

CSEP 573: Artificial Intelligence

Reinforcement Learning

Luke Zettlemoyer

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

Outline

- Reinforcement Learning
 - Passive Learning
 - TD Updates
 - Q-value iteration
 - Q-learning
 - Linear function approximation

What is it doing?

-

Step Delay: 0.10000

+

-

Epsilon: 0.500

+

-

Discount: 0.800

+

-

Learning Rate: 0.800

+



Step: 75

Position: 63

Velocity: -6.04

100-step Avg Velocity: 0.68

Reinforcement Learning

- Reinforcement learning:

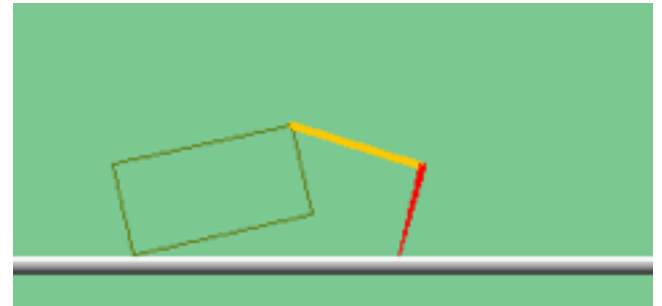
- Still have an MDP:

- A set of states $s \in S$
 - A set of actions (per state) A
 - A model $T(s,a,s')$
 - A reward function $R(s,a,s')$

- Still looking for a policy $\pi(s)$

- New twist: **don't know T or R**

- I.e. don't know which states are good or what the actions do
 - Must actually try actions and states out to learn

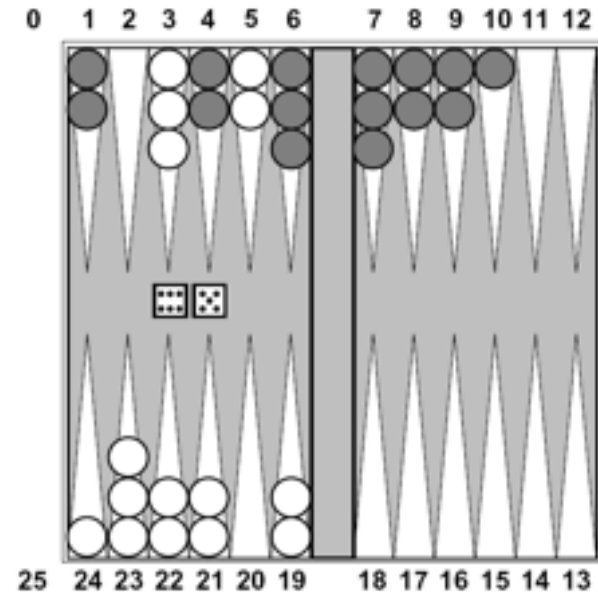


Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
 - Rewards: food, pain, hunger, drugs, etc.
 - Mechanisms and sophistication debated
- Example: foraging
 - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
 - Bees have a direct neural connection from nectar intake measurement to motor planning area

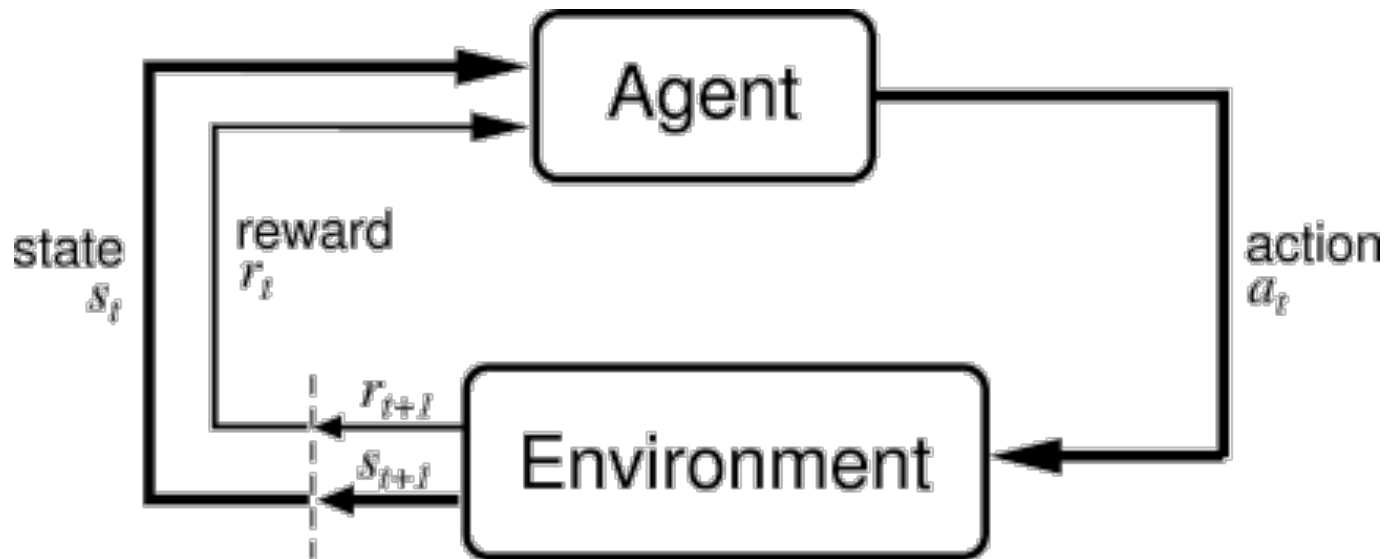
Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to $V(s)$ using a neural network
- Combined with depth 3 search, one of the top 3 players in the world
- You could imagine training Pacman this way...
- ... but it's tricky! (It's also P3)

