Announcement

- Homework 5
  - Due: Dec. 3, 11:59pm
- Programming Assignment 5
  - Due: Dec. 10, 11:59pm
- Next three lectures to be given by Prof. Xuming He

Project

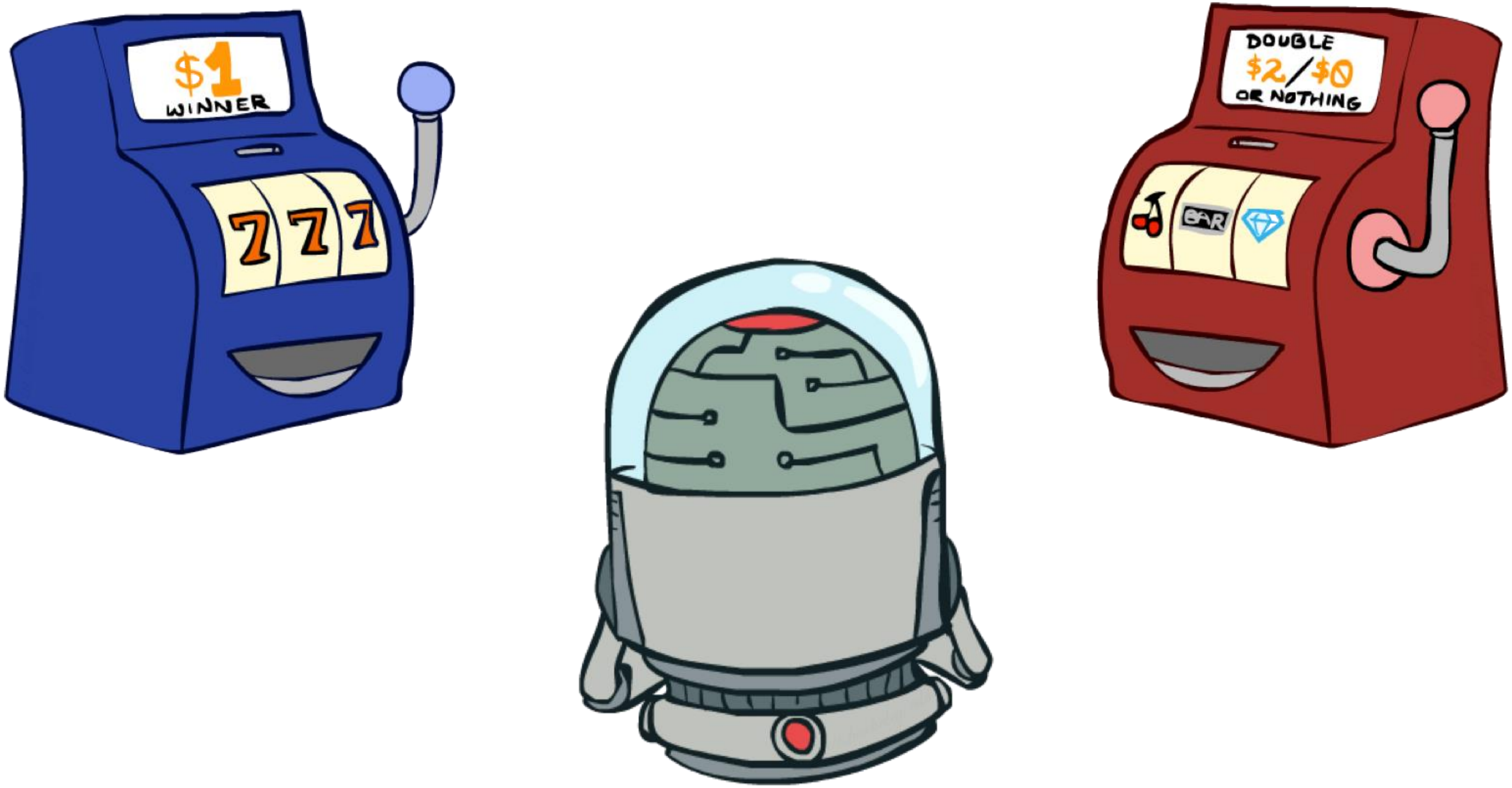
- Project group registration
  - Register your group members by Dec. 8
    - BB → Project → Project group registration
    - Each group only needs to register once
  - No more than 5 people in a group
    - You are encouraged to form large groups
- Proposal presentation
  - Dec. 17, 22
  - 5-7min per group

Reinforcement Learning

AIMA Chapter 21

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley]

