Announcements

Assignments

- HW8: Out today, due Thu, 12/3, 11:59 pm

Schedule next week

- Monday: Recitation in both lecture slots
- No lecture Wednesday
- No recitation Friday

Final exam scheduled

Introduction to Machine Learning

Reinforcement Learning

Instructor: Pat Virtue

Plan

Last time

- Rewards and Discounting
- Finding optimal policies: Value iteration and Bellman equations

Today

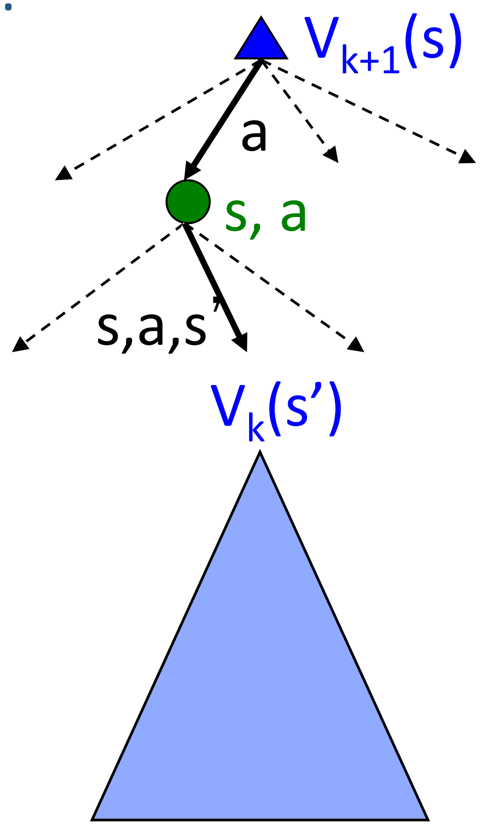
- MDP: How to use optimal values
- Reinforcement learning
  - Models are gone!
  - Rebuilding models
  - Sampling and TD learning
  - Q-learning
  - Approximate Q-learning

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:

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



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

Optimal Quantities

- The value (utility) of a state s:

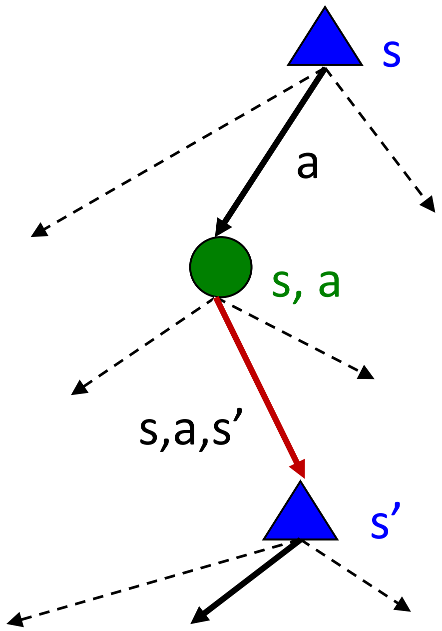
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

[Demo – gridworld values (L8D4)]

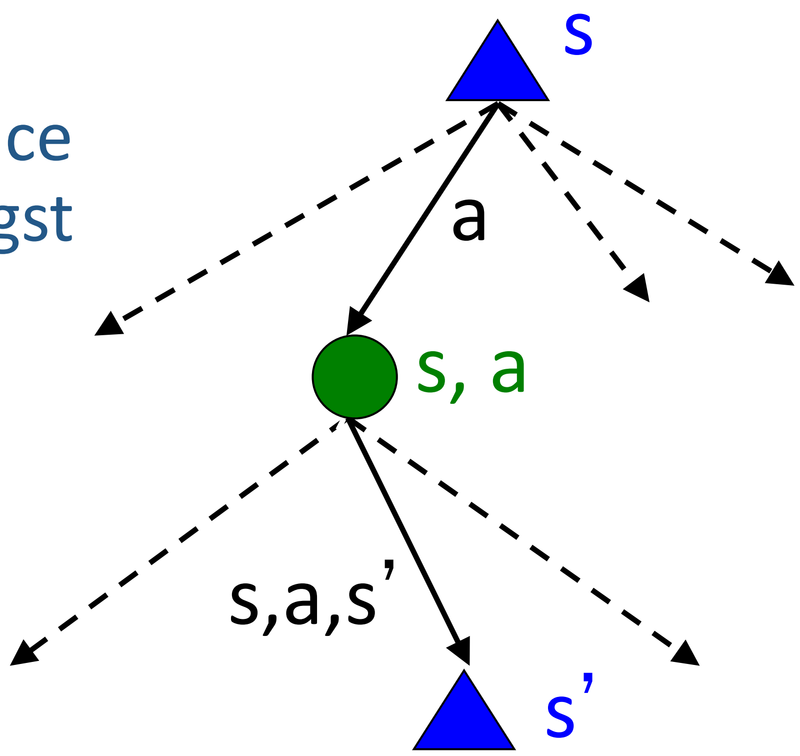
The Bellman Equations

Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

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



These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

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

MDP Notation

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

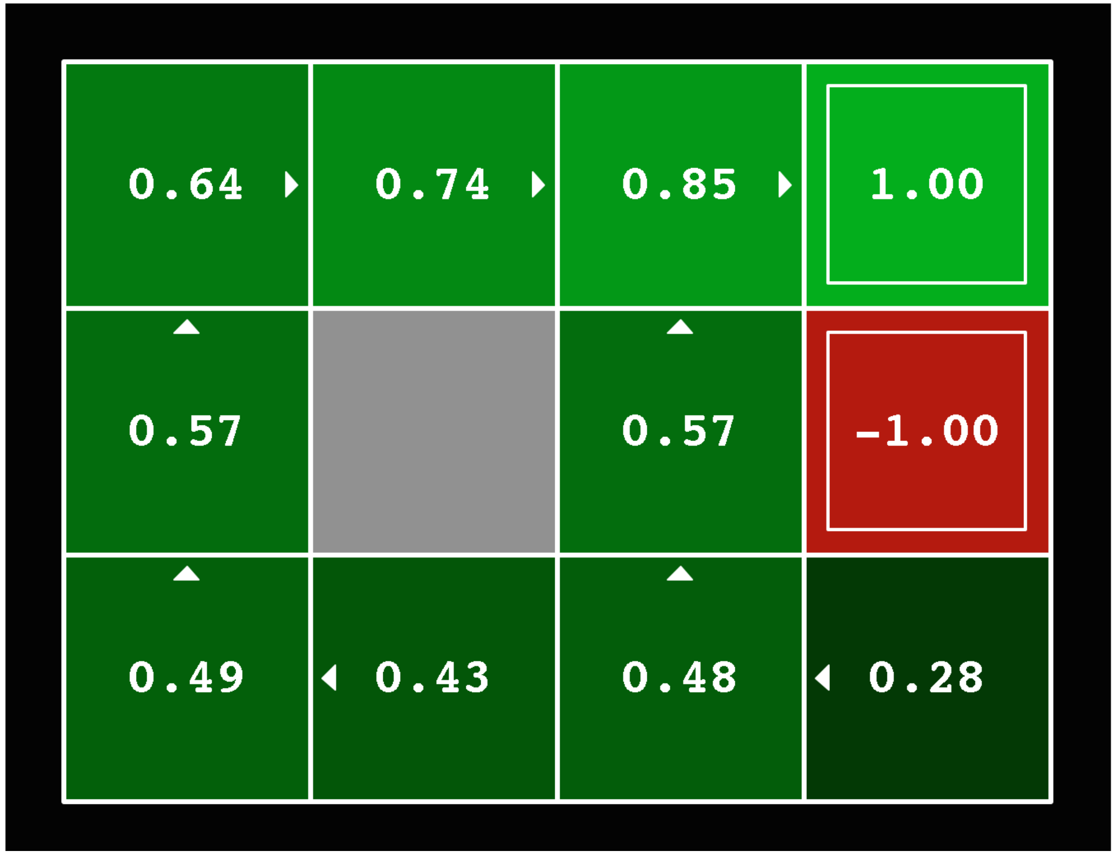
Bellman equations: $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

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

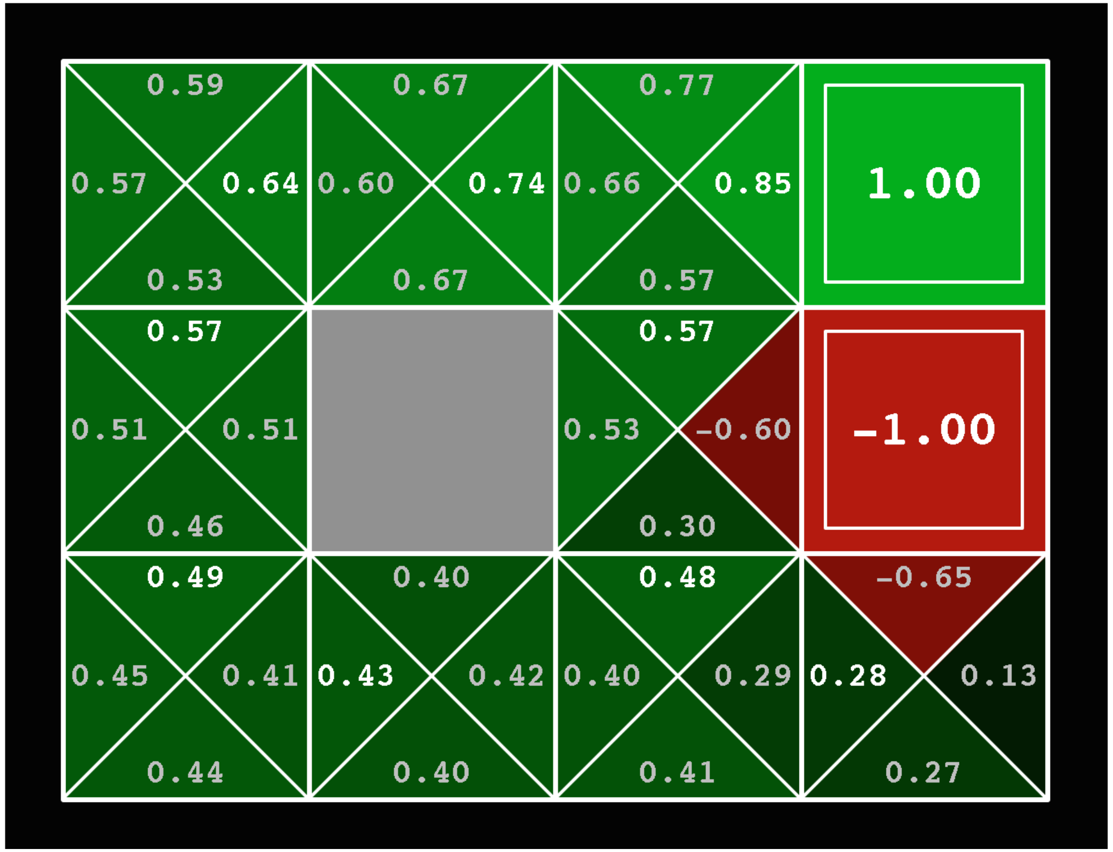
Solved MDP! Now what?

What are we going to do with these values??

