Approximate Policy Iteration with Unsupervised Feature Learning based on Manifold Regularization

Hongliang Li, Derong Liu, *Fellow, IEEE*, Ding Wang

***Abstract*—In this paper, we develop a novel approximate policy iteration reinforcement learning algorithm with unsupervised feature learning based on manifold regularization. The proposed algorithm can automatically learn data-driven smooth basis representations for value function approximation, which can preserve the intrinsic geometry of the state space of Markov decision processes. Moreover, it can provide a direct basis extension for new samples in both policy learning and policy control processes. We evaluate the effectiveness and efficiency of the proposed algorithm on the inverted pendulum task. Simulation results show that this algorithm can learn smooth basis representations and excellent control policies.**

I. Introduction

Reinforcement learning (RL) [1] is a computational approach which can solve goal-directed sequential decision making problems described by Markov decision processes (MDPs). Although dynamic programming [2] is a standard approach to solve MDPs, it suffers from the "curse of dimensionality" and requires the knowledge of the system model. RL algorithms [3] are practical for MDPs with large discrete or continuous state spaces, and can also deal with the learning scenario when the model is unknown. A closely related topic is adaptive or approximate dynamic programming [4]–[9] which adopts a control-theoretic point of view and terminology.

Batch RL [10] is a subfield of dynamic programming based RL, and it allows to solve MDPs by solving a series of supervised learning problems. The goal of batch RL is to learn a best possible policy from the given training data collecting from the unknown system. Therefore, it can make more efficient use of data and avoid stability issues. However, a major challenge is that it is infeasible to represent the solutions exactly for MDPs with large discrete or continuous state spaces. Approximate policy iteration (API) with function approximation methods [11] can provide a compact representation for value function by storing only the parameters of the approximator. API [12] starts from an initial policy, and iterates between policy evaluation and policy improvement to find an approximate solution to the fixed point of Bellman optimality equation. Bradtke and Barto [13] proposed a popular Least-Squares Temporal Difference (LSTD) algorithm to perform policy evaluation. Lagoudakis and Parr [14] developed a Least-Squares Policy Iteration (LSPI) algorithm by extending the LSTD algorithm to control problems. The LSPI algorithm is offline, off-policy and model-free by learning the state-action value function without the generative model of MDPs, and it is easy to implement because of the use of linear parametric architectures.

Although the RL algorithms with hand-crafted features work well, it is still difficult to design suitable features for many real-world RL problems. When the features are improperly designed, the RL algorithms may have poor performance. To avoid designing features by hand-engineering methods, the desire to learn data-driven features has led to some works by incorporating manifold learning into RL algorithms. Mahadevan et al. [15], [16] proposed a representation policy iteration algorithm for learning representations and policies in MDPs based on Laplacian eigenmaps [17]. The basis functions called proto-value functions were automatically constructed by diagonalizing the graph Laplacian matrix, specifically by using the eigenvectors of the graph Laplacian that correspond to the smallest eigenvalues. Xu et al. [18] presented a K-means clustering based graph Laplacian framework for automatic learning of features in MDPs with continuous state spaces. Huang et al. [19] presented an automatic basis learning for the LSPI algorithm based on isometric mapping [20], where the geodesic distances were estimated by searching the shortest paths in the state graph. However, for the above RL algorithms [15], [18], [19], the basis functions of new unexplored states need to be recalculated using the already learned basis functions during both the policy learning and policy control processes.

Manifold regularization [21], [22] combines the graph Laplacian approach with parametric methods to combine the advantages of the two approaches. To the best of our knowledge, there is still not any work on applying the manifold regularization to learn features for RL problems. In this paper, we develop a novel manifold regularization based API (MR-API) algorithm which can automatically learn smooth basis representations for value function approximation. The learned basis functions can preserve the intrinsic geometry of the state space by learning on the collected samples. Compared with other RL algorithms based on manifold learning [15], [18], [19], the proposed MR-API algorithm can provide a direct basis extension for new unexplored samples in both policy learning and policy control. This greatly reduces the complexity and computational burden. The effectiveness and efficiency of the proposed MR-API scheme is evaluated on the

The authors are with The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (phone: +86-10-82544761; fax: +86-10-82544799; e-mail: hongliang.li@ia.ac.cn; derong.liu@ia.ac.cn; ding.wang@ia.ac.cn).

This work was supported in part by the National Natural Science Foundation of China under Grants 61034002, 61233001, 61273140, and 61304086.

inverted pendulum task. Simulation results show that the MR-API algorithm can learn suitable basis representations and can obtain excellent control performance.