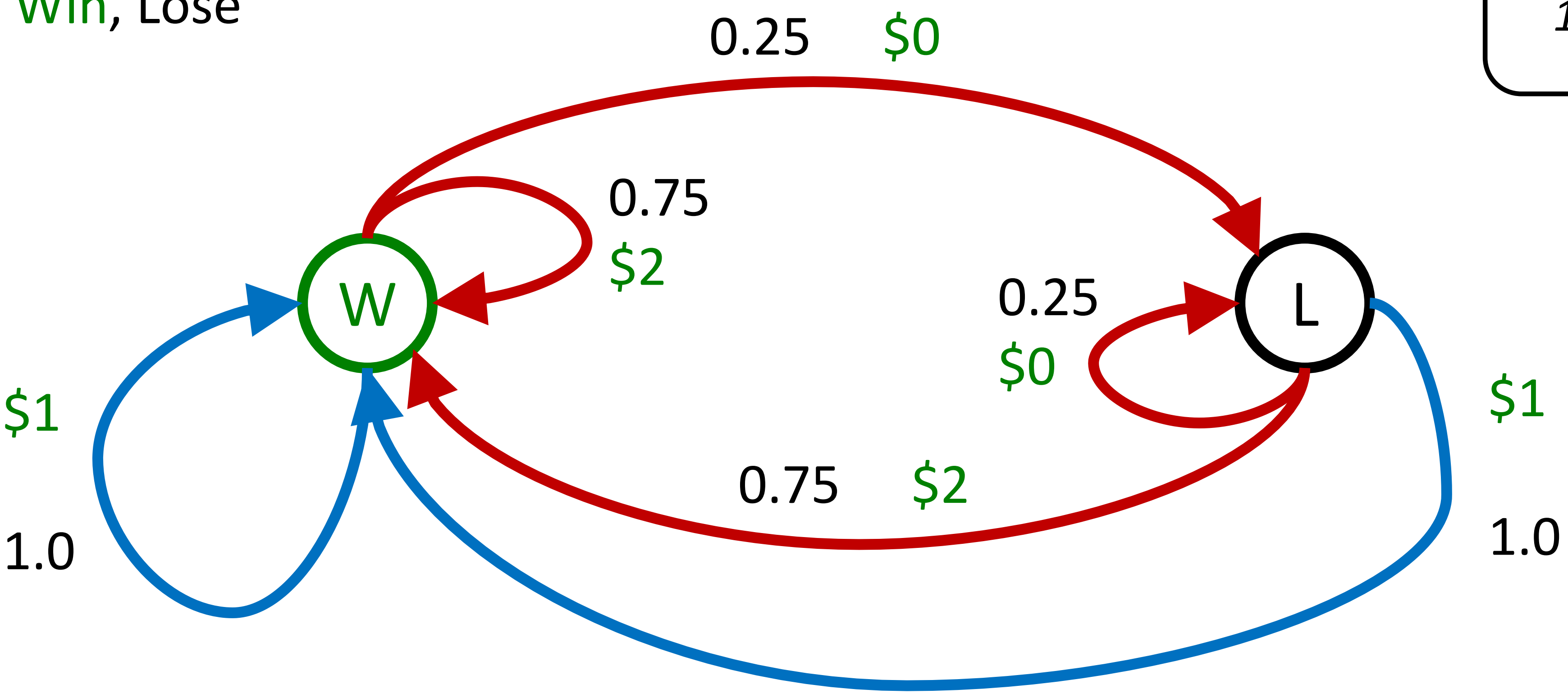
Double Bandits



Double-Bandit MDP

- Actions: *Blue*, *Red*
- States: Win, Lose

No discount
100 time steps

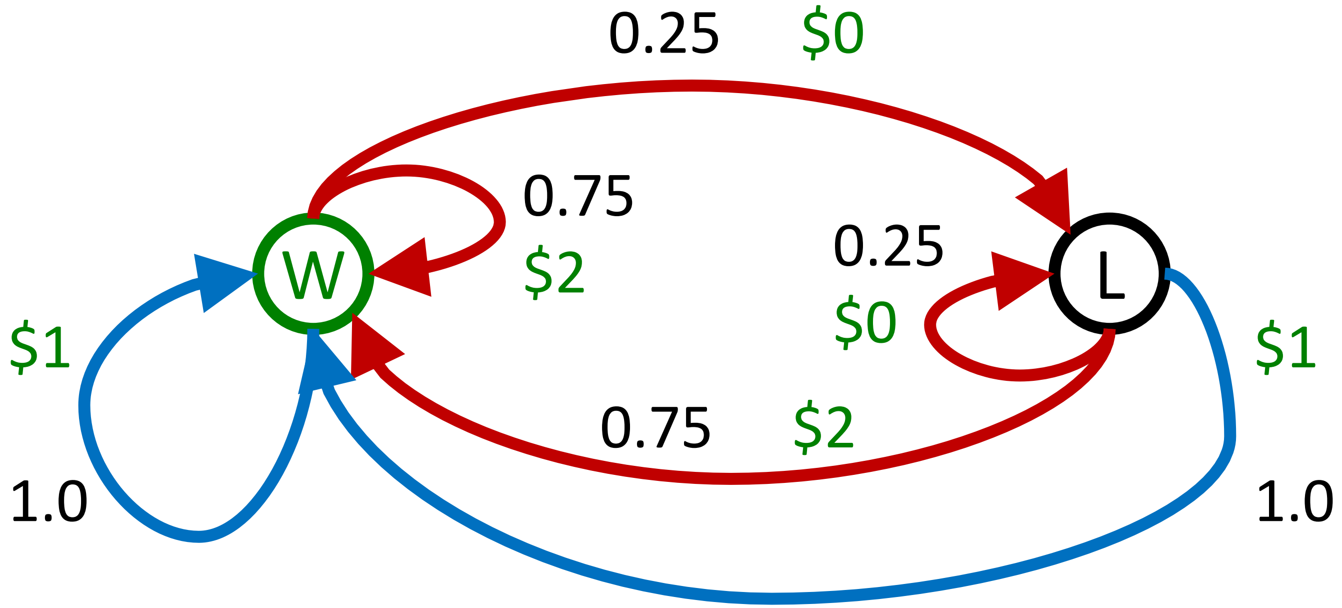


Offline Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

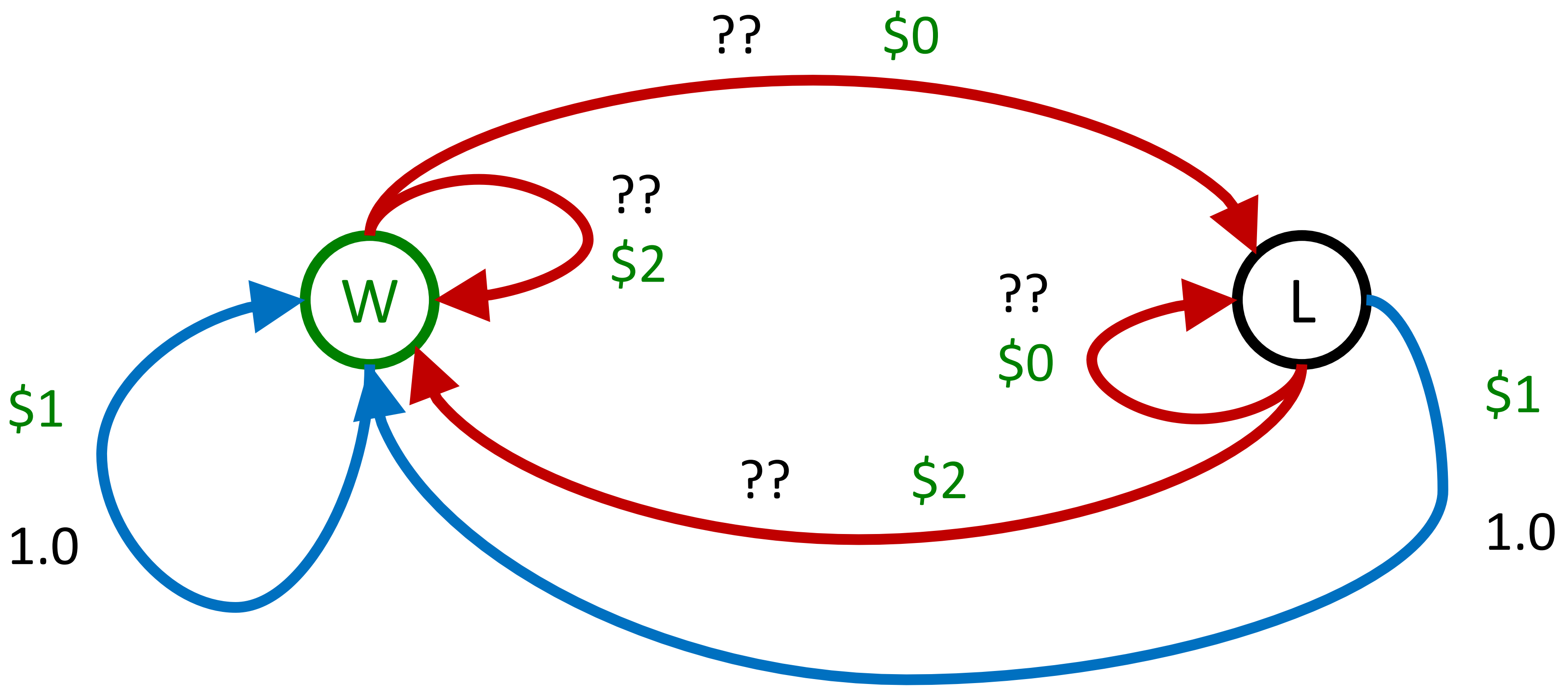
No discount
100 time steps

	Value
Play Red	150
Play Blue	100



Online Planning

- Rules changed! Red's win chance is different.



Let's Play!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

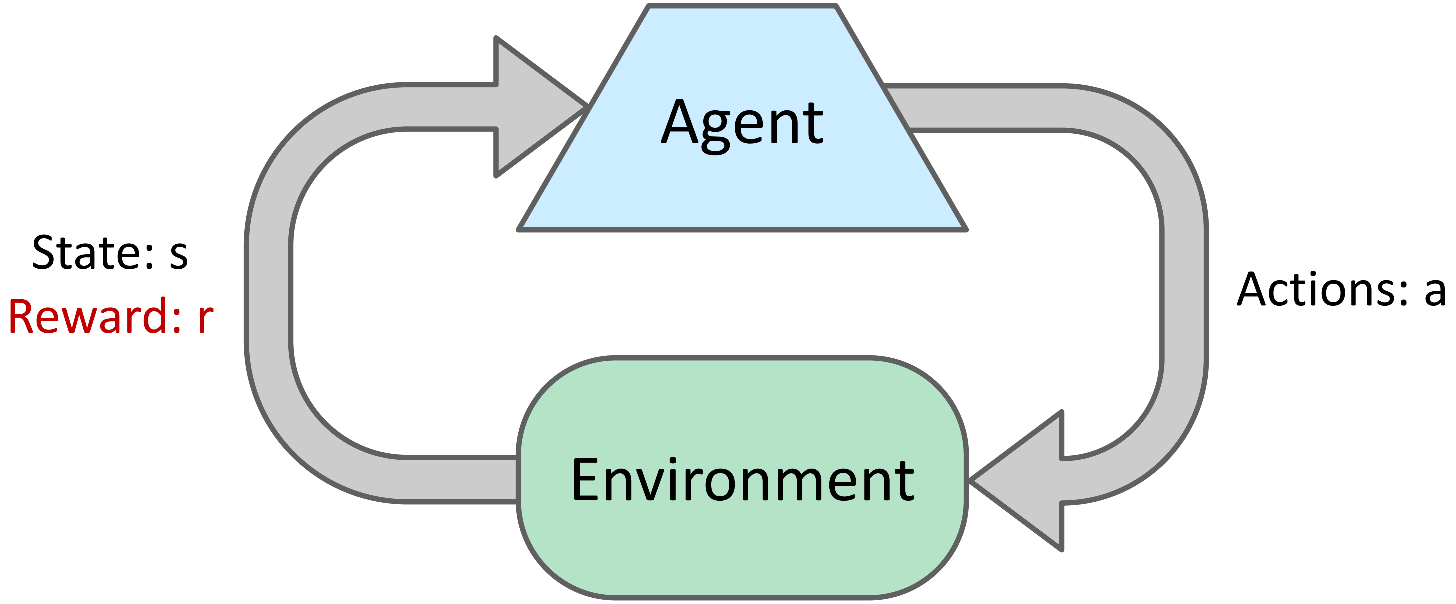
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

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 $\pi(s)$

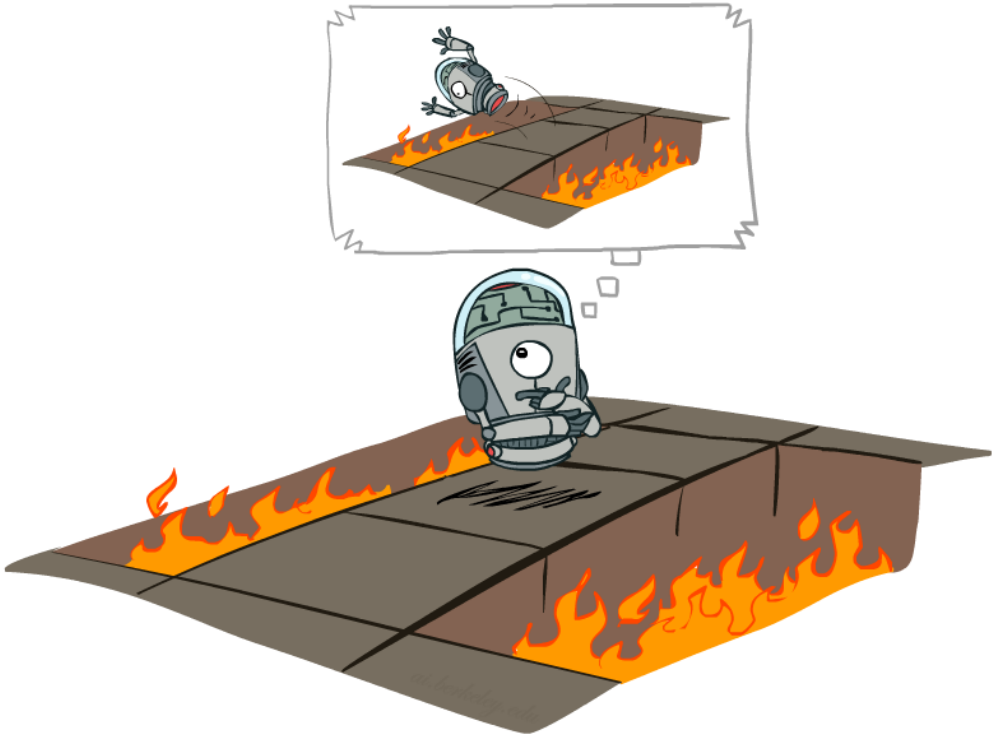
- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning

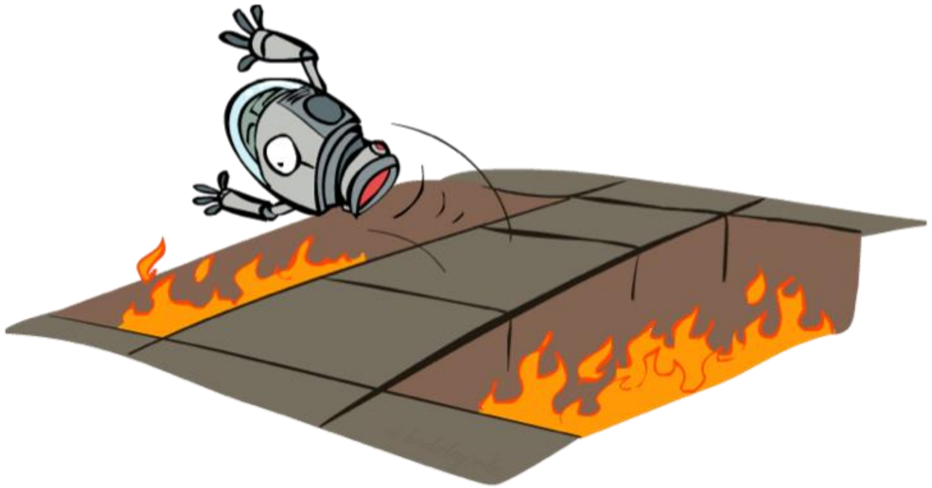


- Basic idea:
  - Take actions and observe outcomes (new states, rewards)
  - Learning is based on observed samples of outcomes
  - Must (learn to) act so as to maximize expected rewards

Offline (MDPs) vs. Online (RL)



