Sparse Kernel-SARSA(λ) with an Eligibility Trace

Matthew Robards[1,2], Peter Sunehag[2], Scott Sanner[1,2], and Bhaskara Marthi[3]

[1] National ICT Australia
Locked Bag 8001
Canberra ACT 2601, Australia
[2] Research School of Computer Science
Australian National University
Canberra, ACT, 0200, Australia
[3] Willow Garage, Inc.,
68 Willow Road, Menlo Park, CA 94025, USA

Abstract. We introduce the *first* online kernelized version of SARSA(λ) to permit sparsification for arbitrary λ for $0 \leq \lambda \leq 1$; this is possible via a novel kernelization of the eligibility trace that is maintained separately from the kernelized value function. This separation is crucial for preserving the functional structure of the eligibility trace when using sparse kernel projection techniques that are essential for memory efficiency and capacity control. The result is a simple and practical Kernel-SARSA(λ) algorithm for general $0 \leq \lambda \leq 1$ that is memory-efficient in comparison to standard SARSA(λ) (using various basis functions) on a range of domains including a real robotics task running on a Willow Garage PR2 robot.

1 Introduction

In many practical reinforcement learning (RL) problems, the state space $\mathcal{S}$ may be very large or even continuous, leaving function approximation as the only viable solution. Arguably, the most popular form of RL function approximation uses a linear representation $\langle \mathbf{w}, \phi(s) \rangle$; although linear representations may seem quite limited, extensions based on kernel methods [14, 2, 13] provide (explicitly or implicitly) a rich feature map ϕ that in some cases permits the approximation of arbitrarily complex, nonlinear functions.

The simplicity and power of kernel methods for function approximation has given rise to a number of kernelized RL algorithms in recent years, e.g., [20, 4, 7, 8, 17]. In this paper we focus on online kernel extensions of SARSA(λ) — a *model-free* algorithm for learning optimal control policies in RL. However, unlike the Gaussian Process SARSA (GP-SARSA) approach and other kernelized extensions of SARSA [20,4], the main contribution of this paper is the *first* kernelized SARSA algorithm to allow for general $0 \leq \lambda \leq 1$ rather than restricting to just $\lambda \in \{0, 1\}$.

While generalizing an online kernelized SARSA algorithm to SARSA(λ) with $0 \leq \lambda \leq 1$ might seem as simple as using an eligibility trace [15], this leads

D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part III, LNAI 6913, pp. 1–17, 2011.

to theoretical and practical issues when sparsity is introduced. Because online kernel methods typically require caching *all* previous data samples, sparsification techniques that selectively discard cached samples and reproject the value representation are necessary [3, 5, 9]. However, in kernelized SARSA(λ), cached samples are required for both the value *and* eligibility trace representation, hence sparsification that focuses on low-error value approximations may inadvertently destroy structure in the eligibility trace. To address these issues, we *separate* the kernelization of the value function from the eligibility function, thus maintaining independent low-error, sparse representations of each. Despite the mathematical complications, we derive simple and efficient value and eligbility updates under this scheme. We point out here that once one wishes to "switch off" the learning and utilize the learned policy, our algorithm becomes linear in the number of samples stored and our experimental section shows that our algorithm seems to be more efficient than regular SARSA (λ).

These novel insights allow us to propose a practical, memory-efficient Kernel-SARSA(λ) algorithm for general $0 \leq \lambda \leq 1$ that scales efficiently in comparison to standard SARSA(λ) (using both radial basis functions and tile coding) on a range of domains including a real robotics task running on a Willow Garage PR2 robot. Crucially we note the use of $0 < \lambda < 1$ often leads to the best performance in the fewest samples, indicating the importance of the two main paper contributions:

1. the first generalization of kernelized SARSA(λ) algorithms to permit $0 \leq \lambda \leq 1$ with sparsification, and
2. the novel kernelization and projection of *separate* value and eligibility functions needed for low-error, memory-efficient approximations of kernelized SARSA(λ) with an eligibility trace.

2 Preliminaries

In this section we briefly review [15] MDPs, the SARSA(λ) algorithm, its extension for function approximation and some background on Reproducing Kernel Hilbert Spaces [1].

2.1 Markov Decision Processes

We assume a (finite, countably infinite, or even continuous) Markov decision process (MDP) [11] given by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$. Here, states are given by $s \in \mathcal{S}$, actions are given by $a \in \mathcal{A}$, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function with $T(s, a, s')$ defining the probability of transitioning from state s to s' after executing action a. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function where $r_t = R(s_t, a_t, s_{t+1})$ is the reward received for time t after observing the transition from state s_t to s_{t+1} on action a_t. Finally, $0 \leq \gamma < 1$ is a discount factor.

A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ specifies the action $\pi(s)$ to take in each state s. The value $Q_\pi(s, a)$ of taking an action a in state s and then following some policy π thereafter is defined using the infinite horizon, expected discounted reward

criterion: $Q_\pi(s,a) = E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0 = s, a_0 = a]$. Our objective is to learn the optimal Q^* s.t. $\forall s, a, \pi\ Q^*(s,a) \geq Q_\pi(s,a)$ in an episodic, online learning setting. Given Q^*, the optimal control policy π^* is simply $\pi^*(s) = \text{argmax}_a\, Q^*(s,a)$.

