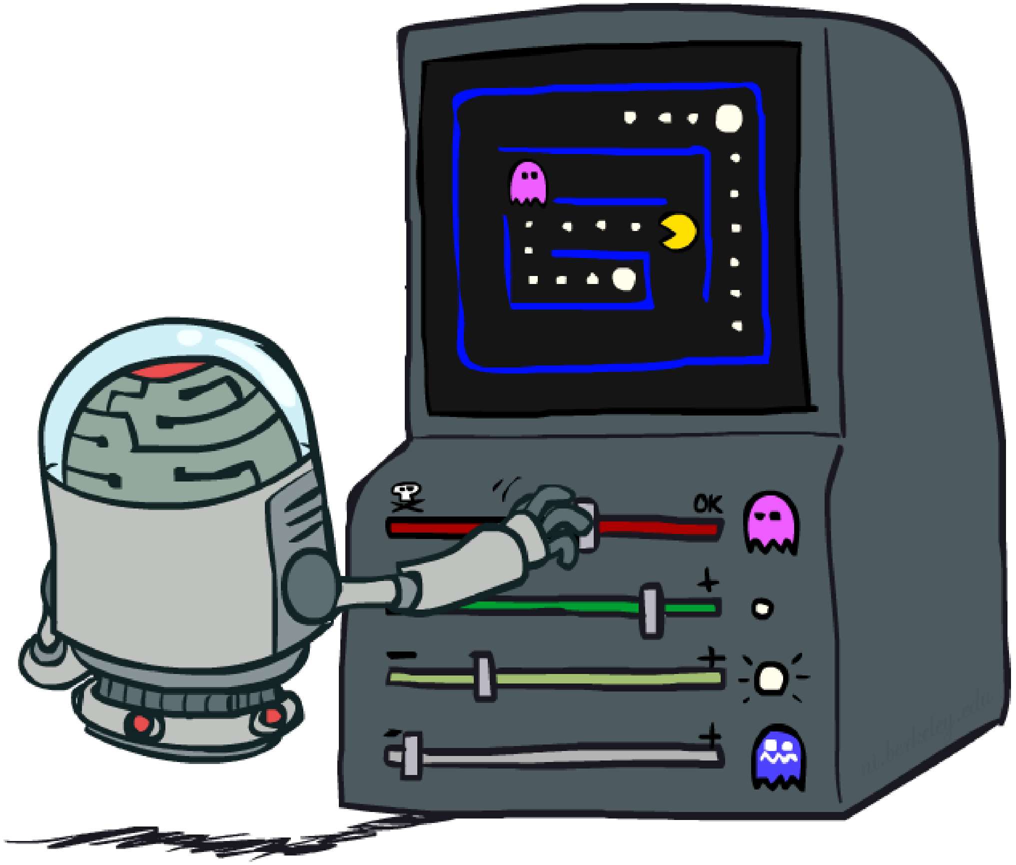
Artificial Intelligence

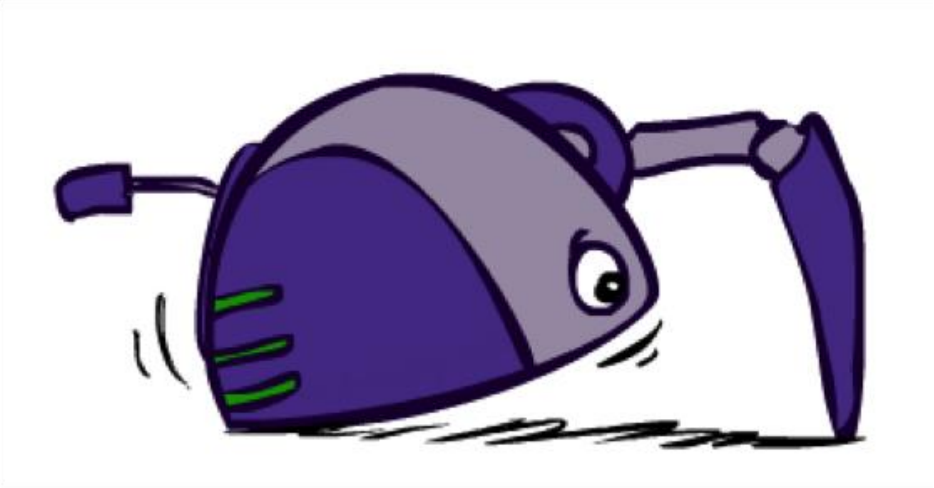
Reinforcement Learning



[These slides were created by Dan Klein, Pieter Abbeel, and Anca Dragan. http://ai.berkeley.edu.]

Reinforcement Learning

- We 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 $R(s,a,s')$
- Still looking for a policy $\pi(s)$



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

- Big 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

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate 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

Analogy: Expected Age

Goal: Compute expected age of UNR students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

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

Sample-Based Policy Evaluation?

- We want to improve our estimate of V (utility) by computing these averages:

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

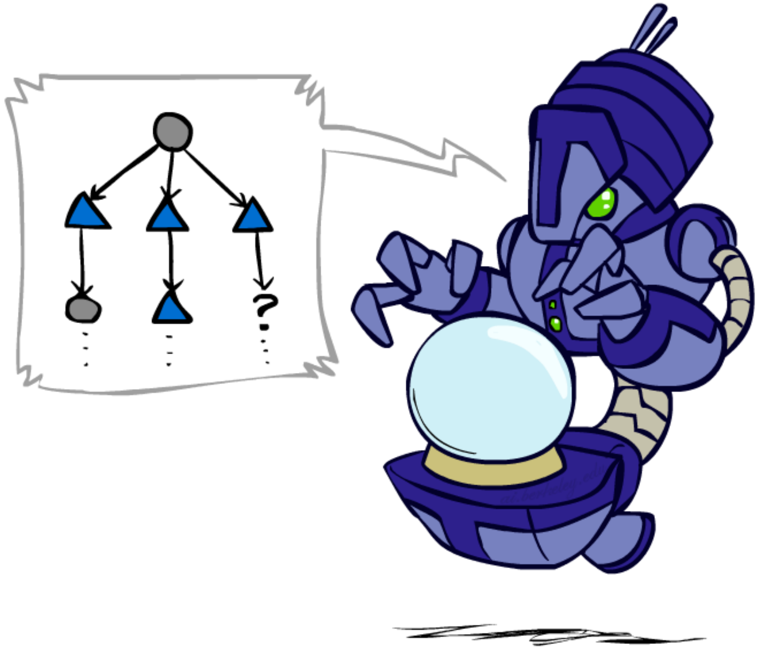
- Idea: Take samples of outcomes s′ (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s, \pi(s), s_2') + \gamma V_k^{\pi}(s_2')$$

$$\ldots$$

$$sample_n = R(s, \pi(s), s_n') + \gamma V_k^{\pi}(s_n')$$



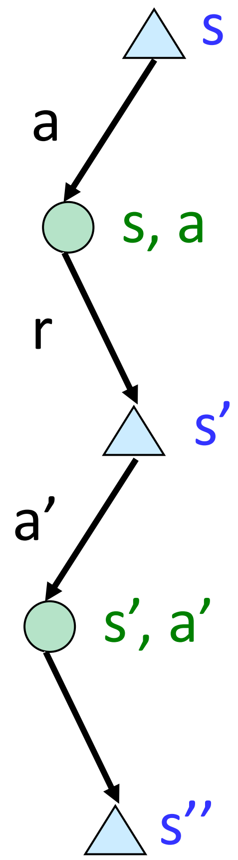
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$

Model-Free Learning

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

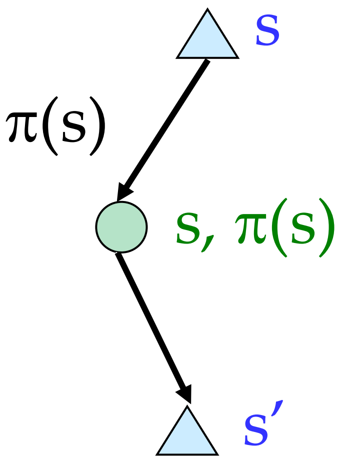
$$(s, a, r, s', a', r', s'', a'', r'', s'''' \ldots)$$

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



Temporal Difference Learning

- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

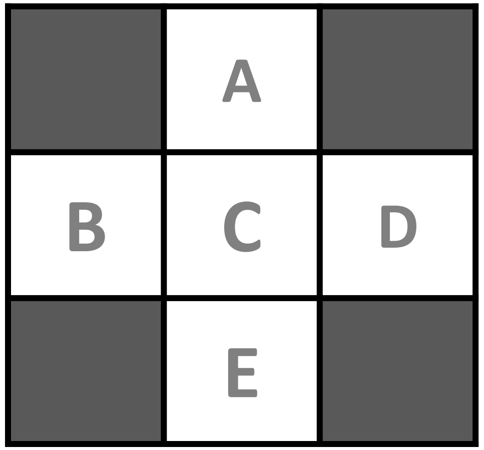


Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s): $$V^{\pi}(s) \leftarrow (1 - \alpha) V^{\pi}(s) + (\alpha) sample$$

