CSE 473: Artificial Intelligence

Reinforcement Learning

Hanna Hajishirzi

Many slides over the course adapted from either Luke Zettlemoyer, Pieter Abbeel, Dan Klein, Stuart Russell or Andrew Moore

MDP 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

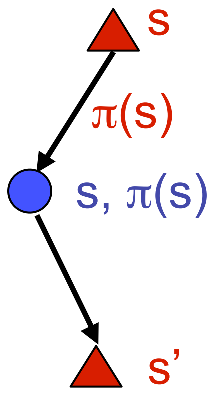
Unknown MDP: Model-Free

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

Passive Learning: TD Learning

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

- Big idea: why bother learning T?
  - Update V each time we experience a transition
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!



$$sample = R(s,\pi(s),s') + \gamma V^\pi(s')$$

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

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

Q-Learning Update

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

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

- Learn Q*(s,a) values
  - 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:

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

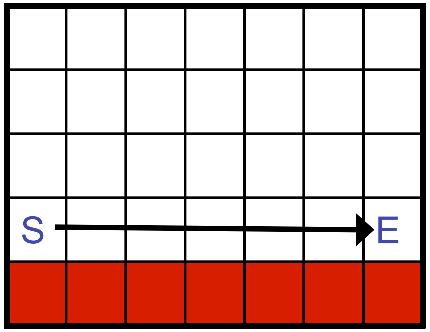
Exploration/Exploitation

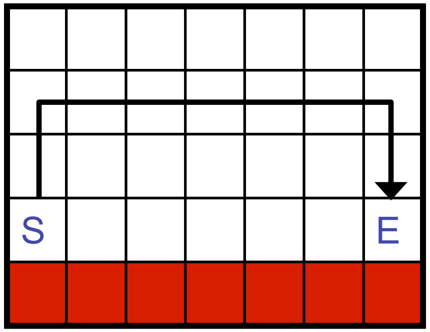
- When to explore
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established
- Exploration function
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$ (exact form not important)
  - Exploration policy $\pi(s')$=

$$\max_{a'} Q_i(s', a') \qquad \text{vs.} \qquad \max_{a'} f(Q_i(s', a'), N(s', a'))$$

Q-Learning Properties

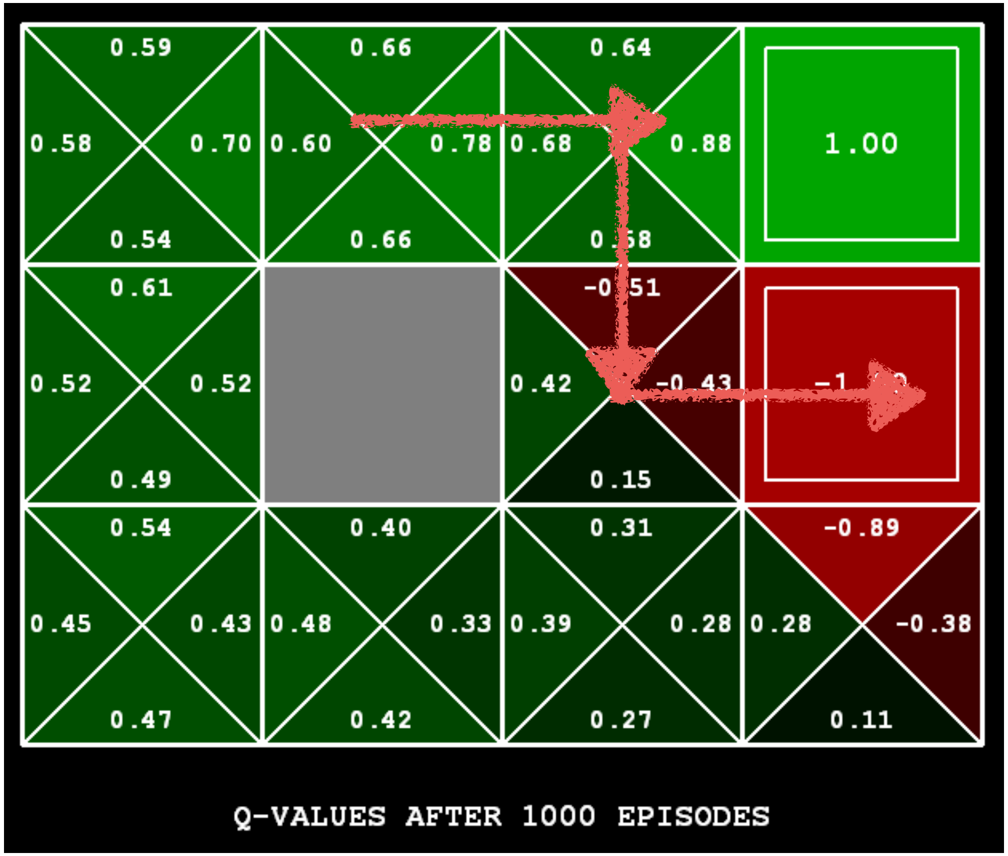
- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Not too sensitive to how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)





