

CS 188: Artificial Intelligence

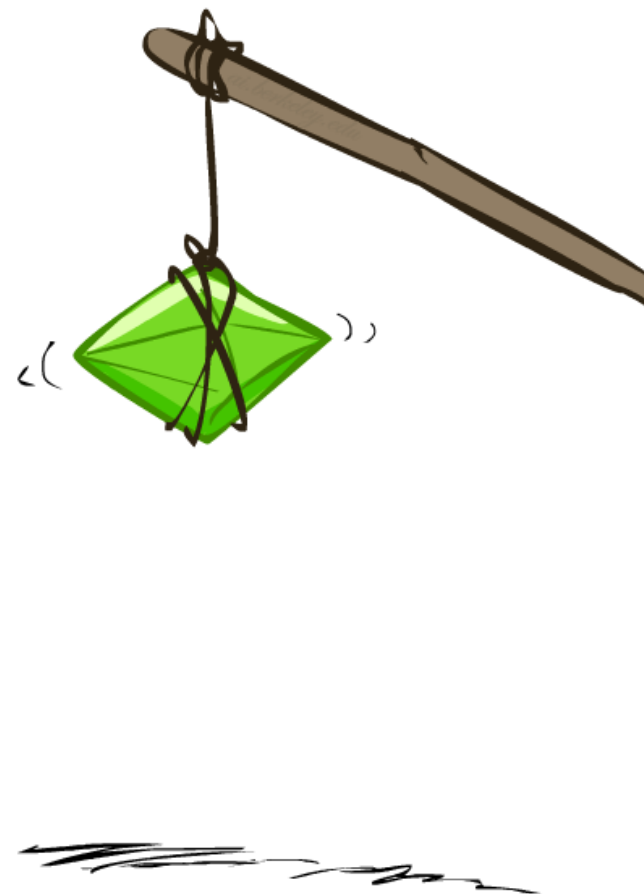
Reinforcement Learning



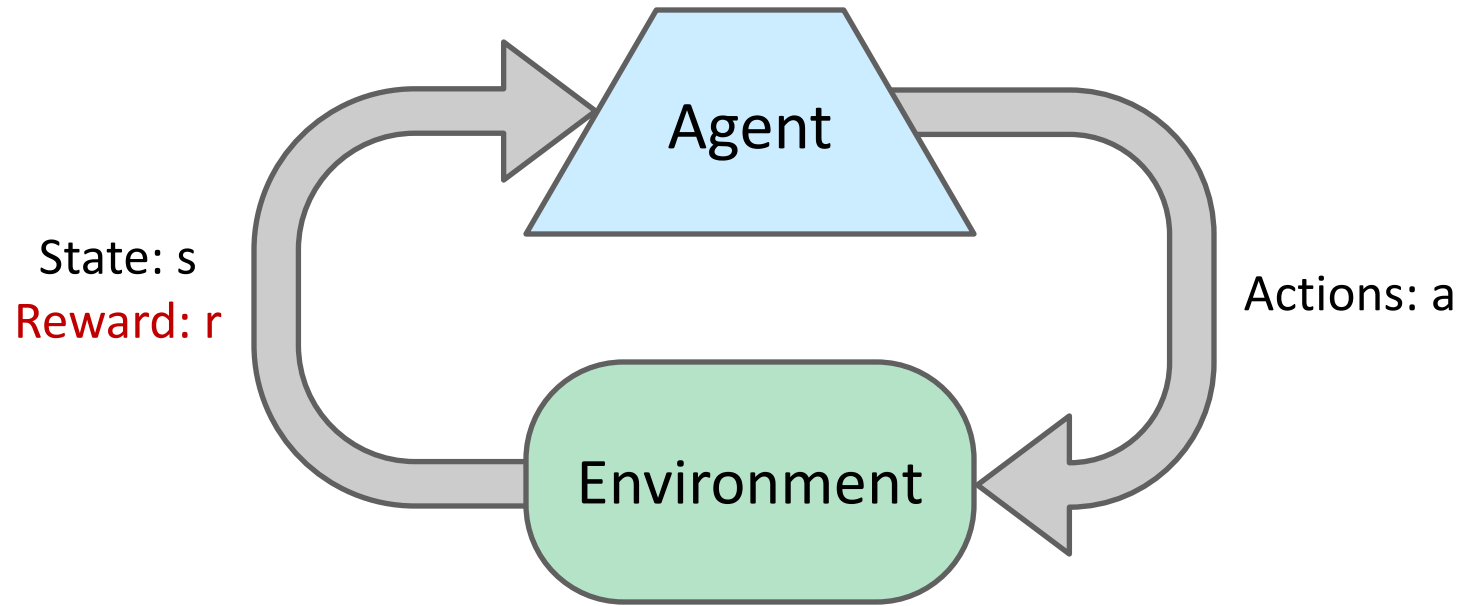
Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