Offline Solution

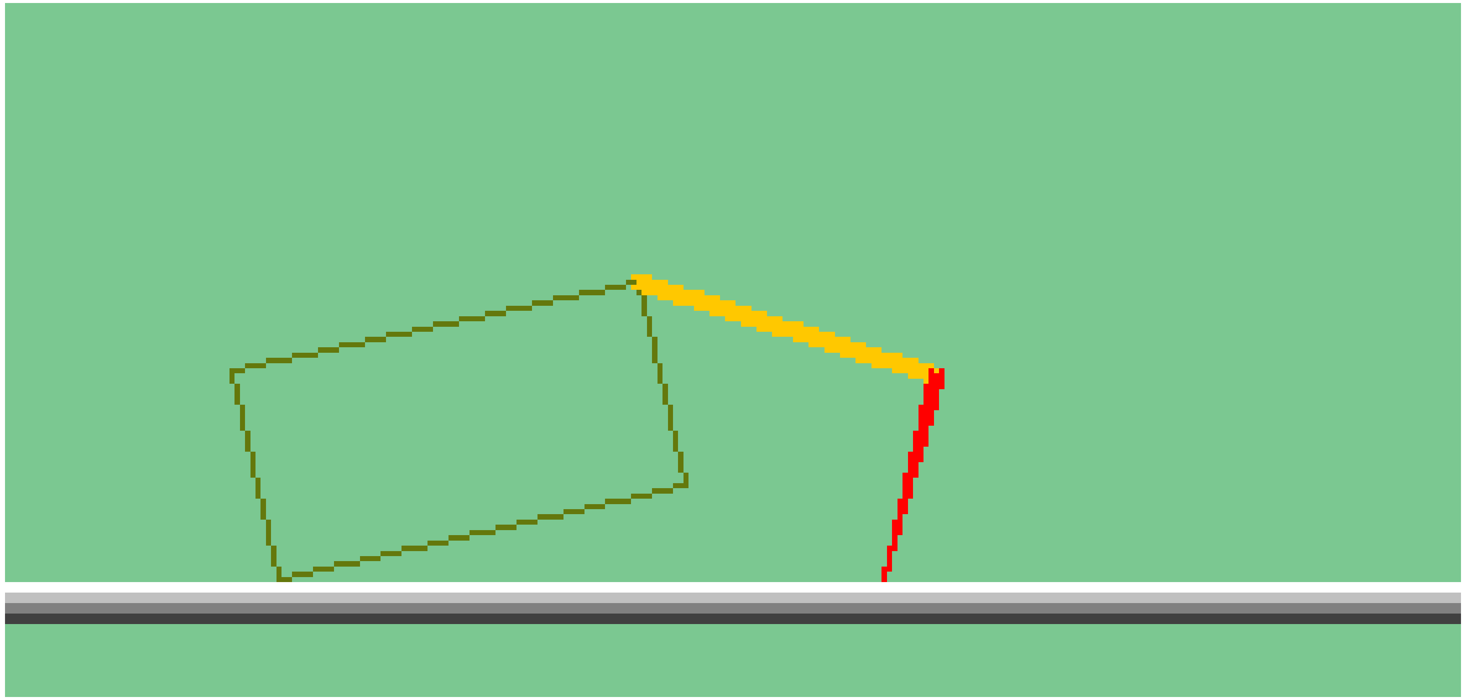


Online Learning

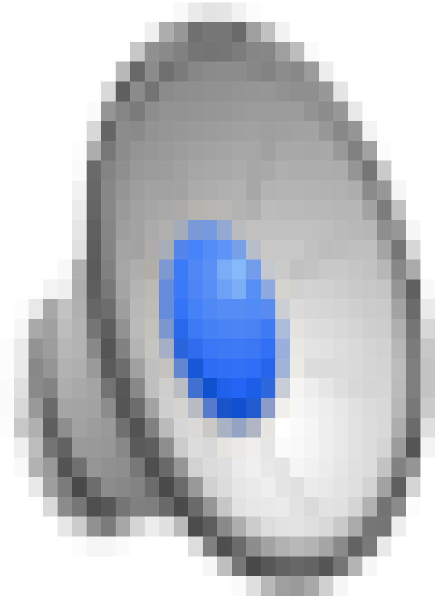
Cheetah

Atari

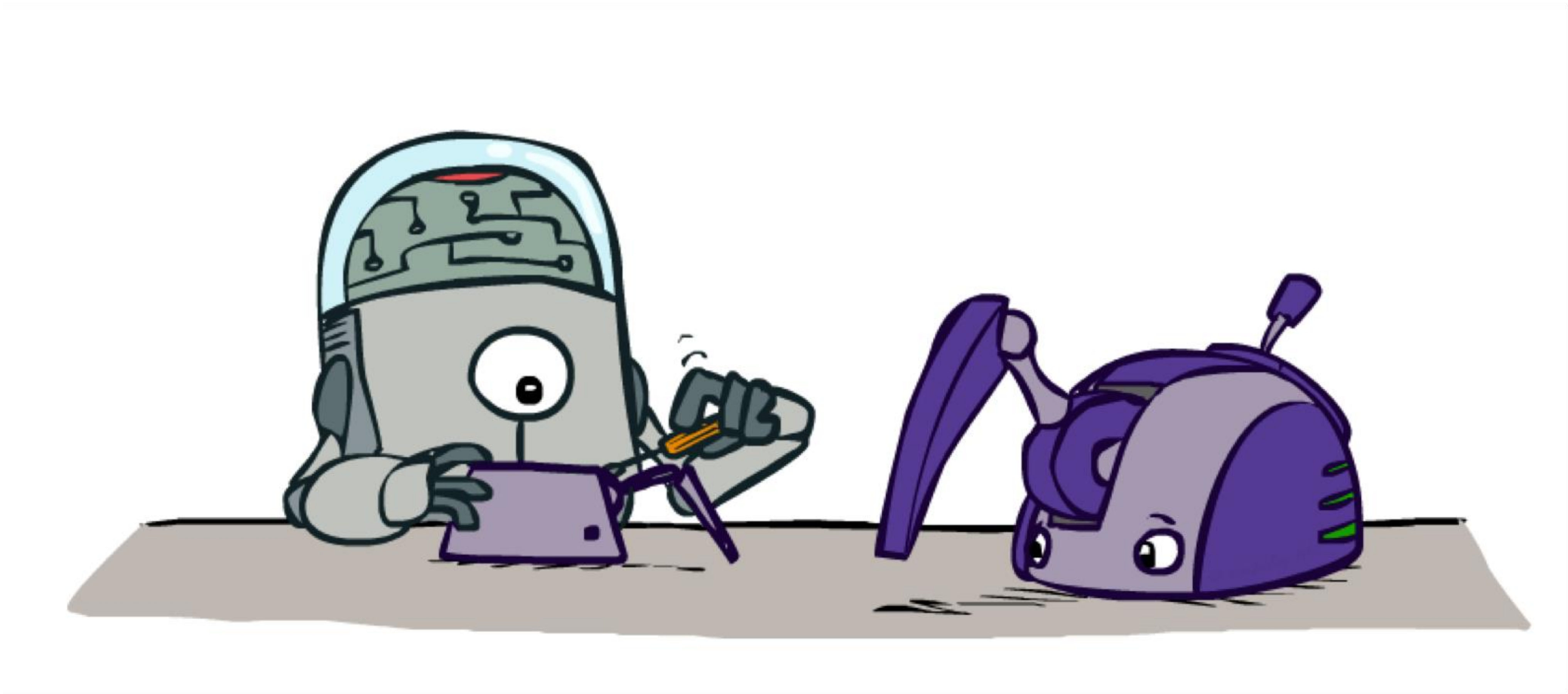
The Crawler!



Video of Demo Crawler Bot

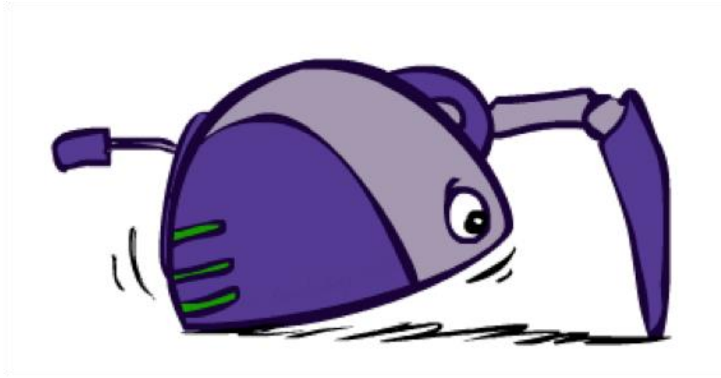


Model-Based Learning



Model-Based Learning

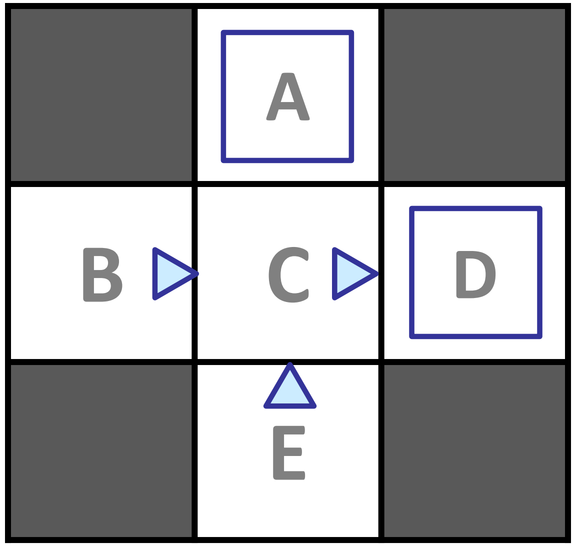
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model was correct



- 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

Example: Model-Based Learning



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

Learned Model

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

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

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

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Model-Based vs. Model-Free

Goal: Compute expected age of ShanghaiTech 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"

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

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

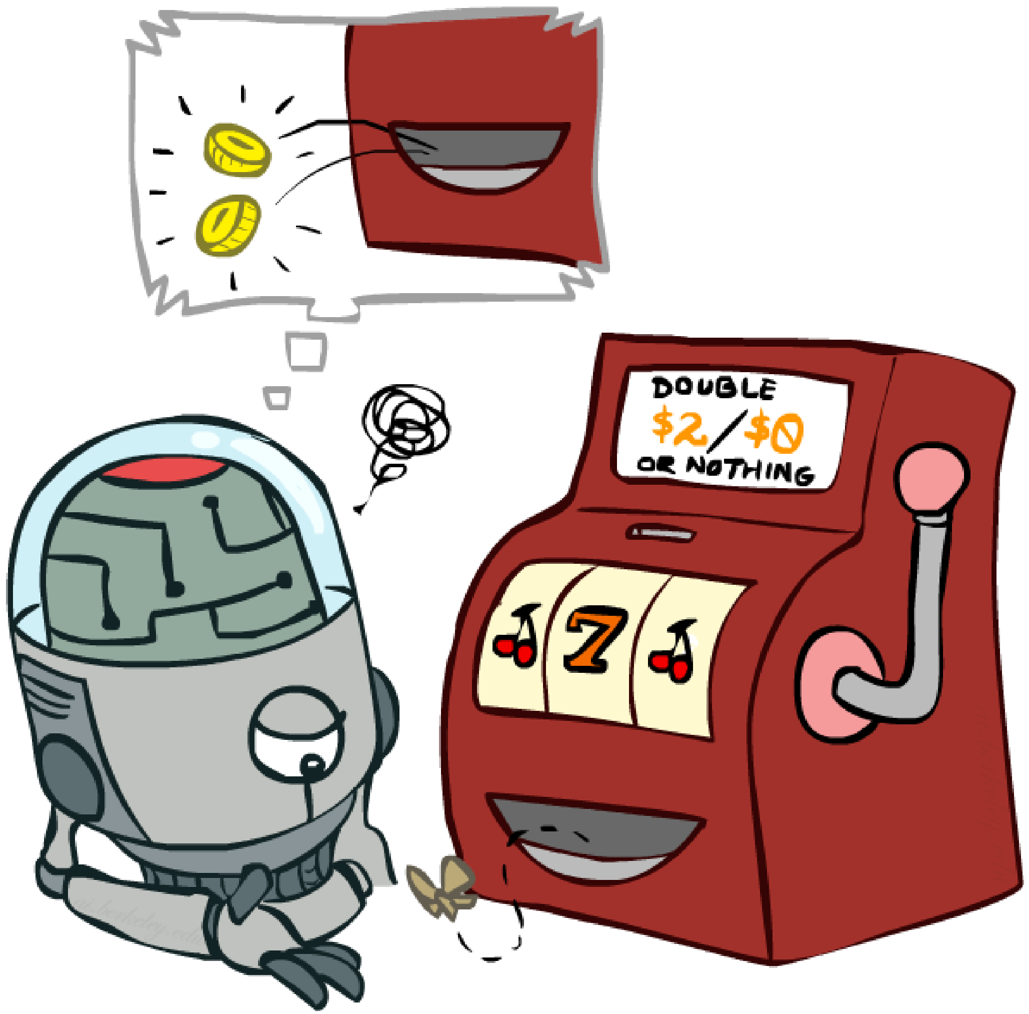
Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

