Recognizers: A study in learning how to model temporally extended behaviors

Jordan Frank
jordan.frank@cs.mcgill.ca
McGill University
Montreal, QC, Canada

Doina Precup
dprecup@cs.mcgill.ca
McGill University
Montreal, QC, Canada

1 Introduction

Using a hierarchy of behaviors often requires learning models of these behaviors in an efficient way from one stream of experience. In this paper we are concerned with computing the reward model of a behavior, when this model has to be represented using function approximation. Off-policy learning methods have been proposed for this goal, but their efficiency when using function approximation has been limited (Precup, Sutton & Dasgupta, 2001). One of the sources of this problem is the fact that existing off-policy methods rely on importance sampling corrections to estimate models for different behaviors from the same stream of experience. These corrections can have very high variance if the policy generating the behavior is very different from the behavior that we want to model. Moreover, the behavior policy has to be known and fixed, otherwise these corrections are not well defined. In order to address these problems, Precup et al. (2006) introduced the notion of *recognizers*. Rather than specifying an explicit policy for a behavior about which we want to make predictions, a recognizer specifies a condition on the actions that are selected. For example, a recognizer for the temporally extended action of picking up a cup would not specify which hand is to be used, or what the motion should be at all different positions of the cup. The recognizer would recognize a whole variety of directions of motion and poses as part of picking the cup. The advantage of this approach is that the behavior may be based on a variety of different strategies, all of which are relevant, and we would like to learn from any of them. In general, a recognizer is a function that recognizes or accepts a range of different policies. Recognizers have two advantages over direct specification of a target policy: 1) they are a natural and easy way to specify a target policy for which importance sampling will be well conditioned, and 2) they do not require the behavior policy to be known. The latter is important because in many cases we may have little knowledge of the behavior policy, or a stationary behavior policy may not even exist. In Precup et al.(2006), the authors show that if the model is represented using state aggregation, even if the behavior policy is unknown, convergence to a good model is achieved. Here, we generalize this to the case of general linear function approximation, and show that this approach works very well empirically. We also discuss how recognizers might be learned from data.

2 Recognizers

We use the standard framework in which an agent interacts with a stochastic environment. At each time step t, the agent receives a state s_t, chooses an action a_t, obtains a numerical reward r_{t+1} and transitions stochastically to a new state s_{t+1}. Assume for the moment that actions are selected according to a fixed behavior policy, $b : \mathcal{S} \times \mathcal{A} \to [0,1]$ where $b(s,a)$ is the probability of selecting action a in state s. The behavior policy is used to generate a sequence of experience (observations, actions and rewards). The goal is to learn, from this data, predictions about different ways of behaving. We assume that the state space is large or continuous, and function approximation must be used to compute any values of interest. In particular, we assume a space of feature vectors Φ and a mapping $\phi : \mathcal{S} \to \Phi$. We denote by ϕ_s the feature vector associated with state s.

An option (Sutton, Precup & Singh, 1999) is defined as a triple $o = \langle I, \pi, \beta \rangle$ where $I \subseteq S$ is the set of states in which the option can be initiated, π is the internal policy of the option and $\beta : S \to [0,1]$ is a stochastic termination condition. In the options work (Sutton, Precup & Singh, 1999), each of these elements has to be explicitly specified and fixed in order for an option to be well defined. Here, we will instead define options implicitly, using the notion of a recognizer.

A recognizer is defined as a function $c : S \times \mathcal{A} \to [0,1]$, where $c(s,a)$ indicates to what extent the recognizer allows action a in state s. An important special case (the only one we addressed so far) is that of binary recognizers. In this case, c is an indicator function, specifying a subset of actions that are allowed, or recognized, given a particular state. Note that recognizers do not specify policies; instead, they merely give restrictions on the policies that are allowed or recognized.

A recognizer c together with a behavior policy b generates a *target policy* π, where:

$$\pi(s,a) = \frac{b(s,a)c(s,a)}{\sum_x b(s,x)c(s,x)} = \frac{b(s,a)c(s,a)}{\mu(s)} \tag{1}$$

The denominator of this fraction, $\mu(s) = \sum_x b(s,x)c(s,x)$, is the *recognition probability* at s, i.e., the probability that an action will be accepted at s when behavior is generated according to b. The policy π is only defined at states for which $\mu(s) > 0$. The numerator gives the probability that action a is produced by the behavior and recognized in s. Note that if the recognizer accepts all state-action pairs, i.e. $c(s,a) = 1, \forall s, a$, then π is the same as b.

