Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Kernel-Based Reinforcement Learning on Representative States

Branislav Kveton
Technicolor Labs
Palo Alto, CA
branislav.kveton@technicolor.com

Georgios Theocharous
Yahoo Labs
Santa Clara, CA
theochar@yahoo-inc.com

Abstract

Markov decision processes (MDPs) are an established framework for solving sequential decision-making problems under uncertainty. In this work, we propose a new method for batch-mode reinforcement learning (RL) with continuous state variables. The method is an approximation to kernel-based RL on a set of k representative states. Similarly to kernel-based RL, our solution is a fixed point of a kernelized Bellman operator and can approximate the optimal solution to an arbitrary level of granularity. Unlike kernel-based RL, our method is fast. In particular, our policies can be computed in $O(n)$ time, where n is the number of training examples. The time complexity of kernel-based RL is $\Omega(n^2)$. We introduce our method, analyze its convergence, and compare it to existing work. The method is evaluated on two existing control problems with 2 to 4 continuous variables and a new problem with 64 variables. In all cases, we outperform state-of-the-art results and offer simpler solutions.

Introduction

Markov decision processes (MDPs) (Puterman 1994) are an established framework for sequential decision making under uncertainty. If a decision problem is Markovian, has a small number of states, and its model is known, it can be typically easily solved as an MDP. However, in practice, a good model is often unavailable and the state of the problem is described by continuous variables. This class of problems is inherently hard to solve and has been subject to active research over the past 30 years (Sutton and Barto 1998).

In this paper, we propose a novel approach to batch-mode reinforcement learning (RL) with continuous state variables. The approach does not need the model of the MDP or a parametric approximation of its value function to solve the problem. Our solution consists of two main steps. First, we cover the state space of the problem using cover trees and discover k representative states. Second, we summarize the dynamics of the problem in these states and solve it by policy iteration. Our solution is intuitive, easily implementable, and has only one tunable parameter, the number of representative states k. In the experimental section, we show that our method learns better policies than state-of-the-art algorithms and is capable of solving high-dimensional MDPs.

Our work addresses a major challenge in RL, learning of a good representation of a problem. We represent the problem as statistics in representative states and extrapolate to unseen data using kernels (Ormoneit and Sen 2002). In comparison to kernel-based RL, our method is computationally efficient. In particular, our solutions are computed in only $O(n)$ time, where n denotes the number of training examples. The time complexity of kernel-based RL is $\Omega(n^2)$. Since our method is a kernel-based approximation, it has many favorable properties. For instance, our solution is a fixed point of a kernelized Bellman operator. Moreover, it converges to the optimal value function as the complexity of the approximation k and the sample size n increase.

We make the following three assumptions. First, all problems can be simulated. The *simulator* of the problem outputs a sequence of 4-tuples $\{(\mathbf{x}_t, a_t, r_t, \mathbf{x}'_t)\}_{t=1}^n$, where $\mathbf{x}_t$, a_t, r_t, and $\mathbf{x}'_t$ refer to the state, action, reward, and next state in the sample t, respectively. Second, the state space is metric and $d(\cdot, \cdot)$ is the corresponding distance function. For simplicity, we assume that the function is given by $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2$ but note that our results generalize to any metric. Finally, the state space is normalized such that $d(\mathbf{x}, \mathbf{x}') \leq 1$ for all $\mathbf{x}$ and $\mathbf{x}'$.

Markov decision processes

A *Markov decision process (MDP)* (Bellman 1957) is given by a 4-tuple $\mathcal{M} = (S, \mathcal{A}, P, R)$, where S is a set of states, $\mathcal{A}$ is a set of actions, $P(s' \mid s, a)$ is a transition function, which describes the dynamics of the MDP, and $R(s, a)$ is a reward model, which assigns rewards to state-action configurations. The quality of MDP policies π is typically measured by their *infinite horizon discounted reward* $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where γ is a *discount factor* and r_t is the immediate reward at time t. In this setting, the *optimal policy* is stationary and deterministic (Puterman 1994), and can be defined greedily with respect to the *optimal value function* V^*, which is the fixed point of the Bellman equation (Bellman 1957):

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s') \right]. \quad (1)$$

The optimal value function can be found by policy iteration, value iteration, or linear programming (Puterman 1994).

This paper focuses on continuous-state MDPs. Formally, a *continuous-state MDP* is a tuple $\mathcal{M} = (\mathbf{X}, \mathcal{A}, P, R)$, where $\mathbf{X} = X_1 \times \cdots \times X_m$ is a state space, which is factored into m continuous variables, $\mathcal{A}$ is a set of actions, $P(\mathbf{x}' \mid \mathbf{x}, a)$ is a transition model, and $R(\mathbf{x}, a)$ is a reward function. Similarly to the discrete-state model, the optimal value function V^* is

the fixed point of the Bellman equation:

$$V^*(\mathbf{x}) = \max_a \left[R(\mathbf{x}, a) + \gamma \mathbb{E}_{P(\mathbf{x}'|\mathbf{x},a)}[V^*(\mathbf{x}')]\right]. \quad (2)$$

In general, the computation of the exact solution V^* is hard because it may not have a finite support.

