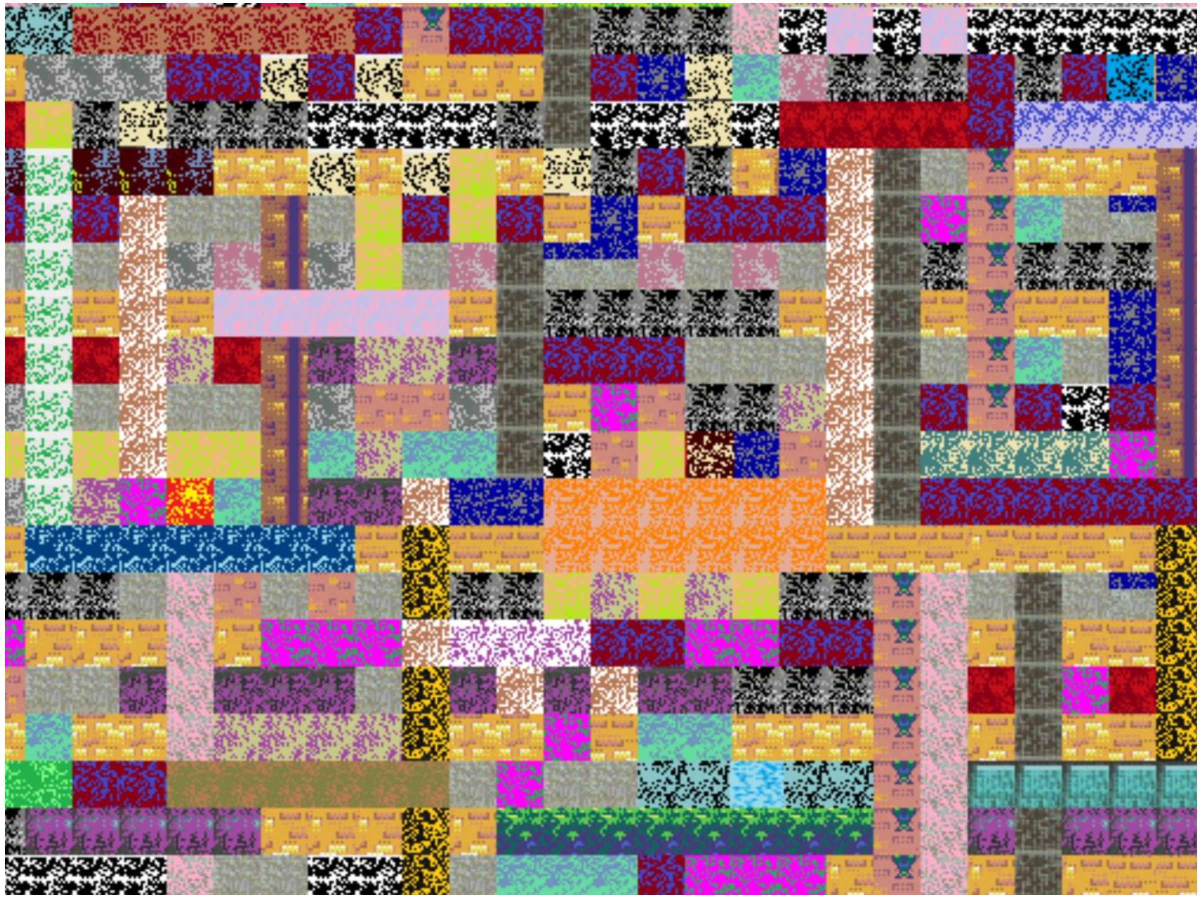
Warm-up as you log in

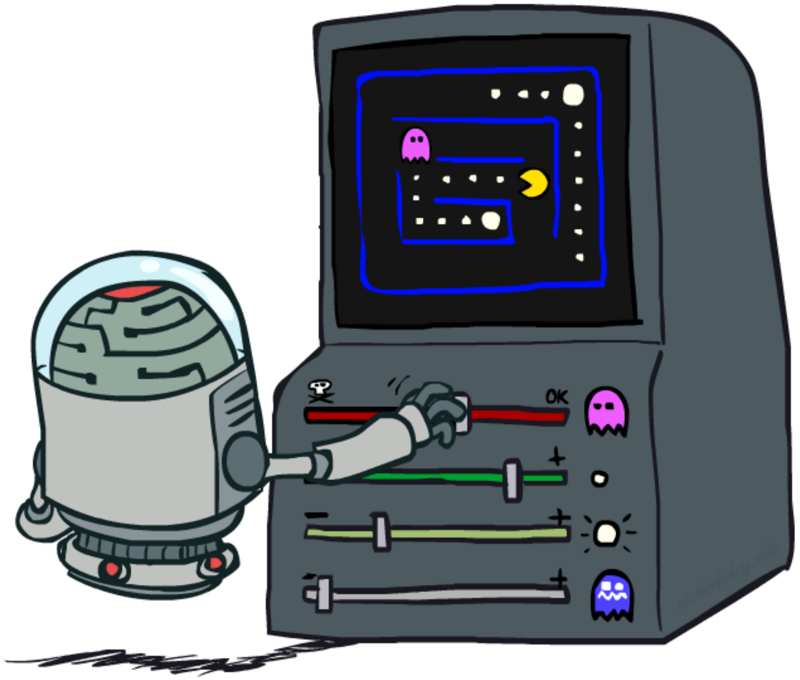
https://high-level-4.herokuapp.com/experiment



https://rachit-dubey.github.io/humanRL_website/

AI: Representation and Problem Solving

Reinforcement Learning II



Instructor: Pat Virtue

Slide credits: CMU AI and http://ai.berkeley.edu

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

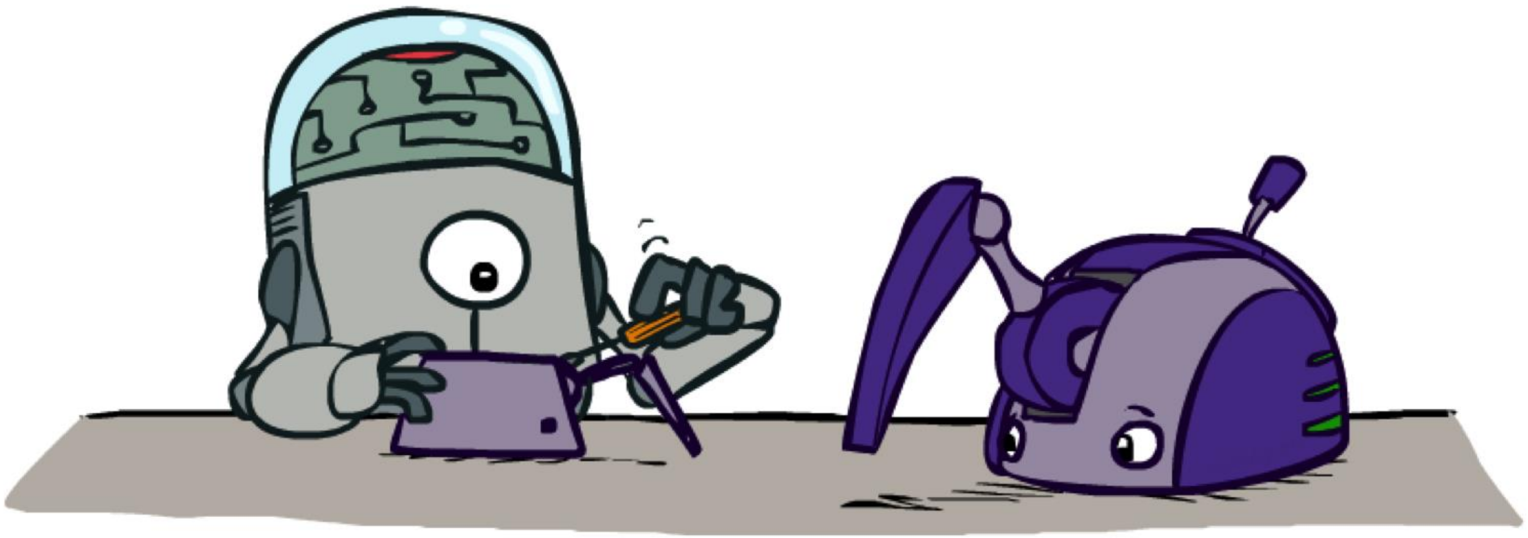
Model-Free

Passive Reinforcement Learning

- Direct Evaluation (simple)
- TD Learning

Active Reinforcement Learning

- Q-Learning



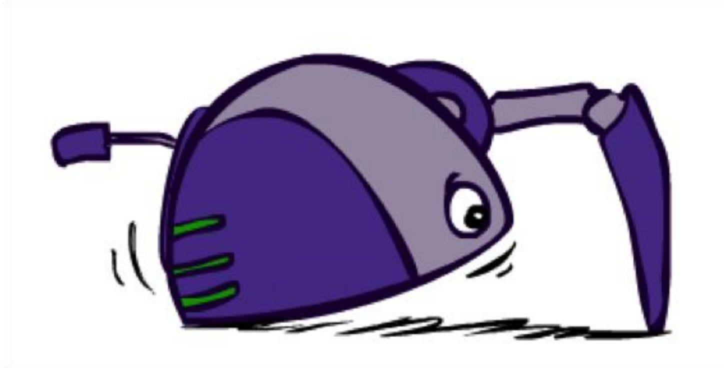
Online Learning

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')

