07-280
AI & ML I

Reinforcement Learning

Instructors:
Pat Virtue & Nihar Shah

Reinforcement learning

What if we didn't know $P(s'|s,a)$ and $R(s,a,s')$?

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

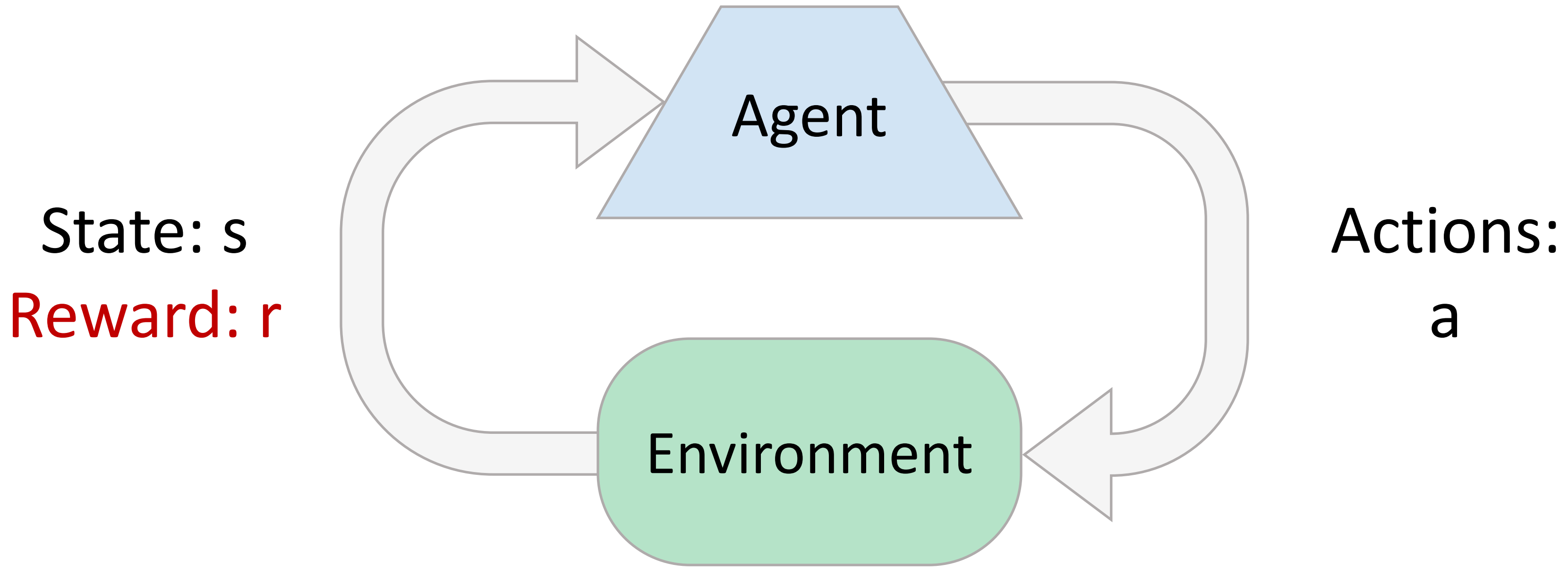
Q-iteration: $$Q_{k+1}(s,a) = \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$$

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

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

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

Reinforcement Learning



Basic idea:

- All learning is based on observed samples of rewards and next states!
- Receive feedback in the form of rewards
- Must (learn to) act so as to maximize expected sum of discounted rewards

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 $R(s,a,s')$

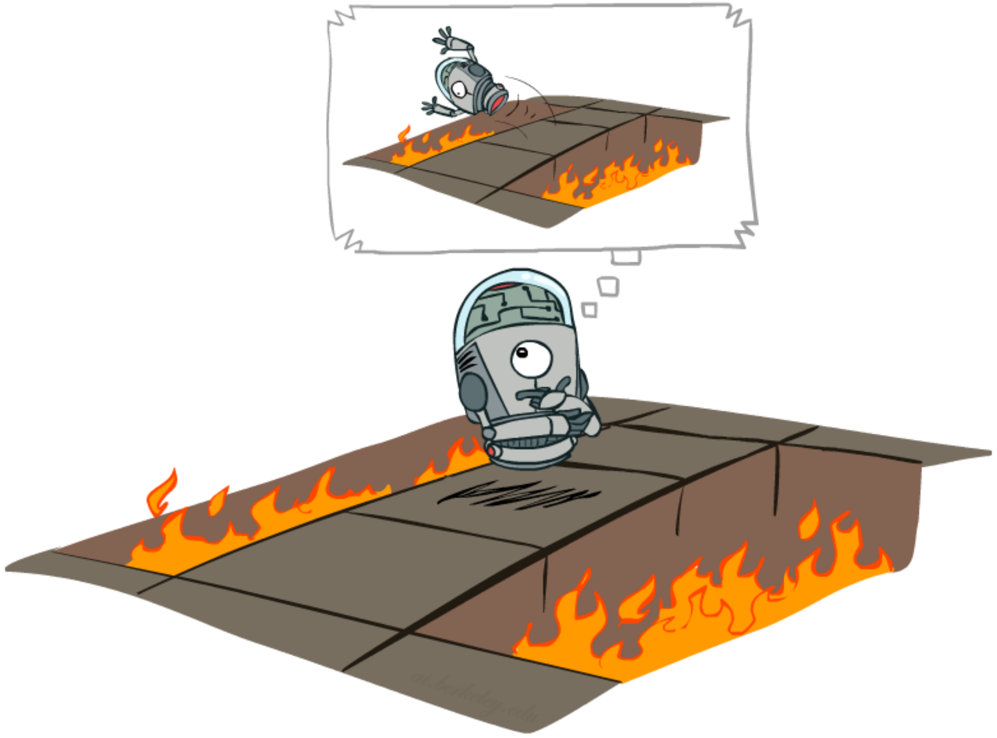
Still looking for a policy $\pi(s)$

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

Slide credit: UC Berkeley AI

Offline (MDPs) vs. Online (RL)



Offline Solution
(Known MDP)

Online Learning
(Unknown MDP)

Slide credit: UC Berkeley AI

Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

Model-Based

Estimate MDP T(s,a,s') and R(s,a,s') from samples of environment

Model-Free

Passive Reinforcement Learning

- Monte Carlo Policy Evaluation
- TD(0) Learning

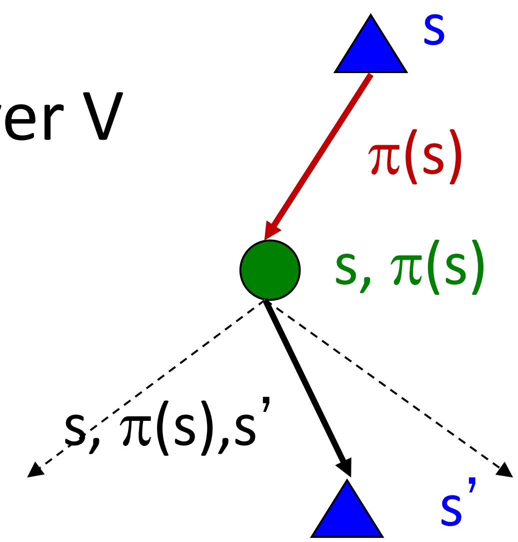
Active Reinforcement Learning

- Q-Learning

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?

Slide credit: UC Berkeley AI

Online Learning

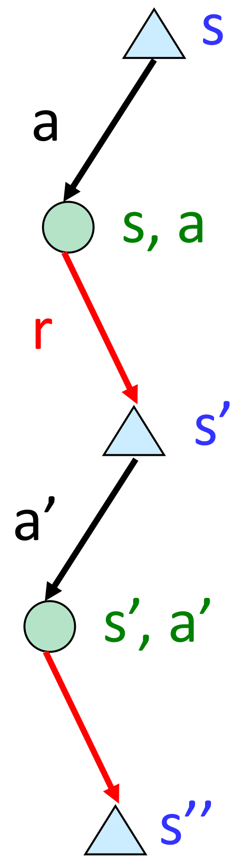
Passive Reinforcement Learning

Monte Carlo Policy Evaluation

Monte Carlo (Sample-Based) Learning

Collect a bunch of trajectories from episodes following π starting at a given state, s.

