CSPB3202 Artificial Intelligence

Search with Uncertainty

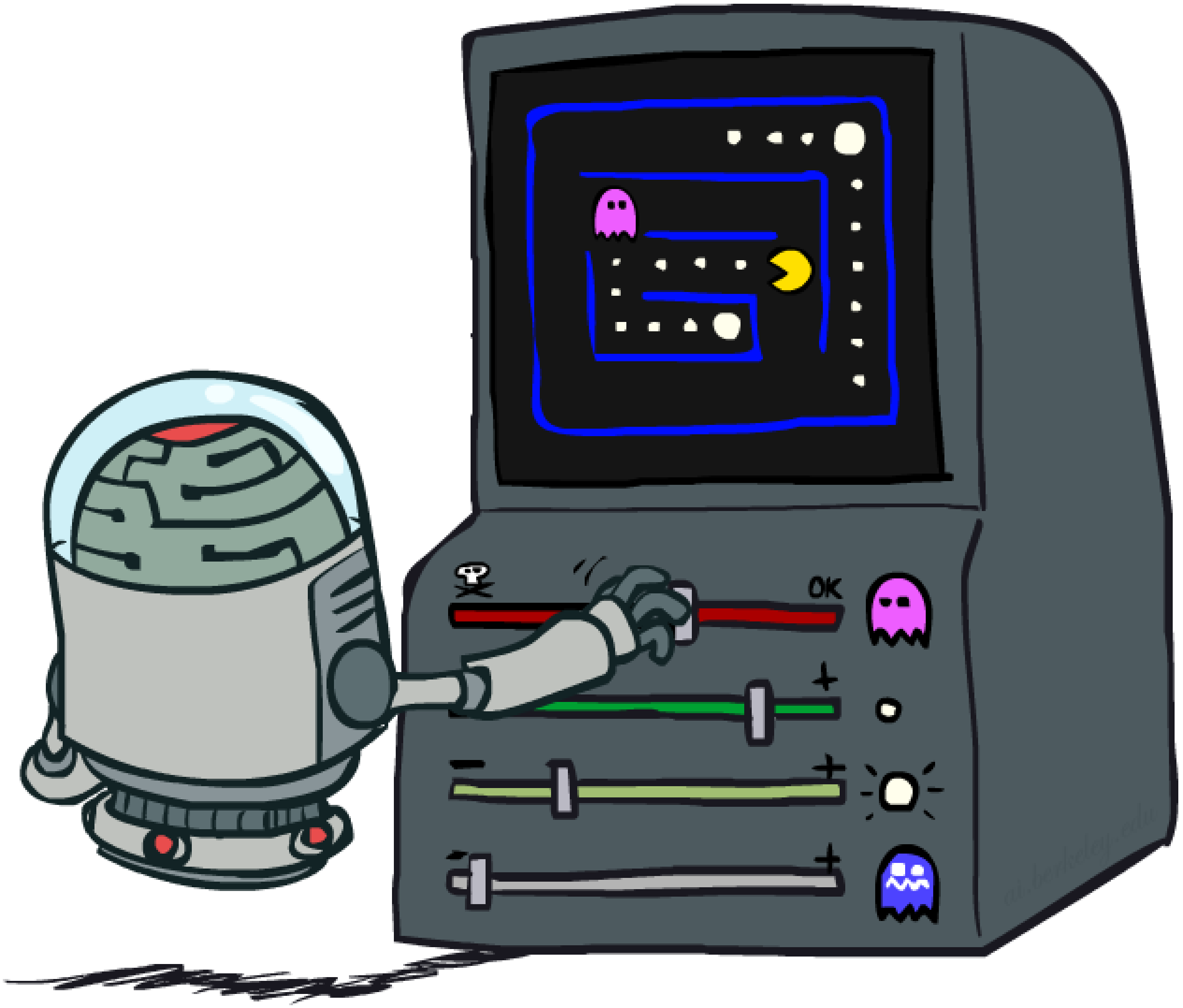
University of Colorado **Boulder**

Note: Some slides were adapted from the materials from Prof. Pieter Abbeel and Prof. Dan Klein at UC Berkeley, and Russell & Norvig's AI book.

