

# CSE 473: Artificial Intelligence

## Reinforcement Learning



Dan Weld

University of Washington

[Most of these slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

## Midterm Postmortem

- It was long, hard... ☹
  - Max 41
  - Min 13
  - Mean & Median 27
- Final
  - Will include some of the midterm problems

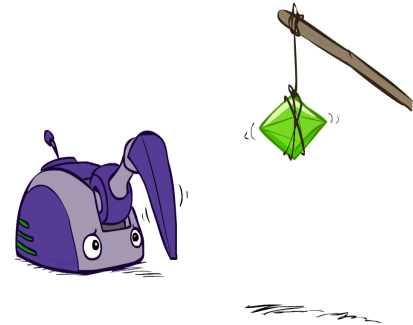
## Office Hour Change (this week)

- Thurs 10-11am
  - CSE 588
  - (Not Fri)



"Listen Simkins, when I said that you could always come to me with your problems, I meant during office hours!"

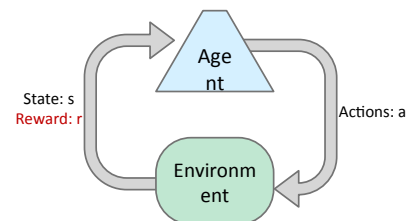
## Reinforcement Learning



## Two Key Ideas

- Credit assignment problem
- Exploration-exploitation tradeoff

## Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of **rewards**
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to **maximize expected rewards**
  - All learning is based on observed samples of outcomes!

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

7

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

" " " 39,      " = 0,   " = 4

8

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

" " " 39,      " = 0,   " = 4

" " " 22,      " = 0,   " = 1

9

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

" " " 39,      " = 0,   " = 4

" " " 22,      " = 0,   " = 1

" " " 21,      " = 0,   " = 1

10

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

" " " 39,      " = 0,   " = 4

" " " 22,      " = 0,   " = 1

" " " 21,      " = 0,   " = 1

" " " 21,      " = 0,   " = 1

11

## The "Credit Assignment" Problem

---

I'm in state 43,      reward = 0,   action = 2

" " " 39,      " = 0,   " = 4

" " " 22,      " = 0,   " = 1

" " " 21,      " = 0,   " = 1

" " " 21,      " = 0,   " = 1

" " " 13,      " = 0,   " = 2

12

## The "Credit Assignment" Problem

I'm in state 43, reward = 0, action = 2  
 " " " 39, " = 0, " = 4  
 " " " 22, " = 0, " = 1  
 " " " 21, " = 0, " = 1  
 " " " 21, " = 0, " = 1  
 " " " 13, " = 0, " = 2  
 " " " 54, " = 0, " = 2

13

## The "Credit Assignment" Problem

I'm in state 43, reward = 0, action = 2  
 " " " 39, " = 0, " = 4  
 " " " 22, " = 0, " = 1  
 " " " 21, " = 0, " = 1  
 " " " 21, " = 0, " = 1

Yippee! I got to a state with a big reward!  
 But which of my actions along the way actually helped me get there??  
 This is the Credit Assignment problem, = 100,



14

## Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for???
- Exploitation:** should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at risk of missing out on a better reward somewhere
- Exploration:** should I look for states w/ more reward?
  - at risk of wasting time & getting some negative reward

15

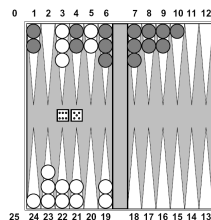
## Example: Animal Learning

- 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! (It's also P3)



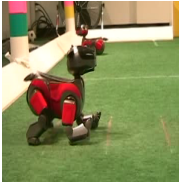
## Demos

- <http://inst.eecs.berkeley.edu/~ee128/fa11/videos.html>

18

## Example: Learning to Walk

---



Initial



A Learning Trial



After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

## Example: Learning to Walk

---



Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initi

## Example: Learning to Walk

---



Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finis

## Example: Sidewinding

---

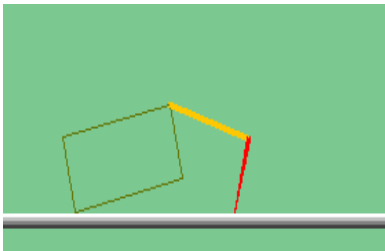


[Andrew Ng]

