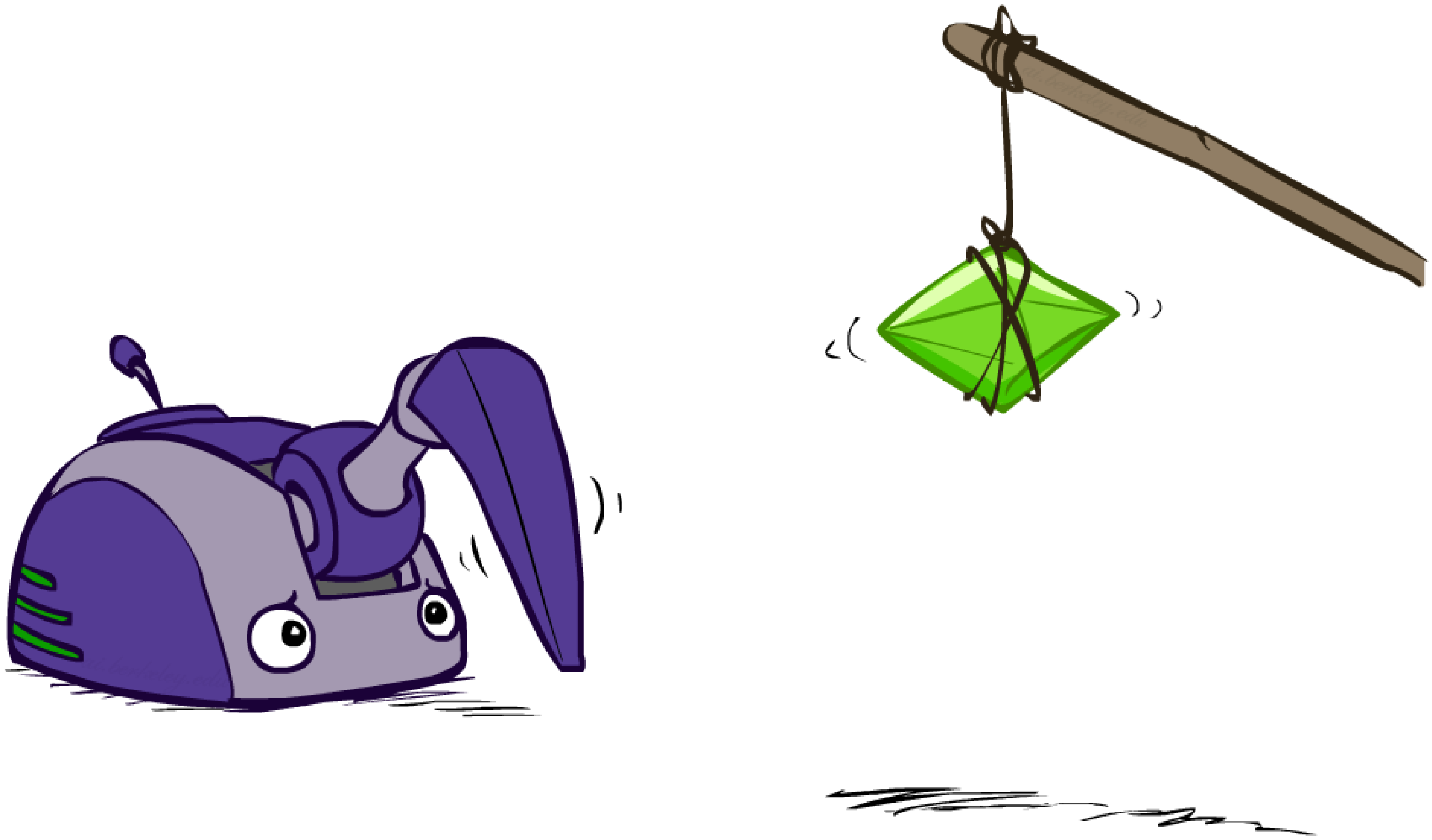
Reinforcement Learning

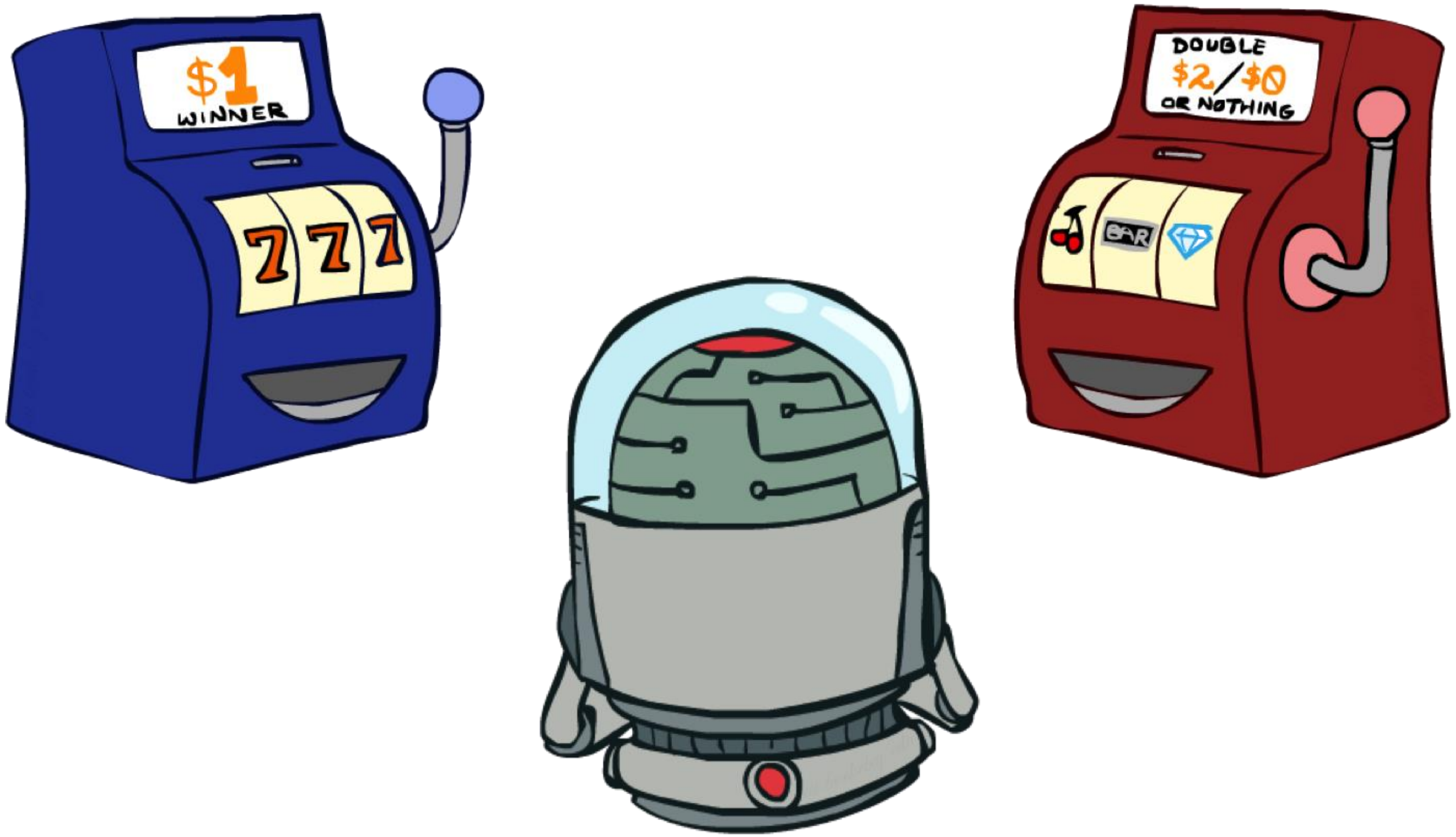


Bandits

- Exactly one state
- Set of actions: A
- Stochastic reward function: $P(r|a)$

Contextual Bandits:

- Set of states $s \in S$
- Transitions always return to start state distribution $P(s'|s, a) = P_0(s')$

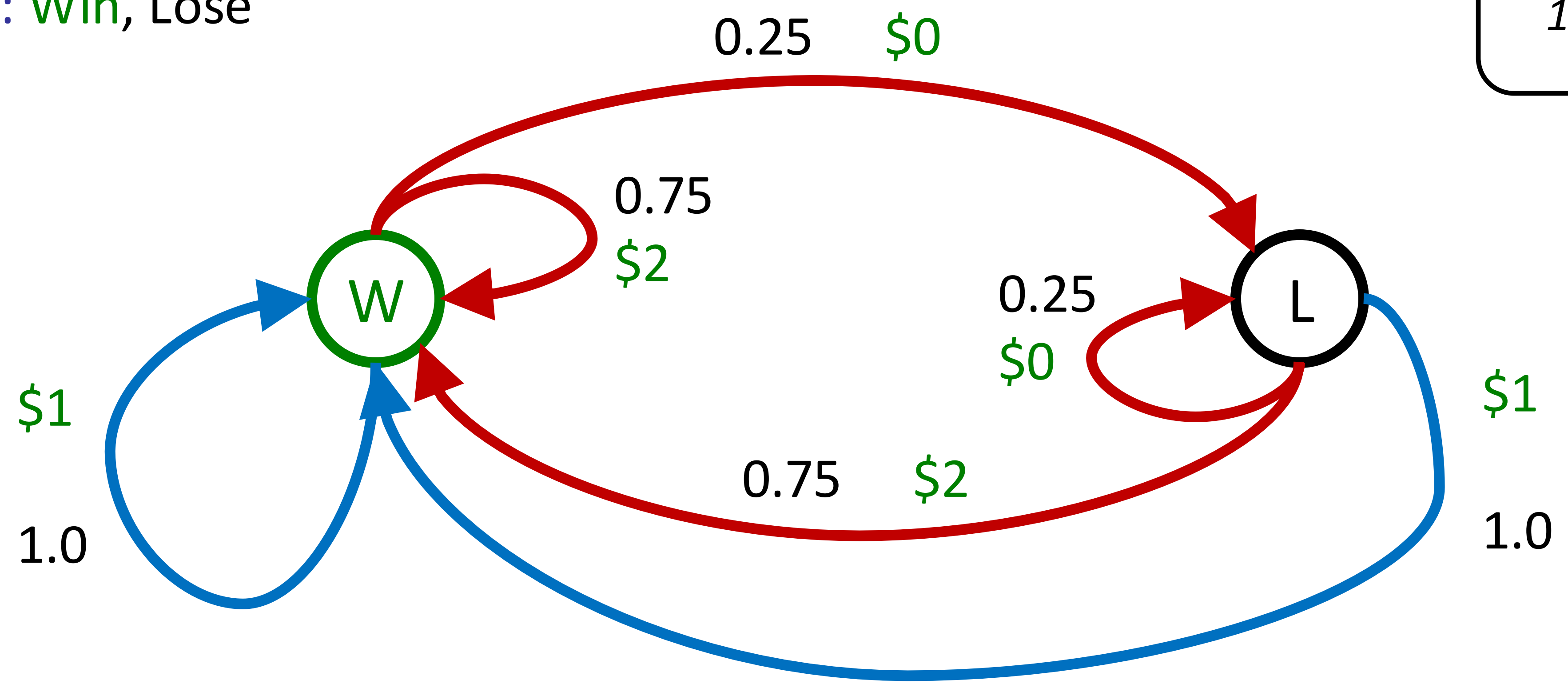


Double-Bandit MDP

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

No discount

100 time steps

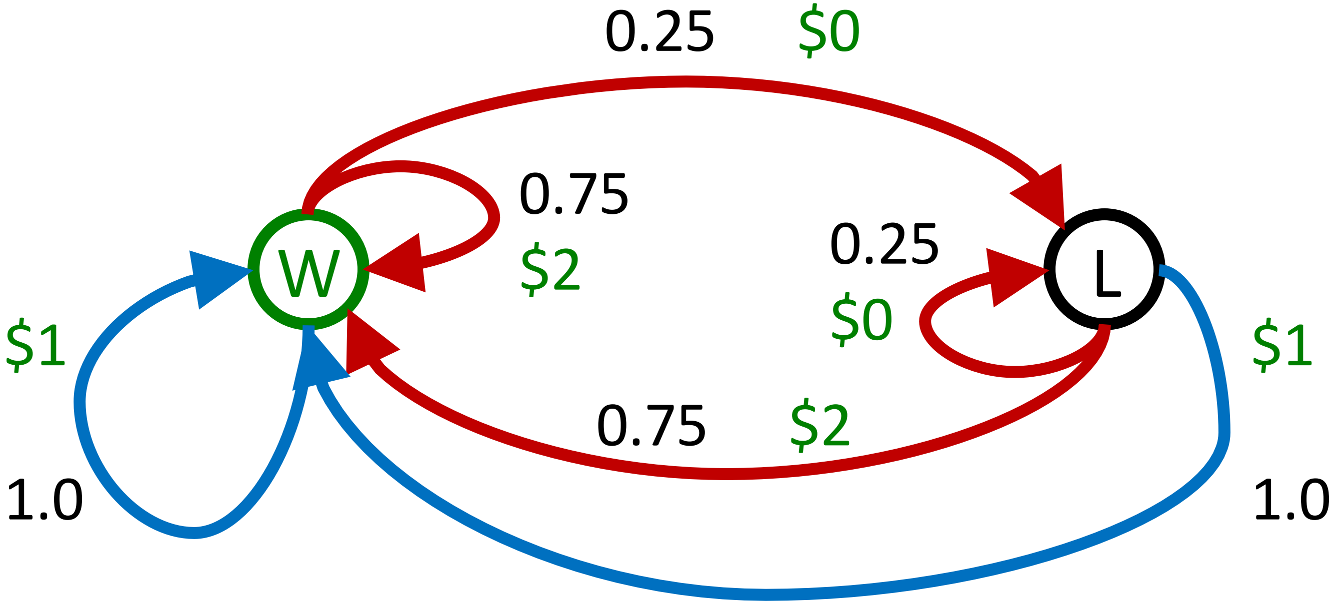


Offline Planning

- 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!

