Announcements

Happy Pi Day!

Assignments before Midterm 2

- HW6 (online)
  - Due Tonight, 3/14 10 pm
- P3: Logic/Classical Planning
  - Due Friday, 3/17 10 pm
- HW7 (written) out tonight, due 3/21 10pm
- HW8 (online) due 3/28 10pm
- P4: Reinforcement Learning
  - Due Thursday, 4/6 10 pm (after midterm 2)

Midterm 2 3/30!

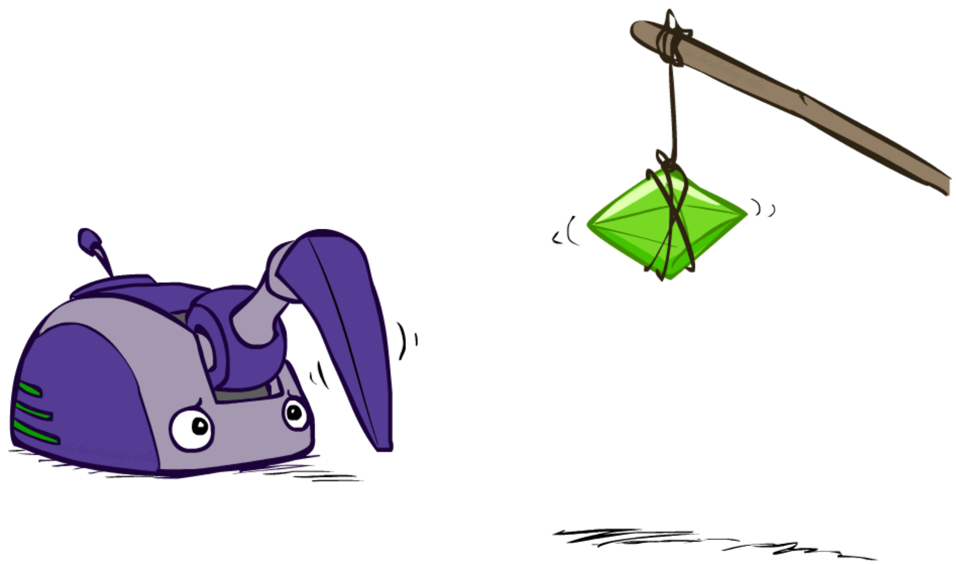
Announcements

Midterm 2

- Covers material from 2/17 (recitation) through 3/28
- Logic/Logical Agents, Classical Planning, MDPs, RL, Bayes Nets
- Calculators allowed - Lots of computation
  - Device must be only a calculator (no phones, ipads, etc)
- One 8.5”x11” handwritten cheatsheet is also allowed

AI: Representation and Problem Solving

Reinforcement Learning



Instructor: Stephanie Rosenthal

Slide credits: CMU AI and http://ai.berkeley.edu

What do you remember about MDPs?

S, A, Transitions $P(s'|s,a)$ Rewards living $R(s,a,s')$

Maximize Expected Reward

Act $\pi(s) \to a$

Value Iteration / Policy Extraction

Policy: Policy Evaluation / Policy Improvement

V^* π^* Bellman

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

Poll 1

Which of the following are used in policy iteration?

A. Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

B. Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

C. Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

D. Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

E. Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

F. No Clue!

Poll 2

s, a, s′

Rewards may depend on any combination of *state*, *action*, *next state*.

Which of the following are valid formulations of the Bellman equations?

Hint: what can you pull out or redistribute based on the parameters of R?

A. $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

B. $V^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$

C. $V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$

D. $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$

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

Reinforcement learning

What if we knew we had an MDP but didn't know $P(s'|s,a)$ and $R(s,a,s')$?

Value iteration: $$V_{k+1}(s) = \max_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma V_k(s')], \qquad \forall\, s$$

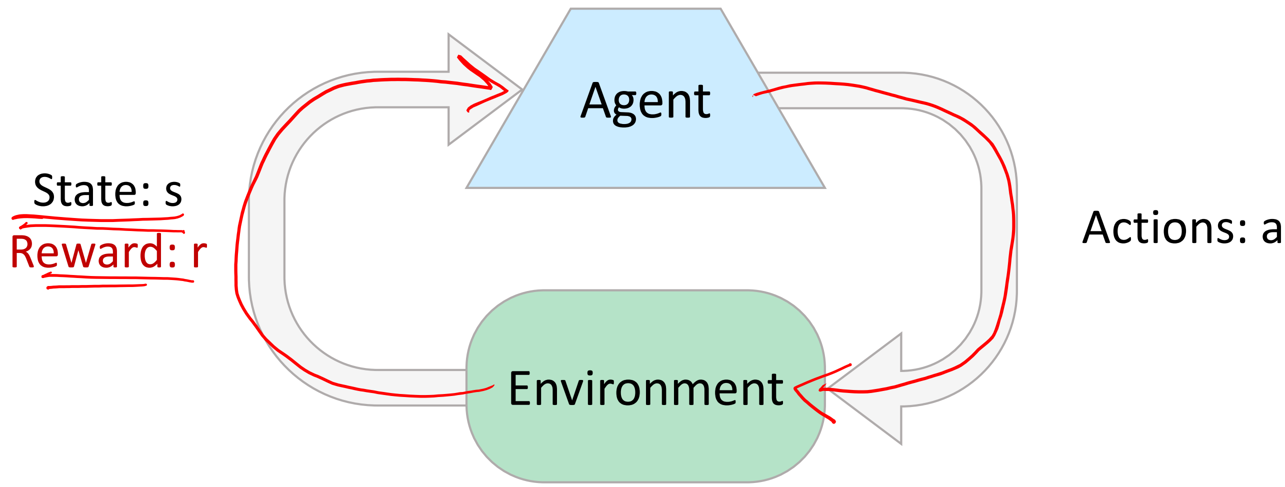
Q-iteration: $$Q_{k+1}(s,a) = \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$$

