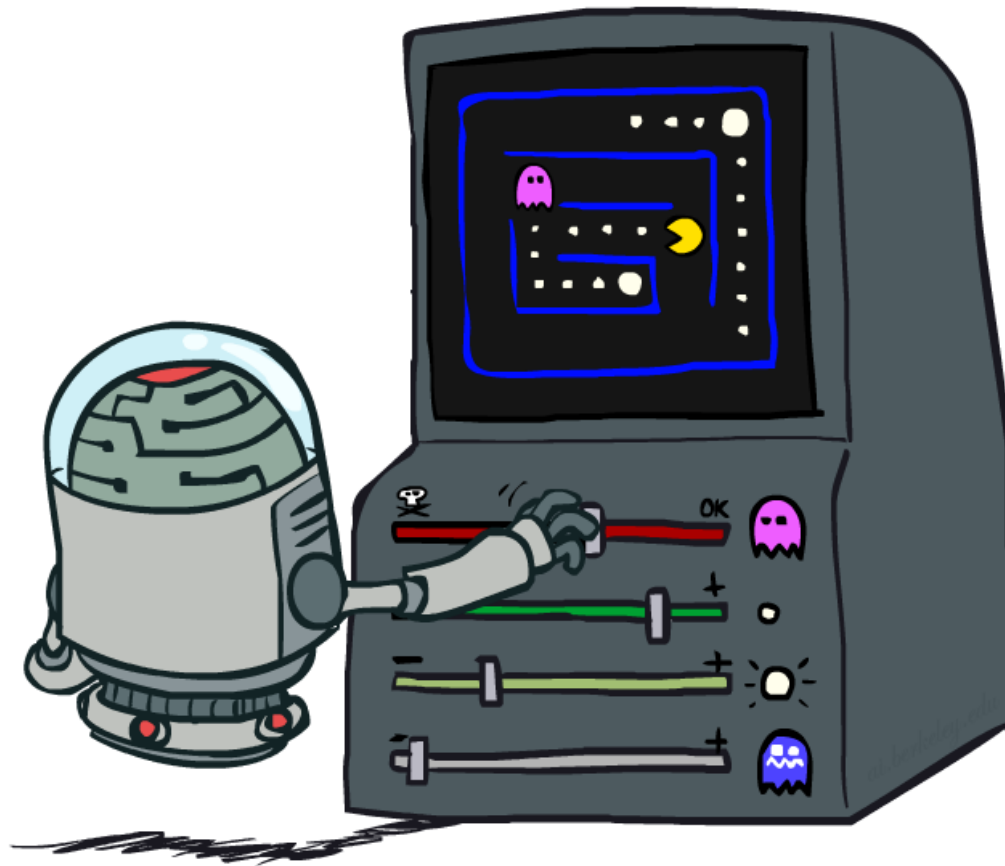


CS 188: Artificial Intelligence

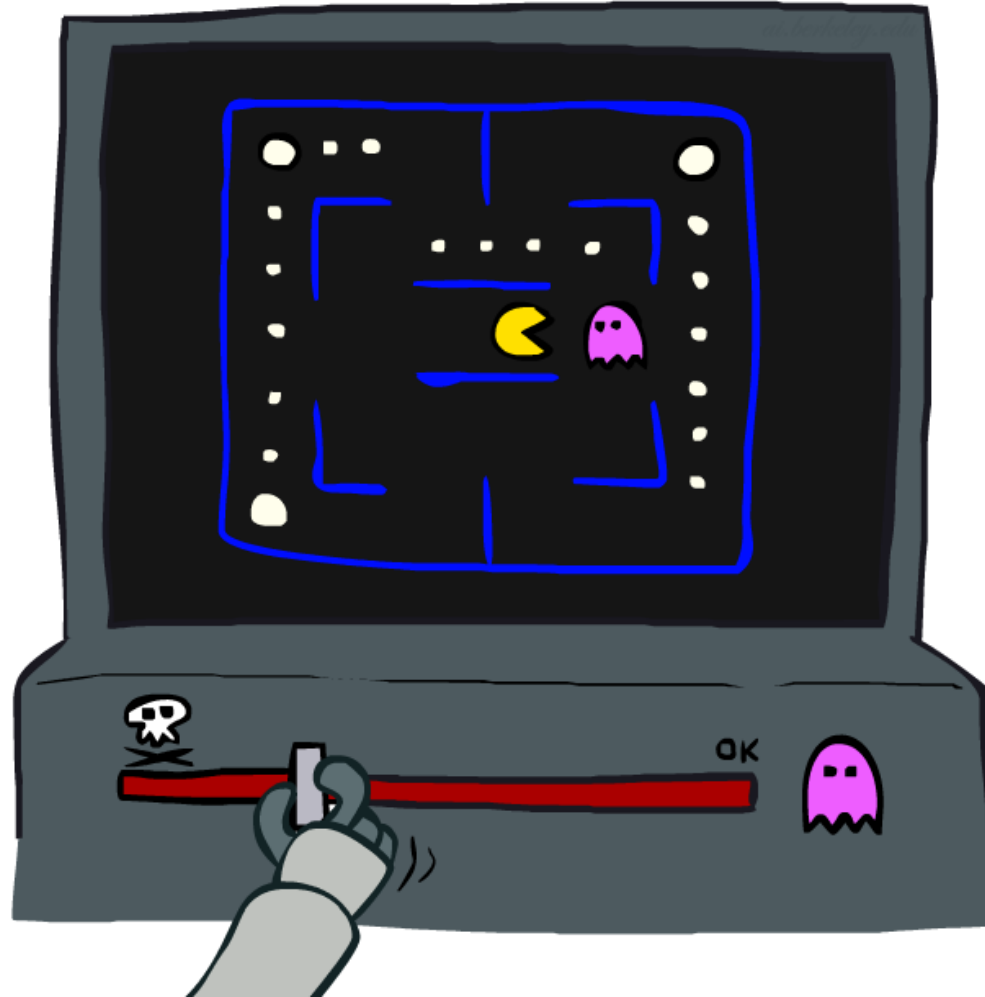
Reinforcement Learning III



Instructor: Brijen Thananjeyan and Aditya Baradwaj, University of California, Berkeley

