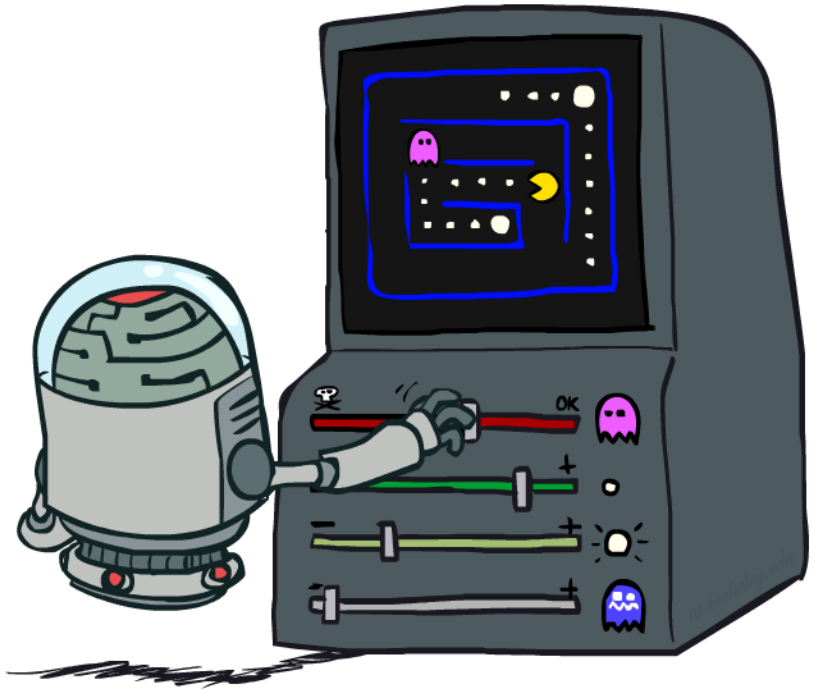
Reinforcement Learning II

CSE 473: Introduction to Artificial Intelligence



Megan Hazen(mh75@cs)
cs.uw.edu/473
acknowledgements

Welcome!
What was your favorite thing to learn?

Administrivia

ANNOUNCEMENTS

- **HW 1** regrades in process
- **Test 1** grades will be released this week

ASSIGNMENTS

- **Homework 2** due next week (10.30)
- **Project 2** due 11.2



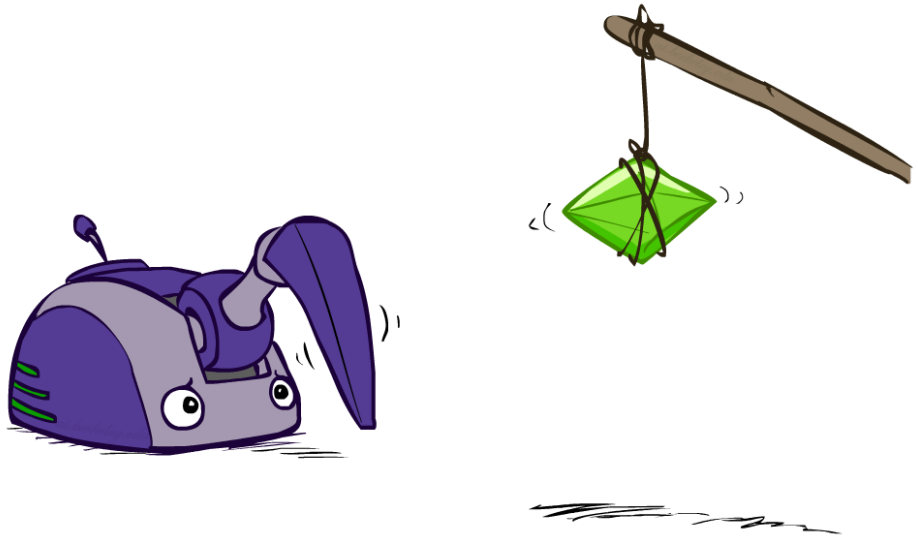
Recap: Reinforcement Learning

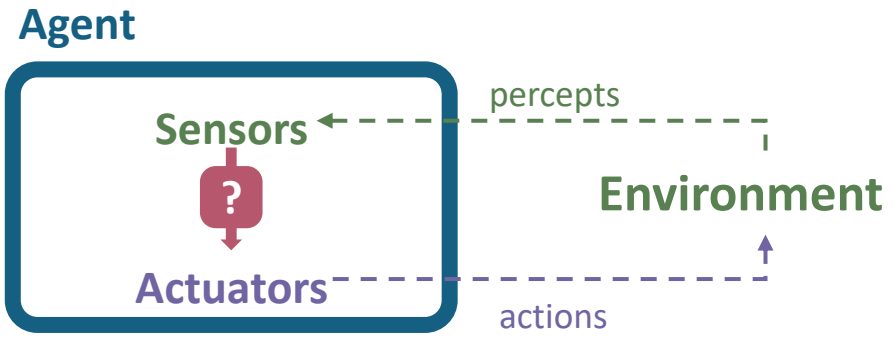
FOUNDATIONS

- Still a Markov Decision Process, with:
  - States $s \in S$
  - Actions $a \in A$
  - Transition model $T(s, a, s')$

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

  - Reward function $R(s, a, s')$
  - Start state s_0
  - Utility function $U(s)$
- But:
  - $\boldsymbol{T}$ and $\boldsymbol{R}$ are unknown.
  - Now must explore to learn best policy

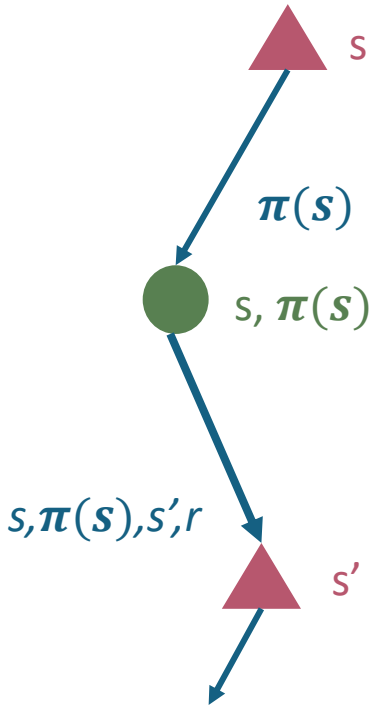




Recap: Temporal Difference Learning

FOUNDATIONS

- Idea: Learn from every experience!
  - Update $\boldsymbol{U(s)}$ each time we experience a transition $\boldsymbol{(s, a, s', r)}$
  - Likely outcomes $\boldsymbol{s'}$ will contribute updates more often
- Temporal Difference Learning
  - Policy is still fixed, still doing evaluation
  - Move $U(s)$ toward value of whichever successor s' occurs (running average)
  - Sample of $\boldsymbol{U(s)}$: $\text{sample} = R(s, \pi(s), s') + \gamma U^{\pi}(s')$
  - Update to $\boldsymbol{U(s)}$: $U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha \cdot \text{sample}$