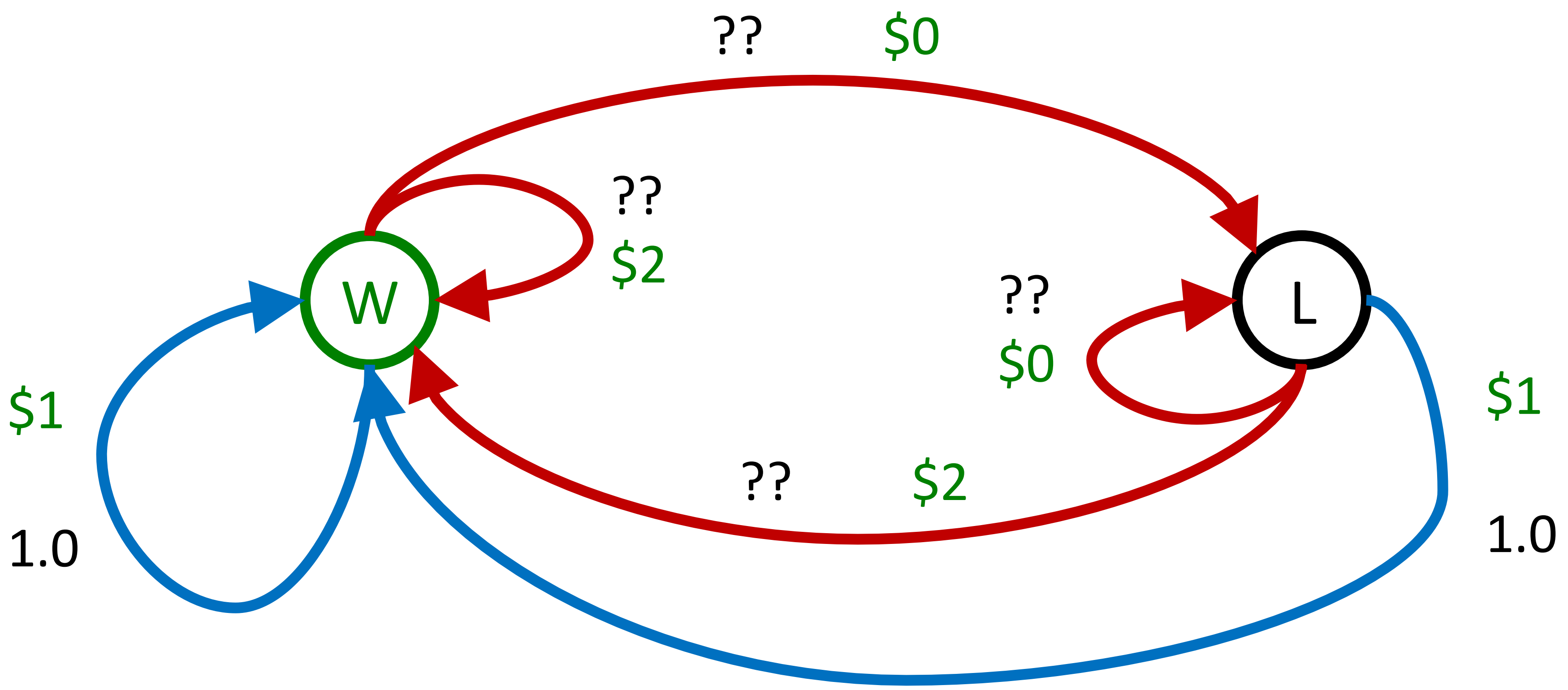
No discount
100 time steps

	Value
Play Red	150
Play Blue	100

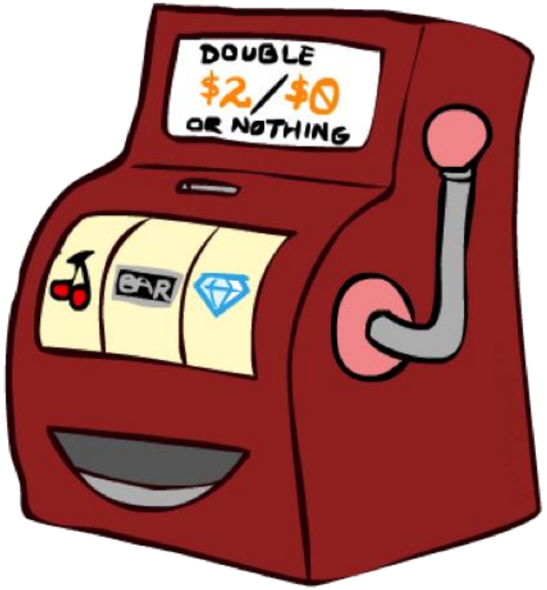


Online Planning

- Rules changed! Red's win chance is different.



Let's Play!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

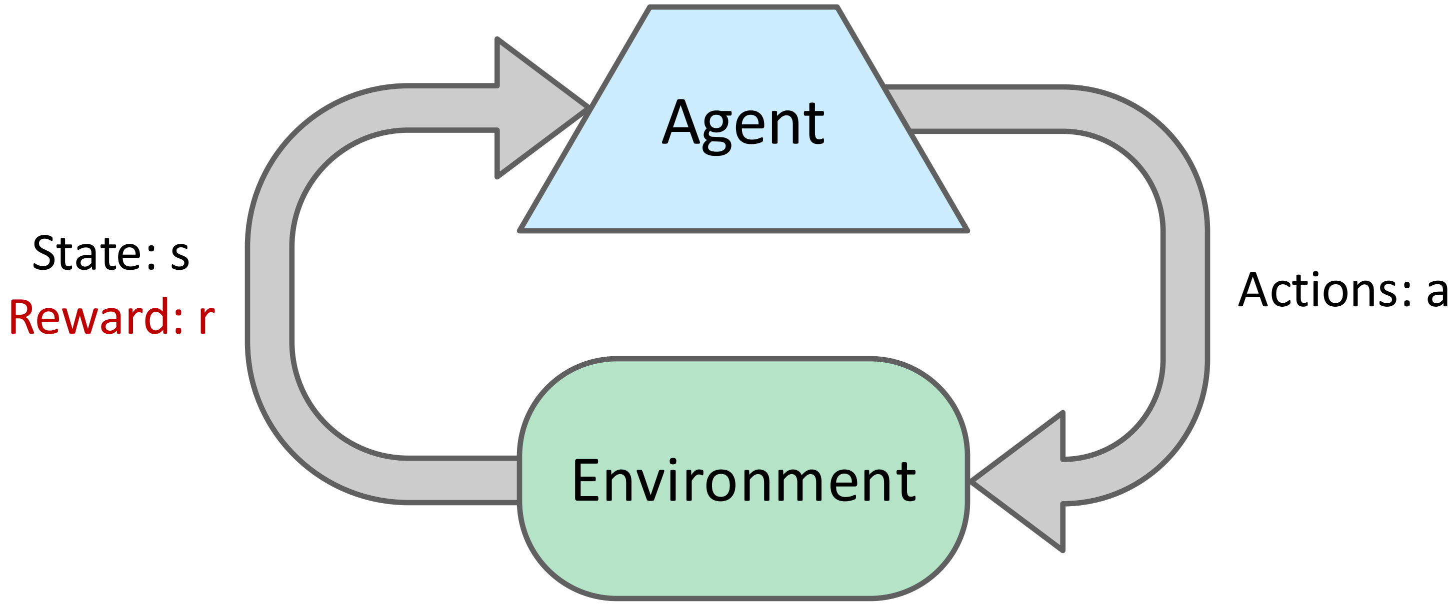
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  1. A set of states $s \in S$
  2. A set of actions (per state) A
  3. A model $T(s,a,s')$
  4. 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

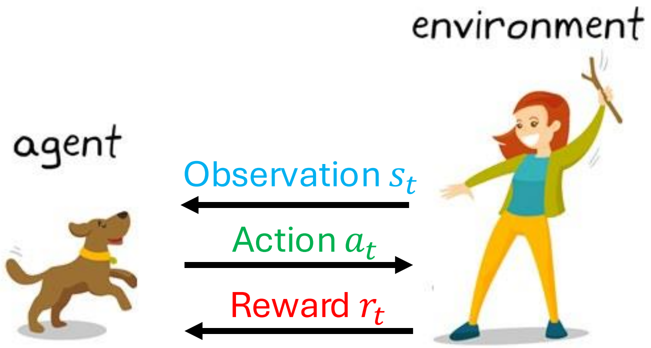
Reinforcement Learning



- 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!

Basis of reinforcement learning*

- Markov decision process (MDP) assumption
- At timestep t, agent following a policy $\pi_\omega(s_t)$,
  - Obtains an observation s_t of the surrounding environment,
  - produces action a_t,
  - then, environment will transmit to s_{t+1},
  - and agent will receive reward r_t,



  - The goal of the learning agent is to *maximize* the expected cumulative rewards as

$$R = \mathrm{E}_\pi\left[\sum_{t=0}^{T-1} r_t(s_t, a_t)\right]$$

Interaction data stored in the transition dataset.

$$\{s_0, a_0, r_0, s_1, a_1, r_1, \dots, \underbrace{s_t, a_t, r_t, s_{t+1}}_{\text{Transition}}, \dots s_{T-1}, a_{T-1}, r_{T-1},\}$$

* Episodic environment with limited timesteps

Reinforcement Learning

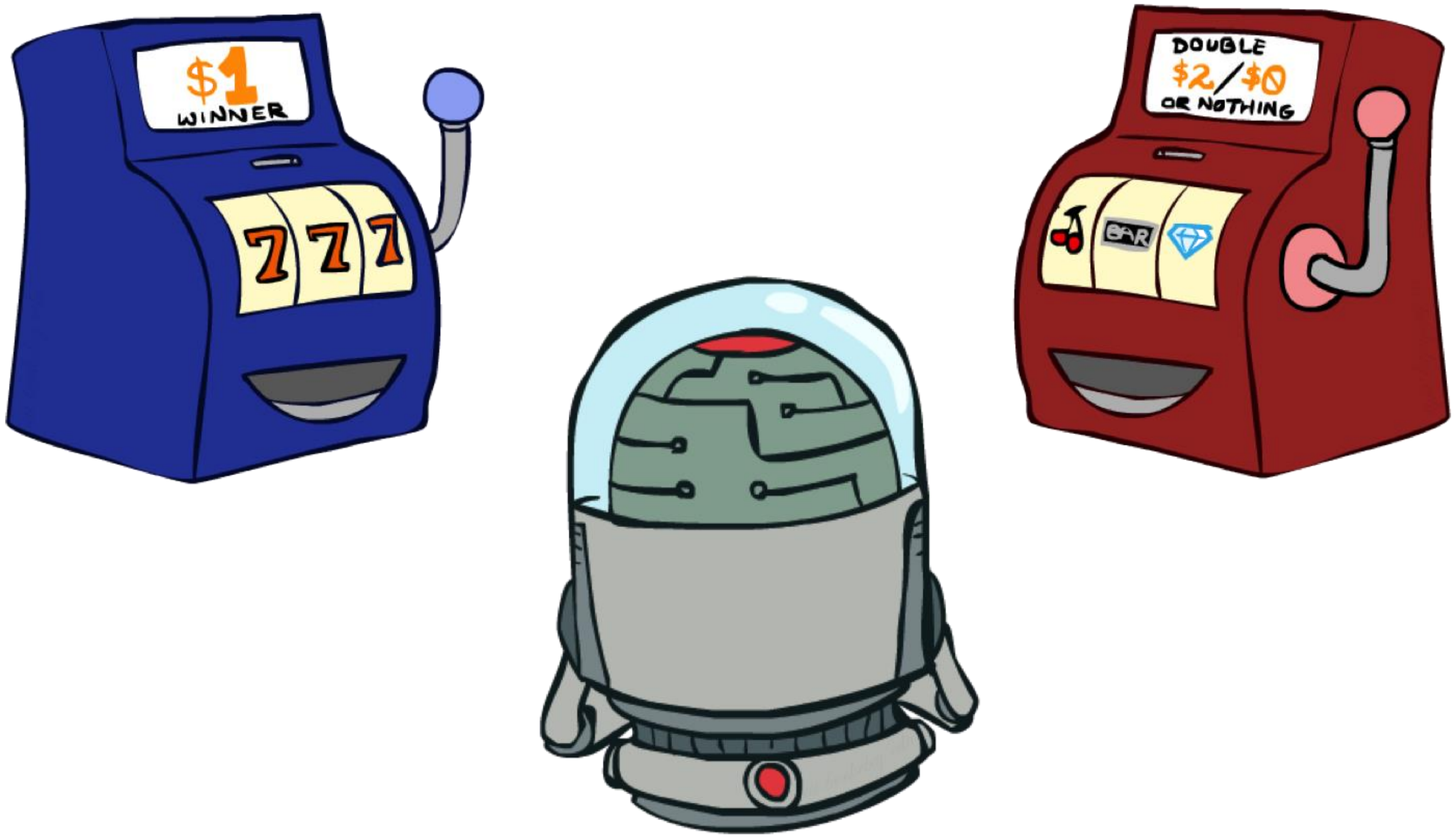
- What if the MDP is initially unknown? Lots of things change!
  - ***Exploration***: you have to ***try unknown actions*** to get information
  - ***Exploitation***: eventually, you have to use what you know
  - ***Regret***: early on, you inevitably "make mistakes" and lose reward
  - ***Sampling***: you may need to repeat many times to get good estimates
  - ***Generalization***: what you learn in one state may apply to others too

Bandits

- Exactly one state
- Set of actions: A
- Stochastic reward function: $P(r|a)$

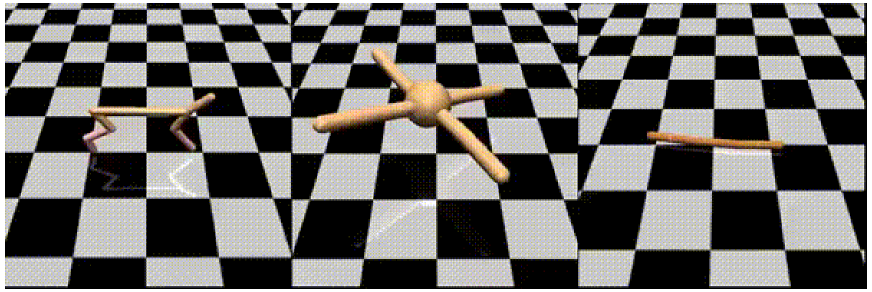
Contextual Bandits:

- Set of states $s \in S$
- Transitions always return to start state distribution $P(s'|s, a) = P_0(s')$



