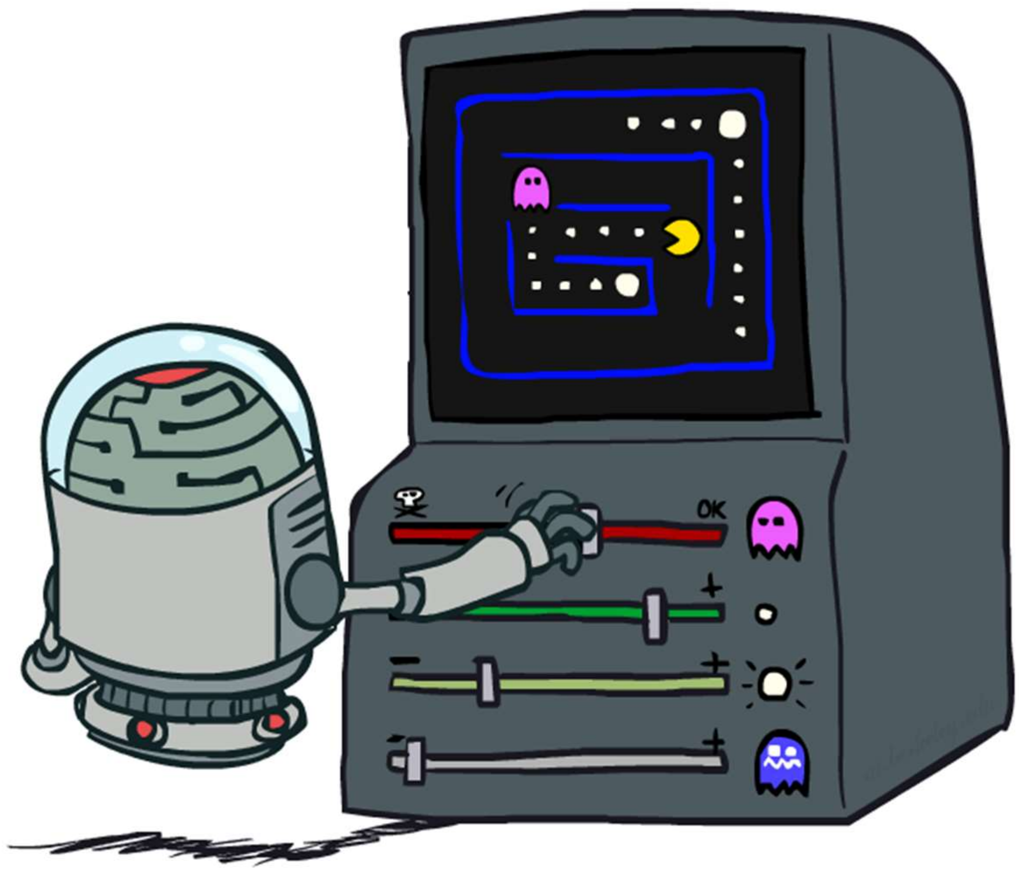
Reinforcement Learning II



Steve Tanimoto

slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkele

Reinforcement Learning

/e still assume 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')

till looking for a policy π(s)

ew twist: don't know T or R, so must try out actions

ig idea: Compute all averages over T using sample outcomes

The Story So Far: MDPs 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

al	Technique
npute V*, Q*, π*	VI/PI on approx. MDP
uate 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

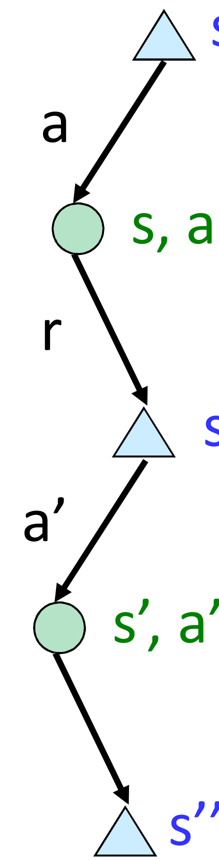
Model-Free Learning

Model-free (temporal difference) learning

- Experience world through episodes

$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

- Update estimates each transition (s, a, r, s')
- Over time, updates will mimic Bellman updates



Q-Learning

- We'd like to do Q-value updates to each Q-state:

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

  - But can't compute this update without knowing T, R

- Instead, compute average as we go
  - Receive a sample transition (s,a,r,s')
  - This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

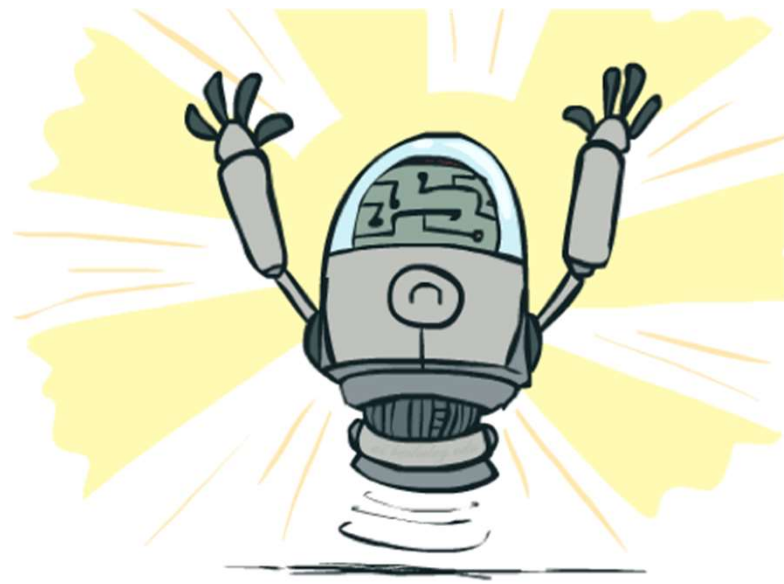
  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a') \right]$$

Q-Learning Properties

mazing result: Q-learning converges to optimal policy -- even
f you're acting suboptimally!

