CS 188: Artificial Intelligence Spring 2009

Lecture 12: Reinforcement Learning II

2/26/2009

John DeNero – UC Berkeley

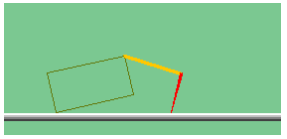
Slides adapted from Dan Klein, Stuart Russell or Sutton & Barto

Announcements

- Project 3:
  - Due a week from yesterday
  - Submission will be enabled tomorrow
- Written Assignment 2:
  - Posted later today or tomorrow
  - Due *in lecture* on Thursday 3/12

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 $\pi(s)$
  - 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



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

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:

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

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

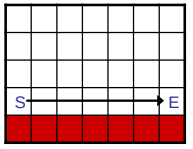
  - Incorporate the new estimate into a running average:

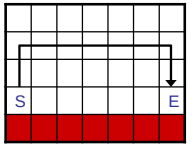
$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \, [sample]$$

5

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 while executing sub-optimal policies





6

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
    - Takes a long time to explore certain spaces
    - 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)

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

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

[Videos]

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

9

Example: Pacman

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

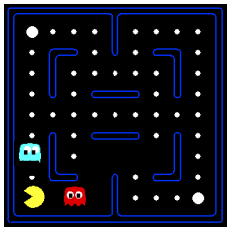
- In naïve q learning, we know nothing about this state or its q states:

- Or even this one!

10

Feature-Based Representations

- Solution: describe a (state,action) pair using a vector of features
  - Features are functions from q-states to real numbers that capture important properties of the state
  - Simple features for the project:
    - Will I collide with the ghost?
    - Distance to closest dot
    - Does the action eat food?
    - Number of ghosts within one step
  - A feature vector is just a Python dict
  - For the algorithm you're about to see to converge reliably, features should be between -1 and 1



+

'North'

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 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:

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

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

$$w_i \leftarrow w_i + \alpha\,[correction]\, f_i(s,a)$$

- 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

13

Example: Q-Pacman

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

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

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

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

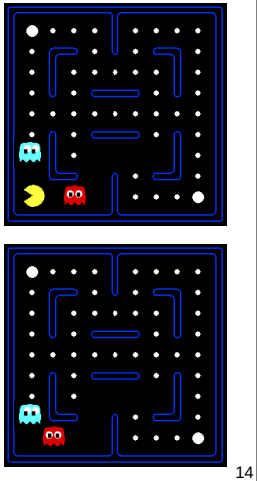
$$R(s,a,s') = -500$$

$$correction = -501$$

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

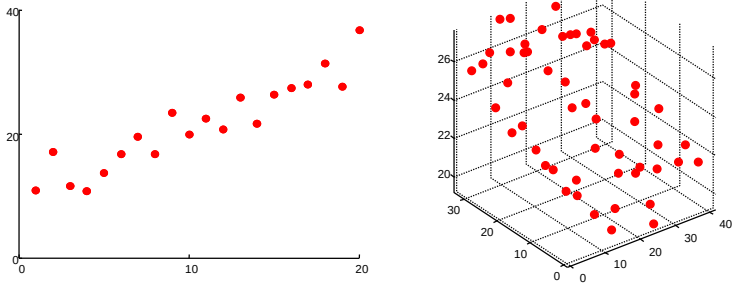
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

