Announcements

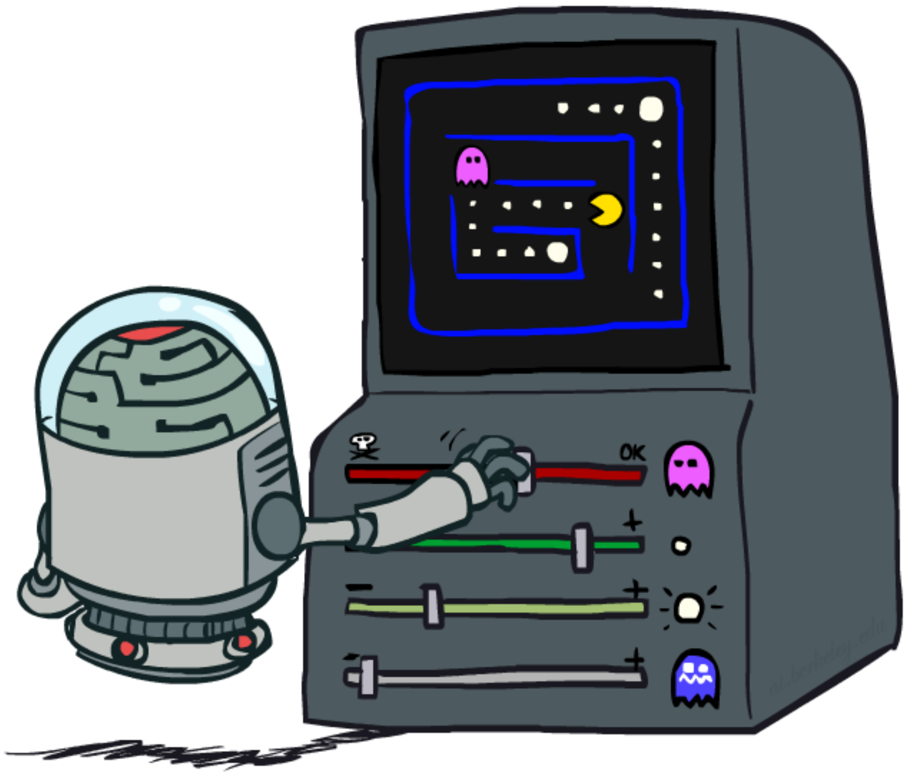
- HW 10 due **Tuesday, April 23, 11:59 PM PT**
- Project 6 due **Friday, April 26, 11:59 PM PT**

- **Course evaluations are live!**
  - Log in at course-evaluations.berkeley.edu
  - Current response rate: 3%
  - Target response rate: 100%

Pre-scan attendance QR code now!

CS 188: Artificial Intelligence

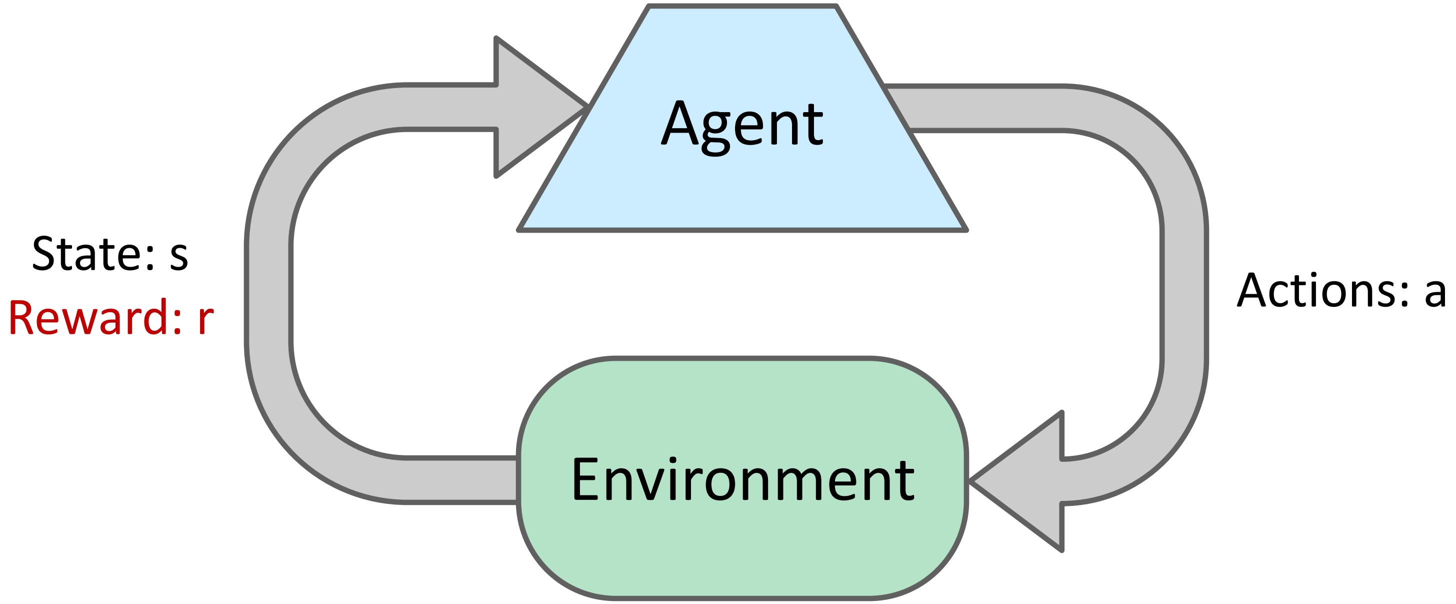
Reinforcement Learning – Part 2



University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Recap: Reinforcement Learning



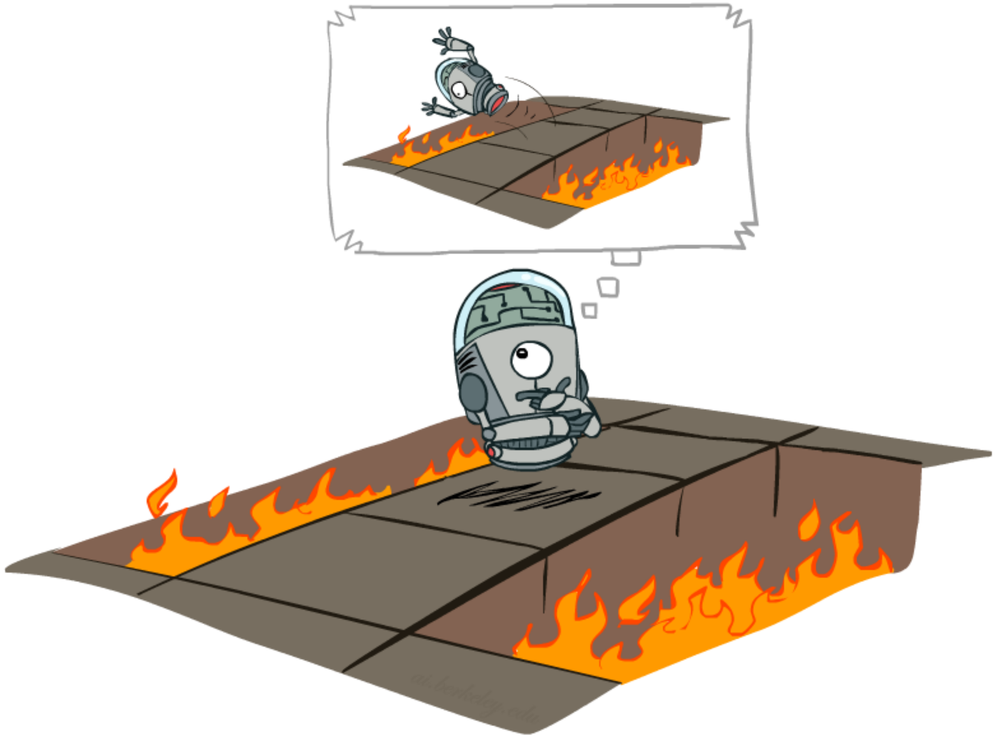
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Recap: Reinforcement Learning

