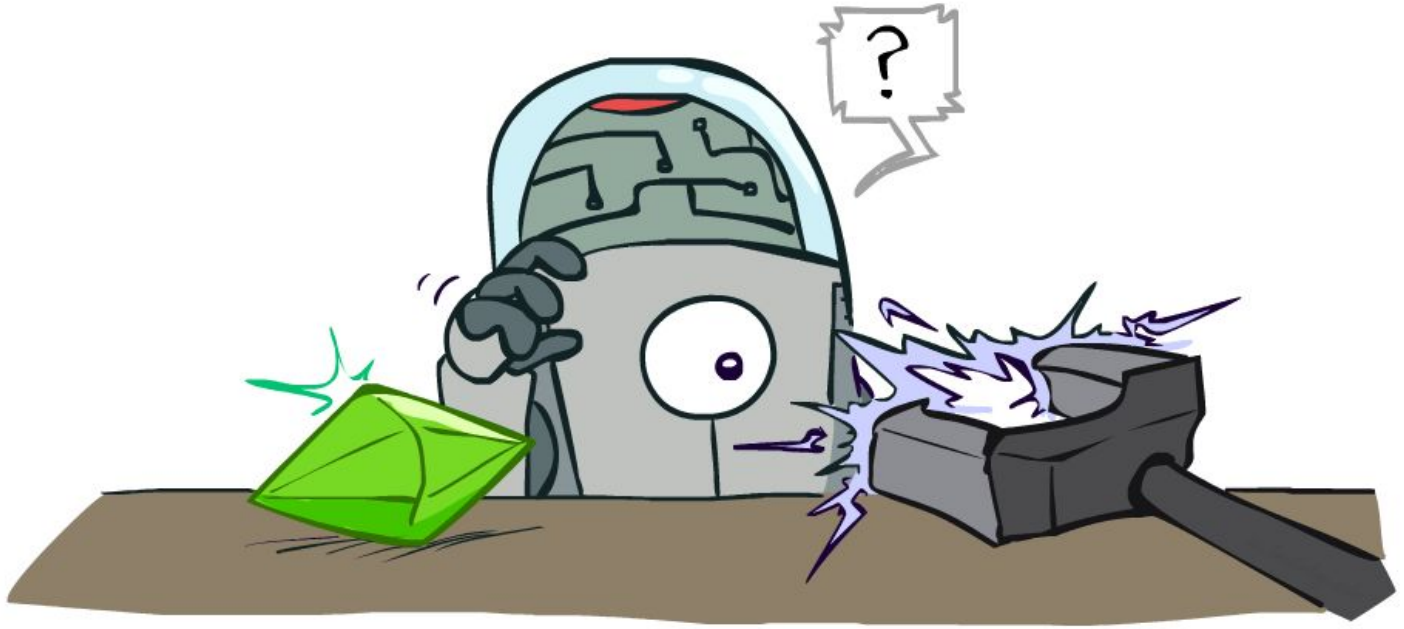
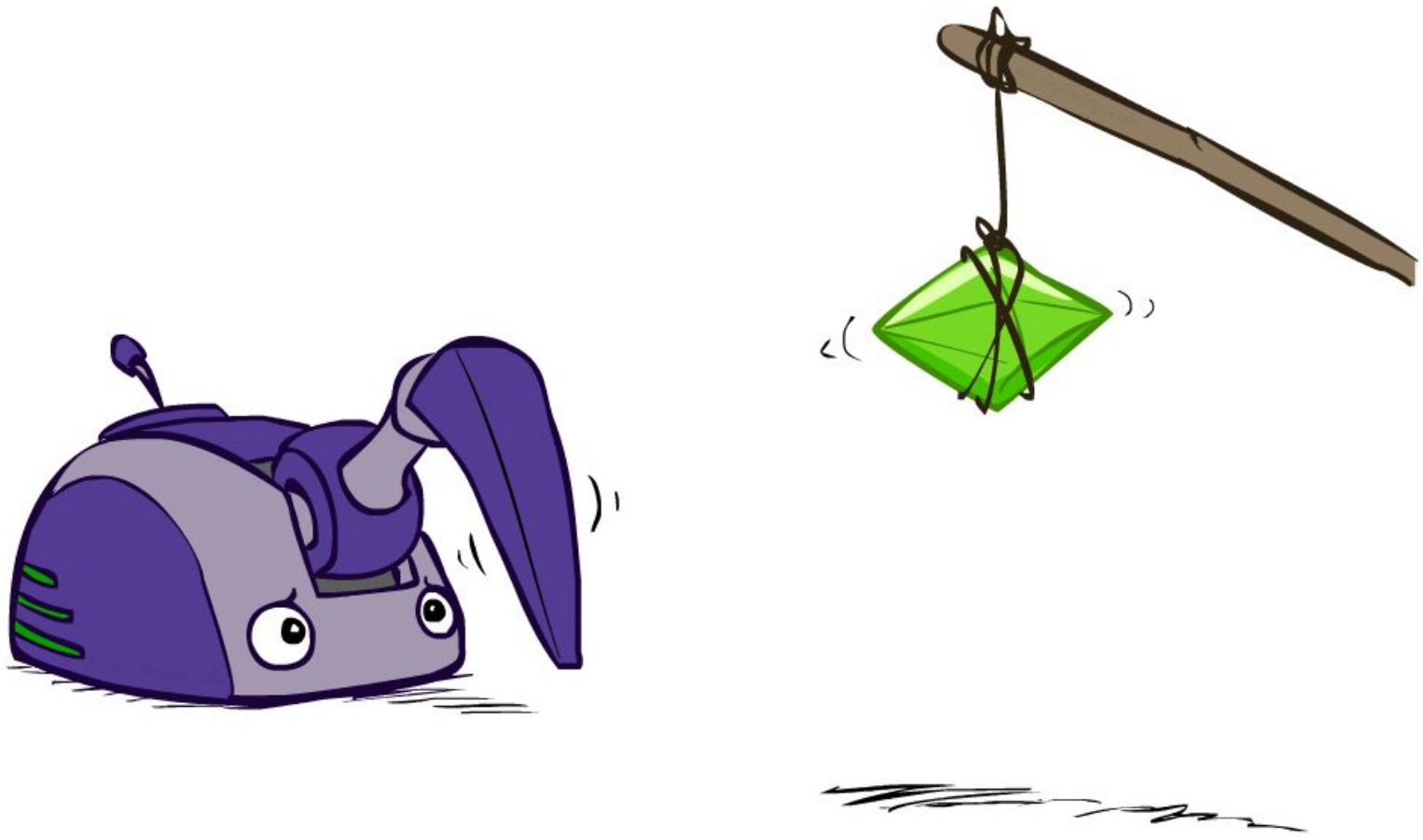
Reinforcement Learning



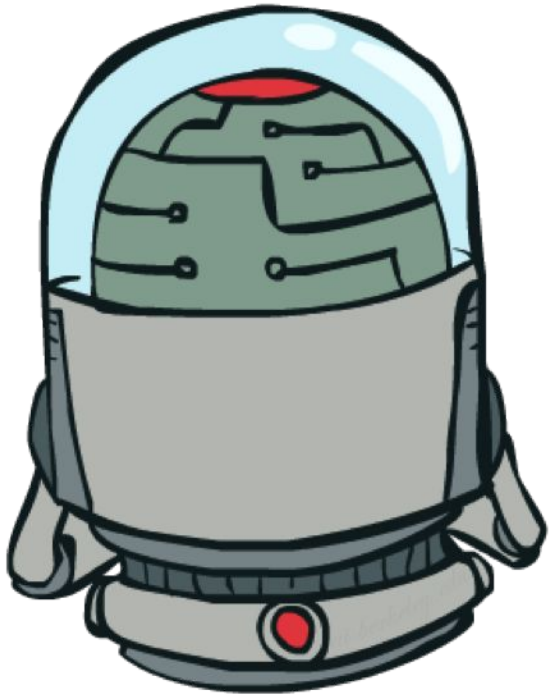
Instructor: Ngoc-Hoang LUONG, PhD.

[These slides were adapted from the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. http://ai.berkeley.edu.]