The rest of this paper is organized as follows: Section II provides some backgrounds on MDPs; Section III presents the MR-API algorithm; Section IV provides simulation study; Section V derives conclusions.

II. Problem Statement

A discounted MDP can be expressed by a 5-tuple $(\mathcal{X}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{X}$ denotes the finite or continuous state space, $\mathcal{A} = \{a_1, a_2, \dots, a_c\}$ denotes the finite action space, $P\colon \mathcal{X} \times \mathcal{A} \to P(\cdot|x_t, a_t)$ denotes the Markov transition model which gives the next-state distribution upon taking action a_t at state x_t, $R\colon \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}$ denotes the bounded deterministic reward function which gives an immediate reward $r_t = R(x_t, a_t, x_t')$, and $\gamma \in [0, 1)$ is a predefined discount factor. For many practical control problems, the underlying MDP model is not fully available: The transition model P and the reward function R are not known in advance; The state space $\mathcal{X}$, the action space $\mathcal{A}$ and the discount factor γ are available.

A mapping $\pi\colon \mathcal{X} \to \mathcal{A}$ is called a deterministic stationary Markov policy, hence $\pi(x_t)$ indicates the action taking at state x_t. The state value function V^π of a policy π is defined as the expected total discounted reward

$$V^\pi(x) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| x_0 = x\right], \tag{1}$$

which satisfies the Bellman equation

$$V^\pi(x) = \mathbb{E}_\pi\big[R(x, a, x') + \gamma V^\pi(x')\big]. \tag{2}$$

The goal of RL algorithms is to find a policy that attains the best possible values

$$V^*(x) = \sup_\pi V^\pi(x), \ \forall x \in \mathcal{X}, \tag{3}$$

where V^* is called the optimal value function. A policy π^* is called optimal if it attains the optimal value $V^*(x)$ for any state $x \in \mathcal{X}$, i.e., $V^{\pi^*}(x) = V^*(x)$. The optimal value function V^* satisfies the Bellman optimality equation

$$V^*(x) = \max_{a \in \mathcal{A}} \mathbb{E}\big[R(x, a, x') + \gamma V^*(x')\big]. \tag{4}$$

To develop model-free RL algorithms, the state-action value function $Q^\pi(x, a)$ of any policy π is defined as

$$Q^\pi(x, a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| x_0 = x, a_0 = a\right]. \tag{5}$$

The state-action value function Q^π satisfies the Bellman equation

$$Q^\pi(x, a) = \mathbb{E}_\pi\big[R(x, a, x') + \gamma Q^\pi(x', a')\big]. \tag{6}$$

A policy is called greedy with respect to the state-action value function if

$$\pi(x) = \arg\max_{a \in \mathcal{A}} Q(x, a). \tag{7}$$

The optimal state-action value function Q^* is defined as

$$Q^*(x, a) = \sup_\pi Q^\pi(x, a), \ \forall x \in \mathcal{X}, \forall a \in \mathcal{A}. \tag{8}$$

The optimal state-action value function Q^* satisfies the following Bellman optimality equation

$$Q^*(x, a) = \mathbb{E}\big[R(x, a, x') + \gamma \max_{a'} Q^*(x', a')\big]. \tag{9}$$

The optimal policy π^* can be obtained by

$$\pi^*(x) = \arg\max_{a \in \mathcal{A}} Q^*(x, a). \tag{10}$$

In LSPI, the action-state value function Q^π is approximated by a linear parametric architecture with free parameters $\vec{w}_i$

$$\hat{Q}^{\pi_i}(x, a) = \sum_{j=1}^{h} \phi_j(x, a) w_j = \vec{\phi}^{\mathrm{T}}(x, a)\vec{w}_i, \tag{11}$$

where $\vec{\phi}(x, a) \in \mathbb{R}^h$ denotes the vector of basis functions or features, and $\vec{w}_i = [w_1, w_2, \dots, w_h]^{\mathrm{T}}$ denotes the weight vector. The parameter vector $\vec{w}_i$ can be adjusted appropriately so that the approximate value function is close enough to the exact one. The basis functions are usually selected manually, and can be selected as polynomial bases, radial basis functions, etc. Usually the radial basis functions are used, and they are tiled over the state space uniformly. However, these basis functions cannot capture topological information of the state space and may make the LSPI algorithm perform poorly.

III. Manifold Regularization based Approximate Policy Iteration

In this section, we will develop the MR-API algorithm which learns feature representations using the manifold regularization method and learns policy using the L_2 regularized LSPI method [23]. Fig. 1 summarizes the main processes of the proposed MR-API algorithm: sampling and sub-sampling, unsupervised feature learning based on manifold regularization, and policy learning using the L_2 regularized LSPI. The dashed line in Fig. 1 means that the algorithm does not interact with the environment during learning.

