CS 188: Artificial Intelligence
Fall 2011

Lecture 10: Reinforcement Learning

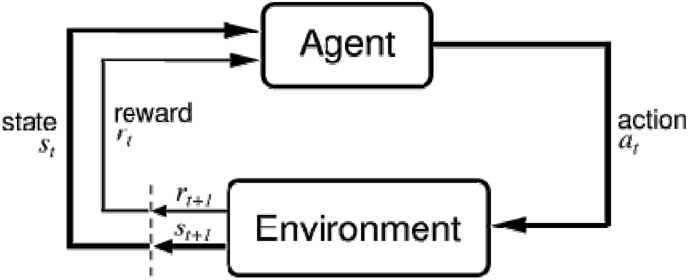
9/27/2011

Dan Klein – UC Berkeley

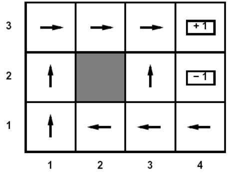
Many slides over the course adapted from either Stuart Russell or Andrew Moore

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



Passive RL



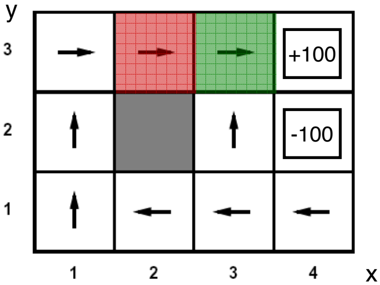
- Simplified task
  - You are given a policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values
  - … what policy evaluation did
- In this case:
  - Learner "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Example: Direct Evaluation

- 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 -1
(4,3) exit +100
(done)

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



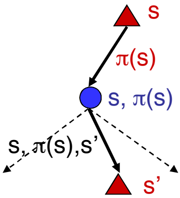
$\gamma = 1$, R = -1

V(2,3) ~ (96 + -103) / 2 = -3.5

V(3,3) ~ (99 + 97 + -102) / 3 = 31.3

Recap: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate V for a fixed policy:
  - New V is expected one-step-look-ahead using current V
  - Unfortunately, need T and R

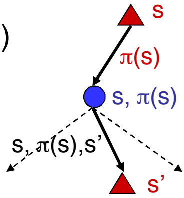


$$V_0^\pi(s) = 0$$

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

Model-Based Learning

- Idea:
  - Learn the model empirically through experience
  - 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



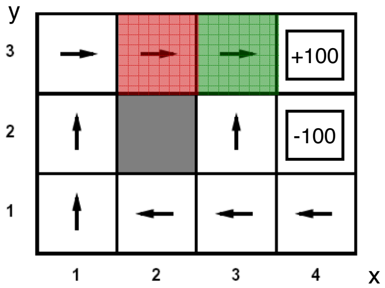
$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$

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 -1
(4,3) exit +100
(done)

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



$\gamma = 1$

T(<3,3>, right, <4,3>) = 1 / 3

T(<2,3>, right, <3,3>) = 2 / 2

Model-Free Learning

- Want to compute an expectation weighted by P(x):

$$E[f(x)] = \sum_x P(x) f(x)$$

- Model-based: estimate P(x) from samples, compute expectation

$$x_i \sim P(x)$$

$$\hat{P}(x) = \text{num}(x)/N$$

$$E[f(x)] \approx \sum_x \hat{P}(x) f(x)$$

- Model-free: estimate expectation directly from samples

$$x_i \sim P(x)$$

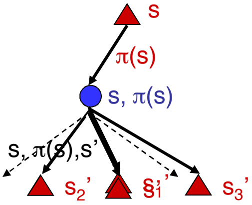
$$E[f(x)] \approx \frac{1}{N} \sum_i f(x_i)$$

- Why does this work? Because samples appear with the right frequencies!

Sample-Based Policy Evaluation?

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

- Who needs T and R? Approximate the expectation with samples of s' (drawn from T!)



$$sample_1 = R(s, \pi(s), s_1') + \gamma V_i^{\pi}(s_1')$$

$$sample_2 = R(s, \pi(s), s_2') + \gamma V_i^{\pi}(s_2')$$

$$\ldots$$

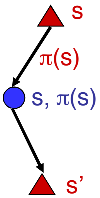
$$sample_k = R(s, \pi(s), s_k') + \gamma V_i^{\pi}(s_k')$$

$$V_{i+1}^{\pi}(s) \leftarrow \frac{1}{k} \sum_i sample_i$$

Almost! But we can't rewind time to get sample after sample from state s.

