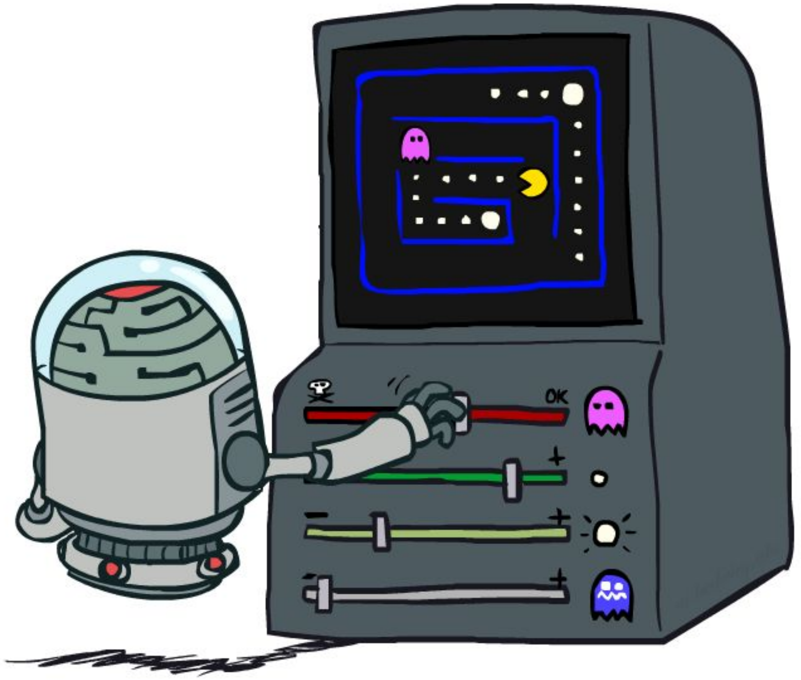
AI: Representation and Problem Solving

Reinforcement Learning II



Instructors: Tuomas Sandholm and Vincent Conitzer

Slide credits: CMU AI and http://ai.berkeley.edu

Overview: MDPs and Reinforcement Learning

Known MDP: Offline Solution

Value Iteration / Policy Iteration

Unknown MDP: Online Learning

Estimate MDP T(s,a,s') and R(s,a,s') from samples of environment

Model-Free

Passive Reinforcement Learning

- Direct Evaluation (simple)
- TD Learning

Active Reinforcement Learning

- Q-Learning

Online Learning

Model-free Learning

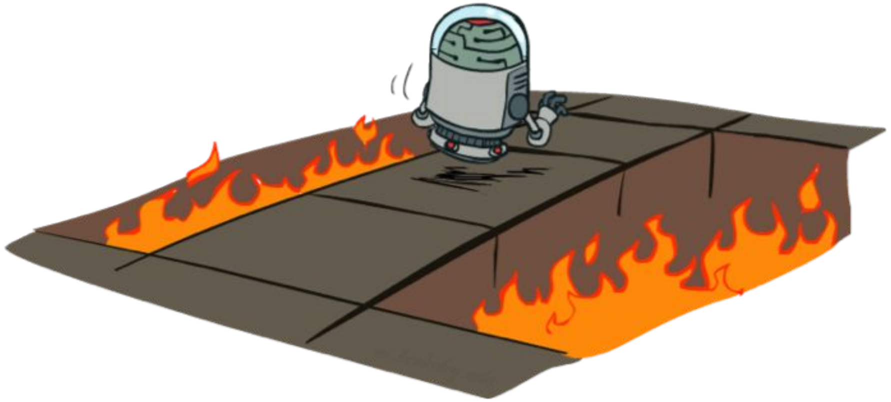
Active Reinforcement Learning

Q-learning

Active Reinforcement Learning

Full reinforcement learning: optimal policies (like value iteration)

- You don't know the transitions T(s,a,s')
- You don't know the rewards R(s,a,s')
- You choose the actions now
- Goal: learn the optimal policy / values



In this case:

- Learner makes choices!
- Fundamental tradeoff: exploration vs. exploitation
- This is NOT offline planning! You actually take actions in the world and find out what happens…

Recall: Q-Value Iteration

Value iteration: find successive (depth-limited) values

- Start with $V_0(s) = 0$, which we know is right
- Given V_k, calculate the depth k+1 values for all states:

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

But Q-values are more useful, so compute them instead

- Start with $Q_0(s,a) = 0$, which we know is right
- Given Q_k, calculate the depth k+1 q-values for all q-states:

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

Q-Learning

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

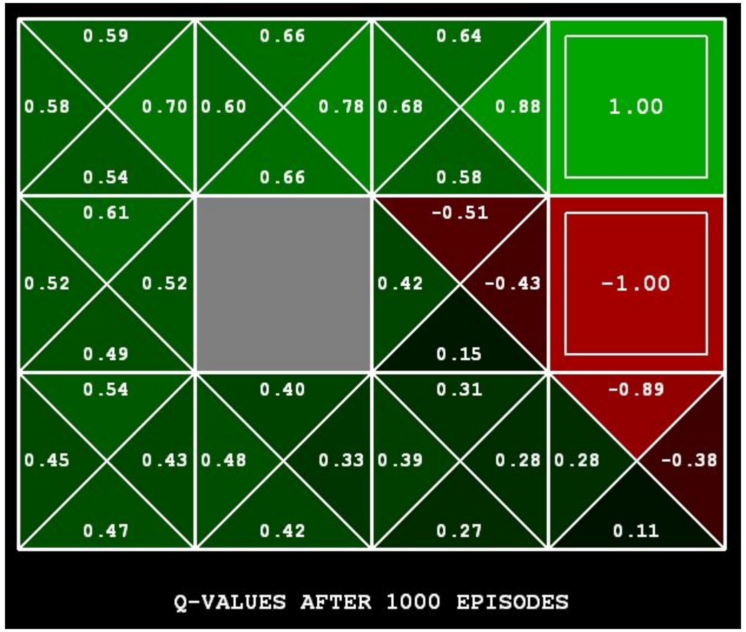
Instead, learn Q(s,a) values as you go

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

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

- Incorporate the new estimate into a running average:

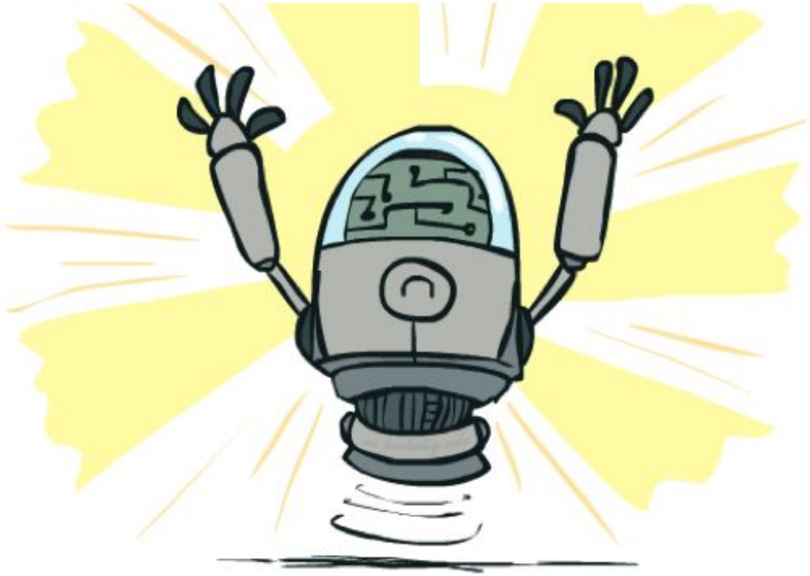
$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) [sample]$$



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
- Basically, in the limit, it doesn't matter how you select actions (!)

Review: MDP/RL Notation

Standard expectimax: $$V(s) = \max_a \sum_{s'} P(s'|s,a) V(s')$$

Bellman equations: $$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Value iteration: $$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$$

Q-iteration: $$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$$

Policy extraction: $$\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$$

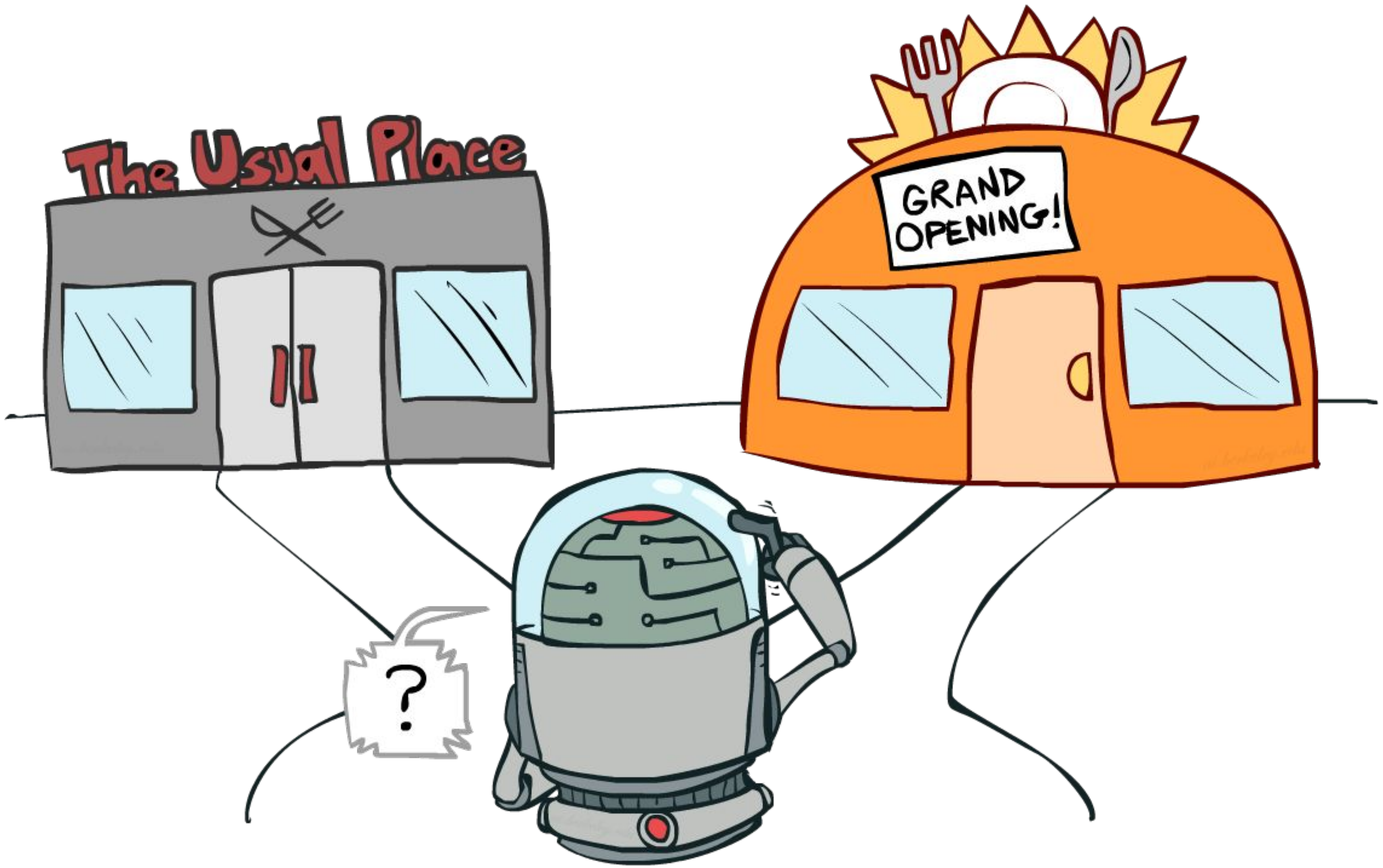
Policy evaluation: $$V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')], \quad \forall\, s$$

Policy improvement: $$\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$$

Value (TD) learning: $$V^{\pi}(s) \;=\; V^{\pi}(s) \;+ \alpha\,[r + \gamma\, V^{\pi}(s') - \; V^{\pi}(s)]$$

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

Exploration vs. Exploitation



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

Poll

Exploration should be

A) Optimistic

B) Pessimistic

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!