- Option 1: Collect N trajectories then evaluate π



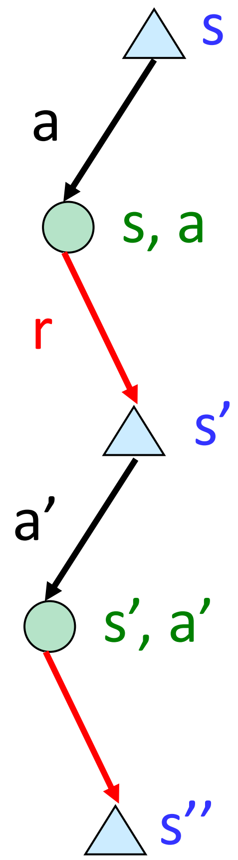
Monte Carlo (Sample-Based) Learning

Collect a bunch of trajectories from episodes following π starting at a given state, s.

- Option 1: Collect N trajectories then evaluate π
- Option 2: Update $V^{\pi}(s)$ after each episode

Sample of V(s):

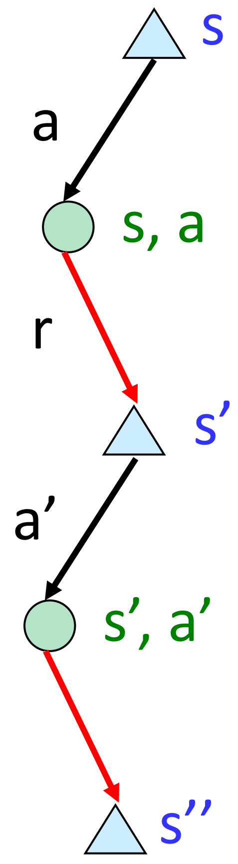
Update to V(s):



Monte Carlo (Sample-Based) Learning

Collect a bunch of trajectories from episodes following π starting at a given state, s.

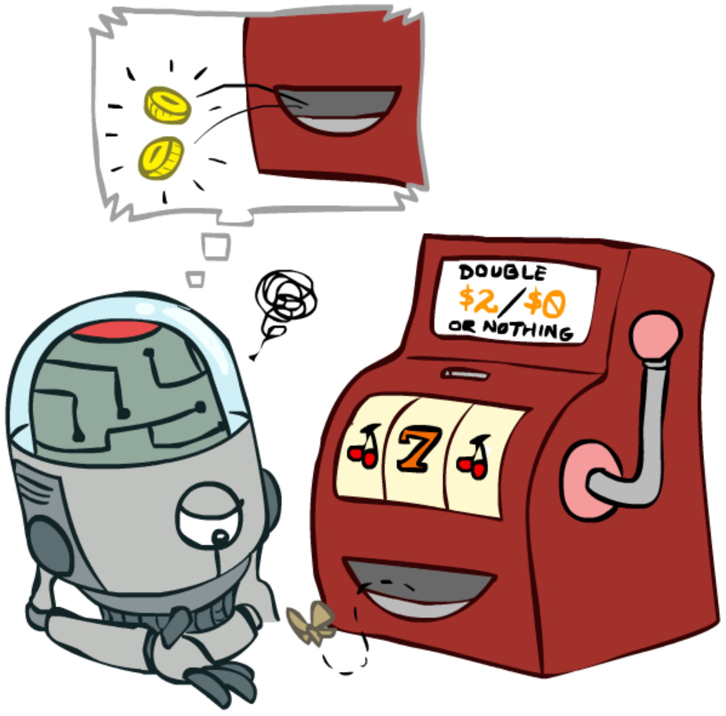
- Option 1: Collect N trajectories then evaluate π
- Option 2: Update $V^{\pi}(s)$ after each episode



Sample of V(s):

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

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



Online Learning

Passive Reinforcement Learning

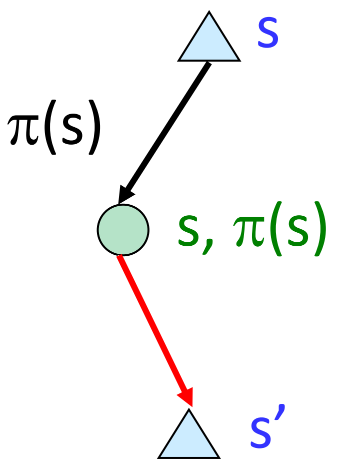
Temporal Difference Learning

Image credit: *Ketrina Yim, UC Berkeley*

Temporal Difference Learning

Big idea: learn from every step!

- Update V(s) each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often
- Bootstrap: Leverage previous estimates of next state, $V_k^{\pi}(s')$ (rather than full trajectory)
- Policy still fixed, still doing evaluation!



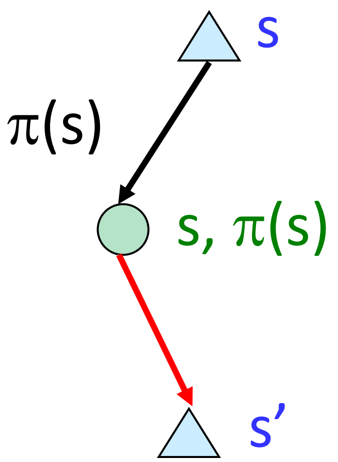
Sample of V(s): $sample = r + \gamma V_k^{\pi}(s')$

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
- Bootstrap: Leverage previous estimates of next state, $V_k^{\pi}(s')$ (rather than full trajectory)
- Policy still fixed, still doing evaluation!



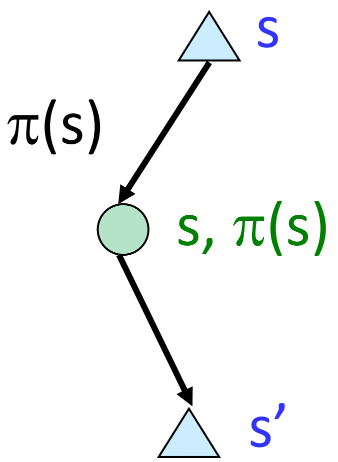
Sample of V(s): $sample = r + \gamma V_k^{\pi}(s')$

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

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
- Bootstrap: Leverage previous estimates of next state, $V_k^{\pi}(s')$ (rather than full trajectory)
- Policy still fixed, still doing evaluation!



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

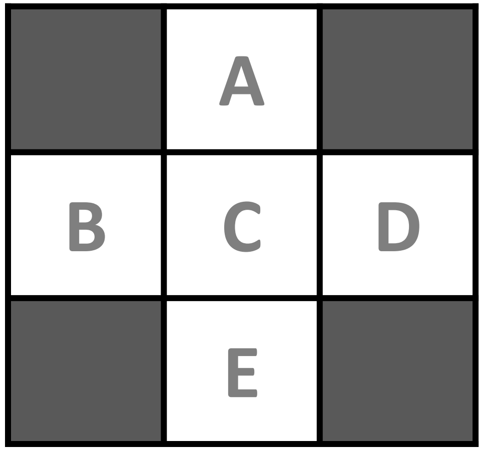
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

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

Slide credit: UC Berkeley AI

Example: Temporal Difference Learning

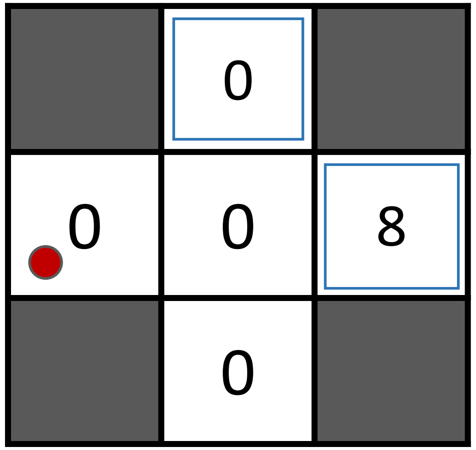
States

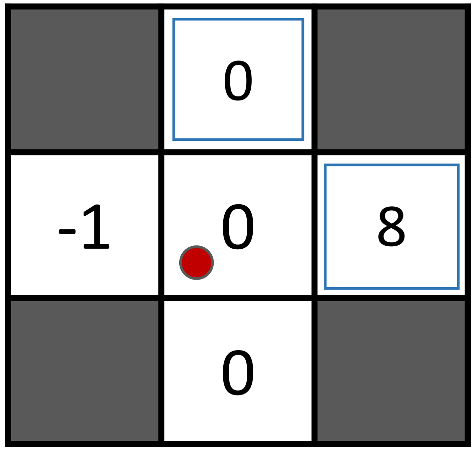


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, C, -2





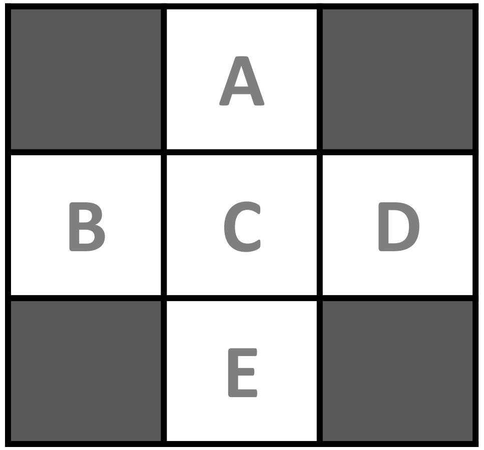
$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

