Markov Decision Processes II

The Bellman Equation

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

- The Bellman equation gives the utility of a state
- If there are n states, there are n Bellman Equations to solve
- This is a system of simultaneous equations
- But the equations are nonlinear because of the max operator

Iterative Solution

Define $U_1(s)$ to be the utility if the agent is at state s and lives for 1 time step

$$U_1(s) = R(s)$$

Calculate this for all states s

Define $U_2(s)$ to be the utility if the agent is at state s and lives for 2 time steps

$$U_2(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_1(s')$$

This has already been calculated above

The Bellman Update

More generally, we have:

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

- This is the maximum possible expected sum of discounted rewards (ie. the utility) if the agent is at state s and lives for i+1 time steps
- This equation is called the Bellman Update

The Bellman Update

- As the number of iterations goes to infinity, $U_{i+1}(s)$ converges to an equilibrium value $U^*(s)$.
- The final utility values $U^*(s)$ are solutions to the Bellman equations. Even better, they are the unique solutions and the corresponding policy is optimal
- This algorithm is called ***Value-Iteration***
- The optimal policy is given by:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')U^*(s')$$

The Value Iteration Algorithm

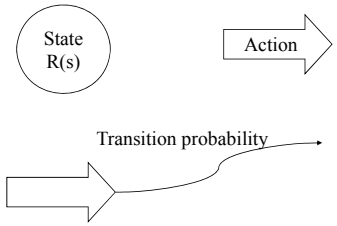
$$U_1(s) = R(s)$$
$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

1. Apply bellman update until it the utility function converges (to $U^*(s)$).
2. The optimal policy is given by:

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')U^*(s')$$

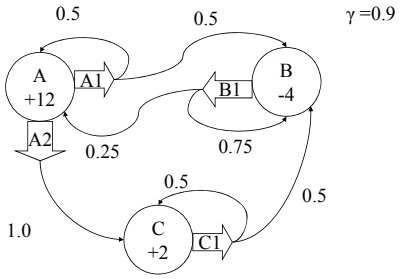
Example

- We will use the following convention when drawing MDPs graphically:



7

Example



8

Example

i=1

$U_1(A) = R(A)=12$

$U_1(B) = R(B)=-4$

$U_1(C) = R(C)=2$

9

Example

$U_1(A)$	$U_1(B)$	$U_1(C)$
12	-4	2

i=2

$U_2(A) = 12 + (0.9) * \max\{(0.5)(12)+(0.5)(-4), (1.0)(2)\}$
$= 12 + (0.9)*\max\{4.0,2.0\} = 12 + 3.6 = 15.6$

$U_2(B) = -4 + (0.9) * \{(0.25)(12)+(0.75)(-4)\} = -4 +$
$(0.9)*0 = -4$

$U_2(C) = 2 + (0.9) * \{(0.5)(2)+(0.5)(-4)\} = 2 + (0.9)*(-1)$
$= 2-0.9 = 1.1$

10

Example

$U_2(A)$	$U_2(B)$	$U_2(C)$
15.6	-4	1.1

i=3

$U_3(A) = 12 + (0.9) * \max\{(0.5)(15.6)+(0.5)(-$
$4),(1.0)(1.1)\} = 12 + (0.9) * \max\{5.8,1.1\} = 12 +$
$(0.9)(5.8) = 17.22$

$U_3(B) = -4 + (0.9) * \{(0.25)(15.6)+(0.75)(-4)\} = -4 +$
$(0.9)*(3.9-3) = -4 + (0.9)(0.9) = -3.19$

$U_3(C) = 2 + (0.9) * \{(0.5)(1.1)+(0.5)(-4)\} = 2 +$
$(0.9)*\{0.55-2.0\} = 2 + (0.9)(-1.45) = 0.695$

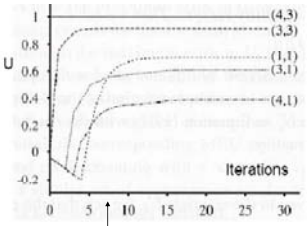
11

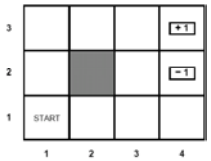
The Bellman Update

- What exactly is going on?
- Think of each Bellman update as an update of each local state
- If we do enough local updates, we end up propagating information throughout the state space

12

Value Iteration on the Maze





Notice that rewards are negative until a path to (4,3) is found, resulting in an increase in U

13

Value-Iteration Termination

When do you stop?

In an iteration over all the states, keep track of the maximum change in utility of any state (call this δ)

When δ is less than some pre-defined threshold, stop

This will give us an approximation to the true utilities, we can act greedily based on the approximated state utilities

14

Comments

- Value iteration is designed around the idea of the utilities of the states
- The computational difficulty comes from the max operation in the bellman equation
- Instead of computing the general utility of a state (assuming acting optimally), a much easier quantity to compute is the utility of a state assuming a policy

15

Utility of a policy at state s

- $U_\pi(s)$: the utility of policy π at state s
- $U^*(s)$ can be considered as $U_{\pi^*}(s)$ where π^* is an optimal policy
- Given a fixed policy, can compute its utility at state ***s*** as follows:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') \cdot U_\pi(s')$$

Note the difference from:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

Evaluating a Policy

Once we compute the utilities, we can easily improve the current policy by one-step look-ahead:

$$\pi'(s) = \arg\max_a \sum_{s'} T(s, a, s') U_\pi(s')$$

This suggests a different approach for finding optimal policy

17

Policy Iteration

- Start with a randomly chosen initial policy π_0
- Iterate until no change in utilities:
  1. Policy evaluation: given a policy π_i, calculate the utility $U_i(s)$ of every state *s* using policy π_i
  2. Policy improvement: calculate the new policy π_{i+1} using one-step look-ahead based on $U_i(s)$ ie.

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') U_i(s')$$

18

Policy Evaluation

- Policy improvement is straightforward
- Policy evaluation requires a simpler version of the Bellman equation
- Compute $U_i(s)$ for every state s using π_i:

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

Notice that there is no max operator, so the above equations are linear! $O(n^3)$ where n is the number of states

19

Policy Evaluation

- $O(n^3)$ is still too expensive for large state spaces
- Instead of calculating exact utilities, we could calculate approximate utilities
- The simplified Bellman update is:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

- Repeat the above k times to get the next utility estimate

This is called modified policy iteration

20

Comparison

- Which would you prefer, policy or value iteration?
- Depends…
  - If you have lots of actions in each state: policy iteration
  - If you have a pretty good policy to start with: policy iteration
  - If you have few actions in each state: value iteration

21

Limitations

- Need to represent the utility (and policy) for every state
- In real problems, the number of states may be very large
- Leads to intractably large tables
- Need to find compact ways to represent the states eg
  - Function approximation
  - Hierarchical representations
  - Memory-based representations

22

What you should know

- How value iteration works
- How policy iteration works
- Pros and cons of both
- What is the big problem with both value and policy iteration

23