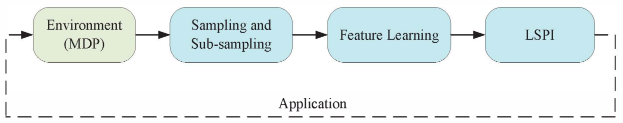

Fig. 1. Framework of MR-API.

In the first step, a set of samples $\mathcal{D}_N = \{(x_k, a_k, r_k, x_k')\}_{k=1}^N$ are sampled from a real-word system or a simulated system. These samples may be collected by using a purely random policy or an arbitrary known policy. The states x_k and x_{k+1} may be sampled independently, or may be sampled along a connected trajectory, i.e., $x_{k+1} = x_k'$. For example, the samples can be collected using an initial policy π_0 for T episodes, and each episode terminates after N_T steps or entering absorbing states. The samples need to cover the state-action space adequately, since the distribution of the training data can greatly affect the performance of the RL algorithm. The training set $\mathcal{D}_N$ usually can be reused at each iteration.

Then a sub-sampling process is usually required to select a much smaller set from the collected samples $\mathcal{D}_N$ to learn basis functions. Some sub-sampling methods can be used, such as the trajectory-based sub-sampling method [15], the random sub-sampling method [15], and the K-means clustering-based sub-sampling method [18]. In this paper, we use the trajectory-based sub-sampling method, which can retain important information about the trajectory. The trajectory-based sub-sampling algorithm follows a greedy strategy: Starting with the null subset, add new samples to the subset whose distances to any sample in the subset are not within the threshold parameter ξ. A maximal subset $\mathcal{D}_M = \{x_k\}_{k=1}^M$ is returned when no more samples can be added. The larger parameter ξ will lead to a smaller number of samples in $\mathcal{D}_M$.

A. Feature Learning

In the feature learning process, the basis functions are expressed by a single hidden layer neural network, where the input weights are generated randomly and the output weights are solved by computing the eigenvectors corresponding to the smallest eigenvalues of the Laplacian matrix. Hence the obtained basis functions are automatically generated by constructing the optimal embedding in a low-dimensional Euclidean space, which preserves the estimated intrinsic geometry of the state space. The target of the manifold regularized feature learning algorithm (see Algorithm 1) is to learn appropriate basis functions for value function approximation in an unsupervised manner. The data set $\mathcal{D}_M = \{x_k\}_{k=1}^M$ for feature learning are obtained by sub-sampling from the original data set $\mathcal{D}_N$.

The basis functions are expressed by a single hidden layer neural network, which is called feature network. The output of the feature network is given by

$$\vec{\varphi}(x_k) = \vec{\sigma}^{\mathrm{T}}(x_k)\beta, \tag{12}$$

where $\vec{\sigma}(x_k) \in \mathbb{R}^{n_h}$ denotes the output vector of the hidden layer with respect to x_k, and $\beta \in \mathbb{R}^{n_h \times n_f}$ denotes the output weight matrix of the feature network. The predefined parameter n_h and n_f are the number of the hidden neurons and the dimension of basis functions, respectively.

The first stage is to construct the hidden layer using a fixed number of neurons. The activation function is selected as the Gaussian function $\sigma(x) = \exp(-\|x - \upsilon\|^2/2b^2)$, where υ is the center and b is the width of the Gaussian function. The center can be randomly generated according to any continuous probability distribution. They can also be selected from the data set $\mathcal{D}_M$ randomly or by the K-means clustering method with $K = n_h$. Therefore, the hidden layer maps the data from the n-dimensional state space into the n_h-dimensional space. We denote a matrix $\Theta = [\vec{\sigma}(x_1), \ldots, \vec{\sigma}(x_M)]^{\mathrm{T}} \in \mathbb{R}^{M \times n_h}$ and a matrix $\Omega = \Theta\beta \in \mathbb{R}^{M \times n_f}$.

