Learning Representation and Control In Continuous Markov Decision Processes

Sridhar Mahadevan *
Department of Computer Science
University of Massachusetts
140 Governor's Drive
Amherst, MA 01003
mahadeva@cs.umass.edu

Mauro Maggioni
Program in Applied Mathematics
Department of Mathematics
Yale University
New Haven,CT,06510
mauro.maggioni@yale.edu

Kimberly Ferguson, Sarah Osentoski
Department of Computer Science
University of Massachusetts
140 Governor's Drive
Amherst, MA 01003
(kferguso,sosentos)@cs.umass.edu

Abstract

This paper presents a novel framework for simultaneously learning representation and control in continuous Markov decision processes. Our approach builds on the framework of proto-value functions, in which the underlying representation or basis functions are automatically derived from a spectral analysis of the state space manifold. The proto-value functions correspond to the eigenfunctions of the graph Laplacian. We describe an approach to extend the eigenfunctions to novel states using the Nystrom extension. A least-squares policy iteration method is used to learn the control policy, where the underlying subspace for approximating the value function is spanned by the learned proto-value functions. A detailed set of experiments is presented using classic benchmark tasks, including the inverted pendulum and the mountain car, showing the sensitivity in performance to various parameters, and including comparisons with a parametric radial basis function method.

Introduction

This paper describes a novel framework for learning control tasks in continuous state spaces, building on the framework of proto-value functions (Mahadevan 2005). Unlike traditional approaches based on parametric function approximation (Bertsekas & Tsitsiklis 1996), where the basis functions are usually designed by a human, proto-value functions are task-independent basis functions whose structure is automatically learned from samples of the underlying state space manifold. Consequently, their structure captures large-scale geometric invariants, including bottlenecks and symmetries. Proto-value functions can be learned in a variety of ways. "Off-policy" Fourier proto-value functions are formed from spectral analysis of the graph Laplacian, whose eigenfunctions form a complete orthonormal basis. In contrast, "on-policy" proto-value functions are learned by multiscale wavelet decomposition of the transition matrix of a policy directly (Maggioni & Mahadevan 2006).

Previous work on proto-value functions was restricted to discrete state spaces. The main contribution of this paper is a detailed study of proto-value functions in continuous state spaces, which present significant challenges not encountered in discrete state spaces. The eigenfunctions can only be computed and stored on sampled real-valued states, and hence must be interpolated to novel states. We apply the Nyström interpolation method. While this approach has been studied previously in kernel methods (Williams & Seeger 2000) and spectral clustering (Belongie *et al.* 2002), our paper presents the first detailed study of the Nyström method for learning control, as well as a detailed comparison of graph normalization methods.

*This research was supported in part by the National Science Foundation under grants IIS-0534999 and DMS-051050.

Markov Decision Processes

A continuous state Markov decision process (MDP) $M = \langle S, A, P^a_{ss'}, R^a_{ss'} \rangle$ is defined by a set of states $S \subset \mathbb{R}^d$, a set of discrete actions A, a transition model $P^a_{ss'}$ specifying the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R^a_{ss'}$ specifying a scalar cost or reward. Usually, continuous control tasks are specified by some underlying *controller* or *plant* $s_{t+1} = f(s_t, a_t, \sigma_t)$ which specifies a functional mapping of a state s_t into a new state s_{t+1} in response to the control or action selected a_t and some (parametrically modeled) noise term σ_t. In our paper, we do not assume either the continuous control system or the noise model is known. A value function is a mapping $S \rightarrow \mathbb{R}$; in discrete state spaces, it can be viewed as a vector $\in \mathbb{R}^{|S|}$. Given a policy $\pi : S \rightarrow A$ mapping states to actions, its corresponding value function V^π specifies the expected long-term discounted sum of rewards received by the agent in a state s when actions are chosen using the policy. Any optimal policy π^* defines the same unique optimal action value function $Q^*(s, a)$ which satisfies the nonlinear constraints

$$Q^*(s,a) = \int_{s'} P^a_{ss'} \left(R^a_{ss'} + \max_{a'} \gamma Q^*(s', a') \right) ds'$$

A more detailed treatment can be found in standard treatises on MDPs (Puterman 1994).