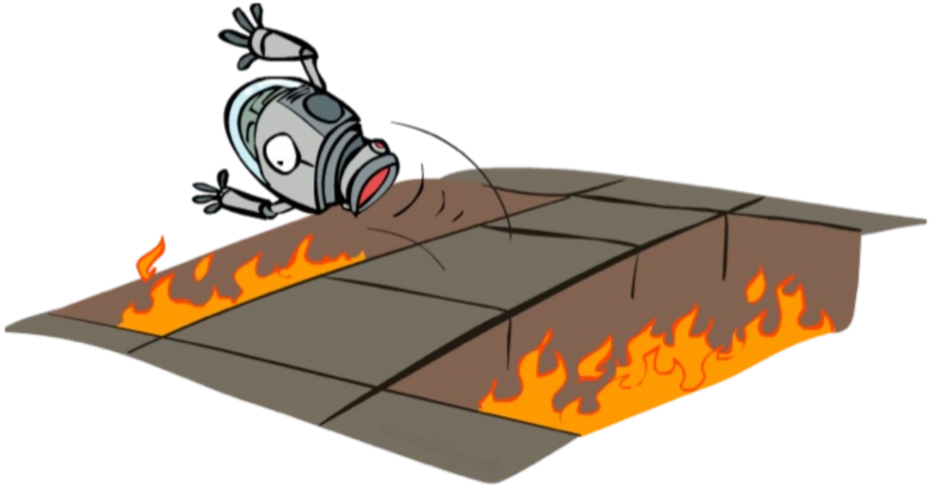
- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) $A(s)$
  - A transition model $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must explore new states and actions to discover how the world works

Recap: Offline (MDPs) vs. Online (RL)

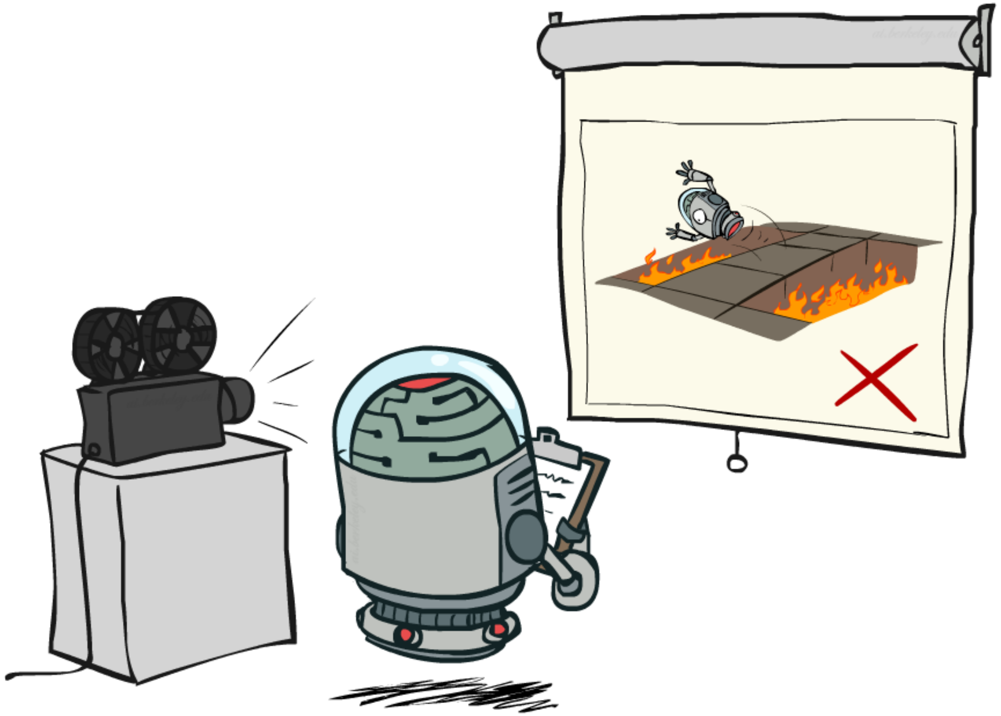


Offline/Planning

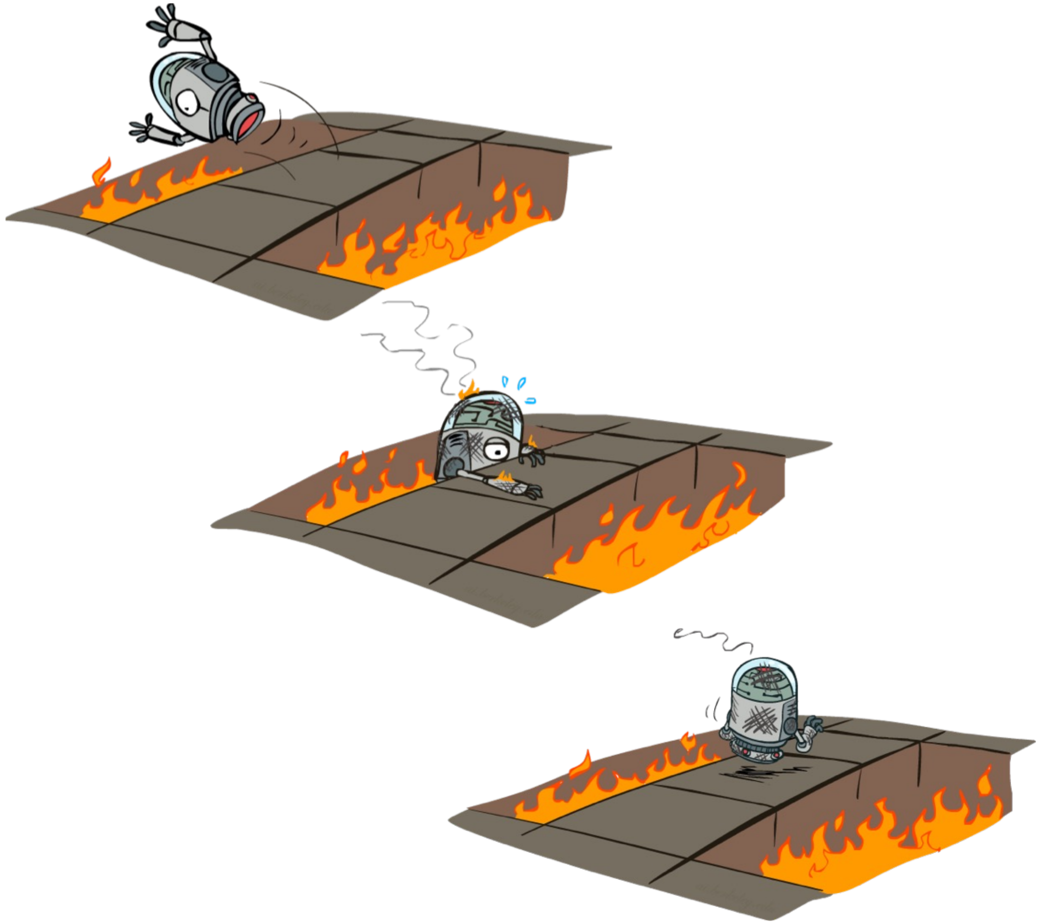


Online/Learning

Recap: Passive vs Active RL



Passive (fixed π)