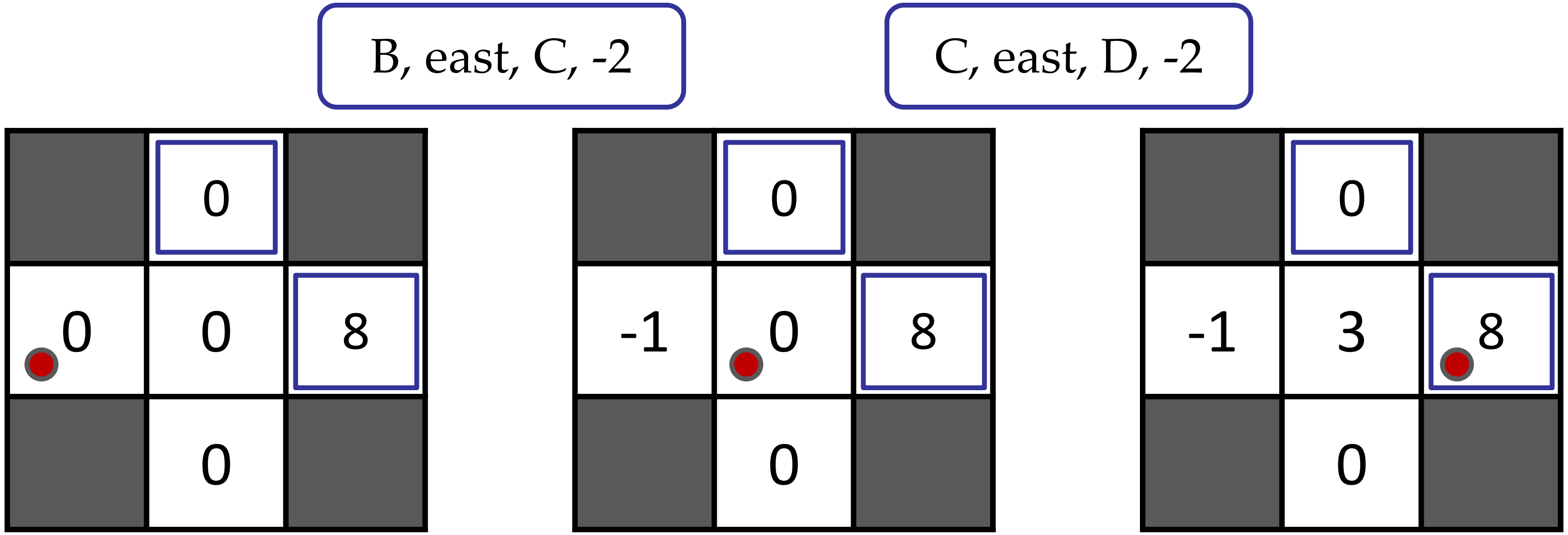
Example: Temporal Difference Learning

States



Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transitions



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

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-Learning: sample-based Q-value iteration

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

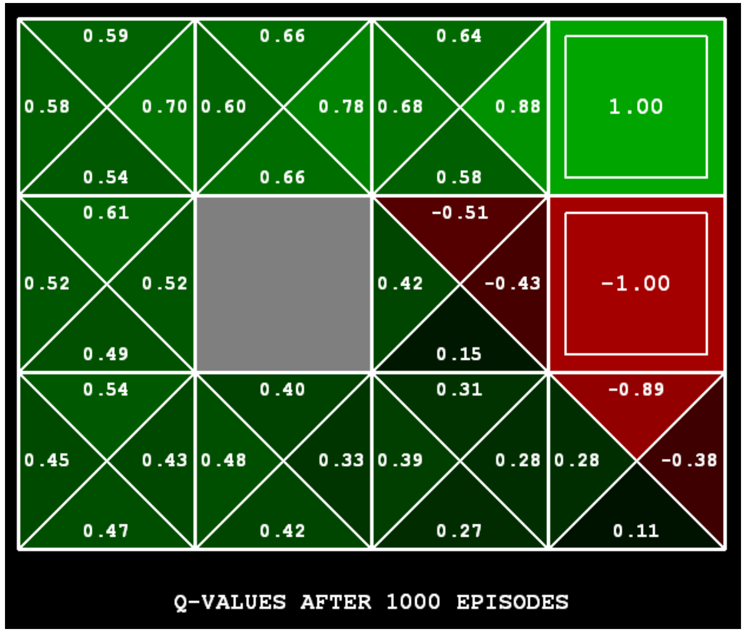
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s’,r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

no longer policy evaluation!

  - Incorporate the new estimate into a running average:

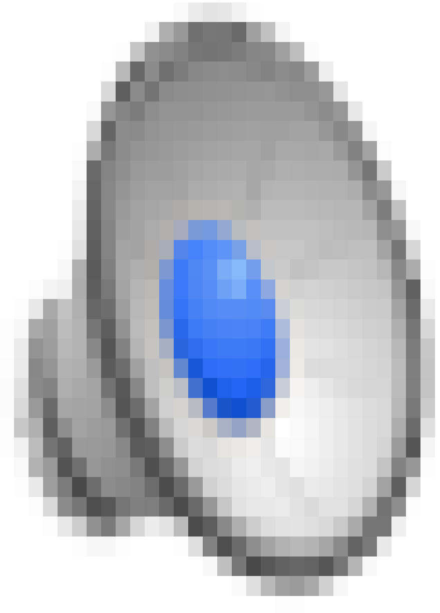
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

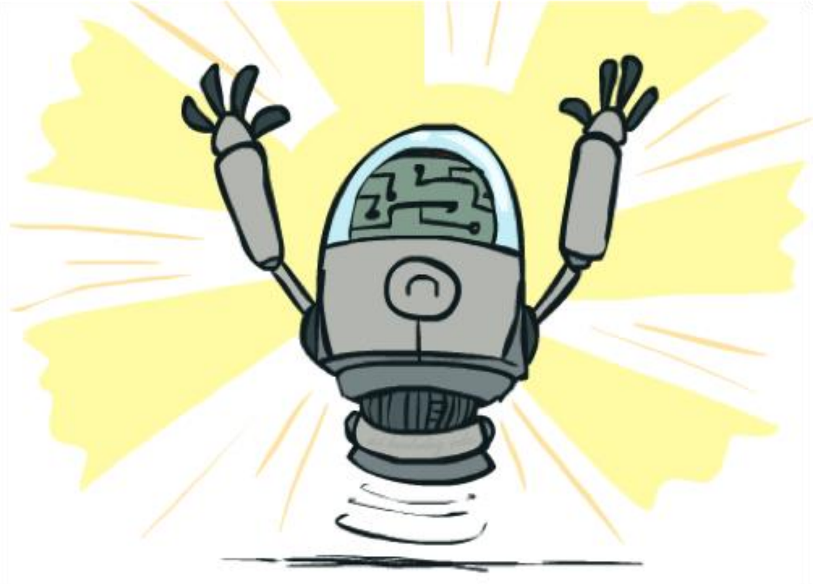


Q-Learning Properties

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

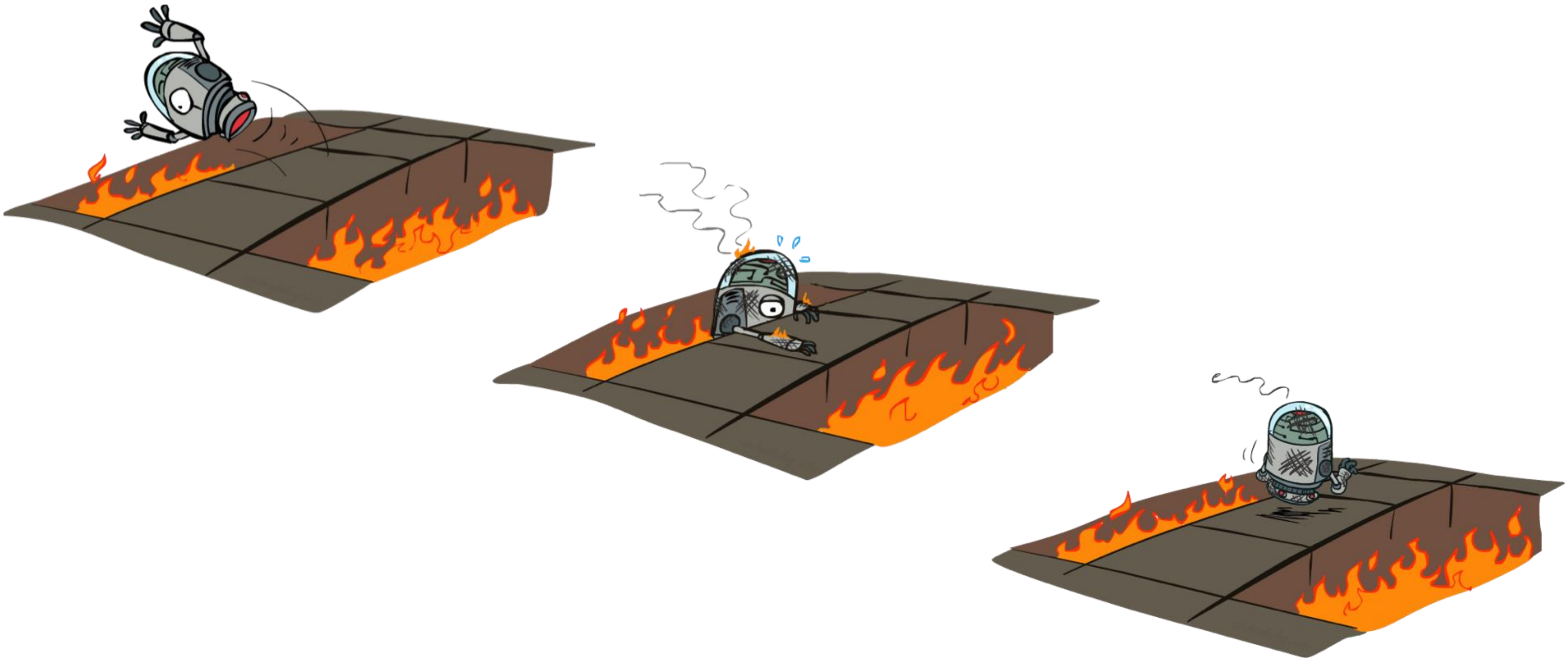
- This is called off-policy learning

- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly



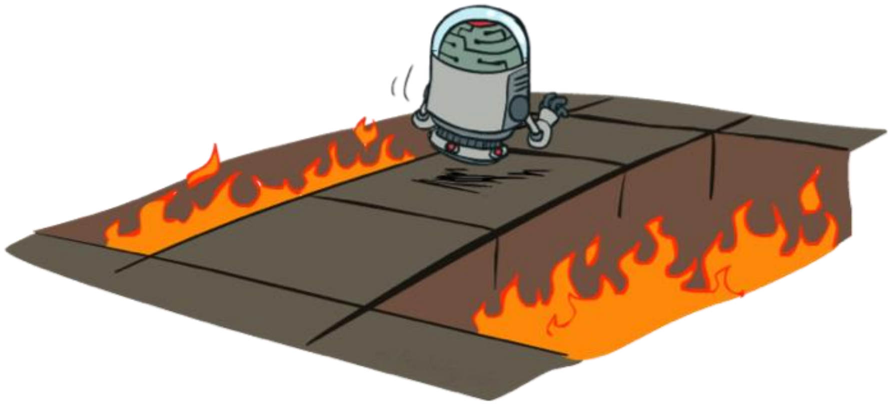
[Demo: Q-learning – auto – cliff grid (L11D1)]