$V^*(s)$



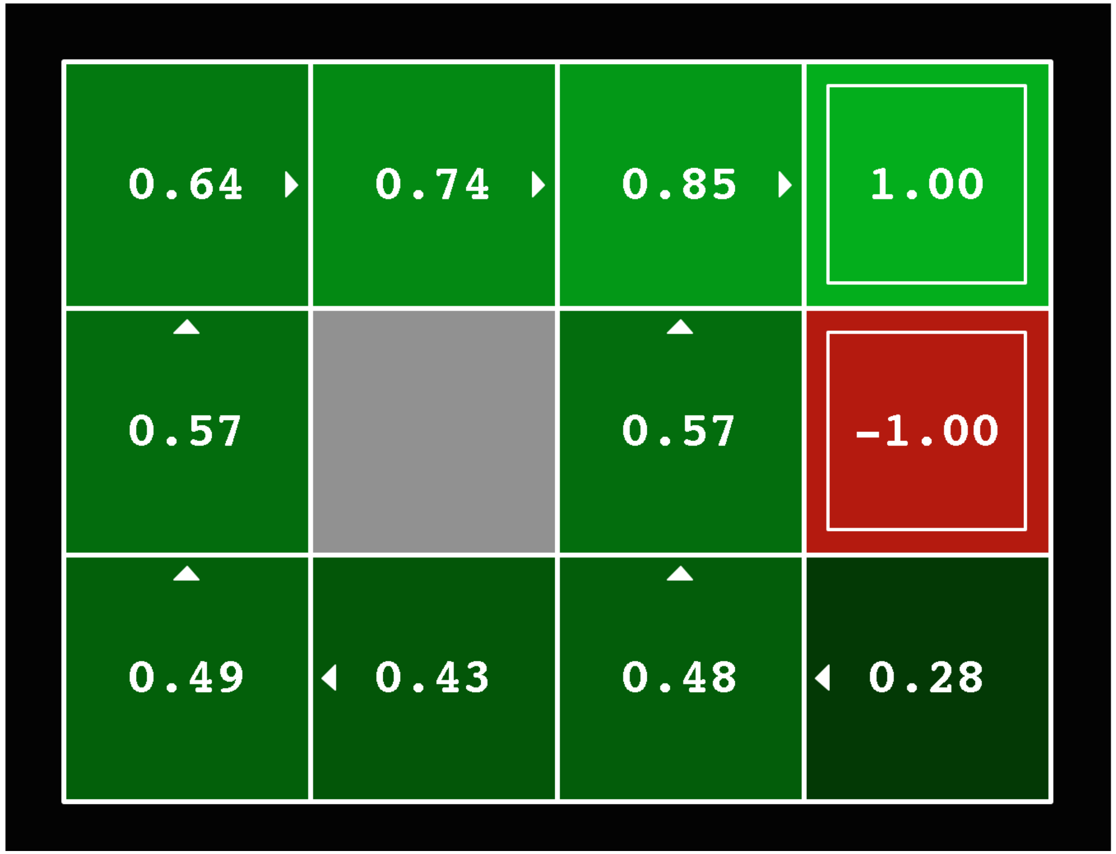
$Q^*(s,a)$

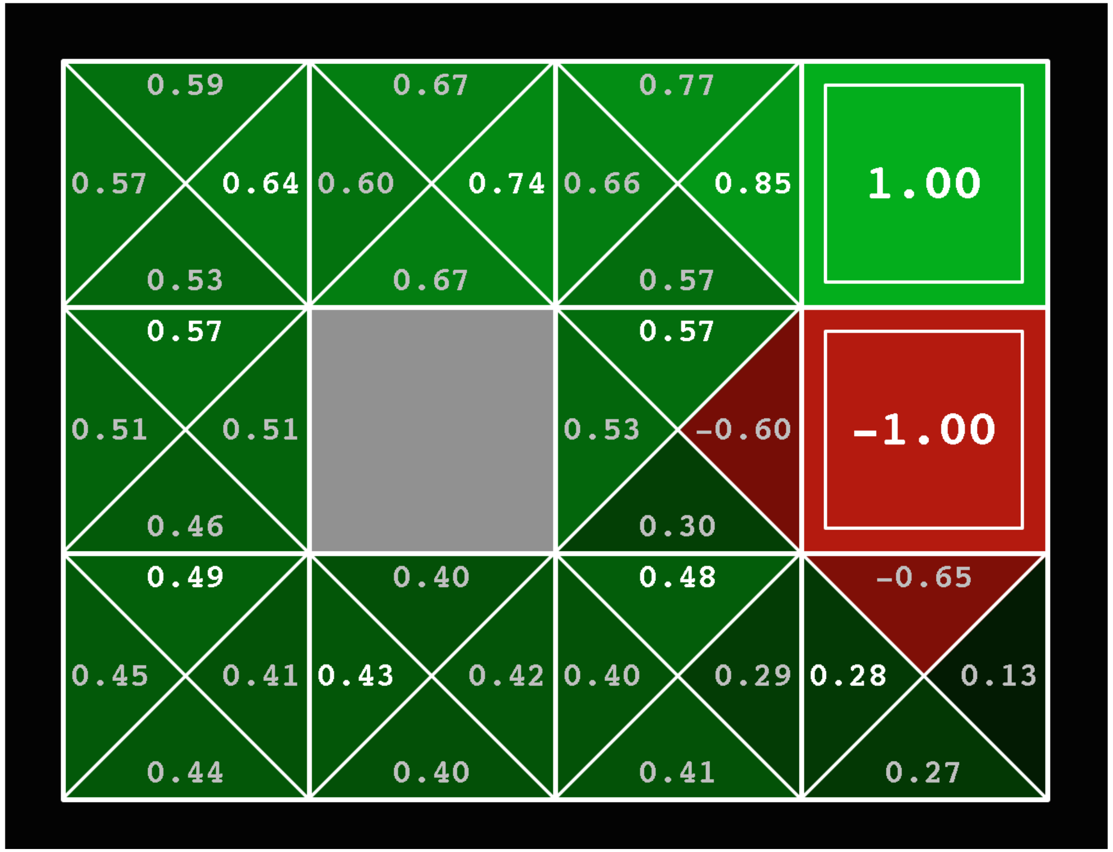


Piazza Poll 1

If you need to extract a policy, would you rather have

A) Values, B) Q-values or C) Z-values?

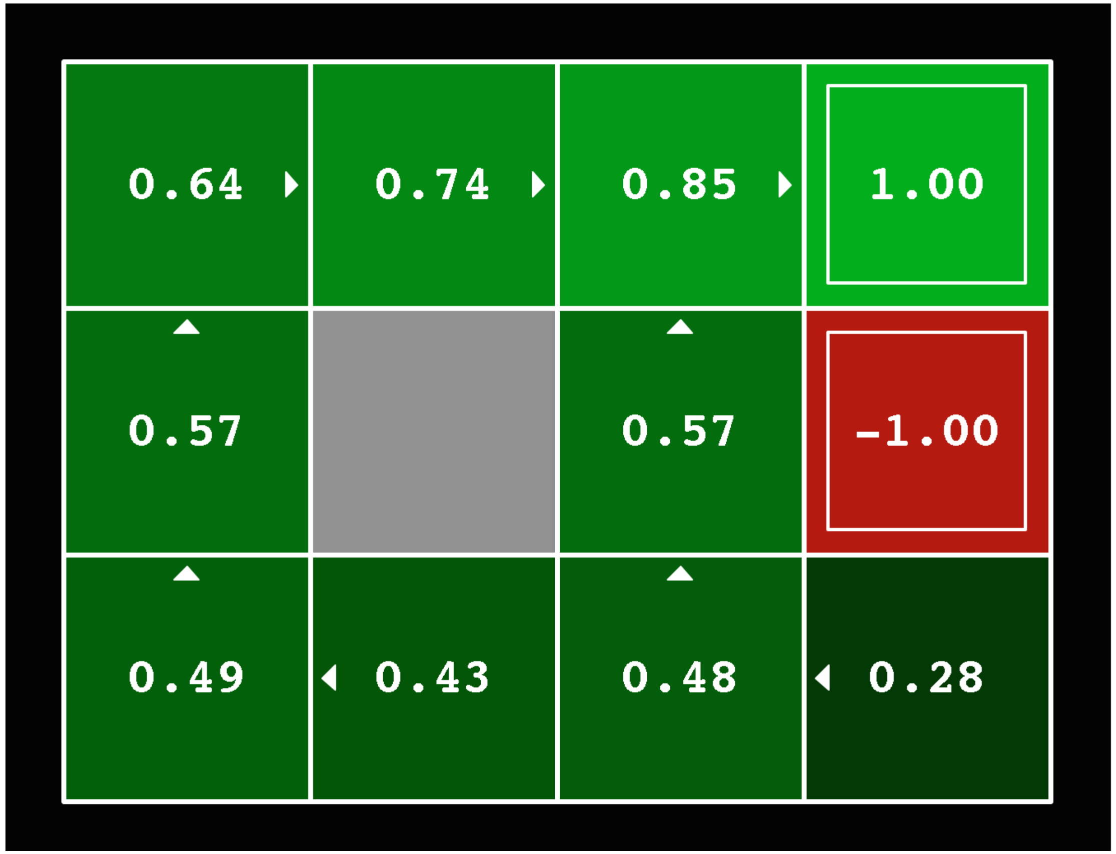


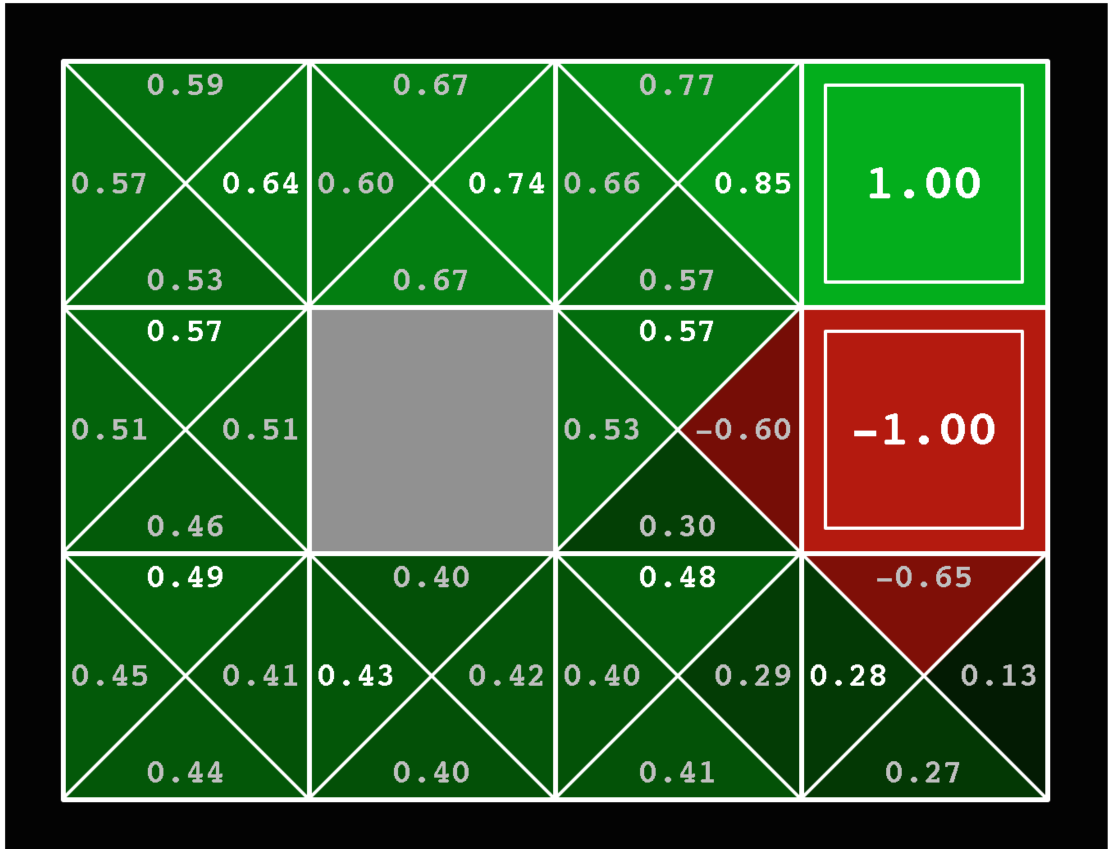


Piazza Poll 1

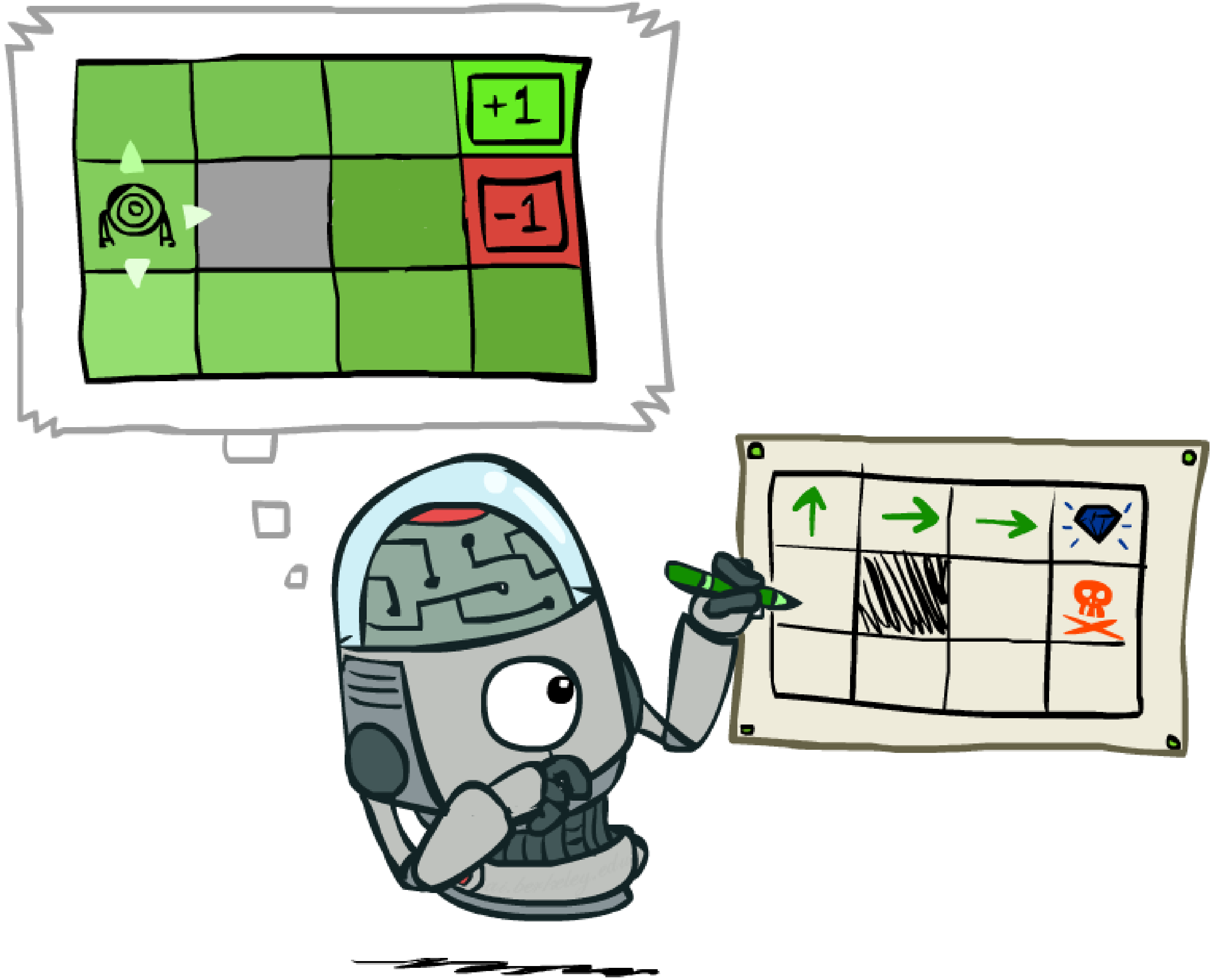
If you need to extract a policy, would you rather have

A) Values, $\boxed{\text{B) Q-values}}$ or C) Z-values?