Reinforcement Learning



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



A Learning Trial



After Learning [1K Trials]

Example: Learning to Walk



Initial

Example: Learning to Walk



Training

Example: Learning to Walk

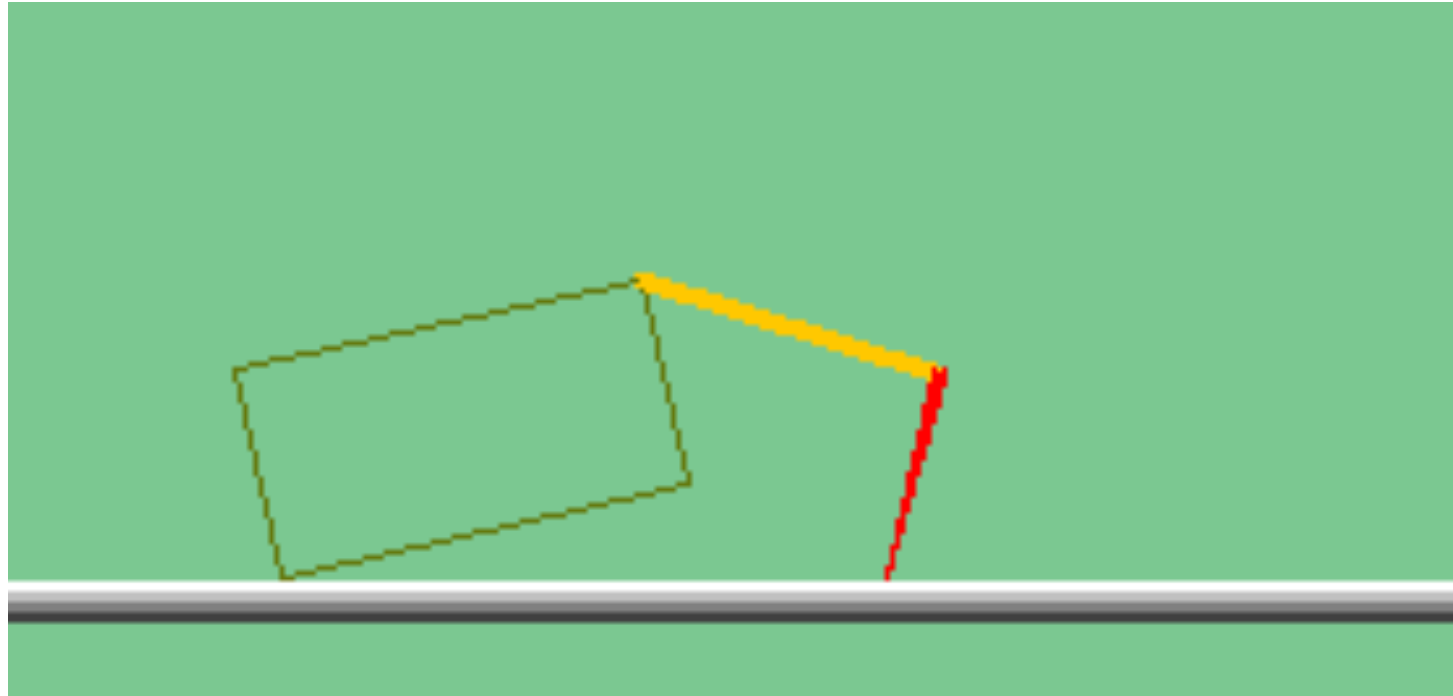


Finished

Example: Toddler Robot



The Crawler!