Policy extraction: $$\pi_V(s) = \operatorname{argmax}_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma V(s')], \qquad \forall\, s$$

Policy evaluation: $$V_{k+1}^{\pi}(s) = \sum_{s'} \cancel{P(s'|s,\pi(s))}[\cancel{R(s,\pi(s),s')} + \gamma V_k^{\pi}(s')], \qquad \forall\, s$$

Policy improvement: $$\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma V^{\pi_{old}}(s')], \qquad \forall\, s$$

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!

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

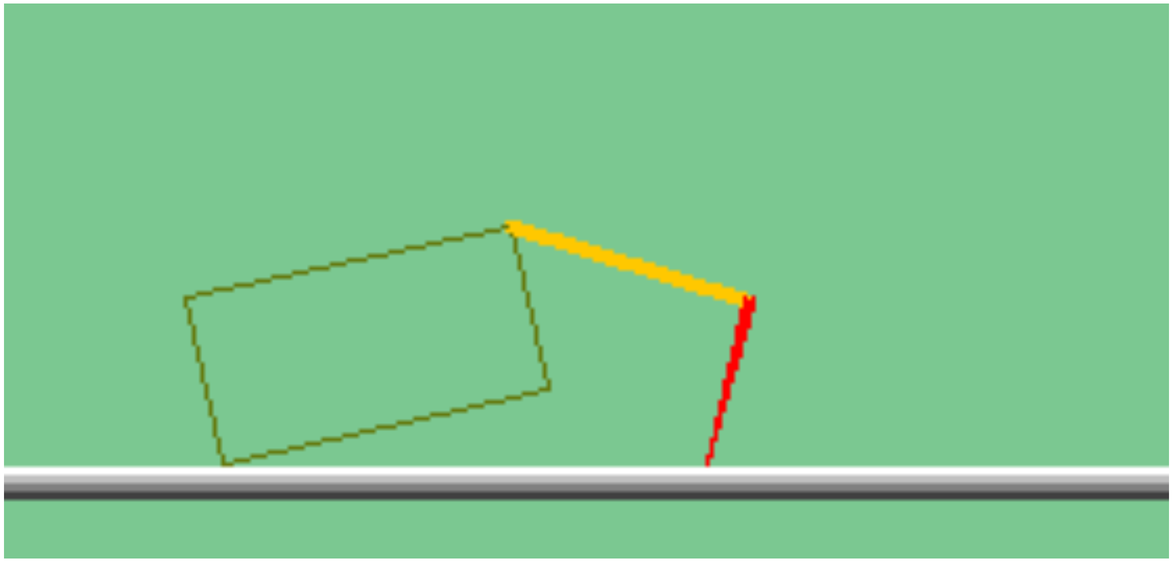
[Video: AIBO WALK – finished]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

The Crawler!



[You, in Project 4]

Reinforcement Learning

Still assume a Markov decision process (MDP):

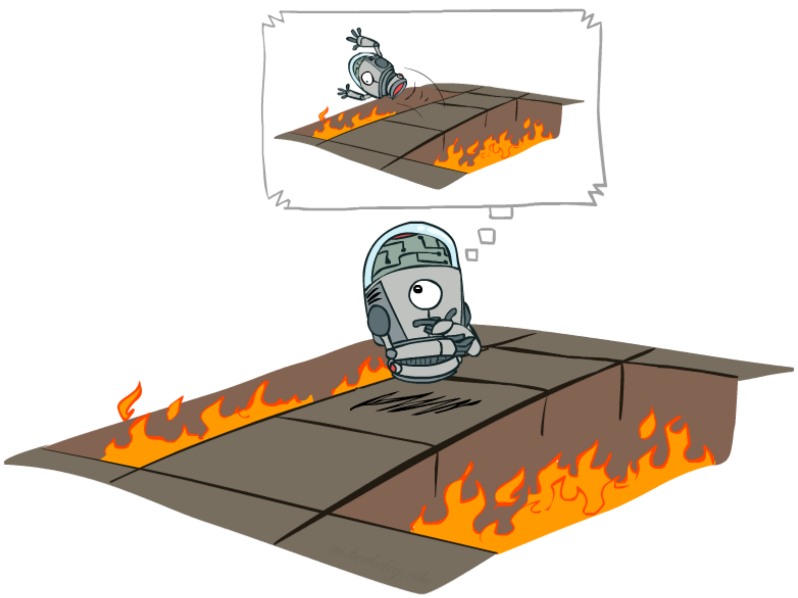
- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s,a,s')
- A reward function R(s,a,s')

