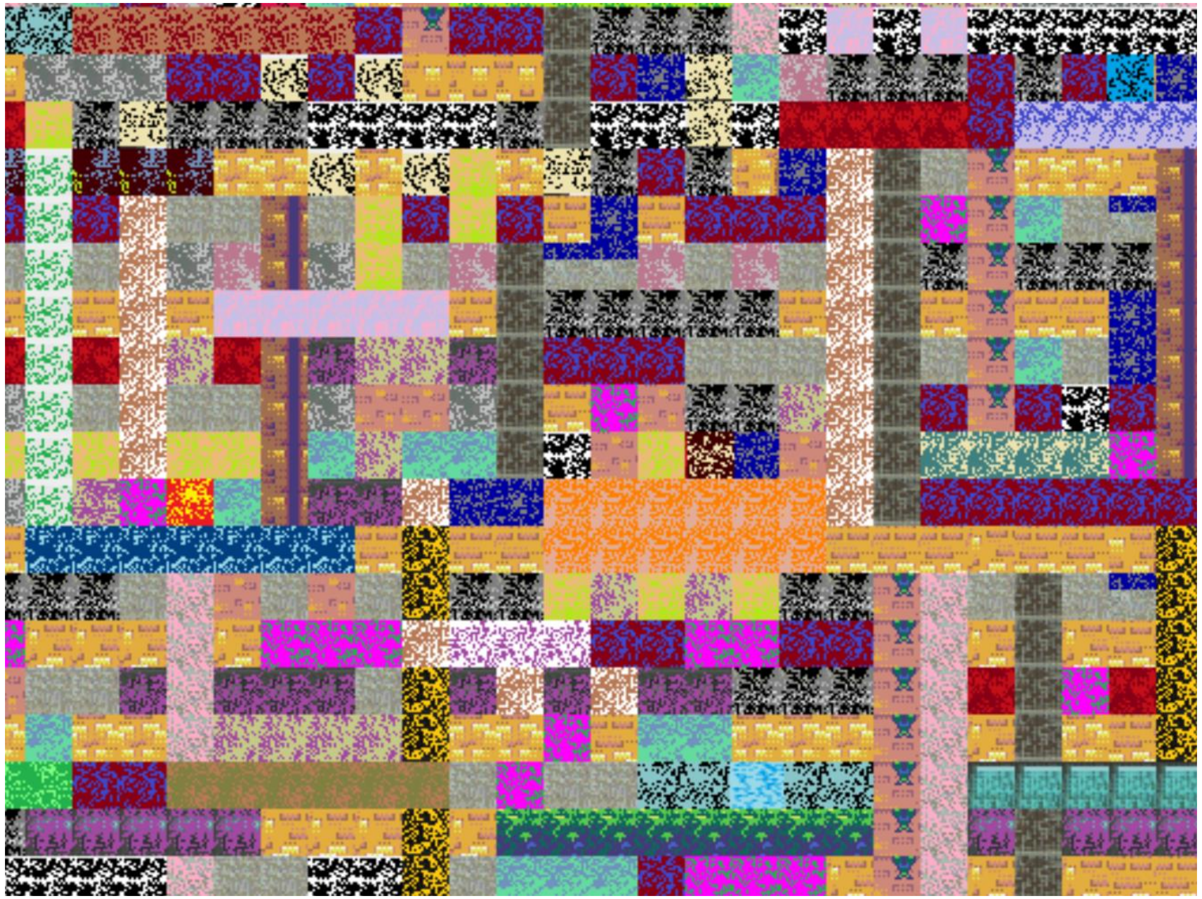
Warm-up as you walk in

https://high-level-4.herokuapp.com/experiment



https://rach0012.github.io/humanRL_website/

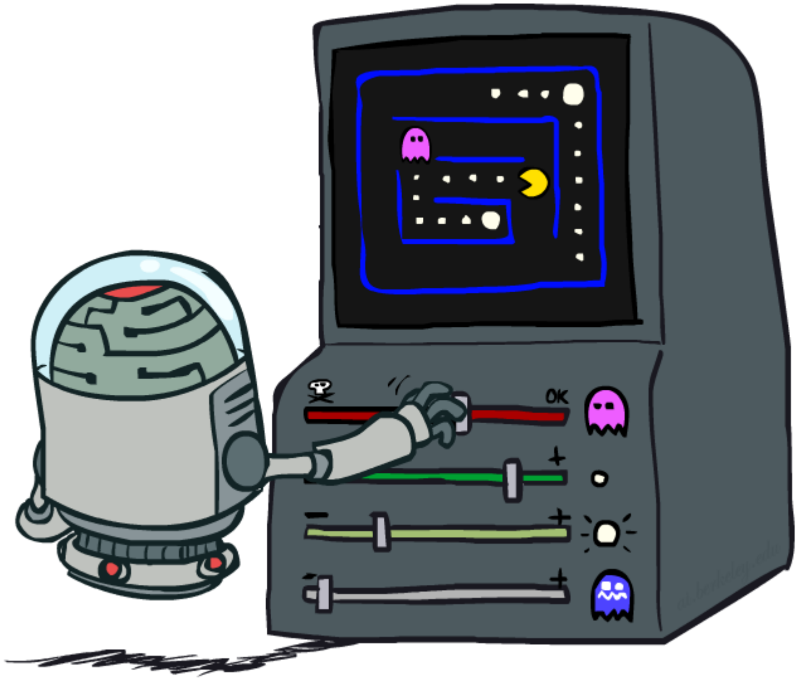
Announcements

Assignments:

- HW8
  - Due Tue 3/26, 10 pm
- P4
  - Due Thu 3/28, 10 pm
- HW9 (written)
  - Plan: Out tomorrow, due Tue 4/2

AI: Representation and Problem Solving

Reinforcement Learning II



Instructors: Pat Virtue & Stephanie Rosenthal

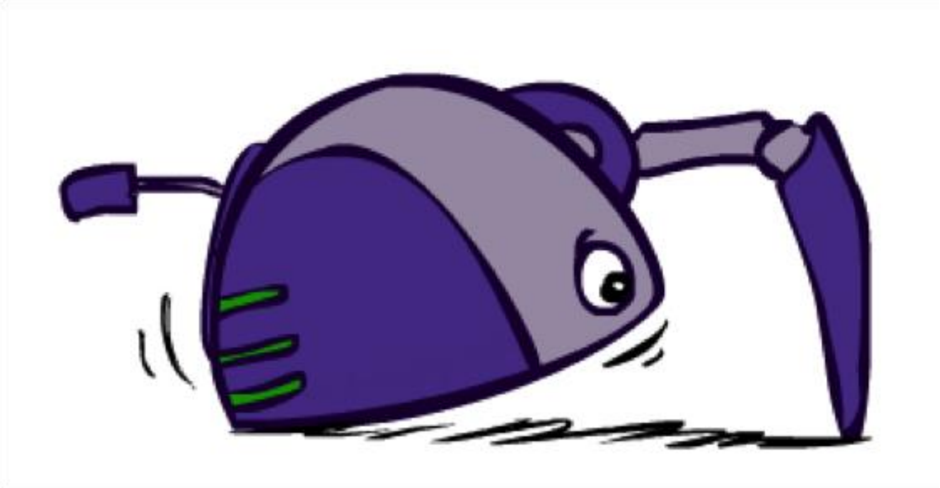
Slide credits: CMU AI and http://ai.berkeley.edu

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')