Problems with TD Value Learning

CONTEXT

TD Value Learning performs model-free policy evaluation

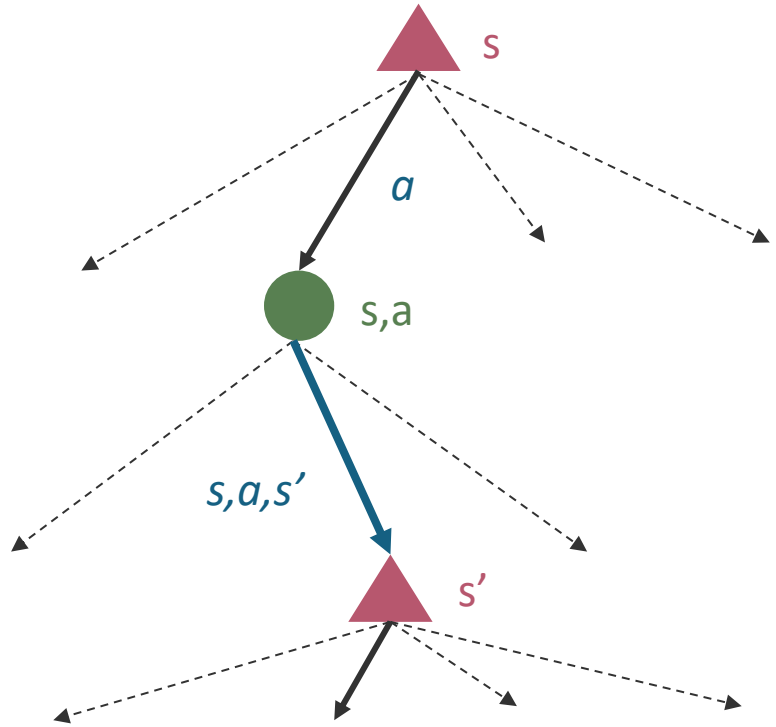
- Mimics Bellman updates with running average samples

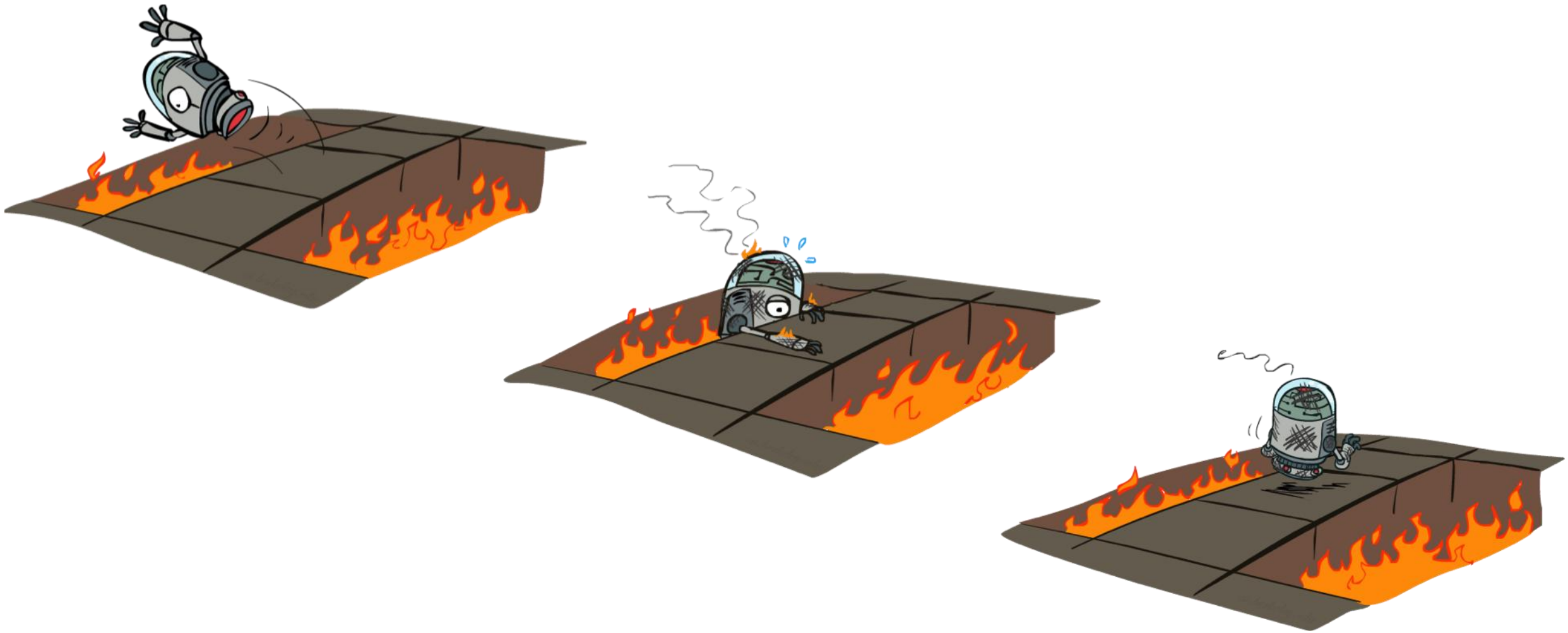
- However, if we want to turn values into policy, we're stuck:

$$\pi(s) = \text{argmax}_a Q(s,a)$$

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

- Idea: Learn **Q-values**, not utilities
  - Makes action selection model-free, too!



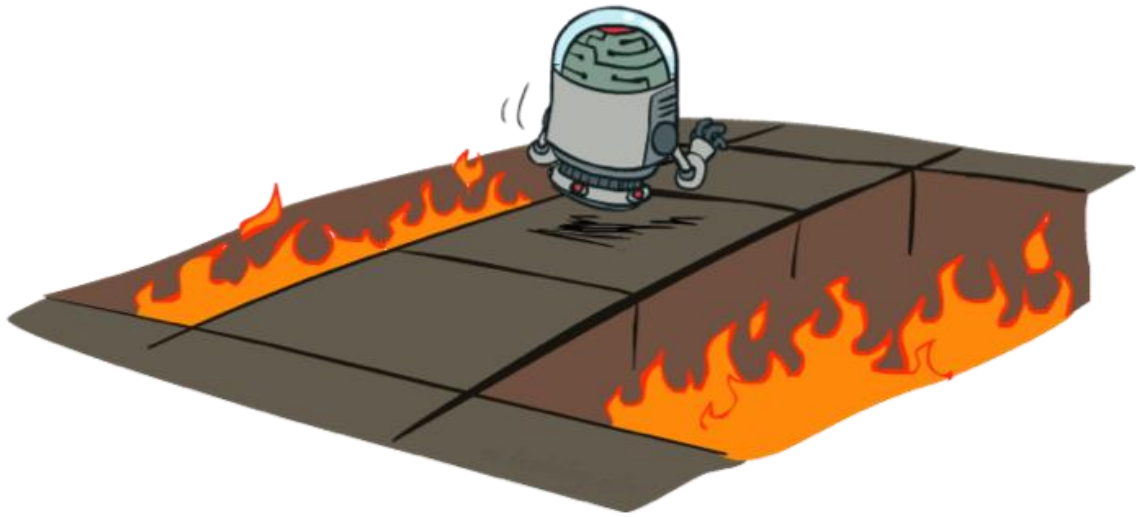


A More Hands-On Approach…

Active Reinforcement Learning

FOUNDATIONS

- *Full* Reinforcement Learning: Derive **optimal policies**
  - Don't know $T(s, a, s') = P(s'|s, a)$
  - Don't know $R(s, a, s)$
  - Choose actions
  - Goal: *learn the state* ***values and policy***



- In this case:
  - Learner makes choices!
  - Fundamental Tradeoff: **exploration** vs. **exploitation**
  - *Not* offline planning
    - Learner actually takes actions and learns from experience

Questions?

