Lecture notes for CS747: Foundations of Intelligent and Learning Agents [Autumn 2018]

Shivaram Kalyanakrishnan
scribed by Sabyasachi Ghosh

November 9, 2018

Contents

1 The Markov Decision Problem (MDP)

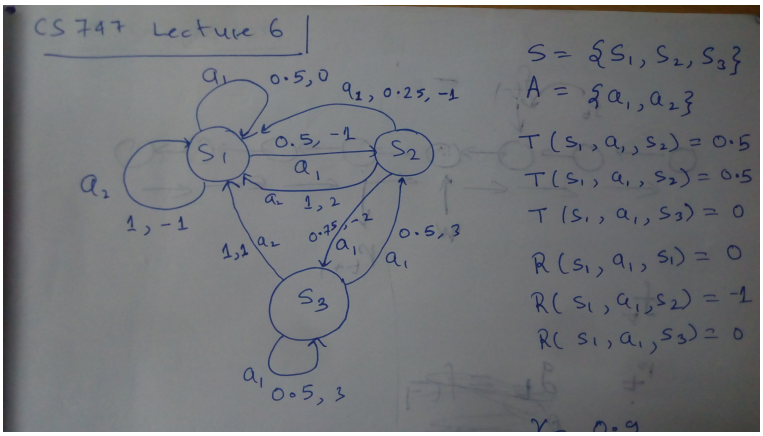
In a Markov Decision Problem, an agent is in an environment which has a notion of states and actions. At any given time, the agent is in a particular state and must decide which action to take. Depending on the action taken, the agent will transition to one of many other states with probabilities depending on the environment. It may even remain in the same state. On each such transition, the agent receives a reward. The goal of the agent is to take a sequence of actions which will maximize its expected long-term reward, with rewards in the future discounted by some factor. The problem is Markovian because the next state and the reward depend only on the current state and the current action taken.

Formally, a Markov Decision Problem is a tuple (S, A, R, T, γ) where

- S is a set of states
- A is a set of actions
- $T : S \times A \rightarrow$ the set of probability distributions over S is the transition function
- $R : S \times A \times S \rightarrow [-R_{max}, R_{max}]$ is the reward function. $R_{max} \in \mathbb{R}$ is a constant bounding the reward values.
- $\gamma \in [0, 1]$ is the reward discount factor

An example of a MDP is shown in 1.1

Figure 1.1: An example of a MDP



Even the game of chess can be formulated as an MDP. Each configuration of the chess board is a state. Each possible move of a player is an action. Since there are two players,

the next state for the first player is defined as the configuration of the board after the second player has made its move. The transition probabilities for the first player depends on the strategy of the second player and vice-versa. The reward function can be defined to be 1 for going to a win state and -1 for a loss. For going to any other state, a reward of 0 is given.

Note the difference between an MDP and a Markov Chain. In a Markov chain, there is no notion of an agent making decisions.

1.1 MDP Policy

Given its current state, the agent must decide which action to take in order to maximize its expected long-term reward. Since the MDP itself is Markovian, hence the action it takes should depend only on its current state and not on any previous states or actions it might have taken. We define a MDP policy to be a mapping from the set of states to the set of actions ($\pi : S \to A$). Note that a policy defined in this way is deterministic. We shall soon see that a stochastic policy will not be needed to maximize expected long-term reward.

1.1.1 Trajectory of a Policy

Suppose the agent starts at the state s^0 and follows policy π. Its trajectory is given by the sequence

$$s^0 a^0 r^0 s^1 a^1 r^1 s^2 \cdots \tag{1.1}$$

where

- $a^t = \pi(s^t)$
- $s^{t+1} = T(s^t, a^t)$
- $R^t = R(s^t, a^t, s^{t+1})$

Note that since the reward and the transition functions are stochastic, the agent may follow different trajectories in different runs even if it starts from the same initial state s^0 and follows the same policy π.

1.1.2 The Value Function of a Policy

The long-term reward of the trajectory in 1.1 is:

$$r^0 + \gamma r^1 + \gamma^2 r^2 + \cdots \tag{1.2}$$

under the discount factor γ. This is the **infinite discounted reward**. The discount factor determines how important future rewards are. The expectation of this quantity gives the value of the state s^0 under policy π. In general, a value function can be defined, $V : S \to \mathbb{R}$ such that

$$V^\pi(s) = \mathbb{E}[r^0 + \gamma r^1 + \gamma^2 r^2 + \cdots | s^0 = s] \tag{1.3}$$

The value function gives us a way to choose between policies. Ideally, the agent would like to choose that policy which maximizes its long-term reward given a start state. In fact, it is possible to do more - for every MDP there exists an optimal policy π^* whose value function dominates the value function of every other policy. That is,

$$V^{\pi^*}(s) \geq V^{\pi}(s) \forall s \in S$$

Note that there can be multiple optimal policies for an MDP, but their value functions are identical, and called the optimal value function V^*.

