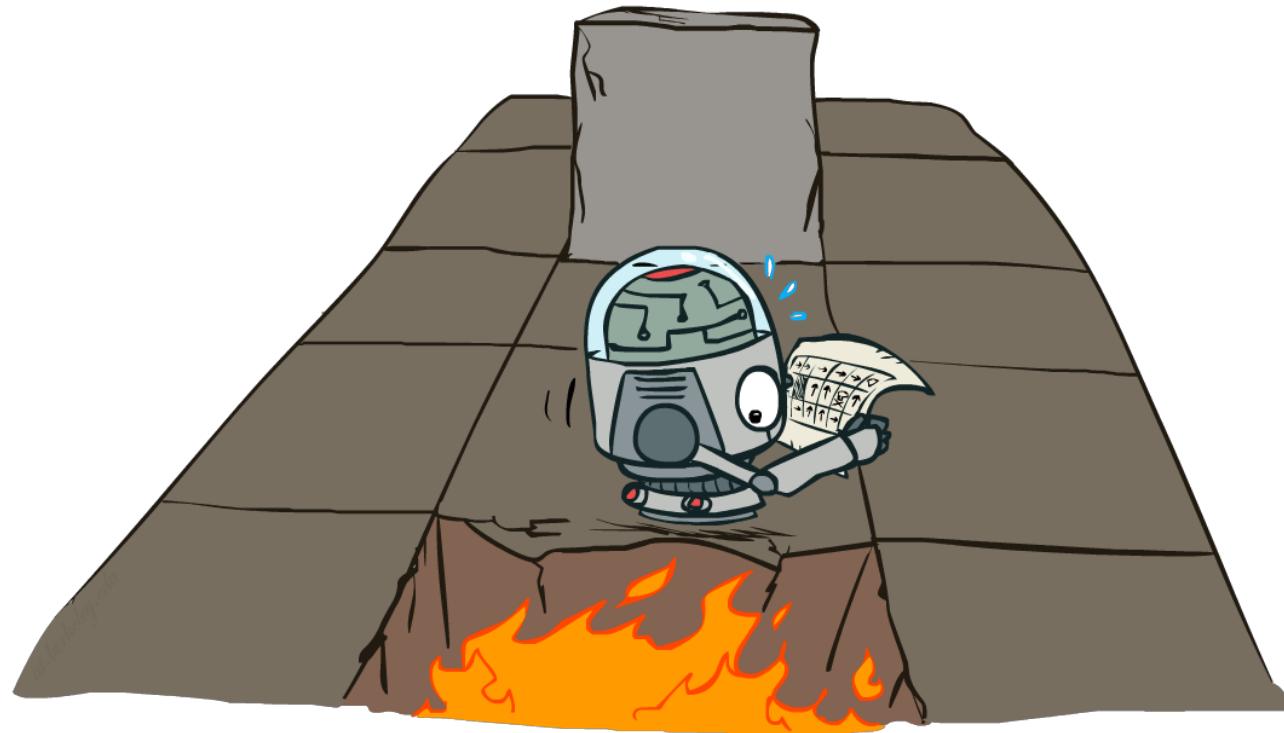


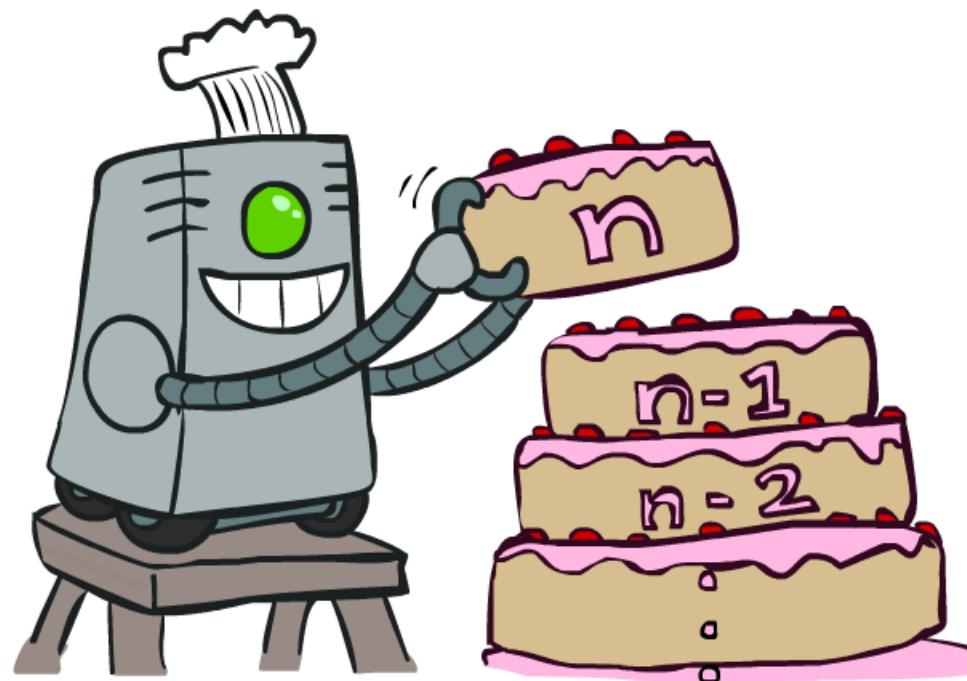
Markov Decision Processes -wrapup



Slides courtesy of Dan Klein and Pieter Abbeel --- University of California, Berkeley

[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>.]

Review: Value Iteration

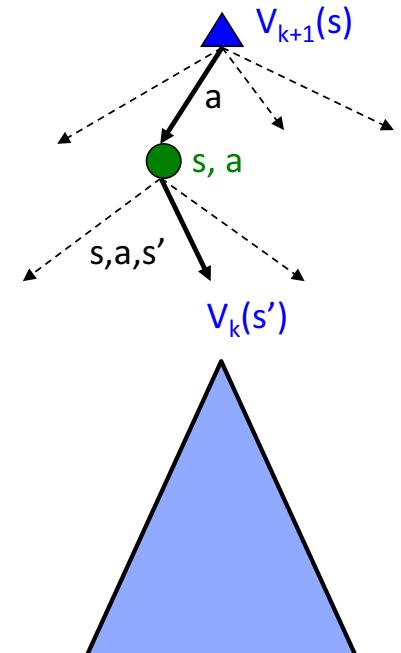


Review: Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- **Theorem: will converge to unique optimal values**
 - Basic idea: approximations get refined towards optimal values
 - Policy may converge long before values do

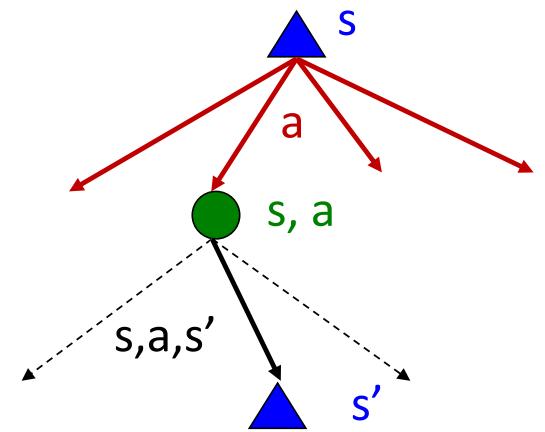


Review: Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The “max” at each state rarely changes
- Problem 3: The policy often converges long before the values