his is called off-policy learning



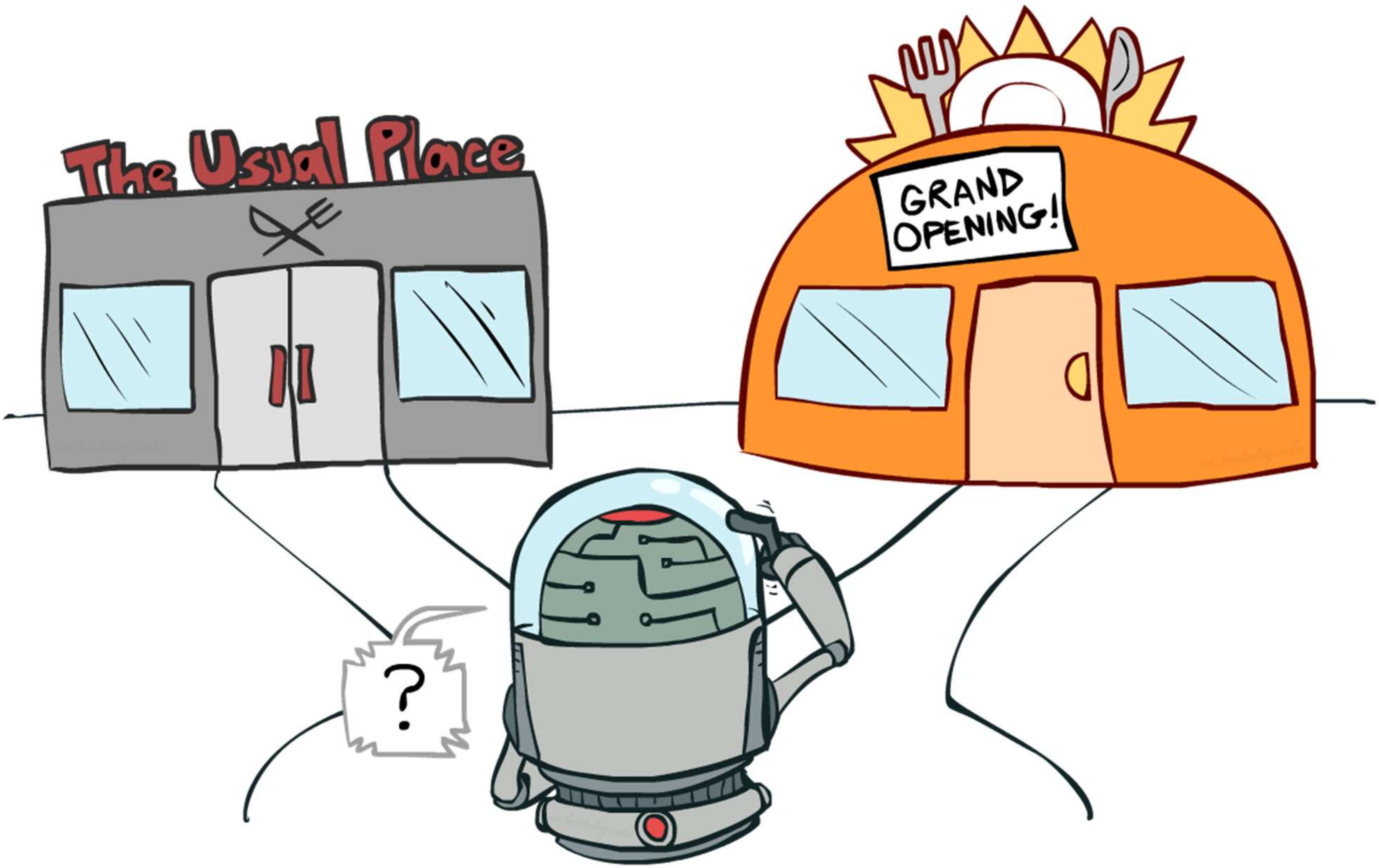
aveats:

- You have to explore enough
- You have to eventually make the learning rate small enough
- … but not decrease it too quickly
- Basically, in the limit, it doesn't matter how you select actions (!)

[Demo: Q-learning – auto – cliff gri

Video of Demo Q-Learning Auto Cliff Grid

Exploration vs. Exploitation



How to Explore?

- everal schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

[Demo: Q-learning – manual exploration – bridge gri
[Demo: Q-learning – epsilon-greedy -- crawle

eo of Demo Q-learning – Manual Exploration – Bridge G

Video of Demo Q-learning – Epsilon-Greedy – Crawler

Exploration Functions

hen to explore?

Random actions: explore a fixed amount

Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

ploration function

Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

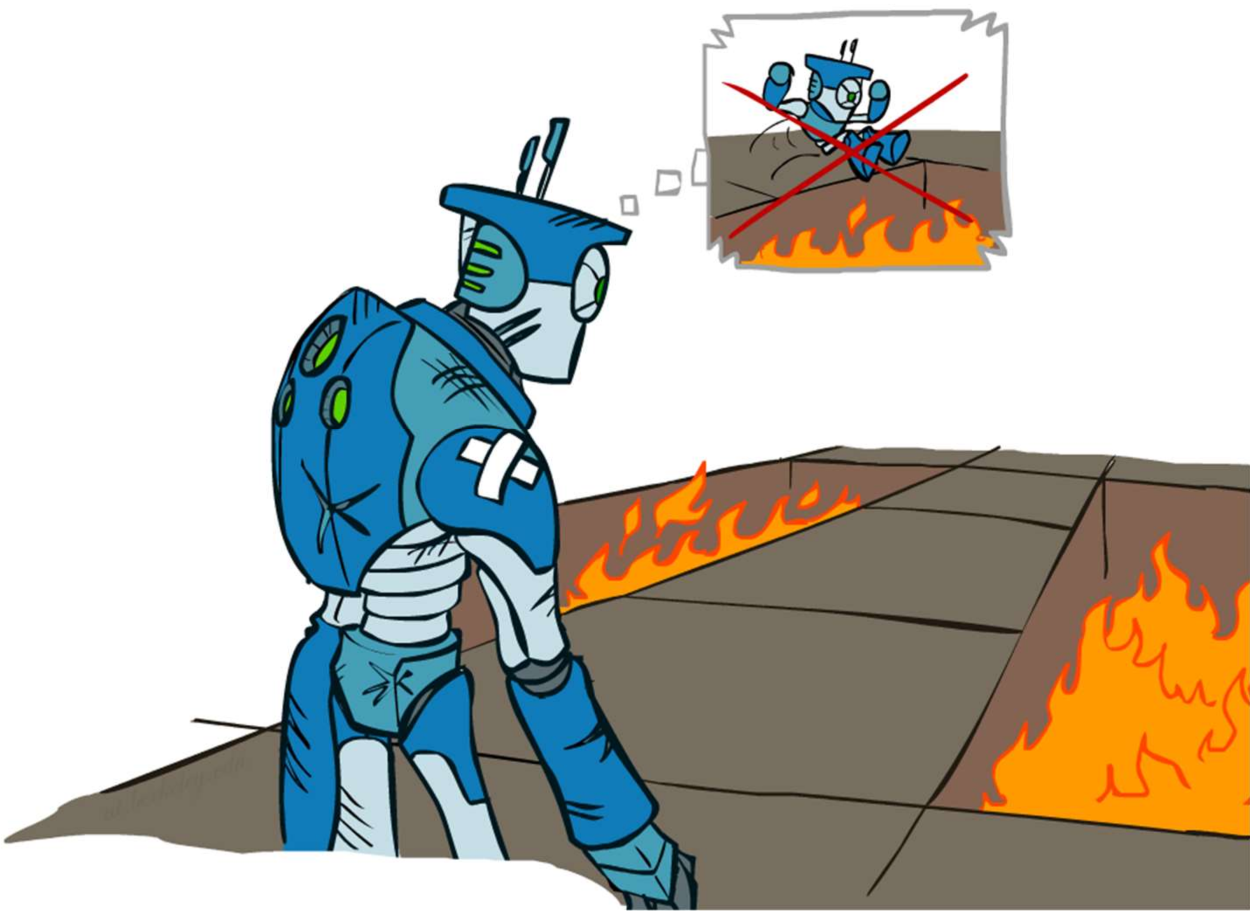
Note: this propagates the "bonus" back to states that lead to unknown states as