Q-Learning Final Solution

- Q-learning produces tables of q-values:



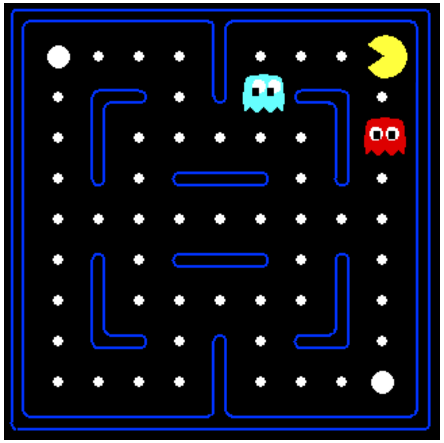
Q-Learning

- 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

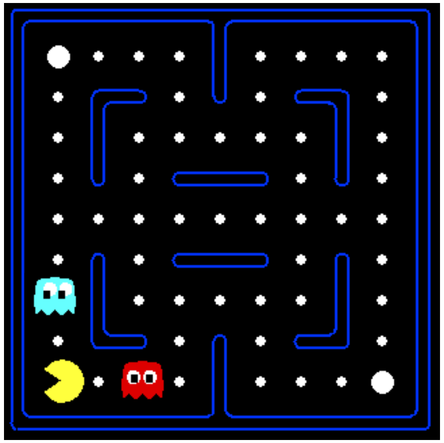
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Example: Pacman

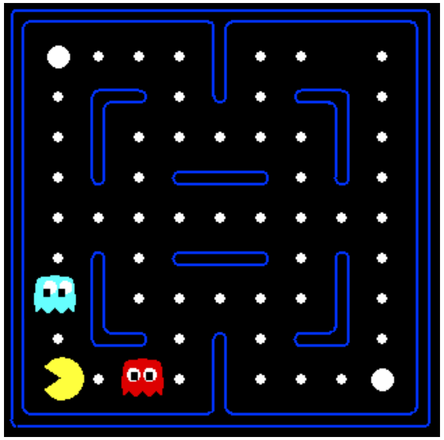
- Let's say we discover through experience that this state is bad:



- In naïve q learning, we know nothing about related states and their q values:

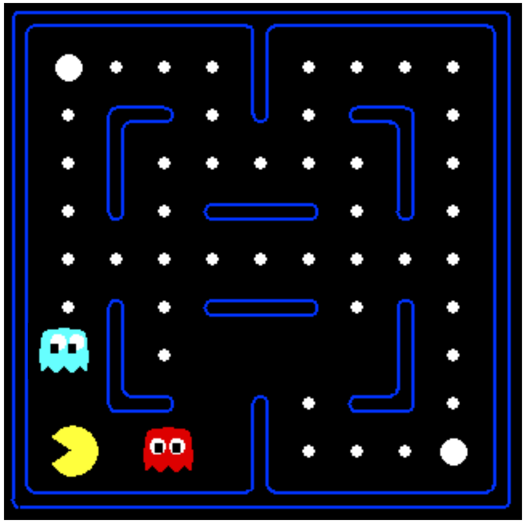


- Or even this third one!



