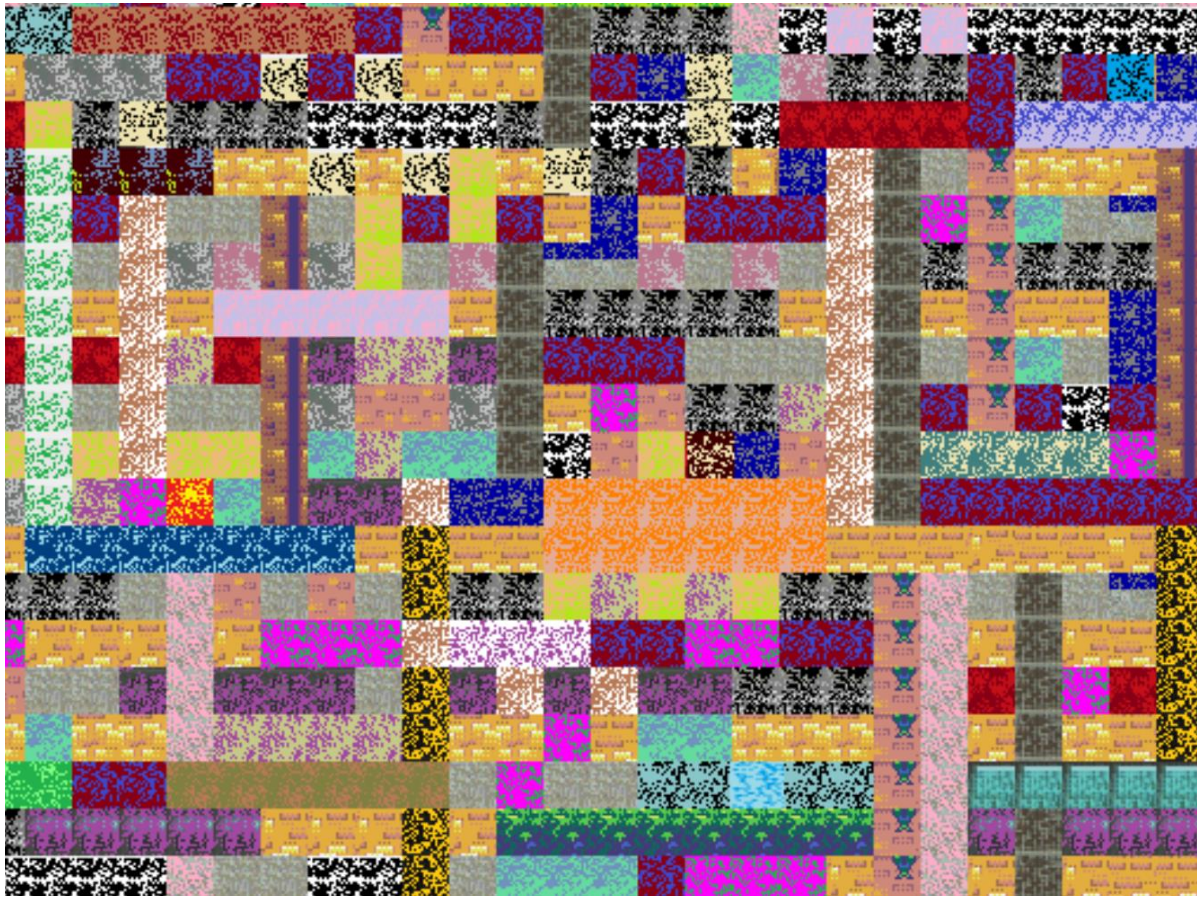
Warm-up as you walk in

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



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

Announcements

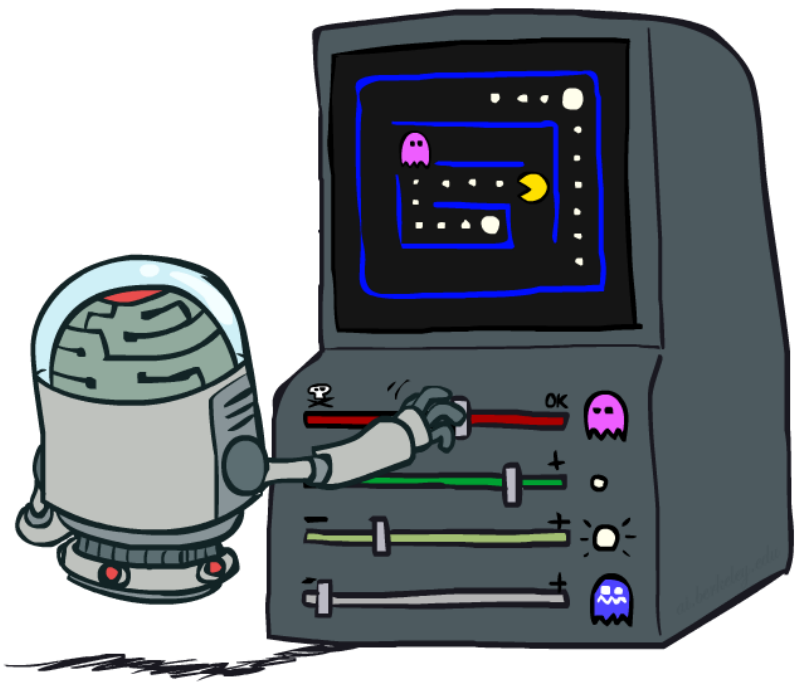
Assignments:

- HW8 (written)
  - Due 10/29 Tue, 10 pm
- P4
  - Due 10/31 Thu, 10 pm

Piazza in-class post is ready to go

AI: Representation and Problem Solving

Reinforcement Learning II



Instructors: Fei Fang & Pat Virtue

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

Learning Objective (RL I&II)

- Describe the relationships and differences between
  - Markov Decision Processes (MDP) vs Reinforcement Learning (RL)
