

# CS 188: Artificial Intelligence

## Fall 2007

### Lecture 10: MDPs

9/27/2007

Dan Klein – UC Berkeley

## High-Low

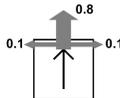
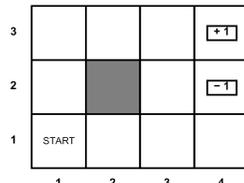
- States: 2, 3, 4, done
- Actions: High, Low
- Model:  $T(s, a, s')$ :
  - $P(s'=done | 4, High) = 3/4$
  - $P(s'=2 | 4, High) = 0$
  - $P(s'=3 | 4, High) = 0$
  - $P(s'=4 | 4, High) = 1/4$
  - $P(s'=done | 4, Low) = 0$
  - $P(s'=2 | 4, Low) = 1/2$
  - $P(s'=3 | 4, Low) = 1/4$
  - $P(s'=4 | 4, Low) = 1/4$
  - ...
- Rewards:  $R(s, a, s')$ :
  - Number shown on  $s'$  if  $s \neq s'$
  - 0 otherwise
- Start: 3



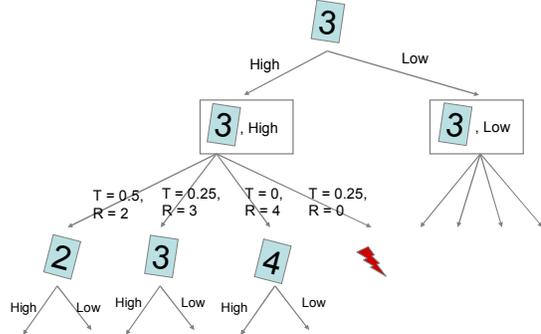
*Note: could choose actions with search. How?*

## Markov Decision Processes

- An MDP is defined by:
  - A set of states  $s \in S$
  - A set of actions  $a \in A$
  - A transition function  $T(s, a, s')$ 
    - Prob that  $a$  from  $s$  leads to  $s'$  i.e.,  $P(s' | s, a)$
    - Also called the model
  - A reward function  $R(s, a, s')$ 
    - Sometimes just  $R(s)$  or  $R(s')$
  - A start state (or distribution)
  - Maybe a terminal state
- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions

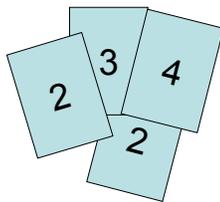


## Example: High-Low



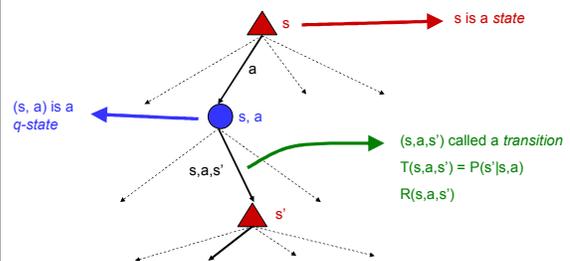
## Example: High-Low

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say "high" or "low"
- New card is flipped
- If you're right, you win the points shown on the new card
- Ties are no-ops
- If you're wrong, game ends
- Differences from expectimax:
  - #1: get rewards as you go
  - #2: you might play forever!



## MDP Search Trees

- Each MDP state gives an expectimax-like search tree



## Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards

- Typically consider **stationary preferences**:

$$[r, r_0, r_1, r_2, \dots] \succ [r', r'_0, r'_1, r'_2, \dots]$$

$$\Leftrightarrow$$

$$[r_0, r_1, r_2, \dots] \succ [r'_0, r'_1, r'_2, \dots]$$

*Assuming that reward depends only on state for these slides!*

- Theorem: only two ways to define stationary utilities

- Additive utility:

$$V([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

- Discounted utility:

$$V([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

## Episodes and Returns

- An episode is a run of an MDP

- Sequence of transitions (s,a,s')
- Starts at start state
- Ends at terminal state (if it ends)
- Stochastic!

- The utility, or return, of an episode

- The discounted sum of the rewards

$$\sum_i \gamma^i R(s_i, a_i, s_{i+1})$$

## Infinite Utilities?!

- Problem: infinite sequences with infinite rewards

- Solutions:

- Finite horizon:
  - Terminate after a fixed T steps
  - Gives nonstationary policy ( $\pi$  depends on time left)
- Absorbing state(s): guarantee that for every policy, agent will eventually "die" (like "done" for High-Low)
- Discounting: for  $0 < \gamma < 1$

$$V([s_0, \dots, s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq R_{\max} / (1 - \gamma)$$

- Smaller  $\gamma$  means smaller "horizon" – shorter term focus

## Utilities under Policies

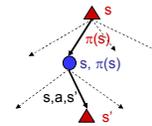
- Fundamental operation: compute the utility of a state s

- Define the value (utility) of a state s, under a fixed policy  $\pi$ :

$V^\pi(s)$  = expected return starting in s and following  $\pi$

- Recursive relation (one-step look-ahead):

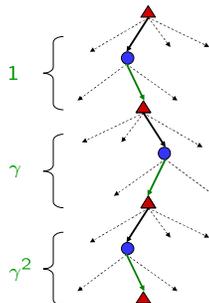
$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]$$



## Discounting

- Typically discount rewards by  $\gamma < 1$  each time step

- Sooner rewards have higher utility than later rewards
- Also helps the algorithms converge



## Policy Evaluation

- How do we calculate values for a fixed policy?