SARSA(λ) is a *temporal difference* RL algorithm for learning Q^* from experience [15]. We use SARSA(λ) in an *on-policy* manner where $Q_t(s,a)$ represents Q-value estimates at time t w.r.t. the greedy policy $\pi_t(s) := \text{argmax}_a\, Q_t(s,a)$. Initializing eligibilities $e_0(s,a) = 0$; $\forall s, a$, SARSA(λ) performs the following online Q-update at time $t+1$:

$$Q_{t+1}(s,a) = Q_t(s,a) + \eta err_t e_t(s,a);\ \forall s, a. \tag{1}$$

Here $\eta > 0$ is the learning rate, $err_t = Q_t(s_t, a_t) - R_t$ is the temporal difference error between the actual prediction $Q_t(s_t, a_t)$ and a bootstrapped estimate of $Q_t(s,a)$:

$$R_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1})$$

and e_t is the *eligibility trace* updated each time step as follows:

$$e_{t+1}(s,a) = \begin{cases} \gamma\lambda e_t(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_t(s,a) & \text{otherwise.} \end{cases} \tag{2}$$

The eligibility trace indicates the degree to which each state-action pair is updated based on future rewards. The parameter λ ($0 \leq \lambda \leq 1$) adjusts how far SARSA(λ) "looks" into the future when updating Q-values; as $\lambda \to 0$, SARSA(λ) updates become more myopic and it may take longer for delayed rewards to propagate back to earlier states.

For large or infinite state-action spaces it is necessary to combine SARSA(λ) with function approximation. Linear value approximation is perhaps the most popular approach: we let $\hat{Q}_t(s,a) = \langle \mathbf{w}_t, \phi(s,a) \rangle$ where $\mathbf{w}_t \in \mathbb{R}^d$ are $d > 0$ learned weights and $\phi : (s,a) \mapsto \phi(s,a)$ maps state-action (s,a) to features $\phi(s,a) \in \Phi \subseteq \mathbb{R}^d$. Because the optimal Q_t may not exist within the span of $\hat{Q}_t$, we minimize the error between Q_t and $\hat{Q}_t$ in an online empirical risk minimization framework; this can be done by gradient descent on the squared error loss function $l[Q_t, s_t, R_t] = \frac{1}{2}(Q_t(s_t,a_t) - R_t)^2$ w.r.t. each observed datum (s_t, a_t, R_t). For SARSA(λ) with general λ and linear function approximation, Sutton and Barto [15] provide the following update rule

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \eta err_t \mathbf{e}_t \phi(s_t, a_t) \tag{3}$$

with an eligibility vector updated through $\mathbf{e}_{t+1} = \gamma\lambda \mathbf{e}_t + \phi(s_t, a_t)$.

2.2 Reproducing Kernel Hilbert Spaces (RKHS)

When using Reproducing Kernel Hilbert Spaces [1] for function approximation (regression, classification) we define a feature map implicitly by defining a similarity measure called a kernel [14], [2], [13], e.g. a Gaussian kernel defined by $k(x,y) = e^{\|x-y\|^2/2\rho}$. Positive definite and symmetric kernels define a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ by completing the span of the functions

$k_x(\cdot) = k(x, \cdot)$ w.r.t. the inner product $\langle k_x, k_y \rangle_{\mathcal{H}} = k(x, y)$. Some kernels such as the Gaussian kernel are universal kernels, which means that the RKHS is dense in the space L^2 of square integrable functions. Using universal kernels means that any such (L^2) function can be approximated arbitrarily well by elements in $\mathcal{H}_k$.

If we have a feature map into a space with an inner product we have defined a kernel through $k(x, y) = \langle \phi(x), \phi(y) \rangle$. However, our intent is to start with a kernel like the Gaussian kernel and then use the feature map that is implicitly defined through that choice. This means that ϕ maps x to the function $k_x \in \mathcal{H}_k$. Note that $\phi(x)$ is not necessarily a finite vector anymore, but a possibly continuous function $k(x, \cdot)$.

3 Kernel-SARSA(λ)

We now generalize SARSA(λ) with function approximation to learn with large or even infinite feature vectors $\phi(s, a)$. We will use a reproducing kernel Hilbert space as our hypothesis space. The "weights" $\mathbf{w}$ that we will end up with are represented in the form $\mathbf{w} = \sum_i \alpha_i k((s_i, a_i), \cdot)$ and are really functions on $S \times A$. The corresponding Q function is

$$Q(s, a) = \sum_i \alpha_i k((s_i, a_i), (s, a)).$$

To define Kernel-SARSA(λ), we extend the SARSA (λ) update rule given in [15] to a RKHS setting. We slightly extend this update rule to include a regularizer term[1]. The update is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \Big[(Q_t(s_t, a_t) - R_t)\mathbf{e}_t - \xi \mathbf{w}_t \Big] \tag{4}$$

where $\mathbf{e}_t$ is the eligibility trace, updated through

$$\mathbf{e}_t = \gamma\lambda \mathbf{e}_{t-1} + \phi(s_t, a_t), \tag{5}$$

s.t. $\phi(s_t, a_t) = k((s_t, a_t), \cdot)$ and $\mathbf{e}_t$ is initialized to $\mathbf{0}$ at the start of each episode. ξ denotes the regularizer. Alternatively we may write the eligibility trace as

$$\mathbf{e}_t = \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} \phi(s_i, a_i) \tag{6}$$

where t_0 is the time at which the current episode began. Typically such a representation would be undesirable since it requires storing all past samples, however

[1] Regularization is important for exact kernel methods that cache all samples since they have no other means of capacity control. Later when we introduce sparsification into the algorithm, the regularization term ξ may be set to zero since sparsity performs the role of capacity control.

kernelizing our online algorithm already necessitates storing all previously visited state-action pairs. Now, by substituting (6) into (4), we get