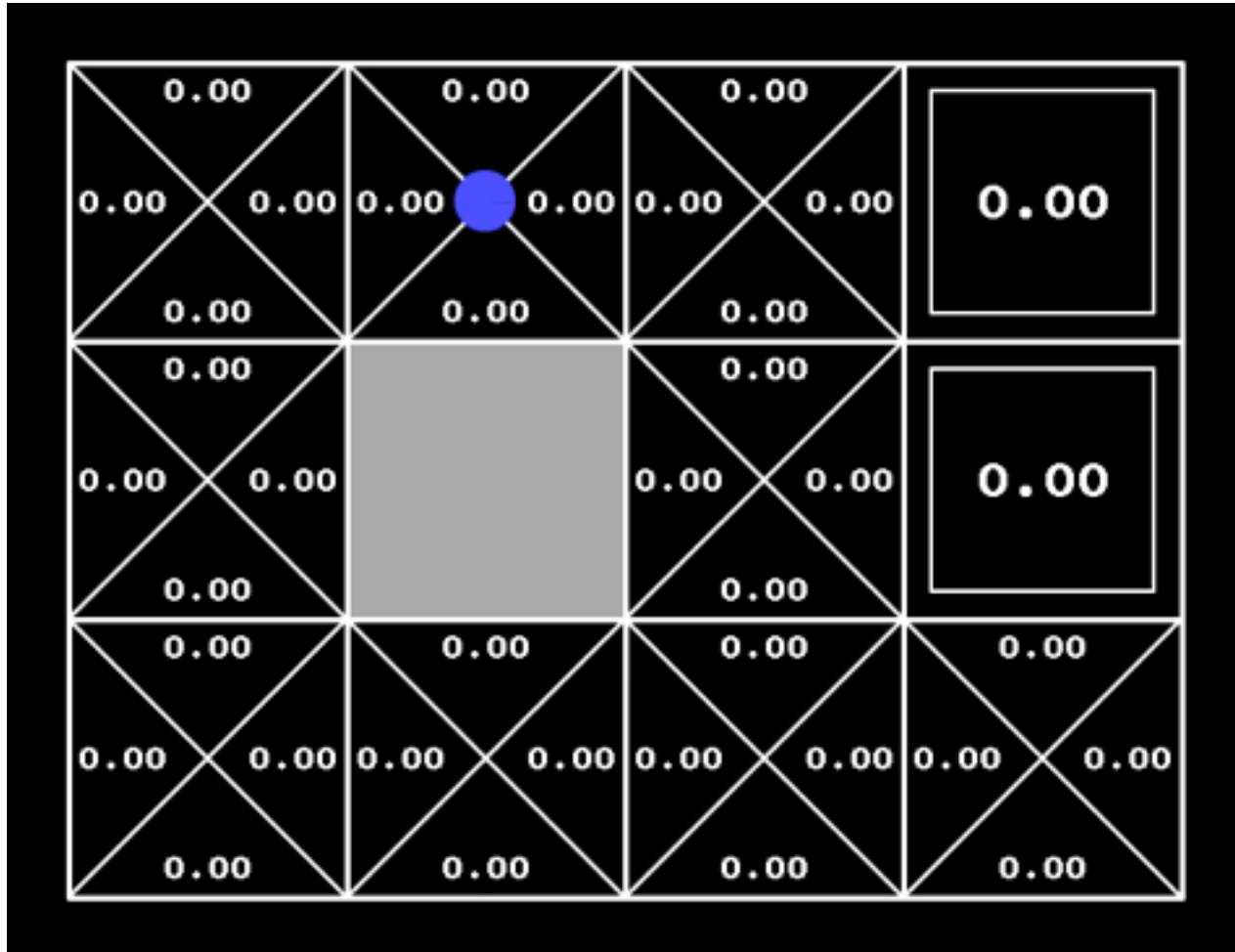


Reinforcement Learning

- 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**



What is the dot doing?



Key Ideas for Learning

- **Online vs. Batch**
 - Learn while exploring the world, or learn from fixed batch of data
- **Active vs. Passive**
 - Does the learner actively choose actions to gather experience? or, is a fixed policy provided?
- **Model based vs. Model free**
 - Do we estimate $T(s,a,s')$ and $R(s,a,s')$, or just learn values/policy directly

Detour: Sampling Expectations

- Want to compute an expectation weighted by $P(x)$:

$$E[f(x)] = \sum_x P(x) f(x)$$

- Model-based:** estimate $P(x)$ from samples, compute expectation

$$\begin{aligned} x_i &\sim P(x) \\ \hat{P}(x) &= \text{count}(x)/k \end{aligned} \quad E[f(x)] \approx \sum_x \hat{P}(x) f(x)$$

- Model-free:** estimate expectation directly from samples

$$x_i \sim P(x) \quad E[f(x)] \approx \frac{1}{k} \sum_i f(x_i)$$

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

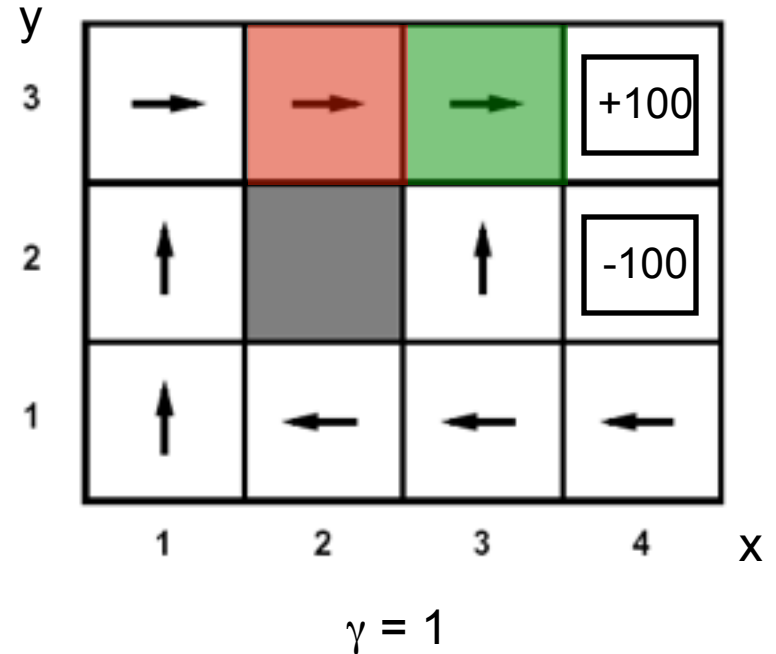
Model-Based Learning

- Idea:
 - Learn the model empirically (rather than values)
 - Solve the MDP as if the learned model were correct
- Empirical model learning
 - Simplest case:
 - Count outcomes for each s,a
 - Normalize to give estimate of $T(s,a,s')$
 - Discover $R(s,a,s')$ the first time we experience (s,a,s')
 - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. “stationary noise”)