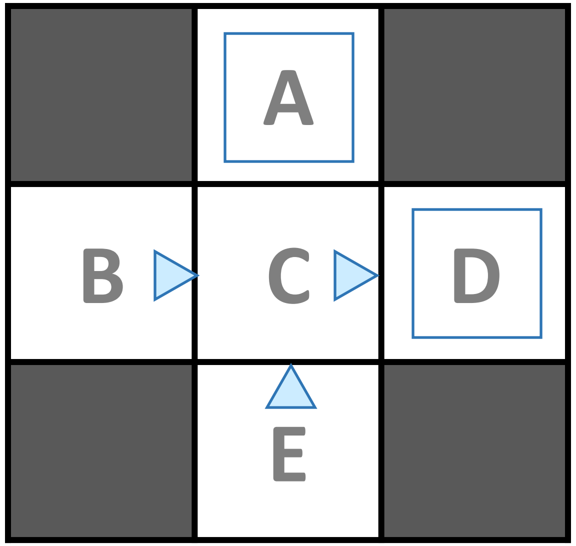
Step 2: Solve the learned MDP

- For example, use value iteration, as before

Example: Model-Based Learning

Episode: a sequence of states actions and rewards sampled from the environment

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

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

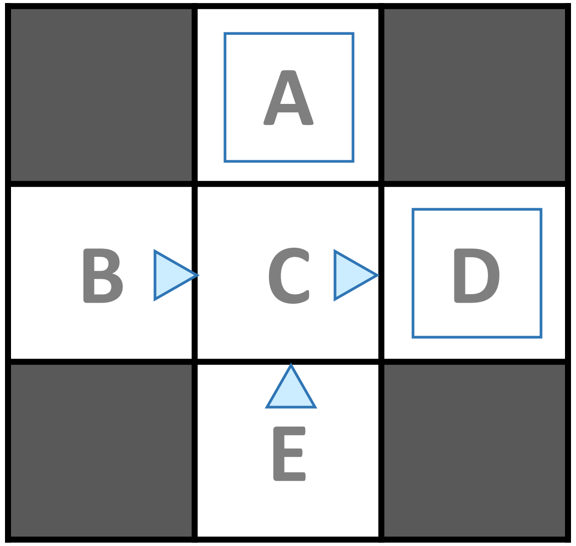
T(B, east, C) =
T(C, east, D) =
T(C, east, A) =
…

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

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

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

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 15-281 students

Known P(A)

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

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

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

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

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.

Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

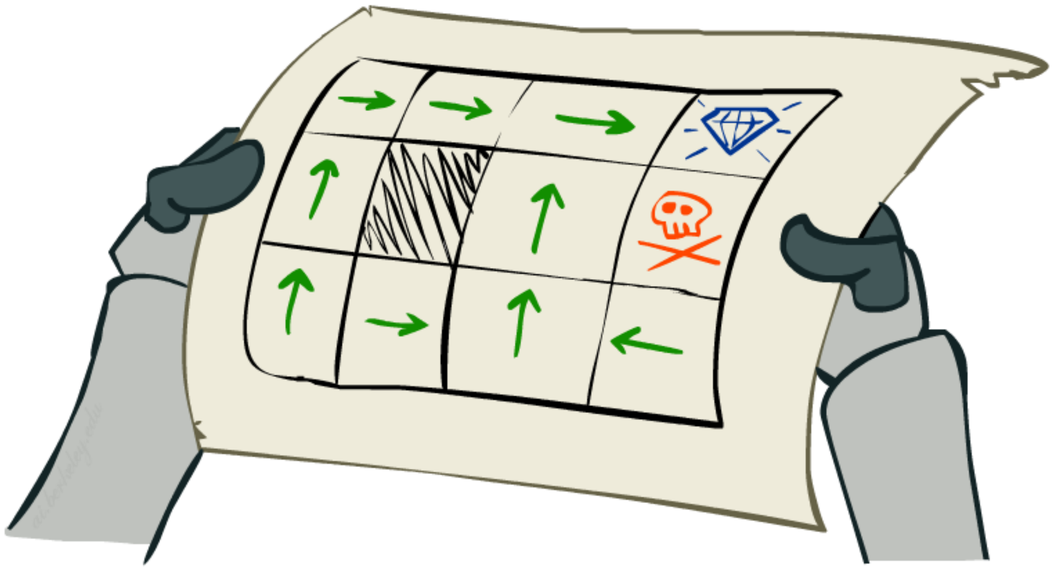
Model-Based

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

Model-Free

Passive Reinforcement Learning
- Direct Evaluation (simple)
- TD Learning

Active Reinforcement Learning
- Q-Learning

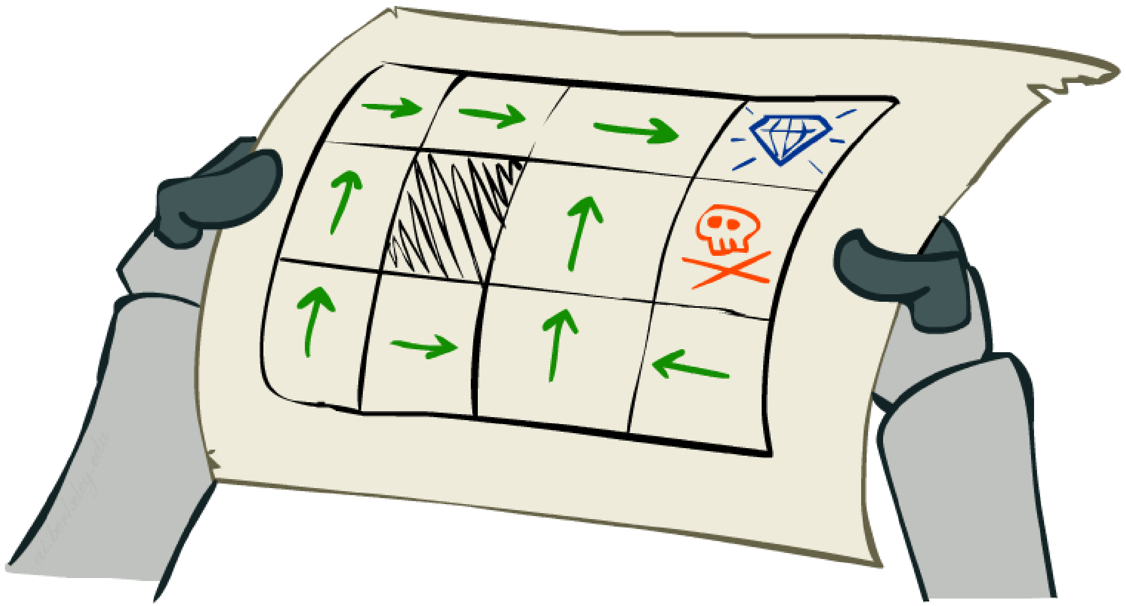


Online Learning

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

Simple Passive Learning: Direct Evaluation

Goal: Compute values for each state under π

Idea: Average together observed sample values

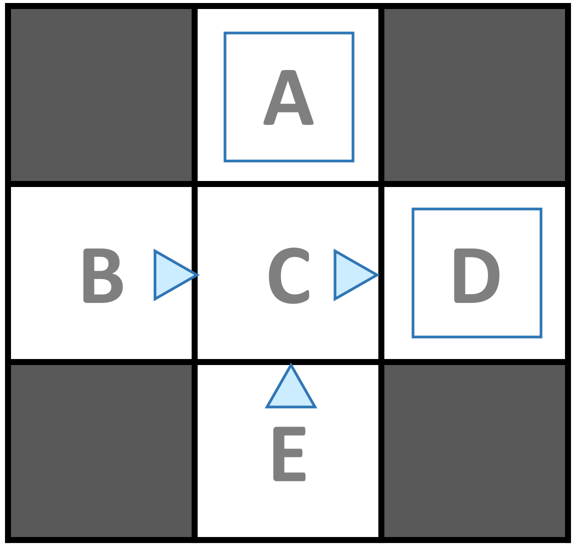
- Act according to π
- Every time you visit a state, write down what the sum of discounted rewards turned out to be
- Average those samples