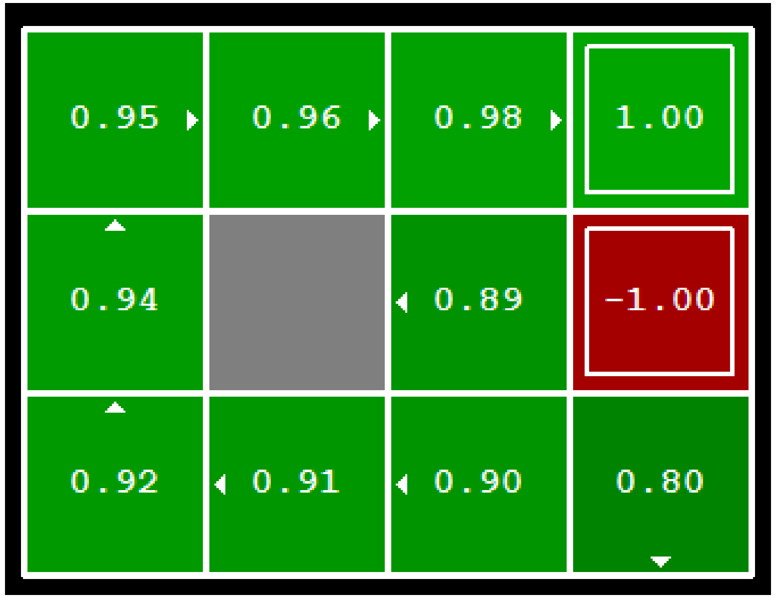




Policy Extraction



Computing Actions from Values



Let’s imagine we have the optimal values V*(s)

How should we act?

- It’s not obvious!

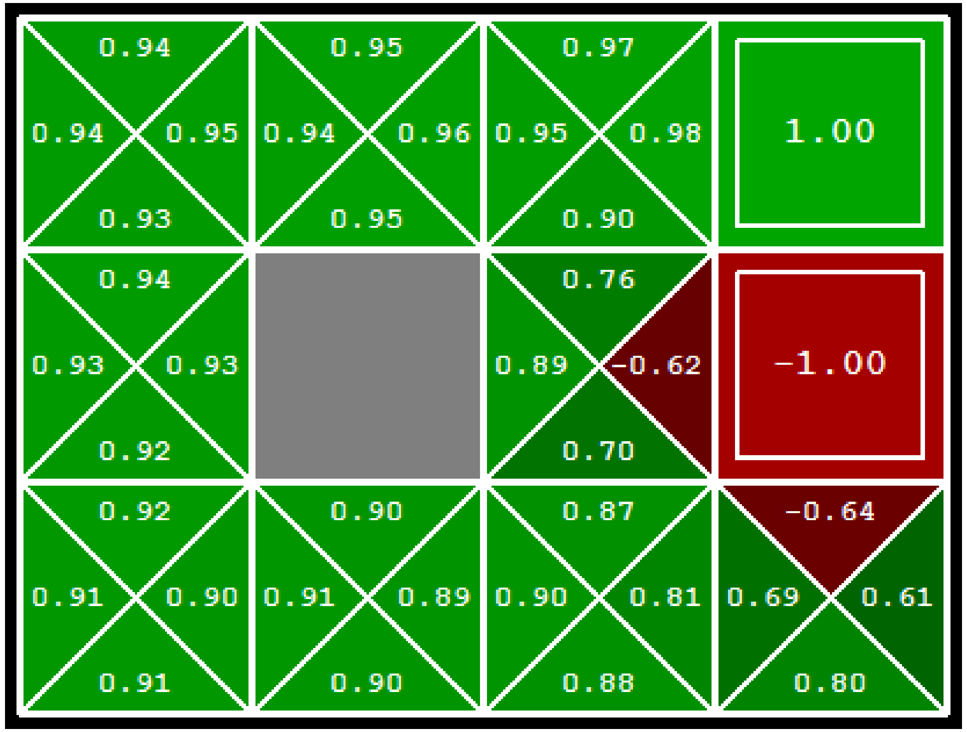
We need to do a mini-expectimax (one step)

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

Let's imagine we have the optimal q-values:



How should we act?

- Completely trivial to decide!

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

Important lesson: actions are easier to select from q-values than values!

Two Methods for Solving MDPs

Value iteration + policy extraction

- Step 1: Value iteration:

  $$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \ \forall s$$ **until convergence**

- Step 2: Policy extraction:

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

Policy iteration (out of scope for this course)

- Step 1: Policy evaluation:

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

- Step 2: Policy improvement:

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

- Repeat steps until policy converges

Summary: MDP Algorithms

So you want to….

- Compute optimal values: use value iteration or policy iteration
- Turn your values into a policy: use policy extraction (one-step lookahead)

All these equations 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

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

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

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

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

Piazza Poll 2

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

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

Select ALL that apply.

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

Piazza Poll 2

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

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

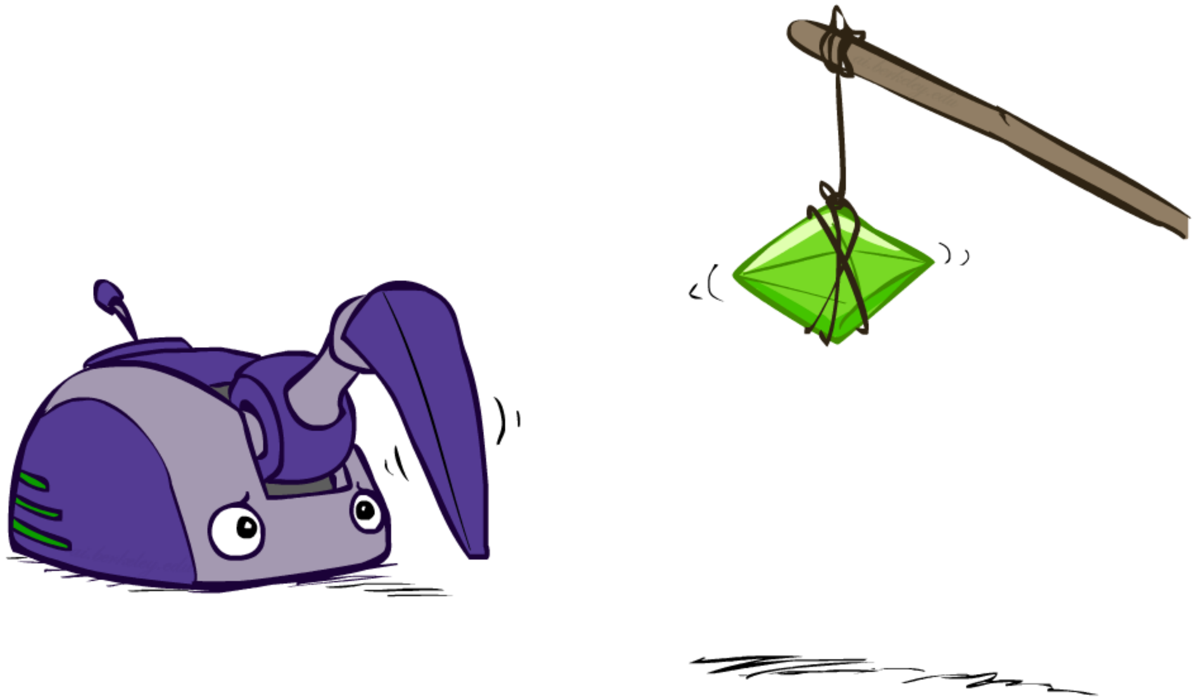
Select ALL that apply.

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

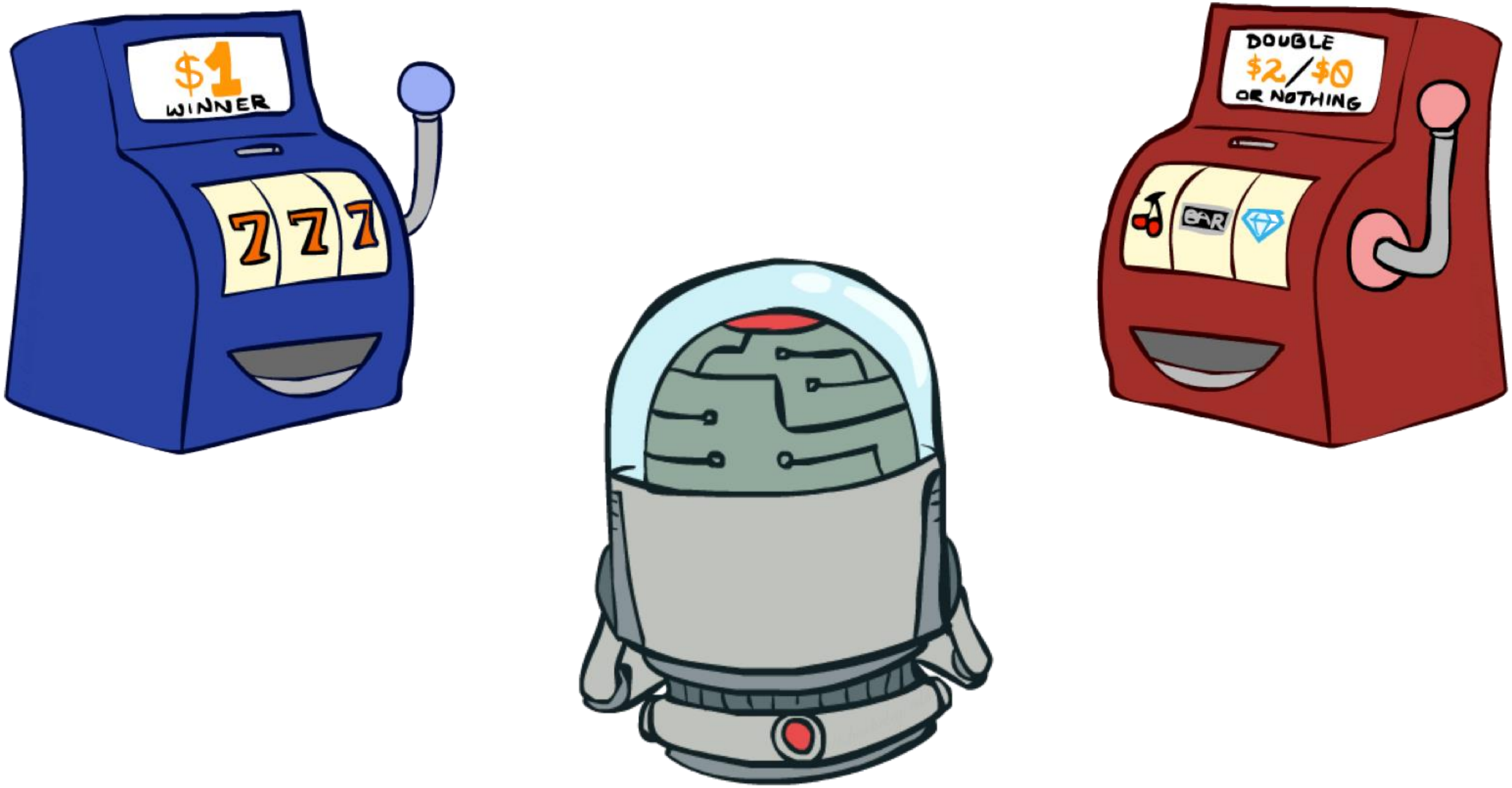
✓ D. $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$