Active Reinforcement Learning



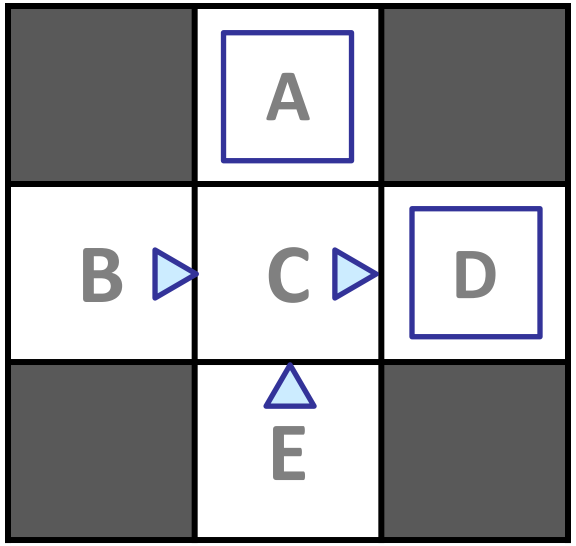
Model-Free Learning

- act according to current optimal (based on Q-Values)
- but also explore…



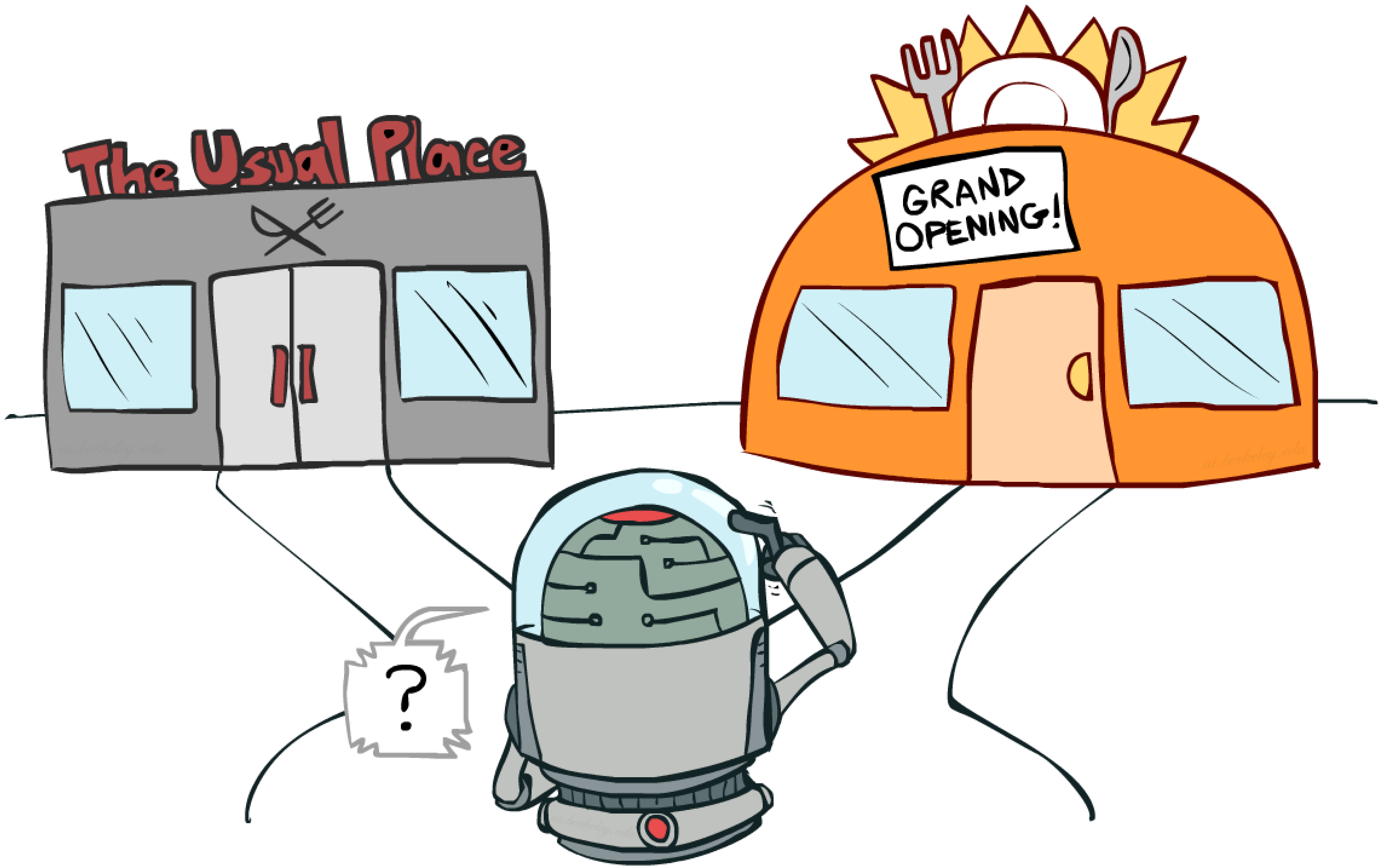
Model-Based Learning

~~Input Policy π~~

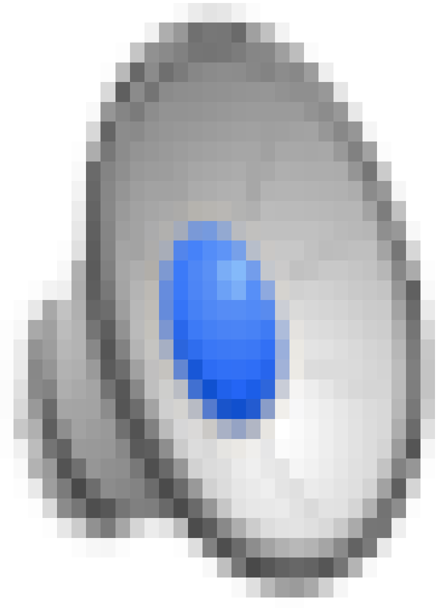


act according to current optimal
also explore!

Exploration vs. Exploitation



Video of Demo Q-learning – Manual Exploration – Bridge Grid

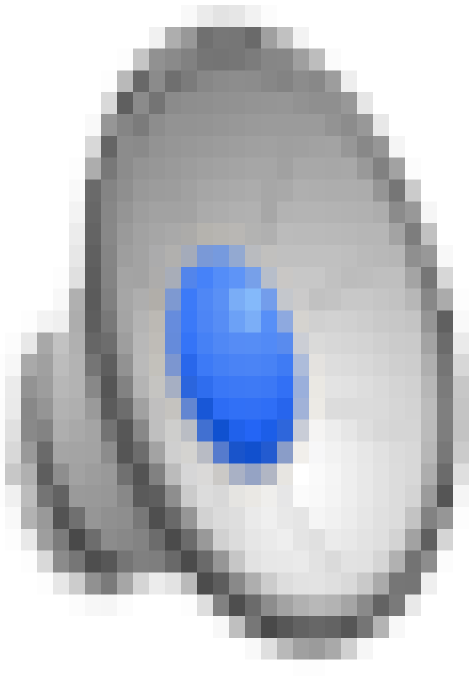


How to Explore?

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

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



Exploration Functions

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

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

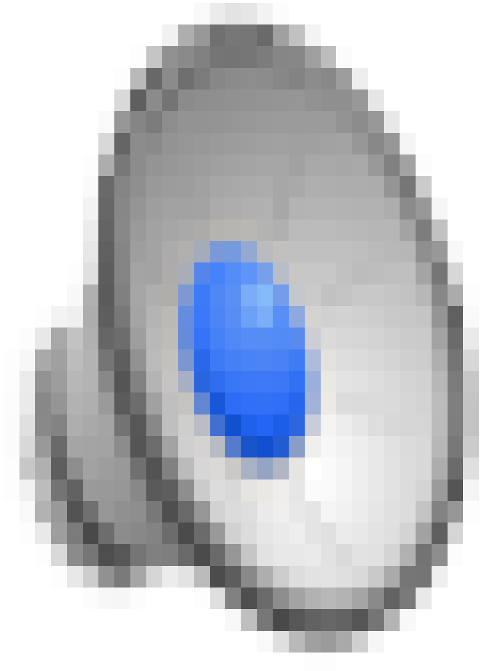
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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 well!