  - Model-based vs Model-free RL
  - Temporal-Difference Value Learning (TD Value Learning) vs Q-Learning
  - Passive vs Active RL
  - Off-policy vs On-policy Learning
  - Exploration vs Exploitation
- Describe and implement
  - TD (Value) Learning
  - Q-Learning
  - ϵ-Greedy algorithm
  - Approximate Q-learning (Feature-based)
- Derive weight update for Approximate Q-learning

This Lecture

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

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

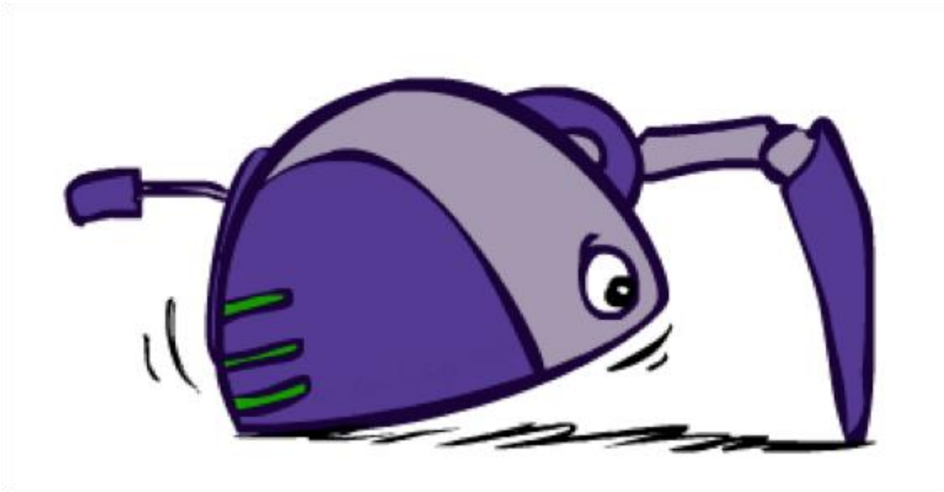
Q-learning: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