But First: Q-Value Iteration

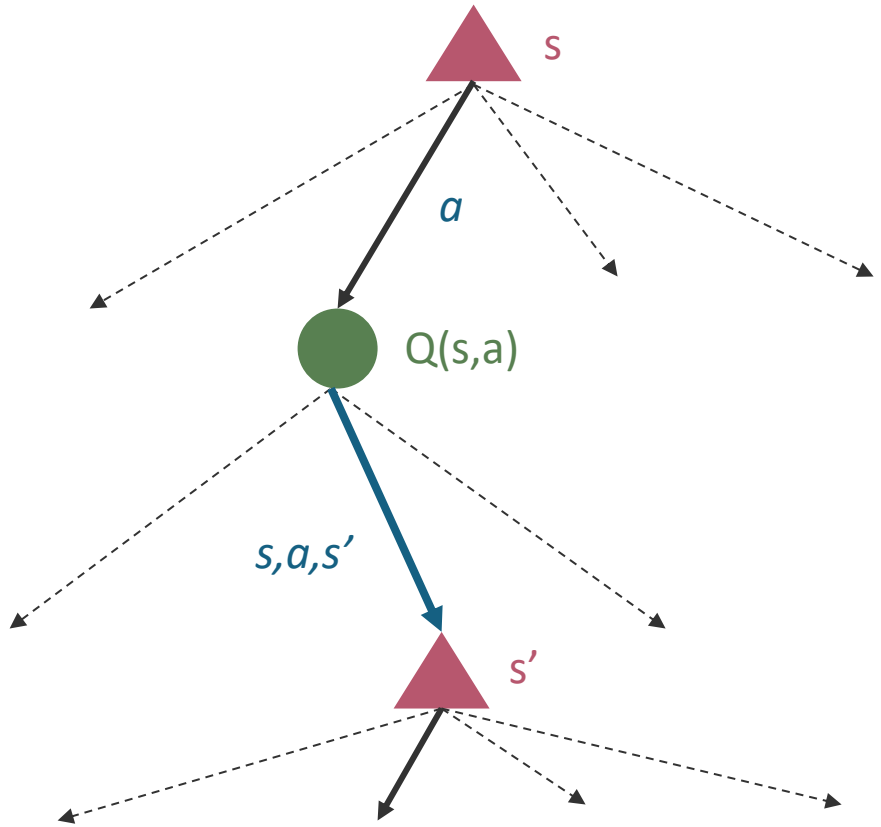
FOUNDATIONS

- **Value Iteration**: Find successive (depth-limited) values
  - Start with $U^{\pi}(s) = 0$
  - Given U_k, calculate depth $k+1$ values for all states:

$$U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

- But **Q-values** are more useful…
  - Start with $Q_0(s,a) = 0$
  - Given Q_k, calculate depth $k+1$ Q-values for all states:

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



Q-Learning

DEFINITION

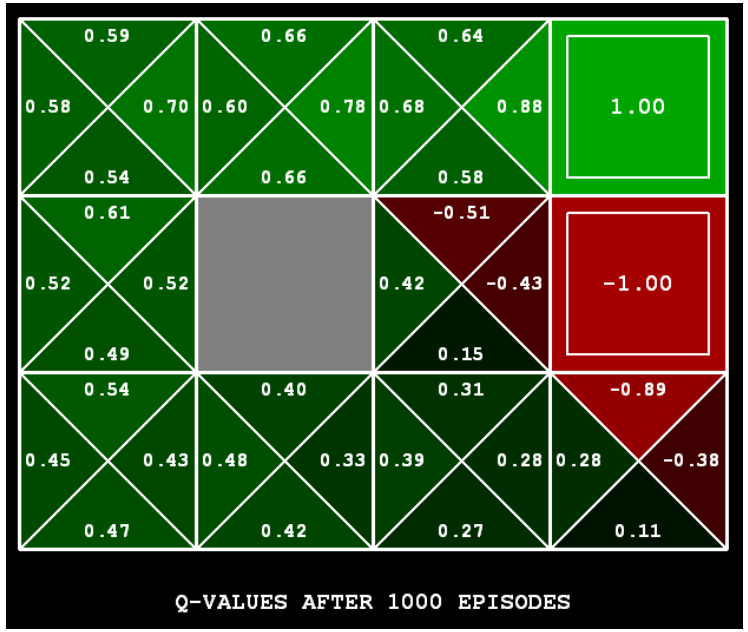
- Q-value iteration:

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

- Q-Learning is a sample-based version:

- Learn Q-values as you go:
  - Receive sample (s,a,s',r)
  - Consider old estimate $Q_k(s,a)$
  - Consider new sample estimate: sample $= R(s,a,s') + \gamma \max_{a'} Q_k(s,a)$
  - Incorporate new estimate into a running average:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha \cdot \text{sample}$$

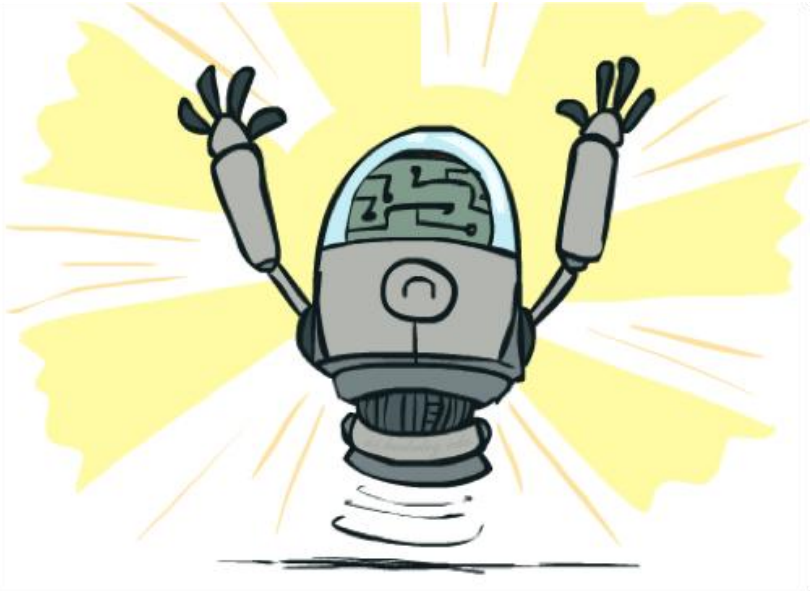


Properties of Q-Learning

DEFINITION

Off-Policy Learning

- Q-Learning converges to optimal policy, even if you're acting sub-optimally!