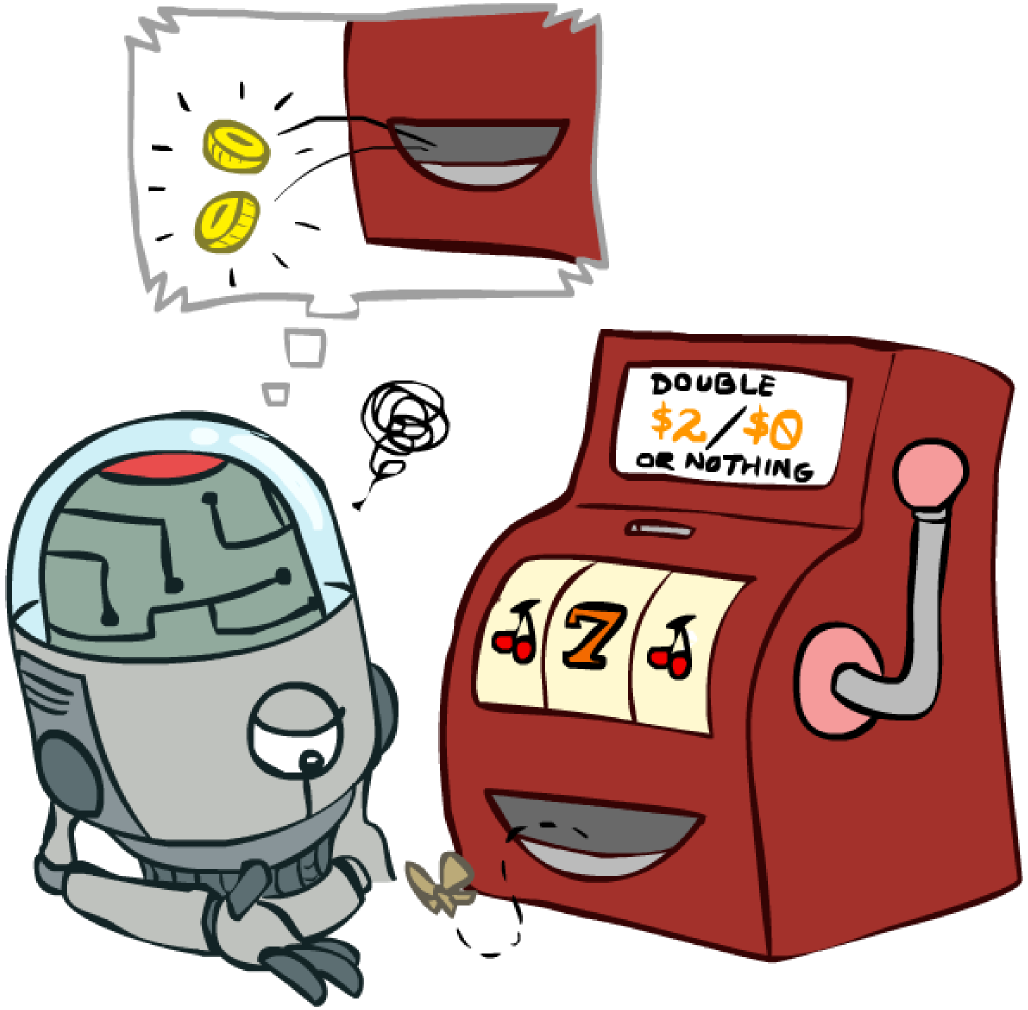
Still looking for a policy $\pi(s)$



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

Big idea: Compute all averages over T using sample outcomes

Temporal Difference Learning



Model-Free Learning

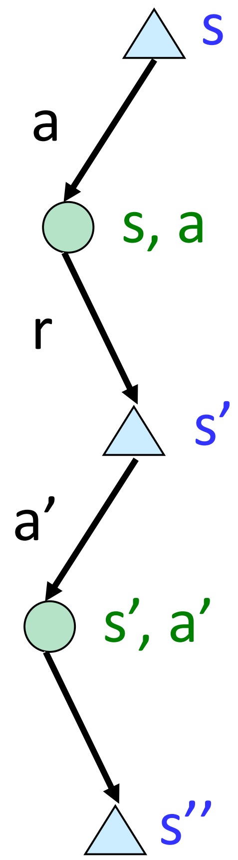
Model-free (temporal difference) learning

- 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



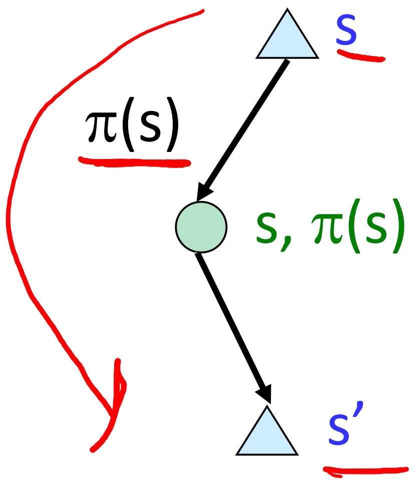
Temporal Difference Learning

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

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

Big idea: learn from every experience!

- Update V(s) each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often



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): $sample = r + \gamma V^{\pi}(s')$

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

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\,[sample - V^{\pi}(s)]$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) - \alpha \nabla Error$

$Error = \frac{1}{2}\left(sample - V^{\pi}(s)\right)^2$

$\frac{dE}{dV} = -(samp - V^{\pi}(s))$

Piazza Poll 1

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

Which converts TD values into a policy?

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: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')]\,, \quad \forall\, s$

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

Piazza Poll 1

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

Which converts TD values into a policy?

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: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')]\,, \quad \forall\, s$

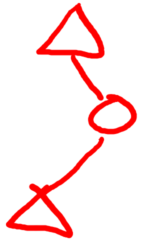
Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')]\,, \quad \forall\, s$

None of the above

MDP/RL Notation

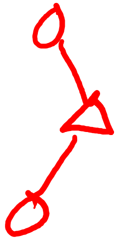
Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

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: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall s$

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

$\pi(s) = \arg\max_a Q(s,a)$

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

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

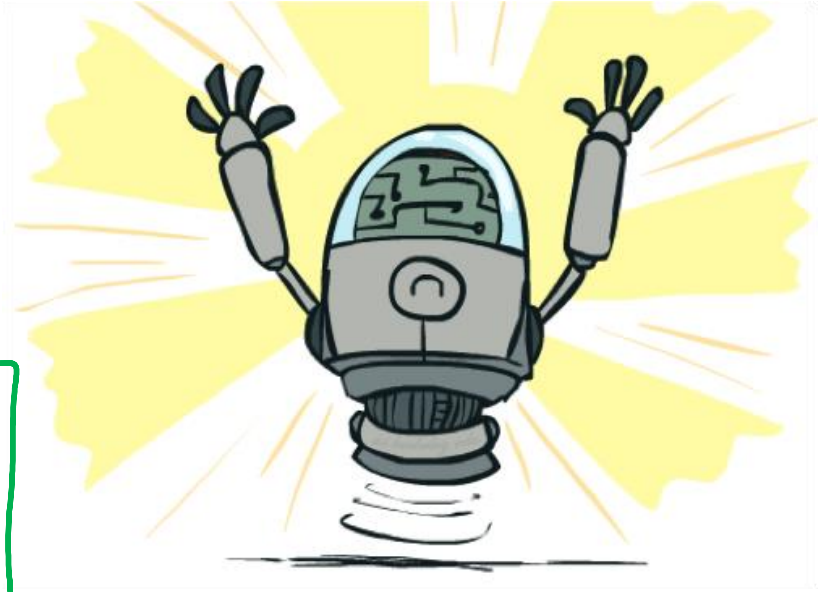
- But we want to average over results from (s,a) (Why?)
- So keep a running average

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

Q-Learning Properties

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 (!)