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta \left(err_t \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} \phi(s_i, a_i) - \xi \mathbf{w}_t \right) \tag{7}$$

and assuming that $\mathbf{w}_0 = 0$ we see that

$$\mathbf{w}_t = \sum_{i=1}^{t-1} \alpha_i k((s_i, a_i), \cdot)$$

which leads us to an alternative formulation of (7). If err_t is the temporal difference error given by $(Q(s_t, a_t) - R_t))$ then

$$\sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot) := \sum_{i=1}^{t-1} (1 - \eta\xi)\alpha_i k((s_i, a_i), \cdot) - \eta err_t \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} k((s_i, a_i), \cdot). \tag{8}$$

Equating the coefficients of the basis functions leads to the update formulae:

$$\alpha_i := (1 - \eta\xi)\alpha_i, \ i = 1, \ldots, t_0 - 1 \tag{9}$$

$$\alpha_t := \eta err_t \tag{10}$$

and otherwise for $i = t_0, ..., t - 1$

$$\alpha_i := (1 - \eta\xi)\alpha_i - \eta err_t (\gamma\lambda)^{t-i-1} \tag{11}$$

As a notational observation crucial for the next section, we note that $\mathbf{w}_t$ is exactly the same object as Q_t. Normally in function approximation one may think of $Q_t(\cdot)$ as $\langle \mathbf{w}_t, \phi(\cdot) \rangle$ which is true here, but since $\phi(s, a)(\cdot) = k((s, a), \cdot)$, we can conclude from the reproducing property that $\mathbf{w}_t(s, a) = \langle \mathbf{w}_t, \phi(s, a) \rangle$. We will henceforth write Q_t instead of $\mathbf{w}_t$ in our equations.

Although surprisingly simple and elegant, we note that this is the *first* kernelization of SARSA(λ) for general $0 \leq \lambda \leq 1$ that the authors are aware of. However, it is impractical in general since it requires storing all state-action pairs; next we provide a memory-efficient variant of this novel kernelized SARSA(λ) approach needed to make it practically viable.

4 Memory-Efficient Kernel-SARSA (λ) Based on the Projectron

We now have the foundations for a powerful kernel reinforcement learning algorithm. Problematically, however, the memory required to store the old samples grows without bound. We will deal with this by extending sparse representation techniques from [3, 5, 9].

Following the Projectron approach [9], in order to bound the memory requirements of the algorithm, we ask ourselves at each time step, "to what extent can the new sample be expressed as a linear combination of old samples?". Consider the "temporal hypothesis" Q'_t given through equation (7), and its projection $Q''_t = P_{t-1}Q'_t$ onto $\mathcal{H}_{t-1}$ which is the span of the set $\mathbb{S}$ of previously stored basis functions. One must be careful when trying to use (7) since our previous update equations made the vital assumption that we store all points allowing the progression from (5) to (6). This assumption no longer holds since we plan to only add those new points which cannot be well represented as a linear combination of the old ones. This is an obstacle that has to be resolved to be able to use the Projectron technique in our setting.

4.1 *Separately* Kernelizing the Eligibility Trace

We note that (6) represents the eligibility trace as a linear combination of previous basis functions. Hence we can write the eligibility trace (which is now really an eligibility function) as a function (separate from the value function) parameterized by $\beta = \{\beta_i\}_{i=1,\dots,t}$ through

$$e_t = \sum_{i=1}^{t} \beta_i k((s_i, a_i), \cdot). \tag{12}$$

By substituting this form of the eligibility trace into its update equation (5) we get

$$\sum_{i=1}^{t} \beta_i k((s_i, a_i), \cdot) := \sum_{i=1}^{t-1} \gamma\lambda\beta_i k((s_i, a_i), \cdot) + k((s_t, a_t), \cdot) \tag{13}$$

and by equating the coefficients of the basis functions we get the parameter updates $\beta_i = \gamma\lambda\beta_i$ for $i = 1, \dots, t-1$ and $\beta = 1$.

4.2 Projected Kernel-SARSA(λ) Updates

We begin by plugging the update of the eligibility trace into the Q update, and call this the temporal hypothesis given by

$$Q'_t = (1 - \eta\xi)Q_{t-1} - \eta err_t \Big[\gamma\lambda \mathbf{e}_{t-1} + k((s_t, a_t), \cdot)\Big] \tag{14}$$

allowing us to write its projection $Q''_t = P_{t-1}Q'_t =$

$$(1 - \eta\xi)Q_{t-1} - \eta err_t \Big[\gamma\lambda \mathbf{e}_{t-1} + P_{t-1}k((s_t, a_t), \cdot)\Big]. \tag{15}$$

Our aim is to examine how well the temporal hypothesis Q'_t is approximated by its projection onto $\mathcal{H}_{t-1}$ which suitably is the hypothesis in $\mathcal{H}_{t-1}$ closest to h. We denote the difference $Q''_t - Q'_t$ by δ_t and note that

$$\delta_t = -\eta err_t \Big[P_{t-1}k((s_t, a_t), \cdot) - k((s_t, a_t), \cdot)\Big]. \tag{16}$$

By letting $\mathbf{K}_{t-1}$ denote the kernel matrix with elements given by $\{\mathbf{K}_{t-1}\}_{i,j} = k((s_i, a_i), (s_j, a_j))$, $\mathbf{k}_t$ denote the vector with ith element $\mathbf{k}_{t_i} = k((s_i, a_i), (s_t, a_t))$ and letting $\mathbf{d}^* = \mathbf{K}_{t-1}^{-1}\mathbf{k}_t$ we can as in [9] derive that

$$\|\delta_t\|^2 = \eta^2 err_t^2[k((s_t, a_t), (s_t, a_t)) - \mathbf{k}_t^T\mathbf{d}^*]. \tag{17}$$