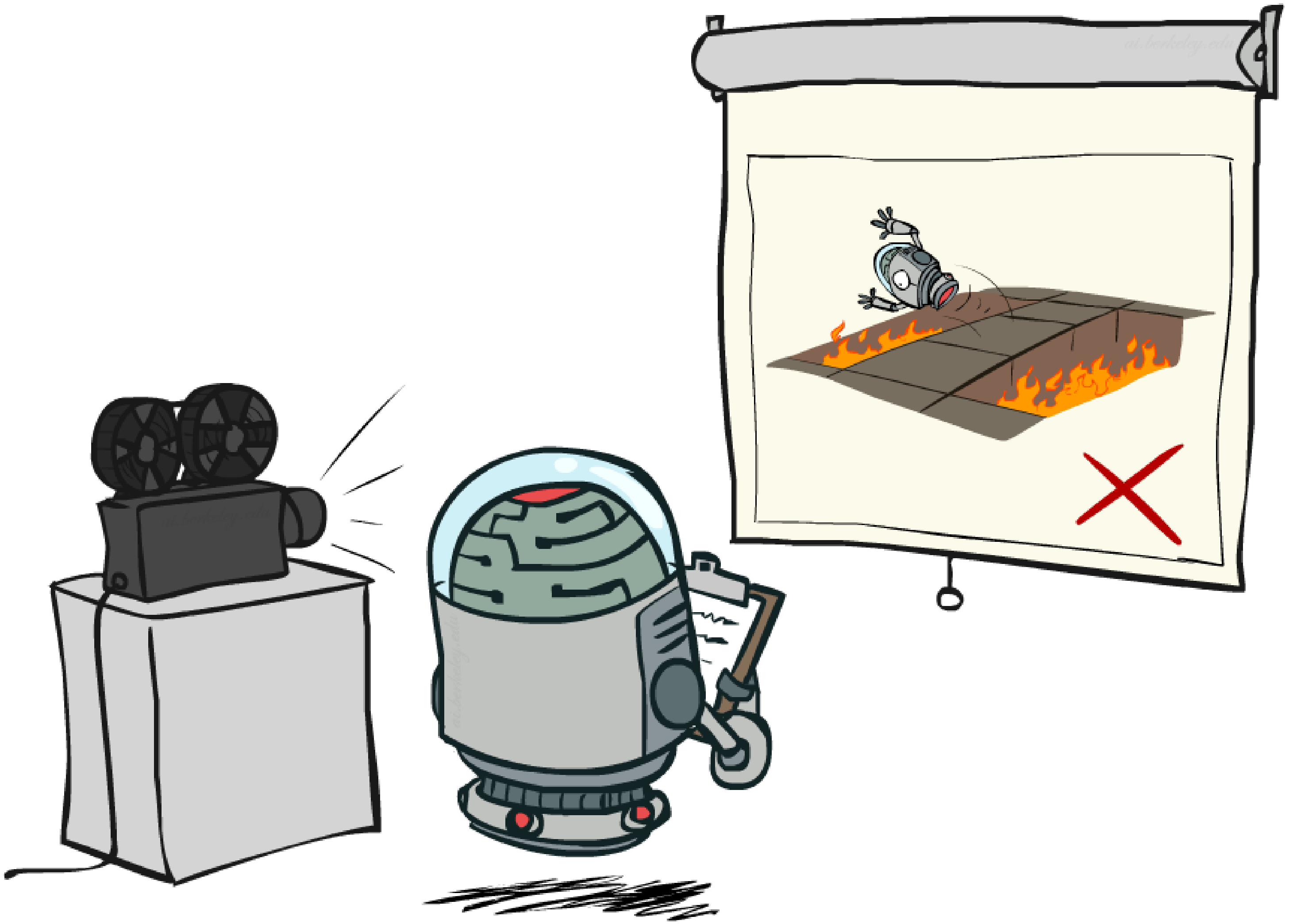
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

Model-Free Learning

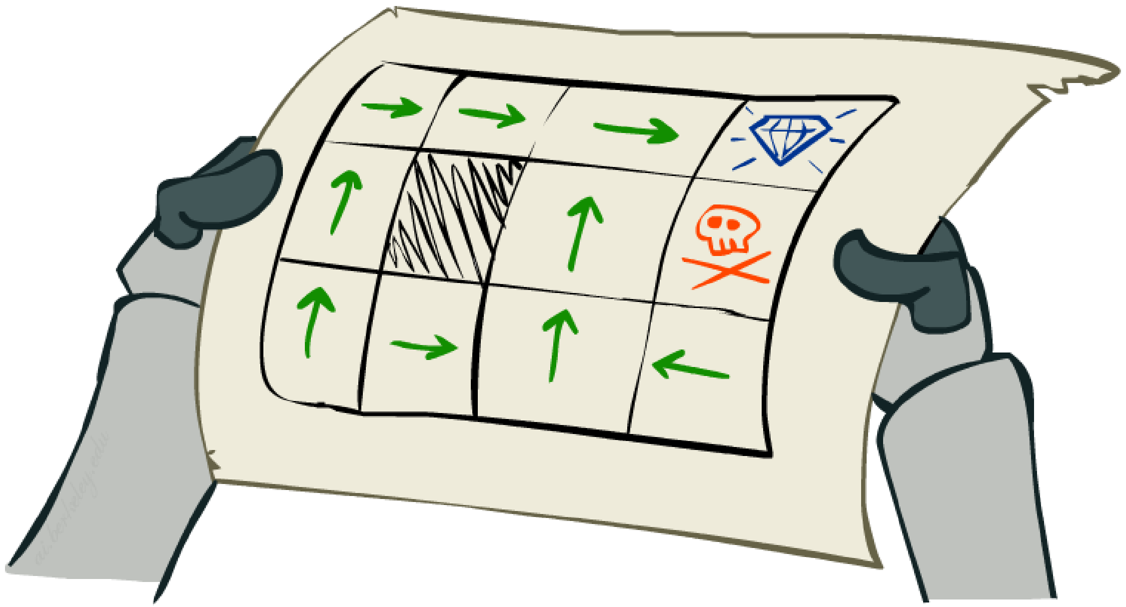


Passive Reinforcement Learning



Passive Reinforcement Learning

- 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



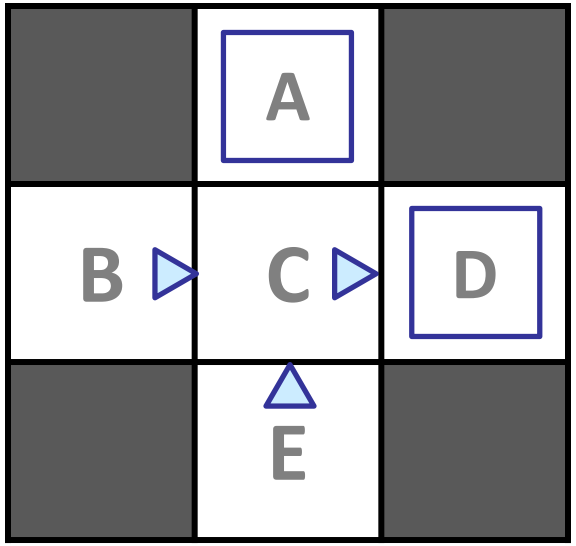
- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

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

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

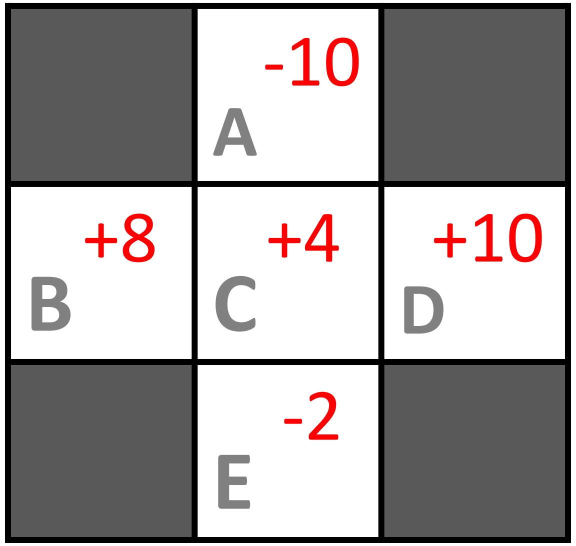
Episode 3

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

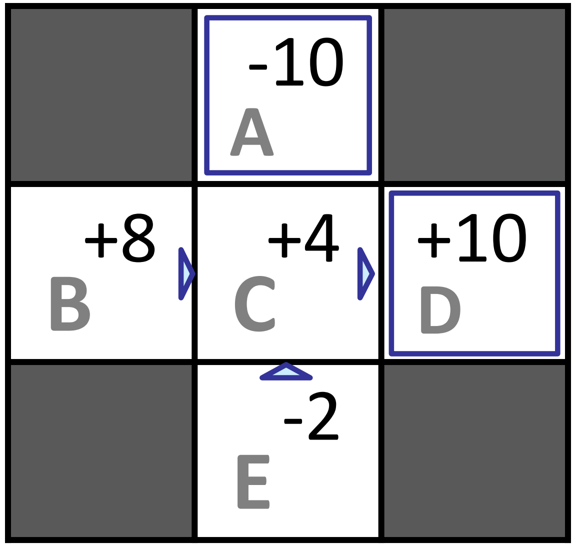


Problems with Direct Evaluation

- 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



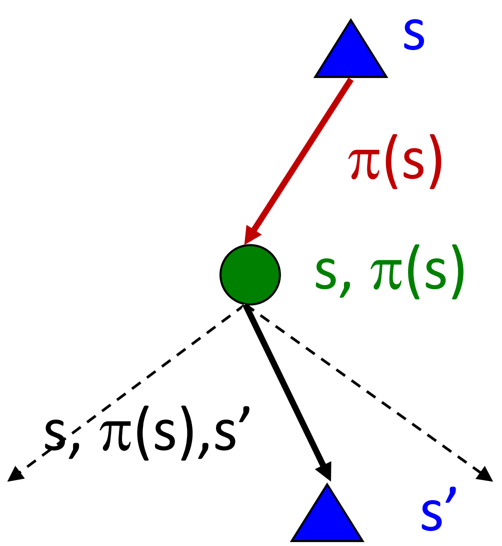
B goes to C, so we may use Bellman equation

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



  - This approach fully exploits 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 do we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

