Bayesian Reinforcement Learning

Aman Taxali
Department of Statistic
University of Michigan
ataxali@umich.edu

Yung-Chun Lee
Department of Statistic
University of Michigan
yungclee@umich.edu

Abstract

1 Introduction

2 Methodology

2.1 Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework for sequential decision-making in systems where the outcome is partly under the control of an external decision maker and partly random. This idea was first proposed by Bellman in 1957, and has since been applied in many domains like robotics, economics and manufacturing.

An MDP is an extension of Markov chains known as discrete time stochastic control process. At each time-step, the process is in some state and a decision maker is required to choose an action. Based on the action, the process moves to a new state and returns to the decision maker the reward corresponding to the movement. Now, a new time-step begins, and based on the process these steps are either repeated until a terminal state is reached or indefinitely.

Model (Markov Decision Process) Define an MDP $\mathcal{M}$ to be a 5-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $P_a(s,s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t results in transition to state s' at time $t+1$
- $R_a(s,s')$ is the reward received after transitioning from state s to s' as a result of action a
- $\gamma \in [0,1]$ is a discount factor that determines the exponential devaluation rate of future rewards.

For an MDP, $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, an action policy π is a function that maps each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$, defined as $\pi : \mathcal{S} \to \mathcal{A}$. The main problem of MDPs is to find the policy π such that the cumulative reward are maximized over a potentially infinite horizon.

For a given policy π, we define the value of a state s, denoted $V^\pi(s)$, as the expected sum of discounted rewards when the agent starts in the state s and follows policy π thereafter, for an infinite horizon.

$$V^\pi(s) = \boldsymbol{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \middle| s_0 = s \right]$$

where $\boldsymbol{E}_\pi$ is the expected value under policy π (1)

A similar state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be defined as the expected sum of discounted rewards when the agent starts in state s and takes action a, and follows policy π thereafter.

$$Q^\pi(s,a) = \boldsymbol{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \middle| s_0 = s, a_t = a \right] \tag{2}$$

The goal for solving an MDP is to find the optimal action policy such that value function in equation (1) is maximized for all states $s \in \mathcal{S}$. So, the optimal policy is such that $V^*(s) \geq V^\pi(s)$ for all $s \in \mathcal{S}$ and all policies π.

Using recursive properties, equation (1) can be written as what is known as the *Bellman Equation* (Bellman, 1957):

$$\begin{aligned} V^\pi(s) &= \boldsymbol{E}_\pi \left[R_a(s_0, s_1) + \gamma V^\pi(s_{t+1}) \middle| s_0 = s \right] \\ &= \sum_{s'} P_{\pi(s)}(s,s') \Bigg(R_a(s,s') + \gamma V^\pi(s') \Bigg) \end{aligned} \tag{3}$$

It can be proven that the solution for the optimal policy V^*, called the *Bellman Optimality Equation*, satisfies:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_a(s,s') \Bigg(R_a(s,s') + \gamma V^*(s') \Bigg) \tag{4}$$

Bellman Optimality Equation (4) states that value of a state under an optimal policy equals the expected discounted reward for the best action in that state. This means the *greedy* optimal action given the optimal state value function V^* is:

$$\pi^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}} P_a(s,s') \Bigg(R_a(s,s') + \gamma V^*(s') \Bigg) \tag{5}$$

The greedy policy is important because it is the optimal policy with respect to V^*. The analogous optimal state-action value function is:

$$Q^*(s,a) = \sum_{s'} P_a(s,s') \Bigg(R_a(s,s') + \gamma \max_{a'} Q^*(s',a') \Bigg) \tag{6}$$

The Q-value functions (2), (6) are useful in cases when the P and R functions are unknown. Instead of forward reasoning, the Q-value functions compute the weighted summation over different alternatives.