[Demo: exploration – Q-learning – crawler – exploration functio

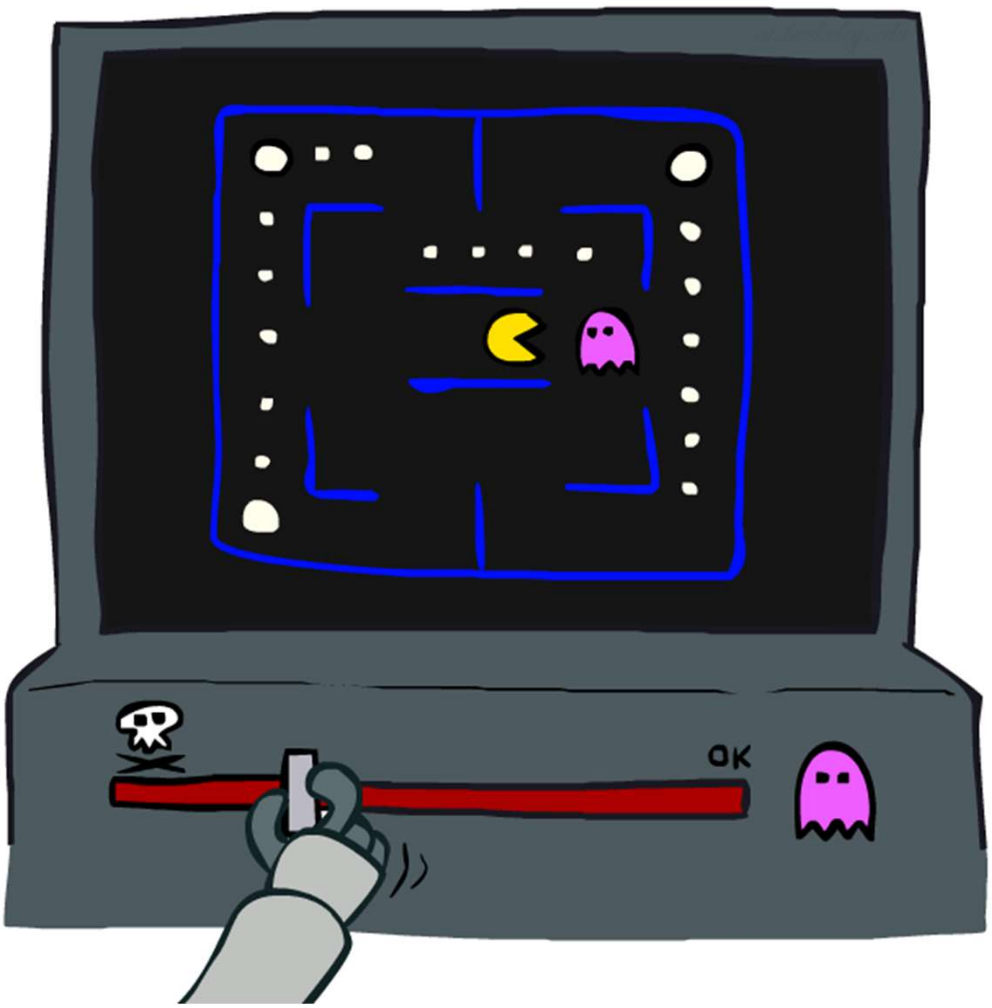
deo of Demo Q-learning – Exploration Function – Craw

Regret

- ven if you learn the optimal policy, ou still make mistakes along the way
- egret is a measure of your total istake cost: the difference between our (expected) rewards, including outhful suboptimality, and optimal xpected) rewards
- inimizing regret goes beyond arning to be optimal – it requires ptimally learning to be optimal
- xample: random exploration and xploration functions both end up ptimal, but random exploration has gher regret

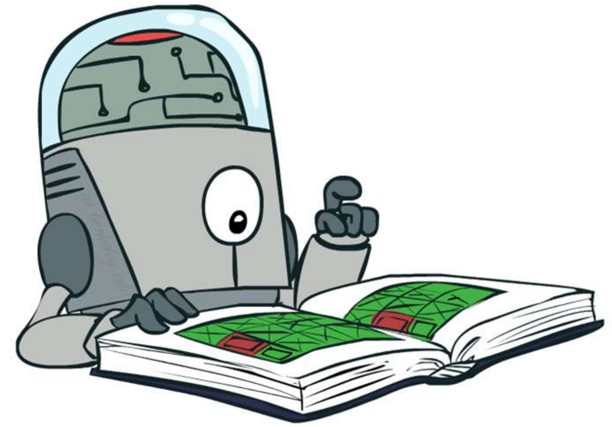


Approximate Q-Learning



Generalizing Across States

asic Q-Learning keeps a table of all q-values



realistic situations, we cannot possibly learn
oout every single state!

