) \quad (3)$$

where α_t is a learning rate at time step t. As (3) shows, when a TD occurs, only the eligible states are assigned credit for the temporal difference.

The tabular TD learning algorithms cannot work for large scale or continuous RL problems [1]. Thus, value function approximators are commonly used to solve such problems. Here is an example, in which TD(λ) is combined with a linear function approximator, where the value function is represented as follows:

$$\widetilde{V}(s) = \Theta \Phi^T(s) = \sum_{i=1}^{M} \theta_i \phi_i(s) \quad (4)$$

where $\Phi(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_M(s)]$ is a vector of feature functions, M is the number of the feature function, and $\Theta = [\theta_1, \theta_2, \ldots, \theta_M]$ is a weight vector.

There are a number of algorithms in the linear TD learning family, e.g., linear TD(λ) [1], normalized TD [35], least squared TD (LSTD) [36], recursive LSTD [36], LSTD(λ) [37], incremental LSTD (iLSTD) [38], gradient TD (GTD) [39], second generation of GTD (GTD2) [40], and linear TD with gradient correction (TDC) [40]. The selected up-to-date algorithms (GTD, GTD2, and TDC) from the TD learning family are used to compare with the proposed algorithm because they can meet the online learning requirement as their complexity is $O(n)$, while the complexity of other algorithms including LSTD, RLSTD, and LSTD(λ) is $O(n^2)$ and iLSTD is with the computational complexity of $O(n)$, but still requires $O(n^2)$ memory.

1) Linear TD(λ): The update rules for the linear TD(λ) algorithm at each iteration t become

$$\Theta_{t+1} \leftarrow \Theta_t + \alpha_t \delta_t e_t \quad (5)$$

where the TD error is evaluated as

$$\delta_t \leftarrow r_t + \gamma \Theta \Phi^T(s_{t+1}) - \Theta \Phi^T(s_t) \quad (6)$$

and the eligibility traces are updated according to

$$e_{t+1} \leftarrow \gamma \lambda e_t + \Phi(s_{t+1}). \quad (7)$$

2) GTD, GTD2, and TDC: GTD, GTD2, and TDC aim to minimize the mean square projected Bellman error (MSPBE) [39], [40]. MSPBE$(\Theta) = ||\widetilde{V}_\Theta - \Pi T \widetilde{V}_\Theta||^2_D$ where T is the Bellman operator ($\widetilde{V} = R + \gamma P \widetilde{V} = T\widetilde{V}$) and Π is the projection operator ($\Pi = \Phi(\Phi^{\mathrm{T}} D \Phi)^{-1} \Phi^{\mathrm{T}} D$), where $\Phi = [\phi(s_1), \phi(s_2), \ldots, \phi(s_N)]^T$.

The update rules for GTD are as follows:

$$\Theta_{t+1} \leftarrow \Theta_t + \alpha_t (\phi_t - \gamma \phi_{t+1})(\phi_t^{\mathrm{T}} w_t) \quad (8)$$

with

$$w_{t+1} \leftarrow w_t + \alpha'_t (\delta_t \phi_t - w_t). \quad (9)$$

GTD2 updates the same Θ as GTD, but with a different w update

$$w_{t+1} \leftarrow w_t + \alpha'_t (\delta_t - \phi_t^{\mathrm{T}} w_t) \phi_t. \quad (10)$$

In TDC

$$\Theta_{t+1} \leftarrow \Theta_t + \alpha_t \delta_t \phi_t - \alpha_t \gamma \phi_{t+1} (\phi_t^{\mathrm{T}} w_t) \quad (11)$$

with w updated as in GTD2.

The convergence of the tabular and linear TD learning algorithms was well studied in [2]. The linear TD algorithm converges with probability 1, given the following assumptions.

1) The learning rates α_t are positive and deterministic (predetermined). In addition, we have $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} {\alpha_t}^2 < \infty$.
2) There exist positive real numbers $\pi(s_j)$ such that $\lim_{t \to \infty} P(s_t = s_j | s_0 = s_i) = \pi(s_j), \forall s_i, s_j \in S$.
3) $M \leq N$ and the matrix Φ has full rank, where N is the size of the state space S.

C. Kernel-Based Reinforcement Learning

As can be seen from Assumption 1, one of the conditions is that the matrix Φ should have a full rank. However, this condition is not easily satisfied in most of RL problems, especially with high dimensionality. Since kernel methods are able to deal with high-dimensional problems and are successfully applied into many machine learning algorithms (e.g., support vector machine), we focus on the KBRL in this paper.

There are two procedures in the KBRL: 1) dictionary construction and kernel-based value function approximation with respect to the constructed dictionary. In the former procedure and 2) a sample dictionary can be constructed online or offline. In this paper, we focus on online, which can be shown as in Fig. 1. When a new sample s_t is presented, it needs to decide whether the feature $\phi(s_t)$ of this sample needs to be added into the sample dictionary D_{t-1}, resulting $D_t = D_{t-1}$ or $D_t = D_{t-1} \cup \{\phi(s_t)\}$.

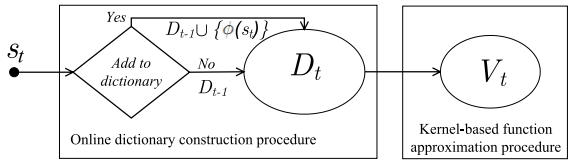


Fig. 1. Two procedures of online KBRL.

In the latter procedure, kernel-based value function V_t is used to predict the value of a sample and the parameters of the kernel-based value function are updated by RL. The kernel-based value function is defined as follows:

$$\widetilde{V}(s_t) = \Theta K^T(s_t) = \sum_{i=1}^{M} \theta_i k(s_t, s_i) \tag{12}$$

where s_i is corresponding to a feature vector $\phi(s_i)$ from the sample dictionary D, M is the size of the dictionary, $k(.,.)$ is a kernel function, and each $k(s_t, s_i)$ represents a basis function for state s_t corresponding to the basis function $\phi_i(s)$ in (4).

In KBRL, the objective is to find the optimal value function $V^*(s)$. To do this effectively, we need to consider the tradeoff between the complexity and accuracy of the approximate value function. The optimization of the value function can be divided into three issues: 1) how to construct a sample dictionary (online sparsification); 2) how to express the value function based on the sparsified sample dictionary; and 3) how to update the parameters of the value function. Issue 1) mainly focuses on the complexity, while issues 2) and 3) mainly focus on the accuracy. In this paper, we will address all the above issues and propose OSKTD learning.

III. Online Sparsification

Online sparsification aims to decrease the complexity of the value function. In this section, firstly, related work about online sparsification is summarized and the motivation of our research is given. Secondly, a new sparsification approach based on selective ensemble learning is proposed. Finally, a kernel distance-based online sparsification method is proposed based on the selective ensemble learning approach.

A. Related Work and Motivation

An open problem in KBRL is online kernel sparsification [15]. The general notion is that the kernel matrix, instead of all samples in the dictionary, should be fully ranked, according to the assumptions introduced in Section II-B. That is to ensure that any sample in the dictionary cannot be linearly represented by the other samples in the dictionary. Then, for online sparsification, the basic idea is that if a new coming sample can be linearly represented by the samples in the dictionary, it should not be added. However, to decide whether a sample can be linearly represented by the existing samples in a dictionary is computationally exhausted. We summarize related work about the kernel sparsification as follows.