Representation Policy Iteration

Representation policy iteration (RPI) is a novel algorithm for approximate policy iteration (Mahadevan 2005), where the subspace for projecting the value function is constructed from the spectral analysis of a graph. RPI can be viewed as an enhancement of least-squares policy iteration (LSPI) to include a basis learning step (Lagoudakis & Parr 2003). The graph is constructed from a random walk of transitions generated from the current policy. RPI approximates the true action-value function $Q^\pi(s, a)$ for a policy π using a set of basis functions $\phi(s, a)$ computed from a graph formed from an initial random walk:

$$\hat{Q}^\pi(s, a; w) = \sum_{j=1}^{k} \phi_j(s, a) w_j$$

where the w_j are weights or parameters that can be determined using a least-squares method. Given a data set $\mathcal{D}$ of states $\in \mathbb{R}^d$ visited during a random walk, where $k = |\mathcal{D}|$, let Q^π be a real (column) vector of size $k \times |A|$. The column vector $\phi(s, a)$ is a real vector of size k where the j^{th} row corresponds to the basis function $\phi_j(s, a)$ evaluated at the state action pair (s, a), where $s \in \mathcal{D}$. The approximate action-value function can be written as $\hat{Q}^\pi = \Phi w^\pi$, where w^π is a real column vector of length k and Φ is a real matrix with $k \times |A|$ rows and k columns. We use the Nystrom extension to compute the value function across the entire state space $S \subset \mathbb{R}^d$. RPI solves a fixed-point approximation $T_\pi Q^\pi \approx Q^\pi$, where T_π is the Bellman backup operator, yielding the following solution for the coefficients:

$$w^\pi = \left(\Phi^T \left(\Phi - \gamma P \Pi_\pi \Phi \right) \right)^{-1} \Phi^T R$$

where Π_π is a $|S| \times |S||A|$ matrix defining a stochastic policy $\Pi_{pi}(s, (s, a)) = \pi(a|s)$. Since the transition dynamics are unknown, a sampling technique is used to learn the state-action value function $\hat{Q}^\pi$.

$$\begin{aligned} \tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t, a_t) \left(\phi(s_t, a_t) - \gamma \phi(s'_t, \pi(s'_t)) \right)^T \\ \tilde{b}^{t+1} &= \tilde{b}^t + \phi(s_t, a_t) r_t \end{aligned}$$

(s_t, a_t, r_t, s'_t) is the t^{th} sample of experience from a trajectory generated by the agent (using some random or guided policy). RPI, like LSPI, solves the linear system of equations $\tilde{A}\tilde{w}^\pi = \tilde{b}$ to find the coefficients $\tilde{w}^\pi$.

Spectral Analysis of Graphs

In this paper, proto-value functions are Fourier global bases computed "off-policy" through a spectral analysis of a random walk on the graph, in particular by a spectral analysis of the graph Laplacian on samples $\in \mathbb{R}^d$ representing states traversed during a random walk. Let $G = (V, E, W)$ denote a weighted undirected graph with vertices V, edges E and weights w_{ij} on edge $(i, j) \in E$. The degree of a vertex v is denoted as d_v. The adjacency matrix A can be viewed as a binary weight matrix W. The graph Laplacian is a discrete version of the Laplacian on a Riemannian manifold. A well-known classical result called *Hodge's* theorem states that the eigenfunctions of the Laplacian provide a complete basis for functions on a Riemannian manifold (Rosenberg 1997). Let D be the valency matrix, in other words a diagonal matrix whose entries are the row sums of W. The *normalized Laplacian* $\mathcal{L}$ of the graph G is defined as $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$, i.e.