Reinforcement Learning

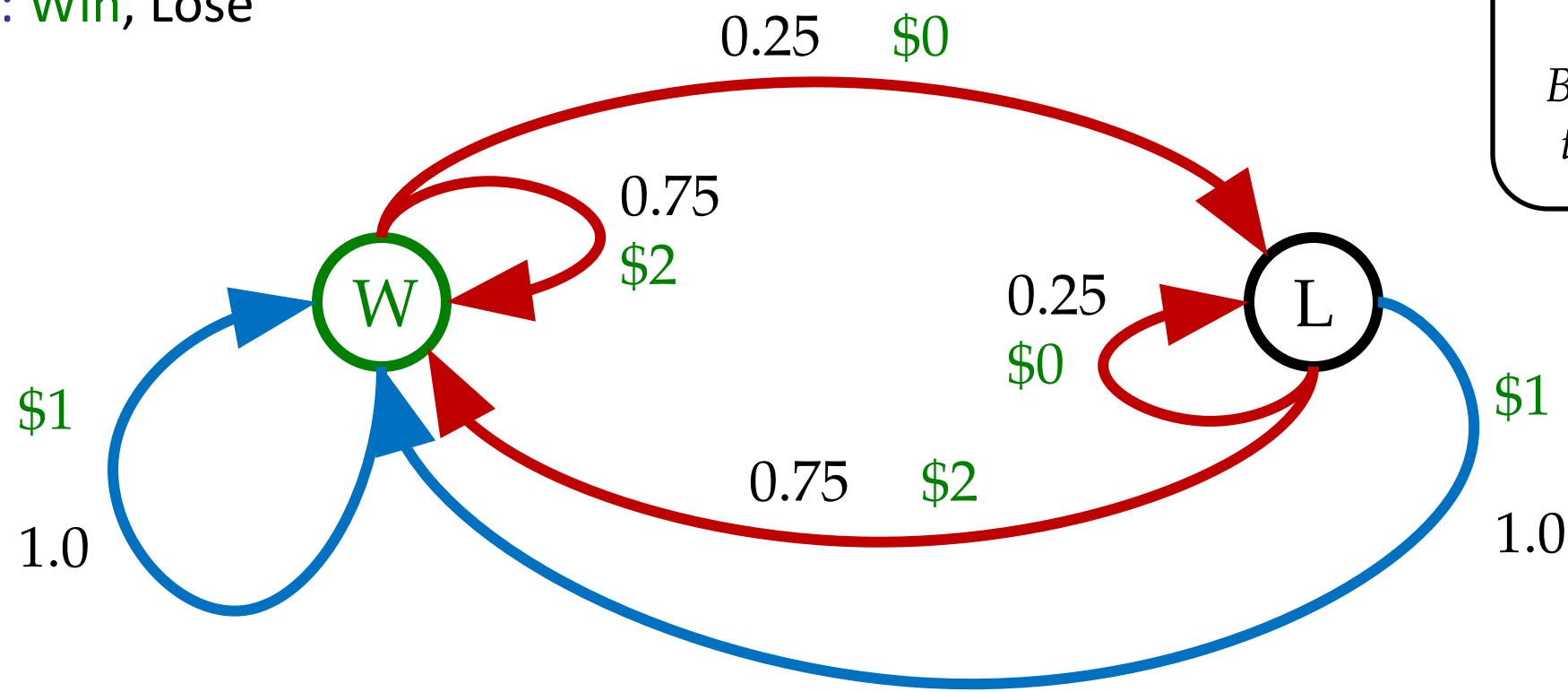


Double Bandits



Double-Bandit MDP

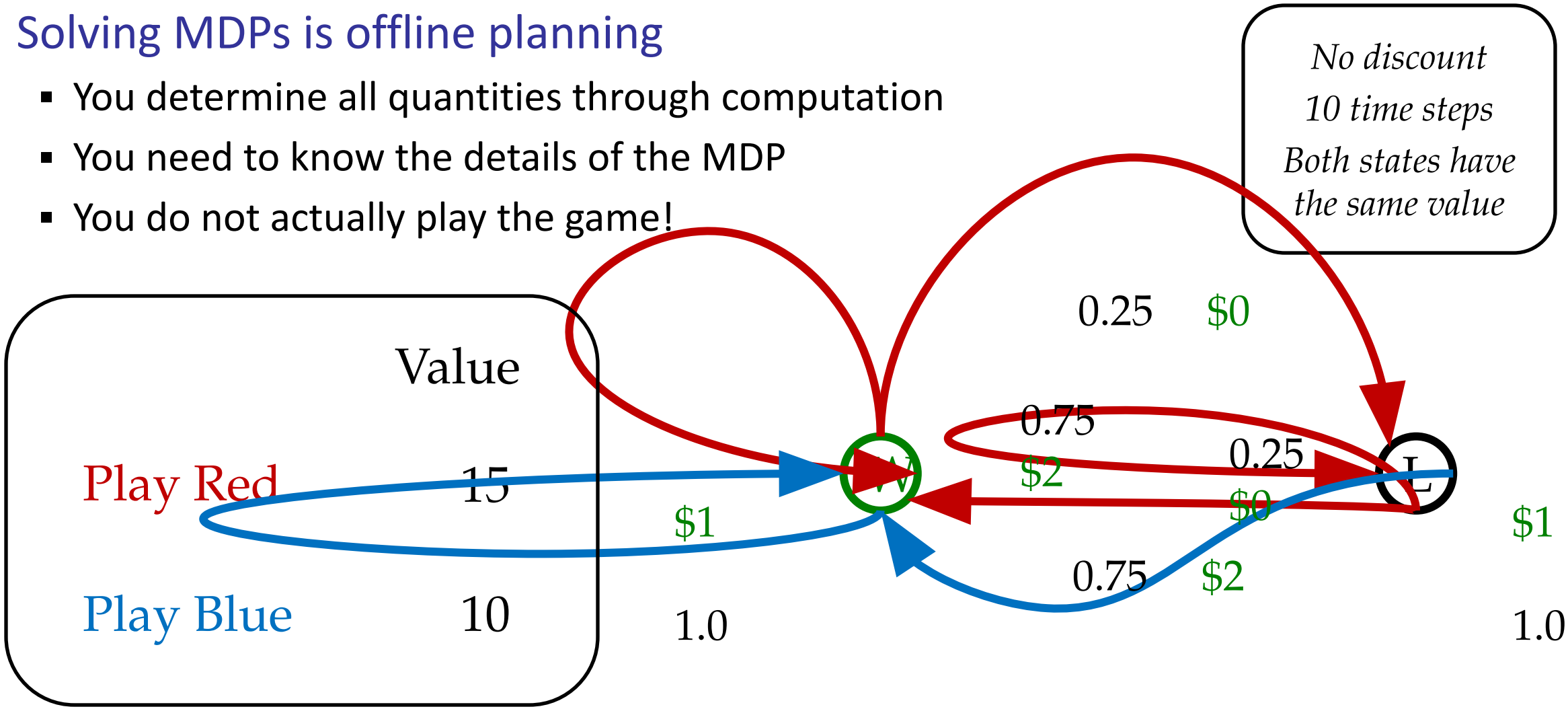
- Actions: *Blue, Red*
- States: Win, Lose



No discount
10 time steps
Both states have the same value

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!



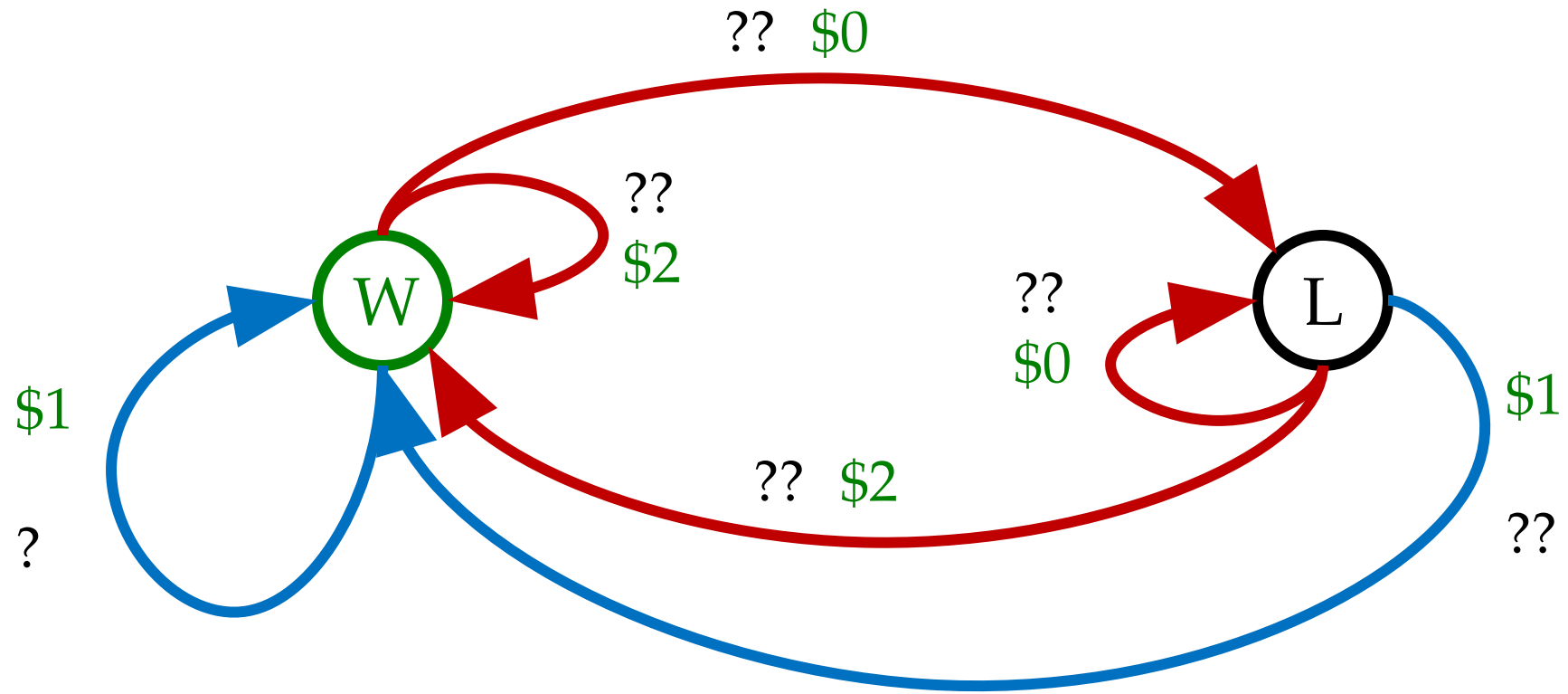
Let's Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

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



Let’s Play!

$1 $1 $1

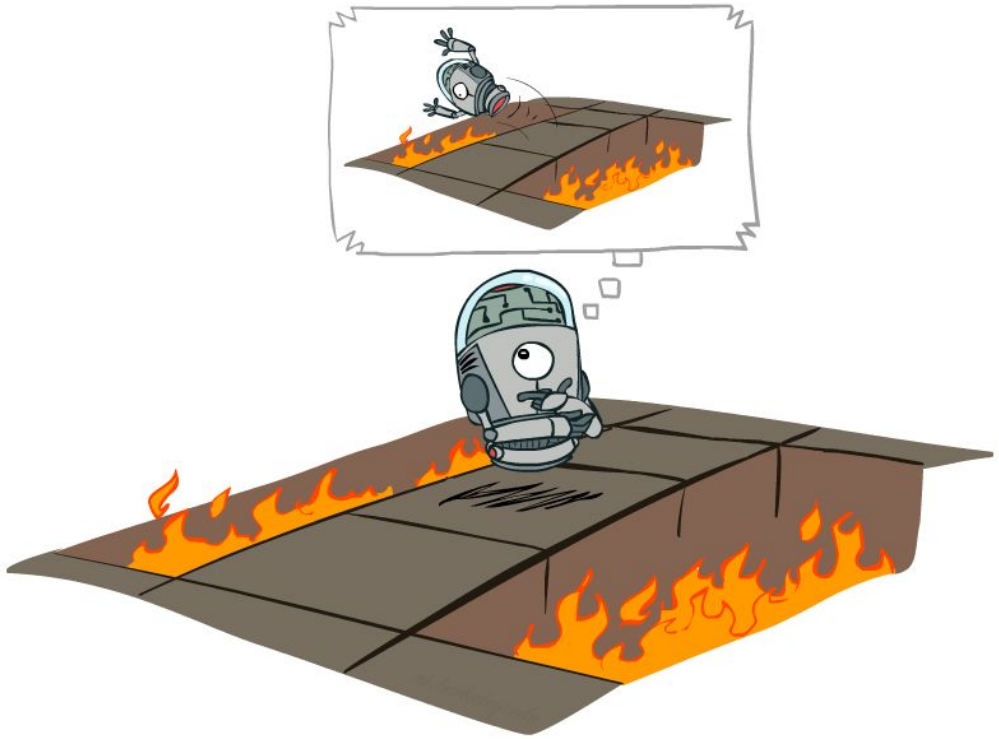
$0 $0 $0 $2

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

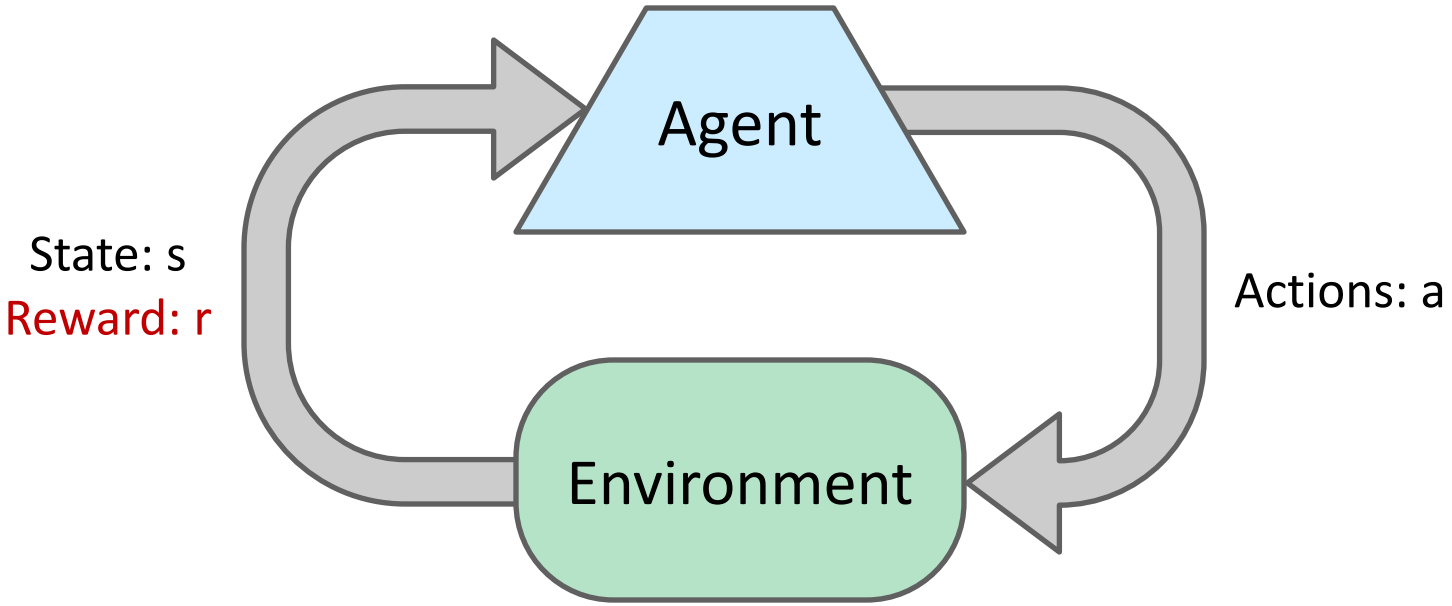
Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Reinforcement Learning