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



14

Linear Regression

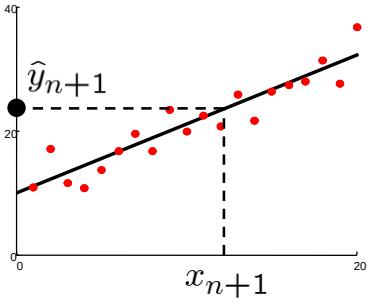


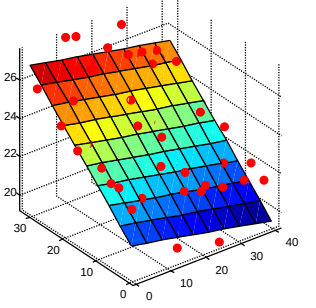
Given examples $(x_i, y_i)_{i=1...n}$

Predict y_{n+1} given a new point x_{n+1}

15

Linear Regression



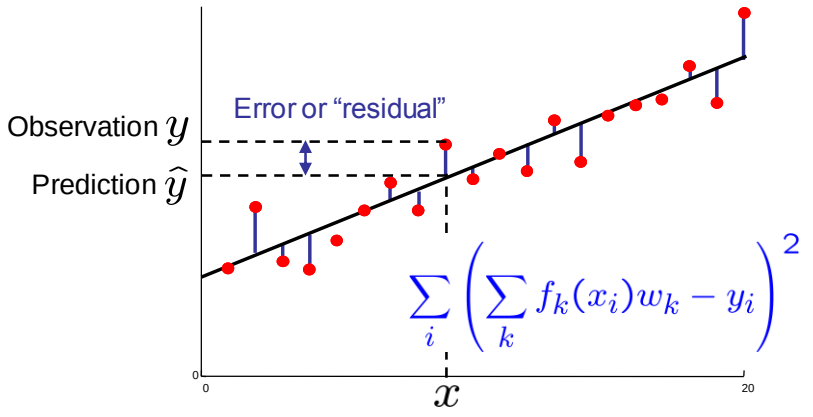


Prediction
$\widehat{y}_i = w_0 + w_1 x_i$

Prediction
$\widehat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2}$

16

Ordinary Least Squares (OLS)



17

Minimizing Error

$$E(w) = \frac{1}{2} \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right)^2$$

$$\frac{\partial E}{\partial w_m} = \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

$$w_m \leftarrow w_m - \alpha \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

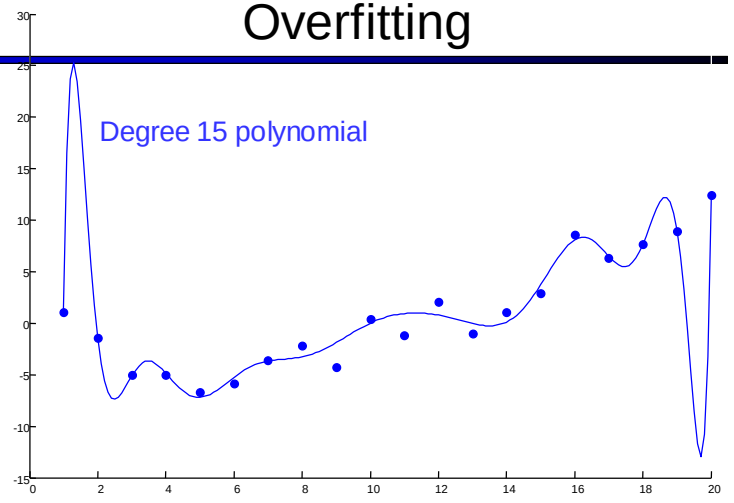
Approximate q update explained:

$$w_m \leftarrow w_m - \alpha \, [error] \, f_m(s,a)$$

$$w_m \leftarrow w_m + \alpha \, [correction] \, f_m(s,a)$$

18

Overfitting



[DEMO]

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

20

Policy Search

- [DEMO – Helicopter]
- Idea: learn the policy that maximizes utility rather than the value that predicts returns
- Justification: exact values often don't matter for making good decisions

21

Policy Search*

- Simplest policy search:
  - Start with an initial linear q-function based on features

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

  - 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

24

Policy Gradient Search*

- Policy Gradient Search:
  - Start with an initial linear q-function based on features

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

  - Compute the change in *w* that will increase utility the fastest -- gradient of utility with respect to *w*
  - After changing *w*, you need to recompute gradient
- Problem:
  - The utility is not a continuous function of the parameters, because a small weight change can change the whole policy

Policy Gradient Search*

- Solution to discontinuous reward function:
  - Use a probabilistic 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, optional material)
  - 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 through time
  - Complex interactions and domains
  - Pacman won't know where the ghosts are!
- Last part of course: machine learning

27