Video of Demo Crawler Bot

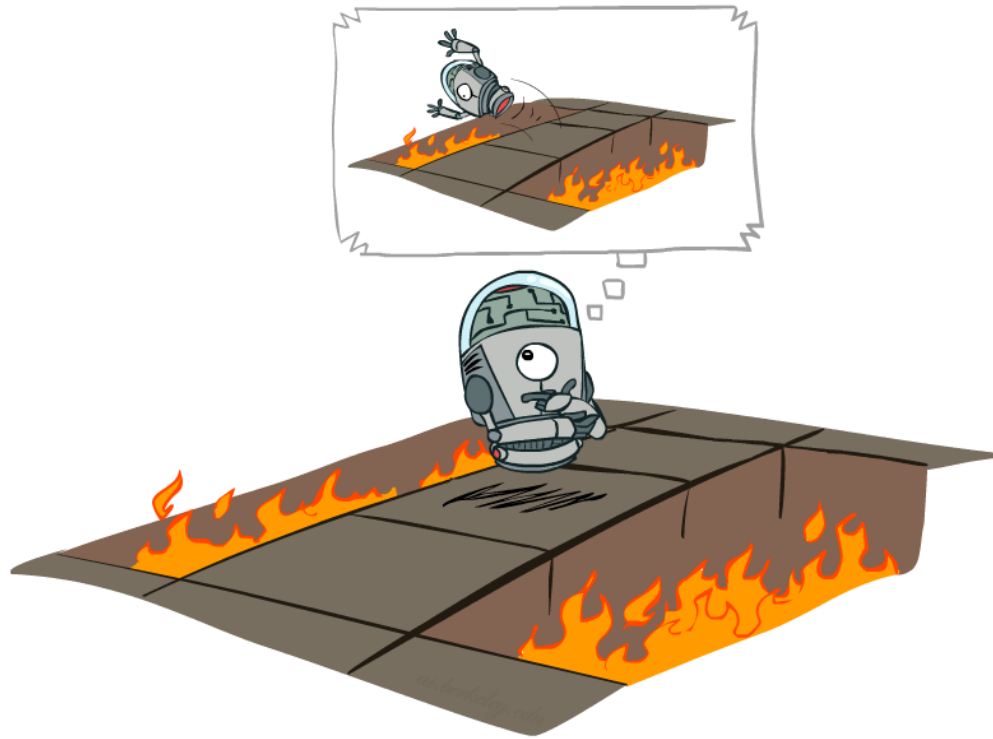


Reinforcement Learning

- 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)

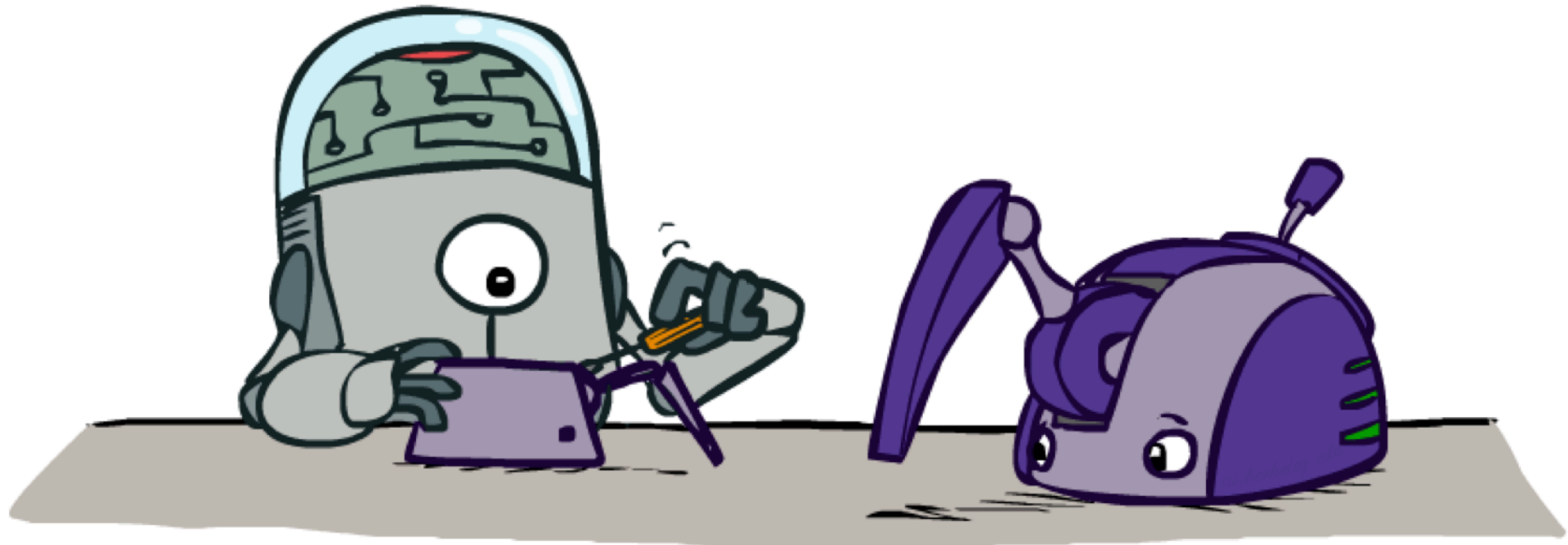


Offline Solution



Online Learning

Model-Based Learning



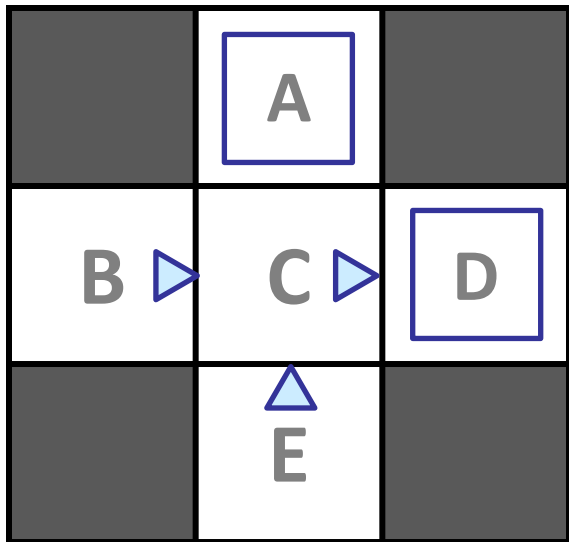
Model-Based Learning

- **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 $\hat{T}(s, a, s')$
 - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
- **Step 2: Solve the learned MDP**
 - For example, use value iteration, as before