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

Slide credit: UC Berkeley AI

Example: Temporal Difference Learning

States

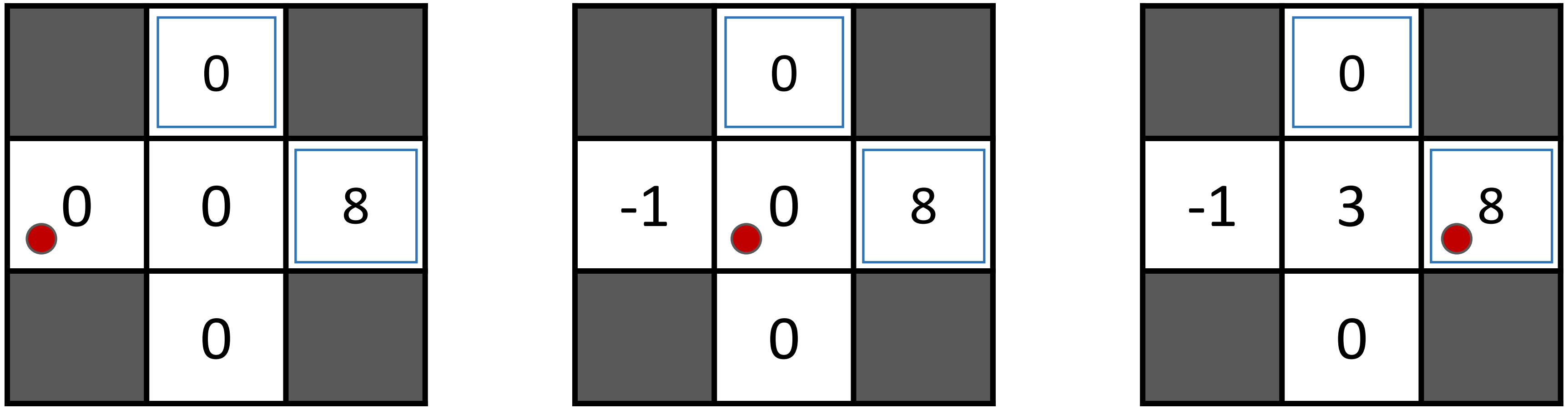


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, C, -2

C, east, D, -2



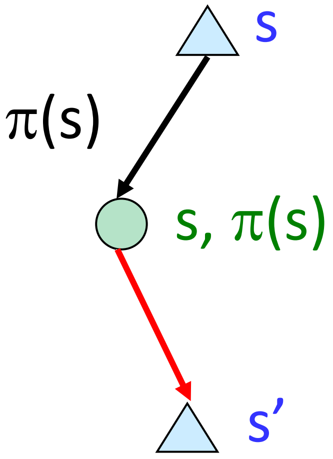
$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

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

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): $sample = r + \gamma V^{\pi}(s')$

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

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\,[sample - V^{\pi}(s)]$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) - \alpha \nabla Error$ $\quad Error = \frac{1}{2}\left(sample - V^{\pi}(s)\right)^2$

ML detour: Quick Calculus Quiz

$$f(x) = \frac{1}{2}(y - x)^2$$

What is $\frac{df}{dx}$?

ML detour: Gradient Descent

$$f(x) = \frac{1}{2}(y - x)^2$$

Goal: find x that minimizes $f(x)$

$$\frac{df}{dx} = -(y - x)$$

1. Start with initial guess, x_0
2. Update x by taking a step in the direction that $f(x)$ is changing fastest (in the negative direction) with respect to x:

   $x \leftarrow x - \alpha \nabla_x f$, where α is the step size or learning rate
3. Repeat until convergence

TD goal: find value(s), V, that minimizes difference between sample(s) and V

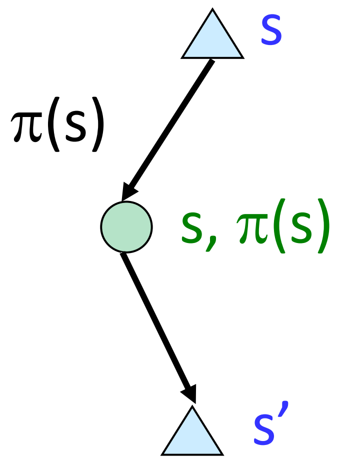
$$V \leftarrow V - \alpha \nabla_V Error$$

$$Error(V) = \frac{1}{2}(sample - V)^2$$

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): $sample = r + \gamma V^{\pi}(s')$

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

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\, [sample - V^{\pi}(s)]$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) - \alpha \nabla Error$ $\quad Error = \frac{1}{2}\left(sample - V^{\pi}(s)\right)^2$

Poll 1

TD update: $V^{\pi}(s) = V^{\pi}(s) + \alpha\,[r + \gamma\, V^{\pi}(s') - V^{\pi}(s)]$

Which converts TD values into a policy?

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

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) None of the above

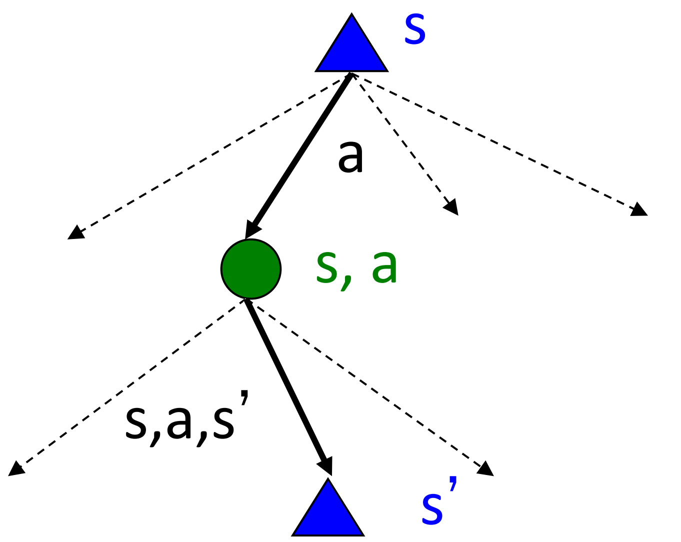
Problems with TD Value Learning

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!

MDP/RL 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: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$

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

Value (TD) learning: $V^{\pi}(s) = V^{\pi}(s) + \alpha\, [r + \gamma\, V^{\pi}(s') - V^{\pi}(s)]$

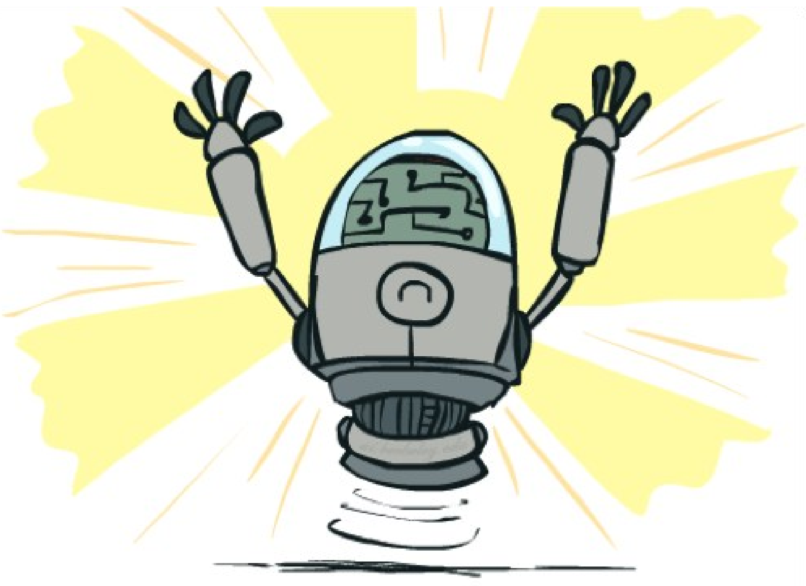
Q-learning: $Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