Active (changing π)

Approaches to reinforcement learning

✅ 1. Model-based: Learn the model, solve it, execute the solution

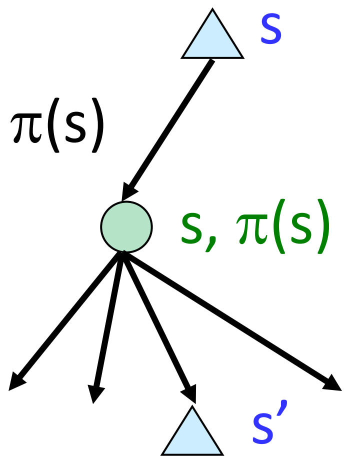
2. Learn values from experiences, use to make decisions
   ✅ a. Direct evaluation
   b. Temporal difference learning
   c. Q-learning

3. Optimize the policy directly

Temporal Difference Learning

- Passive setting (fixed policy π), like policy evaluation:

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



- Modifications:

  1. Don't know T or R; estimate expectation from samples!

     $V^{\pi}(s) = \frac{1}{N}\sum_{i}[\,r_i + \gamma V^{\pi}(s_i')\,]$

  2. Update V(s) after each transition (s,a,s',r) using running average.

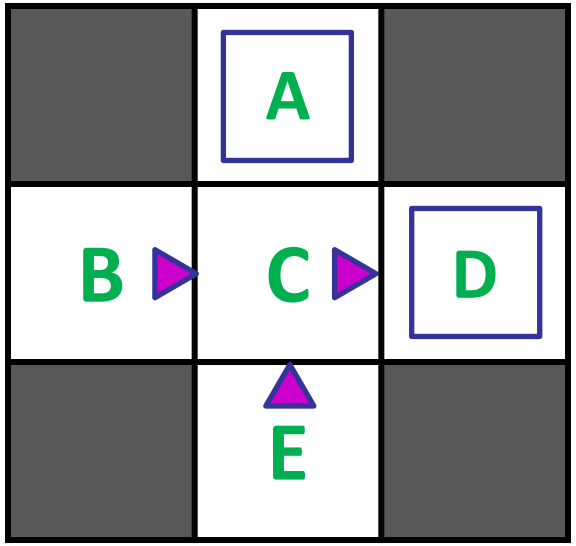
  3. Decay older samples as new ones come in.

Example: TD Value Estimation

- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^\pi(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^\pi(s_i')\right) - V^\pi(s_i)$.
- Update: $V^\pi(s_i)$ += $\alpha_i \cdot \delta_i$.

Example: TD Value Estimation

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

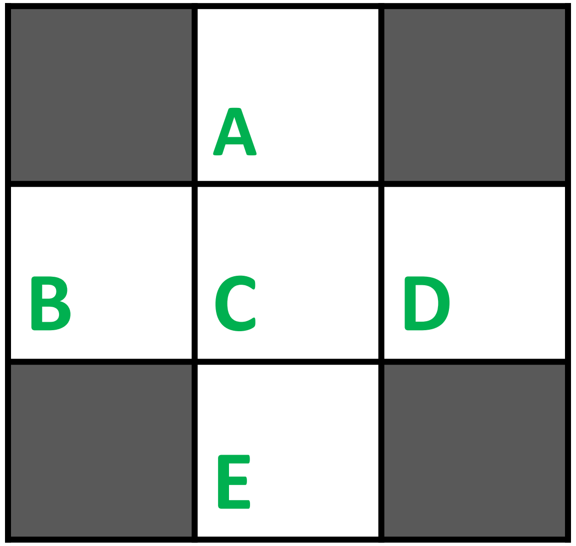
Episode 3

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



Example: TD Value Estimation

- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^\pi(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^\pi(s_i')\right) - V^\pi(s_i)$.
- Update: $V^\pi(s_i)$ += $\alpha_i \cdot \delta_i$.

s	*V(s)*
A	
B	
C	
D	
E	

i	*s*	*a*	*s'*	*r*	$r + \gamma V^\pi(s')$	$V^\pi(s)$	δ
1							
2							
3							
4							
5							
6							
7							

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

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

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

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

Example: TD Value Estimation

- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^\pi(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^\pi(s_i')\right) - V^\pi(s_i)$.
- Update: $V^\pi(s_i) \mathrel{+}= \alpha_i \cdot \delta_i$.

s	***V(s)***
A	0
B	-1
C	9
D	10
E	8

i	***s***	***a***	***s'***	***r***	$r + \gamma V^\pi(s')$	$V^\pi(s)$	δ
1	B	east	C	-1	-1 + 0	0	-1
2	C	east	D	-1	-1 + 0	0	-1
3	D	exit	---	10	10 + 0	0	+10
4	B	east	C	-1	-1 + -1	-1	-1
5	C	east	D	-1	-1 + 10	-1	+9
6	D	exit	---	10	10 + 0	10	0
7	E	north	C	-1	-1 + 9	0	+8

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

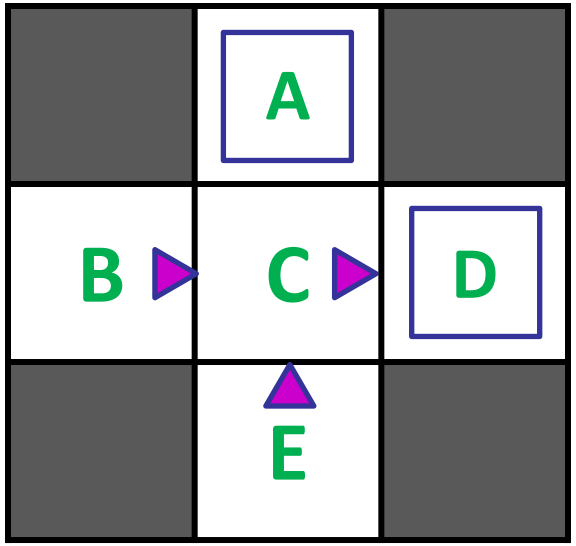
B, east, **C**, -1
C, east, **D**, -1
D, exit, **x**, +10

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

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

Example: TD Value Estimation

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

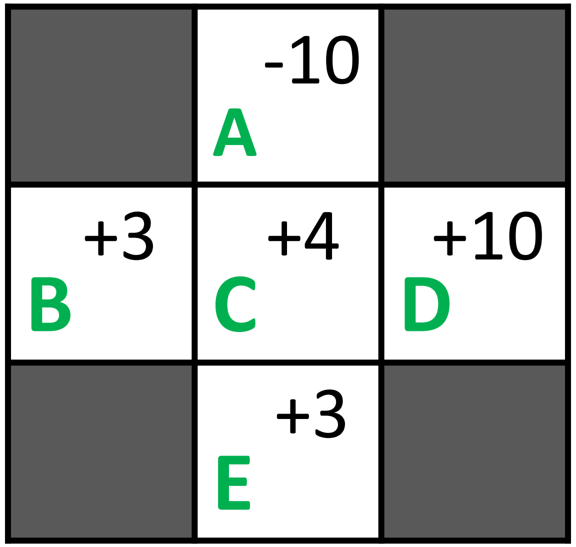
Episode 3

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



Temporal Difference Learning

- Passive setting (fixed policy π), like policy evaluation:

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

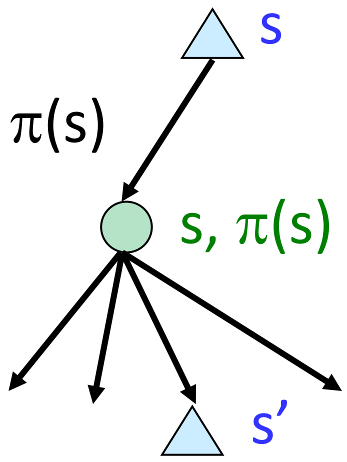
- Modifications:

  1. Don't know T or R; estimate expectation from samples!

     $V^{\pi}(s) = \frac{1}{N} \sum_i [r_i + \gamma V^{\pi}(s_i')]$

  2. Update V(s) after each transition (s,a,s',r) using running average.

  3. Decay older samples as new ones come in.



Running averages

- How do you compute the average of 1, 4, 7?
- Method 1: add them up and divide by N
  - 1+4+7 = 12
  - average = 12/N = 12/3 = 4

- Method 2: keep a running sum, or running mean μ_n, and a running count, n:

$$\mu_n = (\text{sum}_{n-1} + x_n) / \text{count}_n = ((n-1) \cdot \mu_{n-1} + x_n)/n$$

  - n=0 $\mu_0=0$
  - n=1 $\mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1$
  - n=2 $\mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5$
  - n=3 $\mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4$

- Alternate formula:

$$\mu_n = [(n-1)/n]\, \mu_{n-1} + [1/n]\, x_n$$ (weighted average of old mean, new sample)

Running averages contd.

- What if we use a weighted average with a fixed weight?
  - $\mu_n = (1-\alpha)\, \mu_{n-1} + \alpha\, x_n$
  - n=1 $\quad \mu_1 = x_1$
  - n=2 $\quad \mu_2 = (1-\alpha) \cdot \mu_1 + \alpha x_2 = (1-\alpha) \cdot x_1 + \alpha x_2$
  - n=3 $\quad \mu_3 = (1-\alpha) \cdot \mu_2 + \alpha x_3 = (1-\alpha)^2 \cdot x_1 + \alpha(1-\alpha) x_2 + \alpha x_3$
  - n=4 $\quad \mu_4 = (1-\alpha) \cdot \mu_3 + \alpha x_4 = (1-\alpha)^3 \cdot x_1 + \alpha(1-\alpha)^2 x_2 + \alpha(1-\alpha) x_3 + \alpha x_4$
- I.e., ***exponential forgetting*** of old values
- μ_n is a convex combination of sample values (weights sum to 1)
- $E[\mu_n]$ is a convex combination of $E[X_i]$ values, hence unbiased

TD as approximate Bellman update

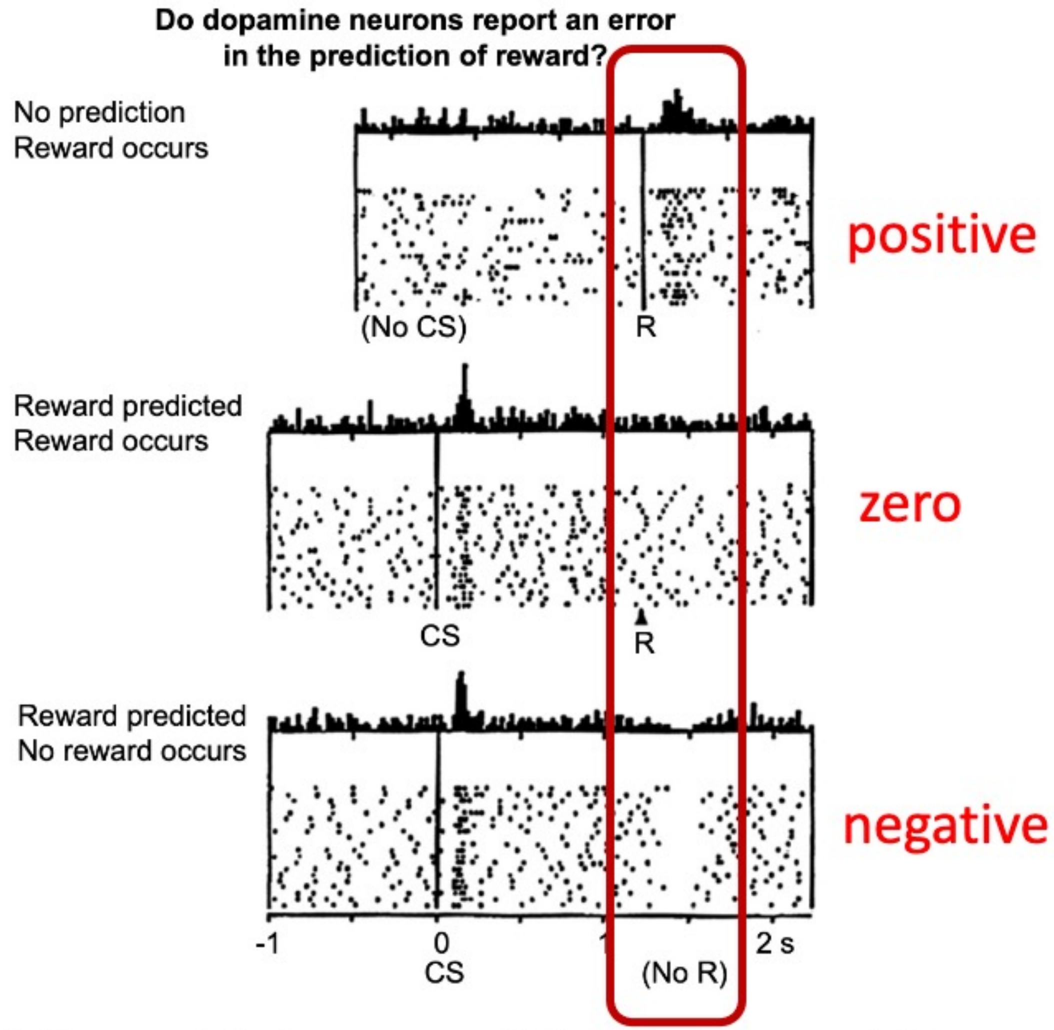
- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^\pi(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^\pi(s_i')\right) - V^\pi(s_i)$.
- Update with TD learning rule:
  - $V^\pi(s_i) \leftarrow V^\pi(s_i) + \alpha \cdot \delta_i$.
  - $V^\pi(s) \leftarrow V^\pi(s) + \alpha \cdot [\text{target} - V^\pi(s)]$
  - $V^\pi(s) \leftarrow (1-\alpha) \cdot V^\pi(s) + \alpha \cdot \text{target}$
  - α is the ***learning rate***
- Observe a sample, move $V^\pi(s)$ a little bit to make it more consistent with its neighbor $V^\pi(s')$

TD Learning Happens in the Brain!