Now if $\|\delta_t\|^2$ is below some threshold ϵ, we update the Q function by setting it to

$$(1 - \eta\xi)Q_{t-1} - \eta err_t\left[\gamma\lambda\mathbf{e}_{t-1} + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot)\right]. \tag{18}$$

We note that the last part of Eq(18) is the projection of the eligibility trace given by

$$P_{t-1}e_t = \gamma\lambda e_{t-1} + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot) \tag{19}$$

$$= \gamma\lambda\sum_{i=1}^{|\mathbb{S}|}\beta_i k((s_i, a_i), \cdot) + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot) \tag{20}$$

giving the updates

$$\beta_i := \gamma\lambda\beta_i + \mathbf{d}_i^*,\ i = 1, ..., |\mathbb{S}|. \tag{21}$$

Finally we write Q_t and e_t in their parameterized form to obtain

$$\begin{aligned}\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) =& (1 - \eta\xi)\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) - \eta err_t\gamma\lambda P_{t-1}e_t \\ =& (1 - \eta\xi)\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) - \eta err_t\gamma\lambda\sum_{i=1}^{|\mathbb{S}|}\beta_i k((s_i, a_i), \cdot)\end{aligned} \tag{22}$$

and by again equating the coefficients of the basis functions we get the α update

$$\alpha_i = (1 - \eta\xi)\alpha_i - \eta err_t\gamma\lambda\beta_i \tag{23}$$

for $i = 1, \ldots, |\mathbb{S}|$ when $\delta_t < \epsilon$. Otherwise α is updated as in Equations (9)-(10). To avoid the costly calculation of the inverse kernel matrix we calculate this incrementally as in [9] when a new sample is added:

$$\mathbf{K}_t^{-1} = \begin{pmatrix} & & 0 \\ & \mathbf{K}_{t-1}^{-1} & \vdots \\ & & 0 \\ 0 & \ldots & 0\ 0 \end{pmatrix} + \frac{1}{k((s_t, a_t), (s_t, a_t)) - \mathbf{k}_t^T\mathbf{d}^*}\begin{pmatrix}\mathbf{d}^* \\ -1\end{pmatrix}\left(\mathbf{d}^{*T} - 1\right). \tag{24}$$

Algorithm 1. Memory Efficient Kernel-SARSA (λ)

INPUTS:

- $\pi_0, \epsilon, \eta, \lambda$
- $\mathbb{S} = \emptyset$

1. DO
   (a) Select action a_t (e.g. greedily) in current state s_t and observe reward r_t
   (b) $\mathbf{d}^* \leftarrow \mathbf{K}_{t-1}^{-1}\mathbf{k}_t$
   (c) $\|\delta_t\|^2 \leftarrow \eta^2 err_t^2[k((s_t,a_t),(s_t,a_t)) - \mathbf{k}_t^T\mathbf{d}^*]$
   (d) if $(\|\delta_t\|^2 < \epsilon)$
   - for $i = 1,\ldots,|\mathbb{S}|$
     - $\beta_i \leftarrow \gamma\lambda\beta_i + \mathbf{d}_i^*$
     - $\alpha_i \leftarrow (1-\eta\xi)\alpha_i - \eta err_t\gamma\lambda\beta_i$

   (e) else
   - Add $k((s_t,a_t),\cdot)$ to $\mathbb{S}$
   - $\beta_{|\mathbb{S}|} \leftarrow 1$
   - for $i = 1,\ldots,|\mathbb{S}|-1$
     - $\beta_i \leftarrow \gamma\lambda\beta_i$
   - for $j = 1,\ldots,|\mathbb{S}|$
     - $\alpha_i \leftarrow (1-\eta\xi)\alpha_i - \eta err_t\gamma\lambda\beta_i$
   - Update $\mathbf{K}_{t-1}^{-1}$ through (24).
2. UNTIL policy update required

From here on, references to Kernel-SARSA(λ) imply the memory-efficient version in Algorithm 1 and not the version in Section 3. This first version from Section 3 (a) cannot be used directly since it stores every sample, and (b) if sparsified naïvely via the Projectron, leads to an algorithm that stops learning long before convergence because most new kernel samples are discarded (since most lie within the span of previous samples) — unfortunately, the eligibility trace is defined in terms of these new samples! In this way, the eligibility trace is not updated and a directly sparsified approach to Section 3 will prematurely cease to learn, rendering it useless in practice.[2]

5 Empirical Evaluation

Having now completed the derivation of our memory efficient Kernel-SARSA(λ) algorithm, we proceed to empirically compare it to two of the most popular and useful function approximation approaches for SARSA(λ): one version using kernel (RBF) basis functions (n.b., not the same as Kernel-SARSA(λ) but with

[2] As such, this paper contributes much more than a simple combination of Section 3 and the Projectron [9] (which does not work) — it makes the crucial point that value function and eligibility function must be *separately* kernelized and projected as presented in Section 4.

similar function approximation characteristics) and the other using standard tile coding [15].

Our experimental objectives are threefold: (1) to show that Kernel-SARSA(λ) learns better with less memory than the other algorithms, (2) to show that $0 < \lambda < 1$ leads to optimal performance for Kernel-SARSA(λ) on each MDP, and (3) that Kernel-SARSA(λ) can learn a smooth nonlinear Q-function in a continuous space with less memory than competing algorithms *and* which is nearly optimal in a real-world domain.

5.1 Problems

We ran our algorithm on three MDPs: two standard benchmarks and one real-world robotics domain.