$$\mathcal{L}(u,v) = \begin{cases} 1 - \frac{w_{vv}}{d_v} & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{w_{uv}}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{L}$ is a symmetric self-adjoint operator, its spectrum (eigenvalues) lie in the interval $\lambda \in [0, 2]$. The random walk operator on a graph, given by $D^{-1}A$ is not symmetric, but is conjugate to $I - \mathcal{L}$, and hence its spectrum is $\{1 - \lambda(\mathcal{L})\}$. The eigenvectors of the random walk operator are the eigenvectors of $I - \mathcal{L}$ scaled by $D^{-\frac{1}{2}}$. The eigenvectors $\{\phi_i\}$ of $\mathcal{L}$ have different degrees of smoothness as measured by the quadratic form $\langle \mathcal{L}\phi_i, \phi_i \rangle = \lambda_i$: the larger λ_i the more oscillatory and less smooth ϕ_i is.

Constructing Graphs from Point Sets in $\mathbb{R}^d$

Given a data set $\{x_i\}$ in $\mathbb{R}^d$, we can associate different weighted graphs to this point set. There are different choices of edges and for any such choice there is a choice of weights on the edges. For example, edges can be inserted between a pair of points x_i and x_j if:

(E1) $||x_i - x_j||_{\mathbb{R}^d} \leq \delta$, where $\delta > 0$ is a parameter;

(E2) x_j is among the k nearest neighbors of x_i, where $k > 0$ is a parameter.

Weights can be assigned to the edges in any of the following ways:

(W1) all edges have the same weight (say, 1);

(W2) $W(i,j) = \alpha(i) e^{-\frac{||x_i \;\; x_j||^2_{\mathbb{R}^d}}{\sigma}}$, where $\sigma > 0$ is a parameter, and α a weight function to be specified;

(W3) $W(i,j) = \alpha(i) e^{-\frac{||x_i \;\; x_j||^2_{\mathbb{R}^d}}{||x_i \;\; x_{k(i)}|| \, ||x_j \;\; x_{k(j)}||}}$ where $k > 0$ is a parameter and $x_{k(i)}$ is the k-th nearest neighbor of x_i (this is called self-tuning Laplacian (L. Zelnik-Manor 2004). Again α a weight function to be specified.

Observe that in case (E2) the graph is in general not undirected, since x_j can be among the K nearest neighbors of x_i but x_i may not be among the K nearest neighbors of x_j. Since here we consider undirected graphs, in such cases we replace the weight matrix W constructed so far by the symmetric $W + W^T$, WW^T or W^TW. If the points $\{x_i\}$ are drawn uniformly from a Riemannian manifold, then it is shown in (Belkin & Niyogi 2004) that (E1)+(W2), with $\alpha = 1$, approximates the continuous Laplace-Beltrami operator on the underlying manifold. If $\{x_i\}$ is not drawn uniformly from the manifold, as it typically happens here when the space is explored by an agent, it is shown in (Lafon 2004) that a pre-processing normalization step can (must) be performed that yields the weight function α, so that (E1)+(W2) yields an approximation to the Laplace-Beltrami operator. In the experiments below, we use the following terminology:

1. 'Ave' means that the eigenvectors of $D^{-1}(D - W)$ are computed. This applies to any combination of (E1,2)-(W1-3)

2. 'GraphMarkov' means that the eigenvectors of the normalized graph Laplacian are computed, with any combination of (E1,2)-(W1-3)
3. 'Beltrami' applies to any combination of (E1,2)-(W1-3), however the only theorem known about approximation to the continuous Laplace-Beltrami is for the combination (E1) (with large δ) together with (W2).

