Eligibility Traces for Options

Ayush Jain

April 22, 2017

Abstract

In this work, I investigate the utility of eligibility traces with temporally abstract actions in MDP setup.

1 Introduction

Many problems in machine learning can be framed as a temporal credit assignment problem. Using the classical reinforcement learning setup, it corresponds to an agent determining a relation between its interactions with the environment (actions) and the feedback signals (rewards). In general, this can be an extremely difficult problem as reward signals can be sparse and their associated actions temporally distant. A huge number of problems fall under this generic problem setup, making it a very interesting general problem. In adition to speeding up covergence in such cases, *eligibity traces* have been shown to make reinforcement learning more robust to hidden states. Traces also provide a link between Monte Carlo and Temporal Difference methods.

Being able to learn, plan and represent knowledge at multiple levels of behavior is a cornerstone of AI. Humans regularly make decisions based on high level behavior over a broad range of time and it only seems natural to incorporate temporally extended courses of actions into reinforcement learning algorithms. Exploration of temporal abstraction in AI started as early as 1970's. Problem of temporal abstraction is how to connect the higher level of behavior with the lower level in the best possible way. The primary aim is to extend reinforcement learning to temporally abstract actions while making minimum changes to the existing framework and maximizing the generality in how rewards and all courses of actions are expressed. Methods like MAXQ in [1], HAM in [2] have been proposed for temporal abstraction but *Options* as introduced in [3] is the most generic and intuitive framework for this.

In RL setup, learning is usually based on experience - the states, actions taken in those states and the observed reward. The experience is generated using a behavior policy and if this experience is used directly, we end up learning something which is a function of this behavior policy. However, we might

be interested in learning about policies other than one followed by the agent - a process called *Off-Policy Learning*. This is especially important for use of temporal abstraction in reinforcement learning - it might be of interest to learn about many different policies, each corresponding to a different temporally extended action.

In this work, the authors propose off-policy eligibility traces for intra-option learning [4] methods.

2 Options

Hierarchical reinforcement learning is one of the key concepts of AI. Temporally extended actions have been shown to generate shorter plans, reduce complexity of choosing actions, increase robustness againt mis-specified models, and improve exploration. The options [3] framework provides a concrete way to incorporate temporal abstraction into reinforcement learning with no change to the existing setup. The term 'options' is a generalization for primitive as well as temporally extended actions.

An option $\mathcal{O}$ is a triple comprising intiation set $\mathcal{I} \subseteq \mathcal{S}$, stationary internal policy π which defines probability distibution over all possible primitive actions from a given state, and termination function β defining the probability of terminating in a given state.

An option is 'feasible' in a state s iff s $\in \mathcal{I}$. Once an option o is selected, actions are taken according to internal policy π until termination as per β. $\mathcal{I}$ and β limit an option's applicability to a part of state space $\mathcal{S}$ and hence policy π needs to be defined for only that section of the state space.

As is clear from the structure of an option, actions can be considered a special case of options - *primitive options*. Each primitive option is available whenever corresponding action is available, it lasts exactly one step and selects the particular action w.p. 1. Given a set of options, their intiation sets define a set of available options $\mathcal{O}_s$ $\forall s \in \mathcal{S}$. A hierarchical Markov policy μ defines a probability distribution over $\mathcal{O}_s$. An option o is selected according to $\mu(s_t,)$, determining action until termination in s_{t+k} as per $\beta(s_{t+k})$, when a new option is selected according to $\mu(s_{t+k},)$.

3 Intra-Option Learning

SMDP learning methods can be applied at high level behavior and be used to learn with temporal abstraction. However, these methods treat options as black boxes - an algorithm based on SMDP methods would execute an option to completion, accumulating rewards at each step and then at termination would make one single update to the executing option. An interesting idea is to take advantage of each fragment of the experience.

If an option is Markov, in some sense, it is intiated at every time step.

By looking inside options, more efficient Intra-Option methods [4] have been devised. For markov options, after each time step experience can be used to update all $\mathcal{O}_s$ options that could take same action. Intra-Option methods are off-policy learning methods because updates can be applied to all relevant options while behaving according to a particular option. A single step intra-option update can be -

$$\begin{aligned} Q(s_t, o) \leftarrow Q(s_t, o) + \alpha[r_{t+1} + \gamma(1 - \beta_o(s_t))Q(s_{t+1}, o) \\ + \gamma\beta_o(s_{t+1}) \max_{o' \in \mathcal{O}} Q(s_{t+1}, o') - Q(s_t, o)] \end{aligned} \tag{1}$$

4 DYNA Planning

DYNA architecture integrates -

1. Direct RL
2. Model Learning
3. Planning: using learned model to quicken learning using simulations
4. Reactive Execution: Selecting actions for real observations

Main idea is to try out actions in different states using modeled enviroment knowledge and use these hipothetical experiences to speed up learning.