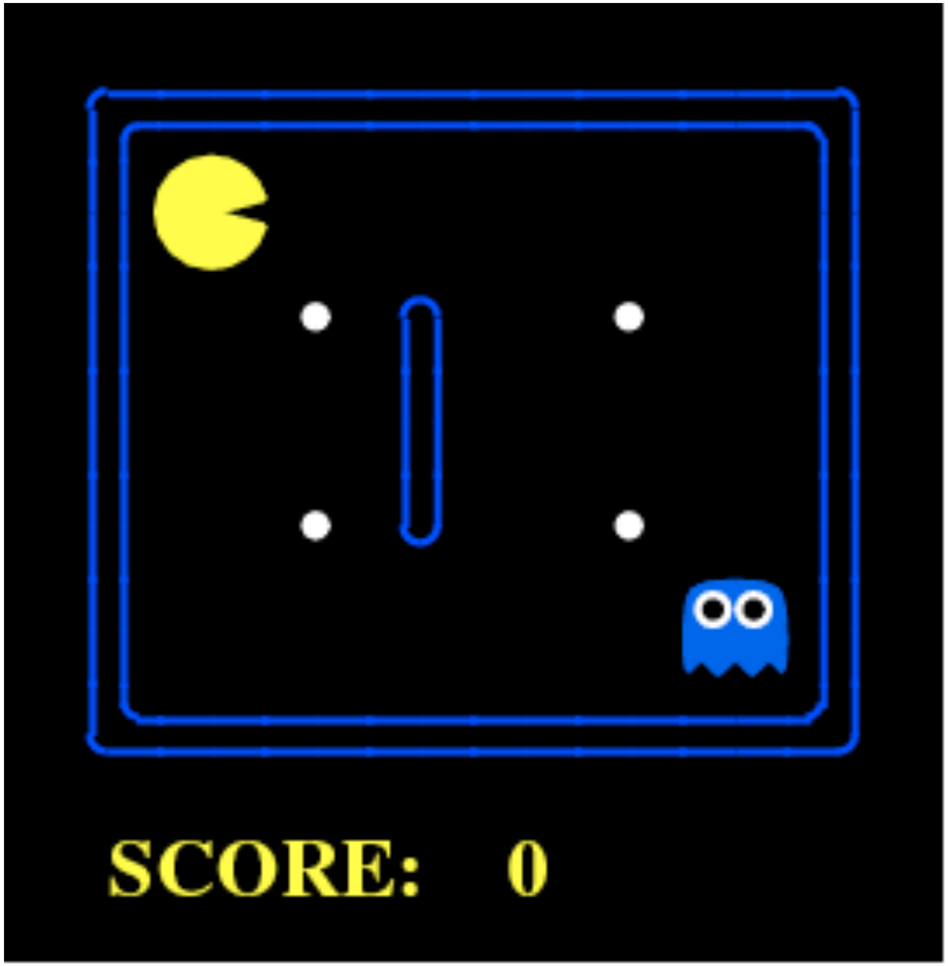
Feature-Based Representations

- 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)



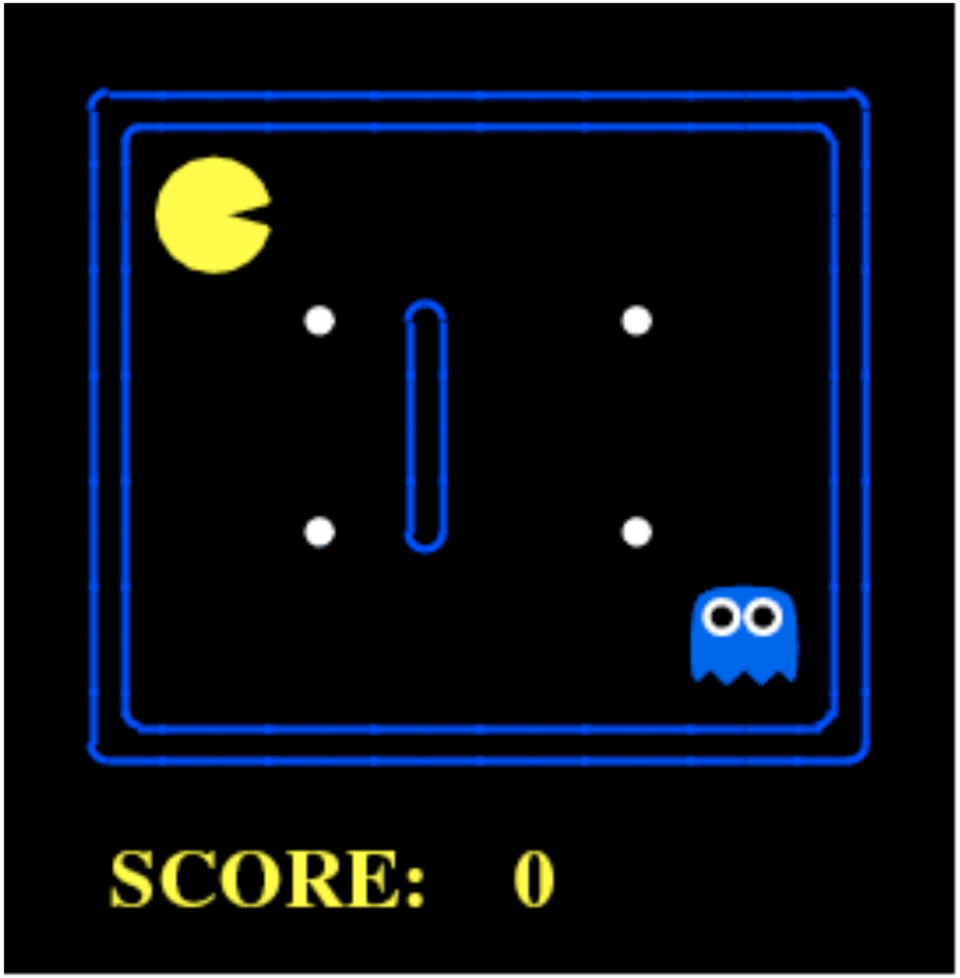
Which Algorithm?