Policy Iteration

- Alternative approach for optimal values:
 - **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
 - **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
 - Repeat steps until policy converges
- **This is policy iteration**
 - It's still optimal!
 - Can converge (much) faster under some conditions

Policy Iteration

- Evaluation: For fixed current policy π , find values with policy evaluation:

- Iterate until values converge:

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

- Improvement: For fixed values, get a better policy using policy extraction

- One-step look-ahead:

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Comparison

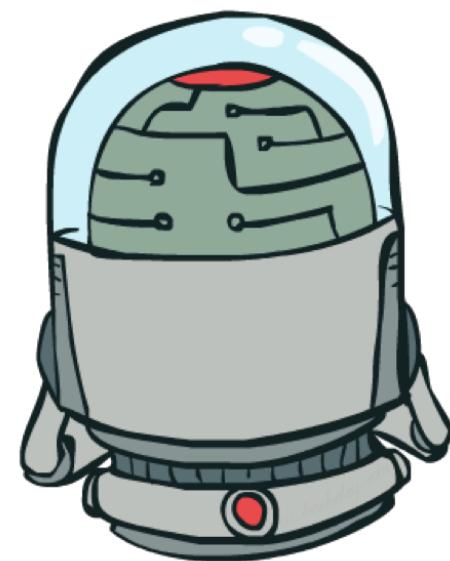
- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
 - Every iteration updates both the values and (implicitly) the policy
 - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
 - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
 - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
 - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

Summary: MDP Algorithms

- So you want to....
 - Compute optimal values: use value iteration or policy iteration
 - Compute values for a particular policy: use policy evaluation
 - Turn your values into a policy: use policy extraction (one-step lookahead)

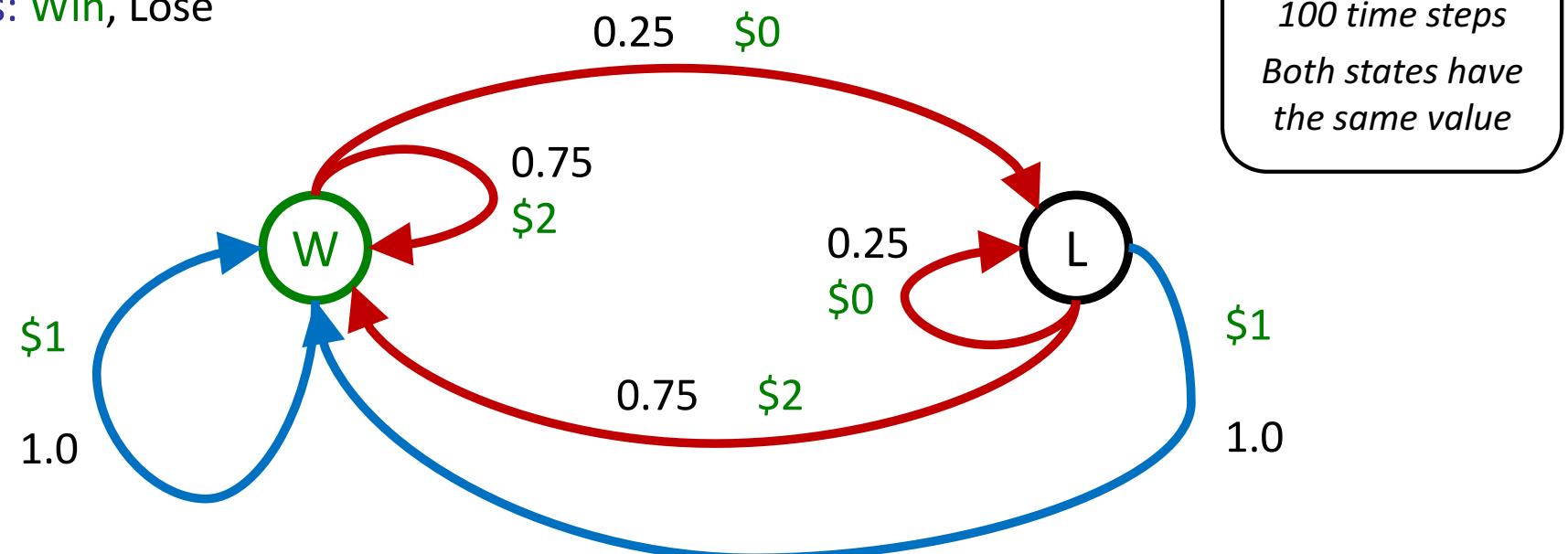
- These all look the same!
 - They basically are – they are all variations of Bellman updates
 - They all use one-step lookahead expectimax fragments
 - They differ only in whether we plug in a fixed policy or max over actions

Double Bandits



Double-Bandit MDP

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

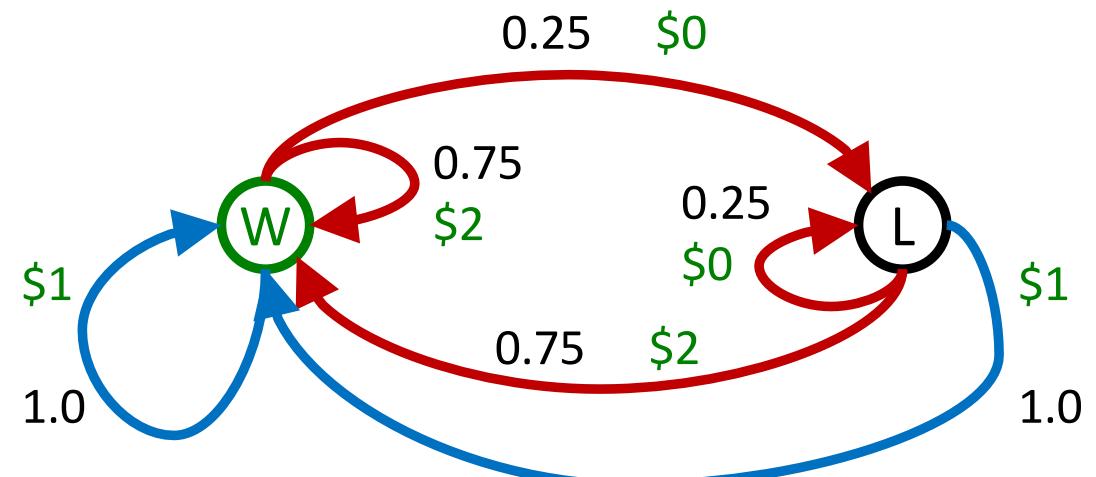


Offline Planning

- 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
Both states have
the same value*