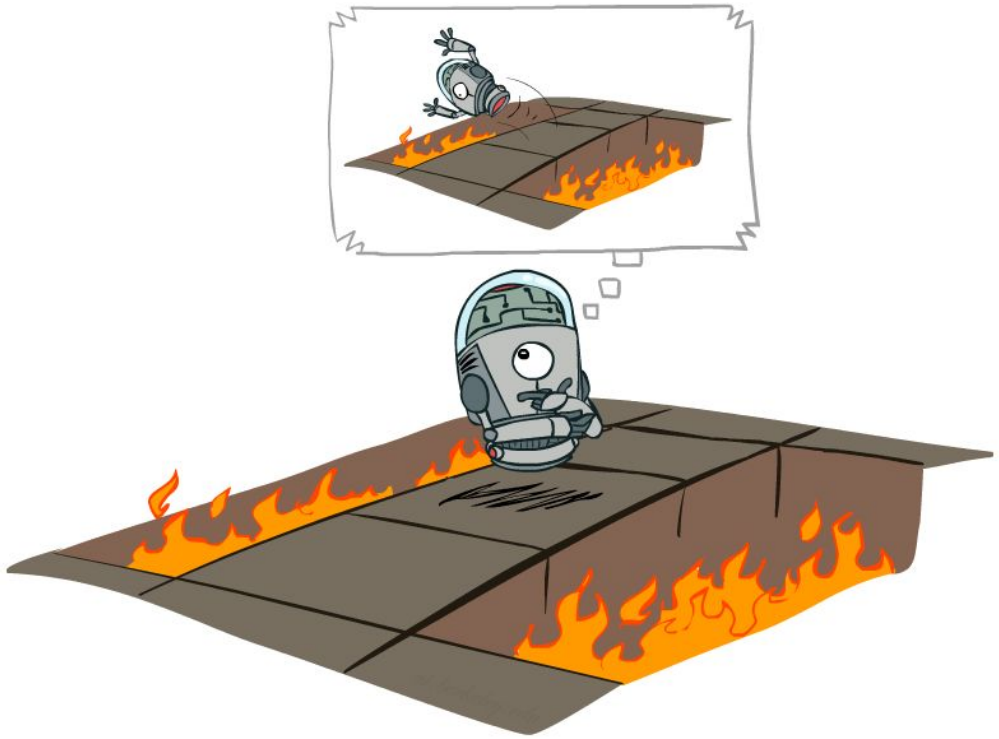
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

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 $T(s,a,s') = P(s'|s,a)$
  - 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

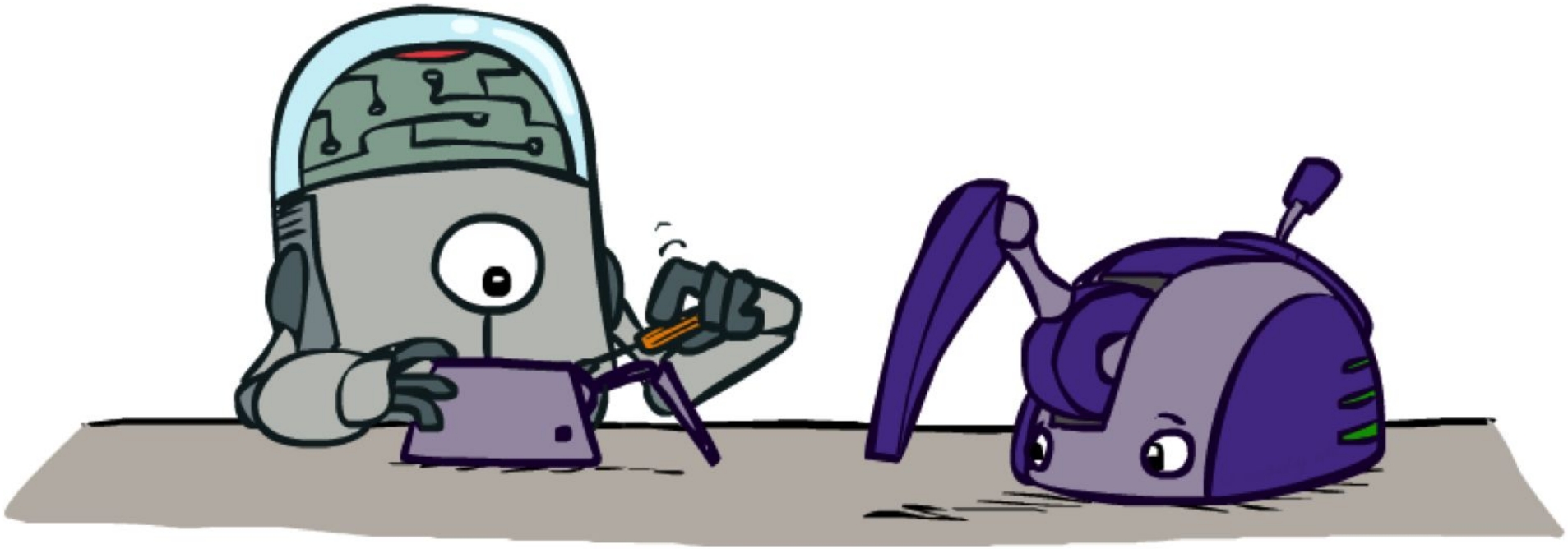
Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Model-Based Learning



Model-Based Learning

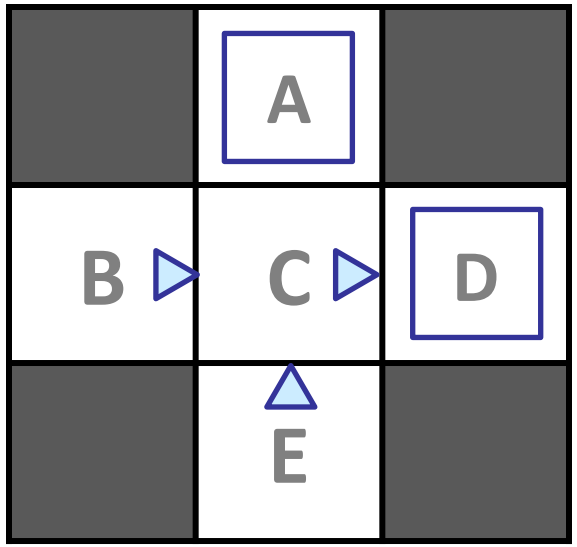
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- 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

Input Policy π



Assume: γ = 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 CS106 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"

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

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

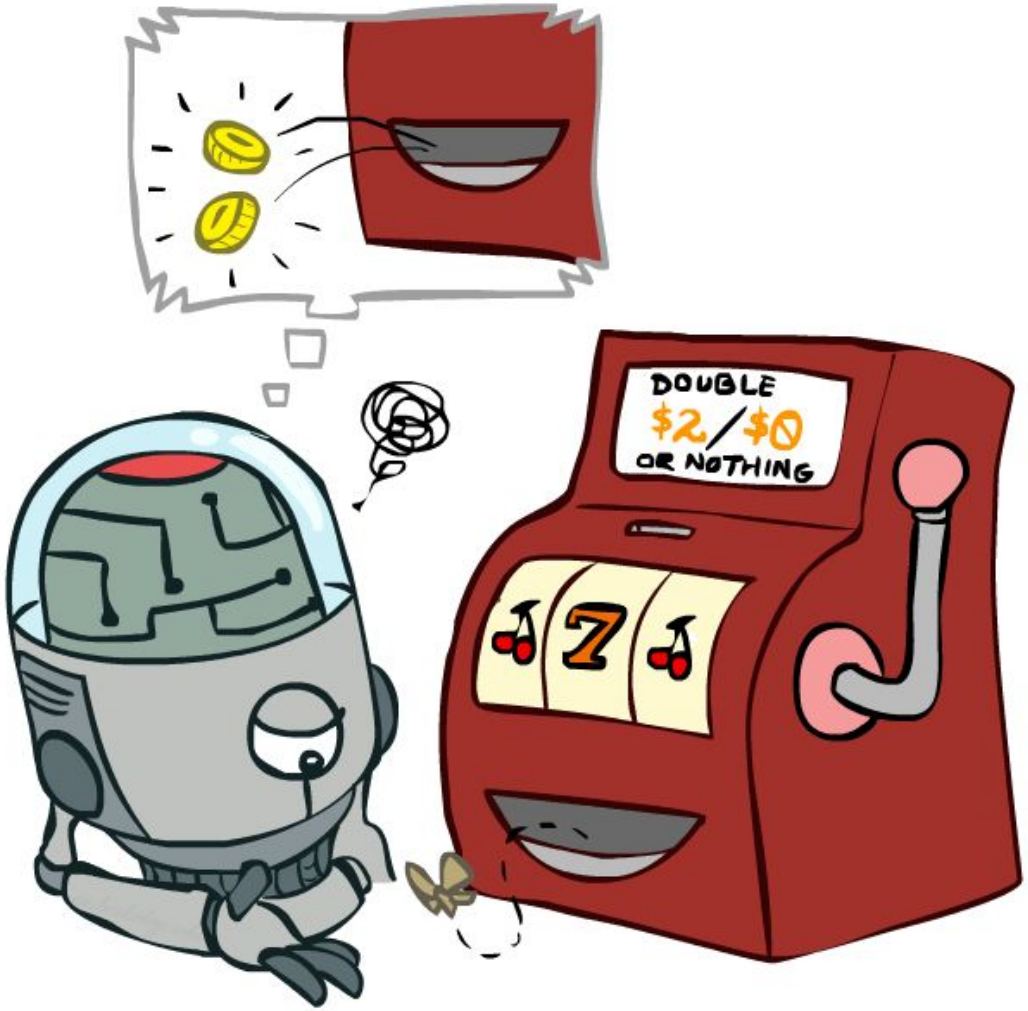
Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

