CSE250A Fall '12: Discussion Week 9

Aditya Menon (akmenon@ucsd.edu)

December 4, 2012

1 Schedule for today

- Recap of Markov Decision Processes.
- Examples: slot machines and maze traversal.
- Planning and learning.

2 Recap of Markov Decision Processes

A Markov Decision Process (MDP) is defined by four elements:

- A state space $\mathcal{S}$,
- An action space $\mathcal{A}$,
- A transition probability function $\Pr[s'|s,a]$, where $s, s' \in \mathcal{S}, a \in \mathcal{A}$,
- A reward function $R(s', s, a)$ where $s, s' \in \mathcal{S}, a \in \mathcal{A}$

An MDP represents models the behaviour of an *environment* over time. At each time step, the environment possesses a state in $\mathcal{S}$. From the environment's point of view, the *actions* are the result of an external entity, known as the *agent*. Each action affects the state of the environment, and endows the agent with a *reward*. The aim of the agent is to perform actions that maximize the long-term reward.

Formally, the interaction between the agent and environment is as follows.

- Initially, the agent finds herself at state $s_0 \in \mathcal{S}$.
- for time steps $t = 0 \dots T$:
  - The agent performs an action $a_t \in \mathcal{A}$.
  - The environment places the agent in the new state s_{t+1} by performing a random draw from the distribution $\Pr[s'|s_t, a_t]$.
  - The environment rewards the agent by an amount $R(s_{t+1}, s_t, a_t)$

The number of time steps T is called the horizon. It is mathematically convenient to assume an infinite horizon[1], $T = \infty$. The figure below shows an illustration of this interaction.

Above, we did not specify how the agent computes the action a_t when she is in state s_t. This is precisely the problem we wish to solve: we want to figure out a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ that tells us how to act given the state

[1] If a finite horizon is genuinely needed, we can imagine the end-cell as having only self-transitions with zero reward.

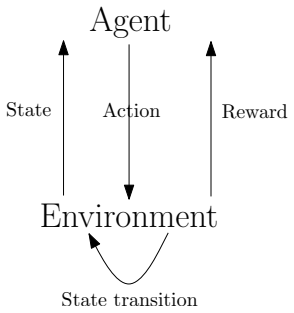

Figure 1: The interaction between agent and environment in an MDP.

we are in. To determine a policy, we need to have a goal in mind. One reasonable goal is to maximize the discounted expected reward, where the expectation is over all possible future states.

$$\text{ExpReward}(\pi) = \mathbb{E}_{s_0, s_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, s_t, \pi(s_t))\right].$$

Here, we assume $s_0 \sim \mu(s)$, some prior probability distribution over the initial state of the MDP, and $\gamma \in (0, 1)$ is the discount factor. Each $s_i \neq s_0$ is drawn from $\Pr[s' | s_{i-1}, \pi(s_{i-1})]$.

Let Π denote the set of all functions from $\mathcal{S}$ to $\mathcal{A}$. Note that if $\mathcal{S}, \mathcal{A}$ are finite, then $|\Pi| = |\mathcal{A}|^{|\mathcal{S}|}$. We define the optimal policy to be

$$\pi^* = \text{argmax}_{\pi \in \Pi} \text{ExpReward}(\pi).$$

3 Examples of MDPs

3.1 Slot machines

Suppose we (the agent) are in a casino (the environment), and there are K slot machines. We are free to play with any of the machines, in any order. Each play of a machine is independent from the previous ones from any other machine (including the current one). Each machine i gives us a payoff v_i with probability p_i. As a Markov decision process:

- The state space $\mathcal{S} = \{0\}$; there is no really relevant concept of state for the environment, because we assume that playing one machine has no impact on playing any other machine in the future.
- The action space $\mathcal{A} = \{1, 2, \ldots, K\}$ denotes the slot machine we decide to play with.
- The transition function $T(s', s, a)$ is trivially just 1 everywhere, as there is one state.
- The reward function R depends only on the action, and $R(a) \sim \begin{cases} v_a & \text{, probability } p_a \\ 0 & \text{, probability } 1 - p_a \end{cases}$.