[Video: SNAKE – climbStep+sidewi

## The Crawler!

---



[Demo: Crawler Bot (L10D1)] [You, in Pr

## Video of Demo Crawler Bot

---



## Other Applications



- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

## Reinforcement Learning

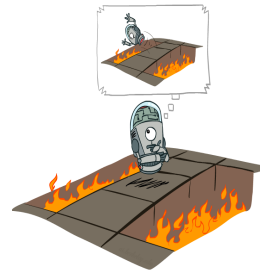
- Still assume a Markov decision process (MDP):
  - A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s')$
  - A reward function  $R(s,a,s')$  & discount  $\gamma$
- Still looking for a policy  $\pi(s)$
- New twist: **don't know T or R**
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn



## Overview

- Offline Planning (MDPs)
  - Value iteration, policy iteration
- Online: Reinforcement Learning
  - Model-Based
  - Model-Free
    - Passive
    - Active

## Offline (MDPs) vs. Online (RL)

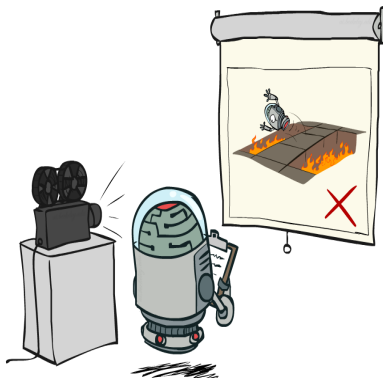


Offline Solution



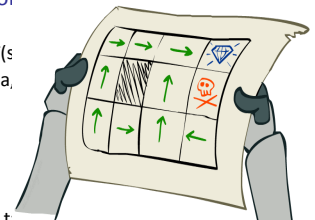
Online Learning

## Passive Reinforcement Learning

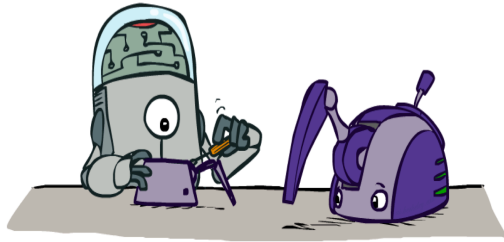


## Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy  $\pi(s)$
  - You don't know the transitions  $T(s,a,s')$
  - You don't know the rewards  $R(s,a,s')$
  - Goal: learn the state values
- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.



## Model-Based Learning



## Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experience
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes  $s'$  for each  $s, a$
  - Normalize to give an estimate of  $T(s, a, s')$
  - Discover each  $\hat{R}(s, a, s')$  experience  $(s, a, s')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

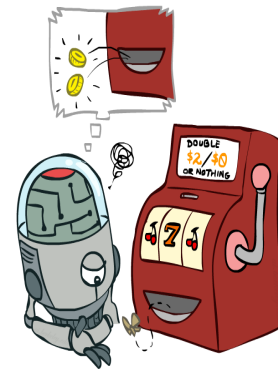


## Example: Model-Based Learning



Input Policy $\pi$	Observed Episodes (Training)	Learned Model $\hat{T}(s, a, s')$													
	<table border="1"> <tr> <th>Episode 1</th> <th>Episode 2</th> </tr> <tr> <td>B, east, C, -1</td> <td>B, east, C, -1</td> </tr> <tr> <td>C, east, D, -1</td> <td>C, east, D, -1</td> </tr> <tr> <td>D, exit, x, +10</td> <td>D, exit, x, +10</td> </tr> </table>	Episode 1	Episode 2	B, east, C, -1	B, east, C, -1	C, east, D, -1	C, east, D, -1	D, exit, x, +10	D, exit, x, +10	<table border="1"> <tr> <td><math>T(B, \text{east}, C) = 1.00</math></td> </tr> <tr> <td><math>T(C, \text{east}, D) = 0.75</math></td> </tr> <tr> <td><math>T(C, \text{east}, A) = 0.25</math></td> </tr> </table>	$T(B, \text{east}, C) = 1.00$	$T(C, \text{east}, D) = 0.75$	$T(C, \text{east}, A) = 0.25$		
Episode 1	Episode 2														
B, east, C, -1	B, east, C, -1														
C, east, D, -1	C, east, D, -1														
D, exit, x, +10	D, exit, x, +10														
$T(B, \text{east}, C) = 1.00$															
$T(C, \text{east}, D) = 0.75$															
$T(C, \text{east}, A) = 0.25$															
Assume: $\gamma = 1$	<table border="1"> <tr> <th>Episode 3</th> <th>Episode 4</th> </tr> <tr> <td>E, north, C, -1</td> <td>E, north, C, -1</td> </tr> <tr> <td>C, east, D, -1</td> <td>C, east, A, -1</td> </tr> <tr> <td>D, exit, x, +10</td> <td>A, exit, x, -10</td> </tr> </table>	Episode 3	Episode 4	E, north, C, -1	E, north, C, -1	C, east, D, -1	C, east, A, -1	D, exit, x, +10	A, exit, x, -10	<table border="1"> <tr> <td><math>\hat{R}(s, a, s')</math></td> </tr> <tr> <td><math>R(B, \text{east}, C) = -1</math></td> </tr> <tr> <td><math>R(C, \text{east}, D) = -1</math></td> </tr> <tr> <td><math>R(D, \text{exit}, x) = +10</math></td> </tr> <tr> <td>...</td> </tr> </table>	$\hat{R}(s, a, s')$	$R(B, \text{east}, C) = -1$	$R(C, \text{east}, D) = -1$	$R(D, \text{exit}, x) = +10$	...
Episode 3	Episode 4														
E, north, C, -1	E, north, C, -1														
C, east, D, -1	C, east, A, -1														
D, exit, x, +10	A, exit, x, -10														
$\hat{R}(s, a, s')$															
$R(B, \text{east}, C) = -1$															
$R(C, \text{east}, D) = -1$															
$R(D, \text{exit}, x) = +10$															
...															