Q-learning, no features, 50 learning trials:



Which Algorithm?

Q-learning, no features, 1000 learning trials:



Linear Feature 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!

Function Approximation

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

- Q-learning with linear q-functions:

$$transition \;= (s,a,r,s')$$

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

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

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

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

Example: Q-Pacman

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

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

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

$$Q(s', \cdot) = 0 \quad Q(s,a) = +1$$

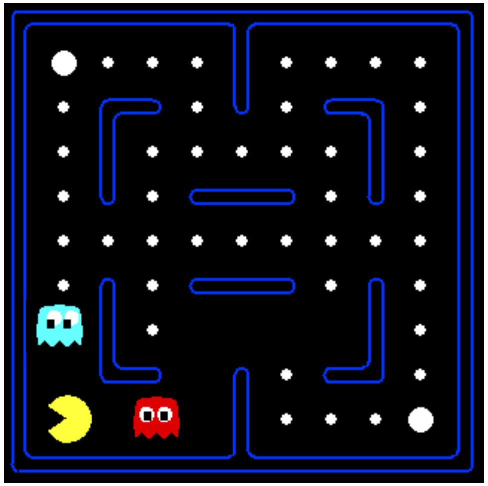
$$R(s,a,s') = -500$$

$$correction = -501$$

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

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

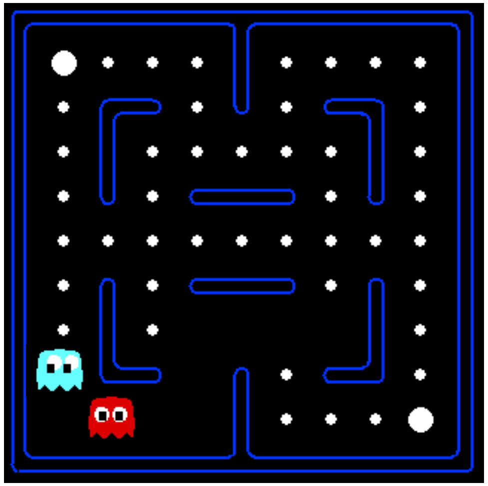
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$



