1 Introduction

1.1 Smart Computers!

How do we create intelligent artificial agents who learn through trial and error? What can we learn from Pavlov’s dog, who learned to anticipate treats (rewards)?

Our aim is to create agents, using the field of reinforcement learning, who learn to act in their environment to maximize the reward they receive.

1.2 A Bit of Notation

Before we delve into the details of reinforcement learning, we need to establish a few notations that will be used throughout the text.

We use the tuple $(s, a)$ to represent the agent’s state and action. The state $s$ is the agent’s current environment, and the action $a$ is what the agent chooses to do in that environment.

The agent receives a reward $r$ for taking action $a$ in state $s$. The reward function $r(s, a)$ is defined as:

$$ r(s, a) = r(s) + r(a) $$

where $r(s)$ is the immediate reward for being in state $s$, and $r(a)$ is the immediate reward for taking action $a$.

The agent’s goal is to learn a policy that maximizes the expected cumulative reward over time, given by the formula:

$$ V(s) = E[R_{t+1} | s_t = s] $$

where $R_{t+1}$ is the sum of all future rewards starting from state $s_t$.

1.3 Previous Methods

SARSA(1) is a temporal difference RL algorithm for learning, from experience, the value $Q(s, a)$ of taking action $a$ when occupying state $s$. We use online SARSA(1) and denote by $Q(s, a)$ the action-value estimates at time $t$ assuming the agent takes what he considers the best action at each step. Initializing $Q(s, a) = 0$, SARSA(1) performs the following $Q$-update at time $t + 1$:

$$ Q(s, a) = Q(s, a) + \alpha_t (r(s, a) + \gamma Q(s', a') - Q(s, a)) $$

where $\alpha_t$ is the learning rate.

The feature map $\phi$ is generally hand engineered a way such that the agent can gain maximal understanding his states within the problem. This approach, however, results in poor generalization across domains. The algorithm presented next discusses a smart way to automatically select this feature map.

1.4 SARSA(λ) in the RKHS

To define this algorithm we extend the traditional online gradient descent SARSA(1) update rule to a RKHS setting by using the dual formulation. We slightly extend this update rule to include a regularization term (which will be set to zero for our final algorithm). The primal update is given by:

$$ w_{t+1} = w_t - \alpha_t \frac{1}{1 - \lambda} \sum_{i=1}^{t} \eta_t^i (Q(s, a) - R_i) $$

where $\eta_t$ is the eligibility trace, updated through:

$$ \eta_t = \gamma \lambda \eta_{t-1} + \phi(s, a) $$

and $s_t$ is set to 0 at the beginning of each episode. The representation and $\phi(S \times A \rightarrow H_0)$ maps a state action pair to function in the RKHS. Alternatively we may write the eligibility trace as:

$$ \eta_t = \gamma \lambda \eta_{t-1} + \phi(s, a) $$

where $s_t$ is the time at which the current episode began. Typically such a representation would be undesirable since it requires storing all past samples, however kernelizing our algorithm means storing all previously visited state action pairs anyway. By substituting (5) into (3), we get

$$ w_{t+1} = w_t - \gamma \lambda \alpha_t \sum_{i=1}^{t} \eta_i (Q(s, a) - R_i) $$

and assuming that $w_0 = 0$ we see that $w_t = \sum_{i=1}^{t} \eta_i (Q(s, a) - R_i)$ which leads us to the dual update formulation of (5), $\hat{w} = \sum_{i=1}^{t} \eta_i (Q(s, a) - R_i)$.

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

$$ a_i = 1 - \frac{1}{\eta_i} (1 - \lambda) t - 1 $$

where $s_t$ is the time at which the current episode began.

2 Experiments

For empirical evaluation we use the cart pole and mountain car problems. The cart pole problem requires a hinged pole atop a cart to maintain a vertical position, within some threshold, by sliding the cart along a frictionless track. The agent in this problem receives zero reward at each time step, except upon failure at which time a reward of –1 is received. The state space is represented by four variables: the cart’s position and velocity, and the pole’s angle and angular velocity. The threshold within which the pole must be held is ±12 degrees. At the beginning of each episode we drew the initial pole angle uniformly from [±20] degrees. Further we cap episode lengths at 1000 time steps. In the mountain car problem the agent is required drive an underpowered car up a hill. This requires first rocking back before powering up the hill. Reward –1 is received at each time step until failure where the reward is received. The state is represented by the cart’s position and velocity.

3 Conclusion

We extend SARSA(1) temporal difference learning to learn in RKHS. The resulting algorithm, RKHS-SARSA(1), represents an important extension of SARSA(1) 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(1) that were restricted to $\lambda = 1$, 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 Projection we dramatically reduce the need for memory or regularization. We evaluate RKHS-SARSA(1) on a continuous RL domain showing that $\lambda > 0$ can yield much more efficient learning than $\lambda = 1$ (emphasizing the importance of $\lambda > 0$ for kernelized RL methods) and that RKHS-SARSA(1) outperforms traditional function approximation techniques proposed in the literature.