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 $\pi(s)$

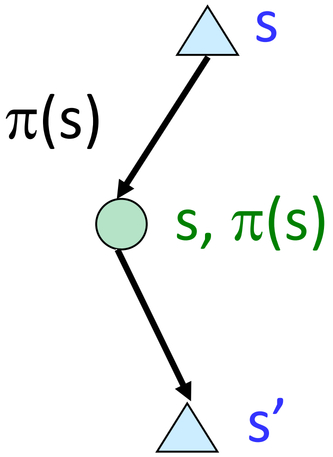


New twist: don't know T or R, so must try out actions

Big idea: Compute all averages over T using sample outcomes

Temporal Difference (Value) Learning

Task: Given policy π, learn state value V^{π}



Learn from every experience

- Update $V^{\pi}(s)$ each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often
- Move values toward latest sample (running average)

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

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

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

Piazza Poll 1

What is the derivative of function $f(x) = \frac{1}{2}(y - x)^2$ w.r.t. x (y is a constant)?

A: $y - x$

B: $-2x$

C: $x - y$

Piazza Poll 1

What is the derivative of function $f(x) = \frac{1}{2}(y - x)^2$ w.r.t. x (y is a constant)?

A: $y - x$

B: $-2x$

C: $x - y$

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

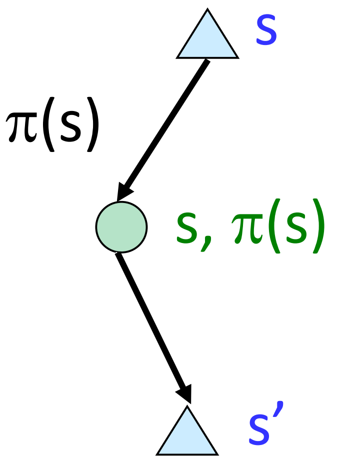
$$\nabla f(x) = \frac{1}{2} \times 2 \times (y - x) \times \nabla(y - x) \ = x - y$$

Temporal Difference (Value) Learning

Task: Given policy π, learn state value V^π

Learn from every experience

- Update $V^\pi(s)$ each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often
- Move values toward latest sample (running average)



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

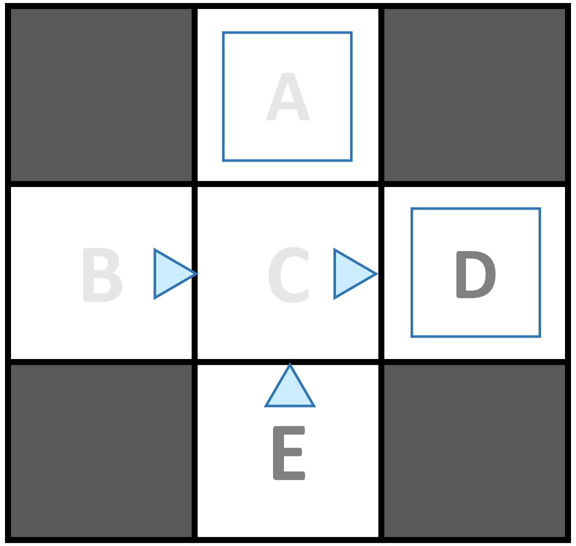
Update to V(s): $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha) sample$

Equivalent to: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

Equivalent to: $V^\pi(s) \leftarrow V^\pi(s) - \alpha \nabla Error$ $Error = \frac{1}{2}\left(sample - V^\pi(s)\right)^2$

