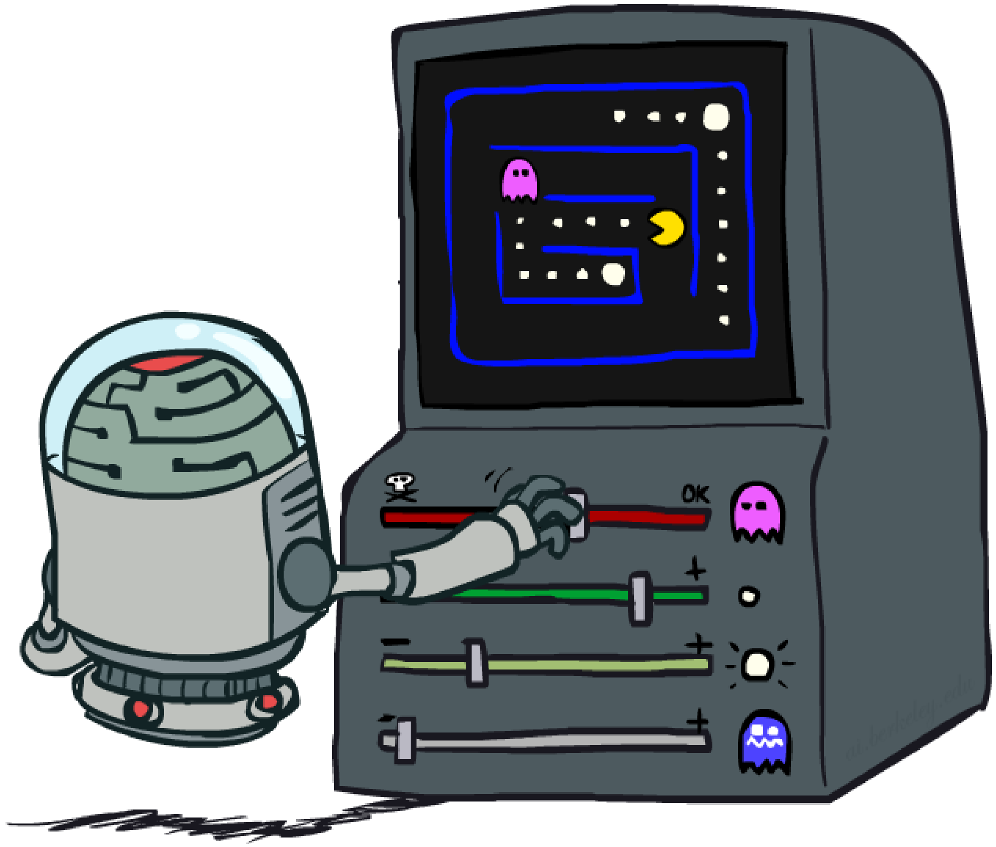
加强学习(reinforcement learning) II



加强学习

- 假定情景仍为一个 MDP:
  - 一个 状态集合 $s \in S$
  - 一个 行动集合 (每个状态) A
  - 一个 转移模型 T(s, a, s’)
  - 一个 奖赏值函数 R(s, a, s’)
- 依然是寻找一个策略 $\pi(s)$

- 不一样的地方是: 不知道 T 或 R, 所以必须去尝试不同的行动(获取相应的奖赏值)
- Big idea: 利用(行动)样本结果来计算基于T的(Q状态)均值

从 MDPs 到 RL

已知 MDP: Offline Solution

Goal	Technique
计算 V*, Q*, π*	Value / policy iteration
评估一个给定的 π	Policy evaluation

未知 MDP: Model-Based

Goal	Technique
计算 V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

未知 MDP: Model-Free

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

不基于模型的 Learning

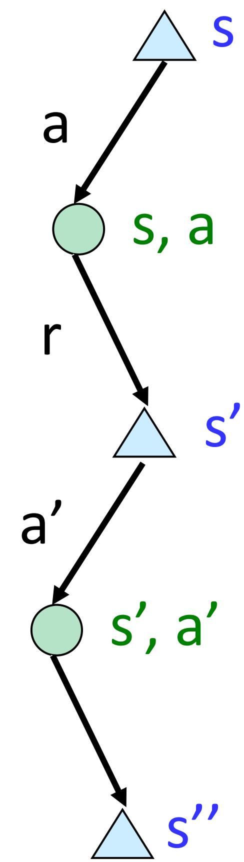
- Model-free (时间差分temporal difference) learning
  - Experience world through episodes

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

  - Update estimates each transition

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

  - 反复更新最终模仿了 Bellman updates



Approximating Values through Samples

- Policy Evaluation:

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

- Value Iteration:

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

- Q-Value Iteration:

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

Q-Learning （Q-值学习）

- 对于每个Q状态，We'd like to do Q-value updates:

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

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

- 转而计算 均值 as we go
  - 每获得一个 样本 transition (s,a,r,s')
  - This sample suggests

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

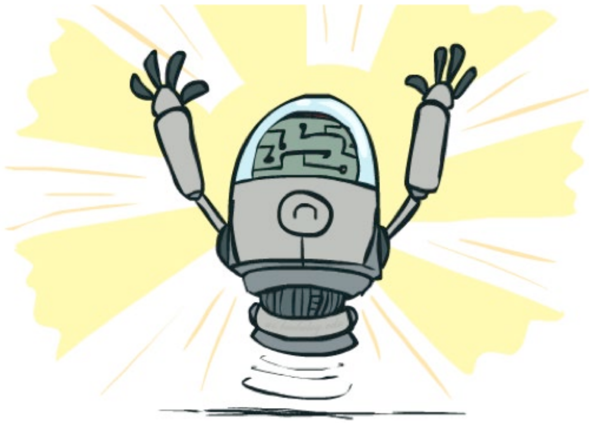
  - But we want to average over results from (s,a) (Why?)
  - So keep a running average

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

Q-Learning 属性

- 优势: Q-learning 收敛于最优（行动）策略 -- even if you're acting suboptimally!