	Value
Play Red	150
Play Blue	100



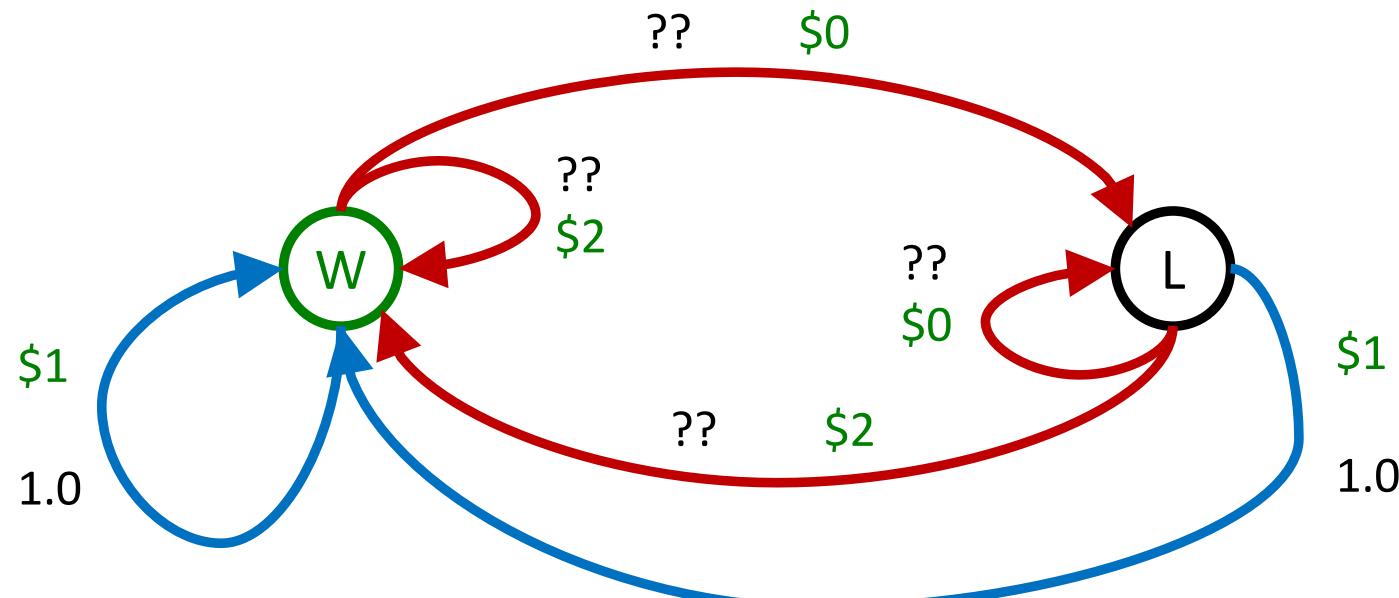
Let's Play!



\$2	\$2	\$0	\$2	\$2
\$2	\$2	\$0	\$0	\$0

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?

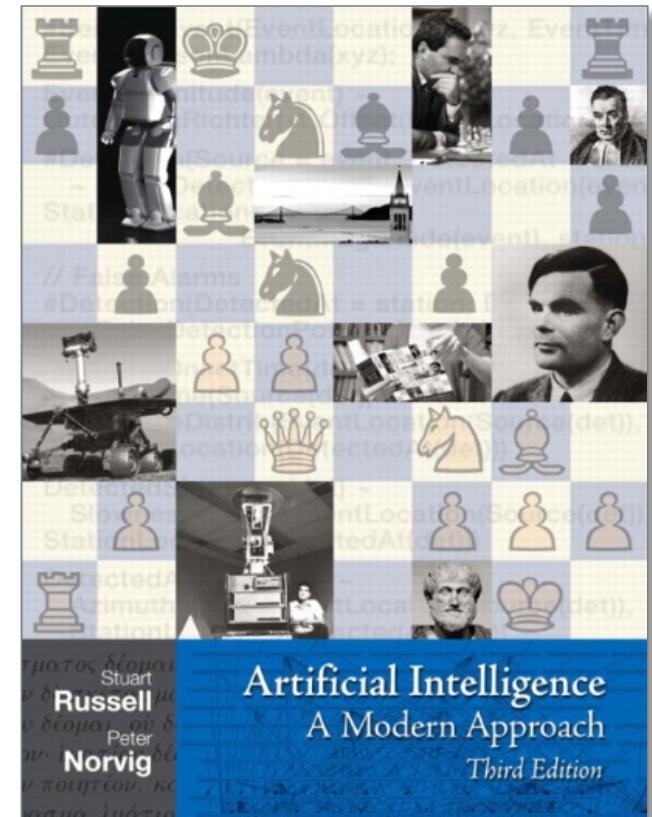
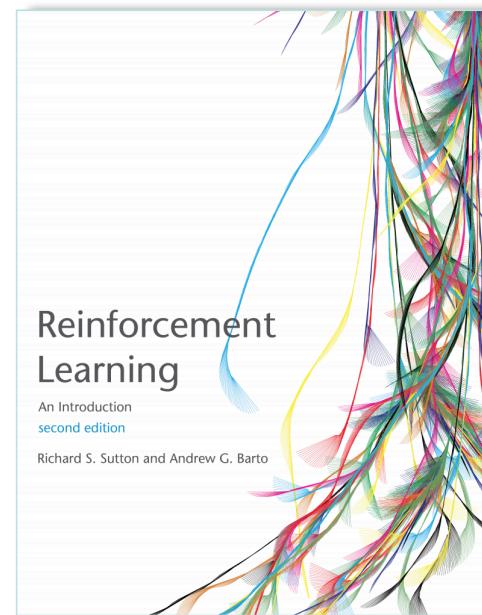
- That wasn't planning, it was learning!
 - Specifically, reinforcement learning
 - There was an MDP, but you couldn't solve it with just computation
 - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
 - **Exploration:** you have to try unknown actions 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



