Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2016, Article ID 2305854, 11 pages
http://dx.doi.org/10.1155/2016/2305854

Research Article

Kernel Recursive Least-Squares Temporal Difference Algorithms with Sparsification and Regularization

Chunyuan Zhang,[1,2] Qingxin Zhu,[1] and Xinzheng Niu[1]

[1] *School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*
[2] *College of Information Science and Technology, Hainan University, Haikou 570228, China*

Correspondence should be addressed to Chunyuan Zhang; zcy7566@126.com

Received 18 January 2016; Revised 23 April 2016; Accepted 28 April 2016

Academic Editor: Manuel Graña

By combining with sparse kernel methods, least-squares temporal difference (LSTD) algorithms can construct the feature dictionary automatically and obtain a better generalization ability. However, the previous kernel-based LSTD algorithms do not consider regularization and their sparsification processes are batch or offline, which hinder their widespread applications in online learning problems. In this paper, we combine the following five techniques and propose two novel kernel recursive LSTD algorithms: (i) online sparsification, which can cope with unknown state regions and be used for online learning, (ii) L_2 and L_1 regularization, which can avoid overfitting and eliminate the influence of noise, (iii) recursive least squares, which can eliminate matrix-inversion operations and reduce computational complexity, (iv) a sliding-window approach, which can avoid caching all history samples and reduce the computational cost, and (v) the fixed-point subiteration and online pruning, which can make L_1 regularization easy to implement. Finally, simulation results on two 50-state chain problems demonstrate the effectiveness of our algorithms.

1. Introduction

The least-squares temporal difference (LSTD) learning may be the most popular approach for policy evaluation in reinforcement learning (RL) [1, 2]. Compared with the standard temporal difference (TD) learning, LSTD uses samples more efficiently and eliminates all step-size parameters. However, LSTD also has some drawbacks. First, LSTD requires a matrix-inversion operation at each time step. To reduce computational complexity, Bradtke and Barto proposed a recursive LSTD (RLSTD) algorithm [1], and Xu et al. proposed a RLSTD(λ) algorithm [3]. But these two algorithms still require many features especially for highly nonlinear RL problems, since the RLS approximator assumes a linear model [4]. Second, when the number of features is larger than the number of training samples, LSTD is prone to overfitting. To overcome this problem, Kolter and Ng proposed an L_1-regularized LSTD algorithm called LARS-TD for feature selection [5], but it is only applicable for batch learning and its implementation is complicated. On this basis, Chen et al. proposed an L_2-regularized RLSTD algorithm [6]. In contrast with LARS-TD, it has an analytical solution, but it cannot obtain a sparse solution. Third, LSTD requires users to design the feature vector manually, and poor design choices can result in estimates that diverge from the optimal value function [7].

In the last two decades, kernel methods have been intensively and extensively studied in supervised and unsupervised learning [8]. The basic idea behind kernel methods can be summarized as follows: By using a nonlinear transform, the origin input data can be mapped into a high-dimensional feature space, and an inner product in this space can be interpreted as a Mercer kernel function. Thus, as long as a linear algorithm can be formulated in terms of inner products, there is no need to perform computations in the high-dimensional feature space [9]. Recently, there have also been many research works on kernelizing least-squares algorithms [9–13]. Here, we only review some works related to our proposed algorithms. One typical work is the sparse kernel recursive least-squares (SKRLS) algorithm with the approximate linear dependency (ALD) criterion [11]. Compared with traditional RLS algorithms, it not only has a good nonlinear approximation ability but also can construct the feature dictionary automatically. Similarly, Chen et al. proposed an L_2-regularized SKRLS algorithm with the online vector quantization [12]. Besides having the good properties of SKRLS-ALD, it can avoid overfitting. In addition, Chen et al. proposed an L_1-regularized SKRLS algorithm with the fixed-point subiteration [13], which can yield a much sparser dictionary.

Intuitively, we can also bring the benefits of kernel machine learning to LSTD algorithms. In fact, kernel-based RL algorithms have become more and more popular in recent years [14–22], and several works have been done for kernelizing LSTD algorithms. In an earlier paper, Xu proposed a sparse kernel-based LSTD(λ) (SKLSTD(λ)) algorithm with the ALD criterion [19]. Although this algorithm can avoid selecting features manually, it is only applicable for batch learning and its derivation is complicated. After that, Xu et al. proposed an incremental version of the SKLSTD(λ) algorithm for policy iteration [20], but this algorithm still requires a matrix-inversion operation at each time step. Moreover, the feature dictionary is required to be constructed offline, which makes this algorithm only approximate the value function correctly in the area of the state space that is covered by the training samples. Recently, Jakab and Csató proposed a sparse kernel RLSTD (SKRLSTD) algorithm by using a proximity graph sparsification method [21]. Unfortunately, its sparsification process is also offline. In addition, all of these algorithms do not consider regularization, whereas many real problems exhibit noise and the high expressiveness of the kernel matrix can result in overfitting [22].

In this paper, we propose two online SKRLSTD algorithms with L_2 and L_1 regularization, called OSKRLSTD-L_2 and OSKRLSTD-L_1, respectively. Compared with the derivation of SKLSTD(λ), our derivation uses Bellman operator along with projection operator and thus is more simple. To cope with unknown state-space regions and avoid overfitting, our algorithms use online sparsification and regularization techniques. Besides, to reduce computational complexity and avoid caching all history samples, our algorithms also use the recursive least-squares and the sliding-window technique. Moreover, different from LARS-TD, OSKRLSTD-L_1 uses the subiteration and online pruning to find the fixed point. These techniques make our algorithms more suitable for online RL problems with a large or continuous state space. The rest of this paper is organized as follows. In Section 2, we present preliminaries and review the LSTD algorithm. Section 3 contains the main contribution of this paper: we derive OSKRLSTD-L_2 and OSKRLSTD-L_1 algorithms in detail. In Section 4, we demonstrate the effectiveness of our algorithms for two 50-state chain problems. Finally, we conclude the paper in Section 5.

2. Background

In this section, we introduce the basic definitions and notations, which will be used throughout the paper without any further mention. We also review the LSTD algorithm, which is needed to establish our algorithms described in Section 3.

2.1. Preliminaries. In RL and dynamic programming (DP), an underlying sequential decision-making problem is often modeled as a Markov decision process (MDP). An MDP can be defined as a tuple $\mathscr{M} = \langle \mathscr{S}, \mathscr{A}, P, r, \gamma, d \rangle$ [5], where $\mathscr{S}$ is a set of states, $\mathscr{A}$ is a set of actions, $P : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow [0,1]$ is a state transition probability function where $P(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ denotes the probability of transitioning to state $\mathbf{s}'$ when taking action $\mathbf{a}$ in state $\mathbf{s}$, $r \in \mathbb{R}$ is a reward function, $\gamma \in [0,1]$ is the discount factor, and d is an initial state distribution. For simplicity of presentation, we assume that $\mathscr{S}$ and $\mathscr{A}$ are finite. Given an MDP $\mathscr{M}$ and a policy $\pi : \mathscr{S} \rightarrow \mathscr{A}$, the sequence $\mathbf{s}_1, r_1, \mathbf{s}_2, r_2, \ldots$ is a Markov reward process $\mathscr{R} = \langle \mathscr{S}, P^\pi, R^\pi, \gamma, d \rangle$, where $P^\pi(\mathbf{s}, \mathbf{s}') = \sum_{\mathbf{a} \in \mathscr{A}} \pi(\mathbf{a} \mid \mathbf{s}) P(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ and $R^\pi(\mathbf{s}) = \sum_{\mathbf{a} \in \mathscr{A}} \pi(\mathbf{a} \mid \mathbf{s}) \sum_{\mathbf{s}' \in \mathscr{S}} P(\mathbf{s}, \mathbf{a}, \mathbf{s}') r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$.

RL and DP often use the state-value function $V^\pi(\mathbf{s})$ to evaluate how good the policy π is for the agent to be in state $\mathbf{s}$. For an MDP, $V^\pi(\mathbf{s})$ can be defined as $V^\pi(\mathbf{s}) = \mathbf{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t \mid \mathbf{s}_0 = \mathbf{s}]$, which must obey the Bellman equation [23],

$$V^\pi(\mathbf{s}) = R^\pi(\mathbf{s}) + \gamma \sum_{\mathbf{s}' \in \mathscr{S}} P^\pi(\mathbf{s}, \mathbf{s}') V^\pi(\mathbf{s}'), \tag{1}$$

or be expressed in vector form,

$$V^\pi = R^\pi + \gamma P^\pi V^\pi. \tag{2}$$

If P^π and R^π are known, V^π can be solved analytically; that is,

$$V^\pi = (\mathbf{I} - \gamma P^\pi)^{-1} R^\pi, \tag{3}$$

where $\mathbf{I}$ is the $|\mathscr{S}| \times |\mathscr{S}|$ identity matrix.

However, different from the case in DP, P^π and R^π are unknown in RL. The agent has to estimate V^π by exploring the environment. Furthermore, many real problems have a large or continuous state space, which makes $V^\pi(\mathbf{s})$ hard to be expressed explicitly. To overcome this problem, we often resort to linear function approximation; that is,

$$\begin{aligned} \widehat{V}^\pi(\mathbf{s}) &= \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{s}) \\ \text{or } \widehat{V}^\pi &= \Phi \mathbf{w}, \end{aligned} \tag{4}$$

where $\mathbf{w} \in \mathbb{R}^m$ is a parameter vector, $\boldsymbol{\phi}(\mathbf{s}) \in \mathbb{R}^m$ is the feature vector of state $\mathbf{s}$, and $\Phi = [\boldsymbol{\phi}(\mathbf{s}_1), \ldots, \boldsymbol{\phi}(\mathbf{s}_{|\mathscr{S}|})]^\top$ is a $|\mathscr{S}| \times m$ feature matrix. Unfortunately, when approximating V^π in this manner, there is usually no way to satisfy the Bellman equation exactly, because $R^\pi + \gamma P^\pi \Phi \mathbf{w}$ may lie outside the span of Φ [5].

