CS 188: Artificial Intelligence Spring 2009

Lecture 11: Reinforcement Learning

2/24/2009

John DeNero – UC Berkeley

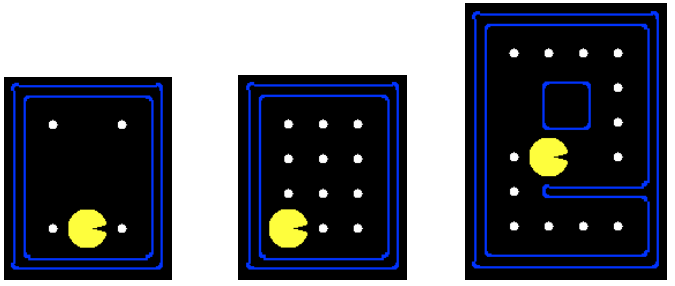
Slides adapted from Dan Klein, Stuart Russell or Sutton & Barto

Announcements

- Project 3:
  - Due a week from Wednesday: 3/4
- Written Assignment 2 timing change:
  - Posted on Thursday
  - Due on Thursday 3/12
- Grades should be posted for P0, P1 & WA1
  - If you have questions or comments, email the staff list
  - Good work! Mean scores are in the A- range

2

Commonly Failed Admissibility Tests



8 steps 11 steps 16 steps

If you used global variables & you think our tests cheated you, ask us

Nice Pacman Agent!

- The best expectimax agents
  - Runner-up: Chan Kuan 1504 pts, 93% won
    - Dying is very bad, eating ghosts is good, etc.
  - Winner: Tim Ingram 1691 pts, 97% won
    - If it's a terminal state, returns the score; otherwise,
    - Current score + 500 pts (for winning) +
    - 50 pts per ghost per uneaten power pellet +
    - 150-d pts per scared ghost +
    - 9 points per uneaten food +
    - A little extra for being close to food

[Demo]

Reinforcement Learning

Known	Unknown	Assumed
•Current state •Available actions •Rewards experienced	•Transition model •Reward structure	•Markov transitions •Fixed reward for (s,a,s')

Problem: Find values for fixed policy π (policy evaluation)

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

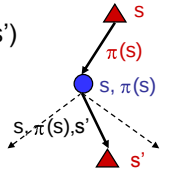
Model-based learning: Learn the model, solve for values

Model-free learning: Solve for values directly (by sampling)

5

Model-Based Learning

- Idea:
  - Learn the model empirically (rather than values)
  - Solve for values as if the learned model were correct
- Simple empirical model learning
  - Count outcomes for each s,a
  - Normalize to give estimate of **T(s,a,s')**
  - Discover **R(s,a,s')** when we experience (s,a,s')
- Solving the MDP with the learned model
  - Iterative policy evaluation, for example



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

6

Example: One Bowl of Fruit

- MDP:
  - State space: What's not in the bowl (2 states)
  - 1 action: Put everything in the bowl, then take something out
  - Rewards: R(s,a,s') only cares about s': +1 for Orange; +0 for Lemon
  - Transition model: Unknown!
  - Assume a discount factor of 0.5
- Let's sample: $\hat{P}(s' = \text{Orange}) = \frac{\#\text{ of Orange}}{\text{total draws}}$

7

Example: One Bowl of Fruit

- Evaluating the policy is easy here!
- Theorem: linearity of expectations
  - Expectation of a sum equals the sum of expectations

$$\mathrm{E}[a \cdot f(X) + b \cdot g(X)] = a \cdot \mathrm{E}[f(X)] + b \cdot \mathrm{E}[g(X)]$$

- Solve policy evaluation equation

$$\begin{aligned} V^{\pi}(\text{Orange}) &= \mathrm{E}[R(s') + 0.5 \cdot V^{\pi}(s')] \\ &= \mathrm{E}[R(s')] + 0.5 \cdot \mathrm{E}[V^{\pi}(s')] \\ &= \hat{P}(s' = \text{Orange}) \cdot 1 + 0.5 \cdot V^{\pi}(\text{Orange}) \\ &= 2 \cdot \hat{P}(s' = \text{Orange}) \end{aligned}$$