1) ALD is used to construct a dictionary of representative states online, resulting from the approximate linear dependency condition in a feature space [20]. Suppose we have obtained a sample dictionary $D_{t-1} = \{\phi(s_1), \phi(s_2), \ldots, \phi(s_{M_{t-1}})\}$ at time step $t-1$, where the size of the dictionary is $|D_{t-1}| = M_{t-1}$. Then, the approximately linearly dependent condition for a new feature vector $\phi(s_t)$ is

$$\delta_t = \min_c || \sum_j c_j \phi(s_j) - \phi(s_t)||^2 \leq \mu \tag{13}$$

where $c = [c_j]$ and μ are the thresholds to determine the approximation accuracy and sparsity level. When the condition (13) is satisfied, the feature vector $\phi(s_t)$ can be ignored; otherwise, it is added into the dictionary. The complexity of ALD at each step is mainly caused by the computation of the inverse kernel matrix, which complexity is $O(n^2)$, where n is the size of the kernel dictionary [21].
2) Kernel principal component analysis (KPCA) [41] is an extension of principal component analysis (PCA) using kernel methods. The originally linear operations of PCA are conducted in a reproducing kernel Hilbert space with a nonlinear mapping. The standard KPCA computation method [41] via eigendecompositions involves a time complexity of $O(n^3)$, where n is the size of training vector. Further, an incremental version of the KPCA is proposed for image processing [42].
3) Novelty criterion (NC) [34] considers both the minimum distance and prediction error. Only if the minimum distance is larger than a specific threshold value and the prediction error is larger than another specific threshold value, the input state is added into the dictionary.

The complexity of ALD, KPCA, and NC are $O(n^2)$, $O(n^3)$ and $O(n)$, respectively. Different from KPCA and NC, the ALD method is combined with RL. In [20], [43], [44], ALD was used as an online method for sparsification. In [15], ALD was used as an offline method for sparsification. However, ALD is still computational exhausted. Thus it is unable to response fast enough to real-time applications. Therefore, in this paper, we propose the OSKTD learning algorithm, where the procedure of the online sparsification is a modified version of NC based on the selective ensemble learning approach and its complexity is $O(n)$, thus it is a qualified online sparsification method.

B. Kernel Sparsification Based on Selective Ensemble Learning

In this section, selective ensemble learning (SEL) is briefly introduced and the new SEL approach for kernel sparsification is proposed.

In supervised learning, ensemble learning [45], [46] trains a finite number of weak learners and combines their results in an attempt to produce a strong learner. It is proved [45] that if each weak learner can get the prediction accuracy more than 0.5, and if all weak learners are independent, then the more weak learners used, the lower the likelihood of a generalization error by an averaged the predictions of all the weak learners. In the limit of an infinite number of learners N, the ensemble

error rate goes to 0. This means that there are two ways to enhance the generalization ability of ensembles: 1) improve the prediction accuracy of each weak learner and 2) decrease the correlation between each learner.

In KBRL, the value function is

$$\widetilde{V}(s_t) = \sum_{i=1}^{M} \theta_i k(s_t, s_i) = \sum_{i=1}^{M} w_i \big(1/w_i \theta_i k(s_t, s_i)\big)$$

where w_i is a weight, $0 < w_i < 1$, and $\sum_{i=1}^{M} w_i = 1$; each s_i is corresponding to the element $\phi(s_i)$ of the dictionary D, $M = |D|$ is the size of dictionary. Let each

$$v_i(s_t) = \frac{1}{w_i} \theta_i k(s_t, s_i) \tag{14}$$

be a weak learner, where $1 \le i \le M$, then

$$\widetilde{V}(s_t) = \sum_{i=1}^{M} w_i v_i(s_t)$$

can be a strong learner.

Theorem 1: Based on the ensemble learning [47], [48], the generalization error of the value function $\widetilde{V}(s)$ in reinforcement learning is

$$E = \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij} \tag{15}$$

where

$$C_{ij} = \int p(s) \big(v_i(s) - V(s)\big) \big(v_j(s) - V(s)\big) ds \tag{16}$$

is the correlation between the weak learners v_i and v_j.

Proof: The generalization error of the value function is defined as the mean squared error [1][1]: $E = \int p(s) \big(\widetilde{V}(s) - V(s)\big)^2 ds$, where $p(s)$ is the distribution of the state s, and $V(s)$ is the true value of the state s. Suppose, $w_i = 1/M$. The generalization error can be derived as

$$\begin{aligned} E &= \int p(s) \big(\widetilde{V}(s) - V(s)\big)^2 ds \\ &= \int p(s) \left(\sum_{i=1}^{M} \frac{1}{M} v_i(s) - V(s) \right)^2 ds \\ &= \int p(s) \left(\sum_{i=1}^{M} \frac{1}{M} \big(v_i(s) - V(s)\big) \right)^2 ds \\ &= \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \int p(s) \big(v_i(s) - V(s)\big) \big(v_j(s) - V(s)\big) ds \\ &= \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij}. \end{aligned}$$

■

Theorem 1 indicates that the less correlation between each weak learners, the less generalization error of the ensemble.

Theorem 2: Based on selective ensemble learning [47], the kth sample associated with the kth value function v_k should be removed from the sample dictionary D without sacrificing the generalization ability, under the condition

$$(2M - 1) \sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij} \le 2M^2 \sum_{i=1, i \ne k}^{M} C_{ik} + M^2 E_k \tag{17}$$

where E_i is the generalization error of the ith learner v_i

$$E_i = \int p(s) (v_i(s) - V(s))^2 ds. \tag{18}$$

Proof: Denote E' as the generalization error of the ensemble $\{v_i\}_{i=1, i \ne k}^{M}$, where the value function v_k is removed. According to Theorem 1

$$\begin{aligned} E' &= \frac{1}{(M-1)^2} \sum_{i=1, i \ne k}^{M} \sum_{j=1, j \ne k}^{M} C_{ij} \\ &= \frac{1}{(M-1)^2} \left(M^2 E - (2 \sum_{i=1, i \ne k}^{M} C_{ik} + E_k) \right). \end{aligned}$$

According to (17)

$$\begin{aligned} E' &\le \frac{1}{(M-1)^2} \left(M^2 E - \frac{2M-1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij} \right) \\ &= \frac{1}{(M-1)^2} \big(M^2 E - (2M-1)E\big) = E. \end{aligned}$$

It indicates that the weak value function v_k can be removed without sacrificing the generalization ability. From (14), we can see that each weak value function v_i is corresponding to a sample $\phi(s_i)$ in the sample dictionary D because $k(s, s_i) = \phi(s)\phi(s_i)$. Thus, the kth sample can be removed without sacrificing the generalization ability under the condition (17). ■

The kernel sparsification approach based on selective ensemble learning is proposed according to Theorem 2 as follows:

In the offline case, the procedure of kernel sparsification can be done in two steps: 1) to train all the weak learners and 2) to continue to remove the bad samples from the sample dictionary until there are no bad samples, which are corresponding to bad learners according to (17).

In the online case, the procedure of kernel sparsification can also be done in two steps: 1) to train the new weak learner (corresponding to the new coming sample) and 2) to check the condition (17), if it holds, the new samples will not be added to the sample dictionary.

Note that it is difficult to directly compute each element of the condition (17) in RL, because: 1) it is computationally exhausted for training the weak learner $v_i(s)$; 2) the true value function $V(s)$ cannot be obtained directly; and 3) the computation for each correlation C_{ij} is exhausted. In the next section, inspired by the approach of selective ensemble learning, a kernel distance-based method is proposed for online sparsification.

C. Kernel Distance-Based Online Sparsification

For the derivation of the generalization error, (15) indicates that the less correlation between each weak learners, the

[1]It is straightforward to extend the result for the discrete case.

less generalization error of the ensemble. Thus, one way to decrease the generalization error is to decrease the correlation between learners.

Corollary 1: According to (16) and (18), the correlation has the property $2C_{ij} \leq E_i + E_j$.

Proof:

$$\begin{aligned} 2C_{ij} &= \int p(s) 2\big(v_i(s) - V(s)\big)\big(v_j(s) - V(s)\big) ds \\ &\leq \int p(s)\Big(\big(v_i(s) - V(s)\big)^2 + \big(v_j(s) - V(s)\big)^2\Big) ds \\ &= E_i + E_j. \end{aligned}$$

■

It indicates that two learners will get a higher correlation if they have closer performances. When $E_i = E_j$, $E_i + E_j = 2C_{ij}$. Besides, from (14), we can see the fact that if two samples get closer to each other, the corresponding weak learners (if they are independently trained by the same learning technique) will get closer performances. Therefore, one way to decrease the generalization error is to enlarge the distance (in the reproducing kernel Hilbert space) between the samples in the sample dictionary.