Related work

State space discretization and value function approximations are two main approaches to solving continuous-state MDPs. Both techniques have been studied and analyzed extensively. For example, Chow and Tsitsiklis (1991) analyze discretization of the state space on a uniform grid, Bertsekas and Tsitsiklis (1996) propose several methods for fitting linear value function approximations, Munos and Moore (1999) use KD-trees to discretize the state space at multiple levels of granularity, and Kveton *et al.* (2006) show how to learn the linear value function using linear programming. All of these methods rely on the model of the MDP and cannot be used when the model is unknown.

When the model is unavailable, the optimal value function can be still approximated using reinforcement learning (RL). Sutton and Barto (1998) provide a comprehensive overview of existing RL algorithms. The most relevant to this work is *kernel-based RL* (Ormoneit and Sen 2002). In kernel-based RL, the Bellman operator is approximated by an operator on the sample $\{(\mathbf{x}_t, a_t, r_t, \mathbf{x}'_t)\}_{t=1}^n$:

$$\mathcal{T}_\lambda V(\mathbf{x}) = \max_a \sum_{t \in \tau^a} \lambda^a_{\mathbf{x}_t \mathbf{x}} \left[r_t + \gamma V(\mathbf{x}'_t)\right], \quad (3)$$

where τ^a is a subset of indices t where $a_t = a$, and $\lambda^a_{\mathbf{x}_t \mathbf{x}}$ is a normalized kernel such that $\sum_{t \in \tau^a} \lambda^a_{\mathbf{x}_t \mathbf{x}} = 1$ for every state $\mathbf{x}$ and action a. Because of the normalization, it is helpful to look at the kernel as a function of $\mathbf{x}_t$, which is parameterized by $\mathbf{x}$ and a. Moreover, the backup operator $\mathcal{T}_\lambda$ can be viewed as being defined by a convex combination of transitions and rewards from the underlying model.

Kernel-based reinforcement learning has many nice properties (Ormoneit and Sen 2002). First, the operator $\mathcal{T}_\lambda$ has a unique fixed point. Second, the fixed point converges to the optimal value function for the *Gaussian kernel*:

$$\lambda^a_{\mathbf{x}_t \mathbf{x}} \propto \exp\left[-\frac{d^2(\mathbf{x}_t, \mathbf{x})}{2\sigma^2}\right] \quad (4)$$

when $n \to \infty$ and $\sigma \to 0$. Finally, note that the operator $\mathcal{T}_\lambda$ (Equation 3) depends on the value function V in n states $\mathbf{x}'_t$. Therefore, the backup of V by $\mathcal{T}_\lambda$ can be computed in $\theta(n^2)$ time because V needs to be updated in only n states and the cost of each update is $\theta(n)$.

The time complexity of kernel-based RL is $\Omega(n^2)$. Therefore, the method is impractical when the sample size is large. To make it practical, Jong and Stone (2006) proposed several speedups, such as sampling of states through exploration and prioritized sweeping. Fitted Q iteration (FQI) (Ernst, Geurts, and Wehenkel 2005) is the first practical and general method for RL with kernels. The method is a variation of Q iteration, where the exact Q function is replaced by its approximation. The approximation is fit by a non-parametric regressor, such as a classification and regression tree (CART). Naturally, the quality of FQI policies and their computation time depend a lot on the regressor. In the experimental section, we compare our policies to fitted Q iteration with CART. This is the best performing regressor that can be learned in time comparable to our method (Ernst, Geurts, and Wehenkel 2005).

Barreto *et al.* (2011) recently proposed factorization of the kernel matrix in RL. The resulting algorithm is similar to our approach because the factorization is done on representative states. Note that Barreto *et al.* (2011) derive an upper bound on the error of their solutions but do not show how to choose representative states to minimize it. The upper bound in our paper can be directly minimized using cover trees.

Kernel-based RL on representative states

In this work, we approximate the kernelized backup operator $\mathcal{T}_\lambda$ (Equation 3) on k representative states $\mathcal{Z} = \{z_1, \ldots, z_k\}$, which summarize the sample. In particular, we define a new operator:

$$\tilde{\mathcal{T}}_\lambda V(\mathbf{x}) = \max_a \sum_{t \in \tau^a} \tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} \left[r_t + \gamma V(\xi(\mathbf{x}'_t))\right], \quad (5)$$

where $\xi : \mathbf{x} \to z$ is a function that maps states to the closest representative state. Similarly to kernel-based RL, the kernel $\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}}$ is normalized such that $\sum_{t \in \tau^a} \tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} = 1$ for every state $\mathbf{x}$ and action a.

The operator $\tilde{\mathcal{T}}_\lambda$ can be restated in terms of the representative states. Before we show how, we introduce new notation. The set of indices t where $a_t = a$, $\xi(\mathbf{x}_t) = z$, and $\xi(\mathbf{x}'_t) = z'$ is denoted by $\tau^a_{zz'}$. Moreover, we introduce two supersets of $\tau^a_{zz'}$, $\tau^a_z = \bigcup_{z'} \tau^a_{zz'}$ and $\tau^a = \bigcup_z \tau^a_z$. The cardinality of the sets $\tau^a_{zz'}$, τ^a_z, and τ^a is $n^a_{zz'}$, n^a_z, and n^a, respectively. Based on Equation 5 and the new notation, it follows that:

$$\begin{aligned}
&\tilde{\mathcal{T}}_\lambda V(\mathbf{x}) \\
&= \max_a \sum_{t \in \tau^a} \tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} \left[r_t + \gamma V(\xi(\mathbf{x}'_t))\right] \\
&= \max_a \sum_z \sum_{t \in \tau^a_z} \tilde{\lambda}^a_{z\mathbf{x}} \left[r_t + \gamma V(\xi(\mathbf{x}'_t))\right] \\
&= \max_a \sum_z \tilde{\lambda}^a_{z\mathbf{x}} n^a_z \left[\sum_{t \in \tau^a_z} \frac{r_t}{n^a_z} + \gamma \sum_{z'} \sum_{t \in \tau^a_{zz'}} \frac{V(\xi(\mathbf{x}'_t))}{n^a_z}\right] \\
&= \max_a \sum_z \tilde{\lambda}^a_{z\mathbf{x}} n^a_z \left[\sum_{t \in \tau^a_z} \frac{r_t}{n^a_z} + \gamma \sum_{z'} \frac{n^a_{zz'}}{n^a_z} V(z')\right]. \quad (6)
\end{aligned}$$

The terms:

$$\hat{\lambda}^a_{z\mathbf{x}} = \tilde{\lambda}^a_{z\mathbf{x}} n^a_z, \quad \hat{r}^a_z = \sum_{t \in \tau^a_z} \frac{r_t}{n^a_z}, \quad \hat{p}^a_{zz'} = \frac{n^a_{zz'}}{n^a_z} \quad (7)$$

can be viewed as a model in the space of representative states and the final form of the operator $\tilde{\mathcal{T}}_\lambda$ is given by:

$$\tilde{\mathcal{T}}_\lambda V(\mathbf{x}) = \max_a \sum_z \hat{\lambda}^a_{z\mathbf{x}} \left[\hat{r}^a_z + \gamma \sum_{z'} \hat{p}^a_{zz'} V(z')\right]. \quad (8)$$

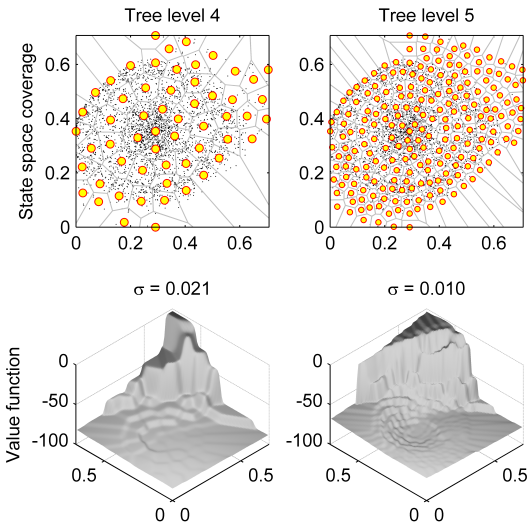

Figure 1: Covering the state space of the mountain car problem by representative states (yellow dots) induced by a cover tree. The representative states correspond to the nodes at the 4th and 5th levels of the tree. The value functions are learned by kernel-based RL on representative states.

Note that the operator $\tilde{\mathcal{T}}_\lambda$ depends on the value function V in k states. Therefore, the backup of V by $\tilde{\mathcal{T}}_\lambda$ can be computed in $\theta(k^3)$ time because V needs to be updated in only k states and the cost of each update is $\theta(k^2)$. In comparison, a single backup in kernel-based RL takes $\theta(n^2)$ time.

Algorithm

Kernel-based RL on representative states involves four steps. First, we sample the problem using its simulator. Second, we map every sampled state $\mathbf{x}_t$ and $\mathbf{x}'_t$ to its representative state. Third, we build a model in the space of representative states (Equation 7). Finally, we use policy iteration to find the fixed point of Equation 8

The time complexity of sampling and mapping to the closest representative state is $O(n)$ and $O(kn)$, respectively. The model can be built in $O(n+k^2)$ time and each step of policy iteration step consumes $O(k^3)$ time. Hence, the overall time complexity of our approach is $O(kn+k^3)$. If the number of representative states k is treated as a constant with respect to n, the time complexity of our approach is $O(n)$.

Figure 1 shows examples of value functions computed by policy iteration. Note that the value functions improve as the number of representative states k increases. In the rest of the section, we analyze our solutions and discuss how to choose the representative states.

Theoretical analysis

First, we show that the operator $\tilde{\mathcal{T}}_\lambda$ is a contraction mapping. Therefore, it has a unique fixed point.

Proposition 1. *The operator $\tilde{\mathcal{T}}_\lambda$ is a contraction mapping.*

Proof: Let V and U be value functions on the state space $\mathbf{X}$. Then:

$$\left\|\tilde{\mathcal{T}}_\lambda V - \tilde{\mathcal{T}}_\lambda U\right\|_\infty \leq \gamma \max_{\mathbf{x},a} \sum_{z,z'} \hat{\lambda}^a_{z\mathbf{x}} \hat{p}^a_{zz'} \left|V(z') - U(z')\right| \leq \gamma \left\|V - U\right\|_\infty .$$