Still looking for a policy $\pi(s)$

New twist: don't know T or R

- I.e. we don't know which states are good or what the actions do
- Must actually try actions and states out to learn

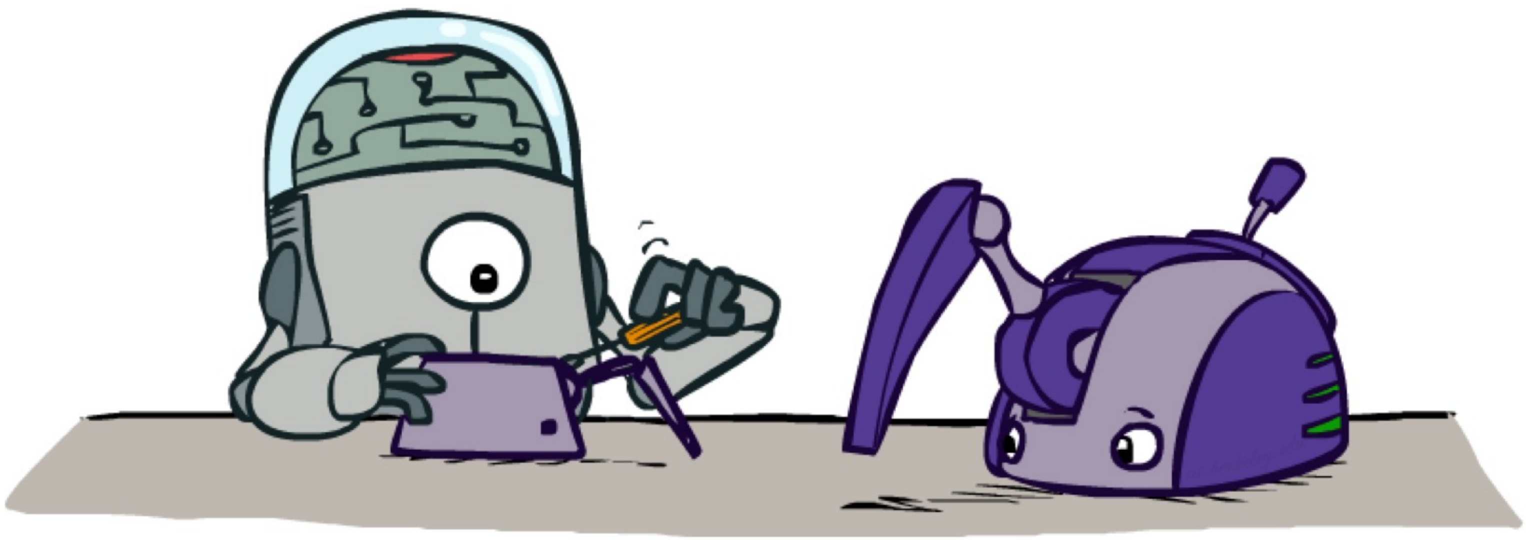
Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Model-Based Learning



Model-Based Learning

Model-Based Idea:

- Learn an approximate model based on experiences
- 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 $\widehat{T}(s, a, s')$
- Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')

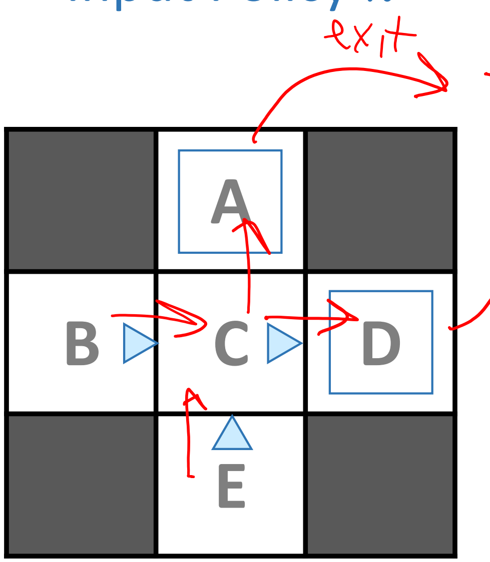
Step 2: Solve the learned MDP

- For example, use value iteration, as before

Learning an Empirical Model

sample <s, a, s', r>

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

T

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$$\widehat{T}(s, a, s')$$

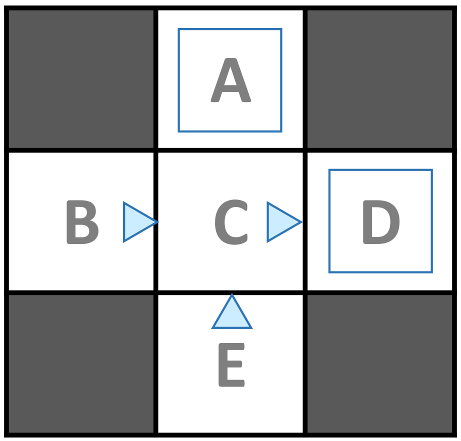
$T(B, e, C) = 2/2 = 1.0$

$$\widehat{R}(s, a, s')$$

$R(B, e, C) = -1$

Example: Model-Based Learning + Poll 3

Input Policy π



Assume: γ = 1

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) =
T(C, east, D) =
T(C, east, A) =
…

T(C, e, D) = ? 3/4

P(D | C, e) = 3/4

$\hat{R}(s, a, s')$

R(B, east, C) =
R(C, east, D) =
R(D, exit, x) =
…

R(C, e, D) = ? -1

Example: Expected Age

Goal: Compute expected age of 15-281 students

Known P(A)

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

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

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.