5 Eligibility Traces for Options

Eligibity traces are one of the basic mechanisms of reinforcement learning. TD methods are essentially 1-step algorithms which bootstraps from the value of subsequent states. Monte Carlo methods on the other hand wait for the exact return at the end of an episode. TD(λ) methods, i.e. TD augmented with eligibility traces, produce a family of methods spanning a spectrum that has Monte Carlo on one end and 1-step TD at another. In this sense, eligibity traces interpolate between TD and Monte Carlo methods.

A more mechanistic way of seeing eligibility traces, called *backward view*, is to see these traces as a short term fading memory of occurrence of events. The trace marks the parameters associated with an event with it's *eligibility* for updates. When an update is to be made, only eligible states get cedit or blame for the error weighed according to their eligibility. Eligibility of events decreases as they get temporally far.

As explained earlier, 1-step intra-option methods are essentially off-policy methods. To extend options with eligibility traces, off-policy nature of the algorithm is important if updates are to be at intra-option level.

5.1 Per Decision Importance Sampling

The primary method to do off-policy learning is to correct the expectation of the update from behavior policy to target policy - like classical Importance Sampling technique. [5] proposed an IS motivated approach for eligibility trace for off-policy evaluation.

5.2 Tree Backup

Previously explained method has two shortcomings - firstly, it requires the behavior policy to be known, and secondly, if the target policy being learnt is too different from executing behavior policy, PDIS approach has high variance. Motivation behind Tree backup, as proposed in [5], is to overcome these two limitations.

Main idea behind this method is to use expectation across all branches of the backup diagram instead of sampling. At each step along the trajectory, there are several possible choices of actions according to target policy. Tree backup algorithm forms target values combining value estimates of actions not taken and the new estimate of value of the action taken, each weighed by corresponding action's probability under target policy. Authors extend this idea to the case of options.

In current setup of options, an option o is selected according to a hierarchical policy over available options $\mathcal{O}_s$ which determines actions until termination as per $\beta_o(s)$. Active option o_t depends on -

1. current state s_t
2. if previously active option o_{t-1} terminated in s_t or not

Folding in the termination conditions with the hierarchical probability distribution over options, a new hierarchical policy over options μ can be written as -

$$\mu(o|s_t, o_{t-1}) = \begin{array}{ll} \beta_{o_{t-1}}(s_t)h(o|s_t) & \text{if } o \neq o_{t-1} \\ (1-\beta_{o_{t-1}}) + \beta_{o_{t-1}}(s_t)h(o_{t-1}|s_t) & \text{otherwise} \end{array} \tag{2}$$

Target value for an update to option-action values can formed as proposed in the tree backup algorithm. Assuming following as a trajectory using options:

$$s_t, o_t, a_t, r_t, s_{t+1}, o_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}, \ldots$$

1-step return for an option $o \in \mathcal{O}_{s_t}$ feasible with action a_t can be written as:

$$\begin{aligned} G_t^1 = r(s_t, a_t) &+ \gamma \sum_{o' \neq o_{t+1}} \mu(o'|s_{t+1}, o) \sum_a \pi_{o'}(a|s_{t+1}) Q(s_{t+1}, o', a) \\ &+ \gamma \mu(o_{t+1}|s_{t+1}, o) \sum_a \pi_{o_{t+1}}(a|s_{t+1}) Q(s_{t+1}, o_{t+1}, a) \end{aligned} \tag{3}$$