Pole balancing (cart pole): requires the agent to balance a pole hinged atop a cart by sliding the cart along a frictionless track. We refer to [15] for a specification of the transition dynamics; rewards are zero except for -1, which is received upon failure (if the cart reaches the end of the track, or the pole exceeds an angle of ± 12 degrees). At the beginning of each episode we drew the initial pole angle uniformly from $[\pm 3]$ degrees. Further, we cap episode lengths at 1000 time steps. We report on a noisy version of this problem where we add $\pm$ 50% noise to the agents actions, that is, when the agent commands a force of 1, the actual force applied is drawn uniformly from the interval $[0.5, 1.5]$. Note that, since we report on time per episode for this task, higher is better.

Mountain car: involves driving an underpowered car up a steep hill. We use the state/action space, and transition/reward dynamics as defined in [15]. In order to solve this problem the agent must first swing backwards to gain enough velocity to pass the hill. The agent receives reward -1 at each step until failure when reward 1 is received. We capped episodes in the mountain car problem to 1000 time steps. The car was initialized to a standing start (zero velocity) at a random place on the hill in each episode. Note that, since we report on time per episode for this task, lower is better.

Robot navigation: requires the agent to successfully drive a (real) robot to a specified waypoint. The state space $\mathcal{S} = (d, \theta, \dot{x}, \dot{\theta})$, where d is the distance to the goal, θ is the angle between the robot's forward direction and the line from the robot to the goal, $\dot{x}$ is the robot's forward velocity, and $\dot{\theta}$ is the robot's angular velocity. The action space is $a \in \{\ddot{x}, \ddot{\theta}\}$, which represents respective linear and angular accelerations. We restrict the accelerations to $1.0ms^{-2}$ and $1.0rads^{-2}$ with decisions made at 10Hz. This corresponds to acceleration of $1.0ms^{-1}$ per $\frac{1}{10}$ seconds for both x and θ. A reward of -1 is received at each time step. Further, a reward of 10 is received for success, -100 for leaving a 3 metre radius from the goal, and -10 for taking more than 1000 time steps; these last three events result in termination of the episode.

Table 1. The parameter setup we gave each algorithm. Here σ is the RBF tile width given to each algorithm as a fraction of the state space in each dimension after the state space was normalized to the unit hyper-sphere.

Domain	Algorithm	γ	λ	η	ξ	ϵ	σ
Mountain car	Kernel-SARSA(λ)	0.9999	0.6	0.5	—	5.0×10^{-5}	0.05
	RBF coding	0.9999	0.7	0.1	0	—	0.05
	Tile coding	0.9999	0.7	0.005	—	—	0.1
Cart pole	Kernel-SARSA(λ)	0.9999	0.7	0.1	—	5.0×10^{-7}	0.05
	RBF coding	0.9999	0.7	0.01	0.01	—	0.05
	Tile coding	0.9999	0.6	0.1	—	—	0.066
Robot navigation	Kernel-SARSA(λ)	0.9999	0.6	0.1	—	0.5	0.1
	RBF coding	0.9999	0.5	0.1	0.0	—	0.1
	Tile coding	0.9999	0.6	0.1	—	—	0.066

We used a high-fidelity simulator for training a Willow Garage PR2 robot[3] and then evaluated the learned Q-value policy on an actual PR2. In the simulator training phase, the robot's task is simply to drive directly to the waypoint. For the *in situ* robot testing phase, we gave the robot a global plan, from which it drives towards the nearest waypoint beyond a 1m radius at each time step. This has the effect of a single waypoint moving away at the same speed as the robot. At the end of the planned path, the waypoint stops moving, and the robot must drive to it. This RL problem requires a nonlinear Q-value approximation over a continuous space, and RL algorithms must deal with a high-noise environment on account of noisy actuators and an unpredictable surface response. Although the transition dynamics are noisy, we note that high-precision PR2 sensors render the state space fully observed for all practical purposes, making this an MDP.

5.2 Results

The above selection of problems demonstrate our performance in both difficult continuous state spaces requiring nonlinear approximation of Q-values and a real-world robotics navigation task.

For each MDP, we compare Kernel-SARSA(λ) to SARSA(λ) with tile coding and SARSA(λ) with RBF coding. The first metric we record for each algorithm and MDP is the memory usage in terms of the number of samples/basis functions vs. the episode number. Obviously, a small memory footprint while achieving near-optimal performance is generally desired. The second metric that we evaluate for each algorithm and MDP is the time/reward per episode, i.e., the number of steps until episode termination for *cart pole* and *mountain car* and the average reward per time step within an episode for *robot navigation*, both being recorded vs. the episode number. For the *mountain car* MDP, smaller episode length is better since episodes terminate with success, whereas for the *cart pole*

[3] `http://www.ros.org/wiki/pr2_simulator`

MDP, longer is better since episodes terminate with failure. In the *navigation task* MDP, larger average rewards per episode are better.

In the following results, each algorithm is configured with the parameter specifications in Table 1. The parameters were chosen from the following search space: $\lambda \in \{0.0, 0.1, \ldots, 0.9, 1.0\}$, $\eta \in \{1, 5\} \times 10^{-k}$, $\xi \in \{0, 1, 5\} \times 10^{-k}$, $\epsilon \in \{1, 5\} \times 10^{-k}$, $\sigma \in \frac{1}{n}$, $n \in \{5, 10, 15, 20\}$. For each algorithm and domain we chose the parameters which obtain the best result.

Memory Efficiency and Performance. Figure 1 shows the growth in the number of stored samples for Kernel-SARSA(λ), compared to the memory requirements of RBF coding and tile coding, for each of the three MDPs. We can see that Kernel-SARSA(λ) is *always* the most memory-efficient.

Figure 3 shows the time/reward for all three algorithms on all three domains. In brief, the results show that Kernel-SARSA(λ) is always among the best in terms of final episode performance (and is the best for both *cart pole* and *robot navigation*). Kernel-SARSA(λ) also learns fastest in *cart pole* while performing mid-range among the other two algorithms on both *mountain car* and *robot navigation*. This is impressive given the small amount of memory used by Kernel-SARSA(λ) relative to the other algorithms.