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience (s,a,s',r)
  - Likely s' will contribute updates more often
- Temporal difference learning
  - 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

- Exponential moving average
  - The running interpolation update

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

  - Makes recent samples more important

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

  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate can give converging averages

Example: TD Policy Evaluation

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

(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 -1

(4,3) exit +100

(done)

(1,1) up -1

(1,2) up -1

(1,3) right -1

(2,3) right -1

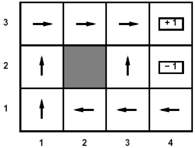
(3,3) right -1

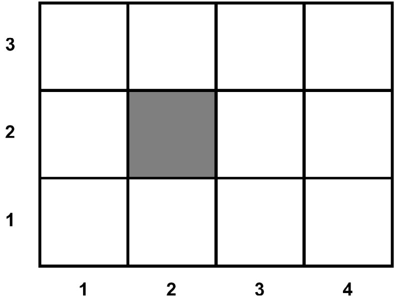
(3,2) up -1

(4,2) exit -100

(done)

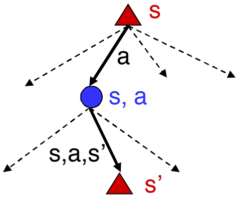
Take $\gamma = 1$, $\alpha = 0.5$





Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we're sunk:

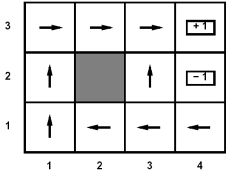


$$\pi(s) = \arg\max_a Q^*(s,a)$$

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

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

Active RL



- 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 / values
  - … what value iteration did!

- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Detour: 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:

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

Q-Learning

- Q-Learning: sample-based Q-value iteration
- 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:

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

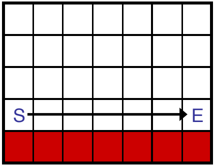
Q-Learning

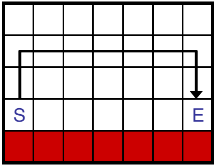
- Q-learning produces tables of q-values:

Q-Learning Properties

- Amazing result: Q-learning converges 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: off-policy learning
  - learn optimal policy without following it (some caveats)





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)

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

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

The Story So Far: MDPs and RL

Things we know how to do:

- If we know the MDP
  - Compute V*, Q*, π* exactly
  - Evaluate a fixed policy π
- If we don't know the MDP
  - We can estimate the MDP then solve
  - We can estimate V for a fixed policy π
  - We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- Model-based DPs
  - Value Iteration
  - Policy evaluation
- Model-based RL
- Model-free RL
  - Value learning
  - Q-learning

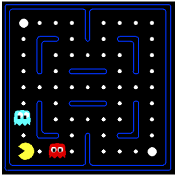
Q-Learning

- 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

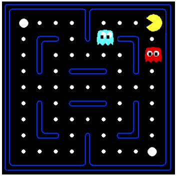
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Example: Pacman

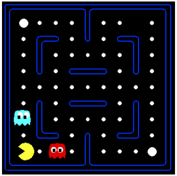
- Let's say we discover through experience that this state is bad:



- In naïve q learning, we know nothing about this state or its q states:

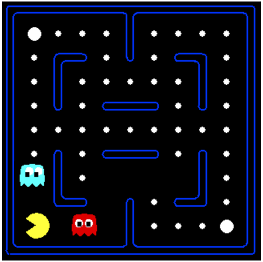


- Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

Function Approximation

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear q-functions:

$$transition \;= (s,a,r,s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$ Exact Q's

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$ Approximate Q's

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

$$f_{DOT}(s, \text{NORTH}) = 0.5$$

$$f_{GST}(s, \text{NORTH}) = 1.0$$

$$Q(s,a) = +1$$

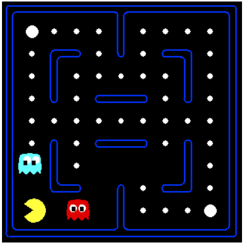
$$R(s,a,s') = -500$$

$$difference = -501$$

$$w_{DOT} \leftarrow 4.0 + \alpha \, [-501] \, 0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha \, [-501] \, 1.0$$

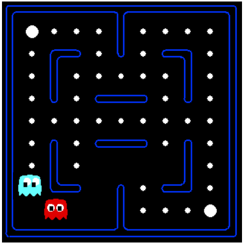
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$



s

$a = \text{NORTH}$

$r = -500$



s'