Extension: Nyström, fast updates and (randomized) low-rank approximations

To learn policies on continuous MDPs, it is necessary to be able to extend eigenfunctions computed on a set of states $\in \mathbb{R}^d$ to new states. We describe here the Nyström method, which can be combined with iterative updates and randomized algorithms for low-rank approximations. The Nyström method interpolates the value of eigenvectors computed on sample states to novel states, and is an application of a classical method used in the numerical solution of integral equations (Baker 1977). It can be viewed as a technique for approximating a semi-positive definite matrix from a low-rank approximation. In this context it can be related to randomized algorithms for low-rank approximation of large matrices (Frieze, Kannan, & Vempala 1998). Let us review the Nyström method in its basic form. Suppose we have a positive semi-definite operator K, with rows and columns indexed by some measure space $(\mathbb{X}, \mu)$. K acts on a vector space of functions on X by the formula

$$Kf(x) = \int_{\mathbb{X}} K(x,y) f(y) d\mu(y)\,,$$

for f in some function space on $\mathbb{X}$. Examples include:

(i) $\mathbb{X} = \mathbb{R}$, μ is the Lebesgue measure, and $K_\sigma(x,y) = e^{-\frac{|x\ \ y|^2}{\sigma}}$, and K acts on square integral functions f on $\mathbb{R}$ by $K_\sigma f(x) = \int_{-\infty}^{+\infty} e^{-\frac{|x\ \ y|^2}{\sigma}} f(y) dy = K_\sigma * f$.

(ii) $\mathbb{X}$ is a compact Riemannian manifold $(\mathcal{M}, \rho)$ equipped with the measure corresponding to the Riemannian volume, Δ is the Laplace-Beltrami operator on $\mathcal{M}$, with Dirichlet or Neumann boundary conditions if $\mathcal{M}$ has a boundary, and $K = (I - \Delta)^{-1}$ is the Green's function or potential operator associated with Δ.

Since K is positive semi-definite, by the spectral theorem it has a square root F, i.e. $K = F^T F$. Sometimes this property is expressed by saying that K is a Gram matrix, since we can interpret $K(x,y)$ as the inner product between the x-th and y-th columns of F. In applications one approximates operators on uncountable spaces (such as $\mathbb{R}$ or a manifold $\mathcal{M}$ as in the examples above) by a finite discretization $x_1, \ldots, x_n$, in which case $\mathbb{X} = \{0, \ldots, n\}$, the measure μ is an appropriate set of weights on the n points, and K is a $n \times n$ matrix acting on n-dimensional vectors. To simplify the notation we use this discrete setting in what follows.

The Nyström approximation starts with a choice of a partition of the columns of F into two subsets F_1 and F_2. Let k be the cardinality of F_1, so that F_1 can be represented as $n \times k$ matrix and F_2 as a $n \times (n-k)$ matrix. One can then write

$$K = \begin{pmatrix} F_1^T F_1 & F_1^T F_2 \\ F_2^T F_1 & F_2^T F_2 \end{pmatrix}$$

The Nyström method consists of the approximation

$$F_2^T F_2 \sim (F_1^T F_2)^T (F_1^T F_1)^{-1} (F_1^T F_2)\,. \qquad (1)$$

The quantity on the righthand side requires only the knowledge of $(F_1^T F_2)$ and $F_1^T F_1$, i.e. the first k rows (or columns) of K. Moreover if the matrix K has rank k and F_1 spans the range of K, then the Nyström approximation is in fact exactly equal to $F_2^T F_2$.

The natural question that arises is of course how to choose F_1 in these situations. Various heuristics exists, and mixed results have been obtained (Platt 2004). The most desirable choice of F_1, when the error of approximation is measured by $||F_2^T F_2 - (F_1^T F_2)^T (F_1^T F_1)^{-1} (F_1^T F_2)||_2$ (or, equivalently, the Fröbenius norm) would be to pick F_1 such that its span is as close as possible to the span of the top k singular vectors of K. Several numerical algorithms exist, which in general require $\mathcal{O}(kN^2)$ computations. One can use randomized algorithms, which pick rows (or columns) of K accordingly to some probability distribution (e.g. dependent on the norm of the row or column). There are guarantees that these algorithms will select with high probability a set of rows whose span is close to that of the top singular vectors (Drineas & Mahoney 2005).