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.

$$f(u,n) = u + k/(n+1)$$

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

Modified Q-Update: $Q(s,a) = Q(s,a) + \alpha \, [r + \gamma \max_{a'} f(Q(s',a'), N(s',a')) \; - Q(s,a)]$

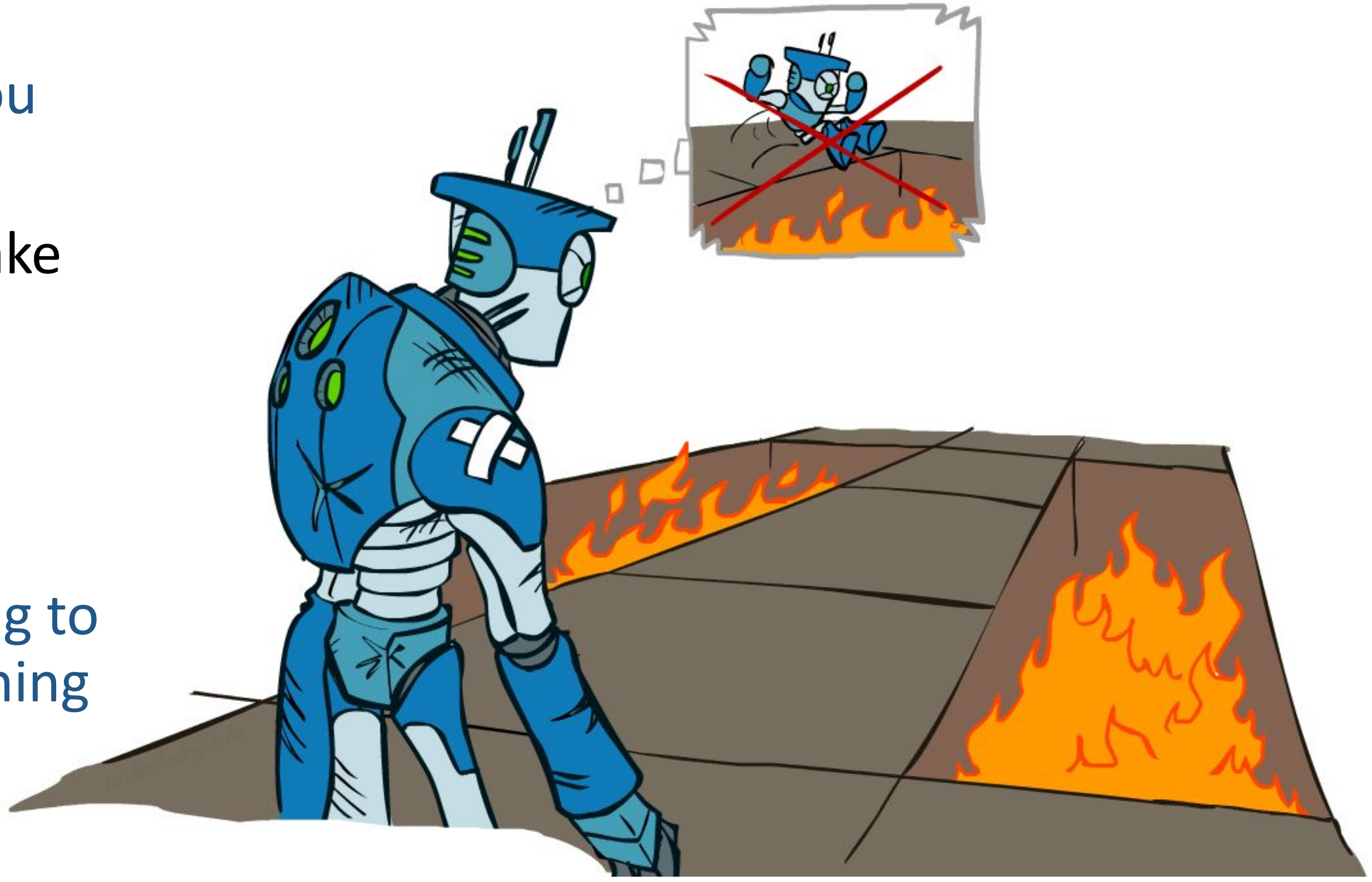
Regret

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

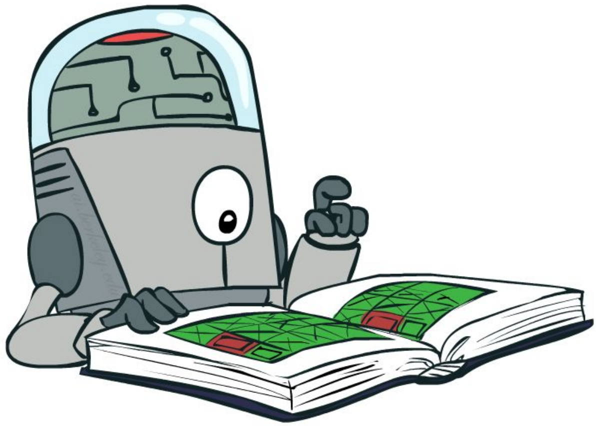
Basic Q-Learning keeps a table of all Q-values

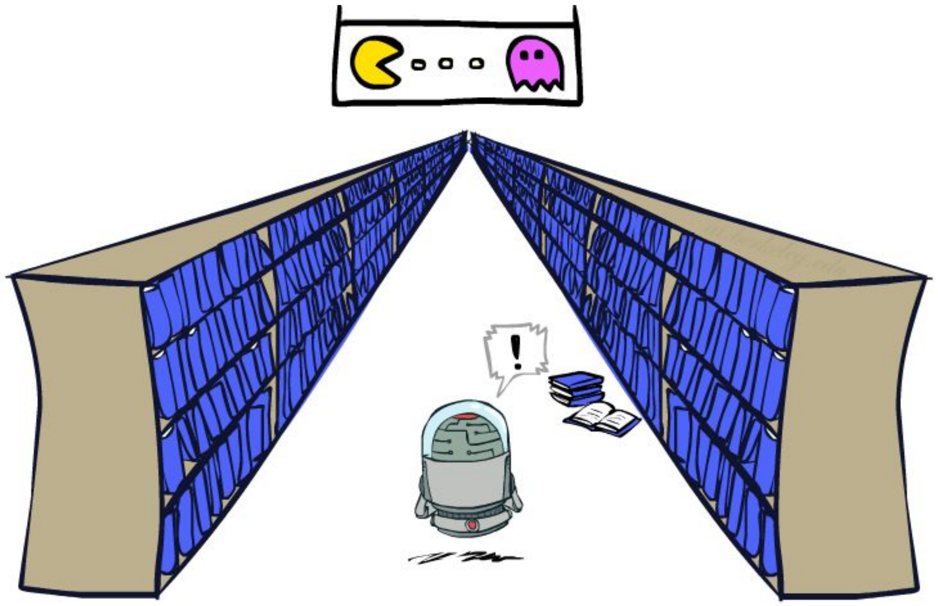
In realistic situations, we cannot possibly learn about every single state!