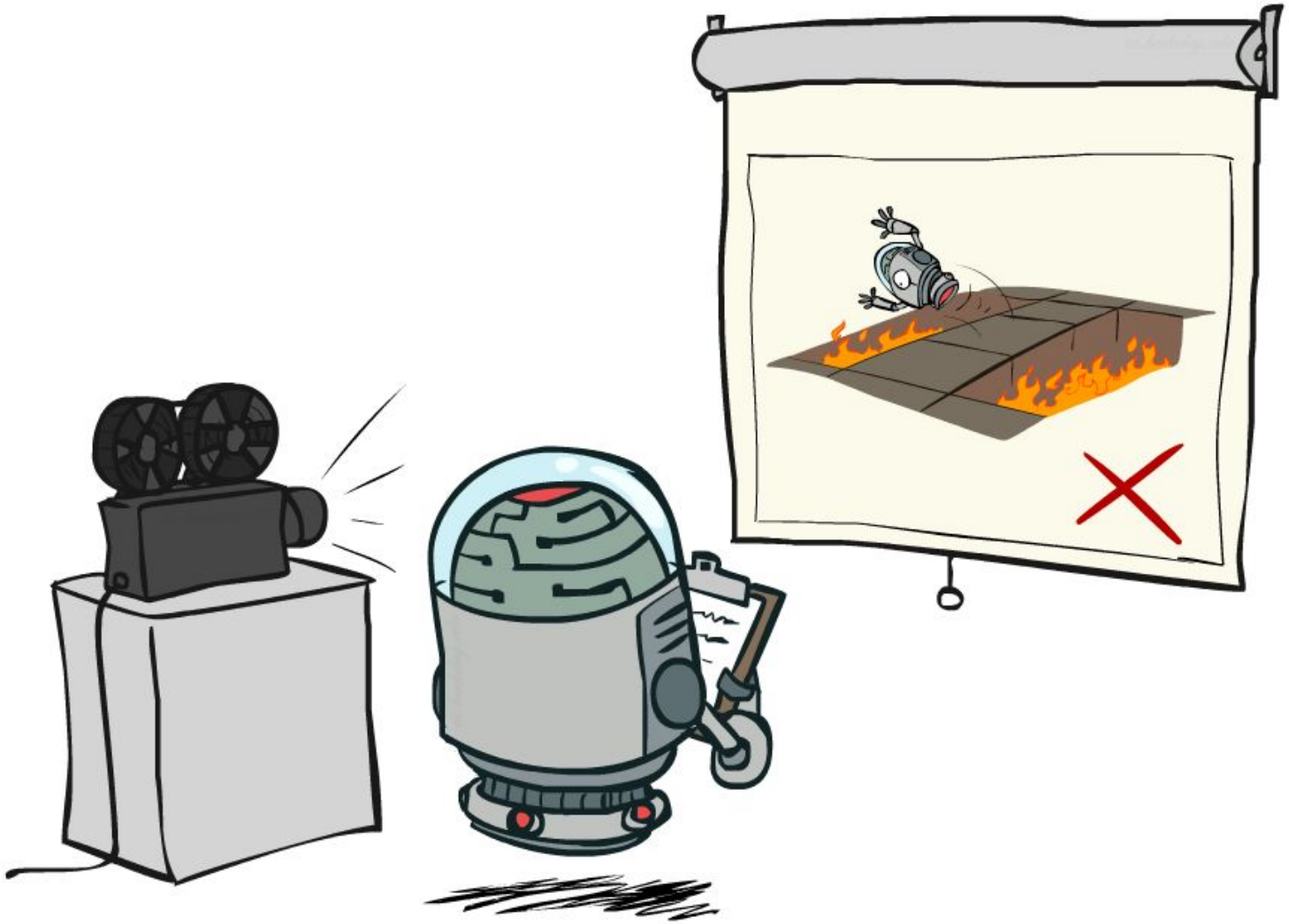
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

Model-Free Learning

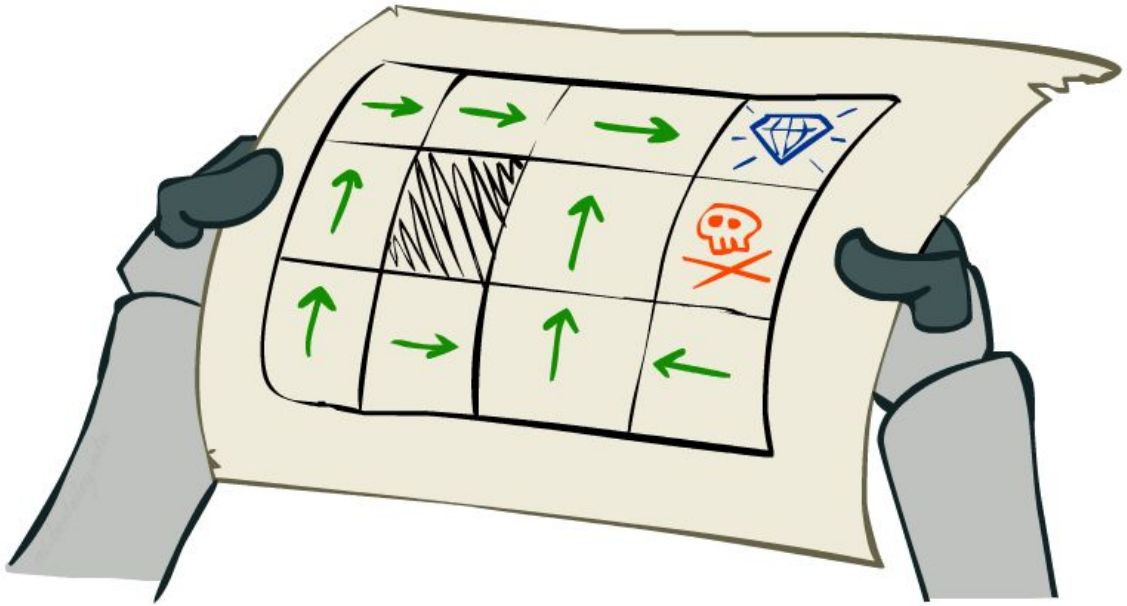


Passive Reinforcement Learning



Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy π(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.

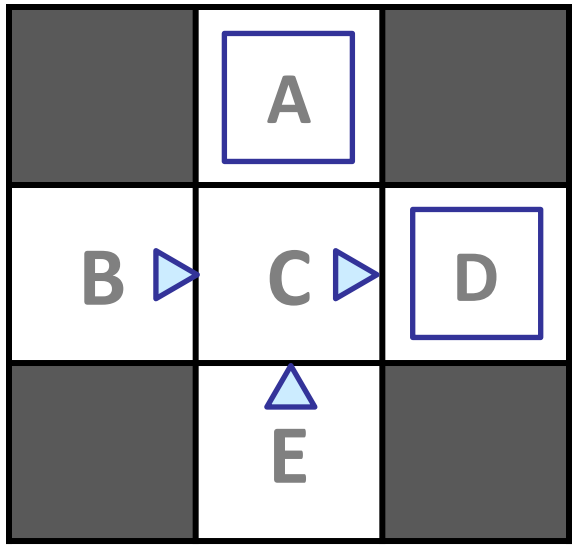
Direct Evaluation

- Goal: Compute $V^{\pi}(s)$ for each state s under given π
- Idea: Average together observed sample values
  1. Act according to π for several episodes
  2. Afterward, for every state s and every time t that s is visited: determine the rewards $r_{t+1}, r_{t+2}, \ldots, r_T$ subsequently received in the episode.
  3. Sample for s at time t = sum of discounted future rewards.
     $$sample = G_t(s) = r_{t+1} + \gamma R_{t+2}(s')$$
  4. Average those samples.

- This is called direct evaluation

Example: Direct Evaluation

Input Policy π



Assume: γ = 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

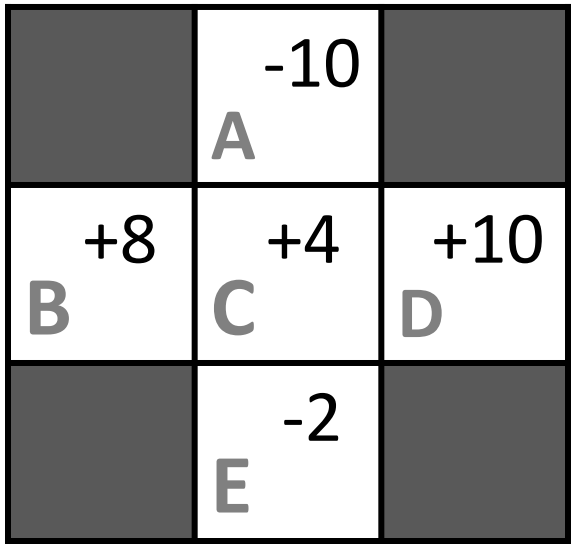
Episode 3

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

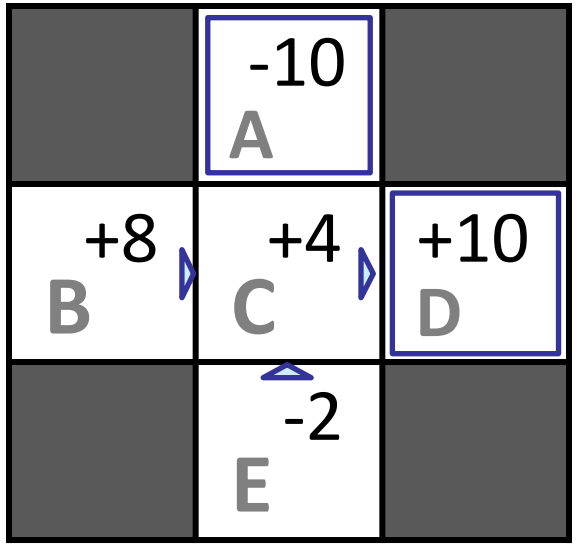
Output Values



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?

First-visit Monte Carlo Policy Evaluation

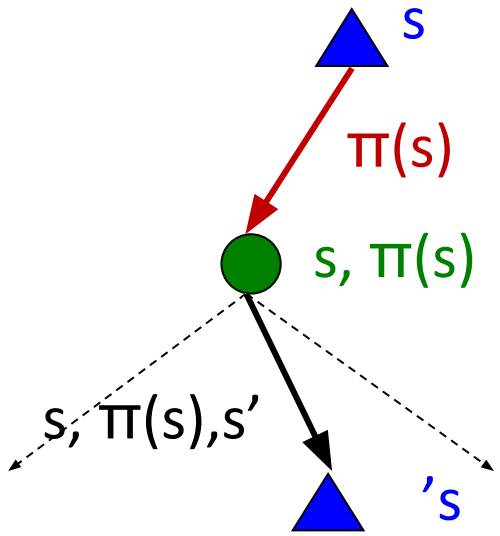
- Input: a policy π to be evaluated.
- Initialize:
  - $V(s)$ arbitrarily
  - $Returns(S_t) \leftarrow$ an empty list
- Loop forever (for each episode):
  - Generate an episode following π: $S_0, A_0, R_1, S_1, A_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  - $G \leftarrow 0$
  - Loop for each step of episode, $t = T-1, T-2, \ldots, 0$
    - $G \leftarrow R_{t+1} + \gamma G_{t+1}$
    - Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

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?