Reinforcement Learning

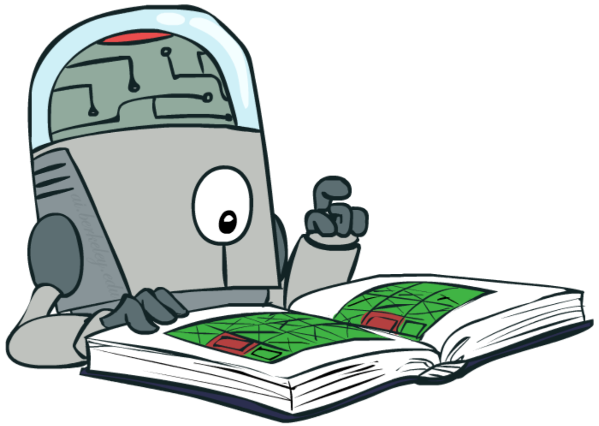
Review- Stories so far

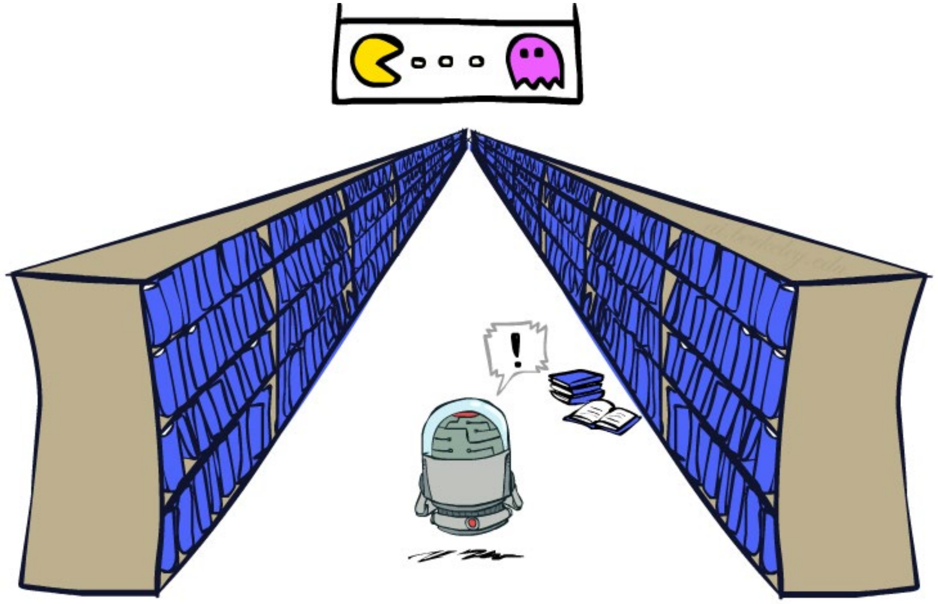
Approximate Q-Learning



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

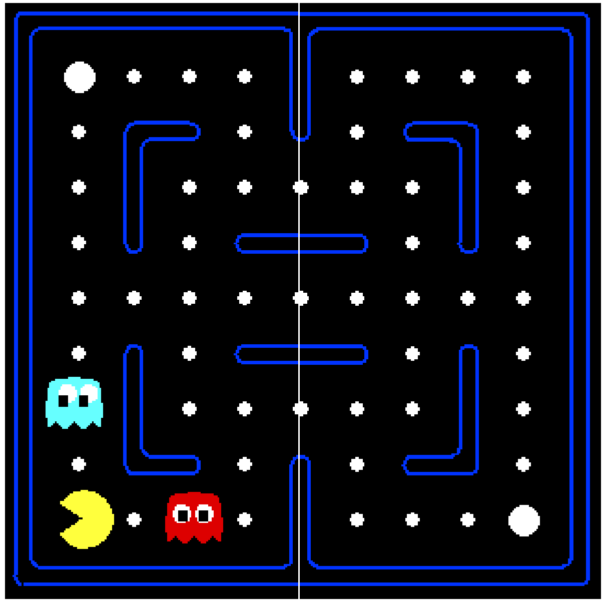




Example: Pacman

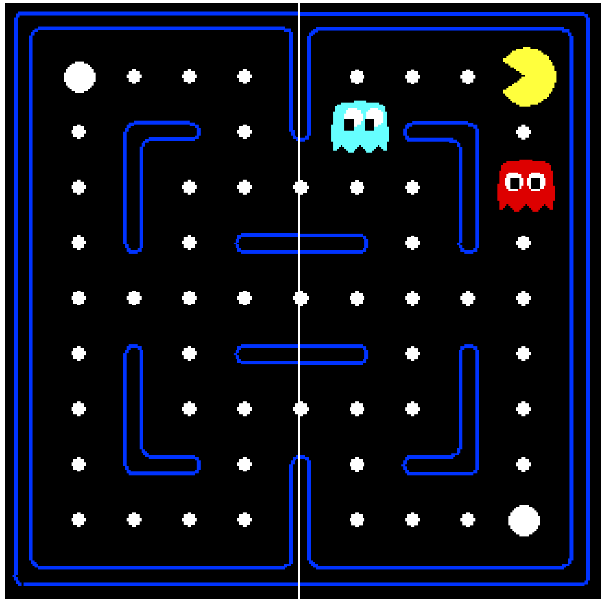
Let's say we discover through experience that this state is bad:

A



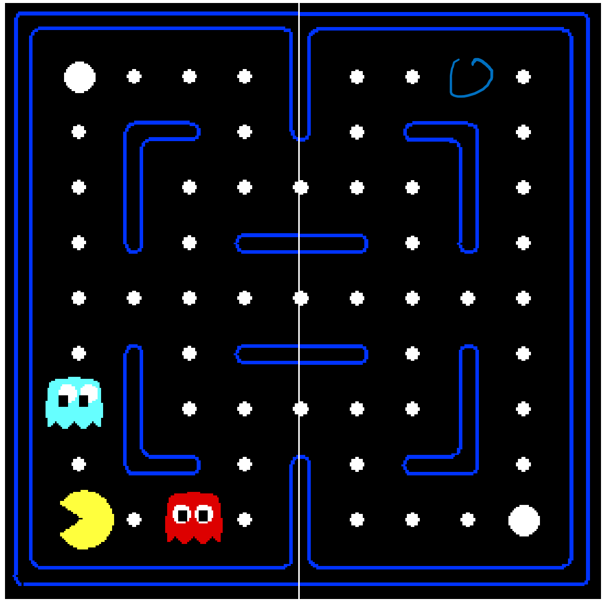
In naïve q-learning, we know nothing about this state:

B



Or even this one!