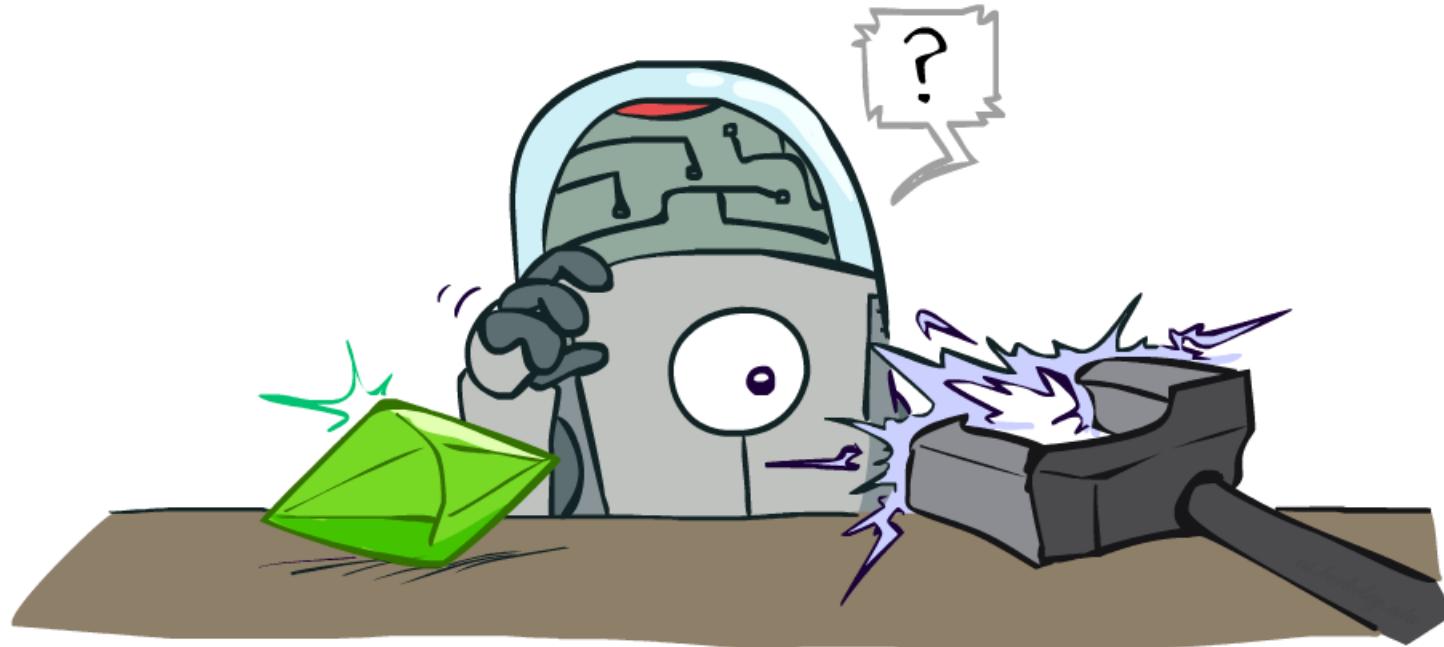
Reinforcement Learning

Read AIMA 21

Also read Sutton and Barto Chapter
6.1, 6.2 and 6.5 (see link on course
web site)



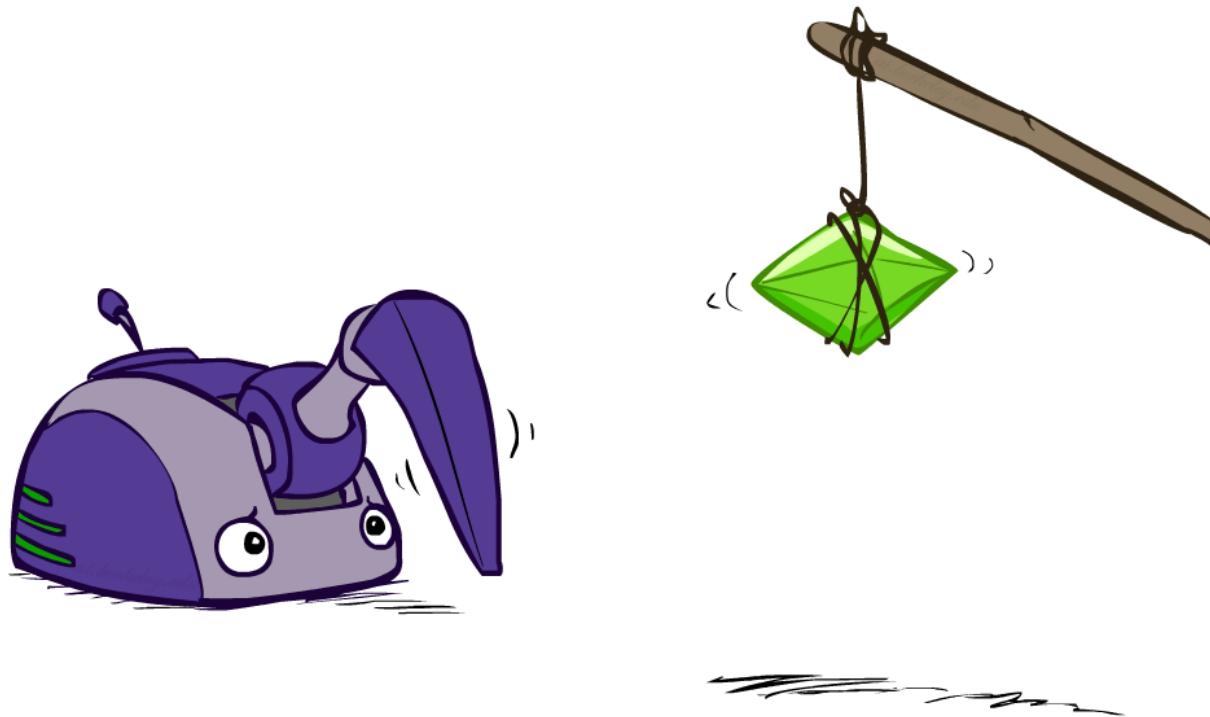
Reinforcement Learning



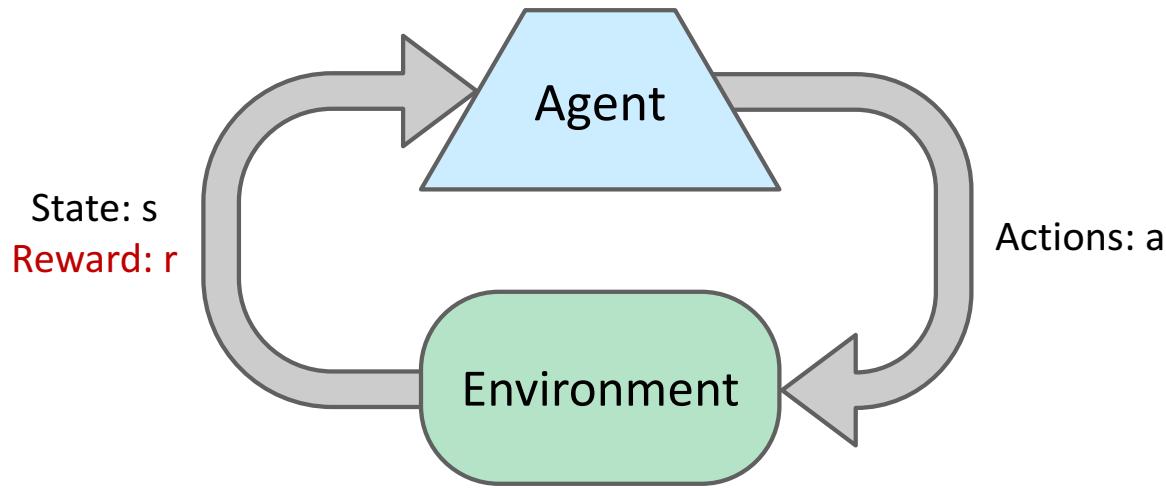
Slides courtesy of Dan Klein and Pieter Abbeel --- University of California, Berkeley

[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>.]