In Eq. 3, second term is the expected return from all non-realised options and third term is the new estimate of expected return from trajectory of the realised option. Proceeding in a similar way, a 2-step return with aditional step at t+2 can be written as Eq. 4

$$\begin{aligned}
G_t^2 = r(s_t, a_t) &+ \gamma \sum_{o' \neq o_{t+1}} \mu(o'|s_{t+1}, o) \sum_a \pi_{o'}(a|s_{t+1}) Q(s_{t+1}, o', a) \\
&+ \gamma \mu(o_{t+1}|s_{t+1}, o) \sum_{a \neq a_{t+1}} \pi_{o_{t+1}}(a|s_{t+1}) Q(s_{t+1}, o_{t+1}, a) \\
&+ \gamma^2 \mu(o_{t+1}|s_{t+1}, o) \pi_{o_{t+1}}(a_{t+1}|s_{t+1}) \sum_{o'' \neq o_{t+2}} \mu(o''|s_{t+2}, o_{t+1}) \sum_a \pi_{o''}(a|s_{t+2}) Q(s_{t+2}, o'', a) \\
&+ \gamma^2 \mu(o_{t+1}|s_{t+1}, o) \pi_{o_{t+1}}(a_{t+1}|s_{t+1}) \mu(o_{t+2}|s_{t+2}, o_{t+1}) \sum_a \pi_{o_{t+2}}(a|s_{t+2}) Q(s_{t+2}, o_{t+2}, a)
\end{aligned} \tag{4}$$

Further steps can be derived in a similar fashion. λ-Return is compound target combining all the n-step return targets, each weighed by λ^{n-1}.

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^n \tag{5}$$

TD error using the G_t^λ for option o can hence be written as:

$$\begin{aligned}
\Delta Q(s_t, o, a_t) &= G_t^\lambda - Q(s_t, o, a_t) \\
&= [r(s_t, a_t) + \gamma \sum_{o'} \mu(o'|s_{t+1}, o) \sum_a \pi_{o'}(a|s_{t+1}) Q(s_{t+1}, o', a) \\
&\quad - Q(s_t, o, a_t)] \\
&\quad + \lambda\gamma \mu(o_{t+1}|s_{t+1}, o) \pi_{o_{t+1}}(a_{t+1}|s_{t+1}) \quad [r(s_{t+1}, a_{t+1}) \\
&\quad + \gamma \sum_{o''} \mu(o''|s_{t+2}, o_{t+1}) \sum_a \pi_{o''}(a|s_{t+2}) Q(s_{t+2}, o'', a) - Q(s_{t+1}, o, a_{t+1})] \\
&\quad + \dots
\end{aligned} \tag{6}$$

Eq. 6 can be generalised as:

$$\Delta Q(s_t, o, a_t) = \sum_{i=t}^{\infty} \delta_i \prod_{j=t+1}^{i} \lambda\gamma \mu(o_j|s_j, o_{j-1}) \pi_{o_j}(a_j|s_j) \tag{7}$$

Backward view equivalent to Eq. 7 can be implemented as:

$$e_t(s, o, a) = \begin{cases} \lambda\gamma \mu(o_{t-1}|s_{t-1}, o_{t-2}) \pi_{o_{t-1}}(a_{t-1}|s_{t-1}) + 1 & \text{if } s, o, a = s_{t-1}, o_{t-1}, a_{t-1} \\ \lambda\gamma \mu(o_{t-1}|s_{t-1}, o_{t-2}) \pi_{o_{t-1}}(a_{t-1}|s_{t-1}) & \text{otherwise} \end{cases} \tag{8}$$

Param	ETD(λ) Value	TD(λ) Value
α	5×10^{-5}	5×10^{-4}
γ	0.9	0.9
λ	0.3	0.2

Table 1: Parameter Values

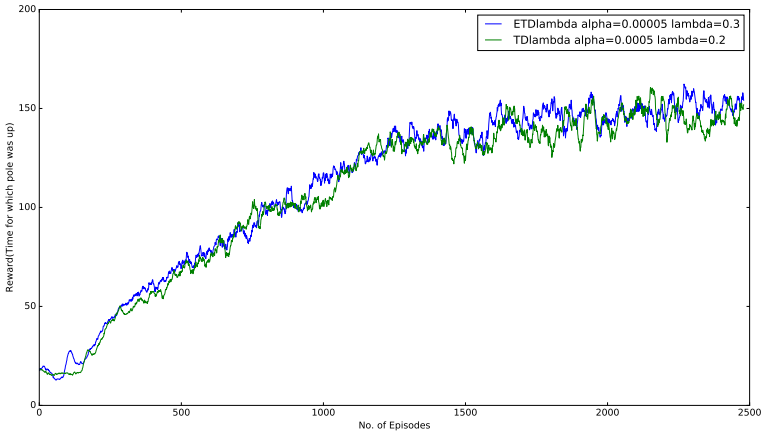

Figure 1: Carpole experiment comparing ETD(λ) with TD(λ)

6 Experiment

6.1 Different Elligibility Traces

Very first experiments I performed were to test feasibility of different tracing mechanisms, as explained in [6], under RL setting. Averaging method didn't perform well but bootstraping method gave decent results in Cartpole environment.