We thus propose an alternative approach, which is a kernel distance-based online sparsification method. Only if a new coming sample is far away in the reproducing kernel Hilbert space (i.e., the kernel distance is greater than a threshold) from all existing samples in the sample dictionary, it will be added into the dictionary.

In the kernel distance-based online sparsification, we start with an empty sample dictionary $D_0 = \emptyset$. Suppose we have obtained a sample dictionary $D_{t-1} = \{\phi(s_1), \phi(s_2), \ldots, \phi(s_{M_{t-1}})\}$ at time step $t-1$ where the size of the dictionary $|D_{t-1}| = M_{t-1}$. The kernel distance-based condition for a new feature vector $\phi(s_t)$ is $\delta_t = \min_{s_i \in D_{t-1}} ||\phi(s_i) - \phi(s_t)||^2 < \mu_1$ where μ_1 is a threshold value.

Using the kernel trick, by substituting $k(s_i, s_t) = \phi(s_i)^T \phi(s_t)$, we can obtain the following:

$$\begin{aligned} \delta_t &= \min_{s_i \in D_{t-1}} ||\phi(s_i) - \phi(s_t)||^2 \\ &= \min_{s_i \in D_{t-1}} \big(\phi(s_i)\phi(s_i) - 2\phi(s_i)\phi(s_t) + \phi(s_t)\phi(s_t)\big) \\ &= \min_{s_i \in D_{t-1}} \{k(s_i, s_i) - 2k(s_i, s_t) + k(s_t, s_t)\}. \end{aligned} \tag{19}$$

Thus, the update rule for the sample dictionary D is

$$D_t = \begin{cases} D_{t-1}, & \text{if } \delta_t < \mu_1 \\ D_{t-1} \cup \{\phi(s_t)\}, & \text{otherwise} \end{cases} \tag{20}$$

where μ_1 is a threshold value.

This method can be viewed as a modified version of NC [34]. NC considers both the distance and the prediction error. Since the prediction error is not included into our kernel sparsification process. Thus, the prediction error is ignored in this paper. The computational complexity of the kernel distance-based kernel sparsification procedure is $O(n)$, which enables kernel sparsification online. It has less computational complexity compared with the ALD method because the computational complexity of ALD is $O(n^2)$ based on (13). However, there are no guarantees for linear independence of the samples in the sample dictionary using the kernel distance-based method, which could lead to a challenge for the convergence of RL algorithms. To ease such a problem, we propose a selective kernel-based value function based on local validity in the next section.

IV. Selective Kernel-Based Value Function

Our approach is based on the tradeoff between complexity and accuracy. The new kernel sparsification approach based on selective ensemble learning is proposed mainly to decrease the complexity, while the selective kernel-based value function based on local validity is used to provide a better accuracy.

A. Local Validity and its Generalization Ability

In selective ensemble learning, local validity is proposed through assembling weak learners from a local instance space in order to improve the generalization ability. In this paper, we extend this result to reinforcement learning.

Assumption 1:

1) The state space S is divided into local subspace $\{S_1, S_2, \ldots, S_n\}$, where $S = \cup S_i$ and $S_i \cap S_j = \emptyset$ for any $1 \leq i, j \leq n$ and $i \neq j$.
2) Each $\widetilde{V}^{\text{opt},S_i}$ is the optimal approximated value function for the local subspace S_i learned by a RL method, e.g., linear TD

$$\widetilde{V}^{\text{opt},S_i} = \arg\min_{\widetilde{V}} \int_{S_i} p(s)(\widetilde{V} - V(s))^2 ds \tag{21}$$

where $p(s)$ is the sampling distribution, and $V(s)$ is the true value for state s. 3) $\widetilde{V}^{\text{opt},S}(s)$ is the optimal approximated value function for the whole space S learned by the same RL method

$$\widetilde{V}^{\text{opt},S} = \arg\min_{\widetilde{V}} \int_{S} p(s)(\widetilde{V} - V(s))^2 ds. \tag{22}$$

Theorem 3: The generalization error E^* for a set of optimal value functions $\widetilde{V}^{\text{opt},S_i}$ defined corresponding to each local subspace S_i is not greater than the generalization error E^{RL} for one optimal value function $\widetilde{V}^{\text{opt},S}(s)$ defined on the whole space S. That is $E^* \leq E^{\text{RL}}$ where

$$E^* = \sum_{S_i} \int_{S_i} p(s)(\widetilde{V}^{\text{opt},S_i}(s) - V(s))^2 ds \tag{23}$$

and

$$E^{\text{RL}} = \int p(s)(\widetilde{V}^{\text{opt},S}(s) - V(s))^2 ds. \tag{24}$$

Proof: Because of Assumption 1, the generalization error E of the value function $\widetilde{V}(s)$ can be rewritten as follows:

$$\begin{aligned} E &= \int p(s)(\widetilde{V}(s) - V(s))^2 ds \\ &= \sum_{S_i} \int_{S_i} p(s)(\widetilde{V}(s) - V(s))^2 ds. \end{aligned} \tag{25}$$

RL algorithms are to find the optimal $\widetilde{V}^{\text{opt},S}$ in an attempt to decrease the generalization error (25).

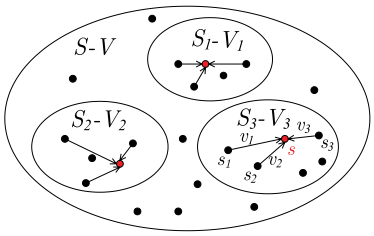


Fig. 2. Space S consists of the local subspaces $S_1, S_2, \ldots$, and the value function V defined on S consists of $V_1, V_2, \ldots$, where each V_i is defined under the corresponding local subspace S_i.

Then, (24) can be rewritten as

$$E^{\mathrm{RL}} = \sum_{S_i} \int_{S_i} p(s)(\widetilde{V}^{\mathrm{opt},S}(s) - V(s))^2 ds. \tag{26}$$

Recall $\widetilde{V}^{\mathrm{opt},S_i}$ is the optimal approximated value function in local subspace S_i as shown in (21). That is for any $S_i \subset S$ and any approximated value function $\widetilde{V}$

$$\int_{S_i} p(s)(\widetilde{V}^{\mathrm{opt},S_i}(s) - V(s))^2 ds \leq \int_{S_i} p(s)(\widetilde{V}(s) - V(s))^2 ds. \tag{27}$$

$\widetilde{V}^{\mathrm{opt},S}$ is the optimal approximated value function in the whole space S as shown in (22). Simultaneously, it is also an approximated value function in S_i. Then

$$\int_{S_i} p(s)(\widetilde{V}^{\mathrm{opt},S_i}(s) - V(s))^2 ds \leq \int_{S_i} p(s)(\widetilde{V}^{\mathrm{opt},S}(s) - V(s))^2 ds. \tag{28}$$

Thus, $E^* \leq E^{\mathrm{RL}}$.

The core of Theorem 3 is that the optimization objectives are different on the left-hand side and right-hand side of (28). Theorem 3 indicates that it will get a better generalization ability when optimizing each value function (to obtain a global value function), respectively, for each local subspace of the state space. ■

Then, to get a better accuracy, the value function can be divided into many value functions, which are defined under the corresponding local subspaces of the state space. Based on such an approach, a selective kernel-based value function is obtained in this paper.

