

CS 188: Artificial Intelligence Spring 2006

Lecture 22: Reinforcement Learning II 4/13/2006

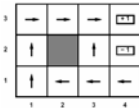
Dan Klein – UC Berkeley

Today

- **Reminder: P3 lab Friday, 2 4pm, 275 Soda**
- **Reinforcement learning**
 - Temporal-difference learning
 - Q-learning
 - Function approximation

Recap: Passive Learning

- **Learning about an unknown MDP**
- **Simplified task**
 - You don't know the transitions $T(s,a,s')$
 - You don't know the rewards $R(s)$
 - You DO know the policy $\pi(s)$
 - **Goal: learn the state values** (and maybe the model)
- **Last time: try to learn T, R and then solve as a known MDP**



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

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

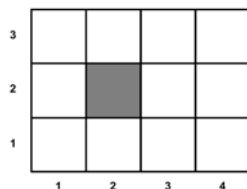
$$U^\pi(s) \leftarrow U^\pi(s) + \alpha (R(s) + \gamma U^\pi(s') - U^\pi(s))$$

$$U^\pi(s) \leftarrow (1 - \alpha)U^\pi(s) + \alpha (R(s) + \gamma U^\pi(s'))$$

Example: Passive TD

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha (R(s) + \gamma U^\pi(s') - U^\pi(s))$$

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



Take $\gamma = 1, \alpha = 0.1$

(Greedy) Active Learning

- **In general, want to learn the optimal policy**
- **Idea:**
 - 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

Example: Greedy Active Learning

- 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

3	?	?	→	+1
2	↓		↑	-1
1	→	→	↑	↓
	1	2	3	4

What Went Wrong?

- Problem with following optimal policy for current model:
 - Never learn about better regions of the space
- 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

3	?	?	→	+1
2	↓		↑	-1
1	→	→	↑	↓
	1	2	3	4

Q-Functions

- Alternate way to learn:
 - Utilities for state-action pairs rather than states
 - AKA Q-functions

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s')$$

$$U(s) = \max_a Q(a, s)$$

$$U(3, 2) = 0.660 \quad \pi(3, 2) = \text{up}$$

$$Q(\text{up}, \{3, 2\}) = 0.660$$

$$Q(\text{right}, \{3, 2\}) = -0.535$$

3	0.812	0.868	0.912	+1
2	0.762			-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Learning Q-Functions: MDPs

- Just like Bellman updates for state values:

$$Q_{i+1}^{\pi}(a, s) \leftarrow R(s) + \gamma \sum_{s'} T(s, a, s') Q_i^{\pi}(\pi(s'), s')$$

- For optimal policy

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

- Main advantage of Q functions over values U is that you don't need a model for learning or action selection!

Q-Learning

- Model free, TD learning with Q-functions:

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

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

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

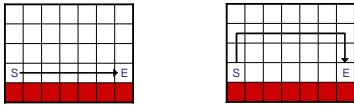
Example

- [DEMOS]

Exploration / Exploitation

- Several schemes for forcing exploration
 - Simplest: random actions
 - Every time step, flip a coin
 - With probability ϵ , act randomly
 - With probability $1-\epsilon$, act according to current policy

- Problems with random actions?



- Will take a non-optimal long route to reduce risk which stems from exploration actions!
- Solution: lower ϵ over time

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. $f(u, n) = u + k/n$

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

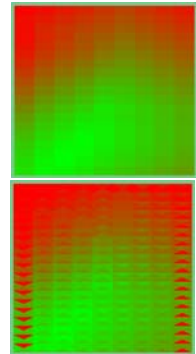
$$Q_{i+1}(a, s) \leftarrow (1 - \alpha)Q_i(a, s) + \alpha \left(R(s) + \gamma \max_{a'} f(Q_i(a', s'), N(a', s')) \right)$$

Function Approximation

- Problem: too slow to learn each state's utility one by one
- Solution: what we learn about one state should generalize to similar states
 - Very much like supervised learning
 - If states are treated entirely independently, we can only learn on very small state spaces

Discretization

- Can put states into buckets of various sizes
 - E.g. can have all angles between 0 and 5 degrees share the same Q estimate
 - Buckets too fine \Rightarrow takes a long time to learn
 - Buckets too coarse \Rightarrow learn suboptimal, often jerky control
- Real systems that use discretization usually require clever bucketing schemes
 - Adaptive sizes
 - Tile coding
- [DEMOS]



Linear Value Functions

- Another option: values are linear functions of features of states (or action-state pairs)

$$\bar{U}_{\theta}(s) = \sum_k \theta_k f_k(s)$$

- Good if you can describe states well using a few features (e.g. for game playing board evaluations)

- Now we only have to learn a few weights rather than a value for each state

3	0.812	0.868	0.912	0.95
2	0.762	0.812	0.860	0.90
1	0.705	0.655	0.611	0.388
	1	2	3	4

3	0.80	0.85	0.90	0.95
2	0.70	0.80	0.85	0.90
1	0.60	0.65	0.70	0.75
	1	2	3	4

$$\bar{U}_{\theta}(s) = 0.3 + 0.05x + 0.1y$$

TD Updates for Linear Values

- Can use TD learning with linear values
 - (Actually it's just like the perceptron!)
 - Old Q-learning update:

$$Q(a, s) \leftarrow Q(a, s) + \alpha \left(R(s) + \gamma \max_{a'} Q(a', s') - Q(a, s) \right)$$

- Simply update weights of features in $Q_{\theta}(a, s)$

$$\theta_k \leftarrow \theta_k + \alpha \left(R(s) + \gamma \max_{a'} Q_{\theta}(a', s') - Q_{\theta}(a, s) \right) f_k(a, s)$$