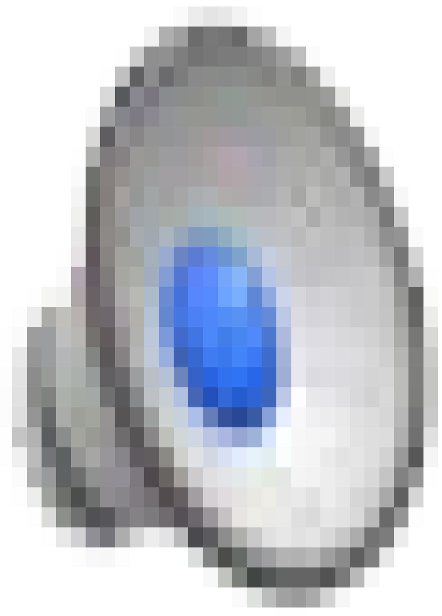
- This is called off-policy learning



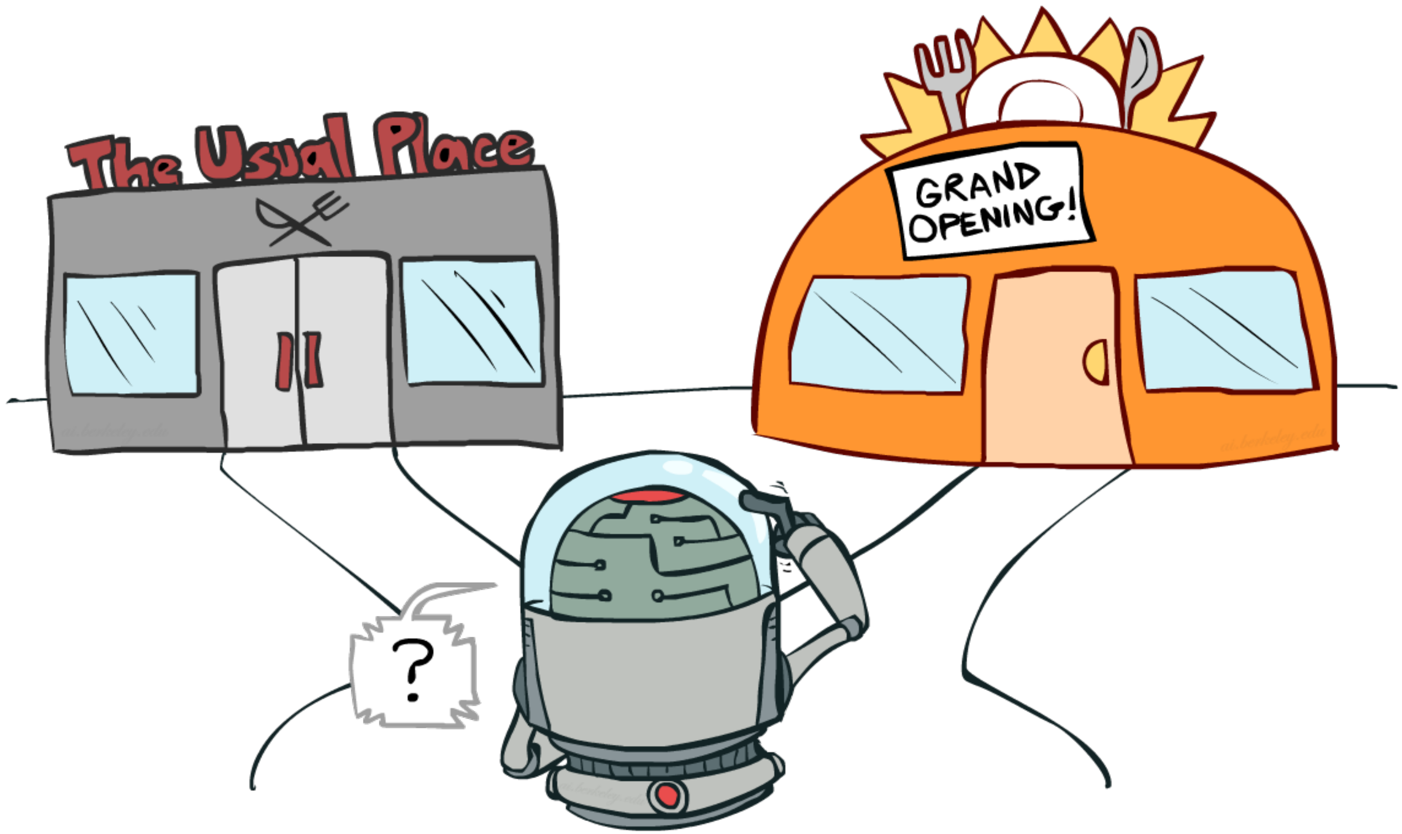
- Caveats 缺点:
  - You have to explore enough
  - 最终要使 learning rate 变得足够小
  - … 但是又不能让它减小的太快
  - Basically, in the limit, it doesn't matter how you select actions (!)

[Demo: Q-learning – auto – cliff grid (L11D1)]

视频演示Q-Learning Auto Cliff Grid



Exploration探索 vs. Exploitation利用



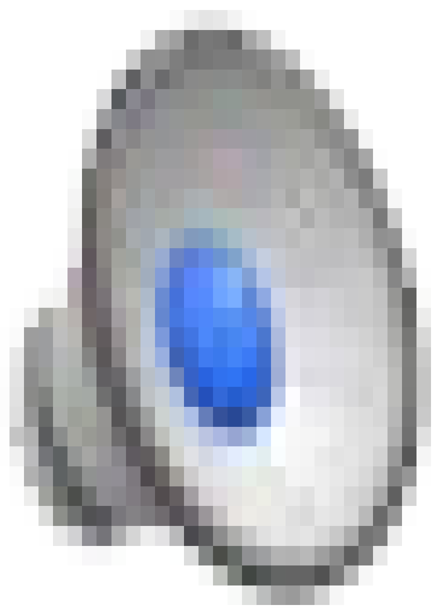
How to Explore?

- 几种方法用来执行 exploration
  - 最简单的: 随机选择行动 (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

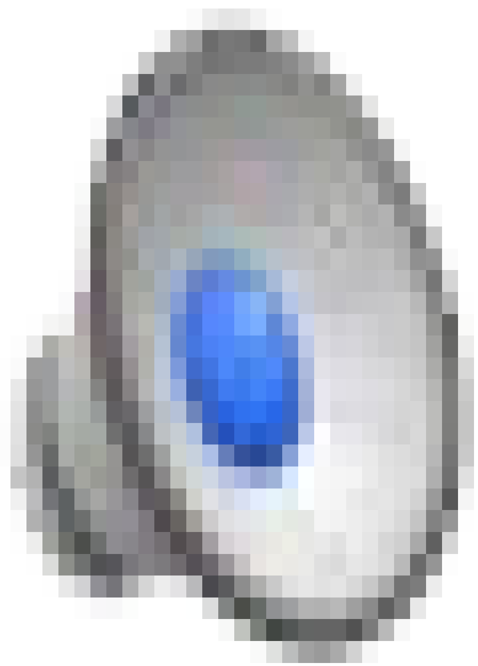
  - 随机选择行动的问题所在?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: 逐渐减小 ε over time
    - Another solution: 使用探索函数 exploration functions

[Demo: Q-learning – manual exploration – bridge grid (L11D2)] [Demo: Q-learning – epsilon-greedy -- crawler (L11D3)]

视频演示Q-learning – Manual Exploration – Bridge Grid



视频演示 Q-learning – Epsilon-Greedy – Crawler



Exploration Functions探索函数

- When to explore?
  - Random actions: explore a fixed amount
  - 更好的想法: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function 探索函数
  - 两个输入，一个是估计的Q值 u，另一个是 访问Q状态的次数 n, and returns an optimistic utility, e.g.

$$f(u,n) = u + k/n$$

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

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

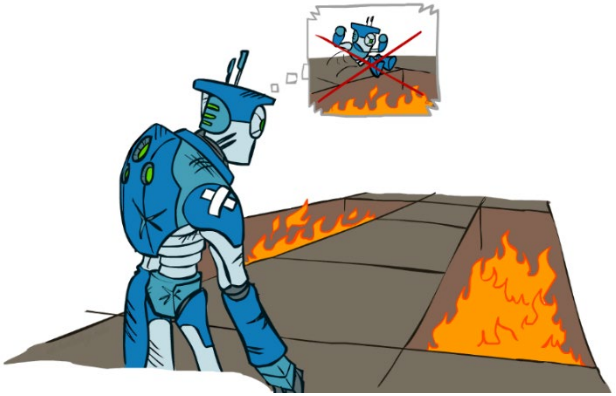
- Note: this propagates the “bonus” back to states that lead to unknown states as well!