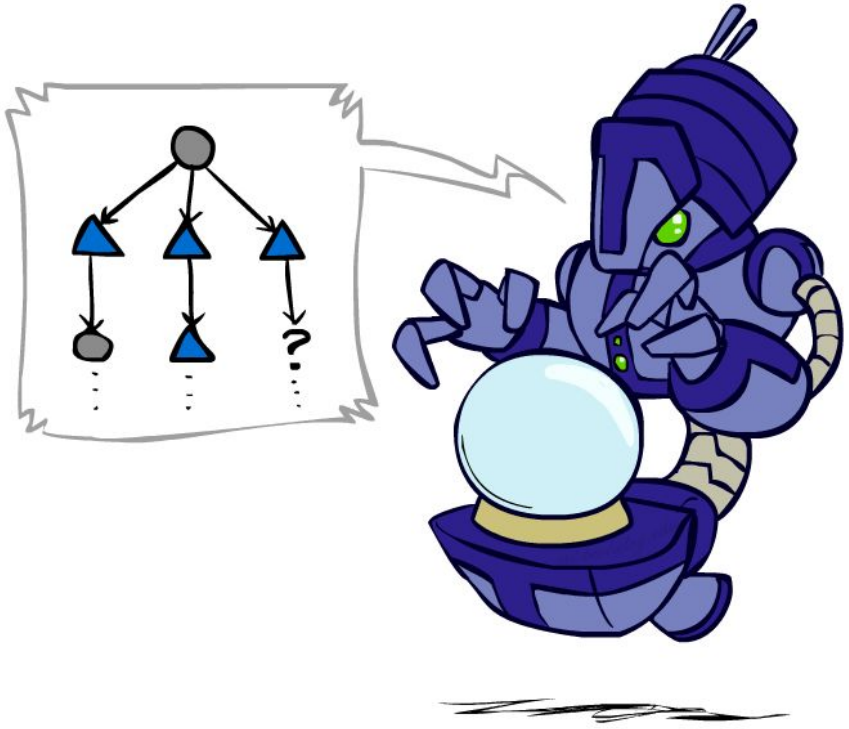
- We want to improve our estimate of V by computing these averages:

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

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

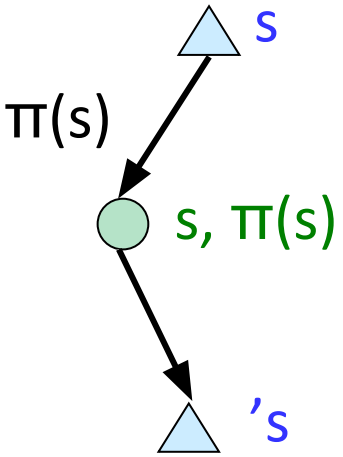
$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\ldots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$



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

Temporal Difference Learning

- Goal: Compute $V^{\pi}(s)$ for each state s under given π
- Idea: update after every experience (s, a,r, s').

→Likely outcomes s' will contribute updates more often

1. Initialize each $V^{\pi}(s)$ with some value.
2. Observe experience (s, π (s), r, s').
3. Use observation in rough estimate of long-term reward $V^{\pi}(s)$

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

4. Update $V^{\pi}(s)$ by moving values slightly toward estimate:

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

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:

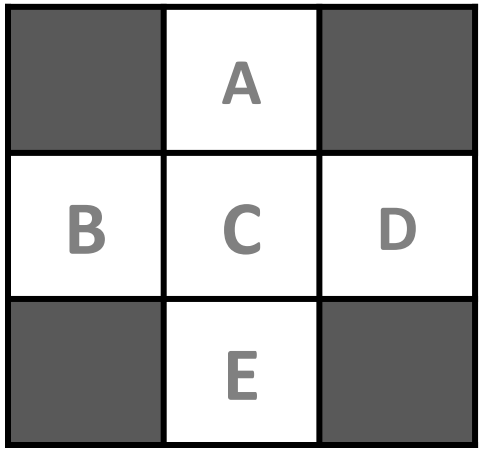
$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

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

- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

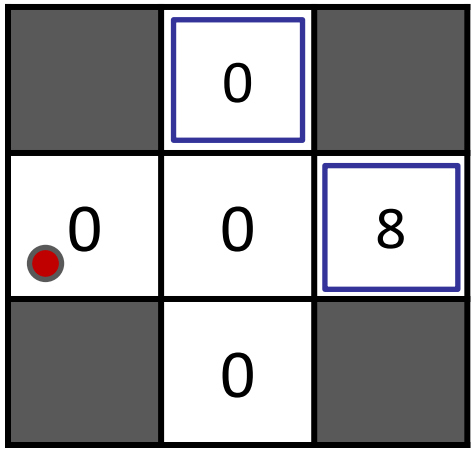
States



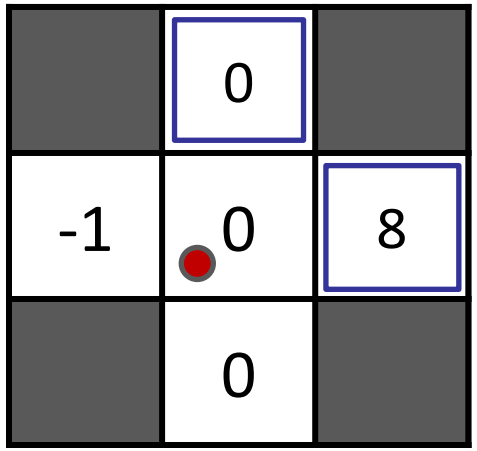
Assume: γ = 1, α = 1/2

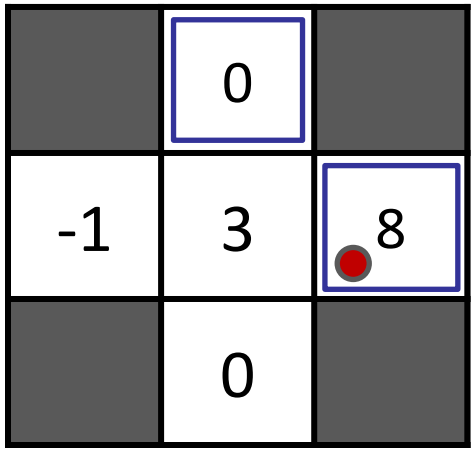
Observed Transitions

B, east, C, -2



C, east, D, -2





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

$$V^{\pi}(B) \leftarrow (1-\alpha)V^{\pi}(B) + \alpha(r + \gamma V^{\pi}(D))$$

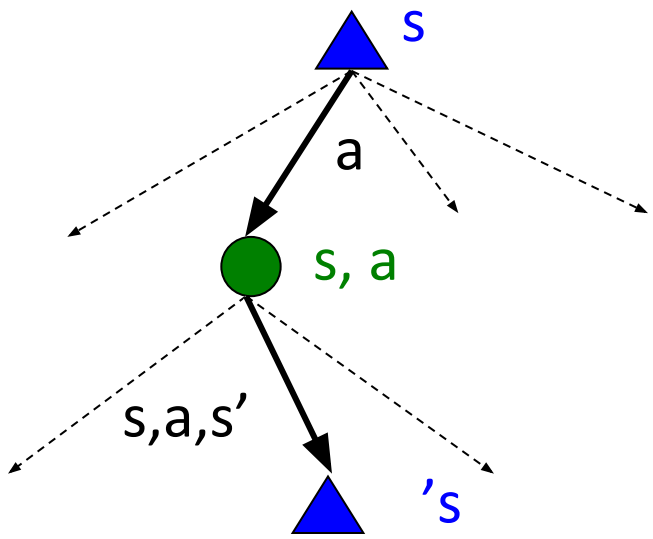
$$V^{\pi}(B) \leftarrow V^{\pi}(B) + \alpha(r + \gamma V^{\pi}(D) - V^{\pi}(B))$$

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:

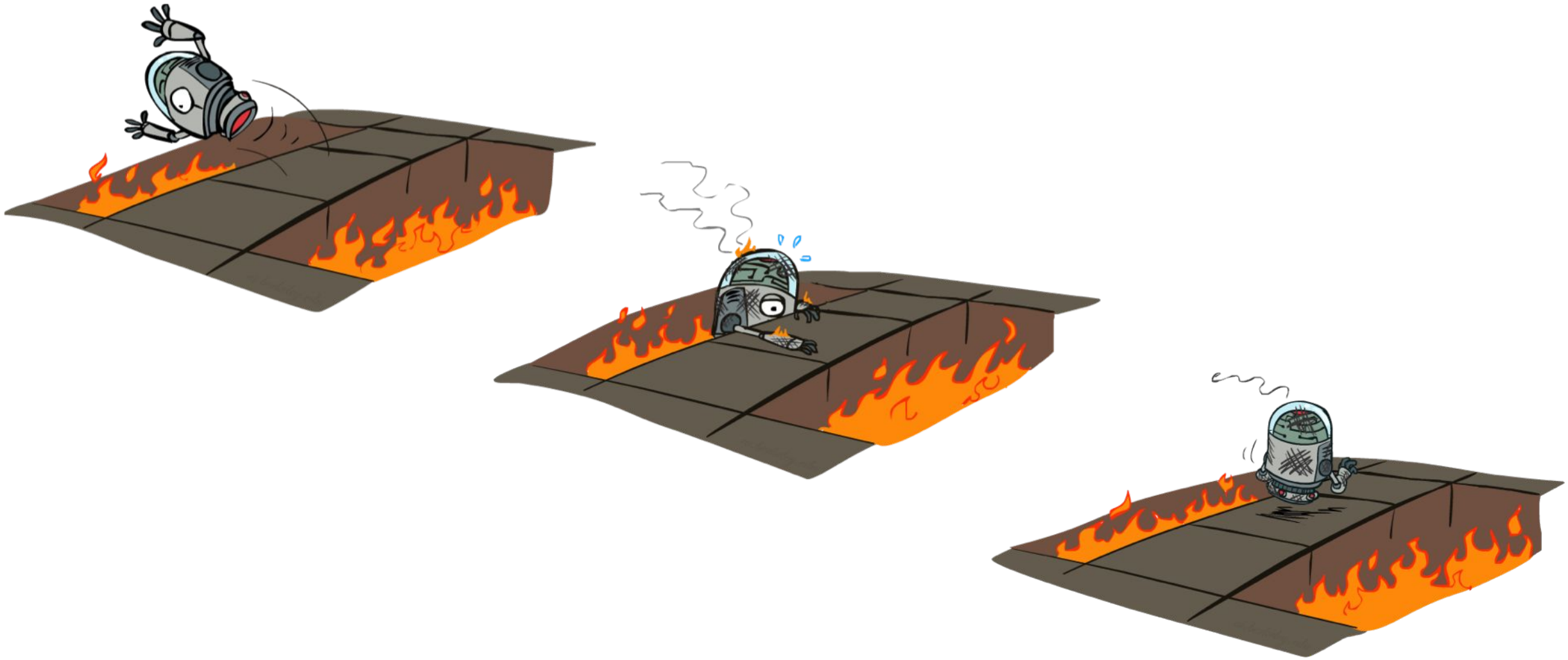
$$\pi(s) = \arg\max_a Q(s,a)$$

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



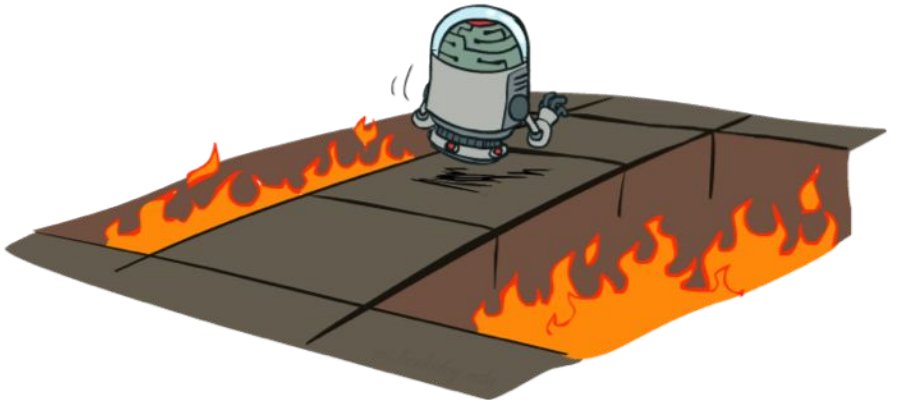
- Idea: learn Q-values, not values
- Makes action selection model-free too!