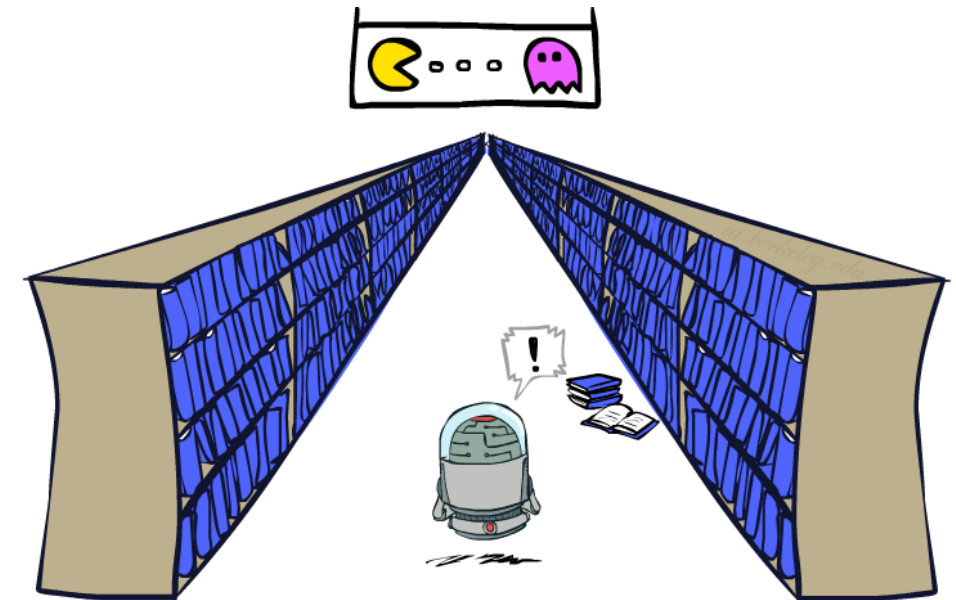
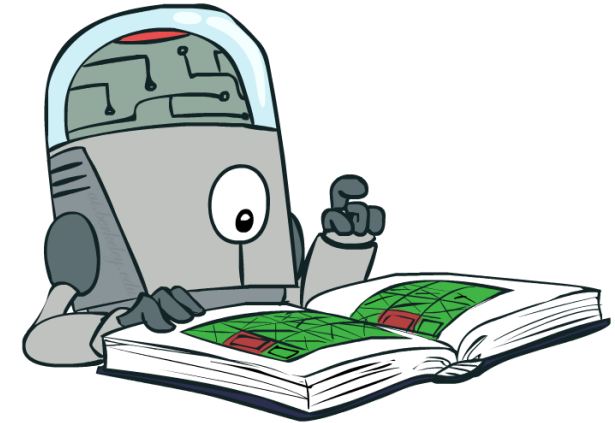
[These slides were created by Dan Klein, Pieter Abbeel, Anca Dragan, Sergey Levine. <http://ai.berkeley.edu>.]

Approximate Q-Learning



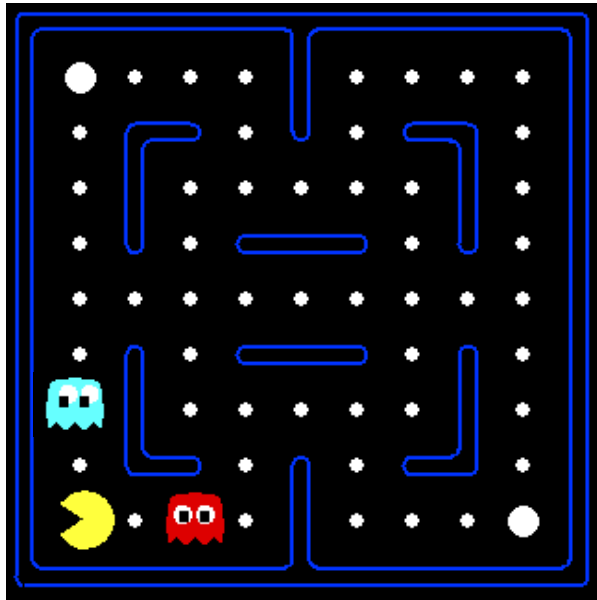
Generalizing Across States

- 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

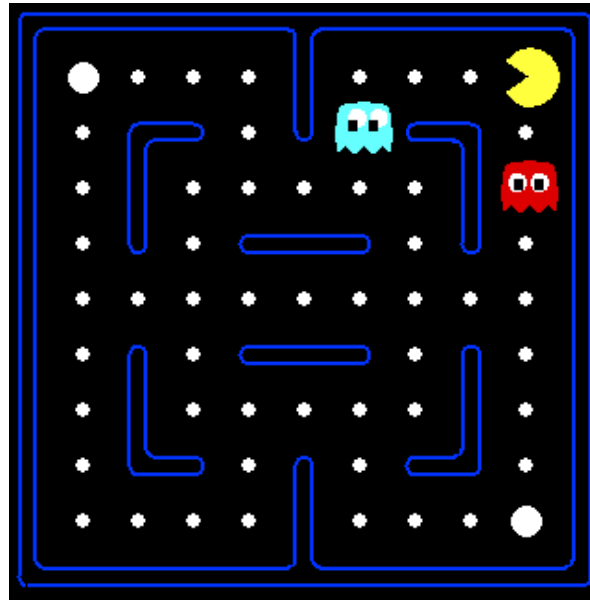


Example: Pacman

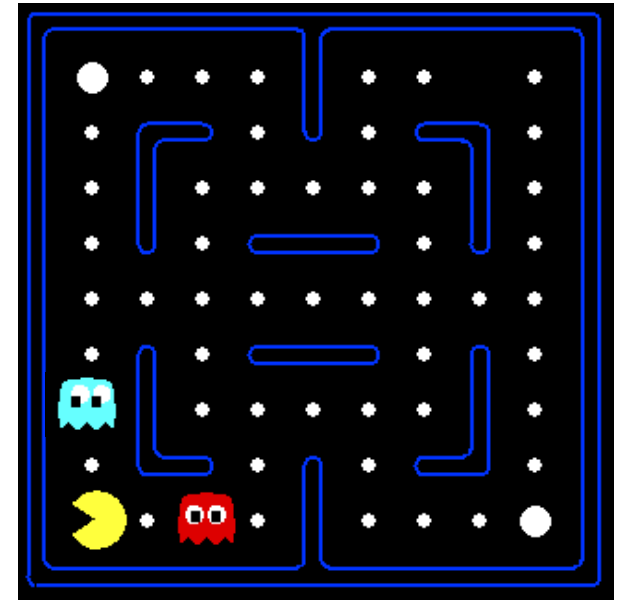
Let's say we discover through experience that this state is bad:



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



Or even this one!