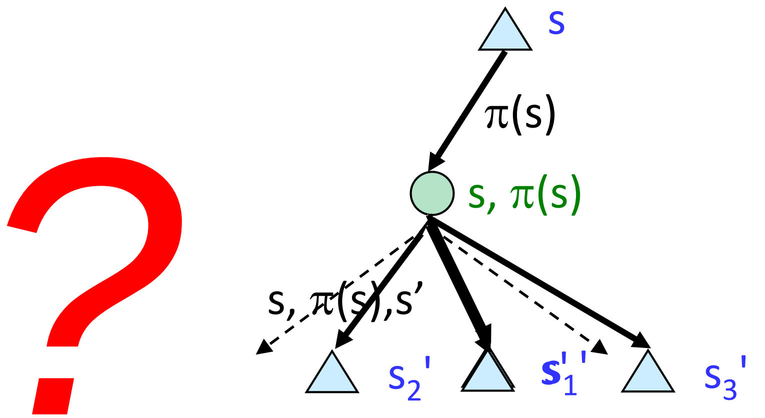
- We want to compute these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s, \pi(s), s_2') + \gamma V_k^{\pi}(s_2')$$
$$\dots$$
$$sample_n = R(s, \pi(s), s_n') + \gamma V_k^{\pi}(s_n')$$

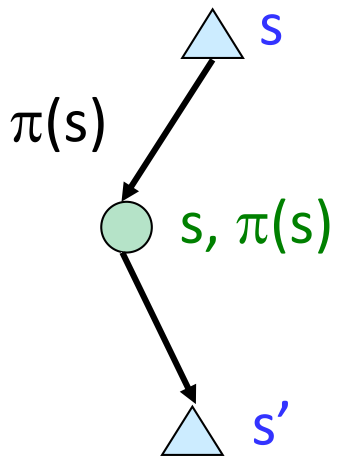
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$$



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

Temporal Difference Learning

- Big idea: learn immediately from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
- Temporal difference learning of values
  - (Policy still fixed, still doing evaluation!)
  - Move the value towards the sample



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

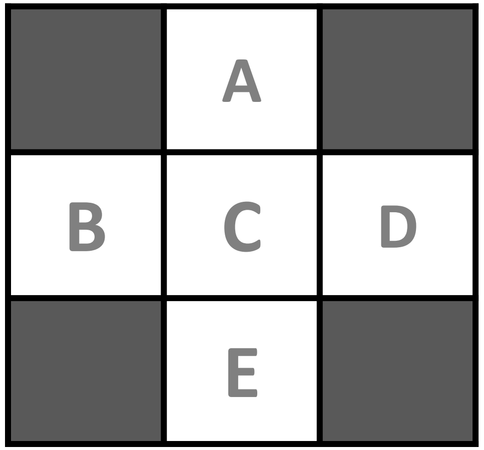
- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important
  - Forgets about the past (distant past values were wrong anyway)

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

- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

States

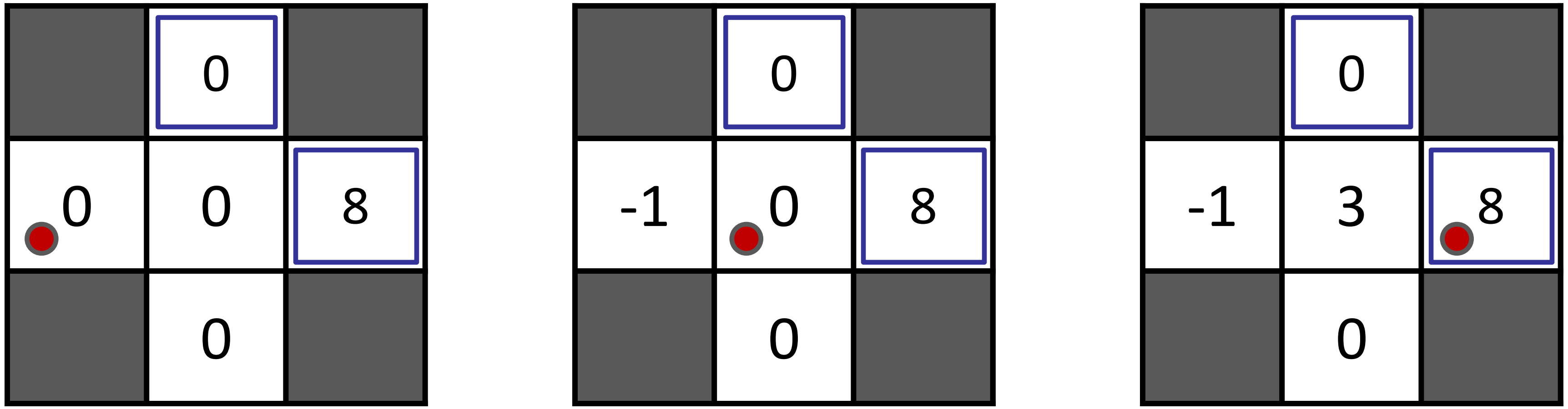


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, C, -2

C, east, D, -2



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

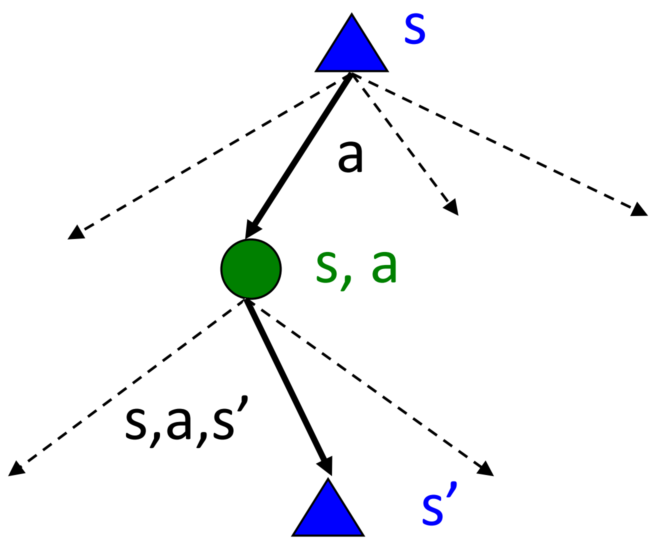
Limitations of TD Value Learning

- 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…

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

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

Unknown!



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

Q-Learning

- Q-value iteration

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

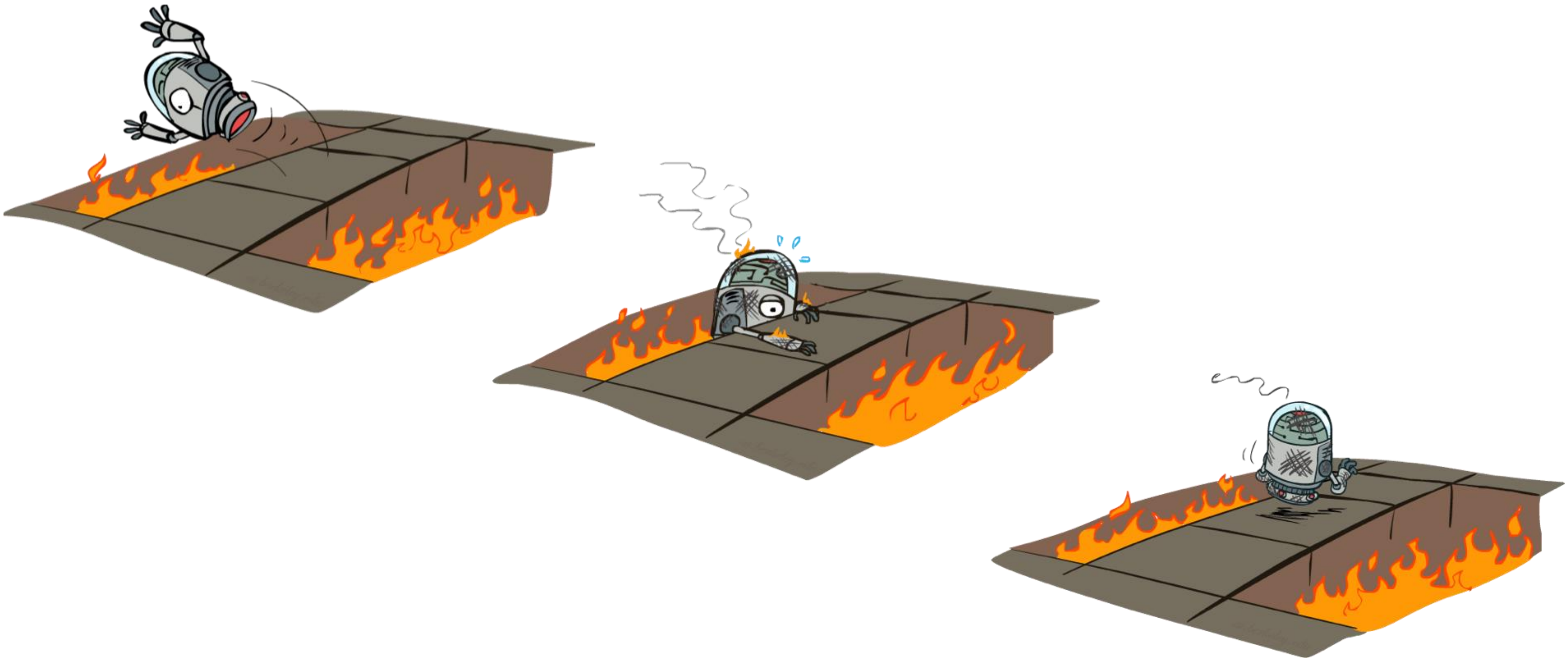
- Q-Learning: learn Q(s,a) values as you go
  - Receive a sample (s,a,s’,r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

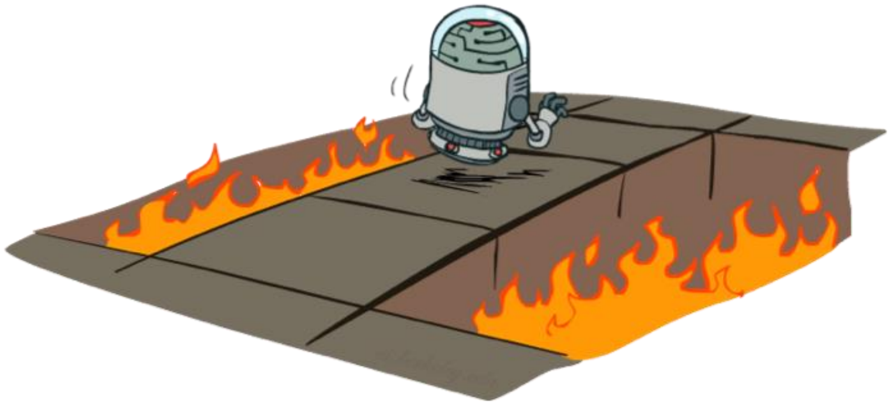
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Active Reinforcement Learning



Active Reinforcement Learning

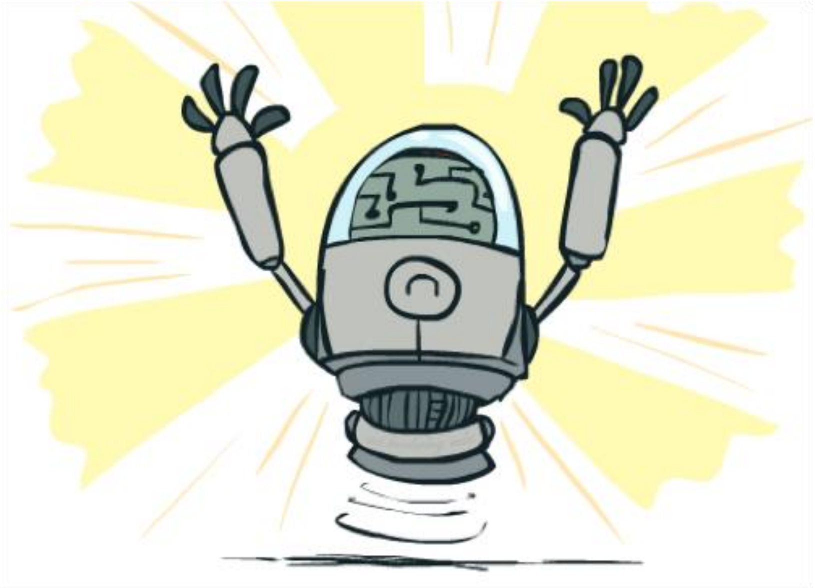
- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



