CS 188: Artificial Intelligence  
Spring 2009

Lecture 9: Markov Decision Processes  
2/17/2009

John DeNero – UC Berkeley  
Slides adapted from Dan Klein, Stuart Russell or Sutton & Barto

Announcements

- Project 2:  
  - Due tomorrow (up to 2 late days)
- Project 3:  
  - Posted tonight or tomorrow  
  - Due in two weeks: Wednesday 3/4
- Don't forget about the midterm:  
  - 6pm-9pm on Thursday 3/19 in Evans 10
- Readings:  
  - For MDPs / reinforcement learning, we're using an online reading: Sutton & Barto
  - Next week is grading week

Connect 4 Challenge

Top 2: 101 (10am) and 103 (2pm)
Depth 1:
101 wins 114 (0.570)  
103 wins 71 (0.355)  
ties 15 (0.075)
Depth 2:
101 wins 141 (0.705)  
103 wins 48 (0.240)  
ties 11 (0.055)
Depth 3:
101 wins 85 (0.425)  
103 wins 46 (0.230)  
ties 69 (0.345)

What's going on here?

Today

- Markov Decision Processes
- Rewards and Utilities
- Dynamic Programming

Reinforcement Learning

- 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
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
- Big rewards come at the end

Markov Decision Processes

- An MDP is defined by:
  - A set of states \( S \)
  - A set of actions \( A \)
  - 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

What is Markov about MDPs?

- Andrey Markov (1856-1922)
- Markov means that given the present state, the future and the past are independent
- For Markov decision processes, Markov means:
  \[
P(S_{t+1} = s' | S_t = s, A_t = a_t) \equiv P(S_{t+1} = s' | S_t = s, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots)
\]

Solving MDPs

- In deterministic single-agent search problem, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy \( \pi^* : S \rightarrow A \)
  - A policy \( \pi \) gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

Example Optimal Policies

- Three card types: 2, 3, 4
- Infinite deck, twice as many 2’s
- Start with 3 showing
- After each card, you predict the next card will be “high” or “low” compared to the current one
- 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

Example: High-Low

- Differences from expectimax:
  - #1: get rewards as you go
  - #2: you might play forever
High-Low as an MDP

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

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
- A very convenient property: stationary preferences
  \[ [r, r_0, r_1, r_2, \ldots] > [r, r'_0, r'_1, r'_2, \ldots] \]
- Theorem: only two ways to define stationary preferences
  - Additive utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \]
  - Discounted utility:
    \[ U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \]

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards
- Solutions:
  - Finite horizon:
    - Terminate sum of rewards after a fixed \( T \) steps
    - Gives nonstationary policy (\( \gamma \) depends on time left)
    - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
  - Discounting: for \( 0 < \gamma < 1 \)
    \[ U([r_0, \ldots, r_n]) = \sum_{i=0}^{\infty} \gamma^i r_i \leq R_{\max}/(1 - \gamma) \]
    - Smaller \( \gamma \) means smaller “horizon” – shorter term focus

Discounting

- Typically discount rewards by \( \gamma < 1 \) each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge quickly
  - Like an interest rate
Recap: Defining 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 \( \gamma \))
  - Start state \( s_0 \)

- Properties of an MDP:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Optimal Utilities

- Fundamental operation: compute the optimal utilities of states \( s \) (all at once)

- Why? Optimal values define optimal policies!

- Define the (optimal) utility of a state \( s \):
  \[ V^*(s) = \text{expected total return starting in } s \text{ and acting optimally} \]

- Define the (optimal) utility of a q-state:
  \[ Q^*(s,a) = \text{expected return starting in } s, \text{ taking action } a \text{ and thereafter acting optimally} \]

- Define the optimal policy:
  \[ \pi^*(s) = \text{optimal action from state } s \]

The Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:
  \[ V^*(s) = \max_a Q^*(s,a) \]
  \[ Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right] \]
  \[ V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right] \]

Value Estimates

- Calculate estimates \( V_i^*(s) \)
  - Not the optimal value of \( s \)!
  - The optimal value considering only next \( i \) time steps (if rewards)
  - As \( i \to \infty \), 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 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 \( i+1 \):
    \[ V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_i^*(s') \right] \]
  - 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
Example: Bellman Updates

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

\[
V_2(3, 3) = \sum_{s'} T((3, 3), \text{right}, s') [R((3, 3)) + 0.9 V_1(s')] 
= 0.9 [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0] 
\]

Example: Value Iteration

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

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:
  \[
  ||U^{t+1} - U^t|| < \epsilon \Rightarrow ||U^{t+1} - U^t|| < 2\epsilon \gamma / (1 - \gamma) 
  \]
  - i.e. once the change in our approximation is small, it must also be close to correct

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!

Utilities for Fixed Policies

- Another basic operation: compute the utility of a state \(s\) under a fixed (general non-optimal) policy
- Define the utility of a state \(s\), under a fixed policy \(\pi\):
  \[
  V(s) = \text{expected total discounted rewards (return) starting in } s \text{ and following } \pi 
  \]
- Recursive relation (one-step look-ahead / Bellman equation):
  \[
  V(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V(s')] 
  \]

Policy Evaluation

- How do we calculate the \(V\)'s for a fixed policy?
  - Idea one: turn recursive equations into updates
    \[
    V_0^\pi(s) = 0 
    \]
    \[
    V_\pi^{t+1}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_\pi^\pi(s')] 
    \]
  - Idea two: it's just a linear system; solve with Matlab (or whatever)
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 using one-step lookahead 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

---

Policy Iteration

- **Policy evaluation:** with fixed current policy $\pi$, find values with simplified Bellman updates:
  - Iterate until values converge
  $$V_{t+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[ R(s, \pi_k(s), s') + \gamma V_t^{\pi_k}(s') \right]$$

- **Policy improvement:** with fixed utilities, find the best action according to one-step look-ahead
  $$\pi_{k+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_t(s') \right]$$

---

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

---

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 $\gamma$)
  - Start state $s_0$

- **Quantities:**
  - Returns = sum of discounted rewards
  - Values = expected future returns from a state (optimal, or for a fixed policy)
  - Q-Values = expected future returns from a q-state (optimal, or for a fixed policy)