Since a recognizer and a behavior policy can specify together a target policy, we can use recognizers as a way to specify policies for options, using (1). An option can only be initiated at a state for which at least one action is recognized, so $\mu(s) > 0, \forall s \in I$. Similarly, the termination condition of such an option, β, is defined as $\beta(s) = 1$ if $\mu(s) = 0$. In other words, the option must terminate if no actions are recognized at a given state. At all other states, β can be defined between 0 and 1 as desired.

In this paper, we focus on computing the reward model of an option o, i.e., the expected sum of rewards obtained while executing the option:

$$E_o\{R(s)\} = E\{r_1 + r_2 + \ldots + r_T | s_0 = s, \pi, \beta\}$$

where $s \in I$, experience is generated according to the policy of the option, π, and T denotes the random variable representing the time step at which the option terminates according to β. We assume that linear function approximation is used to represent these values, i.e.

$$E_o\{R(s)\} \approx \theta^T \phi_s$$

where θ is a vector of parameters.

The algorithm for learning such models, presented in Precup et al (2006), is based on using importance sampling weights at each step of the trajectory, to correct for the difference between the behavior policy b and the target policy π. Because of the way in which π is defined, the importance sampling correction can be re-written as:

$$\rho(s,a) = \frac{\pi(s,a)}{b(s,a)} = \frac{c(s,a)}{\mu(s)}$$

Of course, $\mu(s)$ depends on b. If b is unknown, instead of $\mu(s)$, we will use a maximum likelihood estimate $\hat{\mu} : S \to [0,1]$. The structure used to compute $\hat{\mu}$ will have to be compatible with the feature space used to represent the reward model. More precisely, in the

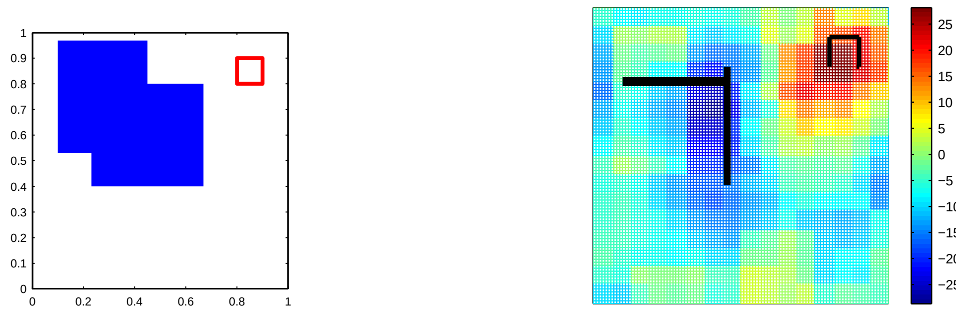

Figure 1: Puddleworld environment (left) and learned reward model (right)

case of linear function approximation, we can use logistic regression, with the same feature mapping ϕ, in order to estimate μ.

On every time step t, the learning algorithm performs the following updates:

Compute recognition probability: $\mu(s_t) = \dfrac{1}{1+e^{-\mathbf{w}_t^T \phi_{s_t}}}$

Compute importance sampling correction: $\rho(s_t, a_t) = c(s_t, a_t)/\mu(s_t)$

Compute TD-error: $\delta_t = \rho_t (r_{t+1} + (1-\beta_{t+1}\theta_t^T \phi_{s_{t+1}}) - \theta_t^T \phi_{s_t}$

Update reward model parameters: $\theta_{t+1} = \theta_t + \alpha \delta_t e_t$

Update eligibility trace increment, based on restarting: $k_{t+1} = \rho_t k_t (1-\beta_{t+1}) + g_{t+1}$

Update eligibility trace: $e_{t+1} = \lambda \rho_t (1-\beta_{t+1}) e_t + k_{t+1} \phi_{s_{t+1}}$

Update recognition probability parameters: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha'(c(s_t, a_t) - \mu(s_t))\phi_{s_t}$

Here, we also use a restart function g in order to prevent the eligibility traces from decaying to 0 too quickly (see (Precup, Sutton & Dasgupta, 2001) for details).

