RKHS Temporal Difference Learning

Matthew W. Robards Peter Sunehag Scott Sanner

Abstract

We introduce a Reproducing Kernel Hilbert Space version of SARSA(λ) including a simple and intuitive formula for the eligibility trace. The eligibility trace which has been missing from previous approaches to kernelized SARSA (Gaussian Process SARSA [13]), is essential for fast learning in practise. Furthermore, inspired by the Projectron algorithm [5], we introduce a memory efficient version which dramatically reduces the number of terms in the representation of the estimated value function. In addition this reduction in the number of parameters improves stability and removes the need for regularization. The result is a simple and practical algorithm that can learn highly non-linear value functions without requiring extensive feature engineering. We show that our method outperforms related classical methods like tile coding and RBF nets.

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 $\langle \mathbf{w}, \phi(s) \rangle$. In machine learning, a popular alternative to explicitly providing a feature map, is to define a similarity measure called a kernel [9, 1, 8] through $k(s,r) = \langle \phi(s), \phi(r) \rangle$ for $s, r \in \mathcal{S}$, e.g., a Gaussian kernel defined by $k(s,r) = e^{\|s-r\|^2/2\rho}$. Positive definite and symmetric kernels define a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ by completing the span of the functions $k_s(\cdot) = k(s,\cdot)$ w.r.t. the inner product $\langle k_s, k_r \rangle_{\mathcal{H}} = k(s,r)$. 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 $\mathcal{H}_k$.

In this paper, we generalize the temporal difference SARSA(λ) algorithm to learn in RKHS. Due to the potential infinite dimensionality of the RKHS, this will necessarily be performed via a dual formulation of SARSA(λ) that we refer to as RKHS-SARSA(λ). In addition to reducing the RL feature engineering process to the selection of an appropriate kernel class, RKHS-SARSA(λ) facilitates efficient learning of nonlinear value functions in possibly continuous spaces. We achieve this by finding an intuitive and elegant dual formulation of the eligibility trace [10] permitting efficient online kernel learning for the case of $\lambda \geq 0$; this frees us from the $\lambda = 0$ restriction of previous Gaussian Processes kernel approaches to SARSA [13]. Furthermore, we dramatically reduce space and improve convergence speed by borrowing from the Projectron [5]. We evaluate RKHS-SARSA(λ) on various RL domains showing that $\lambda > 0$ can yield much faster learning than $\lambda = 0$ and that RKHS-SARSA(λ) outperforms related function approximation techniques proposed in the literature [4].

Equivalence of previous kernel based approaches [13, 7, 12] to reinforcement learning has been proven [11] except for the manner of regularizing. All of these approaches have failed to incorporate eligibility traces. We take a direct approach to using online kernel regression for function approximation that allows for simultaneously using eligibility traces and memory control without the need for explicit regularization.

2 Preliminaries

We assume a (finite, countably infinite, or even continuous) Markov decision process (MDP) [6] given by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$. Here, states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function with $T(s, a, s')$ defining the probability of transitioning from state s to s' after executing action a. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ 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 $\pi : \mathcal{S} \to \mathcal{A}$ 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 an optimal policy π^* s.t. $\forall s, a, \pi'\ Q_\pi(s, a) \geq Q_{\pi'}(s, a)$ in an episodic learning setting.

SARSA(λ) is a *temporal difference* RL algorithm for learning $Q_\pi(s, a)$ from experience [10]. 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 $e_0(s, a) = 0$; $\forall s, a$, SARSA(λ) performs the following Q-update at at time $t + 1$:

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

Here $\eta_t > 0$ is the learning rate, $\delta_t = R_t - Q_t(s_t, a_t)$ is the temporal difference error between the actual prediction $Q_t(s_t, a_t)$ and a bootstrapped estimate of $Q_t(s, a)$: $R_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1})$; 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, [10] provide the following stochastic gradient descent (SGD) update rule

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

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

3 RKHS-SARSA(λ)

We now generalize SARSA(λ) with function approximation from the last section to learn with large or even infinite feature vectors $\phi(s, a)$. We show how to do this via online learning in a Reproducing Kernel Hilbert Space (RKHS) by first introducing regularized risk minimization in the RKHS and then deriving RKHS-SARSA(λ) in this framework.