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

Demo Q-Learning Auto Cliff Grid

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

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

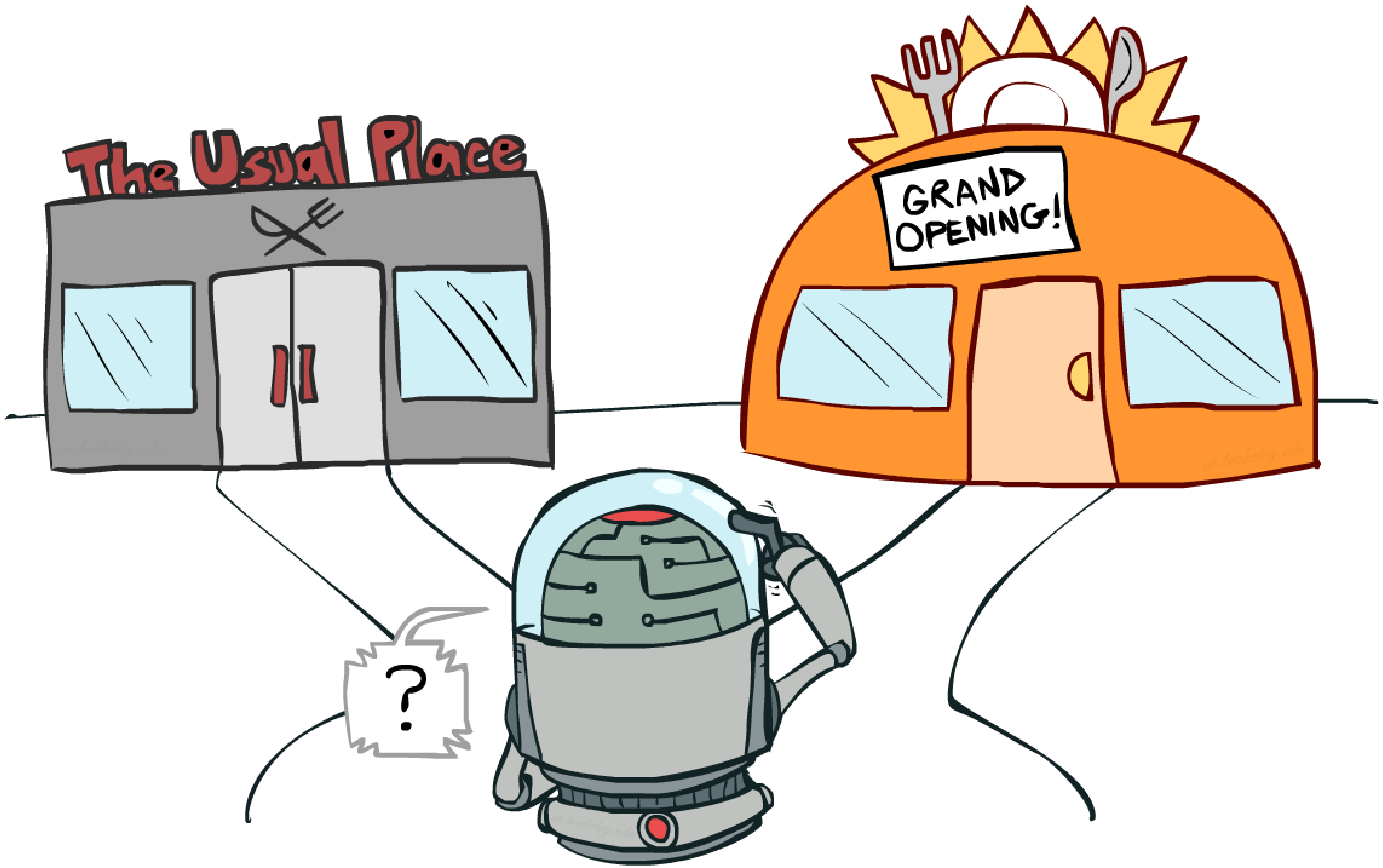
Unknown MDP: Model-Based

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	TD/Value Learning

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

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

Demo Q-learning – Manual Exploration – Bridge Grid

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.

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

10/100

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

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

$f(u, n)$

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

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

Demo Q-learning – Exploration Function – Crawler

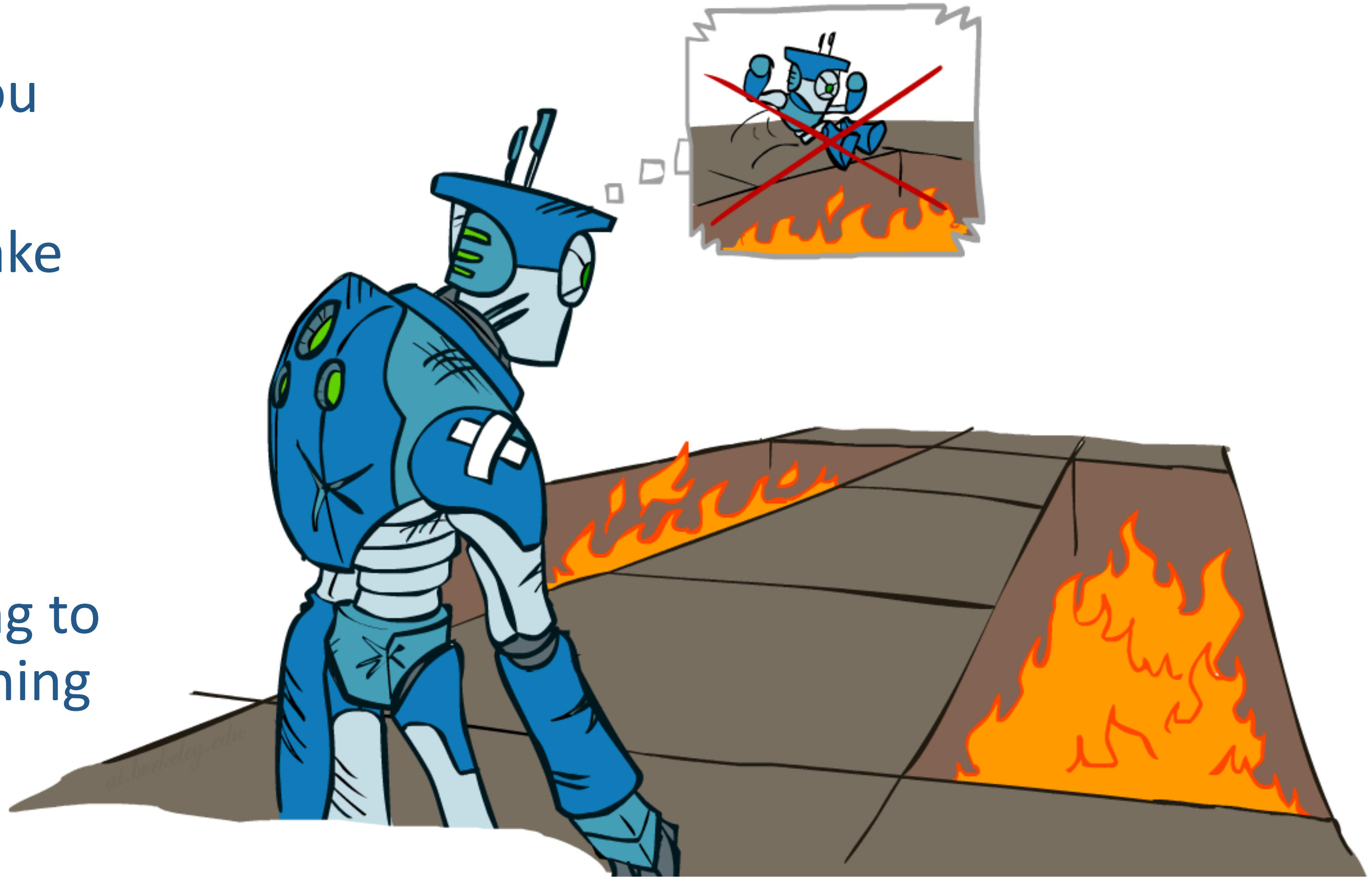
Regret

Even if you learn the optimal policy, you still make mistakes along the way!