2.2. LSTD Algorithm. The LSTD algorithm presents an efficient way to find $\mathbf{w}$ such that $\widehat{V}^\pi$ "approximately" satisfies the Bellman equation [5]. By solving the least-squares problem $\min_{\mathbf{u} \in \mathbb{R}^m} \|\Phi \mathbf{u} - (R^\pi + \gamma P^\pi \Phi \mathbf{w})\|_{\mathbf{D}}^2$, we can find a closest approximation $\Phi \mathbf{u}^*$ in the span of Φ to replace $R^\pi + \gamma P^\pi \Phi \mathbf{w}$. Then, from (2) and (4), we can use $\mathbf{w} = \mathbf{u}^*$ for approximating V^π. That means we can find $\mathbf{w}$ by solving the fixed-point equation:

$$\mathbf{w} = f(\mathbf{w}) = \arg\min_{\mathbf{u} \in \mathbb{R}^m} \|\Phi \mathbf{u} - (R^\pi + \gamma P^\pi \Phi \mathbf{w})\|_{\mathbf{D}}^2, \tag{5}$$

where $\mathbf{D}$ is a nonnegative diagonal matrix indicating a distribution over states. Nevertheless, since P^{π} and R^{π} are unknown and since Φ is too large to form anyway in a large or continuous state space, we cannot solve (5) exactly. Instead, given a trajectory $T_t^{\pi} = \{(\mathbf{s}_i, \mathbf{s}'_i, r_i) \mid i = 1, \ldots, t\}$ following policy π, LSTD uses $\widehat{\Phi}_t = [\boldsymbol{\phi}(\mathbf{s}_1), \ldots, \boldsymbol{\phi}(\mathbf{s}_t)]^{\top}$, $\widehat{\Phi}'_t = [\boldsymbol{\phi}(\mathbf{s}'_1), \ldots, \boldsymbol{\phi}(\mathbf{s}'_t)]^{\top}$, and $\widehat{R}_t = [r_1, \ldots, r_t]^{\top}$ to replace Φ, $P^{\pi}\Phi$, and R^{π}, respectively. Then, (5) can be approximately rewritten as

$$\mathbf{w}_t = \tilde{f}(\mathbf{w}_t) = \arg\min_{\mathbf{u}\in\mathbb{R}^m} \left\| \widehat{\Phi}_t \mathbf{u} - \left(\widehat{R}_t + \gamma \widehat{\Phi}'_t \mathbf{w}_t\right)\right\|_2^2. \tag{6}$$

Let $\partial\|\widehat{\Phi}_t \mathbf{u} - (\widehat{R}_t + \gamma\widehat{\Phi}'_t\mathbf{w}_t)\|_2^2/\partial\mathbf{u} = \mathbf{0}$; we have

$$\tilde{f}(\mathbf{w}_t) = \mathbf{u}^* = \left(\widehat{\Phi}_t^{\top}\widehat{\Phi}_t\right)^{-1}\widehat{\Phi}_t^{\top}\left(\widehat{R}_t + \gamma\widehat{\Phi}'_t\mathbf{w}_t\right). \tag{7}$$

Thus, the fixed point $\mathbf{w}_t = \tilde{f}(\mathbf{w}_t)$ can be found by

$$\mathbf{w}_t = \left(\widehat{\Phi}_t^{\top}\left(\widehat{\Phi}_t - \gamma\widehat{\Phi}'_t\right)\right)^{-1}\widehat{\Phi}_t^{\top}\widehat{R}_t. \tag{8}$$

3. Regularized OSKRLSTD Algorithms

To overcome the weaknesses of the previous kernel-based LSTD algorithms, we propose two regularized OSKRLSTD algorithms in this section.

3.1. OSKRLSTD-L_2 Algorithm.

Now, we use L_2 regularization and online sparsification to derive the first OSKRLSTD algorithm, which is called OSKRLSTD-L_2.

First, we use the kernel trick to kernelize (6). Suppose the feature dictionary $\mathcal{D}_t = \{\mathbf{d}_j \mid \mathbf{d}_j \in \mathcal{S},\ j = 1, \ldots, n_t\}$, and let $\Phi_t = [\boldsymbol{\phi}(\mathbf{d}_1), \ldots, \boldsymbol{\phi}(\mathbf{d}_{n_t})]^{\top}$ denote the corresponding feature matrix. By the Representer Theorem [24], $\mathbf{w}_t$ and $\mathbf{u}$ can be expressed as follows:

$$\begin{aligned}\mathbf{w}_t &= \Phi_t^{\top}\boldsymbol{\alpha}_t = \sum_{j=1}^{n_t}\alpha_j\boldsymbol{\phi}(\mathbf{d}_j),\\ \mathbf{u} &= \Phi_t^{\top}\boldsymbol{\beta} = \sum_{j=1}^{n_t}\beta_j\boldsymbol{\phi}(\mathbf{d}_j),\end{aligned} \tag{9}$$

where $\boldsymbol{\alpha}_t = [\alpha_{t,1}, \ldots, \alpha_{t,n_t}]^{\top}$ and $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_{n_t}]^{\top}$ are the coefficient vector of $\mathbf{w}_t$ and $\mathbf{u}$, respectively. Then, from (6), we have

$$\boldsymbol{\alpha}_t = \hat{f}(\boldsymbol{\alpha}_t) = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{n_t}}\left\|\widehat{\Phi}_t\Phi_t^{\top}\boldsymbol{\beta} - \left(\widehat{R}_t + \gamma\widehat{\Phi}'_t\Phi_t^{\top}\boldsymbol{\alpha}_t\right)\right\|_2^2. \tag{10}$$

By the Mercer Theorem [24], the inner product of two feature vectors can be calculated by $k(\mathbf{s}_i, \mathbf{s}_j) = \boldsymbol{\phi}(\mathbf{s}_i)^{\top}\boldsymbol{\phi}(\mathbf{s}_j)$. Thus, we can define $\mathbf{K}_t = \Phi_t\Phi_t^{\top}$, $\widehat{\mathbf{K}}_t = \Phi_t\widehat{\Phi}_t^{\top}$, and $\widehat{\mathbf{K}}'_t = \Phi_t(\widehat{\Phi}'_t)^{\top}$. On this basis, (10) can be rewritten as

$$\boldsymbol{\alpha}_t = \hat{f}(\boldsymbol{\alpha}_t) = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{n_t}}\left\|\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - \left(\widehat{R}_t + \gamma\left(\widehat{\mathbf{K}}'_t\right)^{\top}\boldsymbol{\alpha}_t\right)\right\|_2^2. \tag{11}$$

Second, we try to derive the L_2-regularized solution of (11). Add an L_2-norm penalty into (11); that is,

$$\begin{aligned}\boldsymbol{\alpha}_t &= \hat{f}(\boldsymbol{\alpha}_t)\\ &= \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^{n_t}}\left\|\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - \left(\widehat{R}_t + \gamma\left(\widehat{\mathbf{K}}'_t\right)^{\top}\boldsymbol{\alpha}_t\right)\right\|_2^2 + \eta\|\boldsymbol{\beta}\|_2^2,\end{aligned} \tag{12}$$

where $\eta \in [0, \infty)$ is a regularization parameter. Let $\partial(\|\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - (\widehat{R}_t + \gamma(\widehat{\mathbf{K}}'_t)^{\top}\boldsymbol{\alpha}_t)\|_2^2 + \eta\|\boldsymbol{\beta}\|_2^2)/\partial\boldsymbol{\beta} = \mathbf{0}$; we have

$$\widehat{\mathbf{K}}_t\left(\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta}^* - \gamma\left(\widehat{\mathbf{K}}'_t\right)^{\top}\boldsymbol{\alpha}_t\right) + \eta\boldsymbol{\beta}^* = \widehat{\mathbf{K}}_t\widehat{R}_t. \tag{13}$$

Since $\mathbf{w}_t = \mathbf{u}^*$, we easily have $\boldsymbol{\alpha}_t = \boldsymbol{\beta}^*$ from (9). Then, the above equation can be rewritten as

$$\left(\widehat{\mathbf{K}}_t\left(\widehat{\mathbf{K}}_t - \gamma\widehat{\mathbf{K}}'_t\right)^{\top} + \eta\mathbf{I}_t\right)\boldsymbol{\alpha}_t = \widehat{\mathbf{K}}_t\widehat{R}_t, \tag{14}$$

where $\mathbf{I}_t$ is the $n_t \times n_t$ identity matrix. Thus, $\boldsymbol{\alpha}_t$ can be analytically solved as

$$\boldsymbol{\alpha}_t = \left(\widehat{\mathbf{K}}_t\left(\widehat{\mathbf{K}}_t - \gamma\widehat{\mathbf{K}}'_t\right)^{\top} + \eta\mathbf{I}_t\right)^{-1}\widehat{\mathbf{K}}_t\widehat{R}_t = \mathbf{A}_t^{-1}\mathbf{b}_t, \tag{15}$$

where $\mathbf{A}_t \in \mathbb{R}^{n_t\times n_t}$ and $\mathbf{b}_t \in \mathbb{R}^{n_t}$ denote

$$\begin{aligned}\mathbf{A}_t &= \widehat{\mathbf{K}}_t\left(\widehat{\mathbf{K}}_t - \gamma\widehat{\mathbf{K}}'_t\right)^{\top} + \eta\mathbf{I}_t\\ &= \sum_{i=1}^{t}\mathbf{k}_t(\mathbf{s}_i)\,\Delta\mathbf{k}_t^{\top}(\mathbf{s}_i, \mathbf{s}'_i) + \eta\mathbf{I}_t,\\ \mathbf{b}_t &= \widehat{\mathbf{K}}_t\widehat{R}_t = \sum_{i=1}^{t}\mathbf{k}_t(\mathbf{s}_i)\,r_i,\end{aligned} \tag{16}$$

where $\mathbf{k}_t(\cdot) = [k(\cdot, \mathbf{d}_1), \ldots, k(\cdot, \mathbf{d}_{n_t})]^{\top}$ and $\Delta\mathbf{k}_t(\mathbf{s}_i, \mathbf{s}'_i) = \mathbf{k}_t(\mathbf{s}_i) - \gamma\mathbf{k}_t(\mathbf{s}'_i)$.