Example: Model-Based Learning

Episodes:

- | | |
|-----------------|-----------------|
| (1,1) up -1 | (1,1) up -1 |
| (1,2) up -1 | (1,2) up -1 |
| (1,2) up -1 | (1,3) right -1 |
| (1,3) right -1 | (2,3) right -1 |
| (2,3) right -1 | (3,3) right -1 |
| (3,3) right -1 | (3,2) up -1 |
| (3,2) up -1 | (4,2) exit -100 |
| (3,3) right -1 | (done) |
| (4,3) exit +100 | |
| (done) | |



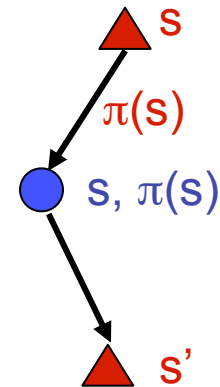
$$T(\langle 3,3 \rangle, \text{right}, \langle 4,3 \rangle) = 1 / 3$$

$$T(\langle 2,3 \rangle, \text{right}, \langle 3,3 \rangle) = 2 / 2$$

Model-free Learning

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

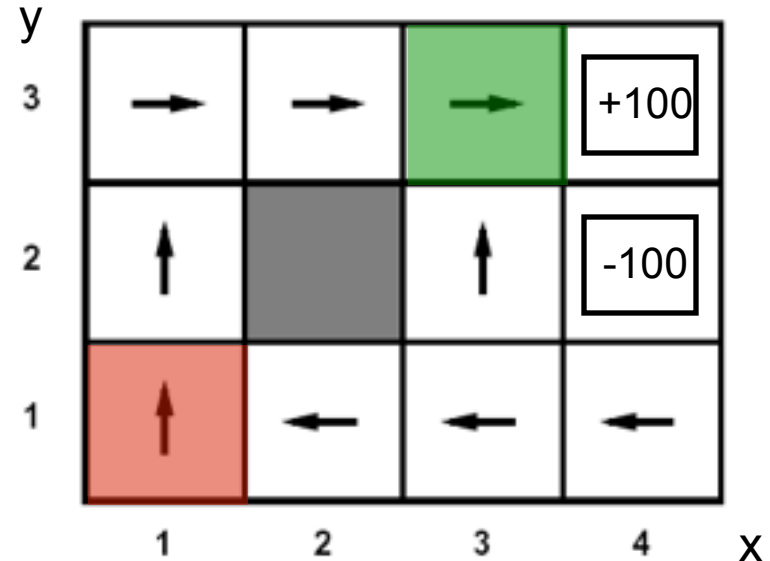
- **Big idea:** why bother learning T ?
- **Question:** how can we compute V if we don't know T ?
 - Use direct estimation to sample complete trials, average rewards at end
 - Use sampling to approximate the Bellman updates, compute new values during each learning step



Simple Case: Direct Estimation

- Average the total reward for every trial that visits a state:

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



$$\gamma = 1, R = -1$$

$$V(1,1) \sim (92 + -106) / 2 = -7$$

$$V(3,3) \sim (99 + 97 + -102) / 3 = 31.3$$

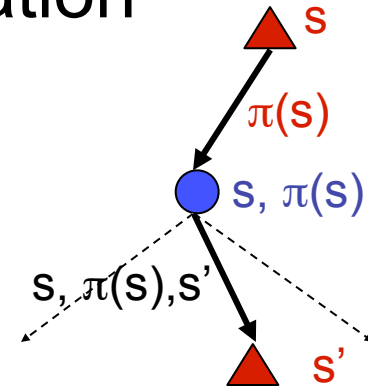
Problems with Direct Evaluation

- What's good about direct evaluation?
 - It is easy to understand
 - It doesn't require any knowledge of T and R
 - It eventually computes the correct average value using just sample transitions
- What's bad about direct evaluation?
 - It wastes information about state connections
 - Each state must be learned separately
 - So, it takes long time to learn

Towards Better Model-free Learning

Review: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate V for a fixed policy:
 - New V is expected one-step-look-ahead using current V
 - Unfortunately, need T and R



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

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

Sample Avg to Replace Expectation?

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

- Who needs T and R? Approximate the expectation with samples (drawn from T!)

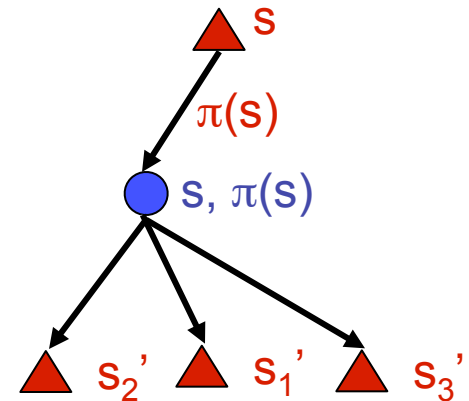
$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_i^\pi(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_i^\pi(s'_2)$$

...

$$\text{sample}_k = R(s, \pi(s), s'_k) + \gamma V_i^\pi(s'_k)$$

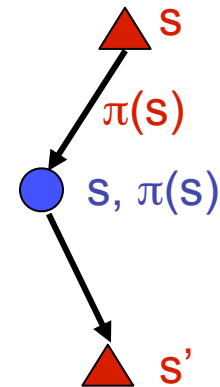
$$V_{i+1}^\pi(s) \leftarrow \frac{1}{k} \sum_i \text{sample}_i$$



Temporal Difference 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!