Practical examples of sequential decision making

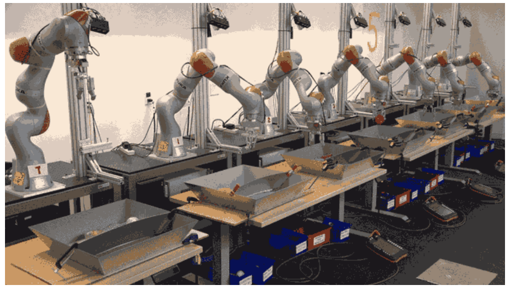
Logistics system



MuJoCo Robot Control

Video Games

Intelligent financial investment

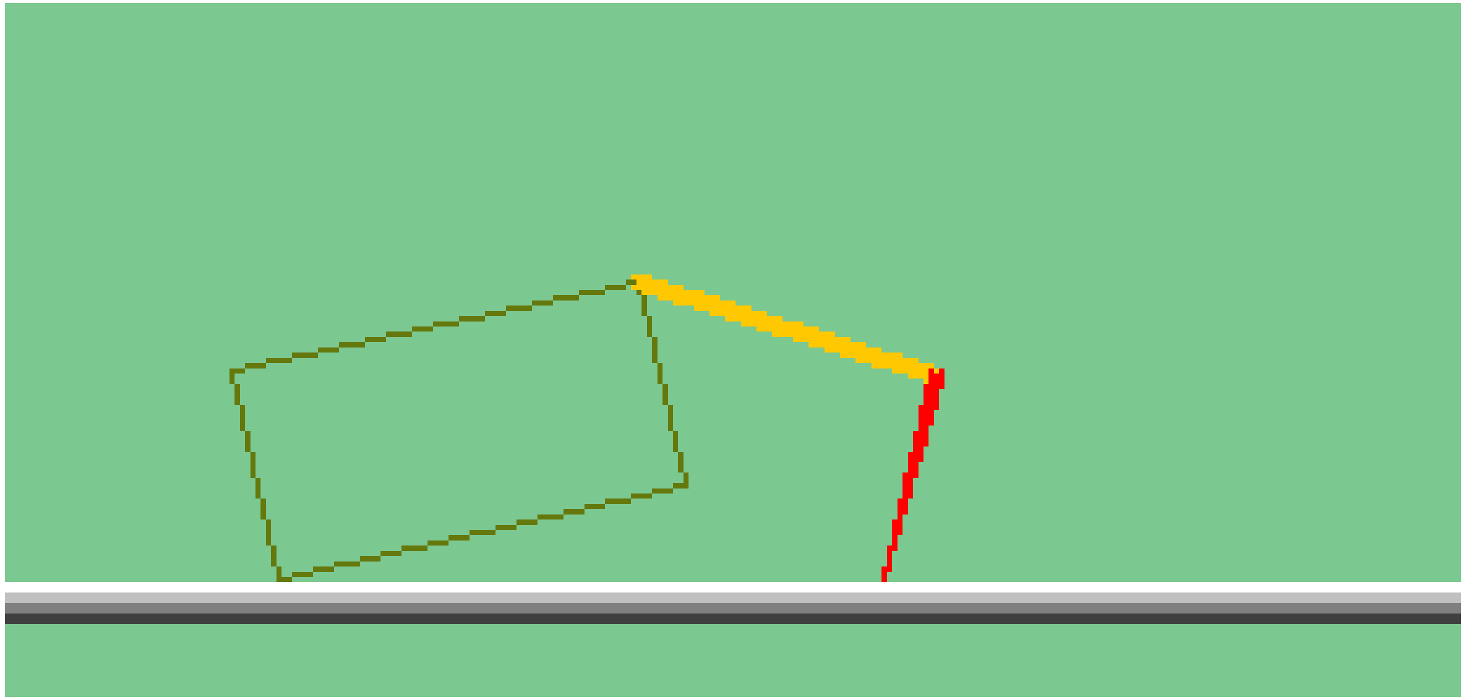


Industrial production

Recommendation and ads

Cheetah

The Crawler!



[Demo: Crawler Bot (L10D1)] [You, in Project 6]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

Example: Breakout (DeepMind)

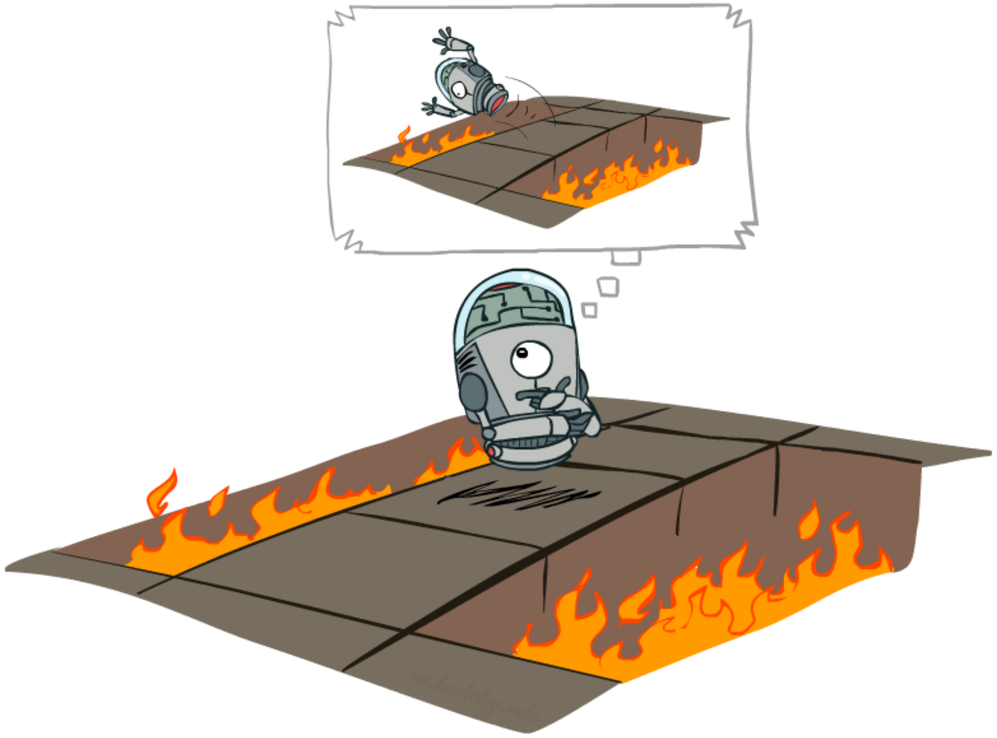
[© TwoMinuteLectures]

Example: AlphaGo (2016)

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

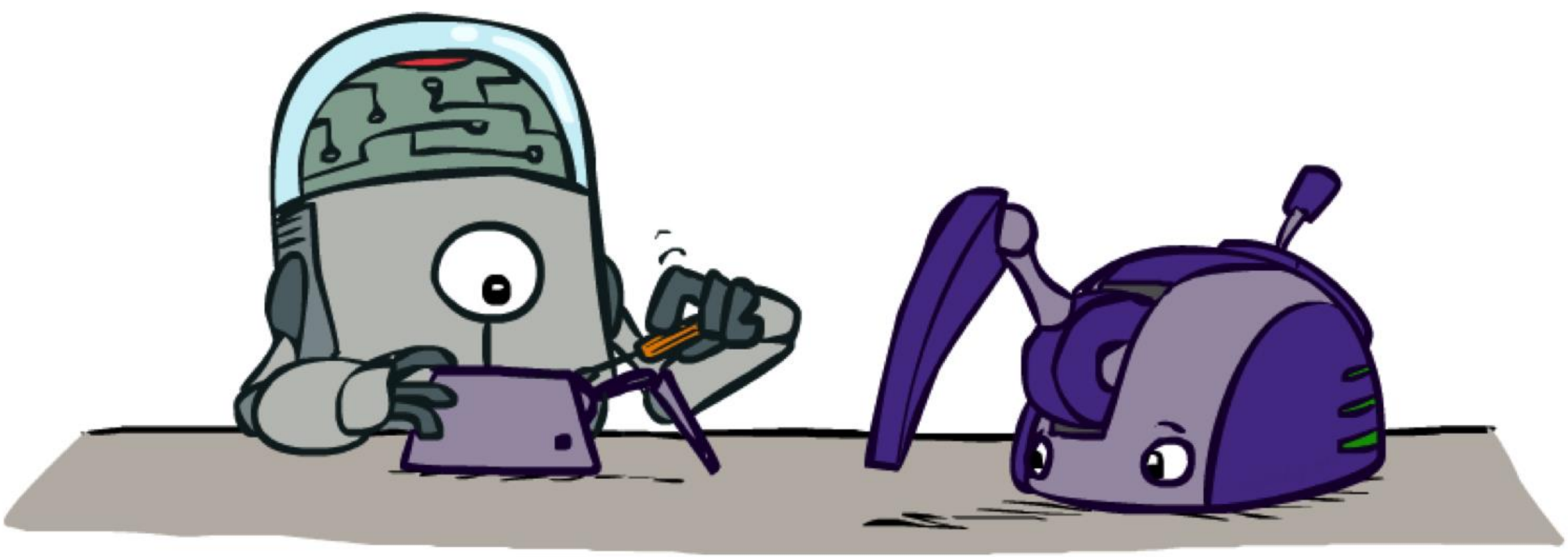
Offline (MDPs) vs. Online (RL)



Offline Solution

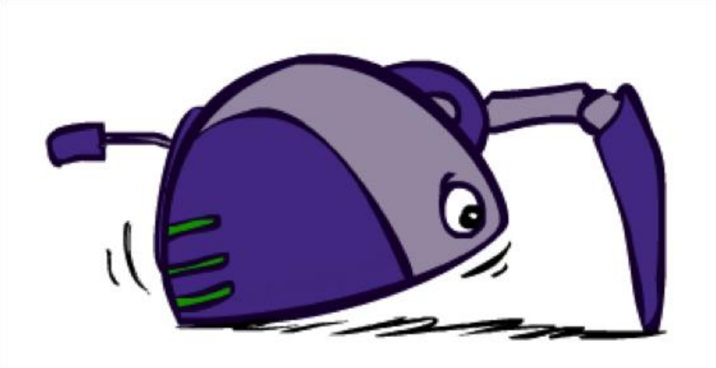
Online Learning

Model-Based RL



Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

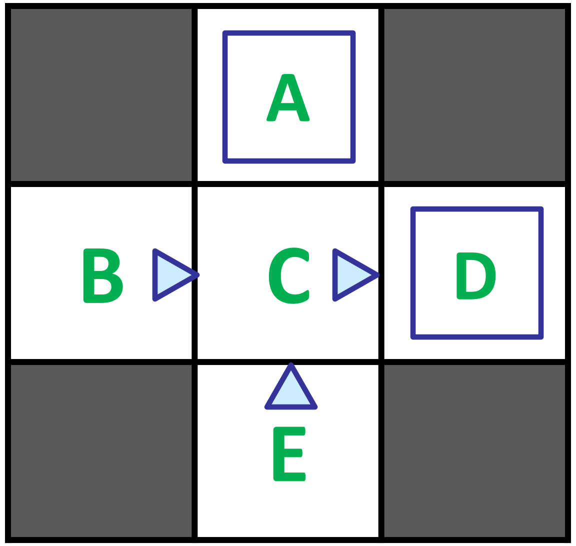


- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Directly estimate each entry in $T(s,a,s')$ from counts
  - Discover each $R(s,a,s')$ when we experience the transition

- Step 2: Solve the learned MDP
  - Use, e.g., value or policy 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

$T(s,a,s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$R(s,a,s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Pros and cons

- Pro:
  - Makes efficient use of experiences (low ***sample complexity***)

- Con:
  - May not scale to large state spaces
    - Solving MDP is intractable for very large $|S|$
  - RL feedback loop tends to magnify small model errors
  - Much harder when the environment is partially observable

Basic idea of model-free methods

- To approximate expectations with respect to a distribution, you can either
  - Estimate the distribution from samples, compute an expectation
  - Or, bypass the distribution and estimate the expectation from samples directly

Model-Based vs. Model-Free

Goal: Compute expected age of ShanghaiTech students

Known P(A)

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

Without P(A), instead collect samples $[a_1, a_2, \ldots a_N]$

Unknown P(A): "Model Based"

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

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

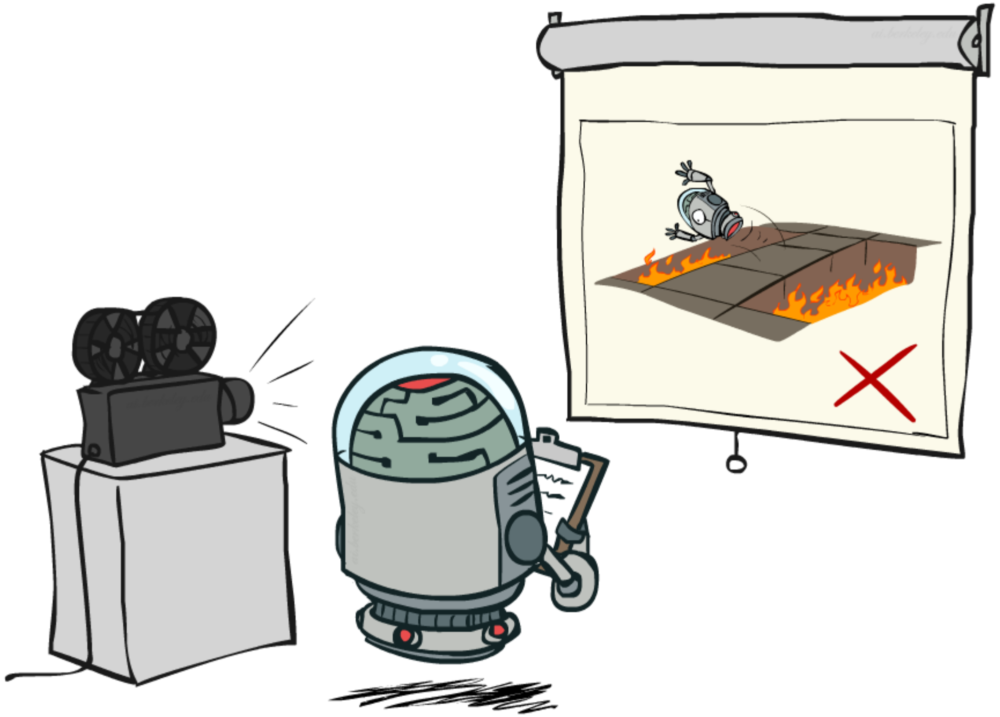
Why does this work? Because eventually you learn the right model.