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]

[Kohl and Stone, ICRA 2004]

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: Sidewinding



[Andrew Ng]

[Video: SNAKE – climbStep+sidewinding]

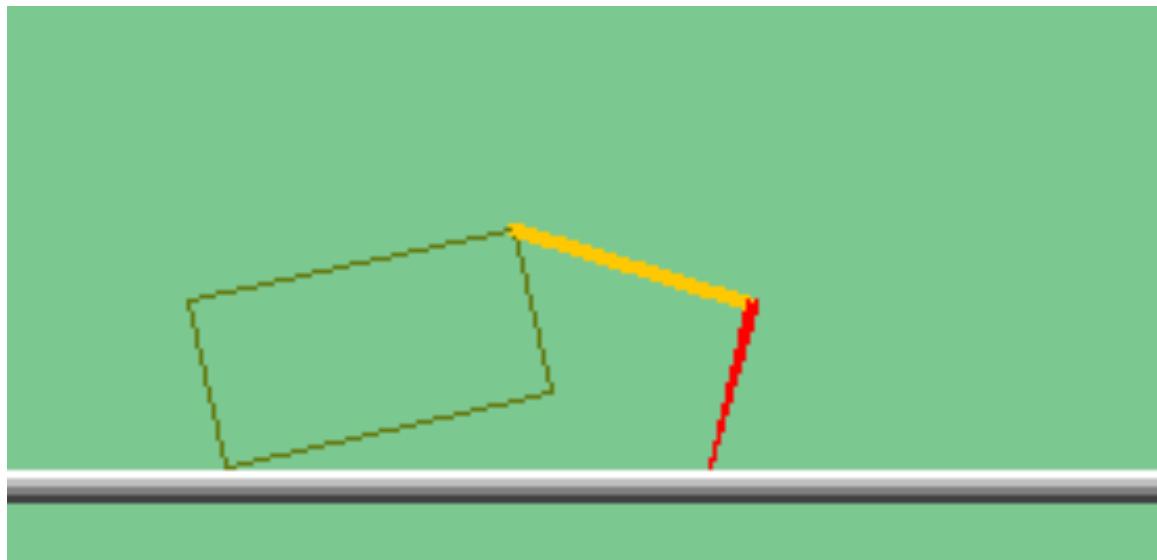
Example: Toddler Robot



[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

The Crawler!



[Demo: Crawler Bot (L10D1)]

Video of Demo Crawler Bot

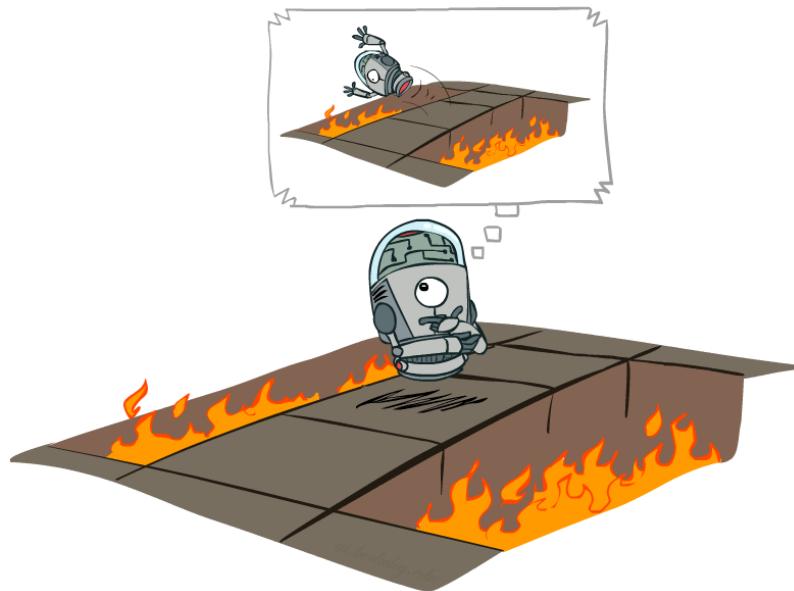


Reinforcement Learning

- 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)

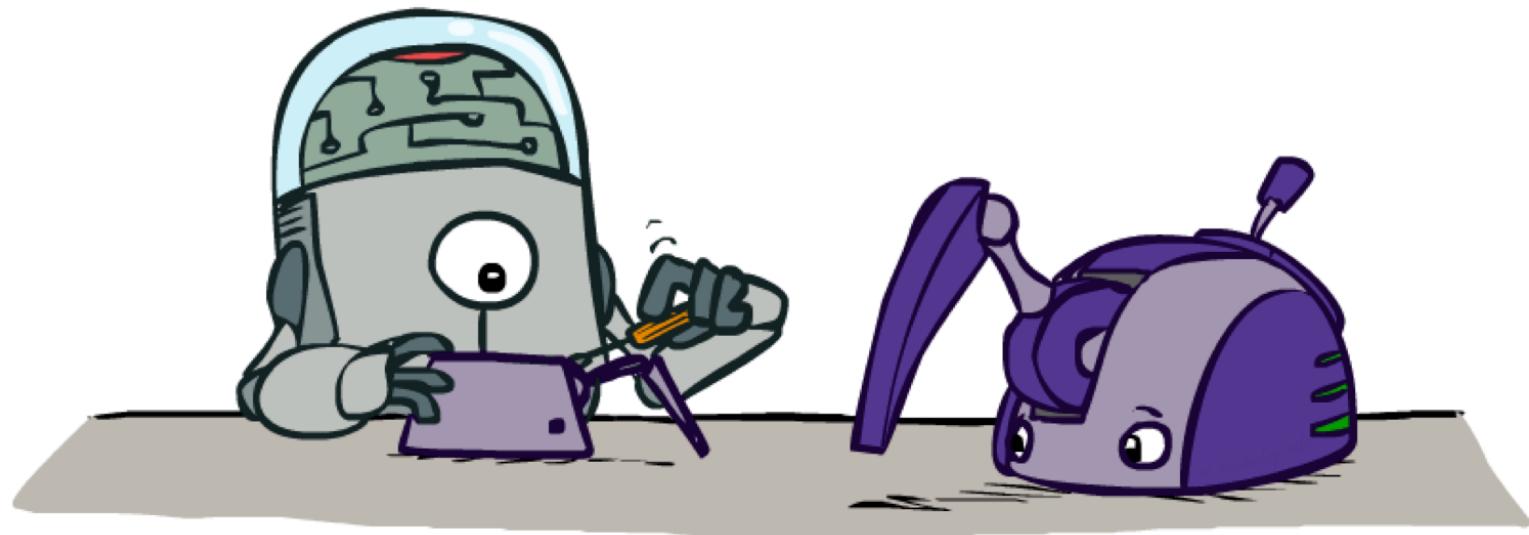


Offline Solution



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	Observed Episodes (Training)	Learned Model
	<p>Episode 1</p> <p>B, east, C, -1 C, east, D, -1 D, exit, x, +10</p> <p>Episode 2</p> <p>B, east, C, -1 C, east, D, -1 D, exit, x, +10</p> <p>Episode 3</p> <p>E, north, C, -1 C, east, D, -1 D, exit, x, +10</p> <p>Episode 4</p> <p>E, north, C, -1 C, east, A, -1 A, exit, x, -10</p>	$\hat{T}(s, a, s')$ 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 ...