RL Summary

Name	Monte Carlo	TD(0)	Q-learning	Approx. Q-learning
Model	$V(s)$ table	$V(s)$ table	$Q(s,a)$ table	θ for $Q_\theta(s,a)$
Action selection	$a = \pi(s)$	$a = \pi(s)$	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)
Sampling technique	MC	Bootstrap	Bootstrap	Bootstrap
Loss	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$
Prediction, $\hat{y}$	$V^\pi(s)$	$V^\pi(s)$	$Q(s,a)$	$Q_\theta(s,a)$
Sample, y				

RL Summary

Name	Monte Carlo	TD(0)	Q-learning	Approx. Q-learning
Model	$V(s)$ table	$V(s)$ table	$Q(s,a)$ table	θ for $Q_\theta(s,a)$
Action selection	$a = \pi(s)$	$a = \pi(s)$	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)
Sampling technique	MC	Bootstrap	Bootstrap	Bootstrap
Loss	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$	$\frac{1}{2}(y-\hat{y})^2$
Prediction, $\hat{y}$	$V^\pi(s)$	$V^\pi(s)$	$Q(s,a)$	$Q_\theta(s,a)$
Sample, y	$r +$ $\gamma r' +$ $\gamma^2 r'' +$ $\gamma^3 r''' +$ $\gamma^4 r''' +$ …	$r +$ $\gamma\, V^\pi(s')$	$r +$ $\gamma \max_{a'} Q(s',a')$	$r +$ $\gamma \max_{a'} Q_\theta(s',a')$



Online Learning
Model-free Learning
Active Reinforcement Learning
Q-Learning

Image credit: *Ketrina Yim, UC Berkeley*

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')
- The sample value is then:

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

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

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

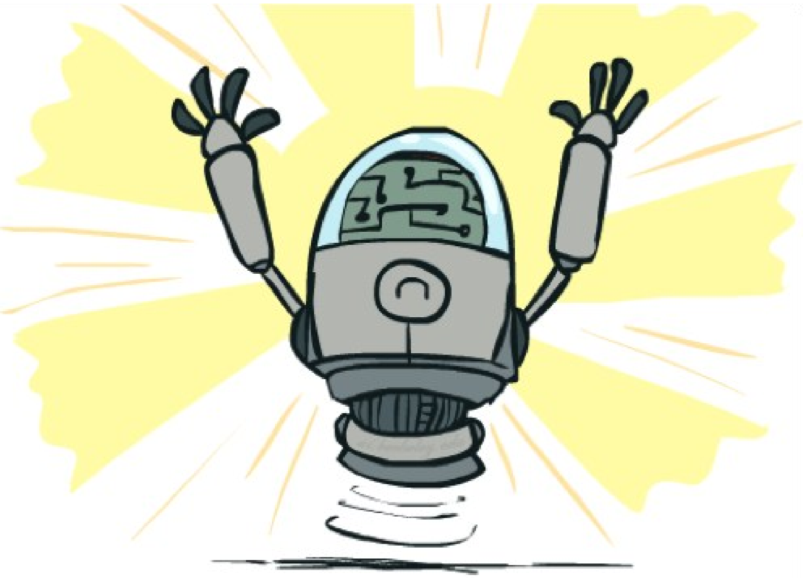
Q-Learning Properties

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 (!)



Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

Model-Based

Estimate MDP T(s,a,s') and R(s,a,s') from samples of environment

Model-Free

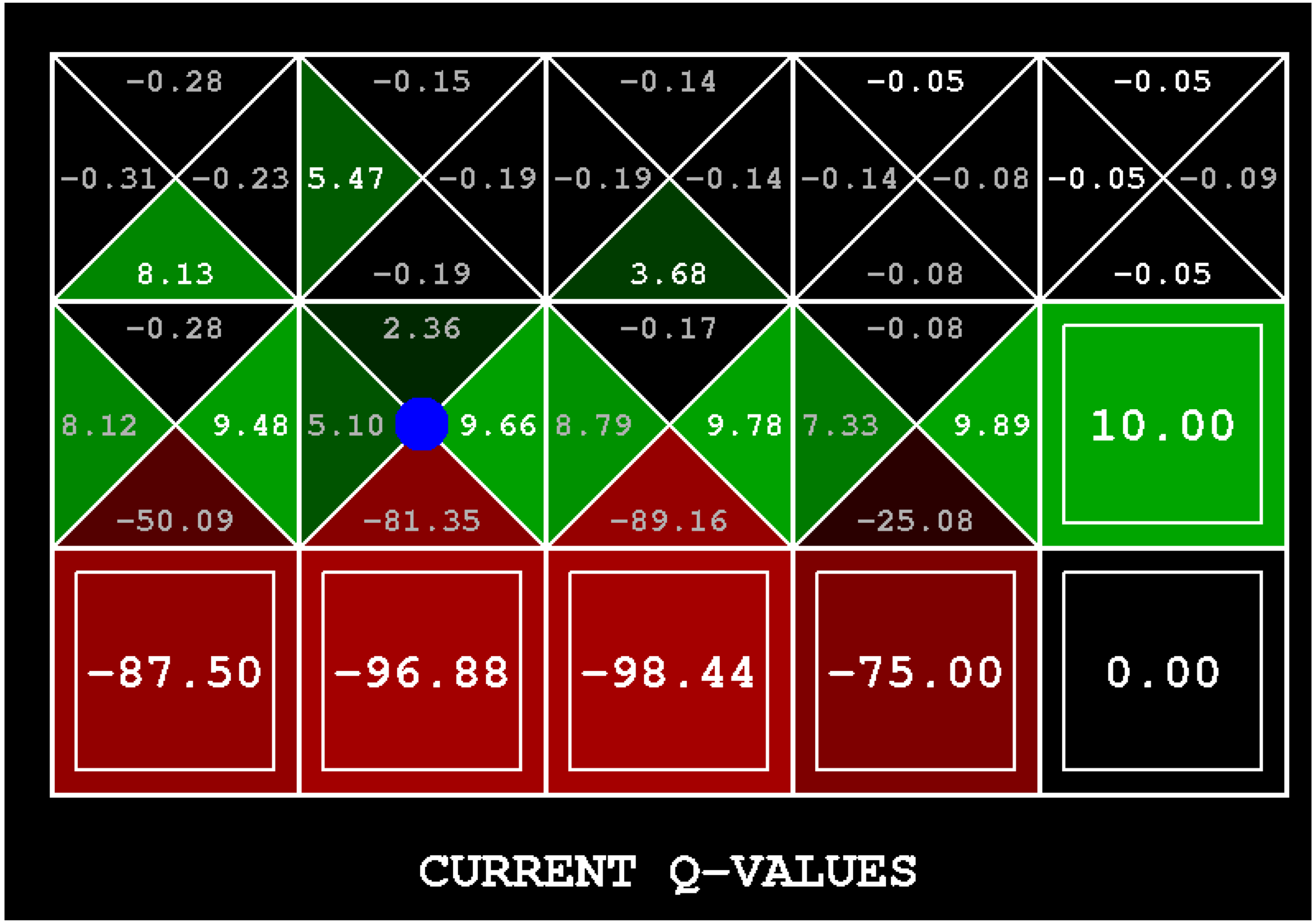
Passive Reinforcement Learning

- Monte Carlo Policy Evaluation
- TD Learning

Active Reinforcement Learning

- Q-Learning

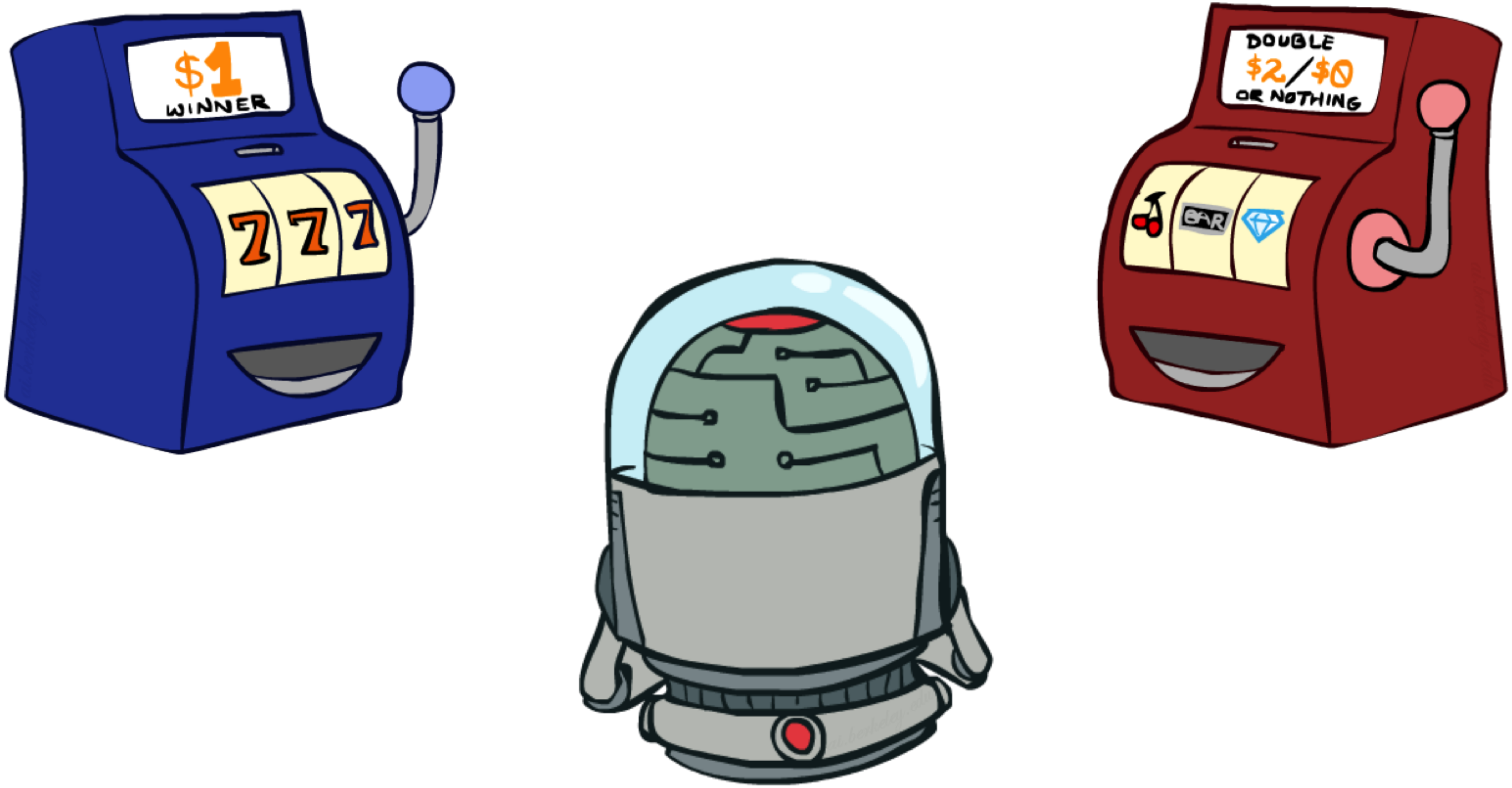
Demo Q-Learning Auto Cliff Grid



Slide credit: UC Berkeley AI

[Demo: Q-learning – auto – cliff grid (L11D1)]

Double Bandits

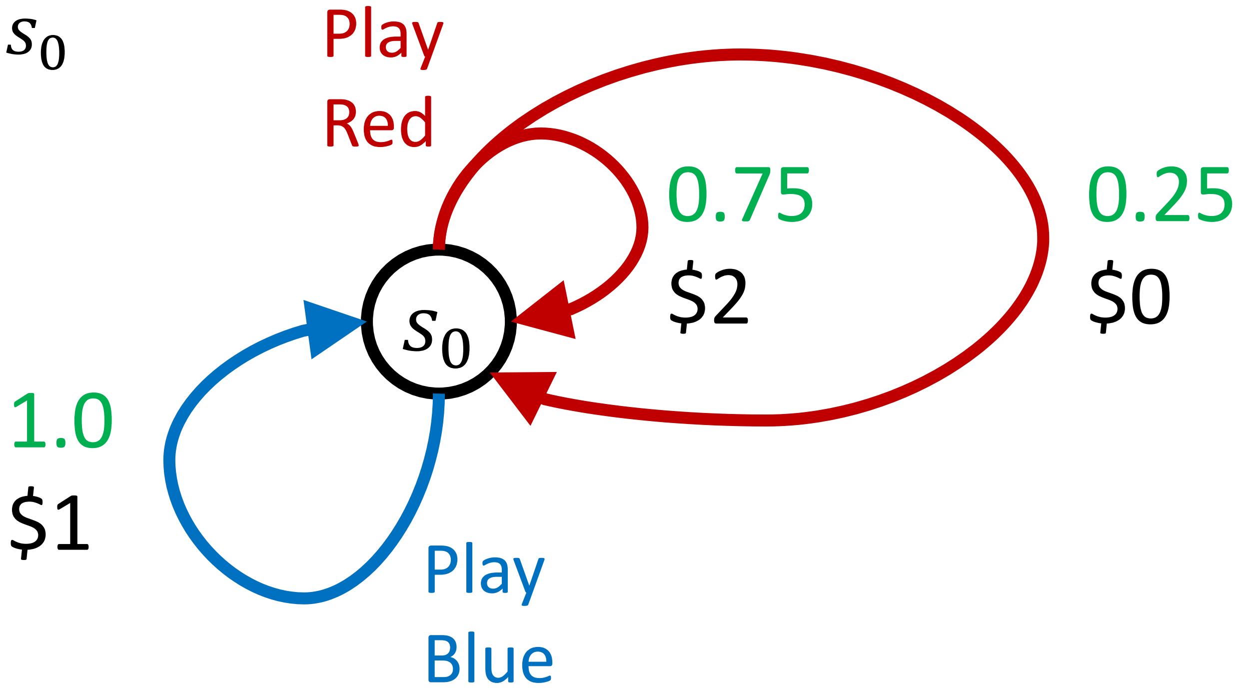


Slide credit: UC Berkeley AI

Double-Bandit MDP

Actions: *Blue*, *Red*

States: (only one) s_0



Slide credit: UC Berkeley AI

Offline Planning

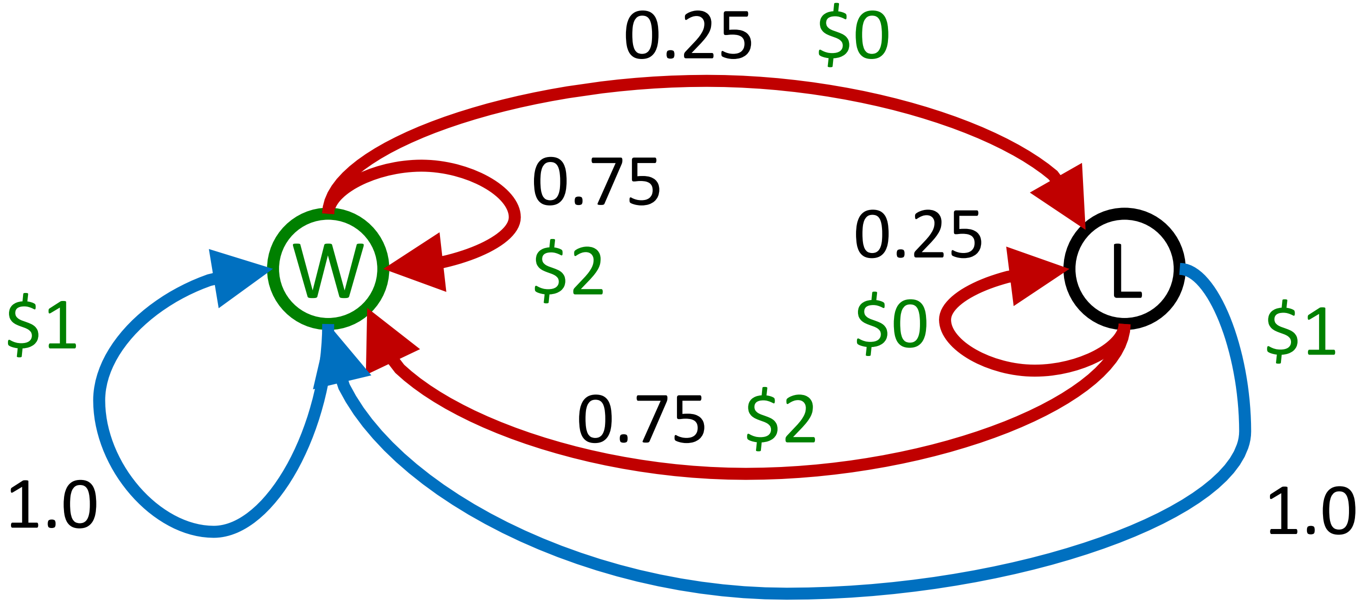
No discount

100 time steps

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!