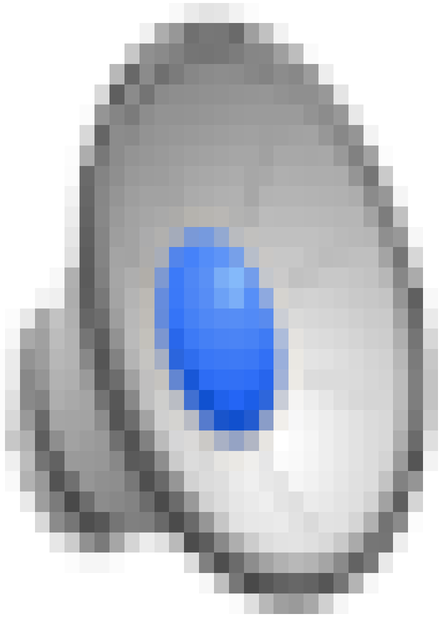
- Q-learning:
  - Learner makes choices (according to current values / policy, and also explore…)
    - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Learning Properties

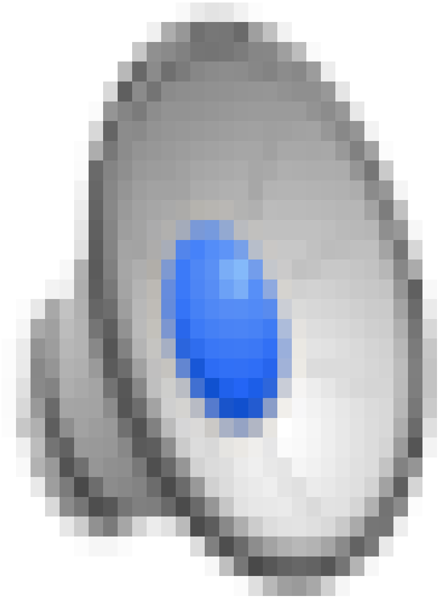
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly



Video of Demo Q-Learning -- Cliff Grid



Video of Demo Q-Learning -- Crawler



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V^*, Q^*, π^*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

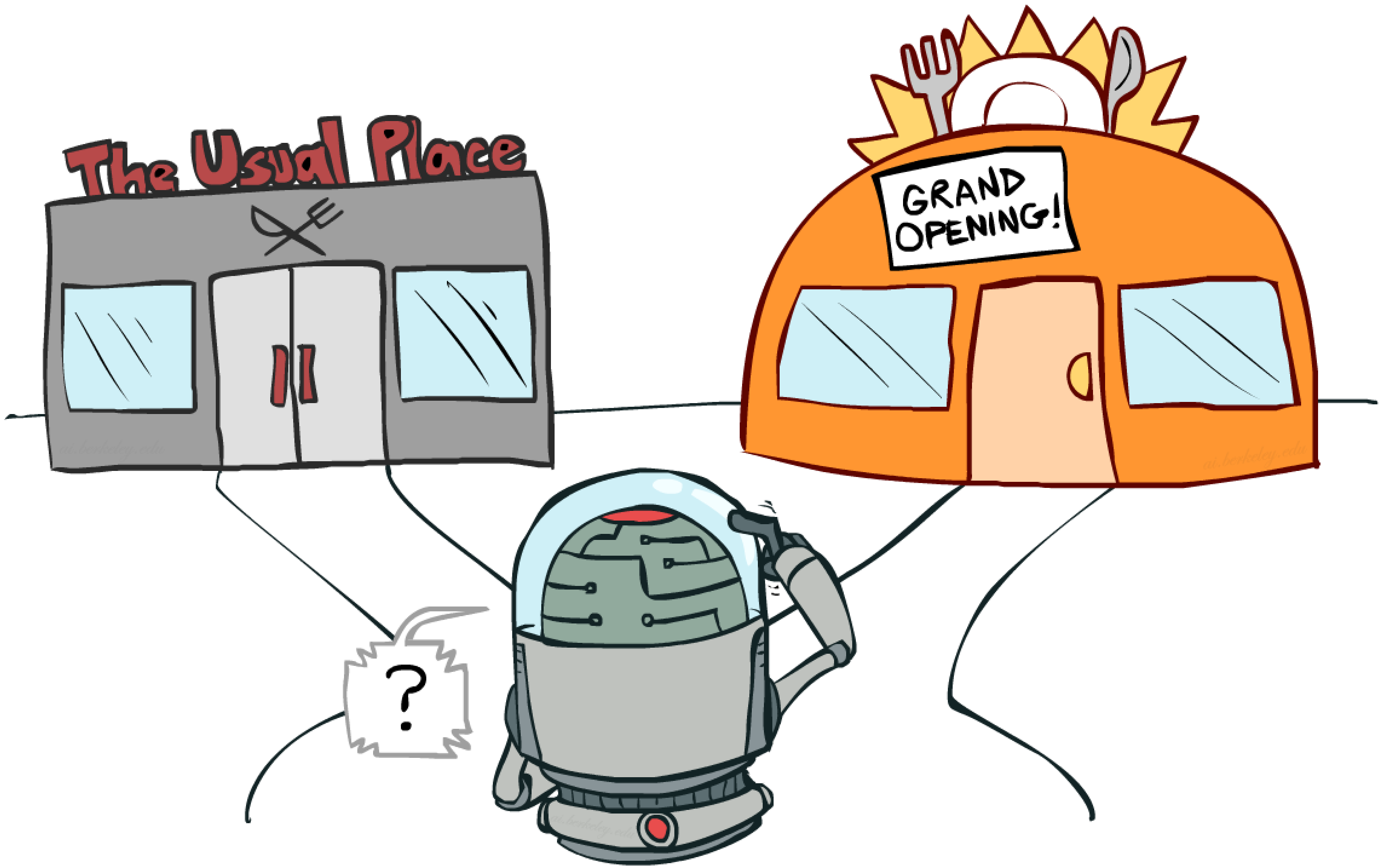
Unknown MDP: Model-Based

Goal	Technique
Compute V^*, Q^*, π^*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V^*, Q^*, π^*	Q-learning
Evaluate a fixed policy π	TD Value Learning

Exploration vs. Exploitation



How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

Video of Demo Q-learning – Epsilon-Greedy – Crawler

How to Explore?

- Several schemes for forcing exploration
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

Exploration Functions

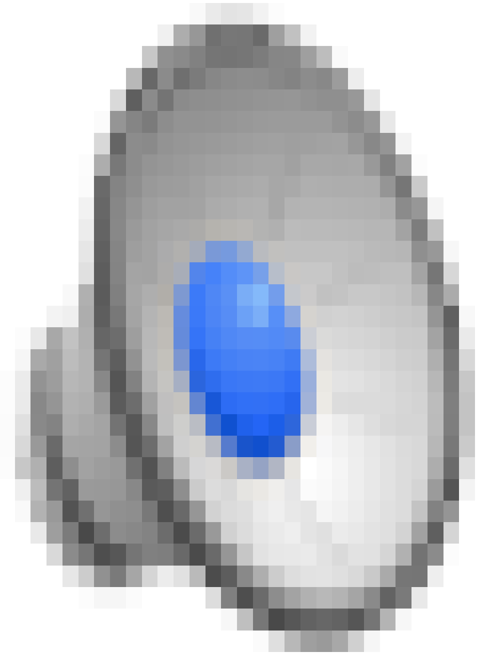
- When to explore?
  - Explore states that haven't been sufficiently explored
  - Eventually stop exploring
- Idea: select actions based on modified Q-value
  - Exploration function: takes a Q-value estimate u and a visit count n, and returns an optimistic utility, e.g.
    $f(u,n) = u + k/n$
  - Q-Update

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

Modified Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

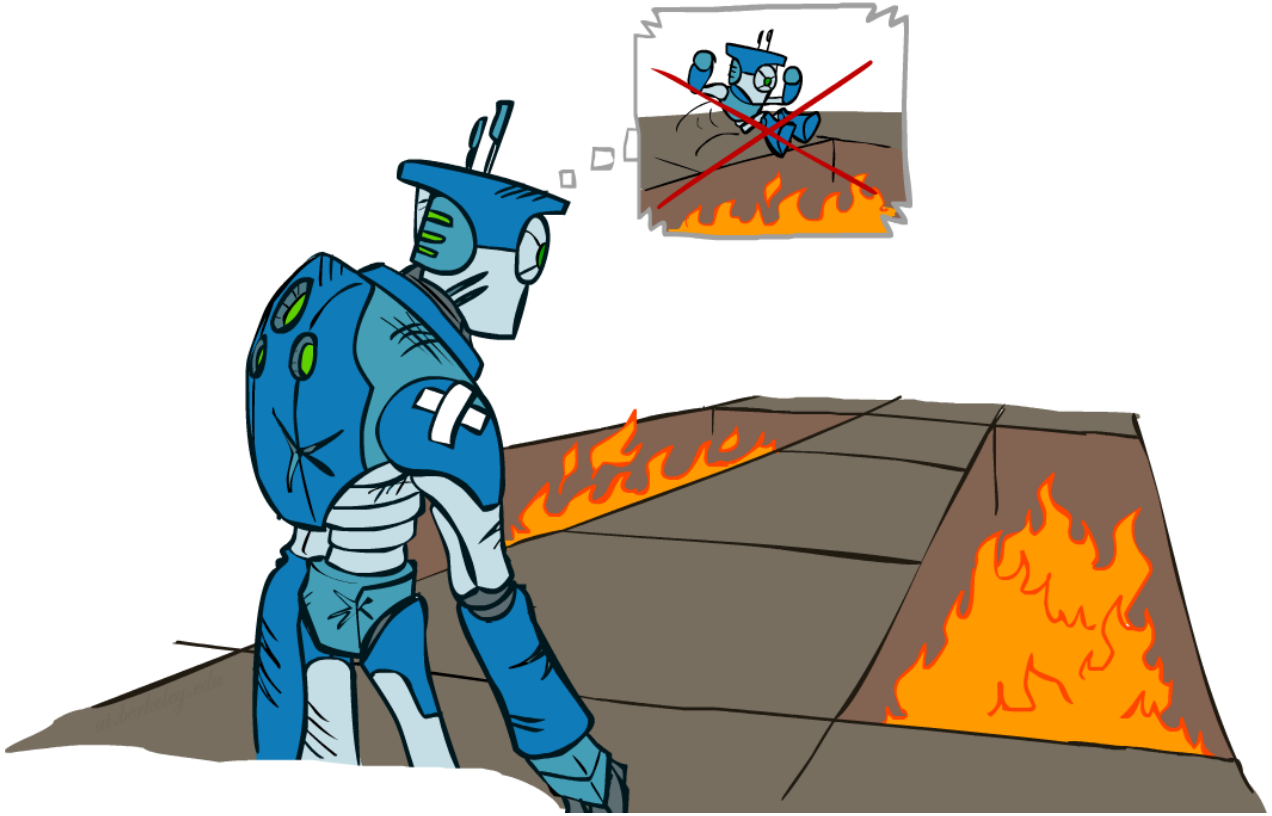
This propagates the "bonus" back to states that lead to under-explored states

