

# CS 188: Artificial Intelligence Spring 2011

## Lecture 11: Reinforcement Learning II 2/28/2010

Pieter Abbeel – UC Berkeley  
Many slides over the course adapted from either Dan Klein,  
Stuart Russell or Andrew Moore

## Announcements

- W2: due right now
- Submission of self-corrected copy for partial credit due Wednesday 5:29pm
- P3 Reinforcement Learning (RL):
  - Out, due Monday 4:59pm
  - You get to apply RL to:
    - Gridworld agent
    - Crawler
    - Pac-man
- Recall: readings for current material
  - Online book: Sutton and Barto

<http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html>

## MDPs and RL Outline

- Markov Decision Processes (MDPs)
  - Formalism
  - Value iteration
  - Expectimax Search vs. Value Iteration
  - Policy Evaluation and Policy Iteration
- Reinforcement Learning
  - Model-based Learning
  - Model-free Learning
    - Direct Evaluation [performs policy evaluation]
    - Temporal Difference Learning [performs policy evaluation]
    - Q-Learning [learns optimal state-action value function  $Q^*$ ]
  - Exploration vs. exploitation

3

## Reinforcement Learning

- Still assume a Markov decision process (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

4

## Reinforcement Learning

- Reinforcement learning:
  - Still assume 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

5

## Example: learning to walk



Before learning (hand-tuned)

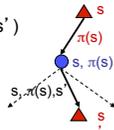
One of many learning runs

After learning  
[After 1000  
field  
traversals]

[Kohl and Stone, ICRA 2004]

## 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
  - Value iteration, or policy iteration

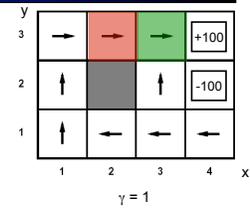


7

## Example: Learn Model in 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)          |



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

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

8

## Model-based vs. Model-free

- Model-based RL
  - First act in MDP and learn  $T, R$
  - Then value iteration or policy iteration with learned  $T, R$
  - Advantage: efficient use of data
  - Disadvantage: requires building a model for  $T, R$
- Model-free RL
  - Bypass the need to learn  $T, R$
  - Methods to evaluate a fixed policy without knowing  $T, R$ :
    - (i) Direct Evaluation
    - (ii) Temporal Difference Learning
  - Method to learn  $\pi^*, Q^*, V^*$  without knowing  $T, R$ 
    - (iii) Q-Learning

10

## Direct Evaluation

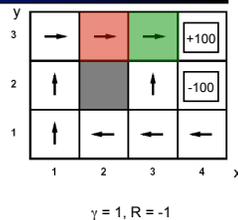
- Repeatedly execute the policy  $\pi$
- Estimate the value of the state  $s$  as the average over all times the state  $s$  was visited of the sum of discounted rewards accumulated from state  $s$  onwards

11

## Example: Direct Evaluation

### 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)          |



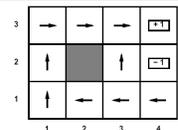
$$V(2,3) \sim (96 + -103) / 2 = -3.5$$

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

12

## Limitations of Direct Evaluation

- Assume random initial state
- Assume the value of state (1,2) is known perfectly based on past runs
- Now for the first time encounter (1,1) --- can we do better than estimating  $V(1,1)$  as the rewards outcome of that run?



13

## 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 (drawn from T!)

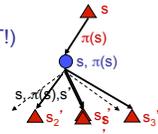
$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_i^\pi(s'_1)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_i^\pi(s'_2)$$

$$\dots$$

$$sample_k = R(s, \pi(s), s'_k) + \gamma V_i^\pi(s'_k)$$

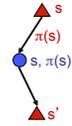
$$V_{i+1}^\pi(s) \leftarrow \frac{1}{k} \sum_i sample_i$$



Almost! (i) Will only be in state  $s$  once and then land in  $s'$  hence have only one sample  $\rightarrow$  have to keep all samples around? (ii) Where do we get value for  $s'$ ? 14

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

15

## Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important

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

- Forgets about the past (distant past values were wrong anyway)
- Easy to compute from the running average

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

- Decreasing learning rate can give converging averages

16

## Policy evaluation when T (and R) unknown --- recap

- Model-based:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct
- Model-free:
  - Direct evaluation:
    - $V(s)$  = sample estimate of sum of rewards accumulated from state  $s$  onwards
  - Temporal difference (TD) value learning:
    - Move values toward value of whatever successor occurs: running average!