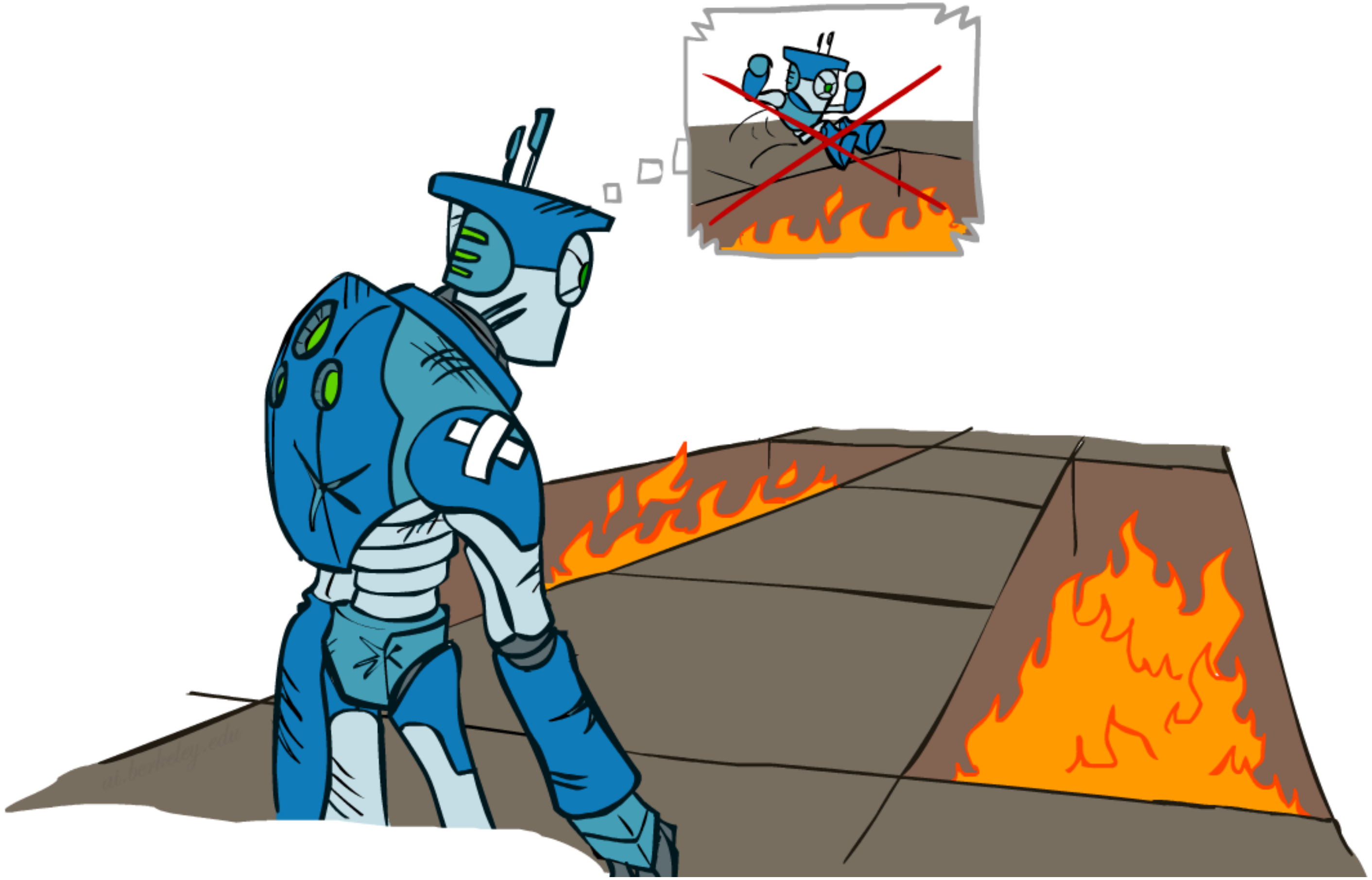
[Demo: exploration – Q-learning – crawler – exploration function (L11D4)]