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

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

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

Detour: Exp. Moving Average

- Exponential moving average
 - 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)
- Easy to compute from the running average

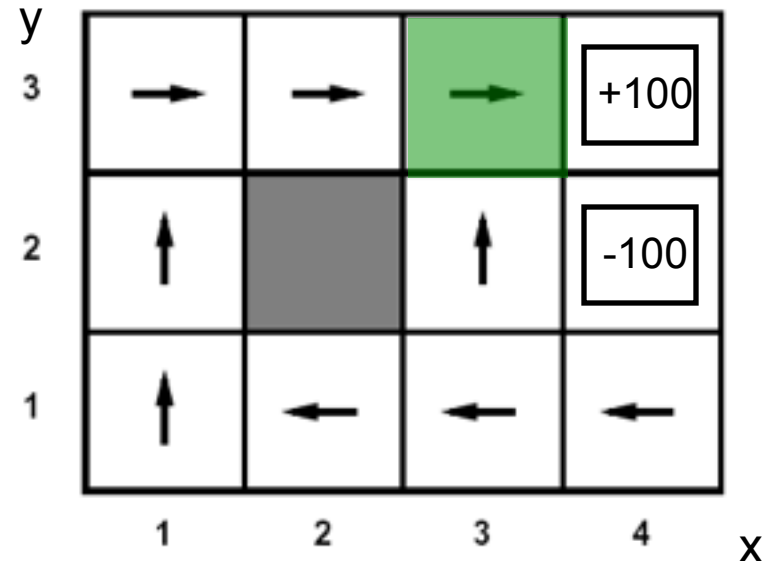
$$\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

TD Policy Evaluation

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

(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,2) up -1	(1,3) right -1
(1,3) right -1	(2,3) right -1
(2,3) right -1	(3,3) right -1
(3,3) right -1	(3,2) up -1
(3,2) up -1	(4,2) exit -100
(3,3) right -1	(done)
(4,3) exit +100	
(done)	



Updates for $V(<3,3>)$:

$$V(<3,3>) = 0.5 \cdot 0 + 0.5 \cdot [-1 + 1 \cdot 0] = -0.5$$

$$V(<3,3>) = 0.5 \cdot -0.5 + 0.5 \cdot [-1 + 1 \cdot 100] = 49.475$$

$$V(<3,3>) = 0.5 \cdot 49.475 + 0.5 \cdot [-1 + 1 \cdot -0.75]$$

Take $\gamma = 1$, $\alpha = 0.5$, $V_0(<4,3>) = 100$, $V_0(<4,2>) = -100$, $V_0 = 0$ otherwise

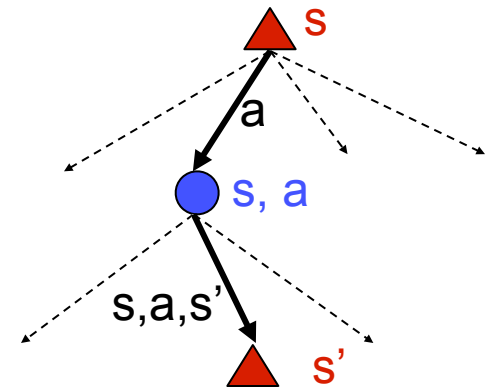
Problems with TD Value Learning

- TD value learning is model-free for policy evaluation (passive learning)
- However, if we want to turn our value estimates into a policy, we're sunk:

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

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

- Idea: learn Q-values directly
- Makes action selection model-free too!



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:

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

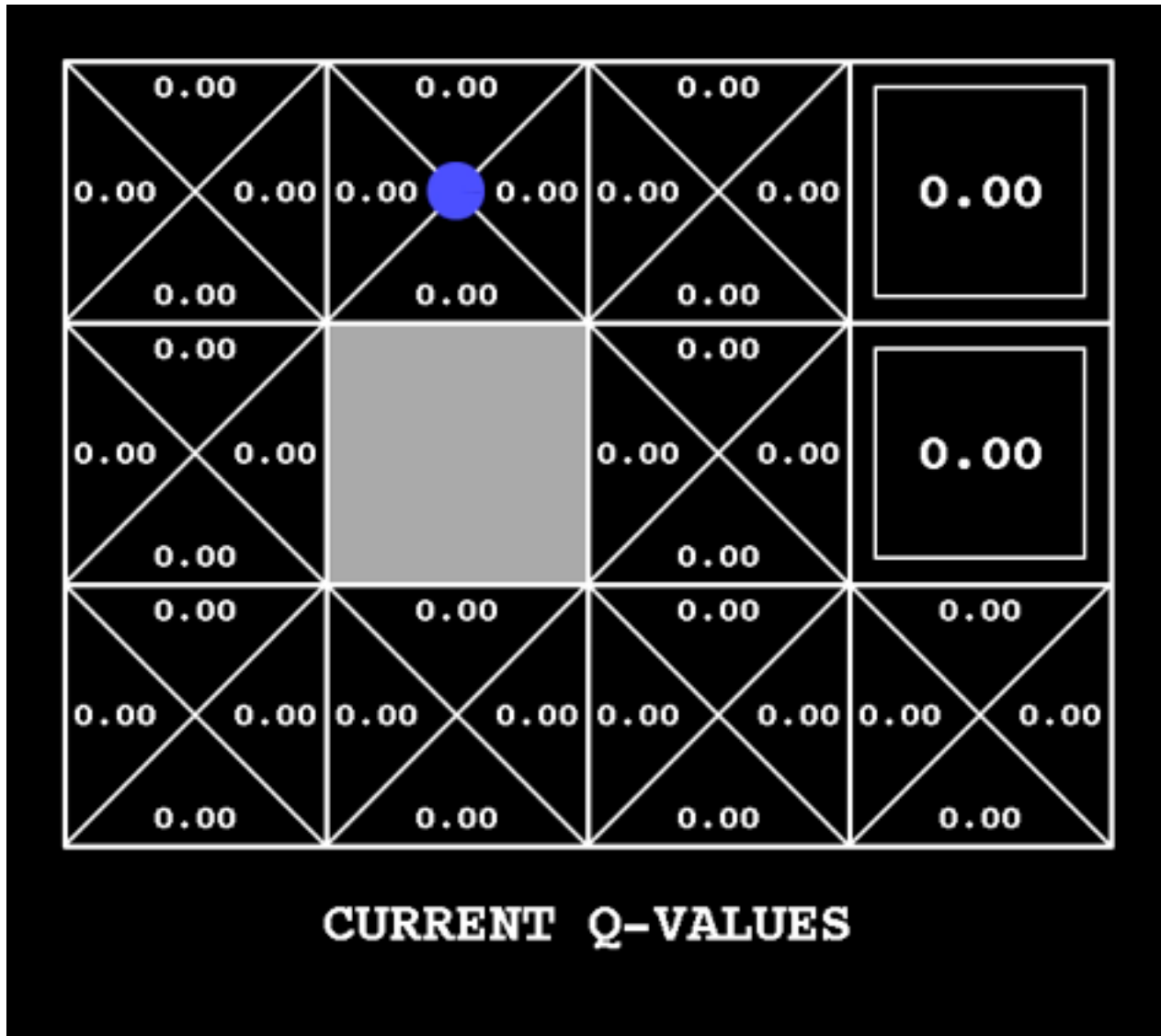
- Incorporate the new estimate into a running average:

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

Exploration / Exploitation

- Several schemes for action selection
 - Simplest: random actions (ϵ greedy)
 - Every time step, flip a coin
 - With probability ϵ , act randomly
 - With probability $1-\epsilon$, act according to current policy
 - Problems with random actions?
 - You do explore the space, but keep thrashing around once learning is done
 - One solution: lower ϵ over time
 - Another solution: exploration functions

Q-Learning: ϵ Greedy



Exploration Functions

- 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 $\pi(s') =$

$$\max_{a'} Q_i(s', a')$$

vs.

$$\max_{a'} f(Q_i(s', a'), N(s', a'))$$

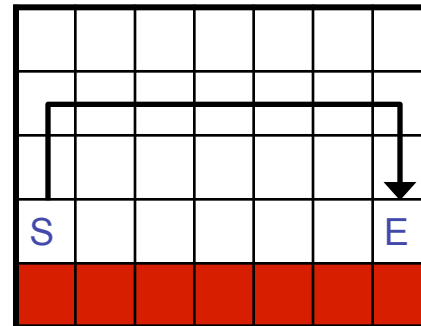
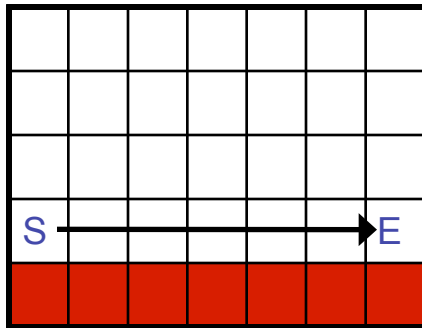
Q-Learning Final Solution

- Q-learning produces tables of q-values:



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



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!



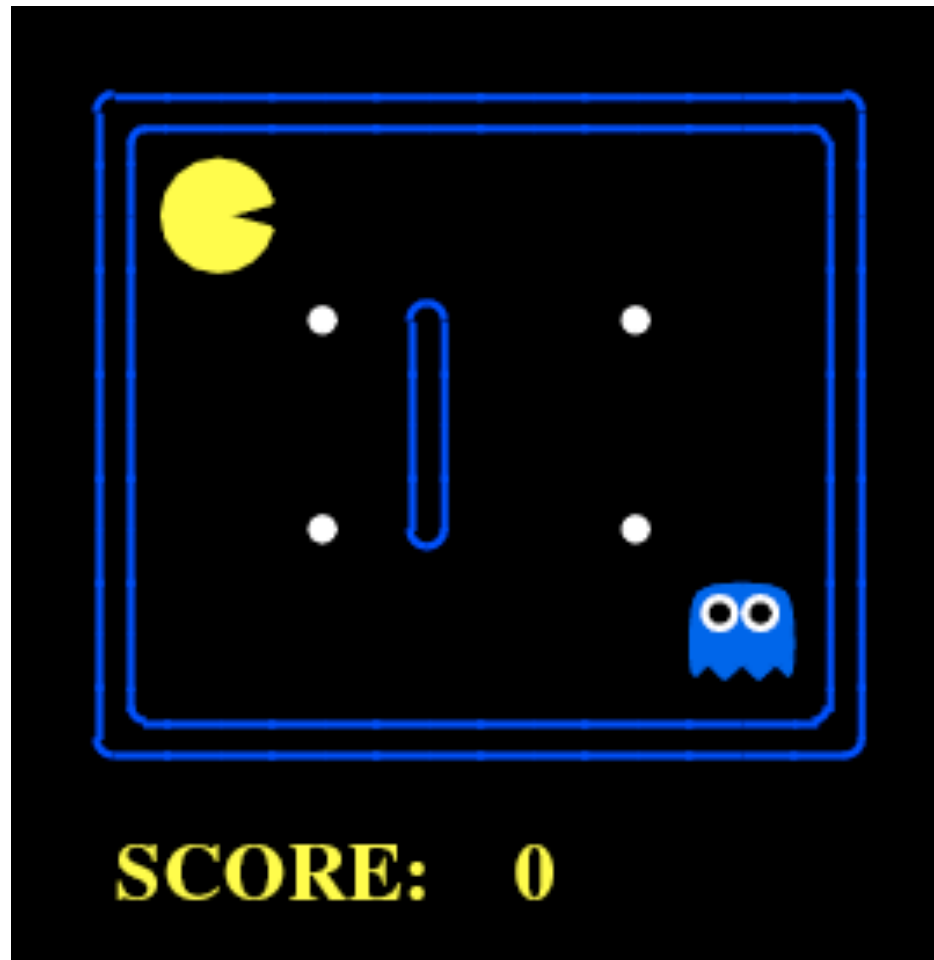
Feature-Based Representations

- 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
 - $1 / (\text{dist to dot})^2$
 - 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)