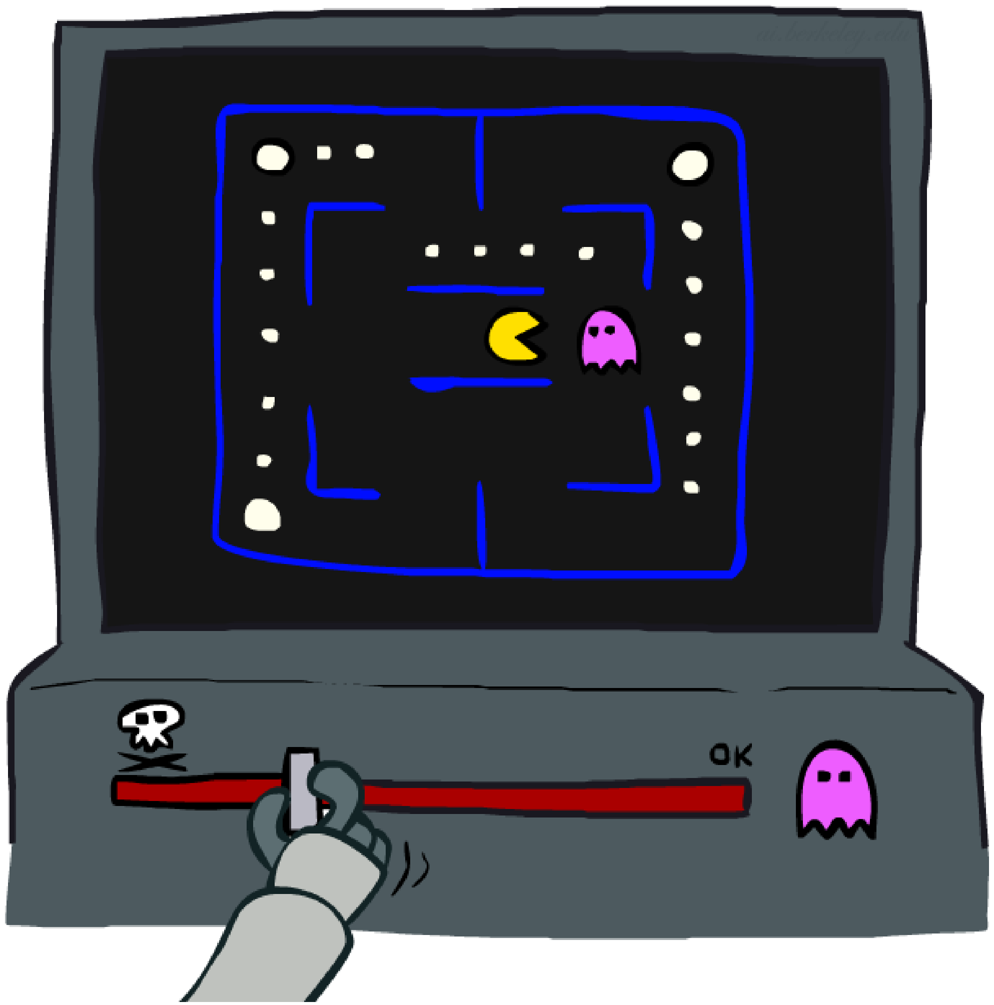
Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards

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

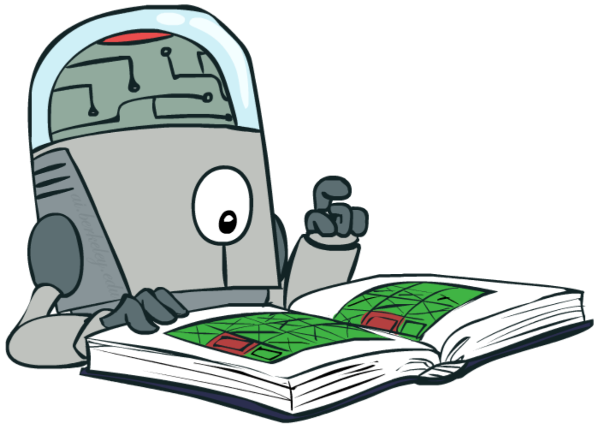
Basic Q-Learning keeps a table of all q-values

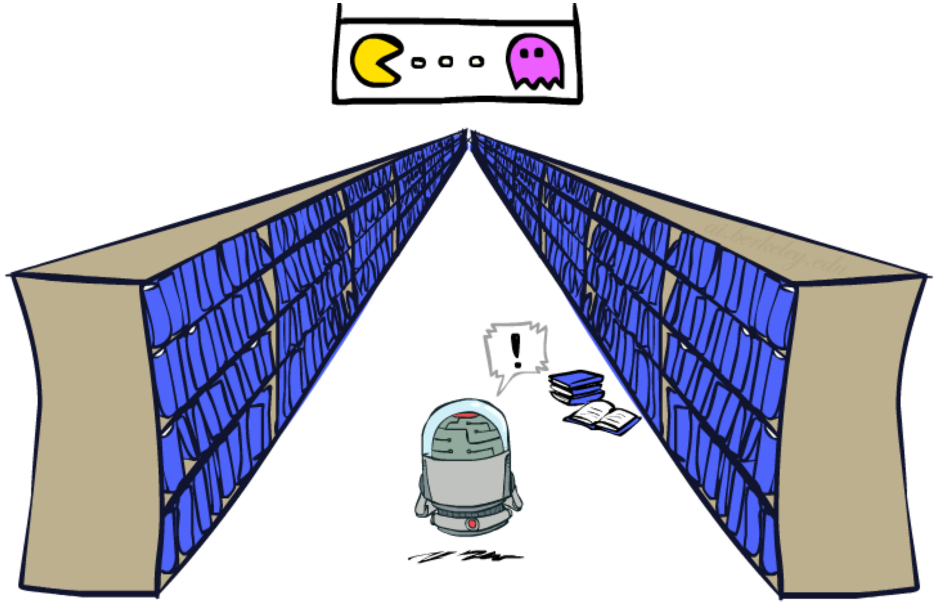
In realistic situations, we cannot possibly learn about every single state!

- 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

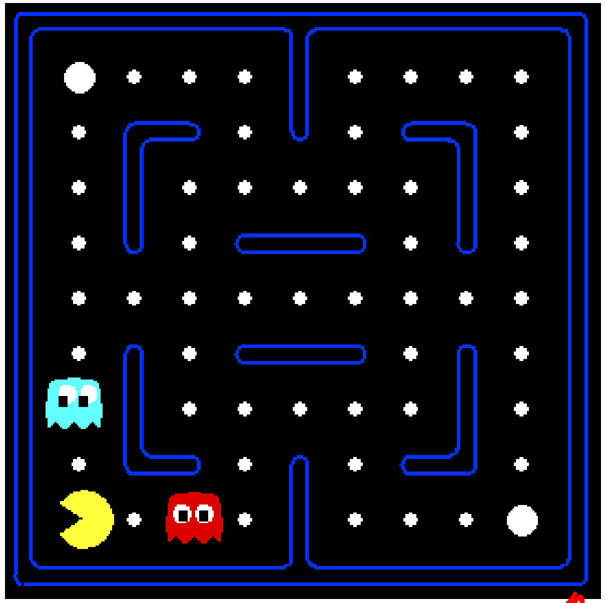




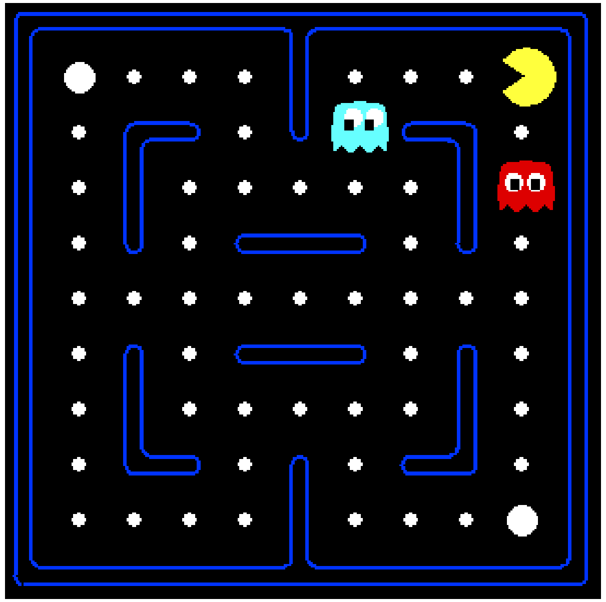
[demo – RL pacman]