In the second stage, the feature learning algorithm aims to solve the optimal weights β^* by the manifold regularization method. According to the smoothness assumption in manifold learning [21], if two points $x_1, x_2 \in \mathcal{X}$ are close to each other, then the conditional probabilities $P(y|x_1)$ and $P(y|x_2)$ are similar. The assumption can be enforced by minimizing the following empirical Laplacian

$$\hat{L} = \frac{1}{2}\sum_{i,j} W_{ij}\|\vec{\varphi}(x_i) - \vec{\varphi}(x_j)\|^2 = \mathrm{tr}\big(\Omega^{\mathrm{T}} L \Omega\big), \tag{13}$$

where $W = [W_{ij}]_{M\times M}$ is a symmetric, sparse adjacency matrix representing the state graph, D is a diagonal matrix with its diagonal elements $D_{ii} = \sum_j^M W_{ij}$, and $L = D - W$ is the combinatorial graph Laplacian which is positive semidefinite. The normalized graph Laplacian $I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ can also be used instead of the combinatorial graph Laplacian. In this paper, we use the K nearest neighbors method and the heat kernel $\exp(-\|x_i - x_j\|^2/\varrho)$ ($\varrho > 0$) to construct the graph matrix. In the manifold regularized feature learning, the optimal weight matrix β^* can be found by solving the following optimization problem

$$\min_{\beta \in \mathbb{R}^{n_h \times n_f}} \|\beta\|^2 + \alpha \mathrm{tr}\big(\Omega^{\mathrm{T}} L \Omega\big), \tag{14}$$

where the first term is an extrinsic regularization, the second term is an intrinsic regularization, and $\alpha > 0$ is a penalty coefficient to trade off these two regularization terms. Substituting $\Omega = \Theta\beta$, the objective function in (14) is equivalent to

$$\min_{\beta \in \mathbb{R}^{n_h \times n_f}} \|\beta\|^2 + \alpha \mathrm{tr}(\beta^{\mathrm{T}} \Theta^{\mathrm{T}} L \Theta \beta). \tag{15}$$

To avoid degenerate solutions, some additional constraint conditions are imposed

$$\min_{\beta \in \mathbb{R}^{n_h \times n_f}} \|\beta\|^2 + \alpha \mathrm{tr}(\beta^{\mathrm{T}} \Theta^{\mathrm{T}} L \Theta \beta) \tag{16}$$

$$\text{s.t.} \quad \beta \Theta^{\mathrm{T}} \Theta \beta = I_{n_f}. \tag{17}$$

According to [21], [24], the optimal weight matrix β^* to the above problem is given by choosing β^* as the matrix whose columns are the smoothest eigenvectors corresponding to the smallest eigenvalues of the generalized eigenvalue problem

$$(I_{n_h} + \alpha \Theta^{\mathrm{T}} L \Theta)\upsilon = \lambda \Theta^{\mathrm{T}} \Theta \upsilon. \tag{18}$$

We discard the first eigenvector corresponding to the smallest eigenvalue 0. Define $\lambda_1, \lambda_2, \ldots, \lambda_{n_f+1}$ ($\lambda_1 \le \lambda_2 \le \cdots \le \lambda_{n_f+1}$) be the $n_f + 1$ smallest eigenvalues of (18) and $\upsilon_1, \upsilon_2, \ldots, \upsilon_{n_f+1}$ be their corresponding eigenvectors. The optimal weight matrix β^* is given by

$$\beta^* = [\tilde{\upsilon}_2, \tilde{\upsilon}_3, \ldots, \tilde{\upsilon}_{n_f+1}], \tag{19}$$

where $\tilde{\upsilon}_i = \upsilon_i/\|\Theta\upsilon_i\|$ are the normalized eigenvectors. Therefore, the feature network $\vec{\varphi}(x) = \vec{\sigma}^{\mathrm{T}}(x)\beta^*$ can give data-driven feature representations and can offer a natural out-of-sample extension for new states which are not in the data set $\mathcal{D}_M$.

Remark 1: The learned basis functions by this manifold regularized feature learning algorithm can reflect the intrinsic geometric structure of the state space. This proposed algorithm is based on the graph Laplacian built from the state data, and

Algorithm 1 Manifold Regularized Feature Learning

```
// $\mathcal{D}_M = \{x_i\}_{i=1}^M$:  the training data set;
// $K$:  the number of the nearest neighbors of each state;
// $\varrho$:  a positive constant;
// $\alpha$:  a penalty coefficient;
// $n_h$:  the number of the neurons in the hidden layer;
// $n_f$:  the number of the neurons in the output layer.
1 Construct graph Laplacian from $\mathcal{D}_M$.
  If $x_i$ is among $K$ nearest neighbors of $x_j$ or $x_j$ is among $K$ nearest neighbors of $x_i$, put an edge between $x_i$ and $x_j$, and set the weight $W_{ij} = e^{-\|x_i - x_j\|^2/\varrho}$; otherwise, set the weight $W_{ij} = 0$. Compute the Laplacian matrix $L$.
2 Initialize a feature network of $n_h$ hidden neurons with random input weights. Calculate the output matrix of the hidden layer $\Theta$.
3 Compute the generalized eigenvectors $\upsilon_2, \upsilon_3, \ldots, \upsilon_{n_f+1}$ in (18). Obtain the optimal weights $\beta^* = [\tilde{\upsilon}_2, \tilde{\upsilon}_3, \ldots, \tilde{\upsilon}_{n_f+1}]$ where $\tilde{\upsilon}_i = \upsilon_i/\|\Theta\upsilon_i\|$.
4 Return the feature network $\vec{\varphi}(x) = \vec{\sigma}^{\mathrm{T}}(x)\beta^*$.
```