Active Reinforcement Learning



Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values



- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

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 V_{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 Q_{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

- Q-Learning: sample-based Q-value iteration

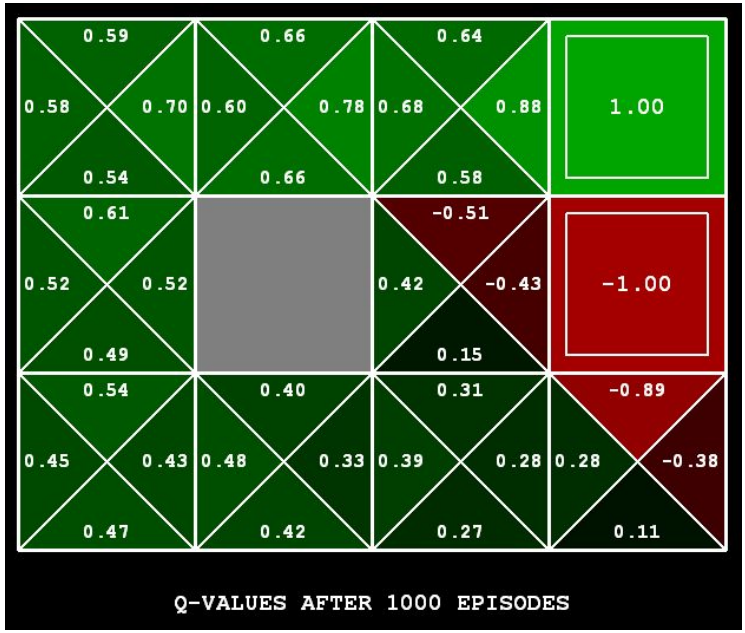
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  1. Initialize $Q(s,a) = 0$ for each s,a pair
  2. Select an action an observe an experience (s,a,r,s')
  3. Consider your old estimate: $Q(s,a)$
  4. Use observation in rough estimate of $Q(s,a)$

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

  5. Update $Q(s,a)$ by moving values slightly toward estimate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big)$$
$$= (1-\alpha)Q(s,a) + \alpha \cdot sample$$

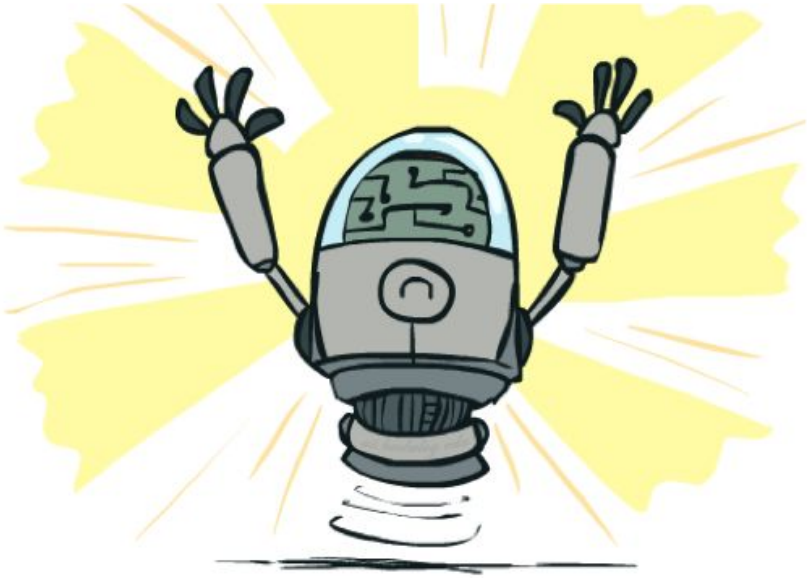


[Demo: Q-learning – gridworld (L10D2)]

Video of Demo Q-Learning -- Gridworld

Q-Learning Properties

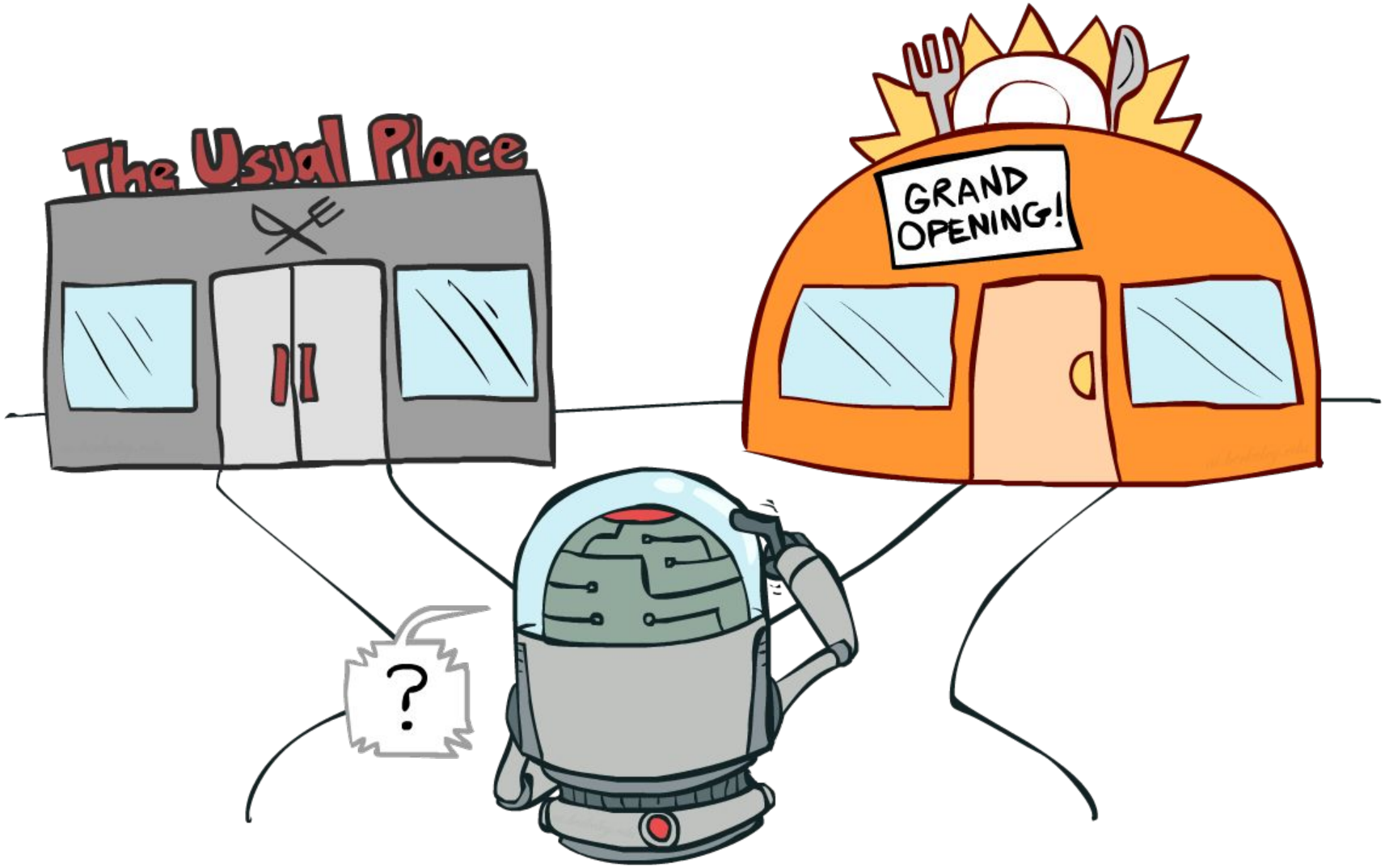
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



Summary

- Reinforcement Learning: learn from experience, not from a ‘teacher’
- RL problems can be cast as MDPs with unknown T and R
- Model-based RL: estimate R and T from experience
- Model-free RL: estimate V(s) or Q(s,a) from experience. Q-Learning can give optimal policies.

Exploration vs. Exploitation



ε-greedy Policies

- Simple idea to balance exploration and exploitation
- Let $|A|$ be the number of actions
- Then an ε-greedy policy w.r.t a state-action value $Q(s,a)$ is

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a), & \text{with prob. } 1-\varepsilon+\frac{\varepsilon}{|A|} \\ a' \neq \arg\max_a Q(s,a), & \text{with prob. } \frac{\varepsilon}{|A|} \end{cases}$$

Q-Learning with ε-greedy Exploration

1. Initialize $Q(s,a) = 0$ for each s, a pair, $t = 0$
2. Set $\pi_b(s) = \arg\max_{a'} Q(s,a)$ with prob. $1-\varepsilon$, else random.
3. Loop
    1. Sample action a from policy π_b an observe an experience (s, a, r, s')
    2. Consider your old estimate: $Q(s,a)$
    3. Use observation in rough estimate of $Q(s,a)$

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

    4. Update $Q(s,a)$ by moving values slightly toward estimate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big) = Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

$$\pi_b(s) = \arg\max_{a'} Q(s,a) \text{ with prob. } 1-\varepsilon, \text{ else random.}$$

    5. $s \leftarrow s'$

SARSA with ε-greedy Exploration

1. Initialize $Q(s,a) = 0$ for each s, a pair, $t = 0$
2. Set $\pi_b(s) = \arg\max_{a'} Q(s,a)$ with prob. $1-\varepsilon$, else random.
3. Sample action a from policy π_b an observe an experience (s, a, r, s')
4. Loop
    1. Sample action a' from policy π_b an observe an experience (s', a', r'', s'')
    2. Consider your old estimate: $Q(s,a)$
    3. Use observation in rough estimate of $Q(s,a)$

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

    4. Update $Q(s,a)$ by moving values slightly toward estimate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big) = Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$

$$\pi_b(s) = \arg\max_{a'} Q(s,a) \text{ with prob. } 1-\varepsilon, \text{ else random.}$$

5. $s \leftarrow s'; a \leftarrow a'$

SARSA & Q-Learning

- SARSA is an on-policy learning algorithm.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$

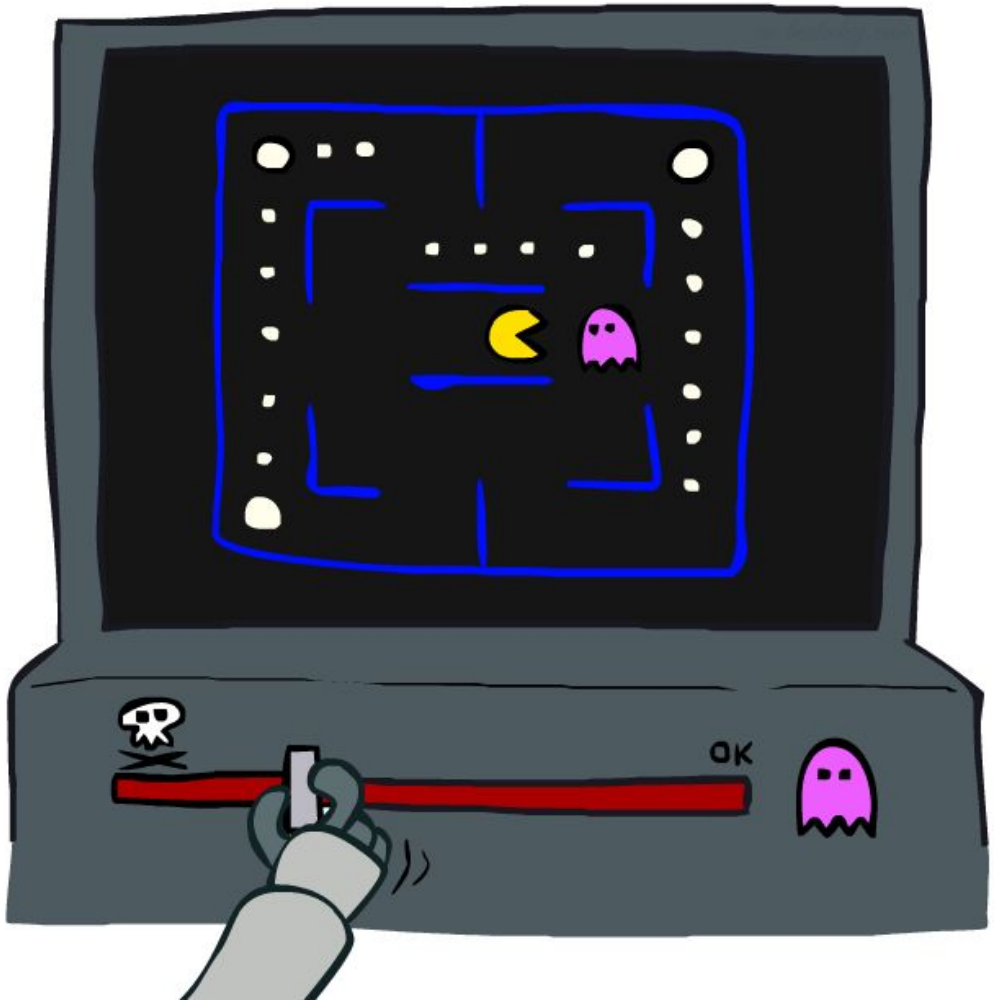
- SARSA estimates the value of the current behavior policy π_b, and then update the policy trying to estimate.