Working backwards and adapting Bootstraping method for reinforcement learning naturally lead to [7]'s Emphatic TD. Experiments comparing effectiveness of emphatic trace and accumulative trace in Miner and Cartpole environments show both methods converge to similar values with former performing better as can be seen in fig 1. For the cartpole experiment I used linear function approximation to model action-state pairs. Observations were tile-coded to get a feature vector of length 1000. A grid search on parameters was conducted to get the best performing values. Table 6.1 specifies the final values of parameters used.

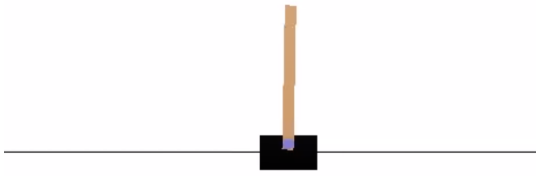

Figure 2: The cartpole domain. Cart with an inverted pendulum moves on a frictionless surface controlled by applying force of +1/-1. Aim is to learn a policy to keep pole upright for maximum timesteps.

6.1.1 Cartpole Domain

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pole starts upright, and the goal is to prevent it from falling over. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. Agent controlling this cartpole gets a reward of zero for every timestep that the pole remains upright except at episode termination when agent gets a reward of -100. See fig 2.

6.2 Temporally extended actions with Eligibility Traces

This section explains my experiment to learn option models and State-option-action pairs. DYNA architecture is used in the experiments to integrate learning and planning steps. Actions are reactively chosen according to a behaviour policy. Corrected TD(λ) updates are applied to state-action values of all feasible options from last observed state. Observed reward and next state is used to learn model of the environment in a Intra-Option manner. For planning stage, environment is reset to an arbitrary state and a random option is executed till option-termination. Values from learned model are used to get expected reward and next feature vector. These values are used to update selected option's Q values. This experiment is performed on a gridworld problem from [7] called *Miner problem.*

6.2.1 Miner Gridworld

It is a gridworld problem as seen in fig 3. The miner starts from state S and continually takes one of actions: *up, down, left, and right.* Invalid actions result in no movement. The miner gets zero reward for every action except when it arrives at Goal state where a reward of +1 is received.There are two different routes to Goal state from S - via block D or roundabout via block B. A trap can be active in block D with activation probability of 0.25. When activated, a

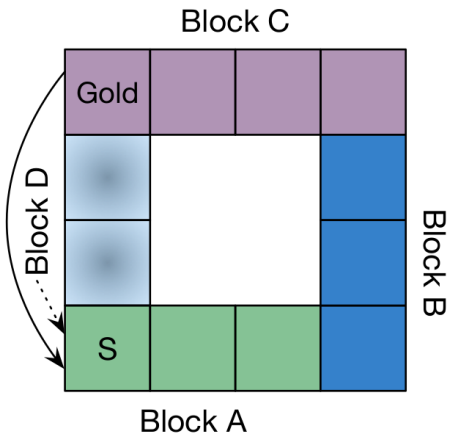

Figure 3: The Miner problem where the miner continually collects gold from Goal State until it falls into a trap.

trap remains there for 3 timesteps and only one is active at once. When miner arrives at goal state or falls into a trap, it is transported back to S. It is not an episodic task and miner wanders on continually even after being transported back to S.

In my experiment miner follows a fixed *behavior policy* according to which it takes any of the four actions in Block A with equal probability, is more inclined to go up in both Block B and Block D, and more inclined to go left in Block C, in each case with probability 0.4. The rest of the actions are equally likely.

I learn about three different options. *Uniform option*, where all actions are equally likely. Under *Cautious option*, miner is more inclined to go right in Block A, go up in both Block B and Block D, and go left in Block C, in each case with probability 0.6. The rest of the actions are equally likely. In *Headfirst option*, miner chooses to go up in Block A and D with 0.9 probability while other actions from those blocks were equally likely. All the actions from other blocks have equal probabilities.