Assume: $\gamma = 1$

Example: Expected Age

Goal: Compute expected age of CIS 421/521 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”

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

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

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

Unknown $P(A)$: “Model Free”

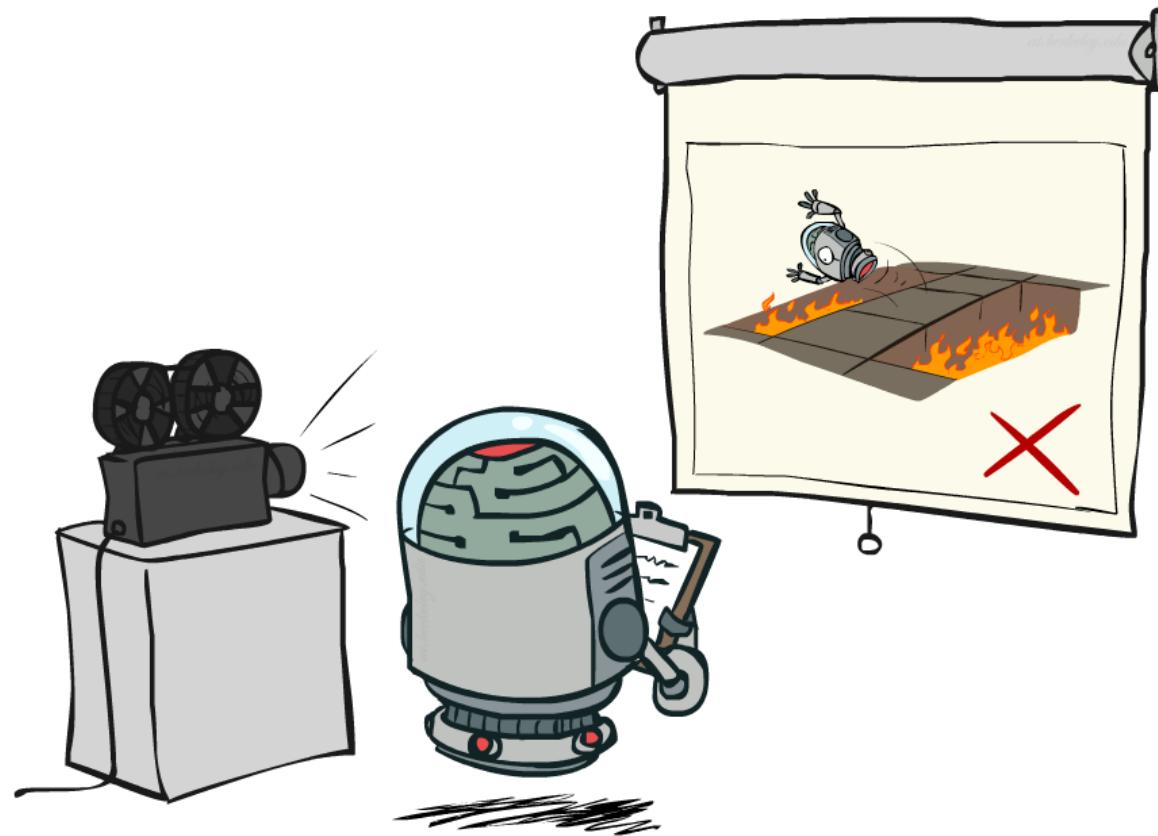
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

Model-Free Learning

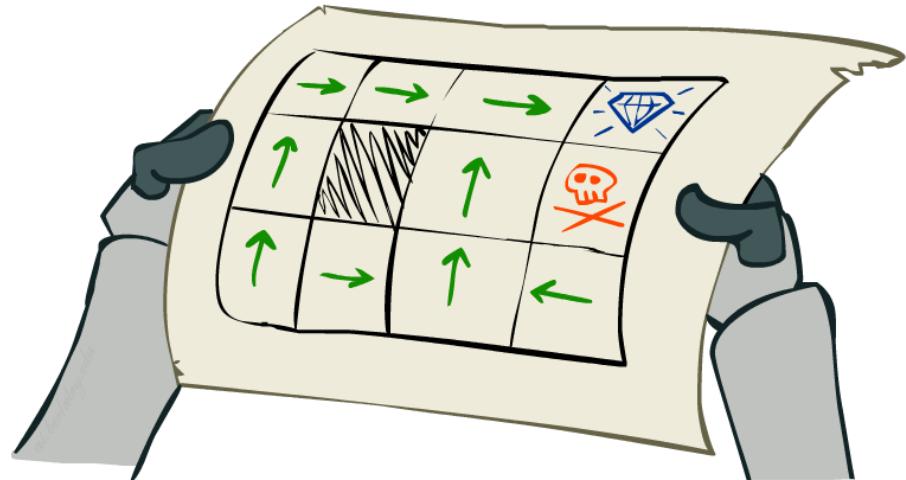


Passive Reinforcement Learning



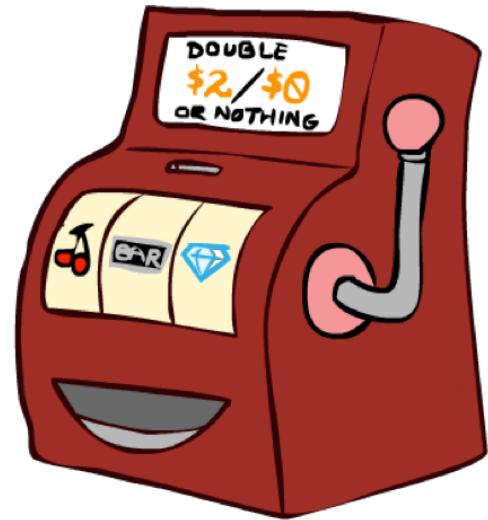
Passive Reinforcement Learning

- 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.



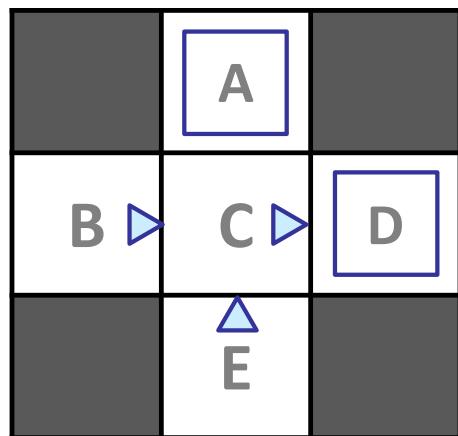
Direct Evaluation

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

Problems with Direct Evaluation

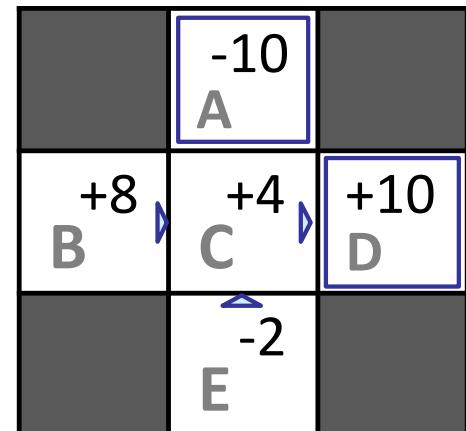
- What's good about direct evaluation?