- Too many states to visit them all in training
- Too many states to hold the Q-tables in memory

Instead, we want to ***generalize***:

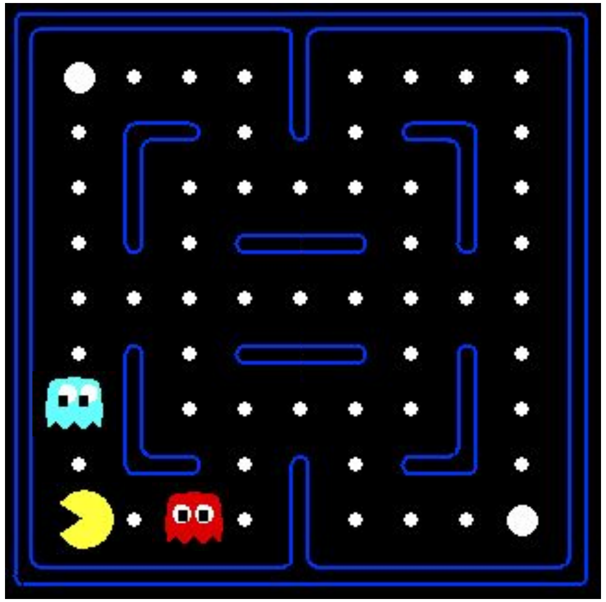
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
- (This is a fundamental idea in many types of machine learning)



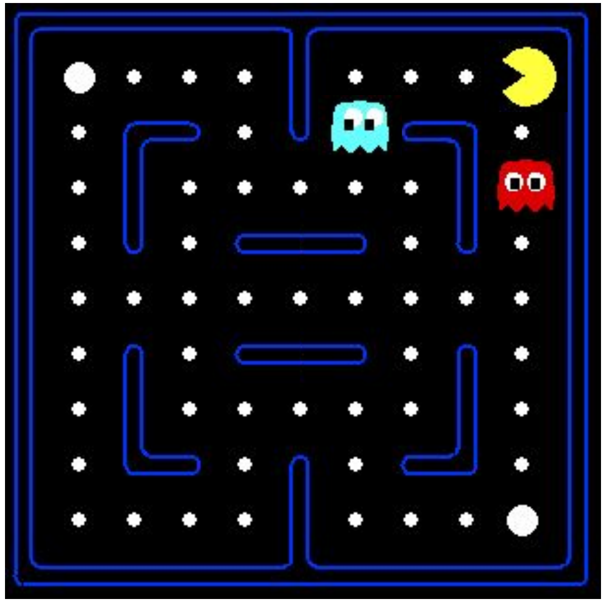


Example: Pacman

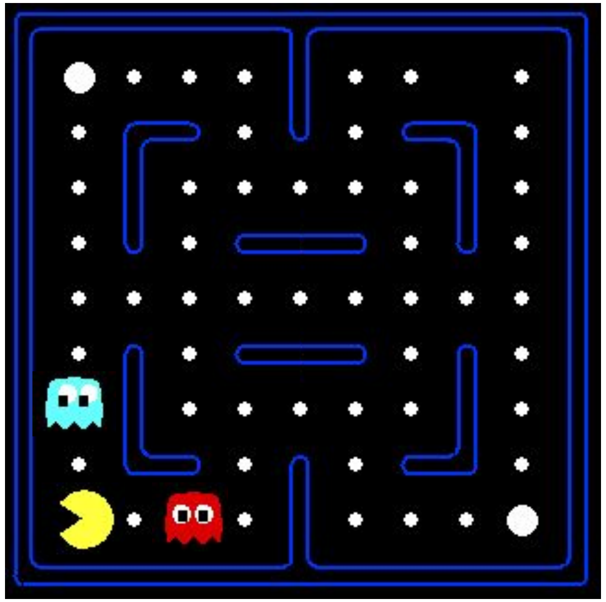
Let’s say we discover through experience that this state is bad:



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



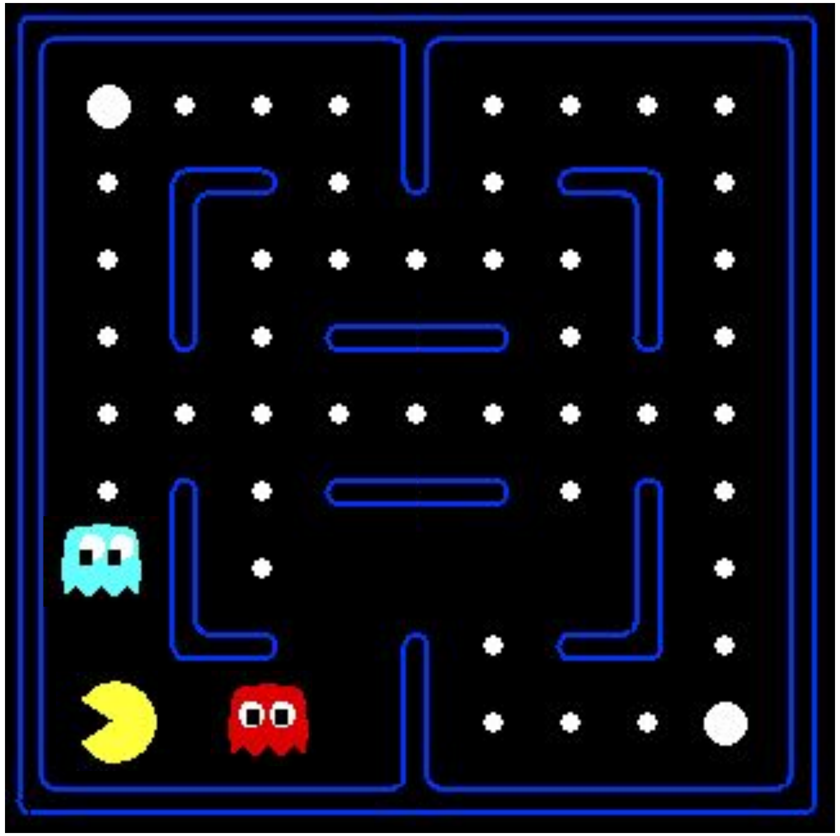
Or even this one!



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:

- $V_{\mathbf{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- $Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$

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

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

Updating a linear value function

Original Q learning rule tries to reduce prediction error at s, a:

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

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$

  $= w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, f_i(s,a)$

Quick Calculus Quiz

$$Error(w) = \frac{1}{2}\left(y - wf(x)\right)^2$$

What is $\frac{dError}{dw}$?

