CS 188: Artificial Intelligence
Spring 2009

Lecture 10: Markov Decision Processes II
2/19/2009

John DeNero – UC Berkeley

Slides adapted from Dan Klein, Stuart Russell or Sutton & Barto

Announcements

- Project 3:
  - Posted yesterday
  - Due in two weeks: Wednesday 3/4

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

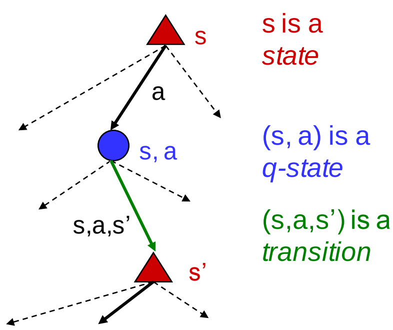
- Quantities:
  - Policy = map of states to actions
  - Episode = one run of an MDPs
  - Utility (Returns) = sum of discounted rewards
  - Values = expected future returns from a state
  - Q-Values = expected future returns from a q-state

Optimal Utilities

- The utility of a state s:
  $V^*(s)$ = expected return starting in s and acting optimally

- The utility of a q-state (s,a):
  $Q^*(s,a)$ = expected return starting in s, taking action a and thereafter acting optimally

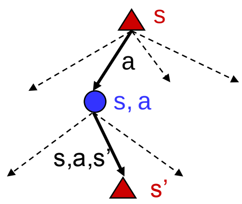
- The optimal policy:
  $\pi^*(s)$ = optimal action from state s



The Bellman Equations

- One-step lookahead relationship amongst optimal utility values:

  Optimal rewards = maximize over first action, then follow the optimal policy



- Formally:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

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

Review: 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!

Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth 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

[DEMO]

Convergence*

- Define the max-norm: $||U|| = \max_s |U(s)|$

- Theorem: For any two approximations U and V

$$||U^{t+1} - V^{t+1}|| \leq \gamma \, ||U^t - V^t||$$

  - 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:

$$||U^{t+1} - U^t|| < \epsilon, \Rightarrow ||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$$

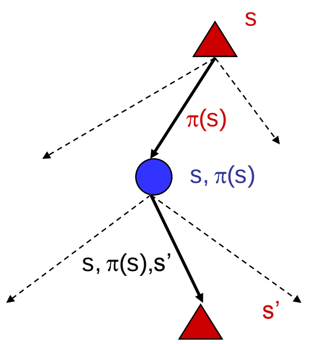
  - I.e. once the change in our approximation is small, it must also be close to correct

8

Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fixed (perhaps non-optimal) policy
- Define the utility of a state s, under a fixed policy π:

  $V^\pi(s)$ = expected total discounted rewards starting in s and following π



- Recursive relation (one-step look-ahead / Bellman equation):

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

9

Policy Evaluation

- How do we calculate the V's for a fixed policy?
- Idea one: turn recursive equations into 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; ask Matlab

10

Policy Iteration

- Alternative to value iteration:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step lookaheah with resulting converged (but not optimal!) utilities
  - Repeat steps until policy converges

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

11

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

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

[DEMO]

Comparison

- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (implicitly, based on current utilities)
  - Tracking the policy isn't necessary; we take the max

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

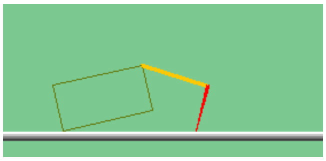
- In policy iteration:
  - Several passes to update utilities with fixed policy
  - After policy is evaluated, a new policy is chosen
- Together, these are dynamic programming for MDPs

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

Reinforcement Learning

- Reinforcement learning:
  - Still have an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
  - Still looking for a policy $\pi(s)$



[DEMO]

  - New twist: don't know T or R
    - I.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

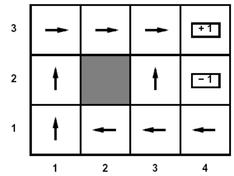
15

Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated
- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