The Nyström method is applied to the approximation of the eigenfunctions of the graph Laplacian $\mathcal{L}\phi_i = \lambda_i \phi_i$ by letting F_1 be the matrix with k eigenfunctions as columns: equation (1) yields

$$\phi_i(x) = \frac{1}{1-\lambda_i} \sum_{y \sim x} \frac{w(x,y)}{\sqrt{d(x)d(y)}} \phi_i(y)\,, \qquad (2)$$

where $d(z) = \sum_{y \sim z} w(z,y)$, and x is a new vertex in the graph. The Nyström method can be refined with fast iterative updates as follows: first compute an extension of the eigenvectors to new points (states), to obtain approximated eigenvectors of the extended graph $\{\tilde{\phi}_i\}$. Input these eigenvectors into an iterative eigensolver as initial approximate eigenvectors: after very few iterations the eigensolver will refine these initial approximate eigenvectors into more precise eigenvectors on the larger graph. The extra cost of this computation is $\mathcal{O}(IN)$ if I iterations are necessary, and if the adjacency matrix of the extended graph is sparse (only $\mathcal{O}(N)$ non-zero entries).

Algorithmic Details and Experimental Results

This section presents detailed experimental results of the RPI algorithm illustrated in Figure 1 on the inverted pendulum and the mountain car benchmark problems. The experiments measure the variance in performance with respect to various parameters, such as the local distance metric used, and also compare the performance of RPI with using hand-coded function approximators, such as radial basis functions.

`RPI Algorithm` $(T, N, Z, \epsilon, k, P, \mathcal{O})$:

// T: Number of initial random walk trials
// N: Maximum length of each trial
// Z: Size of the subsampled data to build the graph from
// ϵ : Convergence condition for policy iteration
// k: Number of nearest neighbors
// P: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used

1. **Representation Learning Phase**: Perform a random walk of T trials, each of maximum N steps, and store the states visited in the dataset $\mathcal{D}$. Construct a smaller dataset $\mathcal{D}_Z$ of size Z from the collected data $\mathcal{D}$ by random subsampling.
2. Build an undirected weighted graph G from $\mathcal{D}_Z$, in one of the ways described in the section on the construction of graphs from point sets. Construct one of the operators $\mathcal{O}$ on graph G as discussed in the section on graph construction.
3. Compute the k smoothest eigenvectors of $\mathcal{O}$ on the subsampled graph $\mathcal{D}_Z$, and collect them as columns of the basis function matrix Φ, a $|\mathcal{D}_Z| \times k$ matrix. The embedding of a state action pair $\phi(s,a)$ where $s \in \mathcal{D}_Z$ is given as $e_a \otimes \phi(s)$, where e_a is the unit vector corresponding to action a, $\phi(s)$ is the s^{th} row of Φ, and $\otimes$ is the tensor product.
4. **Control Learning Phase**: Initialize $w^0 \in \mathcal{R}^k$ to a random vector. **Repeat** the following steps
   (a) Set $i \leftarrow i+1$. For each transition $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}$, compute low rank approximations of matrix A and b as follows:
   $$\begin{aligned} \tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t,a_t)\left(\phi(s_t,a_t) - \gamma\phi(s'_t,a'_t)\right)\big)^T \\ \tilde{b}^{t+1} &= \tilde{b}^t + \phi(s_t,a_t) r_t \end{aligned}$$
   where $\phi(s_t,a_t)$ is approximated using the Nystrom extension whenever $s_t \notin \mathcal{D}_Z$.
   (b) Solve the system $\tilde{A}w^i = \tilde{b}$
5. **until** $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
6. Return $\hat{Q}^\pi = \sum_i w^i \Phi$ as the approximation to the optimal value function.

end

Figure 1: Pseudo-code of the representation policy iteration algorithm for continuous MDPs.