Reinforcement Learning

Image: ai.berkeley.edu

Double Bandits



Slide: ai.berkeley.edu

Double-Bandit MDP

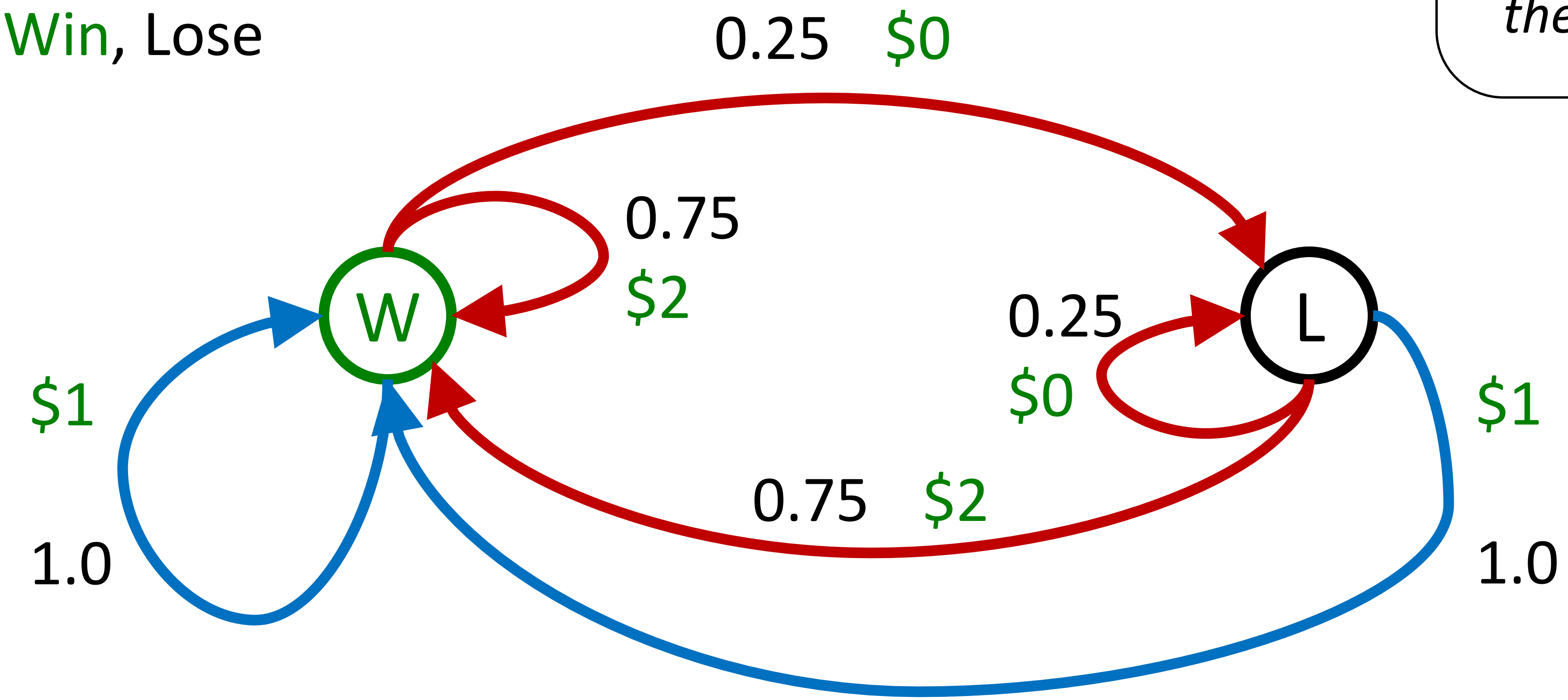
Actions: *Blue*, *Red*

States: Win, Lose

No discount

100 time steps

Both states have the same value



Slide: ai.berkeley.edu

Offline Planning

Solving MDPs is offline planning

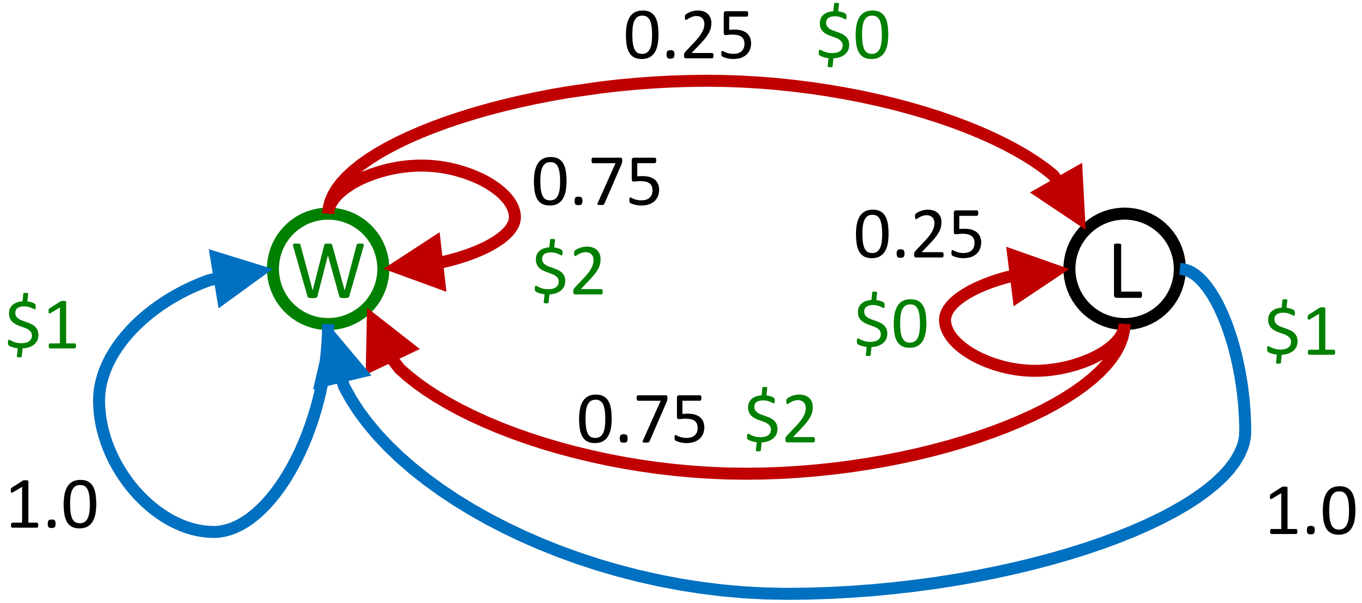
- You determine all quantities through computation
- You need to know the details of the MDP
- You do not actually play the game!

No discount

100 time steps

Both states have the same value

	Value
Play Red	150
Play Blue	100



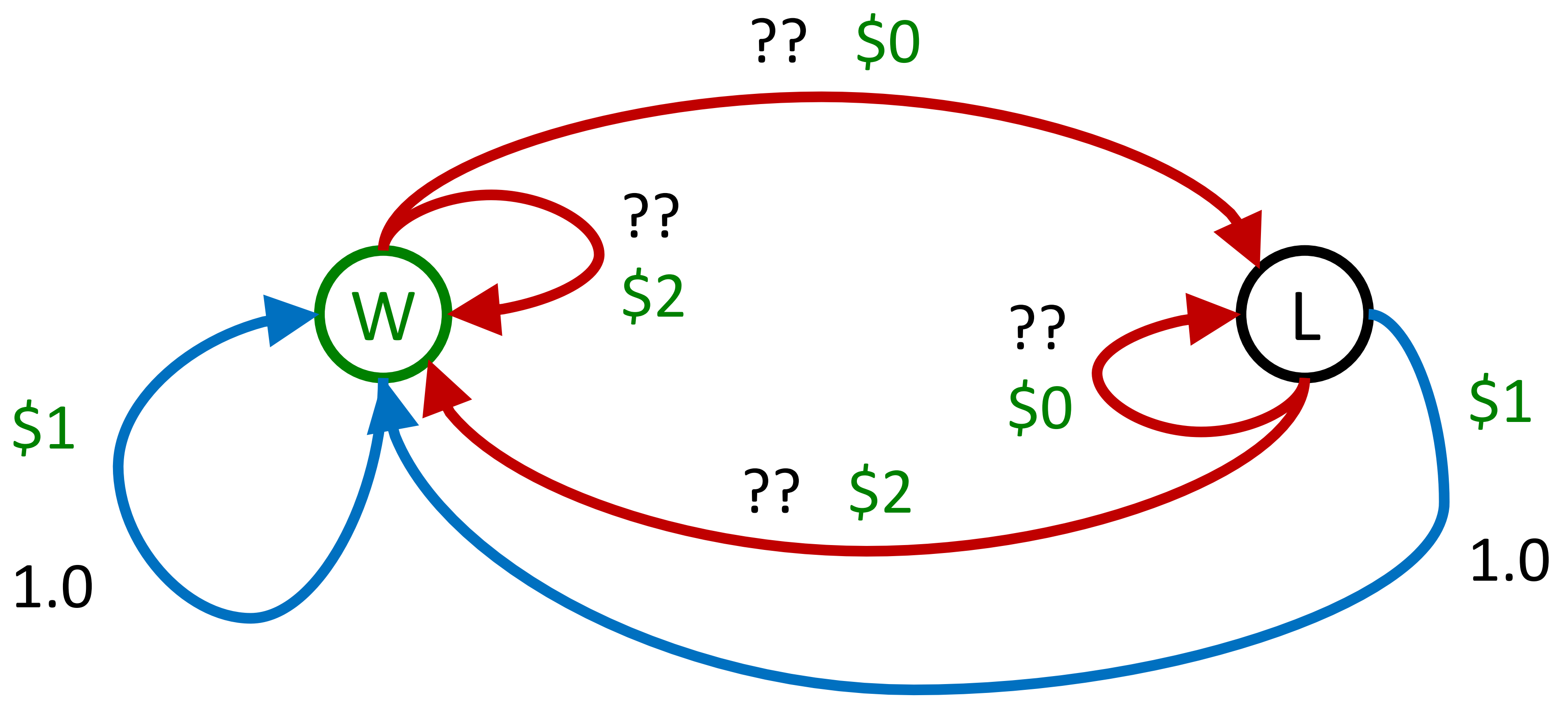
Let’s Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

Rules changed! Red's win chance is different.



Slide: ai.berkeley.edu

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

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

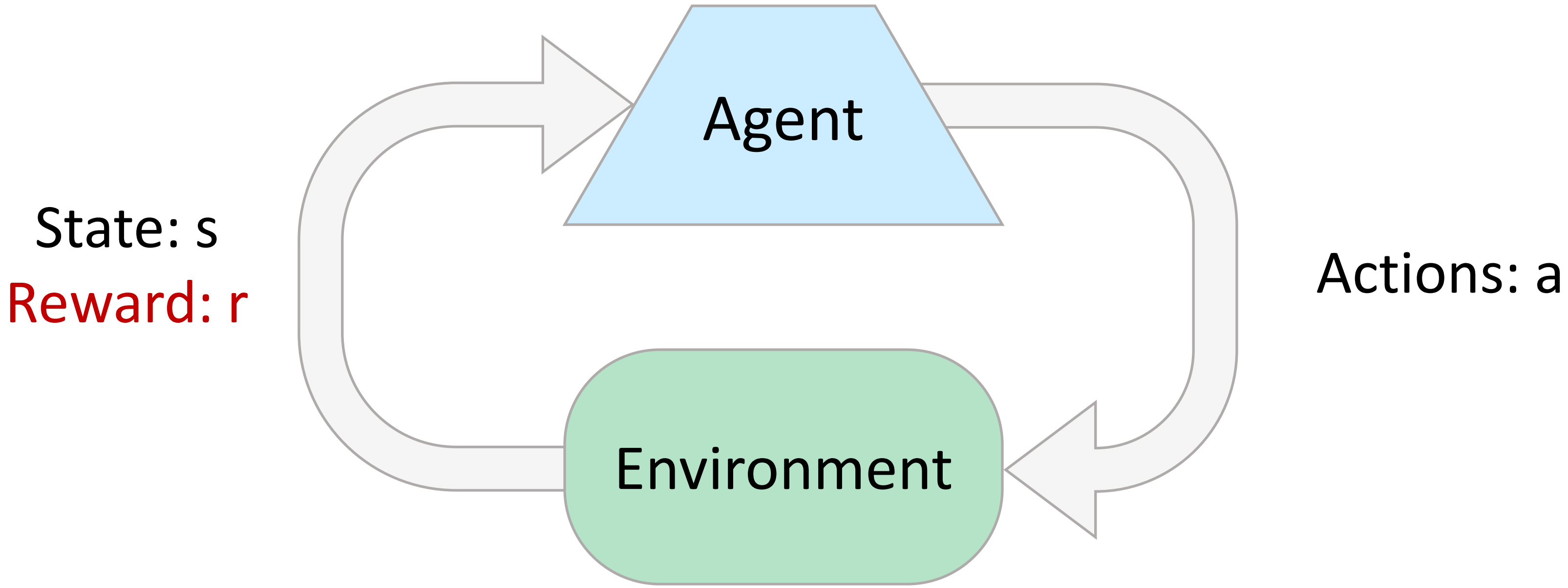
Value iteration: $$V_{k+1}(s) = \max_a \sum_{s'} \cancel{P(s'|s,a)}[\cancel{R(s,a,s')} + \gamma V_k(s')], \qquad \forall\, s$$

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

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!

Slide: ai.berkeley.edu

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]

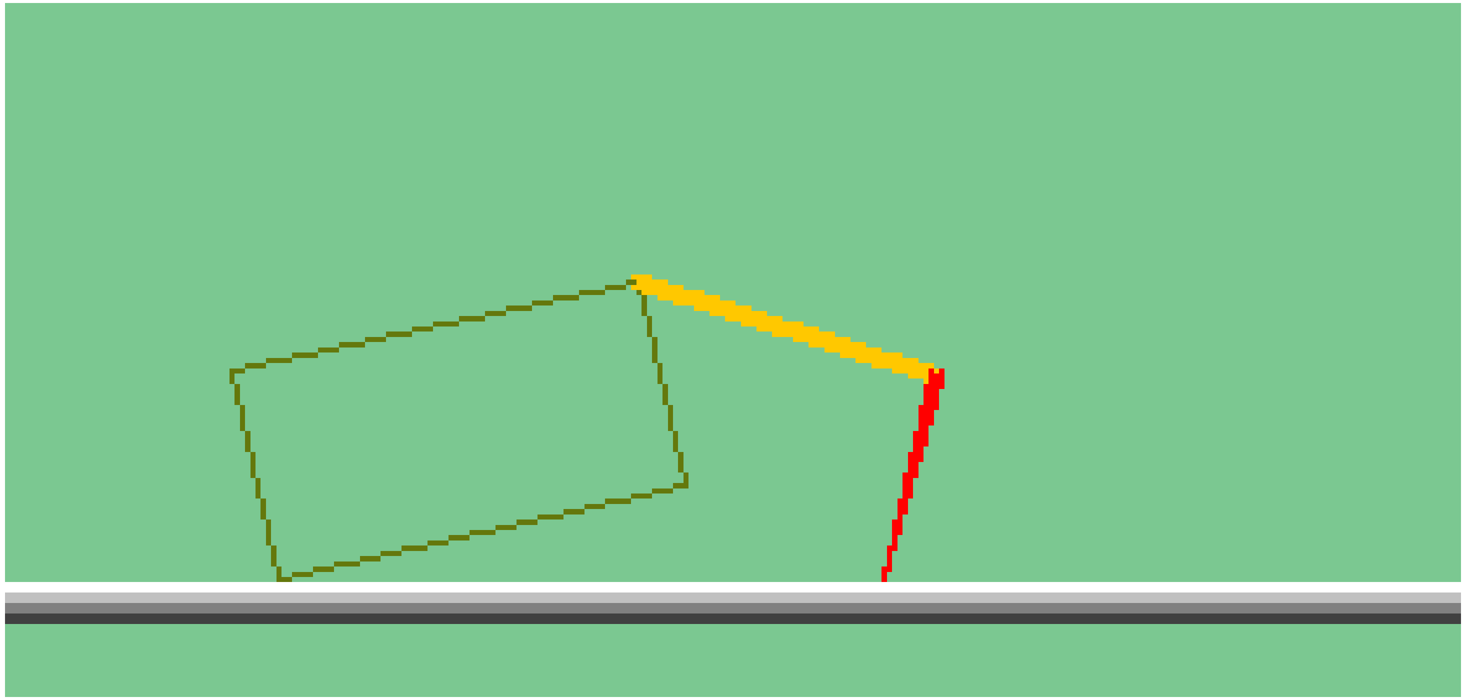
[Video: SNAKE – climbStep+sidewinding]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

The Crawler!