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

Video of Demo Q-Learning Pacman – Tiny – Watch All



Video of Demo Q-Learning Pacman – Tiny – Silent Train

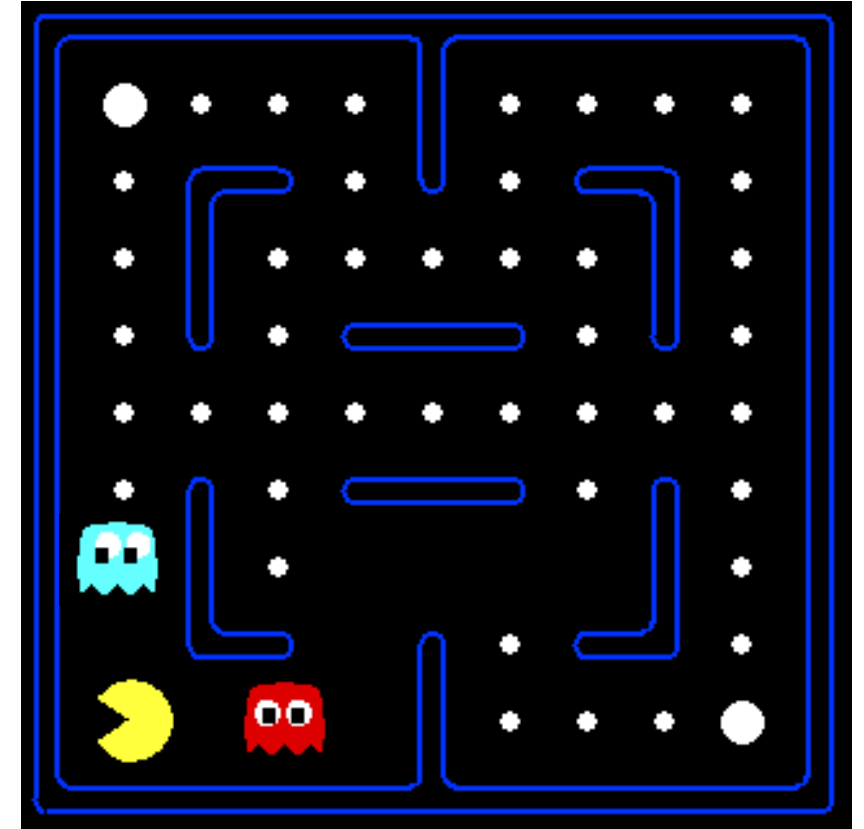


Video of Demo Q-Learning Pacman – Tricky – Watch All



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
 - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
 - Example features:
 - Distance to closest ghost
 - Distance to closest dot
 - Number of ghosts
 - $1 / (\text{dist to dot})^2$
 - Is Pacman in a tunnel? (0/1)
 - etc.
 - Is it the exact state on this slide?
 - Can also describe (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:

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

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

- 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

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

- Q-learning with linear Q-functions:

transition = (s, a, r, s')

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

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

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

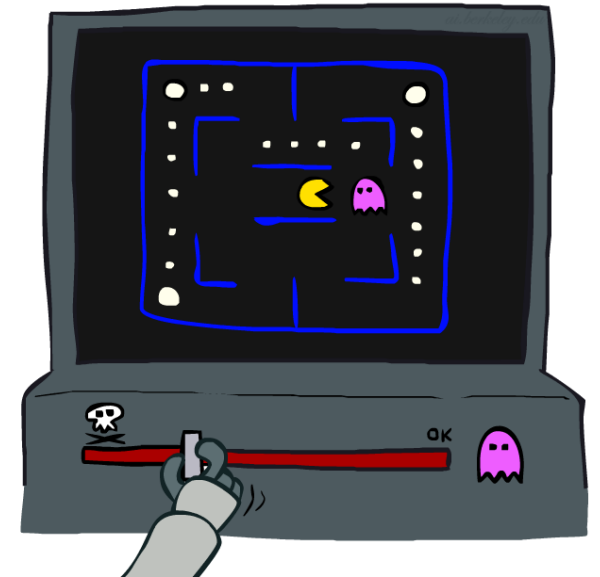
Exact Q's

Approximate Q's

- Intuitive interpretation:

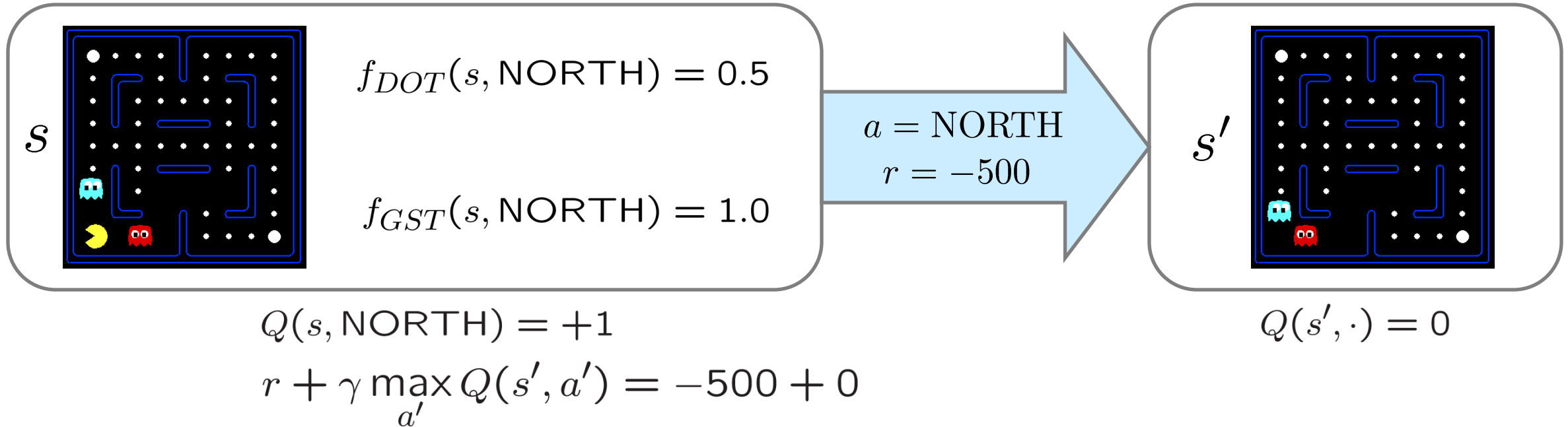
- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares



Example: Q-Pacman

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



difference = -501

$$\begin{aligned} w_{DOT} &\leftarrow 4.0 + \alpha [-501] 0.5 \\ w_{GST} &\leftarrow -1.0 + \alpha [-501] 1.0 \end{aligned}$$

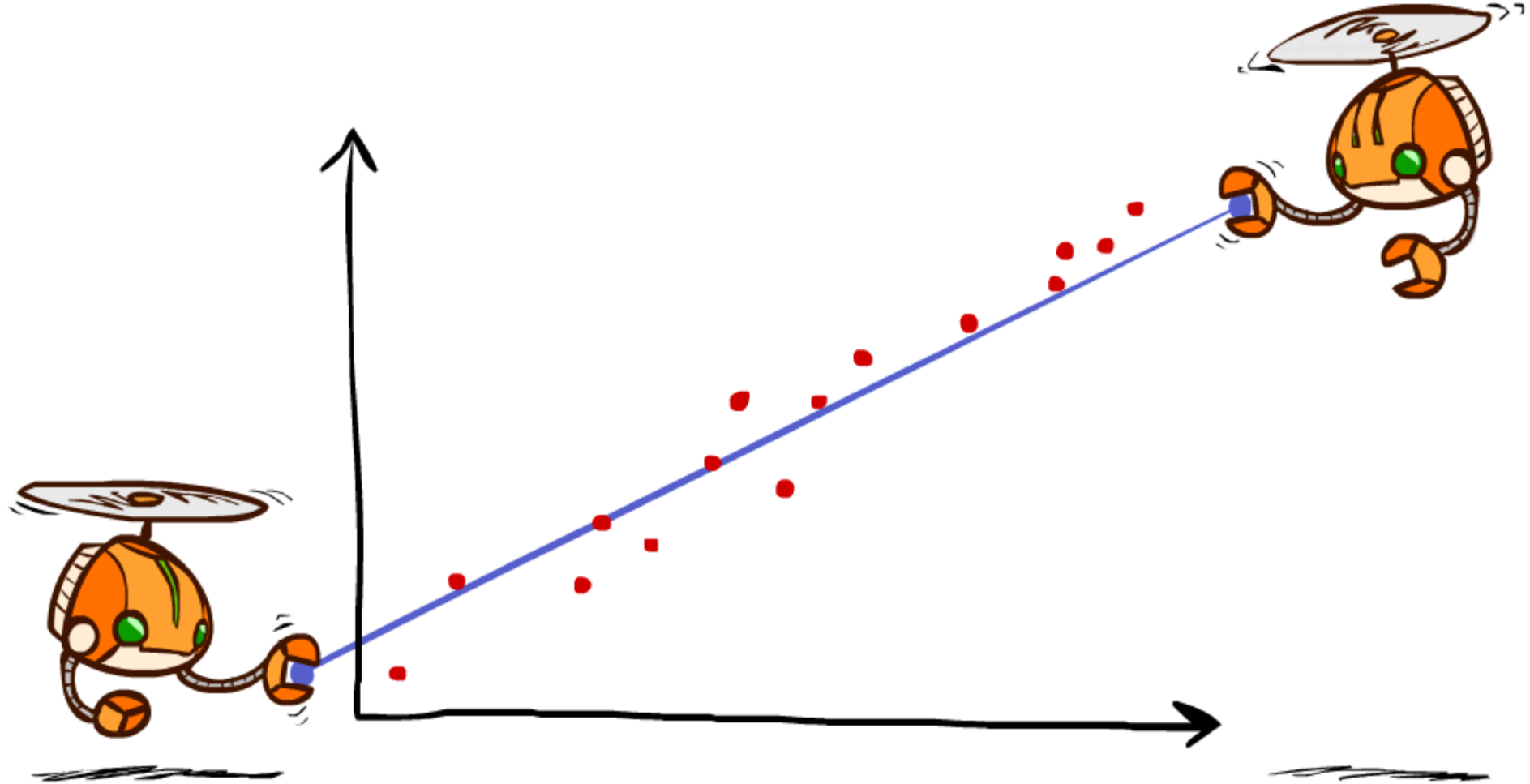
$$Q(s, a) = 3.0 f_{DOT}(s, a) - 3.0 f_{GST}(s, a)$$

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

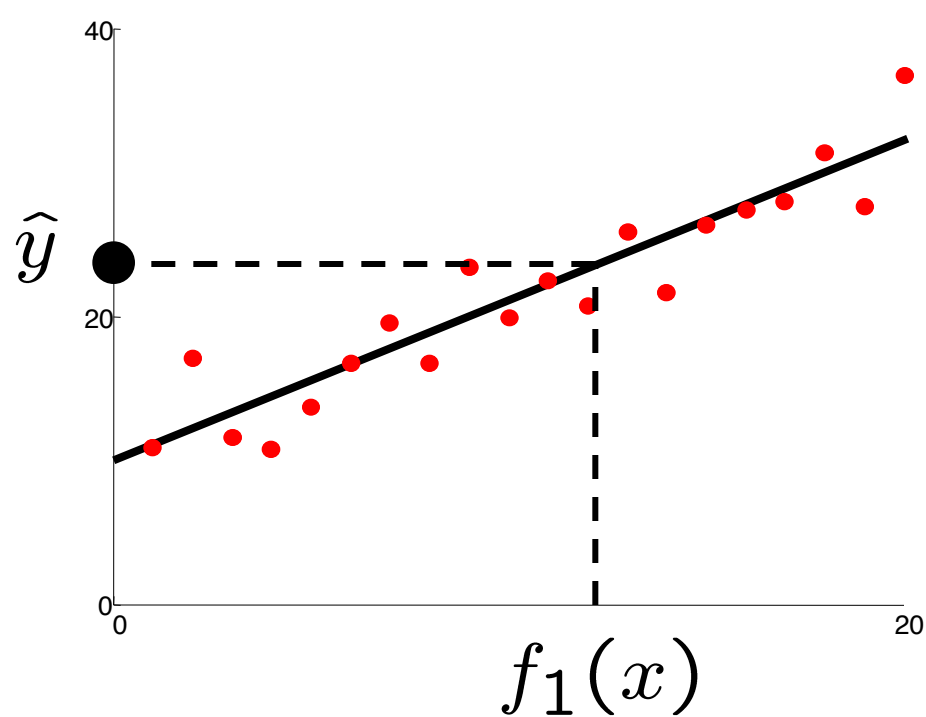
Video of Demo Approximate Q-Learning -- Pacman