This is called direct evaluation

Pieces Available	Take 1	Take 2
2	0	**100**
3	2	0
4	**75**	2
5	4	**68**
6	5	6
7	**60**	5

Example: Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

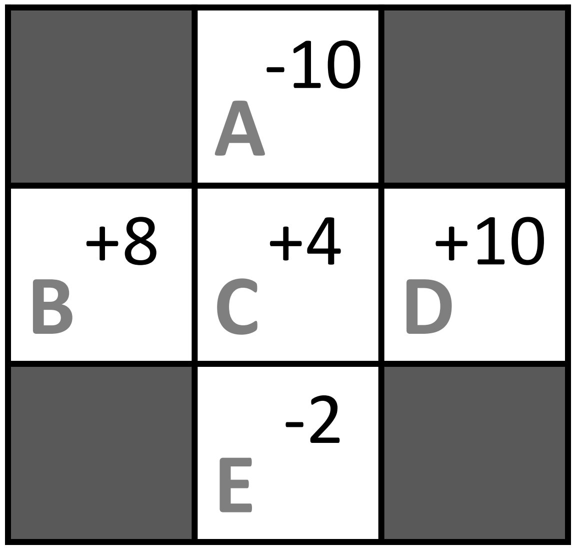
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Problems with Direct Evaluation

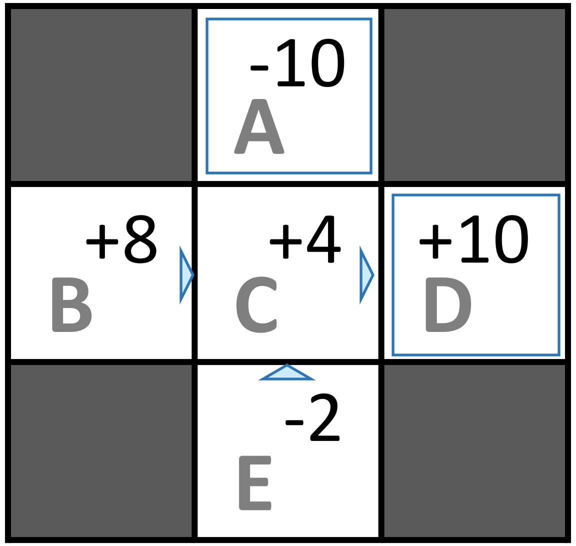
What's good about direct evaluation?

- It's easy to understand
- It doesn't require any knowledge of T, R
- It eventually computes the correct average values, using just sample transitions

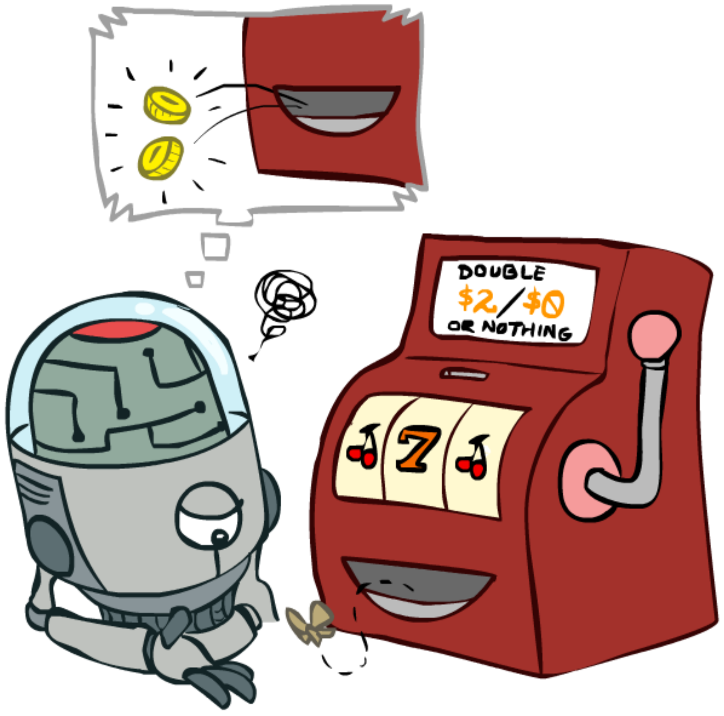
What bad about it?

- It wastes information about state connections
- Each state must be learned separately
- So, it takes a long time to learn

Output Values



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



Online Learning

Model-free Learning

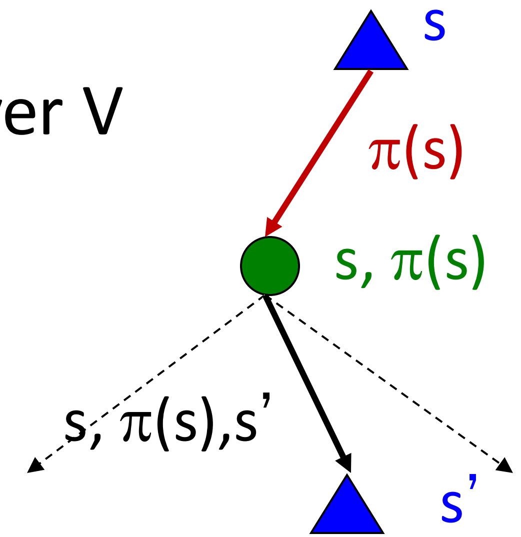
Passive Reinforcement Learning

Temporal Difference 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?

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

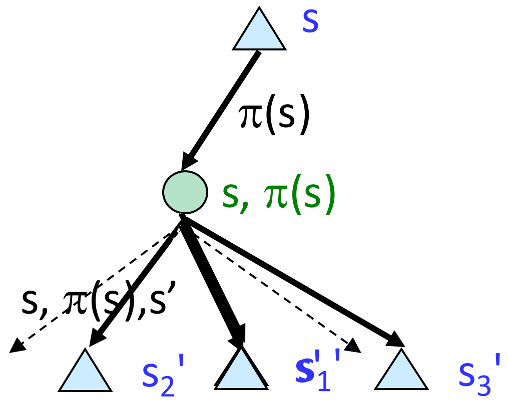
$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s, \pi(s), s_2') + \gamma V_k^{\pi}(s_2')$$

$$\ldots$$

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

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

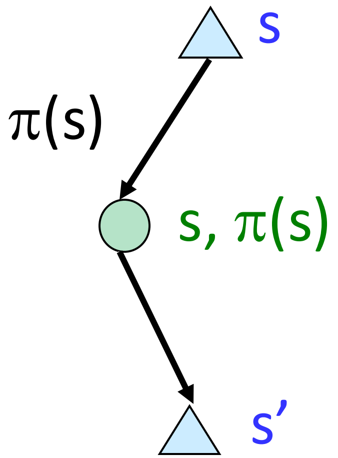


Almost! But we can't rewind time to get sample after sample from state s.

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

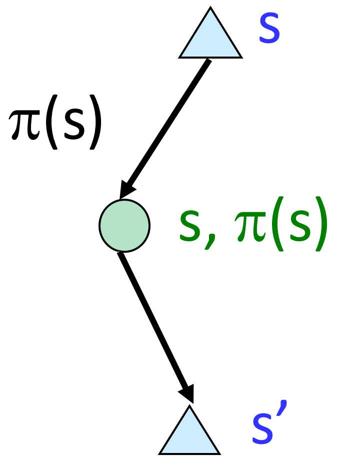
Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s):

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:

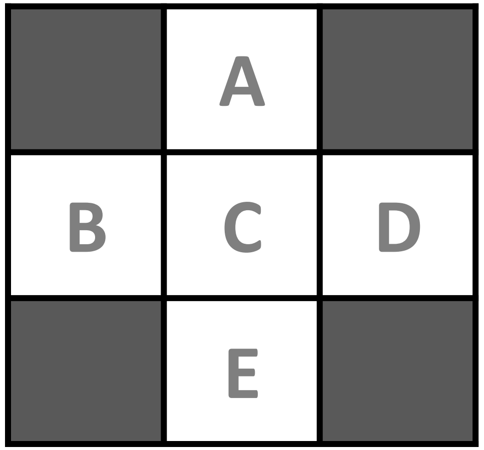
$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2\cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

- Forgets about the past (distant past values were wrong anyway)

Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

States

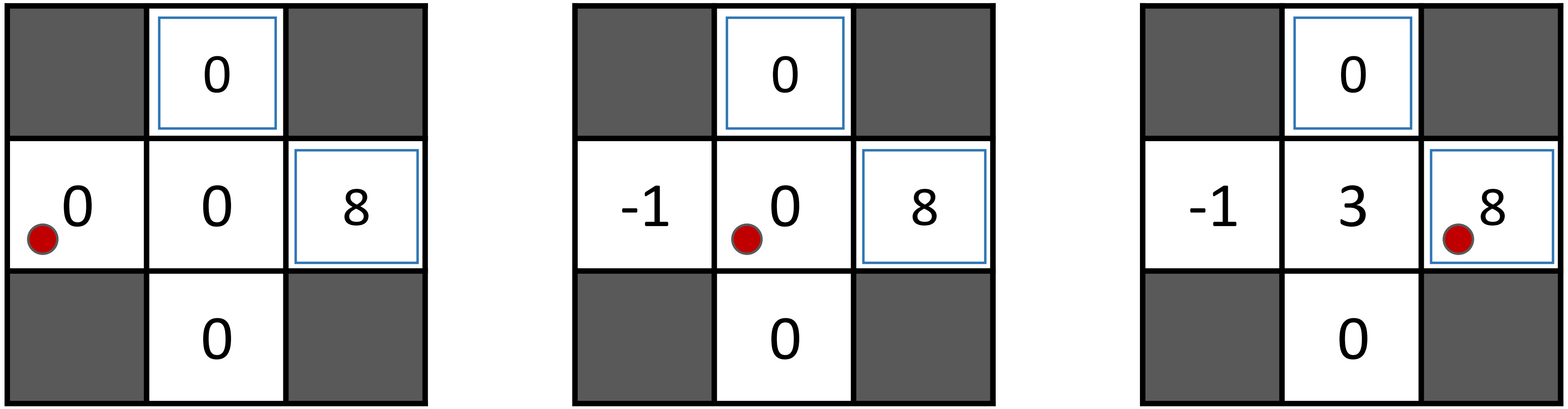


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

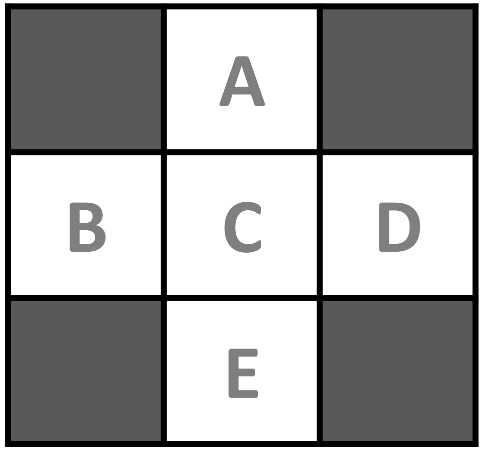
B, east, C, -2

C, east, D, -2



Example: Temporal Difference Learning

States

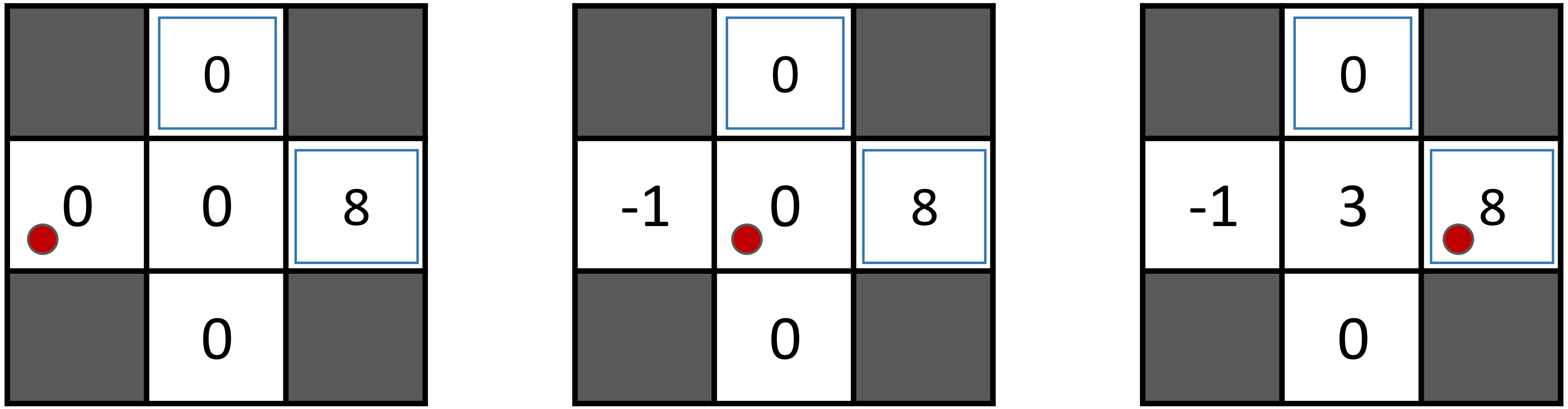


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, C, -2

C, east, D, -2



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

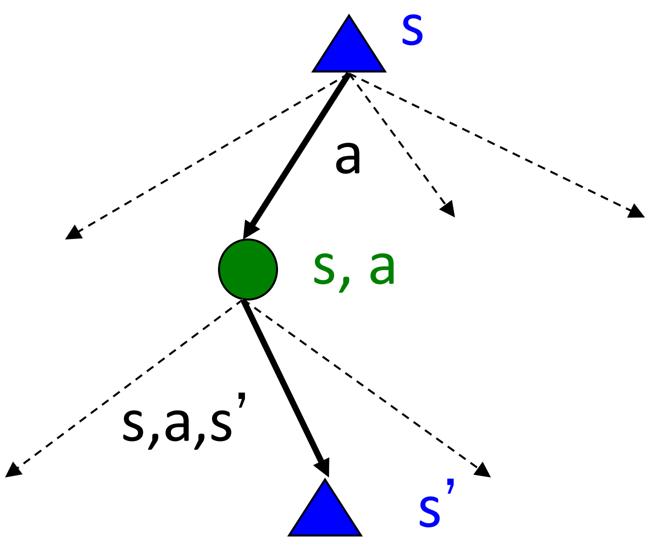
Problems with TD Value Learning

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!

Model-Free Learning

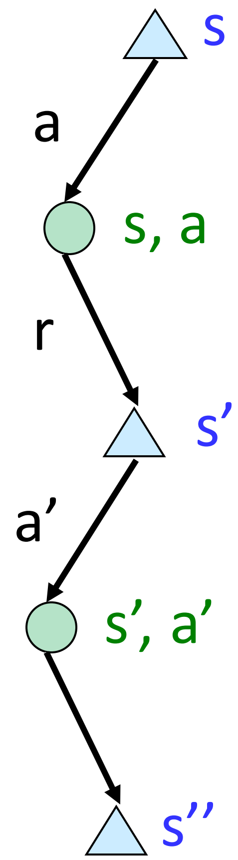
Model-free (temporal difference) learning

- Experience world through episodes

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

- Update estimates each transition (s, a, r, s')

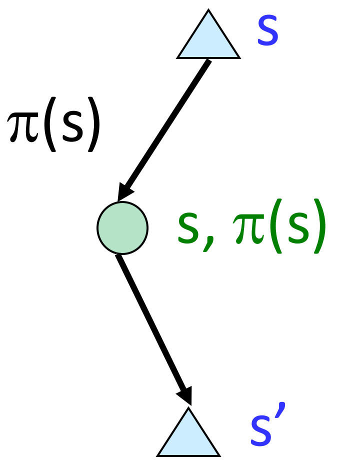
- Over time, updates will mimic Bellman updates



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

Quick Calculus Quiz

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

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

Gradient Descent

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

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

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

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

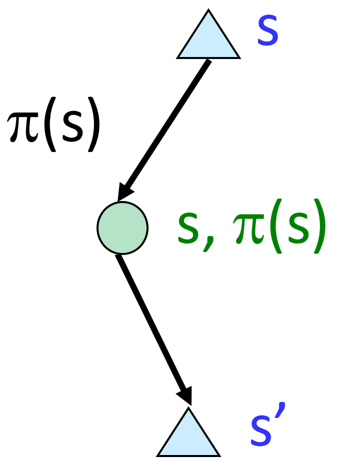
$$V \leftarrow V - \alpha \nabla_V Error$$