Example: Model-Based Learning

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

Learned Model

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

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

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

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

Example: Expected Age

Goal: Compute expected age of cs188 students

Known $P(A)$

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

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

Unknown $P(A)$: “Model Based”

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$
$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

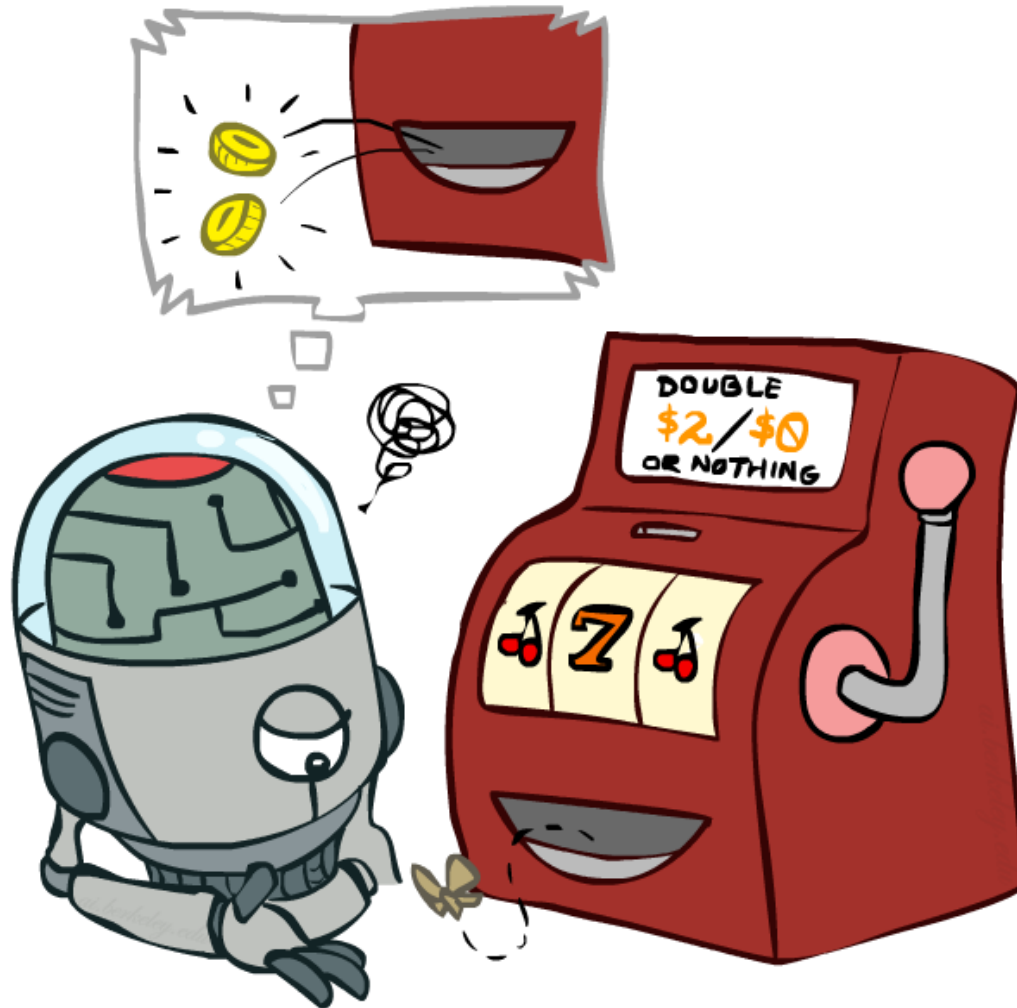
Why does this work? Because eventually you learn the right model.

Unknown $P(A)$: “Model Free”