Too many states to visit them all in training

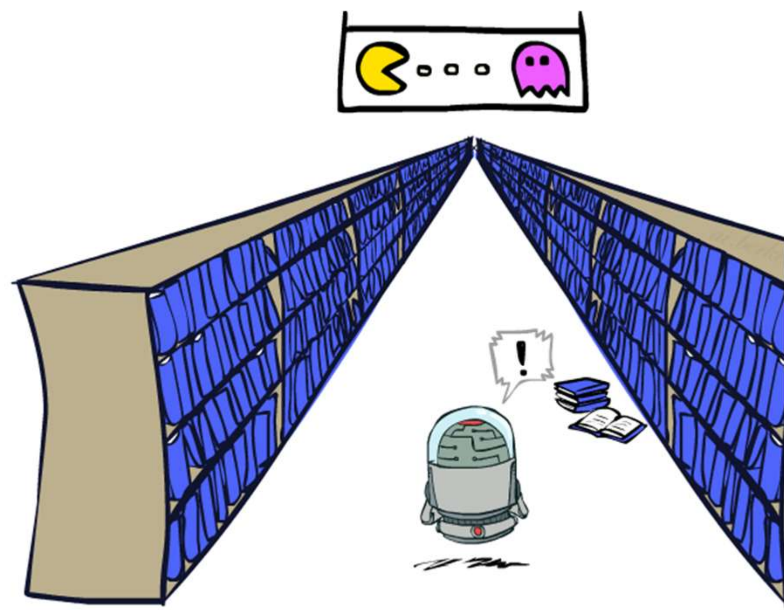
Too many states to hold the q-tables in memory

stead, we want to generalize:

Learn about some small number of training states from experience

Generalize that experience to new, similar situations

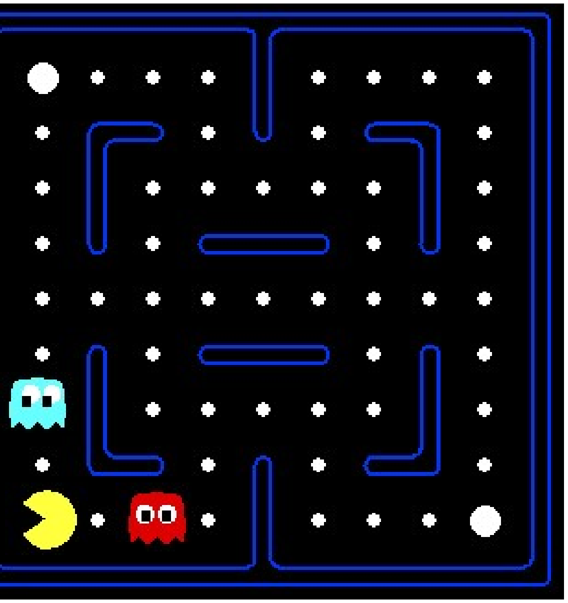
This is a fundamental idea in machine learning, and we'll see it over and over again



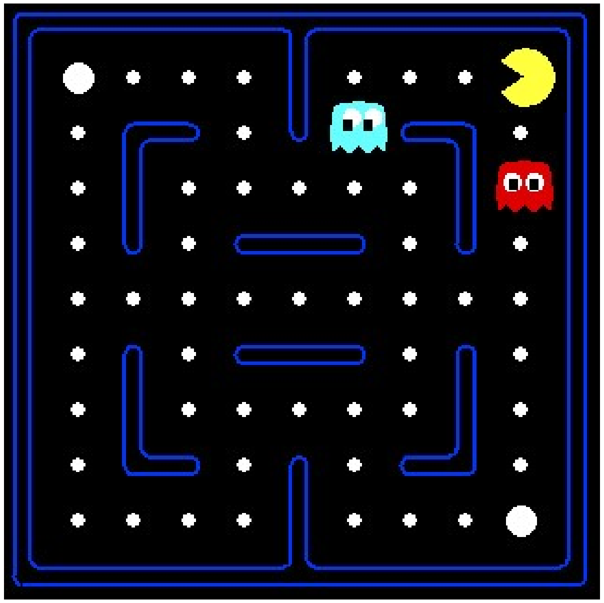
[demo –

Example: Pacman

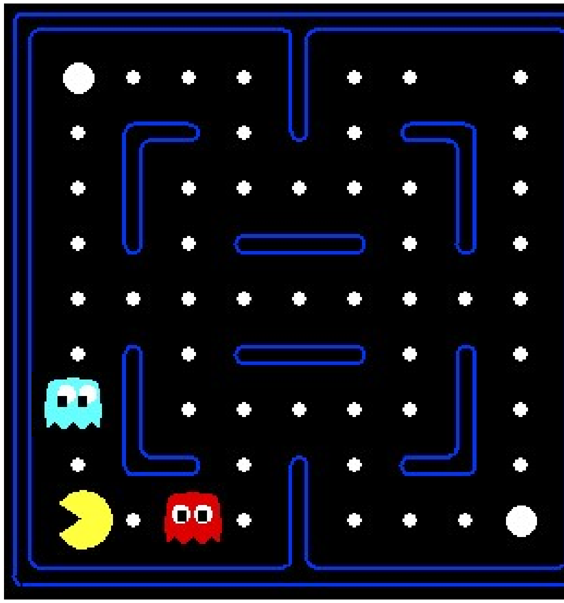
Let's say we discover through experience that this state is bad:



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



Or even this one!