Last time

$$Error(x) = \frac{1}{2}(y - x)^2$$

$$\frac{dError}{dx} = -(y - x)$$

Updating a linear value function

Original Q learning rule tries to reduce prediction error at s, a:

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

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_{\mathbf{w}}(s,a)/\partial w_i$

  $= w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, f_i(s,a)$

$$Q_{\mathbf{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a)$$

$$\frac{\partial Q}{\partial w_2} =$$

$$Error(w) = \frac{1}{2}\left(y - wf(x)\right)^2$$

$$\frac{dError}{dw} = -\left(y - wf(x)\right)f(x)$$

Updating a linear value function

Original Q learning rule tries to reduce prediction error at s, a:

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

Instead, we update the weights to try to reduce the error at s, a:

- $w_i \leftarrow w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]\ \partial Q_{\mathbf{w}}(s,a)/\partial w_i$

  $= w_i + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]\ f_i(s,a)$

Qualitative justification:

- Pleasant surprise: increase weights on +ve features, decrease on –ve ones
- Unpleasant surprise: decrease weights on +ve features, increase on –ve ones

Approximate Q-Learning

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

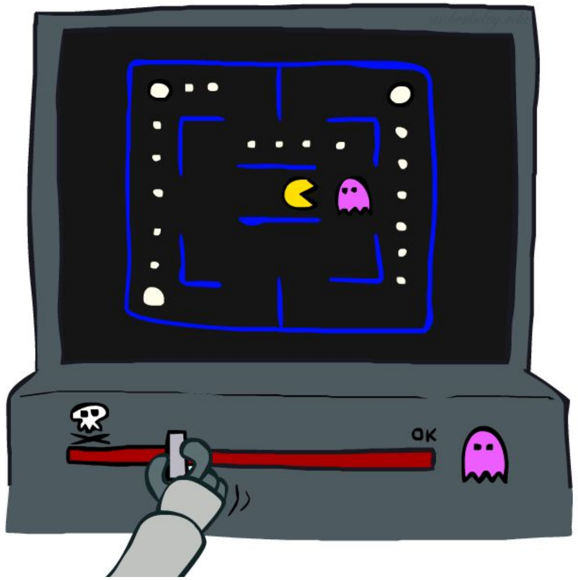
Q-learning with linear Q-functions:

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

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

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

$$w_i \leftarrow w_i + \alpha \,[\text{difference}]\, f_i(s,a)$$ 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: online least squares

Example: Q-Pacman

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

s

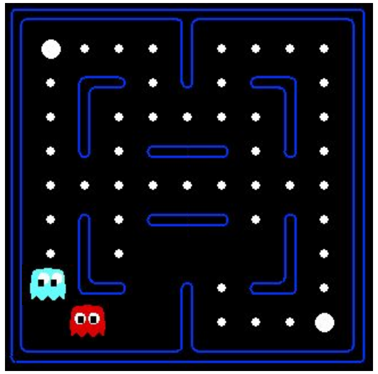
$$f_{DOT}(s, \text{NORTH}) = 0.5$$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