B. Selective Kernel-Based Value Function

Based on the local validity approach, we can decrease the overall generalization error by optimizing the value functions respectively for each local subspace of the state space. Assume that the space S is divided into n local subspace $S_i \subset S$ where $S = \cup S_i$ and $S_i \cap S_j = \emptyset$ for any $1 \leq i, j \leq n$ and $i \neq j$. In each local subspace S_i, the value function is noted as $V_i(s)$, Fig. 2. Then, the objective is to find optimal value function $V_i^*(s)$ for each local subspace S_i

There are two approaches to optimize each $V_i^*(s)$: 1) given a sample dictionary D, the optimal $V_i^*(s)$ is then related to the parameter vectors Θ_i for each $V_i^*(s)$. The result is a set of optimal vectors $\{\Theta_i\}_{i=1}^n$ and (2) without restrictions to the sample dictionary, the optimal value function $V_i^*(s)$ then can be optimized by two procedures. One is online construction of the sample subdictionary D_i for each local subspace S_i. The other is to optimize the parameter vectors Θ_i based on the sample subdictionary D_i for each local subspace S_i. The result is a set of dictionary-parameter pairs $\{< D_i, \Theta_i >\}_{i=1}^n$.

Note that the motivation of this paper is to give a solution to online learning and give a unified kernel-based value function. Thus, we take approach 2) to optimize each $V_i^*(s)$.

Since the procedure of the online construction of the sample dictionary is considered in Section III, assume we have obtained the sparsified sample subdictionary D_i for each local subspace S_i, $D = \cup D_i$, where $D_i \cap D_j = \emptyset$ for any $1 \leq i, j \leq n$ and $i \neq j$. Thus, the value function can be unified as: $\widetilde{V}(s) = \sum_{i=1}^n V_i(s)$ where

$$V_i(s) = \begin{cases} \sum_{s_j \in D_i} \theta_j k(s_j, s), & s \in S_i \\ 0, & \text{otherwise.} \end{cases} \tag{29}$$

Function approximation $\widetilde{V}(s)$ can be viewed as a procedure of selecting a value function V_i associated with the local subspace S_i, where $s \in S_i$. Since each D_i is a sparsified subdictionary in the local subspace S_i, it is equal to select subdictionary D_i. Note that selecting a subdictionary D_i corresponding to the local subspace S_i depends on the division of the state space (into local subspaces). In this paper, a selective function $\beta(.,.)$, which selects a best subset for the new coming sample s from the sample dictionary D, is proposed to facilitate flexibility for the division of the state space

$$\beta(s, s_i) = \begin{cases} 1, & s_i \text{ is appropriate to approximate } s \\ 0, & \text{otherwise} \end{cases} \tag{30}$$

where $s \in S$, $s_i \in D$.

Therefore, the general form of the value function, selective kernel-based value function, is defined as

$$\widetilde{V}(s) = \sum_{i=1}^{M} \theta_i \beta(s, s_i) k(s, s_i) \tag{31}$$

where $\beta(.,.)$ is described as (30), and $M = |D|$ is the size of the sample dictionary D.

Note that the state s can be replaced by an explicit feature abstraction function $\phi(s)$ in (31).

C. Properties of the Selective Kernel-Based Value Function

In this subsection, we analyze two properties of the selective kernel-based value function: 1) the selective function builds a bridge between the tabular value function and kernel-based value function and 2) the selective function combined with the kernel function can be viewed as kernel redefinition.

1) Relationship With the Tabular Value Function: Firstly, let us define a special sample dictionary D as follows: $D = \{\phi(s_1), \phi(s_2), \ldots, \phi(s_N)\} = \{s_1, s_2, \ldots, s_N\}$, where N is the size of the state space S and $\phi(s) = s$. In other words, D is the same as the state space S.

Secondly, assume any $x, y \in S$

$$\beta(x, y) = \begin{cases} 1, & x = y \\ 0, & \text{otherwise.} \end{cases} \tag{32}$$

Finally, considering (31), we obtain $\widetilde{V}(s) = \theta_i$, where i satisfies $s = s_i$, given the fact $k(s, s) = 1$. This shows that each state s of the state space S is associated with a parameter θ, which is a classic form of the tabular value function. Thus, the tabular value function is a special case of the selective kernel-based value function.

2) Approach of Kernel Redefinition: Let us consider the elements $\beta(s, s_i)$ and $k(s, s_i)$ in the value function (31). Let $K(x, y) = \beta(x, y)k(x, y)$, where $k(., .)$ is a kernel function, then $V(s) = \sum_{i=1}^{M} \theta_i K(s, s_i)$. According to Mercer's Theorem, $k(x, y)$ is symmetric and positive semidefinite. That is, $k(x, y)$ satisfies the following two properties:

1) Symmetric

$$k(x, y) = k(y, x). \tag{33}$$

2) Cauchy-Schwarz Inequality

$$k(x, y)^2 \leq k(x, x)k(y, y). \tag{34}$$

Considering (30), the selective function has the following properties:

$$\beta(x, y) = \beta(y, x) \tag{35}$$

$$\beta(x, x) = 1 \tag{36}$$

$$\beta(x, y) \leq \beta(x, x). \tag{37}$$

Considering (33), (34), (36), (37), and (37), obviously

$$K(x, y) = \beta(x, y)k(x, y) = \beta(y, x)k(y, x) = K(y, x) \tag{38}$$

$$\begin{aligned} K(x, y)^2 &= \beta(x, y)^2 k(x, y)^2 \\ &\leq \beta(x, x)\beta(y, y)k(x, x)k(y, y) \\ &= \beta(x, x)k(x, x)\beta(y, y)k(y, y) \\ &= K(x, x)K(y, y). \end{aligned} \tag{39}$$

Hence, $K(x, y) = \beta(x, y)k(x, y)$ is a redefined kernel function. Thus, the selective kernel-based value function (31) can be viewed as an approach of the kernel function redefinition for the traditional kernel-based value function (12).

V. Online Selective Kernel-Based Temporal Difference Learning

There are four issues in applying the selective kernel-based value function to online KBRL: 1) online sparsification: the construction of the sample dictionary online; 2) kernel function $k(., .)$: in different problems, using different kernel functions can have different performances; 3) selective function $\beta(., .)$: to decide which samples should be used to compute the value of a new coming sample. This can also be different for different problems; and 4) learning algorithm: to update the weight $\Theta = [\theta_1, \theta_2, \ldots, \theta_M]$, where M is the size of the sample dictionary.

Issue 1) is discussed in Section III. We will address issues 2) and 3) in Section V-A and propose OSKTD learning to address issue 4) in Section V-B.

A. Details of the Selective Kernel-Based Value Function

According to the approach of kernel redefinition, the redefined kernel function of the selective kernel-based value function is $K(s, s_i) = \beta(s, s_i)k(s, s_i)$, which consists of two parts: the conventional kernel function $k(., .)$ and the selective function $\beta(., .)$.

The first part is the basic kernel function, which is a conventional Mercer kernel function, e.g., Gaussian kernel (40), polynomial kernel (41) or inverse multiquadric kernel (42), etc

$$k(s, s') = e^{-\frac{||s-s'||^2}{2\sigma^2}} \tag{40}$$

$$k(s, s') = (s^T s' + c)^d \tag{41}$$

$$k(s, s') = \frac{1}{\sqrt{||s - s'||^2 + c^2}}. \tag{42}$$

The second part is the selective function, which is described as (30). Recall that the objective is to find an optimal value function $\widetilde{V}^*(s)$, which minimizes the generalization error (25). Using the selective kernel-based value function, the generalization error can be derived as

$$E = \int p(s) \left(\sum_{i=1}^{M} \theta_i \beta(s, s_i)k(s, s_i) - V(s) \right)^2 ds. \tag{43}$$

In this paper, a kernel distance-based selective method is proposed for the selective function, where the kernel distance $d(s_t, s_i)$ for the state s_t and each $s_i \in D$ at time step t is

$$d(s_t, s_i) = \phi(s_t)^T \phi(s_t) - 2\phi(s_t)^T \phi(s_i) + \phi(s_i)^T \phi(s_i). \tag{44}$$

Using the kernel trick, $d(s_t, s_i) = 2 - 2k(s_t, s_i)$. The kernel distance-based selective method is proposed given the fact that a sample of the dictionary, which is too far from the new sample in the reproducing kernel Hilbert space, is less relevant to this sample. Thus, it has less representation ability. Then, the selective function can be defined as

$$\beta(s_t, s_i) = \begin{cases} 1, & d(s_t, s_i) < \mu_2 \\ 0, & \text{otherwise} \end{cases} \tag{45}$$

where μ_2 is a threshold value.