s

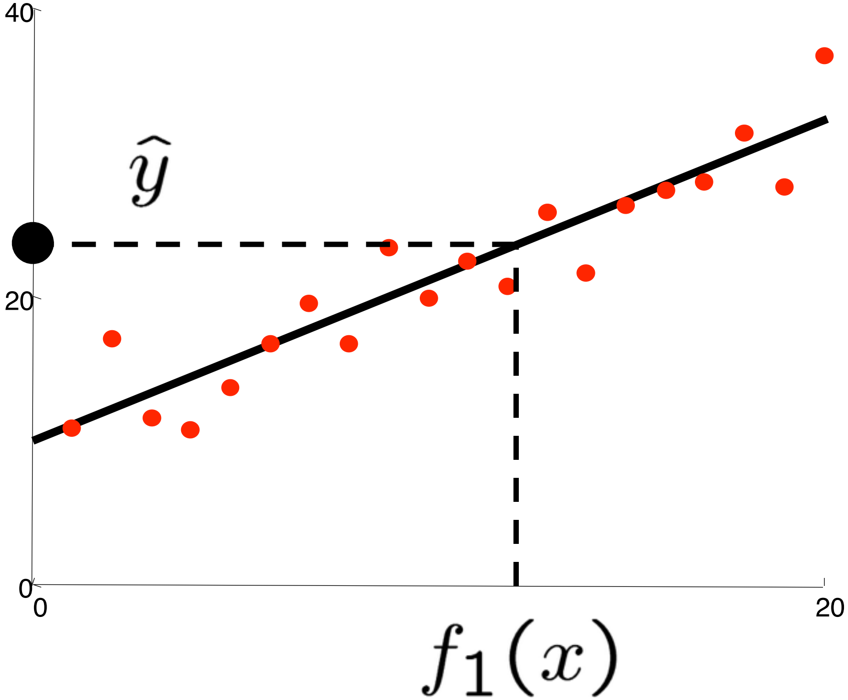
$$a = \text{NORTH}$$

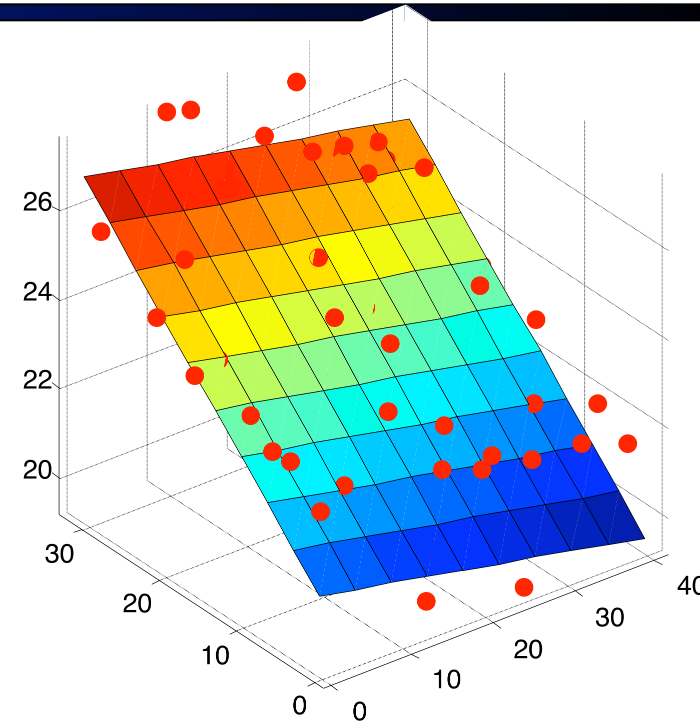
$$r = -500$$



s'

Linear Regression





Prediction

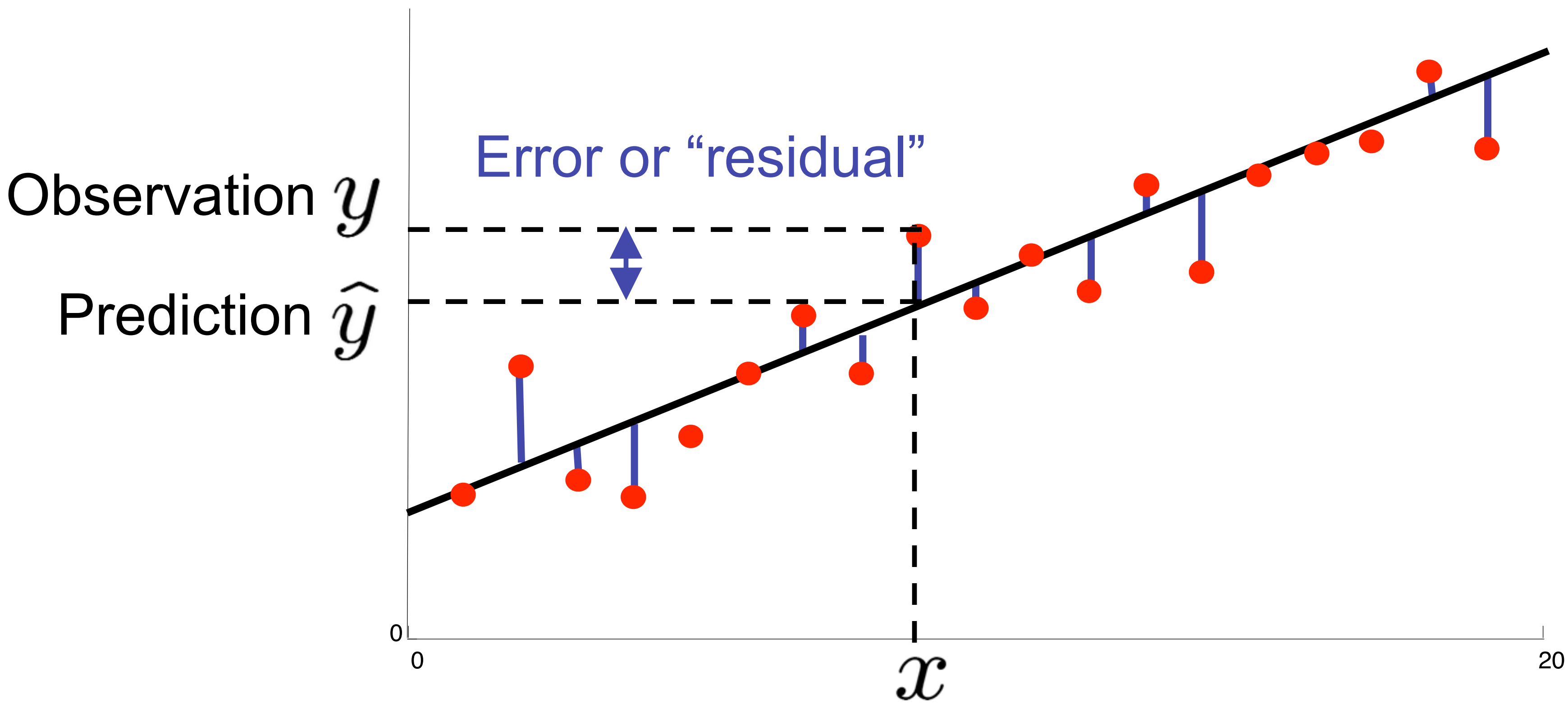
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction

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

Ordinary Least Squares (OLS)

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



Minimizing Error

Imagine we had only one point x with features f(x):

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

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

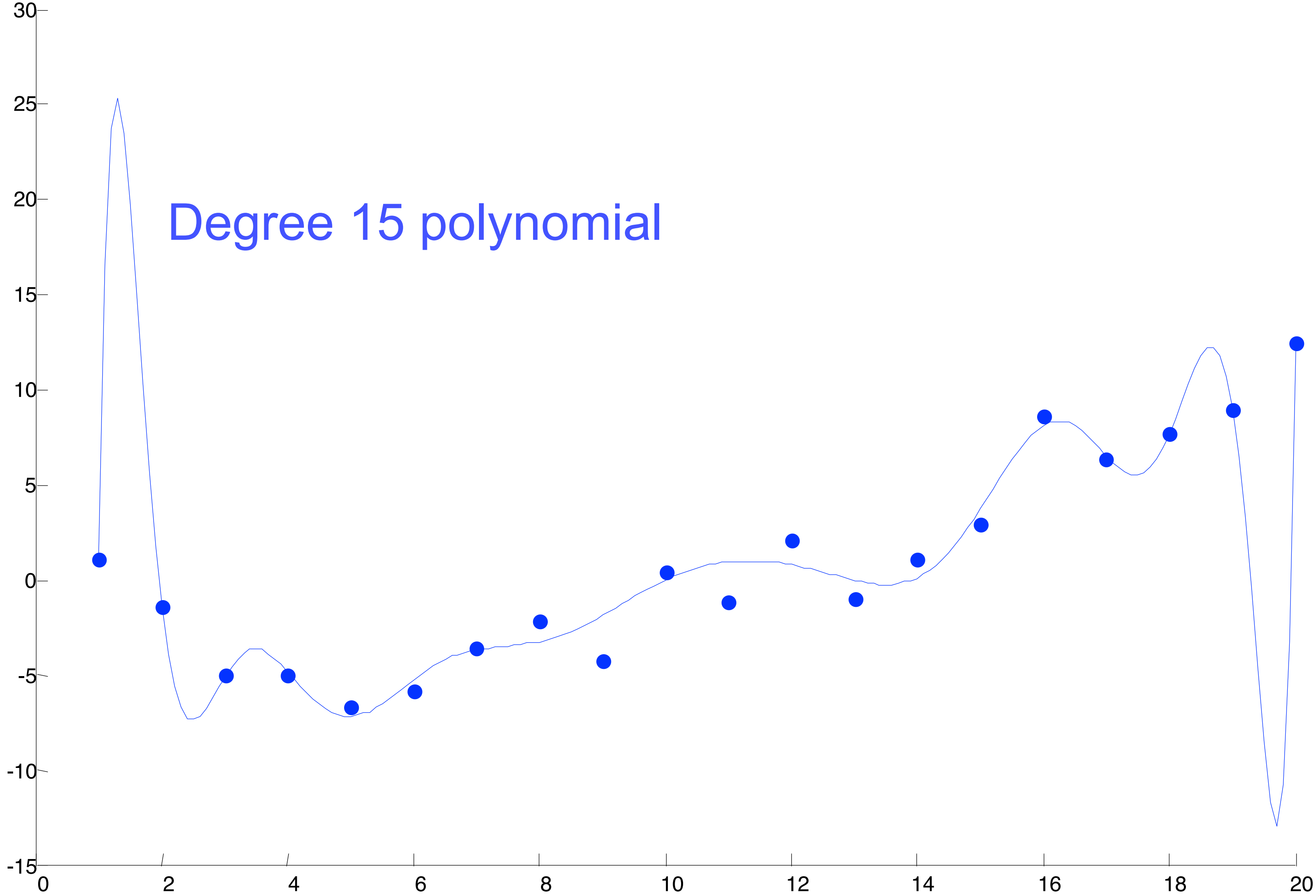
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update:

"target" "prediction"