Example: Pacman

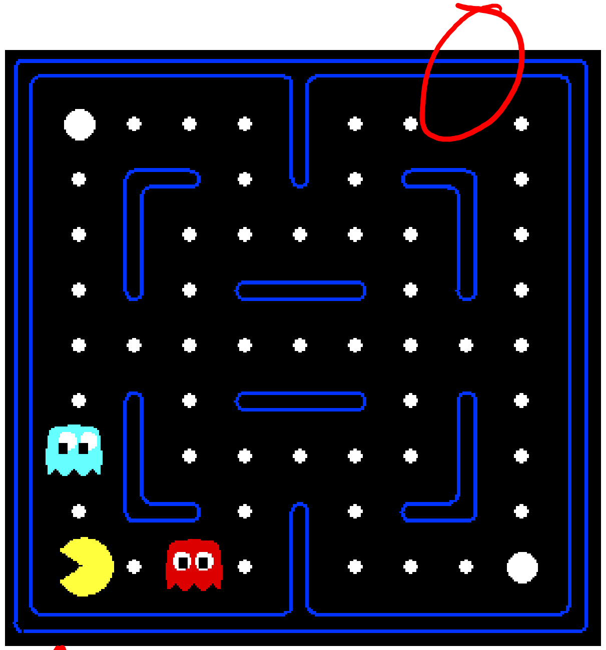
Let's say we discover through experience that this state is bad:



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)]

Demo Q-Learning Pacman – Tiny – Watch All

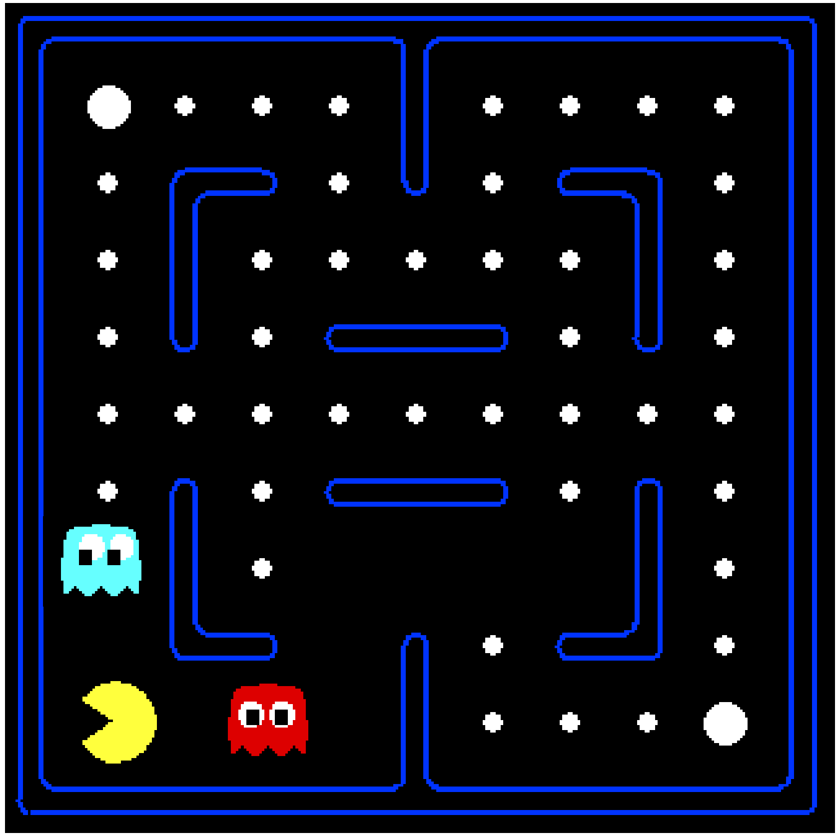
Demo Q-Learning Pacman – Tiny – Silent Train

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:

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

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

$E = \frac{1}{2}(y - x)^2$ $\quad E = \frac{1}{2}(y - w f(x))^2$

$\frac{dE}{dx} = -(y - x)$ (error) $\quad \frac{dE}{dw} = -(y - w f(x)) f(x)$

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

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

  (sample: $R(s,a,s') + \gamma \max_{a'} Q(s',a')$; error: $R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$)

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

- $w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \, \partial Q_w(s,a)/\partial w_i$

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

$Q_w(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a)$ $\quad \frac{dQ}{dw_2} = f_2(s,a)$

Updating a linear value function

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

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

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

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

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

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

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

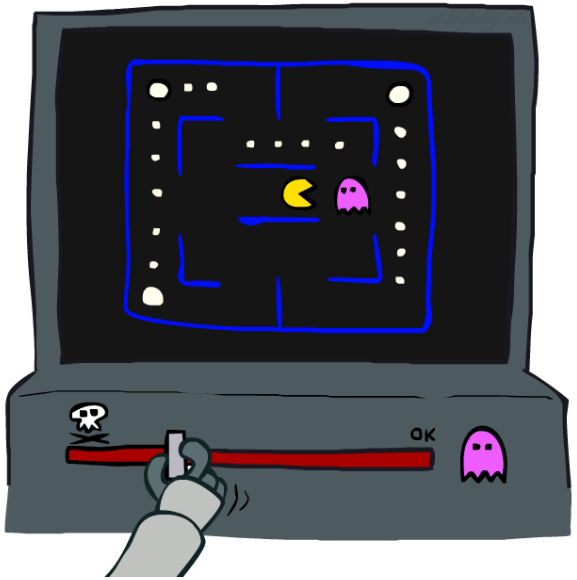
Q-learning with linear Q-functions:

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

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

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

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



Intuitive interpretation:

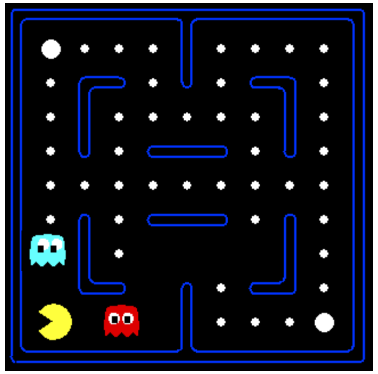
- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

Formal justification: online least squares

Example: Q-Pacman

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

s

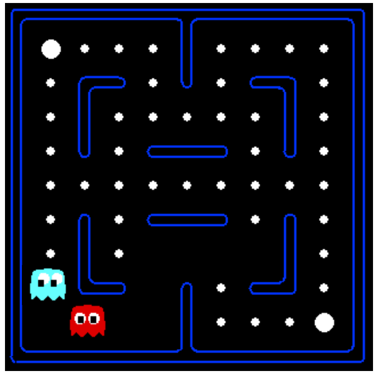


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

s'