Besides, k-nearest neighbor (KNN) method is to select KNNs of the current state s_t from the sample dictionary D for the selective kernel-based value function based on the kernel distance, where k is the parameter. In some cases, there can be some heuristic knowledge telling us which two are near or faraway for solving a problem. For example, in the Maze problem, two grids that are near in the real world may be not reachable if they are blocked by a wall. The selective function can combine the prior knowledge with the distance-based selective method.

B. OSKTD Algorithm

In the following, we try to address issue (iv). The parameter vector μ of the selective function $\beta(., .)$ is a threshold manually set as a small value. Thus, the objective is to optimize Θ for (43). We focus on the traditional RL paradigm and the classical framework of TD(λ) [1].

The objective is to find a global optimal parameter Θ^*, for which $E(\Theta^*) \leq E(\Theta)$ for all Θ. Gradient-descent methods try

1. Initialize parameters (ϵ, α, λ, γ, μ_1, μ_2),
empty sample dictionary D
2. Repeat(for each episode):
2.1 $s \leftarrow$ initial state in episode Initialize s,a
2.2 Repeat(for each step of episode):
2.2.1 $a \leftarrow$ action given by π (e.g., ϵ-greedy) for s
2.2.2 Take action a, observe reward r, and next state, s'
2.2.3 $\delta \leftarrow r + \gamma \widetilde{V}(s') - \widetilde{V}(s)$,
where $\widetilde{V}(s)$ is computed by (31)
2.2.4 for each element $d_i \in D$
2.2.4.1 $s_i, e_i, \theta_i \leftarrow$ get state,
eligibility trace and weight associated with d_i
2.2.4.2 $e_i \leftarrow \gamma\lambda e_i + \beta(s, s_i)k(s_i, s)$
2.2.4.3 $\theta_i \leftarrow \theta_i + \alpha\delta e_i$
2.2.5 compute the minimum distance according to (19)
2.2.6 update the sample dictionary D according to (20)
2.2.7 $s \leftarrow s'$
2.3 Until s is terminal

Fig. 3. Pseudocode for the OSKTD learning algorithm.

to adjust the parameter after each example by a small amount in the direction that would decrease the error on that example

$$\Theta_{t+1} \leftarrow \Theta_t + \alpha(V(s_t) - \widetilde{V}(s_t)) \bigtriangledown_{\Theta_t} \widetilde{V}(s_t) \tag{46}$$

where α is the learning rate; the temporal difference error $(V(s_t) - \widetilde{V}(s_t))$ is replaced by $\delta = r + \gamma \widetilde{V}(s') - \widetilde{V}(s)$, where γ is a discounted factor.

According to (31), $\bigtriangledown_{\Theta_t} V_t(s_t) = [\beta(s, s_i)k(s, s_i)]_{i=1}^{M\ T}$. Thus, the update rule for each $\theta_i \in \Theta$ based on the gradient descent TD(λ) is

$$\theta_i \leftarrow \theta_i + \alpha\delta_t e_i \tag{47}$$

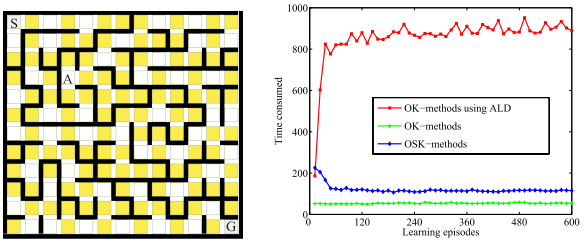
where $\delta_t \leftarrow r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$, $e_i \leftarrow \gamma\lambda e_i + \beta(s, s_i)k(s, s_i)$ and each θ_i, e_i is associated with the sample $\phi(s_i)$ in the sample dictionary D.

Fig. 3 shows the pseudocode of the OSKTD learning algorithm. OSKTD is based on the traditional RL paradigm and the classical framework of TD(λ), where the value function is the selective kernel-based value function described as (31), a kernel distance-based kernel sparsification method is proposed for kernel sparsification, and the parameters of the selective kernel-based value function are updated by the gradient descent technique.

Note that OSKTD is a method for prediction. For learning control problems, OSKTD can be easily combined with some conventional control methods, such as on-policy [SARSA(λ)] and off-policy [Watkins's Q(λ)] [1]. When OSKTD is used for learning control, the following steps need to be passed: 1) the state value function, $\widetilde{V}(s)$, is replaced by the state-action value function, $\widetilde{Q}(s, a)$; 2) the successive state value function $\widetilde{V}(s')$ is replaced by a successive state-action value function $\widetilde{Q}(s', a')$ (i.e., SARSA) or $\max_{a'} \widetilde{Q}(s', a')$ (i.e., Q-learning); and 3) each sample, $\phi(s_i)$ in the sample dictionary D is replaced by state-action pair in the feature space, $\phi(< s_i, a_i >)$.

VI. Experiments

In this paper, we use two experiments to demonstrate the effectiveness of the OSKTD algorithm: Maze and Mountain Car, which are corresponding to a traditional discrete RL problem and a continuous RL problem, respectively [1]. Mountain Car is one of the domains from the standard benchmarks in the NIPS Workshop 2005. They are used to evaluate the performances of various reinforcement learning algorithms [1], [20]–[23], [26], [31]. Note that OSKTD is a method for prediction. In this section, OSKTD combined SARSA(λ) learning (but still noted as OSKTD) is used to learn for control in Maze and Mountain Car.



(a) (b)

Fig. 4. (a) Maze with 111 samples in yellow. (b) Times curves of different algorithms in Mountain Car.

Algorithms used to compare with OSKTD include: 1) traditional algorithms, such as Q-learning and TD(λ) and 2) up-to-date algorithms, e.g., GTD, GTD2, and TDC. The reason why we select GTD, GTD2, and TDC to conduct comparisons is that the proposed OSKTD algorithm can meet the online learning requirement as its complexity is $O(n)$ including both the sparsification procedure and the parameters learning procedure. The complexity of other algorithms including LSTD, RLSTD, and LSTD(λ) is $O(n^2)$ and iLSTD is with the computational complexity of $O(n)$ but still requires $O(n^2)$ memory. In this paper, OSK-GTD, OSK-GTD2, and OSK-TDC refer to the online selective kernel-based value function combined with algorithms, i.e., GTD, GTD2, and TDC, respectively. OK-TD, OK-GTD2, OK-TDC, and OK-GTD mean that there is not a selective function $\beta(.,.)$ in the kernel based value function. In the experiments, parameters are varied to demonstrate the robustness of the proposed algorithm.

A. Experimental Settings

The same settings for each experiment are: 1) the curves are generated by averaging the sum of rewards (or running time) for five runs; 2) the discount factor γ is 0.9; and 3) an episode is limited to 1000 steps.

1) Maze: Consider a maze shown in Fig. 4(a), where S is the start position; G is the goal position, and A is the position of the agent. The objective of the learning agent is to escape from the maze as soon as possible. In each position, there are four actions, up, down, left, and right, which takes the agent deterministically to the corresponding neighbor position, except when a movement is blocked by an obstacle or the edge of the maze. Reward is -1 on all transitions, except those into the goal state, which reward is 0.