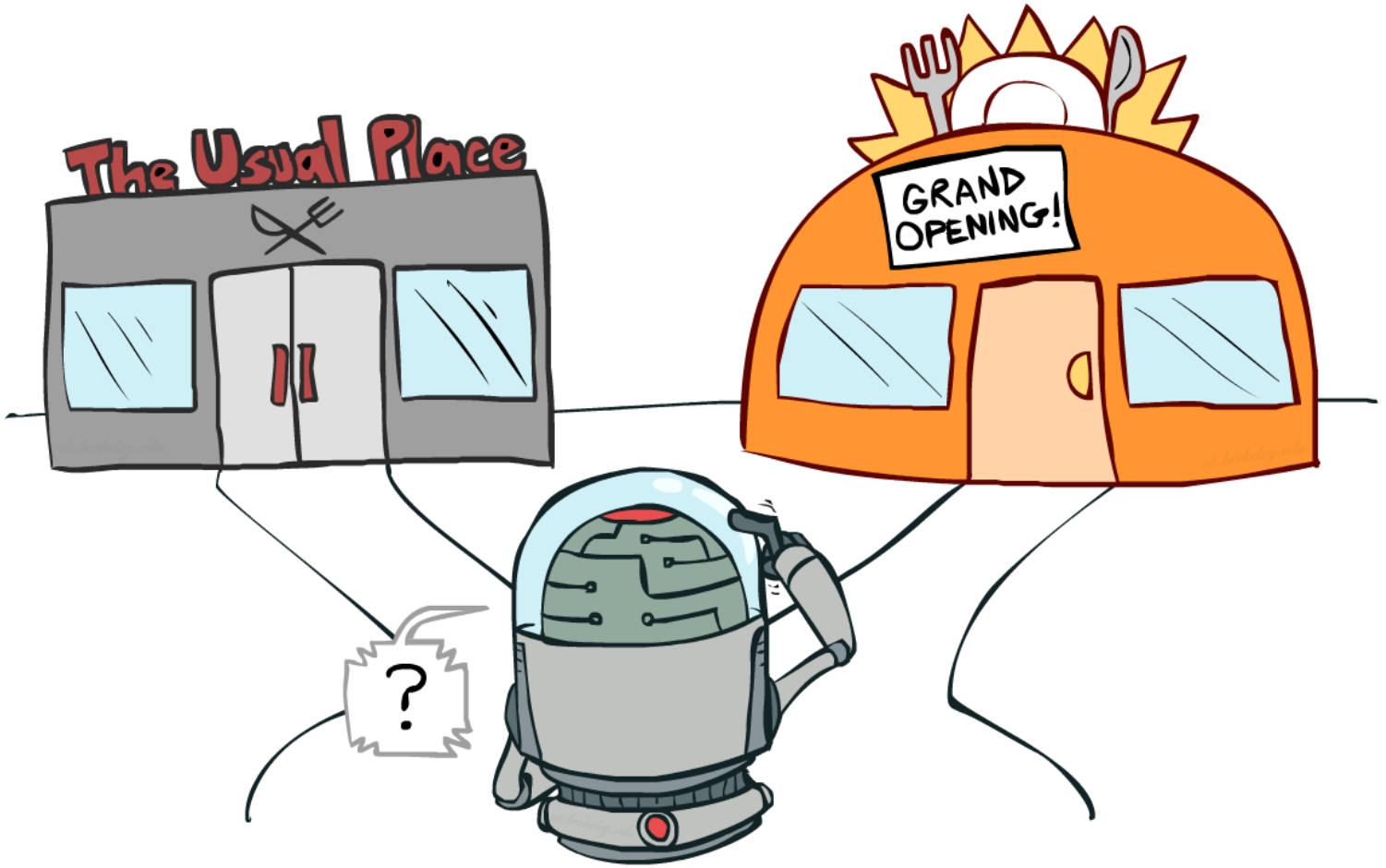


- Caveats:
  - Must explore enough
  - Must eventually make the learning rate small enough
  - In the limit, it doesn't matter how actions are selected

Questions?

Exploration vs. Exploitation

CONTEXT



How to Explore?

CONTEXT

How should the agent balance exploration and exploitation?

- Simple Approach: Random Chance (ε-greedy)
  - With (small) probability ε, take a random action
  - With (large) probability $1 - \varepsilon$, act on current policy
- *Problems with Random Actions?*
  - Eventually explore the space, but do so with a lot of thrashing around
    - One solution: lower ε over time
  - Another solution: **exploration functions**



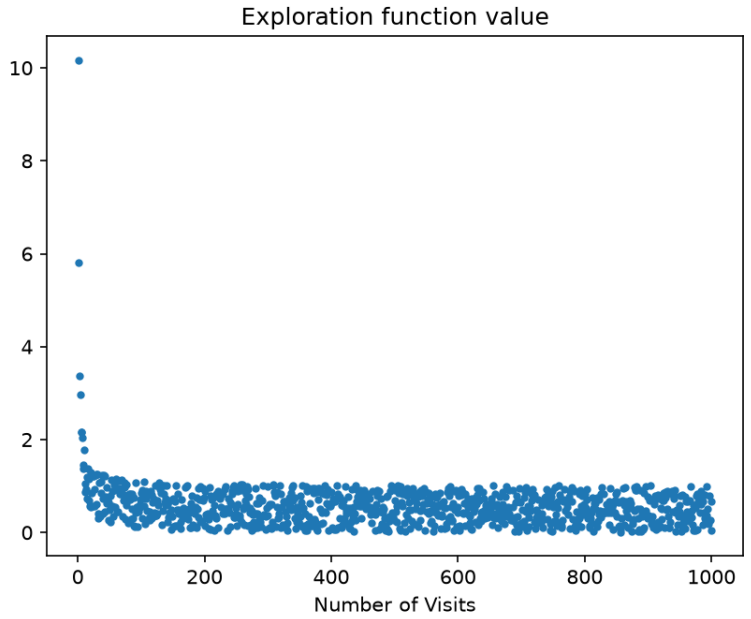
Exploration Functions

FOUNDATIONS

- An **exploration function** takes value estimate u and visit count n and returns an optimistic utility $f(u, n)$

$$\text{E.g., } \mathrm{f(u, n)} = \mathrm{u} + \mathrm{c/n} \text{ with constant } c$$
$$\text{E.g., } \mathrm{f(u, n)} = \mathrm{u} \text{ iff } \mathrm{n} > 10, \text{else } \mathrm{u} + \mathrm{c}$$

- Early on – dominated by ***c***.
  - Larger c encourages visitation of unknown states
- As $n \to \infty f(u, n) \to u$



Exploration Functions in Q-learning

FOUNDATIONS

- An **exploration function** takes value estimate u and visit count n and returns an optimistic utility $f(u, n)$

  E.g., $\mathrm{f(u, n)} = \mathrm{u} + \mathrm{c/n}$ with constant c

- **Modified Q-Update**:

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

  becomes

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

  $N(s, a)$ is the number of times Q-state (s, a) has been visited



Q-value calculations

FOUNDATIONS

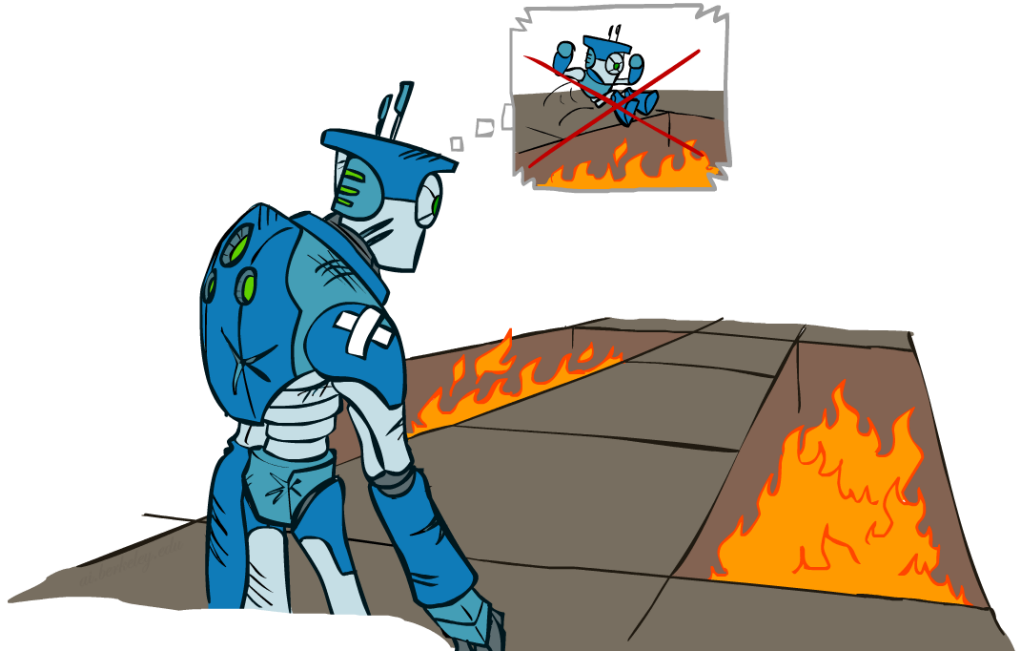
- New Q calculated recursively
  - $Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$
- New Q calculated using running-average $(1-\alpha)\mathrm{Q}_{\mathrm{old}} + \alpha \cdot \mathrm{Q}_{\mathrm{sampled}}$
  - *Sample represents* $R(s,a,s') + \gamma \max_{a'} Q_k(s,a)$
  - $Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q_k(s,a)\ ]$
- New Q calculated using an exploration function
  - $Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha \cdot (\mathrm{f}(\mathrm{sample},\mathrm{k}))$