Video of Demo Q-learning – Exploration Function – Crawler

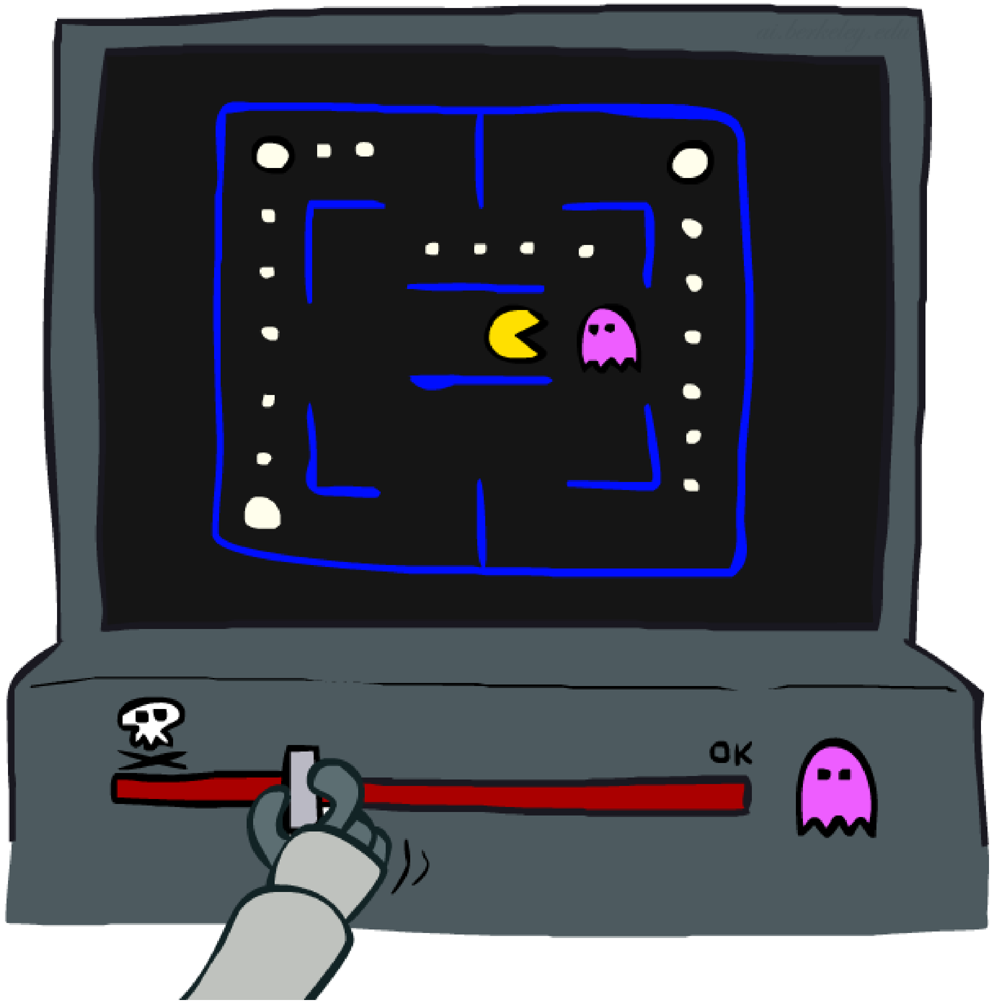


Regret

- 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

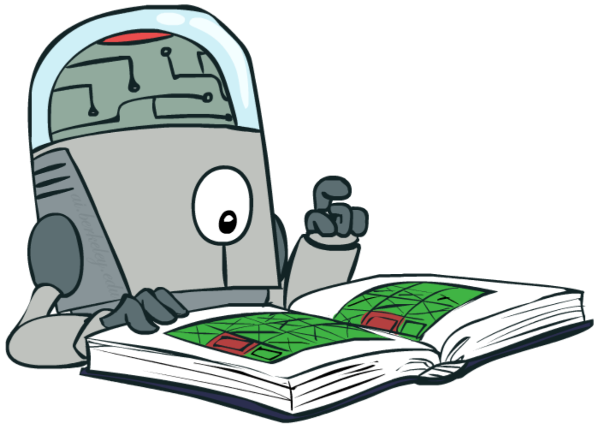


Approximate Q-Learning

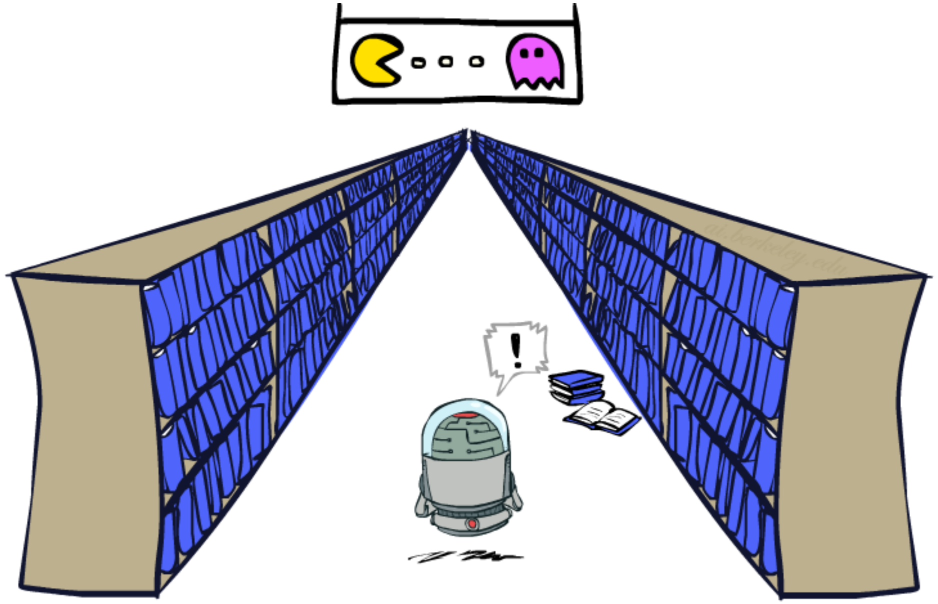


Generalizing Across States

- Basic Q-Learning keeps a table of all q-values

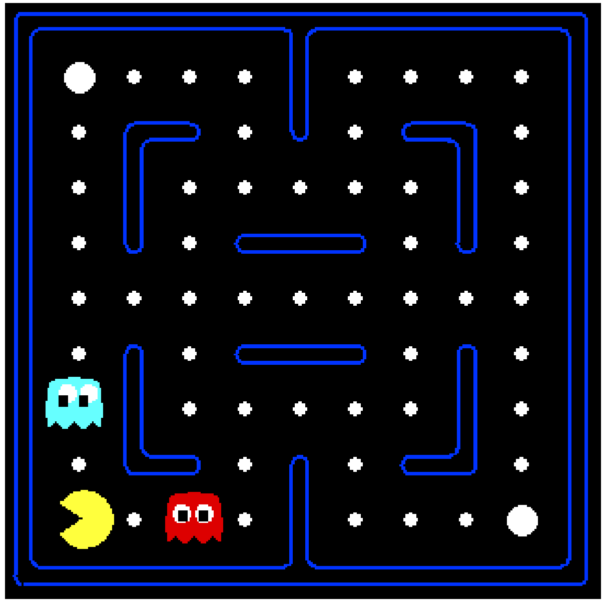


- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

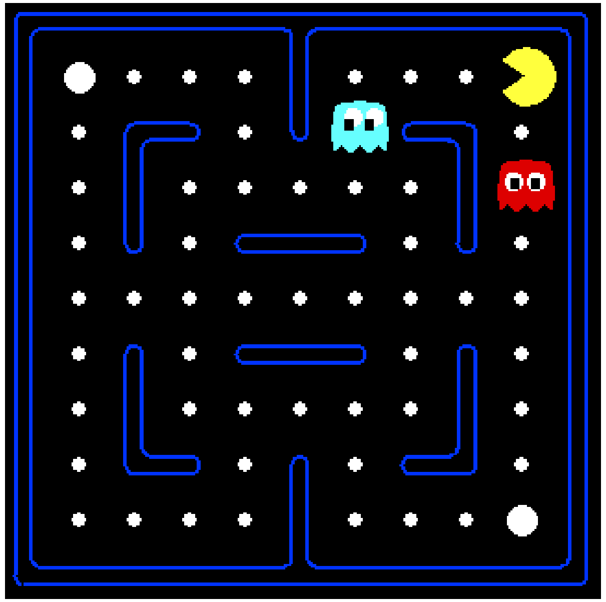


Example: Pacman

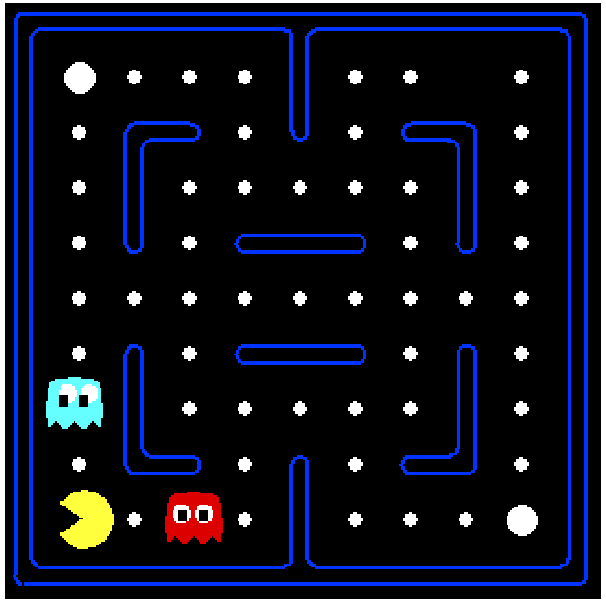
Let's say we discover through experience that this state is bad:



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

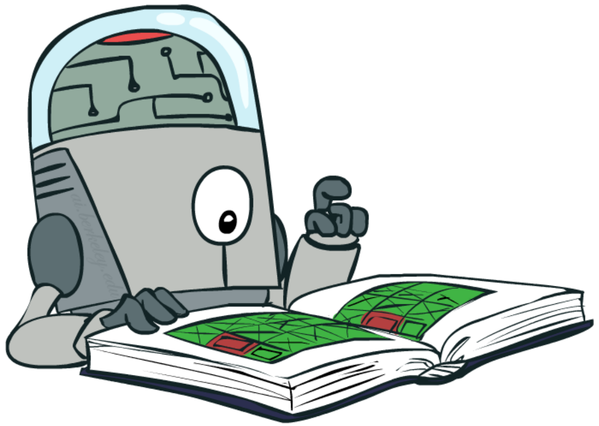


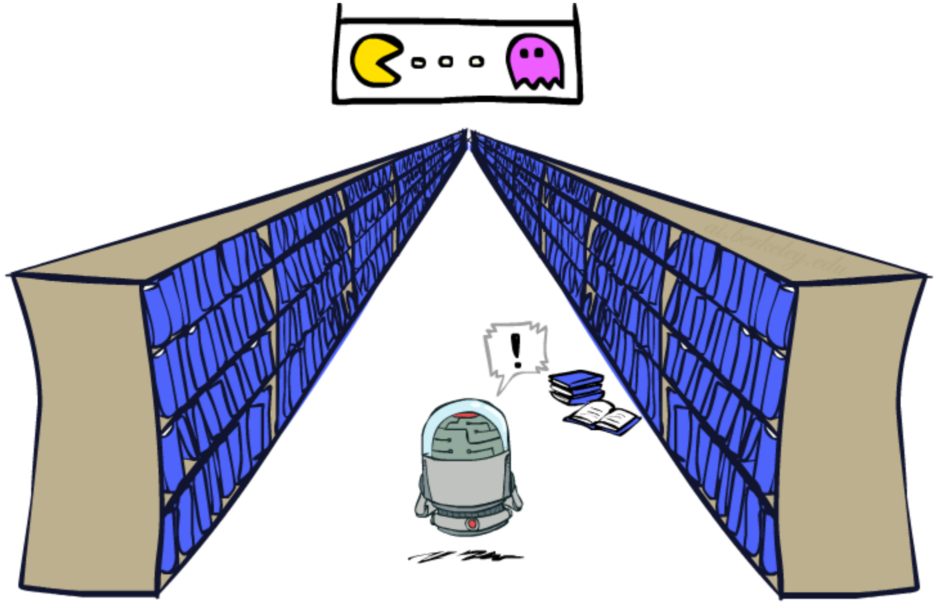
Or even this one!