## Model-Free Learning



## Simple Example: Expected Age

Goal: Compute expected age of CSE 473 students

$$E[A] = \sum_a P(a) \cdot a = 0.35 \times 20 + \dots$$

Without  $P(A)$ , instead collect samples  $[a_1, a_2, \dots, a_N]$

Unknown  $P(A)$ : "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown  $P(A)$ : "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

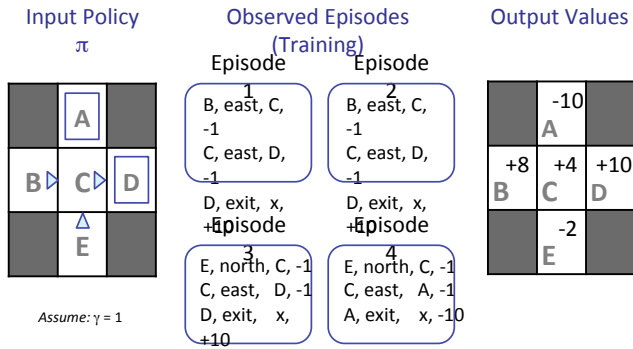
Why does this work? Because samples appear with the right frequencies.

## Direct Evaluation

- Goal: Compute values for each state under  $\pi$
- Idea: Average together observed sample values
  - Act according to  $\pi$
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation



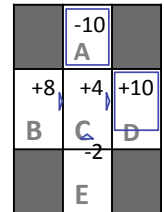
## Example: Direct Evaluation



## Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Ignores Bellman equations
  - Each state must be learned separately
  - So, it takes a long time to learn

### Output Values

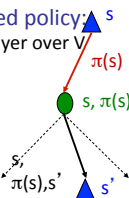


If B and E both go to C under this policy, how can their values be different?

## Why Not Use Policy Evaluation?

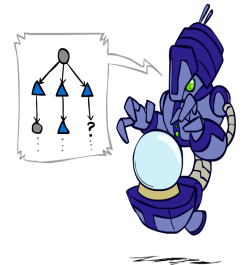
- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V
$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$
  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!
- Key question: how can we do this update to V without knowing T and R?
  - In other words, how to take a weighted average without knowing the weights?



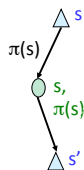
## Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:
 
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$
- Idea: Take samples of outcomes  $s'$  (by doing the action!) and average
  - $sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^\pi(s'_1)$
  - $sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^\pi(s'_2)$
  - ...
  - $sample_n = R(s, \pi(s), s'_n) + \gamma V_k^\pi(s'_n)$
$$V_{k+1}^\pi(s) \leftarrow \frac{1}{n} \sum_i sample_i$$



## Temporal Difference Learning

- Big idea: learn from every experience!
    - Update V(s) each time we experience a transition (s, a, s', r)
    - Likely outcomes  $s'$  will contribute updates more often
  - Temporal difference learning of values
    - Policy still fixed, still doing evaluation!
    - 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:  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$



## Exponential Moving Average

- Exponential moving average
  - The running interpolation update:  $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - 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)
- Decreasing learning rate (alpha) can give converging averages