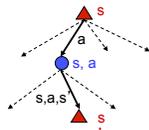
$$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$$

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

18

## Problems with TD Value Learning

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



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

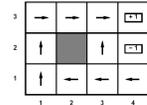
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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

19

## Active Learning

- 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
  - ... 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...

20

## 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  $i+1$ :

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

- 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  $i+1$ :

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

21

## 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:

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

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

- Incorporate the new estimate into a running average:

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

23

## 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

27

## 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

28

## 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)

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

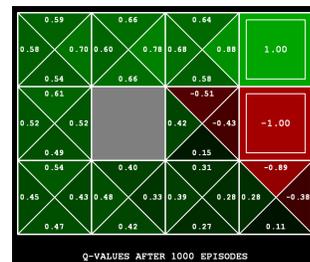
now becomes:

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

30

## Q-Learning

- Q-learning produces tables of q-values:



32

## The Story So Far: MDPs and RL

### Things we know how to do:

- We can solve small MDPs exactly, offline
- We can estimate values  $V^\pi(s)$  directly for a fixed policy  $\pi$ .
- We can estimate  $Q^*(s,a)$  for the optimal policy while executing an exploration policy

### Techniques:

- Value and policy iteration
- Temporal difference learning
- Q-learning
- Exploratory action selection

33

## 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

34

## Example: Pacman

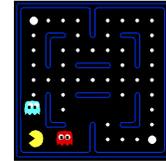
- Let's say we discover through experience that this state is bad:
- In naïve q learning, we know nothing about this state or its q states:
- Or even this one!



35

## 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)



36

## 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!

37

## 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:

$$\text{transition} = (s, a, r, s')$$

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

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

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

38

## Example: Q-Pacman

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

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

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

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

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

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

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

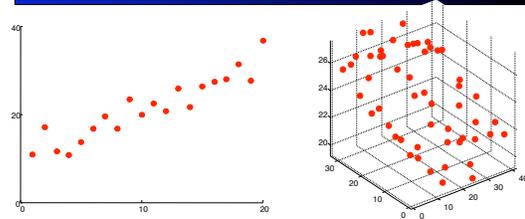
$$w_{GST} \leftarrow -1.0 + \alpha [-501] 1.0$$

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



39

## Linear regression

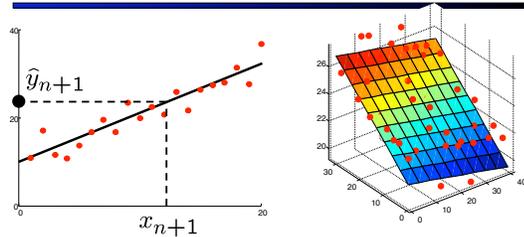


Given examples  $(x_i, y_i)_{i=1 \dots n}$

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

40

## Linear regression

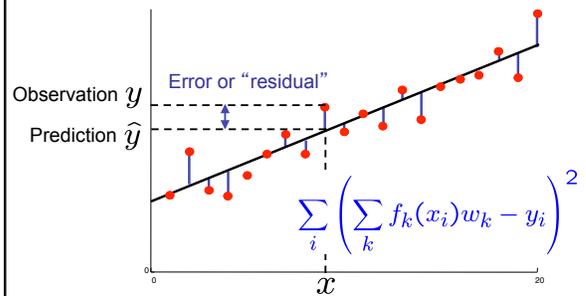


Prediction  
 $\hat{y}_i = w_0 + w_1 x_i$

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

41

## Ordinary Least Squares (OLS)



42

## Minimizing Error

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

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

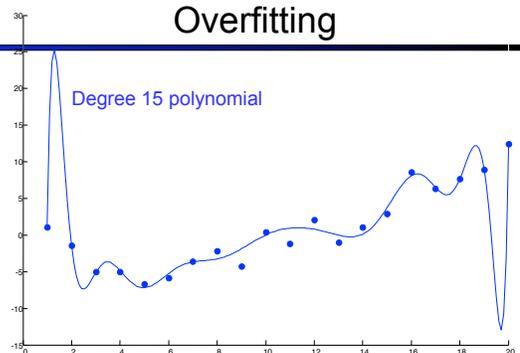
$$E \leftarrow E + \alpha \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