In the implementation of the OSKTD algorithm, settings are summarized as follows: 1) the ultimate sample dictionary is sparsified shown as yellow grids in Fig. 4(a), the number of samples is 111; 2) the selective function in the selective kernel-based value function: to compute state x. We send a

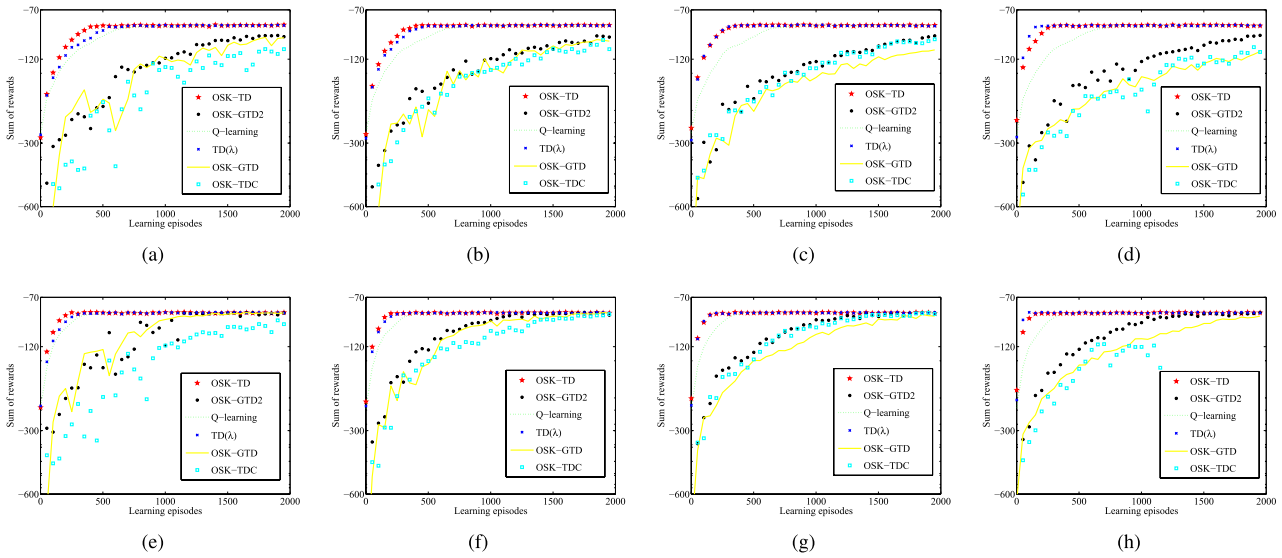


Fig. 5. Learning curves of different algorithms in Maze, where $\alpha \in \{0.1, 0.2\}$; $\alpha' \in \{0.02, 0.05, 0.5, 1.0\}$; and $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$. (a) $\alpha = 0.1$, $\alpha' = 0.02$ and $\lambda = 0.2$. (b) $\alpha = 0.1, \alpha' = 0.05$ and $\lambda = 0.4$. (c) $\alpha = 0.1, \alpha' = 0.05$ and $\lambda = 0.6$. (d) $\alpha = 0.1, \alpha' = 1.0$ and $\lambda = 0.8$. (e) $\alpha = 0.2, \alpha' = 0.02$ and $\lambda = 0.2$. (f) $\alpha = 0.2, \alpha' = 0.05$ and $\lambda = 0.4$. (g) $\alpha = 0.2, \alpha' = 0.5$ and $\lambda = 0.6$. and (h) $\alpha = 0.2, \alpha' = 1.0$ and $\lambda = 0.8$.

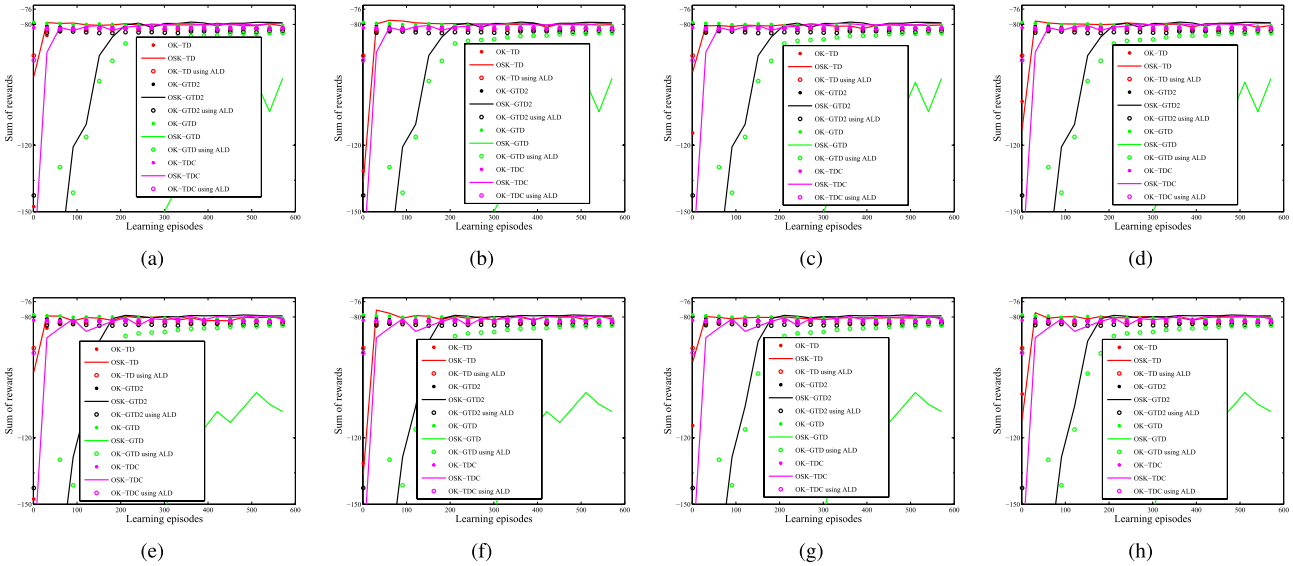


Fig. 6. Learning curves of different algorithms in Mountain Car, where $\alpha = 0.001$, $\alpha' = 0.01$; $\mu_1 = 0.04$, $\mu_2 \in \{0.08, 0.1\}$, and $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$. (a) $\lambda = 0.2\mu_2 = 0.08$. (b) $\lambda = 0.4\mu_2 = 0.08$. (c) $\lambda = 0.6\mu_2 = 0.08$. (d) $\lambda = 0.8\mu_2 = 0.08$. (e) $\lambda = 0.2\mu_2 = 0.1$. (f) $\lambda = 0.4\mu_2 = 0.1$. (g) $\lambda = 0.6\mu_2 = 0.1$. and (h) $\lambda = 0.8\mu_2 = 0.1$.

query to the sample dictionary. If x exists in the dictionary, we directly obtain the corresponding weight θ. Otherwise, we select the conjoined states y from the sample dictionary, excluding the states which are blocked with x; and 3) the basic kernel function in the selective kernel-based value function, the inverse multiquadric kernel, is used to compute the similarity between the grids x and y: $k(x, y) = 1/(\sqrt{||x - y||^2 + c^2})$, where c is a constant.

2) Mountain Car: In the Mountain Car simulation [1], the learning agent needs to drive an underpowered car up a steep mountain road. The reward in this problem is -1 for all time steps until the car reaches its goal position at the top of the mountain, then the episode ends. There are three possible actions: full throttle forward ($+1$), full throttle reverse (-1), and zero throttle (0). The car moves based on simple physics. Its position x_t, and velocity $\dot{x}_t$ are updated by: $x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}]$, $\dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t - 0.0025\cos(3x_t)]$ where the bound operation enforces $-1.2 \leq x_{t+1} \leq 0.5$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. The car starts at the position $x_0 = -0.3$, and its goal position is $x_T = 0.5$.

In the implementation of the OSKTD algorithm, settings are summarized as follows: 1) in action selection, the ϵ-greedy method is used based on the values of the afterstates, where $\epsilon = 0.1$ and 2) Gaussian kernel function is used as the base kernel function $k(.,.)$ to compute the similarity of the positions (40): $k(s, s') = e^{-(||s-s'||^2)/2\sigma^2}$ where $\sigma = 1$, each sample is a state s.