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

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

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

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

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

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

F) None of the above

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

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

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

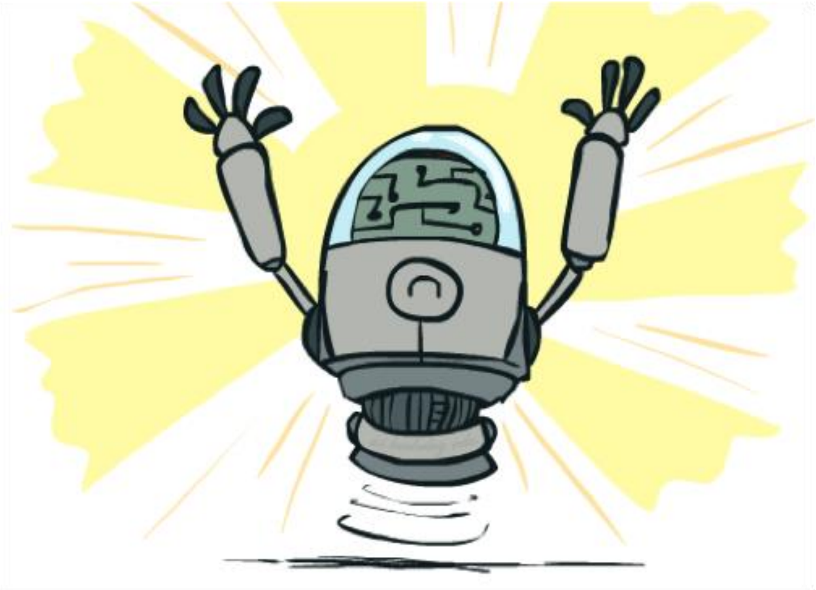
- But we want to average over results from (s,a) (Why?)
- So keep a running average

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

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

- Direct Evaluation (simple)
- TD Learning

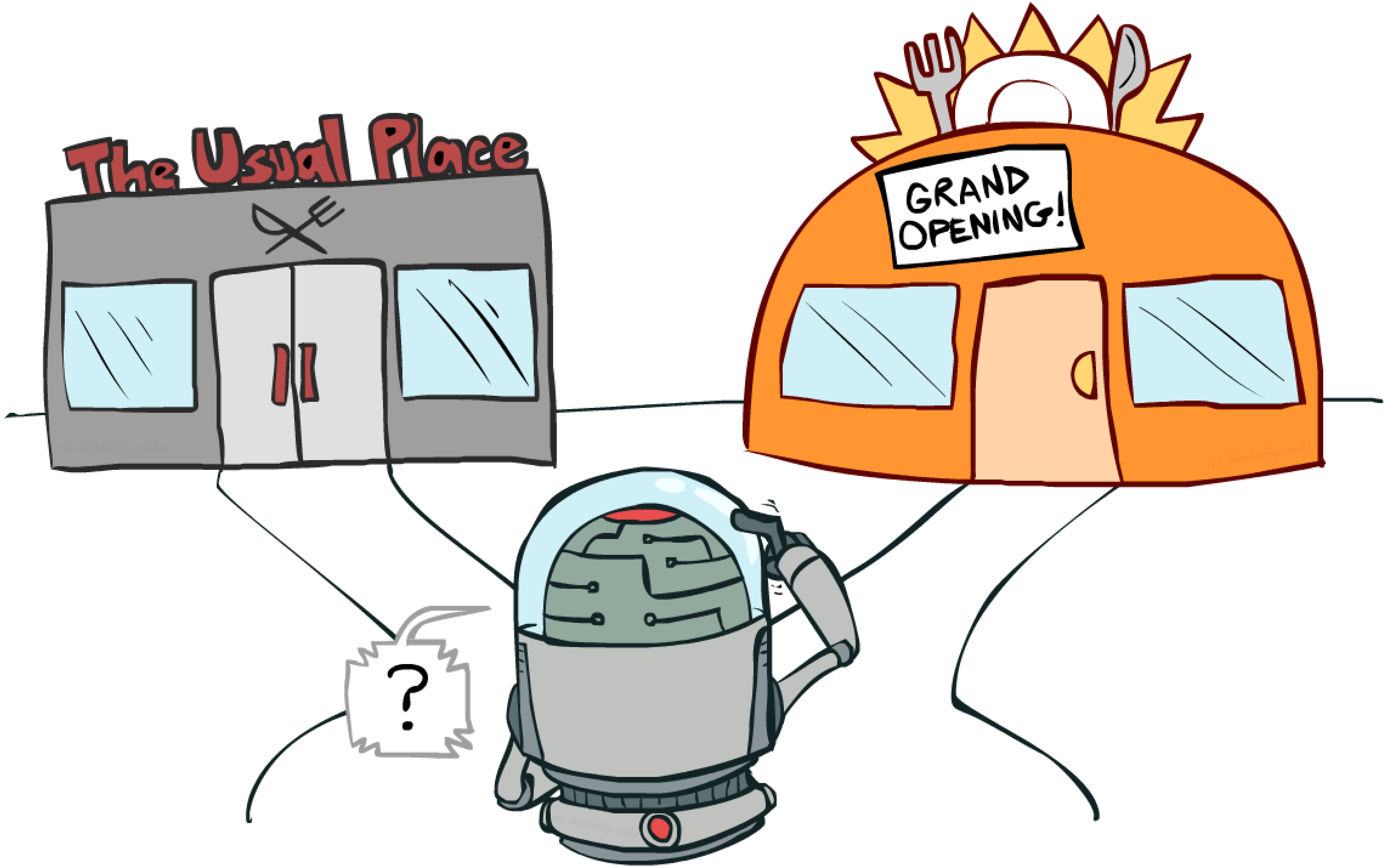
Active Reinforcement Learning

- Q-Learning

Demo Q-Learning Auto Cliff Grid

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

Exploration vs. Exploitation



How to Explore?

Several schemes for forcing exploration

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

- 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

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!

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

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!

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

Demo Q-learning – Exploration Function – Crawler

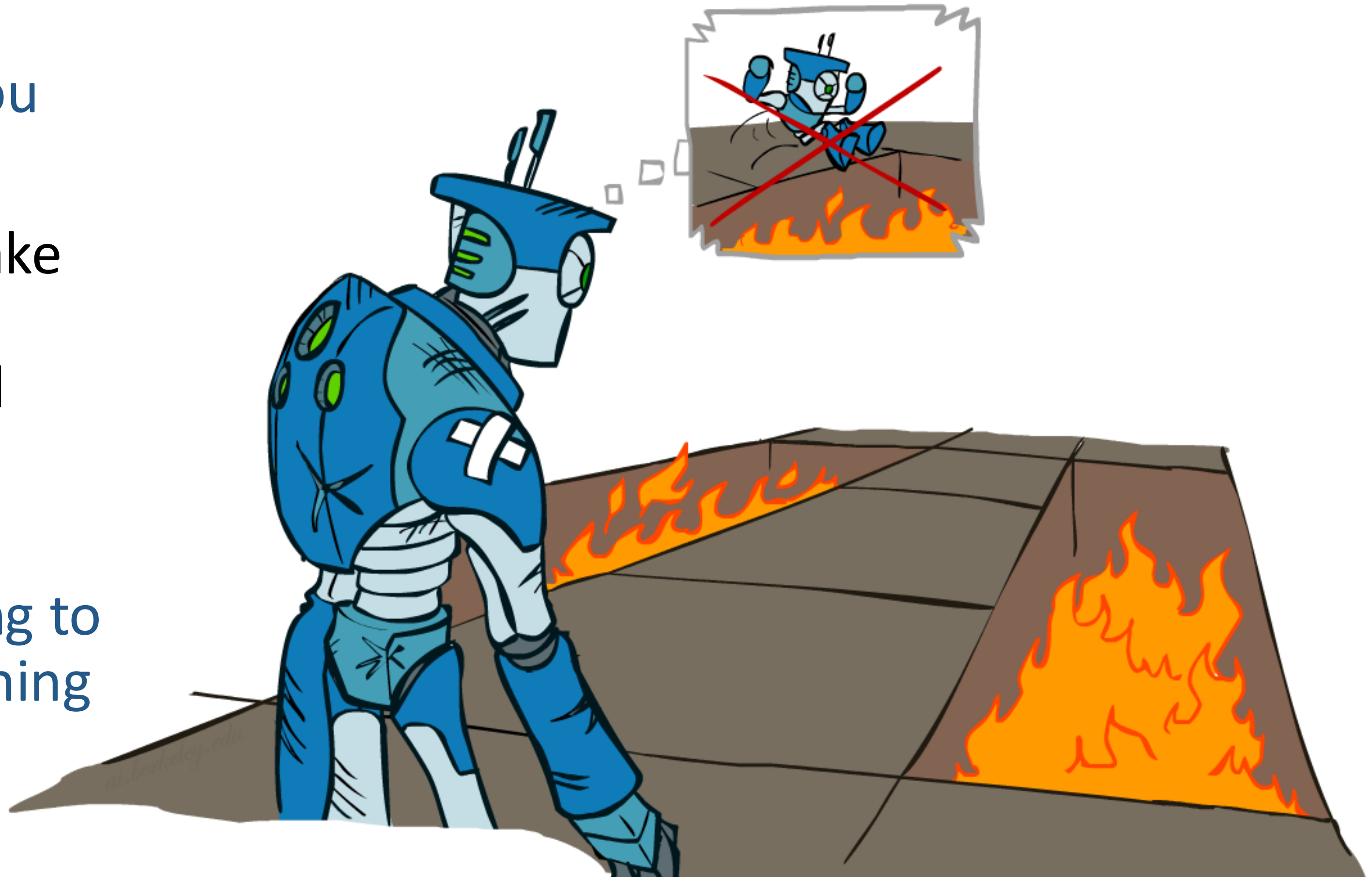
Regret

Even if you learn the optimal policy, you still make mistakes along the way!