Regret

FOUNDATIONS

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

- **Regret** is a measure of the total mistake cost
  - Difference between (expected) rewards (including early suboptimality), and optimal (expected) rewards
  - For example, random exploration incurs more regret than exploring with exploration functions
- Minimizing regret applies optimization to the process of learning to be optimal

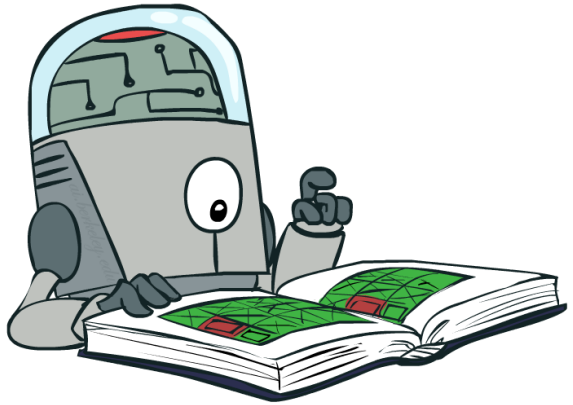


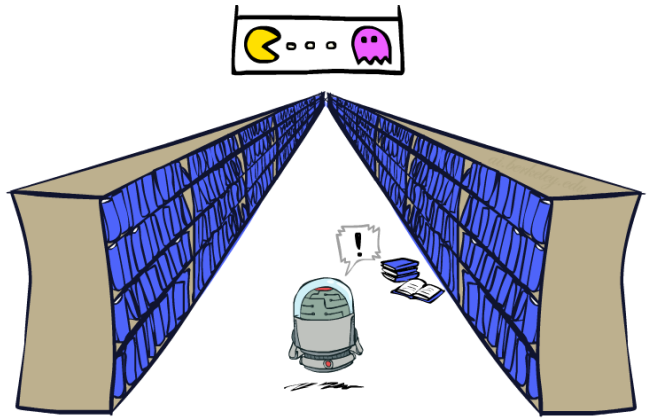
Generalizing Across States

CONTEXT

- Naïve Q-Learning keeps track of all Q-states
  - But in realistic situations, we cannot possibly learn about every single state!
    - Too many states to visit them all in training
    - Too many states to track in memory
- Instead, we want to *generalize*:
  - Learn about some small number of states from experience
  - Generalize experiences to new, but similar situations

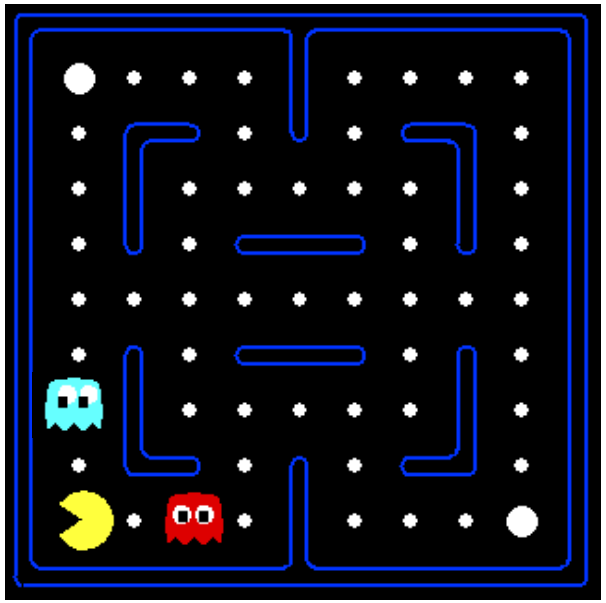
Generalizing learning is a fundamental challenge in AI!