Our goal is to learn a policy[2] $\pi : \mathcal{S} \to \mathcal{A}$. As there is only one state, we are just trying to determine which slot machine to play. We assume that our goal is to maximize the discounted expected return, which in this case is

$$\text{ExpReward}(\pi) = \frac{1}{1-\gamma}\mathbb{E}\left[R(\pi(0))\right].$$

That is, we want to pick the slot machine which gives maximum expected reward.

[2]This problem is also known as the K-armed bandit.

3.2 Traversing a maze

Suppose we have an $N \times N$ grid with designated start and end cells. The agent controls an entity that exists on a single cell of the grid. The environment controls a destructive flood that also exists on a single cell. At each time step, we assume that the agent takes an action wherein she can move in one of the four directions. We assume that the environment also places a flood on a random cell. The agent wishes to go from the start cell to the end cell, without encountering a flooded cell. The states of the system are:

- The state space $\mathcal{S} = \{(x_1, y_1, x_2, y_2) \in [N]^4\}$, representing the (x, y) coordinates of the agent and the flooded cell in the N dimensional game grid.
- The action space $\mathcal{A} = \{U, D, L, R\}$ denotes the direction that the agent decides to move.
- The transition function $T(s', s, a)$ specifies the new position for the agent, as well as the new position for the flooded cell. The former can be assumed to be largely deterministic based on the action that the agent chosen, but with some probability goes in the opposite direction to what is desired: for example, maybe the agent is disoriented, so travels in the wrong direction by mistake. The latter reflects the new position of the flood. We'll assume that flooding just arises from a force of nature, which has some distribution over the cells.
- The reward function R is, say, -100 if the agent is on a flooded cell, and 100 if the agent is on the end cell, and -1 otherwise.

4 Planning and learning

Now we consider how the agent might learn an optimal policy. This depends on how much information is available about the environment.

4.1 Case I: Full Information

Consider for concreteness the maze traversal problem. Suppose that the agent knows both the transition and reward function. The latter is reasonable: the agent knows that it incurs a penalty if it collides with a flooded cell, but is otherwise free to roam. The former is questionable: it is not clear that the agent knows *a-priori* the rules that govern how floods arise. At any rate, in this setting, the agent can learn an optimal policy by *policy iteration*, as follows.

1. Pick a random initial policy $\pi^{(0)}$. Note that a policy has N^4 entries, one for each possible combination of agent and flood position, telling the agent how to move given these positions. For example, the agent may decide on a (not very good!) policy that says to always go down.
2. Compute the expected reward for the agent under this policy. This averages over all possible sequences of positions of the agent and flooded cell, and for each computes the discounted reward that the agent receives. Note that the policy $\pi^{(0)}$ affects how likely different states are, and hence the value of this expected reward: if the agent always decides to go down, for example, then they may never reach the end cell, so that state will have zero probability.
3. Compute a refined policy $\pi^{(1)}$ as follows: for each state, pick the action a which maximizes the expected reward if we follow a for our next action, and then follow $\pi^{(0)}$ for all future actions. Essentially, we evaluate which direction is the best for each possible state, based on the expected return, and use this as our new policy.
4. Repeat steps (2)-(3) until convergence.

Note that the agent does all of this without actually making a physical action: it basically *simulates* the mechanics of a path traversal, because it already knows everything about the underlying MDP. Once it does this calculation, it can interact with the environment optimally.

Step 2 is called *policy evaluation* and Step 3 *policy improvement*. Step 4 involves testing convergence. We elaborate on the steps in turn.