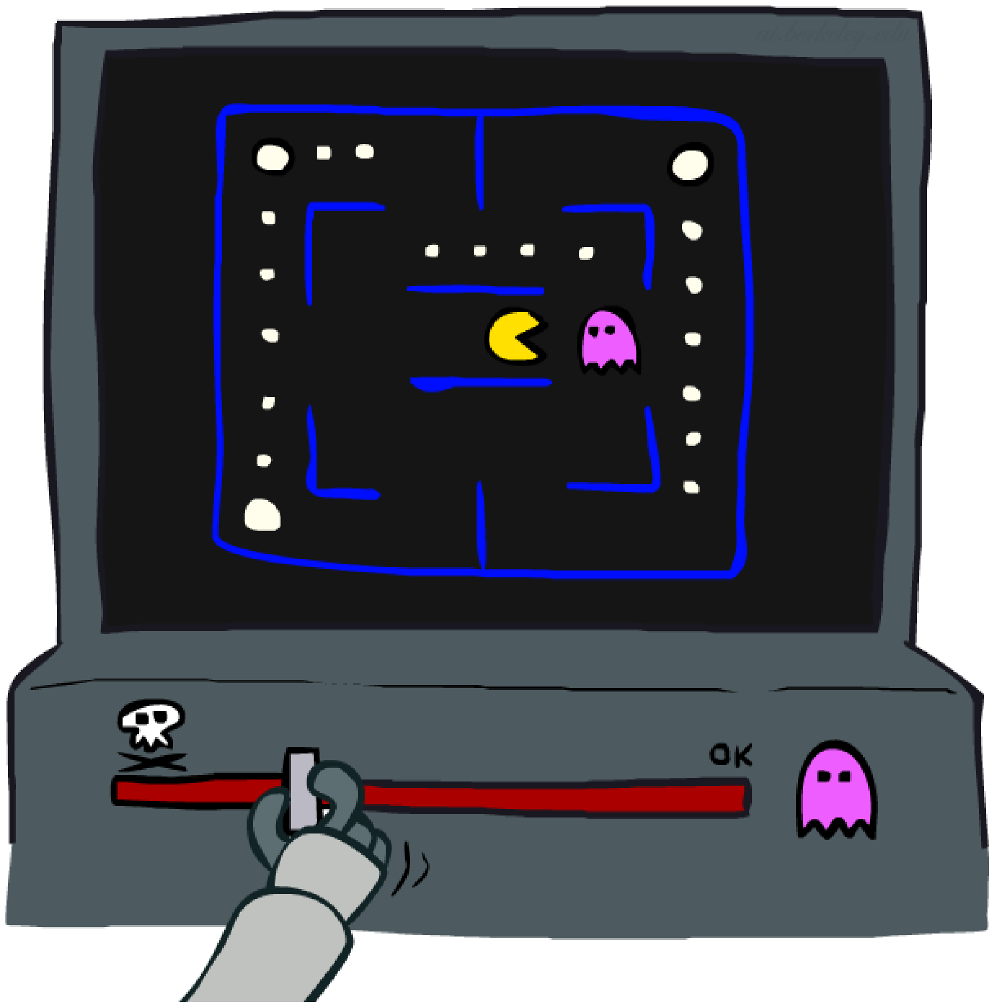
Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards

Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal

Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



Approximate Q-Learning



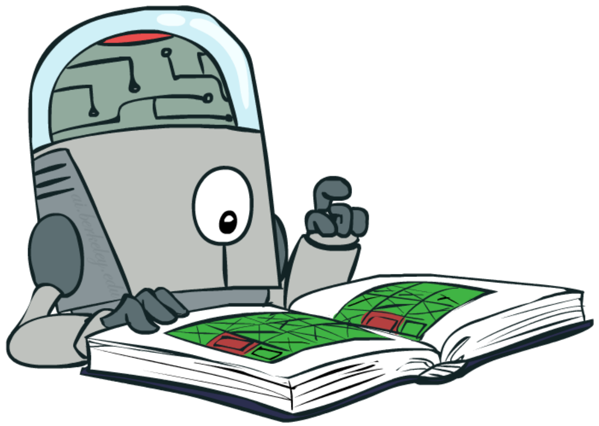
Approximate Q-Learning

What happens when we change Candy Grab to start with 1000 pieces?

Pieces Available	Take 1	Take 2
2	0%	**100%**
3	2%	0%
4	**75%**	2%
5	4%	**68%**
6	5%	6%
7	**60%**	5%

Generalizing Across States

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

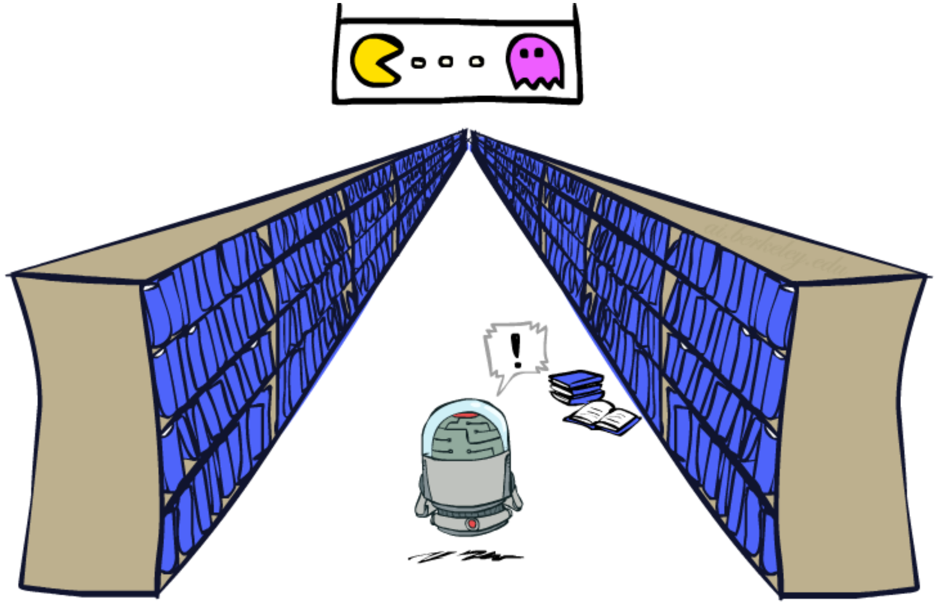


In realistic situations, we cannot possibly learn about every single state!

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

Instead, we want to generalize:

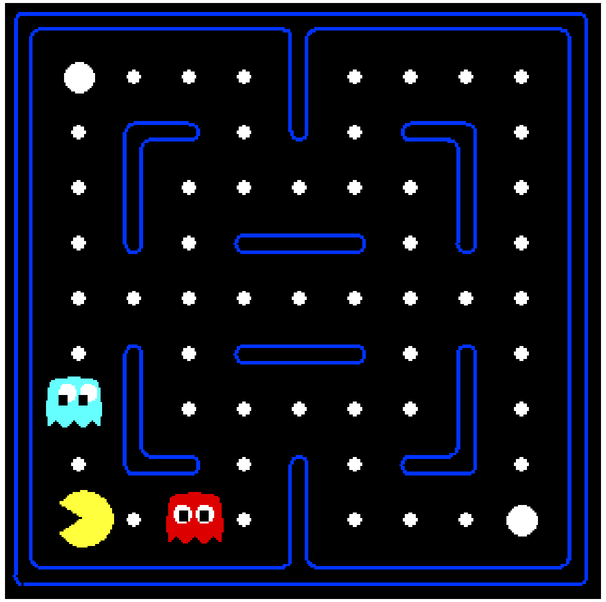
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
- This is a fundamental idea in machine learning, and we'll see it over and over again