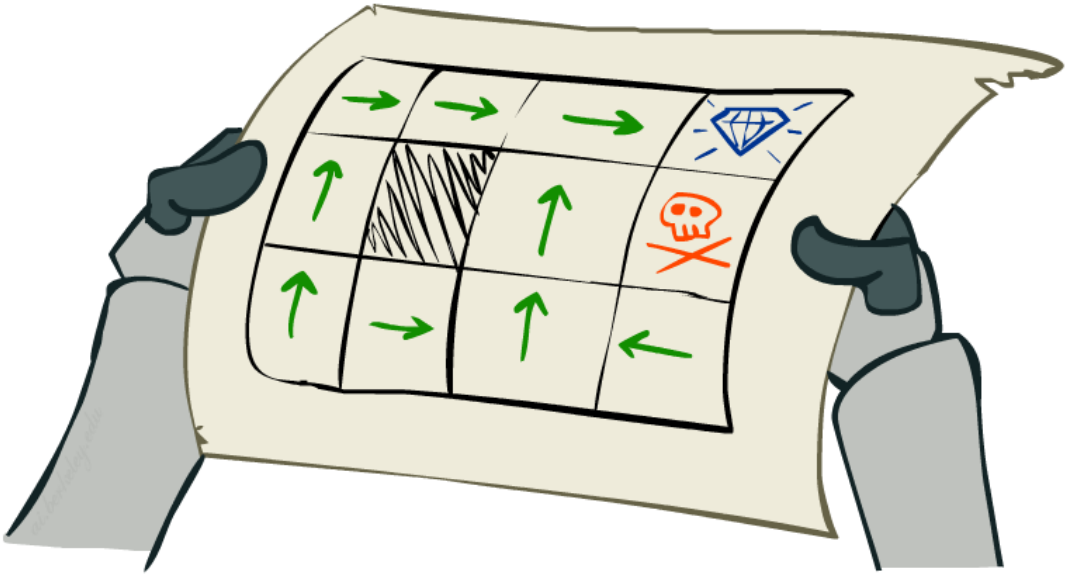
Unknown P(A): "Model Free"

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

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

Model-free Learning

Passive Reinforcement Learning

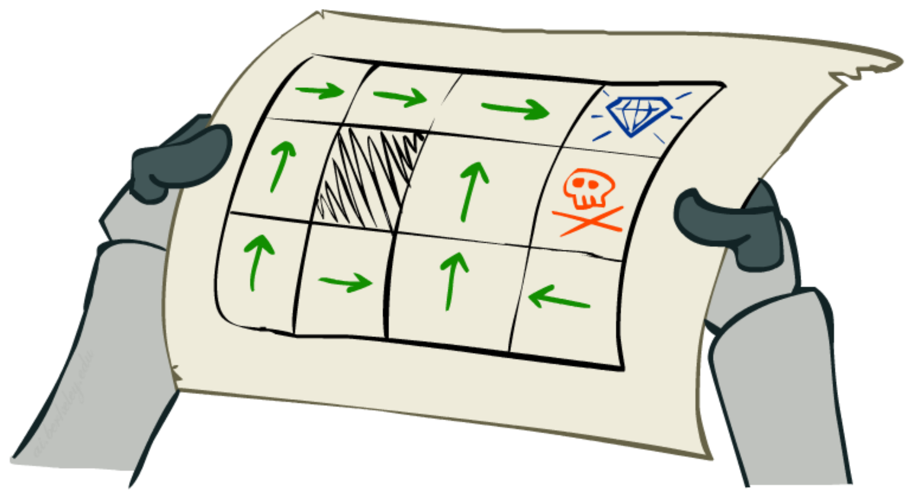


Passive Reinforcement Learning

someone else chooses your actions

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.

Direct Evaluation

Goal: Compute values for each state under π

Idea: Average together observed sample values

- Act according to π
- 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

Direct Evaluation

Goal: Compute values for each state under π

Idea: Average together observed sample values

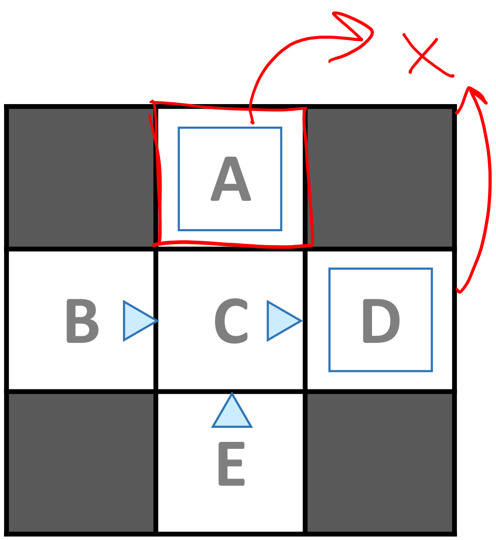
- Act according to π
- Every time you visit a state, write down what the sum of discounted rewards turned out to be
- Average those samples