Video of Demo Q-learning – Exploration Function – Crawler

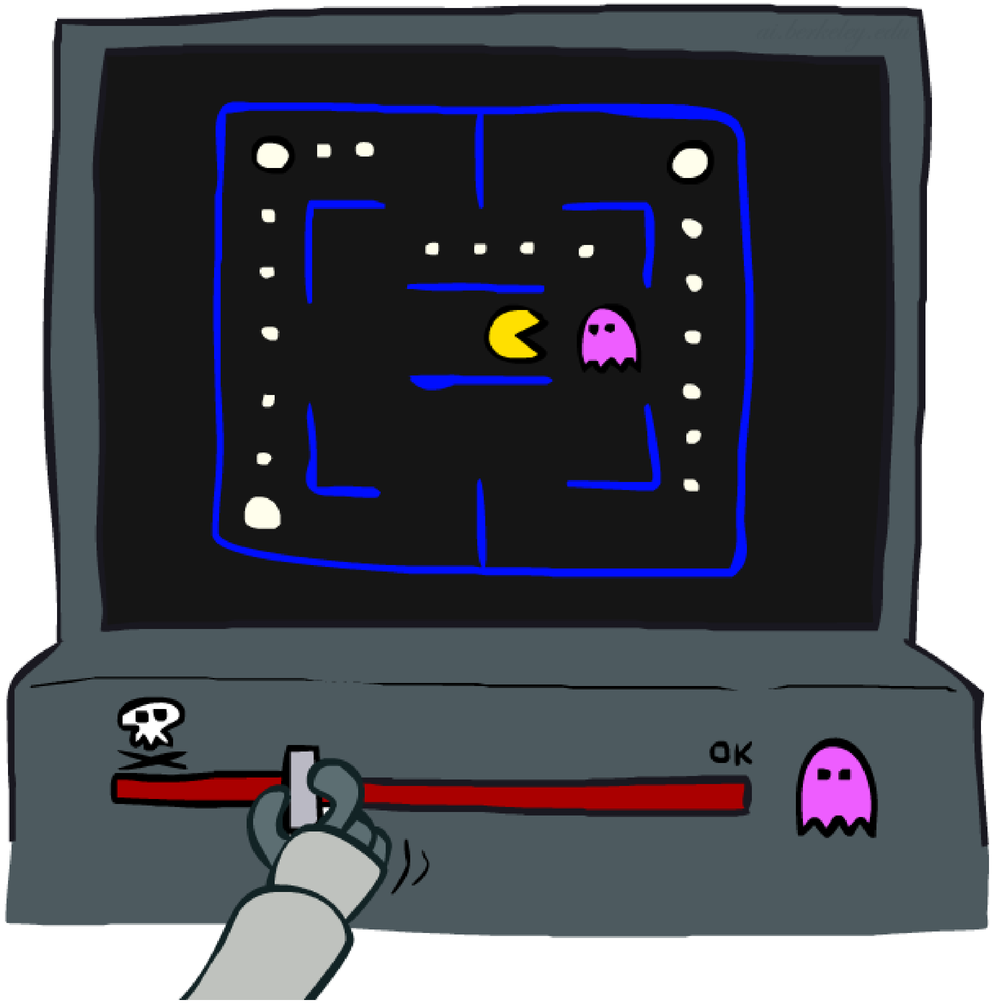


Regret

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



Approximate Q-Learning

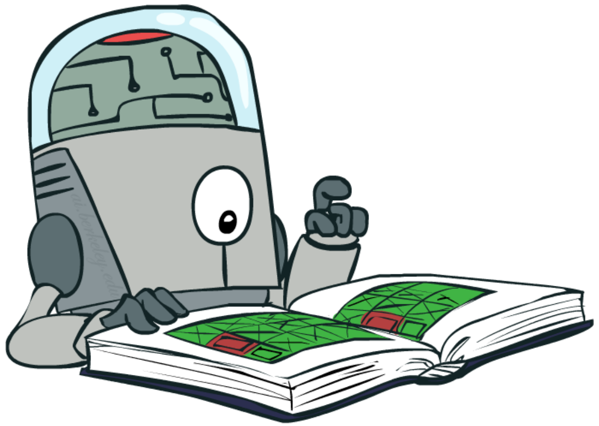


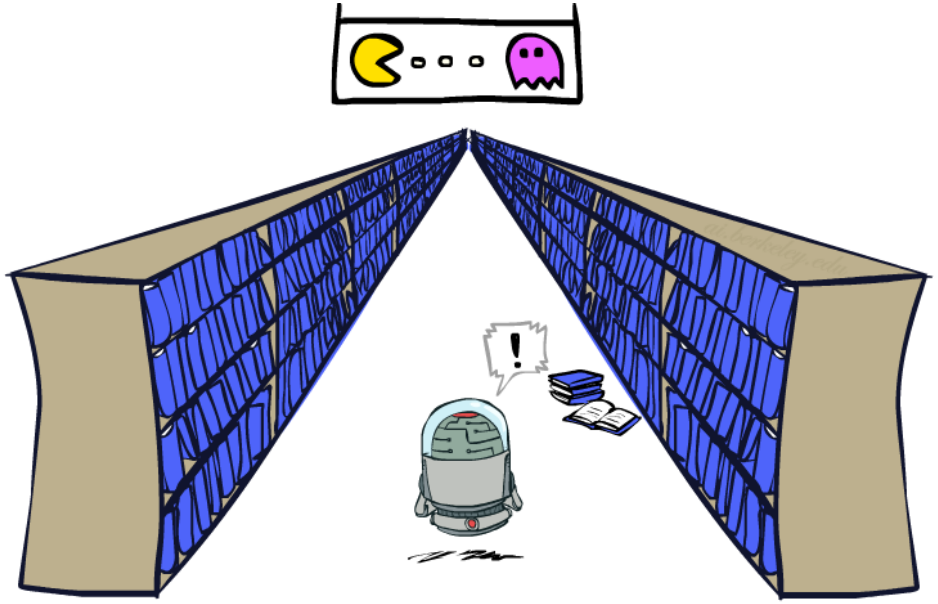
Generalizing Across States

- 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

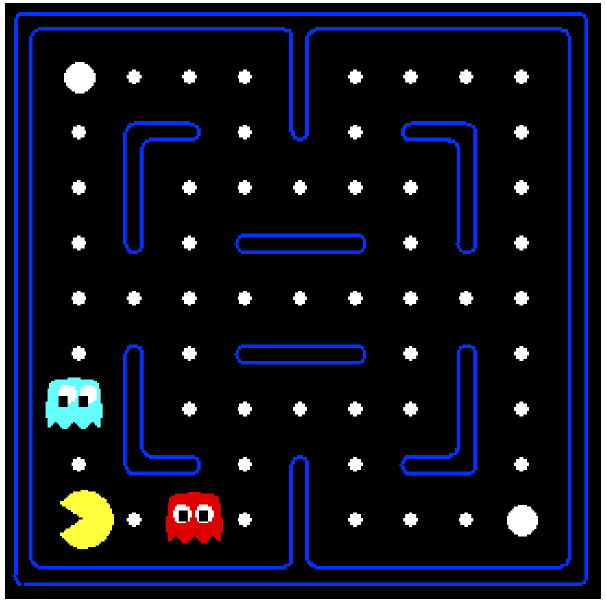




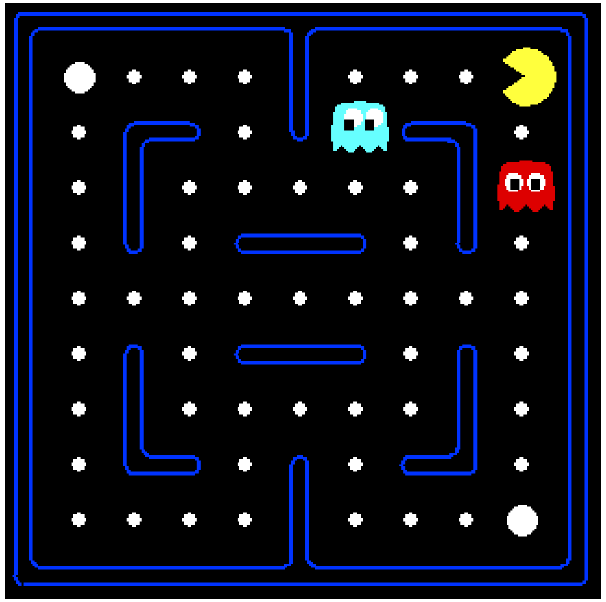
[demo – RL pacman]

Example: Pacman

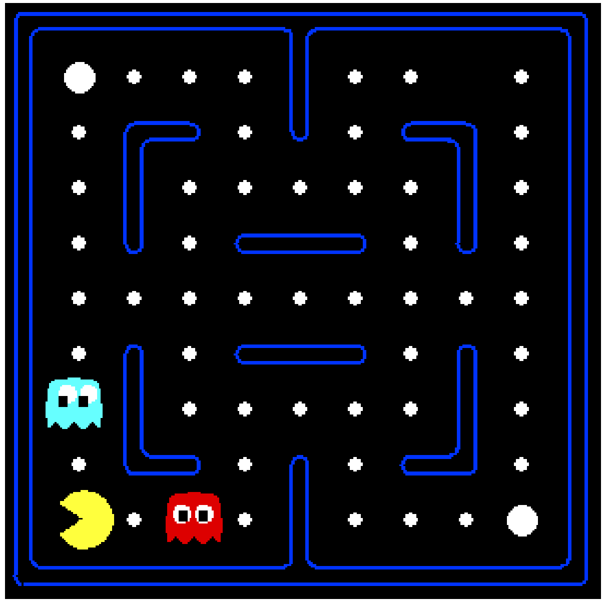
Let's say we discover through experience that this state is bad:



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

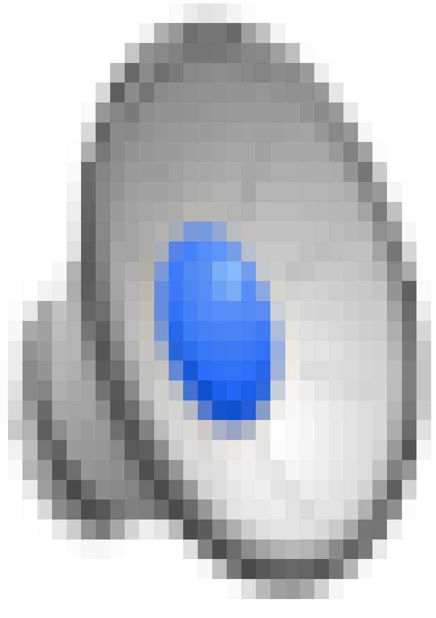


Or even this one!

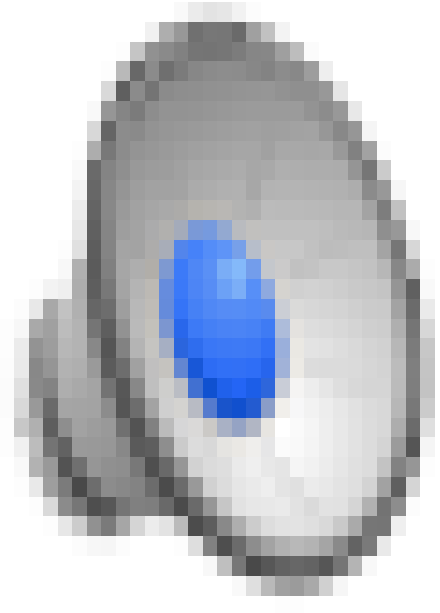


[Demo: Q-learning – pacman – tiny – watch all (L11D5)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D7)]

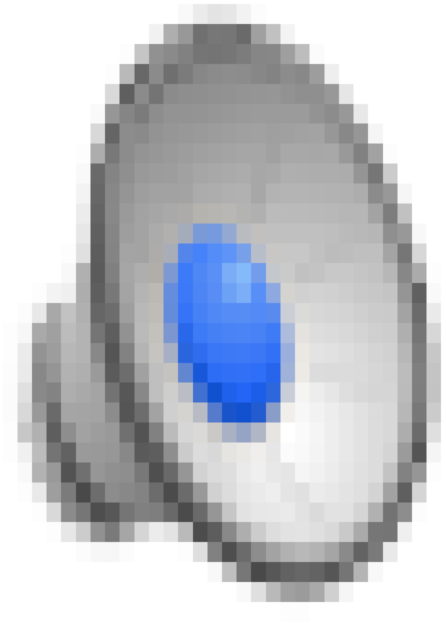
Video of Demo Q-Learning Pacman – Tiny – Watch All



Video of Demo Q-Learning Pacman – Tiny – Silent Train

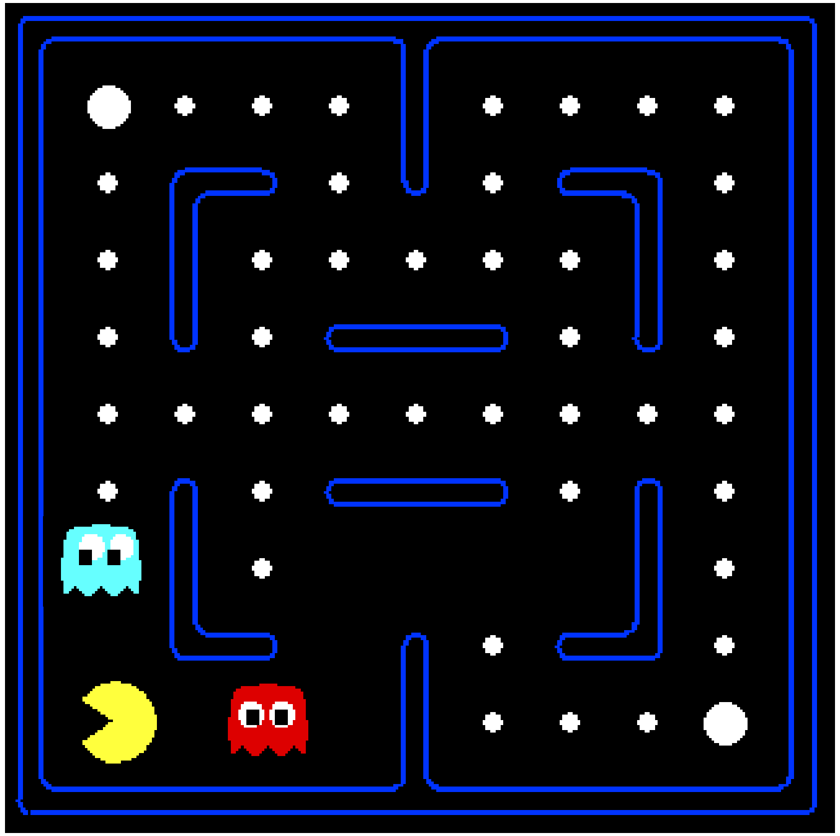


Video of Demo Q-Learning Pacman – Tricky – Watch All



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 / (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

- 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 actually be very different in value!

Error (least square)

What is the partial derivative of function

$$g(w_1, w_2) = \frac{1}{2}(y - (w_1 f_1(x) + w_2 f_2(x)))^2$$

w.r.t. w_1, i.e., $\frac{\partial g(w_1,w_2)}{\partial w_1}$?