We now discuss results by domain in more detail:

Mountain Car: All three methods can solve this domain rather quickly. In Figure 2 (top) we see that the RBF basis functions provide the steepest decline in time needed to complete the task. Kernel-SARSA(λ) starts somewhat slower because it needs to accumulate basis functions to be able to learn the optimal policy. RBF nets and Kernel-SARSA(λ) reached an optimal policy in approximately the same number of episodes while tile coding needed many more. RBF coding showed some instabilities much later on while memory efficient Kernel-SARSA(λ) remained stable. Kernel-SARSA(λ) stores less than 150 samples, an order of magnitude smaller than best performing tile coding.

Cart Pole: As can be seen in Figure 2 (middle) Kernel-SARSA(λ) clearly outperforms both RBF nets and tile coding and learns to indefinitely balance the pole after a very small number of episodes. Neither of the comparison methods learn to reliably balance the pole during the 100 episodes. Impressively, Kernel-SARSA(λ) only stores a total of 60 samples in its representation of a near-optimal policy.

Robot Navigation: We can see that Kernel-SARSA(λ) performs the best in the long run from Figure 2 (bottom) and that for this problem, the other algorithms are far less memory efficient by at least an order of magnitude as shown in Figure 1 (bottom). While it takes a little while for Kernel-SARSA(λ) to collect an appropriate set of kernel samples before asymptoting in memory, it appears able to learn a better performing Q-function by the final episode.

Benefits of General λ. Figure 2 shows the time/reward for Kernel-SARSA(λ) for varying values of λ on all three MDPs. The basic trend here is quite clear

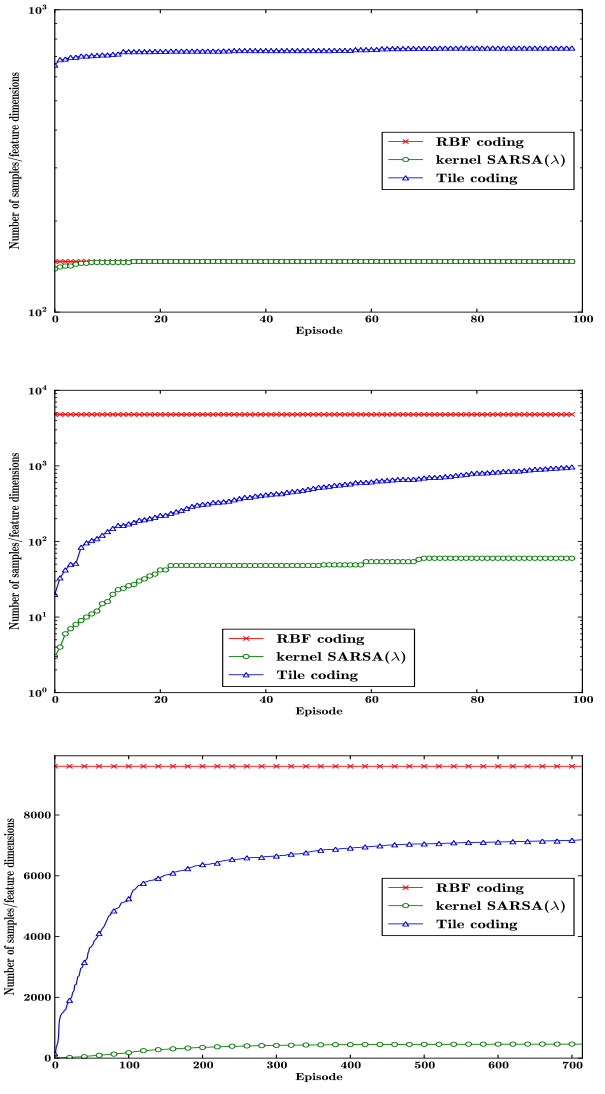

Fig. 1. Number of samples/basis functions vs. episode for all algorithms on *mountain car* (top), *cart pole* (middle), and *robot navigation* (bottom) problems. Note the vertical axis log scale on the top two plots.

— the best performing λ on all three domains satisfies $.4 \leq \lambda \leq .8$, which indicates that both for a fast initial learning rate and good asymptotic learning performance, the best $\lambda \notin \{0, 1\}$. Even further we note that $\lambda = 1$ leads to poor performance on all problems and $\lambda = 0$ leads to only mid-range performance in general.