depends on solving the smoothest generalized eigenvectors. These eigenvectors are not used for feature representations directly, but are used as the weight parameters of the feature network. Once the feature network is solved, it can be applied to any new data in the original state space. Therefore, this proposed algorithm can give a straightforward extension for out-of-sample examples by a function without recomputing eigenvectors as in [15] and [19]. Furthermore, the basis extension step for new states during policy learning and policy control in the next subsection is also not required. This greatly reduces the complexity and computational burden.

B. Policy Learning

In the policy learning process, a linear parametric structure based on the learned basis functions is used to approximate the state-action value function. The L_2 regularized least-squares algorithm is used to update the weight vector of the approximate state-action value function. Algorithm 2 gives the whole process of the proposed MR-API algorithm.

The value function can be approximated by the learned basis function directly. To approximate the state-action value function for MDPs with continuous or finite states and c finite actions, the learned basis functions can be repeated for each of the actions

$$\vec{\psi}(x,a) = [\mathrm{I}(a=a_1)\vec{\varphi}(x), \mathrm{I}(a=a_2)\vec{\varphi}(x), \ldots, \mathrm{I}(a=a_c)\vec{\varphi}(x)]^{\mathrm{T}},$$

where $\mathrm{I}(a=a_i)$ is an indicator function, i.e., if $a = a_i$, $\mathrm{I}(a = a_i) = 1$, otherwise, $\mathrm{I}(a=a_i) = 0$. Then, the action-state value function $Q^{\pi_i}(x,a)$ can be approximated by the linear structure

$$\hat{Q}^{\pi_i}(x,a) = \sum_{i=1}^{p} \psi_j(x,a)\omega_j = \vec{\psi}^{\mathrm{T}}(x,a)\vec{\omega}_i, \tag{20}$$

where $p = n_f \times c$ is the number of basis functions of the approximate action-state value function.

In this paper, we use the L_2 regularized LSTD [23] to evaluate the policy π_i. On the data set $\mathcal{D}_N = \{(x_k, a_k, r_k, x_k')\}_{k=1}^N$, define

$$\Psi = \begin{bmatrix} \vec{\psi}(x_1,a_1)^{\mathrm{T}} \\ \vdots \\ \vec{\psi}(x_N,a_N)^{\mathrm{T}} \end{bmatrix}, \Psi' = \begin{bmatrix} \vec{\psi}(x_1',a_1')^{\mathrm{T}} \\ \vdots \\ \vec{\psi}(x_N',a_N')^{\mathrm{T}} \end{bmatrix}, \mathcal{R} = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix}$$

with $a_j' = \pi_i(x_j')$, $j = 1, \ldots, N$. The weights $\vec{\omega}_i$ can be found by solving the following optimization problem

$$\vec{\mu}_i = \arg\min_{\vec{\mu}\in\mathbb{R}^p} \delta\|\vec{\mu}\|^2 + \|\Psi\vec{\mu} - (\mathcal{R} + \gamma\Psi'\vec{\omega})\|^2 \tag{21}$$

$$\vec{\omega}_i = \arg\min_{\vec{\omega}\in\mathbb{R}^p} \|\Psi\vec{\omega} - \Psi\vec{\mu}_i\|^2, \tag{22}$$

where δ controls the complexity of the value function. By setting the gradient of (21) to zero, we have

$$\delta\vec{\mu} + \Psi^{\mathrm{T}}\big[\Psi\vec{\mu} - (\mathcal{R} + \gamma\Psi'\vec{\omega})\big] = 0. \tag{23}$$

If Ψ has more rows than columns and is of full column rank, which is usually the case where the number of training samples is larger than the number of the features, the above equation (23) is overdetermined. Therefore, we have the following closed-form solution

$$\vec{\mu}_i = (\delta I_p + \Psi^{\mathrm{T}}\Psi)^{-1}\Psi^{\mathrm{T}}(\mathcal{R} + \gamma\Psi'\vec{\omega}). \tag{24}$$