Assume y is a constant and f is a known function that maps a vector in $\mathbb{R}^n$ to a scalar

A: $f_1(x)$

B: $y - (w_1 f_1(x) + w_2 f_2(x))$

C: $w_1 f_1(x) + w_2 f_2(x) - y$

D: $(w_1 f_1(x) + w_2 f_2(x) - y) f_1(x)$

Error (least square)

What is the partial derivative of function

$$g(w_1, w_2) = \frac{1}{2}(y - (w_1 f_1(x) + w_2 f_2(x)))^2$$

w.r.t. w_1, i.e., $\frac{\partial g(w_1,w_2)}{\partial w_1}$?

Assume y is a constant and f is a known function that maps a vector in $\mathbb{R}^n$ to a scalar

A: $f_1(x)$

B: $y - (w_1 f_1(x) + w_2 f_2(x))$

C: $w_1 f_1(x) + w_2 f_2(x) - y$

D: $(w_1 f_1(x) + w_2 f_2(x) - y) f_1(x)$

Let $h(w_1, w_2) = y - (w_1 f_1(x) + w_2 f_2(x))$

Then $g(w_1, w_2) = \frac{1}{2}\left(h(w_1, w_2)\right)^2$

$$\frac{\partial g(w_1,w_2)}{\partial w_1} = \frac{\partial g(w_1,w_2)}{\partial h(w_1,w_2)} \frac{\partial h(w_1,w_2)}{\partial w_1}$$

$$= \frac{1}{2} \times 2 \times h(w_1, w_2) \times (-f_1(x))$$

$$= -h(w_1, w_2) f_1(x)$$

Updating a linear value function

Original Q-learning: Update Q values directly (stored in a table)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\underbrace{\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Latest sample}} - \underbrace{Q(s,a)}_{\text{Previous estimate}}}_{\text{Difference}}]$$

Can be viewed as trying to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) - \alpha \nabla Error \qquad Error = \frac{1}{2}\left(sample - Q(s,a)\right)^2$$

Approximate Q-Learning with Linear Q-Value Function:

$$Q_{\boldsymbol{w}}(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$$

Update weights to reduce prediction error at s, a:

$$w_i \leftarrow w_i - \alpha \frac{\partial Error(w_1, w_2, \ldots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2}\left(sample - Q_w(s,a)\right)^2$$

Updating a linear value function

$$Q_w(s,a) = w_1 f_1(s,a) \; + \; \dots + \; w_n f_n(s,a)$$

$$w_i \leftarrow w_i - \alpha \frac{\partial Error(w_1, w_2, \dots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2}\left(sample - Q_w(s,a)\right)^2$$

$$\begin{aligned} \frac{\partial Error(w)}{\partial w_i} &= (Q_w(s,a) - sample)\frac{\partial Q_w(s,a)}{\partial w_i} \\ &= (Q_w(s,a) - sample) f_i(s,a) \end{aligned}$$

Final Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$\boxed{w_i} \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)\boxed{f_i(s,a)}$$

Original Q-Learning Update Rule:

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

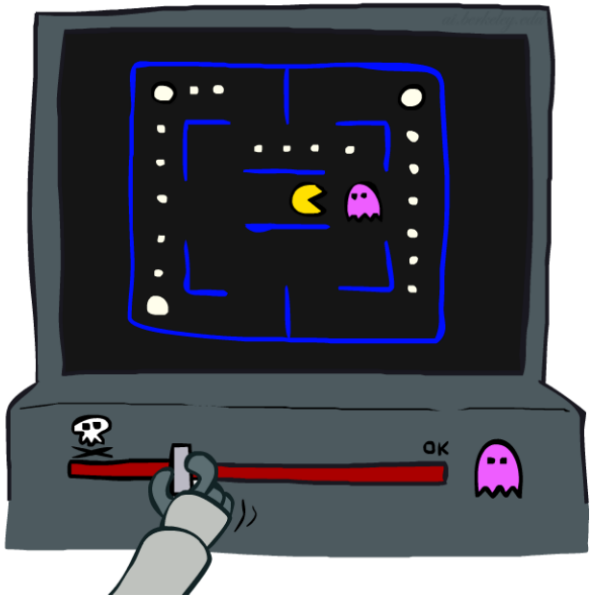
Approximate Q-Learning

Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)\right) f_i(s, a)$$

Qualitative justification:

- Pleasant surprise: increase weights on +valued features, decrease on – ones
  - As a result, Q_w increased for states with the same (similar) features too. Will now prefer all states with that state's features.
- Unpleasant surprise: decrease weights on +valued features, increase on – ones
  - Disprefer all states with that state's features



Approximate Q-Learning

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

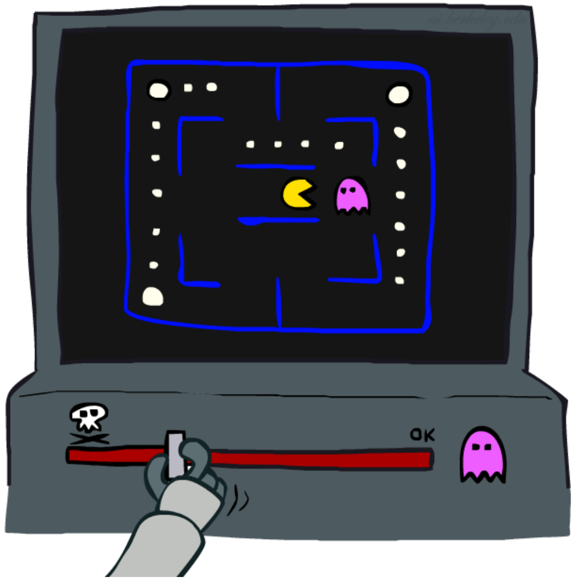
- Q-learning with linear Q-functions:

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

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

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

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



- 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: least squares

What if non-linear value function

Update Rule for Q-Learning:

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

Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right) f_i(s,a)$$

Update Rule for Approximate Q-Learning with differentiable Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right) \frac{\partial Q_w(s,a)}{\partial w_i}$$

If $Q_w(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$

$$\frac{\partial Q_w(s,a)}{\partial w_i} = f_1(s,a)$$

What if non-linear value function

Update Rule for Approximate Q-Learning with Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)\frac{\partial Q_w(s,a)}{\partial w_i}$$

Why?

$$w_i \leftarrow w_i - \alpha\frac{\partial Error(w_1, w_2, \ldots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2}\left(sample - Q_w(s,a)\right)^2$$

$$\frac{\partial Error(w)}{\partial w_i} = (Q_w(s,a) - sample)\frac{\partial Q_w(s,a)}{\partial w_i}$$

$$w_i - \alpha\frac{\partial Error(w_1, w_2, \ldots, w_n)}{\partial w_i} = w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)\frac{\partial Q_w(s,a)}{\partial w_i}$$

What if non-linear value function

Update Rule for Approximate Q-Learning with Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)\frac{\partial Q_w(s,a)}{\partial w_i}$$

Example: $Q_w(s,a) = \exp(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a))$

$$\begin{aligned}\frac{\partial Q_w(s,a)}{\partial w_i} &= \exp\big(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)\big)\, f_i(s,a) \\ &= Q_w(s,a) f_i(s,a)\end{aligned}$$

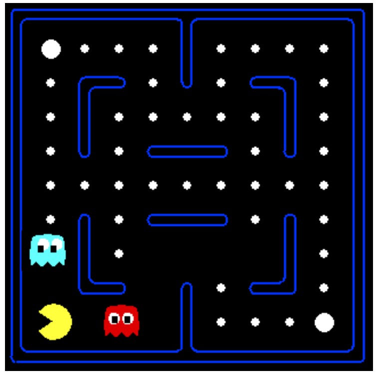
Update Rule:

$$w_i \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right) Q_w(s,a) f_i(s,a)$$

Example: Q-Pacman

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

s

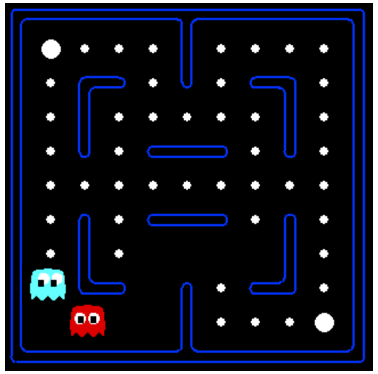


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

difference $= -501$

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

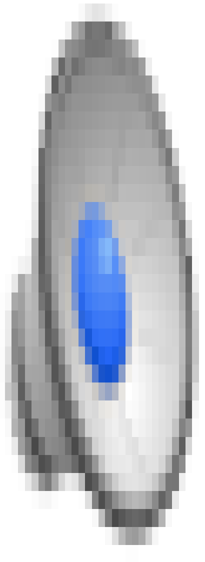
$w_{GST} \leftarrow -1.0 + \alpha \, [-501] \, 1.0$