Third, we derive the recursive formulas of $\mathbf{A}_t^{-1}$ and $\boldsymbol{\alpha}_t$. Under online sparsification, there are two cases: (1) $\mathcal{D}_t = \mathcal{D}_{t-1}$, $n_t = n_{t-1}$, $\mathbf{k}_t(\cdot) = \mathbf{k}_{t-1}(\cdot)$, $\Delta\mathbf{k}_t(\mathbf{s}_i, \mathbf{s}'_i) = \Delta\mathbf{k}_{t-1}(\mathbf{s}_i, \mathbf{s}'_i)$, and $\mathbf{I}_t = \mathbf{I}_{t-1}$; (2) $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{s}_t\}$, $n_t = n_{t-1} + 1$, $\mathbf{k}_t(\cdot) = [\mathbf{k}_{t-1}^{\top}(\cdot), k(\cdot, \mathbf{s}_t)]^{\top}$, $\Delta\mathbf{k}_t(\mathbf{s}_i, \mathbf{s}'_i) = [\Delta\mathbf{k}_{t-1}^{\top}(\mathbf{s}_i, \mathbf{s}'_i), \Delta k_t(\mathbf{s}_i, \mathbf{s}'_i)]^{\top}$, where $\Delta k_t(\mathbf{s}_i, \mathbf{s}'_i) = k(\mathbf{s}_i, \mathbf{s}_t) - \gamma k(\mathbf{s}'_i, \mathbf{s}_t)$, and $\mathbf{I}_t$ is expanded as

$$\mathbf{I}_t = \begin{bmatrix}\mathbf{I}_{t-1} & \mathbf{0}_{t-1}\\ \mathbf{0}_{t-1}^{\top} & 1\end{bmatrix}, \tag{17}$$

where $\mathbf{0}_{t-1}$ is the n_{t-1} dimensional zero vector.

For the first case, (16) can be rewritten as follows:

$$\mathbf{A}_t = \mathbf{A}_{t-1} + \mathbf{k}_{t-1}(\mathbf{s}_t)\,\Delta\mathbf{k}_{t-1}^{\top}(\mathbf{s}_t, \mathbf{s}'_t), \tag{18}$$

$$\mathbf{b}_t = \mathbf{b}_{t-1} + \mathbf{k}_{t-1}(\mathbf{s}_t)\,r_t. \tag{19}$$

Applying the matrix-inversion lemma [25] for $\mathbf{A}_t^{-1}$, we get

$$\mathbf{A}_t^{-1} = \mathbf{A}_{t-1}^{-1} - \frac{\mathbf{A}_{t-1}^{-1}\mathbf{k}_{t-1}(\mathbf{s}_t)\,\Delta\mathbf{k}_{t-1}^{\top}(\mathbf{s}_t, \mathbf{s}'_t)\,\mathbf{A}_{t-1}^{-1}}{1 + \Delta\mathbf{k}_{t-1}^{\top}(\mathbf{s}_t, \mathbf{s}'_t)\,\mathbf{A}_{t-1}^{-1}\mathbf{k}_{t-1}(\mathbf{s}_t)}. \tag{20}$$

Thus, plugging (19) and (20) into (15), we obtain

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} + \frac{\left(r_t - \boldsymbol{\alpha}_{t-1}^{\top}\Delta\mathbf{k}_{t-1}\left(\mathbf{s}_t,\mathbf{s}_t'\right)\right)\mathbf{A}_{t-1}^{-1}\mathbf{k}_{t-1}\left(\mathbf{s}_t\right)}{1 + \Delta\mathbf{k}_{t-1}^{\top}\left(\mathbf{s}_t,\mathbf{s}_t'\right)\mathbf{A}_{t-1}^{-1}\mathbf{k}_{t-1}\left(\mathbf{s}_t\right)}. \tag{21}$$

For the second case, (16) can be rewritten as follows:

$$\begin{gathered}\mathbf{A}_t = \begin{bmatrix}\widetilde{\mathbf{A}}_t & \mathbf{h}_t \\ \mathbf{g}_t^{\top} & p_t\end{bmatrix}, \\ \mathbf{b}_t = \begin{bmatrix}\widetilde{\mathbf{b}}_t \\ q_t\end{bmatrix},\end{gathered} \tag{22}$$

where $\widetilde{\mathbf{A}}_t$ and $\widetilde{\mathbf{b}}_t$ are the same as the updated $\mathbf{A}_t$ and $\mathbf{b}_t$ when the feature dictionary keeps unchanged, $\mathbf{h}_t = \sum_{i=1}^{t}\Delta k_t(\mathbf{s}_i,\mathbf{s}_i')\mathbf{k}_{t-1}(\mathbf{s}_i)$, $\mathbf{g}_t = \sum_{i=1}^{t}k(\mathbf{s}_i,\mathbf{s}_t)\Delta\mathbf{k}_{t-1}(\mathbf{s}_i,\mathbf{s}_i')$, $p_t = \sum_{i=1}^{t}k(\mathbf{s}_i,\mathbf{s}_t)\Delta k_t(\mathbf{s}_i,\mathbf{s}_i') + \eta$, and $q_t = \sum_{i=1}^{t}k(\mathbf{s}_i,\mathbf{s}_t)r_i$. However, computing $\mathbf{h}_t$, $\mathbf{g}_t$, p_t, and q_t requires caching all history samples, and the computational cost will become more and more expensive as t increases. Inspired by the work of Van Vaerenbergh et al. [26], we introduce a sliding window $\mathscr{H}_t$ to deal with these problems. Let $\mathscr{H}_t = \{(\mathbf{s}_j,\mathbf{s}_j',r_j) \mid j = \max(1, t-M+1),\ldots,t\}$, where M is the window size. We only use the samples in $\mathscr{H}_t$ to evaluate $\mathbf{h}_t$, $\mathbf{g}_t$, p_t, and q_t; that is,

$$\begin{aligned}\widetilde{\mathbf{h}}_t &= \sum_{j\in\mathscr{H}_t}\Delta k_t\left(\mathbf{s}_j,\mathbf{s}_j'\right)\mathbf{k}_{t-1}\left(\mathbf{s}_j\right), \\ \widetilde{\mathbf{g}}_t &= \sum_{j\in\mathscr{H}_t}k\left(\mathbf{s}_j,\mathbf{s}_t\right)\Delta\mathbf{k}_{t-1}\left(\mathbf{s}_j,\mathbf{s}_j'\right), \\ \widetilde{p}_t &= \sum_{j\in\mathscr{H}_t}k\left(\mathbf{s}_j,\mathbf{s}_t\right)\Delta k_t\left(\mathbf{s}_j,\mathbf{s}_j'\right) + \eta, \\ \widetilde{q}_t &= \sum_{j\in\mathscr{H}_t}k\left(\mathbf{s}_j,\mathbf{s}_t\right)r_j.\end{aligned} \tag{23}$$

Then, similar to those in the first case, $\mathbf{A}_t^{-1}$ and $\boldsymbol{\alpha}_t$ can be derived as follows:

$$\mathbf{A}_t^{-1} = \frac{1}{m_t}\begin{bmatrix} m_t\widetilde{\mathbf{A}}_t^{-1} + \widetilde{\mathbf{A}}_t^{-1}\widetilde{\mathbf{h}}_t\widetilde{\mathbf{g}}_t^{\top}\widetilde{\mathbf{A}}_t^{-1} & -\widetilde{\mathbf{A}}_t^{-1}\widetilde{\mathbf{h}}_t \\ -\widetilde{\mathbf{g}}_t^{\top}\widetilde{\mathbf{A}}_t^{-1} & 1\end{bmatrix}, \tag{24}$$

$$\boldsymbol{\alpha}_t = \frac{1}{m_t}\begin{bmatrix} m_t\widetilde{\boldsymbol{\alpha}}_t - \widetilde{\mathbf{A}}_t^{-1}\widetilde{\mathbf{h}}_t\left(\widetilde{q}_t - \widetilde{\mathbf{g}}_t^{\top}\widetilde{\boldsymbol{\alpha}}_t\right) \\ \widetilde{q}_t - \widetilde{\mathbf{g}}_t^{\top}\widetilde{\boldsymbol{\alpha}}_t\end{bmatrix}, \tag{25}$$

where $m_t = \widetilde{p}_t - \widetilde{\mathbf{g}}_t^{\top}\widetilde{\mathbf{A}}_t^{-1}\widetilde{\mathbf{h}}_t$ and $\widetilde{\boldsymbol{\alpha}}_t$ is the same as the updated $\boldsymbol{\alpha}_t$ when the dictionary keeps unchanged.

Finally, we summarize the whole algorithm in Algorithm 1.