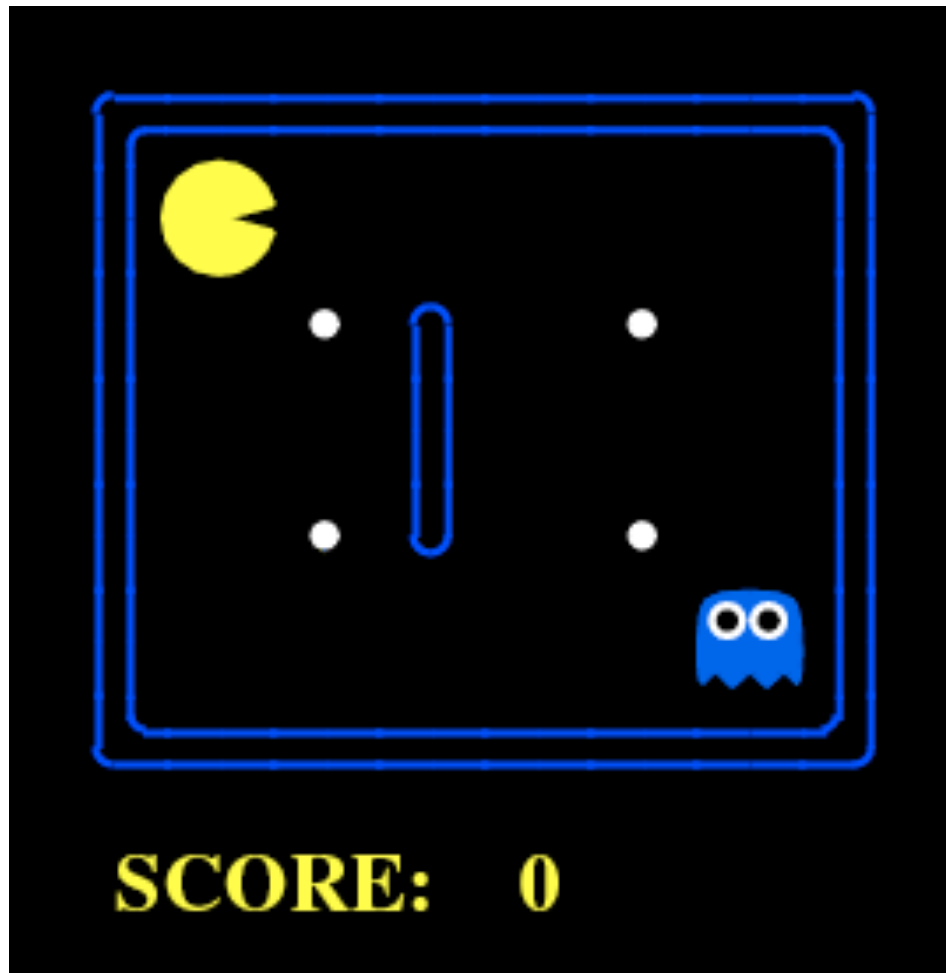
Which Algorithm?

Q-learning, no features, 50 learning trials:



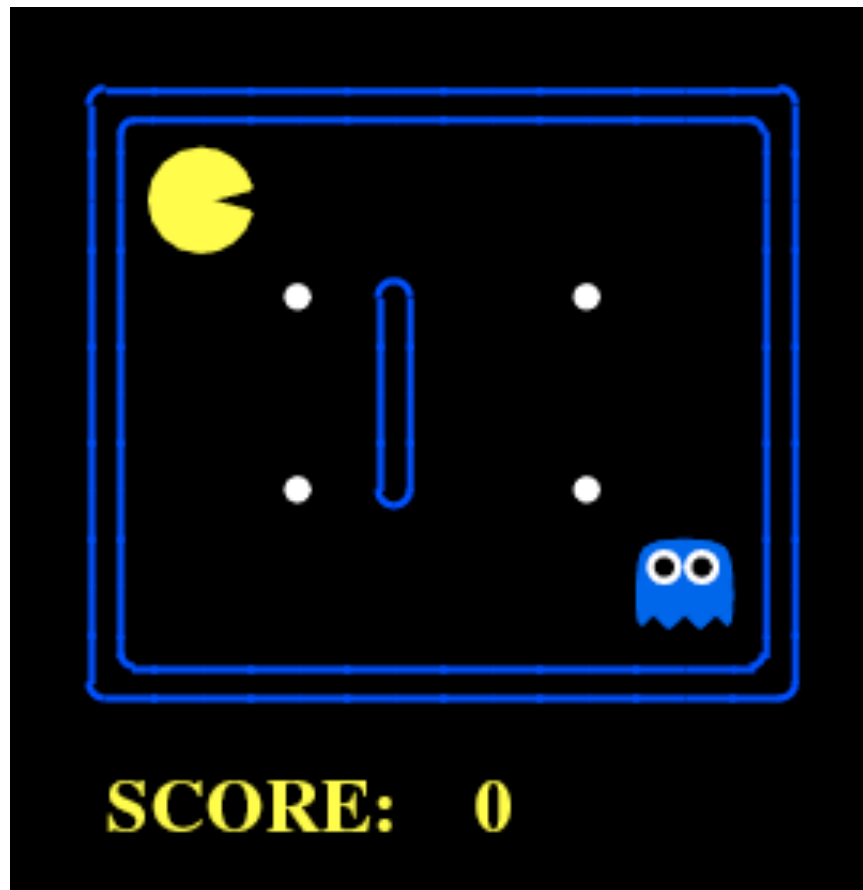
Which Algorithm?

Q-learning, no features, 1000 learning trials:



Which Algorithm?

Q-learning, simple features, 50 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:

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

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

- **Advantage:** our experience is summed up in a few powerful numbers
- **Disadvantage:** states may share features but actually be very different in value!