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

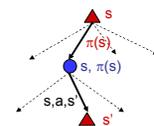
- Idea two: turn recursive equations into updates

- $V_i^\pi(s)$  = expected returns over the next i transitions while following  $\pi$

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

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

Equivalent to doing depth i search and plugging in zero at leaves



## Example: High-Low

- Policy: always say "high"
- Iterative updates:

$$V_0 = \{2 : 0, \quad 3 : 0, \quad 4 : 0, \quad d : 0\}$$

$$V_1(2) = \frac{1}{2}(R(2,H,2) + V_0(2)) + \frac{1}{2}(R(2,H,3) + V_0(3)) +$$

$$\frac{1}{4}(R(2,H,4) + V_0(4)) + 0(R(2,H,d) + V_0(d)) +$$

$$V_1(2) = \frac{1}{2}(0 + 0) + \frac{1}{2}(3 + 0) + \frac{1}{4}(4 + 0) + 0(0 + 0);$$

$$V_1(2) = \frac{7}{4}$$

$$V_1 = \{2 : \frac{7}{4}, \quad 3 : 1, \quad 4 : 0, \quad d : 0\}$$

[DEMO]

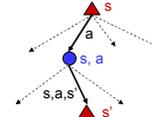
## Optimal Utilities

- Fundamental operation: compute the optimal utilities of states  $s$

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

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

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



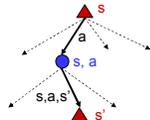
3	0.812	0.669	0.912	←
2	0.792	0.660	0.660	←
1	0.705	0.695	0.611	←
	1	2	3	4

3	←	←	←	←
2	↑	←	↑	←
1	↑	←	←	←
	1	2	3	4

## Q-Functions

- Also, define a **q-value**, for a state and action (q-state)
  - $Q^\pi(s)$  = expected return starting in  $s$ , taking action  $a$  and following  $\pi$  thereafter



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

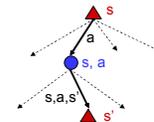
$$V^\pi(s) = Q^\pi(s, \pi(s))$$

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

## The Bellman Equations

- Definition of utility leads to a simple relationship amongst optimal utility values:

Optimal rewards = maximize over first action and then follow optimal policy



- Formally:

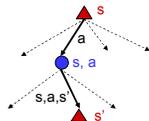
$$V^*(s) = \max_a Q^*(s, a)$$

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

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

## Recap: MDP Quantities

- Return = Sum of future discounted rewards in one episode (stochastic)



- $V$ : Expected return from a state under a policy

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

- $Q$ : Expected return from a q-state under a policy

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

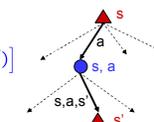
## Solving MDPs

- We want to find the **optimal policy**  $\pi$
- Proposal 1: modified expectimax search:

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

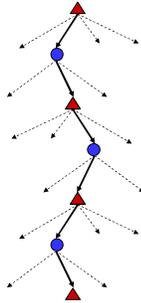
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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



## MDP Search Trees?

- Problems:
  - This tree is usually infinite (why?)
  - The same states appear over and over (why?)
  - There's actually one tree per state (why?)
- Ideas:
  - Compute to a finite depth (like expectimax)
  - Consider returns from sequences of increasing length
  - Cache values so we don't repeat work

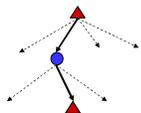


## Value Iteration

- Problems with the recursive computation:
  - Have to keep all the  $V_k(s)$  around all the time
  - Don't know which depth  $\pi_k(s)$  to ask for when planning
- Solution: value iteration
  - Calculate values for all states, bottom-up
  - Keep increasing  $k$  until convergence

## Value Estimates

- Calculate estimates  $V_k^*(s)$ 
  - Not the optimal value of  $s!$
  - The optimal value considering only next  $k$  time steps ( $k$  rewards)
  - As  $k \rightarrow \infty$ , it approaches the optimal value
- Why:
  - If discounting, distant rewards become negligible
  - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
  - Otherwise, can get infinite expected utility and then this approach actually won't work



## 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  $i+1$ :

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

- 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

## Memoized Recursion?

- Recurrences:
  - $V_0^*(s) = 0$
  - $V_i^*(s) = \max_a Q_i^*(s, a)$
  - $Q_i^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_{i-1}^*(s')]$
  - $\pi_i(s) = \arg \max_a Q_i^*(s, a)$
- Cache all function call results so you never repeat work
- What happened to the evaluation function?

## Example: Bellman Updates

3	0	0	0	→ +1	3	0	0	-0.72	→ +1
2	0	0	0	← -1	2	0	0	0	← -1
1	0	0	0	0	1	0	0	0	0
	1	2	3	4		1	2	3	4

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

$$V_{i+1}((3, 3)) = \sum_{s'} T((3, 3), \text{right}, s') [R((3, 3)) + 0.9 V_i(s')] = 0.9 [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

## Example: Value Iteration

	$V_2$				$V_3$			
3	0	0	0.72	[-1]	0	0.52	0.78	[-1]
2	0	0	0	[-1]	0	0	0.43	[-1]
1	0	0	0	0	0	0	0	0
	1	2	3	4	1	2	3	4

- Information propagates outward from terminal states and eventually all states have correct value estimates

[DEMO]

## Policy Iteration

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

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

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

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

## 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

## 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

## Policy Iteration

- Alternative approach:

- Step 1: Policy evaluation:** calculate utilities for a fixed policy (not optimal utilities!) until convergence
- Step 2: Policy improvement:** update policy based on resulting converged (but not optimal!) utilities
- Repeat steps until policy converges

- This is policy iteration

- Can converge faster under some conditions