Remark 1. Here, we do not restrict the OSKRLSTD-L_2 algorithm to a specific online sparsification method. That means it can be combined with many popular sparsification methods such as the novelty criterion (NC) [27] and the ALD criterion.

```
Input: $\pi$ to be evaluated, $k(\cdot,\cdot)$, $\gamma$, $\eta$, $M$
for $t = 1, 2, \ldots$ do
    if $t == 1$ then
        Initialize $\mathbf{s}_1$, $\mathscr{D}_1 = \{\mathbf{s}_1\}$
        Take $\mathbf{a}_1$ given by $\pi$, and observe $\mathbf{s}_1'$, $r_1$
        Initialize $\mathscr{H}_1 = \{(\mathbf{s}_1, \mathbf{s}_1', r_1)\}$
        Initialize $\mathbf{A}_1^{-1} = (k(\mathbf{s}_1,\mathbf{s}_1)(\Delta k_1(\mathbf{s}_1,\mathbf{s}_1') + \eta))^{-1}$
        Initialize $\boldsymbol{\alpha}_1 = \mathbf{A}_1^{-1}k(\mathbf{s}_1,\mathbf{s}_1)r_1$
    else
        Take $\mathbf{a}_t$ given by $\pi$, and observe $\mathbf{s}_t'$, $r_t$
        Update $\mathscr{H}_t$, $\mathscr{D}_t = \mathscr{D}_{t-1}$
        Update $\boldsymbol{\alpha}_t$, $\mathbf{A}_t^{-1}$ by (21) and (20)
        if $\mathbf{s}_t$ satisfies the sparsification condition then
            $\mathscr{D}_t = \mathscr{D}_{t-1} \cup \{\mathbf{s}_t\}$, $\widetilde{\boldsymbol{\alpha}}_t = \boldsymbol{\alpha}_t$, $\widetilde{\mathbf{A}}_t^{-1} = \mathbf{A}_t^{-1}$
            Compute $\widetilde{\mathbf{h}}_t$, $\widetilde{\mathbf{g}}_t$, $\widetilde{p}_t$ and $\widetilde{q}_t$ by (23)
            Update $\boldsymbol{\alpha}_t$, $\mathbf{A}_t^{-1}$ by (25) and (24)
        end if
    end if
    $\mathbf{s}_{t+1} = \mathbf{s}_t'$
end for
```

Algorithm 1: OSKRLSTD-L_2.

Remark 2. Although the OSKRLSTD-L_2 algorithm is designed for infinite horizon tasks, it can be modified for episodic tasks. When $\mathbf{s}_t'$ is an absorbing state, it only requires setting $\gamma = 0$ temporarily and setting $\mathbf{s}_{t+1}$ as the start state of next episode.

Remark 3. Our simulation results show that a big sliding window cannot help improve the convergence performance of the OSKRLSTD-L_2 algorithm. Thus, to save memory and reduce the computational cost, M should be set to a small integer.

3.2. OSKRLSTD-L_1 Algorithm.

In this subsection, we use L_1 regularization and online sparsification to derive the second OSKRLSTD algorithm, which is called OSKRLSTD-L_1.

First, we try to derive the L_1-regularized solution of (11). Add an L_1-norm penalty into (11); that is,

$$\begin{aligned}\boldsymbol{\alpha}_t &= \widehat{f}\left(\boldsymbol{\alpha}_t\right) \\ &= \underset{\boldsymbol{\beta}\in\mathbb{R}^{n_t}}{\arg\min}\left\|\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - \left(\widehat{R}_t + \gamma\left(\widehat{\mathbf{K}}_t'\right)^{\top}\boldsymbol{\alpha}_t\right)\right\|_2^2 + 2\xi\left\|\boldsymbol{\beta}\right\|_1,\end{aligned} \tag{26}$$

where $\xi \in [0,\infty)$ is a regularization parameter. However, $\|\boldsymbol{\beta}\|_1$ is not differentiable. Similar to Painter-Wakefield and Parr in [28], we resort to the subdifferential of $g(\boldsymbol{\beta}) = \|\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - (\widehat{R}_t + \gamma(\widehat{\mathbf{K}}_t')^{\top}\boldsymbol{\alpha}_t)\|_2^2 + 2\xi\|\boldsymbol{\beta}\|_1$; that is,

$$\begin{aligned}\nabla g\left(\boldsymbol{\beta}\right) = {}& 2\widehat{\mathbf{K}}_t\left(\widehat{\mathbf{K}}_t^{\top}\boldsymbol{\beta} - \left(\widehat{R}_t + \gamma\left(\widehat{\mathbf{K}}_t'\right)^{\top}\boldsymbol{\alpha}_t\right)\right) \\ & + 2\xi\,\mathrm{sgn}\left(\boldsymbol{\beta}\right),\end{aligned} \tag{27}$$

where $\operatorname{sgn}(\boldsymbol{\beta})$ is the set-valued function defined componentwise as

$$\operatorname{sgn}\left(\beta_j\right) = \begin{cases} \{+1\} & \beta_j > 0 \\ [-1, +1] & \beta_j = 0 \\ \{-1\} & \beta_j < 0. \end{cases} \tag{28}$$

Let $\nabla g(\boldsymbol{\beta}) = \mathbf{0}$, so that

$$\widehat{\mathbf{K}}_t \left(\widehat{\mathbf{K}}_t^{\top} \boldsymbol{\beta}^* - \gamma \left(\widehat{\mathbf{K}}_t' \right)^{\top} \boldsymbol{\alpha}_t \right) = \widehat{\mathbf{K}}_t \widehat{R}_t - \xi \operatorname{sgn}\left(\boldsymbol{\beta}^*\right). \tag{29}$$

Since $\mathbf{w}_t = \mathbf{u}^*$, we also have $\boldsymbol{\alpha}_t = \boldsymbol{\beta}^*$ from (9). Then, the above equation can be rewritten as

$$\widehat{\mathbf{K}}_t \left(\widehat{\mathbf{K}}_t - \gamma \widehat{\mathbf{K}}_t' \right)^{\top} \boldsymbol{\alpha}_t = \widehat{\mathbf{K}}_t \widehat{R}_t - \xi \operatorname{sgn}\left(\boldsymbol{\alpha}_t\right), \tag{30}$$

where $\operatorname{sgn}(\boldsymbol{\alpha}_t)$ has the same meaning as $\operatorname{sgn}(\boldsymbol{\beta})$. To avoid the singularity of $\widehat{\mathbf{K}}_t(\widehat{\mathbf{K}}_t - \gamma \widehat{\mathbf{K}}_t')^{\top}$ and further reduce the complexity of the subsequent derivation, we introduce $\eta \boldsymbol{\alpha}_t$ into both sides; that is,

$$\begin{aligned} &\left(\widehat{\mathbf{K}}_t \left(\widehat{\mathbf{K}}_t - \gamma \widehat{\mathbf{K}}_t' \right)^{\top} + \eta \mathbf{I}_t \right) \boldsymbol{\alpha}_t \\ &\quad = \widehat{\mathbf{K}}_t \widehat{R}_t + \eta \boldsymbol{\alpha}_t - \xi \operatorname{sgn}\left(\boldsymbol{\alpha}_t\right), \end{aligned} \tag{31}$$

where $\eta \in [0, \infty)$ is a regularization parameter. Obviously, the left hand side of (31) is the same as that of (14). Thus, from (16), the above equation can be rewritten as

$$\mathbf{A}_t \boldsymbol{\alpha}_t = \mathbf{b}_t + \eta \boldsymbol{\alpha}_t - \xi \operatorname{sgn}\left(\boldsymbol{\alpha}_t\right). \tag{32}$$

Then, we have the following fixed-point equation:

$$\boldsymbol{\alpha}_t = \boldsymbol{\mu}_t + \mathbf{A}_t^{-1} \left(\eta \boldsymbol{\alpha}_t - \xi \operatorname{sgn}\left(\boldsymbol{\alpha}_t\right) \right), \tag{33}$$

where $\boldsymbol{\mu}_t$ denotes

$$\boldsymbol{\mu}_t = \mathbf{A}_t^{-1} \mathbf{b}_t. \tag{34}$$

Unfortunately, here, $\boldsymbol{\alpha}_t$ cannot be solved analytically.

Second, we investigate how to find the fixed point of (33). In L_1-regularized LSTD algorithms [5, 29], researchers often used the LASSO method to tackle this problem. However, the LASSO method is inherently a batch method and is unsuitable for online learning. Instead, we resort to the fixed-point subiteration method introduced in [13]. We first use the sign function $\operatorname{sign}(\boldsymbol{\alpha}_t)$ to replace $\operatorname{sgn}(\boldsymbol{\alpha}_t)$ in (33). Then, we can construct the following subiteration:

$$\boldsymbol{\alpha}_t^{l+1} = \boldsymbol{\mu}_t + \mathbf{A}_t^{-1} \left(\eta \boldsymbol{\alpha}_t^{l} - \xi \operatorname{sign}\left(\boldsymbol{\alpha}_t^{l}\right) \right), \tag{35}$$

where $l \in \mathbb{N}^+$ denotes the lth subiteration and $\boldsymbol{\alpha}_t^1$ is initialized to $\boldsymbol{\mu}_t$ since the fixed point will be close to $\boldsymbol{\mu}_t$ if η and ξ are small. If the subiteration number reaches a preset value $N \in \mathbb{N}^+$ or $\|\boldsymbol{\alpha}_t^{\nu+1} - \boldsymbol{\alpha}_t^{\nu}\|$ is less than or equal to a preset threshold $\varepsilon \in \mathbb{R}^+$, the subiteration will stop. From (32) and (28), if $|(\mathbf{b}_t + \eta \boldsymbol{\alpha}_t - \mathbf{A}_t \boldsymbol{\alpha}_t)_j| < \xi$, $\alpha_{t,j}$ should be 0. Obviously, the replacement of $\operatorname{sgn}(\boldsymbol{\alpha}_t)$ makes $\boldsymbol{\alpha}_t$ lose the ability to select features. To remedy this situation, after the whole subiteration, we remove the weakly dependent elements from $\mathscr{D}_t$ according to the magnitude of $\boldsymbol{\alpha}_t$; that is,