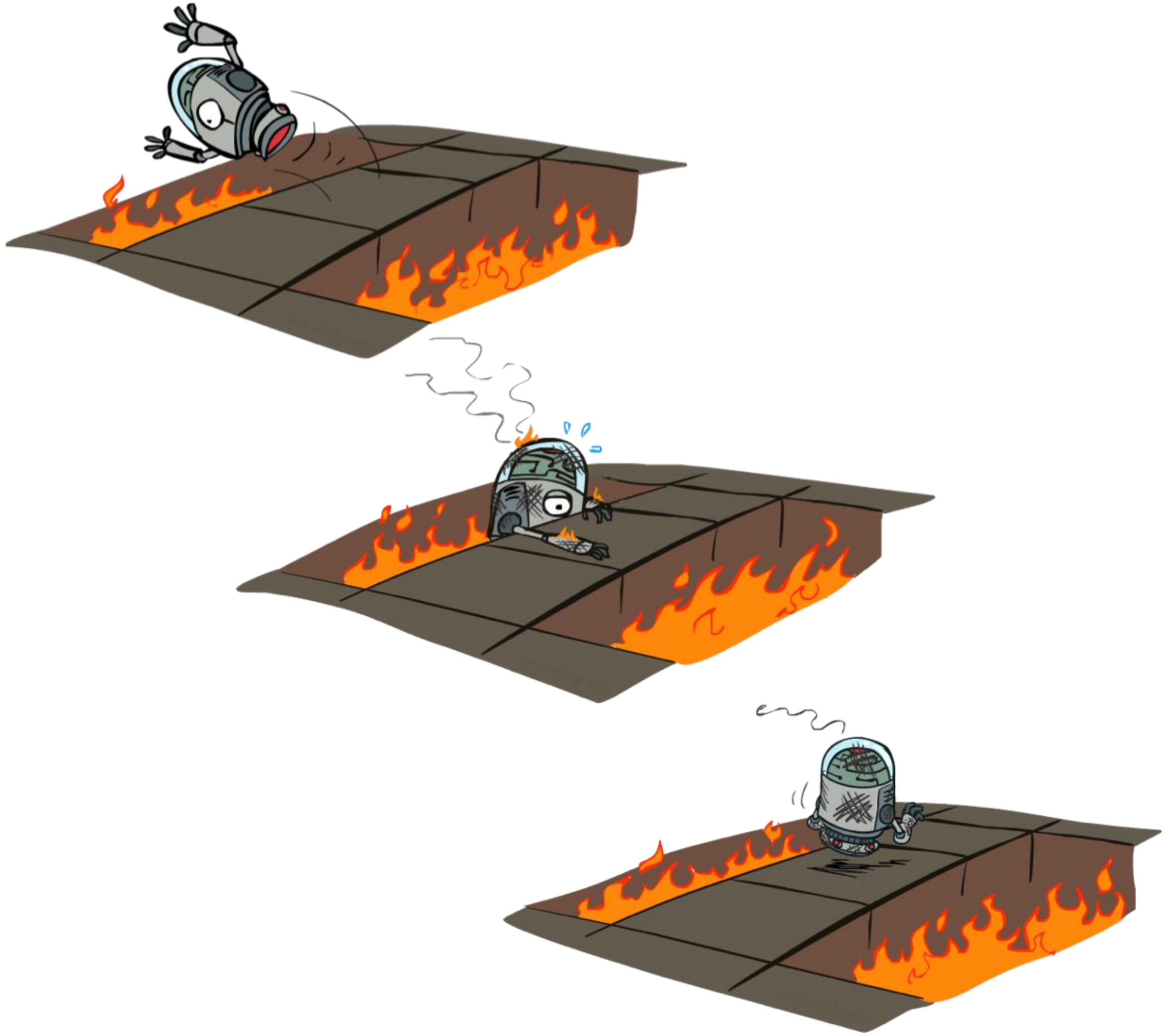
Unknown P(A): "Model Free"

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

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

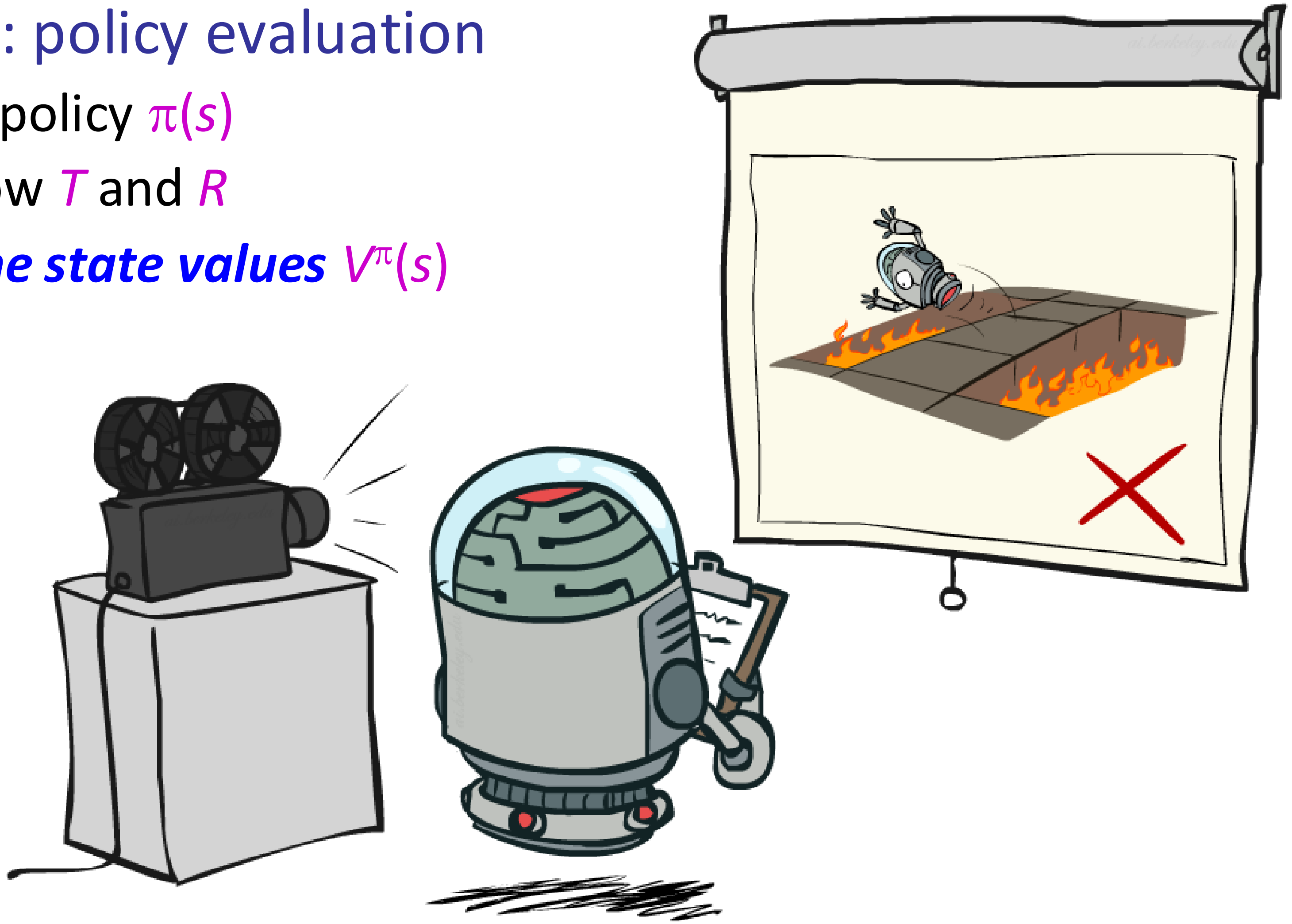
Passive vs Active Reinforcement Learning





Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know T and R
  - ***Goal: learn the state values*** $V^{\pi}(s)$



Direct evaluation

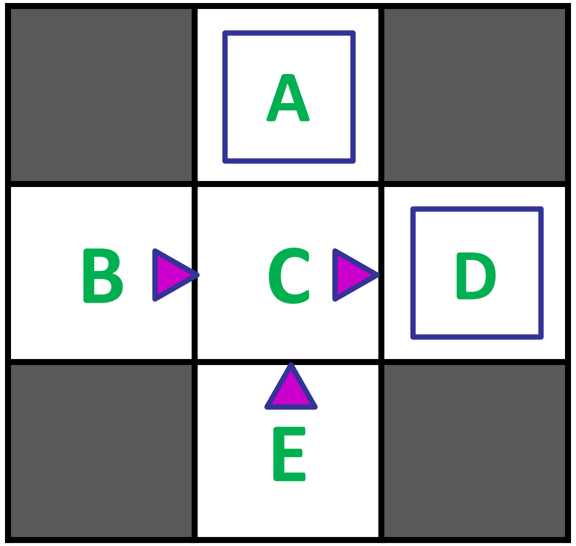
- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from s onwards

- Idea:
  - Use ***returns***, the *actual* sums of discounted rewards from s
  - Average over multiple trials and visits to s

- This is called ***direct evaluation*** (or direct utility estimation)

Example: Direct 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

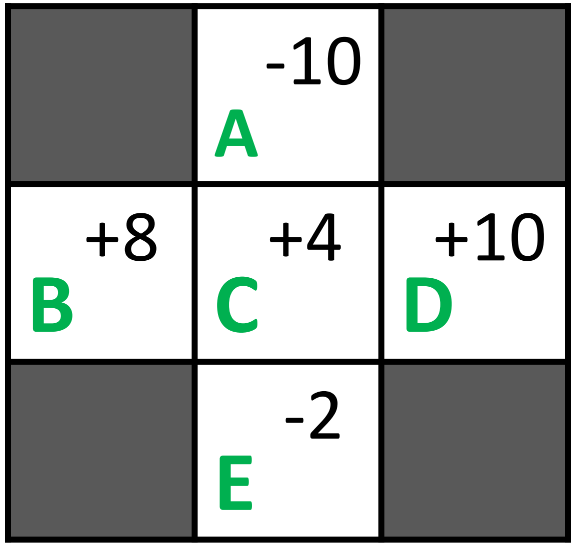
Episode 3

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

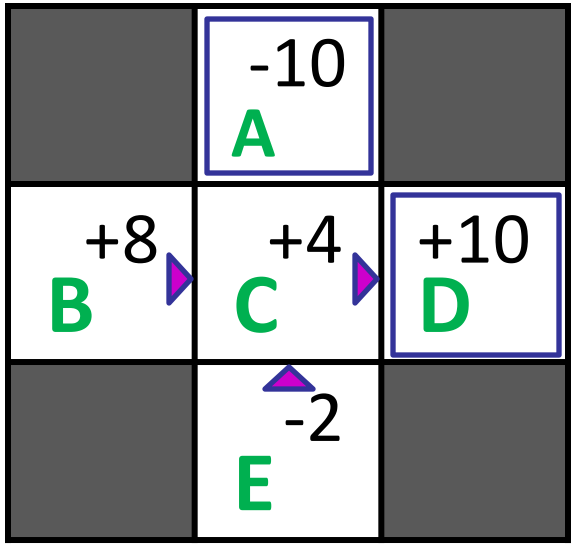


Problems with Direct Estimation

- What's good about direct estimation?
  - It's easy to understand
  - It doesn't require any knowledge of *T* and *R*
  - It converges to the right answer in the limit

- What's bad about it?
  - Each state must be learned separately (fixable)
  - It ***ignores information about state connections***
  - 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?

E.g., B=at home, study hard
E=at library, study hard
C=know material, go to exam

Approaches to reinforcement learning

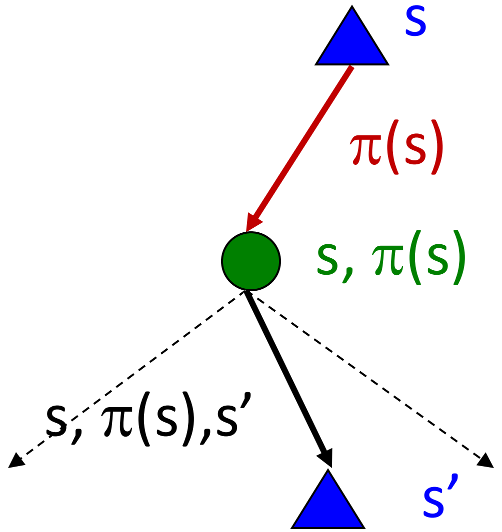
1. Model-based: Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
   a. Direct evaluation
   b. Temporal difference learning
   c. Q-learning
3. Optimize the policy directly

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:

$$V_0^\pi(s) = 0$$

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

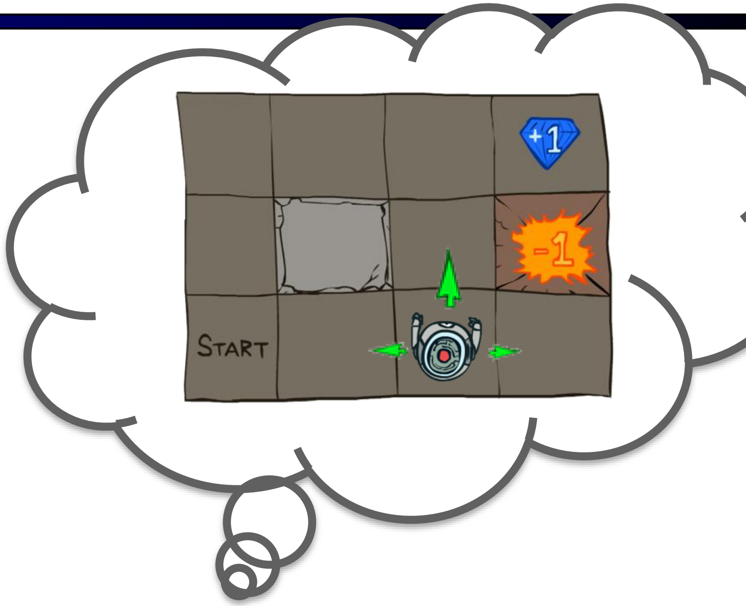


  - This approach fully exploits the connections between the states
  - Unfortunately, we need T and R to do it!

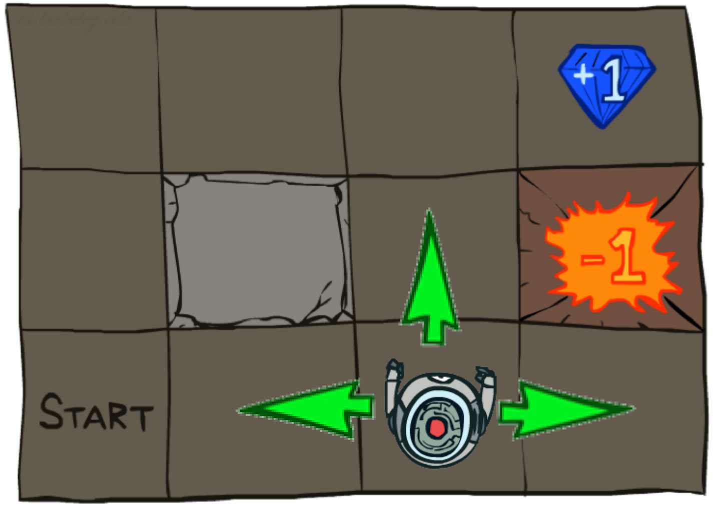
- Key question: how can we do this update to V without knowing T and R?
  - In other words, how do we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

- Given a fixed policy, the value of a state is an expectation over next-state values:
  - $V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s') [R(s,\pi(s),s') + \gamma V^{\pi}(s')]$



- Idea 1: Use actual samples to estimate the expectation:
  - $\text{sample}_1 = R(s,\pi(s),s_1') + \gamma V^{\pi}(s_1')$
  - $\text{sample}_2 = R(s,\pi(s),s_2') + \gamma V^{\pi}(s_2')$
  - …
  - $\text{sample}_N = R(s,\pi(s),s_N') + \gamma V^{\pi}(s_N')$
  - $V^{\pi}(s) \leftarrow 1/N \sum_i \text{sample}_i$



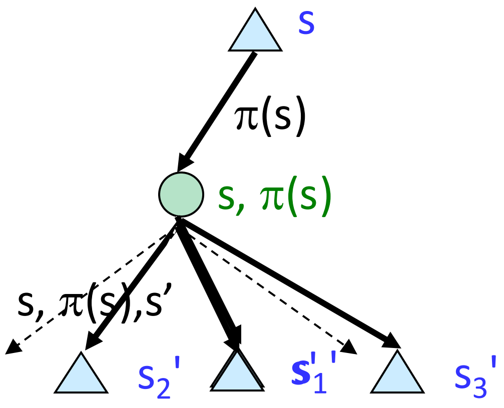
Sample-Based Policy Evaluation?