The first step of the proof follows from Equation 8, the upper bound $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$ that holds for any two functions f and g, and the observation that $\hat{\lambda}^a_{z\mathbf{x}}$ and $\hat{p}^a_{zz'}$ (Equation 7) are non-negative. The second step is based on the fact that $\sum_{z,z'} \hat{\lambda}^a_{z\mathbf{x}} \hat{p}^a_{zz'} = 1$ for every state $\mathbf{x}$ and action a. Thus, all terms $\sum_{z,z'} \hat{\lambda}^a_{z\mathbf{x}} \hat{p}^a_{zz'} |V(z') - U(z')|$ can be bounded by $\|V - U\|_\infty$. This concludes our proof. ■

Second, we show that the fixed point of $\tilde{\mathcal{T}}_\lambda$ approximates the fixed point of $\mathcal{T}_\lambda$. Since the fixed point of $\mathcal{T}_\lambda$ approaches V^* as $n \to \infty$ and $\sigma \to 0$ (Ormoneit and Sen 2002), we conclude that our solution can approximate the optimal value function V^* to an arbitrary level of granularity.

We make the following assumptions. First, all value functions V are Lipschitz continuous with the Lipschitz factor of L_V. Second, $\lambda^a_{\mathbf{x}_t\mathbf{x}}$ (Equation 3) and $\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}}$ (Equation 5) are normalized Gaussian kernels (Equation 4). Note that for any given $\mathbf{x}$ and a, the difference in the values of the kernels can be bounded as $|\lambda^a_{\mathbf{x}_t\mathbf{x}} - \tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}}| \leq \frac{L_\sigma}{\Delta|\tau^a|} d(\mathbf{x}_t, \xi(\mathbf{x}_t))$, where L_σ is the Lipschitz constant of a Gaussian kernel, which has the radius of σ, and $\Delta\,|\tau^a|$ is the minimum normalizing constant, where $\Delta = \exp[-1/(2\sigma^2)]$. The normalizing constant can be bounded from below because the distance of any two states $\mathbf{x}$ and $\mathbf{x}'$ is bounded as $d(\mathbf{x}, \mathbf{x}') \leq 1$.

Proposition 2. *The max-norm error between the fixed points of the operators $\mathcal{T}_\lambda$ and $\tilde{\mathcal{T}}_\lambda$ is bounded by:*

$$\frac{d_{\max}}{1-\gamma}\left[\frac{L_\sigma}{\Delta} r_{\max} + \gamma\left(L_V + \frac{L_\sigma}{\Delta} V_{\max}\right)\right],$$

where $d_{\max}$ is the maximum distance between a state and its representative state, $r_{\max}$ is the maximum reward, and $V_{\max}$ is the maximum of the value function.

Proof: First, note that the max-norm error between the fixed points of two contraction mappings $\mathcal{T}_\lambda$ and $\tilde{\mathcal{T}}_\lambda$ is bounded as:

$$\begin{aligned}\left\|\tilde{\mathcal{T}}^{(n)}_\lambda V - \mathcal{T}^{(n)}_\lambda V\right\|_\infty &\leq \left\|\tilde{\mathcal{T}}_\lambda \tilde{\mathcal{T}}^{(n-1)}_\lambda V - \tilde{\mathcal{T}}_\lambda \mathcal{T}^{(n-1)}_\lambda V\right\|_\infty + \\ &\quad \left\|\tilde{\mathcal{T}}_\lambda \mathcal{T}^{(n-1)}_\lambda V - \mathcal{T}_\lambda \mathcal{T}^{(n-1)}_\lambda V\right\|_\infty \\ &\leq \gamma \left\|\tilde{\mathcal{T}}^{(n-1)}_\lambda V - \mathcal{T}^{(n-1)}_\lambda V\right\|_\infty + \epsilon \\ &\leq \frac{\epsilon}{1-\gamma},\end{aligned}$$

where $\epsilon = \left\|\tilde{\mathcal{T}}_\lambda V - \mathcal{T}_\lambda V\right\|_\infty$ is the max-norm error between one $\mathcal{T}_\lambda$ and $\tilde{\mathcal{T}}_\lambda$ backup for any value function V. Second, the error ϵ can be rewritten based on Equations 3 and 5 as:

$$\epsilon \leq \max_{\mathbf{x},a} \left|\sum_{t\in\tau^a} (\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} r_t - \lambda^a_{\mathbf{x}_t\mathbf{x}} r_t)\right| + \gamma \max_{\mathbf{x},a} \left|\sum_{t\in\tau^a} (\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} V(\xi(\mathbf{x}'_t)) - \lambda^a_{\mathbf{x}_t\mathbf{x}} V(\mathbf{x}'_t))\right| .$$

The two terms in the right-hand side of the inequality can be written in the form $|\langle p, f\rangle - \langle q, g\rangle|$, where p, f, q, and g are vectors such that $\|p\|_1 = 1$ and $\|q\|_1 = 1$, and bounded as:

$$\begin{aligned} |\langle p, f\rangle - \langle q, g\rangle| &= |\langle p, f\rangle - \langle p, g\rangle + \langle p, g\rangle - \langle q, g\rangle| \\ &\leq |\langle p, f - g\rangle| + |\langle p - q, g\rangle| \\ &\leq \|p\|_1 \|f - g\|_\infty + \|p - q\|_1 \|g\|_\infty , \end{aligned}$$

where the last step is due to the Hölder's inequality. Finally, we bound the resulting $\mathcal{L}_1$ and $\mathcal{L}_\infty$ norms as:

$$\begin{aligned} \textstyle\sum_{t\in\tau^a} |\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}}| &= 1 \\ \textstyle\sum_{t\in\tau^a} |\tilde{\lambda}^a_{\xi(\mathbf{x}_t)\mathbf{x}} - \lambda^a_{\mathbf{x}_t\mathbf{x}}| &\leq \frac{L_\sigma}{\Delta} d_{\max} \\ \max_{t\in\tau^a} |r_t - r_t| &= 0 \\ \max_{t\in\tau^a} |r_t| &\leq r_{\max} \\ \max_{t\in\tau^a} |V(\xi(\mathbf{x}'_t)) - V(\mathbf{x}'_t)| &\leq L_V d_{\max} \\ \max_{t\in\tau^a} |V(\mathbf{x}'_t)| &\leq V_{\max} \end{aligned}$$

and get an upper bound:

$$\epsilon \leq \frac{L_\sigma}{\Delta} d_{\max} r_{\max} + \gamma \left(L_V d_{\max} + \frac{L_\sigma}{\Delta} d_{\max} V_{\max} \right).$$

The claim of the proposition follows directly from substituting the upper bound into the first inequality. ■

Cover tree quantization

Proposition 2 suggests that the max-norm error between the fixed points of the operators $\mathcal{T}_\lambda$ and $\tilde{\mathcal{T}}_\lambda$ is bounded when the *cover error* is bounded:

$$d_{\max} = \max_t \min_{z\in\mathcal{Z}} d(\mathbf{x}_t, z). \tag{9}$$

Unfortunately, finding the set that minimizes the error of the cover is NP hard. Suboptimal solutions can be computed by *data quantization* (Gray and Neuhoff 1998) techniques. Two most popular approaches are k-means clustering and random sampling. In this work, we utilize cover trees (Beygelzimer, Kakade, and Langford 2006) because they allow us to find a set that approximately minimizes the cover error.

A *cover tree* (Beygelzimer, Kakade, and Langford 2006) is a tree-like data structure that covers data in a metric space at multiple levels of granularity. At depth j, the tree induces a set of representative points that are at least $1/2^j$ apart from each other and no data point is farther than $1/2^{j-1}$ from the closest representative point. Therefore, the error of the cover at depth j is $1/2^{j-1}$. Figure 1 shows examples of two cover tree covers.

Cover trees have a lot of nice properties. First, the deepest level of the tree with no more than k nodes covers data points within a multiplicative factor of 8 of the error of the optimal cover with k points. This is guaranteed for all $k \leq n$. Hence, the granularity of discretization does not have to be specified in advance. Second, cover trees can be easily updated online in $O(\log n)$ time per data point. Finally, the time complexity of building a cover tree on n data points is $O(n \log n)$. Thus, when $k > \log n$, the cover tree can be built faster than doing k-means clustering.

Practical issues

Our last discussion suggests that for a given k, the representative states z should be the nodes at the deepest level of the cover tree with no more than k nodes. Note that the number of the nodes c_j at the deepest level j is typically smaller than k, sometimes even by an order of magnitude, which impacts the quality of the approximation for a given k. To get a better cover, we choose the remaining $k - c_j$ nodes from the next level of the tree. The nodes are chosen in the order in which we inserted them into the tree.

The heat parameter σ in the Gaussian kernel (Equation 4) can be chosen using the cover tree. In particular, note that at depth j, no representative points are closer than $1/2^j$ and all data points are covered within $1/2^{j-1}$. Therefore, if the heat parameter is set as $\sigma = 1/(\kappa 2^j)$, no representative points are closer than $\kappa\sigma$ and each data point is covered within $2\kappa\sigma$. In our experiments, κ is set to 3. Since the representative states are chosen from two consecutive levels of the cover tree, the parameter σ is interpolated linearly as:

$$\sigma = \frac{1}{\kappa} \frac{1}{c_{j+1} - c_j} \left(\frac{k - c_j}{2^{j+1}} + \frac{c_{j+1} - k}{2^j} \right), \tag{10}$$

where j is the deepest level of the tree such that $c_j \leq k$, and c_j and c_{j+1} is the number of the nodes at levels j and $j + 1$, respectively.

Finally, note that our model (Equation 7) becomes imprecise and unstable as k approaches the sample size n, because each sample $\mathbf{x}_t$ is mapped to only one representative state z. To make the model more stable, we suggest substituting the counts in Equation 7 for a *smoothing kernel* $\psi^a_{\mathbf{x}_t z}$:

$$\begin{aligned} \hat{\lambda}^a_{z\mathbf{x}} &= \tilde{\lambda}^a_{z\mathbf{x}} \textstyle\sum_{t\in\tau^a} \psi^a_{\mathbf{x}_t z} \\ \hat{r}^a_z &= [\textstyle\sum_{t\in\tau^a} \psi^a_{\mathbf{x}_t z} r_t][\sum_{t\in\tau^a} \psi^a_{\mathbf{x}_t z}]^{-1} \\ \hat{p}^a_{zz'} &= [\textstyle\sum_{t\in\tau^a} \psi^a_{\mathbf{x}_t z} \psi^a_{\mathbf{x}'_t z'}][\sum_{t\in\tau^a} \psi^a_{\mathbf{x}_t z}]^{-1}, \end{aligned} \tag{11}$$

which is normalized such that $\sum_z \psi^a_{\mathbf{x}_t z} = 1$ for all states $\mathbf{x}_t$ and actions a. The kernel is defined as:

$$\psi^a_{\mathbf{x}_t z} \propto \begin{cases} \exp\left[-\frac{d^2(\mathbf{x}_t, z)}{2\sigma^2}\right] & d(\mathbf{x}_t, z) \leq d_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Since the kernel is truncated at $d_{\max}$ and normalized, we can bound the error of the corresponding Bellman operator as in Proposition 2. The model in our experiments is smoothed as described in Equations 11 and 12.