This is called direct evaluation

Pieces Available	Take 1	Take 2
2	0%	**100%**
3	2%	0%
4	**75%**	2%
5	4%	**68%**
6	5%	6%
7	**60%**	5%

Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values

A: -10

B: $\frac{8+8}{2} = 8$

C: $4 = \frac{9+9+9+-11}{4}$

D: $\frac{10+10+10}{3} = 10$

E: $\frac{8+-12}{2} = -2$

Algorithm: Average all total/future rewards that start at each state

Problems with Direct Evaluation

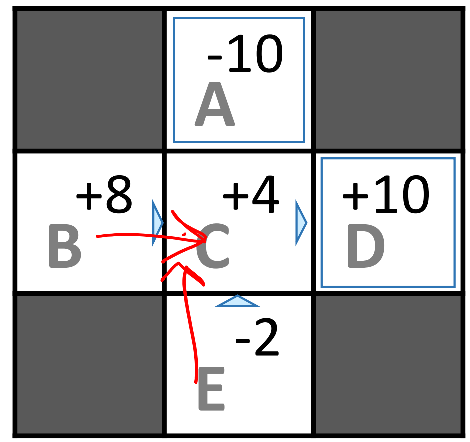
What's good about direct evaluation?

- 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
- 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?

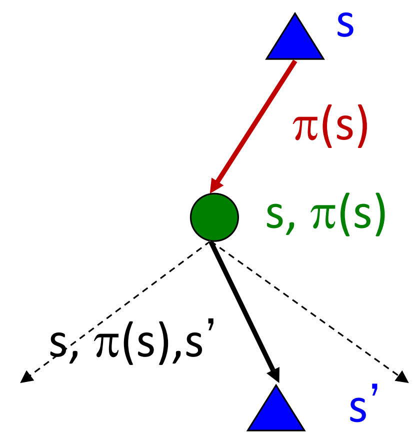
Why Not Use Policy Evaluation?

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 we 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

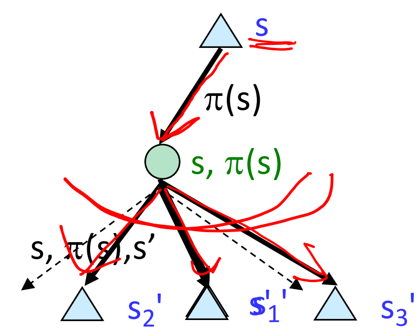
$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$

$$\ldots$$

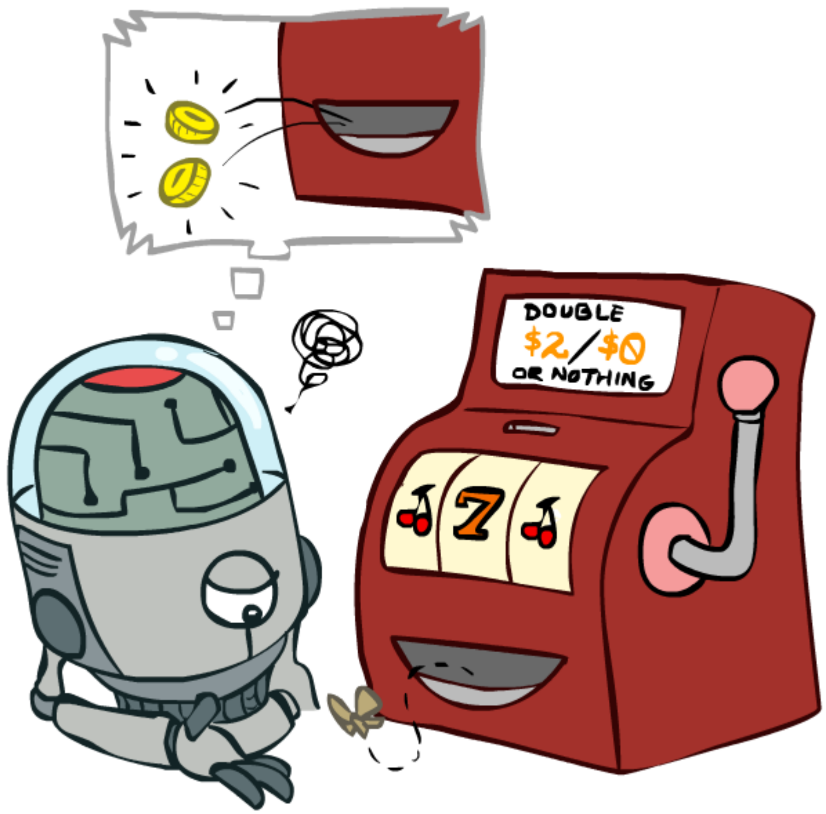
$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

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



Almost! But we can't rewind time to get sample after sample from state s.