- We want to compute 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

$$sample_1 = R(s,\pi(s),s'_1) + \gamma V_k^{\pi}(s'_1)$$
$$sample_2 = R(s,\pi(s),s'_2) + \gamma V_k^{\pi}(s'_2)$$
$$\ldots$$
$$sample_n = R(s,\pi(s),s'_n) + \gamma V_k^{\pi}(s'_n)$$



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

Any problems?

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

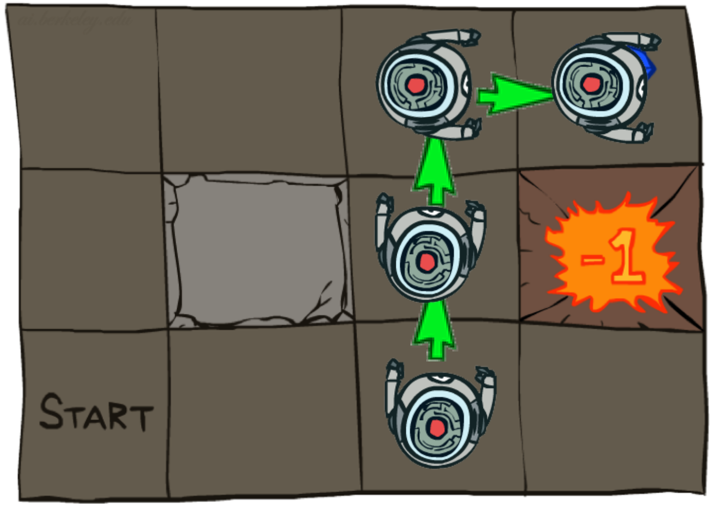
Sample-Based Policy Evaluation?

- Idea 2: Update value of s after each transition s,a,s',r :

- Update $V^{\pi}([3,1])$ based on $R([3,1],up,[3,2])$ and $\gamma V^{\pi}([3,2])$
- Update $V^{\pi}([3,2])$ based on $R([3,2],up,[3,3])$ and $\gamma V^{\pi}([3,3])$
- Update $V^{\pi}([3,3])$ based on $R([3,3],right,[4,3])$ and $\gamma V^{\pi}([4,3])$

Any problems?

One sample estimation may not be accurate.
Early estimation function may not be accurate.



Sample-Based Policy Evaluation?

- Idea 3: Update values by maintaining a ***running average***

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 average μ_n and a running count n
  - $n=0 \quad \mu_0=0$
  - $n=1 \quad \mu_1 = (0 \cdot \mu_0 + x_1)/1 = (0 \cdot 0 + 1)/1 = 1$
  - $n=2 \quad \mu_2 = (1 \cdot \mu_1 + x_2)/2 = (1 \cdot 1 + 4)/2 = 2.5$
  - $n=3 \quad \mu_3 = (2 \cdot \mu_2 + x_3)/3 = (2 \cdot 2.5 + 7)/3 = 4$
  - General formula: $\mu_n = ((n-1) \cdot \mu_{n-1} + x_n)/n$
  - $= [(n-1)/n]\, \mu_{n-1} + [1/n]\, x_n$ (weighted average of old mean, new sample)

Running averages contd.

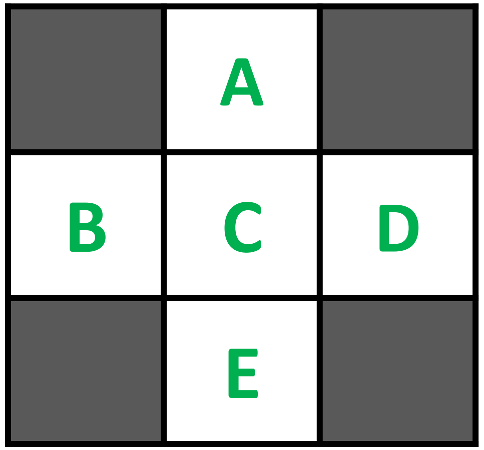
- What if we use a weighted average with a fixed weight?
  - $\mu_n = (1-\alpha)\,\mu_{n-1} + \alpha\, x_n$
  - n=1 $\quad \mu_1 = x_1$
  - n=2 $\quad \mu_2 = (1-\alpha)\cdot\mu_1 + \alpha x_2 = (1-\alpha)\cdot x_1 + \alpha x_2$
  - n=3 $\quad \mu_3 = (1-\alpha)\cdot\mu_2 + \alpha x_3 = (1-\alpha)^2\cdot x_1 + \alpha(1-\alpha)x_2 + \alpha x_3$
  - 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

- Idea 3: Update values by maintaining a ***running average***
  - sample = $R(s,\pi(s),s') + \gamma V^{\pi}(s')$
  - $V^{\pi}(s) \leftarrow (1-\alpha) \cdot V^{\pi}(s) + \alpha \cdot \text{sample}$
  - $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \cdot [\text{sample} - V^{\pi}(s)]$
  - This is the ***temporal difference learning rule***
  - $[\text{sample} - V^{\pi}(s)]$ is the "TD error"
  - α is the ***learning rate***
- Observe a sample, move $V^{\pi}(s)$ a little bit to make it more consistent with its neighbor $V^{\pi}(s')$

Example: Temporal Difference Learning

States

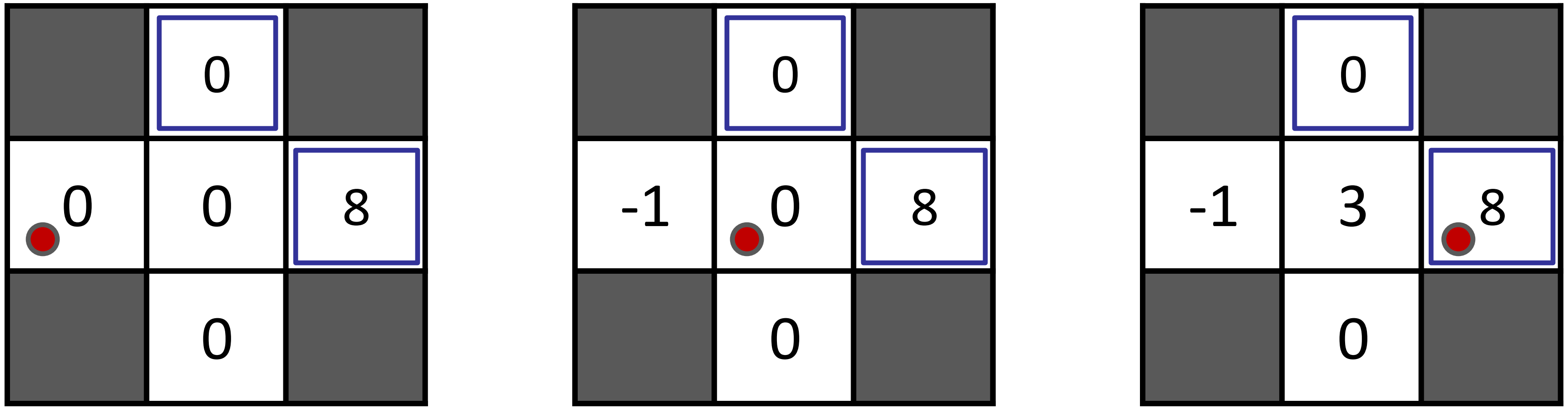


Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions

B, east, **C**, -2

C, east, **D**, -2



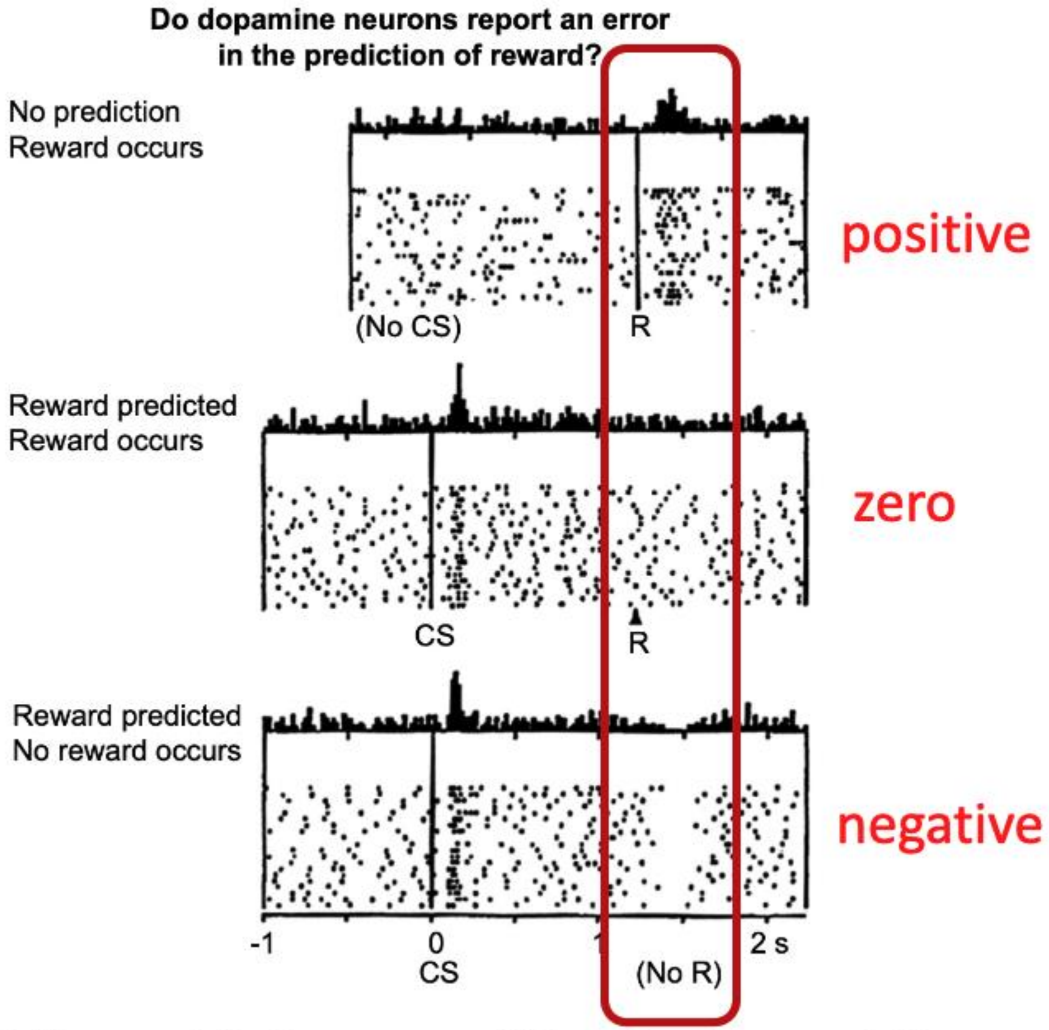
$$V^{\pi}(s) \leftarrow (1-\alpha)\, V^{\pi}(s) + \alpha \cdot [R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

TD Learning Happens in the Brain!

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

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

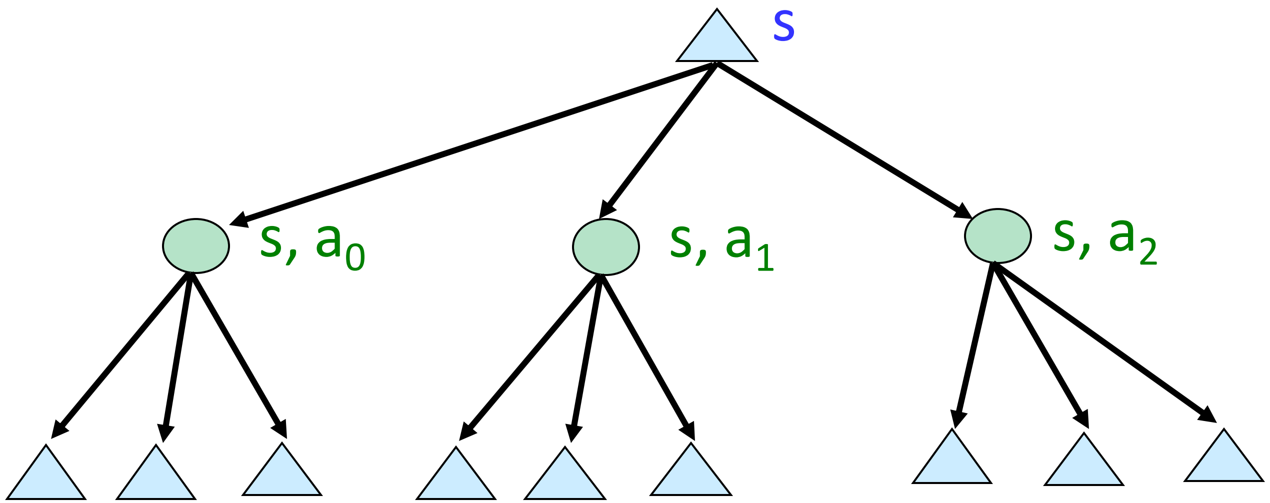
- Example of Neuroscience & AI informing each other



[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)