Experiments

We perform three experiments. First, we study how our policies improve with the number of representative states k. We also show that kernel-based RL with cover-tree quantization produces better policies than k-means and random quantization. Second, we compare our policies to three state-of-the-art baselines. Finally, we solve a high-dimensional problem with no apparent structure.

Our solution is evaluated on two benchmark control problems with 2 to 4 continuous state variables (Sutton and Barto 1998) and one new problem with 64 variables. All problems are solved as discounted MDPs and our results are averaged

Problem	Training		
	Number of episodes	Episode length	Discount factor γ
Mountain car	50 to 500	300	0.99
Acrobot	10 to 100	5,000	1.00
Favorite images	100	100	0.99
Problem	**Evaluation**		
	Number of episodes	Episode length	
Mountain car	100	500	
Acrobot	100	1,000	
Favorite images	100	300	

Figure 2: Training and testing parameters in our problems.

over 50 randomly initialized runs. The setting of our parameters is shown in Figure 2. The exploration policy is random. How to combine our approach with more intelligent policies is discussed in the conclusions.

Benchmark control problems

Mountain car (Sutton and Barto 1998) is a problem in which an agent drives an underpowered car up to a steep hill. Since the car is underpowered, it cannot be driven directly up to the hill and must oscillate at its bottom to build enough momentum. The state of the problem is described by two variables, the position and velocity of the car, and the agent can choose from three actions: accelerate forward, accelerate backward, or no acceleration. The objective is to drive the car to the top of the hill in the minimum amount of time.

Acrobot (Sutton and Barto 1998) is a problem in which an agent operates a two-link underactuated robot that resembles a gymnast swinging on a high bar. The state of the problem is defined by four variables, the position and velocity of two acrobot's joints, and the agent can choose from three actions: a positive torque of a fixed magnitude at the second acrobot's joint, a negative torque of the same magnitude, or no torque. The goal is to swing the tip of the acrobot to a given height in the minimum amount of time.

Favorite images problem

In this paper, we introduce a new synthetic problem in which an agent browses a collection of images (Figure 3). The state of the problem is given by image features, which summarize the currently shown image. The agent can take two actions, ask for more or less similar images. The objective is to learn a policy that browses images that the agent likes.

The states in our problem are the first 10 thousand images in the CIFAR-10 dataset (Krizhevsky 2009). We extract 512 GIST descriptors from each image and project them on their 64 principal components. The projections represent our features. The transition model is defined as follows. If the agent asks for more similar images, the next image is selected from 10 most similar images to the current image. Otherwise, the next image is chosen at random. The reward of 1 is assigned to all images that the agent likes. For simplicity, we assume that the agent likes images of airplanes, which is about 10% of our dataset. For this reward model, we expect the optimal policy to seek airplane images and then browse among them.

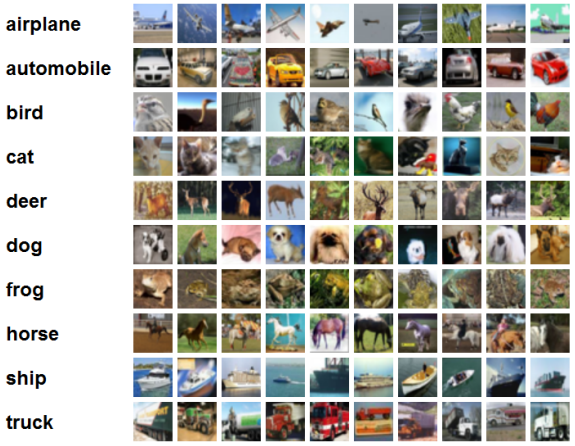

Figure 3: Examples of images from the CIFAR-10 dataset. The images are divided into 10 categories.

In our experiments, we show how to learn a better, and much less intuitive, policy.

The favorite images problem is motivated by real systems. In summary, we study media browsing, where the user seeks an object of interest. The preferences of the user are encoded in the reward function, and we want to compute a policy that guides the user to interesting images through similar content.

Another reason for introducing a new benchmark problem is that all popular RL problems are too small to demonstrate the benefits of our approach. Large-scale network problems, such as those studied by Kveton *et al.* (2006), have apparent structure that can be used to solve these problems efficiently. Our method is more suitable for high-dimensional problems where structure may exist but it is not obvious. The proposed problem has these characteristics.

Number of representative states k

In the first experiment, we study how the quality of our policies improves with the number of representative states k. We also show that kernel-based RL with cover-tree quantization produces better policies than k-means and random quantization. This experiment is performed on two control problems, mountain car and acrobot, which are simulated for the maximum number of training episodes (Figure 2). Our results are shown in Figure 4. We observe three major trends.