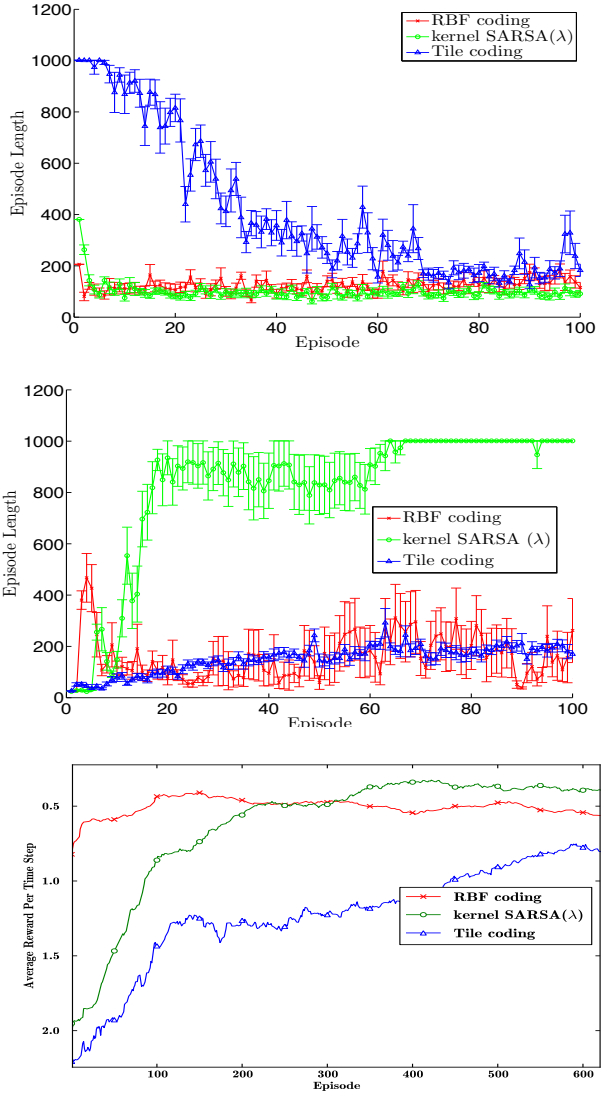

Fig. 2. Average time per episode for all algorithms and standard error over 10 runs on *mountain car* (top), *cart pole* (middle) and **moving average reward per episode** evaluated on the *robot navigation* (bottom)

Evaluation on Robot Navigation. When learning was complete in the simulator, learned Q-values were transferred to a Willow Garage PR2 robot, which was given two paths to follow as described previously. These two paths are shown in Figure 4, both demonstrating how well the agent has learned to navigate.

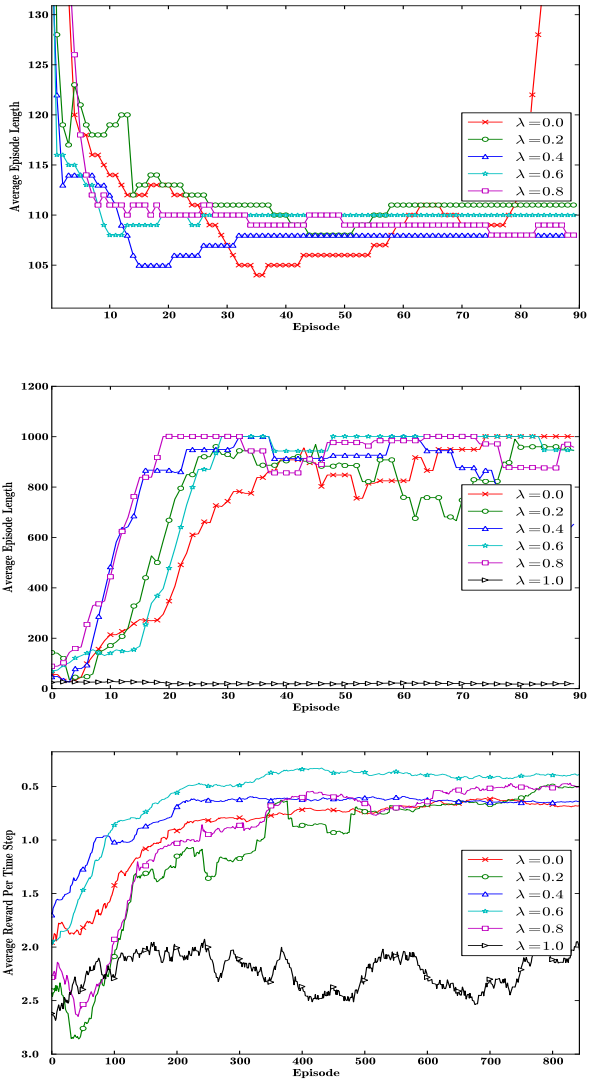

Fig. 3. Average time per episode for Kernel-SARSA(λ) with various values of λ on *mountain car* (top), *cart pole* (middle) and **moving average reward** per episode evaluated on the *robot navigation* (bottom)

We note that the more kernel samples that are stored, the more irregular the function approximation surface may be. However, Kernel-SARSA(λ)'s Projectron-based RKHS sparsification stored relatively few samples compared to other algorithms as shown in Figure 5.2, leading to a smooth Q-value approximation as exhibited in the smoothness of the navigation paths in Figure 4.

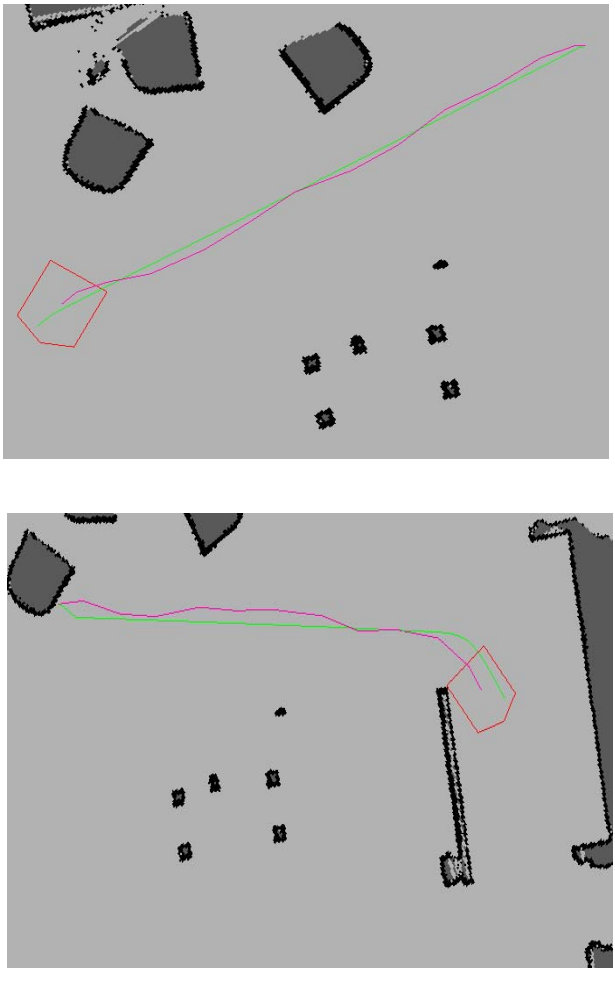

Fig. 4. The **ideal path (green), and the near-optimal path followed by Kernel-SARSA(λ) (purple)**. Here we provided the robot a virtually straight path (top) and a turn into a corridor (bottom).