3.1 Regularized Risk Minimization in RKHS

Suppose that samples $\{(x_t, r_t)\}_{t=1}^M$ (where $x_t = (s_t, a_t)$ in our RL setting) are drawn from a distribution on $\mathcal{X} \times \mathbb{R}$. Also suppose that we have a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ for a positive definite and symmetric kernel function $k(\cdot, \cdot)$ defined on $\mathcal{X}$ ($= \mathcal{S} \times \mathcal{A}$ later) and a loss function $l : \mathcal{H}_k \times \mathcal{X} \times \mathbb{R} \to \mathbb{R}_+$, which for $Q \in \mathcal{H}_k$ tells us how large the loss

is for having $Q(x)$ instead of r. We want to find the function that minimizes the expected loss (i.e., *risk*), which we estimate via the empirical risk:

$$R_{emp}(f) = \frac{1}{M}\sum_{i=1}^{M} l[Q, x_t, r_t]. \tag{4}$$

Minimizing the empirical risk over a large space of functions tends to cause overfitting and instead the regularized empirical risk $R_{reg,\xi}(Q) = R_{emp}(Q) + \frac{\xi}{2}\|Q\|^2_{\mathcal{H}_k}$ is minimized (ξ is a regularization parameter). The representer theorem tells us the minimizer $Q_M = \text{argmin}_Q R_{reg,\xi}(Q)$ can be written as

$$Q_M(\cdot) = \sum_{t=1}^{M} \alpha_t \langle \phi(x_t), \phi(\cdot)\rangle = \sum_{t=1}^{M} \alpha_t k(x_t, \cdot). \tag{5}$$

Since $Q(x) = \langle \mathbf{w}, \phi(x)\rangle$ (n.b., $\phi(x)$ may be in an infinite dimensional feature space), the representer theorem [2] tells us that $\mathbf{w} = \sum_{i=1}^{M} \alpha_t \phi(x_t)$. This allows learning in an infinite dimensional space since learning the optimal $\mathbf{w}$ depends on only finitely many parameters $\alpha_1, \ldots, \alpha_M$.

For the remainder of this paper we will assume the squared loss as traditionally used in RL with function approximation:

$$l[Q_t, x_t, r_t] = \frac{1}{2}(Q_t(x_t) - r_t)^2. \tag{6}$$

In the online setting, which is of interest here, we receive samples one at a time and we update parameters after each sample. The NORMA algorithm [3] utilizes the reproducing kernel property $Q_t(x_t) = \langle Q_t, k(x_t, \cdot)\rangle_{\mathcal{H}}$ and the simple derivative property $\partial_{Q_t}\langle Q_t, k(x_t, \cdot)\rangle_{\mathcal{H}} = k(Q_t, \cdot)$, to derive the SGD update formula

$$Q_{t+1} = (1 - \eta_t \xi)Q_t - (Q_t(x_t) - r_t)k(x_t, \cdot)\eta_t. \tag{7}$$

3.2 SARSA(λ) in the RKHS

To define this algorithm we extend the online gradient descent SARSA (λ) update rule given in [10] to a RKHS setting by using the dual formulation. We slightly extend this update rule to include a regularizer term (which will be set to zero for our final algorithm). The primal update is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \Big[(Q(s_t, a_t) - R_t)\mathbf{e}_t - \xi \mathbf{w}_t \Big] \tag{8}$$

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

$$\mathbf{e}_t := \gamma\lambda \mathbf{e}_{t-1} + \phi(s_t, a_t),\ \ \phi(s, a) = k((s, a), \cdot) \tag{9}$$

and $\mathbf{e}_t$ is set to $\mathbf{0}$ at the beginning of each episode. ξ denotes the regularizer and $\phi : \mathcal{S} \times \mathcal{A} \to \mathcal{H}_k$ maps a state action pair to function in the RKHS. Alternatively we may write the eligibility trace as

$$\mathbf{e}_t := \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} \phi(s_i, a_i) \tag{10}$$