First, our policies improve as the number of representative states k increases. In the acrobot problem, the terminal state is initially reached in more than 800 steps. When the number of representative states increases to 1024, the state is reached in only 117 steps on average.

Second, cover-tree quantization usually yields better policies than k-means and random quantization, especially when the number of representative states k is small. These policies are learned in the same way as the other two policies, except for the representative states. Thus, the increase in the quality of the policies can be only explained by better representative states. Cover trees minimize the maximum distance between states and their representative states (Equation 9), uniformly across the state space. In comparison, k-means and random

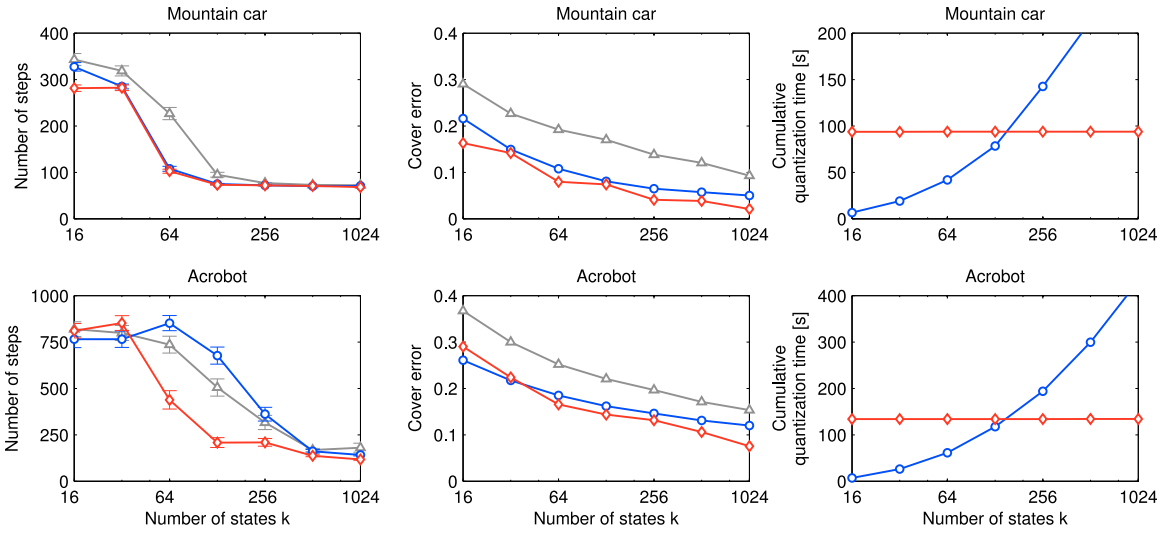

Figure 4: Kernel-based RL on k representative states, which are chosen using cover-tree quantization (red lines with diamonds), k-means quantization (blue lines with circles), and random quantization (gray lines with triangles). For each solution, we report the number of steps to reach a terminal state, the error of the state space cover (Equation 9), and quantization time.

quantization focus mostly on densely sampled regions of the space. Figure 4 compares errors of the state space covers for all three quantization methods. Cover trees usually yield the smallest error.

Third, the number of representative states k that produces good solutions may vary from problem to problem. One way of learning a good value of k is by searching through models of increasing complexity, for instance by doubling k. Cover trees are especially suitable for this search because only one cover tree is constructed for all k, and the computational cost of quantization for each additional k is close to 0 (Figure 4). In comparison, the cumulative cost of k-means quantization increases with each new k because the clustering of the state space needs to be recomputed.

Finally, note that our policies are computed fast. In particular, both the mountain car and acrobot problems are solved for 1024 states in less than 3 minutes. On average, the size of the sample n in the problems is 130k and 190k, respectively.

State-of-the-art solutions

In the second experiment, we compare our solutions to state-of-the-art results on the mountain car (Jong and Stone 2006) and acrobot problems (Sutton and Barto 1998). The number of training episodes varies according to the schedule given in Figure 2. Our policies are learned using 1024 representative states. This setting corresponds to our best results in the first experiment. In addition, we implemented in MATLAB fitted Q iteration with CART (Ernst, Geurts, and Wehenkel 2005). The CART is parameterized such that FQI policies are stable and improve as the sample size increases. More specifically, the tree is not pruned and the minimum number of examples in its leaf nodes is set to 20. Note that FQI with CART rarely converges to a fixed point. As a result, it is unclear when the algorithm should be terminated. In our experiments, we stop FQI when it consumes 3 times as much time as our approach at a given number of training episodes. At this point in time, FQI always oscillates around some solution. Our results are reported in Figure 5. We observe three major trends.

In the mountain car domain, we outperform the method of Jong and Stone (2006) for larger sample sizes and can reach the goal in only 69 steps. The main difference between Jong and Stone (2006) and our method is that we solve exactly an approximation to the original problem while Jong and Stone (2006) solve this problem approximately by heuristics. Note that the kernel width in our solutions is chosen automatically while Jong and Stone (2006) fine-tuned their kernel.

In the acrobot domain, we outperform the policy of Sutton and Barto (1998), which is learned from 25k basis functions. In comparison, our policies are induced by only 1024 states.