Example: Temporal Difference (Value) Learning

Input Policy π



Assume: $\gamma = 1$

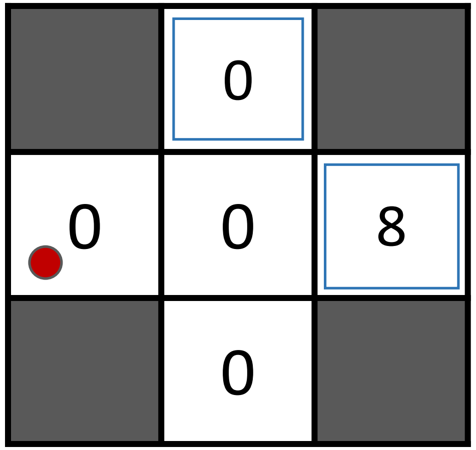
$\alpha = 1/2$

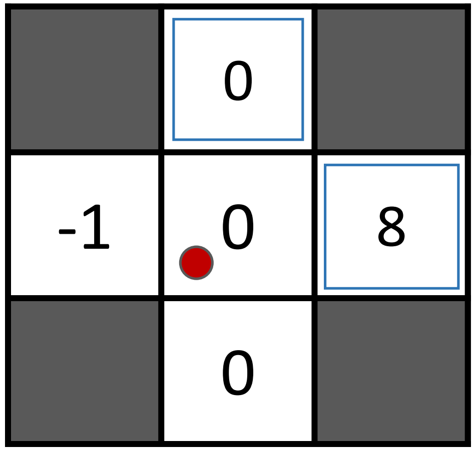
$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha[r + \gamma V^{\pi}(s') - V^{\pi}(s)]$$

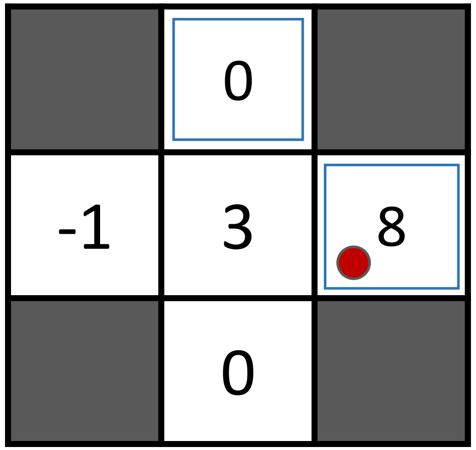
Observed Transitions

B, east, C, -2

C, east, D, -2



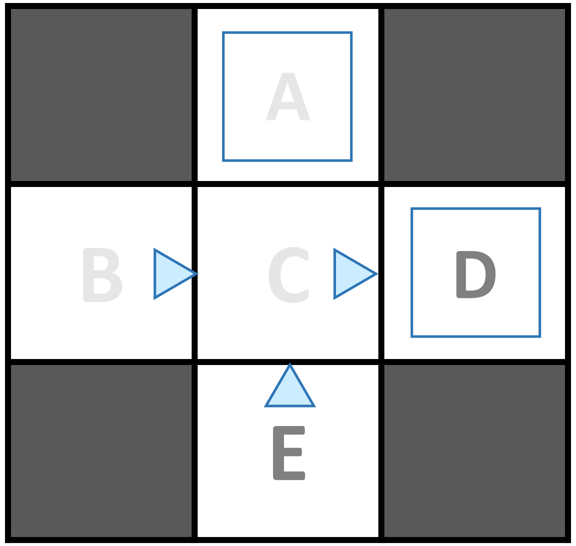




Example: Temporal Difference (Value) Learning

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha[r + \gamma V^{\pi}(s') - V^{\pi}(s)]$$