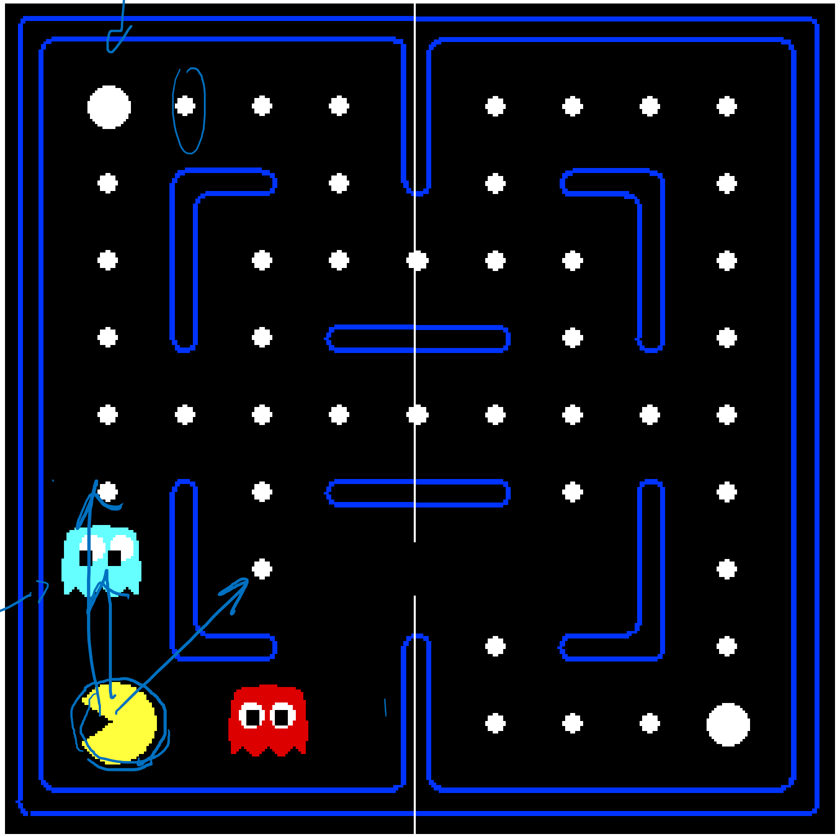
C



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)

$V(s) = F(x_1, x_2 \ldots x_n)$



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

weights := parameters

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

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

→ Linear regression

- 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"

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

- Q-learning with linear Q-functions:

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

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

TD-control (off-policy)

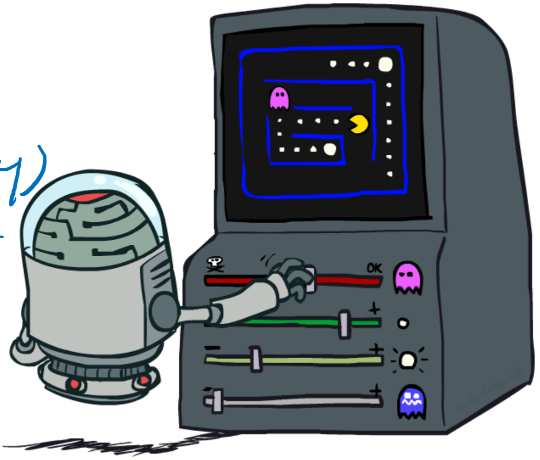
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$

Exact Q's

$$w_i \leftarrow w_i + \underbrace{\alpha}_{\text{learning rate}}\,\underbrace{[\text{difference}]\, f_i(s,a)}_{\Delta w_i}$$

Approximate Q's

Weight update rule



- 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

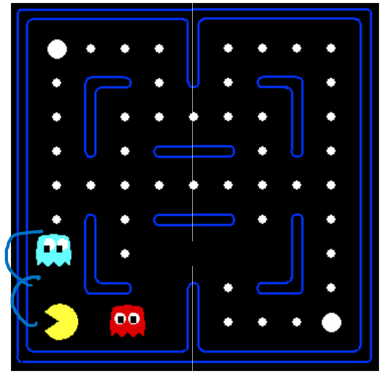
stepsize ; optimization

Example: Q-Pacman

w_{Dot} w_g

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

s



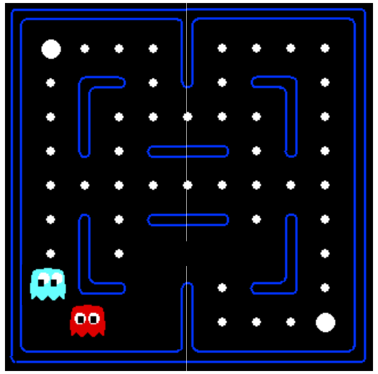
$f_{DOT}(s, \text{NORTH}) = 0.5$ $\quad 1/2 = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$ $\Rightarrow 1$