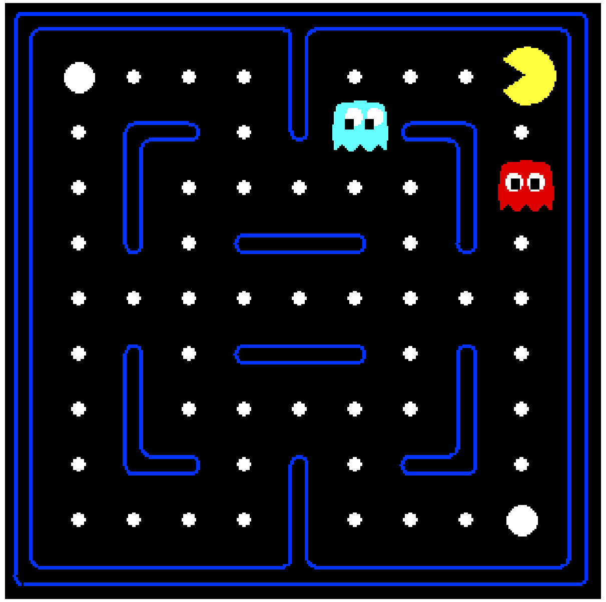
[demo – RL pacman]

Example: Pacman

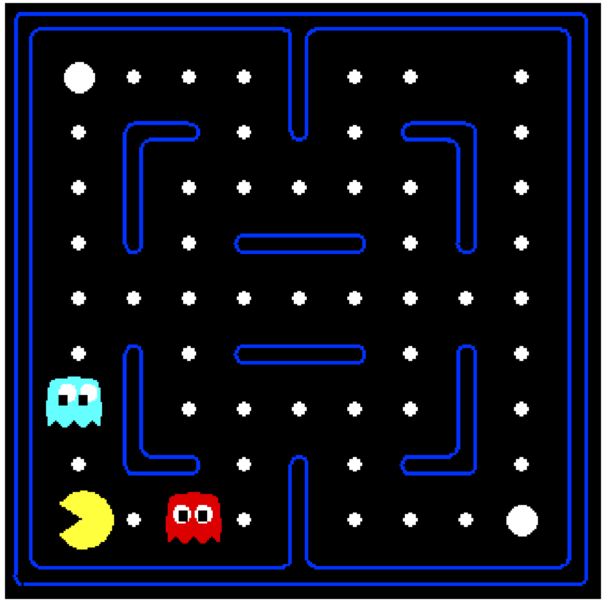
Let's say we discover through experience that this state is bad:



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



Or even this one!

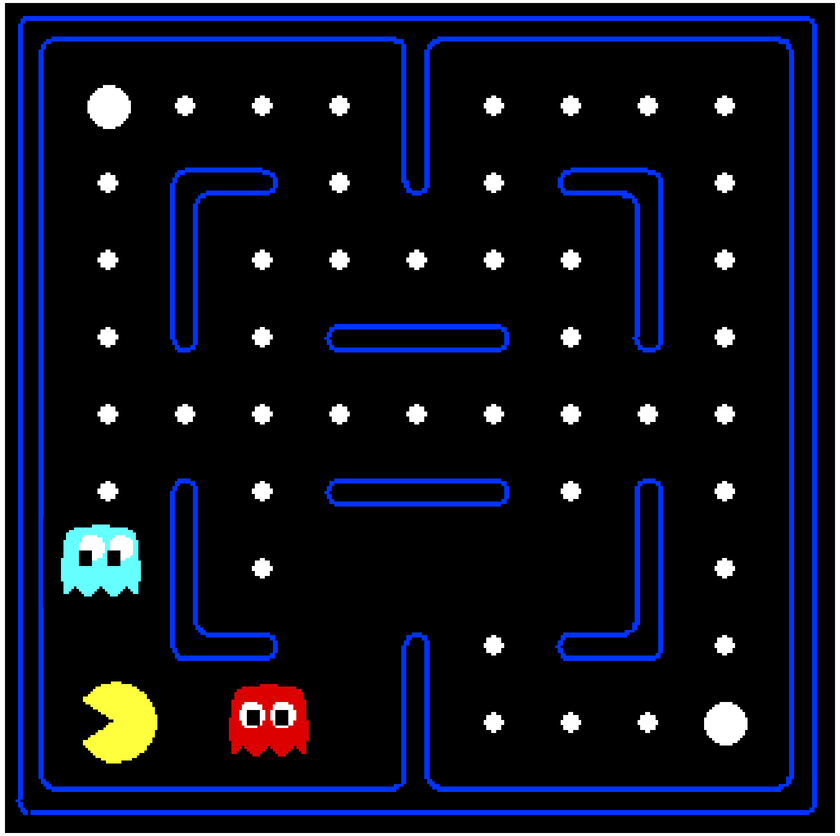


[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

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
  - $1 / (\text{dist to dot})^2$
  - 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:

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

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

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

Updating a linear value function

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

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

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

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

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

Quick Calculus Quiz

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

What is $\frac{dError}{dw}$?

Last time

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

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

Updating a linear value function

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

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

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

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

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

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

$$Error(w) = \frac{1}{2}\big(y - wf(x)\big)^2$$

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

$$\frac{dError}{dw} = -\big(y - wf(x)\big)f(x)$$

Updating a linear value function

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

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

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

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

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

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

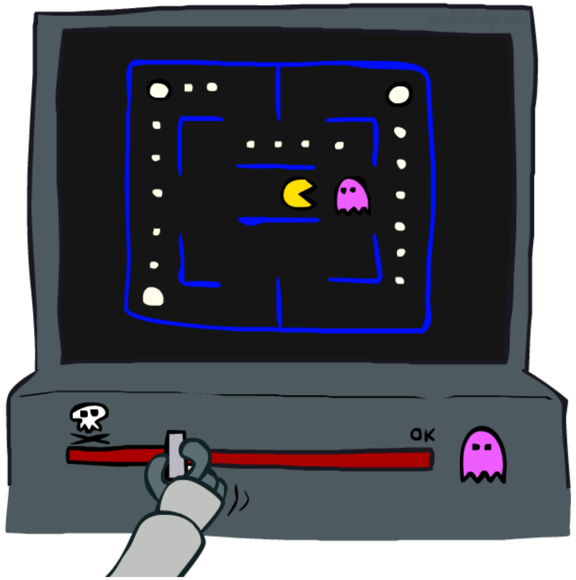
Q-learning with linear Q-functions:

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

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

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

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



Intuitive interpretation:

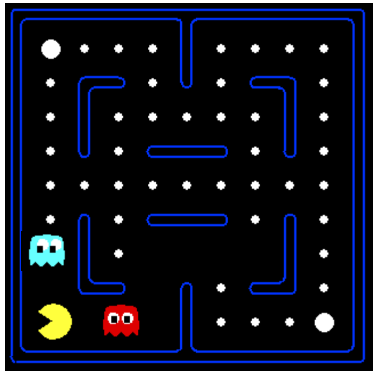
- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

Formal justification: online least squares

Example: Q-Pacman

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

s

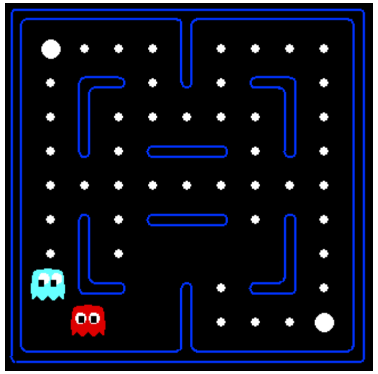


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

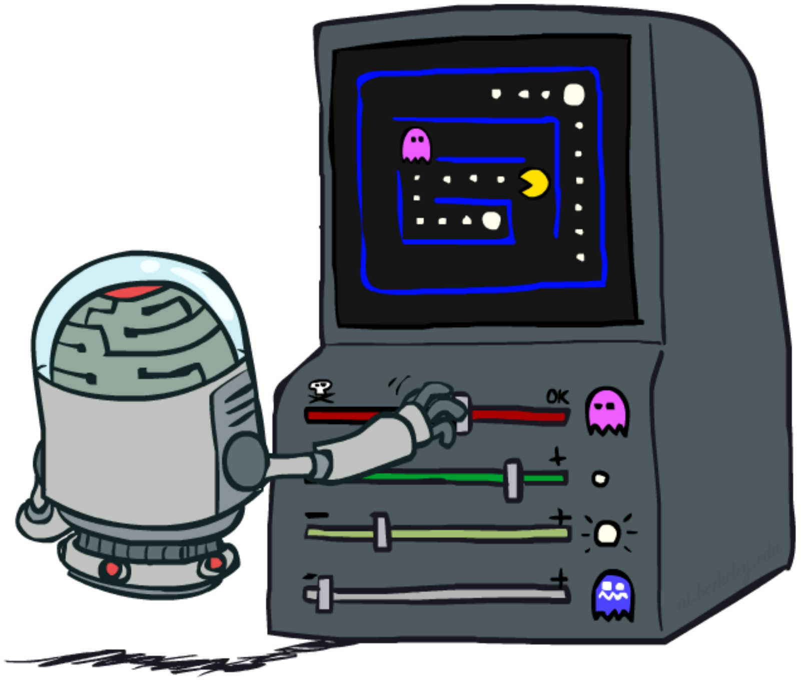
difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