8

Example: Two Bowls of Fruit

- MDP:
  - State space: What's in each bowl & outside
  - 1 action: Flip to choose a bowl; put what's outside into that bowl and then pick a fruit
  - Rewards: R(s,a,s') only cares about s': +1 for Orange; +0 for Lemon
  - Transition model: Unknown and complicated!
  - Assume a discount factor of 0.5
- We could estimate transition probabilities, but there are a lot of them

9

Expectations from Samples

- Values of states are *expectations* of the sum of discounted future rewards
- What we've been doing so far:
  - Estimate probabilities from counts
  - Compute expectation from these estimates
- Alternative: estimate expectations directly
  - Average samples of the value function
  - In math:

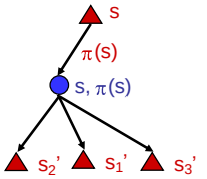
$$\hat{E}[f(X)] = \frac{\sum_{i=1}^{n} f(x_i)}{n}$$

10

Sample Avg to Replace Expectation?

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

- Who needs T and R? Approximate the expectation with samples (drawn from T!)

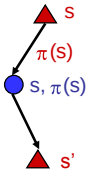


$$sample_1 = R(s, a, s'_1) + \gamma V^{\pi}_{i}(s'_1)$$
$$sample_2 = R(s, a, s'_2) + \gamma V^{\pi}_{i}(s'_2)$$
$$\cdots$$
$$sample_k = R(s, a, s'_k) + \gamma V^{\pi}_{i}(s'_k)$$

Problem: We need to estimate these too!

Model-Free Learning

- Big idea: why bother learning T?
  - Update V(s) each time we experience (s,a,s')
  - Likely s' will contribute updates more often



- Temporal difference learning (TD or TD(0))
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!

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

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha)sample$

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

Exponential Moving Average

- Weighted averages emphasize certain samples

$$\frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

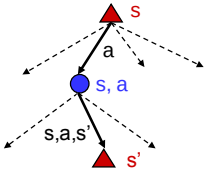
  - Forgets about the past (which contains mistakes in TD)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

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:



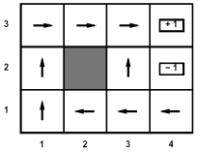
$$\pi(s) = \arg\max_a Q^*(s,a)$$

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

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

Active Learning

- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You can choose any actions you like
  - Goal: learn the optimal policy (maybe values)



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning!

- Concerns:
  - Learning time
  - Negative rewards during exploration sometimes matter

Q-Value Iteration

- Value iteration: find successive approx optimal values
  - 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:

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

- But Q-values are more useful!
  - Start with $Q_0^*(s,a) = 0$, which we know is right (why?)
  - Given Q_i^*, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a') \right]$$

[DEMO – Grid Q's]

Q-Learning

- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

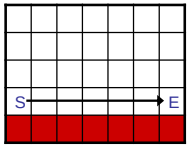
  - Incorporate the new estimate into a running average:

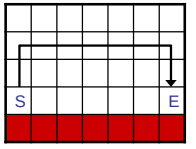
$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \, [sample]$$

17

Q-Learning Properties

- Will converge to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - But not decrease it too quickly!
  - Basically doesn't matter how you select actions (!)
- Neat property: learns optimal q-values regardless of action selection noise (some caveats)





18

[DEMO – RL Pacman]

Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Exploration Functions

- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established
- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$ (exact form not important)

$$Q_{i+1}(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

$$Q_{i+1}(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a'))$$

[DEMO – Crawler Q's]

Q-Learning

- Q-learning produces tables of q-values:

The Story So Far: MDPs and RL

Things we know how to do:

- We can solve small MDPs exactly
- If we don't know T(s,a,s'), we can estimate it, then solve the MDP
- We can estimate values $V^{\pi}(s)$ directly for a fixed policy π.
- We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- Value and policy Iteration
- Adaptive dynamic programming
- Temporal difference learning
- Q-learning
- Exploration functions

What About Large Known MDPs

- Simulated Q-learning is a good bet
  - Q-learning storage is only O(|S|*|A|): might be smaller than the MDP
  - Solving policy evaluation equations is $O(|S|^3)$
  - Every value iteration update to V(s) is O(|A|*|S|)
  - A Q-learning update to Q(s,a) is O(|A|)
- When simulating, you can make q-learning updates to any (s,a) in any order

23

The Problem with Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

24