$$\mathscr{D}_t = \Psi_{\mathscr{I}_t}\left(\mathscr{D}_t\right), \tag{36}$$

where $\Psi_{\mathscr{I}_t}(\cdot)$ denotes the operation to remove the elements indexed by the set $\mathscr{I}_t$, which is determined by

$$\mathscr{I}_t = \left\{ j \mid -\nu \le \alpha_{t,j} \le \nu,\ j = 1, \ldots, n_t - 1 \right\}, \tag{37}$$

where $\nu \in \mathbb{R}^+$ is a preset threshold. Note that we do not remove the last element $\mathbf{d}_{n_t}$ of $\mathscr{D}_t$, since $|\alpha_{n_t}|$ is probably very small, especially when $\mathbf{d}_{n_t}$ is just added to $\mathscr{D}_t$. Similarly, we perform $\Psi_{\mathscr{I}_t}(\boldsymbol{\alpha}_t)$ and $\Psi_{\mathscr{I}_t}(\boldsymbol{\mu}_t)$ to remove the weakly dependent coefficients. From (16), $\mathbf{A}_t^{-1}$ also requires removing some rows and columns. Unfortunately, we cannot use the method in [30] to do this like Chen et al. in [13], since $\mathbf{A}_t^{-1}$ is not a symmetry matrix. Considering that $\mathbf{b}_t$ will remove the corresponding elements if $\mathscr{D}_t$ is pruned, we directly perform $\Psi_{\mathscr{I}_t}(\mathbf{A}_t^{-1})$ to remove the rows and columns indexed by $\mathscr{I}_t$. Although this method may bring some bias into $\mathbf{A}_t^{-1}$, our simulation results show that it is feasible and effective. The whole fixed-point subiteration and online pruning algorithm are summarized in Algorithm 2.

```
Input: $\mathscr{D}_t$, $\boldsymbol{\mu}_t$, $\mathbf{A}_t^{-1}$, $\eta$, $\xi$, $N$, $\varepsilon$, $\nu$
Initialize: $\boldsymbol{\alpha}_t^1 = \boldsymbol{\mu}_t$
for $l = 1$ to $N$ do
    Update $\boldsymbol{\alpha}_t^l$ by (35)
    if $\|\boldsymbol{\alpha}_t^{l+1} - \boldsymbol{\alpha}_t^l\| \le \varepsilon$ then
        Break out of the loop
    end if
end for
$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_t^{l+1}$
Determine the index set $\mathscr{I}_t$ by (37)
Perform $\Psi_{\mathscr{I}_t}(\mathscr{D}_t)$, $\Psi_{\mathscr{I}_t}(\boldsymbol{\alpha}_t)$, $\Psi_{\mathscr{I}_t}(\boldsymbol{\mu}_t)$ and $\Psi_{\mathscr{I}_t}(\mathbf{A}_t^{-1})$
```

Algorithm 2: Fixed-point subiteration and online pruning.

Remark 4. Our simulation results show that Algorithm 2 will converge in few iterations. Thus, Algorithm 2 does not become the computational bottleneck of the OSKRLSTD-L_1 algorithm, and the maximum subiteration number N can be set to a small positive integer.

Third, we derive the recursive formulas of $\mathbf{A}_t^{-1}$ and $\boldsymbol{\mu}_t$. Although the dictionary can be pruned by using Algorithm 2, it still has the risk of rapidly growing if new samples are allowed to be added continually. Thus, the conventional sparsification method is also required to be considered here. Similar to Section 3.1, there are two cases under online sparsification. Since $\mathbf{A}_t$ and $\boldsymbol{\mu}_t$ have the same definitions as $\mathbf{A}_t$ and $\boldsymbol{\alpha}_t$ in the OSKRLSTD-L_2 algorithm, we can directly use (20) and (24) for updating $\mathbf{A}_t^{-1}$ and rewrite (21) and (25)

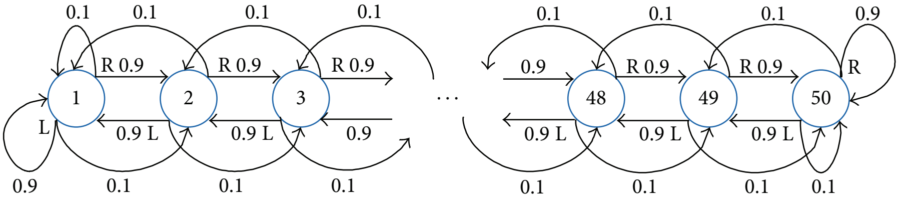


Figure 1: The 50-state chain problem.

```
Input: $\pi$ to be evaluated, $k(\cdot,\cdot)$, $\gamma$, $\eta$, $\xi$, $M$, $N$, $\varepsilon$, $\nu$
for $t = 1, 2, \ldots$ do
    if $t == 1$ then
        Initialize $\mathbf{s}_1$, $\mathscr{D}_1 = \{\mathbf{s}_1\}$
        Take $\mathbf{a}_1$ given by $\pi$, and observe $\mathbf{s}_1'$, $r_1$
        Initialize $\mathscr{H}_1 = \{(\mathbf{s}_1, \mathbf{s}_1', r_1)\}$
        Initialize $\mathbf{A}_1^{-1} = (k(\mathbf{s}_1, \mathbf{s}_1)(\Delta k_1(\mathbf{s}_1, \mathbf{s}_1') + \eta))^{-1}$
        Initialize $\boldsymbol{\mu}_1 = \mathbf{A}_1^{-1} k(\mathbf{s}_1, \mathbf{s}_1) r_1$
        Perform Algorithm 2
    else
        Take $\mathbf{a}_t$ given by $\pi$, and observe $\mathbf{s}_t'$, $r_t$
        Update $\mathscr{H}_t$, $\mathscr{D}_t = \mathscr{D}_{t-1}$
        Update $\boldsymbol{\mu}_t$, $\mathbf{A}_t^{-1}$ by (38) and (20)
        $\widehat{\mathbf{A}}_t^{-1} = \mathbf{A}_t^{-1}$, $\widehat{\boldsymbol{\mu}}_t = \boldsymbol{\mu}_t$
        Perform Algorithm 2
        if $\mathbf{s}_t$ satisfies the sparsification condition then
            $\mathscr{D}_t = \mathscr{D}_{t-1} \cup \{\mathbf{s}_t\}$, $\widetilde{\boldsymbol{\mu}}_t = \widehat{\boldsymbol{\mu}}_t$, $\widetilde{\mathbf{A}}_t^{-1} = \widehat{\mathbf{A}}_t^{-1}$
            Compute $\widetilde{\mathbf{h}}_t$, $\widetilde{\mathbf{g}}_t$, $\widetilde{p}_t$ and $\widetilde{q}_t$ by (23)
            Update $\boldsymbol{\mu}_t$, $\mathbf{A}_t^{-1}$ by (39) and (24)
            Perform Algorithm 2
        end if
    end if
    $\mathbf{s}_{t+1} = \mathbf{s}_t'$
end for
```

Algorithm 3: OSKRLSTD-L_1.

for updating $\boldsymbol{\mu}_t$. If $\mathbf{s}_t$ dissatisfies the sparsification condition, $\boldsymbol{\mu}_t$ will be updated by

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \frac{\left(r_t - \boldsymbol{\mu}_{t-1}^{\top} \Delta \mathbf{k}_{t-1}\left(\mathbf{s}_t, \mathbf{s}_t'\right)\right) \mathbf{A}_{t-1}^{-1} \mathbf{k}_{t-1}\left(\mathbf{s}_t\right)}{1 + \Delta \mathbf{k}_{t-1}^{\top}\left(\mathbf{s}_t, \mathbf{s}_t'\right) \mathbf{A}_{t-1}^{-1} \mathbf{k}_{t-1}\left(\mathbf{s}_t\right)}. \tag{38}$$

Otherwise, $\boldsymbol{\mu}_t$ will be updated by

$$\boldsymbol{\mu}_t = \frac{1}{m_t} \begin{bmatrix} m_t \widetilde{\boldsymbol{\mu}}_t - \widetilde{\mathbf{A}}_t^{-1} \widetilde{\mathbf{h}}_t \left(\widetilde{q}_t - \widetilde{\mathbf{g}}_t^{\top} \widetilde{\boldsymbol{\mu}}_t\right) \\ \widetilde{q}_t - \widetilde{\mathbf{g}}_t^{\top} \widetilde{\boldsymbol{\mu}}_t \end{bmatrix}, \tag{39}$$

where $\widetilde{\mathbf{h}}_t$, $\widetilde{\mathbf{g}}_t$, $\widetilde{p}_t$, and $\widetilde{q}_t$ are also calculated by (23). Since $\mathscr{D}_t$, $\mathbf{A}_t^{-1}$, and $\boldsymbol{\mu}_t$ will be pruned by Algorithm 2 after the update, it is important to note that $\widetilde{\mathbf{A}}_t^{-1}$ and $\widetilde{\boldsymbol{\mu}}_t$ in (39) denote $\mathbf{A}_t^{-1}$ and $\boldsymbol{\mu}_t$ updated by $\mathscr{D}_{t-1}$ but unpruned by $\Psi_{\mathcal{I}_t}(\cdot)$. Likewise, when (24) is used here, $\widetilde{\mathbf{A}}_t^{-1}$ has the same meaning.

Finally, we summarize the whole algorithm in Algorithm 3. For episodic tasks, the modification is the same as Remark 2. In addition, similar to Remark 3, the sliding-window size M should also be set to a small integer.