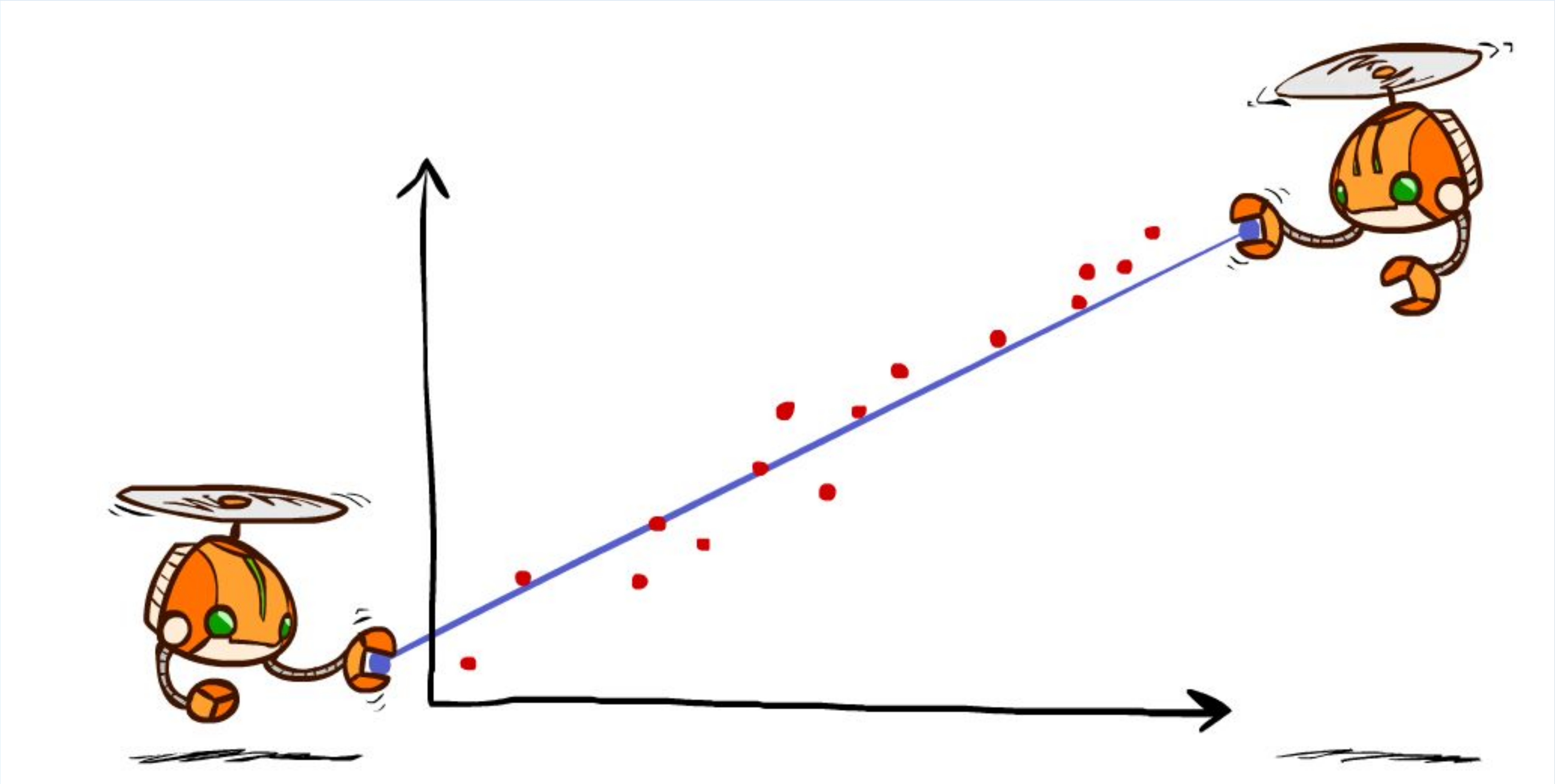
$$\text{difference} = -501$$

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

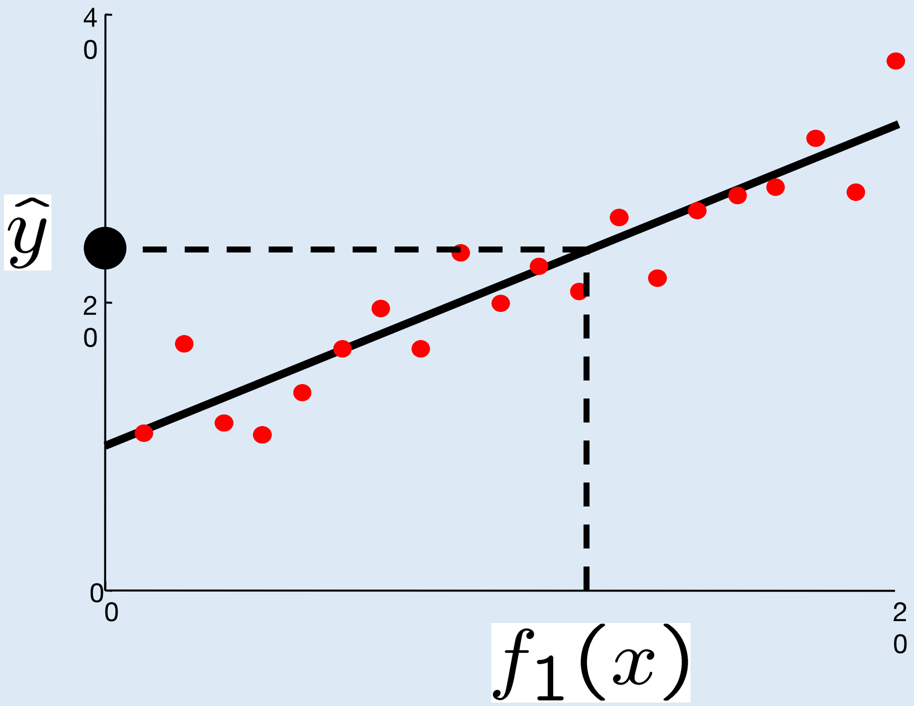
$$w_{GST} \leftarrow -1.0 + \alpha \,[-501]\, 1.0$$

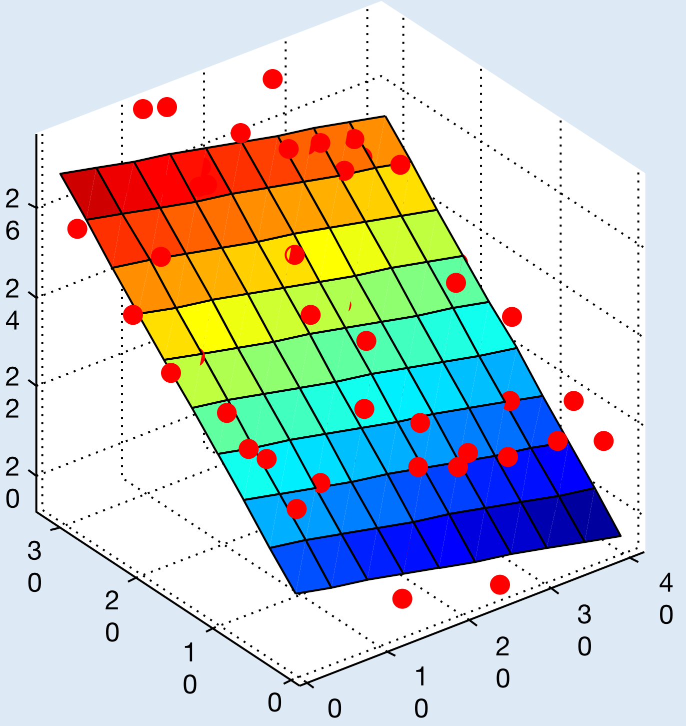
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

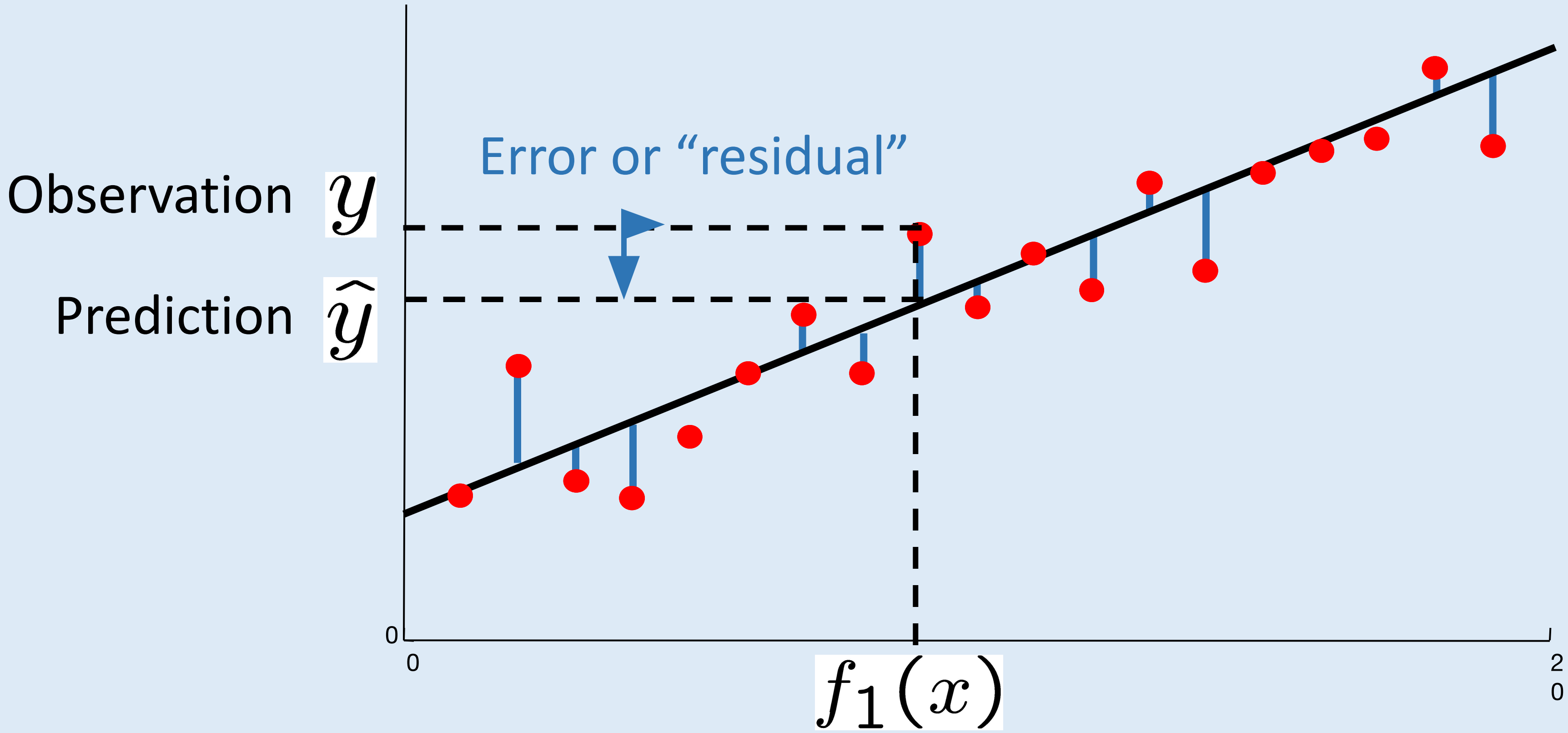
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



