About this class

Markov Decision Processes

The Bellman Equation

Dynamic Programming for finding value functions and optimal policies

Basic Framework

[Most of this lecture from Sutton & Barto]

The world still evolves over time. We still describe it with certain state variables. These variables exist at each time period. For now we'll assume that they are observable. The big change now will be that the agent's actions affect the world. The agent is trying to optimize reward received over time (think back to the lecture on utility).

Agent/environment distinction − anything that the agent doesn't directly and arbitrarily control is in the environment.

States, Actions and Rewards define the whole problem. Plus the Markov assumption.

We'll usually see two different types of reward structures — big reward at the end, or "flow" rewards as time goes on.

We're going to deal with two different kinds of problems: episodic and continuing.

The reward the agent tries to optimize for an episodic task can just be the sum of individual rewards over time.

The reward the agent tries to optimize for a continuing task must be discounted.

The MDP and it's partially observable cousin the POMDP, are the standard representation for many problems in control, economics, robotics, etc.

MDPs: Mathematical Structure

What do we need to know?

Transition probabilities (now dependent on actions!)

$$P^a_{ss'} = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

Expected rewards

$$R^a_{ss'} = E[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$$

Rewards are sometimes associated with states and sometimes with (State, Action) pairs.

Note: we lose distribution information about rewards in this formulation.

Policies and Value Functions

A policy is a mapping from (State, Action) pairs to probabilities.

$\pi(s, a)$ = prob. of taking action a in state s

States have values under policies.

$$V^\pi(s) = E_\pi[R_t | s_t = s]$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s]$$

It is also sometimes useful to define an action-value function:

$$Q^\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a]$$

Note that in this definition we fix the current action, and *then* follow policy π

Finding the value function for a policy:

$$V^\pi(s) = E_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s]$$

$$= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s]]$$

$$= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

This is the Bellman equation for V^π

An Example: Gridworld

Actions: L,R,U,D

If you try to move off the grid you don't go anywhere.

The top left and bottom right corners are absorbing states.

The task is episodic and undiscounted. Each transition earns a reward of -1, except that you're finished when you enter an absorbing state

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

A			
			A

What is the value function of the policy π that takes each action equiprobably in each state?

Optimal Policies

One policy is better than another if it's expected return is greater across all states. An optimal policy is one that is better than or equal to all other policies.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman optimality equation: the value of a state under an optimal policy must equal the expected return of taking the best action from that state.

$$V^*(s) = \max_{a} E[r_{t+1} + \gamma V^*(s')|a_t = a]$$

$$= \max_{a} \sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s'))$$

Given the optimal value function, it is easy to compute the actions that implement the optimal policy. V^* allows you to solve the problem greedily!

Dynamic Programming

How do we solve for the optimal value function? We turn the Bellman equations into update rules that converge.

Keep in mind: we must know model dynamics perfectly for these methods to be correct.

Two key cogs:

1. Policy evaluation

2. Policy improvement

Policy Evaluation

How do we derive the value function for any policy, leave alone an optimal one?

If you think about it,

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

is a system of linear equations.

We use an iterative solution method. The Bellman equation tells us there is a solution, and it turns out that solution will be the fixed point of an iterative method that operates as follows:

1. Initialize $V(s) \leftarrow 0$ for all s

2. Repeat until convergence

   (a) For all states s

i. $v \leftarrow V(s)$

ii. $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

Actually works faster when you update the array in place instead of maintaining two separate arrays for the sweep over the state space!

Back to Gridworld and the equiprobable action selection policy:

$t = 0:$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$t = 1:$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

Policy Improvement

$t = 2 :$

0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0

$t = 3 :$

0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0

$t = 10 :$

0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0

$t = \infty :$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

Suppose you have a deterministic policy π and want to improve on it. How about choosing a in state s and then continuing to follow π?

Policy improvement theorem:

If $Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$ for all states s, then:

$$V^{\pi'}(s) \geq V^{\pi}(s)$$

Relatively easy to prove by repeated expansion of $Q^{\pi}(s, \pi'(s))$.

Consider a short-sighted greedy improvement to the policy π, in which, at each state we choose the action that appears best according to $Q^{\pi}(s, a)$

$$\pi'(s, a) = \arg\max_{a} Q^{\pi}(s, a)$$

9

$$= \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

What would policy improvement in the Grid-world example yield?

	L	L	L/D
U	L/U	L/D	D
U	U/R	R/D	D
U/R	R	R	

Note that this is the same thing that would happen from $t = 3$ onwards!

Only guaranteed to be an improvement over the random policy but in this case it happens to also be optimal.

If the new policy π' is no better than π then it must be true for all s that

$$V^{\pi'}(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi'}(s')]$$

This is the Bellman optimality equation, and therefore $V^{\pi'}$ must be V^*.

The policy improvement theorem generalizes to stochastic policies under the definition:

$$Q^\pi(s, \pi'(s)) = \sum_a \pi'(s, a) Q^\pi(s, a)$$

Policy Iteration

Interleave the steps. Start with a policy, evaluate it, then improve it, then evaluate the new policy, improve it, etc., until it stops changing.

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

Algorithm:

1. Initialize with arbitrary value function and policy

2. Perform policy evaluation to find $V^\pi(s)$ for all $s \in S$. That is, repeat the following update until convergence

$$V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$$

3. Perform policy improvement:

$$\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$$

If the policy is the same as last time then you are done!

Takes very few iterations in practice, even though the policy evaluation step is itself iterative.

Value Iteration

Initialize V arbitrarily

Repeat until convergence:

For each $s \in S$

- $V(s) \leftarrow \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

Output policy π such that

$$\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$$

Convergence criterion: the maximum change in the value of any state in the state set in the last iteration was less than some threshold

Note that this is simply turning the Bellman equation into an update rule! It can also be thought of as an update that cuts off policy evaluation after one step...

Discussion of Dynamic Programming

We can solve MDPs with millions of states. Efficiency isn't as bad as you'll sometimes hear. There is a problem in that the state representation must be relatively compact. If your state representation, and hence your number of states, grows very fast, then you're in trouble. But that's a feature of the problem, not the method.

Asynchronous dynamic programming: a lead in...

Instead of doing sweeps of the whole state space at each iteration, just use whatever values are available at any time to update any state. In place algorithms.

Convergence has to be handled carefully, be-
cause in general convergence to the value func-
tion only occurs if we then visit all states in-
finitely often in the limit — so we can't stop
going to certain states if we want the guaran-
tee to hold.

But we can run an iterative DP algorithm *on-
line* at the same time that the agent is actually
in the MDP. Could focus on important regions
of the state space, perhaps at the expense of
true convergence?

What's next? What if we don't have a correct
model of the MDP? How do we build one while
also acting? We'll start by going through really
simple MDPs, namely Bandit problems.