[Demo: Q-learning – pacman – tiny – watch a

[Demo: Q-learning – pacman – tiny – silent trai

[Demo: Q-learning – pacman – tricky – watch a

deo of Demo Q-Learning Pacman – Tiny – Watch A

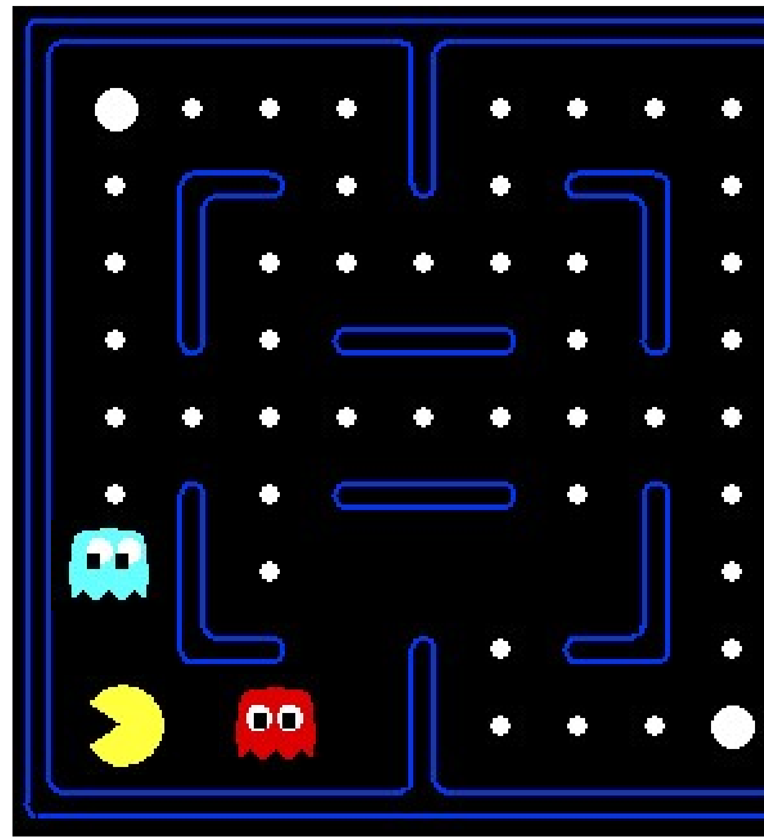
eo of Demo Q-Learning Pacman – Tiny – Silent Tr

eo of Demo Q-Learning Pacman – Tricky – Watch

Feature-Based Representations

olution: describe a state using a vector of eatures (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)



Linear Value Functions

sing a feature representation, we can write a q function (or value function) for an
tate using a few weights:

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

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

dvantage: our experience is summed up in a few powerful numbers

isadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

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

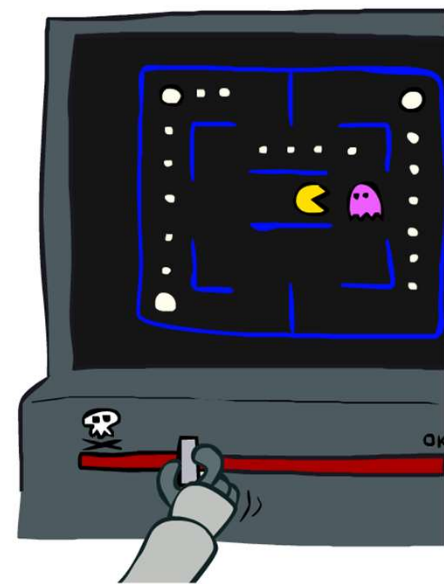
-learning with linear Q-functions:

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

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

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

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



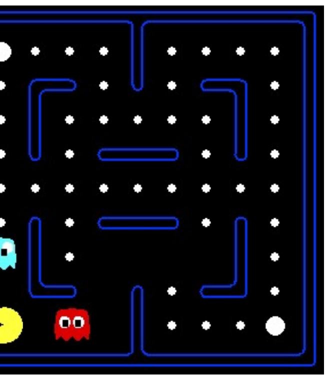
ntuitive interpretation:

- 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

ormal justification: online least squares

Example: Q-Pacman

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

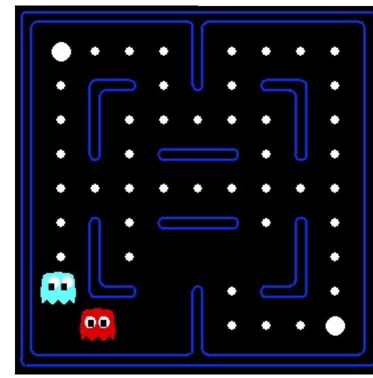


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



$Q(s, \text{NORTH}) = +1$

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

$Q(s', \cdot) = 0$

difference $= -501$

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

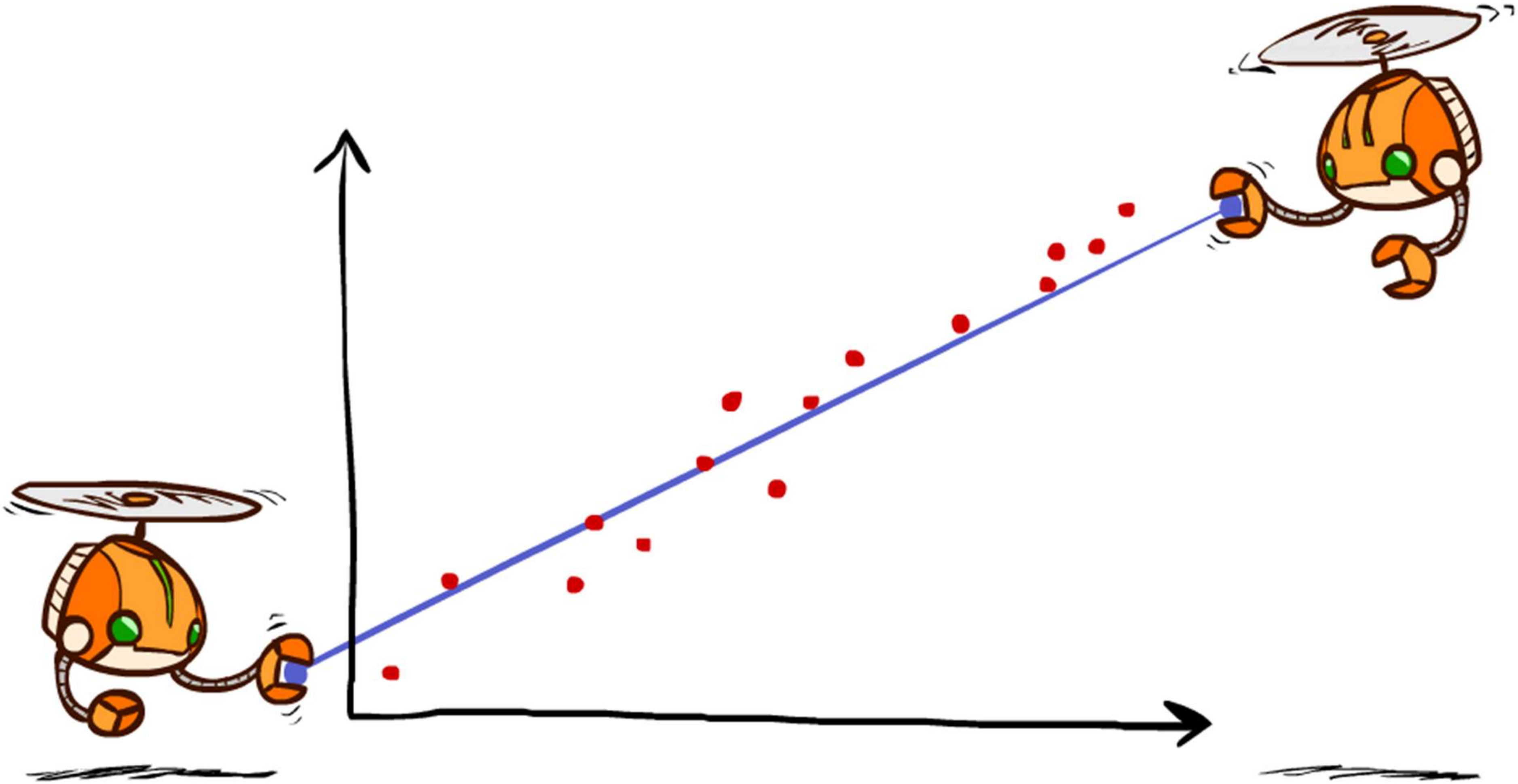
$w_{GST} \leftarrow -1.0 + \alpha \, [-501] \, 1.0$

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

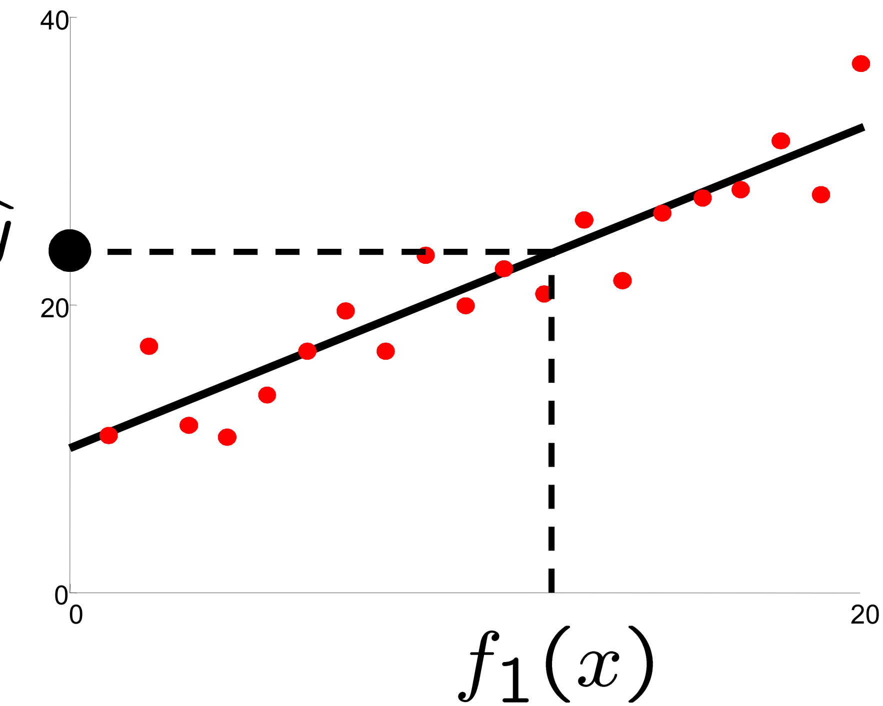
[Demo: app
learning pacma

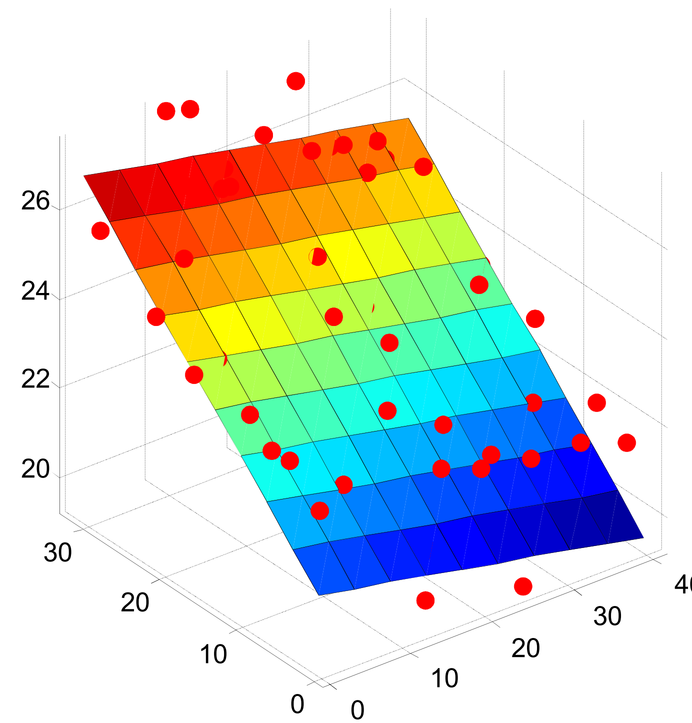
eo of Demo Approximate Q-Learning -- Pacm