3 Illustration

In order to illustrate this approach we use the puddle world environment from the RL-Glue library (University of Alberta). The environment is presented in Figure 1. Stepping in the puddle generates a negative reward which goes from 0 at the edges to -100 at the center of the puddle. When the agent is in the goal state, it is given a reward of 100. Otherwise, if the agent is not in the puddle or the goal region, it receives a reward of -1. The task is continuing, and does not have a terminal state, and so the agent may remain in the goal state for a number of time steps, generating a reward of 100 at each step. The dimension of the world is 1 in each direction and there are 16 possible actions, moving the agent a distance of 0.05 in one of 16 directions. After an action, the agent is also moved in a random direction by an amount drawn from a normal distribution with mean 0 and standard deviation 0.01.

We use standard tile coding to provide state features. We use four overlapping tilings, each of dimension 4×4. The behaviour policy is uniformly random. We have a deterministic, binary recognizer that recognizes all actions that move the agent up and to the right, which would generally move the agent towards the goal region from most states. Since four actions are recognized, and the actions are generated uniformly randomly, the recognition probability should be exactly 0.25. The values of the parameters are $\lambda = 0.6$, $\beta = g_0 = 0.05$, $\alpha = 0.001$, $\alpha' = 0.005$. We use a decreasing schedule for the learning rates, where we halve the learning rates after time steps $T, 2T, 4T, 8T, 16T, 32T, 64T$, and $128T$, where $T = 100,000$. The experiment runs for $256T$ time steps.

The reward model for the recognizer is shown in Figure 1. Figure 2 shows the recogni-

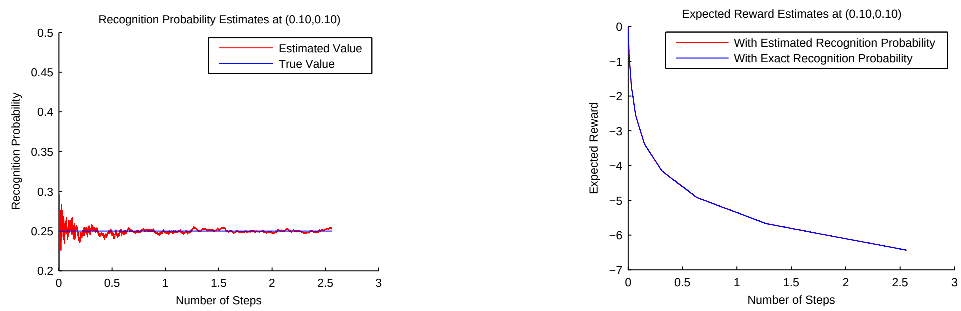

Figure 2: Recognition probabilities at a given state in the PuddleWorld experiment (left). Reward estimates for the same state (right)

tion probabilities and reward model for a particular given state. As seen in the figure, the recognition probabilities are learned correctly very quickly. More importantly, the imprecise recognition probabilities have almost no effect on learning the reward estimates: the reward estimate using the exact recognition probability is nearly identical to the reward estimate using the learned recognition probability.

These results are consistent over different states and different behavior policies. We have also experimented with this method in the Ship steering task (also included in the RL repository). The results are almost identical to those described here, and are omitted for the moment due to lack of space.

4 Recognizer improvement

So far, we have assumed that both the recognition function and the termination condition have to be specified in advance. However, these can be learned from data as well. In order to learn the recognition function for good recognizers, we can use the action elimination approach proposed by Even-Dar et al (2003). Learning the termination probabilities is more complicated. The basic idea we have been experimenting so far is tuning these in order to keep the importance sampling ratios well behaved. Hence, trajectories are cut when ratios that are too large or too small are encountered.

References

Even-Dar, E., Mannor, S. & Mansour, Y. (2003). “Action Elimination and Stopping Conditions for Reinforcement Learning”. In *Proceedings of the 20th International Conference on Machine Learning*, pp. 162–169. Morgan Kaufmann.

Precup, D., Sutton, R.S. & Dasgupta, S. (2001) “Off-policy temporal-difference learning with function approximation ”. In *Proceedings of the 18th International Conference on Machine Learning*, pp.417–424. Morgan Kaufmann.

Precup, D., Sutton, R.S., Paduraru, C., Koop, A. & Singh, S. (2006). “Off-policy Learning with Options and Recognizers”. In *Advances in Neural Information Processing Systems 18*, pp. 1097–1104. MIT Press.

Sutton, R.S., Precup D.& Singh, S. (1999). “Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In *Artificial Intelligence*, vol. 112, pp.181–211.