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

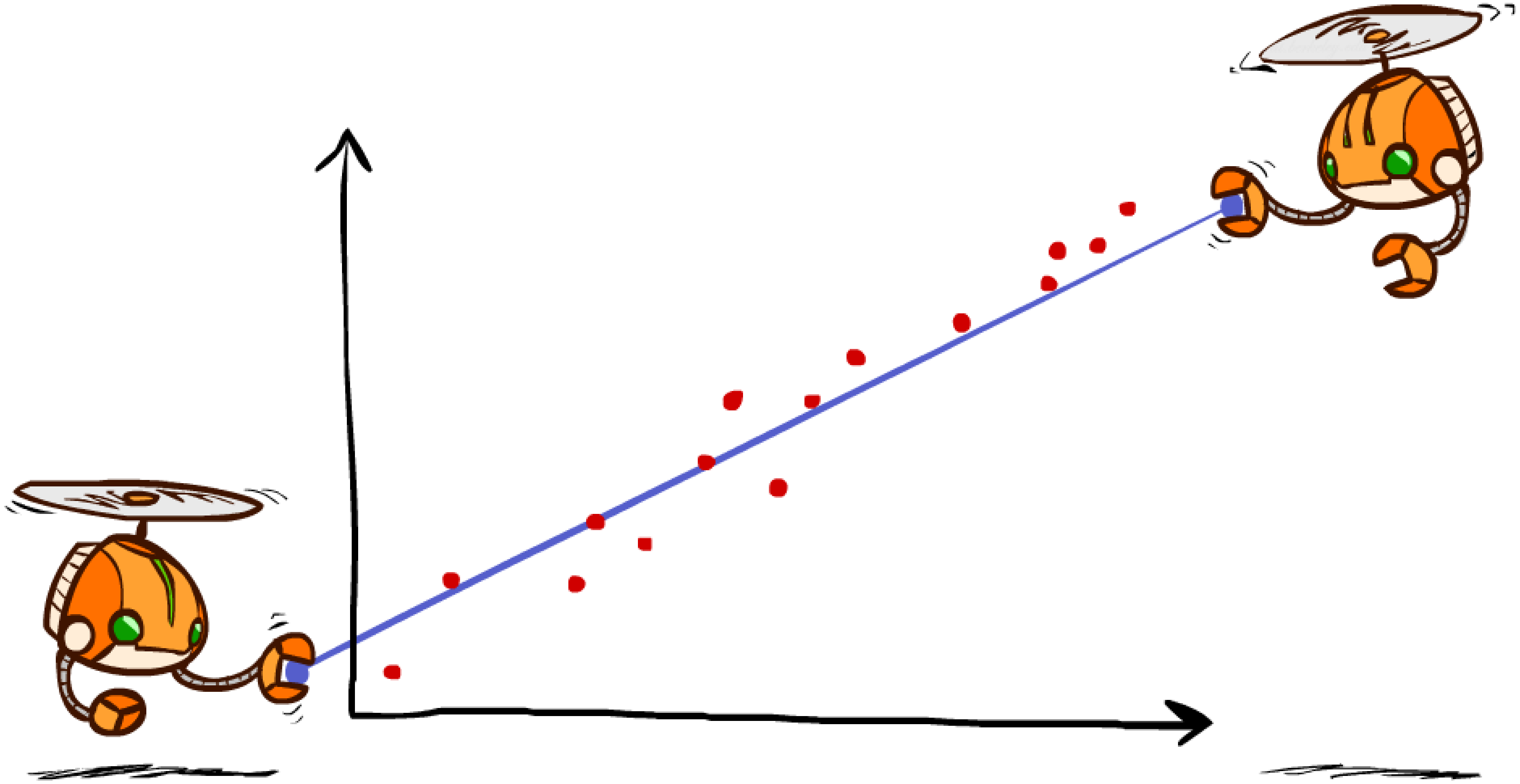
[Demo: approximate Q-learning pacman (L11D10)]

Video of Demo Approximate Q-Learning -- Pacman

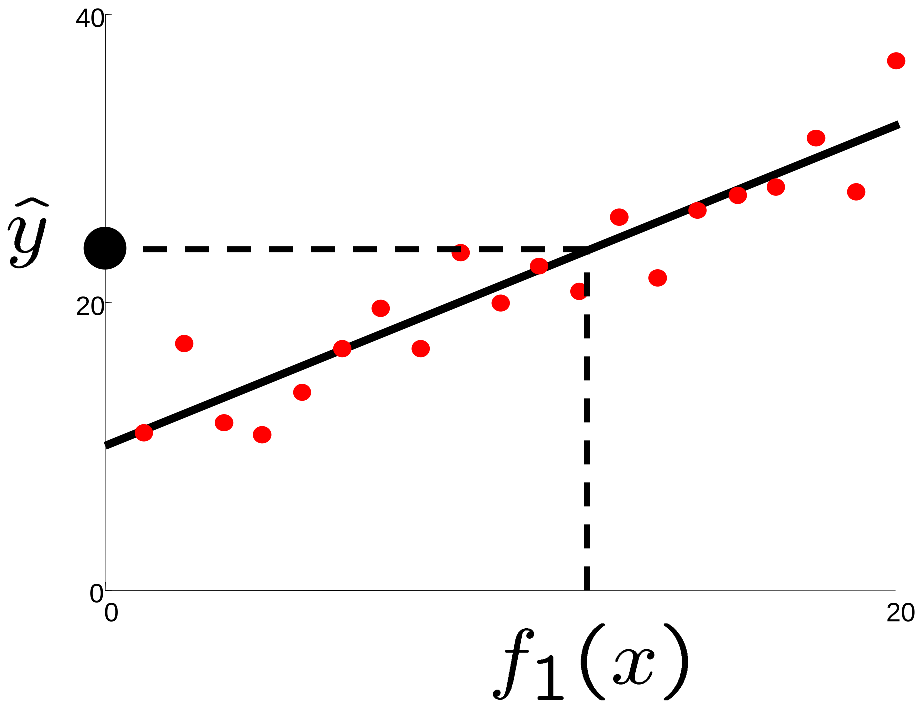
DeepMind Atari (©Two Minute Lectures) approximate Q-learning with neural nets

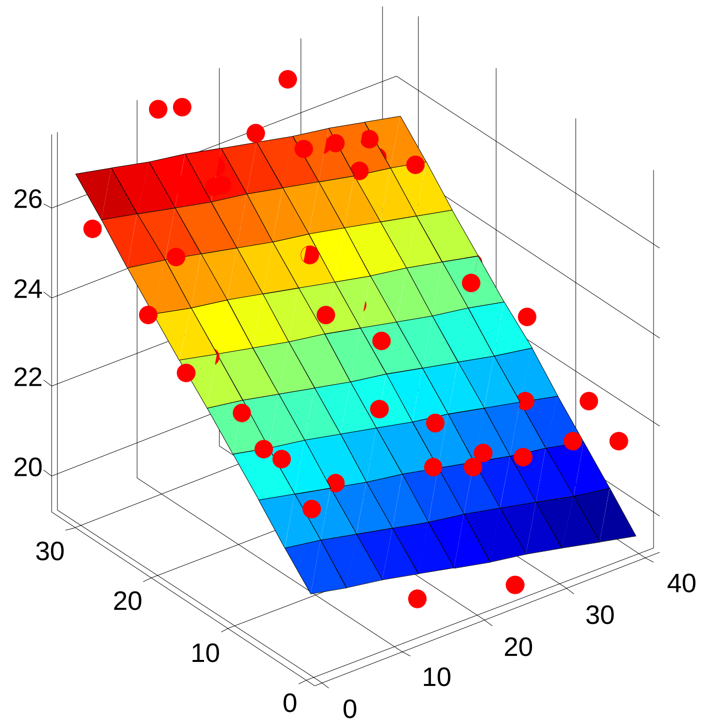


Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

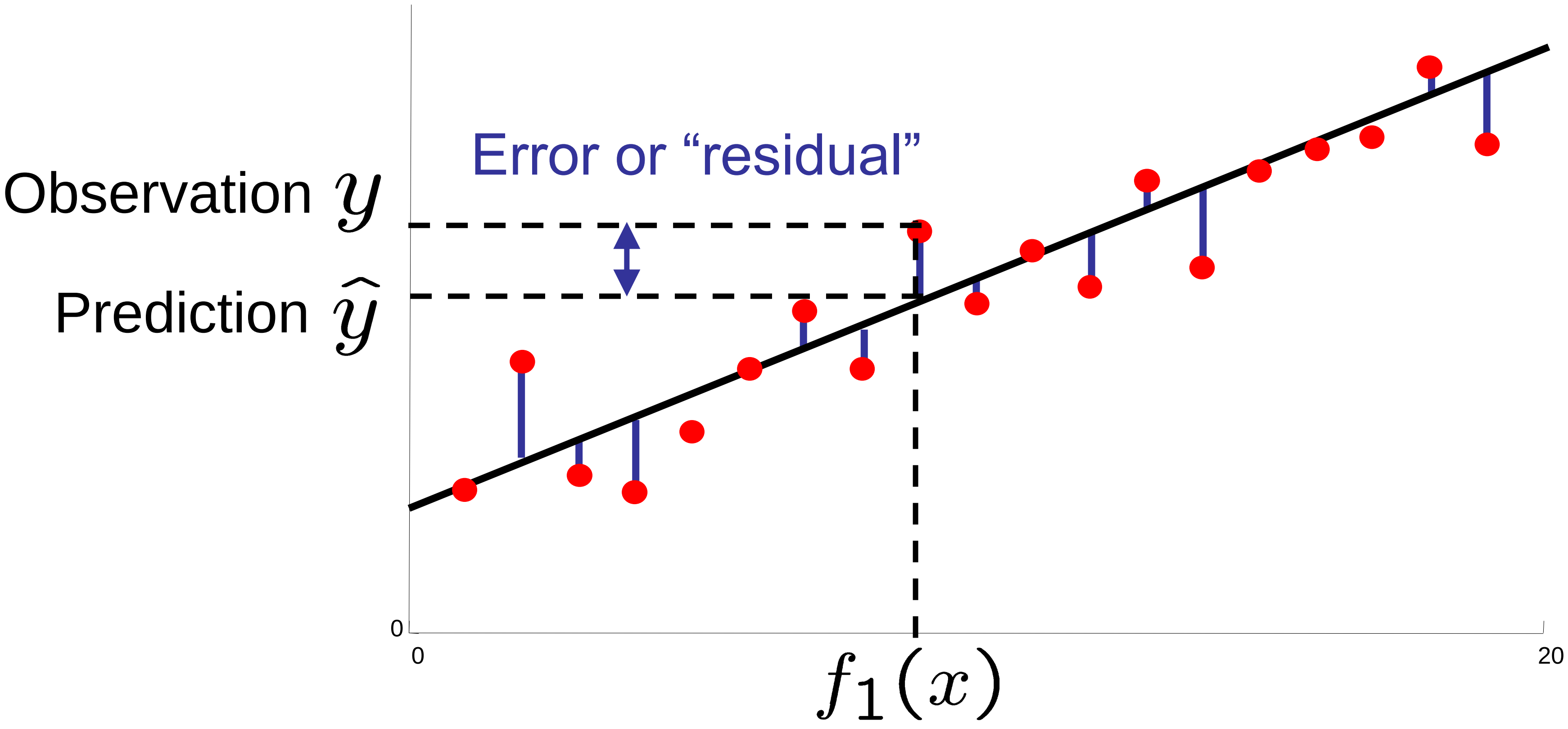
$$\widehat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