Input Policy π



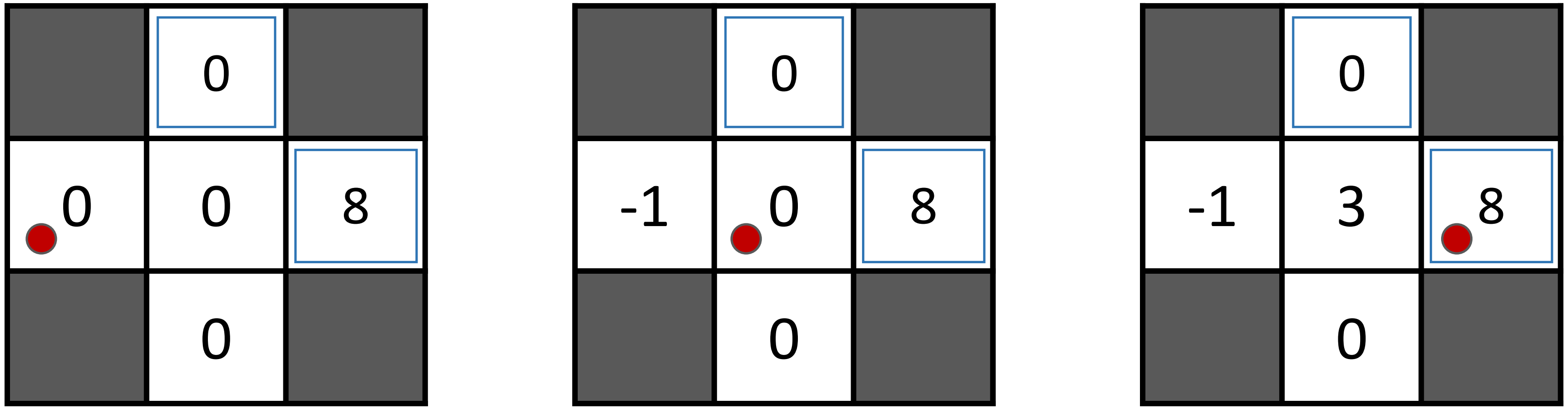
Assume: γ = 1

α= 1/2

Observed Transitions

B, east, C, -2

C, east, D, -2



$$V^{\pi}(B) \leftarrow V^{\pi}(B) + 0.5 * [-2 + 1 * V^{\pi}(C) - V^{\pi}(B)] = 0 + 0.5 * (-2) = -1$$

$$V^{\pi}(C) \leftarrow V^{\pi}(C) + 0.5 * [-2 + 1 * V^{\pi}(D) - V^{\pi}(C)] = 0 + 0.5 * (-2 + 8) = 3$$

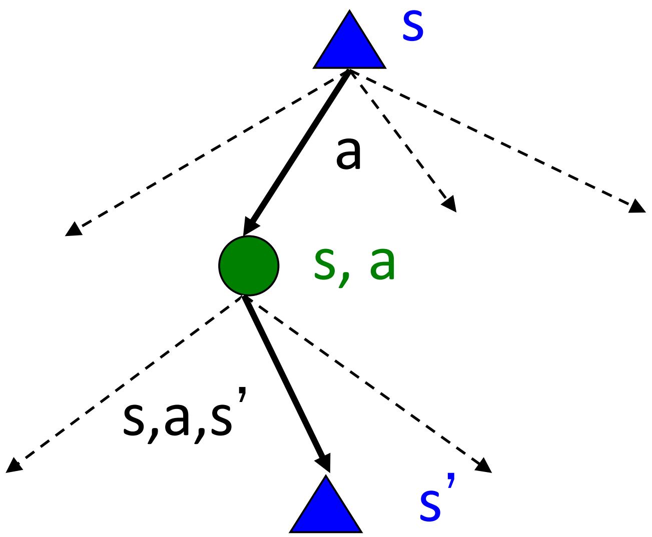
Problems with TD Value Learning

TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages

However, if we want to turn values into an improved policy, we're sunk:

$$\pi^{new}(s) = \underset{a}{\operatorname{argmax}}\, Q^{\pi_{old}}(s,a)$$

$$= \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_{old}}(s')]$$



Again, we don't know R and T!