[Demo: Crawler Bot (L10D1)]

Demo Crawler Bot

Reinforcement Learning

Still assume a Markov decision process (MDP):

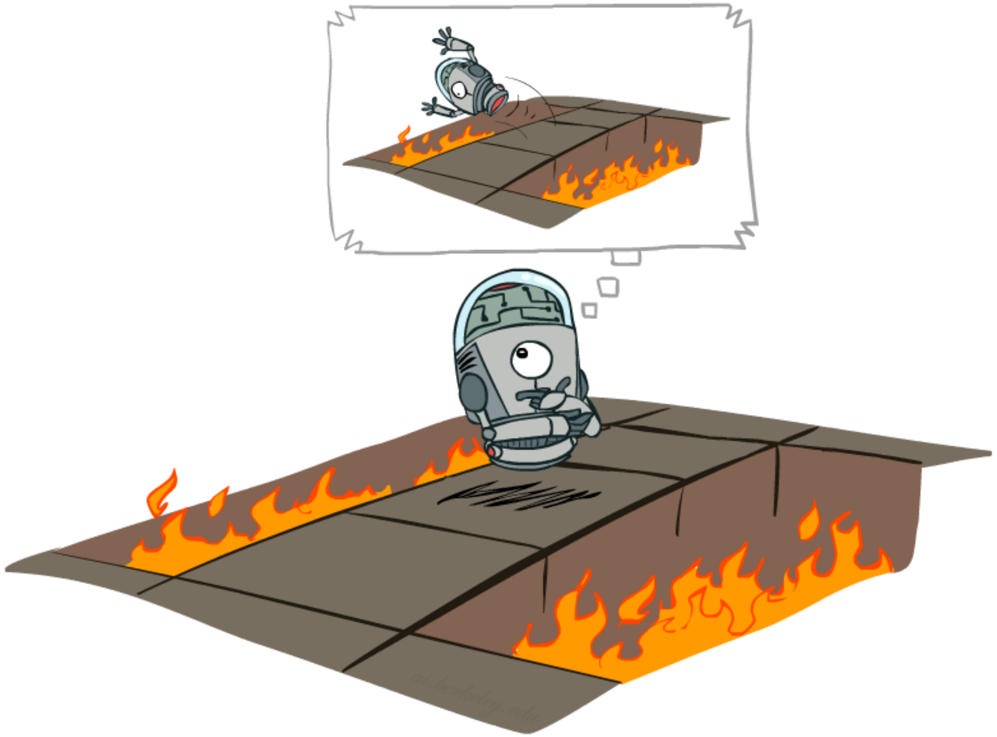
- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s,a,s')
- A reward function R(s,a,s')

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

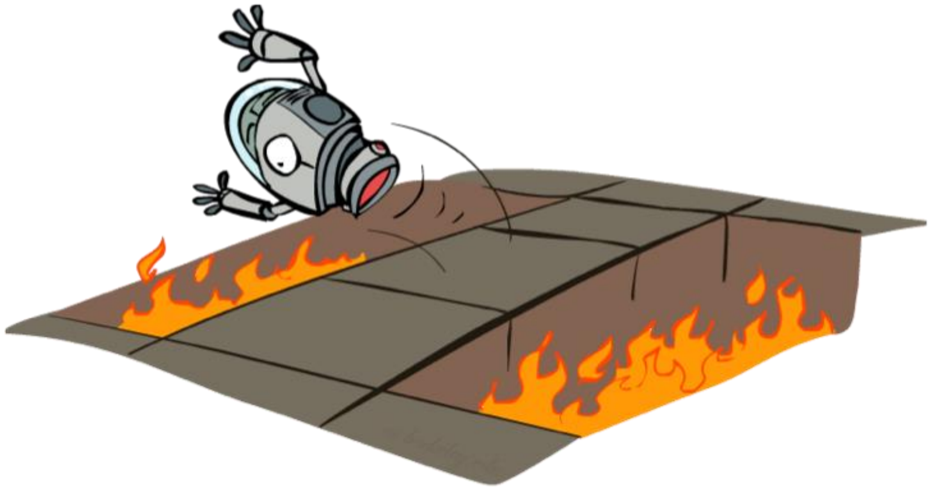
New twist: don't know T or R

- 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)

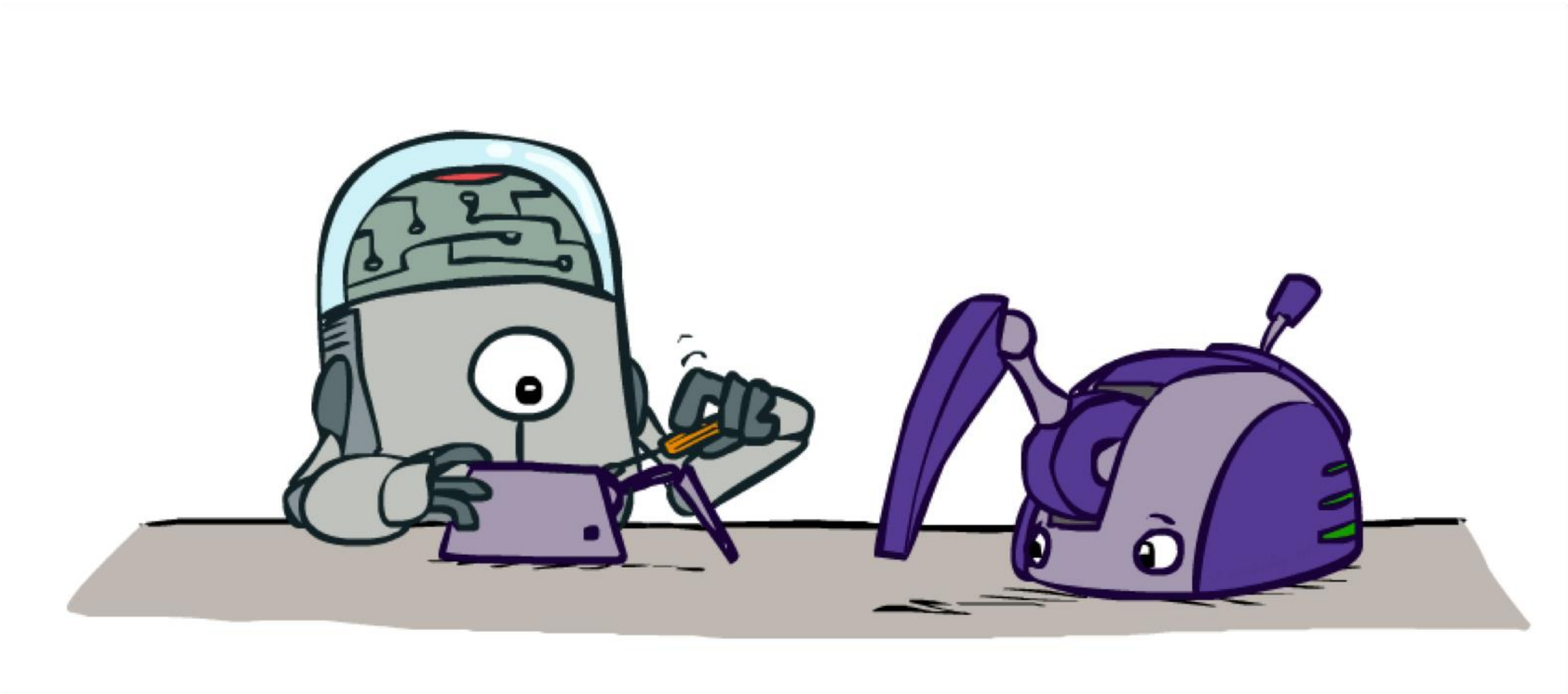


Offline Solution



Online Learning

Model-Based Learning



Slide: ai.berkeley.edu

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

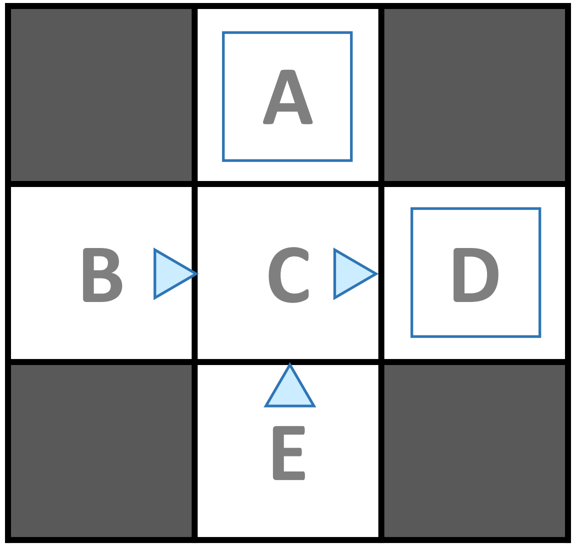
- Count outcomes s' for each s, a
- Normalize to give an estimate of $\widehat{T}(s, a, s')$
- Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s')

Step 2: Solve the learned MDP

- For example, use value iteration, as before

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

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

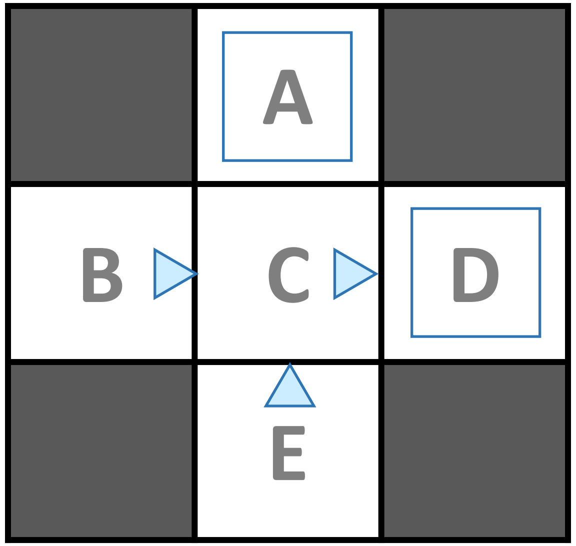
T(B, east, C) =
T(C, east, D) =
T(C, east, A) =
…

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

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

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

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

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
…

Example: Expected Age

Goal: Compute expected age of 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, … 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.

Sample-Based Policy Evaluation?

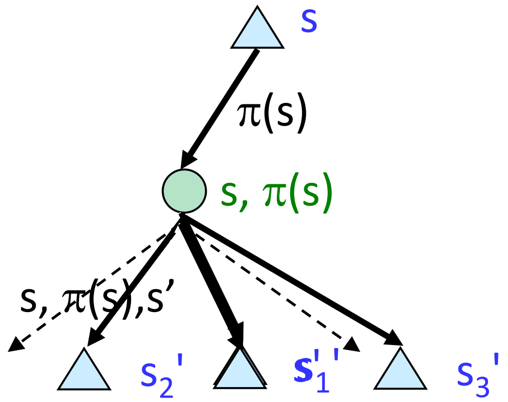
We want to improve our estimate of V by computing these averages:

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

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

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

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

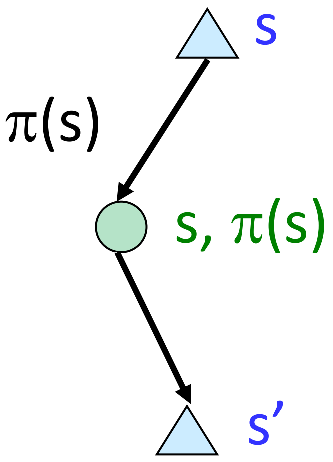


Almost! But we can't rewind time to get sample after sample from state s.