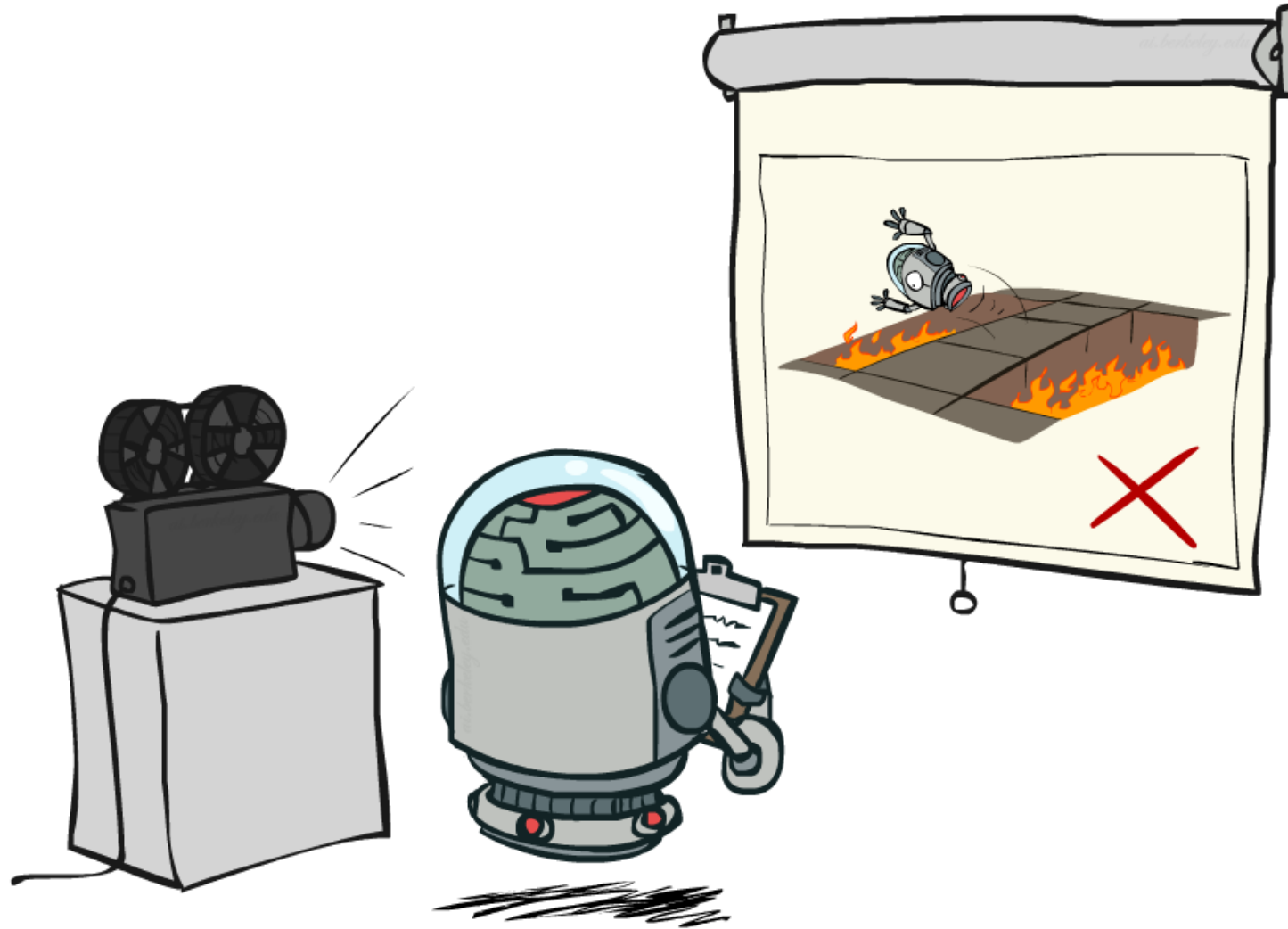
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

Model-Free Learning



Passive Reinforcement Learning



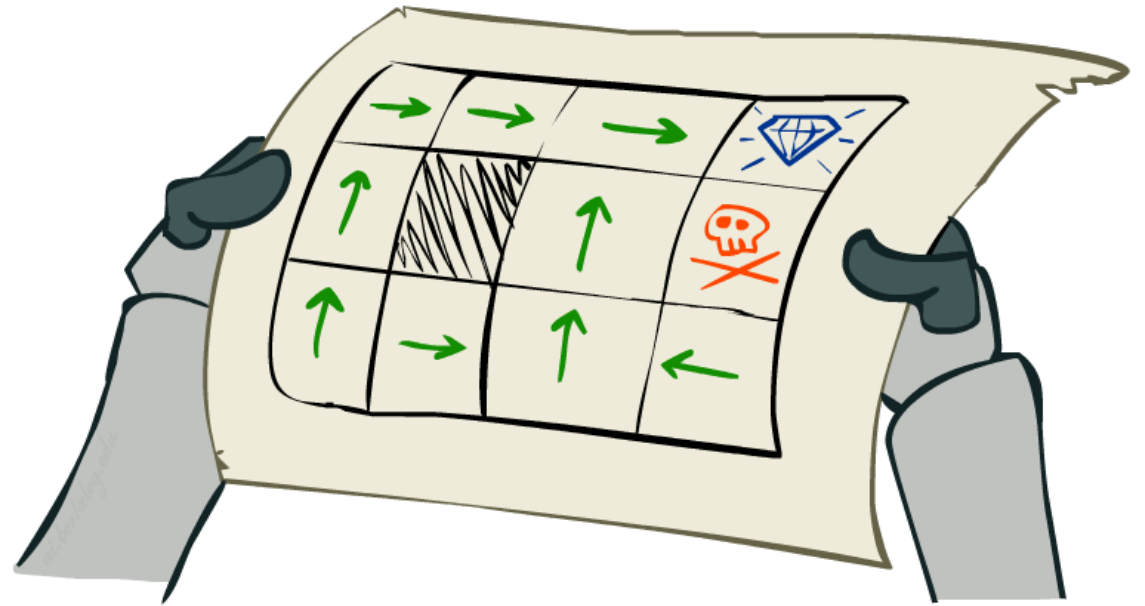
Passive Reinforcement Learning

- 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:

- 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.