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

Q new − Q old = −501 $\quad 0 < \alpha < 1$

w^D_{old} + α

difference $= -501$

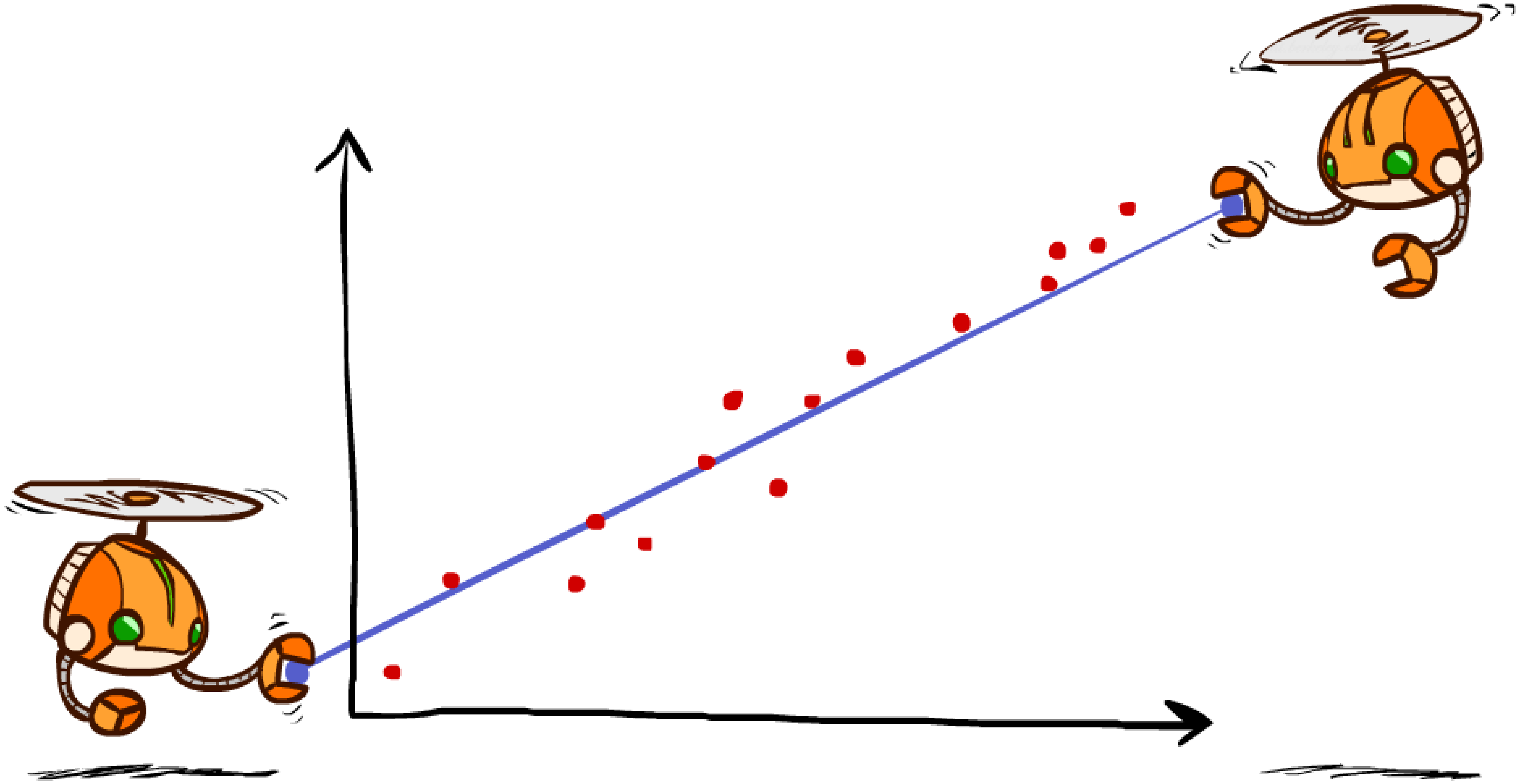
$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$ $\quad f_D$ $\quad -1$