Temporal Difference Learning



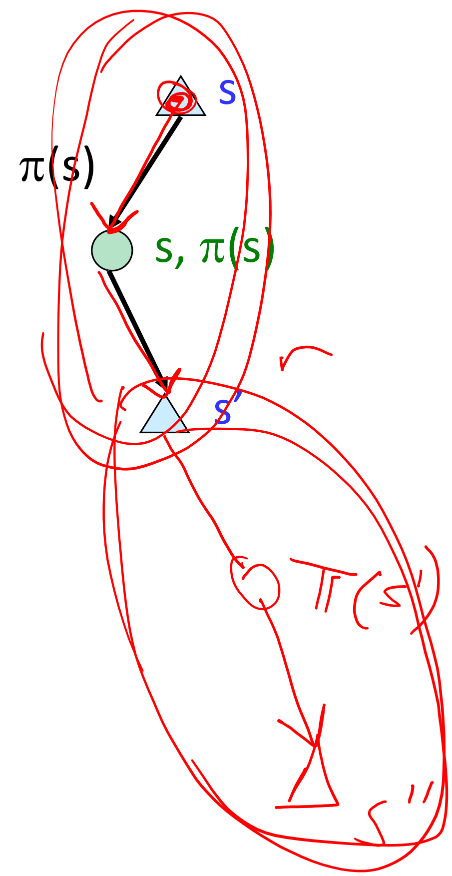
Temporal Difference Learning

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

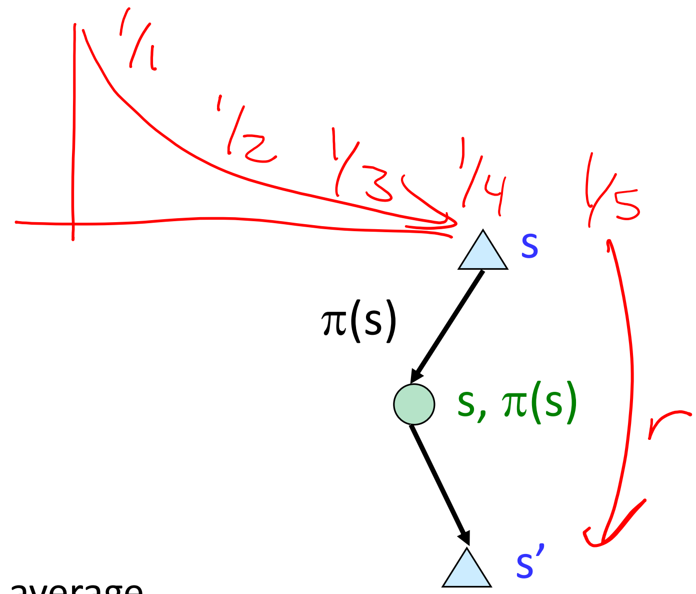
- Policy still fixed, still doing evaluation!
- Move values toward value of whatever successor occurs: running average



Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s):

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

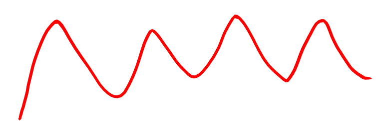
Exponential moving average

- The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot\bar{x}_{n-1} + \alpha\cdot x_n$
- Makes recent samples more important:

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

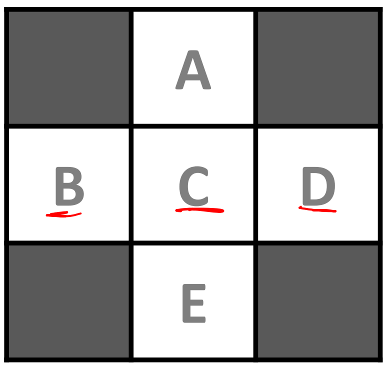
- Forgets about the past (distant past values were wrong anyway)

Decreasing learning rate (alpha) can give converging averages



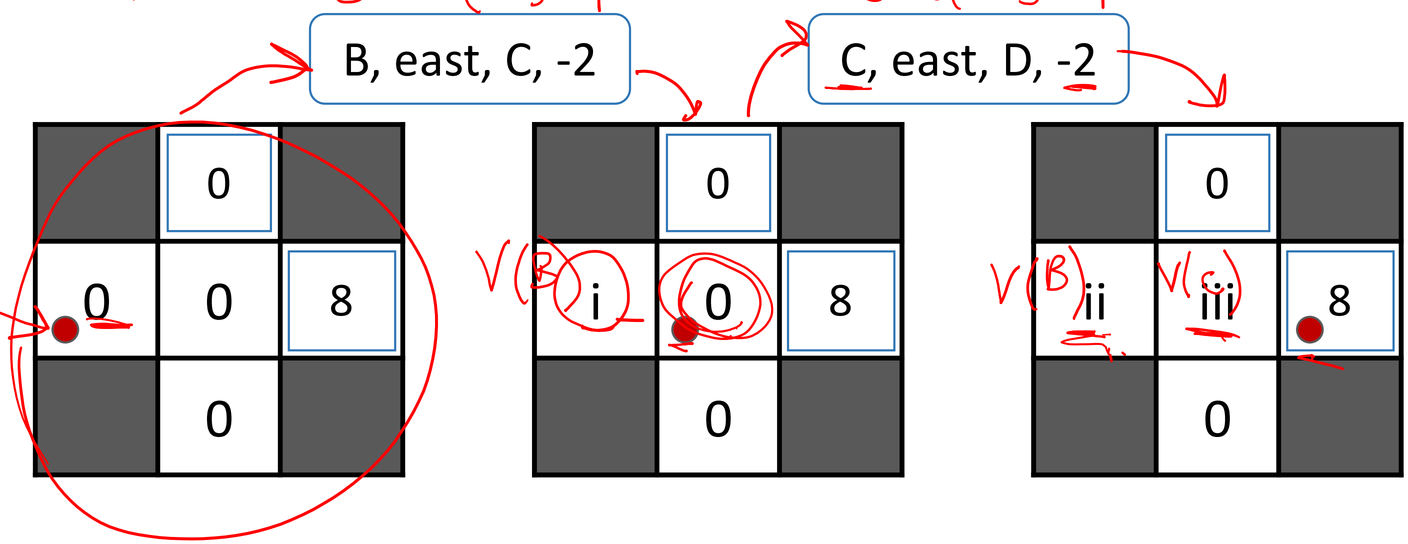
Example: Temporal Difference Learning: Poll 4