[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

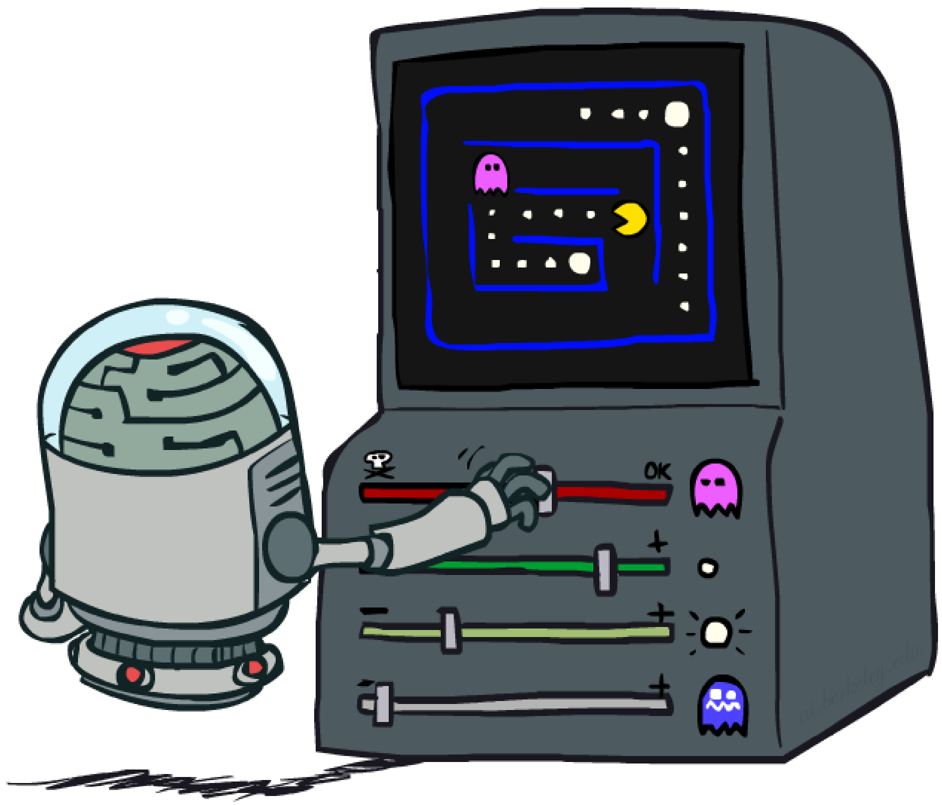
视频演示Q-learning – 探索函数 – Crawler

Regret

- 在学习最优策略的过程中，会犯错误（选错行动，导致不好的结果）
- Regret 是一个衡量对于你的机器人所犯错误导致的代价之和：你的期望奖励值之差值，即你所获得的奖励值和最优奖励值的差值
- 最小化 regret 本身涉及到学习优化策略以外的事 - 需要你的学习方法（算法）本身也是最优化的
- 例如：随机探索算法random exploration 和 利用探索函数exploration functions 的算法，都可以找到最优行动策略，但是前者有更大的 regret

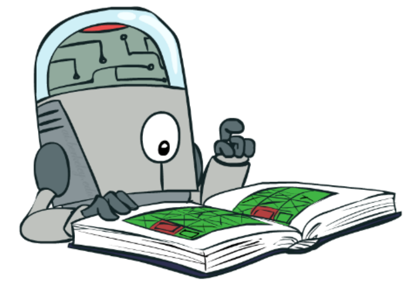


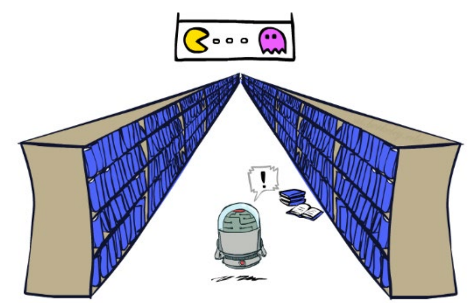
Approximate Q-Learning



Generalizing Across States（泛化状态）

- Basic Q-Learning keeps a table of all q-values
- 在现实情境中, 不可能学习关于每个状态!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- 转而, 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

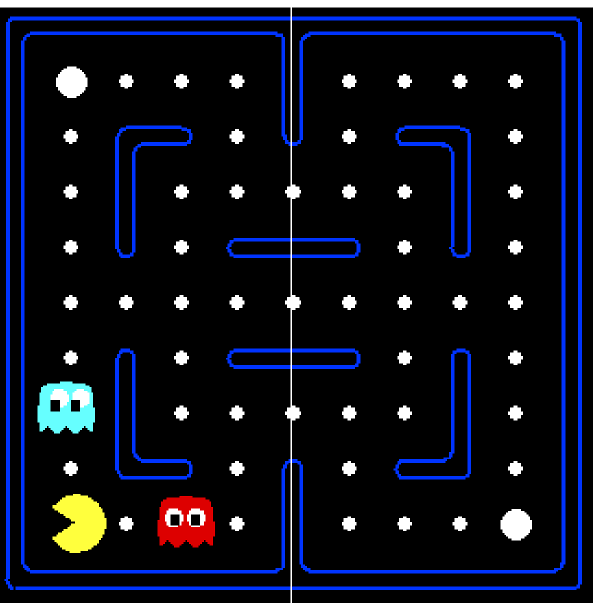




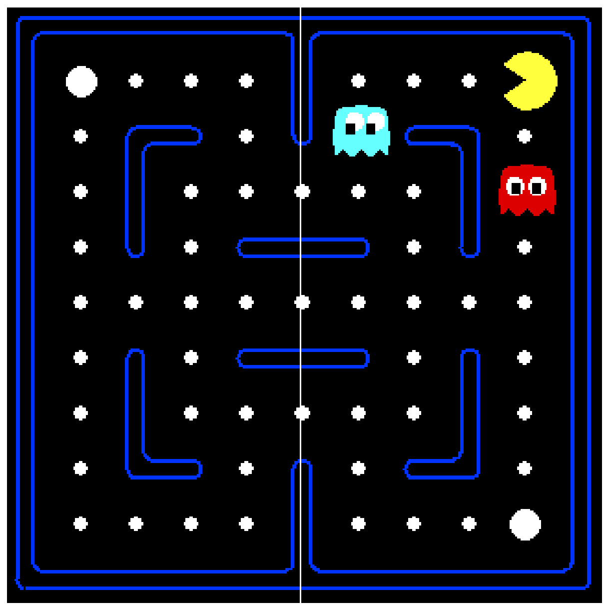
[demo – RL pacman]

Example: Pacman

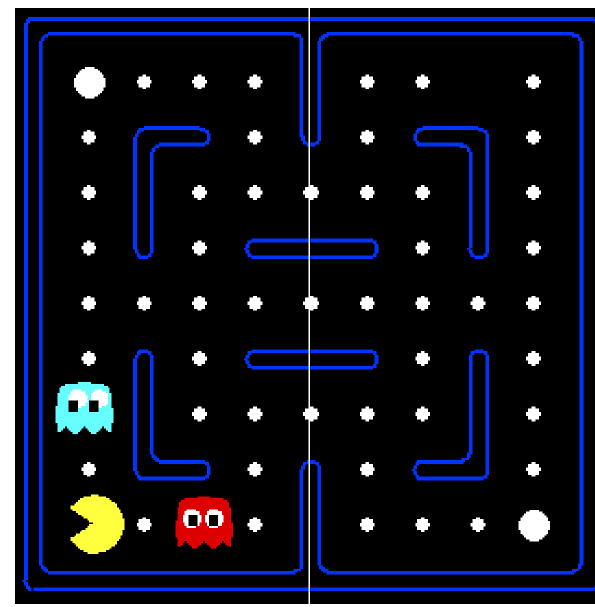
假设我们通过经验发现这个状态是槽糕的:



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

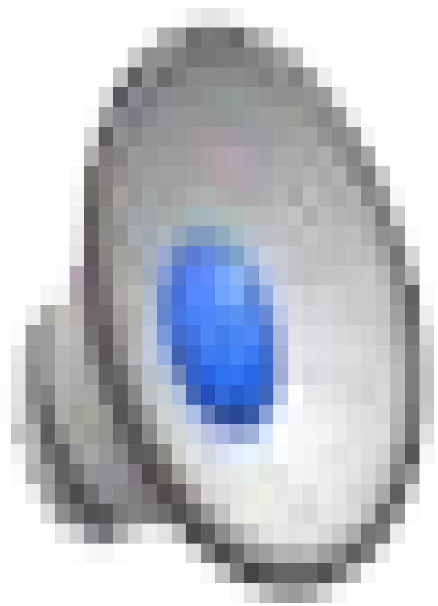


Or even this one!

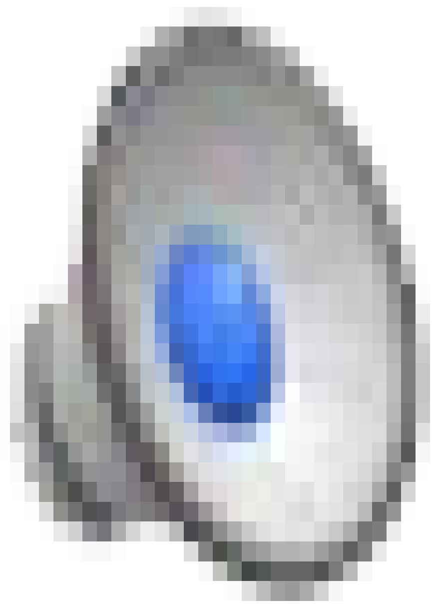


[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

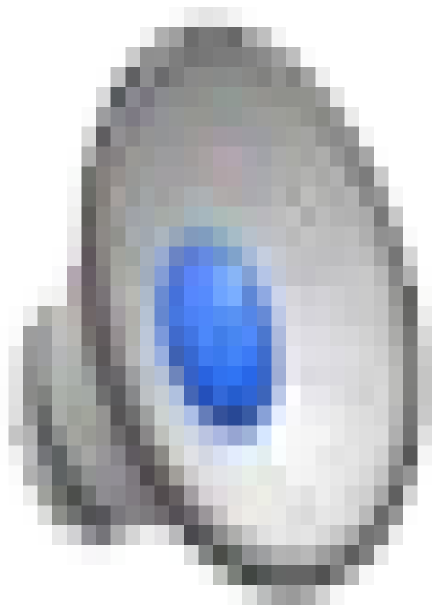
视频演示Q-Learning Pacman – Tiny – Watch All



视频演示Q-Learning Pacman – Tiny – Silent Train

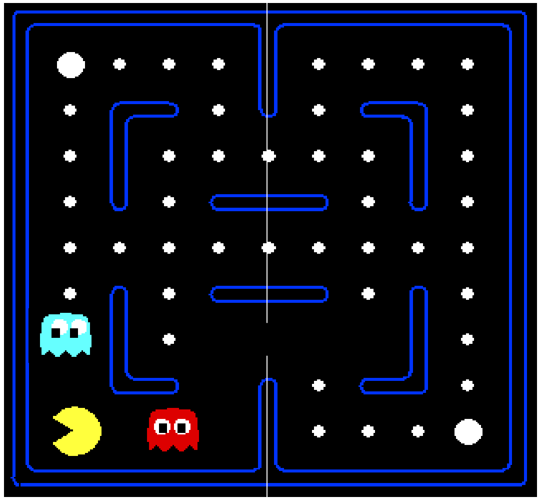


视频演示Q-Learning Pacman – Tricky – Watch All



基于特征的状态表达

- Solution: describe a state using a vector of features (properties) 使用特征向量
  - 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)



特征值的线性拟合

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

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

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

- 优势: our experience is summed up in a few powerful numbers

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

Approximate 近似 Q-Learning

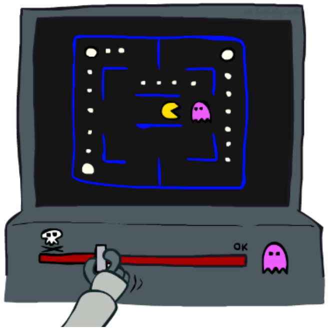
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear Q-functions:

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

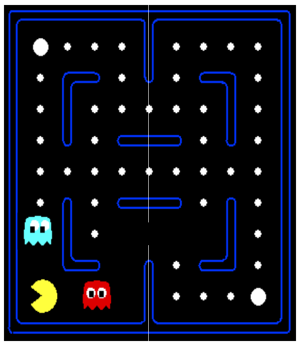
- 简单的解释:
  - 调整激活的特征值的权重
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares 在线最小二值法

举例: Q-Pacman

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

s



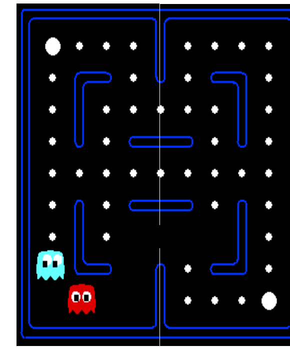
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