Generalizing Across States

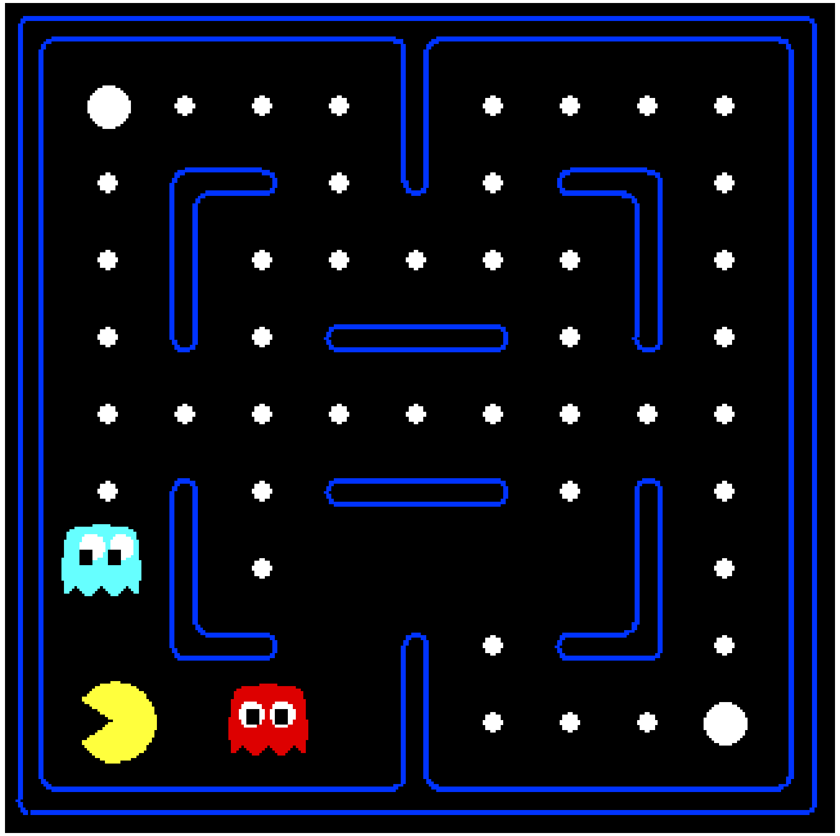
- We want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it again later





Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - 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 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value 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 actually be very different in value!

Approximate Q-Learning

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

- Q-learning with linear Q-functions:

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

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

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

Exact Q's

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

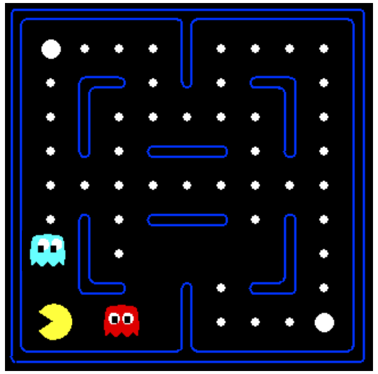
Approximate Q's
(based on online least squares)

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

Example: Q-Pacman

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

s



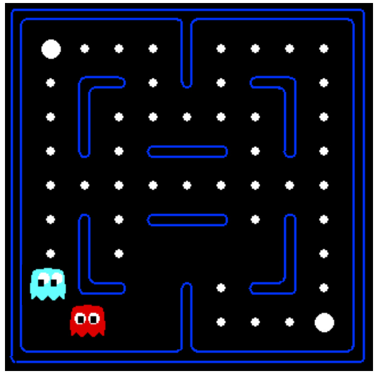
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

$Q(s', \cdot) = 0$

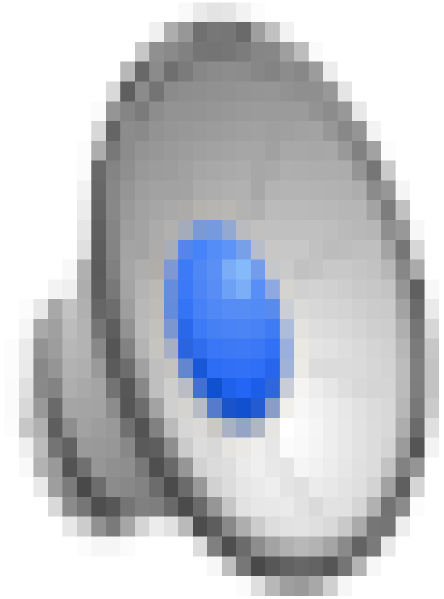
difference $= -501$

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

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

If $\alpha = 0.004$: $Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$

Video of Demo Approximate Q-Learning -- Pacman



More Powerful Functions

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

Polynomial: $Q(s,a) = w_{11} f_1(s,a) + w_{12} f_1(s,a)^2 + w_{13} f_1(s,a)^3 + \ldots$

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

learn these too

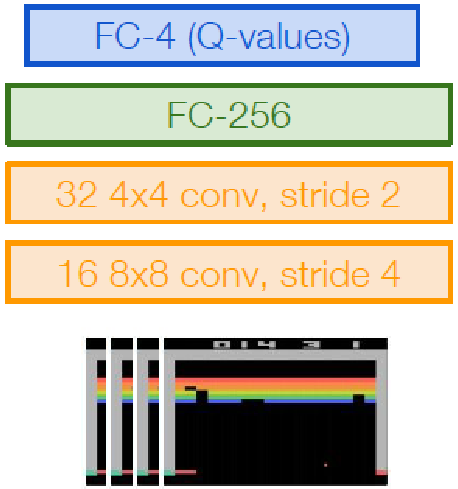
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s',a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$

$= f_m(s,a)$ in linear case

Example: Atari Games

- Q-network

$Q(s, a; \theta)$:
neural network
with weights θ

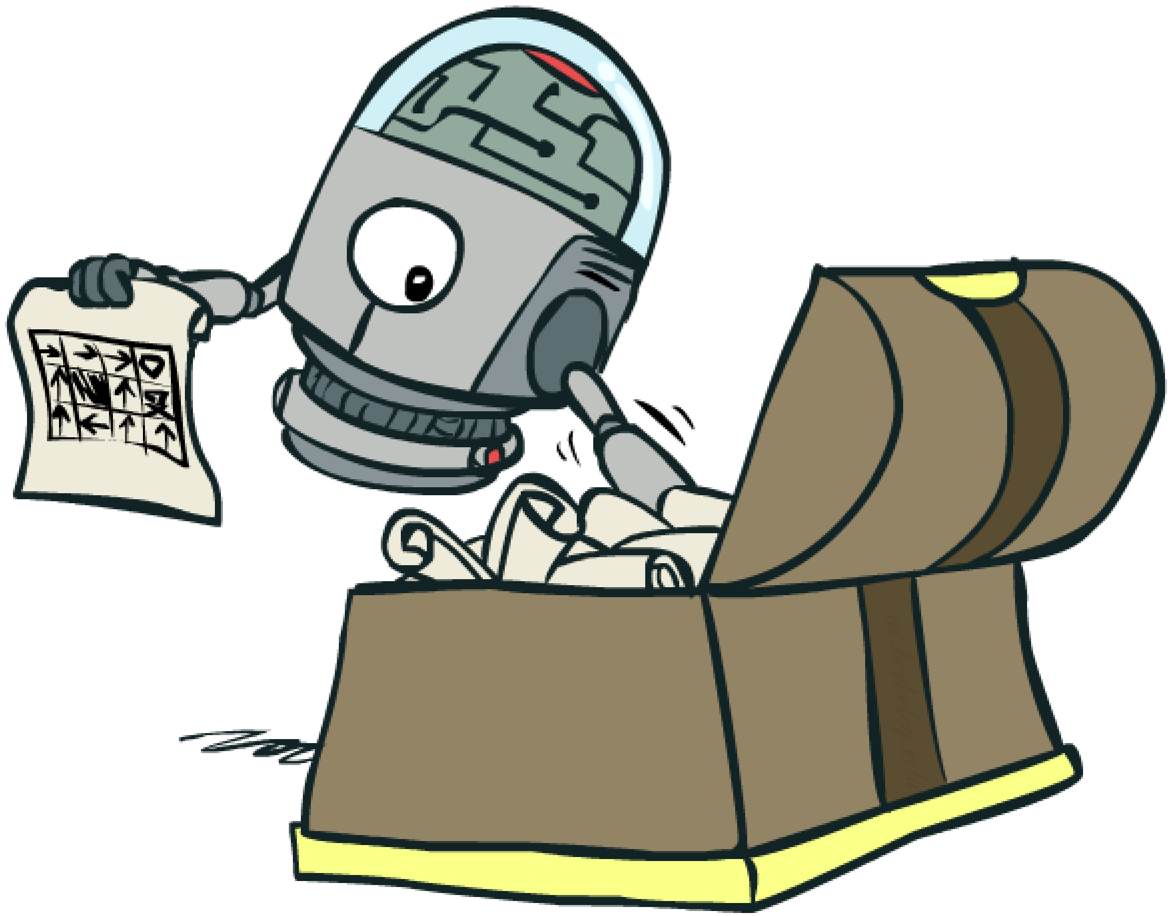


Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Policy Search



Policy Search

- Q-learning's priority: get Q-values close
- Observation: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from PA 1b were probably horrible estimates of future rewards, but they still produced good decisions
  - The real priority: get ordering of Q-values right (action prediction)

- Idea: learn policies that maximize rewards, not the values that predict them
- Policy search: start with an OK solution (e.g., approximate Q-learning), then fine-tune feature weights to find a better policy

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Change 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
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Summary

- Reinforcement learning
  - MDP without knowing T and R
- Model-based learning
- Model-free learning
  - Policy evaluation: TD Learning
  - Computing q-values/policy: Q-Learning
- Exploration vs. Exploitation
  - Random exploration, exploration function
- Feature-based representation of states
- Policy Search