		Q Value
a	Play Red	150
a	Play Blue	100



Slide credit: UC Berkeley AI

Let’s Play!

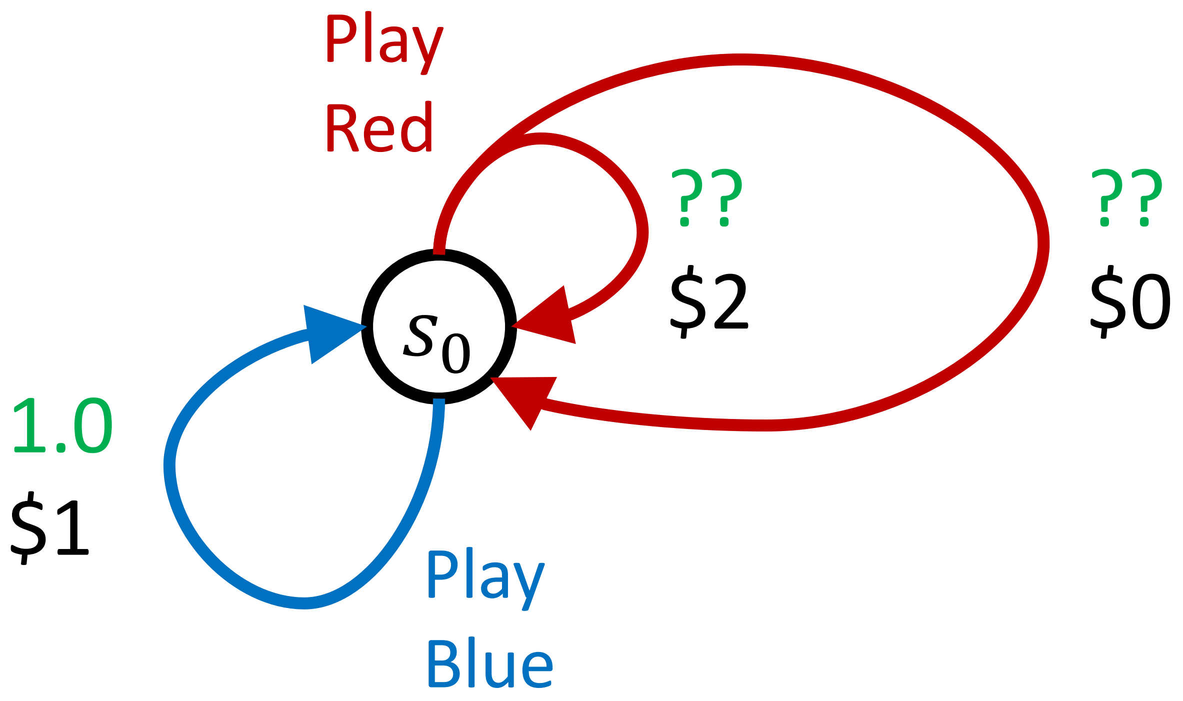
$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Slide credit: UC Berkeley AI

Online Planning

Rules changed! Red's win chance is different.



Slide credit: UC Berkeley AI

Let's Play!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

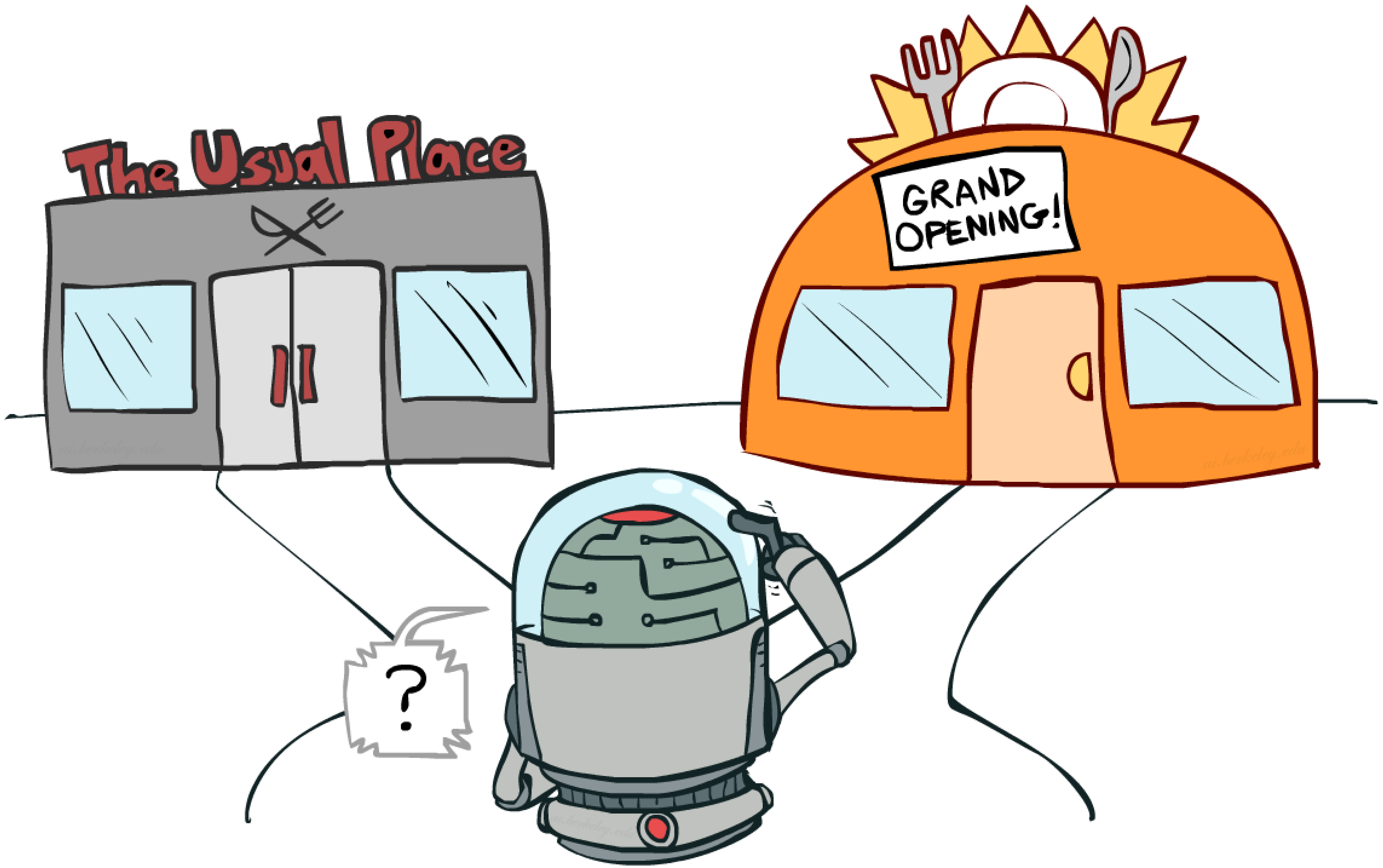
That wasn't planning, it was learning!

- 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

Exploration vs. Exploitation



Slide credit: UC Berkeley AI

How to Explore?