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

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

Recent Reinforcement Learning Milestones



TDGammon

1992 by Gerald Tesauro, IBM

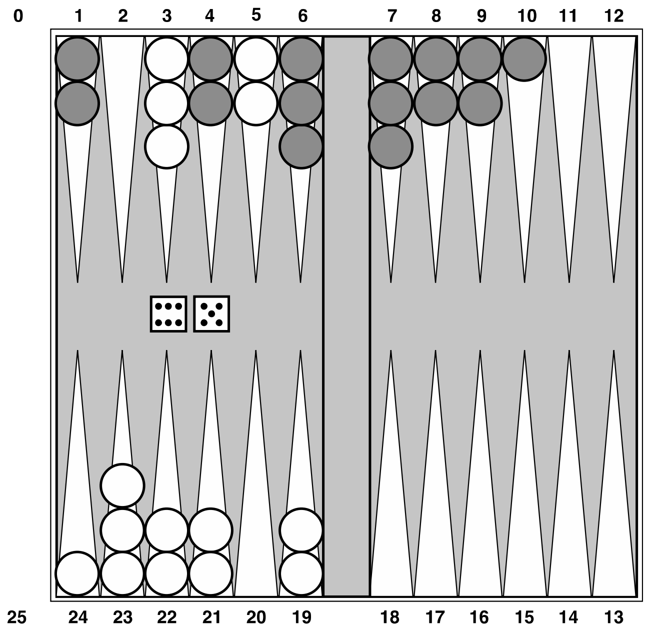
4-ply lookahead using V(s) trained from 1,500,000 games of self-play

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

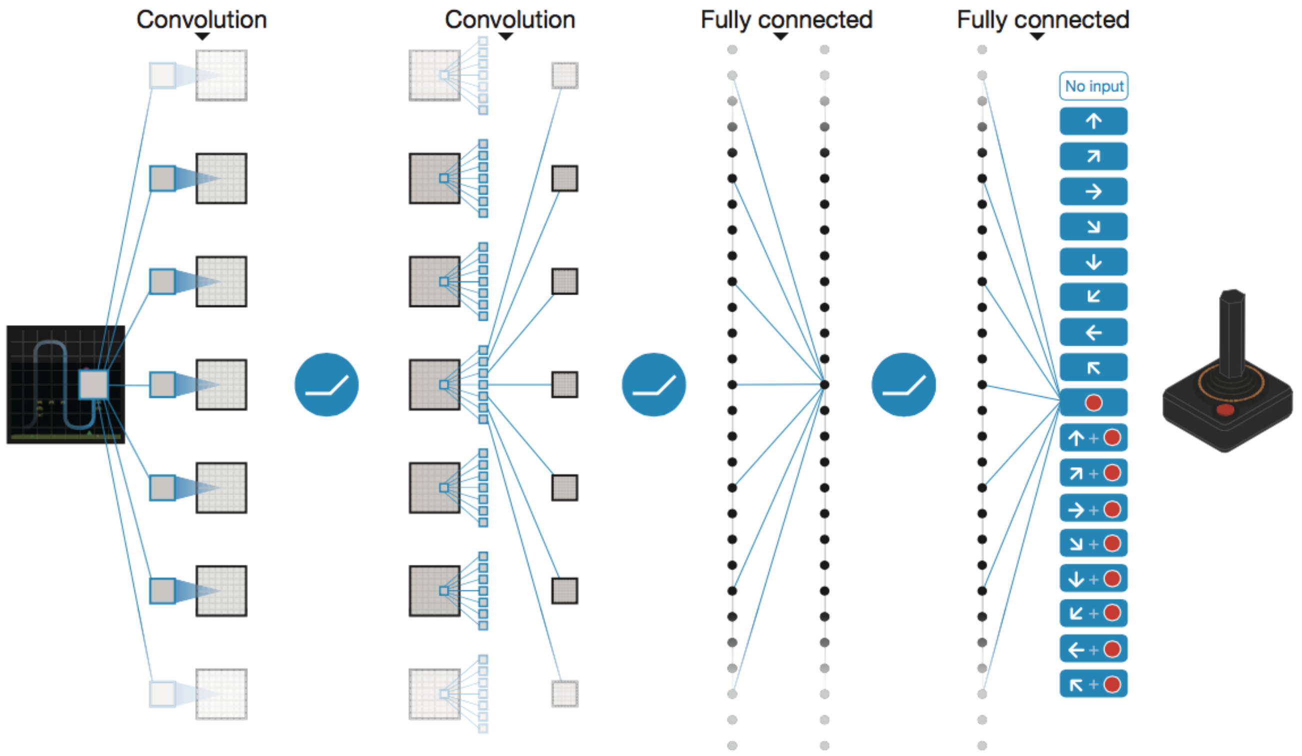
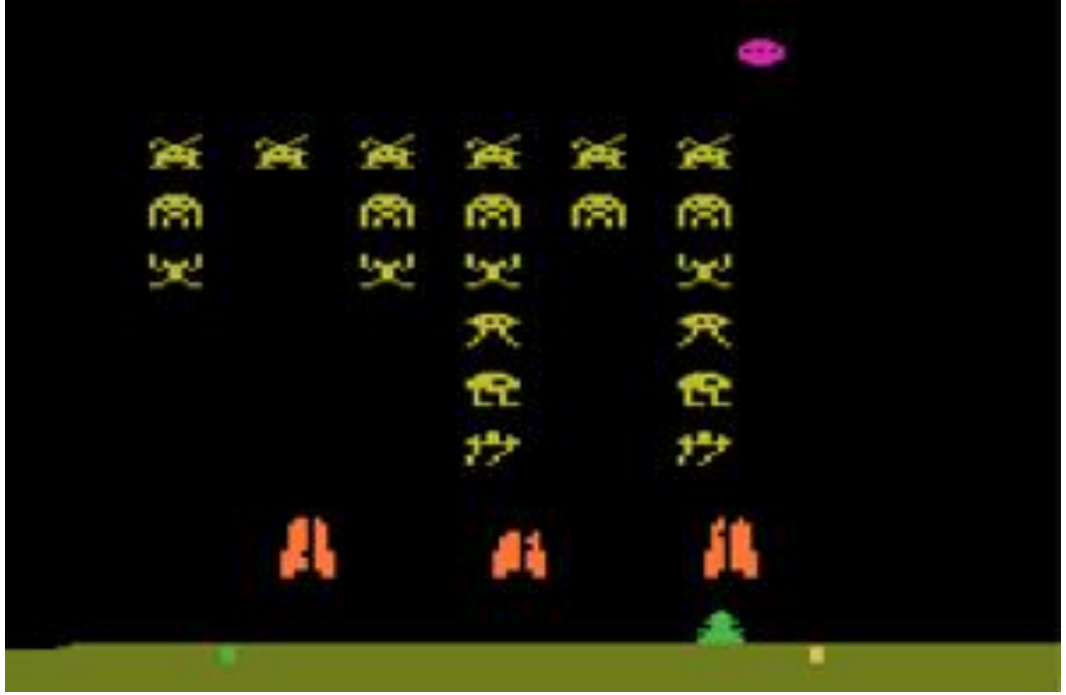


Image: Deep Mind



Images: Open AI, Atari

OpenAI Gym

2016+

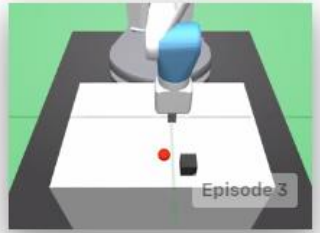
Benchmark problems for learning agents

https://gym.openai.com/envs

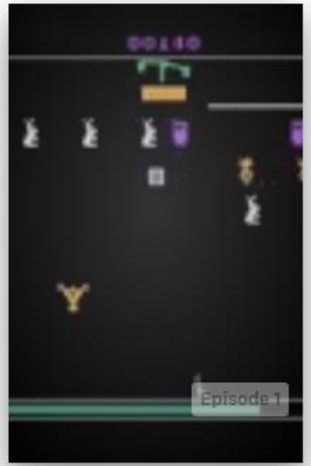
Breakout-ram-v0
Maximize score in the game Breakout, with RAM as input

Acrobot-v1
Swing up a two-link robot.

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



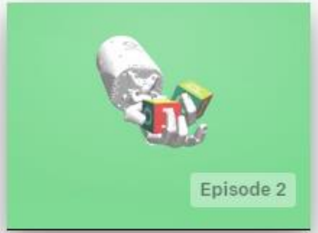
FetchPush-v0
Push a block to a goal position.



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?