Function Approximation

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

- Q-learning with linear q-functions:

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

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

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

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

- 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

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

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

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

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

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

$$\text{correction} = -501$$

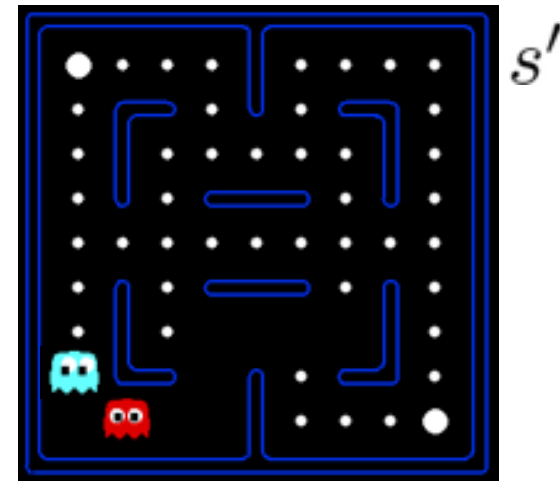
$$w_{DOT} \leftarrow 4.0 + \alpha [-501] 0.5$$

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

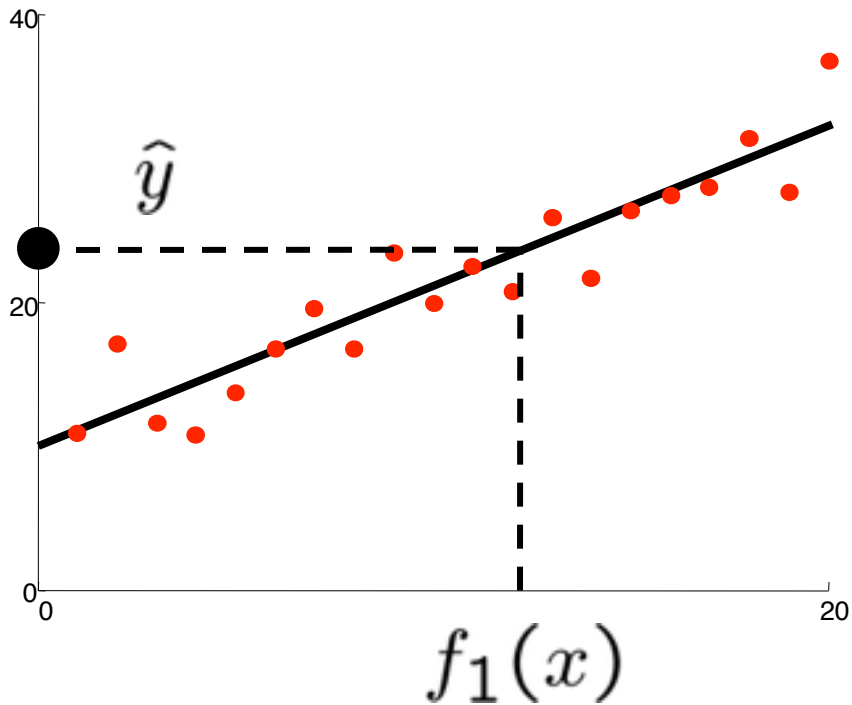
$$Q(s, a) = 3.0f_{DOT}(s, a) - 3.0f_{GST}(s, a)$$