Substituting $\vec{\mu}_i$ into (22), the weight vector $\vec{w}_i$ has a closed-form solution

$$\vec{\omega}_i = (\delta I_p + \Psi^{\mathrm{T}}(\Psi - \gamma\Psi'))^{-1}\Psi^{\mathrm{T}}\mathcal{R} \triangleq A^{-1}B. \tag{25}$$

If $\delta = 0$, the L_2 regularized LSTD algorithm (25) reduces to the standard LSTD algorithm. For a sample $(x_k, a_k, r_k, x_k') \in \mathcal{D}_N$, an incremental update equation can be given as follows

$$A \leftarrow A + \delta I_p + \vec{\psi}(x_k,a_k)\big(\vec{\psi}(x_k,a_k) - \gamma\vec{\psi}(x_k', \pi_i(x_k'))\big)^{\mathrm{T}}$$
$$B \leftarrow B + \vec{\psi}(x_k,a_k)r_k.$$

The policy improvement is performed by the greedy policy

$$\pi_{i+1}(x) = \arg\min_{a\in\mathcal{A}} \vec{\psi}^{\mathrm{T}}(x,a)\vec{\omega}_i. \tag{26}$$

These two steps are repeated until there is no change in the policy in which case the iteration has converged to the optimal policy.

IV. Simulation Study

In this section, we illustrate the effectiveness and efficiency of the proposed MR-API algorithm on the inverted pendulum problem. The inverted pendulum problem has been studied as a standard benchmark domain with continuous state spaces and nonlinear dynamics. The task of the inverted pendulum problem is to balance a pendulum of unknown parameters at the upright position by applying forces to the cart to which the pendulum is attached. The state space is 2-dimensional continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The three actions are applying a force of -50 Newtons (left force), 0 Newtons (no force), or 50 Newtons (right force). Uniform noise in $[-10,10]$ is added

Algorithm 2 MR-API

```
// $\varepsilon$: the error condition;
// $i_{\max}$: the maximum number of iterations;
// $\gamma$: the discount factor;
// $\pi_1$: an initial policy;
// $\vec{\omega}_0$: the initial weights.
1  **Sample Collection and Sub-sampling**
2  Collect a set of samples $\mathcal{D}_N = \{(x_k, a_k, r_k, x'_k)\}_{k=1}^N$;
3  Construct a subset of samples $\mathcal{D}_M = \{x_k\}_{k=1}^M \subseteq \mathcal{D}_N$.
4  **Manifold Regularized Feature Learning**
5  Learn basis functions $\vec{\varphi}(x) = \vec{\sigma}^{\mathrm{T}}(x)\beta^*$ on $\mathcal{D}_M$.
6  **Policy Learning**
7  $i \leftarrow 0$
8  **Repeat**
9      $i \leftarrow i + 1$
10     $A \leftarrow 0$, $B \leftarrow 0$
11     **for** each $(x_k, a_k, r_k, x'_k) \in \mathcal{D}_N$ **do**
12     $\vec{\psi}(x_k, a_k) = [\mathrm{I}(a_k = a_1)\vec{\varphi}(x_k), \ldots, \mathrm{I}(a_k = a_c)\vec{\varphi}(x_k)]^{\mathrm{T}}$
13     $\vec{\psi}(x'_k, \pi_i(x'_k)) = [\mathrm{I}(\pi_i(x'_k) = a_1)\vec{\varphi}(x'_k), \ldots,$
                           $\mathrm{I}(\pi_i(x'_k) = a_c)\vec{\varphi}(x'_k)]^{\mathrm{T}}$
14     $A \leftarrow A + \delta I_p + \vec{\psi}(x_k, a_k)\big(\vec{\psi}(x_k, a_k) - \gamma\vec{\psi}(x'_k, \pi_i(x'_k))\big)^{\mathrm{T}}$
15     $B \leftarrow B + \vec{\psi}(x_k, a_k)r_k$
16     **end for**
17     $\vec{\omega}_i = A^{-1}B$
18     $\pi_{i+1}(x'_k) = \arg\max_{a'_k \in \mathcal{A}} \vec{\psi}^{\mathrm{T}}(x'_k, a'_k)\vec{\omega}_i$
19 **Until** $\|\vec{\omega}_i - \vec{\omega}_{i-1}\| < \varepsilon$ or $i = i_{\max}$
20 **Return** $\pi(x) = \max_{a \in \mathcal{A}} \vec{\psi}^{\mathrm{T}}(x, a)\vec{\omega}_i$.
```

to the chosen action. The nonlinear dynamics of the system is defined by

$$\ddot{\theta} = \frac{g\sin(\theta) - \kappa m l \dot{\theta}^2 \sin(2\theta)/2 - \kappa\cos(\theta)u}{4l/3 - \kappa m l \cos^2(\theta)}. \tag{27}$$