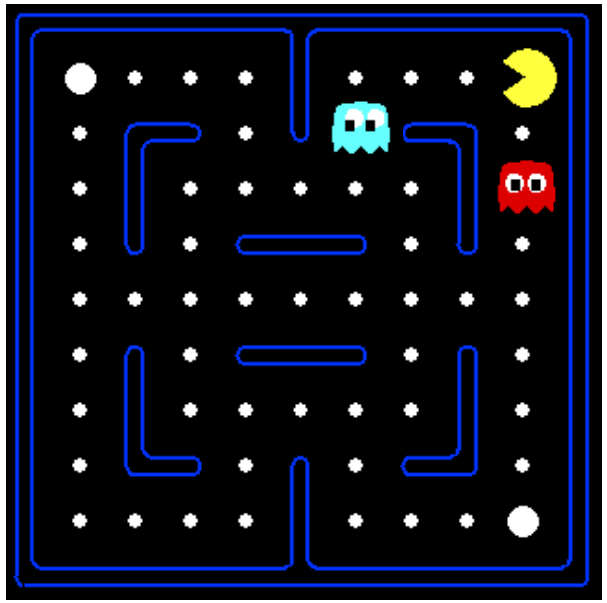


Poor Generalization

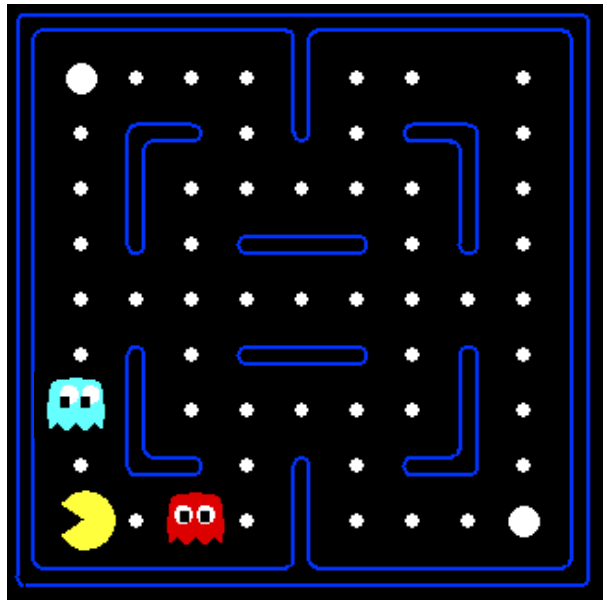
EXAMPLE



With naïve Q-learning, learning from experience in this state…



…doesn't provide information about this state…

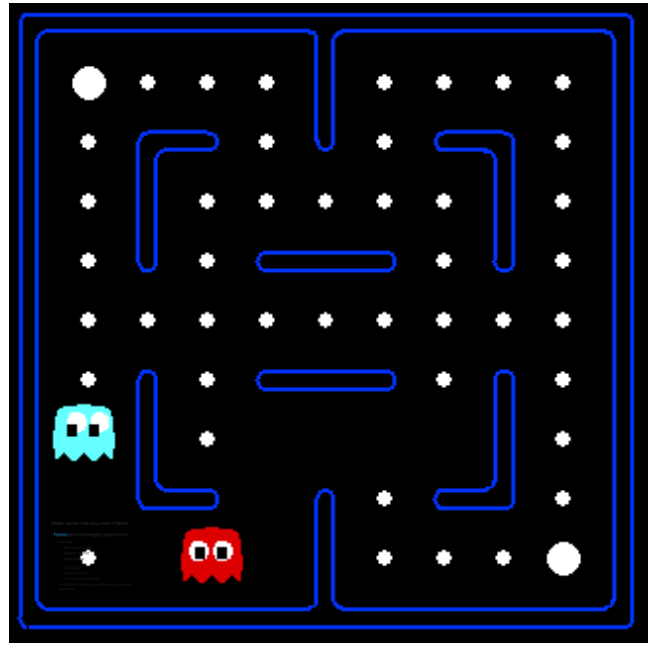


…or even this state!…

Feature-Based Learning

FOUNDATIONS

Solution: describe a state using a vector of features



- **Features** are functions mapping states to real numbers
  - For example:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $\frac{1}{(\text{dist to closest dot})^2}$
    - Is Pacman in a tunnel?
    - 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

FOUNDATIONS

- Using a feature representation, define a **Q-function** (Q-value utility function)

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

(for n features)

- **Advantage**: Experience is summed up in a few powerful numbers
- **Disadvantage**: States may share features but differ in actual utility!

Approximate Q-Learning

DEFINITION

Approximate Q-Learning with linear combination of features as Q-function

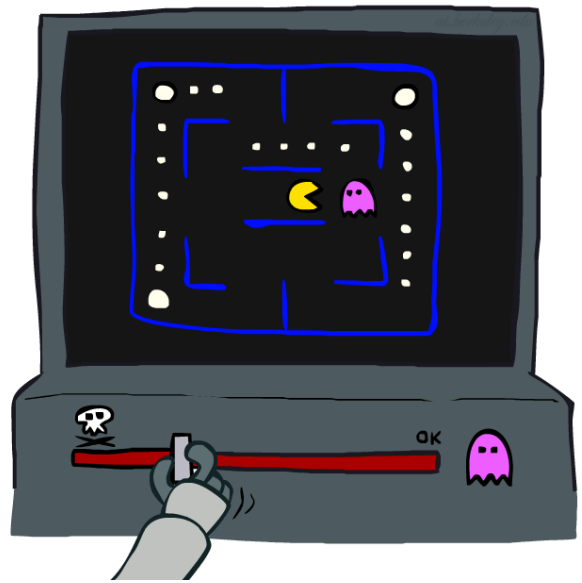
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

- Process:
  - Collect transition samples (s, a, s', r)
  - Let difference $\leftarrow [r + \gamma \cdot \max_{a'} Q(s', a')] - Q(s,a)$
  - **Exact Q-Values:**

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \cdot \text{difference}$$

  - ***Approximate* Q-Values:**

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$



Intuition Behind Approximate Q-Learning

DEFINITION

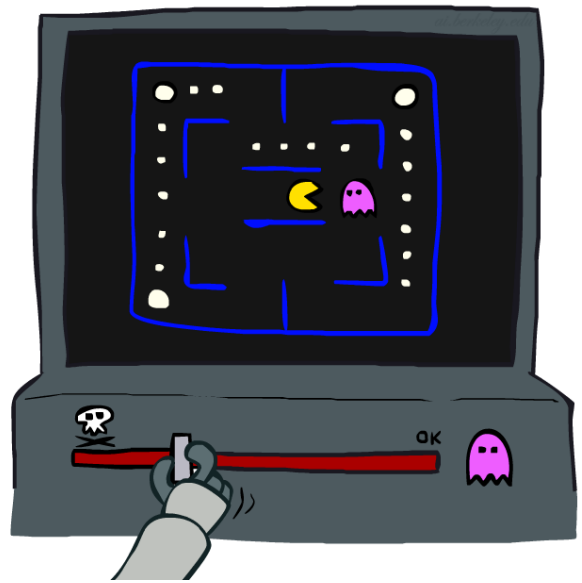
Approximate Q-Learning with linear combination of features as Q-function

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

- **Approximate Q-Values**:

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$

- Intuition
  - Adjust weights of active features
  - If something unexpectedly bad happens, blame the features that were "on" — avoid all states with similar features



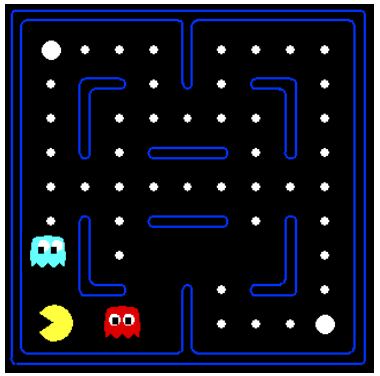
Questions?

Q-Pacman (1)

EXAMPLE

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

s

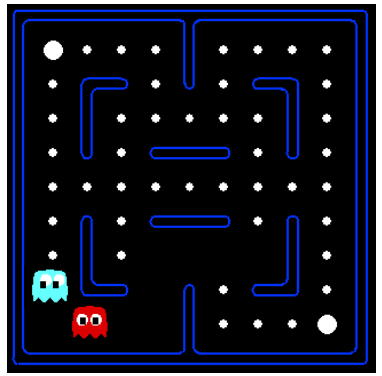


$f_{\text{DOT}}(s, \text{NORTH}) = 0.5$

$f_{\text{GST}}(s, \text{NORTH}) = 1$

$a = \text{NORTH}$

$r = -500$



s'

$Q(s,a) = Q(s, NORTH) = +1$

$Q(s,\cdot) = 0$

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

$\text{difference} = -501$

$w_{\text{DOT}} \leftarrow (4.0) + \alpha \cdot [-501] \cdot (0.5)$

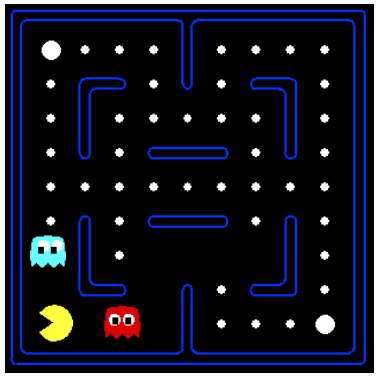
$w_{\text{GST}} \leftarrow (-1.0) + \alpha \cdot [-501] \cdot (1)$

Q-Pacman (2)

EXAMPLE

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

s

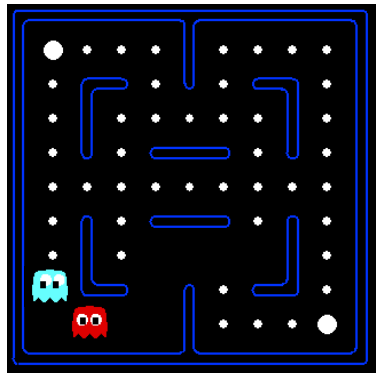


$f_{\text{DOT}}(s, \text{NORTH}) = 0.5$

$f_{\text{GST}}(s, \text{NORTH}) = 1$

$a = \text{NORTH}$

$r = -500$



s'

