

CS 188: Artificial Intelligence

Fall 2007

Lecture 12: Reinforcement Learning

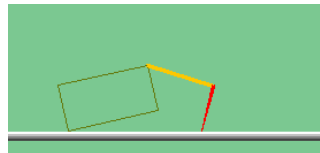
10/4/2007

Dan Klein – UC Berkeley

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]

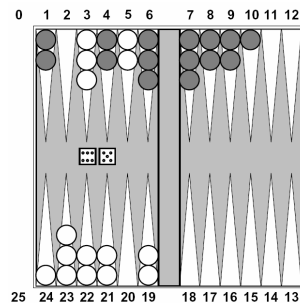
- New twist: **don't know T or R**
 - I.e. don't know which states are good or what the actions do
 - Must actually try actions and states out to learn

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

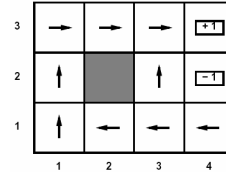
- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to $V(s)$ using a neural network
- Combined with depth 3 search, one of the top 3 players in the world
- You could imagine training Pacman this way...
- ... but it's tricky!



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)



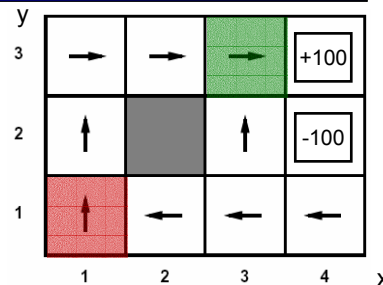
- **In this case:**

- 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) | |



$\gamma = 1, R = -1$

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

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

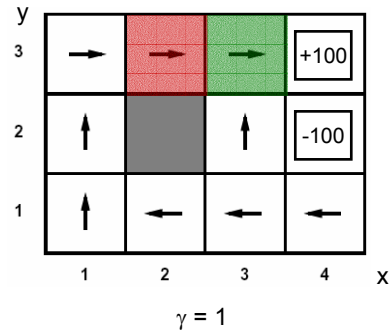
Model-Based Learning

- 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) | |



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

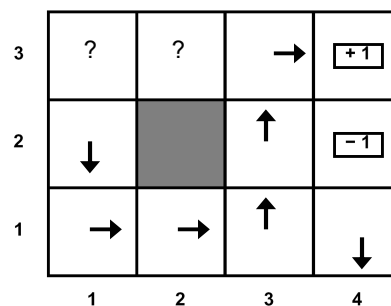
$$T(\langle 2,3 \rangle, \text{right}, \langle 3,3 \rangle) = 2 / 2$$

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



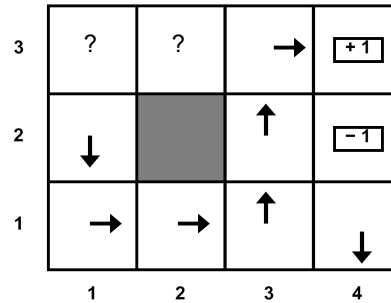
What Went Wrong?

- 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



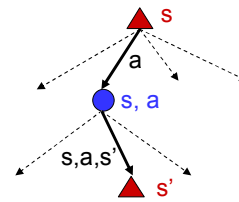
Model-Free Learning

- Big idea: why bother learning T?

- Update V 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



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

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

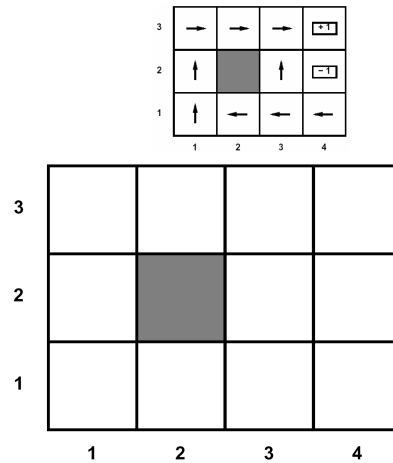
$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$$

Example: Passive TD

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha) [R(s, a, s') + \gamma V^\pi(s')]$$

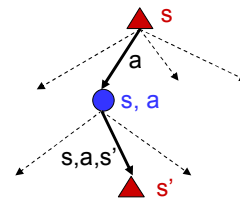
(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	