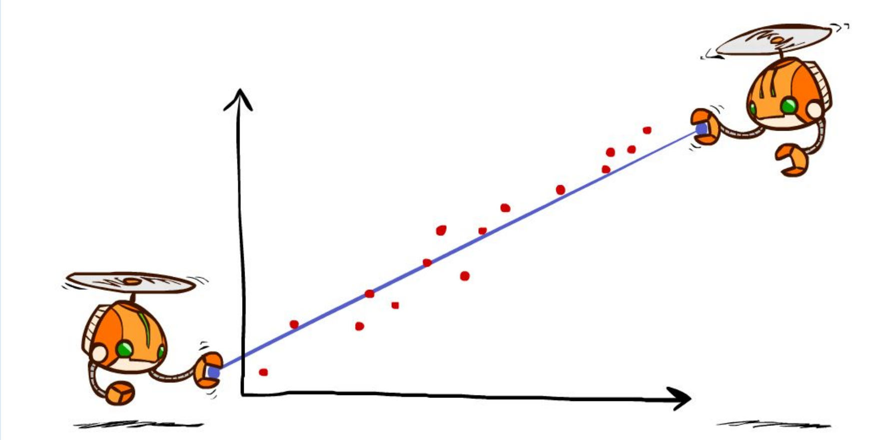
Minimizing Error

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"

Recent Reinforcement Learning Milestones

TDGammon

1992 by Gerald Tesauro, IBM

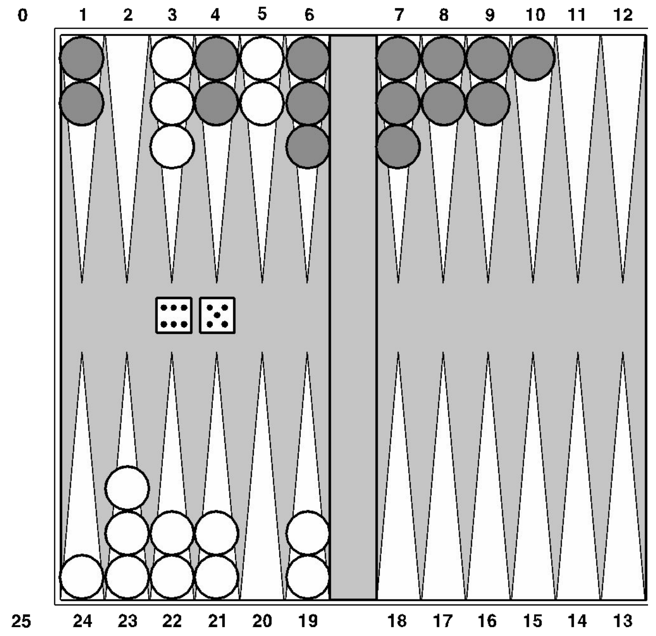
4-ply lookahead using V(s) trained from 1,500,000 games of self-play

3 hidden layers, ~100 units each

Input: contents of each location plus several handcrafted features

Experimental results:

- Plays approximately at parity with world champion
- Led to radical changes in the way humans play backgammon



Deep Q-Networks

$sample = r + \gamma \max_{a'} Q_w(s',a')$

$Q_w(s,a)$: Neural network

Deep Mind, 2015

Used a deep learning network to represent Q:

- Input is last 4 images (84x84 pixel values) plus score

49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro

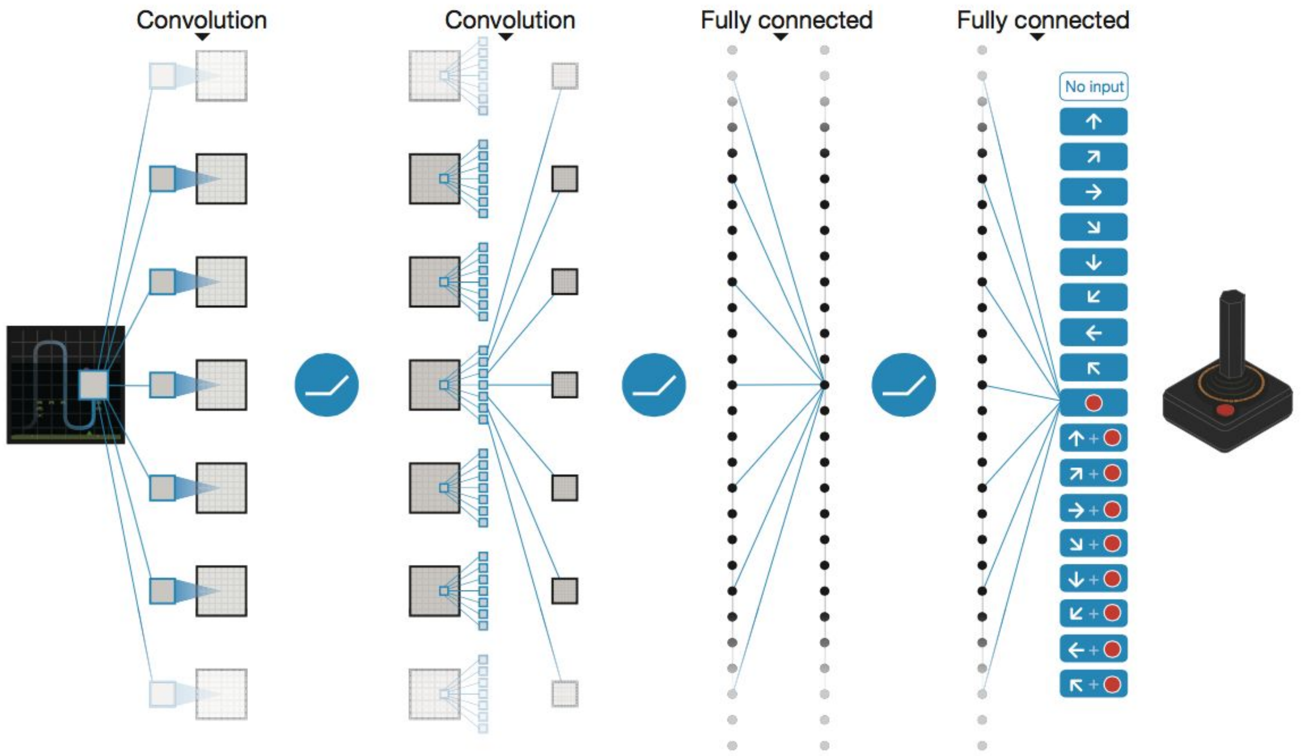
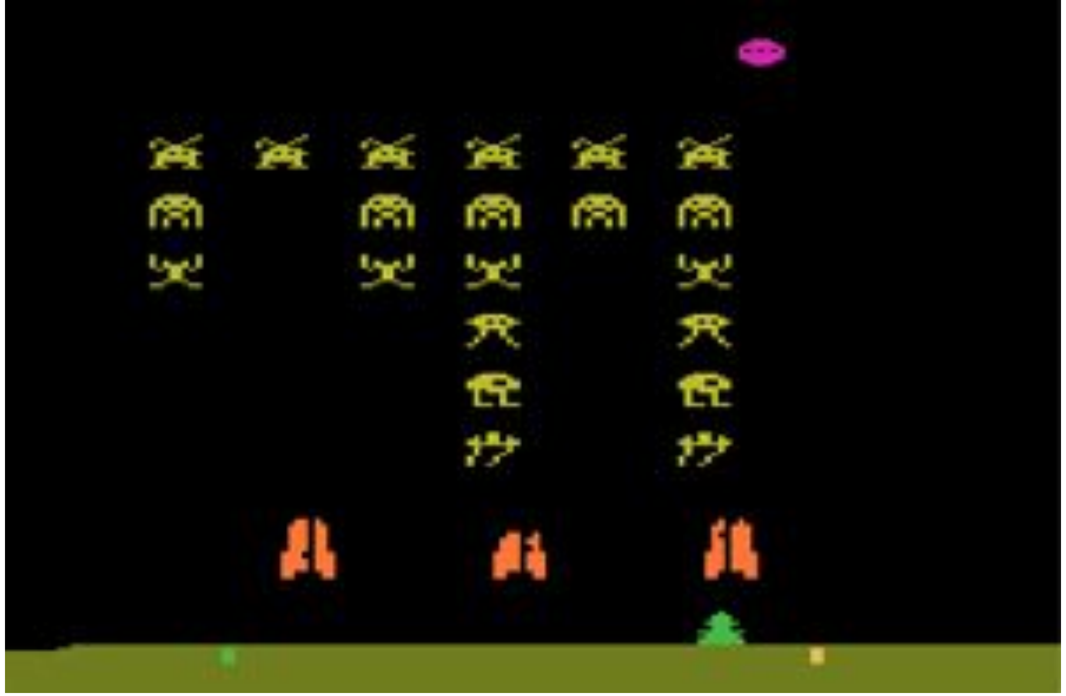


Image: Deep Mind



Images: Open AI, Atari

OpenAI Gym

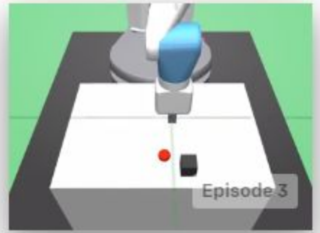
2016+

Benchmark problems for learning agents

https://gym.openai.com/envs

Acrobot-v1
Swing up a two-link robot.

Ant-v2
Make a 3D four-legged robot walk.

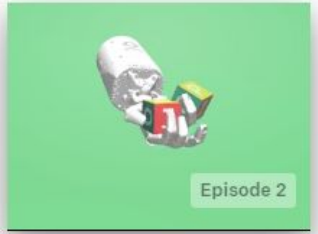


FetchPush-v0
Push a block to a goal position.

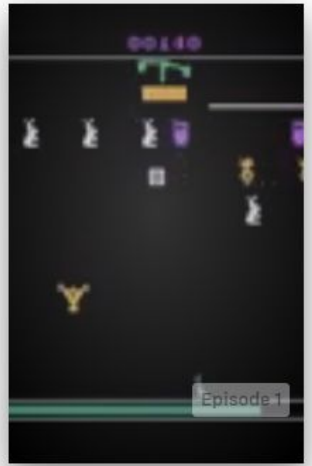
Breakout-ram-v0
Maximize score in the game Breakout, with RAM as input

MountainCarContinuous-v0
Drive up a big hill with continuous control.

Humanoid-v2
Make a 3D two-legged robot walk.



HandManipulateBlock-v0
Orient a block using a robot hand.



Carnival-v0
Maximize score in the game Carnival, with screen images as input

Images: Open AI

AlphaGo, AlphaZero

Deep Mind, 2016+

Autonomous Vehicles?

Reinforcement Learning from Human Feedback (RLHF)

Successful applications:

- Videogame bots
- Simulated robotics
- Fine-Tuning Large Language Models (LMMs), e.g., ChatGPT, Gemini, Claude
- Text-to-image models