Table 1.1 lists the value functions for all policies for the MDP in Figure 1.1. Can you spot the optimal policy? Note as well that there exist some policies which are not comparable to each other. For example, for $\pi_1 = a_2a_1a_1$ and $\pi_2 = a_2a_2a_2$, $V^{\pi_1}(s_2) < V^{\pi_2}(s_2)$, but $V^{\pi_1}(s_3) > V^{\pi_2}(s_3)$.

Table 1.1: Value functions for all policies for the MDP in Figure 1.1

$\pi(s_1, s_2, s_3)$	$V^{\pi}(s_1)$	$V^{\pi}(s_2)$	$V^{\pi}(s_3)$
$a_1a_1a_1$	4.45	6.55	10.82
$a_1a_1a_2$	-5.61	-5.75	-4.05
$a_1a_2a_1$	2.76	4.48	9.12
$a_1a_2a_2$	2.76	4.48	3.48
$a_2a_1a_1$	10	9.34	13.10
$a_2a_1a_2$	10	7.25	10
$a_2a_2a_1$	10	11	14.45
$a_2a_2a_2$	10	11	10

1.2 Continuing and Episodic Tasks

In chess there are only a finite number of states. Also, if the same state is reached thrice, then the game is called a draw. Hence all chess games either end in one of the players winning, or a draw. Hence it is an example of an **episodic task**.

In MDPs which are episodic tasks, two conditions must be fulfilled:

1. There exists a terminal state
2. With probability 1, the agent will reach a terminal state under *every* policy.

Continuing tasks are those which do not have a terminal state. Although most real-world processes eventually terminate, it is useful to model long-running processes as continuing tasks. Examples of these may be events in long-running factories, high frequency stock trading etc.

1.3 MDP Planning

In MDP planning, the full specification of the MDP is known, and the task is to find the optimal policy. This is as opposed to a learning problem in which reward and transition functions of the MDP may not be known. Multi-armed bandit is a learning problem.

1.3.1 Bellman's Equations

$$V^\pi(s) = \mathbb{E}_\pi[r^0 + \gamma r^1 + \gamma^2 r^2 + \cdots | s^0 = s] \tag{1.4}$$

By linearity of expectations, this becomes

$$V^\pi(s) = \mathbb{E}_\pi[r^0 | s^0 = s] + \gamma \mathbb{E}_\pi[r^1 + \gamma r^2 + \cdots | s^0 = s] \tag{1.5}$$

Expanding the expectation over all possible values of the next state s^1, we get

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') R(s, \pi(s), s') + \gamma \sum_{s' \in S} T(s, \pi(s), s') \mathbb{E}_\pi[r^1 + \gamma r^2 + \cdots | s^0 = s, s^1 = s'] \tag{1.6}$$

Note that in the second term in the above equation, we can remove the condition $s^0 = s$ because of the Markovian property (rewards $r^1, r^2 \cdots$ do not depend on the state s^0 once s^1 is specified). We also note that the expectation term inside the sum then becomes $V^\pi(s')$ by the definition of the value function. Hence,

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') \{R(s, \pi(s), s') + \gamma V^\pi(s')\} \tag{1.7}$$

1.7 gives $|S|$ linear equations in as many unknowns. These are known as Bellman's Equations. If $0 \leq \gamma < 1$, then it has a unique solution[1].

1.3.2 Policy Evaluation

Finding $V^\pi(s) \forall s \in S$ is called policy evaluation. This may be done by solving Bellman's Equations.

Is there a way to find the optimal policy given we can do policy evaluation? A naive way will be to evaluate all policies, and then choose the policy which dominates all others.

1.3.3 The Action Value Function

There exist more efficient ways of finding the optimal policy than brute-force evaluation of all policies. To help with such algorithms, we will need to define the action value function, $Q^\pi : S \times A \to \mathbb{R}$

$$Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') \{R(s, a, s') + \gamma V^\pi(s')\} \tag{1.8}$$

That is, $Q^\pi(s, a)$ gives the value of taking the action a from the state s once, and thereafter following policy π.

[1] These equations are of the form $Ax = b$ where A is a stochastic matrix

1.3.4 Bellman's Optimality Equations

Under the optimal policy π^*, the agent must take that action a from a state s whose value $Q^*(s,a)$ is maximum when following π^* amongst all actions in A. Hence the optimal value function V^* must satisfy the following equation:

$$\forall s \in S,\ V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V^*(s')\} \tag{1.9}$$

These are Bellman's Optimality Equations. Unlike Bellman's equations, one cannot directly solve them. These equations are not linear because of the 'max' operation.

1.3.5 Finding the optimal policy from the optimal value function

If one has the optimal policy, the optimal value function may easily be found using policy evaluation. On the other hand, in order to find the optimal policy from the optimal value function, one must find $Q^{\pi^*}(s,a) \forall s \in S, a \in A$ and find the action maximizing $Q^{\pi^*}(s,a)$ for each state.