Q-Learning and Least Squares

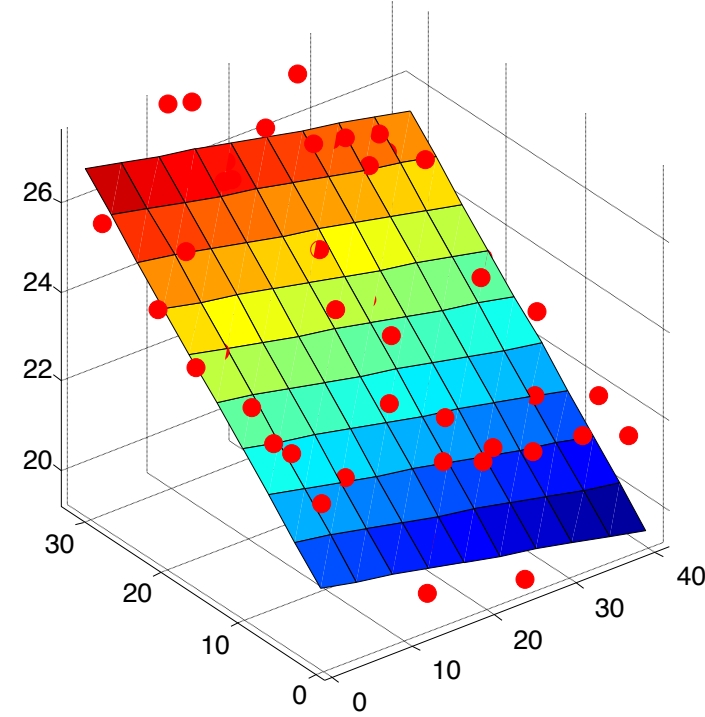


Linear Approximation: Regression*



Prediction:

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

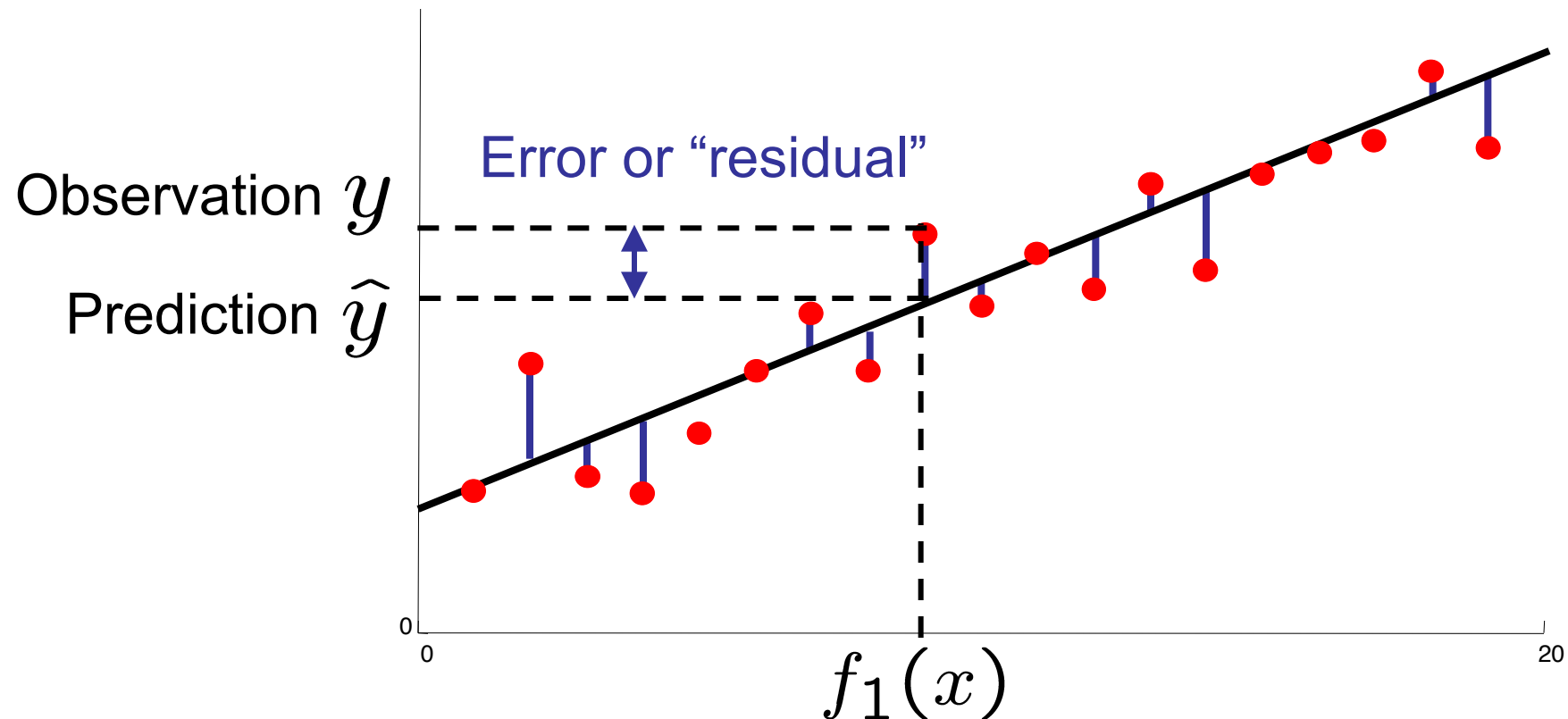


Prediction:

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

Optimization: Least Squares*

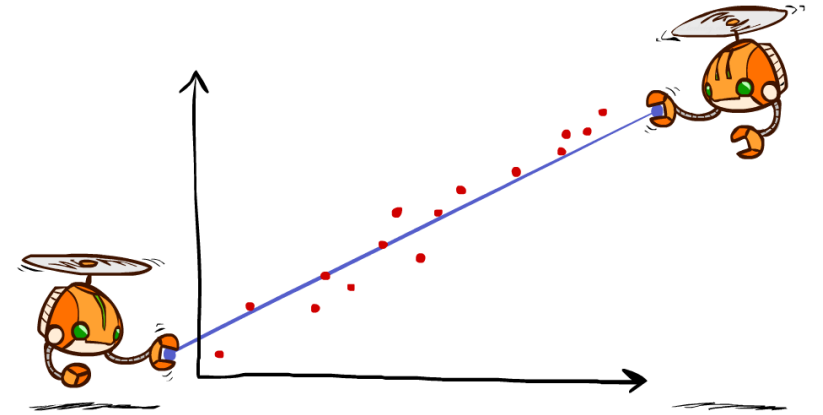
$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



Minimizing Error*

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

$$\begin{aligned}\text{error}(w) &= \frac{1}{2} \left(y - \sum_k w_k f_k(x) \right)^2 \\ \frac{\partial \text{error}(w)}{\partial w_m} &= - \left(y - \sum_k w_k f_k(x) \right) f_m(x) \\ w_m &\leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x) \right) f_m(x)\end{aligned}$$



Approximate q update explained:

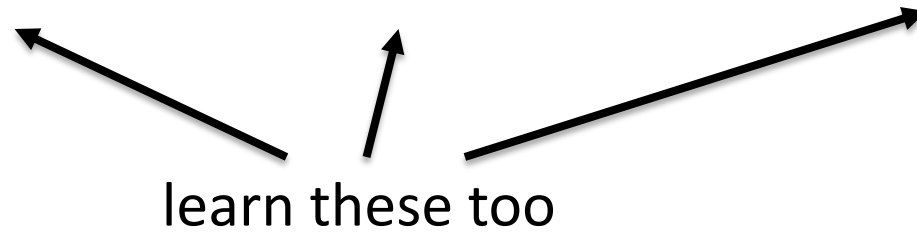
$$w_m \leftarrow w_m + \alpha \left[\underset{\text{“target”}}{r + \gamma \max_a Q(s', a')} - \underset{\text{“prediction”}}{Q(s, a)} \right] f_m(s, a)$$

More Powerful Function Approximation

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

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

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



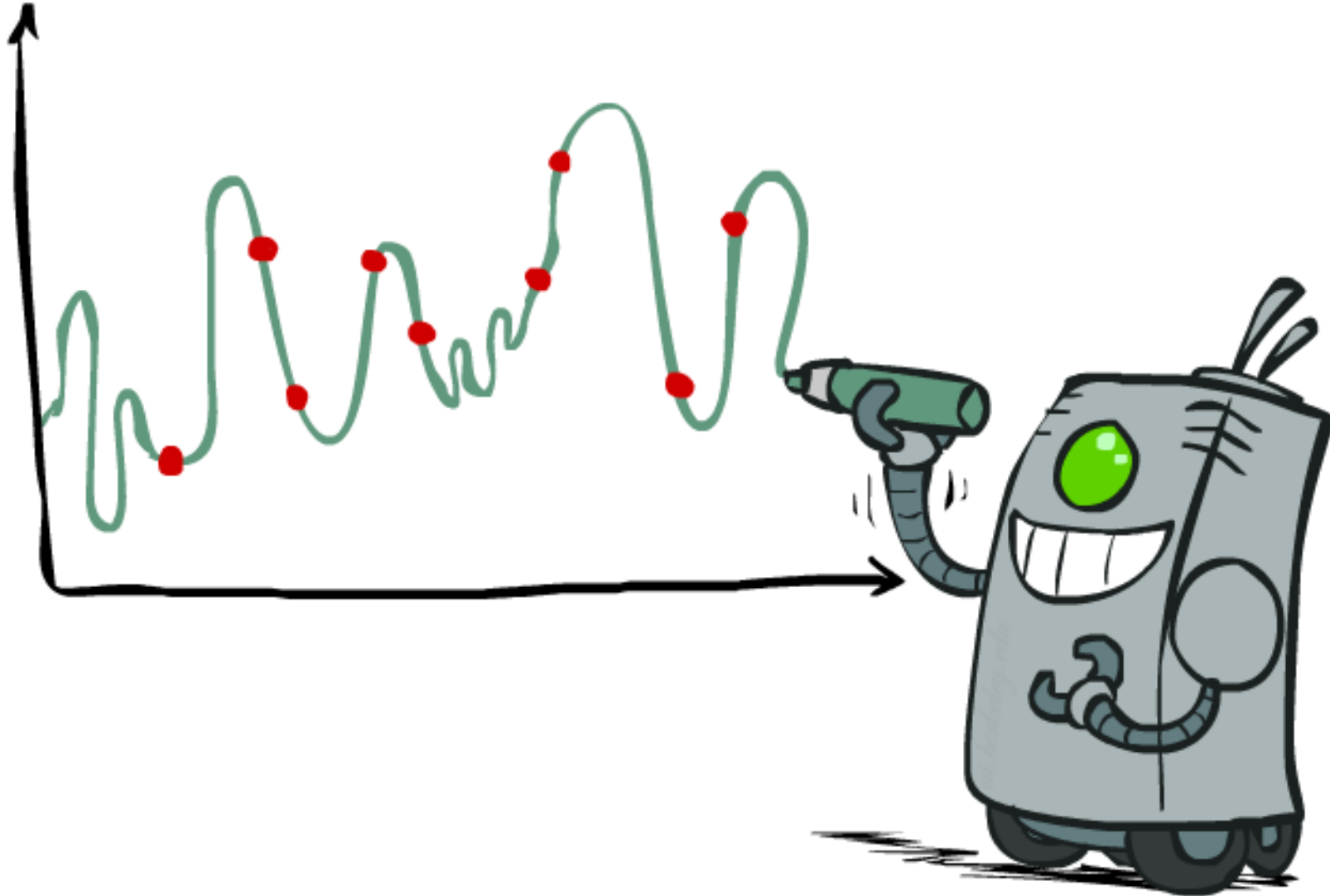
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a) \right] \frac{dQ}{dw_m}(s, a)$$