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 kernalizing our algorithm means storing all previously visited state action pairs anyway. By substituting (10) into (8), we get $\mathbf{w}_{t+1} =$

$$\mathbf{w}_t - \eta_t err(s_t, a_t, R_t) \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} \phi(s_i, a_i) - \xi \mathbf{w}_t \tag{11}$$

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 the dual update formulation of (11). If $err(s_t, a_t, R_t)$ is the temporal difference error given by $(Q(s_t, a_t) - R_t))$ then

$$\sum_{i=1}^{t} \alpha_i k((s_i, a_i) = \sum_{i=1}^{t-1} (1 - \eta\xi)\alpha_i k((s_i, a_i), \cdot) - \eta_t err(s_t, a_t, R_t) \sum_{i=t_0}^{t} (\gamma\lambda)^{t-i} k((s_i, a_i), \cdot). \tag{12}$$

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

$$\alpha_i = (1 - \eta\xi)\alpha_i \, i = 1, \ldots, t_0 - 1 \tag{13}$$

$$\alpha_i = (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)(\gamma\lambda)^{t-i-1}, i = t_0, \ldots, t - 1 \tag{14}$$

$$\alpha_t = \eta_t err(s_t, a_t, R_t). \tag{15}$$

4 Memory Efficient RKHS-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 easily blows out of control. We will deal with this by using a technique from the Projectron algorithm [5].

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 (11), 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 (11) since our previous update equations made the vital assumption that we store all points allowing the progression from Eq. (9) to Eq. (10). 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 Dealing With The Eligibility Trace

We note that Eq. (10) 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 parameterized by $\beta = \{\beta_i\}_{i=1,\ldots,t}$ through

$$e_t = \sum_{i=1}^{t} \beta_i k((s_i, a_i), \cdot). \tag{16}$$

By substituting this form of the eligibility trace into its update equation (9) we get

$$\sum_{i=1}^{t} \beta_i k((s_i, a_i), \cdot) := \sum_{i=1}^{t-1} \gamma\lambda\beta_i k((s_i, a_i), \cdot) + k((s_t, a_t), \cdot) \tag{17}$$

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

4.2 Projected Updates

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

$$Q'_t = (1 - \eta\xi)Q_{t-1} - \eta err(s_t, a_t, R_t)\Big[\gamma\lambda\mathbf{e}_{t-1} + k((s_t, a_t), \cdot)\Big] \tag{18}$$

allowing us to write its projection

$$Q''_t = P_{t-1}Q'_t = (1 - \eta\xi)Q_{t-1} - \eta err(s_t, a_t, R_t)\Big[\gamma\lambda\mathbf{e}_{t-1} + P_{t-1}k((s_t, a_t), \cdot)\Big]. \tag{19}$$

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. To this end we introduce the difference

$$\delta_t = Q''_t - Q'_t = -\eta err(s_t, a_t, R_t)\Big[P_{t-1}k((s_t, a_t), \cdot) - k((s_t, a_t), \cdot)\Big]. \tag{20}$$

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 [5] derive that

$$\|\delta_t\|^2 = \eta^2 err(s_t, a_t, R_t)^2[k((s_t, a_t), (s_t, a_t)) - \mathbf{k}_t^T\mathbf{d}^*]. \tag{21}$$

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

$$Q''_t := (1 - \eta\xi)Q_{t-1} - \eta err(s_t, a_t, R_t)\Big[\gamma\lambda\mathbf{e}_{t-1} + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot)\Big]. \tag{22}$$