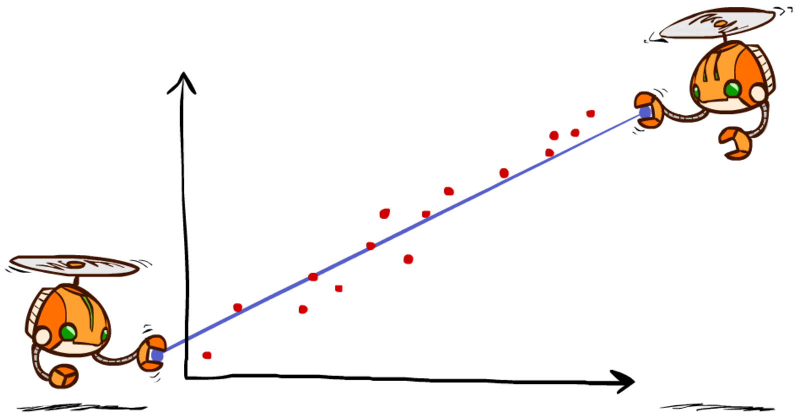
Minimizing Error

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

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

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

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

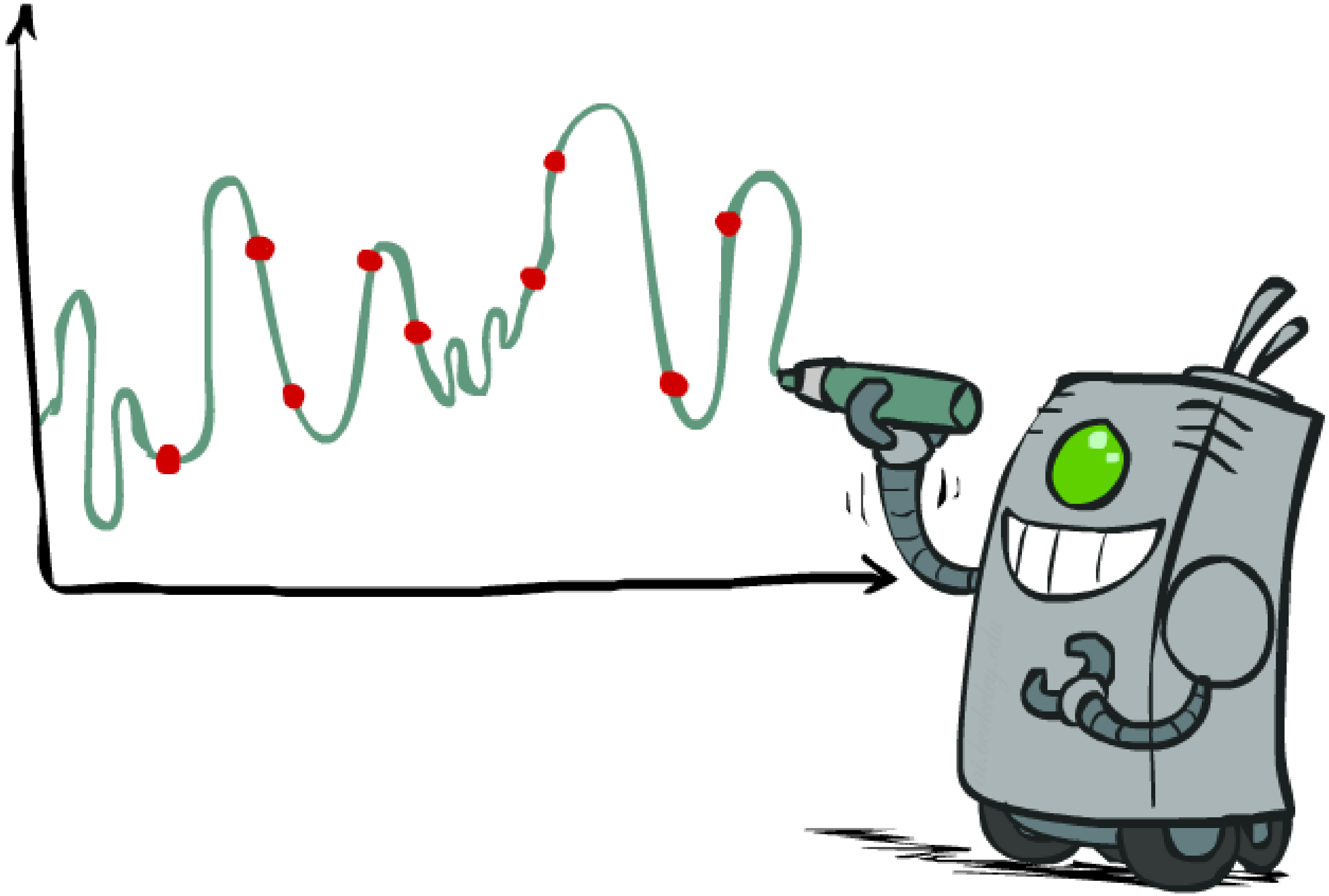


Approximate q update explained:

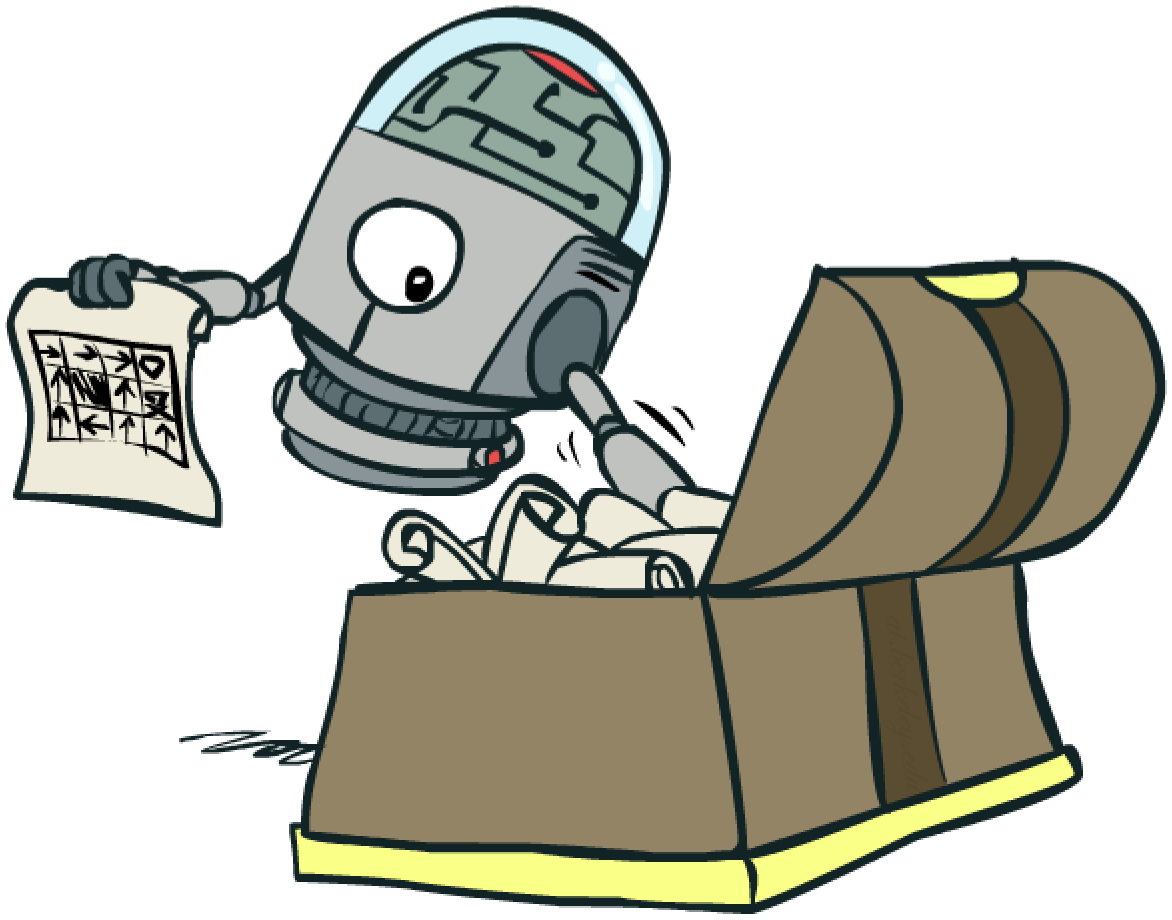
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help



Policy Search



Policy Search

- 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…

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

**use features to generalize*

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

Unknown MDP: Model-Free

**use features to generalize*

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