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

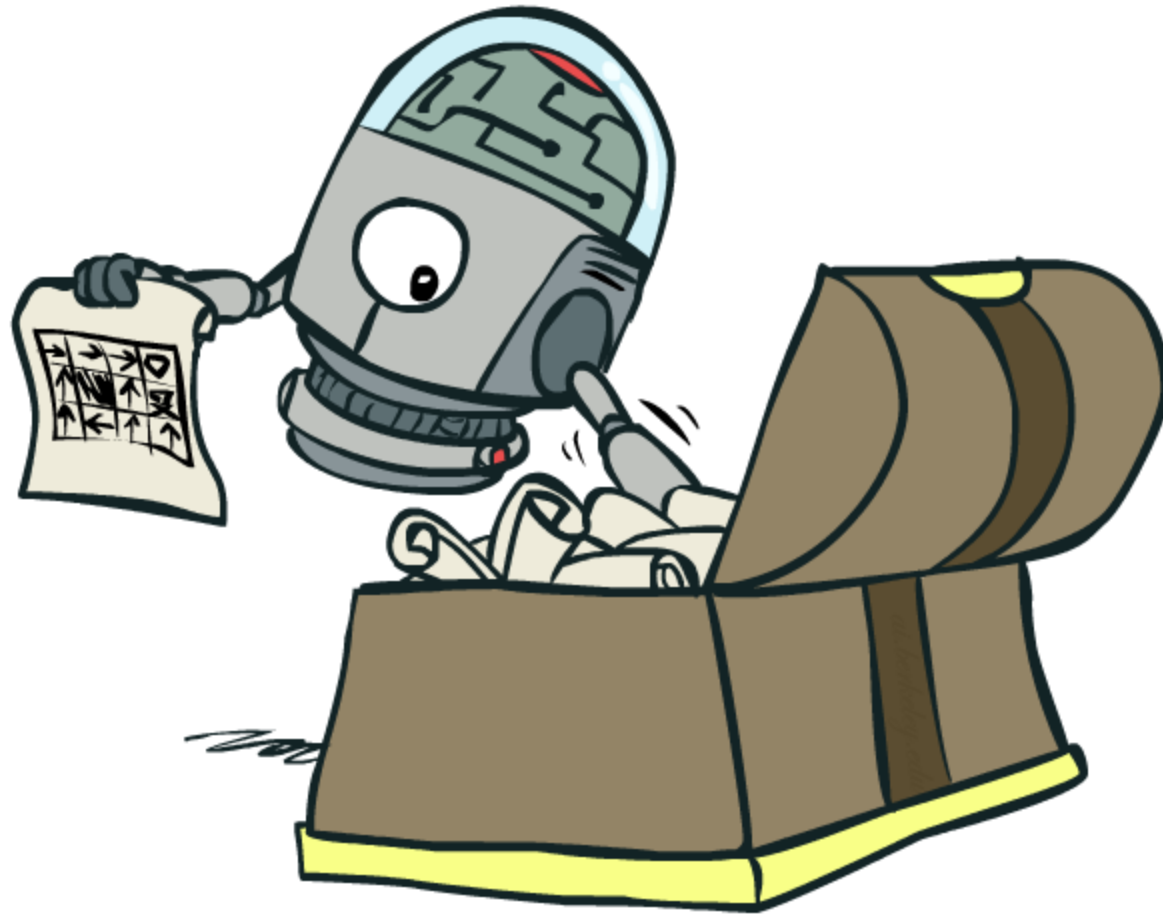
Example: Q-Learning with Neural Nets



Overfitting: Why Limiting Capacity Can Help*



Policy Search



Policy Search

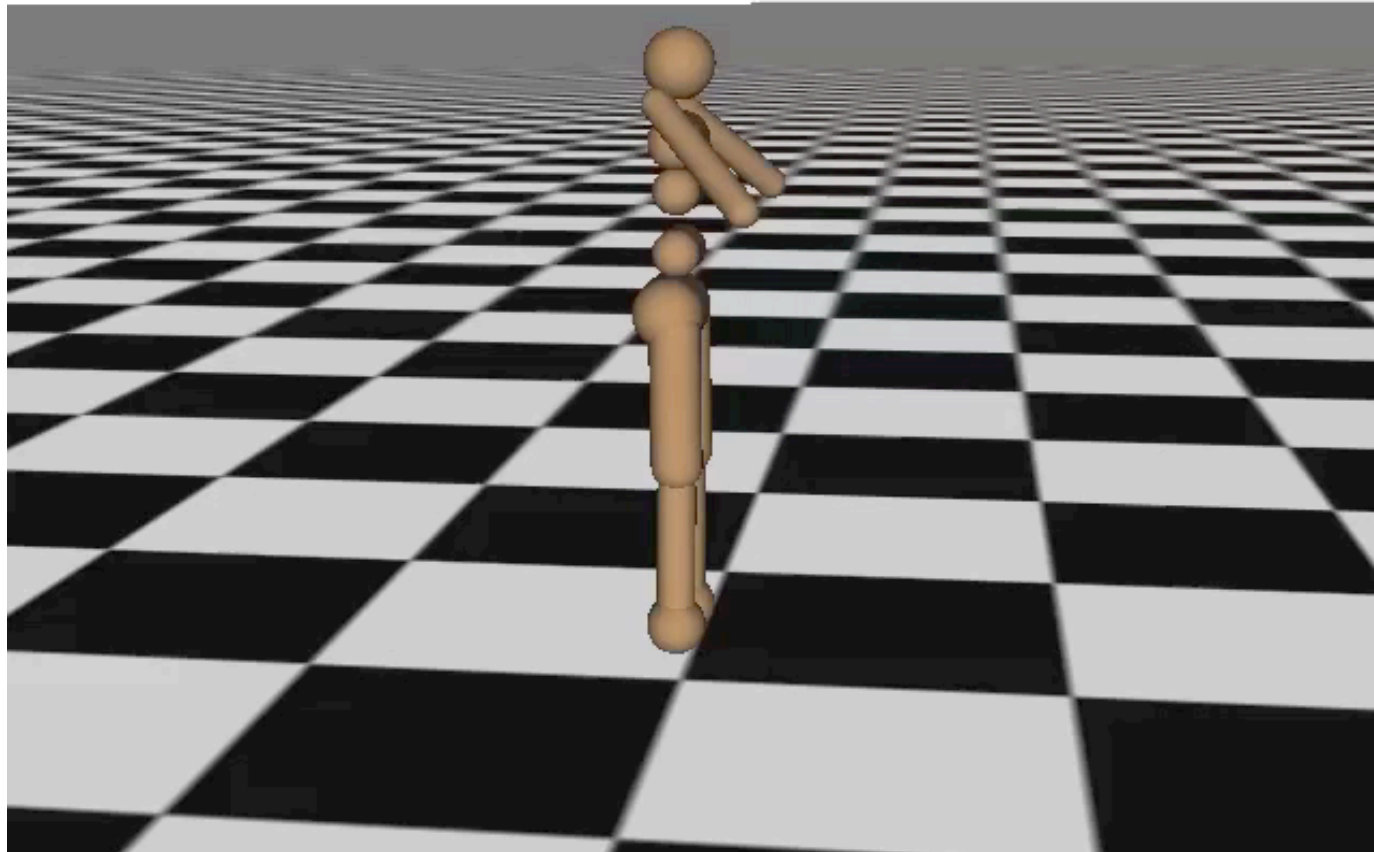
- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
 - E.g. your value functions from project 2 are probably horrible estimates of future rewards, but they still produced good decisions
 - Q-learning's priority: get Q-values close (modeling)
 - Action selection priority: get ordering of Q-values right (prediction)
 - We'll see this distinction between modeling and prediction again later in the course
- Solution: learn policies that maximize rewards, not the values that predict them
- Policy search: directly optimize the policy to attain good rewards via hill-climbing