$Q(s,a) = Q(s, NORTH) = +1$

$Q(s,\cdot) = 0$

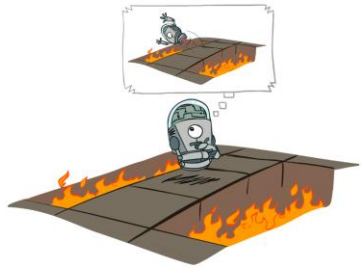
$r + \gamma \cdot \max_{a'} Q(s,a) = -500 + 0$

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

with $\alpha \approx 0.004$

Summary: Reinforcement Learning

CONTEXT



Offline Learning

Known MDP



Online Learning

Unknown MDP

Goal	Approach
Compute U^*, Q^*, π^*	Value and Policy Iteration
Evaluate fixed policy π^*	Policy Evaluation

Model-Based

Goal	Approach
Compute U^*, Q^*, π^*	Value, Policy Iter. on approx. MDP
Evaluate fixed policy π^*	Policy Eval. on approx. MDP

Model-Free

Goal	Approach
Compute U^*, Q^*, π^*	Q-Learning
Evaluate fixed policy π^*	Value Learning

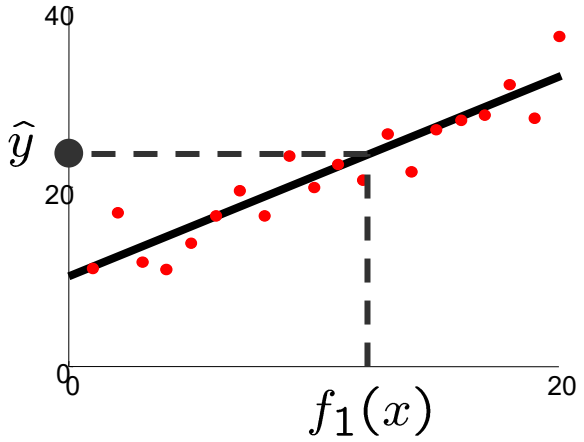
Questions?

Approximate Q-Learning Least Squares

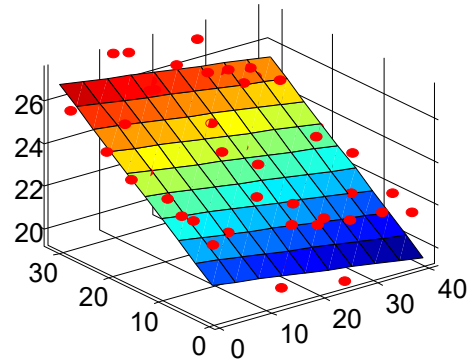
FOUNDATIONS

- **Approximate Q-Values**:

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



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



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

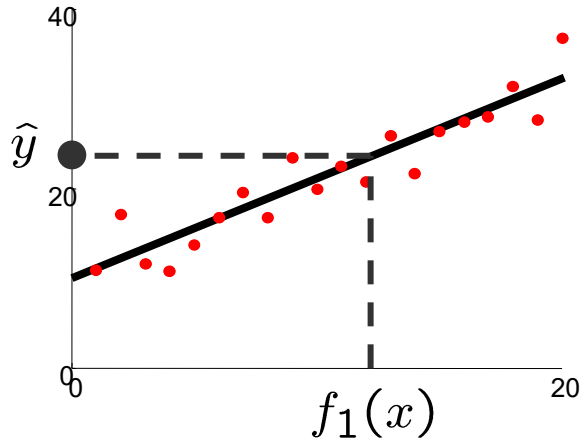
Least Squares Minimizing Error

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

1. Calculate the total error: $\mathrm{E} = \frac{1}{2}\sum_i(\mathrm{y_i} - \hat{\mathrm{y}}_i)^2 = \frac{1}{2}\sum_i(\mathrm{y_i} - \sum_k \mathrm{w_k f_k})^2$

2. Calculate the slope with respect to the weight: $\frac{\partial \mathrm{E}}{\partial \mathrm{w_k}} = -(\mathrm{y} - \sum_k \mathrm{w_k f_k(x)})\mathrm{f_k(x)}$

3. Update weights in downhill direction: $\mathrm{w_k} \leftarrow \mathrm{w_k} + \alpha(\mathrm{y} - \sum_k \mathrm{w_k f_k(x)})\mathrm{f_k(x)}$



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

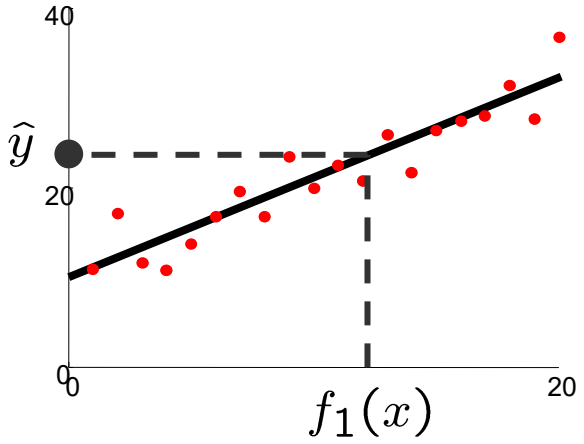
Update Q-values or Least-square weights

FOUNDATIONS

- **Approximate Q-Values:**

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

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$



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

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

Or

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

Linear Separable Classes

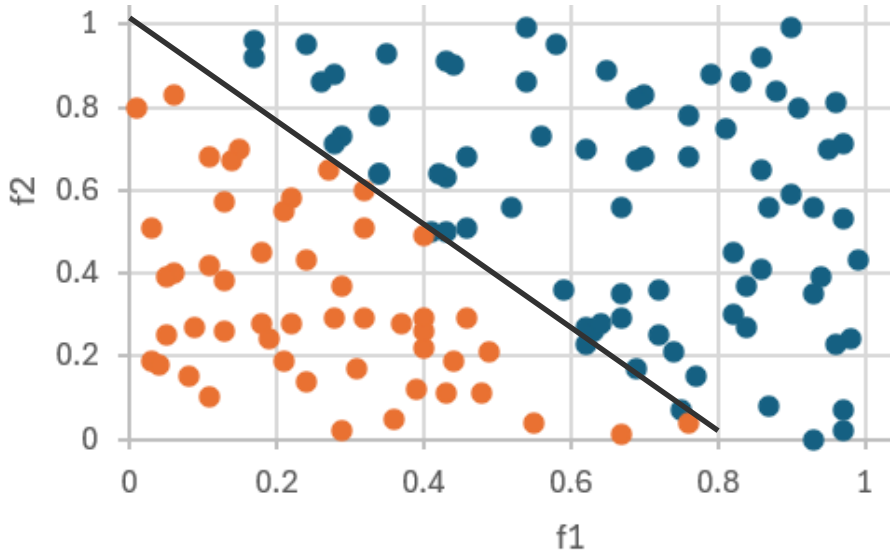
EXAMPLE

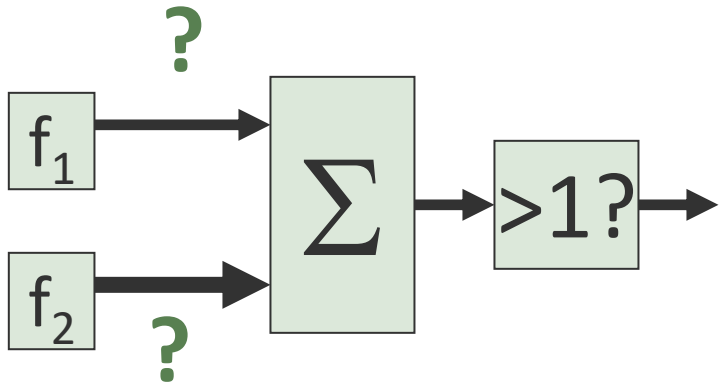
- What if we want to classify something?