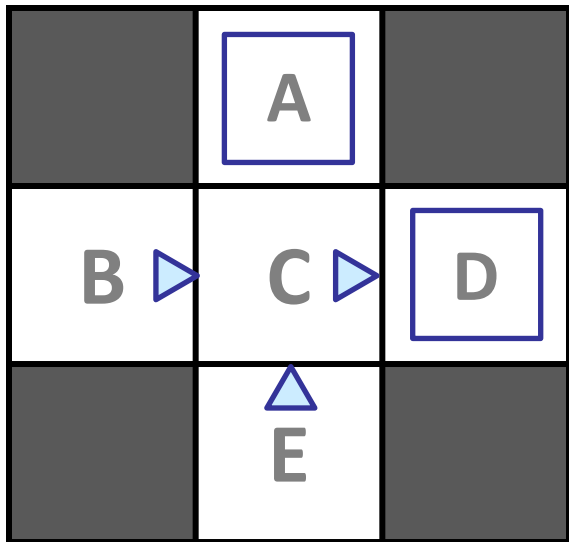
Direct Evaluation

- 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



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

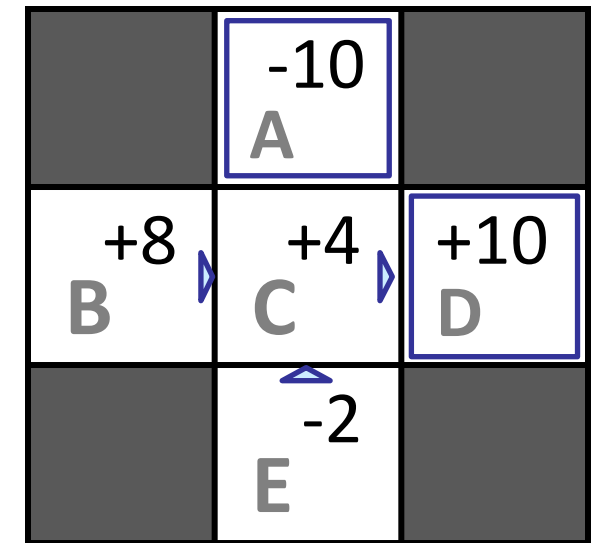
Output Values

	-10 A	
+8 B	+4 C	+10 D
	-2 E	

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?

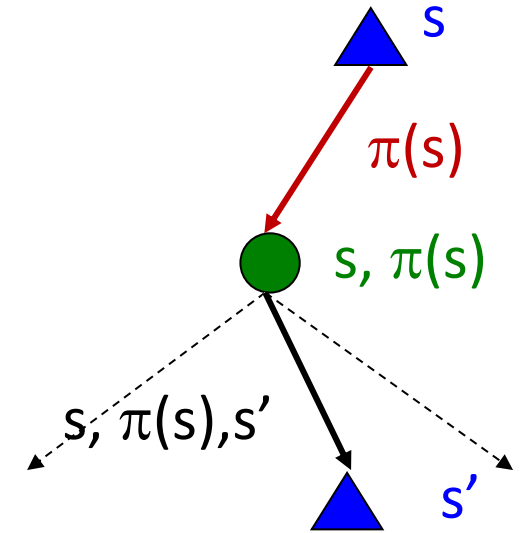
Why Not Use Policy Evaluation?