- Neurons transmit *Dopamine* to encode reward or value prediction error:

  $\delta_i = \left(r_i + \gamma V^\pi(s_i')\right) - V^\pi(s_i).$

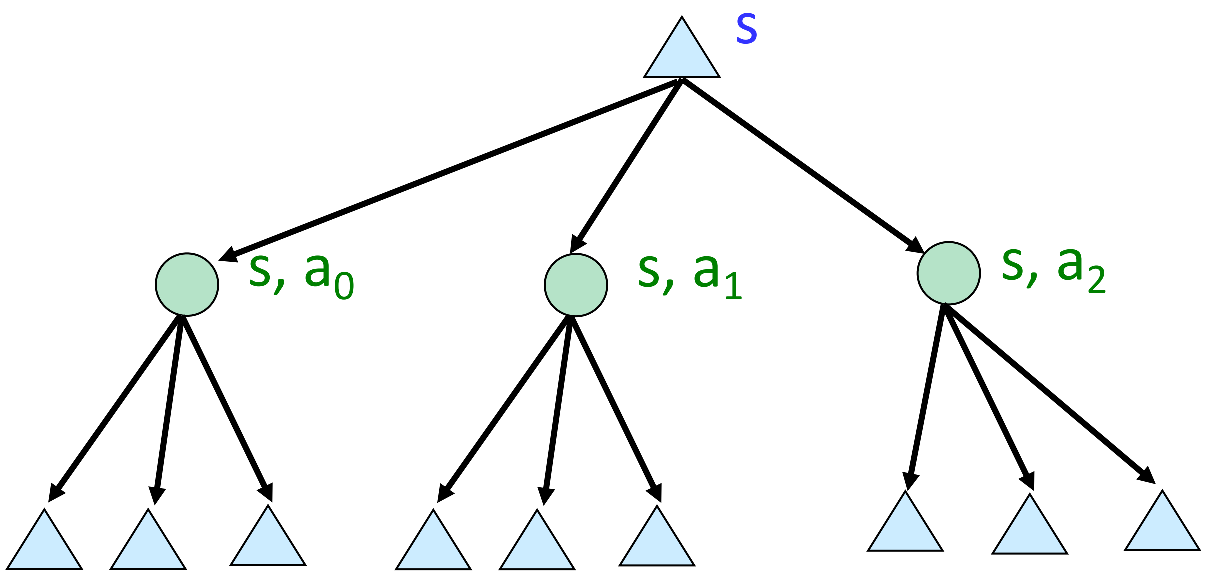
- Example of Neuroscience & AI informing each other



[A Neural Substrate of Prediction and Reward. Schultz, Dayan, Montague. 1997]

Problems with TD Value Learning

- Model-free policy evaluation! 🎉
- Bellman updates with running sample mean! 🎉



- Need the transition model to improve the policy! 😱

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

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

- 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 depth (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 as approximate Q-iteration

- Recall the definition of Q values:
  - $Q^*(s,a)$ = expected return from doing a in s and then behaving optimally thereafter; and $\pi^*(s) = \max_a Q^*(s,a)$
- Bellman equation for Q values:
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$
- Approximate Bellman update for Q values:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
- We obtain a policy from learned Q(s,a), with no model!
  - (No free lunch: Q(s,a) table is |A| times bigger than V(s) table)