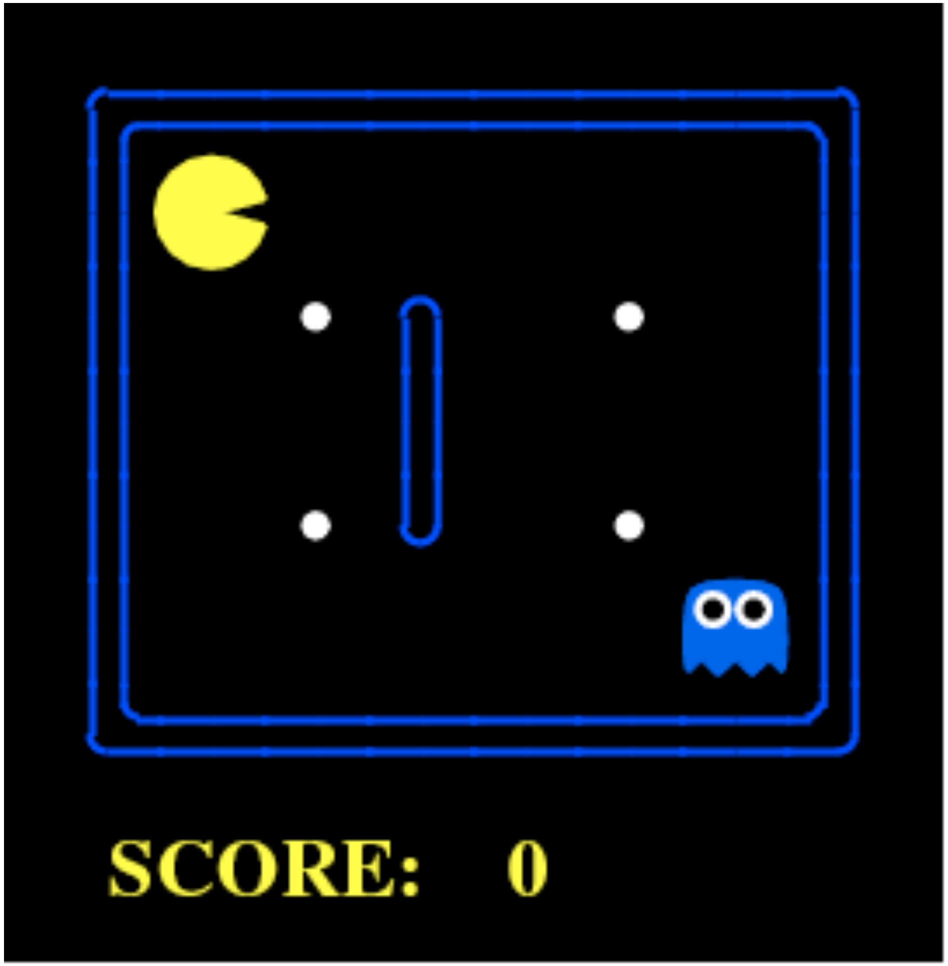
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s,a)\right] f_m(s,a)$$

Overfitting



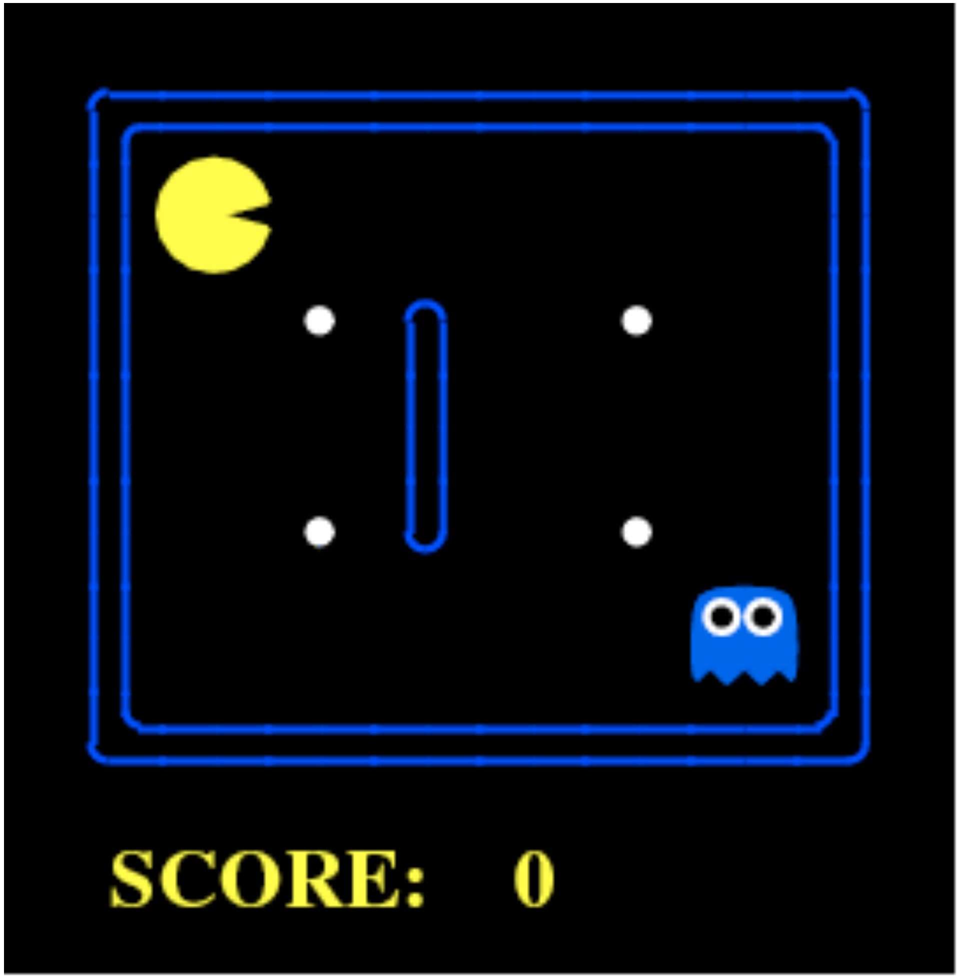
Which Algorithm?