- 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

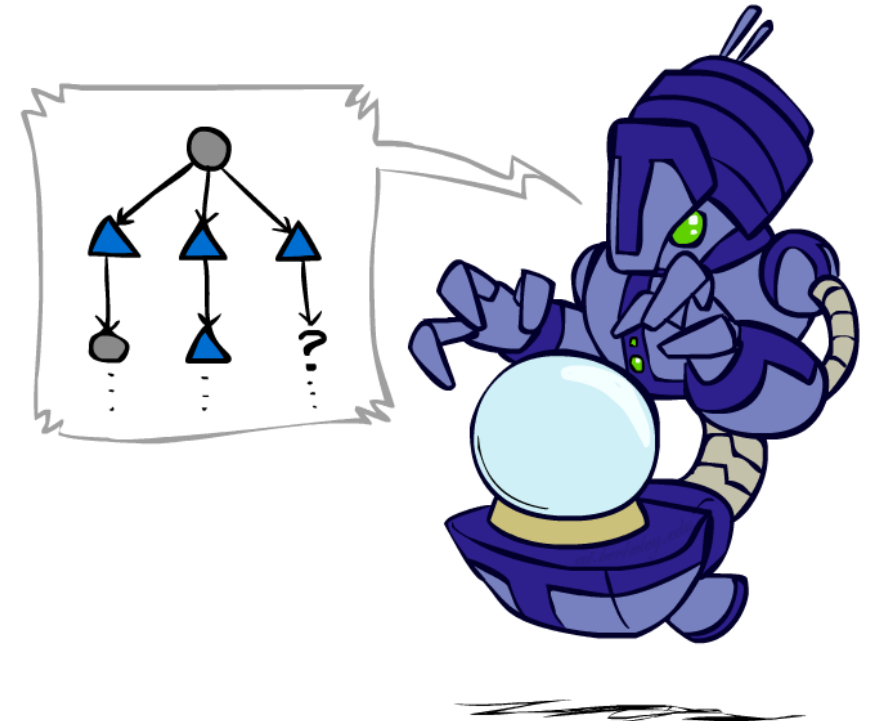
$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

...

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

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



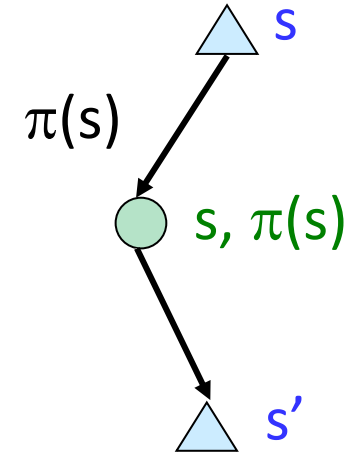
Temporal Difference Learning

- 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

B, east, C, -2

	0	
0	0	8
	0	

C, east, D, -2

	0	
-1	0	8
	0	

	0	
-1	3	8
	0	

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

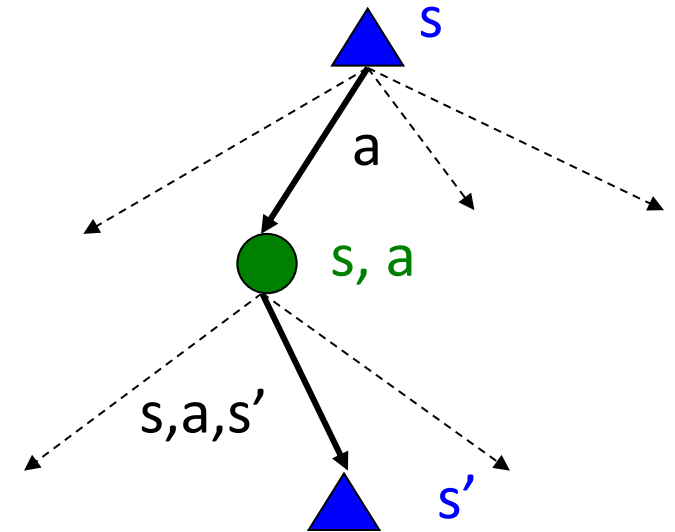
Problems with TD Value Learning

- 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:

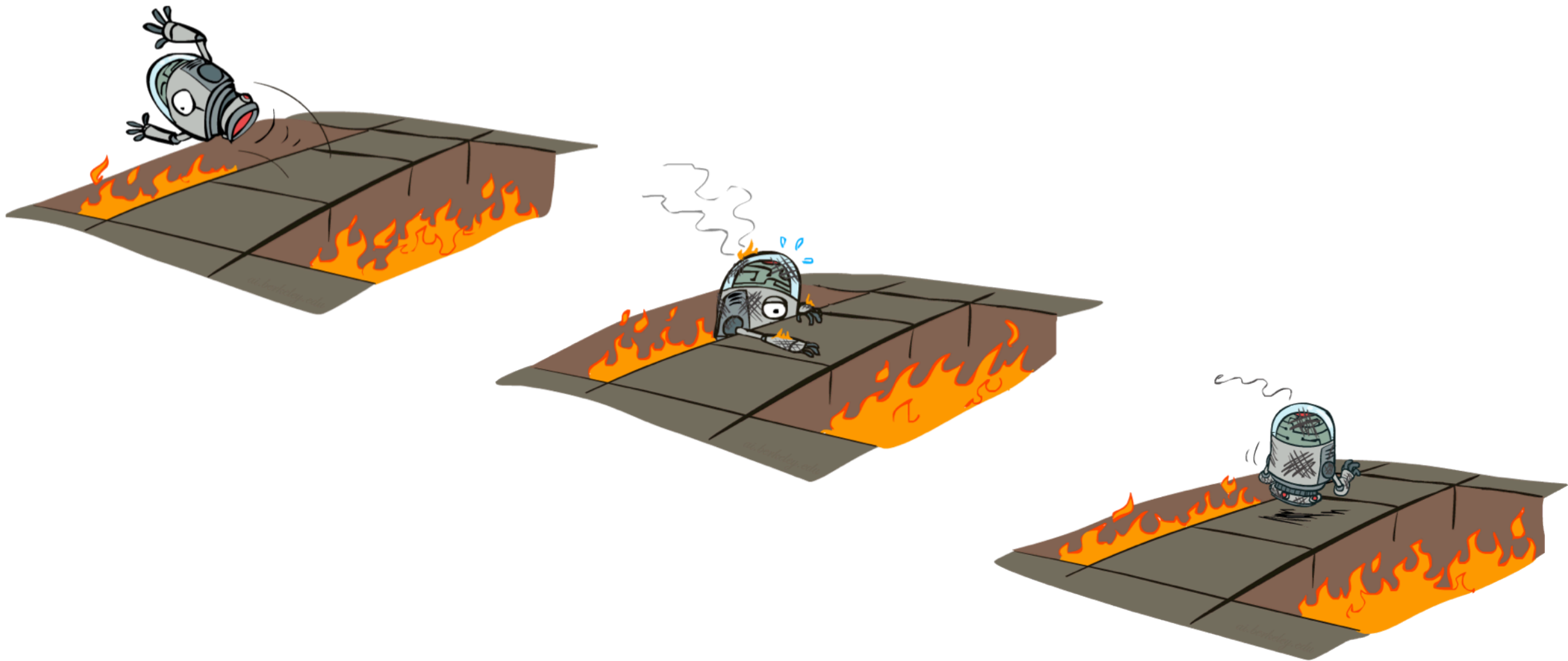
$$\pi(s) = \arg \max_a Q(s, a)$$

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

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

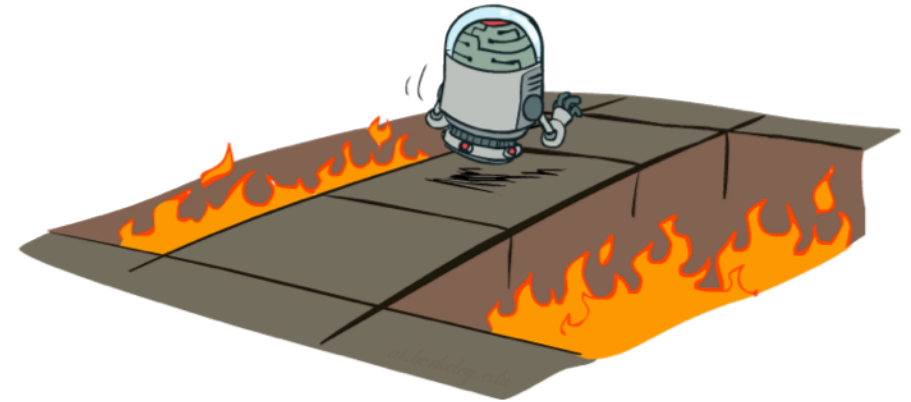


Active Reinforcement Learning



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...



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:

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

- 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:

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

Q-Learning

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

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

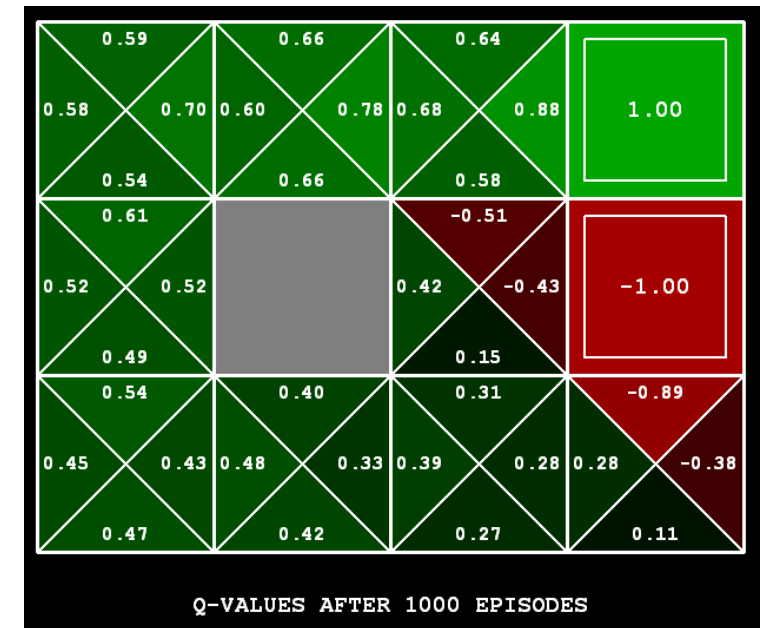
- Learn $Q(s,a)$ values as you go

- 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 (L10D2)]

[Demo: Q-learning – crawler (L10D3)]

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 learning rate small enough
 - ... but not decrease it too quickly
 - Basically, in the limit, it doesn't matter how you select actions (!)

