

CS 188: Artificial Intelligence

Fall 2006

Lecture 9: MDPs

9/26/2006

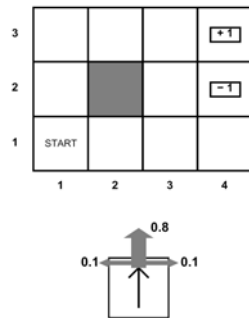
Dan Klein – UC Berkeley

Reinforcement Learning

- [DEMOS]
- Basic idea:
 - Receive feedback in the form of **rewards**
 - Agent's utility is defined by the reward function
 - Must learn to act so as to **maximize expected rewards**
 - **Change the rewards, change the behavior**
- Examples:
 - Playing a game, reward at the end for winning / losing
 - Vacuuming a house, reward for each piece of dirt picked up
 - Automated taxi, reward for each passenger delivered

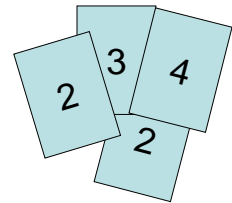
Markov Decision Processes

- Markov decision processes (MDPs)
 - A set of states $s \in S$
 - A model $T(s, a, s') = P(s' | s, a)$
 - Probability that action a in state s leads to s'
 - A reward function $R(s, a, s')$ (sometimes just $R(s)$ for leaving a state or $R(s')$ for entering one)
 - A start state (or distribution)
 - Maybe a terminal state
- MDPs are the simplest case of reinforcement learning
 - In general reinforcement learning, we don't know the model or the reward function



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



High-Low

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

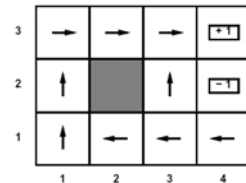


Note: could choose actions with search. How?

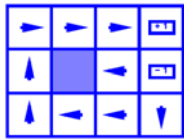
MDP Solutions

- In deterministic single-agent search, want an optimal sequence of actions from start to a goal
- In an MDP, like expectimax, want an optimal policy $\pi(s)$
 - A policy gives an action for each state
 - Optimal policy maximizes expected utility (i.e. expected rewards) if followed
 - Defines a reflex agent

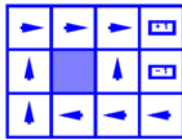
Optimal policy when $R(s, a, s') = -0.04$ for all non-terminals s



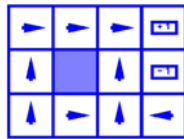
Example Optimal Policies



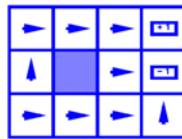
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Stationarity

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

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

Infinite Utilities?!

- Problem: infinite state sequences with infinite rewards

- Solutions:

- Finite horizon:
 - Terminate after a fixed T steps
 - Gives nonstationary policy (π 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 γ means smaller horizon

How (Not) to Solve an MDP

- The inefficient way:
 - Enumerate policies
 - For each one, calculate the expected utility (discounted rewards) from the start state
 - E.g. by simulating a bunch of runs
 - Choose the best policy
- Might actually be reasonable for High Low...
- We'll return to a (better) idea like this later

Utility of a State

- Define the utility of a state under a policy:

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

- Recursive definition (one step look ahead):

$$V^\pi(s) = E_{P(s'|\pi(s),s)}[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

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

Policy Evaluation

- 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, solve with Matlab (or Mosek, or Cplex)

Example: High-Low

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

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

$$V_1(2) = \frac{1}{2}(R(2, H, 2) + V_0(2)) + \frac{1}{4}(R(2, H, 3) + V_0(3)) + \frac{1}{4}(R(2, H, 4) + V_0(4)) + 0(R(2, H, d) + V_0(d))$$

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

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

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

Example: GridWorld

- [DEMO]

Q-Functions

- To simplify things, introduce a **q-value**, for a state and action under a policy
 - Utility of taking starting in state s , taking action a , then following π thereafter

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

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

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

Optimal Utilities

- Goal: calculate the optimal utility of each state
 - $V^*(s)$ = expected (discounted) rewards with optimal actions
- Why: Given optimal utilities, MEU tells us the optimal policy



Practice: Computing Actions

- Which action should we choose from state s :
 - Given optimal q-values Q ?

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

- Given optimal values V ?

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

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:

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

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

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

Example: GridWorld

3	0.812	0.868	0.912	⊕1
2	0.776		0.660	⊖1
1	0.640	0.611	0.388	
	1	2	3	4

$$V^*(1,1) = -0.04$$

$$+ \gamma \max \{ 0.8V^*(1,2) + 0.1V^*(2,1) + 0.1V^*(1,1), \text{ up} \\ 0.9V^*(1,1) + 0.1V^*(1,2), \text{ left} \\ 0.9V^*(1,1) + 0.1V^*(2,1), \text{ down} \\ 0.8V^*(2,1) + 0.1V^*(1,2) + 0.1V^*(1,1) \} \text{ right}$$

Value Iteration

- Idea:
 - Start with bad guesses at all utility values (e.g. $V_0(s) = 0$)
 - Update all values simultaneously using the Bellman equation (called a **value update** or **Bellman update**):

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

- Repeat until convergence
- Theorem: will converge to unique optimal values
 - Basic idea: bad guesses get refined towards optimal values
 - Policy may converge long before values do

Example: Bellman Updates

3	0	0	0	⊕1
2	0		0	⊖1
1	0	0	0	0
	1	2	3	4

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

Example: Value Iteration

3	0	0	0.72	⊕1
2	0		0	⊖1
1	0	0	0	0
	1	2	3	4

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

[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. one the change in our approximation is small, it must also be close to correct

Policy Iteration

- Alternate approach:
 - Policy evaluation**: calculate utilities for a fixed policy until convergence (remember the beginning of lecture)
 - Policy improvement**: update policy based on resulting converged utilities
 - Repeat until policy converges
- This is **policy iteration**
 - Can converge faster under some conditions

Policy Iteration

- If we have a fixed policy π , use simplified Bellman equation to calculate utilities:

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

- For fixed utilities, easy to find the best action according to one step look ahead

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

Comparison

- In value iteration:
 - Every pass (or "backup") updates both policy (based on current utilities) and utilities (based on current policy)
- In policy iteration:
 - Several passes to update utilities
 - 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

Next Class

- In real reinforcement learning:
 - Don't know the reward function $R(s,a,s')$
 - Don't know the model $T(s,a,s')$
 - So can't do Bellman updates
- Need new techniques:
 - Q-learning
 - Model learning
 - Agents actually have to interact with the environment rather than simulate it