$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$ $\quad \alpha = 2/501$

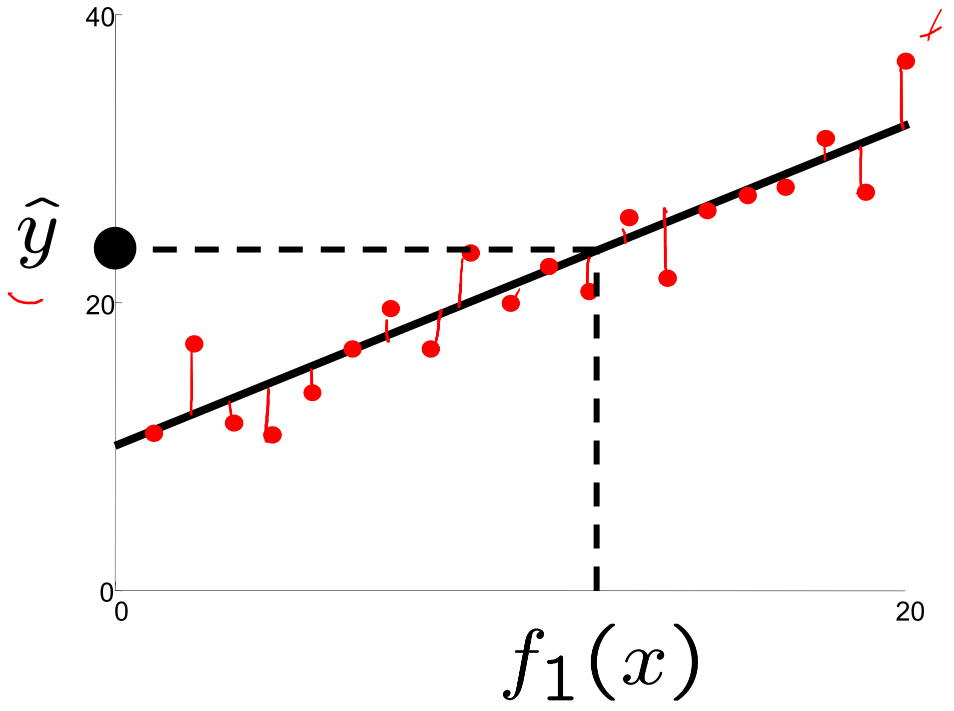
$w_D = 3$ $\quad w_G = -3$ $\quad -2$

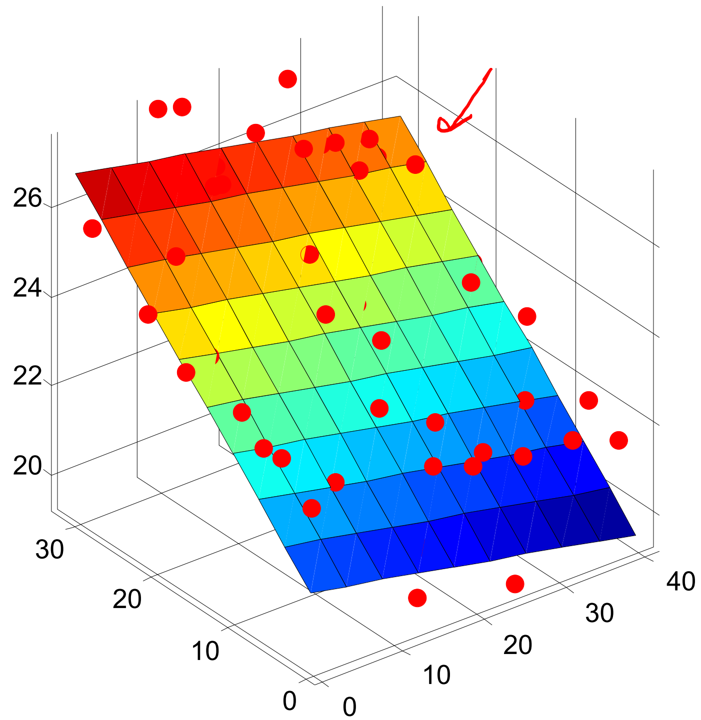
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

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

b

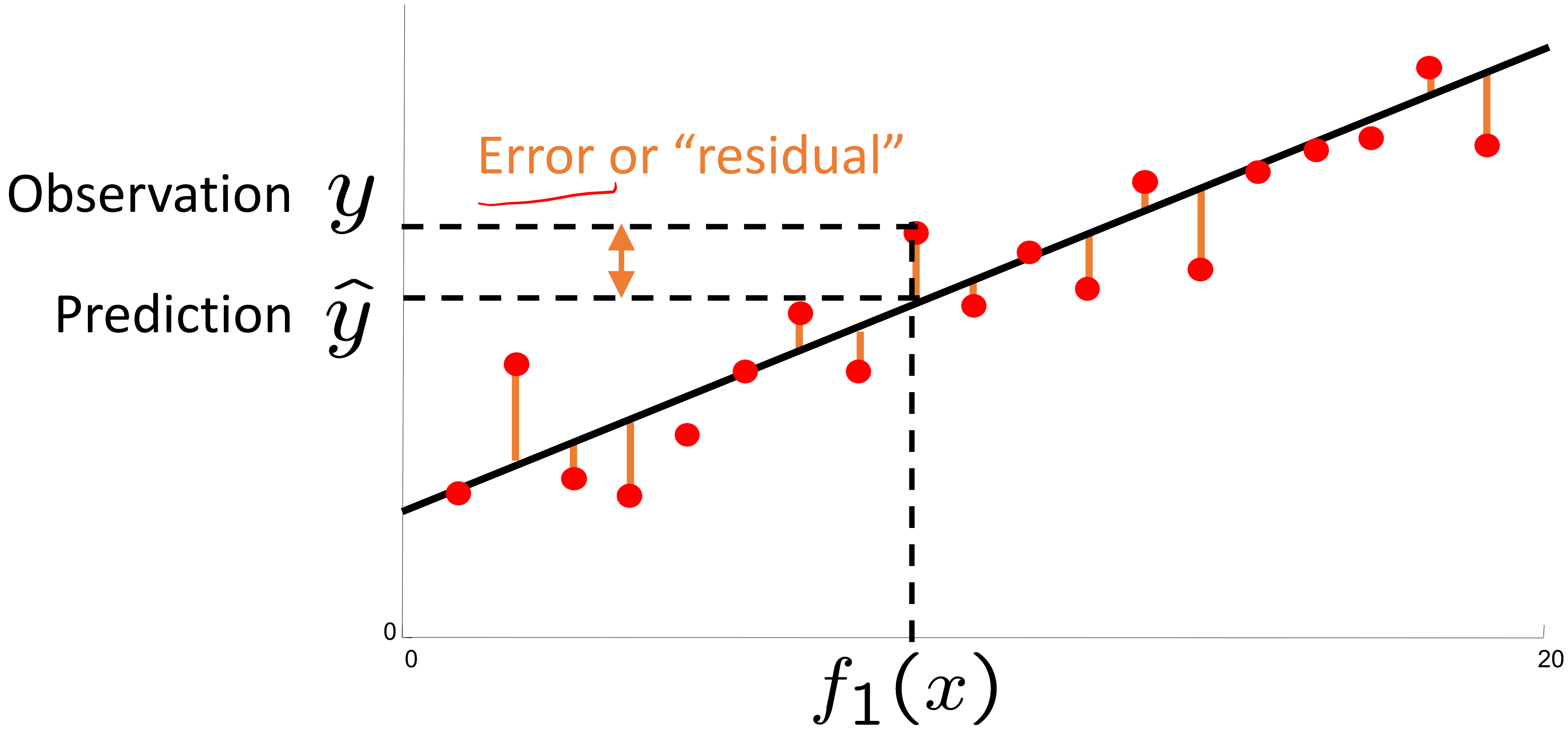
Prediction:

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

Optimization: Least Squares*

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

RSS



Weight update rule- Gradient Descent*

Least squres Method (Linear Regression)

$f(w,x) \to \hat{y} \in \mathbb{R}$

$\frac{\partial f}{\partial w_j} \sum_i^N 2\cdot\left(y_i - \sum_j^P w_j x_{ji}\right) \cdot (-x_{ji})$

Loss ft $\mathcal{L}(w,x) =$ RSS or MSE

(J)

$$RSS = \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \boxed{\sum_i^N \left(y_i - \sum_j^P w_j \cdot x_{ji}\right)^2}$$

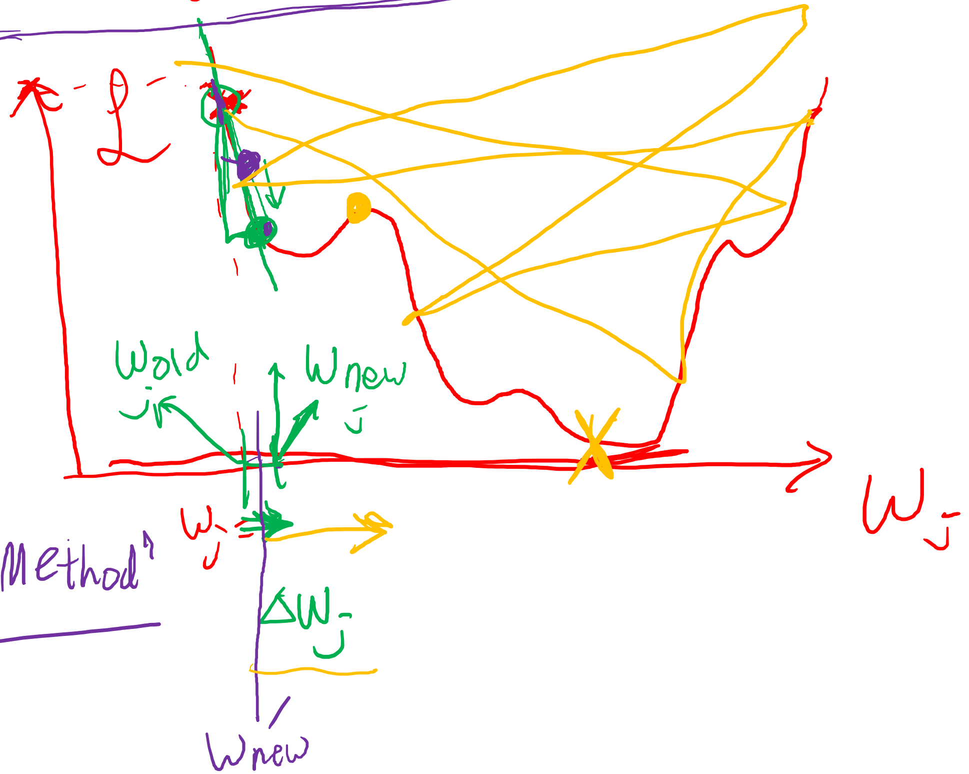
$$MSE = \frac{1}{N}\cdot RSS$$

$$\to \quad w_j^{new} \leftarrow w_j^{old} - \alpha \cdot \frac{\partial L}{\partial w_j}$$

$$w_j^{new} \leftarrow w_j^{old} + \alpha\left(y^{(i)} - \hat{y}^{(i)}\right)\cdot x_j^{(i)}$$

"Gradient Descent Method"

→ ML



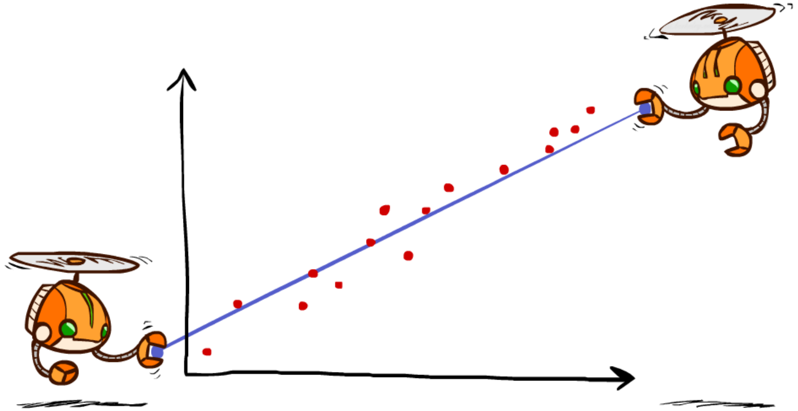
Minimizing Error*

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

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

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

$$w_m \leftarrow w_m + \alpha \left(y - \underbrace{\sum_k w_k f_k(x)}_{\text{predicted Q}}\right) f_m(x)$$