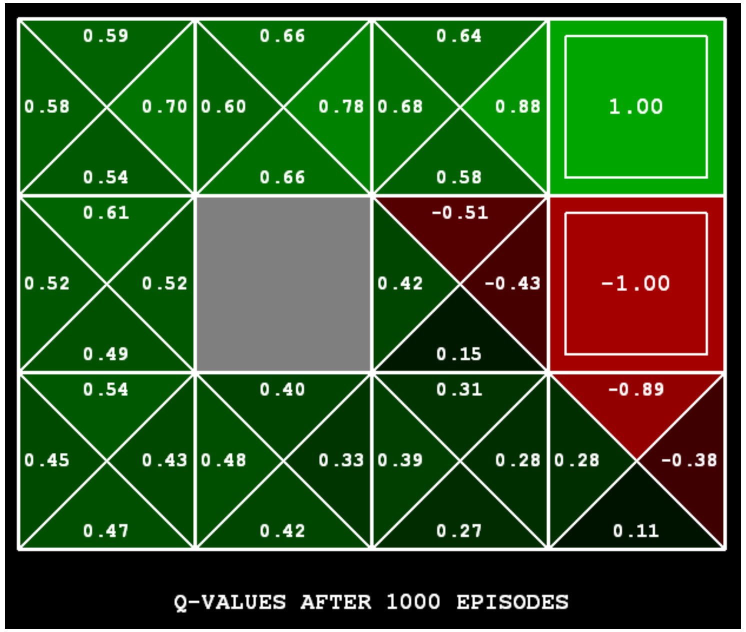
Q-Learning

- 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:

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

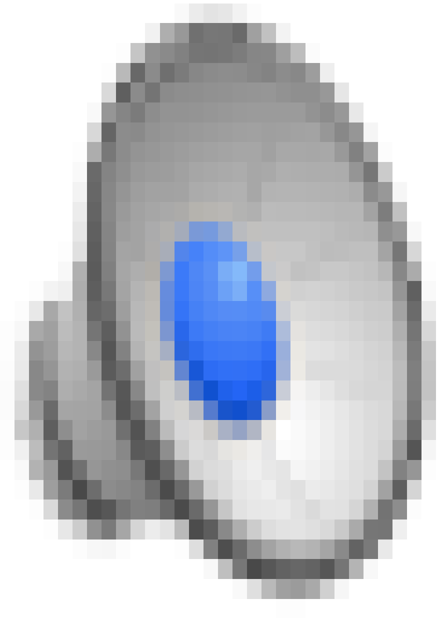
  - Incorporate the new estimate into a running average:

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

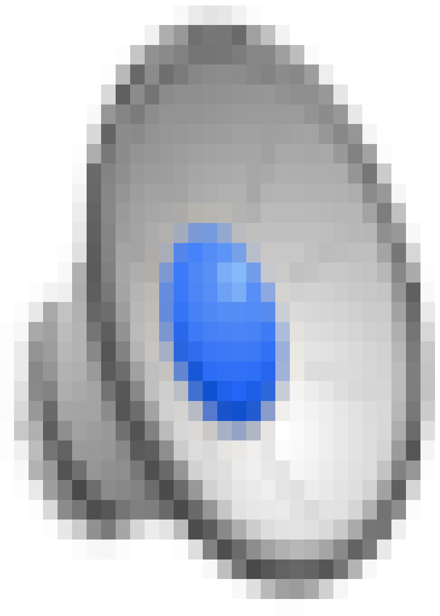


[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

Video of Demo Q-Learning -- Gridworld



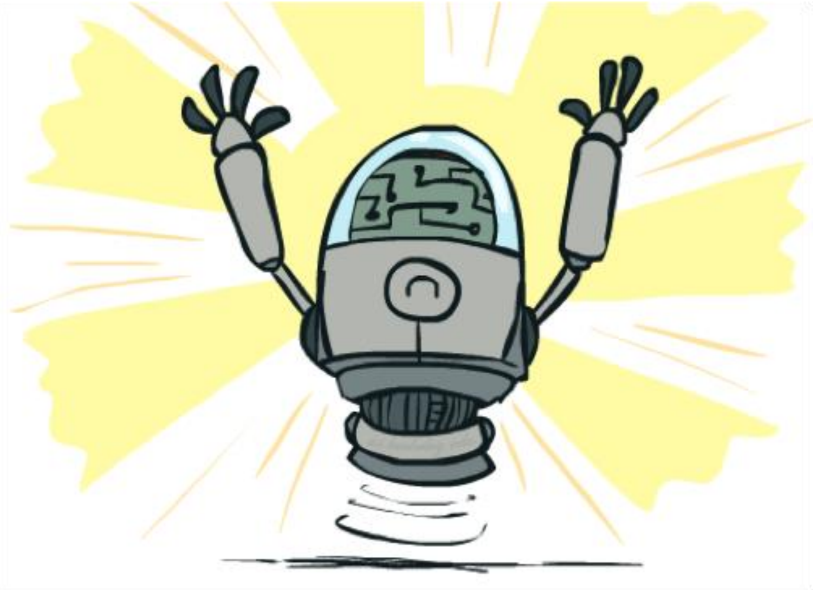
Video of Demo Q-Learning -- Crawler



Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if 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 (!)

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

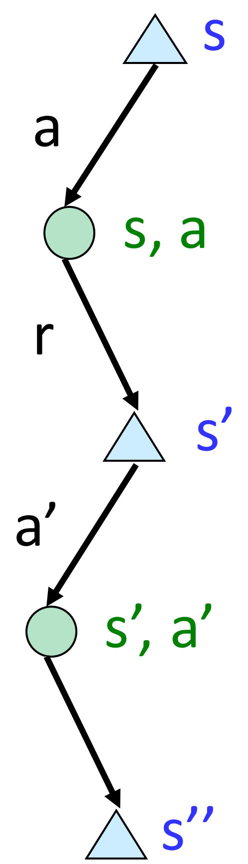
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	TD Value Learning

Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through episodes

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

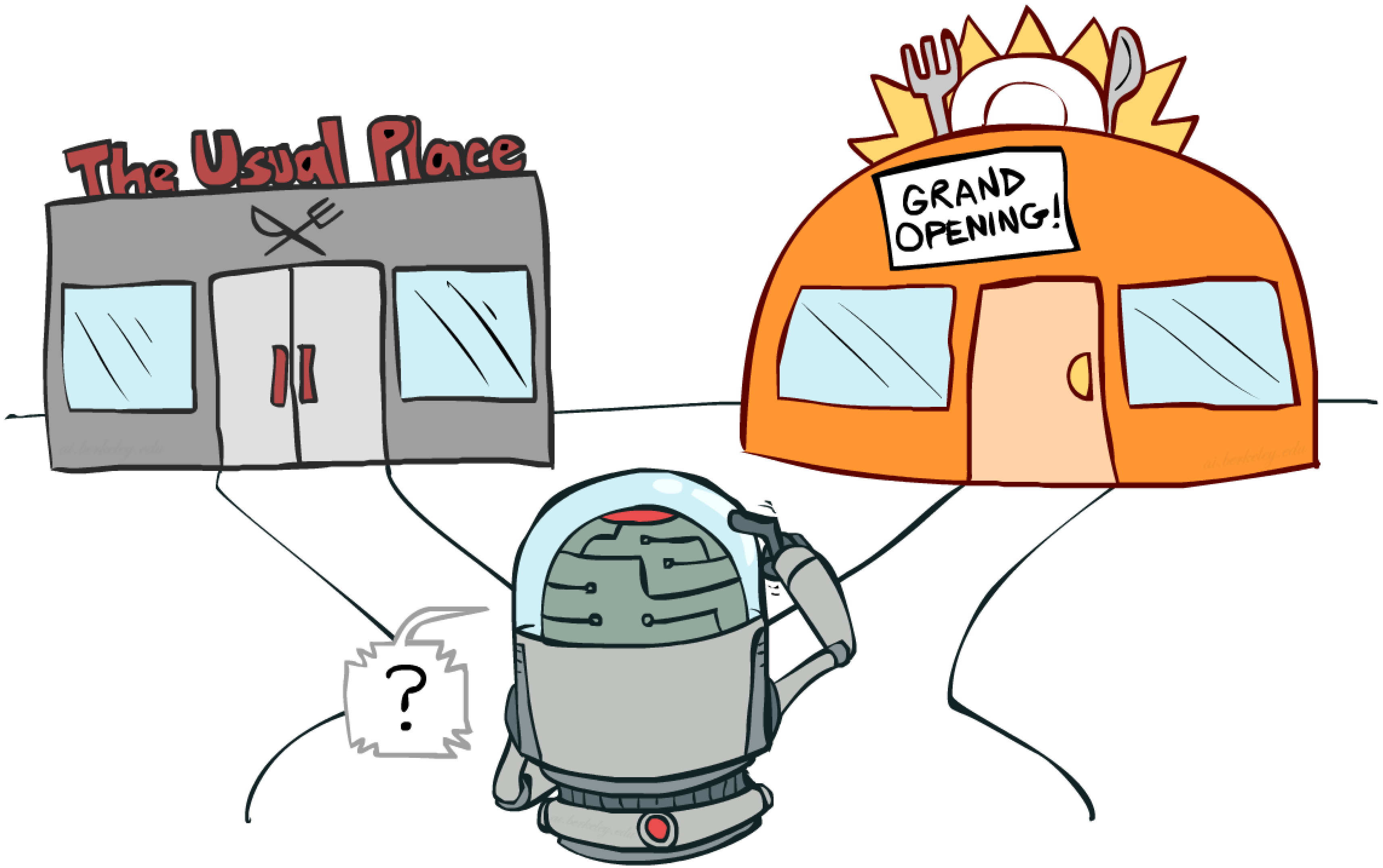
  - Update estimates each transition (s, a, r, s')
  - Over time, updates will mimic Bellman updates



Summary of previous RL knowledge

- 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
  - (and about 100 other variations)
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris (10^{60}), Go (10^{172}), StarCraft ($|A|=10^{26}$)?

Exploration vs. Exploitation



Exploration vs. Exploitation

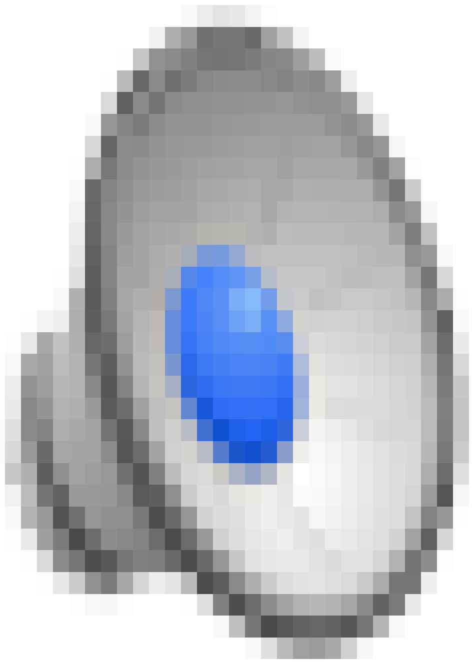
- ***Exploration***: try new things
- ***Exploitation***: do what's best given what you've learned so far
- Key point: pure exploitation often gets ***stuck in a rut*** and never finds an optimal policy!

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



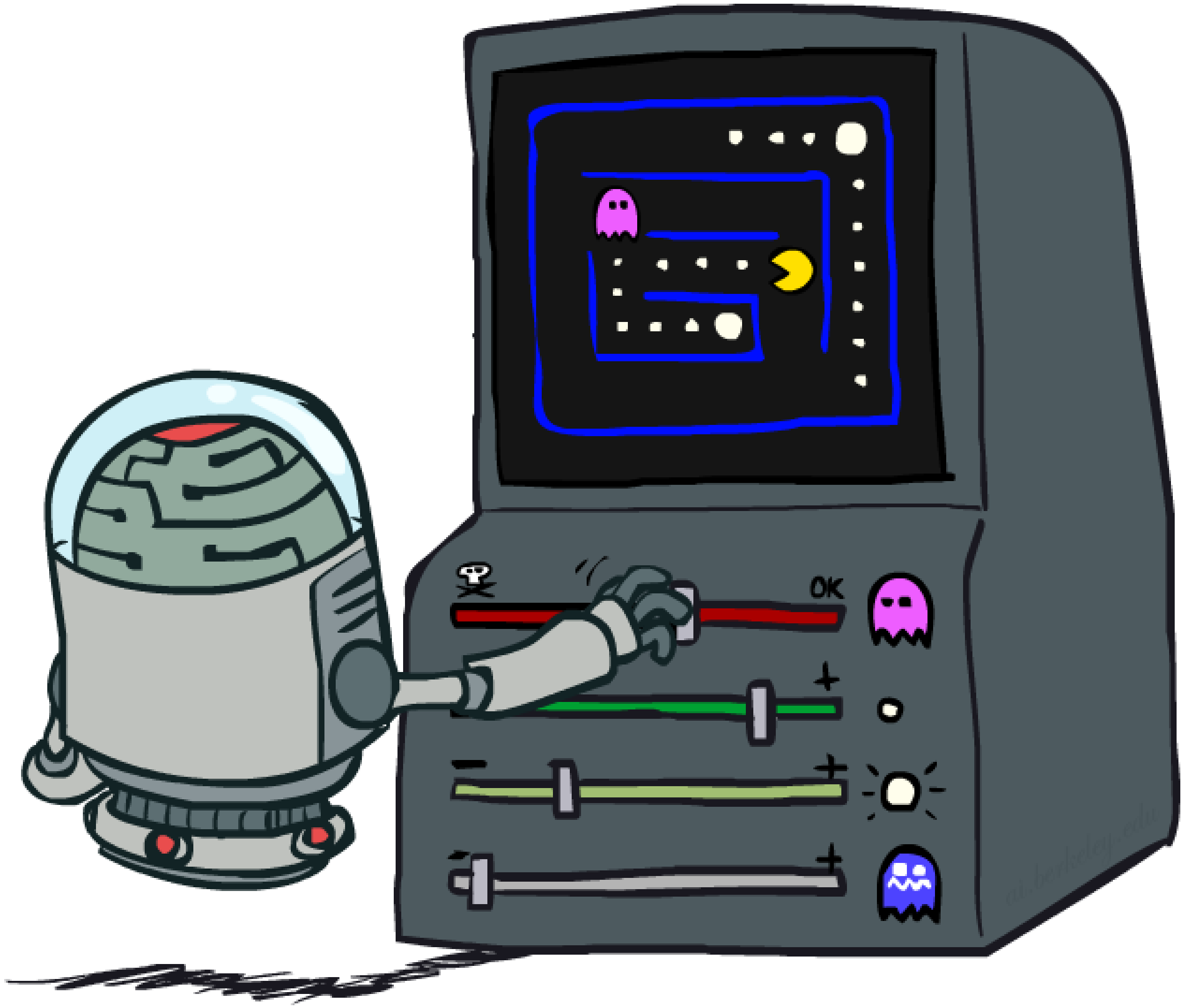
Method 2: Optimistic Exploration Functions

- ***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!