- Q-Learning is an off-policy learning algorithm.

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

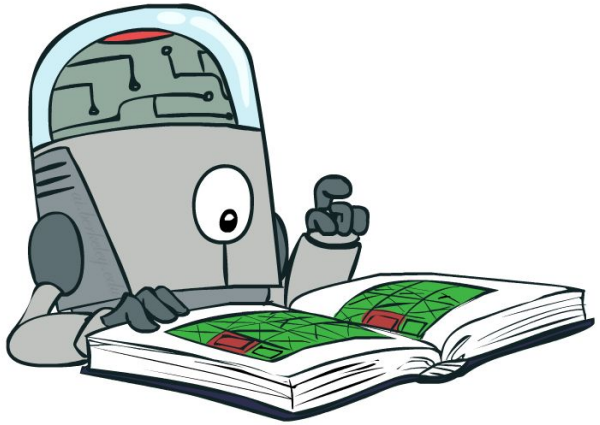
- Q-Learning estimates the value of π^* while acting with another behavior policy π_b.

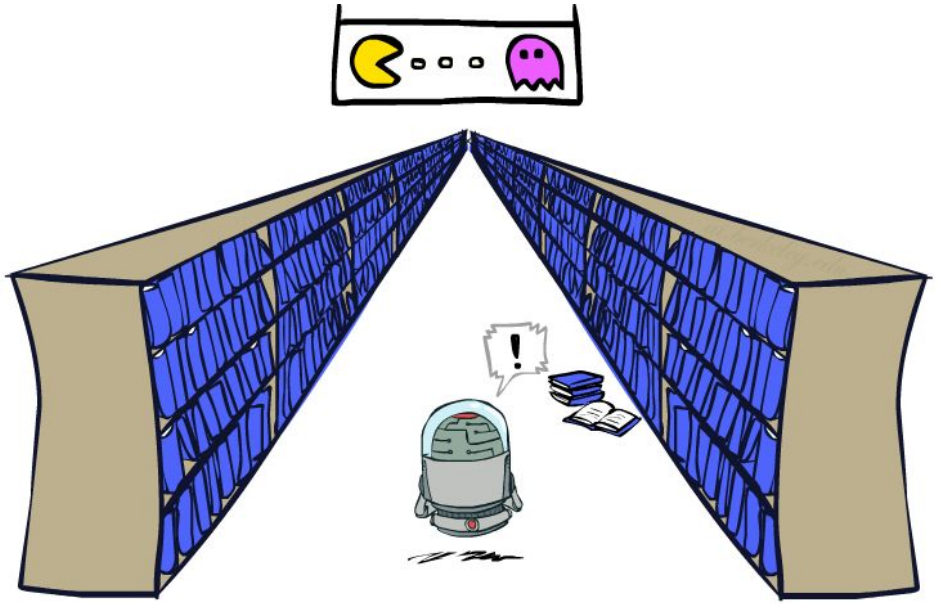
Approximate Q-Learning



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
  - This is a fundamental idea in machine learning, and we'll see it over and over again

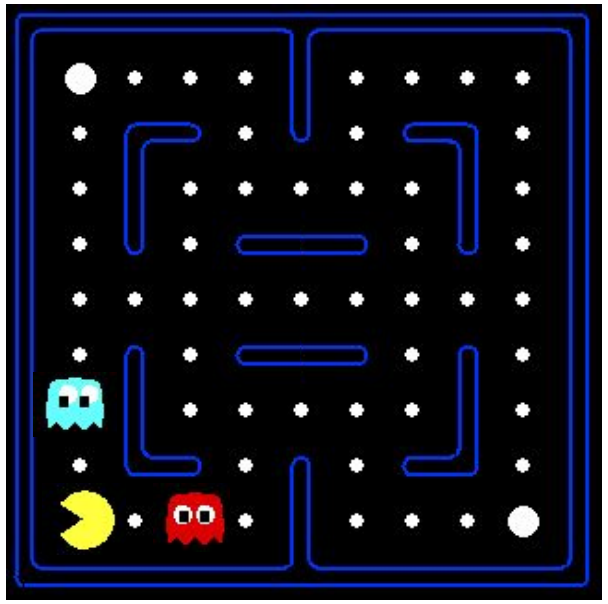




[demo – RL pacman]

Example: Pacman

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

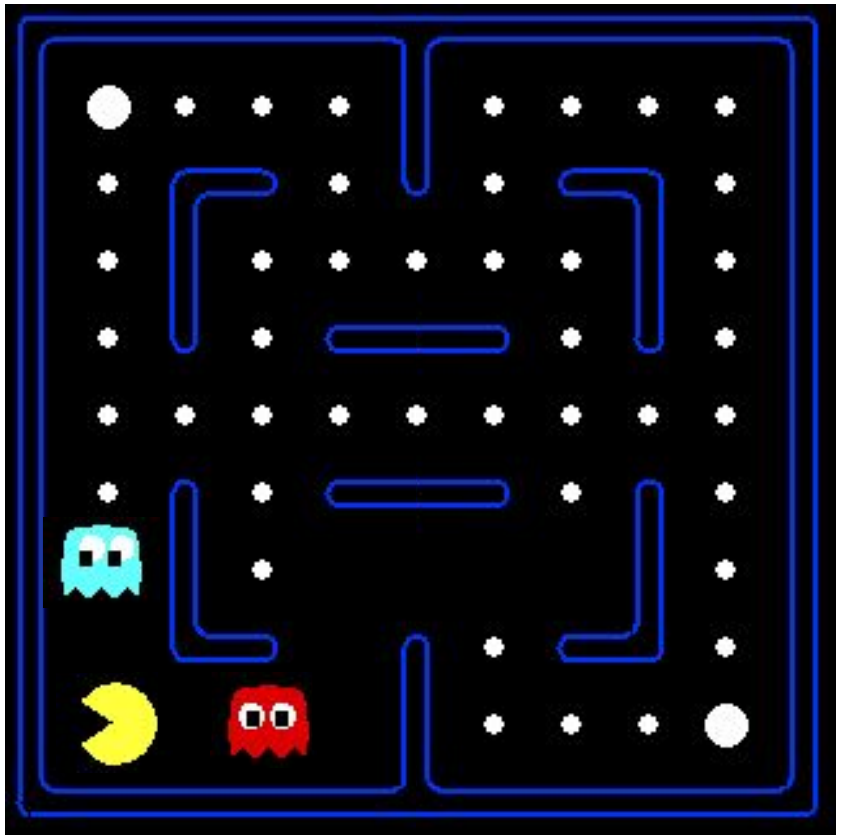


In naïve q-learning, we know nothing about this state:

Or even this one!

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:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

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

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

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:

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

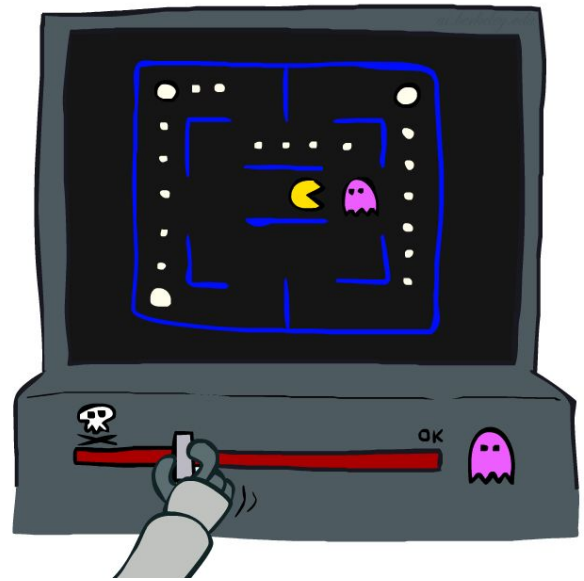
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

Exact Q's

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

Approximate Q's

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

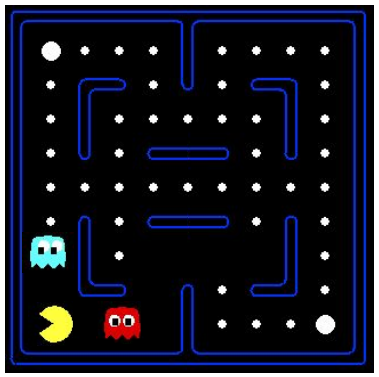


- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features
- Formal justification: online least squares

Example: Q-Pacman

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

s



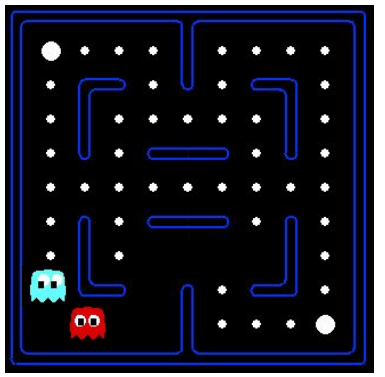
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