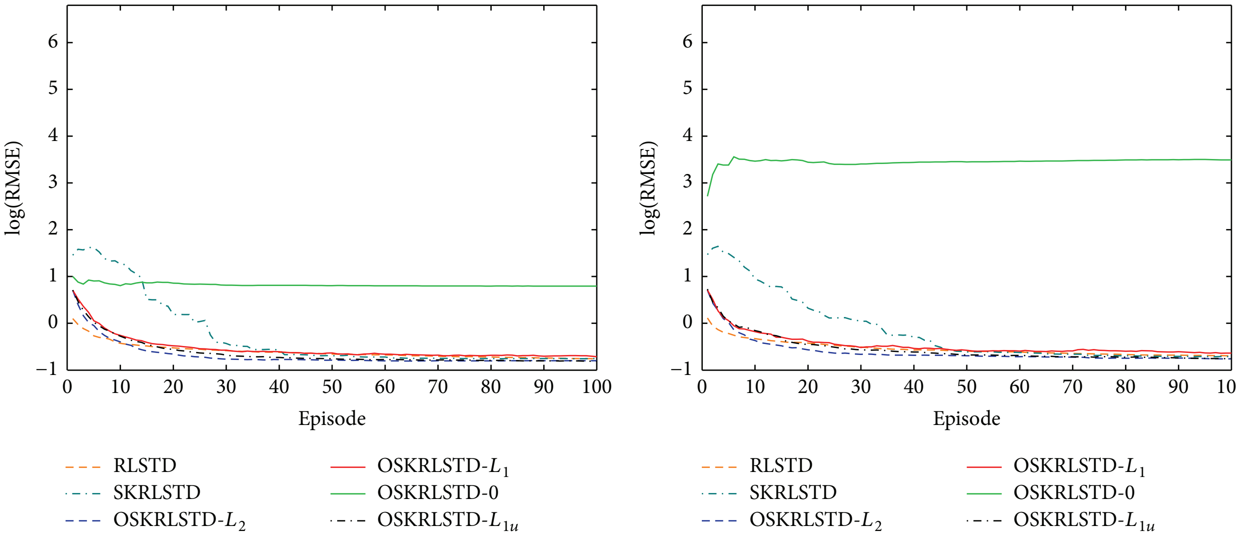
Remark 5. By pruning the weakly dependent features, the OSKRLSTD-L_1 algorithm can yield a much sparser solution than the OSKRLSTD-L_2 algorithm.

4. Simulations

In this section, we use a nonnoise chain and a noise chain [2, 20, 31] to demonstrate the effectiveness of our proposed algorithms. For comparison purposes, RLSTD [1] and SKRLSTD [21] algorithms are also tested in the simulations. To analyze the effect of regularization and online pruning on the performance of our algorithms, the OSKRLSTD-L_2 algorithm with $\eta = 0$ and the OSKRLSTD-L_1 algorithm with $\nu = 0$ (called OSKRLSTD-0 and OSKRLSTD-L_{1u}, resp.) are tested here, too. In addition, the effects of the sliding-window size on the performance of our algorithms and OSKRLSTD-L_{1u} are evaluated as well.

4.1. Simulation Settings. As shown in Figure 1, in both chain problems, each chain consists of 50 states, which are numbered from 1 to 50. For each state, there are two actions available, that is, "left" (L) and "right" (R). Each action succeeds with probability 0.9, changing the state in the intended direction, and fails with probability 0.1, changing the state in the opposite direction. The two boundaries of each chain are dead-ends, and the discount factor γ of each chain is set to 0.9. For the nonnoise chain, the reward is 1 only in states 10 and 41, whereas, for the noise chain, the reward is corrupted by an additive Gaussian noise $0.3\mathcal{N}(0,1)$. Due to the symmetry, the optimal policy for both chains is to go right in states 1–9 and 26–41 and left in states 10–25 and 42–50. Here, we use it as the policy π to be evaluated. Note that the state transition probabilities are available only for solving the true state-value functions V^{π}, and they are assumed to be unknown for all algorithms compared here.

In the implementations of all tested algorithms for both chain problems, the settings are summarized as follows: (i) For all OSKRLSTD algorithms, the Mercer kernel is defined as $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/16)$, the sparsification condition is defined as $\min_{\mathbf{d}_j \in \mathscr{D}_{t-1}} \|\mathbf{s}_t - \mathbf{d}_j\| > 2$, and the sliding-window size M is set to 5. Besides, for the OSKRLSTD-L_1 algorithm, the regularization parameters η and ξ are set to 0.8 and 0.3, the maximum subiteration number N is set to 10, the precision threshold ε is set to 0.1, and the pruning threshold ν is set to 0.4; for the OSKRLSTD-L_{1u} algorithm,



(a) In the nonnoise chain

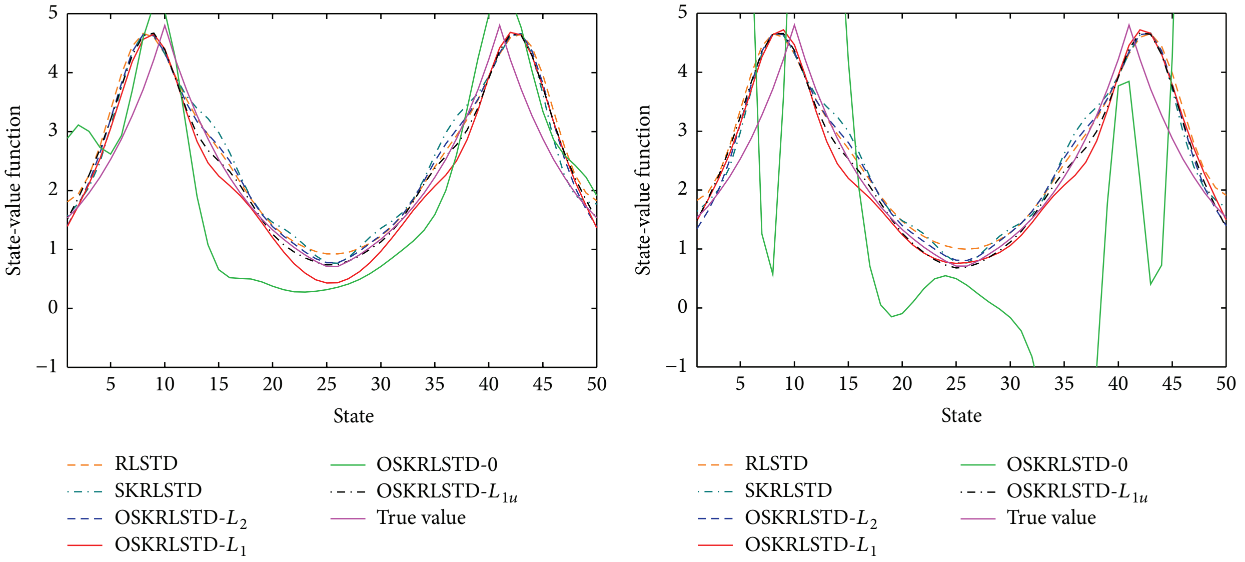
(b) In the noise chain

Figure 2: Learning curves of all tested algorithms.

η, ξ, and N are the same as those in the OSKRLSTD-L_1 algorithm; for the OSKRLSTD-L_2 algorithm, ξ is set to 1. (ii) For the SKRLSTD algorithm, the Mercer kernel and the sparsification condition are the same as those in each OSKRLSTD algorithm. (iii) For the RLSTD algorithm, the feature vector $\boldsymbol{\phi}(\mathbf{s})$ consists of 19 Gaussian radius basis functions (GRBFs) plus a constant term 1, resulting in a total of 20 basis functions. The GRBF has the same definition as the Mercer kernel used in each OSKRLSTD algorithm, and the centers of GRBFs are uniformly distributed over $[1, 50]$. In addition, the variance matrix C_0 of RLSTD is initialized to $0.4\mathbf{I}$, where $\mathbf{I}$ is the 20×20 identity matrix. (iv) In the simulations, each algorithm performs 50 runs, each run includes 100 episodes, and each episode is truncated after 100 time steps. In particular, the SKRLSTD algorithm requires an extra run for offline sparsification before each regular run.

4.2. Simulation Results. We first report the comparison results of all tested algorithms with the simulation settings described in Section 4.1. Their learning curves are shown in Figure 2. At each episode, the root mean square error (RMSE) of each algorithm is calculated by RMSE $= (1/50)\sum_{j=1}^{50}((1/50)\sum_{\mathbf{s}=1}^{50}(\widehat{V}_j^{\pi}(\mathbf{s}) - V^{\pi}(\mathbf{s}))^2)^{0.5}$, where $V^{\pi}(\mathbf{s})$ is solved by (1) and $\widehat{V}_j^{\pi}(\mathbf{s})$ is the approximate value of the jth run. From Figure 2, we can observe that (i) OSKRLSTD-L_2 and OSKRLSTD-L_1 can obtain the similar performance as RLSTD and converge much faster than SKRLSTD. (ii) Without regularization, the performance of OSKRLSTD-0 becomes very poor, especially in the noise chain. In contrast, OSKRLSTD-L_2 and OSKRLSTD-L_1 still perform well. (iii) The performance of OSKRLSTD-L_{1u} is only slightly better than that of OSKRLSTD-L_1, which indicates that online pruning has little effect on the performance. Figure 3 illustrates $\widehat{V}^{\pi}(\mathbf{s})$ approximated by all tested algorithms at the final episode. Clearly, OSKRLSTD-0 has lost the ability to approximate $V^{\pi}(\mathbf{s})$ of the noise chain. Figure 4 shows the dictionary growth curves of all tested algorithms. Compared with RLSTD and SKRLSTD, all OSKRLSTD algorithms can construct the dictionary automatically, and OSKRLSTD-L_1 yields a much sparser dictionary. Figure 5 shows the average subiterations per time step in OSKRLSTD-L_1 and OSKRLSTD-L_{1u}. As episodes increase, the subiterations decline gradually. In addition, online pruning can reduce the subiterations significantly. Even in the noise chain, the subiterations are small. Finally, the main simulation results of all tested algorithms at the final episode are summarized in Table 1.