States



Assume: $\gamma = 1$, $\alpha = 0.2$

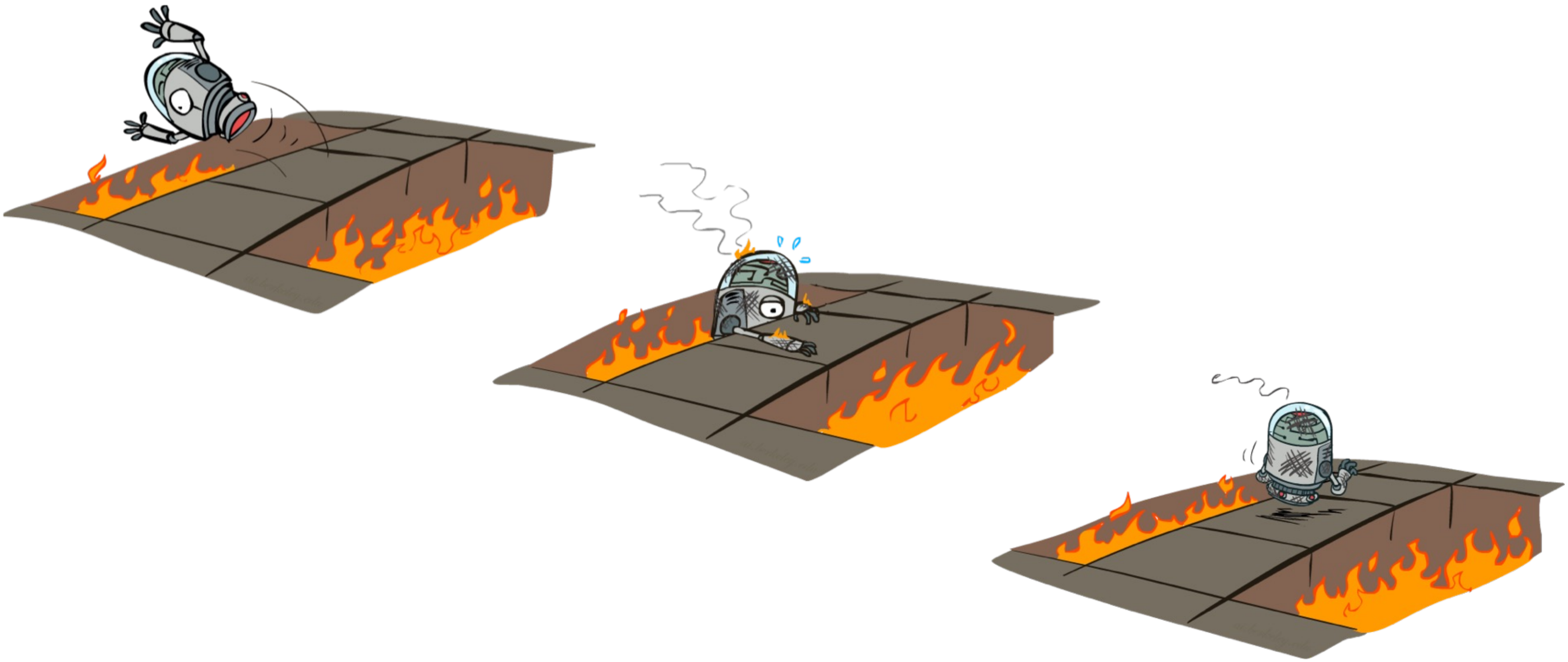
Observed Transitions



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

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

Active Reinforcement Learning



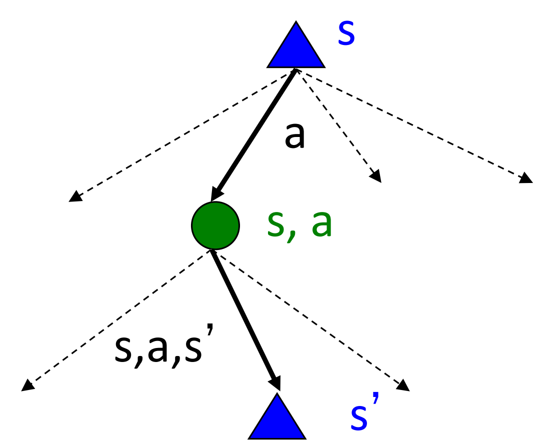
Problems with TD Value Learning

TD value leaning 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:

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

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



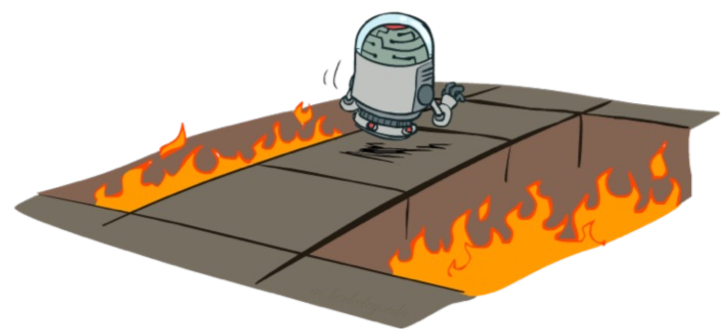
Idea: learn Q-values, not values

Makes action selection model-free too!

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…

Detour: Q-Value Iteration

Value iteration: find successive (depth-limited) values

- Start with $V_0(s) = 0$, which we know is right
- Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

But Q-values are more useful, so compute them instead

- Start with $Q_0(s,a) = 0$, which we know is right
- Given Q_k, calculate the depth k+1 q-values for all q-states:

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

Q-Learning

Q-Learning: sample-based Q-value iteration

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

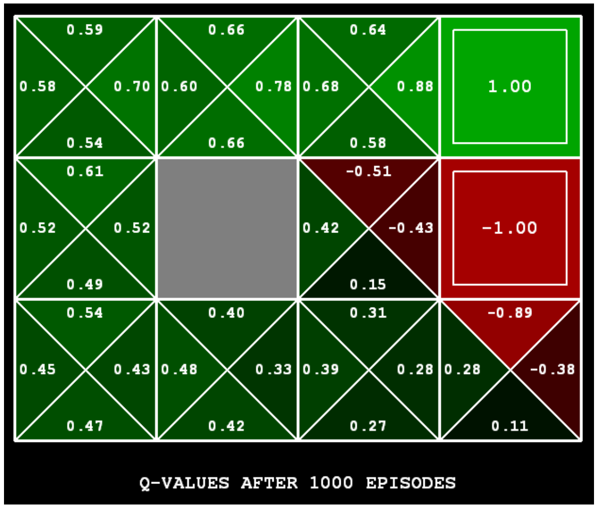
Learn Q(s,a) values as you go

- 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:

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



Q-Learning

We'd like to do Q-value updates to each Q-state:

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

- But can't compute this update without knowing T, R

Instead, compute average as we go

- Receive a sample transition (s,a,r,s')
- This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

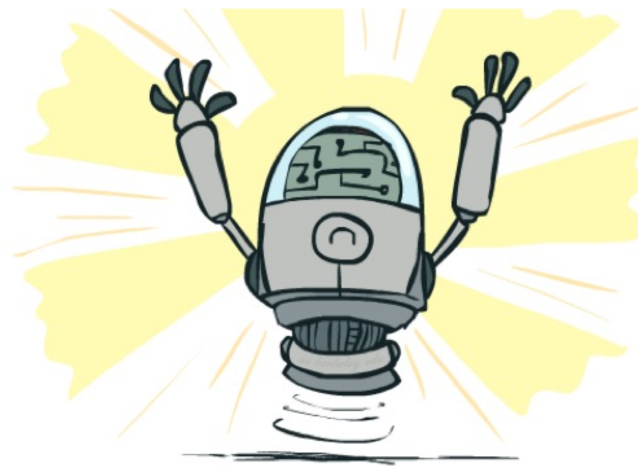
- But we want to average over results from (s,a) (Why?)
- So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Q-Learning Properties

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

This is called off-policy learning

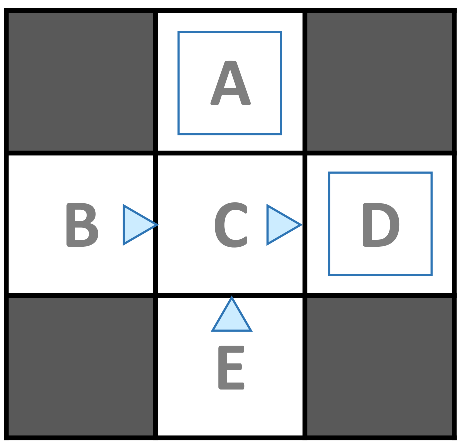


Caveats:

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

Example: Q-Learning + Poll 5

Input Policy π



Assume: γ = 1

α = 0.5

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

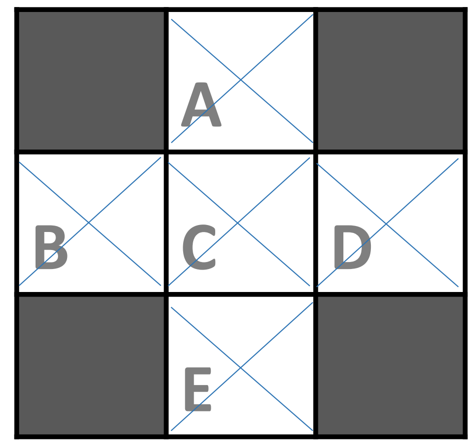
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$