6 Related Work

Both model-based and model-free approaches to using kernel methods in reinforcement learning have been proposed. In the model-based approaches, kernelized regression is used to find approximate transition and reward models which are used to obtain value function approximations. In the model-free approaches, the task of finding an approximation of the value function through regression is addressed directly as in Kernel-SARSA(λ). Gaussian Process kernel regression has been used for both approaches: [12, 8] studied the model-based setting

and [6, 20] studied the model-free setting. [19, 18] took a direct approach to replacing the inner product with a kernel in LSTD, similar to our approach in Kernel-SARSA(λ) but offline. An earlier approach at Kernel-Based Reinforcement Learning [10] that calculated a value approximation offline was modified by [7] into a model-based online approach. These approaches used kernels for "local averaging" and can be viewed as a direct approach to kernelization. Equivalence of previous kernel based approaches [20, 12, 18] to reinforcement learning has been proven by [16] except for the manner of regularization. But *crucially,, all of the sparse, online, model-free approaches have failed to incorporate eligibility traces for* $0 < \lambda < 1$ as we provided in the novel contribution of kernelized SARSA(λ) in this paper — the first online kernelized SARSA(λ) algorithm to show how kernelization can be extended to the eligibility trace.

7 Conclusion

We contributed the first online kernelized version of SARSA(λ) to permit arbitrary λ for $0 \leq \lambda \leq 1$ with sparsification; this was made possible via a novel kernelization of the eligibility trace maintained separately from the kernelized value function. We showed the resulting algorithm was up to an order of magnitude more memory-efficient than standard function approximation methods, while learning performance was generally on par or better. We applied Kernel-SARSA(λ) to a continuous state robotics domain and demonstrated that the learned Q-values were smoothly and accurately approximated with little memory, leading to near-optimal navigation paths on a Willow Garage PR2 robot. Importantly, we showed $.4 < \lambda < .8$ was crucial for optimal learning performance on all test problems, *emphasizing the importance of general* λ *for efficient online kernelized RL in complex, nonlinear domains* as contributed by the novel kernelization and efficient projection of the eligibility trace in Kernel-SARSA(λ) as introduced in this paper.

Acknowledgements. The first and third authors are supported by NICTA; NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Part of this work was carried out while the first author was an intern at Willow Garage.

References

1. Aronszajn, N.: Theory of reproducing kernels. Transactions of the American Mathematical Society 68 (1950)
2. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics), 1st edn. Springer, Heidelberg (October 2007)
3. Csato, L., Opper, M.: Sparse representation for gaussian process models. In: Advances in Neural Information Processing Systems, vol. 13 (2001)

4. Engel, Y.: Algorithms and Representations for Reinforcement Learning. PhD thesis, Hebrew University (April 2005)
5. Engel, Y., Mannor, S., Meir, R.: Sparse online greedy support vector regression. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) ECML 2002. LNCS (LNAI), vol. 2430, pp. 84–96. Springer, Heidelberg (2002)
6. Engel, Y., Mannor, S., Meir, R.: Bayes meets bellman: The gaussian process approach to temporal difference learning. In: Proc. of the 20th International Conference on Machine Learning, pp. 154–161 (2003)
7. Jong, N., Stone, P.: Kernel-based models for reinforcement learning in continuous state spaces. In: ICML Workshop on Kernel Machines and Reinforcement Learning (2006)
8. Jung, T., Stone, P.: Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In: Proceedings of the European Conference on Machine Learning (September 2010)
9. Orabona, F., Keshet, J., Caputo, B.: The projectron: a bounded kernel-based perceptron. In: ICML 2008: Proceedings of the 25th International Conference on Machine Learning, pp. 720–727. ACM, New York (2008)
10. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning 49, 161–178 (2002)
11. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
12. Rasmussen, C.E., Kuss, M.: Gaussian processes in reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 16, pp. 751–759. MIT Press, Cambridge (2003)
13. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
14. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
15. Sutton, R., Barto, A.: Reinforcement Learning. The MIT Press, Cambridge (1998)
16. Taylor, G., Parr, R.: Kernelized value function approximation for reinforcement learning. In: ICML 2009: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 1017–1024. ACM, New York (2009)
17. Xu, X.: A sparse kernel-based least-squares temporal difference algorithm for reinforcement learning. In: Jiao, L., Wang, L., Gao, X.-b., Liu, J., Wu, F. (eds.) ICNC 2006. LNCS, vol. 4221, pp. 47–56. Springer, Heidelberg (2006)
18. Xu, X., Hu, D., Lu, X.: Kernel-based least squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks 18(4), 973–992 (2007)
19. Xu, X., Xie, T., Hu, D., Lu, X., Xu, X., Xie, T., Hu, D., Lu, X.: Kernel least-squares temporal difference learning kernel least-squares temporal difference learning. International Journal of Information Technology, 54–63 (2005)
20. Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with Gaussian processes. In: 22nd International Conference on Machine Learning (ICML 2005), Bonn, Germany, pp. 201–208 (2005)