## Example: Temporal Difference Learning

States

	A	
B	C	D
	E	

Assume:  $\gamma = 1$ ,  $\alpha = 1/2$

Observed Transitions

	0		
0	0	8	
	0		

B, east, C, -2

	0		
-1	0	8	
	0		

C, east, D, -2

	0		
-1	3	8	
	0		

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

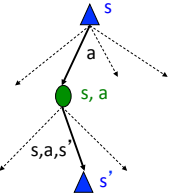
## Problems with TD Value Learning

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

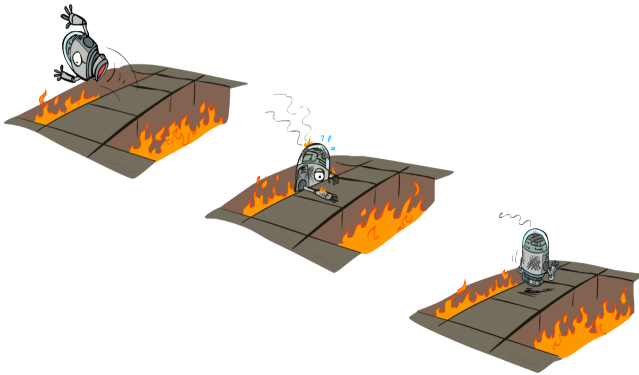
$$\pi(s) = \arg \max_a Q(s, a)$$

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

- Idea: learn Q-values, not values
- Makes action selection model-free too!



## Active Reinforcement Learning



## Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)

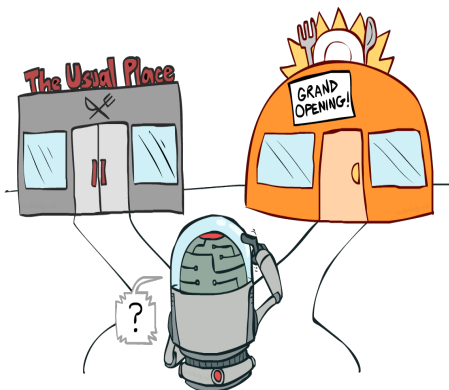
- You don't know the transitions  $T(s, a, s')$
- You don't know the rewards  $R(s, a, s')$
- You choose the actions now
- Goal: learn the optimal policy / values



- 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...

## Exploration vs. Exploitation



## How to Explore?

- Several schemes for forcing exploration

- Simplest: random actions ( $\epsilon$ -greedy)
  - Every time step, flip a coin
  - With (small) probability  $\epsilon$ , act randomly
  - With (large) probability  $1 - \epsilon$ , act on current policy

- Problems with random actions?

- You do eventually explore the space, but keep thrashing around once learning is done
- One solution: lower  $\epsilon$  over time
- Another solution: exploration functions





## Reminder: Q-Value Iteration

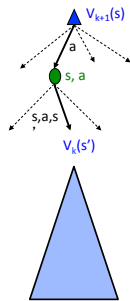
- For all  $s$ , Initialize  $V_0(s) = 0$  no time steps left means an expected reward of zero

- Repeat do Bellman backups

$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_a Q_k(s, a)]$$

$$K += 1$$

- Until convergence



## Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

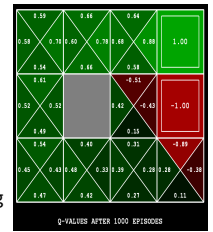
- Learn  $Q(s, a)$  values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate  $Q(s, a)$
- Consider your new sample estimate:

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

- Incorporate the new estimate into a running average:

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



[Demo: Q-learning - gridworld]  
[Demo: Q-learning - crawler]

## Video of Demo Q-Learning -- Gridworld



## Video of Demo Q-Learning -- Crawler



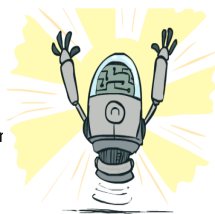
## Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

- This is called **off-policy learning**

- Caveats:

- You have to explore enough
- You have to eventually make the learner small enough
- ... but not decrease it too quickly
- Basically, in the limit, it doesn't matter how you select actions (!)



## Exploration Functions

- When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness (yet) established, eventually stop exploring

- Exploration function

- Takes a value estimate  $u$  and a visit count  $n$  and returns an optimistic utility, e.g.

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

$$\text{Regular Q-Update: } Q(s, a) \leftarrow \alpha R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

- Modified propagation:  $Q(s, a) \leftarrow \alpha R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$
- Unknown states as well!



[Demo: exploration - Q-learning - crawler - exploration]

## Video of Demo Q-learning – Exploration Function – Crawler

---



## Regret

---

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret