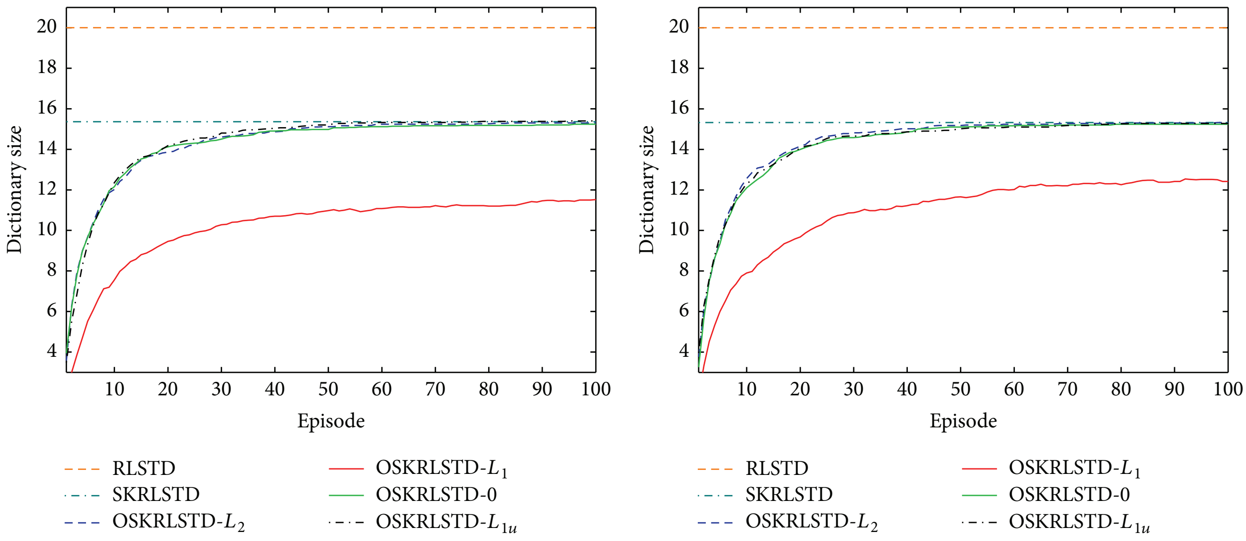
Next, we evaluate the effect of the sliding-window size on our proposed algorithms and OSKRLSTD-L_{1u} with $M = 1, 5, 10, \ldots, 45, 50$. The logarithmic RMSEs of each algorithm at the final episode are illustrated in Figure 6. Note that the parameter settings of these algorithms are the same as those described in Section 4.1 except for M. From Figure 6, OSKRLSTD-L_1 and OSKRLSTD-L_{1u} obviously become worse rather than better as the window size increases, and OSKRLSTD-L_2 has a strong adaptability to different window sizes. The reason for this result is analyzed as follows: From the derivation of our algorithms, the influence of the window size is mainly manifest in $\mathbf{A}_t^{-1}$. Since here $\mathbf{A}_t^{-1}$ is calculated by recursive update instead of matrix inversion and samples are used one by one, using too many history samples together may increase the calculation error. In OSKRLSTD-L_2, a moderate regularization parameter η can relieve the influence of this error. In contrast, in OSKRLSTD-L_1 and OSKRLSTD-L_{1u}, the subiteration may expand the influence. Especially for OSKRLSTD-L_1, online pruning can introduce the new error, which further worsens the convergence performance. To verify the above analysis, we reset $\eta = 0.6$, $\xi = 0.3$, and $N = 1$ for OSKRLSTD-L_1 and OSKRLSTD-L_{1u} and reevaluate the effect of the window



(a) In the nonnoise chain

(b) In the noise chain

Figure 3: $\widehat{V}^{\pi}(\mathbf{s})$ approximated by all tested algorithms at the final episode.



(a) In the nonnoise chain

(b) In the noise chain

Figure 4: Dictionary growth curves of all tested algorithms.

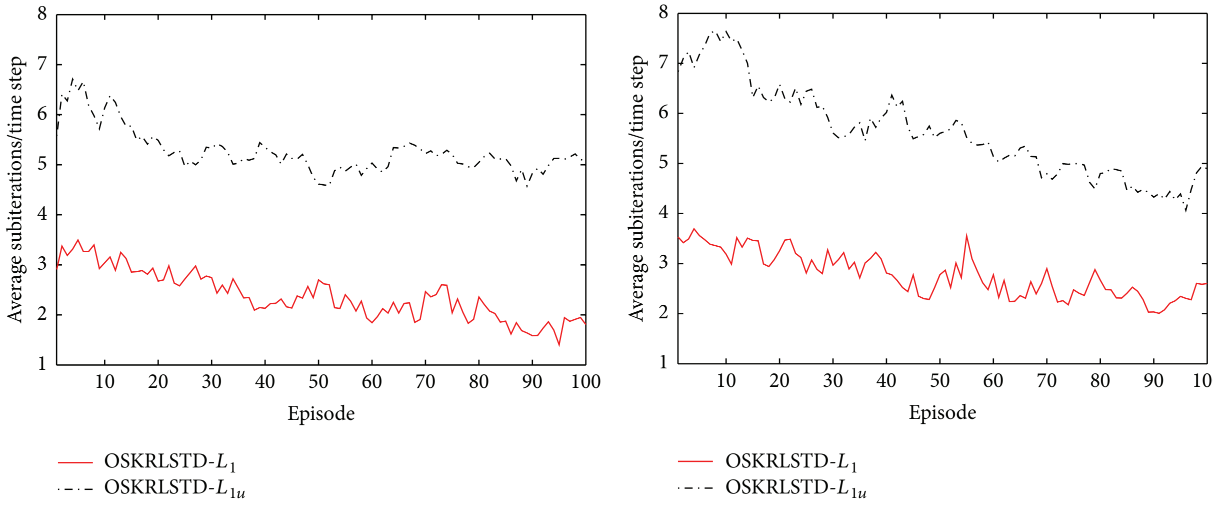
size. The new results are illustrated in Figure 7. As expected, OSKRLSTD-L_1 and OSKRLSTD-L_{1u} can also adapt to M. Nevertheless, there is still no proof that a big window size can help improve the convergence performance of OSKRLSTD-L_2 and OSKRLSTD-L_1. Thus, as stated in Remark 3, M is suggested to be set to a small integer in practice.

5. Conclusion

As an important approach for policy evaluation, LSTD algorithms can use samples more efficiently and eliminate all step-size parameters. But they require users to design the feature vector manually and often require many features to approximate state-value functions. Recently, there are some works on these issues by combining with sparse kernel methods. However, these works do not consider regularization and their sparsification processes are batch or offline. In this paper, we propose two online sparse kernel recursive least-squares TD algorithms with L_2 and L_1 regularization, that is, OSKRLSTD-L_2 and OSKRLSTD-L_1. By using Bellman operator along with projection operator, our derivation is more simple. By combining online sparsification, L_2 and L_1 regularization, recursive least squares, a sliding window, and the fixed-point subiteration, our algorithms not only can construct the feature dictionary online but also can avoid overfitting and eliminate the influence of noise. These

Table 1: Main simulation results on both chains at the final episode.

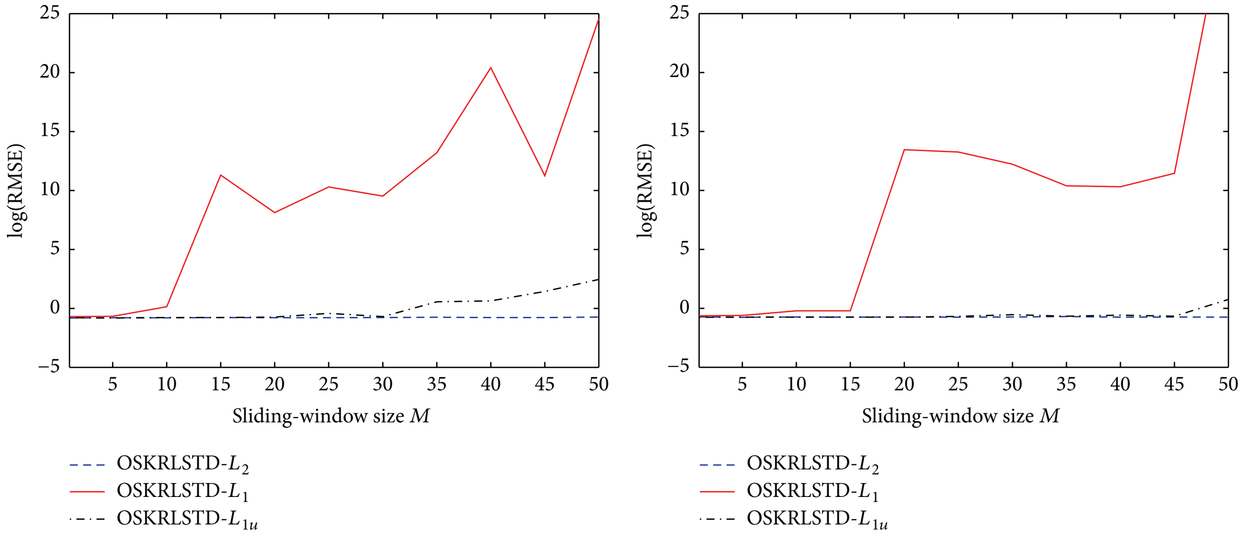
Algorithm		Nonnoise chain			Noise chain	
	RMSE	Dictionary size	Subiterations	RMSE	Dictionary size	Subiterations
RLSTD	0.47 ± 0.03	20	—	0.50 ± 0.04	20	—
SKRLSTD	0.47 ± 0.05	15.36 ± 0.78	—	0.49 ± 0.06	15.32 ± 0.71	—
OKRLSTD-L_2	0.45 ± 0.05	15.30 ± 0.81	—	0.47 ± 0.04	15.32 ± 0.84	—
OKRLSTD-L_1	0.49 ± 0.08	11.52 ± 1.16	1.81 ± 1.82	0.53 ± 0.10	12.42 ± 1.13	2.60 ± 2.56
OKRLSTD-0	2.21 ± 0.05	15.25 ± 0.87	—	32.92 ± 68.67	15.24 ± 0.77	—
OKRLSTD-L_{1u}	0.44 ± 0.05	15.40 ± 0.76	5.08 ± 3.24	0.47 ± 0.05	15.28 ± 0.88	4.90 ± 3.26