I used linear function approximators for the environment models and state-action pairs for the options. Miner's current state is expressed as a feature vector of length 12, each bit represents one of the state. Values of the parameters used are shown in the table 6.2.1

6.2.2 Taxi

In the taxi domain, an agent navigates on a grid with immovable objects in episodic manner. Figure 4 shows a 5X5 grid environment for the agent. The taxi agent has to transport passengers from and to some locations predefined

Param	Value
α	10^{-4}
γ	0.6
λ	0.7

Table 2: Parameter Values for miner gridworld

Param	Value
α_{model}	10^{-2}
α_q	10^{-4}
γ	0.9
λ	0.8

Table 3: Parameter Values used for taxi environment found after grid search over parameters.

by the environment in this girdworld - *Red, Green, Yellow and Blue*. In each episode, taxi starts in a random location with a passenger waiting randomly at one of these four locations and wishing to be dropped-off at one of the four locations chosen stocastically. The taxi agent can only move in one of the four cardinal directions, pickup the passenger when in same location as the passenger and drop off the passenger. Every episode ends with the passenger dropped off at the desired location.

There is a reward of -1 for every timestep. Aim is to seek a policy which maximises the total reward at the end of an episode. There are 500 different possible states the agent can be in - 25 locations on grid, 5 location of passenger and 4 drop-off spots.

In the tabular case, a feature vector of 500 length gives the probability on being in a state. Values of the parameters used are shown in the table 6.2.2

Figure 5 shows the performance of the Tree Backup for Options againts a Naive PDIS algorithm. In the naive PDIS approach, ρ is set as ratio of policy of option being updated and active option's policy.

7 Discussion

As seen in the figure 5, there is very small difference in performance of Tree Backup approach with different λ values. The fact that all the options in my implementations are too different and internal policies of the options are very steep, traces in Tree Backup approach are cutoff prematurely. It can also be noted that Naive PDIS algorithm has obvious flaws, more recent work is on fixing this issue.

SS 

Figure 4: 5X5 gridworld of Taxi environment.

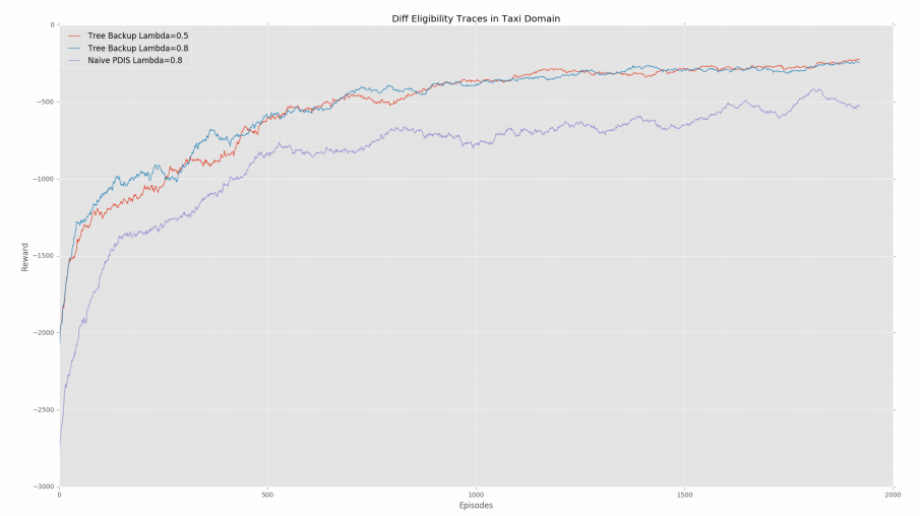

Figure 5: Comparison of Tree Backup algorithm with different λ values and a Naive PDIS algorithm in Taxi environment.

References

[1] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *J. Artif. Intell. Res.(JAIR)*, vol. 13, pp. 227–303, 2000.

[2] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *Advances in neural information processing systems*, pp. 1043–1049, 1998.

[3] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181 – 211, 1999.

[4] R. S. Sutton and D. Precup, "Intra-option learning about temporally abstract actions," in *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 556–564, Morgan Kaufman, 1998.

[5] D. Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.

[6] H. van Hasselt and R. S. Sutton, "Learning to predict independent of span," *CoRR*, vol. abs/1508.04582, 2015.

[7] A. R. Mahmood, H. Yu, M. White, and R. S. Sutton, "Emphatic temporal-difference learning," *CoRR*, vol. abs/1507.01569, 2015.