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

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

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

difference $= -501$

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

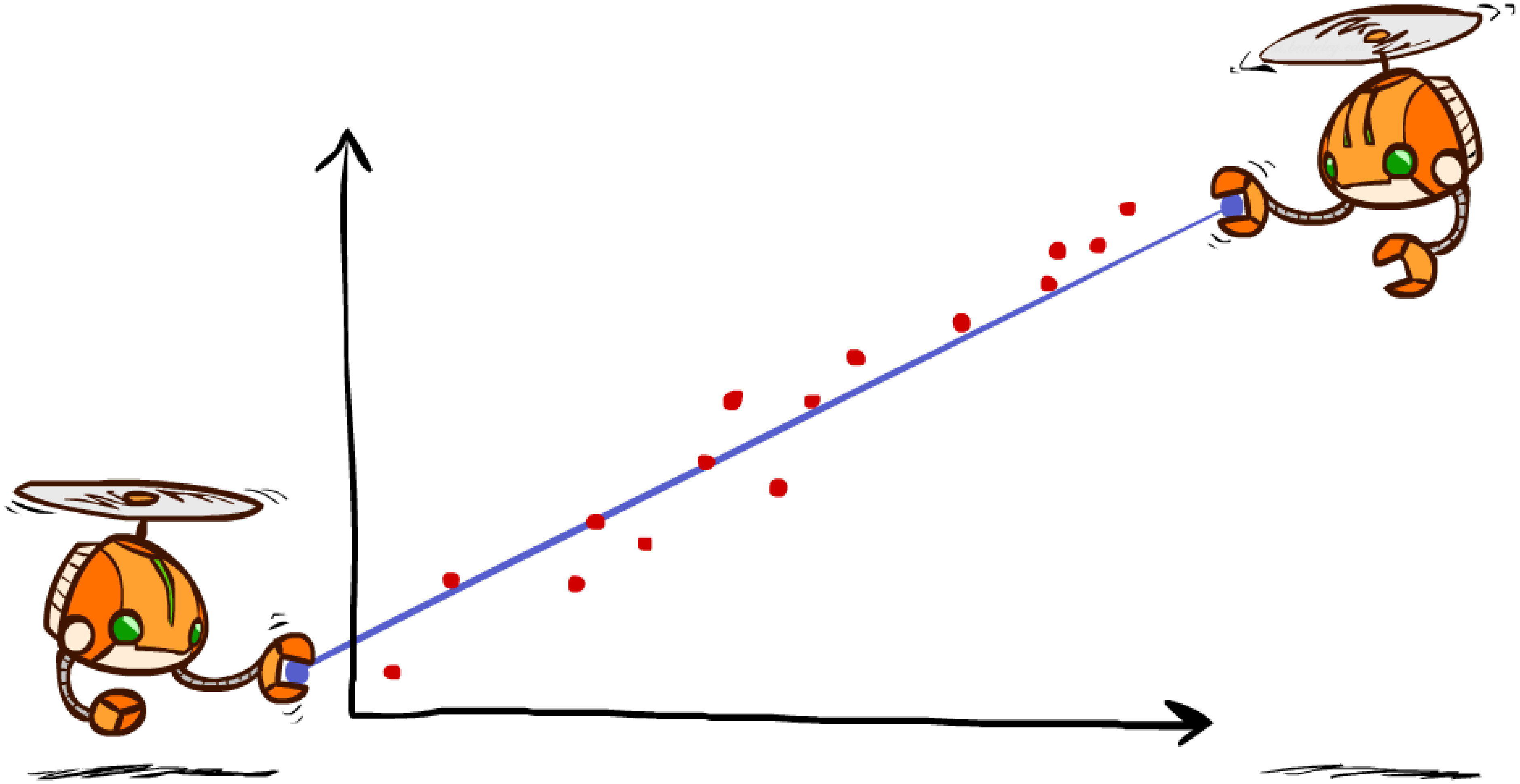
$w_{GST} \leftarrow -1.0 + \alpha \, [-501] \, 1.0$

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

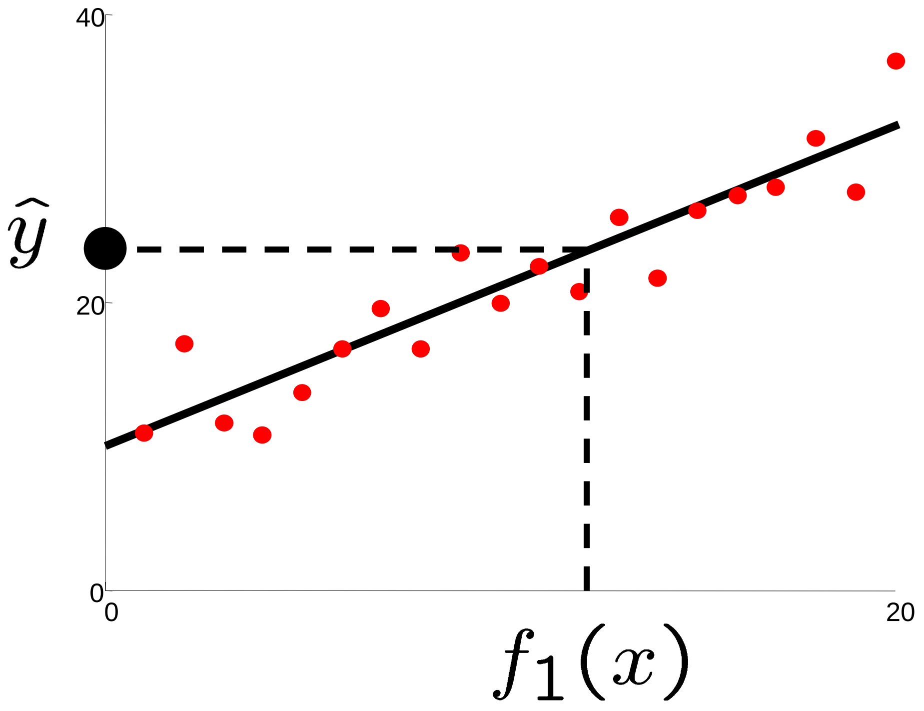
[Demo: approximate Q-learning pacman (L11D10)]