We use the same parameters described in [14], where $g = 9.8\mathrm{m/s}^2$ is the gravity constant, $m = 2.0$kg is the mass of the pendulum, $M = 8.0$kg is the mass of the cart, $l = 0.5$m is the length of the pendulum, and $\kappa = 1/(m + M) = 0.1$. The simulation time step is set to 0.1 seconds. A reward of 0 is given as long as the absolute value of the angle of the pendulum does not exceed $\pi/2$. If the absolute value of the angle is greater than $\pi/2$, the episode ends with a penalty reward of -1. The discount factor is set to 0.95. A policy is considered to be successful if it can balance the pendulum for 3000 steps (5 minutes).

Training samples are collected using a random policy that selects actions uniformly, starting in a randomly perturbed state near the equilibrium state $(0, 0)$. This process is repeated for T episodes, and each episode terminates after 100 steps or entering absorbing state. The trajectory-based sub-sampling method is used with $\xi = 0.05$. The structure of the feature network is 2-50-17. The Gaussian functions are used as the activation functions of the feature network, and their parameters are randomly generated according to the uniform distribution in $[-1, 1]$. The combinatorial graph Laplacion is used. The number of nearest neighbor is set to $K = 25$, and the weight is set by the heat kernel with the width $\varrho = 0.5$. The plenty coefficients α and δ are set to 10 and 0.1, respectively. The error condition ε is set to 10^{-5}, and the maximum number of iterations $i_{\max}$ is set to 20.

For the L_2 regularized LSPI algorithm, a set of 17 Gaussian basis functions for each action, i.e., a total of 51 basis functions, are used to approximate the action-state value function. The 17 basis functions includes 16 Gaussian functions and 1 constant term, where the centers are the 16 points of the grid $\{-0.3\pi, -0.1\pi, 0.1\pi, 0.3\pi\} \times \{-1.2, -0.4, 0.4, 1.2\}$ and the width is 0.25. Other parameters are the same as those described in the MR-API algorithm.

The simulation experiment is conducted as follows: Different numbers of sampling episodes T are chosen as $0, 50, \ldots, 500$. For each size of training episodes, the learned policy is evaluated 20 times to estimate the average number of balancing steps. This simulation experiment is repeated for 20 times. The performance of the MR-API is shown in Fig. 2. This performance shows average balancing steps and the 95% confidence intervals over different sample sets, and it also shows the worst and best balancing steps. With 500 training episodes, the average number of balancing steps is about 2950 steps. With only 50 training episodes, successful policies can be found occasionally among the 20 runs. With 500 training episodes, even the worst policy can balance the pendulum for about 2750 steps. Therefore, the MR-API algorithm can learn very good policies given a few hundred training episodes. To compare, the performance of the L_2 regularized LSPI algorithm is given in Fig. 3. The average number of balancing steps in L_2 regularized LSPI is lower than that in MR-API for nearly all the training episodes. Therefore, the MR-API algorithm can return good policy on fewer training samples. The worst policy in L_2 regularized LSPI can only balance the pendulum for about 10 steps, which are much lower than that in MR-API. The failures are mostly caused by a bad distribution of samples. The MR-API can learn data-driven feature representations, thus has better performance than L_2 regularized LSPI on a bad distribution of samples. The first four basis functions using the combinatorial graph Laplacian are plotted in Fig. 4, where the dots indicate the values of the basis functions of the training samples. The obtained approximate value function is shown in Fig. 5. We can see that the learned basis functions in MR-API are quite smooth.

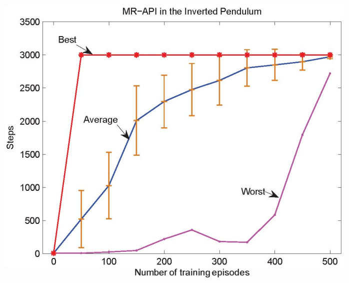

Fig. 2. Performance of MR-API.

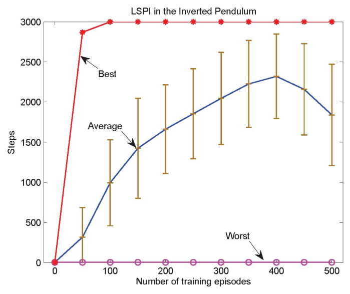

Fig. 3. Performance of L_2 regularized LSPI.