- 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



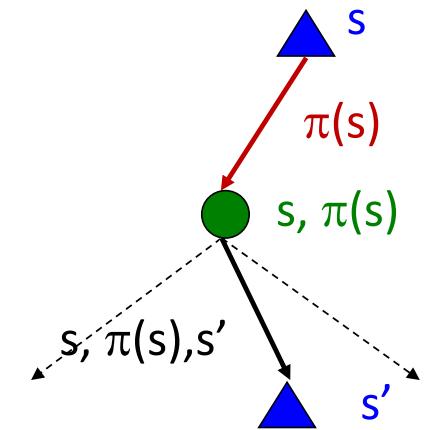
If B and E both go to C under this policy, how can their values be different?

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

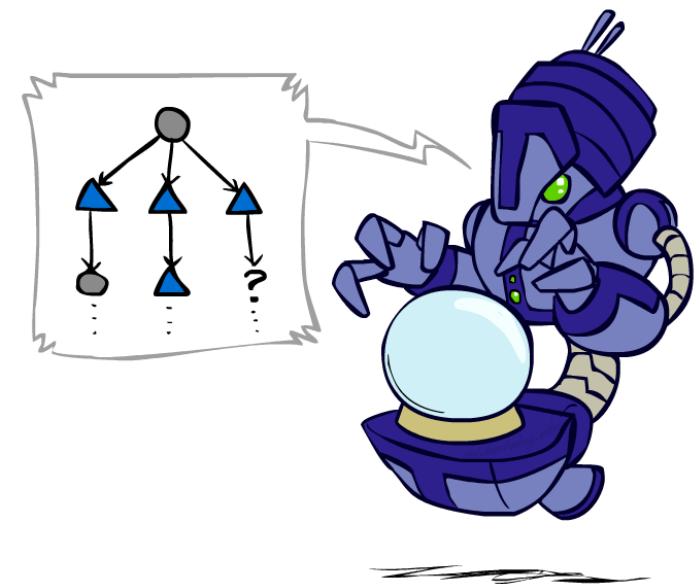
$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

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

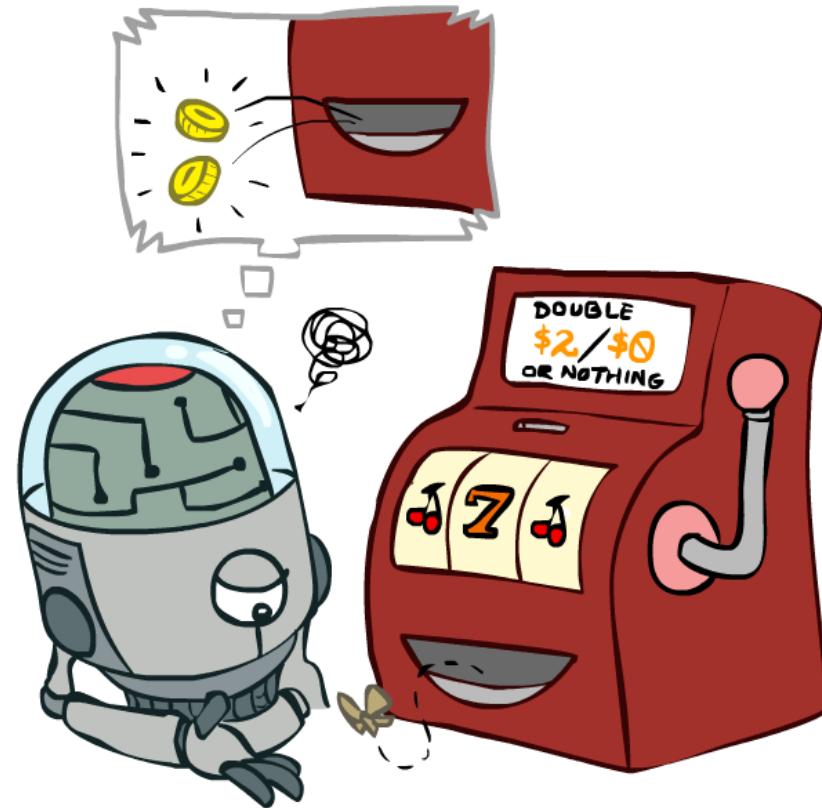
...

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

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



Temporal Difference Learning



Temporal Difference Learning

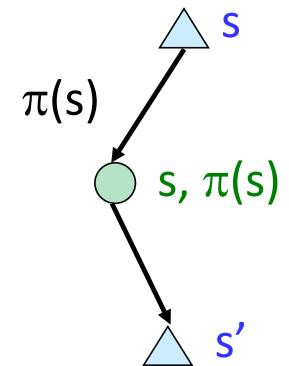
- Big idea: learn from every experience!
 - Update $V(s)$ each time we experience a transition (s, a, s', r)
 - Likely outcomes s' will contribute updates more often

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

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



Exponential Moving Average

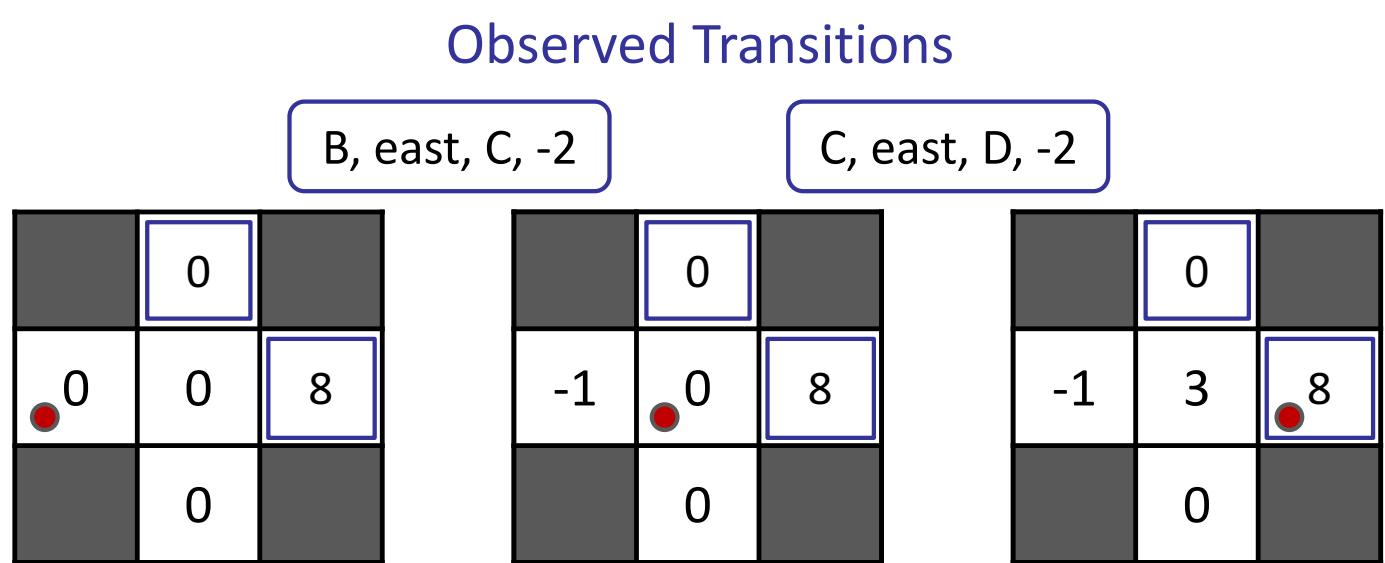
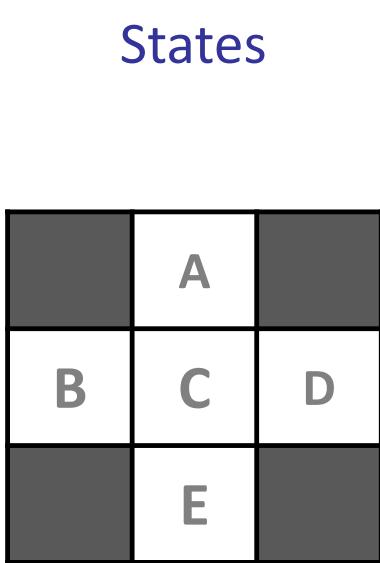
- Exponential moving average