B. Experimental Results and Analysis

1) Maze: The reasons that we select Q-learning and TD(λ) as comparison can be explained as follows: 1) traditional table-based RL algorithms performed well in this problem [1]; 2) the table-based approach can be seen as a special case of OSKTD as described in Section IV-C.1; and 3) based on our experiments, we find that the kernel methods (including kernel-based TD, GTD, GTD2, and TDC) without a selective function cannot handle Maze regardless using the ALD sparsification method or the distance-based one.[2]

The learning curves for different parameters α, α', and λ are shown in Fig. 5. We can find: 1) Q-learning can be seen as a baseline as there are not the parameters α' and λ; 2) OSKTD is better than tabular TD(λ) in terms of learning speed when $\lambda \leq 0.6$, see Fig. 5. (a)-(h). In addition, they are obviously better than Q-learning; 3) with the increase of the learning rate α, the convergence of all algorithms to the optimal policy goes faster, except that OSK-TDC becomes unstable when $\alpha = 0.2$ and $\alpha' = 1.0$; and 4) in all curves, the up-to-date algorithms (i.e., GTD, GTD2, and TDC) combined with the selective kernel-based value function converge slower than Q-learning. The reason is that the updates of the parameter vector in these algorithms depends on the updates of another parameter vector, refer to [39] and [40] for detail.

2) Mountain Car: The learning curves are shown in Fig. 6. We can find: 1) with the increase of the parameter λ, see Fig. 6. (a)-(h), OSKTD, OK-TD, and OK-TD using ALD converge faster; 2) with the increase of the parameter μ_2, OSKTD, OST-GTD2, and OSK-TDC get more stable; 3) OSKTD, OST-GTD2, and OSK-TDC get a better local optima than the other algorithms due to the local validity, but converges slower than the other algorithms because they only update part of the parameter vector using a selective function in a step, while the others update the whole; 4) OSKTD converges faster than OST-GTD2 and OSK-TDC; and 5) OSK-GTD is unstable. The reason is unclear, maybe because that GTD is not a true gradient compared with GTD2 and TDC.

From the time curves shown in Fig. 4(b), we can see[3] that the time consumed in each episode of the algorithms can be ranked in a descending order as follows: 1) OK-methods (including OK-TD, OK-GTD, OK-GTD2, and OK-TDC) using ALD; 2) OSK-methods (including OSKTD, OSK-GTD, OSK-GTD2, and OSK-TDC); and 3) OK-methods. Note that: 1) the complexity of sparsification is $O(n^2)$ using ALD, and $O(n)$ using a kernel distance based method. That is why the time consumed using ALD is far longer than that in OSK-methods and OK-methods and 2) compared with OK-methods, there is a selection procedure in OSK-methods. That is why the time consumed in OSK-methods is a bit longer than that in OK-methods. In addition, note that the ALD method is used usually as an online method [20], [43], [44] (in [15], ALD is used as an offline method) for sparsification. In the experiments, kernel-based algorithms using ALD is implemented as an online method. The motivation of comparison in the terms of time consumed for different algorithms is to show intuitively that OSKTD enables the online procedure for kernel sparsification.

Actually in this section, OSKTD (i.e., OSK-SARSA) is a learning method for control, which optimizes the state-action value function instead of the state value function via optimizing the proposed selective kernel-based value function. From the experimental results, we can see that: 1) compared with the tabular methods (including Q and TD), OSKTD reaches the optimal value function, and converges much faster because OSKTD does not visit all states or state-action pairs; 2) compared with the kernel-based methods (including OK-TD, OK-GTD, OK-GTD2, and OK-TDC), OSKTD reaches a more optimal value function because of the selective function in the selective kernel-based value function; and 3) compared with the selective kernel-based methods (including OSK-GTD, OSK-GTD2, and OSK-TDC), OSKTD reaches a competitive value function, but converges much faster because of the fast convergence of TD learning.

VII. Conclusion

In this paper, the OSKTD learning algorithm was proposed to deal with large scale and/or continuous RL problems. OSKTD had the following two procedures: online kernel sparsification and parameters updating for the selective kernel-based value function.

In the procedure of online kernel sparsification, the kernel sparsification approach was proposed based on selective ensemble learning. Based on the proposed approach, the kernel distance-based online sparsification method, which was less complex computationally compared with other methods, was proposed to construct the sample dictionary, the complexity of which is $O(n)$. Further, the kernel distance-based online sparsification method can be seen as a modified version of NC by omitting the factor of the prediction error.

In the procedure of parameters updating, firstly, based on local validity, a selective kernel-based value function was proposed. Then, the classic framework of TD(λ) with the gradient descent technique was applied to parameters updating for the selective kernel-based value function, then we can get the OSKTD learning algorithm.

Note that two properties of the selective kernel-based value function were analyzed: 1) the traditional table-based value function can be seen as a special case of our selective kernel-based value function with some special settings and 2) the selective kernel-based value function can be viewed as an approach of the kernel function redefinition.

The algorithm compared with both traditional and up-to-date algorithms was tested in two experiments: Maze and Mountain Car, corresponding to a discrete RL problem and a continuous RL problem, respectively. In the Maze problem, OSKTD converges to the optima policy and converges faster than both traditional and up-to-date algorithms. In addition, in the Mountain Car problem, OSKTD converges, requires less computation, gets a better local optima than the traditional algorithms and converges faster than the up-to-date algorithms with an competitive local optima. Because the complexity of

[2]One possible explanation is that Maze is so detail sensitive that one global approximated value function (without a selective function) cannot describe all features of the maze.

[3]Only one figure is shown because there are no remarkable changes with different parameters.

sparsification in the kernel-based TD(λ) with ALD was $O(n^2)$ and the complexity of the OSKTD algorithm with a selective function was $O(n)$, this means that the complexity of our OSKTD can satisfy the requirement of real-time applications.

Future work includes: 1) OSKTD mainly focus on RL problems with large-scale and/or continuous state space. To deal with RL problems with large-scale and/or continuous action space, the OSKTD algorithm can be combined with fast action selection techniques, such as [49]–[51]. 2) In this paper, the distance between two states in the selective function is based on the kernel distance, other metrics between two states in a MDP can be further studied [52], [53]. In addition, different selective functions should be further studied. 3) The convergence of OSKTD will be analyzed theoretically, and the error bounds due to the local validity should also be analyzed theoretically.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees and the editor for their helpful comments and suggestions.

REFERENCES

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

[2] D. Bertsekas, J. Tsitsiklis, and A. Scientific, *Neuro-Dynamic Programming*. Athens, Greece: Athena Scientific, 1996.

[3] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 4, pp. 875–889, Jul. 2001.

[4] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[5] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, 2003.

[6] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 1, pp. 1633–1685, 2009.

[7] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, Jul. 2008, pp. 752–759.

[8] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, no. 1, pp. 503–556, 2005.

[9] Z. Chen and S. Jagannathan, "Generalized Hamilton–Jacobi–Bellman formulation-based neural network control of affine nonlinear discrete-time systems," *IEEE Trans. Neural Netw.*, vol. 19, no. 1, pp. 90–106, Jan. 2008.

[10] B. Li and J. Si, "Approximate robust policy iteration using multilayer perceptron neural networks for discounted infinite-horizon Markov decision processes with uncertain correlated transition matrices," *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1270–1280, Aug. 2010.

[11] F. 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 Trans. Neural Netw.*, vol. 22, no. 1, pp. 1–13, Jan. 2011.

[12] S. van den Dries and M. A. Wiering, "Neural-fitted TD-leaf learning for playing Othello with structured neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 11, pp. 1701–1713, Nov. 2012.

[13] A. Heydari and S. N. Balakrishnan, "Finite-horizon control-constrained nonlinear optimal control using single network adaptive critics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 1, pp. 145–157, Jan. 2013.

[14] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Mach. Learn.*, vol. 49, nos. 2–3, pp. 161–178, 2002.

[15] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 973–992, Jul. 2007.

[16] F. Orabona, L. Jie, and B. Caputo, "Multi kernel learning with online-batch optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 227–253, Feb. 2012.

[17] D. Nguyen-Tuong and J. Peters, "Online kernel-based learning for task-space tracking robot control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1417–1425, Sep. 2012.