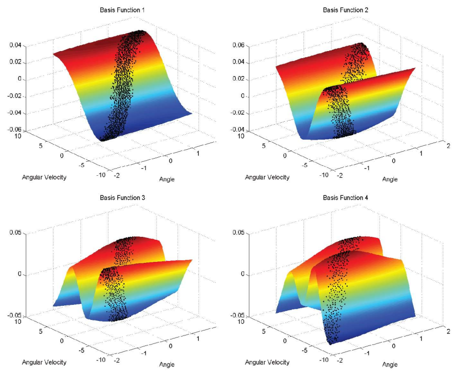

Fig. 4. The first four basis functions learned by MR-API.

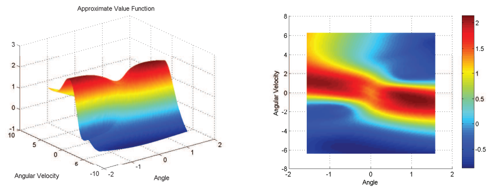

Fig. 5. Left: A 3D view of the approximate value function learned by MR-API. Right: a 2D view of the approximate value function.

V. Conclusions

In this paper, we developed a novel manifold regularization based approximate policy iteration algorithm which can automatically learn data-driven smooth basis representation for value function approximation. Compared with manifold learning based RL algorithms, the proposed algorithm can provide a direct basis extension for new samples in both policy learning and policy control processes. This greatly reduces the complexity and computational burden, compared with RL algorithms based on manifold learning. Simulation results demonstrated that the MR-API algorithm can learn smooth basis representations and can obtain better control performance than the LSPI algorithm. In the future, it is interesting to analyze the error performance for the MR-API algorithm and apply it to more complex control tasks.

References

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

[2] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[3] C. Szepesvari, *Algorithms for Reinforcement Learning*. San Mateo, CA, USA: Morgan & Claypool, 2010.

[4] F. L. Lewis and D. Liu, *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control.* Hoboken, NJ: Wiley, 2013.

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

[6] D. Liu, D. Wang, and H. Li, "Decentralized stabilization for a class of continuous-time nonlinear interconnected systems using online learning optimal control approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 418–428, Feb. 2014.

[7] D. Liu, D. Wang, F. Y. Wang, H. Li, and X. Yang, "Neural-network-based online HJB solution for optimal robust guaranteed cost control of continuous-time uncertain nonlinear systems," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2834–2847, Dec. 2014.

[8] D. Wang, D. Liu, and H. Li, "Policy iteration algorithm for online design of robust control of a class of continuous-time nonlinear systems," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 627–632, Apr. 2014.

[9] D. Wang, D. Liu, H. Li, and H. Ma, "Neural-network-based robust optimal control design for a class of uncertain nonlinear systems via adaptive dynamic programming," *Information Sciences*, vol. 282, pp. 167–179, Oct. 2014.

[10] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement Learning: State-of-the-Art (Adaptation, Learning, and Optimization)*, M. Wiering and M. van Otterlo, Eds. New York, USA: Springer-Verlag, 2012, ch. 2.

[11] M. Geist and O. Pietquin, "Algorithmic survey of parametric value function approximation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 6, pp. 845–867, June 2013.

[12] D. P. Bertsekas, "Approximate policy iteration: a survey and some new methods," *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310–335, 2011.

[13] 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.

[14] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, Dec. 2003.

[15] S. Mahadevan and M. Maggioni, "Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes," *Journal of Machine Learning Research*, vol. 8, no. 10, pp. 2169–2231, 2007.

[16] S. Mahadevan, "Learning representation and control in Markov decision processes: new frontiers," *Foundations and Trends in Machine Learning*, vol. 1, no. 4, pp. 403–565, 2008.

[17] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[18] X. Xu, Z. Huang, D. Graves, and W. Pedrycz, "A clustering-based graph Laplacian framework for value function approximation in reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2613–2625, Dec. 2014.

[19] Z. Huang, X. Xu, and L. Zuo, "Reinforcement learning with automatic basis construction based on isometric feature mapping," *Information Sciences*, vol. 286, pp. 209–227, 2014.

[20] J. Tenenbaum, V. de Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[21] M. Belkin, P. Niyogi, V. Sindhwani, "Manifold regularization: a geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434, 2006.

[22] P. Niyogi, "Manifold regularization and semi-supervised learning: Some theoretical analyses," *Journal of Machine Learning Research*, vol. 14, pp. 1229–1250, 2013.

[23] M. W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos, "Regularized least squares temporal difference learning with nested ℓ_2 and ℓ_1 penalization," in *Proceedings of the 9th European Workshop on Reinforcment Learning*, 2011, pp. 1–12.

[24] G. Huang, S. Song, J. Gupta, and C. Wu, "Semi-supervised and unsupervised extreme learning machines," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2405–2417, Dec. 2014.