The Inverted Pendulum

The inverted pendulum problem requires balancing a pendulum of unknown mass and length by applying force to the cart which it is attached to. We used the implementation described in (Lagoudakis & Parr 2003). The state space is defined by two variables: θ, the vertical angle of the pendulum, and $\dot{\theta}$, the angular velocity of the pendulum. The three actions are applying a force of -50, 0, or 50 Newtons. Uniform noise from -10 and 10 is added to the chosen action. State transitions are defined by the nonlinear dynamics of the system, and depend upon the current state and the noisy control signal, u.

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

where g is gravity, 9.8 m/s^2, m is the mass of the pendulum, 2.0 kg, M is the mass of the cart, 8.0 kg, l is the length of the pendulum, .5 m, and $\alpha = 1/(m+M)$. The simulation time step is set to 0.1 seconds. The agent is given a reward of 0 as long as the absolute value of the angle of the pendulum does not exceed $\pi/2$. If the angle is greater than this value the episode ends with a reward of -1. The discount factor was set to 0.95. The maximum number of episodes the pendulum was allowed to balance was fixed at 3000 steps. Each learned policy was evaluated 10 times.

Mountain Car

The goal of the *mountain car* task is to get a simulated car to the top of a hill as quickly as possible (Sutton & Barto 1998). The car does not have enough power to get there immediately, and so must oscillate on the hill to build up the necessary momentum. This is a minimum time problem, and thus the reward is -1 per step. The state space includes the position and velocity of the car. There are three actions: full throttle forward (+1), full throttle reverse (-1), and zero throttle (0). It's position, x_t and velocity $\dot{x}_t$, are updated by

$$x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}] \qquad (3)$$

$$\dot{x}_{t+1} = \text{bound}[\dot{x}_t + 0.001a_t + -0.0025cos(3x_t)], \qquad (4)$$

where the bound operation enforces $-1.2 \leq x_{t+1} \leq 0.6$ and $-0.07 \leq \dot{x}_{t+1} \leq 0.07$. The episode ends when the car successfully reaches the top of the mountain, defined as position $x_t >= 0.5$. In our experiments we allow a maximum of 500 steps, after which the task is terminated without success. The discount factor was set to 0.99.

Experimental Settings

Table 1 summarizes the range of parameters over which the RPI algorithm was tested in the two domains, as well as the settings for the radial basis function experiments. Values in boldface are the defaults for experiments in which those parameters were not being varied. The results for the following experiments were averaged over 20 runs. The RBFs for both domains were evenly spaced within the range of each axis. The Number of RBFs as seen in Table 2 as well as Figure 6 includes plus one for a constant.

For the mountain car task, lower numbers indicate better performance, since we are measuring the steps to reach the top of the hill. In the inverted pendulum, however, since we are measuring the number of steps that the pole remained upright, higher numbers indicate better performance. Figure 2 displays the first set of experiments, which varied the number of random samples $\mathcal{D}_Z$ over which proto-value functions were computed. In the mountain car task, performance monotonically improves as the number of random samples is increased, upto a maximum of 1500 states. However, interestingly, in the pendulum task, a sample size of 500 produced the best results, and performance appears to degrade for larger sizes.

Parameter	Inverted Pendulum	Mountain Car
T	(100 to 600)	(50 to 550)
N	500	90
Z	$\{100, \mathbf{500}, 1000\}$	$\{200, \mathbf{500}, 800, 1500\}$
ϵ	10^{-5}	10^{-3}
k	$\{25, \mathbf{50}, 100\}$	$\{15, \mathbf{25}, 40, \text{Gaussian}\}$
P	$\{25, 30, \mathbf{50}, 100, 150\}$	$\{50, \mathbf{100}, 200\}$
δ	0.95	1
$\mathcal{O}$	varied	graph markov

Table 1: Parameter values (as defined in Figure 1) for inverted pendulum and mountain car experiments. Default values are in bold.