$a = \text{NORTH}$
 $r = -500$

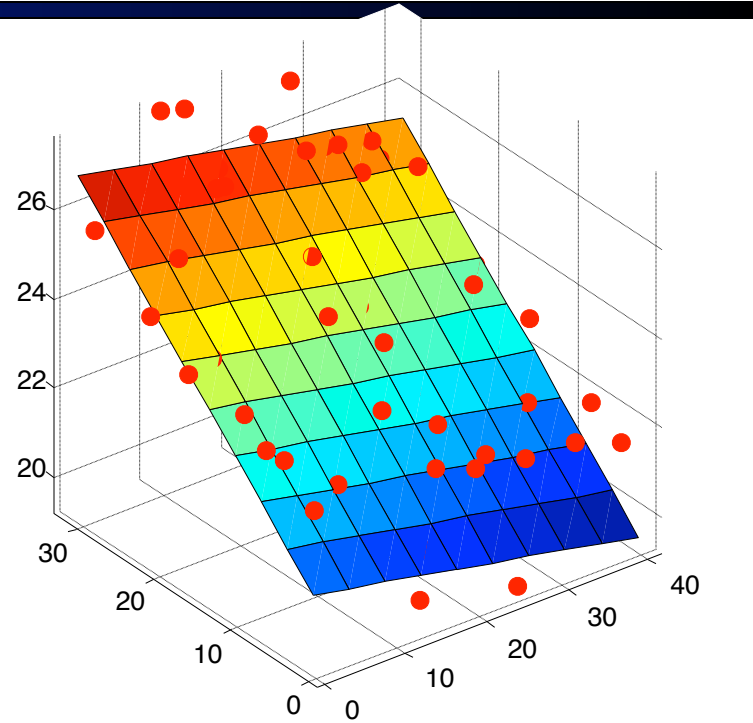


Linear Regression



Prediction

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

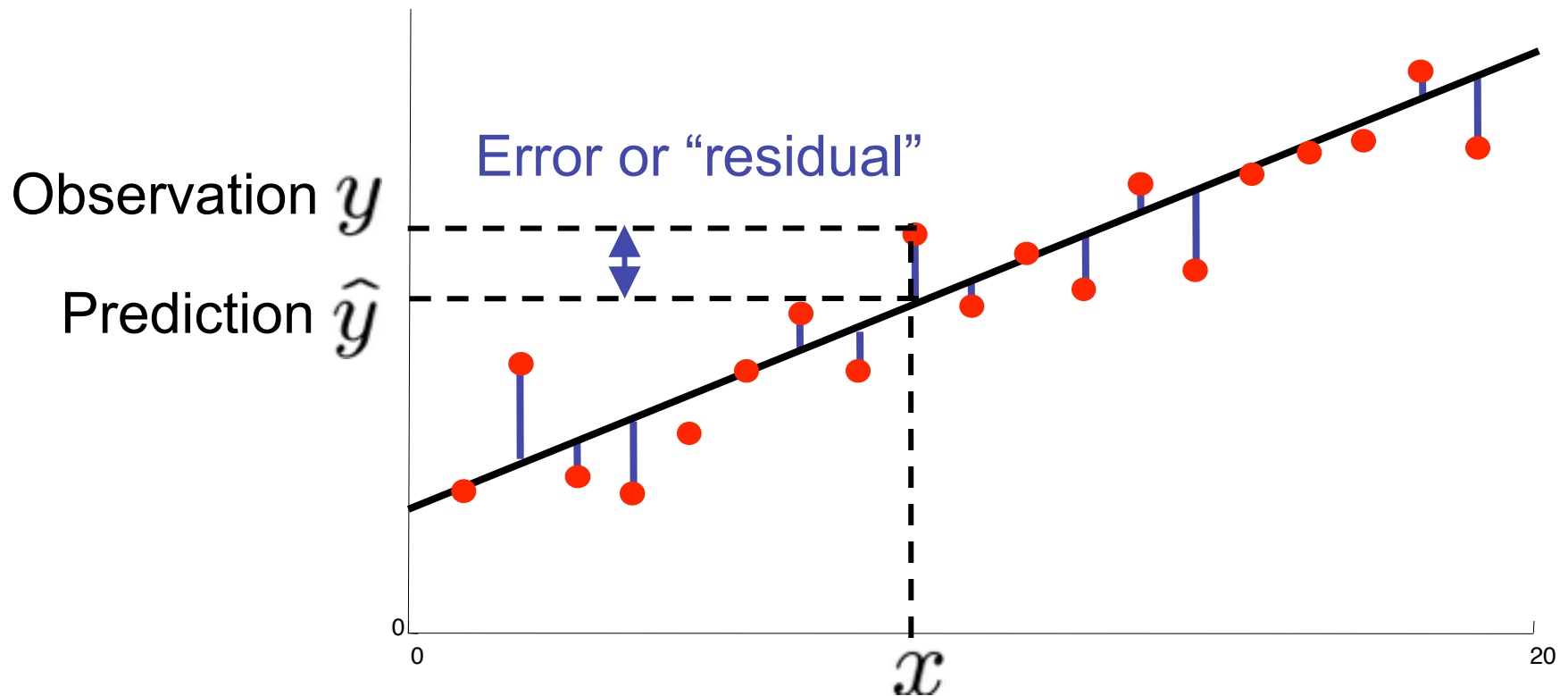


Prediction

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

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 $f(x)$:

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

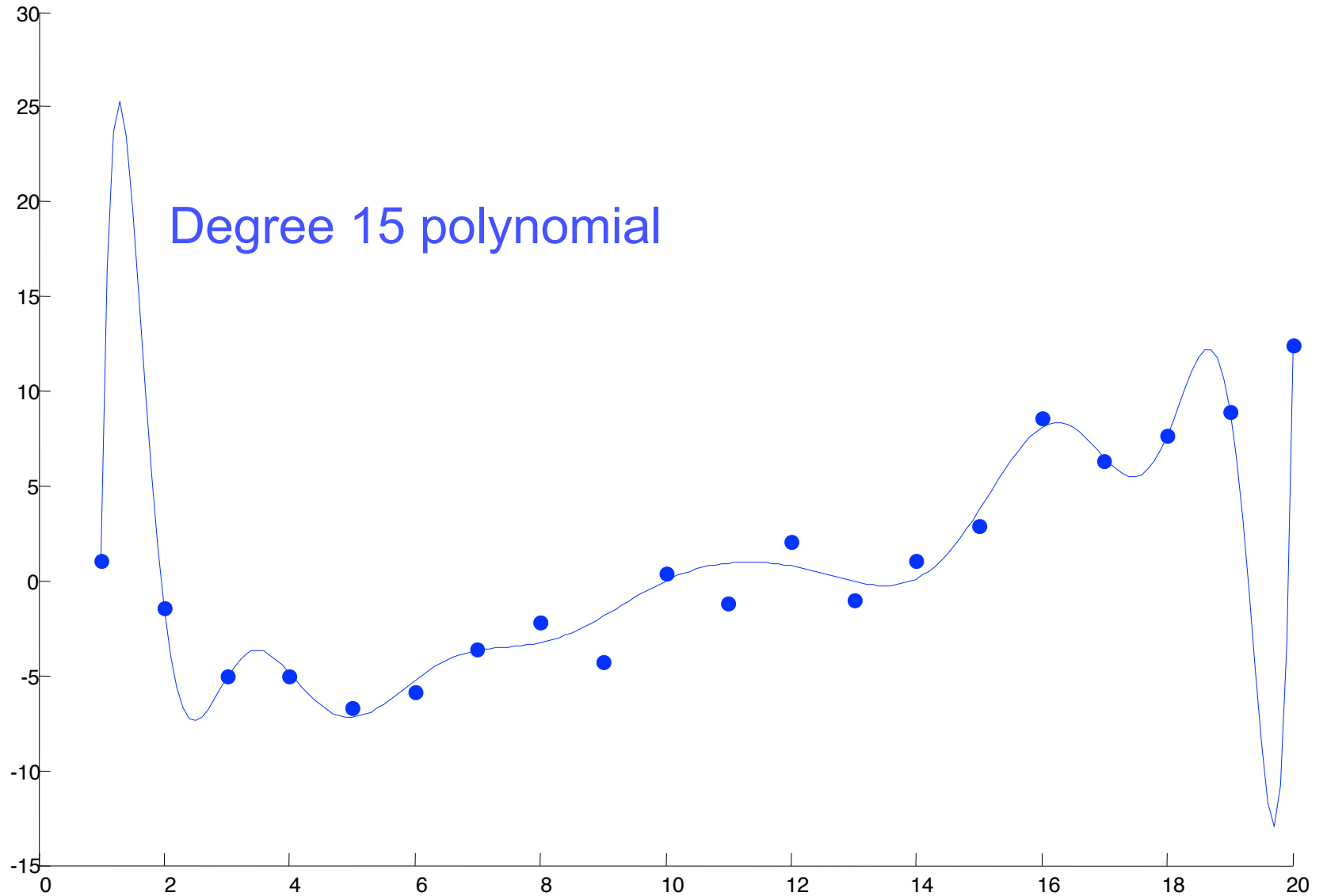
$$\frac{\partial \text{error}(w)}{\partial w_m} = - \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$

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

Approximate q update:

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

Overfitting



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.

Policy Search*



MDP 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

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

VI/PI on approx. MDP

PE on approx. MDP

Unknown MDP: Model-Free

Goal

Compute V^*, Q^*, π^*

Evaluate a fixed policy π

Technique

Q-learning

Value Learning