Reinforcement Learning II: Q-learning

Hal Daumé III

Computer Science
University of Maryland

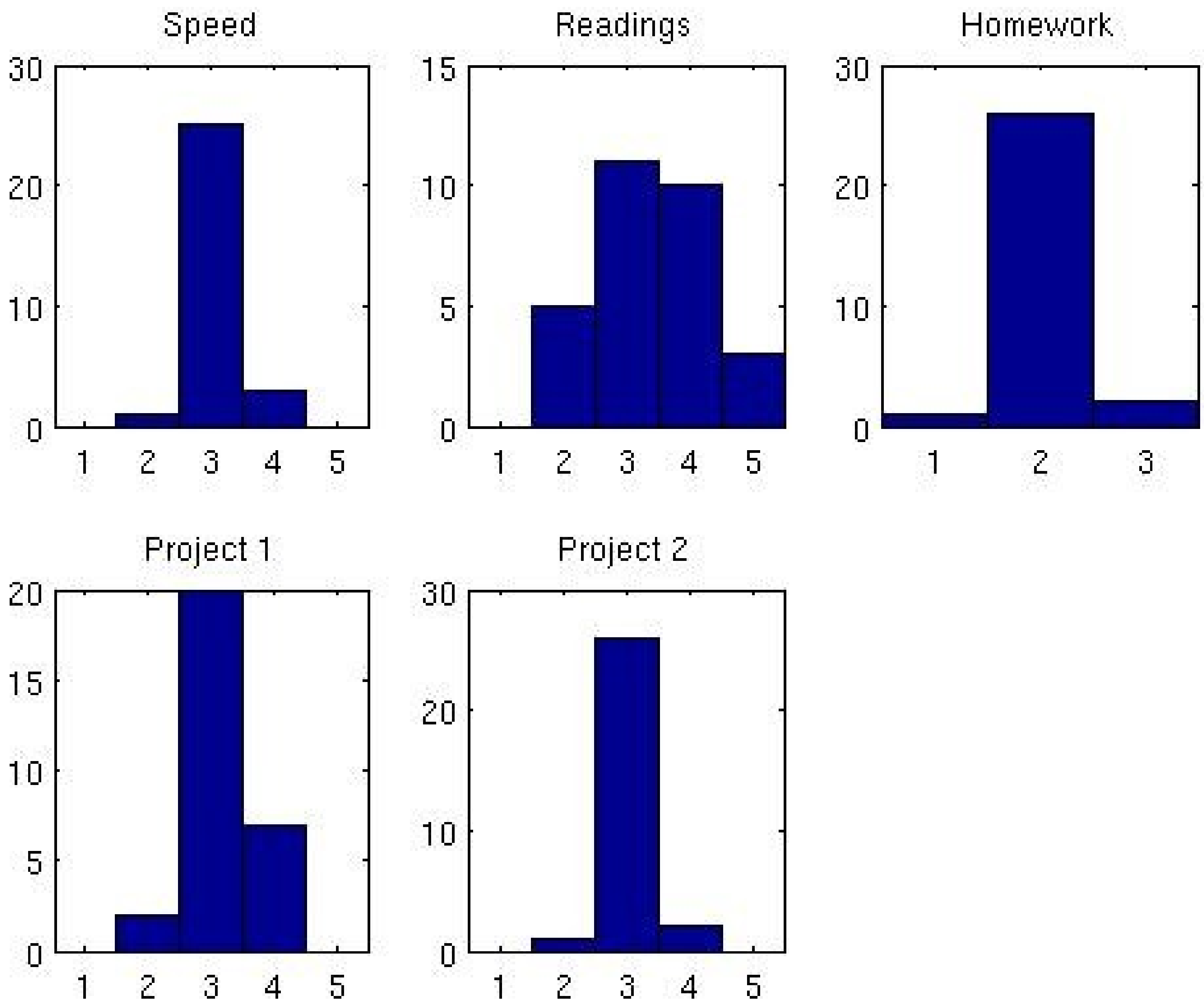
me@hal3.name

CS 421: Introduction to Artificial Intelligence

28 Feb 2012

Many slides courtesy of
Dan Klein, Stuart Russell,
or Andrew Moore

Midcourse survey, quantitative



Midcourse survey, qualitative

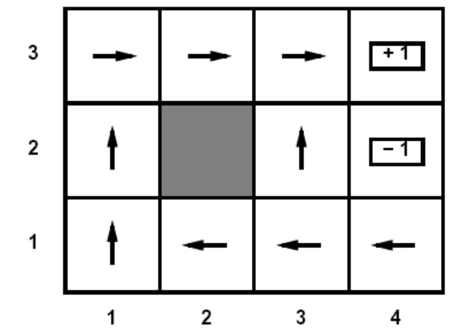
- (3) Too much class time on minutiae of homeworks
- (2) Project 1 not discussed much: made heuristics hard
- (2) More motivating examples (products or research)
- (2) Practice problems for exams, more HW examples
- (2) Reduce overall number of topics, or point toward important ones
- (2) Handin link should be at the top of the web page
- (1) Textbook too wordy with too few visuals
- (1) Talk about (dis)advantages of approaches in class
- (1) More time going over algos in class
- (1) Make sure exam stuff is on slides
- (1) Tweak homeworks toward readings