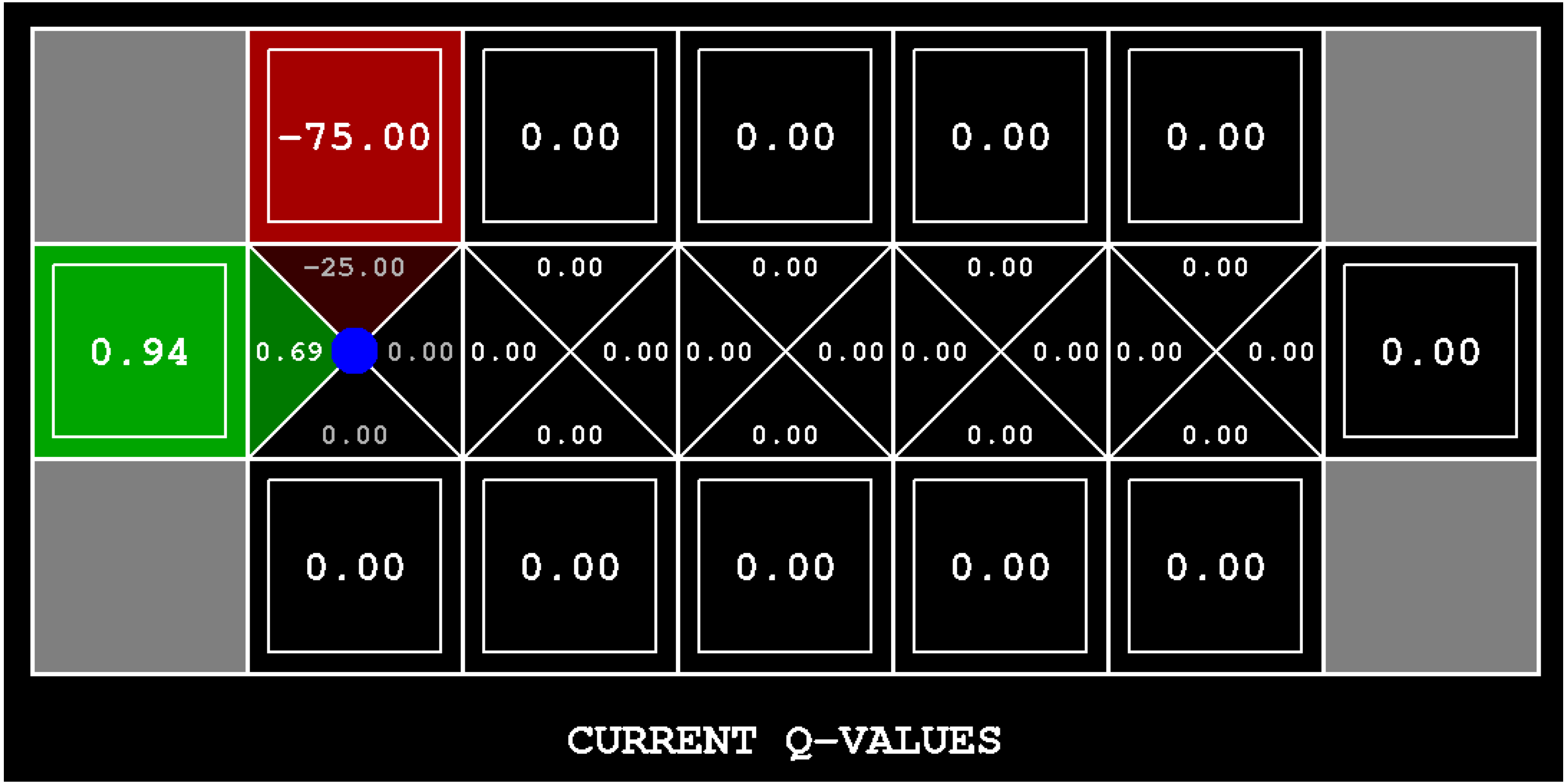
Several schemes for forcing exploration

- 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

- 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

[Demo: Q-learning – manual exploration – bridge grid (L11D2)]
[Demo: Q-learning – epsilon-greedy -- crawler (L11D3)]

Demo Q-learning – Manual Exploration – Bridge Grid



Slide credit: UC Berkeley AI

Exploration Functions

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

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

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

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

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

- Note: this propagates the “bonus” back to states that lead to unknown states as well!

Slide credit: UC Berkeley AI

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Exploration Functions $k=100$

$$\frac{Q}{10} \quad \frac{n}{1} \qquad \frac{Q}{100} \quad \frac{n}{100}$$

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

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

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

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

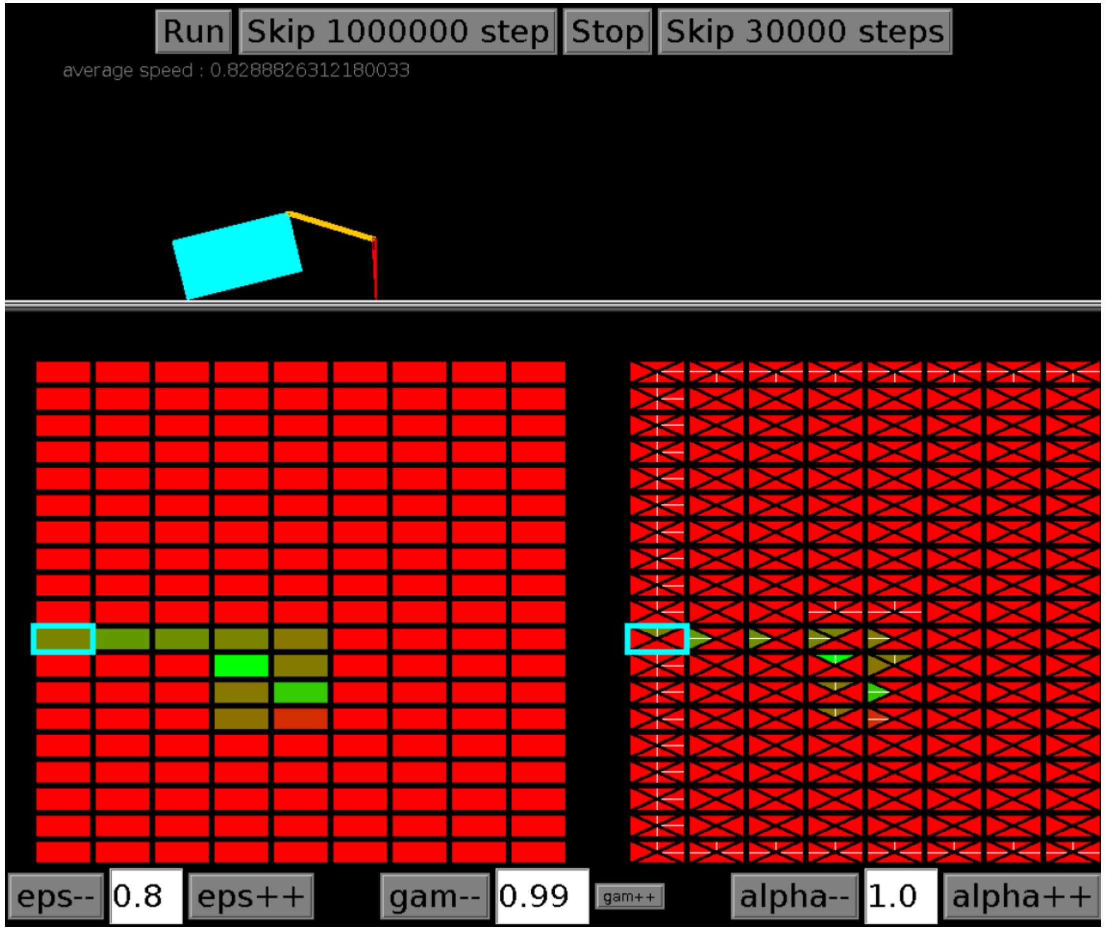
Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