Q-learning, no features, 50 learning trials:



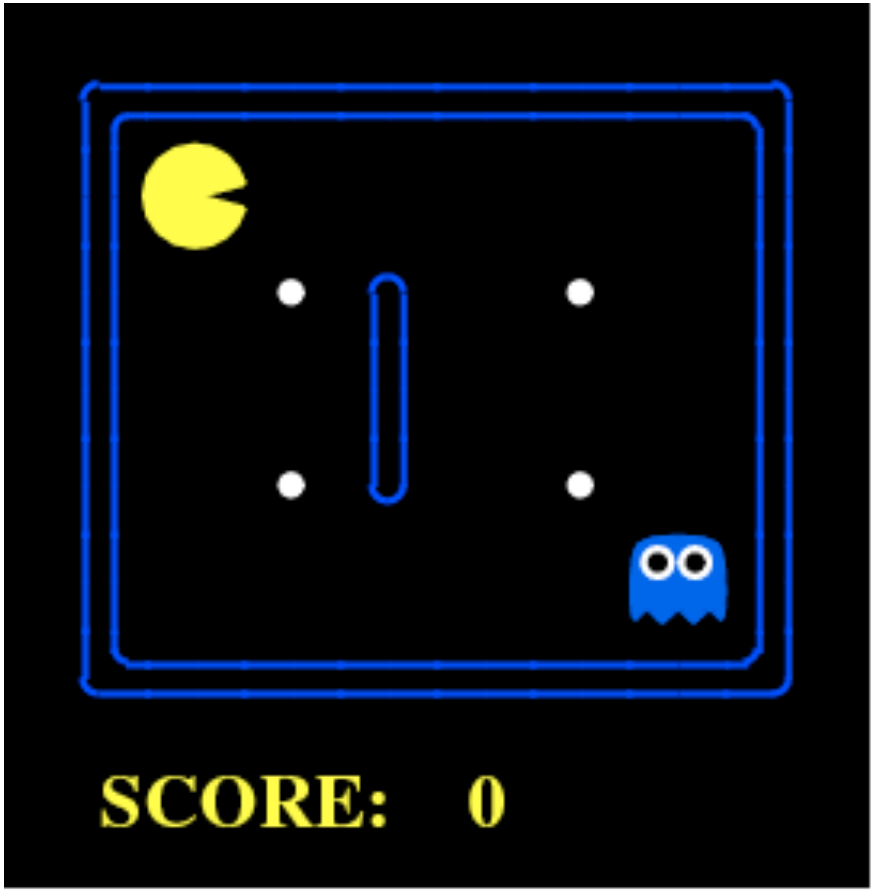
Which Algorithm?

Q-learning, no features, 1000 learning trials:



Which Algorithm?

Q-learning, simple features, 50 learning trials:



Policy Search*

Policy Search*

- Problem: often the feature-based policies that work well 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
  - We'll see this distinction between modeling and prediction again later in the course
- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards
- This is the idea behind policy search, such as what controlled the upside-down helicopter

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

- 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

Policy Search*

- Advanced policy search:
  - Write a stochastic (soft) policy:

$$\pi_w(s) \propto e^{\sum_i w_i f_i(s,a)}$$

  - Turns out you can efficiently approximate the derivative of the returns with respect to the parameters w (details in the book, optional material)
  - Take uphill steps, recalculate derivatives, etc.