CS 188: Artificial Intelligence
Fall 2006

Lecture 10: MDPs II
9/28/2006

Dan Klein – UC Berkeley

Announcements

- Midterm prep page is up
- Project 2.1 will be up in a day or so
  - Due after midterm
  - But start it now, because MDPs are on the midterm!
- Project 1.4 (Learning Pacman) and the Pacman contest will be after the midterm
- Review session TBA, check the web page
Recap: MDPs

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

- **Examples:**
  - Gridworld, High-Low, Pacman, N-Armed Bandit
  - Any process where the result of your action is stochastic

- **Goal:** find the “best” policy $\pi$
  - Policies are maps from states to actions
  - What do we mean by “best”?
  - This is like search – it’s planning using a model, not actually interacting with the environment

---

Example: Autonomous Helicopter

![Image of autonomous helicopter]
Example: High-Low

High-Low

T = 0.5, R = 2
T = 0.25, R = 3
T = 0, R = 4
T = 0.25, R = 0

MDP Search Trees

- Can view an MDP as a branching search tree

(s, a) is a q-state

(s, s′) called a transition

T(s, a, s′) = P(s′|s,a)
R(s, a, s′)
Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

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

- We want to find the optimal policy $\pi$.

- Option 1: modified expectimax search:

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

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

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

MDP Search Trees

- Problems:
  - This tree is usually infinite (why?)
  - The same states appear over and over (why?)

- Solution:
  - Compute to a finite depth (like expectimax)
  - Consider returns from sequences of increasing length
  - Cache values so we don't repeat work
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 $k \to \infty$, 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

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?
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

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

---

Example: Bellman Updates

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

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

$$= 0.9 \left[ 0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 \right]$$
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:

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

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

### If we have a fixed policy \( \pi \), use simplified Bellman equation to calculate utilities:

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

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

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

Reinforcement Learning

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

- 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
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

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to $U(s)$ using a neural network
- Combined with depth 3 search, one of the top 3 players in the world
- You could imagine training Pacman this way...
- … but it’s tricky!
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 \( \pi(s) \)
  - **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 the general case soon

---

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

\[ U(1,1) \sim \frac{93 + (-105)}{2} = -6 \]
\[ U(3,3) \sim \frac{100 + 98 + (-101)}{3} = 32.3 \]
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 outcomes for each \(s,a\)
    - Normalize 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, e.g. "stationary noise")

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 +100
  - \((done)\)

\[
T(\langle 3,3\rangle, \text{right, } \langle 4,3\rangle) = 1 / 3
\]

\[
T(\langle 2,3\rangle, \text{right, } \langle 3,3\rangle) = 2 / 2
\]