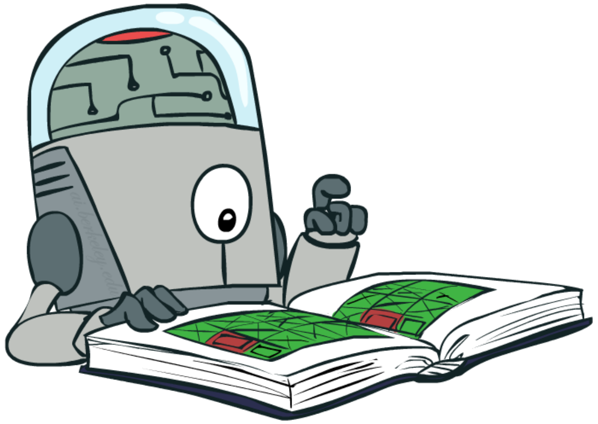
Demo Q-learning – Exploration Function – Crawler

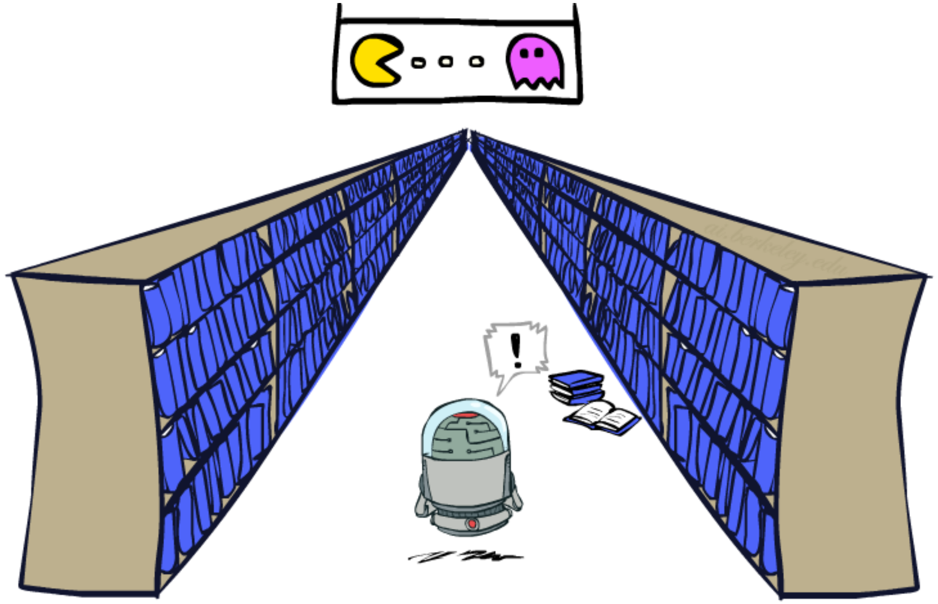
Approximate Q-Learning



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?

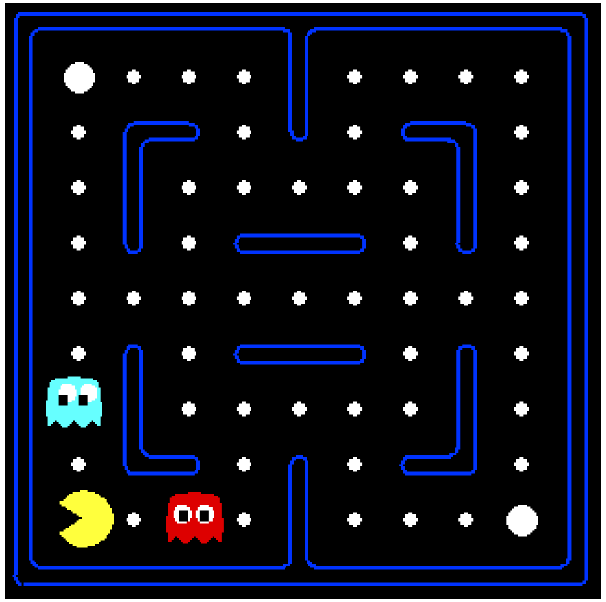




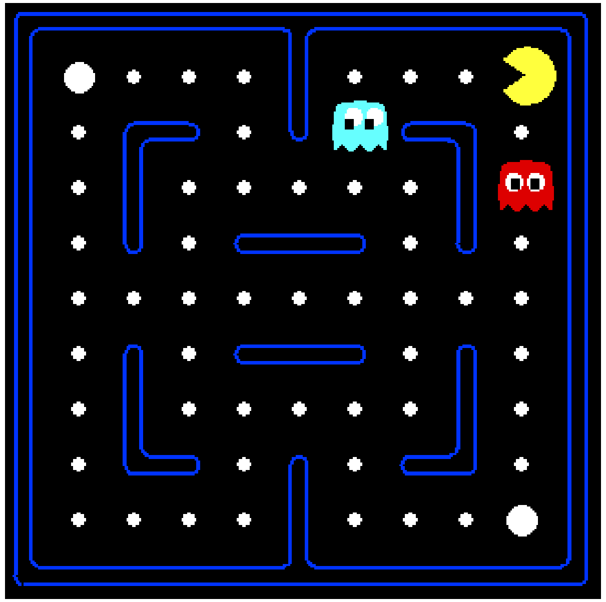
[demo – RL pacman]

Example: Pacman

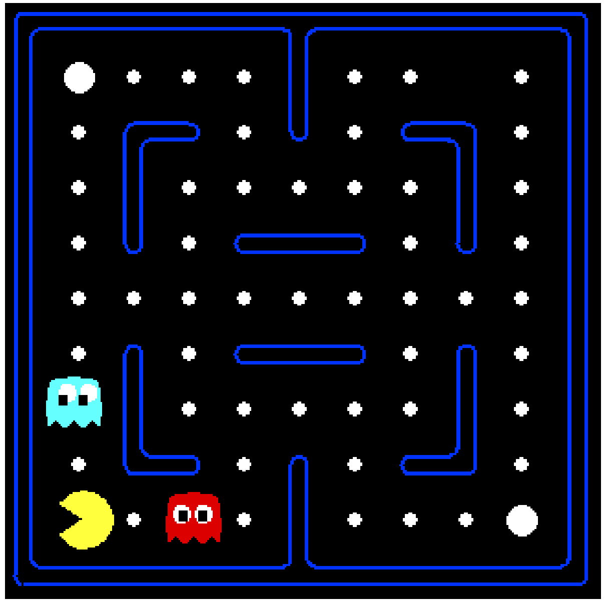
Let’s say we discover through experience that this state is bad:



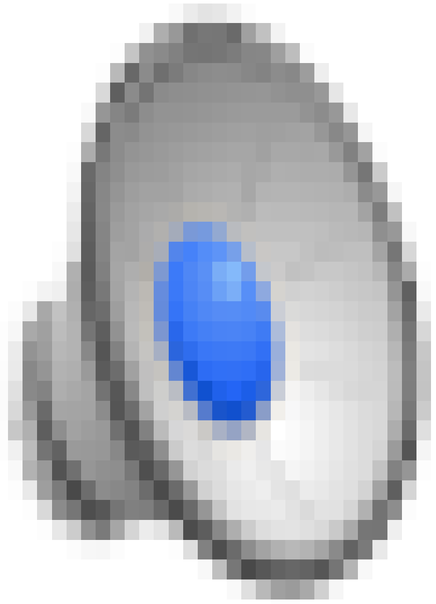
In naïve q-learning, we know nothing about this state:



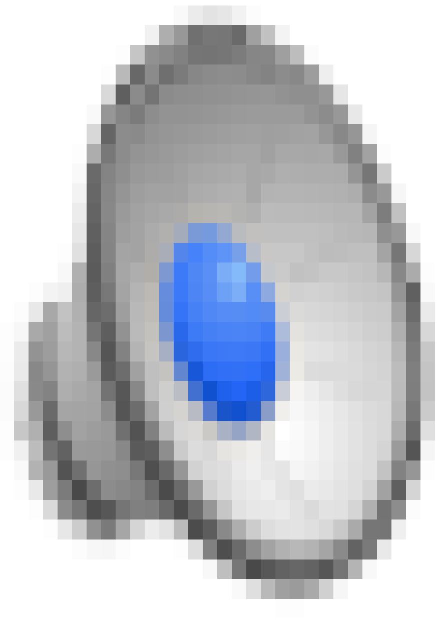
Or even this one!



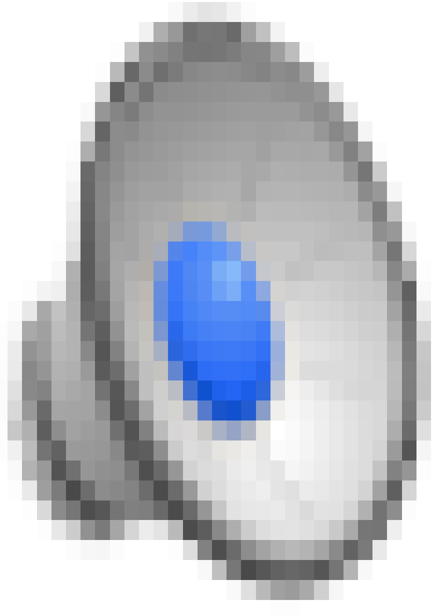
Demo Q-Learning Pacman – Tiny – Watch All



Demo Q-Learning Pacman – Tiny – Silent Train

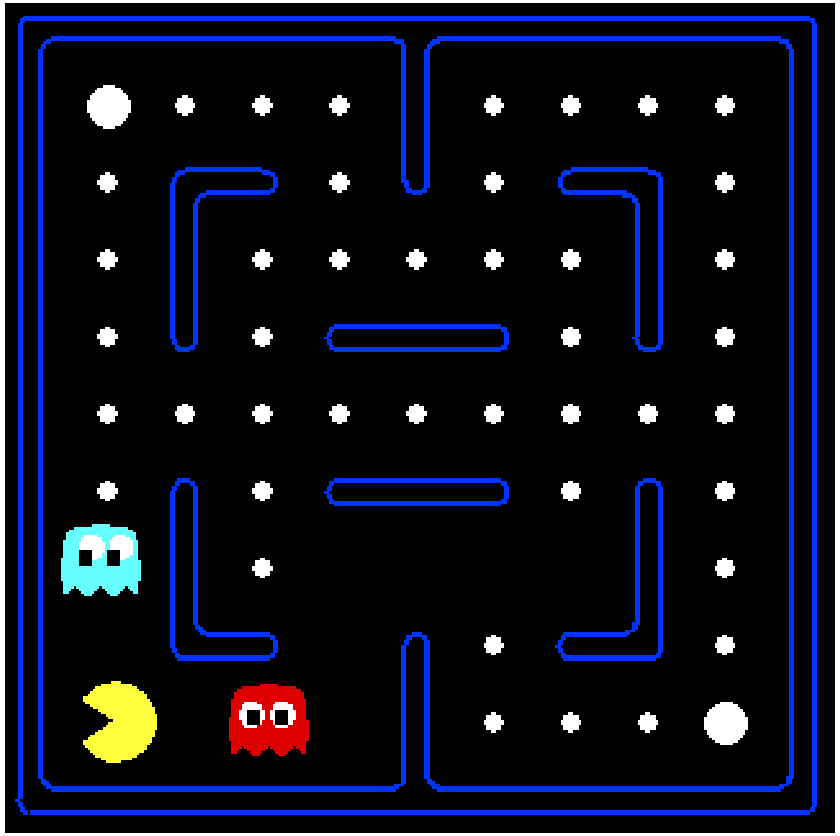


Demo Q-Learning Pacman – Tricky – Watch All



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

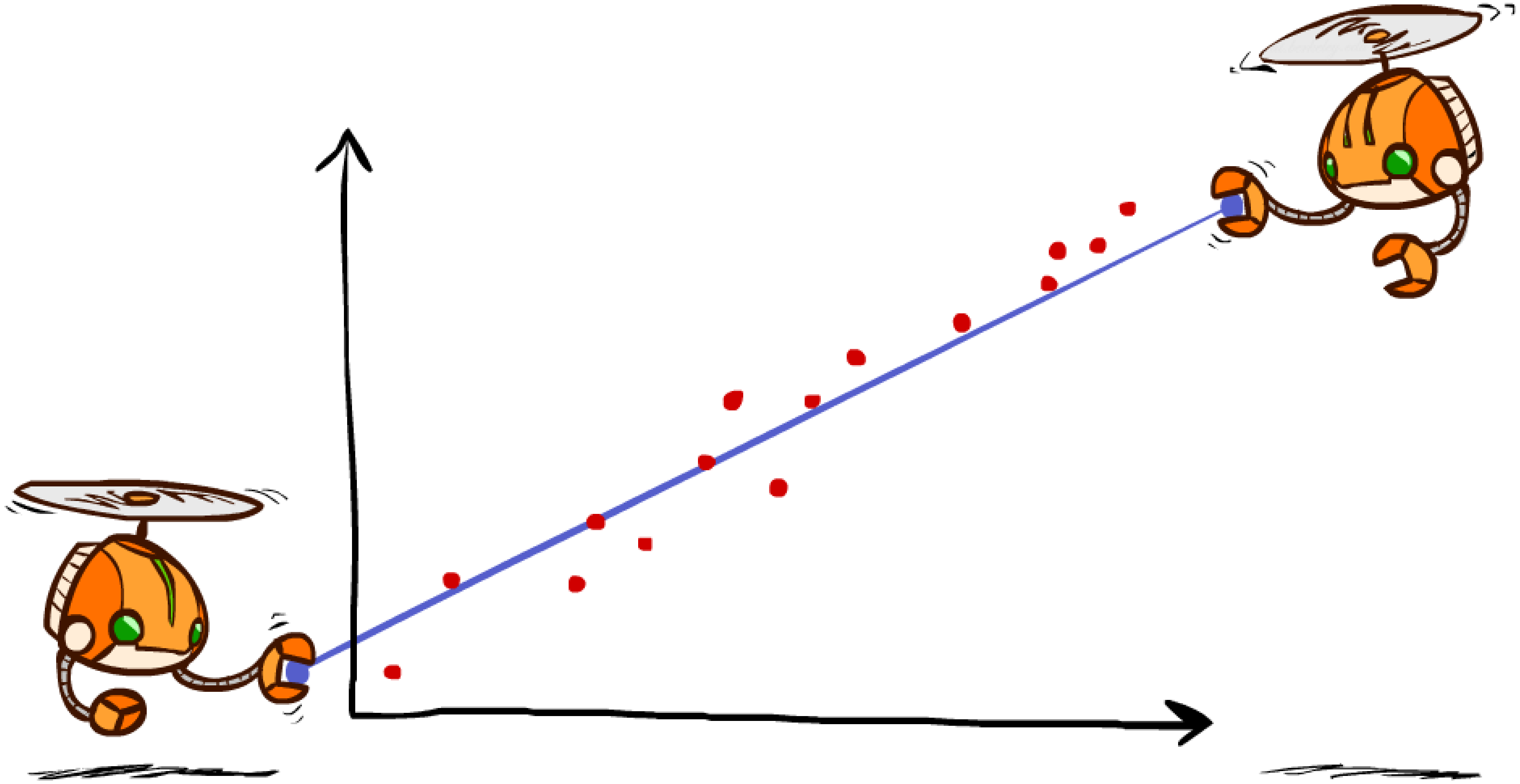
- Original Q-learning rule tries to reduce prediction error at s,a:
  - $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- Instead, we update the weights to try to reduce the error at s,a:

$$L = \frac{1}{2}(\text{diff})^2 \Longrightarrow w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$$