The relationship between Q^* and V^* is:

$$V^*(s) = \max_a Q^*(s,a) \tag{7}$$

And equation (5) can be written as:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{8}$$

2.2 Solving MDPs: Model-based Methods

In situations where a (perfect) model of the MDP is available, and the state space is manageable in size, the optimal policy of the MDP can be solved iteratively using *dynamic* or *linear programming*. Two core model-based methods are *policy iteration* (Howard, 1960) and *value iteration* (Bellman, 1957). However, in this report we will focus on the alternative *model-free* methods.

2.3 Solving MDPs: Model-free Methods

Reinforcement learning (RL) attempts to solve the MDP problem in which an agent (or action taker) interacts with a dynamic and incompletely known environment; with the goal of finding an action policy that optimizes the rewards in the long-term. RL is a *model-free* method because it does not rely on prior knowledge of the MDP's transition and reward functions. RL also adds to the MDP problem the topic of incomplete information and approximation, and thus the need for sampling and exploration. This is especially useful when either the state space is too large, the system is only available as a simulator or no system model is available at all.

Within model-free RL, there are two choices for modeling the optimal action policy. The first learns the transition and reward functions from the interaction with the environment. Once the learned model of the environment is sufficiently correct, dynamic programming methods can be applied. This is called *indirect RL*. The second method attempts to directly estimate value for actions, without estimating the underlying model of the MDP. This is called *direct RL*. It is worth noting that mixed forms between these two methods also exist.

Model-free RL methods fall under the general class of algorithms that interact with an environment one time-step at a time, and update their estimates after each experience. This is called *online learning*, and Algorithm 1 provides a general template for it.

Algorithm 1: *Online Reinforcement Learning*

```
foreach time-step do
    $s \in \mathcal{S}$ is initialized as the starting state
    $t := 0$
    repeat
        choose an action $a \in \mathcal{A}(s)$
        perform action $a$
        observe the new state $s'$ and received
          reward $r$
        update $\tilde{P}, \tilde{R}, \tilde{Q}$ and/or $\tilde{V}$ using the
          experience $\langle s, a, r, s' \rangle$
        $s := s'$
    until end of training time
```

2.4 Bayesian Reinforcement Learning

2.5 Bayesian Sparse Sampling

In 2005, Wang et al suggest a approximation of optimal Bayesian selection for reinforcement learning by combining the Model-Based Bayesian approach and Non-bayesian approach from Dearden et al in 1999 and Kearns et al in 2002, respectively.

Dearden et al advocate a Bayesian approach to model-based reinforcement learning, showing that under fairly reasonable assumption. This approach is able to represent the posterior distribution over possible given past experience. The cost for this approach is claimed to be the same as maintaining a point estimate while is able to keep updating throughout the interaction between agent and the system.

On the other side, the non-bayesian approach, Kearns et al propesed a general *sparse sampling* method for realistic situation where the complexity of the planning scales with the size of an MDP; which contains a large number or infinite number of states s. In the paper, it also showed that for *any* given belief state MDP it is guaranteed to find a near-optimal action.

Algorithm 2: *Sparse Bayesian Tree*

```
GrowSparseBayesianTree (node, budget, $\rho$, horizon):
    while #nodes <budget do
        branchnode = BayesDescend(root, $\rho$)
        if branchnode.type = decision then
            Add outcome then leaf node below branchnode
        if branchnode.type = outcome then
            Add leaf decision node below branchnode
    return EvaluateTree(root)
BayesDesecnd (node, $\rho$):
    if node.type = decision then
        $a$ = ThompsanSample(node.belstate)
        if $a \notin$ node.childeren then
            return [node,a]
        else
            return BayesDescendTree(node.child(a))
    if node.type = outcome then
        $a$ = ThompsanSample(node.belstate)
        if possible to branch, with probability $\rho$% new branch then
            return node
        else
            [rew,obs] = sample(node.belstate,node.act)
            return BayesDescendTree(node.child([rew,obs]))
algorithm adapted from "Bayesian Sparse Sampling for On-line Reward Optimization", Wang et al. 2005.
```

In the Algorithm 2 shows the outline of the Sparse Bayesian tree, which the algorithm incorporate a prior distribution in the descending step using the Thompson sampling.

