

# CS 188: Artificial Intelligence

## Fall 2006

Lecture 11: Reinforcement Learning  
10/3/2006

Dan Klein – UC Berkeley

## Announcements

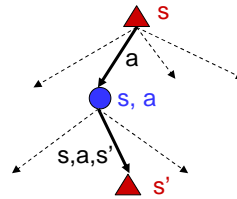
---

- Midterm prep page is up
- Tell us NOW about midterm conflicts
- Project 2.1 up soon, due after midterm
- Project 1.4 (Learning Pacman) and the Pacman contest will be after the midterm
- Review session this Sunday, details on web

## 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  $s_0$



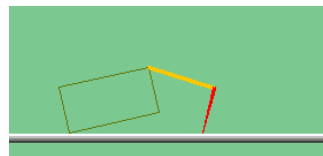
- Quantities:

- 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)

## Reinforcement Learning

- Reinforcement learning:

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



[DEMO]

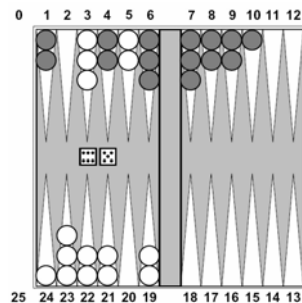
- New twist: **don't know  $T$  or  $R$** 
  - I.e. don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

## 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

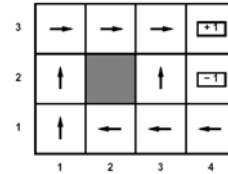
## Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to  $V(s)$  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

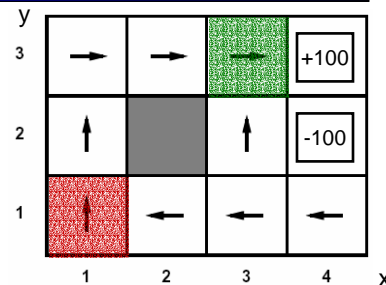
- **Simplified task**
  - You don't know the transitions  $T(s,a,s')$
  - You don't know the rewards  $R(s,a,s')$
  - You are given a policy  $\pi(s)$
  - **Goal: learn the state values** (and maybe the model)
  
- **In this case:**
  - No choice about what actions to take
  - Just execute the policy 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) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$\gamma = 1, R = -1$

$$U(1,1) \sim (92 + -106) / 2 = -7$$

$$U(3,3) \sim (99 + 97 + -102) / 3 = 31.3$$

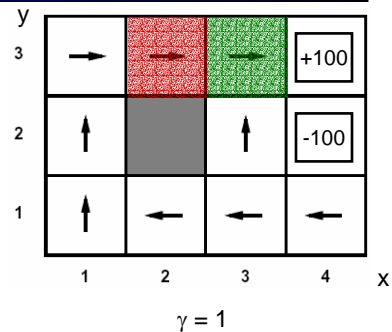
# Model-Based Learning

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



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

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

## Model-Based Learning

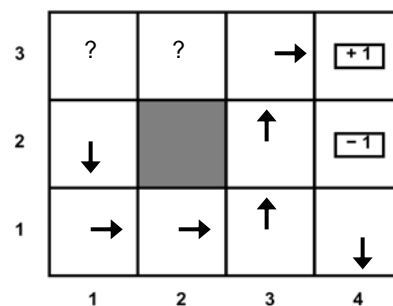
---

- In general, want to learn the optimal policy, not evaluate a fixed policy
- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model

## Example: Greedy ADP

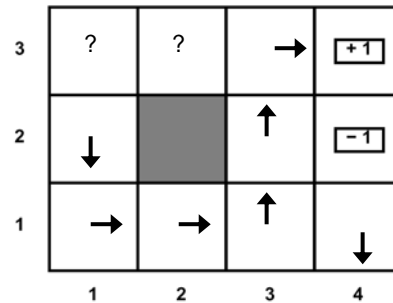
---

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We'll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy



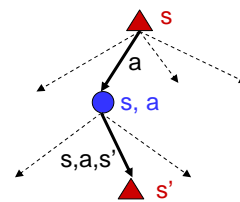
# What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit



# Model-Free Learning

- Big idea: why bother learning  $T$ ?
  - Update each time we experience a transition
  - Frequent outcomes will contribute more updates (over time)
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs



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

$$sample = R(s, a, s') + \gamma V^\pi(s')$$

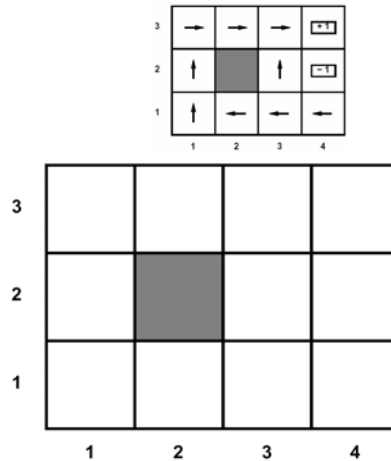
$$V^\pi(s) \leftarrow V^\pi(s) + \alpha (sample - V^\pi(s))$$

## Example: Passive TD

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha [R(s, a, s') + \gamma V^\pi(s') - V^\pi(s)]$$

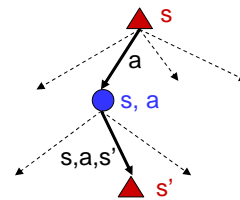
(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	

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



## Problems with TD Value Learning

- TD value learning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy, we're sunk:



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

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

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

# 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:

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

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

- Nudge the old estimate towards the new sample:

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

# Q-Learning Example

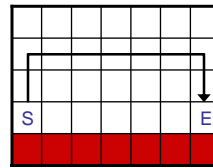
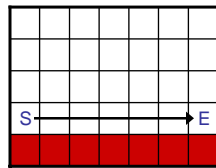
---

- [DEMO]

## Q-Learning Properties

---

- Will converge to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
- Neat property: does not learn policies which are optimal in the presence of action selection noise



## Exploration / Exploitation

---

- Several schemes for forcing exploration
  - Simplest: random actions ( $\epsilon$ -greedy)
    - Every time step, flip a coin
    - With probability  $\epsilon$ , act randomly
    - With probability  $1-\epsilon$ , 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  $\epsilon$  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)

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

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