[18] B. Schölkopf, R. Herbrich, and A. Smola, "A generalized representer theorem," in *Proc. 14th Comput. Learn. Theory*, Amsterdam, The Netherlands, Jul. 2001, pp. 416–426.

[19] D. D. Castro and S. Mannor, "Adaptive bases for reinforcement learning," in *Proc. 21st Eur. Conf. Mach. Learn.*, Barcelona, Spain, Sep. 2010, pp. 312–327.

[20] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. 20th Int. Conf. Mach. Learn.*, Washington, DC, USA, Aug. 2003, pp. 154–161.

[21] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proc. 22nd Int. Conf. Mach. Learn.*, New York, NY, USA, Aug. 2005, pp. 201–208.

[22] D. Schneegaß, S. Udluft, and T. Martinetz, "Kernel rewards regression: An information efficient batch policy iteration approach," in *Proc. 24th IASTED Int. Conf. Artif. Intell. Appl.*, Innsbruck, Austria, Feb. 2006, pp. 428–433.

[23] N. Jong and P. Stone, "Kernel-based models for reinforcement learning," in *Proc. ICML Workshop Kernel Mach. Reinforcement Learn.*, Pittsburgh, PA, USA, Jun. 2006, pp. 1–53.

[24] X. Xu, "A sparse kernel-based least-squares temporal difference algorithm for reinforcement learning," in *Proc. 2nd Int. Conf. Adv. Natural Comput.*, vol. 4221. Xi'an, China, Sep. 2006, pp. 47–56.

[25] M. Deisenroth, J. Peters, and C. Rasmussen, "Approximate dynamic programming with Gaussian processes," in *Proc. 23rd AAAI Conf. Artif. Intell.*, Seattle, WA, USA, Jun. 2008, pp. 4480–4485.

[26] J. Reisinger, P. Stone, and R. Miikkulainen, "Online kernel selection for Bayesian reinforcement learning," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, Jul. 2008, pp. 816–823.

[27] A. Antos, C. Szepesvári, and R. Munos, "Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path," *Mach. Learn.*, vol. 71, no. 1, pp. 89–129, 2008.

[28] B. Bethke and J. How, "Approximate dynamic programming using Bellman residual elimination and Gaussian process regression," in *Proc. Amer. Control Conf.*, Piscataway, NJ, USA, Jul. 2009, pp. 745–750.

[29] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Proc. 26th Int. Conf. Mach. Learn.*, Montreal, QC, Canada, Jun. 2009, pp. 1017–1024.

[30] O. B. Kroemer and J. Peters, "A non-parametric approach to dynamic programming," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 23. Granada, Spain, Dec. 2011, pp. 1719–1727.

[31] M. Robards, P. Sunehag, S. Sanner, and B. Marthi, "Sparse kernel-SARSA (λ) with an eligibility trace," in *Proc. 22nd Eur. Conf. Mach. Learn.*, Athens, Greece, Sep. 2011, pp. 1–17.

[32] F. Orabona, J. Keshet, and B. Caputo, "The projectron: A bounded kernel-based perceptron," in *Proc. 25th Int. Conf. Mach. Learn.*, Helsinki, Finland, Jul. 2008, pp. 816–823.

[33] F. Orabona, J. Keshet, and B. Caputo, "Bounded kernel-based online learning," *J. Mach. Learn. Res.*, vol. 10, pp. 2643–2666, Jan. 2009.

[34] W. Liu, I. Park, and J. Príncipe, "An information theoretic approach of designing sparse kernel adaptive filters," *IEEE Trans. Neural Netw.*, vol. 20, no. 12, pp. 1950–1961, Dec. 2009.

[35] S. Bradtke, "Incremental dynamic programming for on-line adaptive optimal control," Ph.D. dissertation, Dept. Comput. Sci., Univ. Massachusetts, Amherst, MA, USA, Sep. 1994.

[36] S. Bradtke and A. Barto, "Linear least-squares algorithms for temporal difference learning," *Mach. Learn.*, vol. 22, no. 1, pp. 33–57, 1996.

[37] J. Boyan, "Technical update: Least-squares temporal difference learning," *Mach. Learn.*, vol. 49, no. 2, pp. 233–246, 2002.

[38] A. Geramifard, M. Bowling, and R. Sutton, "Incremental least-squares temporal difference learning," in *Proc. 21st AAAI Conf. Artif. Intell.*, Boston, Massachusetts, Jul. 2006, pp. 356–361.

[39] R. Sutton, C. Szepesvári, and H. Maei, "A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 21. Vancouver, BC, Canada, Dec. 2008, pp. 1609–1616.

[40] R. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. 26th Int. Conf. Mach. Learn.*, Montreal, PQ, Canada, Jul. 2009, pp. 993–1000.

[41] B. Schölkopf, A. Smola, and K. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, 1998.

[42] T. Chin and D. Suter, "Incremental kernel principal component analysis," *IEEE Trans. Image Process.*, vol. 16, no. 6, pp. 1662–1674, Jun. 2007.

[43] W. Liu and J. Principe, "Kernel affine projection algorithms," *EURASIP J. Adv. Signal Process.*, vol. 2008, no. 1, pp. 1–13, 2008.

[44] W. Liu, I. Park, Y. Wang, and J. Príncipe, "Extended kernel recursive least squares algorithm," *IEEE Trans. Signal Process.*, vol. 57, no. 10, pp. 3801–3814, Oct. 2009.

[45] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.

[46] H. Lappalainen and J. Miskin, "Ensemble learning," in *Advances in Independent Component Analysis*, M. Girolami, Ed. Berlin, Germany: Springer-Verlag, 2000.

[47] Z. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: Many could be better than all," *Artif. Intell.*, vol. 137, nos. 1–2, pp. 239–263, 2002.

[48] X. Tan, S. Chen, Z. Zhou, and F. Zhang, "Recognizing partially occluded, expression variant faces from single training image per person with SOM and soft k-NN ensemble," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 875–886, Apr. 2005.

[49] H. van Hasselt and M. Wiering, "Reinforcement learning in continuous action spaces," in *Proc. IEEE Int. Symp. Approximate Dyn. Program. Reinforcement Learn.*, Apr. 2007, pp. 272–279.

[50] A. Bonarini, "Reinforcement learning in continuous action spaces through sequential Monte Carlo methods," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 20. Vancouver, BC, Canada, Dec. 2007, pp. 833–840.

[51] X. Xu, C. Liu, and D. Hu, "Continuous-action reinforcement learning with fast policy search and adaptive basis function selection," *Soft Comput., A Fusion of Found., Methodol. Appl.*, vol. 15, no. 6, pp. 1055–1070, 2011.

[52] N. Ferns, P. Castro, D. Precup, and P. Panangaden, "Methods for computing state similarity in Markov decision processes," in *Proc. 22nd Conf. Uncertainty Artif. Intell.*, Cambridge, MA, USA, Jul. 2006, pp. 174–181.

[53] N. Ferns, P. Panangaden, and D. Precup, "Bisimulation metrics for continuous markov decision processes," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1662–1714, 2011.

Xingguo Chen received the B.S. degree from Nanjing University, Nanjing, China, in 2007, where he is currently pursuing the Ph.D. degree in computer application technology.

His current research interests include game AI, transfer learning, and reinforcement learning.

Yang Gao (M'05) received the Ph.D. degree in computer science and technology from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2000.

He is a Professor with the Department of Computer Science and Technology, Nanjing University. He has published more than 50 papers in top conferences and journals. His current research interests include artificial intelligence and machine learning.

Ruili Wang received the Ph.D. degree in computer science from Dublin City University, Dublin, Ireland, in 2003.

He is a Senior Lecturer of computer science and information technology with the School of Engineering and Advanced Technology, Massey University, Auckland, New Zealand. He has published more than 80 papers in top conferences and journals. His current research interests include artificial intelligence (e.g., machine learning and speech processing) and complex systems.