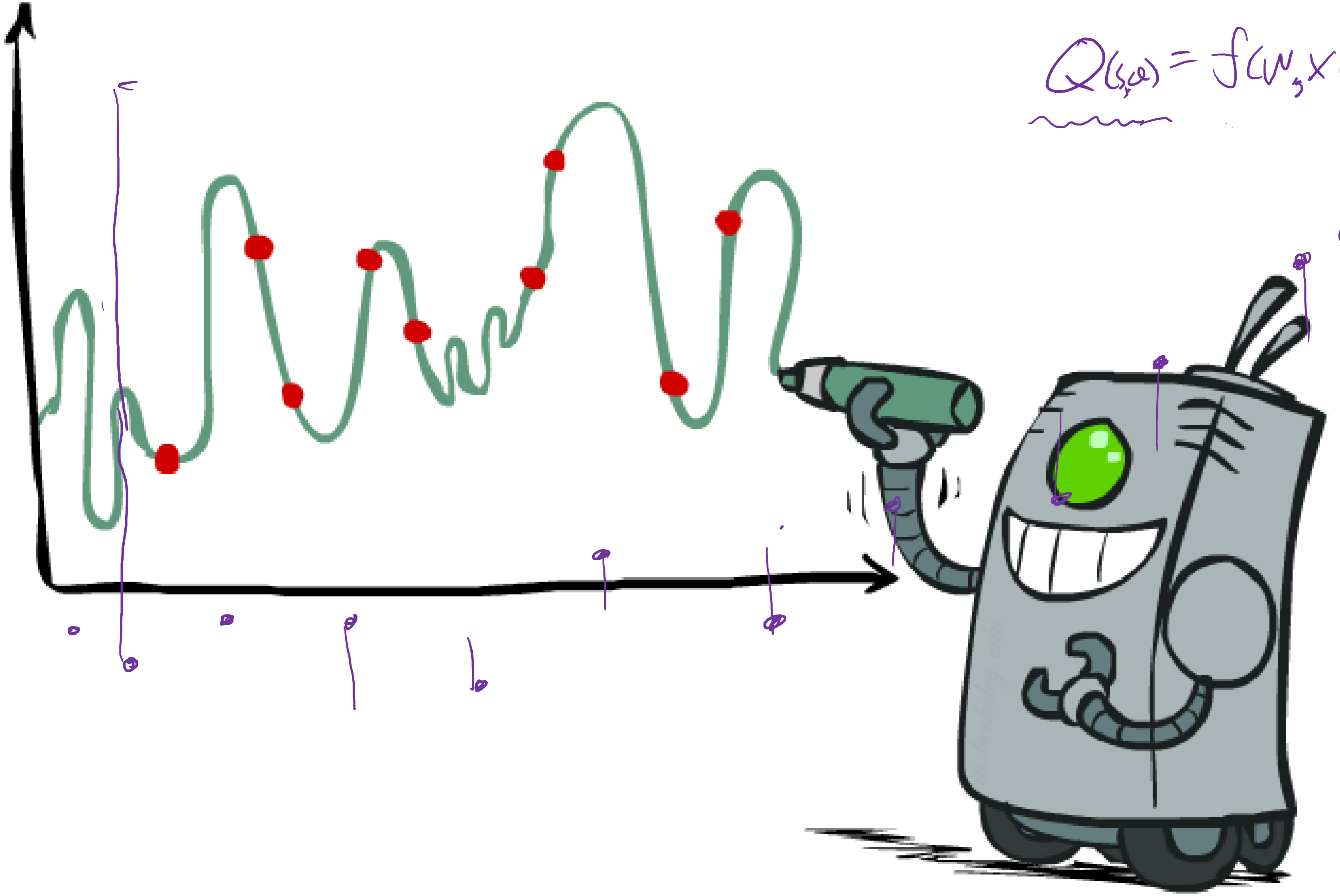


Approximate q update explained:

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

"target" "prediction"