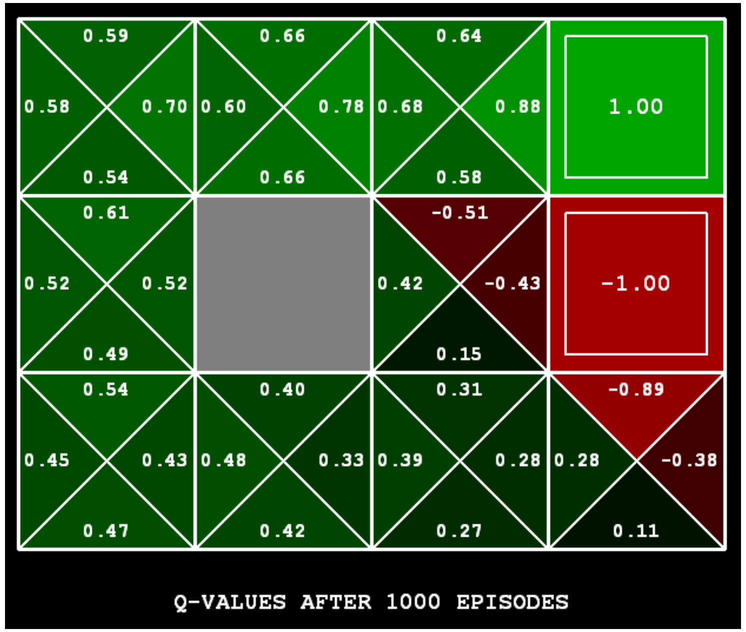
Q-Learning

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

    $$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \cdot [q\_target]$$



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

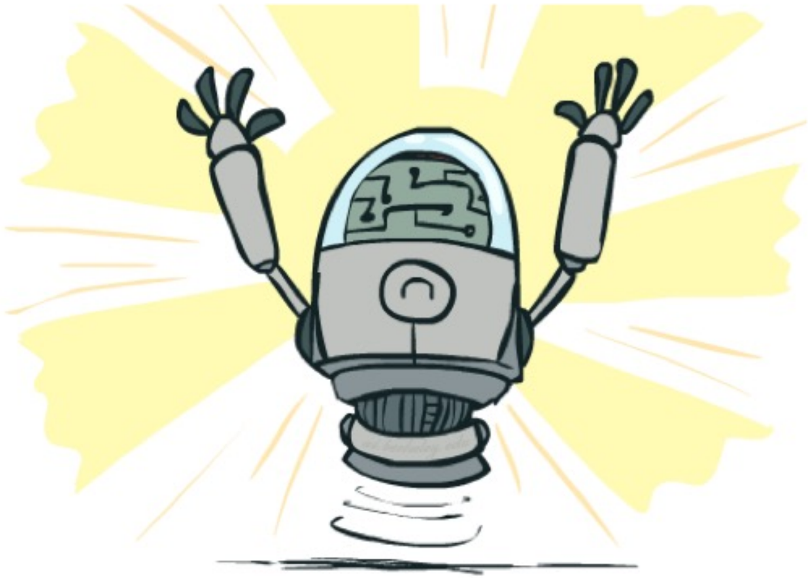
Video of Demo Q-Learning -- Gridworld

Q-Learning Properties

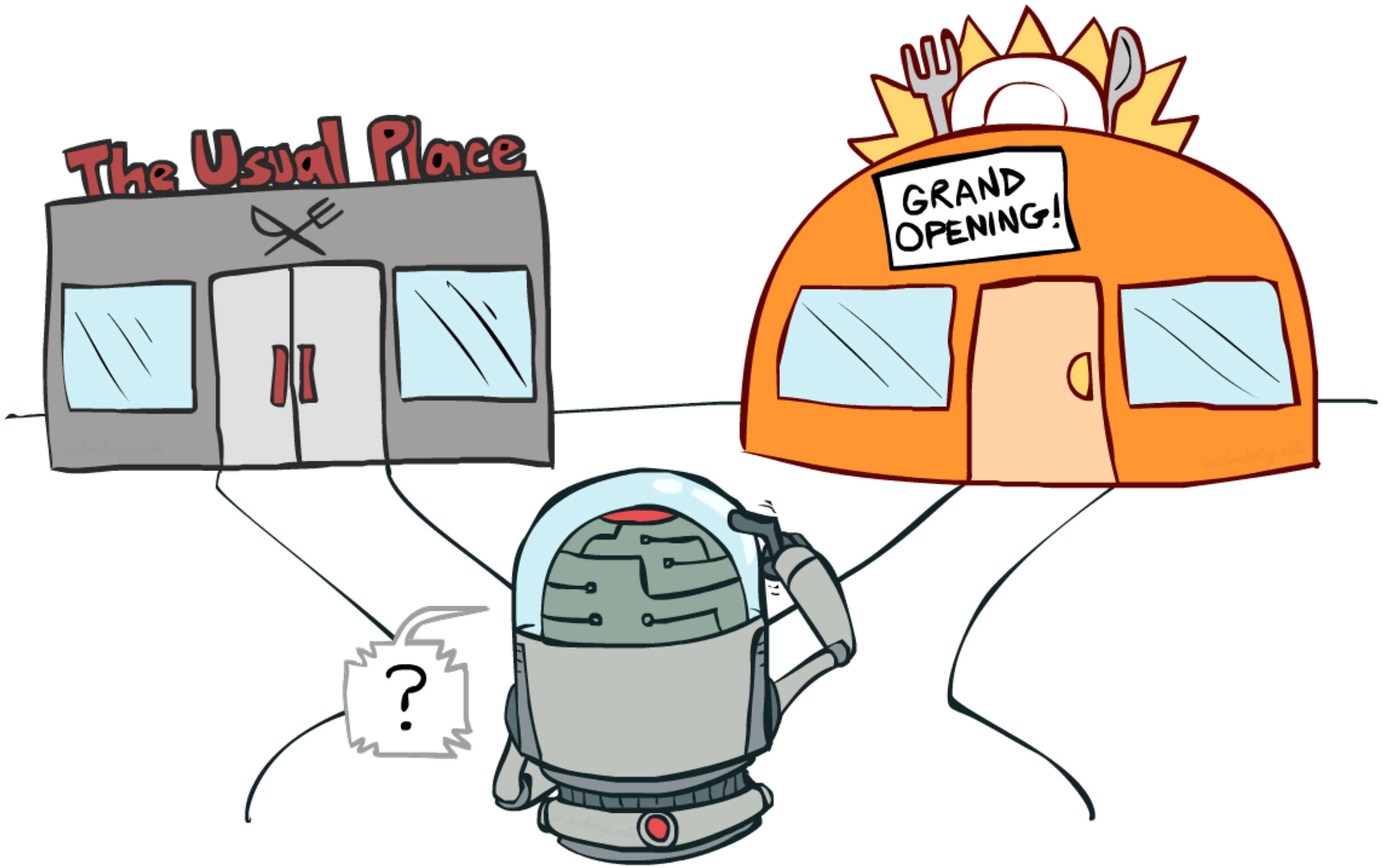
- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!

- 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 (!)



Exploration vs. Exploitation



Exploration vs. Exploitation

- ***Exploration***: try new things
- ***Exploitation***: do what's best given what you've learned so far
- Key point: pure exploitation often gets ***stuck in a rut*** and never finds an optimal policy!

Exploration method 1: ε-greedy

- ε-greedy exploration
  - Every time step, flip a biased coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Properties of ε-greedy exploration
  - Every s,a pair is tried infinitely often
  - Does a lot of stupid things
    - Jumping off a cliff *lots of times* to make sure it hurts
  - Keeps doing stupid things for ever
    - Decay ε towards 0

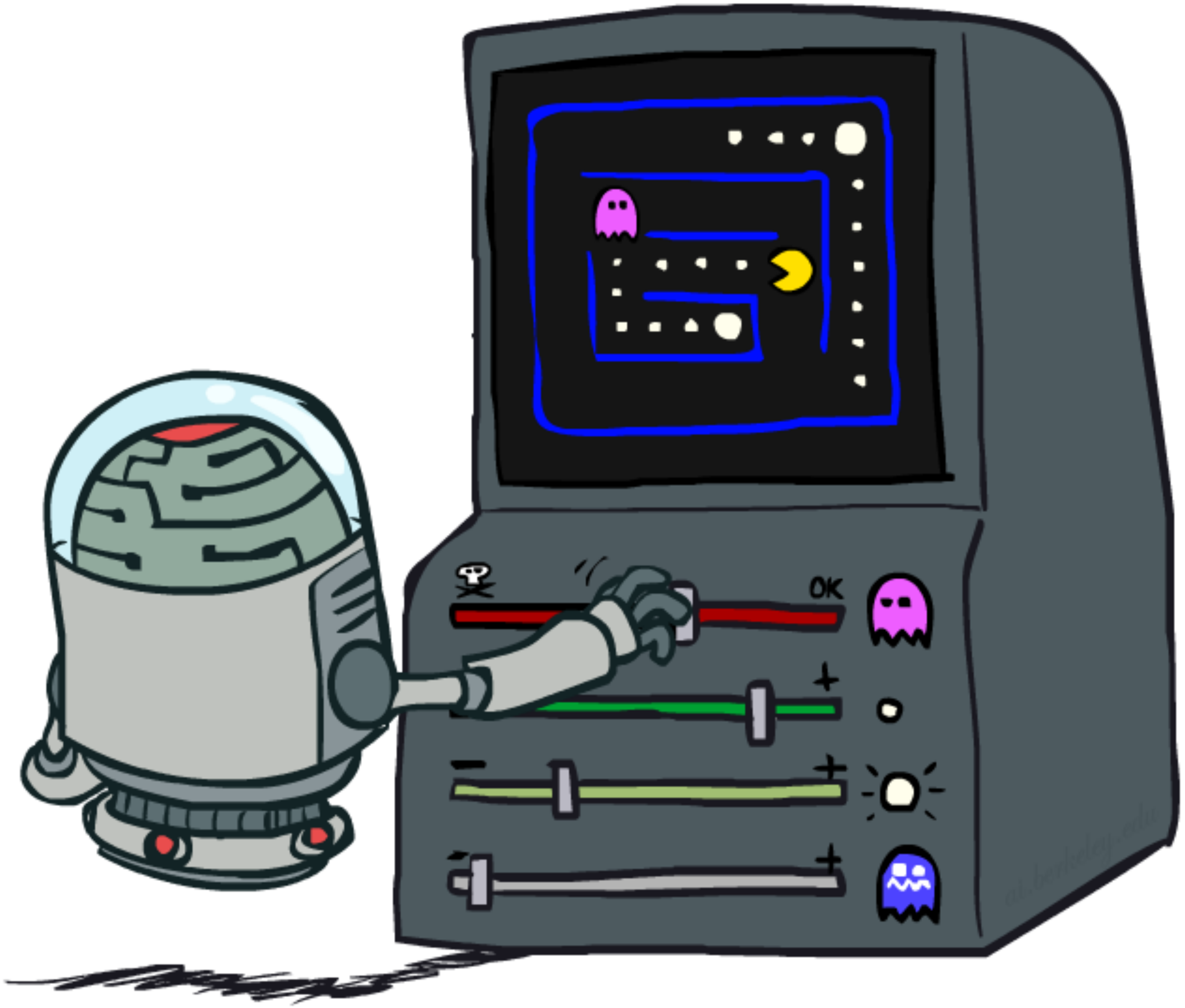
Demo Q-learning – Epsilon-Greedy – Crawler

Method 2: Optimistic Exploration Functions

- ***Exploration functions*** implement this tradeoff
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g., $f(u,n) = u + k/\sqrt{n}$
- Regular Q-update:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a Q(s',a)]$
- Modified Q-update:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a f(Q(s',a'),n(s',a'))]$
- Note: this propagates the "bonus" back to states that lead to unknown states as well!