Solution: Directly learn Q-values, not state values

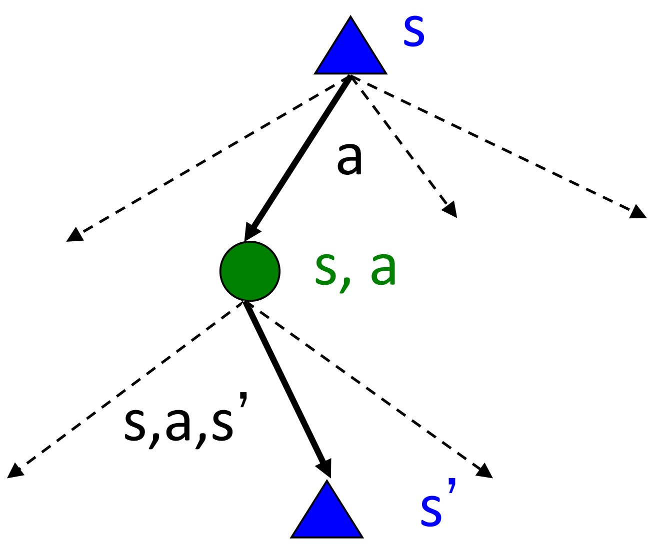
In fact, can directly learn true/optimal Q-values in a model-free way (Q-Learning)

Keep in mind that our ultimate goal is to find optimal policy!

Extending TD Learning to Q-Value

Task: Given policy π, learn state value V^π

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha[R(s,a,s') + \gamma V^\pi(s') - V^\pi(s)]$$
where $a = \pi(s)$

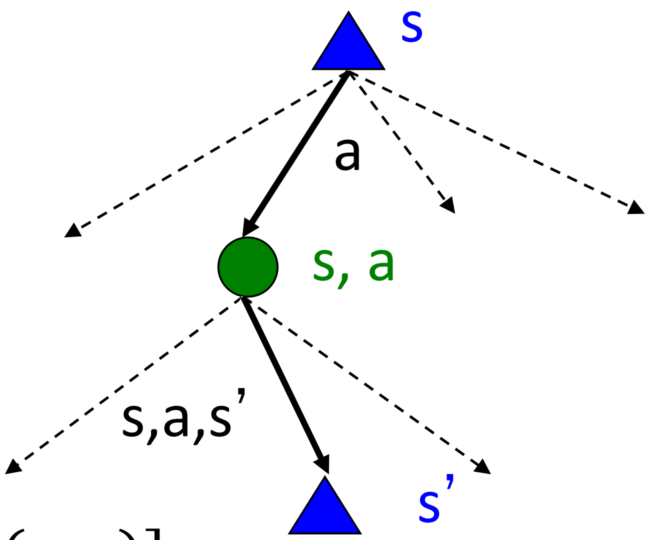


Task: Given policy π, learn Q-state value Q^π

Task: Directly learn true/optimal Q-state value Q

Q-Learning. No given policy π.

Extending TD Learning to Q-Value



Task: Given policy π, learn state value V^π

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha[R(s,a,s') + \gamma V^\pi(s') - V^\pi(s)]$$
where $a = \pi(s)$