We note that the last part of Eq(22) is the projection of the eligibility trace given by

$$P_{t-1}e_t = \gamma\lambda e_{t-1} + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot) \tag{23}$$

$$= \gamma\lambda\sum_{i=1}^{|\mathbb{S}|}\beta_i k((s_i, a_i), \cdot) + \sum_{i=1}^{|\mathbb{S}|}\mathbf{d}_i^* k((s_i, a_i), \cdot) \tag{24}$$

giving the updates

$$\beta_i := \gamma\lambda\beta_i + \mathbf{d}_i^*,\ i = 1, ..., |\mathbb{S}|. \tag{25}$$

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

$$\begin{aligned}\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) &:= (1 - \eta\xi)\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) - \eta err(s_t, a_t, R_t)\gamma\lambda P_{t-1}e_t. \\ &:= (1 - \eta\xi)\sum_{i=1}^{|\mathbb{S}|}\alpha_i k((s_i, a_i), \cdot) - \eta err(s_t, a_t, R_t)\gamma\lambda\sum_{i=1}^{|\mathbb{S}|}\beta_i k((s_i, a_i), \cdot),\end{aligned} \tag{26}$$

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

$$\alpha_i = (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)\gamma\lambda\beta_i, \qquad \text{for } i = 1, \ldots, |\mathbb{S}| \tag{27}$$

for all i when $\delta_t < \epsilon$. Otherwise α is updated as in Equations (13)-(15). To avoid the costly calculation of the inverse kernel matrix we calculate this incrementally as in [5] when a new sample is added:

$$\mathbf{K}_t^{-1} = \begin{pmatrix} & & & 0 \\ & \mathbf{K}_{t-1}^{-1} & & \vdots \\ & & & 0 \\ 0 & \ldots & 0 & 0 \end{pmatrix} + \frac{1}{k((s_t, a_t), (s_t, a_t)) - \mathbf{k}_t^T\mathbf{d}^*}\begin{pmatrix} \mathbf{d}^* \\ -1 \end{pmatrix}\begin{pmatrix} \mathbf{d}^{*T} & -1 \end{pmatrix}. \tag{28}$$

Algorithm 1: Memory Efficient RKHS-SARSA (λ)

INPUTS:

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

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

   (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 $j = 1, \ldots, |\mathbb{S}|$
     - $\alpha_i \leftarrow (1 - \eta\xi)\alpha_i - \eta_t err(s_t, a_t, R_t)\gamma\lambda\beta_i$
   - Update $\mathbf{K}_{t-1}^{-1}$ through (28).

2. UNTIL policy update required

5 Experiments And Results

In this section we provide an empirical evaluation of our memory efficient RKHS-SARSA(λ) algorithm. For this algorithm we also let $\xi = 0$. This means that for this algorithm, unlike RBF nets and the full memory RKHS-SARSA(λ), we do not need a regularizer to achieve stability.

5.1 Problems

We ran our algorithm on three different MDPs:

Pole balancing (cart pole) [10] requires the agent to balance a pole hinged atop a cart by sliding the cart along a frictionless track. The only reward received is -1 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 $[\pm 5]$ degrees. Further we cap episode lengths at 1000 time steps.
Mountain car [10] involves driving an underpowered car up a steep hill. 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 did not cap episodes in the mountain car problem. The car was initialized to a standing start (zero velocity) at the bottom of the hill in each episode (the hardest starting position).
Black hole is a continuous "maze"-like world in which the agent must navigate from the top left corner to the bottom right corner. In the center of the world is a black hole which attracts the agent with a force inversely proportional to the agents distance to the black hole. The agent receives reward -1 at each step except success when a reward of 10 is received, and failure (falling into the black hole) when reward -10000 is received. In this domain the agent must select an action $a \in [0, 360]$ corresponding to an angle. The agent then takes a constant sized step in the direction chosen. We did not cap episodes in the black hole MDP.

5.2 Results

The above selection of problems will show our performance in continuous state spaces, also with a large action space in the black hole MDP. For each MDP we compare our algorithms performance to that of tile coding SARSA (λ) and SARSA (λ) with RBF coding by reporting the time per episode. For the mountain car MDP, obviously smaller is better since episodes terminate with success whereas for the cart pole MDP, longer is better since episodes terminate with failure. In the black hole MDP we report average reward per time step during each episode.