16

Passive Learning



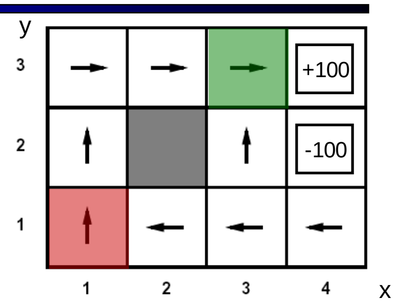
- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy π(s)
  - Goal: learn the state values (and maybe the model)
- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to action selection soon

18

Example: Direct Estimation

- Episodes:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



γ = 1, R = -1

U(1,1) ~ (92 + -106) / 2 = -7

U(3,3) ~ (99 + 97 + -102) / 3 = 31.3

19

Model-Based Learning

- In general, want to learn the optimal policy, not evaluate a fixed policy
- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model (the whole state space)

20

Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
- Empirical model learning
  - Simplest case:
    - Count how many of each s' for each s,a
    - Divide by total times in s,a to give estimate of T(s,a,s')
    - Discover R(s,a,s') the first time we experience (s,a,s')
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes like "stationary noise")

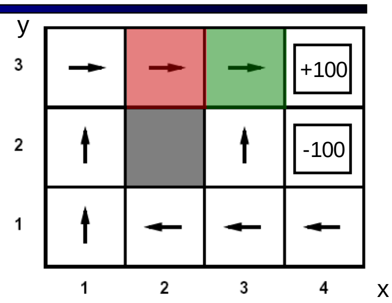
21

Example: Model-Based Learning

- Episodes:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



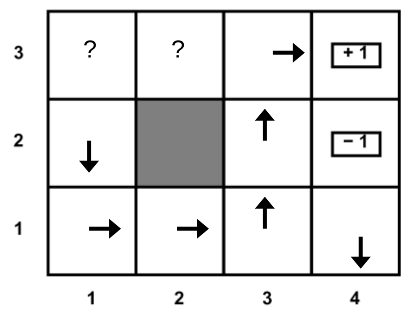
$\gamma = 1$

T(<3,3>, right, <4,3>) = 1 / 3

T(<2,3>, right, <3,3>) = 2 / 2

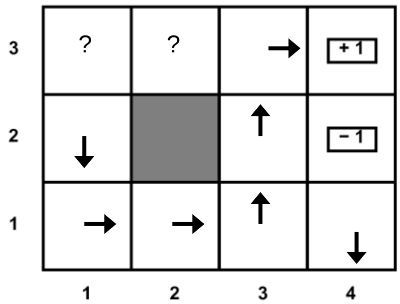
Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We'll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy



What Went Wrong?

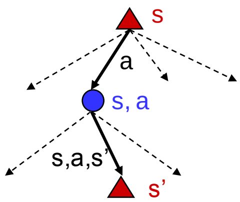
- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit



24

Model-Free Learning

- Big idea: why bother learning T?
  - Update V each time we experience a transition
  - Frequent outcomes will contribute more updates (over time)
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs



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

$$sample = R(s, a, s') + \gamma V^{\pi}(s')$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

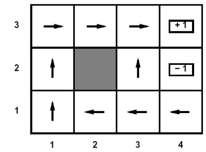
25

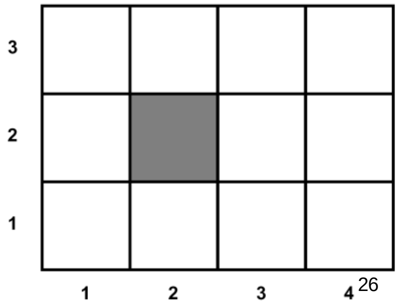
Example: Passive TD

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \left[R(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)\right]$$

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	

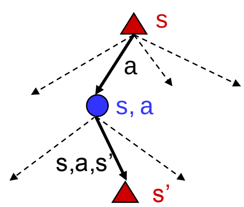
Take $\gamma = 1$, $\alpha = 0.5$





Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy, we're sunk:



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

$$Q^{*}(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{*}(s')\right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!