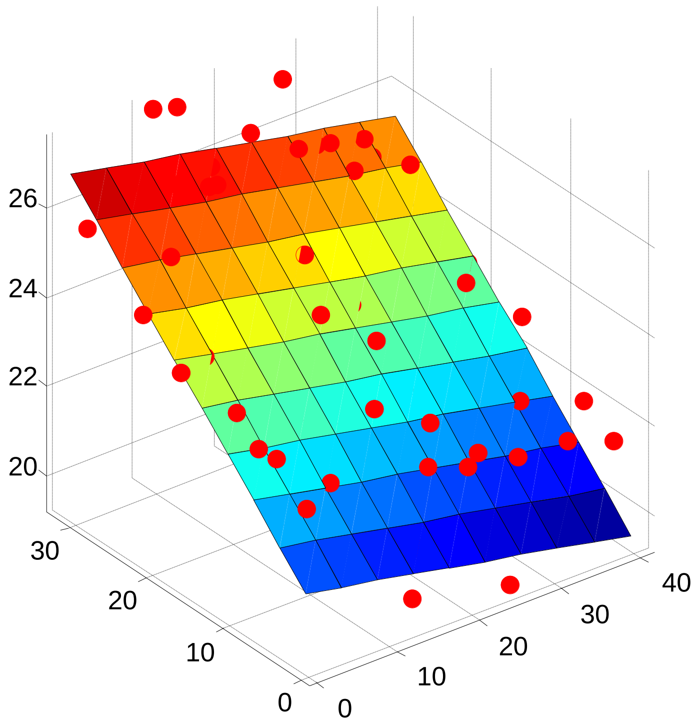
Demo Approximate Q-Learning -- Pacman

Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

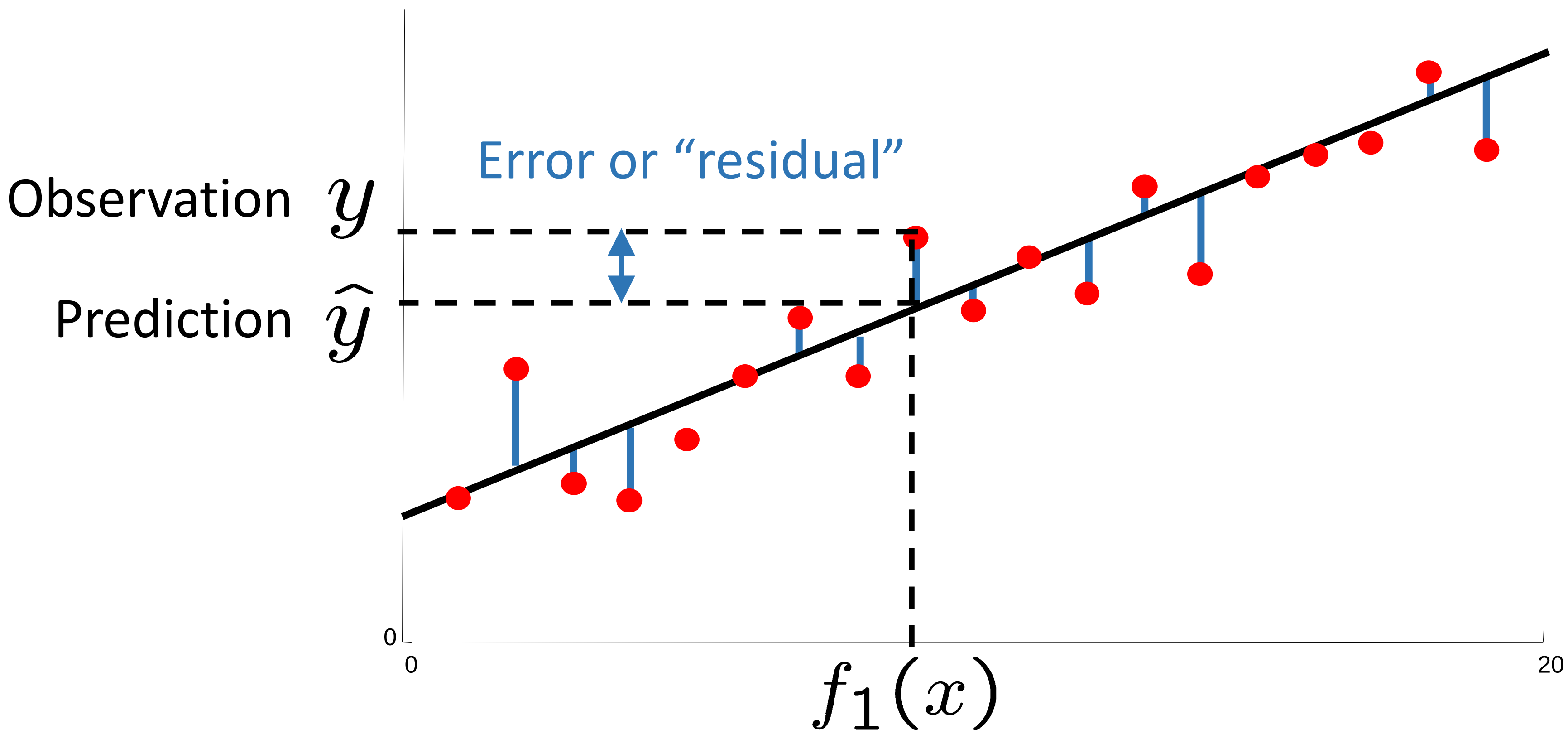
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



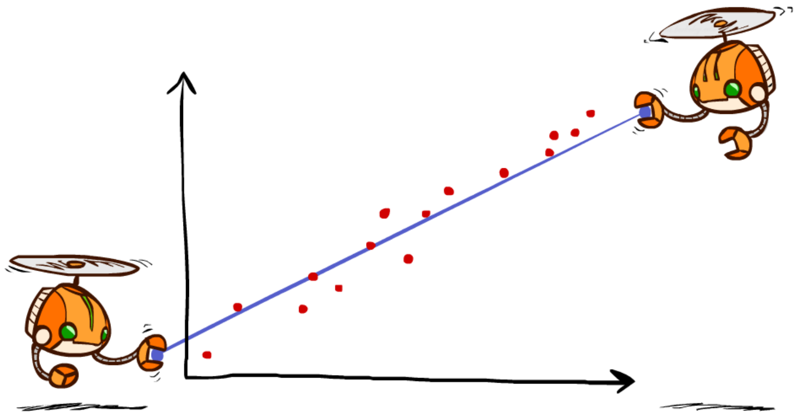
Minimizing Error

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

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

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

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

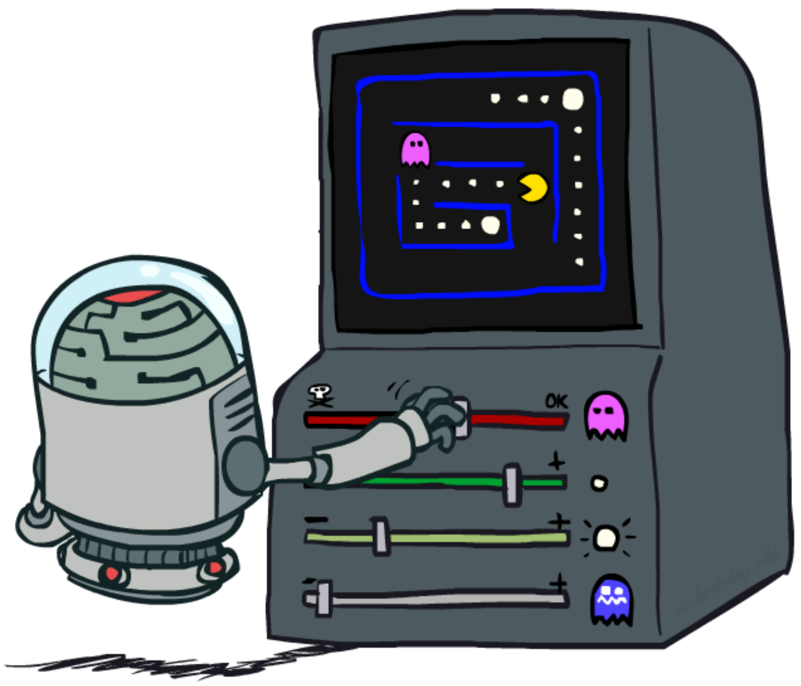


Approximate q update explained:

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

"target" "prediction"

Recent Reinforcement Learning Milestones



TDGammon

1992 by Gerald Tesauro, IBM

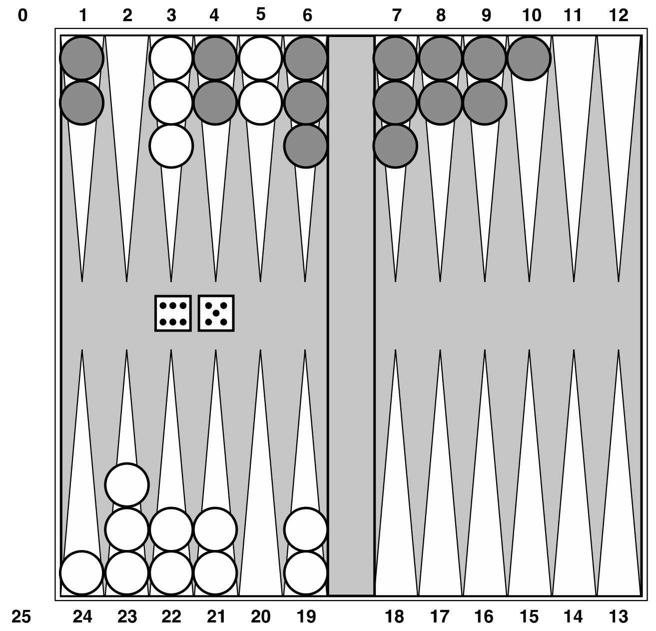
4-ply lookahead using V(s) trained from 1,500,000 games of self-play