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:

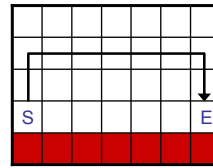
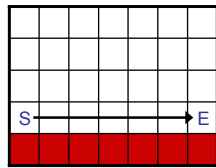
$$Q(s,a) \leftarrow (1 - \alpha)Q(s,a) + (\alpha) [sample]$$

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 (ϵ greedy)
 - Every time step, flip a coin
 - With probability ϵ , 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 ϵ over time
 - Another solution: exploration functions

Exploration Functions

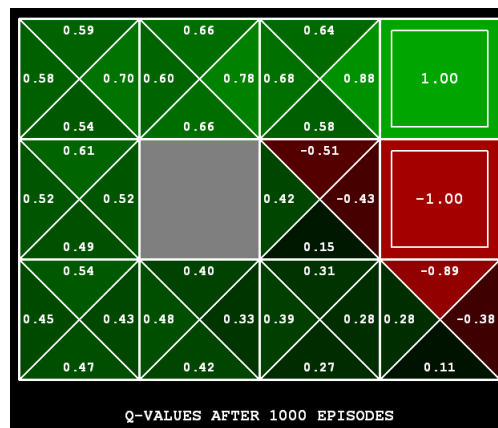
- When to explore
 - Random actions: explore a fixed amount
 - Better idea: explore areas whose badness is not (yet) established
- Exploration function
 - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$ (exact form not important)

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

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

Q-Learning

- Q-learning produces tables of q-values:

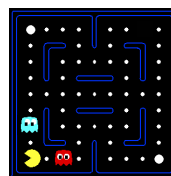
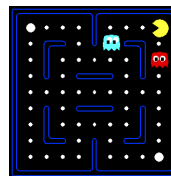


Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
 - Too many states to visit them all in training
 - Too many states to even 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

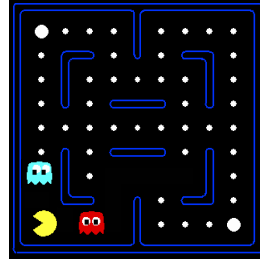
Example: Pacman

- 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
 - 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
 - $1 / (\text{dist to dot})^2$
 - Is Pacman in a tunnel? (0/1)
 - etc.
 - 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:

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

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

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

Function Approximation

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

- Q-learning with linear q-functions:

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

$$w_i \leftarrow w_i + \alpha [\text{error}] f_i(s, a)$$

- 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 (much later)

Example: Q-Pacman

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

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

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

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

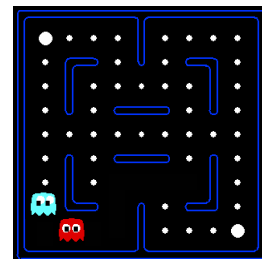
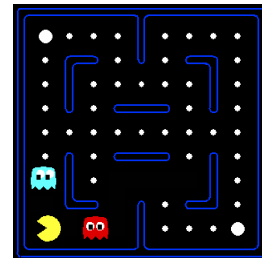
$$R(s, a, s') = -500$$

$$\text{error} = -501$$

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

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

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



Policy Search



Policy Search

- Problem: often the feature-based policies that work well aren't the ones that approximate V / Q best
 - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
 - We'll see this distinction between modeling and prediction again later in the course
- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
- This is the idea behind policy search, such as what controlled the upside-down helicopter

Policy Search

- **Simplest policy search:**
 - Start with an initial linear value function or q-function
 - Nudge each feature weight up and down and see if your policy is better than before
- **Problems:**
 - How do we tell the policy got better?
 - Need to run many sample episodes!
 - If there are a lot of features, this can be impractical

Policy Search*

- **Advanced policy search:**
 - Write a stochastic (soft) policy:

$$\pi_w(s) \propto e^{\sum_i w_i f_i(s,a)}$$

- Turns out you can efficiently approximate the derivative of the returns with respect to the parameters w (details in the book, but you don't have to know them)
- Take uphill steps, recalculate derivatives, etc.

Take a Deep Breath...

- We're done with search and planning!
- Next, we'll look at how to reason with probabilities
 - Diagnosis
 - Tracking objects
 - Speech recognition
 - Robot mapping
 - ... lots more!
- Last part of course: machine learning