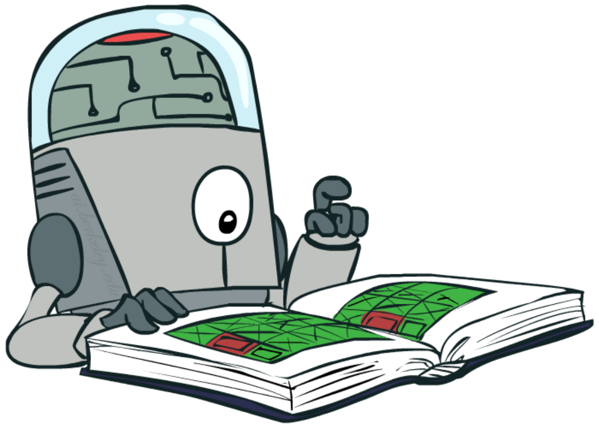
Demo Q-learning – Exploration Function – Crawler

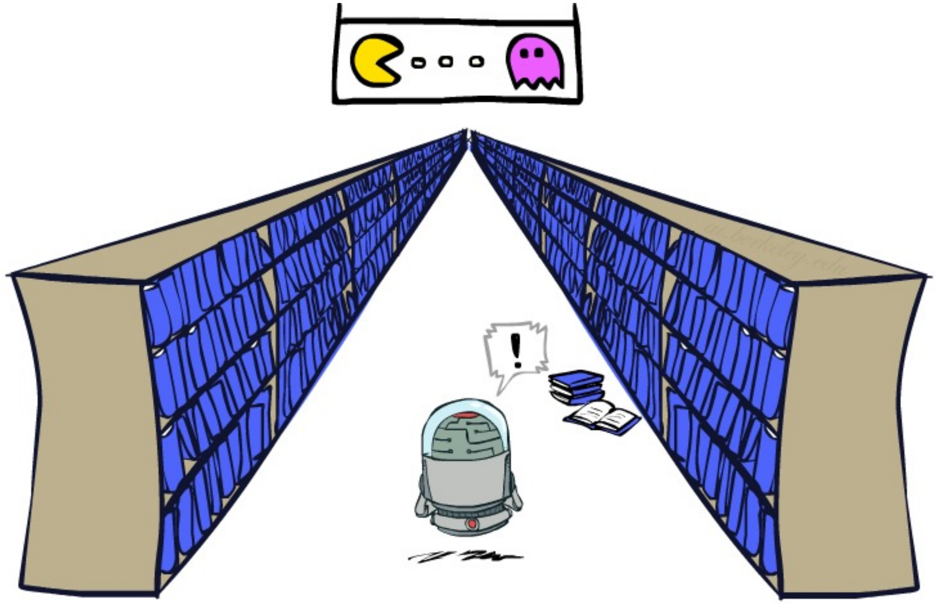
Approximate Q-Learning



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
  - Can we apply some machine learning tools to do this?

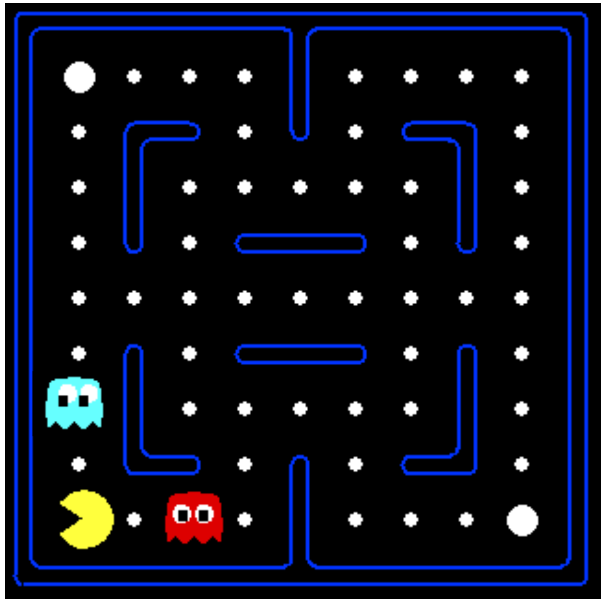




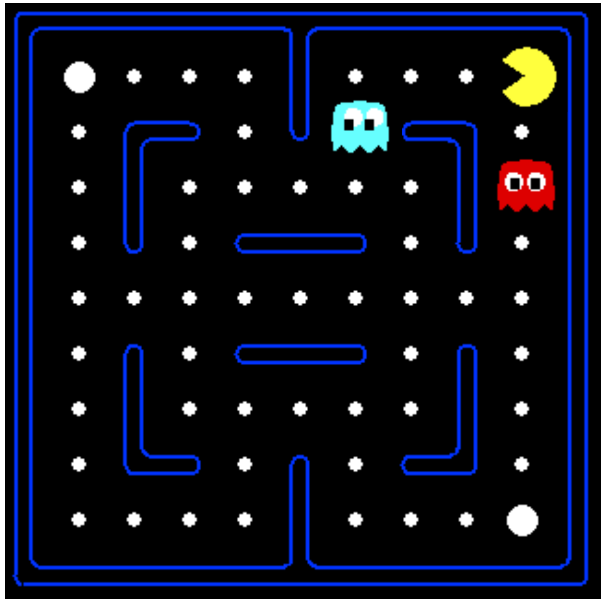
[demo – RL pacman]

Example: Pacman

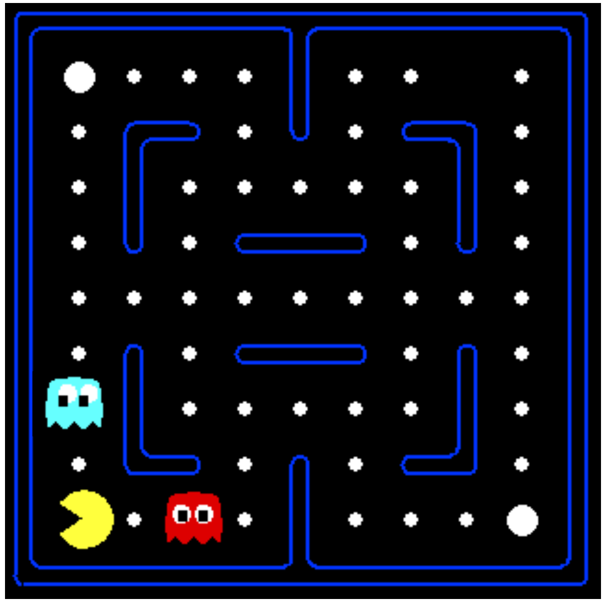
Let's say we discover through experience that this state is bad:



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



Or even this one!



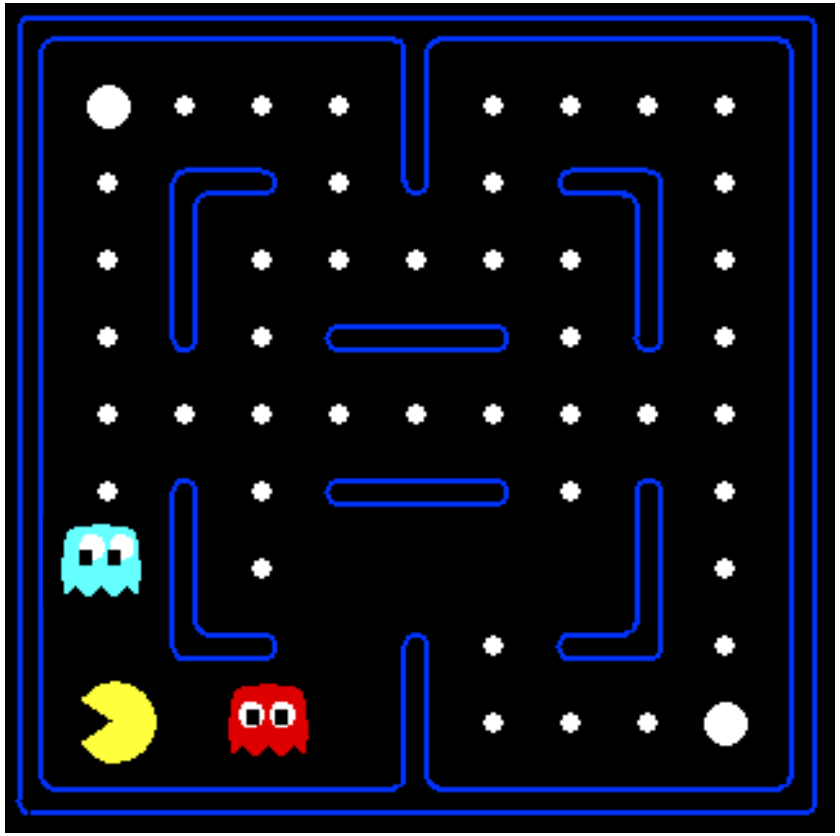
Demo Q-Learning Pacman – Tiny – Watch All