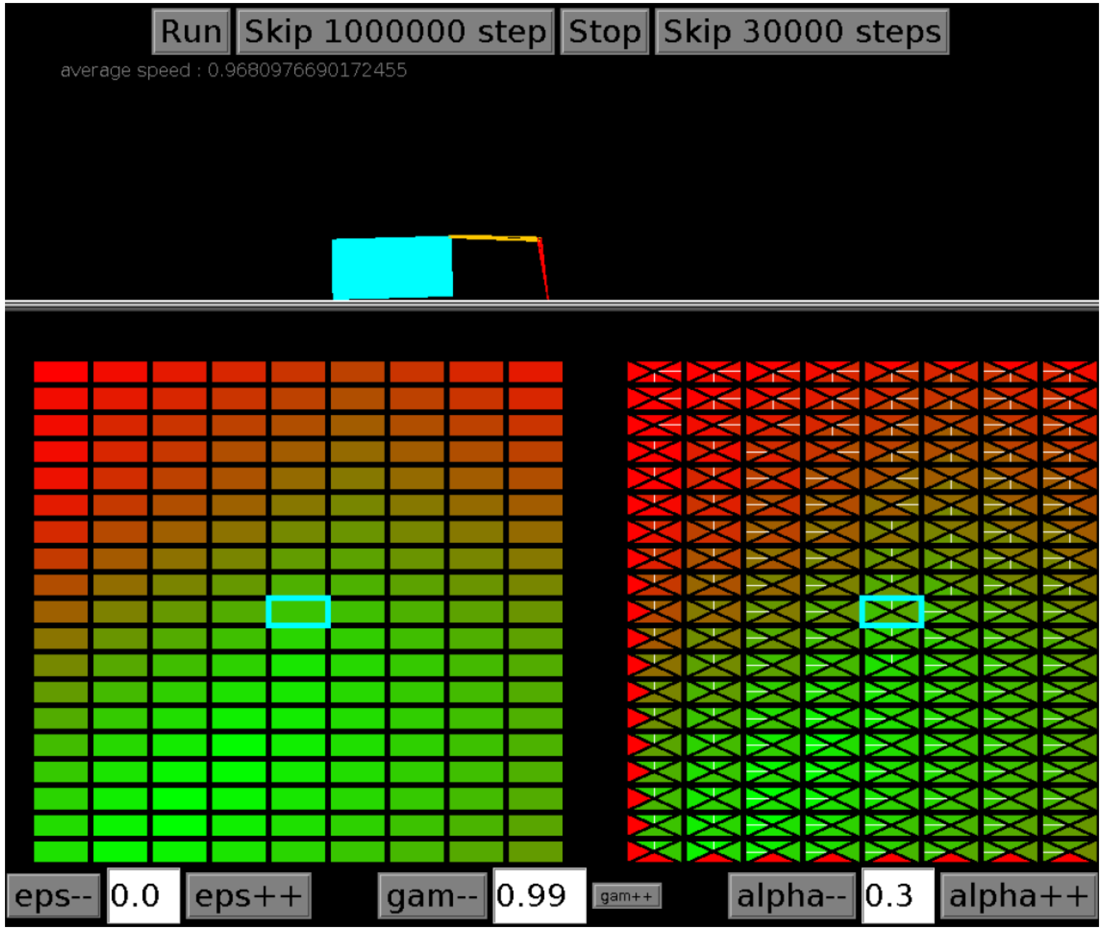
- Note: this propagates the "bonus" back to states that lead to unknown states as well!

Slide credit: UC Berkeley AI

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Demo Q-learning – Epsilon-Greedy – Crawler





Slide credit: UC Berkeley AI

Exploration Functions

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

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

$$f(u,n) = u + k/(n+1)$$

Regular Q-Update: $Q(s,a) = Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') \; - Q(s,a)]$

Modified Q-Update: $Q(s,a) = Q(s,a) + \alpha\,[r + \gamma \max_{a'} f(Q(s',a'), N(s',a')) \; - Q(s,a)]$

- Note: this propagates the "bonus" back to states that lead to unknown states as well!

Slide credit: UC Berkeley AI

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

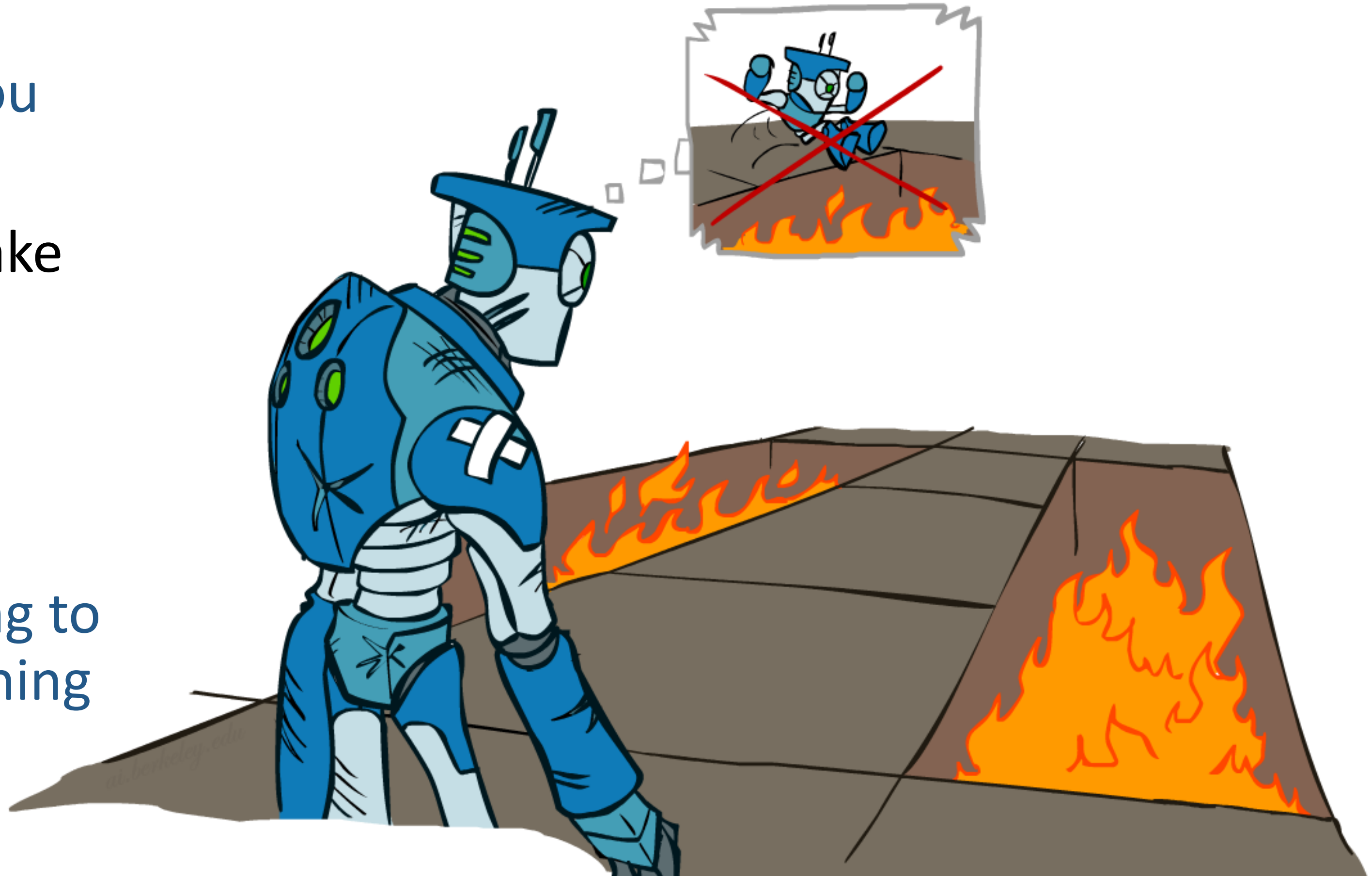
Regret

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



RL Summary

Name	Monte Carlo	TD(0)	Q-learning	Approx. Q-learning
Model	$V(s)$ table	$V(s)$ table	$Q(s,a)$ table	θ for $Q_\theta(s,a)$
Action selection	$a = \pi(s)$	$a = \pi(s)$	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)	$a' = \text{argmax}\, Q(s',a')$ (and/or exploration)
Sampling technique	MC	Bootstrap	Bootstrap	Bootstrap
Loss	$\frac{1}{2}(\hat{y} - y)^2$	$\frac{1}{2}(\hat{y} - y)^2$	$\frac{1}{2}(\hat{y} - y)^2$	$\frac{1}{2}(\hat{y} - y)^2$
Prediction, $\hat{y}$	$V^\pi(s)$	$V^\pi(s)$	$Q(s,a)$	$Q_\theta(s,a)$
Sample, y	$r +$ $\gamma r' +$ $\gamma^2 r'' +$ $\gamma^3 r''' +$ $\gamma^4 r''' +$ ...	$r +$ $\gamma\, V^\pi(s')$	$r +$ $\gamma \max_{a'} Q(s',a')$	$r +$ $\gamma \max_{a'} Q_\theta(s',a')$