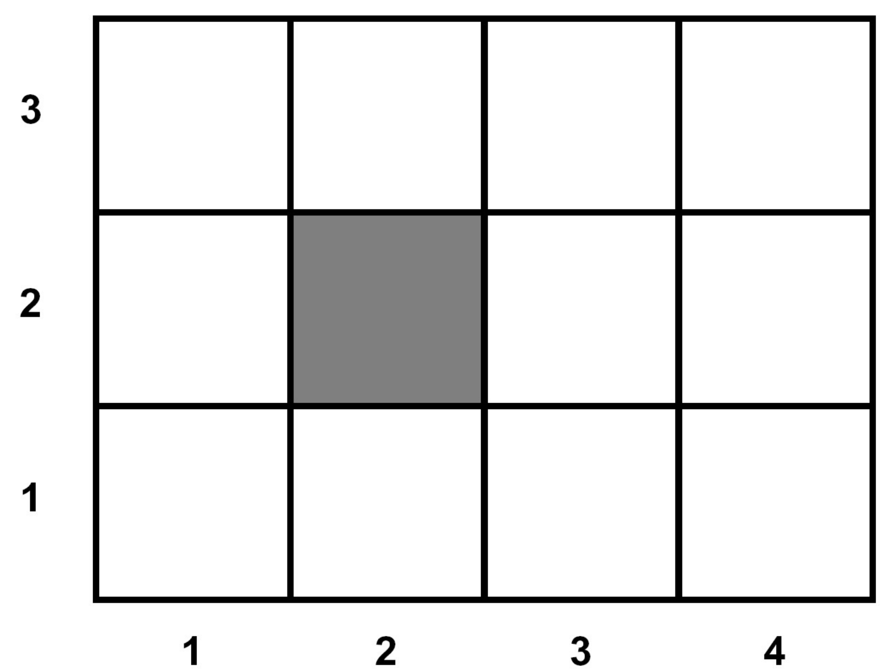
Example: TD Policy Evaluation

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

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

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

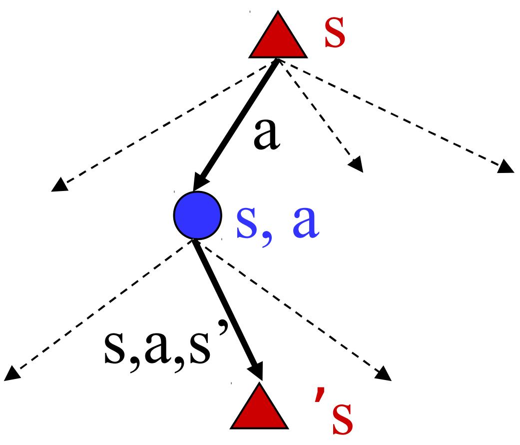




Take $\gamma = 1$, $\alpha = 0.5$

Problems with TD Value Learning

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

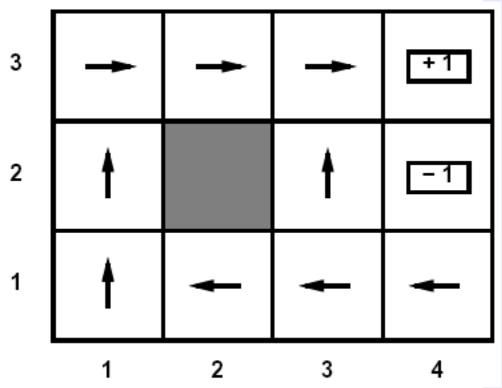


$$\pi(s) = \arg\max_a Q^*(s,a)$$

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

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