Number of RBFs	Inverted Pendulum RBF Parameters
10	3 x-axis, 3 y-axis, sigma 1
37	6 x-axis, 6 y-axis, sigma 0.7
50	7 x-axis, 7 y-axis, sigma 0.3
Number of RBFs	**Mountain Car RBF Parameters**
10	3 x-axis, 3 y-axis, sigma 1
37	6 x-axis, 6 y-axis, sigma 1

Table 2: RBF parameter settings for inverted pendulum and mountain car experiments.

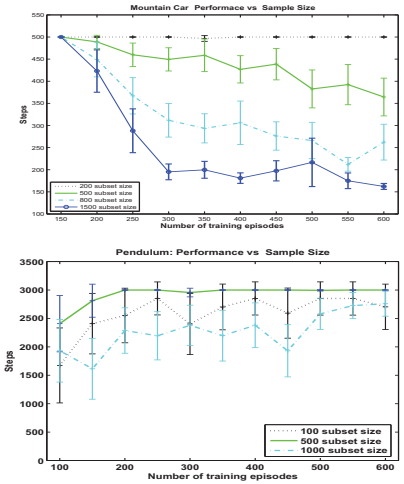

Figure 2: Performance on mountain car and inverted pendulum as a function of the number of subsampled points on which proto-value functions were computed.

In the second experiment, illustrated in Figure 3, the effect of varying the local distance metric was evaluated. In the mountain car domain, the nearest neighbor metric outperformed the gaussian distance metric. Also, using a lower number of nearest neighbors improved the performance. However, in the inverted pendulum task, performance improved as the number of nearest neighbors was increased, upto 50, after which performance seemed to degrade.

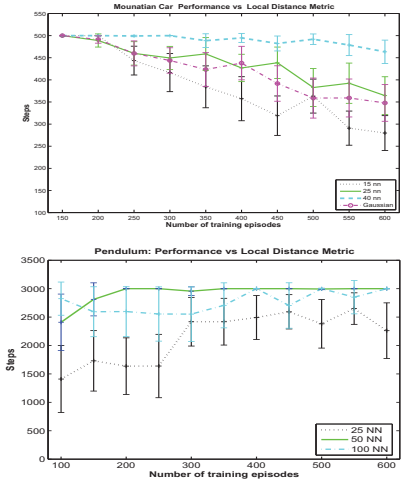

Figure 3: Performance on mountain car and inverted pendulum as a function of the nearest neighbors.

Figure 4 varied the number of protovalue functions (PVFs) used. Here, there were significant differences in the two tasks. In the inverted pendulum task, performance dramatically improved from 25 to 50 proto-value functions, whereas in the mountain car domain, performance differences were less acute (note that the mountain car results used only 500 samples, which results in worse performance than using 1500 samples, as shown earlier in Figure 2).

Figure 5 measures the effect of varying the type of graph normalization in the pendulum task, which seems to cause little difference in the results. The same behavior was observed for the mountain car task (not shown).

Figure 6 compares the performance of proto-value functions with radial basis functions (RBFs). These comparative experiments are to be viewed with caution: only a limited range of the possible set of parameter values were explored. In the mountain car task, proto-value functions (bottom-most curve) seem to outperform the choices of RBFs that are listed in Table 2. For the inverted pendulum, the performance of proto-value functions (top curve) appears significantly better than RBFs as well.

Conclusions

This paper presented a framework for learning both representation and control to solve continuous Markov decision processes. Representations are learned by computing the eigenfunctions of the graph Laplacian formed from an initial random walk of the state space. The eigenfunctions are interpolated to novel states using the Nyström extension. A least-squares policy iteration method was used to learn the control policy. Detailed results from two classic domains showed the feasibility of the approach. We are exploring a variety of extensions to scale this approach to higher-dimensional control tasks.