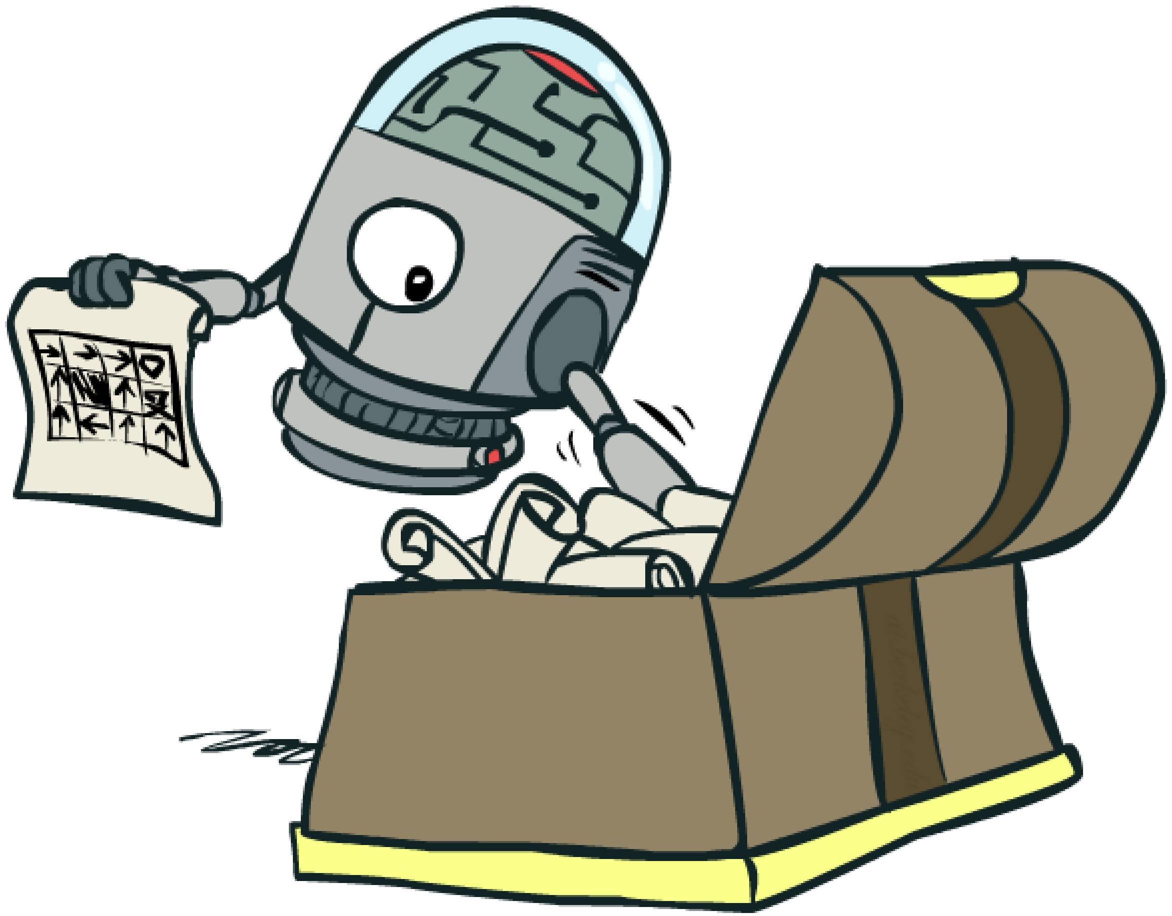
Overfitting: Why Limiting Capacity Can Help*



$Q(s,a) = f(w, x(s,a))$

overfitting

Policy Search



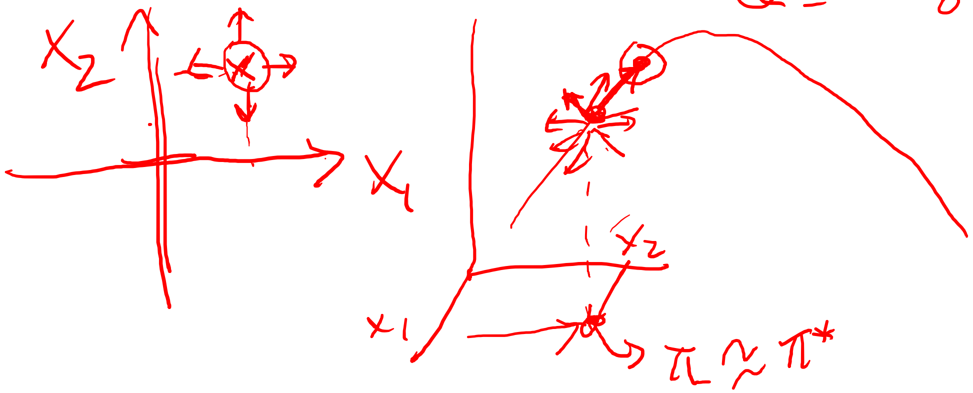
Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)

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

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

$Q = w_0 + w_1 x_1 + w_2 x_2$



Policy Search

N time
a_j

$Q = \sum_j w_j x_j$

$\pi \to ?$

$x = f(s)$

$\pi = P(a \mid s)$

$\pi_w(s,a)$ = softmax

$= e^{\hat{Q}_w(s,a)} / \sum_{a'} e^{\hat{Q}_w(s,a')}$

$a \leftarrow \{a'\}$

$\frac{df}{f} = d(\log f)$

2

hill climbing

policy gradient

$\rho \leftarrow$ policy value $\neq Q$

$\nabla_w \rho(w) = \nabla_w \left(\sum_a \pi_w(a|s) R(a) \right)$

$= \sum_a \frac{\nabla_w(\pi_w) R}{\pi_w} \cdot \pi_w = \sum_a (\nabla_w \log \pi_w) R \cdot \pi_w$

$\approx \frac{1}{N} \sum_j^N (\nabla_w \log \pi(w)) \cdot R$

REINFORCE

Policy Search

Conclusion

- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Process
  - Reinforcement Learning