difference $= -501$

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

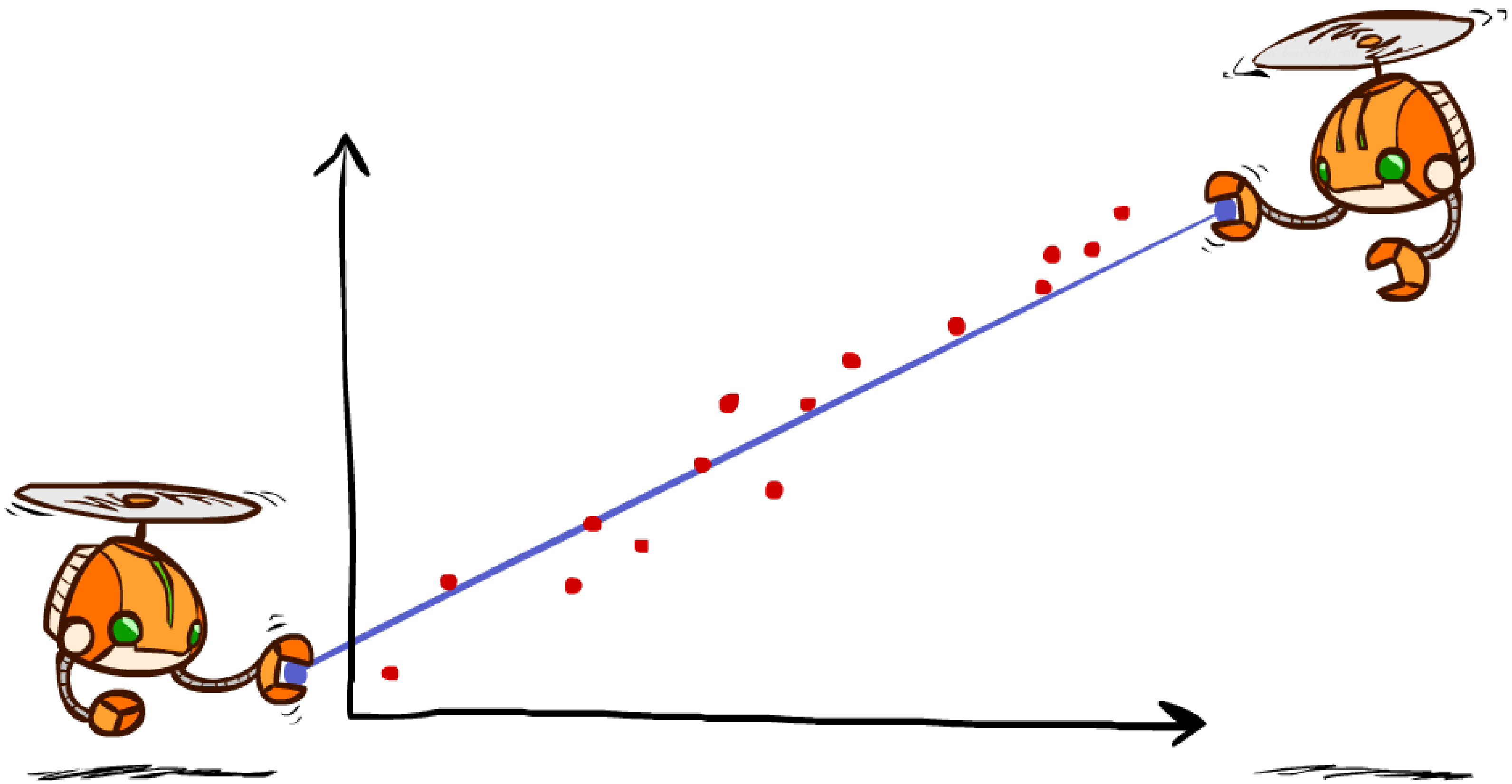
$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$

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

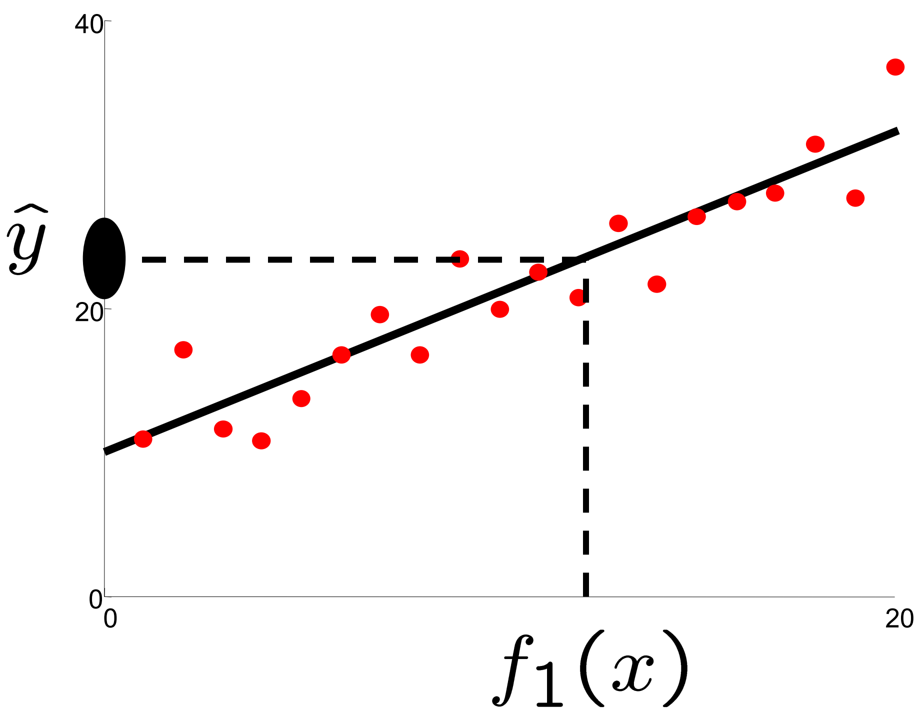
[Der
learning

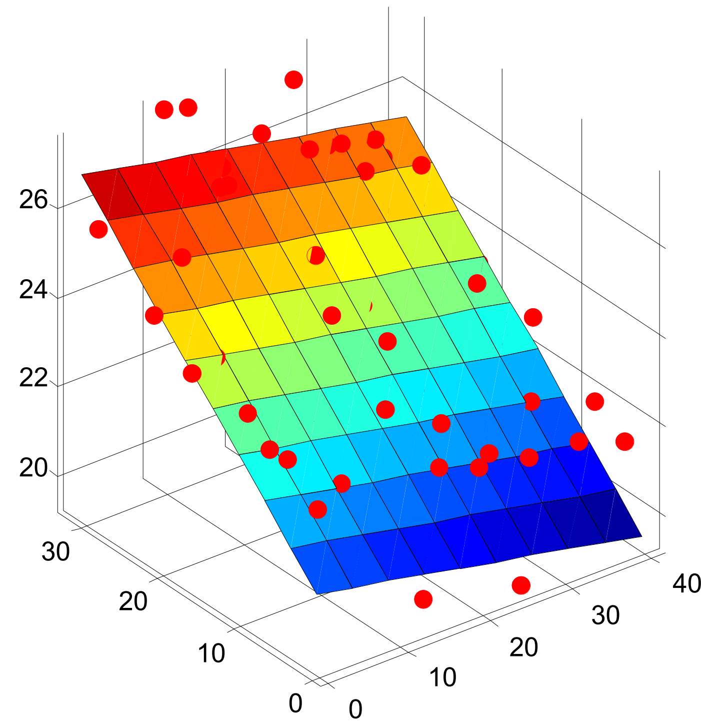
视频演示Approximate Q-Learning -- Pacman

Q-Learning and Least Squares



Linear Approximation: Regression*





Prediction:

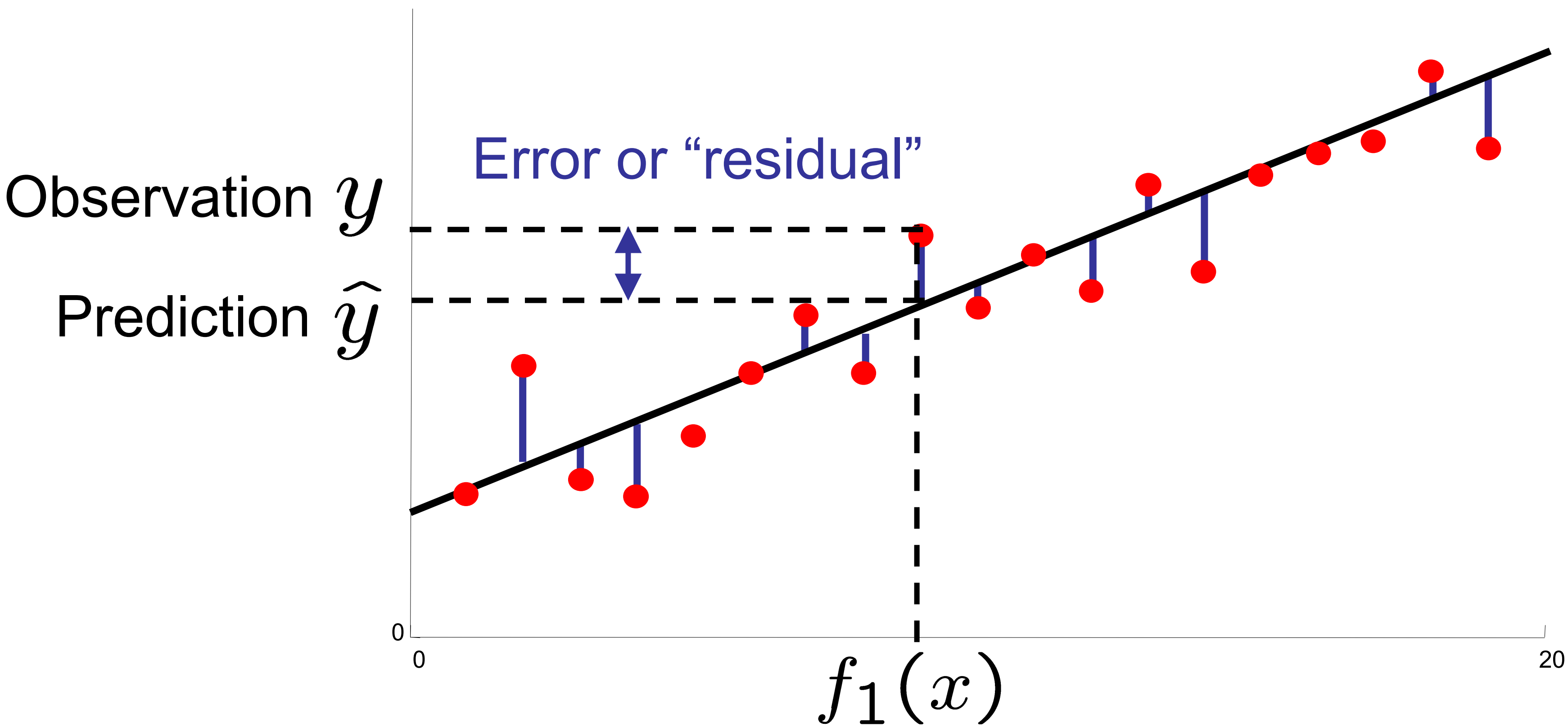
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

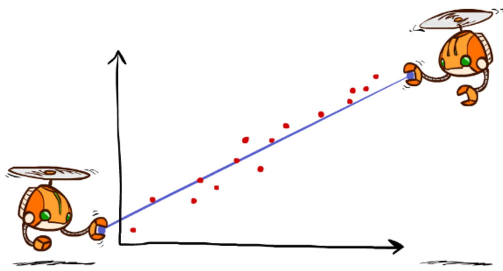
Optimization: Least Squares*

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



Minimizing Error*

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



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

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

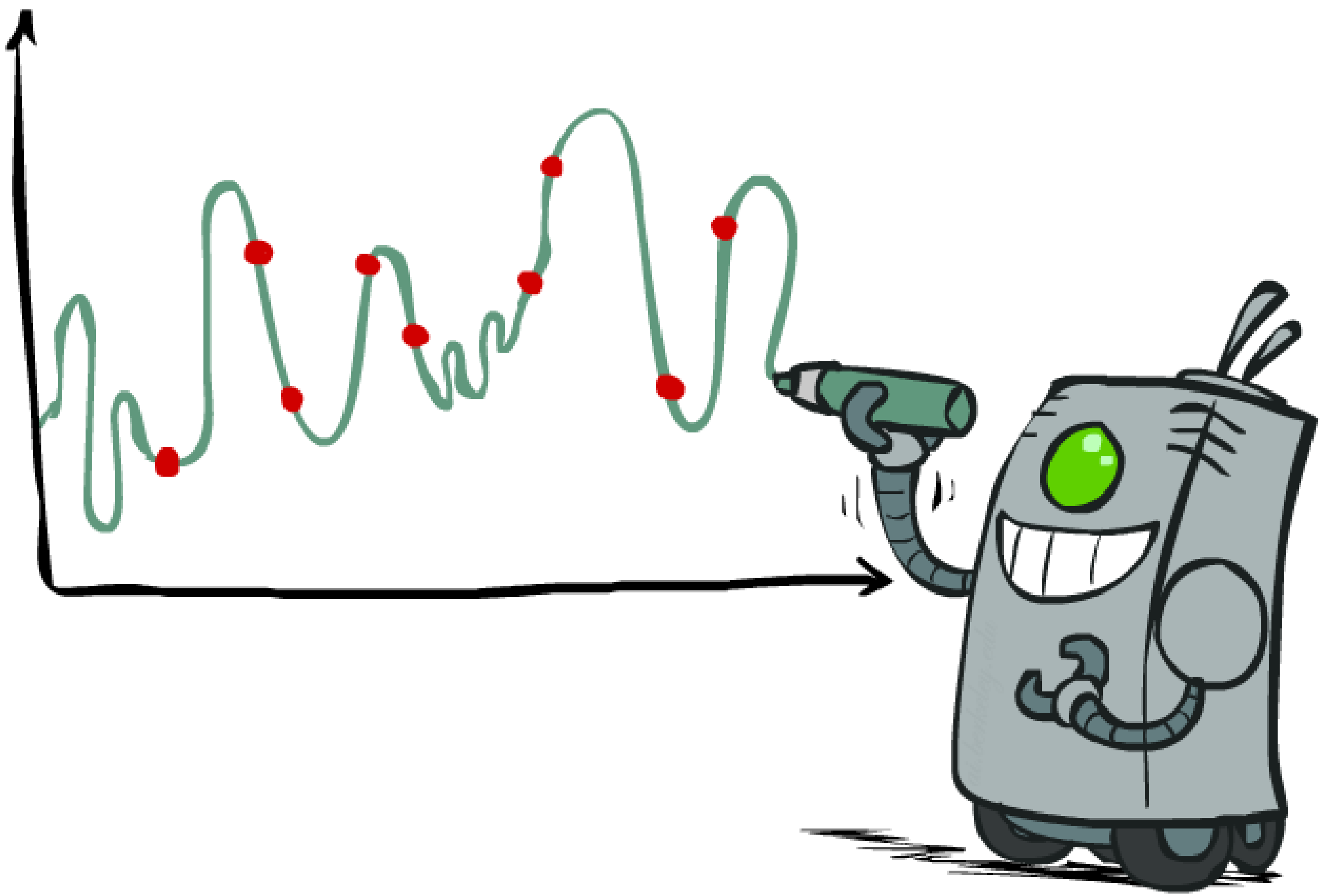
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update explained:

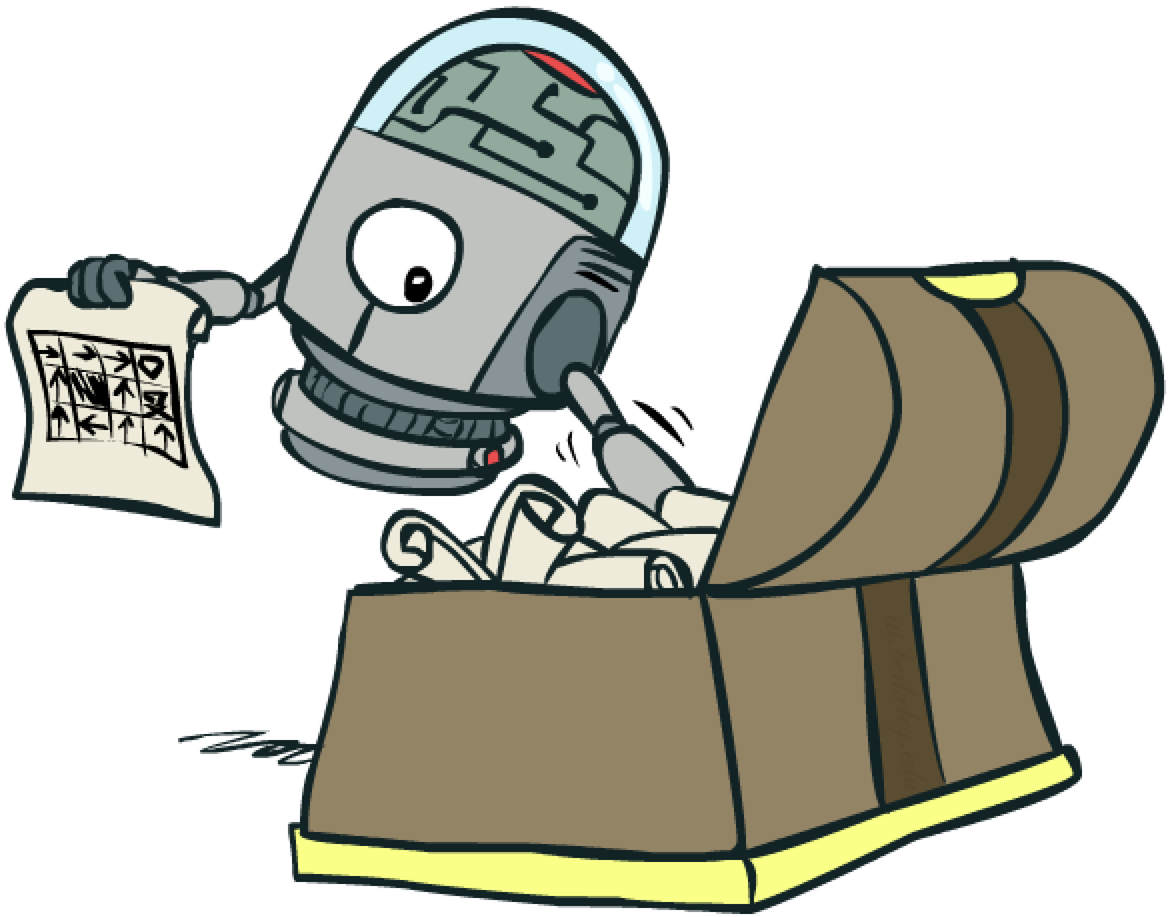
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

“target” “prediction”

Overfitting: Why Limiting Capacity Can Help*



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
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they 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

- 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

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
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Policy Search

[Andrew Ng]

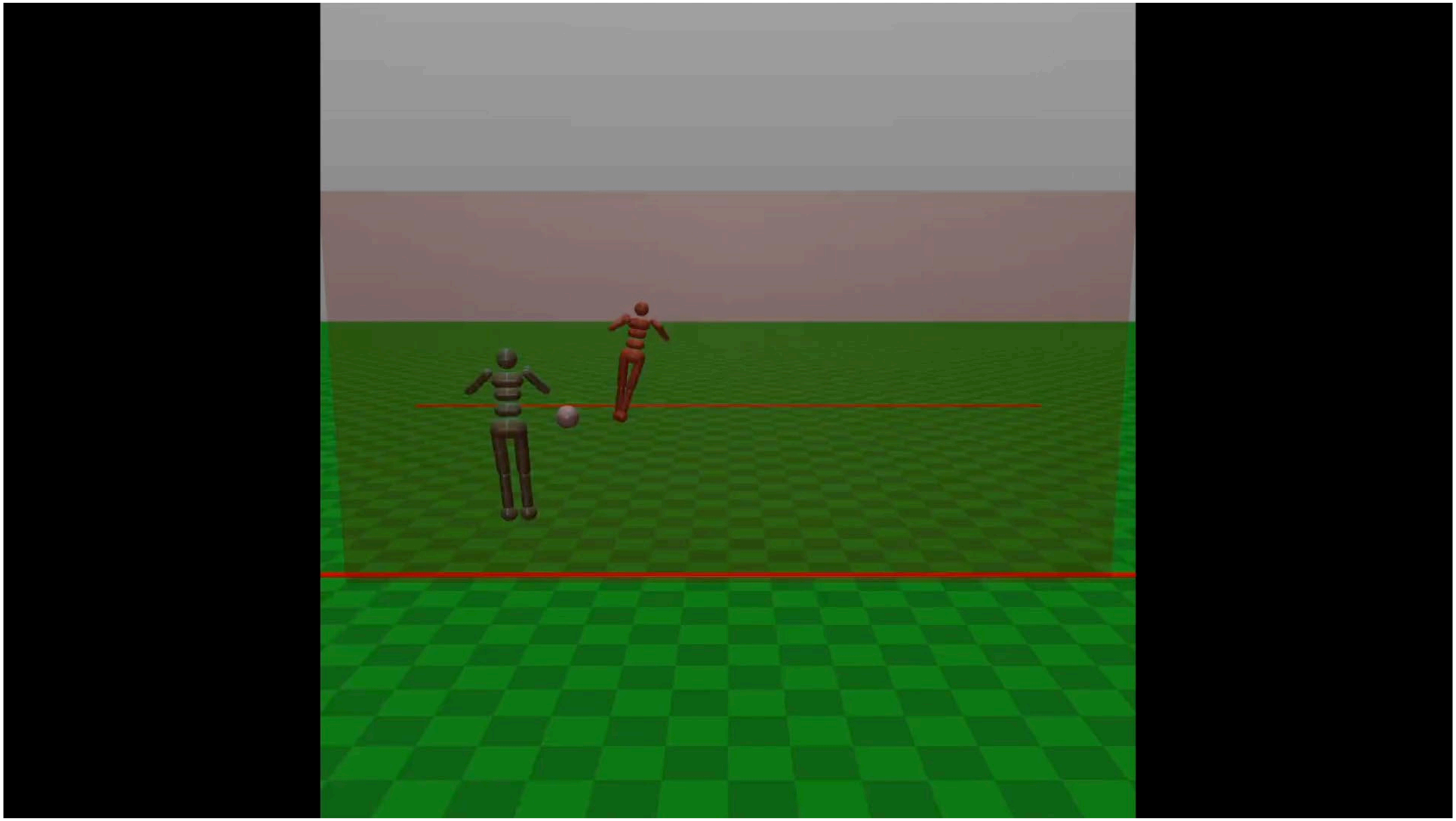
[Video: HELICOPTER]

RL: Learning Locomotion

[Schulman, Moritz, Levine, Jordan, Abbeel, ICLR 2016]

[Video: GAE]

RL: Learning Soccer



[Bansal et al, 2017]