Finally, in both domains, we outperform fitted Q iteration with CART. It is possible that FQI with more complex averagers, such as ensembles of trees, can learn as good policies as our approach. However, it is unlikely that this can be done in comparable time because updating of the ensembles tends to be an order of magnitude slower than learning with CART (Ernst, Geurts, and Wehenkel 2005).

Large state spaces

In the third experiment, we apply our method to the favorite images problem and compare our policies to three baselines. The first baseline takes actions at random. The second baseline asks for more similar images if the immediate reward is positive. Otherwise, it chooses a random action. This policy acts greedily and henceforth we refer to it as a greedy policy. The third baseline is FQI with CART. The policy is parameterized as in the second experiment. The quality of solutions is measured by their discounted reward $\mathbb{E}\left[\sum_{t=0}^{300} \gamma^t r_t\right]$ in the first 300 steps.

Our results are shown in Figure 5. The reward of the random policy is 8.6. This reward is pretty low, even lower than the reward of the policy that constantly asks for different im-

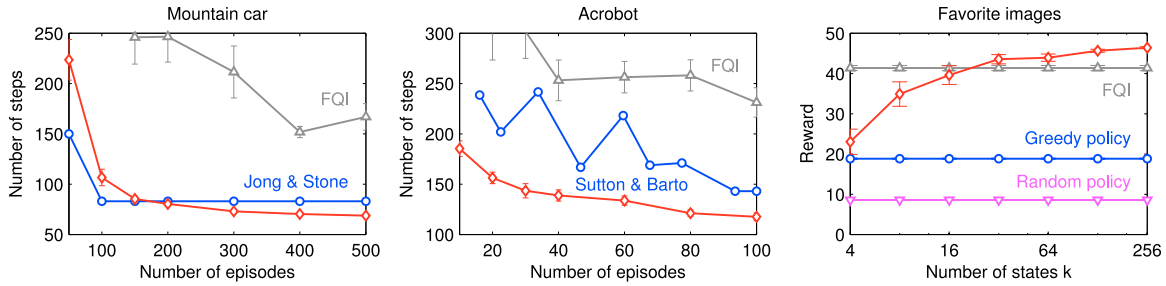

Figure 5: Comparison of kernel-based RL on k representative states (red lines with diamonds) to state-of-the-art solutions on the mountain car and acrobot problems, and heuristic baselines on the favorite images problem.

ages. The reward of the greedy baseline is 18.8. This reward can be derived analytically as follows. On average, 4.5 in 10 most similar images to an airplane are airplanes. As a result, a policy that asks for a similar image after seeing an airplane should earn about $\sum_{t=0}^{\infty} 0.45^t \approx 2$ times higher reward than the random policy, which is in line with our observations.

The reward of our policies increases as the number of representative states increases, and reaches as high as 46.4. This is 150% higher than the reward of the greedy policy and 12% higher than the reward of FQI. Our policies are significantly better than the greedy baseline because they redirect random walks on images to the parts of the space that mostly contain airplanes, and then keep asking for similar images.

Conclusions

In this paper, we propose a new approach to batch-mode RL with continuous state variables. The method discovers k representative states of a problem and then learns a kernel-based approximation on these states. Our solution is intuitive, easy to implement, and has only one tunable parameter, the number of representative states k. We outperform state-of-the-art solutions on two benchmark problems and also show how to solve a high-dimensional problem with 64 state variables.

The proposed approach is offline but we believe that it can be easily adapted to the online setting. In particular, note that cover-tree quantization is an online method and that the time complexity of $\tilde{\mathcal{T}}_\lambda$ backups is independent of the sample size n. Thus, the main challenge in making our method online is to update the model (Equation 7) efficiently. We believe that this can be done in $O(k^2)$ time per example, by updating the statistics in Equation 7 when the example is inserted into the cover tree. We also believe that our model can be built more intelligently, for instance by applying R-MAX (Brafman and Tennenholtz 2003) exploration from the top of the tree to the bottom. Jong and Stone (2009) recently studied another way of combining kernel-based RL and R-MAX.

Although this paper is focused on continuous-state MDPs, note that our ideas also apply to solving partially-observable MDPs (POMDPs). For instance, a POMDP can be treated as a belief-state MDP and solved by our method. Alternatively, cover trees can help in downsampling points for point-based value iteration (Pineau, Gordon, and Thrun 2003).

References

Barreto, A.; Precup, D.; and Pineau, J. 2011. Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems 24*, 720–728.

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

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

Beygelzimer, A.; Kakade, S.; and Langford, J. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, 97–104.

Brafman, R., and Tennenholtz, M. 2003. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Chow, C.-S., and Tsitsiklis, J. 1991. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control* 36(8):898–914.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.

Gray, R., and Neuhoff, D. 1998. Quantization. *IEEE Transactions on Information Theory* 44(6):2325–2383.

Jong, N., and Stone, P. 2006. Kernel-based models for reinforcement learning. In *ICML 2006 Workshop on Kernel Methods and Reinforcement Learning*.

Jong, N., and Stone, P. 2009. Compositional models for reinforcement learning. In *Proceeding of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Kveton, B.; Hauskrecht, M.; and Guestrin, C. 2006. Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research* 27:153–201.

Munos, R., and Moore, A. 1999. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1348–1355.

Ormoneit, D., and Sen, S. 2002. Kernel-based reinforcement learning. *Machine Learning* 49:161–178.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 1025–1032.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons.

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