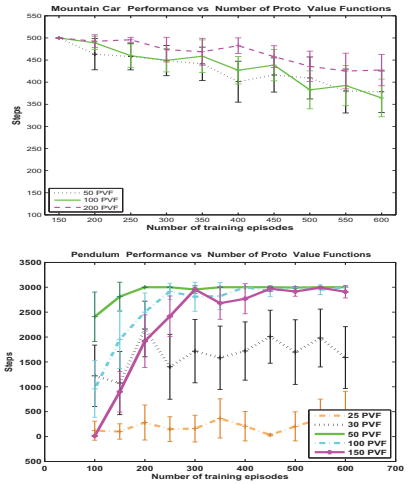

Figure 4: Performance on mountain car and inverted pendulum as a function of the number of proto-value functions.

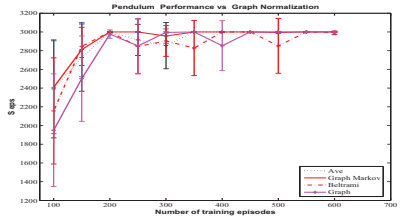

Figure 5: Results from the pendulum task, showing the variation in performance as a function of the type of graph normalization. Similar results were seen in the mountain car domain (not shown).

References

Baker, C. T. H. 1977. *The numerical treatment of integral equations*. Oxford: Clarendon Press.

Belkin, M., and Niyogi, P. 2004. Semi-supervised learning on Riemannian manifolds. *Machine Learning* 56:209–239.

Belongie, S.; Fowlkes, C.; Chung, F.; and Malik, J. 2002. Spectral partitioning with indefinite kernels using the Nystrom extension. *ECCV*.

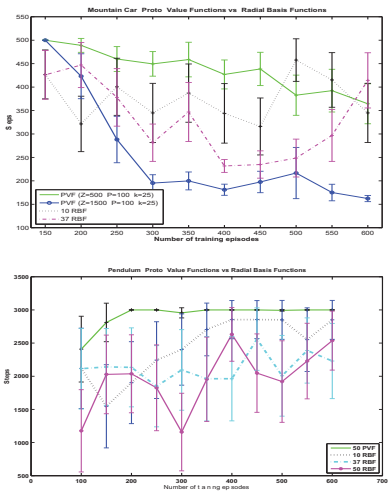

Figure 6: A comparison of RBFs and PVFs on the mountain car and inverted pendulum.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Belmont, Massachusetts: Athena Scientific.

Drineas, P., and Mahoney, M. W. 2005. On the Nystrom method for approximating a Gram matrix for improved kernel-based learning. *J. Machine Learning Research* (6):2153–2175.

Frieze, A.; Kannan, R.; and Vempala, S. 1998. Fast Monte Carlo algorithms for finding low-rank approximations. In *Proceedings of the 39th annual IEEE symposium on foundations of computer science*, 370–378.

L. Zelnik-Manor, P. P. 2004. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*.

Lafon, S. 2004. *Diffusion maps and geometric harmonics*. Ph.D. Dissertation, Yale University, Dept of Mathematics & Applied Mathematics.

Lagoudakis, M., and Parr, R. 2003. Least-squares policy iteration. *Journal of Machine Learning Research* 4:1107–1149.

Maggioni, M., and Mahadevan, S. 2006. Fast direct policy evaluation using multiscale analysis of markov diffusion processes. In *International Conference on Machine Learning (ICML)*.

Mahadevan, S. 2005. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the* 22^{nd} *International Conference on Machine Learning*.

Platt, J. C. 2004. Fastmap, metricmap, and landmark mds are all nystrom algorithms. Technical Report MSR-TR-2004-26, Microsoft Research.

Puterman, M. L. 1994. *Markov decision processes*. New York, USA: Wiley Interscience.

Rosenberg, S. 1997. *The Laplacian on a Riemannian Manifold*. Cambridge University Press.

Sutton, R., and Barto, A. G. 1998. *An Introduction to Reinforcement Learning*. MIT Press.

Williams, C. K. I., and Seeger, M. 2000. Using the nystrom method to speed up kernel machines. In *NIPS*, 682–688.