Policy Search

- Simplest policy search:
 - Start with an initial linear estimator (e.g., random weights on features, like the ones you used for Q-learning)
 - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
 - 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
- Better methods exploit lookahead structure, sample wisely, change multiple parameters...

Policy Search

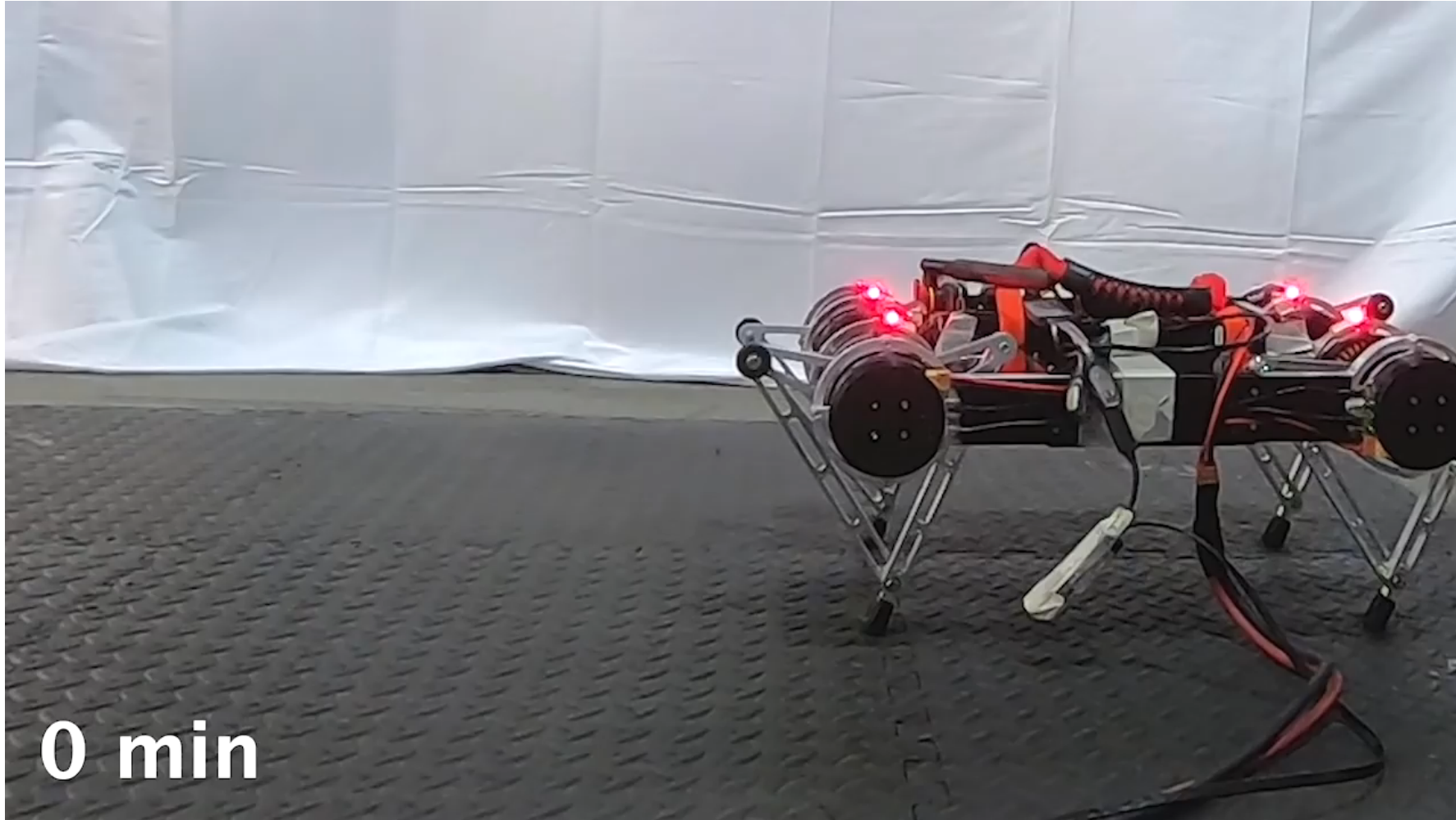
Iteration 0



Pancake Search



Another Example



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

Value / policy iteration

Policy evaluation

Unknown MDP: Model-Based

Goal

**use features
to generalize*

Technique

Compute V^*, Q^*, π^*

VI/PI on approx. MDP

Evaluate a fixed policy π

PE on approx. MDP

Unknown MDP: Model-Free

Goal

**use features
to generalize*

Technique

Compute V^*, Q^*, π^*

Q-learning

Evaluate a fixed policy π

Value Learning