difference $= -501$

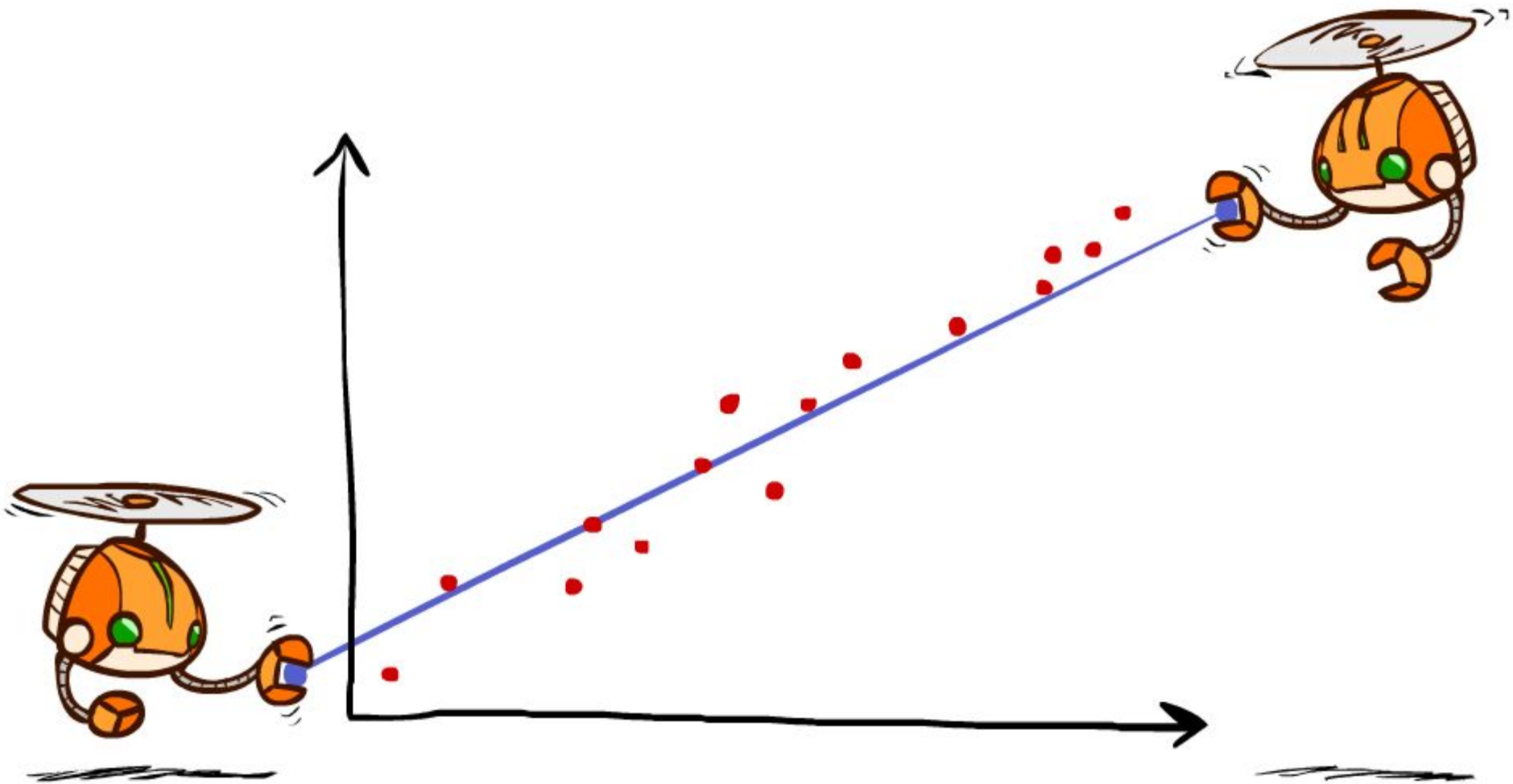
$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$

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

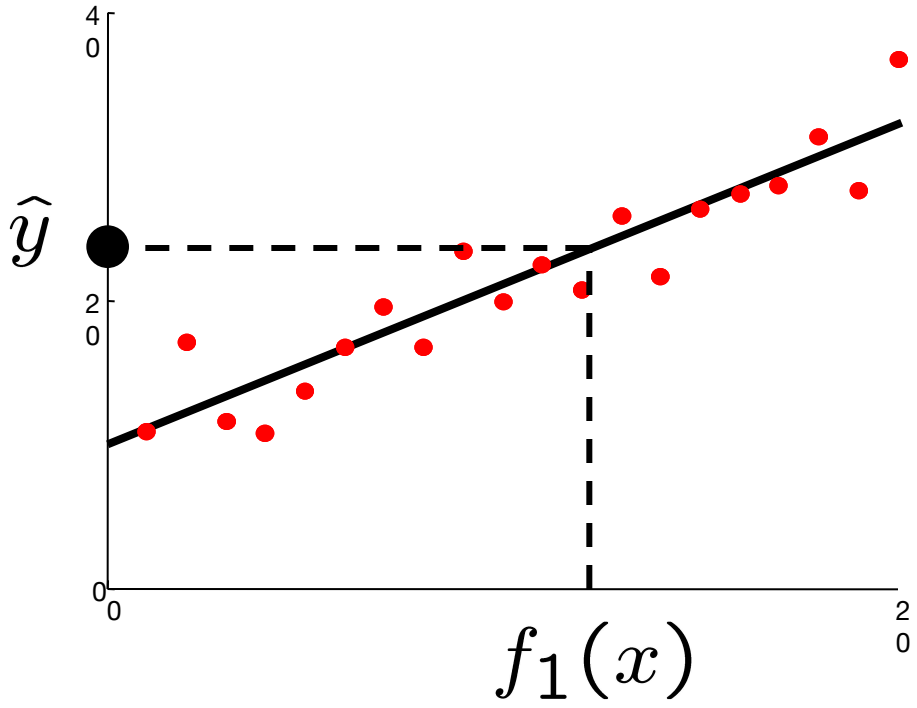
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

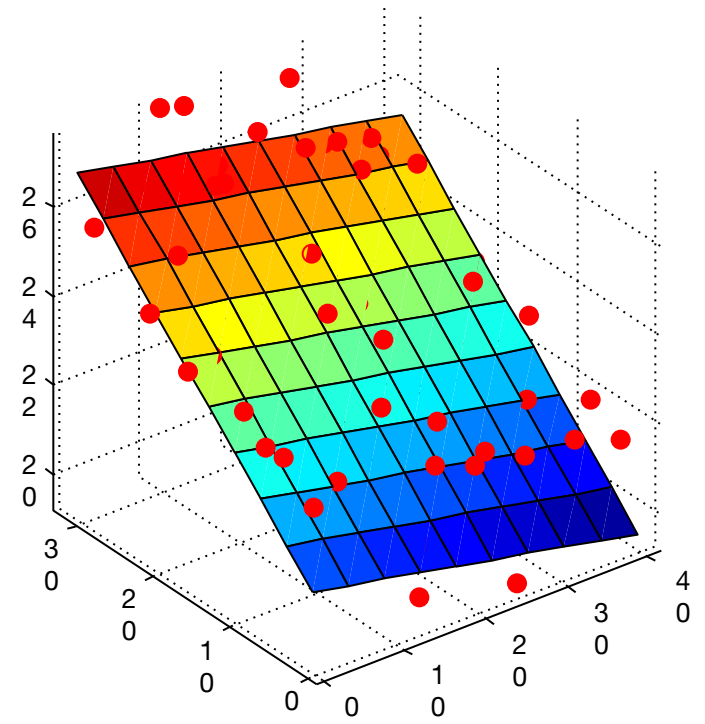
Video of Demo Approximate Q-Learning -- Pacman

Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

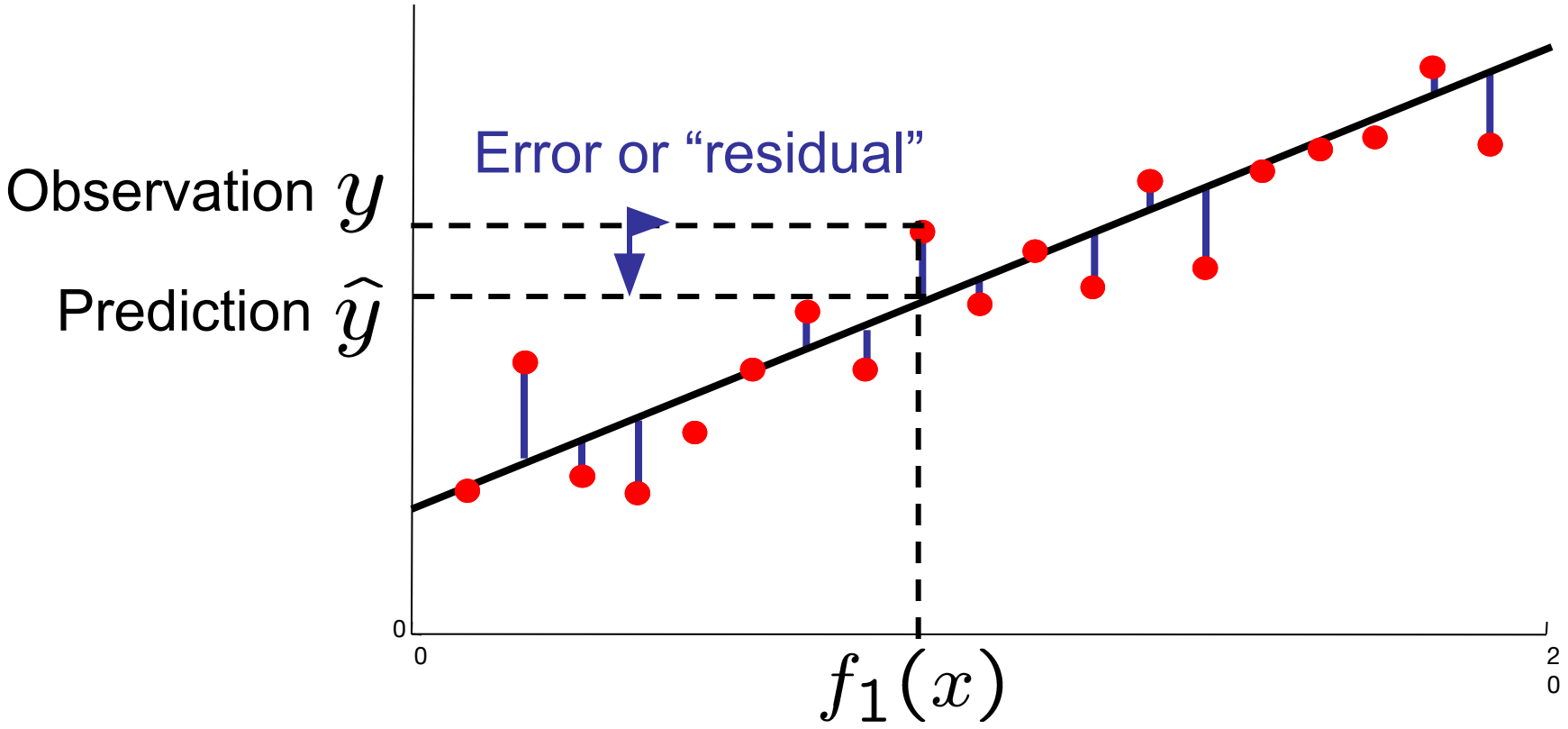
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



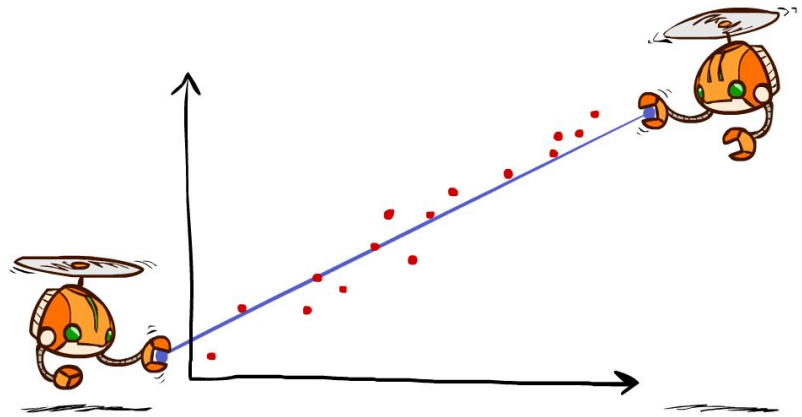
Minimizing Error

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:

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

"target" "prediction"

More Powerful Function Approximation

- Linear:

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

- Polynomial:

$$Q(s,a) = w_{11} f_1(s,a) + w_{12} f_2(s,a)^2 + + w_{13} f_3(s,a)^3 + \cdots$$

- Neural Network:

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

More Powerful Function Approximation

- Summary of iterative feature update equation!

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

$$= f_m(s, a) \text{ when linear approximation}$$