Temporal Difference Learning

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

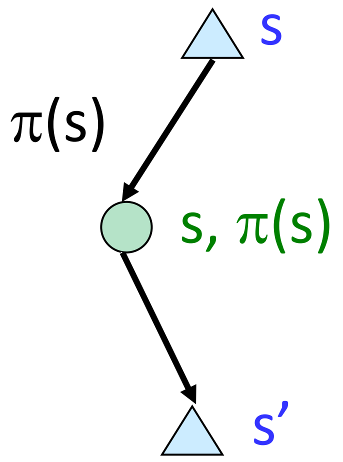
Sample of V(s): $sample = r + \gamma V^{\pi}(s')$

Update to V(s):

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

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

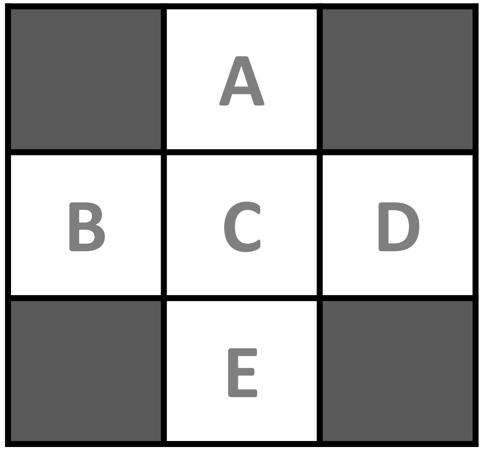
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha) V^{\pi}(s) + (\alpha)\, sample$

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

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

Example: Temporal Difference Learning

States

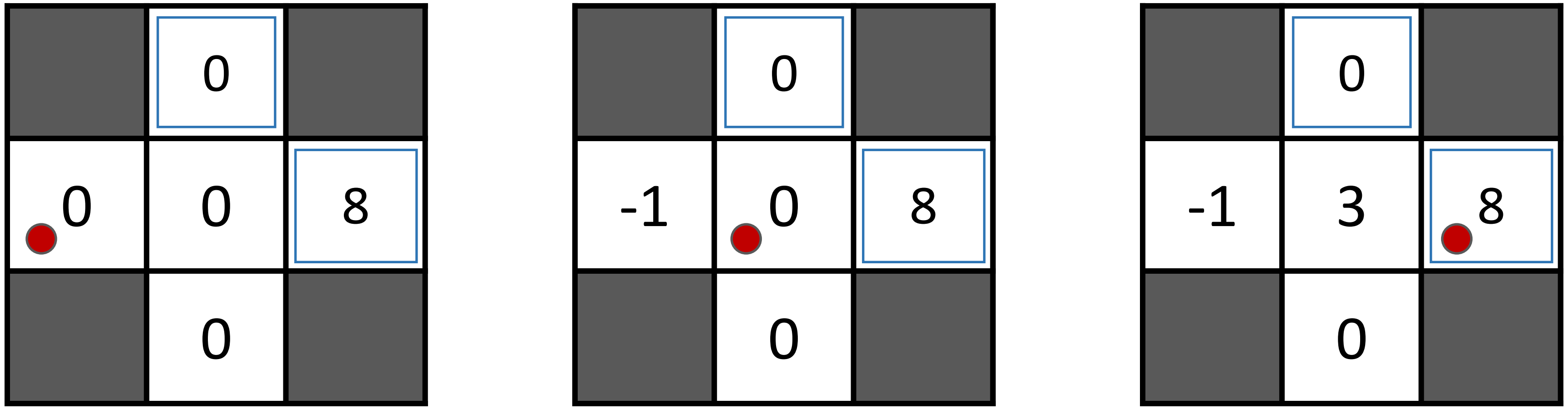


Assume: γ = 1,
α = 1/2

Observed Transitions

B, east, C, -2

C, east, D, -2



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

Piazza Poll 3

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) None of the above

Piazza Poll 3

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) None of the above

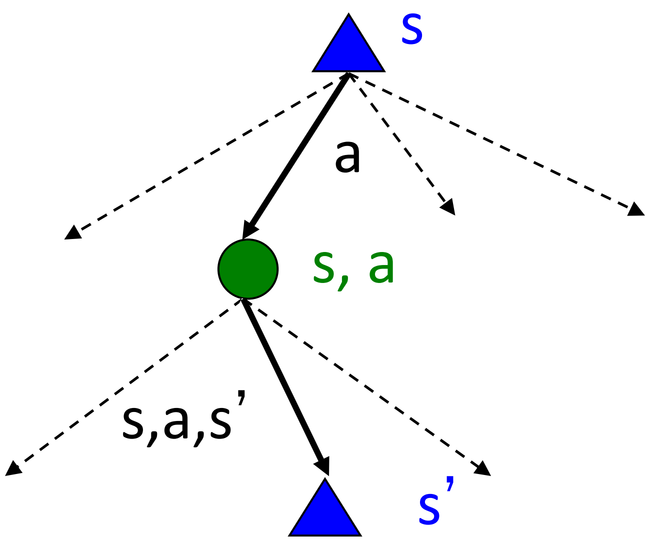
Problems with TD Value Learning

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!

Slide: ai.berkeley.edu

Detour: Q-Value Iteration

Value iteration:

- Start with $V_0(s) = 0$
- Given V_k, calculate the iteration 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 iteration 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

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

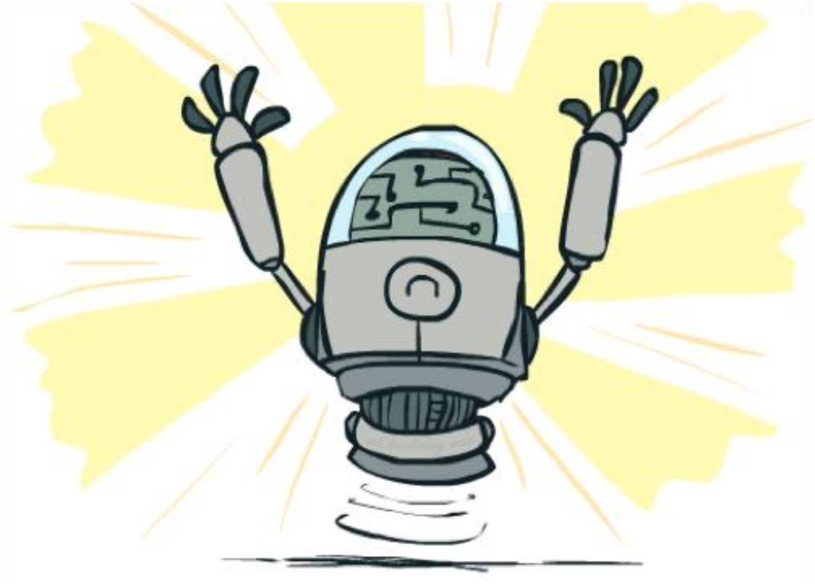
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Slide: ai.berkeley.edu

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 (!)

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

Slide: ai.berkeley.edu

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

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

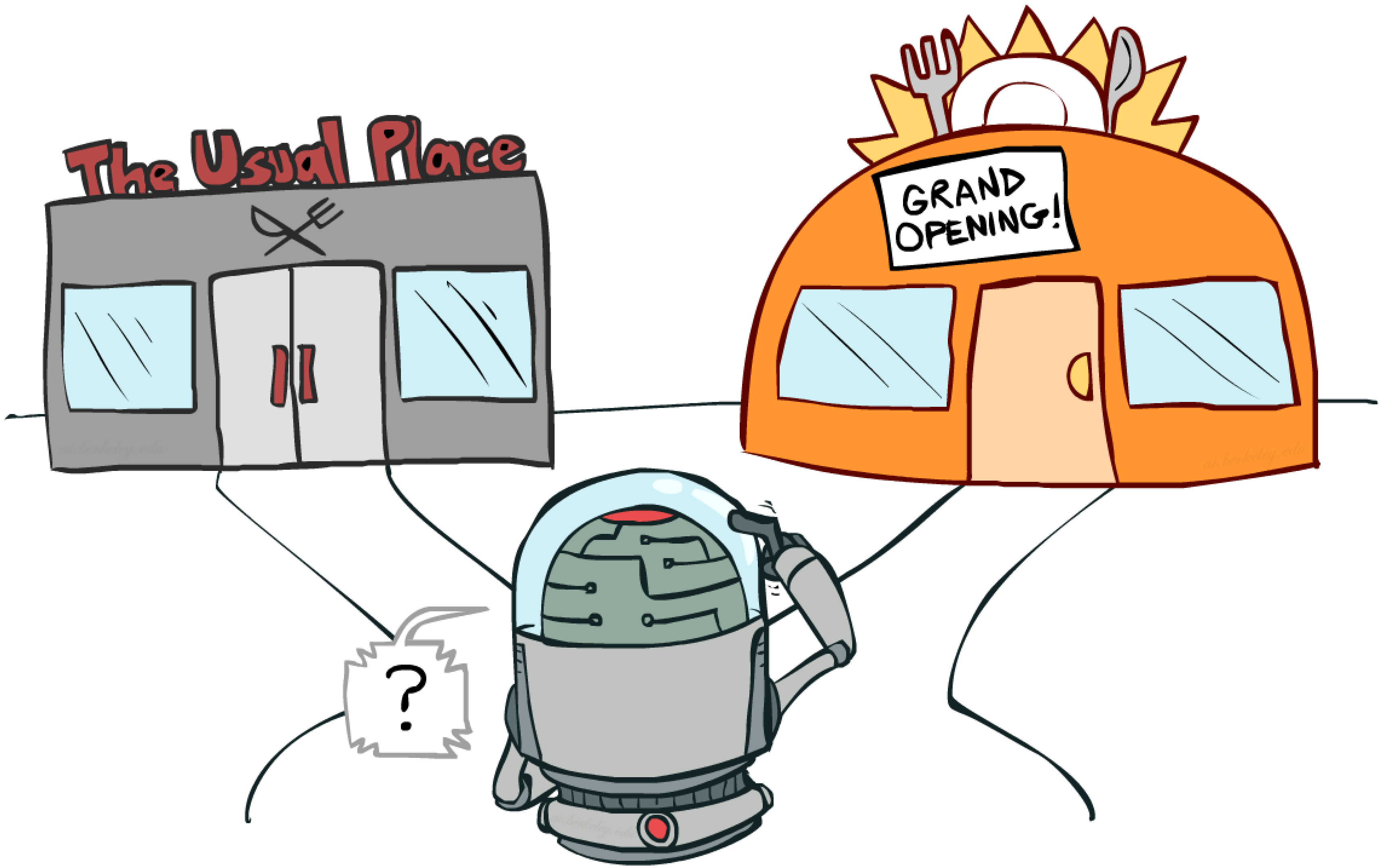
Q-learning: $Q(s,a) \;=\; Q(s,a) \;+ \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Demo Q-Learning Auto Cliff Grid

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

[python gridworld.py -n 0.0 -a q -k 400 -g CliffGrid -d 1.0 -r -0.1 -e 1.0]

Exploration vs. Exploitation



Slide: ai.berkeley.edu

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

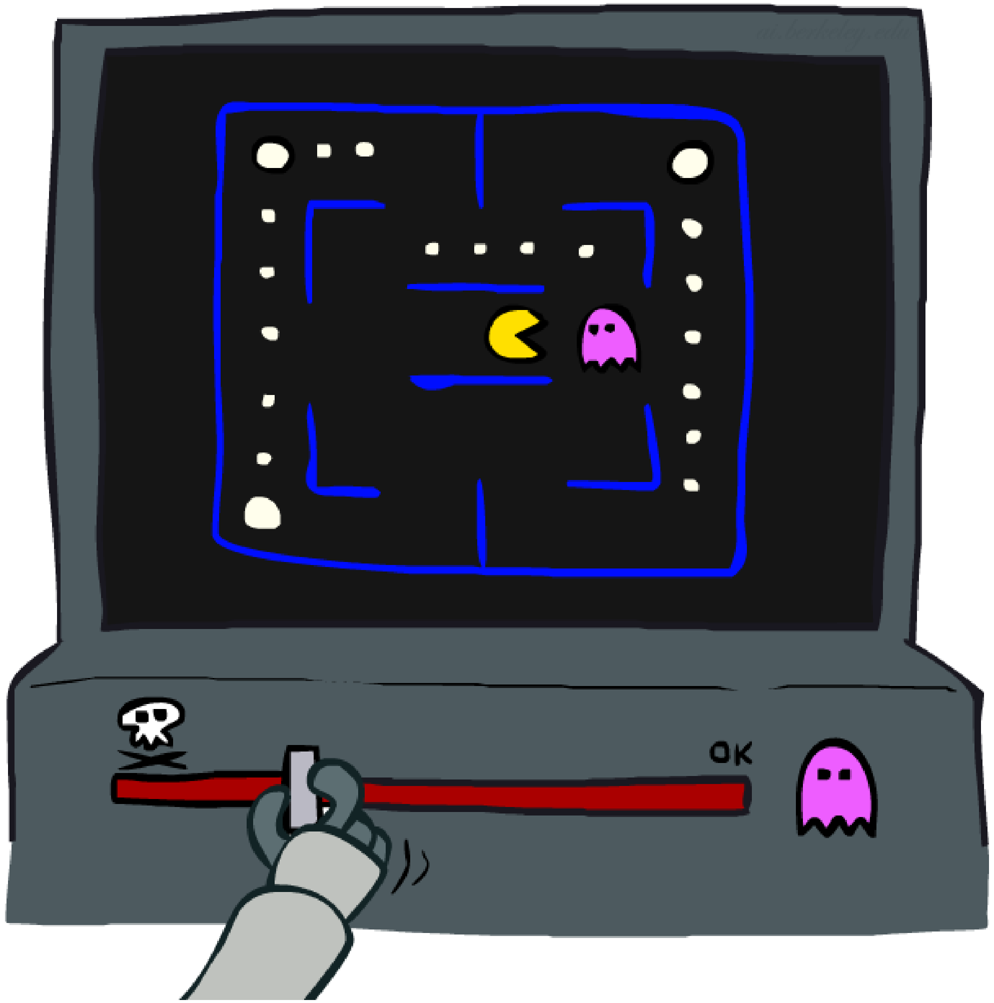
- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

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

Demo Q-learning – Manual Exploration – Bridge Grid

Demo Q-learning – Epsilon-Greedy – Crawler

Approximate Q-Learning



Example: Pacman

How many possible states?