Active Learning



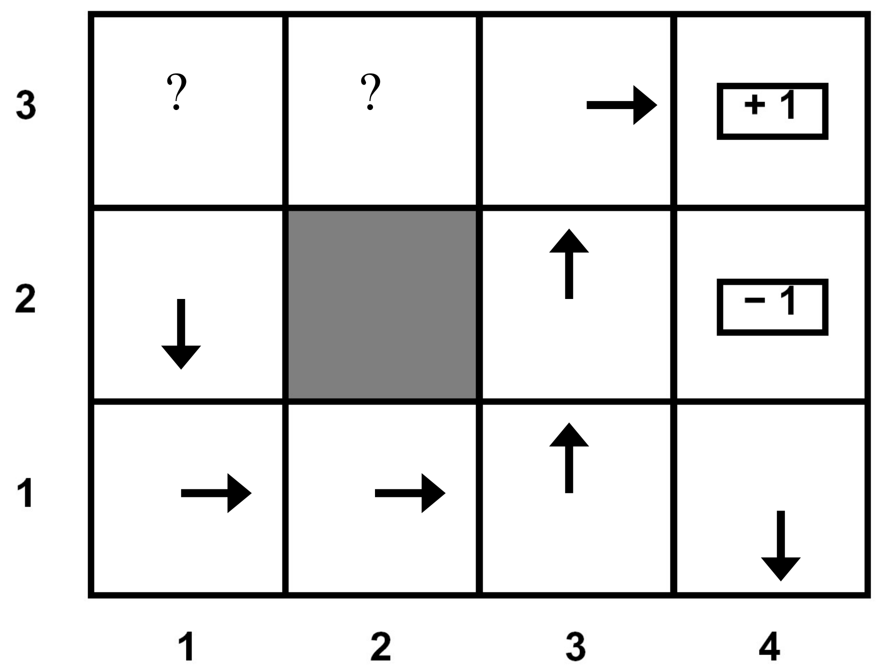
- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You can choose any actions you like
  - Goal: learn the optimal policy (maybe values)
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning!

Model-Based Learning

- In general, want to learn the optimal policy, not evaluate a fixed policy
- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model

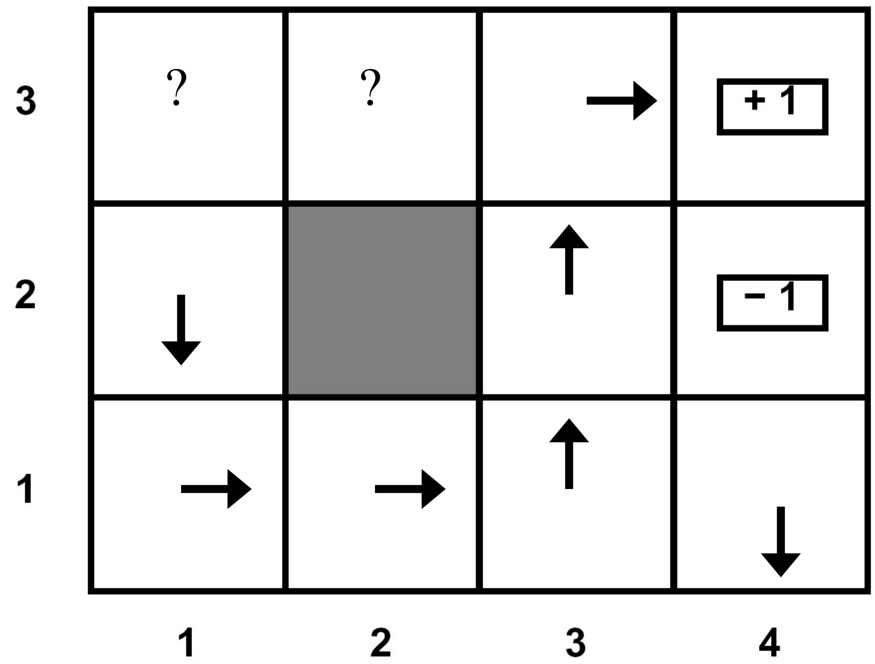
Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We'll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy



What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit



Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

- But Q-values are more useful!
  - Start with $Q_0^*(s,a) = 0$, which we know is right (why?)
  - Given Q_i^*, calculate the q-values for all q-states for depth i+1:

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

Q-Learning

[DEMO – Grid Q's]

- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

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

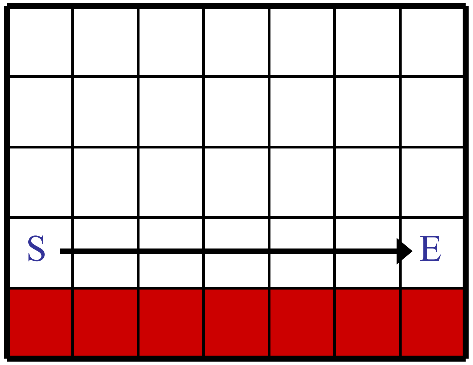
  - Incorporate the new estimate into a running average:

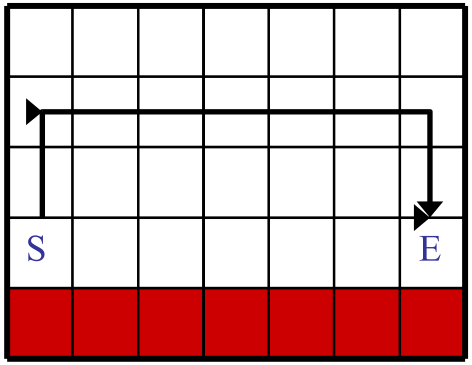
$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[sample \right]$$

Q-Learning Properties

[DEMO – Grid Q's]

- Will converge to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - But not decrease it too quickly!
  - Basically doesn't matter how you select actions (!)
- Neat property: learns optimal q-values regardless of action selection noise (some caveats)





Exploration / Exploitation

[DEMO – RL Pacman]

- Several schemes for forcing exploration
  - Simplest: random actions (ε greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, 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

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)

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a'))$$

Q-Learning

[DEMO – Crawler Q's]

- Q-learning produces tables of q-values:

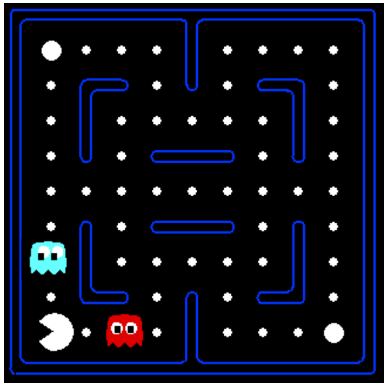
Q-Learning

- 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

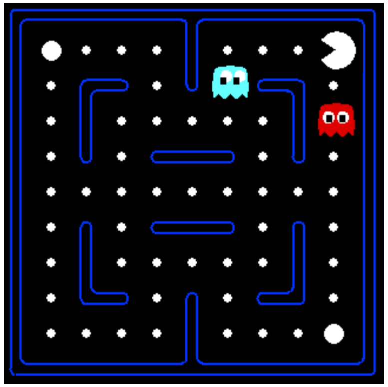
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Example: Pacman

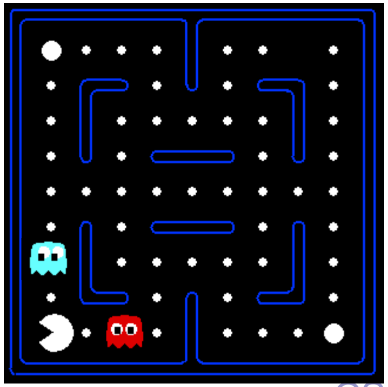
- Let’s say we discover through experience that this state is bad:



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

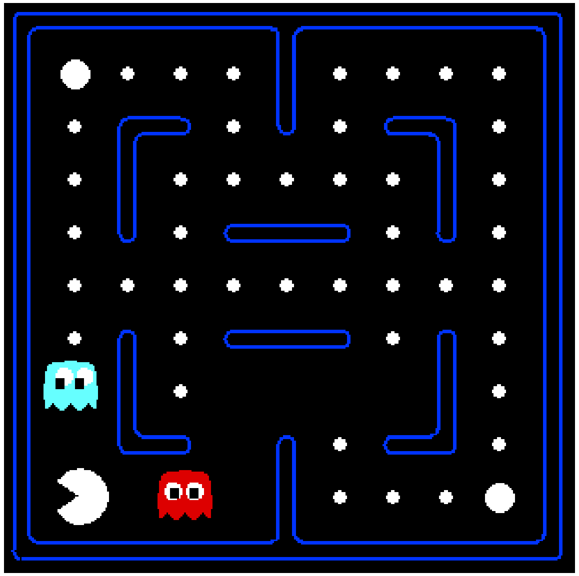


- Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features
  - 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.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



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 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:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[error]$$

$$w_i \leftarrow w_i + \alpha\,[error]\,f_i(s,a)$$

- 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

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

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

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

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

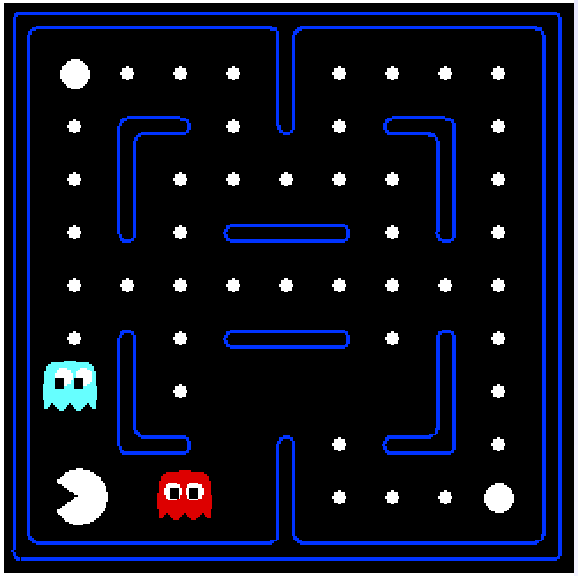
$$R(s,a,s') = -500$$

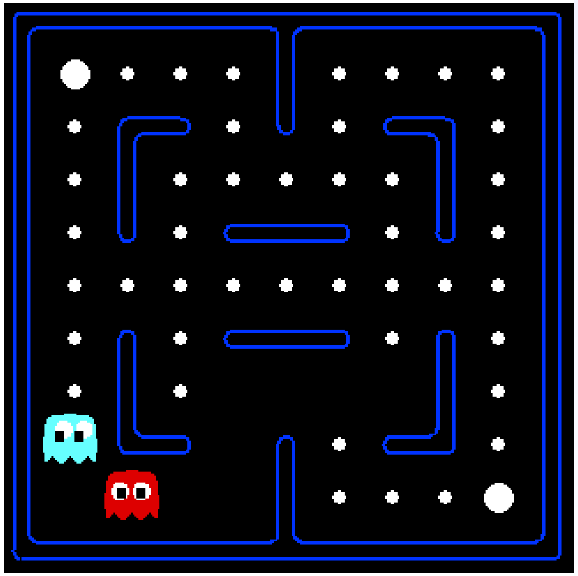
$$error = -501$$

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

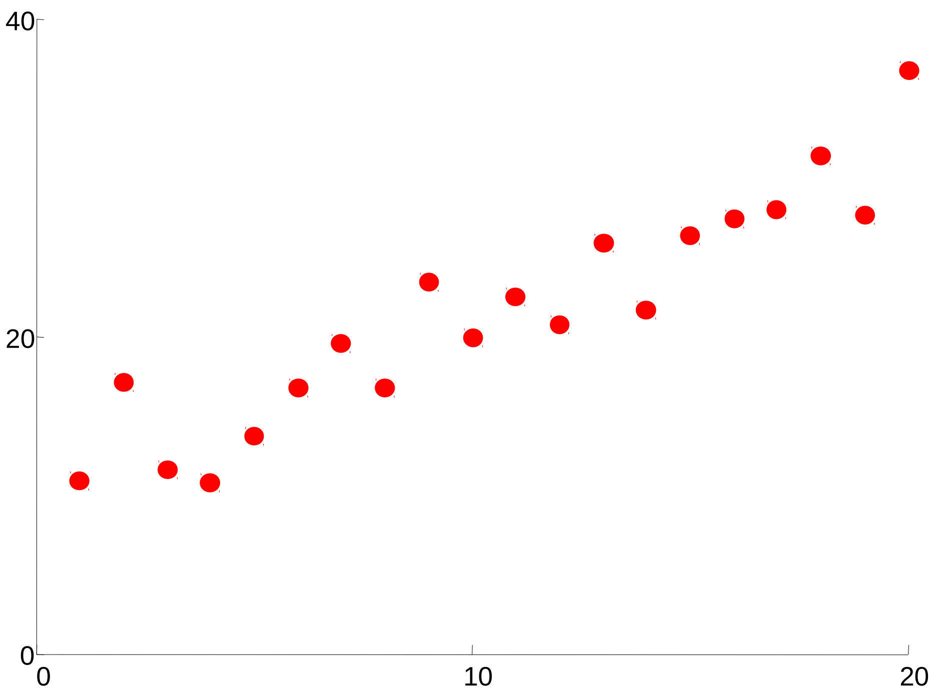
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

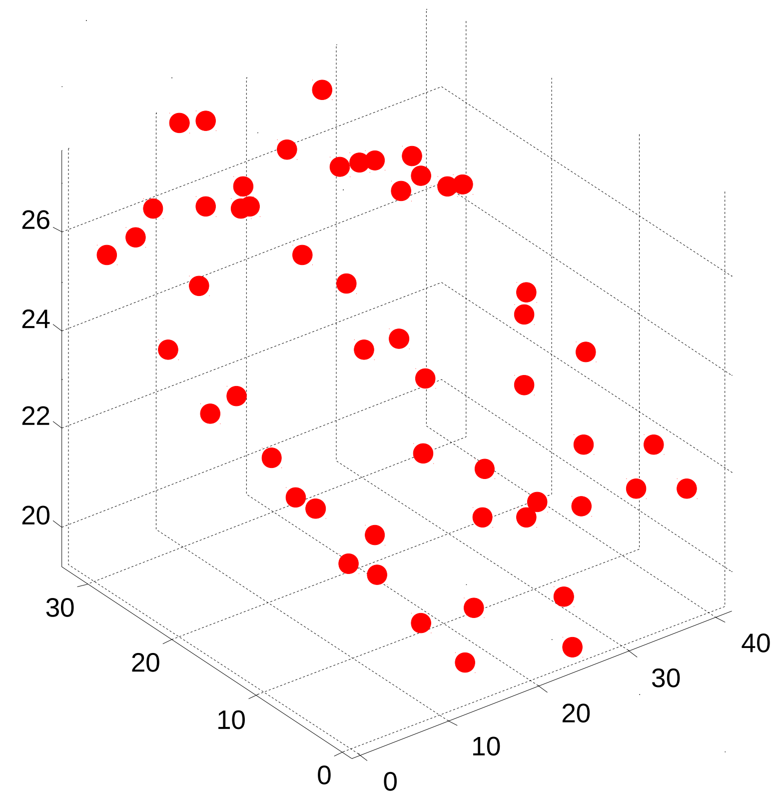
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$