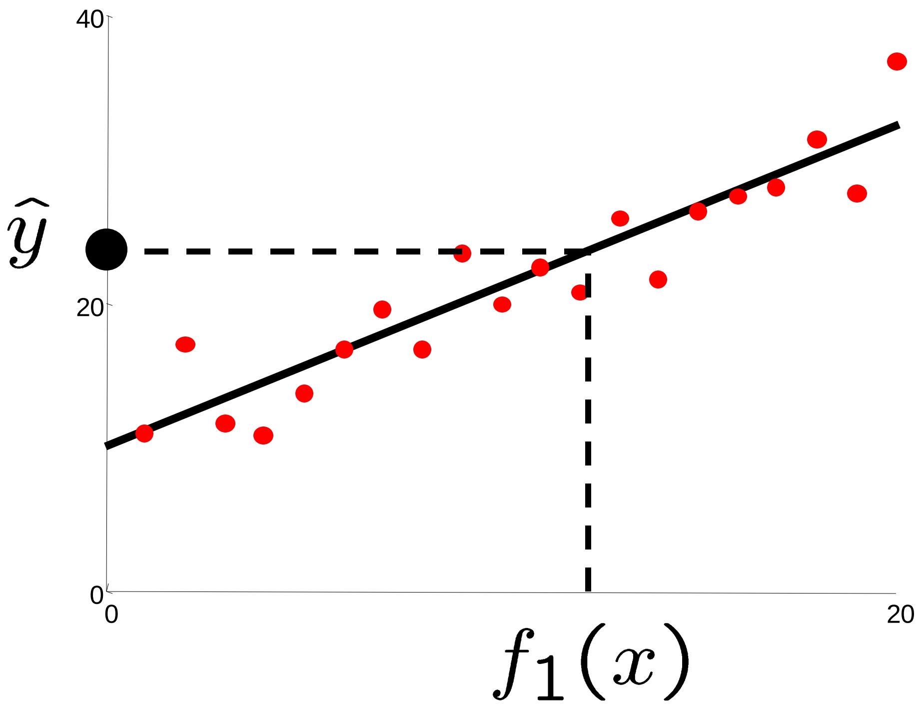
  - $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]\, \partial Q_w(s,a)/\partial w_i$

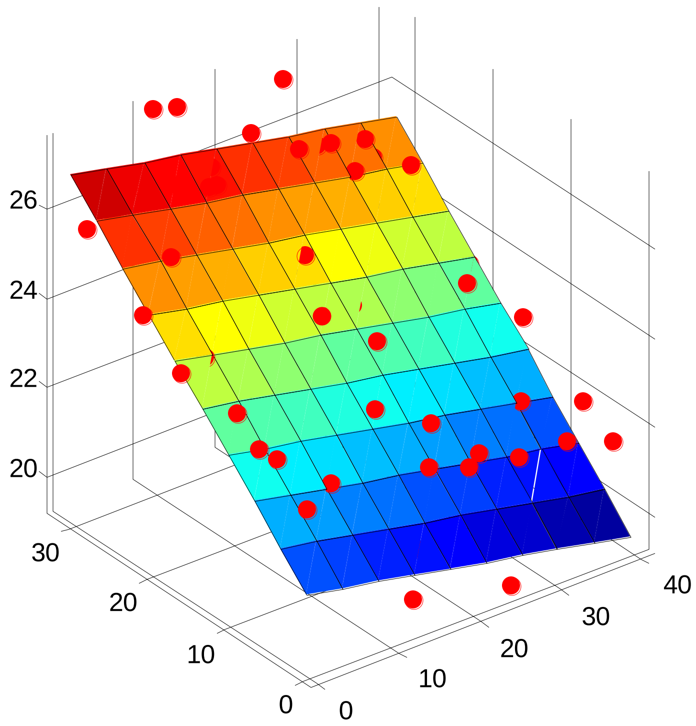
    $= w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]\, f_i(s,a)$
- Intuitive interpretation:
  - Adjust weights of active features
  - If something bad happens, blame the features we saw; decrease value of states with those features. If something good happens, increase value!

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

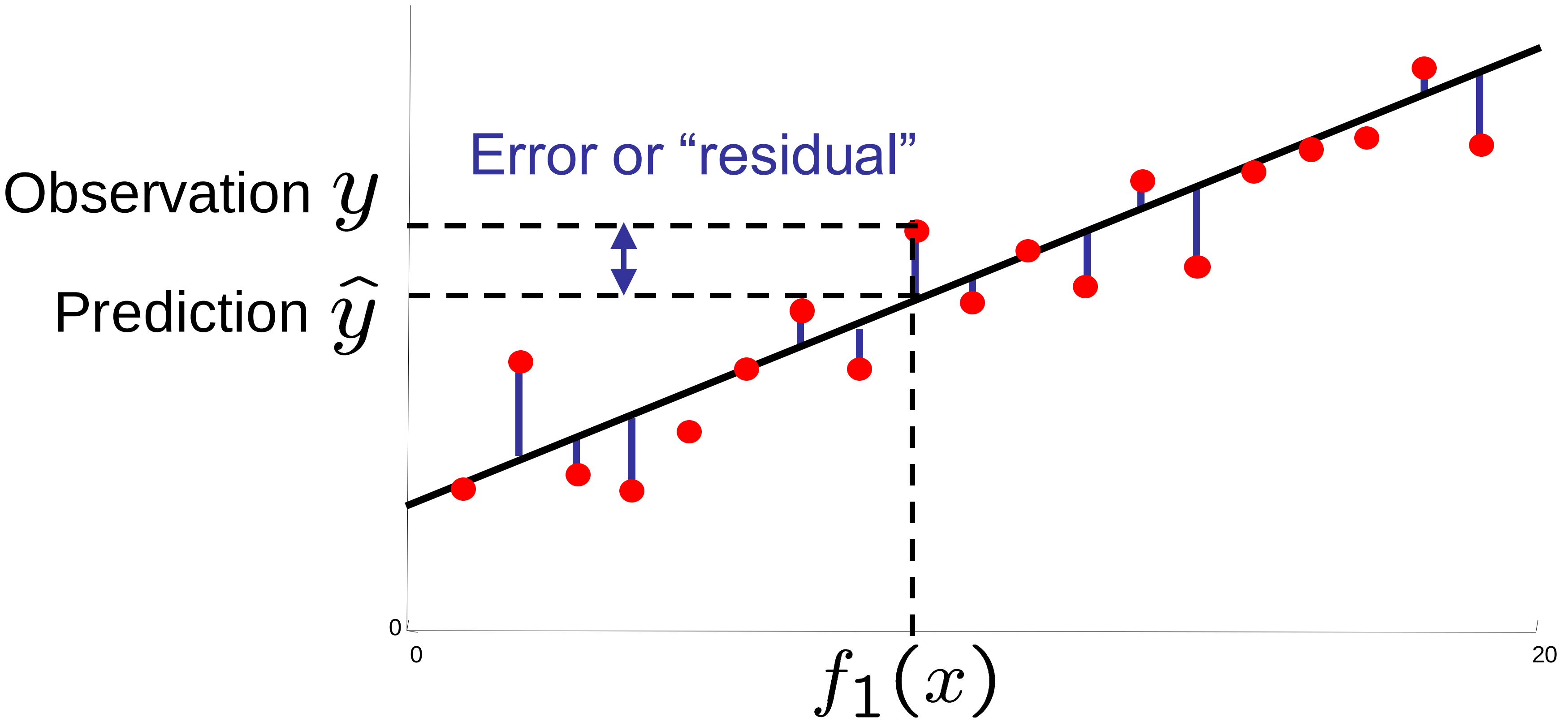
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

$$\hat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$$

Optimization: Least Squares*

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



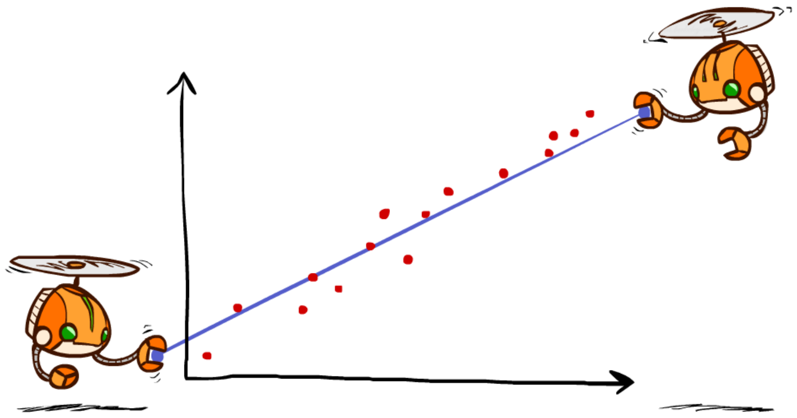
Minimizing Error*

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

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

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

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



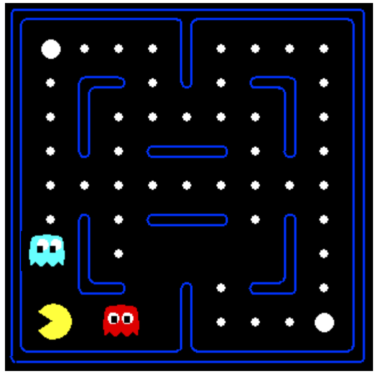
Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[\underbrace{r + \gamma \max_a Q(s', a')}_{\text{“target”}} - \underbrace{Q(s, a)}_{\text{“prediction”}} \right] f_m(s, a)$$

Example: Q-Pacman

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

s

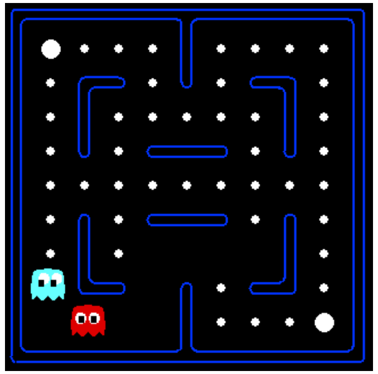


$f_{DOT}(s,NORTH) = 0.5$

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

a = NORTH
r = –500

s'



$Q(s,NORTH) = +1$

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

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

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

More Powerful Functions

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

Polynomial: $Q(s,a) = w_{11} f_1(s,a) + w_{12} f_1(s,a)^2 + w_{13} f_1(s,a)^3 + \ldots$

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

learn these too

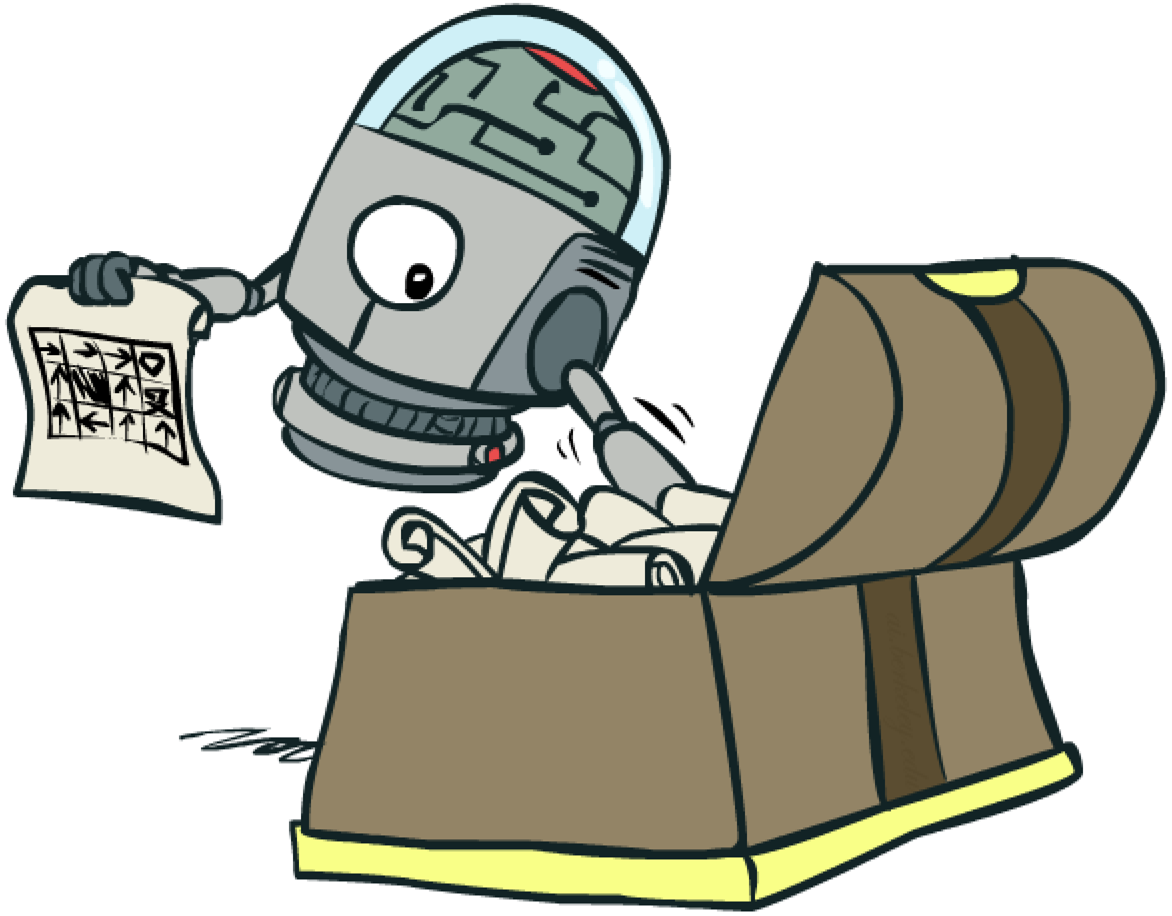
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s',a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$

$= f_m(s,a)$ in linear case

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]

Policy gradient

- We will cover it in “deep reinforcement learning” part.

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