CS 188: Artificial Intelligence
Spring 2010

Lecture 10: MDPs
2/18/2010

Pieter Abbeel – UC Berkeley

Many slides over the course adapted from either Dan Klein, Stuart Russell or Andrew Moore

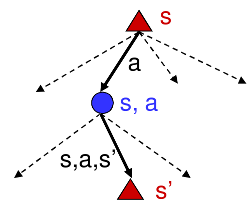
Announcements

- P2: Due tonight
- W3: Expectimax, utilities and MDPs---out tonight, due next Thursday.
- Online book: Sutton and Barto

  http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html

Recap: MDPs

- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0

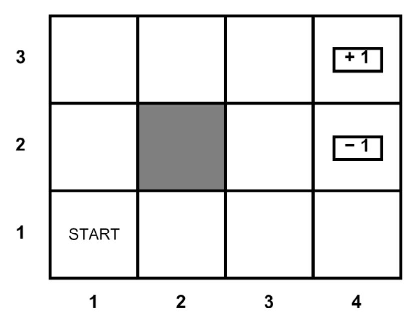


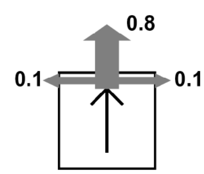
- Quantities:
  - Policy = map of states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state

4

Recap MPD Example: Grid World

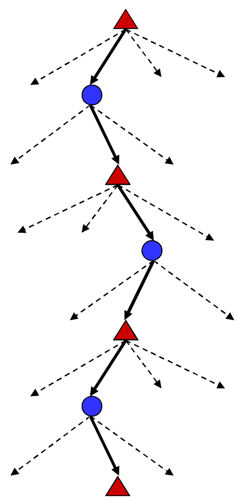
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards





Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!



6

Value Iteration

- Idea:
  - $V_i^*(s)$: the expected discounted sum of rewards accumulated when starting from state s and acting optimally for a horizon of i time steps.
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for horizon i+1:

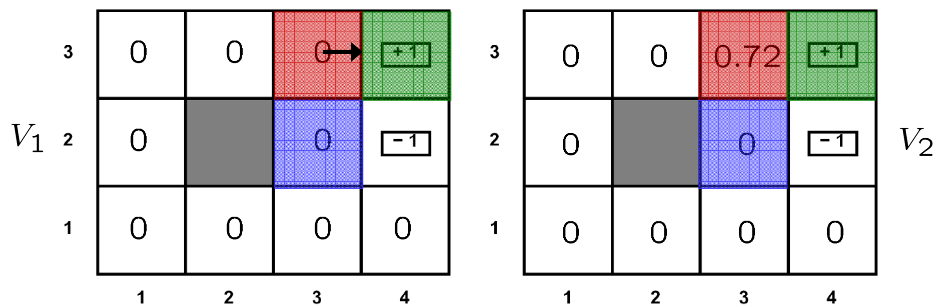
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

7

Example: γ=0.9, living reward=0, noise=0.2

Example: Bellman Updates



$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_i(s')\right]$$

$$V_2(\langle 3,3\rangle) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s')\left[R(\langle 3,3\rangle) + 0.9\, V_1(s')\right]$$

max happens for a=right, other actions not shown

$$= 0.9\,[0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

8

Convergence*

- Define the max-norm: $||U|| = \max_s |U(s)|$
- Theorem: For any two approximations U and V

$$||U_{i+1} - V_{i+1}|| \leq \gamma\, ||U_i - V_i||$$

  - I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution
- Theorem:

$$||U_{i+1} - U_i|| < \epsilon,\ \Rightarrow\ ||U_{i+1} - U|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

10

At Convergence

- At convergence, we have found the optimal value function V* for the discounted infinite horizon problem, which satisfies the Bellman equations:

$$\forall s \in S: \quad V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

12

Practice: Computing Actions

- Which action should we chose from state s:
  - Given optimal values V?

$$\arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

  - Given optimal q-values Q?

$$\arg\max_a Q^*(s,a)$$

  - Lesson: actions are easier to select from Q's!

13

Complete procedure

- 1. Run value iteration (off-line)

→Returns V, which (assuming sufficiently many iterations is a good approximation of V*)

- 2. Agent acts. At time t the agent is in state s_t and takes the action a_t:

$$\arg\max_a \sum_{s'} T(s_t, a, s')[R(s_t, a, s') + \gamma V^*(s')]$$

14

Complete procedure

① Offline

$V_0(s) = 0 \quad \forall s$

for $i = 0, 1, 2, 3, 4, \ldots$

for all s : $V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$

$\left\{ \begin{array}{l} \text{for } a = 1, \ldots, |A| \\ \quad Q_{i+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')] \\ V_{i+1}(s) = \max_a Q_{i+1}(s,a) \end{array} \right.$

if $(\|V_{i+1} - V_i\| \leq \varepsilon)$

break and return $V_{i+1} \approx V^*$ $\quad \left[\|V_{i+1} - V^*\| \leq \frac{2\varepsilon\gamma}{1-\gamma} \right]$

② Choosing actions online

for(;;){

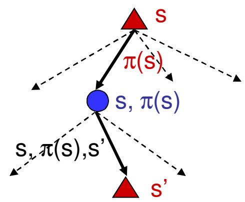
observe current state s_t; compute $a_t = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

execute a_t }

15

Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fix (general non-optimal) policy
- Define the utility of a state s, under a fixed policy π:
  $V^\pi(s)$ = expected total discounted rewards (return) starting in s and following π
- Recursive relation (one-step look-ahead / Bellman equation):



$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

18

Policy Evaluation

- How do we calculate the V's for a fixed policy?
- Idea one: modify Bellman updates

$$V_0^\pi(s) = 0$$

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^\pi(s')]$$

- Idea two: it's just a linear system, solve with Matlab (or whatever)

19

Policy Iteration

- Alternative approach:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

20

Policy Iteration

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

23

Comparison

- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- In policy iteration:
  - Several passes to update utilities with frozen policy
  - Occasional passes to update policies
- Hybrid approaches (asynchronous policy iteration):
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

25

Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change:
  If $|V_{i+1}(s) - V_i(s)|$ is large then update predecessors of s

MDPs recap

- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
  - Start state s_0
- Solution methods:
  - Value iteration (VI)
  - Policy iteration (PI)
  - Asynchronous value iteration
- Current limitations:
  - Relatively small state spaces
  - Assumes T and R are known