4.1.1 Step 2: Policy evaluation

Assuming an infinite horizon, and a reward function that depends only on the current state,

$$\begin{aligned}\text{ExpReward}(\pi) &= \mathbb{E}_{s_0,s_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\big|\pi\right] \\ &= \sum_s \Pr[S_0 = s]\cdot\mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\big|s_0 = s;\pi\right] \\ &:= \sum_s \Pr[S_0 = s]\cdot V(s;\pi).\end{aligned}$$

The function V computes the expected reward from following policy π, given that we start in state s. It is called the *state value function*. How do we compute it? For each initial state s, observe that

$$\begin{aligned}V(s;\pi) &= \mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)\big|s_0 = s;\pi\right] \\ &= R(s) + \mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=1}^{\infty}\gamma^t R(s_t)\big|s_0 = s;\pi\right] \\ &= R(s) + \gamma\mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=1}^{\infty}\gamma^{t-1} R(s_t)\big|s_0 = s;\pi\right] \\ &= R(s) + \gamma\sum_{s'}\Pr[S_1 = s'|S_0 = s,\pi(s)]\cdot\mathbb{E}_{s_2,s_3,\ldots}\left[\sum_{t=1}^{\infty}\gamma^{t-1} R(s_t)\big|s_1 = s';\pi\right] \\ &= R(s) + \gamma\sum_{s'}\Pr[S_1 = s'|S_0 = s,\pi(s)]\cdot\mathbb{E}_{\tilde{s}_1,\tilde{s}_2,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t R(\tilde{s}_t)\big|\tilde{s}_1 = s';\pi\right] \text{ where } \tilde{s}_t := s_{t-1} \\ &=: R(s) + \gamma\sum_{s'}\Pr[s'|s,\pi(s)]\cdot V(s';\pi).\end{aligned}$$

This evaluates the expected utility if we begin from state s and keep following the policy π. The set of equations are known as the *Bellman equations*, and it admits a simple closed form solution:

$$\begin{aligned}V &= R + \gamma PV \\ \implies (I-\gamma P)V &= R \\ \implies V &= (I-\gamma P)^{-1}R,\end{aligned}$$

where P is the matrix with entries $\Pr[s'|s,\pi(s)]$. Note that we can compute this solution because we have full knowledge about the system, i.e. because we know the matrix P and the reward R. In practice we don't actually have to do the inverse, and can instead solve the system of linear equations implied by $(I-\gamma P)V = R$. Further, the inverse of $I-\gamma P$ exists because P is a stochastic matrix, and so has eigenvalues of maximum magnitude 1 by the Perron-Frobenius theorem, which are further scaled down by $\gamma < 1$.

4.1.2 Step 3: Policy improvement

If we write our current policy as $\pi^{(t)}$, policy improvement involves simply finding the action a that yields maximum reward if we perform it on state s now, and then follow the current policy:

$$\begin{aligned}\pi^{(t+1)}(s) &= \text{argmax}_a \left[R(s,a) + \gamma \sum_{s'} \Pr[s'|s,a] \cdot V(s'; \pi^{(t)}) \right] \\ &:= \text{argmax}_a \, Q(s,a;\pi^{(t)}).\end{aligned}$$

The $Q(s,a;\pi)$ function encodes the expected reward if we start from state s, follow action a, and then follow the policy π. It is called the *state-action value function*. Observe that by definition, at any iteration t,

$$V(s;\pi^{(t)}) = Q(s, \pi^{(t)}(s); \pi^{(t)}).$$

4.1.3 Step 4: Convergence of policy iteration

The condition for policy iteration to terminate is that we are unable to further improve the value of our policy. That is, we have

$$V(s;\pi^{(t+1)}) = V(s;\pi^{(t)}) := V^*(s).$$

Note that there may be many policies π for which $V(s;\pi) = V^*(s)$. When we talk of an optimal policy, we generally mean any policy for which the state value function is maximized.

We can derive an important property the optimal state value function by using the fact that $\pi^{(t+1)}$ is greedily derived from $\pi^{(t)}$:

$$\begin{aligned}V^*(s &= V(s;\pi^{(t+1)}) \\ &= R(s) + \gamma \left[\sum_{s'} \Pr[s'|s, \pi^{(t+1)}(s)] \cdot V(s;\pi^{(t+1)}) \right] \\ &= R(s) + \gamma \left[\sum_{s'} \Pr[s'|s, \pi^{(t+1)}(s)] \cdot V(s;\pi^{(t)}) \right] \\ &= R(s) + \gamma \max_a \left[\sum_{s'} \Pr[s'|s,a] \cdot V(s;\pi^{(t)}) \right].\end{aligned}$$

This is similar, but distinct to the Bellman equation we derived earlier. This equation is only satisfied by the optimal state value function. If we could solve this equation directly, we could compute an optimal policy's value; but the system is nonlinear due to the $\max$ operator. The above set of nonlinear equations are known as the *Bellman optimality equations*. (This criterion may be used to compute the optimal policy with a different strategy, *value iteration*, that we do not discuss here.)