2 Reinforcement Learning

In the reinforcement learning setup, there is an agent in an environment. The environment is an MDP. However, the MDP is not fully known to the agent. The states, actions and the reward discount factor are known, but at least one of the transition function and the reward function is not known. The agent must learn these through interactions with the environment. Thus it follows some trajectory

$$s^0 a^0 r^0 s^1 a^1 r^1 s^2 \cdots \tag{2.1}$$

The question is, can the agent "eventually" learn to take optimal actions? The short answer is yes. If each (s, a) pair is taken a large number of times, then the reward and transition functions can be estimated empirically. There is an assumption here that $\forall s_1, s_2 \in S, \forall \pi \in \Pi$, the probability of reaching s_2 starting from s_1 is strictly positive. In other words, the **Markov Chain induced by every policy π is irreducible**.

2.1 Challenges in Reinforcement Learning

What are the general challenges that we expect in trying to solve the problem of reinforcement learning?

2.1.1 Exploration

How to optimally explore states? For example, if we know that our estimates for the variables for certain states and actions are not good enough, then is it better to take a sequence of actions to get to those states and then explore that state? We will not deal with questions such as this in this course.

2.1.2 Generalizing over states and actions

n and k might be too large. For example, in a game of chess, the number of states is 8^{64}. In tennis and soccer, the number of states and actions are uncountably infinite. Hence the states and the actions may need to be approximated by a manageable number of parameters. We will cover this in some detail in future chapters.

2.2 Prediction

Prediction is the problem of finding the value function V^π given a policy π for an MDP S, A, R, T, γ. The key is that at least one of R and T are not known.

In order to make our job easy, we are going to assume the following:

- The task is episodic
- The agent is allowed to do exploring starts, i.e., it can start at any state s and take any action a, and thereafter follow π

Although these assumptions aren't necessary for the prediction algorithms we'll describe, but they make it easier to analyse them.

As a first-cut, one might think of the following to estimate the value function: start at a state s, follow π till the end of the episode, and find the discounted long-term reward for this episode. Repeat this many times starting at the same state s.

Now we have n trajectories as:

$$\begin{gathered} s_1^0 a_1^0 r_1^0 s_1^1 a_1^1 r_1^1 s_1^2 \cdots s_1^{T_1} \\ s_2^0 a_2^0 r_2^0 s_2^1 a_2^1 r_2^1 s_2^2 \cdots s_2^{T_2} \\ \vdots \\ s_n^0 a_n^0 r_n^0 s_n^1 a_n^1 r_n^1 s_n^2 \cdots s_n^{T_n} \end{gathered}$$

For the i^{th} trajectory, the cumulative discounted reward is:

$$R_i = r_i^0 + \gamma r_i^1 + \gamma^2 r_i^2 + \cdots$$

By definition of V^π,

$$V^\pi(s) = \mathbb{E}[R_i]$$

We can approximate it with the empirical estimate:

$$\widehat{V}^\pi(s) = \mathbb{E}[R_i]$$

We can similarly estimate $Q^\pi(s, a)$ by have $s_i^0 = s, a_i^0 = a$ and thereafter following π. Then we get the empirical estimate,

$$\widehat{Q}^\pi(s, a) = \mathbb{E}[R_i]$$

2.3 Control

The problem of Control is to find an optimal policy for the given MDP. Again, R and T may not be fully known. To begin with, we will give a control algorithm which works in practice:

Algorithm 1: Monte Carlo ES (exploring starts)

$Q[\,][\,] \leftarrow 0$;
$\pi \leftarrow$ Arbitrary Policy;
$Visits[\,][\,] \leftarrow 0$;
for *each episode* **do**
  pick $s \in S$ u.a.r. ;
  pick $a \in A$ u.a.r. ;
  Start at s, take a, then follow π ;
  Let the return for the trajectory be R ;
  $Q[s][a] \leftarrow \frac{Q[s][a]*Visits[s][a]+R}{Visits[s][a]+1}$;
  $Visits[s][a]$ += 1 ;
  $\pi \leftarrow Greedy(Q)$;
end
Greedy(Q) is defined as: $\forall s, \pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\, Q(s,a)$;

In Algorithm 1, if we updated Q only after an infinite number of episodes while following a policy and then set $\pi \leftarrow Greedy(Q)$, then we would be doing policy improvement at each step. Hence this algorithm will be equivalent to doing policy iteration. Hence intuitively, we can see why the above algorithm might work. However, it is not known whether this algorithm converges to π^* or not. In practice it does converge to π^*.

Note that this algorithm cannot converge to a suboptimal policy. $Greedy(Q^\pi) \succ \pi$ by Policy improvement theorem for a suboptimal policy π.

However, if we use an ϵ-greedy policy, then we do converge to the optimal ϵ-greedy policy. Note as well that the optimal ϵ-greedy policy (with a particular value of ϵ) need not be the same as the optimal policy made ϵ-greedy because the distribution of state visits may be very different.

It is also possible to modify this algorithm and update Q values for all (s, a) pairs which are visited for the first time in an episode.

Bibliography