- 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



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

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

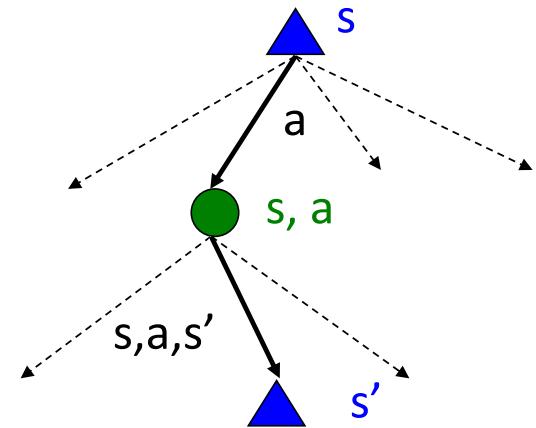
Problems with TD Value Learning

- 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:

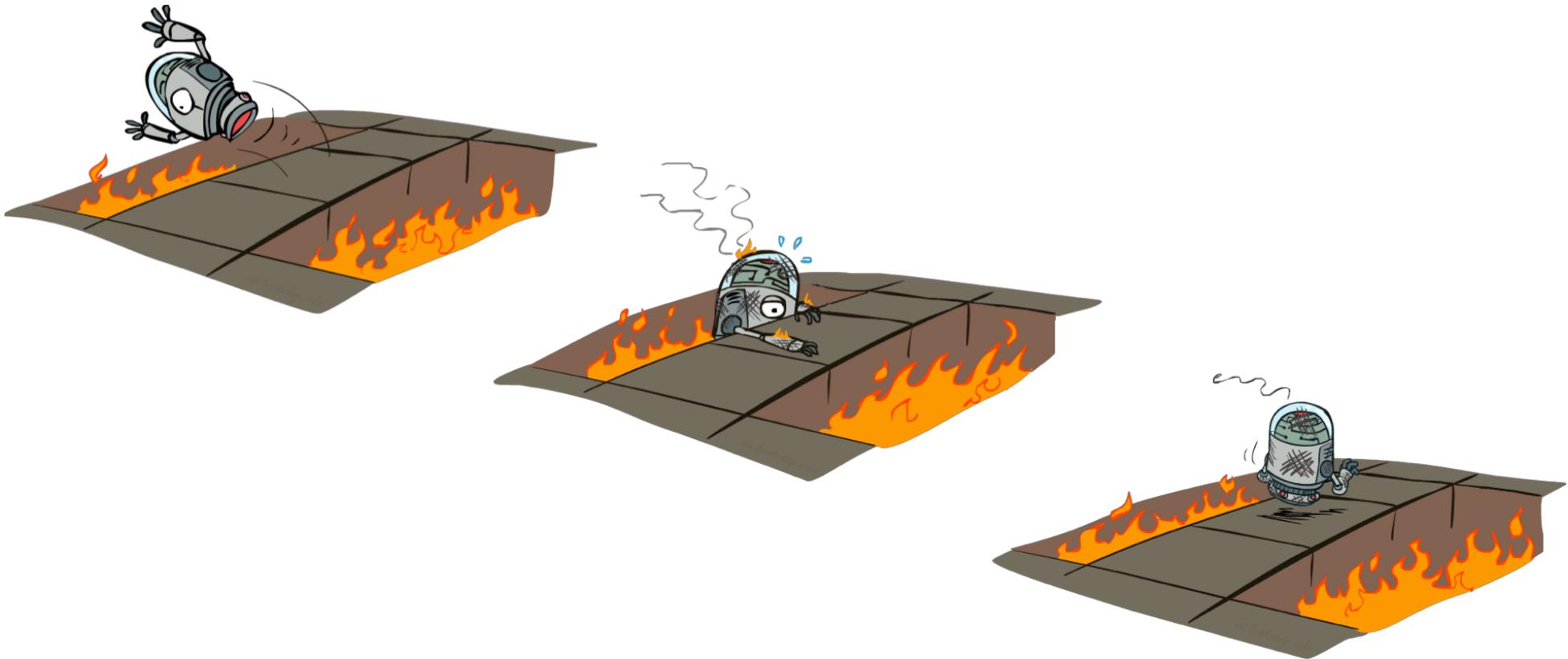
$$\pi(s) = \arg \max_a Q(s, a)$$

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

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

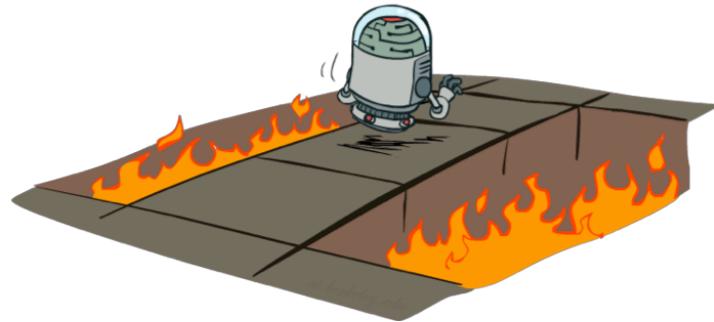


Active Reinforcement Learning



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

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

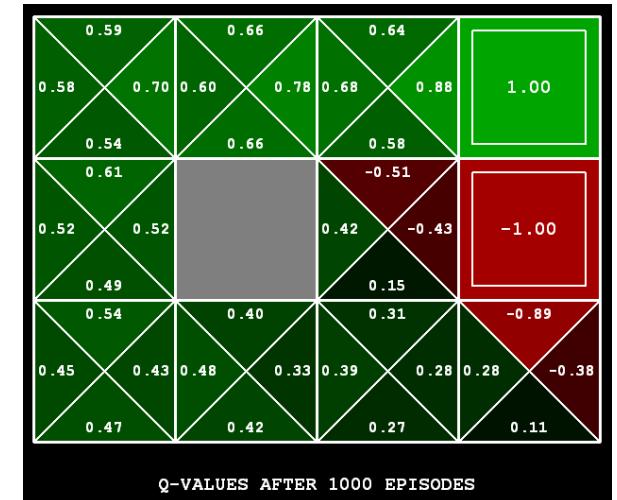
- Learn $Q(s, a)$ values as you go

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

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

- Incorporate the new estimate into a running average:

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



[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 (!)