(a) In the nonnoise chain

(b) In the noise chain

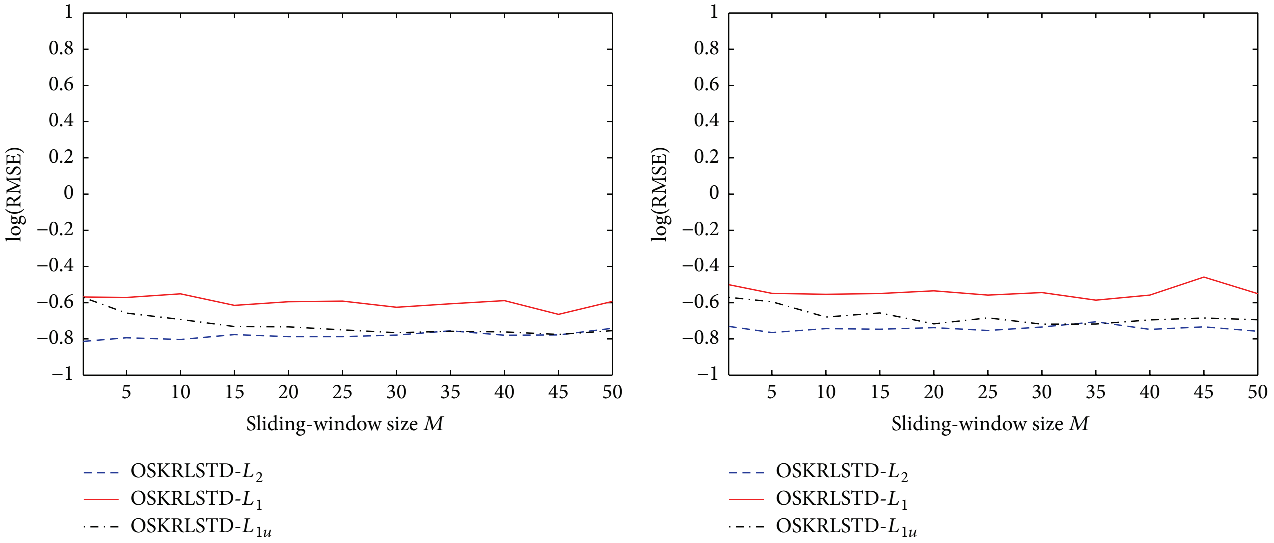
Figure 5: Average subiterations in OSKRLSTD-L_1 and OSKRLSTD-L_{1u} algorithms.



(a) In the nonnoise chain

(b) In the noise chain

Figure 6: Effect of the sliding-window size M on three OSKRLSTD algorithms.



(a) In the nonnoise chain

(b) In the noise chain

Figure 7: Effect of the sliding-window size M on three OSKRLSTD algorithms with new parameters.

advantages make them more suitable for online RL problems with a large or continuous state space. In particular, compared with the OSKRLSTD-L_2 algorithm, the OSKRLSTD-L_1 algorithm can yield a much sparser dictionary. Finally, we illustrate the performance of our algorithms and compare them with RLSTD and SKRLSTD algorithms by several simulations.

There are also some interesting topics to be studied in future work: (i) How to select proper regularization parameter should be investigated. (ii) A more thorough simulation analysis is needed, including an extension of our algorithms to learning control problems. (iii) Eligibility traces would be combined for further improving the performance of our algorithms. (iv) The convergence and prediction error bounds of our algorithms will be analyzed theoretically.

Competing Interests

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

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant nos. 61300192 and 11261015, the Fundamental Research Funds for the Central Universities under Grant no. ZYGX2014J052, and the Natural Science Foundation of Hainan Province, China, under Grant no. 613153.

References

[1] S. J. Bradtke and A. G. Barto, “Linear least-squares algorithms for temporal difference learning,” *Machine Learning*, vol. 22, no. 1–3, pp. 33–57, 1996.

[2] J. A. Boyan, “Technical update: least-squares temporal difference learning,” *Machine Learning*, vol. 49, no. 2-3, pp. 233–246, 2002.

[3] X. Xu, H.-G. He, and D. Hu, “Efficient reinforcement learning using recursive least-squares methods,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 259–292, 2002.

[4] Z. Wu, J. Shi, X. Zhang, W. Ma, and B. Chen, “Kernel recursive maximum correntropy,” *Signal Processing*, vol. 117, pp. 11–16, 2015.

[5] J. Z. Kolter and A. Y. Ng, “Regularization and feature selection in least-squares temporal difference learning,” in *Proceedings of the 26th International Conference on Machine Learning (ICML ’09)*, pp. 521–528, ACM, Montreal, Canada, June 2009.

[6] S. Chen, G. Chen, and R. Gu, “An efficient L2-norm regularized least-squares temporal difference learning algorithm,” *Knowledge-Based Systems*, vol. 45, pp. 94–99, 2013.

[7] L. Baird, “Residual algorithms: reinforcement learning with function approximation,” in *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37, Morgan Kaufmann, Tahoe City, Calif, USA, 1995.

[8] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, Cambridge, UK, 2004.

[9] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*, John Wiley & Sons, 2010.

[10] B. Chen, S. Zhao, P. Zhu, and J. C. Principe, “Quantized kernel least mean square algorithm,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 22–32, 2012.

[11] 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.

[12] B. Chen, S. Zhao, P. Zhu, and J. C. Principe, “Quantized kernel recursive least squares algorithm,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 9, pp. 1484–1491, 2013.

[13] B. Chen, N. Zheng, and J. C. Principe, “Sparse kernel recursive least squares using L1 regularization and a fixed-point sub-iteration,” in *Proceedings of the IEEE International Conference on*

Acoustics, Speech, and Signal Processing (ICASSP '14), pp. 5257–5261, Florence, Italy, May 2014.

[14] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*, pp. 201–208, ACM, Bonn, Germany, August 2005.

[15] S. Yahyaa and B. Manderick, "Knowledge gradient exploration in online kernel-based LSPI," in *Proceedings of the 25th Belgium-Netherlands Artificial Intelligence Conference*, pp. 263–270, Delft, The Netherlands, 2013.

[16] M. Robards, P. Sunehag, S. Sanner, and B. Marthi, "Sparse kernel-sarsa(λ) with an eligibility trace," in *Machine Learning and Knowledge Discovery in Databases—European Conference, ECML PKDD 2011, Part III, Athens, Greece, September 5–9, 2011*, pp. 1–17, Springer, 2011.

[17] X. Chen, Y. Gao, and R. Wang, "Online selective kernel-based temporal difference learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 12, pp. 1944–1956, 2013.

[18] J. Bae, L. G. Sanchez Giraldo, E. A. Pohlmeyer, J. T. Francis, J. C. Sanchez, and J. C. Príncipe, "Kernel temporal differences for neural decoding," *Computational Intelligence and Neuroscience*, vol. 2015, Article ID 481375, 17 pages, 2015.

[19] X. Xu, "A sparse kernel-based least-squares temporal difference algorithm for reinforcement learning," in *Advances in Natural Computation: Second International Conference, Part I, Xi'an, China, September 24–28, 2006*, pp. 47–56, Springer, Berlin, Germany, 2006.

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

[21] H. S. Jakab and L. Csató, "Novel feature selection and kernel-based value approximation method for reinforcement learning," in *Proceedings of the 23rd International Conference on Artificial Neural Networks*, pp. 170–177, Springer, Sofia, Bulgaria, September 2013.

[22] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Proceedings of the 26th International Conference on Machine Learning (ICML '09)*, pp. 1017–1024, ACM, Montreal, Canada, June 2009.

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

[24] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, 2002.

[25] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006.

[26] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "A sliding-window kernel RLS algorithm and its application to nonlinear channel identification," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, pp. V789–V792, Toulouse, France, May 2006.

[27] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[28] C. Painter-Wakefield and R. Parr, "L_1 regularized linear temporal difference learning," Tech. Rep. CS-2012-01, Department of Computer Science, Duke University, Durham, NC, USA, 2012.

[29] M. Loth, M. Davy, and P. Preux, "Sparse temporal difference learning using LASSO," in *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 352–359, IEEE, Honolulu, Hawaii, USA, April 2007.

[30] S. Van Vaerenbergh, I. Santamaría, W. Liu, and J. C. Príncipe, "Fixed-budget kernel recursive least-squares," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '10)*, pp. 1882–1885, Dallas, Tex, USA, March 2010.

[31] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, no. 6, pp. 1107–1149, 2003.