  Let's fine the line: $w_2 f_2(y) = w_0 + w_1 f_1(x)$

  - Class 1 < $w_0 + w_1 f_1(x) + w_2 f_2(y)$
  - Class 2 > $w_0 + w_1 f_1(x) + w_2 f_2(y)$

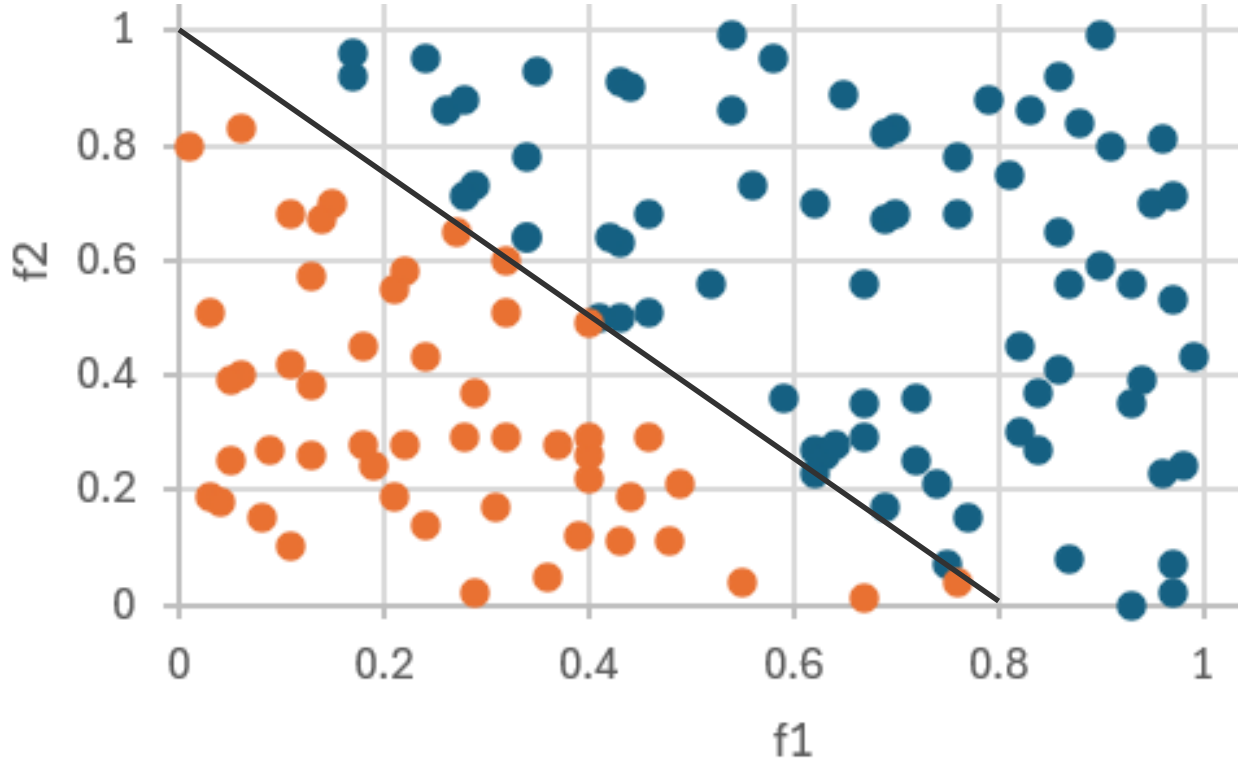
- Solve in a similar manner
  - $w_0 = -1, w_0 = \frac{1}{.8} = 1.25, w_2 = 1$

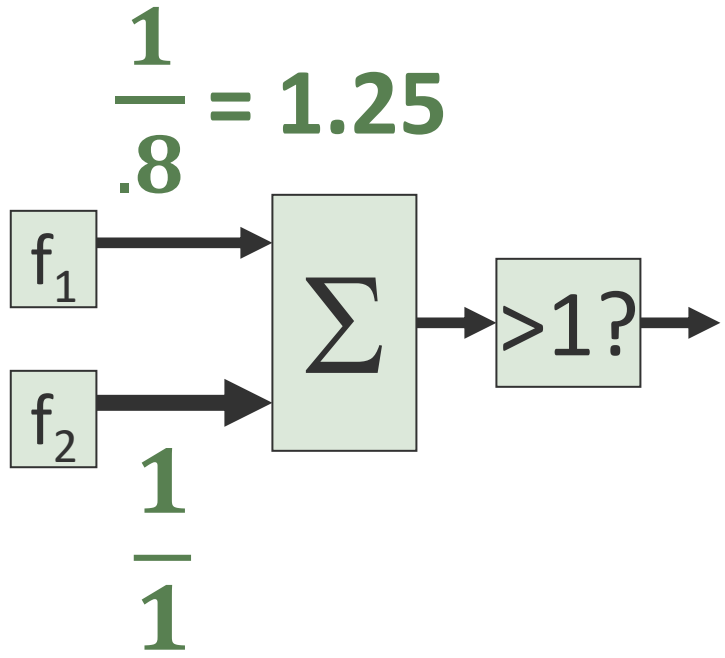




Linear Separable Classes

EXAMPLE

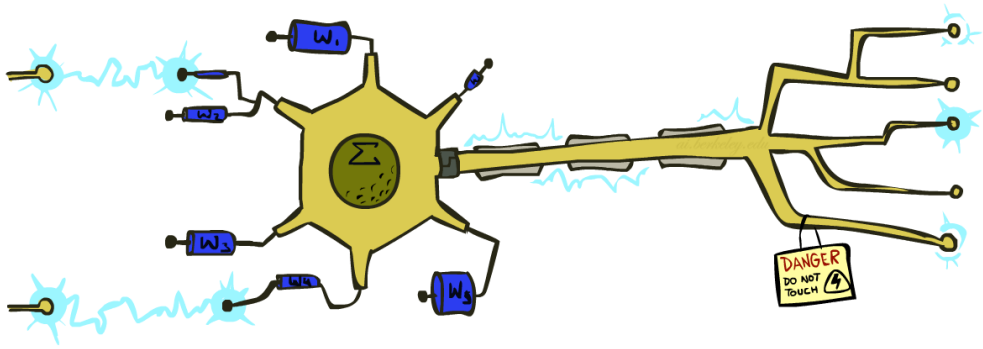




Perceptrons (spoiler!)

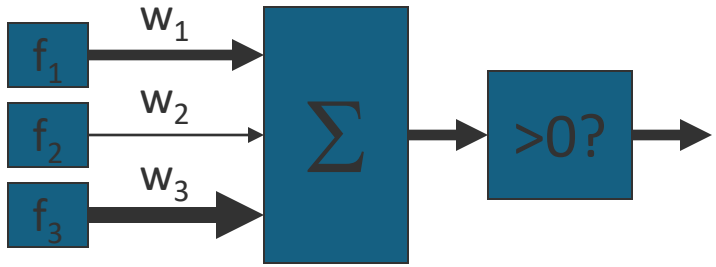
DEFINITION

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



that's it for today!

SUMMARY

- Q-Learning
- Exploration vs. Exploitation Tradeoff
- Feature-Based Approximation

UPCOMING

- Reinforcement Learning in Practice
- Statistical modeling

REMINDERS

- Finish **Homework 2** (due Friday)