Cart Pole Results As can be seen in Figure 1(a) RKHS-SARSA(λ) clearly outperforms both RBF nets and tile coding and learns to indefinitely balance the pole after a very small number of episodes. These comparison methods do not learn to reliable balance the pole during the 100 episodes that are displayed in the plot. Furthermore we see that the memory efficient version is close to as good as the full memory version of RKHS-SARSA(λ) while a version that simply stores the latest 20000 samples deteriorate immediately after the buffer is full and it starts "forgetting". The memory efficient version only stores a total of 30 samples. This means an optimal policy is represented with a very small number of useful parameters.

In Figure 1(c) we compare memory efficient RKHS-SARSA(λ) for different values of λ and clearly see how useful the eligibility trace is.

Mountain Car This is a simpler problem than Cart Pole and all three methods can solve it rather fast. In Figure 1(b) we see that the RBF net has the steepest decline in time needed to complete the task. RKHS-SARSA(λ) starts somewhat slower because it needs to accumulate basis functions to be able to learn the optimal policy. RBF nets and RKHS-SARSA(λ) reached an optimal policy in approximately the same number of episodes while tile coding needed many more. RBF coding showed some continuing instabilities much later on while memory efficient RKHS-SARSA(λ) remained stable. These instabilities were also showing up for the full memory RKHS-SARSA(λ) which we have chosen not to plot. The memory efficient version stores 488 samples. Figure 1(d) compares different values for λ and the performance enhancement that comes from the eligibility trace is clearly visible.

Black Hole The black hole problem is more difficult than the other two since it has such a large action space. We were unable to get reasonable results for the comparison methods with the small number of episodes that we display in Figure 2. Hence, we only report on memory-efficient RKHS-SARSA(λ) for different values of λ. The highest value for which it worked well was $\lambda = 0.7$ and, therefore, we display $\lambda \in \{0, 0.25, 0.5, 0.7\}$. We notice that for the two higher values a successful policy is quickly learnt and maintained while for the two lower values reaching a good policy took longer. Furthermore, late failures for the lower λ values (which indicate that the agent fell into the hole) show that it has not fully learnt to reliably solve the problem. 677 samples are stored when $\lambda = 0.7$ is used. This rather high number can be expected in this kind of problem because the square that is the world needs to covered. It is, however, still a huge improvement over storing every sample.

6 Conclusion

We extend SARSA(λ) temporal difference learning to learn in RKHS. The resulting algorithm, RKHS-SARSA(λ), represents an important extension of SARSA(λ) that substantially reduces the need for feature engineering in the function approximation setting and permits learning of highly nonlinear value functions in a convex optimization framework. Unlike previous kernel versions of SARSA(λ) that were restricted to $\lambda = 0$, we find an intuitive and elegant dual formulation of the eligibility trace permitting efficient online kernel learning for the case of $\lambda > 0$. By borrowing from the Projectron we dramatically reduce the need for memory or regularization. We evaluate RKHS-SARSA(λ) on a continuous RL domain showing that $\lambda > 0$ can yield much more efficient learning than $\lambda = 0$ (emphasizing the importance of $\lambda > 0$ for kernelized RL methods) and that RKHS-SARSA(λ) outperforms traditional function approximation techniques proposed in the literature.