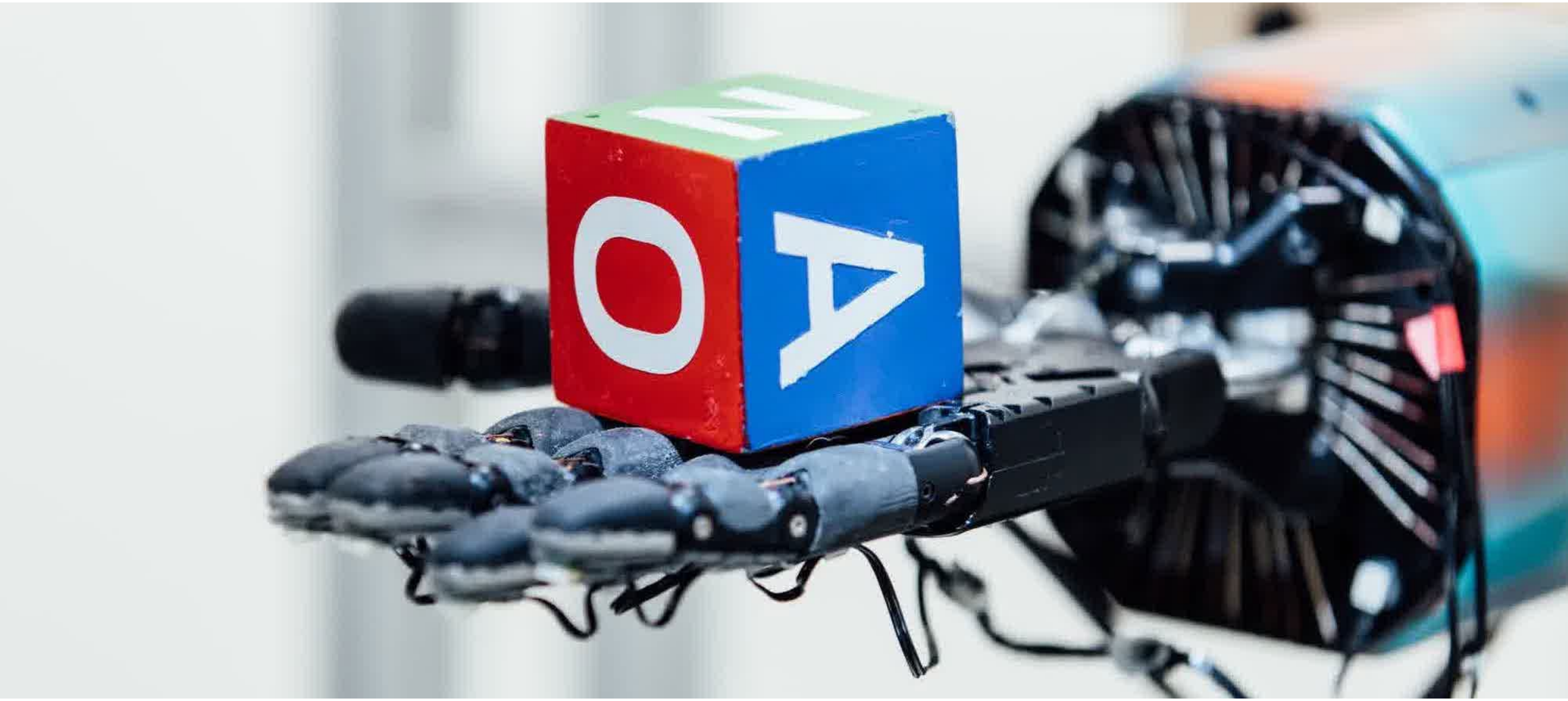
RL: Learning Manipulation

[Levine*, Finn*, Darrell, Abbeel, JMLR 2016]

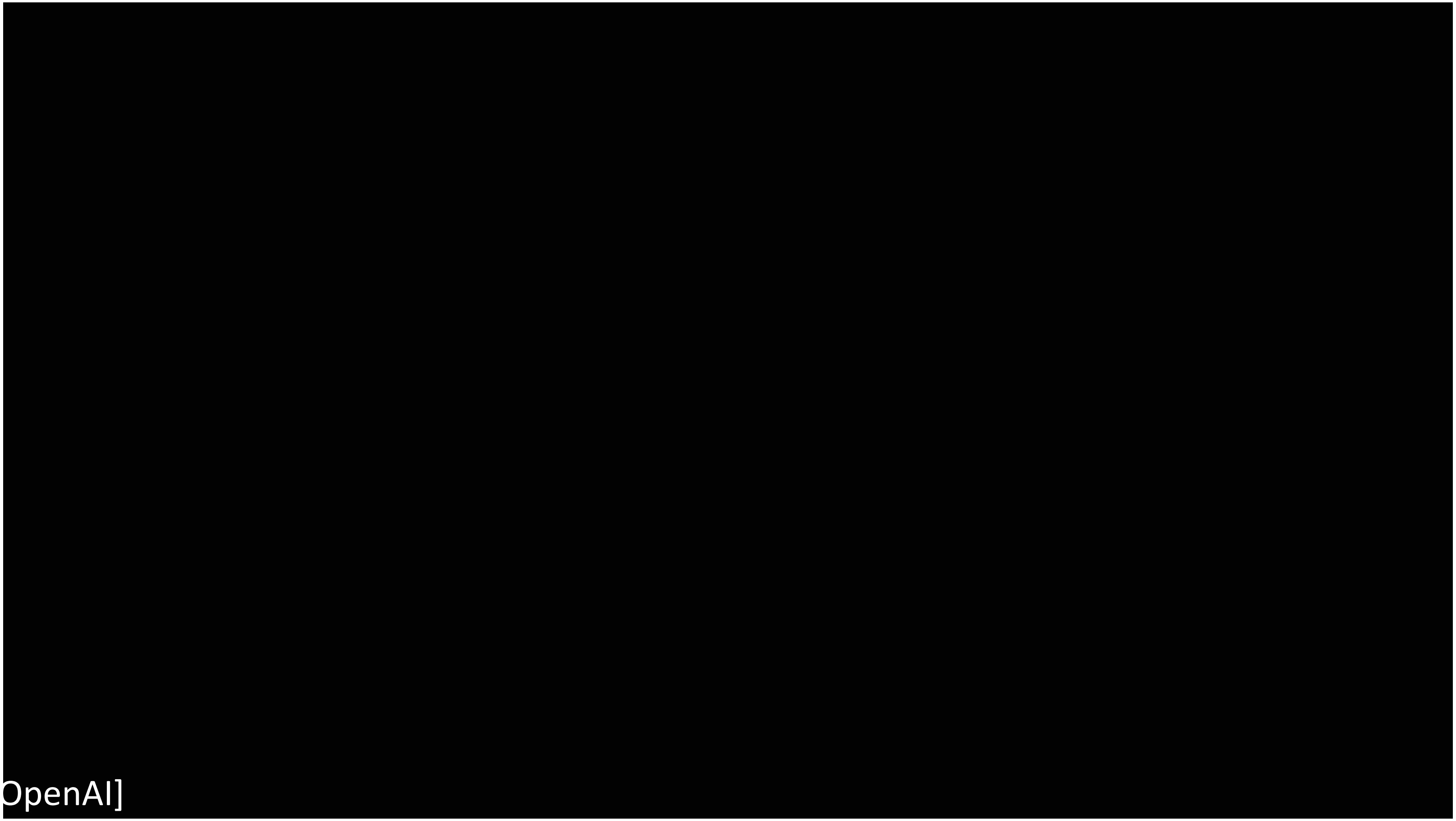
RL: NASA SUPERball

[Geng*, Zhang*, Bruce*, Caluwaerts, Vespignani, Sunspiral, Abbeel, Levine, ICRA 2017]

RL: In-Hand Manipulation



OpenAI: Dactyl



Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning

- Next up: Part II: Uncertainty and Learning!