Algorithm 3: *Thompson Sampling*

```
ThompsonSampling ($P_{prior}$):
    $P_{post} := P_{prior}$
    foreach t do
        sample $\hat{\theta}$ from $P_{post}$
        do action $a_t = \arg\max_{a \in \mathcal{A}} \boldsymbol{E}_{y \sim P_a(s,\cdot)}[R(y)]$
        observe outcome and update $P_{post}$
```

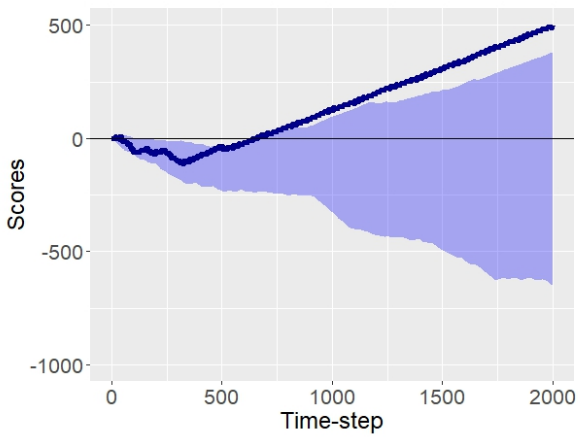

Figure 1: Bayes Optimal

2.6 Thompson Sampling

2.7 Bootstrap

3 Experimental results

a;sdiofha;sdjn;fasd

Time	Min	Mean	Max	St.Dev	Pct. Win
100	-99.1	-32.50	10.4	26.47	13.79
500	-355.1	-114.09	-7.0	75.62	0.00
1000	-491.4	-139.74	187.0	135.28	17.24
1500	-639.8	-160.49	307.5	218.16	20.69
2000	-856.7	-186.07	499.9	313.00	20.69

Table 1: Bayes Optimal

Time	Min	Mean	Max	St.Dev	Pct. Win
100	-89.2	-49.75	-9.4	20.19	0.00
500	-256.5	-166.60	-67.6	43.22	0.00
1000	-454.6	-246.16	-65.3	98.10	0.00
1500	-870.9	-300.92	-44.0	212.01	0.00
2000	-1285.0	-397.60	67.8	335.25	3.33

Table 2: Pruned

Acknowledgments

The acknowledgments should go immediately before the references. Do not number the acknowledgments section. Do not include this section when submitting your paper for review.

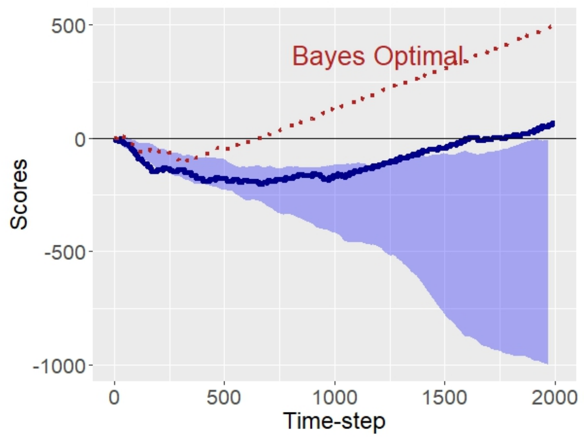

Figure 2: Train Pruned

References

Richard Bellman. 1957. *A Markovian Decision Process*, volume 6, pages 679–684 *Indiana University Mathematics Journal*

Martijn van Otterlo. 2009. *Markov Decision Processes: Concepts and Algorithms SIKS course on Learning and Reasoning*

Martin L. Puterman. 1957. *Markov Decision Processes*. *Wiley Interscience*

Michael Kearns, Yishay Mansour, and Andrew Y, NG. 2002. *A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes*, 49, 193-208 *Machine Learning*

Richard Dearden, Nir Friedman, and David Andre. 2002. *Model-Based Bayesian Exploration*. *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*