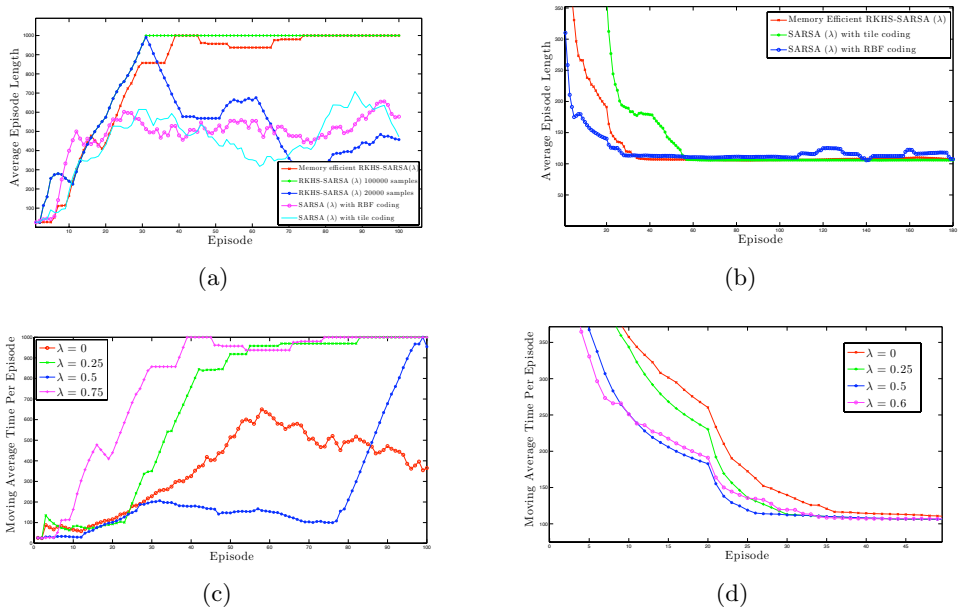

Figure 1: Moving average time per episode with window 10, evaluated for various algorithms at the end of each episode on cart pole 1(a), and mountain car 1(b), and evaluated for our algorithm with various values of λ on the cart pole problem 1(c), and mountain car 1(d).

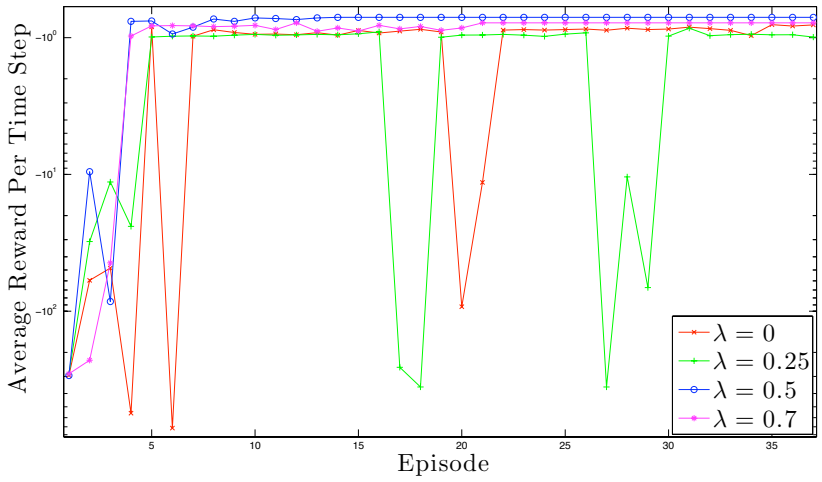

Figure 2: Average reward per time step for memory efficient RKHS-SARSA(λ) on the black hole problem, evaluated over each episode.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, October 2007.

[2] G. S. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Mathematical Analysis and Applications*, 33:82–95, 1971.

[3] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 785–792. MIT Press, 2001.

[4] M. Kretchmar and C. W. Anderson. Comparison of cmacs and radial basis functions for local function approximators in reinforcement learning. In *In International Conference on Neural Networks*, pages 834–837, 1997.

[5] F. Orabona, J. Keshet, and B. Caputo. The projectron: a bounded kernel-based perceptron. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 720–727, New York, NY, USA, 2008. ACM.

[6] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[7] C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16*, pages 751–759. MIT Press, 2003.

[8] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

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

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, 2001.

[11] G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1017–1024, New York, NY, USA, 2009. ACM.

[12] X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

[13] R. M. Y. Engel, S. Mannor. Reinforcement learning with Gaussian processes. In *22nd International Conference on Machine Learning (ICML-05)*, pages 201–208, Bonn, Germany, 2005.