- 55 (non-wall) positions
- 1 Pacman
- 2 Ghosts
- Dots eaten or not

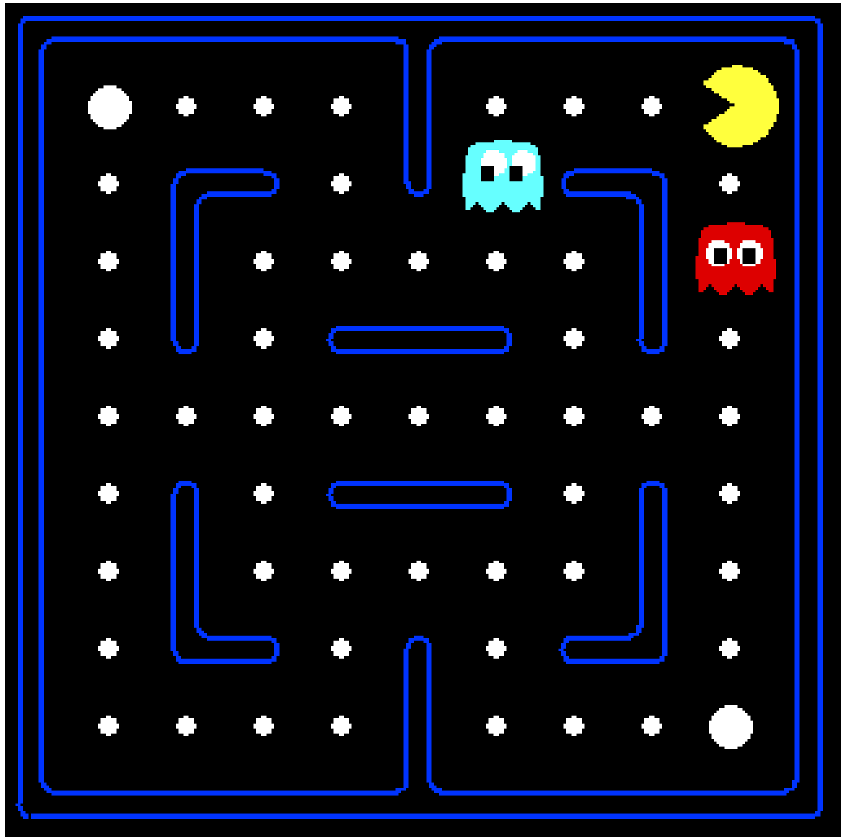


Image: ai.berkeley.edu

Generalizing Across States

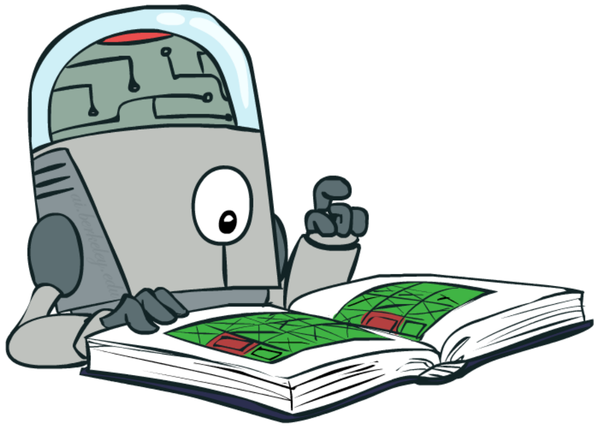
Basic Q-Learning keeps a table of all q-values

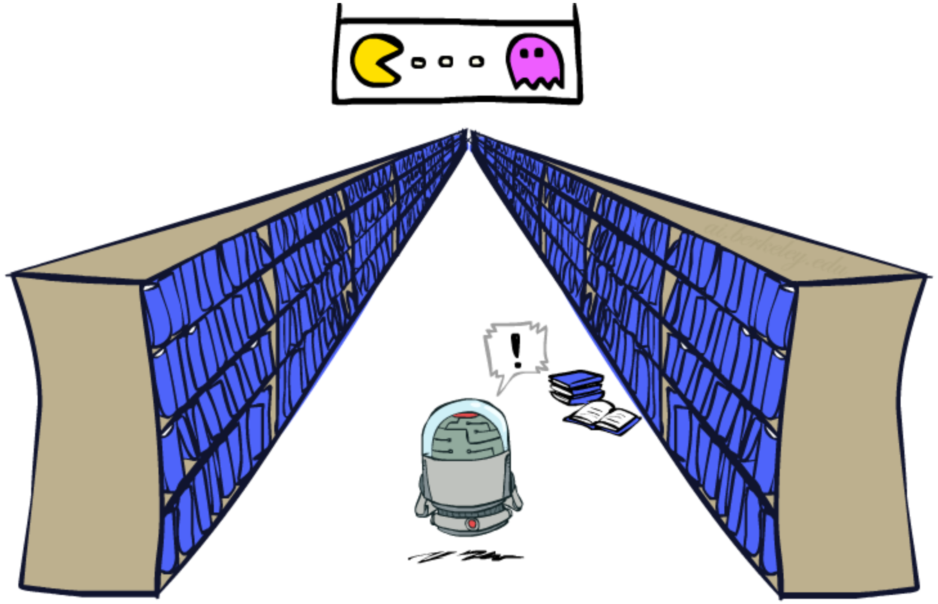
In realistic situations, we cannot possibly learn about every single state!

- Too many states to visit them all in training
- Too many states to hold the q-tables in memory

Instead, 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 over and over again



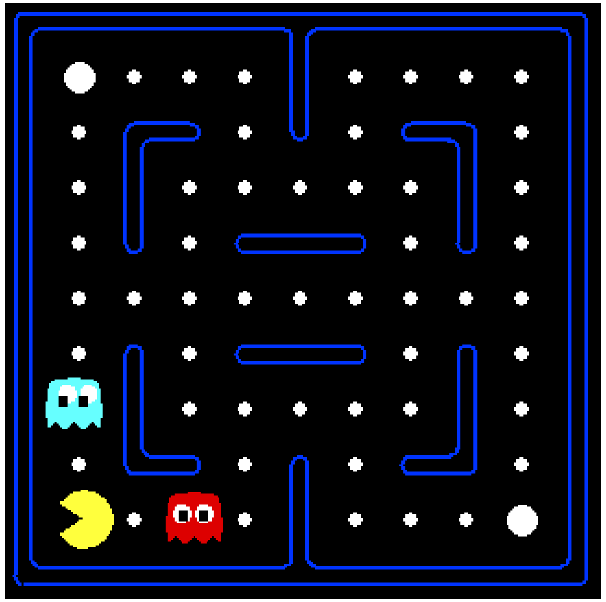


Slide: ai.berkeley.edu

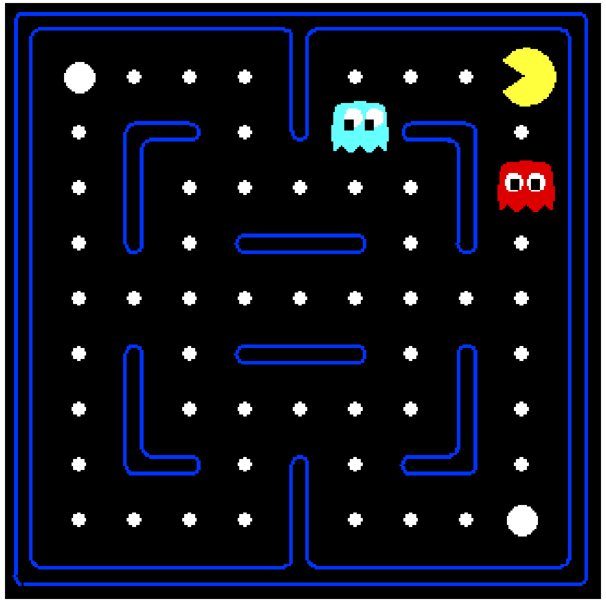
[demo – RL pacman]

Example: Pacman

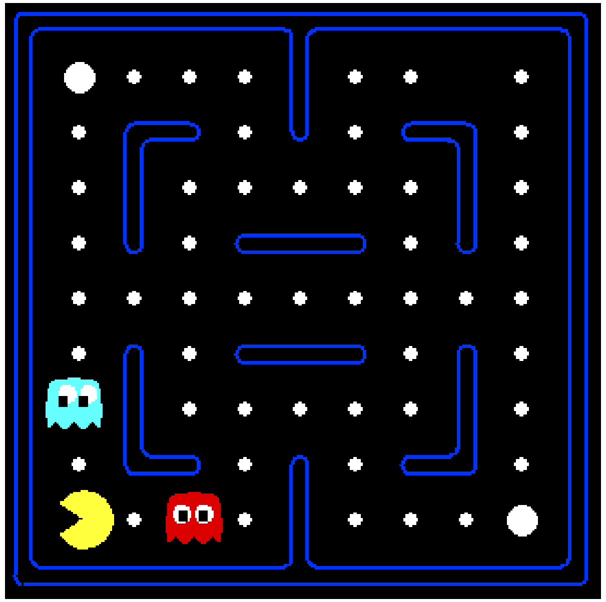
Let's say we discover through experience that this state is bad:



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



Or even this one!

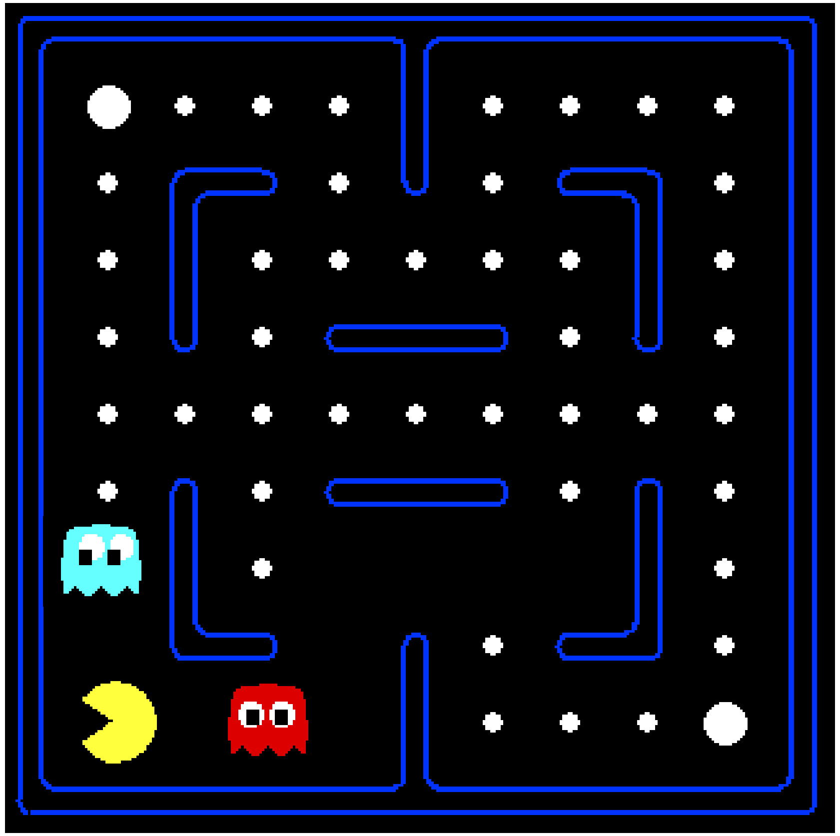


[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

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:

- $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_M f_M(s)$
- $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_M f_M(s,a)$

Advantage: our experience is summed up in a few powerful numbers

Disadvantage: states may share features but actually be very different in value!

Updating a linear value function

$Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + \ldots + w_M f_M(s,a)$

Original Q learning rule tries to reduce prediction error at s, a:

- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$

  $= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)] \, f_i(s,a)$

$Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + \ldots + w_M f_M(s,a)$

Updating a linear value function

Original Q learning rule tries to reduce prediction error at s, a:

- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)]\, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$
  $= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)]\, f_i(s,a)$

$$Q_{\boldsymbol{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a)$$

$$Error(w) = \frac{1}{2}\left(y - \boldsymbol{w}^T f(x)\right)^2$$

$$\frac{\partial Q}{\partial w_2} =$$

$$\frac{\partial Error}{\partial \boldsymbol{w}} = -\left(y - \boldsymbol{w}^T f(x)\right) f(x)$$

Updating a linear value function

Original Q learning rule tries to reduce prediction error at s, a:

- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)]\, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$
  $= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_{\mathbf{w}}(s',a') - Q_{\mathbf{w}}(s,a)]\, f_i(s,a)$

Qualitative justification:

- Pleasant surprise: increase weights on +ve features, decrease on –ve ones
- Unpleasant surprise: decrease weights on +ve features, increase on –ve ones

Approximate Q-Learning

$$Q_w(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_M f_M(s,a)$$

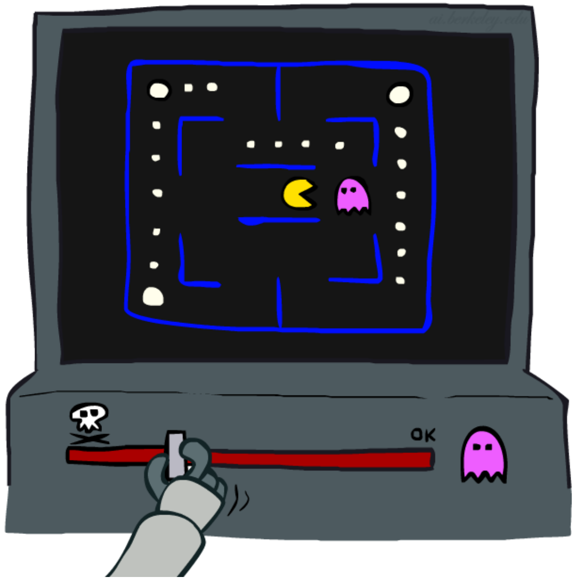
Q-learning with linear Q-functions:

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

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

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

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



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

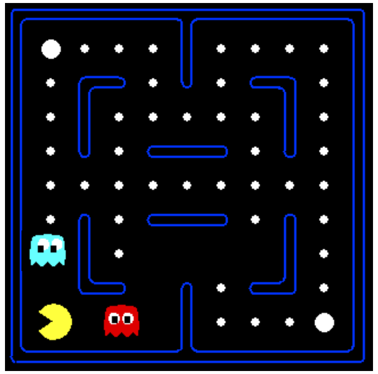
Formal justification: online least squares

Slide: ai.berkeley.edu

Example: Q-Pacman

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

s

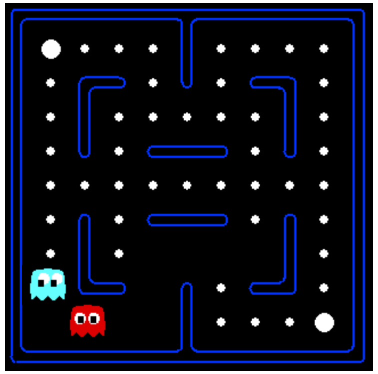


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$
$r = -500$

s'



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

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

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

difference $= -501$

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

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

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

[Demo: approximate Q-learning pacman (L11D10)]

Demo Approximate Q-Learning -- Pacman

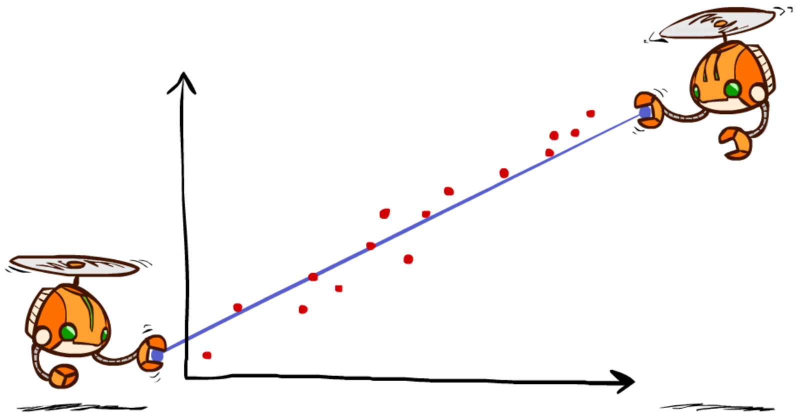
Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a) \right] f_m(s, a)$$

"target" "prediction"

Slide: ai.berkeley.edu

Reinforcement Learning Milestones

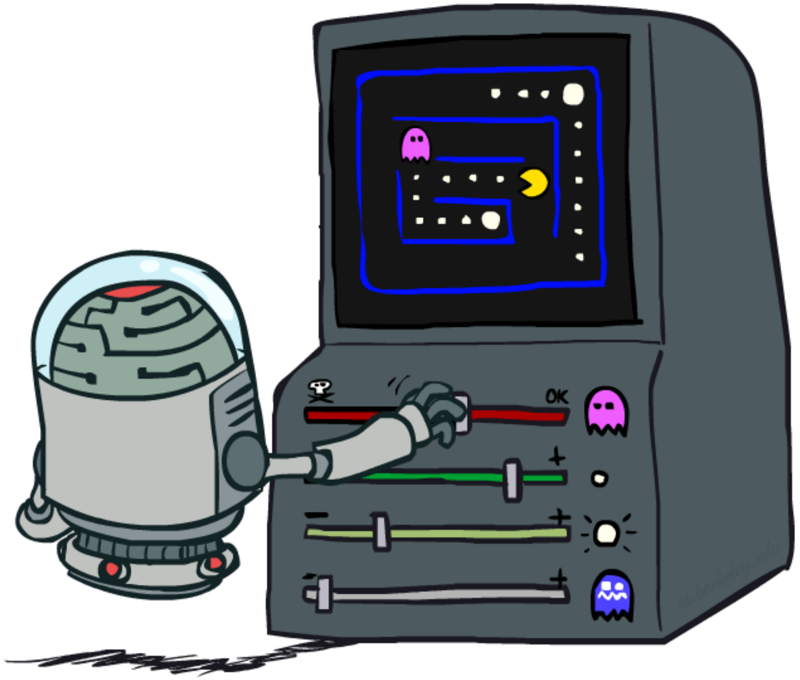


Image: ai.berkeley.edu

TDGammon

1992 by Gerald Tesauro, IBM

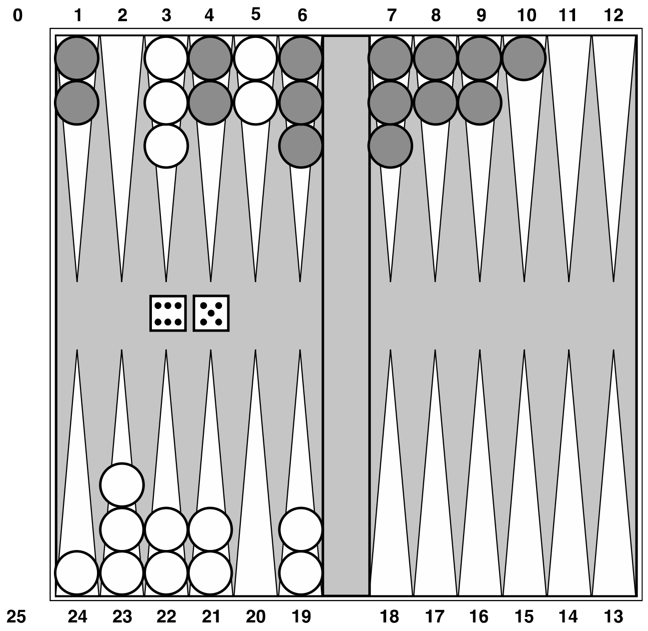
4-ply lookahead using V(s) trained from 1,500,000 games of self-play

3 hidden layers, ~100 units each

Input: contents of each location plus several handcrafted features

Experimental results:

- Plays approximately at parity with world champion
- Led to radical changes in the way humans play backgammon



Deep Q-Networks

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

$Q_w(s,a)$: Neural network

Deep Mind, 2015

Used a deep learning network to represent Q:

- Input is last 4 images (84x84 pixel values) plus score

49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro

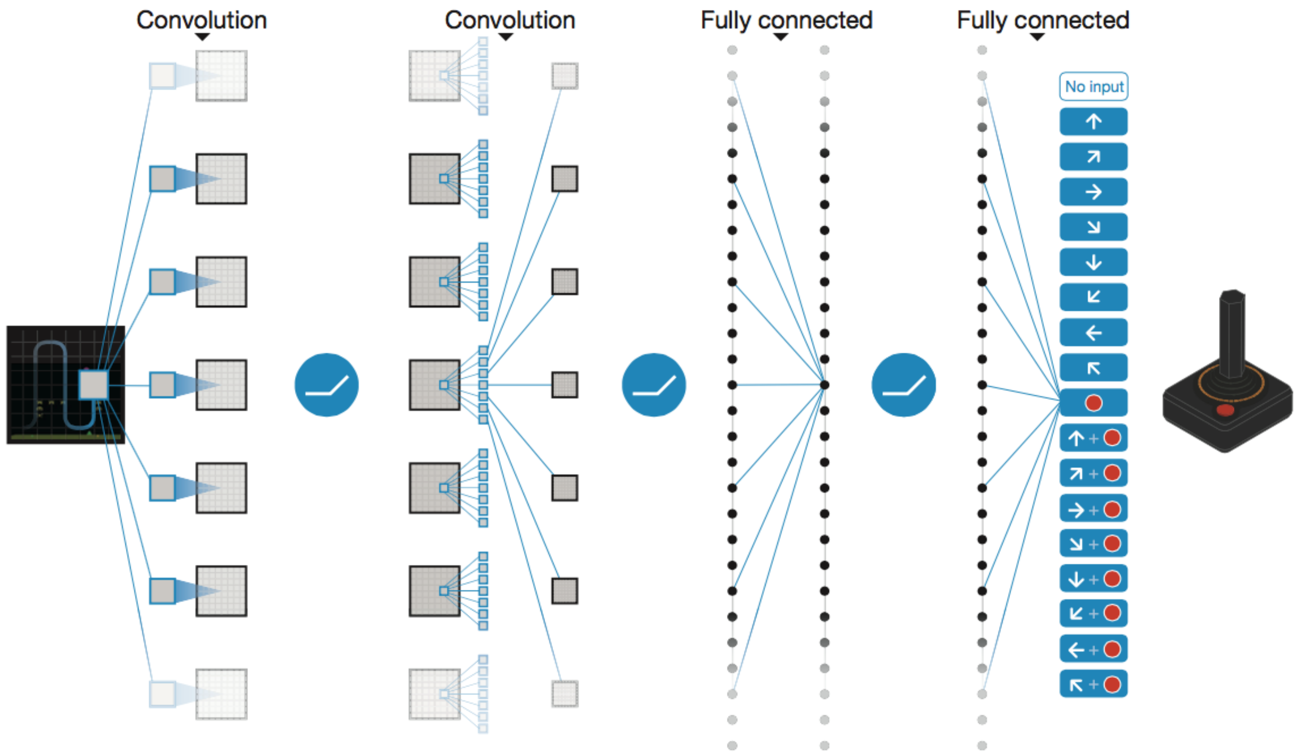
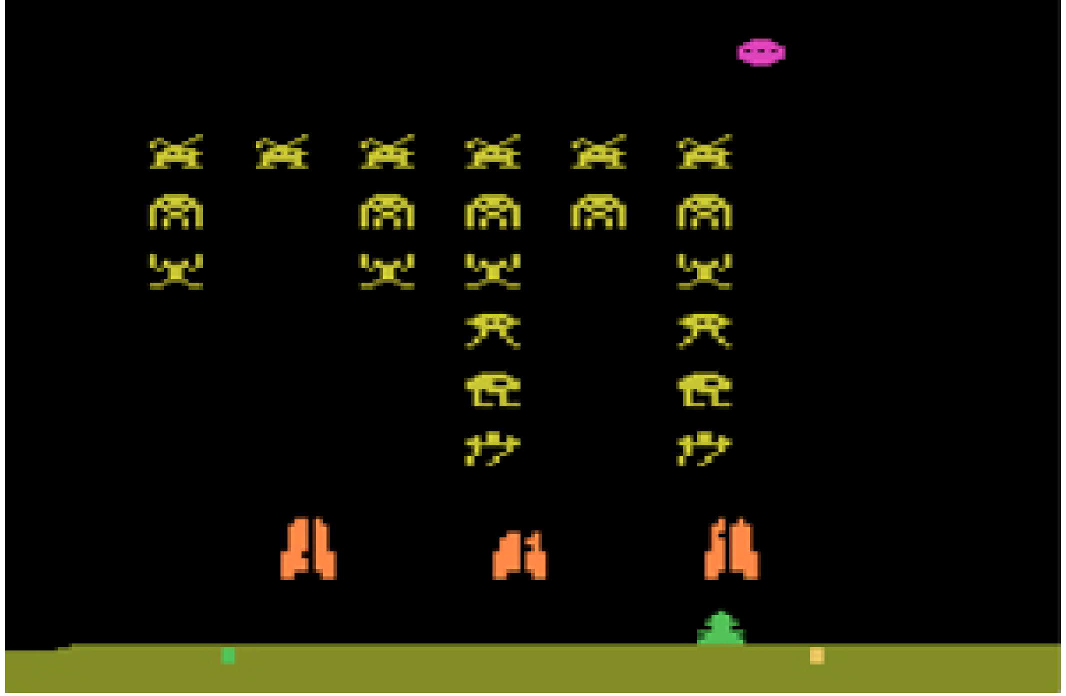


Image: Deep Mind



Images: Open AI, Atari

OpenAI Gym

2016+

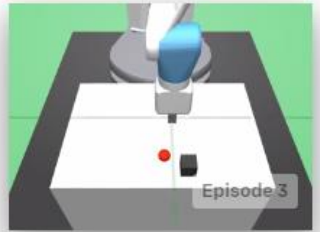
Benchmark problems for learning agents

https://gym.openai.com/envs

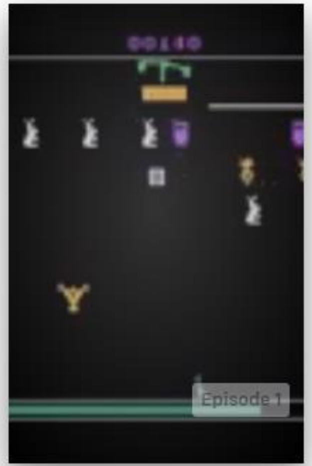
Breakout-ram-v0
Maximize score in the game Breakout, with RAM as input

Acrobot-v1
Swing up a two-link robot.

Ant-v2
Make a 3D four-legged robot walk.



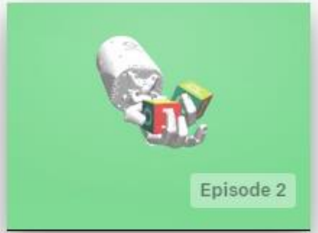
FetchPush-v0
Push a block to a goal position.



Carnival-v0
Maximize score in the game Carnival, with screen images as input

MountainCarContinuous-v0
Drive up a big hill with continuous control.

Humanoid-v2
Make a 3D two-legged robot walk.



HandManipulateBlock-v0
Orient a block using a robot hand.

Images: Open AI

AlphaGo, AlphaZero

Deep Mind, 2016+

Autonomous Vehicles?