Linear regression

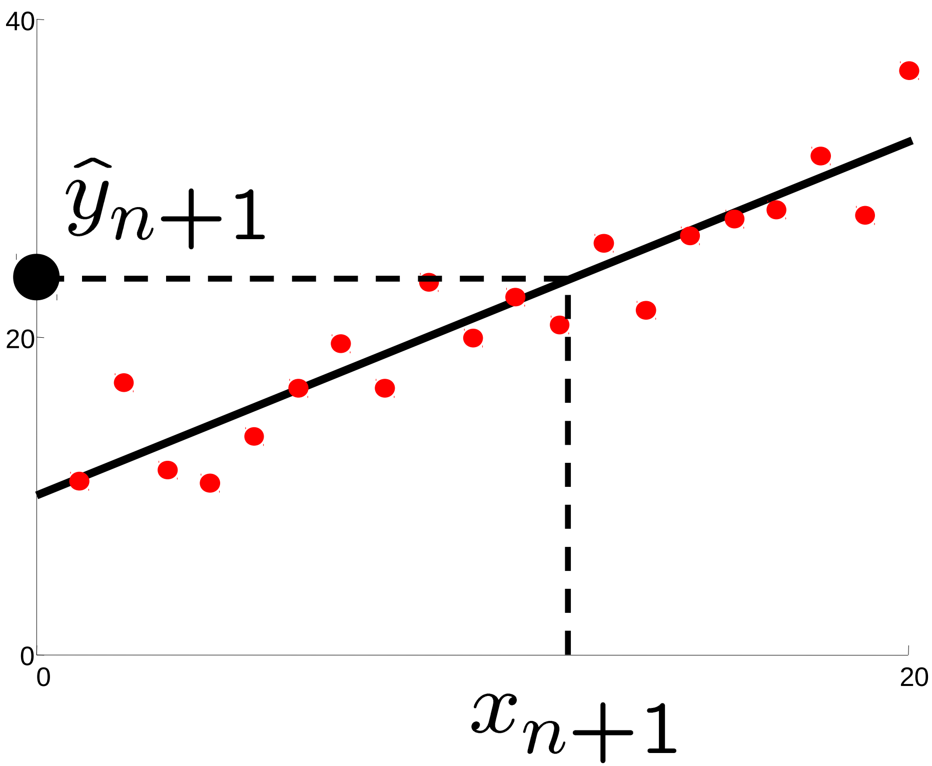


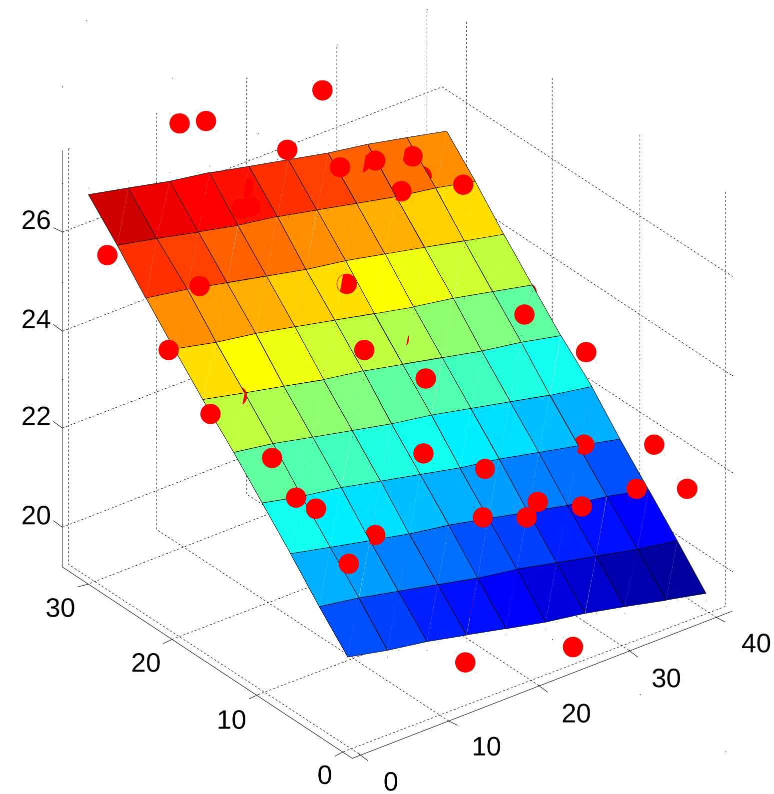


Given examples $(x_i, y_i)_{i=1\ldots n}$

Predict y_{n+1} given a new point x_{n+1}

Linear regression





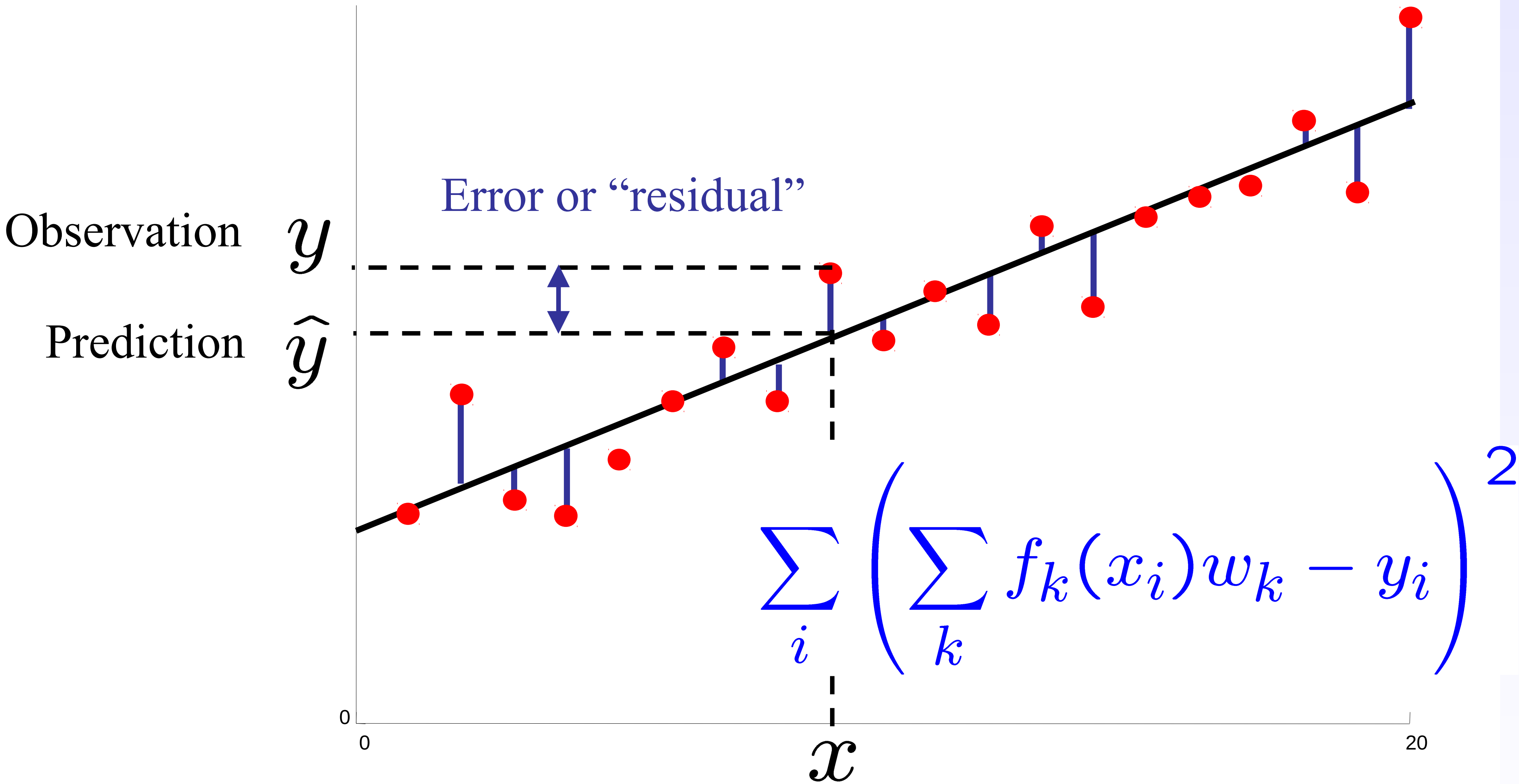
Prediction

$$\widehat{y}_i = w_0 + w_1 x_i$$

Prediction

$$\widehat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2}$$

Ordinary Least Squares (OLS)



Minimizing Error

$$E(w) = \frac{1}{2}\sum_i \left(\sum_k f_k(x_i)w_k - y_i\right)^2$$

$$\frac{\partial E}{\partial w_m} = \sum_i \left(\sum_k f_k(x_i)w_k - y_i\right) f_m(x_i)$$

$$E \leftarrow E + \alpha \sum_i \left(\sum_k f_k(x_i)w_k - y_i\right) f_m(x_i)$$

Value update explained:

$$w_i \leftarrow w_i + \alpha\,[error]\,f_i(s,a)$$

Overfitting