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

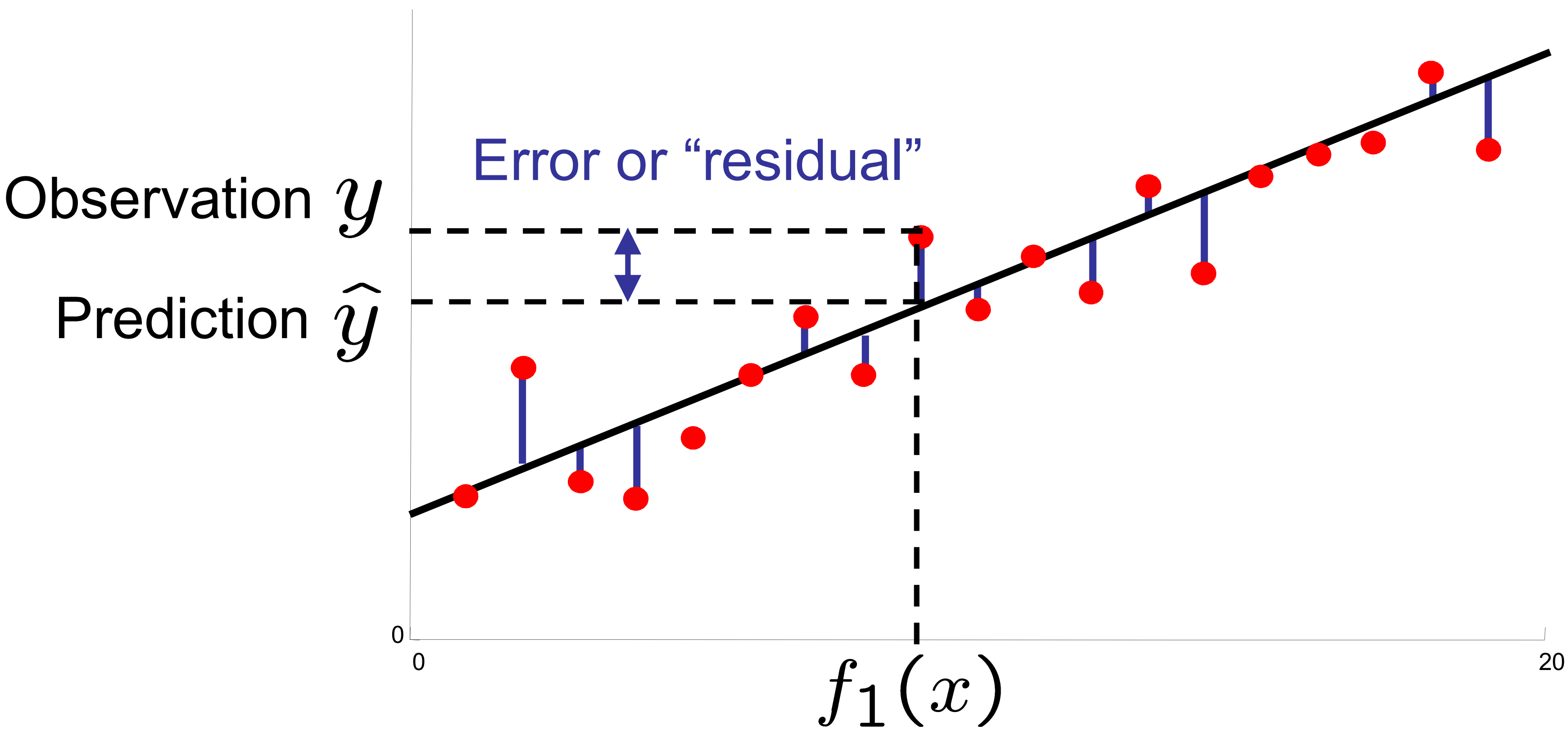
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares*