Demo Q-Learning Pacman – Tiny – Silent Train

Demo Q-Learning Pacman – Tricky – Watch All

Feature-Based Representations

- Solution: describe a state using a vector of ***features***
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost f_{GST}
    - Distance to closest dot
    - Number of ghosts
    - 1 / (distance to closest dot) f_{DOT}
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g., action moves closer to food)



Linear Value Functions

- We can express V and Q (approximately) as weighted linear functions of feature values:
  - $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$
- Advantage: our experience is summed up in a few powerful numbers
  - Can compress a value function for chess (10^{43} states) down to about 30 weights!
- Disadvantage: states may share features but have very different expected utility!

Updating a linear value function

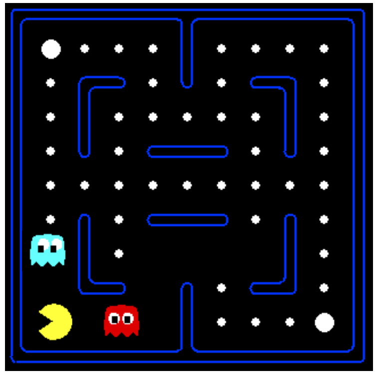
- Original Q-learning rule tries to reduce prediction error at s,a:
  - $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- Instead, we update the weights to try to reduce the error at s,a:
  - $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$

    $= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, f_i(s,a)$
- Intuitive interpretation:
  - Adjust weights of active features
  - If something bad happens, blame the features we saw; decrease value of states with those features. If something good happens, increase value!

Example: Q-Pacman

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

s

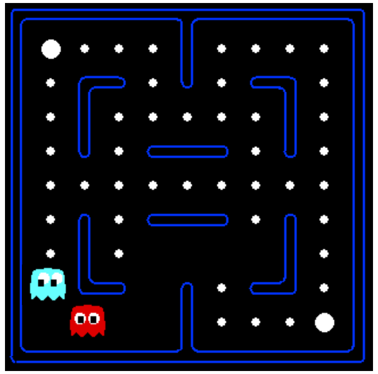


$f_{DOT}(s,\text{NORTH}) = 0.5$

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

a = NORTH
r = –500

s'



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

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

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

difference = –501

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

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

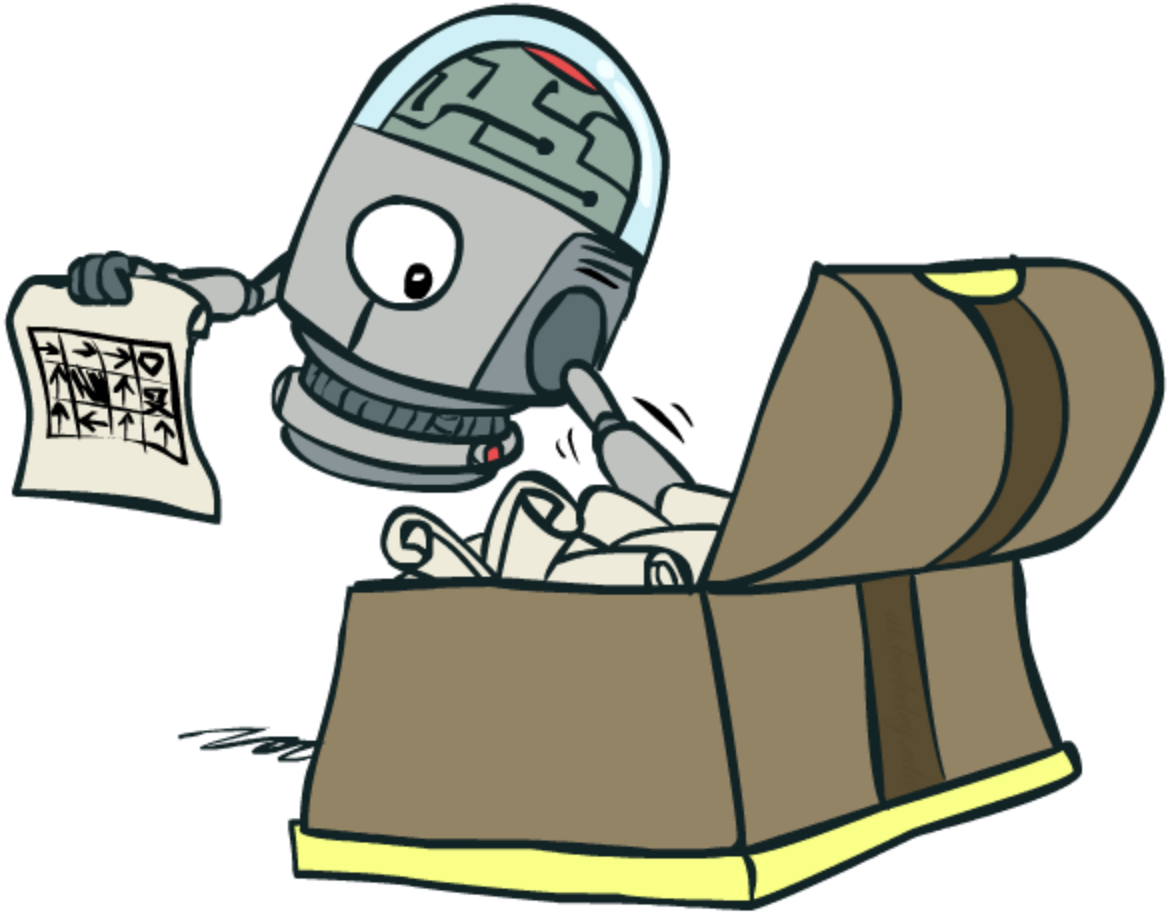
$Q(s,a) = 3.0\, f_{DOT}(s,a) - 3.0\, f_{GST}(s,a)$

Demo Approximate Q-Learning -- Pacman

Approaches to reinforcement learning

1. Model-based: Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
   a. Direct evaluation
   b. Temporal difference learning
   c. Q-learning
3. Optimize the policy directly

Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)

- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by **hill climbing** (or gradient ascent!) on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Pros:
  - Works well for partial observability / stochastic policies
- Cons:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

Policy Search

[Andrew Ng]

[Video: HELICOPTER]

Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several approaches:
  - Learn the MDP model and solve it
  - Learn V directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn Q by local Q-learning adjustments, use it directly to pick actions
  - Optimize the policy directly
- Scaling up with feature representations and approximation