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



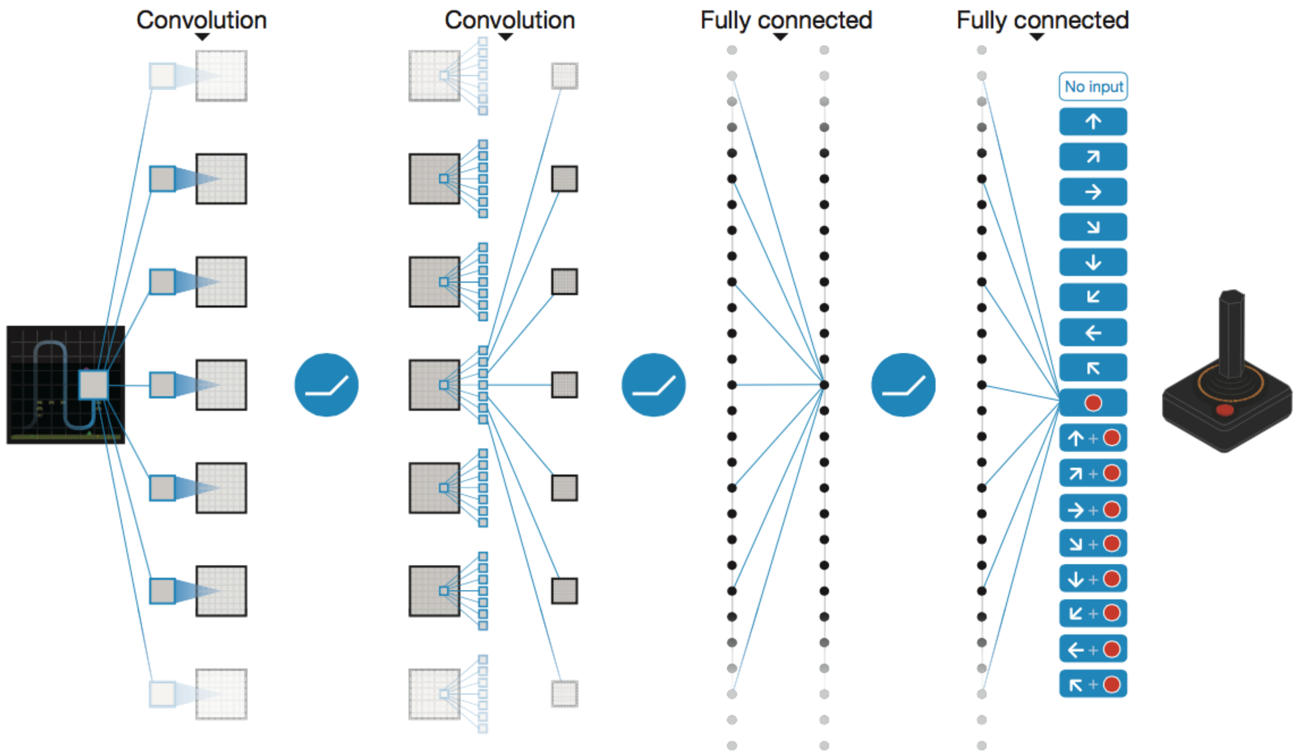
Deep Q-Networks

Deep Mind, 2015

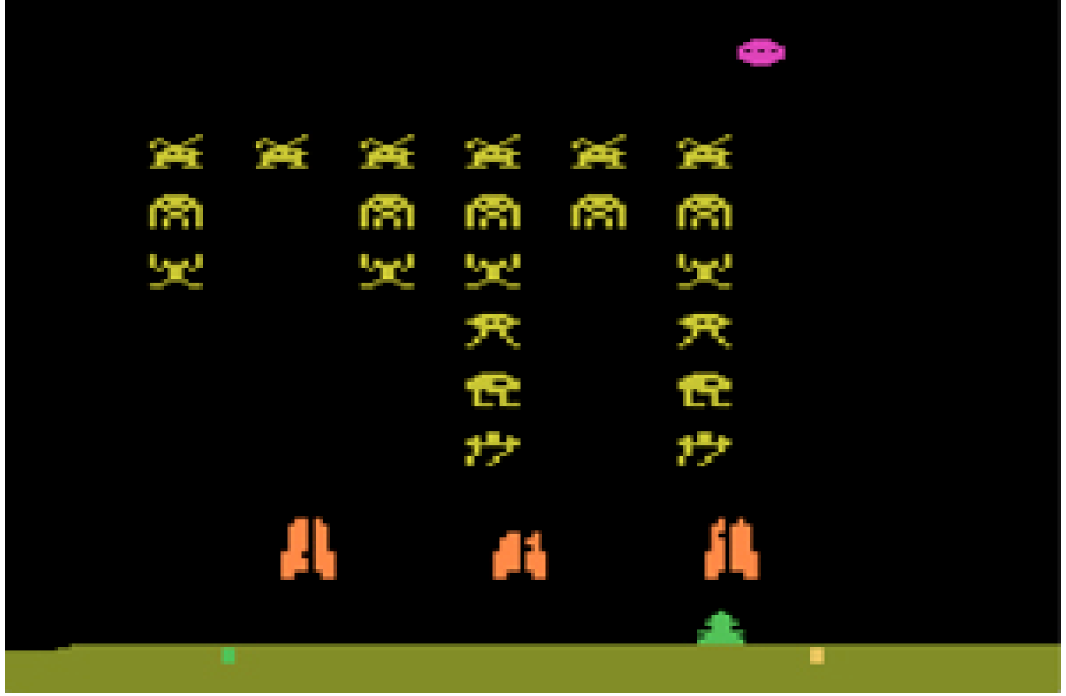
Used a deep learning network to represent Q:

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

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




001 2 1




0
OXYGEN
ACTIVISION


0000025
01 180
142 KM/H

OpenAI Gym

2016+

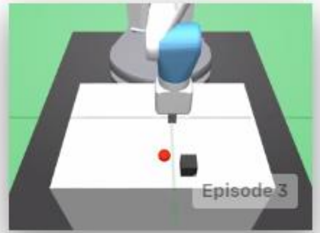
Benchmark problems for learning agents

https://gym.openai.com/envs

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

Acrobot-v1
Swing up a two-link robot.

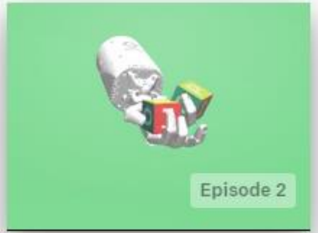
Ant-v2
Make a 3D four-legged robot walk.



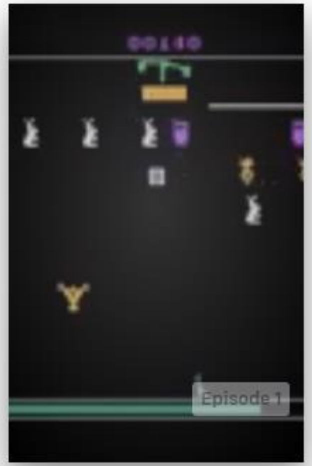
FetchPush-v0
Push a block to a goal position.

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

AlphaGo, AlphaZero

Deep Mind, 2016+

Autonomous Vehicles?