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



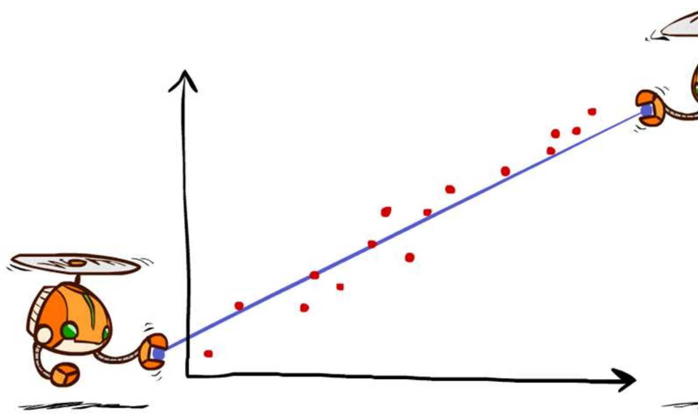
Minimizing Error*

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

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

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

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

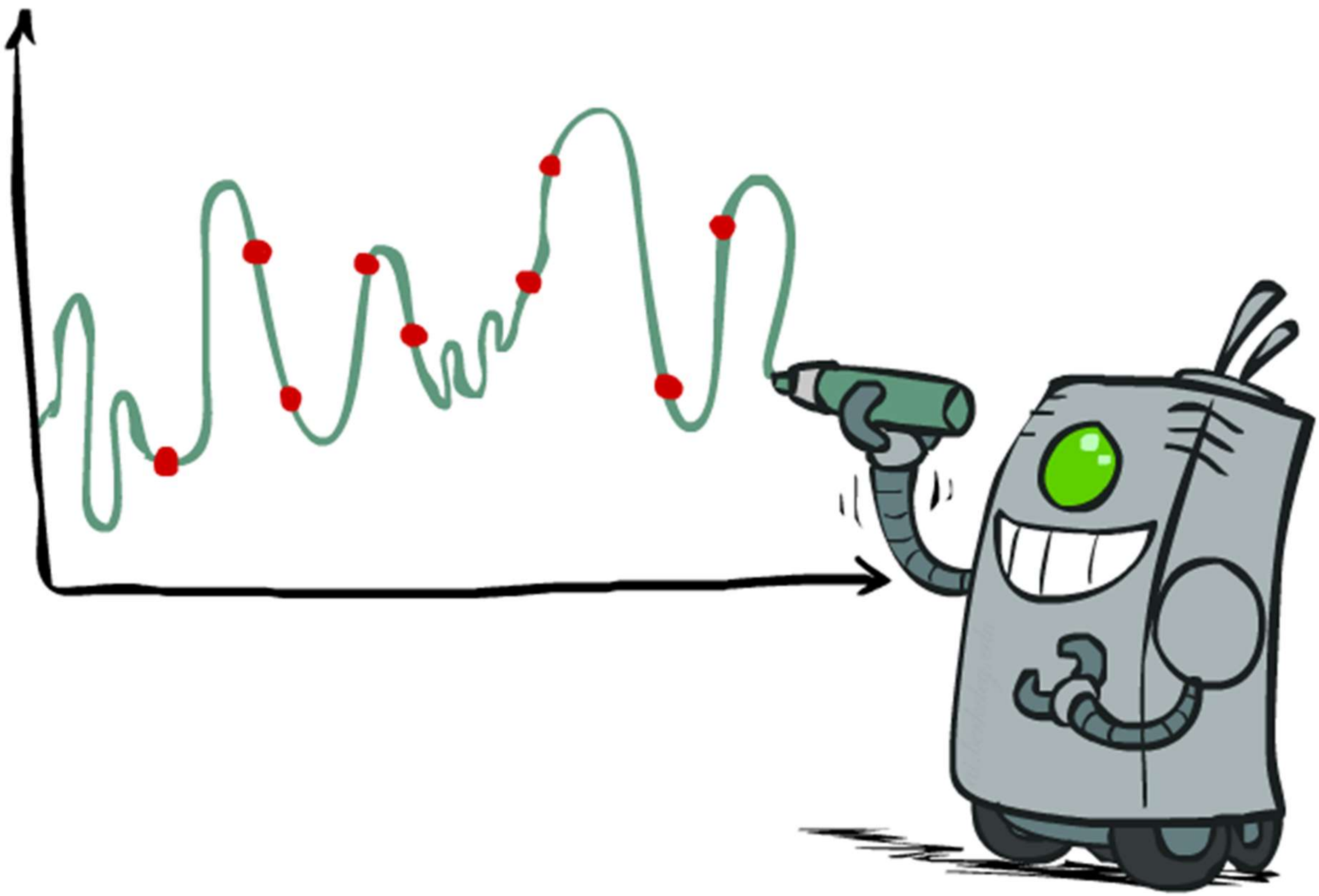


roximate q update explained:

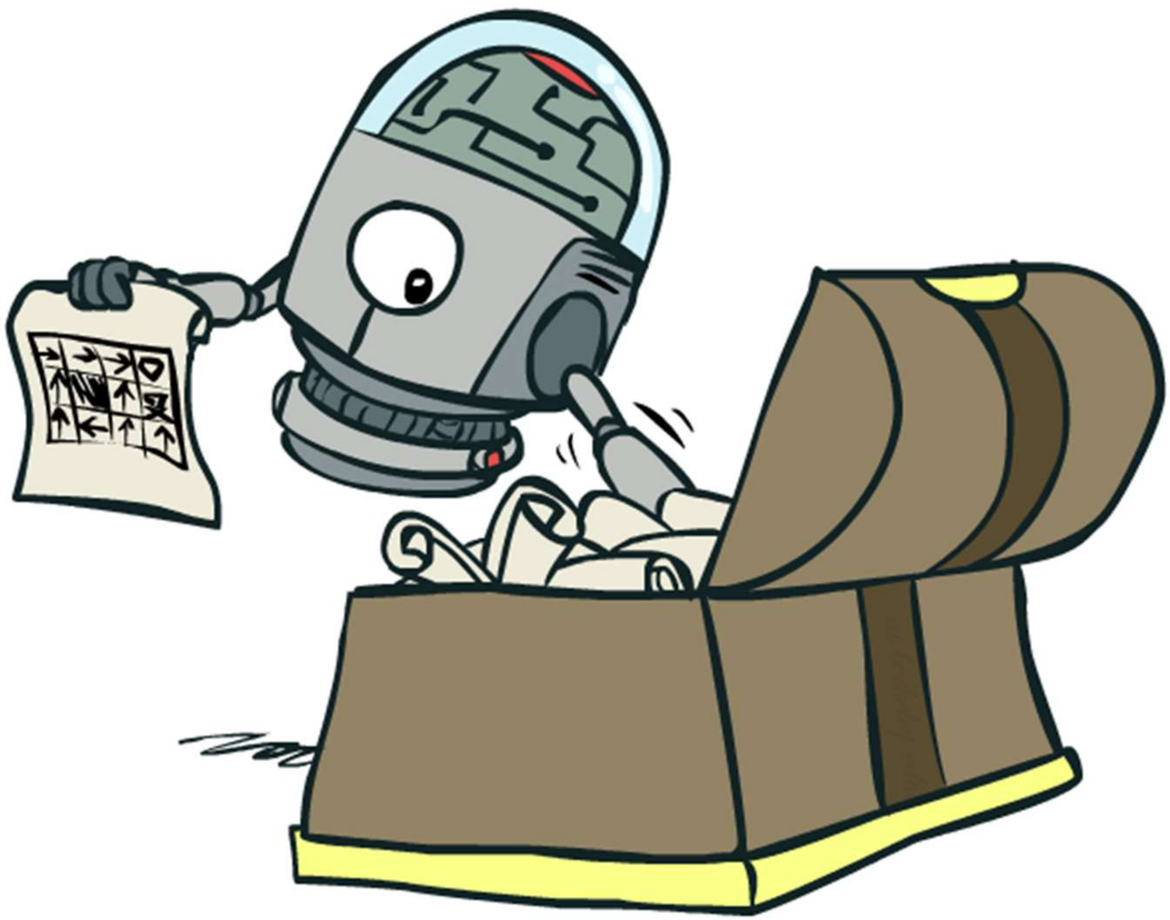
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help*



Policy Search



Policy Search

- roblem: often the feature-based policies that work well (win games, maximize tilities) 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 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

- olution: learn policies that maximize rewards, not the values that predict them

- olicy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbi n feature weights

Policy Search

mplest 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 be

roblems:

- 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

etter methods exploit lookahead structure, sample wisely, change
nultiple parameters…

Policy Search

lg]

[Video: HE

Conclusion

e’re done with Part I: Search and Planning!

e’ve seen how AI methods can solve
oblems in:

- Search
- Constraint Satisfaction Problems
- Games
- Markov Decision Problems
- Reinforcement Learning

ext up: Part II: Uncertainty and Learning!