4.2 Case II: Partial Information

Suppose now that the agent doesn't know the transition or reward function. In this more realistic setting, we have to either estimate the parameters of the MDP and use these to derive a policy, or somehow implicitly build up knowledge of the environment to construct a policy. Options include TD-learning, Q-learning and LSPI. LSPI builds upon a method called LSTD, which attempts approximate the state value function using a linear representation. This is useful when the state space is large and/or continuous. Specifically, we assume that

$$V(s) = w^T \phi(s) + \Delta(s),$$

where $\phi(s)$ is some vector representation of the state s, w is a set of weights, and $\Delta(s)$ is an error term. We'd like to pick w so as to get a good approximation to V, or equivalently that the elements of $\Delta(s)$ are small in

some sense. Recall that V was defined by the Bellman equation,

$$V = R + \gamma PV.$$

Trivially, this can be thought of as the *fixed point* of the function

$$f(v) = R + \gamma Pv,$$

where v is a vector. So, there are at least two possible criteria for picking a good w, each potentially yielding a different solution:

- Find the fixed point of f. Then, pick w to yield the best approximation to this fixed point.
- Construct a function $\tilde{f}$ that mimics f, but only yields outputs that are of the form Φw. Then, find the fixed point of this $\tilde{f}$, and extract the corresponding weight w.

It turns out that the latter has better performance, so we will focus on that. We will consider the function

$$f(v) = R + \gamma Pv$$

where v is constrained to be of the form Φw for some w. Does this equation have a fixed point? In general, no, because we need

$$v = R + \gamma Pv \implies \Phi w = R + \gamma P \Phi w,$$

and the latter linear system may be overdetermined in general. So, analyzing this particular function seems futile. Intuitively, the problem is that the left hand side, Φw, necessarily returns a vector that is a linear combination of the columns of Φ, i.e. an element of the columnspace of Φ. The RHS is not guaranteed to lie in this space, and so an exact solution is impossible. What if we *force* the values of f to lie in this columnspace? Then, we can guarantee that there exists a fixed point solution. In particular, consider

$$\tilde{f}(v) = \Phi(\Phi^T\Phi)^\dagger\Phi^T(R + \gamma Pv),$$

where as before v is constrained to be of the form $v = \Phi w$. Here, $X^\dagger$ denotes the pseudo-inverse of X. If the columns of Φ are linearly independent, it turns out that the pseudo-inverse of $\Phi^T\Phi$ equals the inverse. We'll assume this is so from hereon in. It turns out that we can guarantee a solution to the new fixed point equation

$$v = \Phi(\Phi^T\Phi)^{-1}\Phi^T(R + \gamma Pv) \implies \Phi w = \Phi(\Phi^T\Phi)^{-1}\Phi^T(R + \gamma P\Phi w).$$

So, we need to solve

$$\Phi w = \Phi(\Phi^T\Phi)^{-1}\Phi^T(R + \gamma P\Phi w).$$

First, note that by left-multiplying both sides by Φ^T, we get

$$(\Phi^T\Phi)w = \Phi^T(R + \gamma P\Phi w).$$

Rearranging, we get

$$w = (\Phi^T(I - \gamma P)\Phi)^{-1}\Phi^T R.$$

Thus, we can just use the above as our weights, and use this to approximate the value function as

$$\hat{V} = \Phi(\Phi^T(I - \gamma P)\Phi)^{-1}\Phi^T R.$$

Of course, this doesn't really address the fact that we don't know the transition function, and must estimate it from data. The details of this are in the lecture notes.