Value update explained:

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

43

## Overfitting



44

## Policy Search

---



45

## Policy Search

---

- Problem: often the feature-based policies that work well aren't the ones that approximate  $V$  /  $Q$  best
- 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

46

## 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

47

## MDPs and RL Outline

---

- Markov Decision Processes (MDPs)
  - Formalism
  - Value iteration
  - Expectimax Search vs. Value Iteration
  - Policy Evaluation and Policy Iteration
- Reinforcement Learning
  - Model-based Learning
  - Model-free Learning
    - Direct Evaluation [performs policy evaluation]
    - Temporal Difference Learning [performs policy evaluation]
    - Q-Learning [learns optimal state-action value function  $Q^*$ ]
    - Policy Search [learns optimal policy from subset of all policies]

48

## To Learn More About RL

---

- Online book: Sutton and Barto  
<http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html>
- Graduate level course at Berkeley has reading material pointers online:  
<http://www.cs.berkeley.edu/~russell/classes/cs294/s11/>

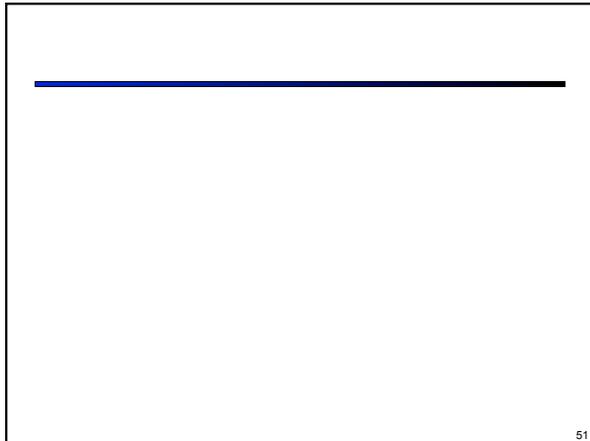
49

## 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!
- Third part of course: machine learning

50



## Helicopter dynamics

- **State:**  $(\dot{A}; \mu; \tilde{A}; \dot{\mu}; \tilde{A}; x; y; z; \underline{x}; \underline{y}; \underline{z})$   
 [ (roll, pitch, yaw, roll rate, pitch rate, yaw rate, x, y, z, x velocity, y velocity, z velocity) ]
- **Control inputs:**
  - Roll cyclic pitch control (tilts rotor plane)
  - Pitch cyclic pitch control (tilts rotor plane)
  - Tail rotor collective pitch (affects tail rotor thrust)
  - Collective pitch (affects main rotor thrust)
- **Dynamics:**
  - $s_{t+1} = f(s_t, a_t) + w_t$   
 [ encodes helicopter dynamics ]

## Helicopter policy class

$$a_1 = w_0 + w_1 \dot{A} + w_2 \underline{x} + w_3 \text{err}_x$$

$$a_2 = w_4 + w_5 \mu + w_6 \underline{y} + w_7 \text{err}_y$$

$$a_3 = w_8 + w_9 \tilde{A}$$

$$a_4 = w_{10} + w_{11} \underline{z} + w_{12} \text{err}_z$$

→ Total of 12 parameters

## Reward function

$$R(s) = \frac{1}{2} (x - x^*)^2 + \frac{1}{2} (y - y^*)^2 + \frac{1}{2} (z - z^*)^2 + \frac{1}{2} (\tilde{A} - \tilde{A}^*)^2$$

## Toddler (Tedrake + al.)\*



- Uses policy gradient from trials on actual robot
  - Leverages value function approximation to improve the gradient estimates
- **Policy parameterization:**
  - Dynamics analysis enables separation roll and pitch. Roll turns out the hardest control problem.
  - ankle roll torque  $\tau = w^* \dot{A}(q_{\text{roll}}, \dot{q}_{\text{roll}})$
  - $\dot{A}$  tiles  $(q_{\text{roll}}, \dot{q}_{\text{roll}})$  into  $5 \times 7$  --- i.e., encodes a look-up table
- On board sensing: 3 axis gyro, 2 axis tilt sensor

## Toddler (Tedrake et al.)