Task: Given policy π, learn Q-state value Q^π

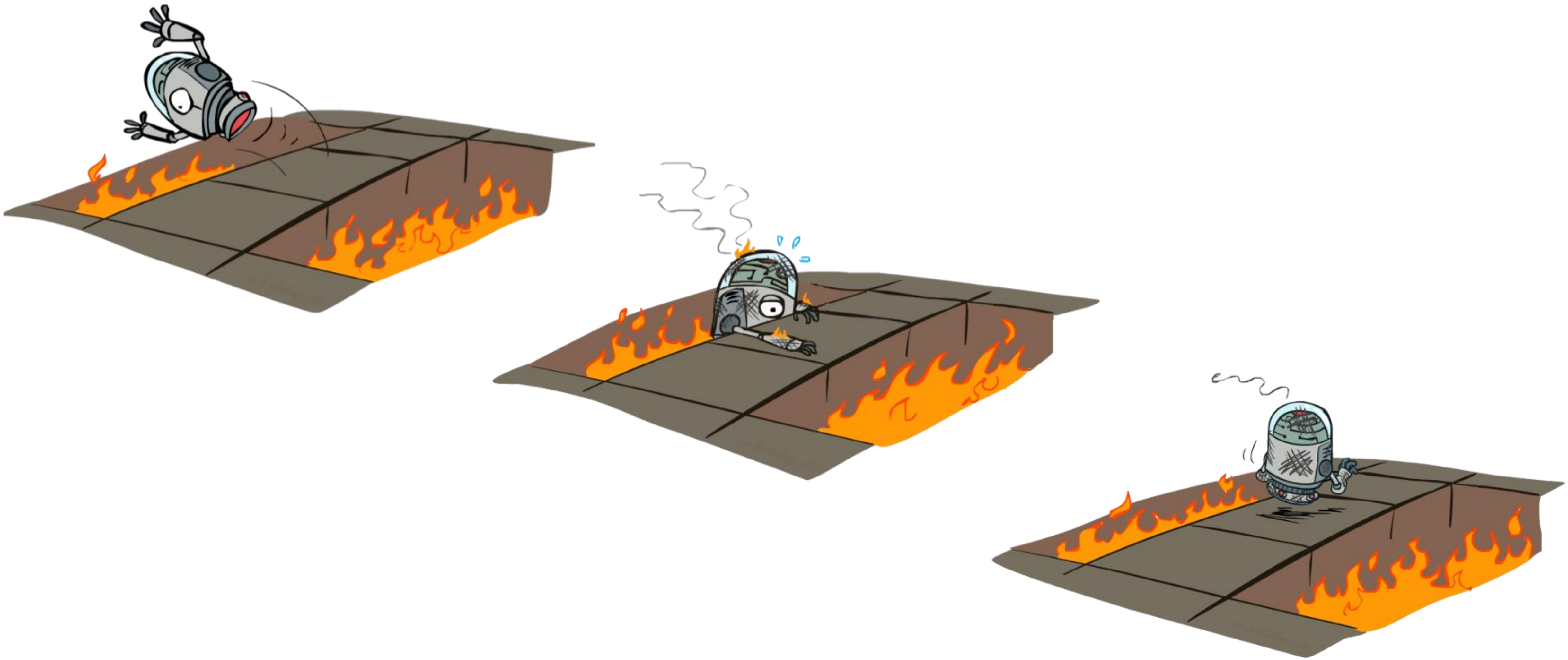
$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha[R(s,a,s') + \gamma Q^\pi(s',\pi(s')) - Q^\pi(s,a)]$$
where $a = \pi(s)$

Task: Directly learn true/optimal Q-state value Q

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

Q-Learning. No given policy π.

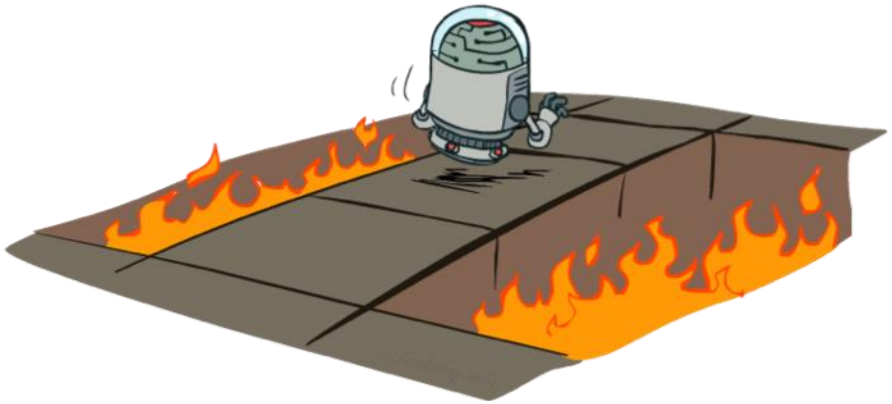
Active Reinforcement Learning



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 (no given policy π)
- Goal: learn the optimal policy / values



In this case:

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

Demo Q-Learning -- Gridworld

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

Demo Q-Learning -- Crawler

[Demo: Q-learning – crawler (L10D3)]

Detour: 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, 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 consider our previous value of $Q(s,a)$ (Why?)
- So keep a running average

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

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

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

Demo Q-Learning Auto Cliff Grid

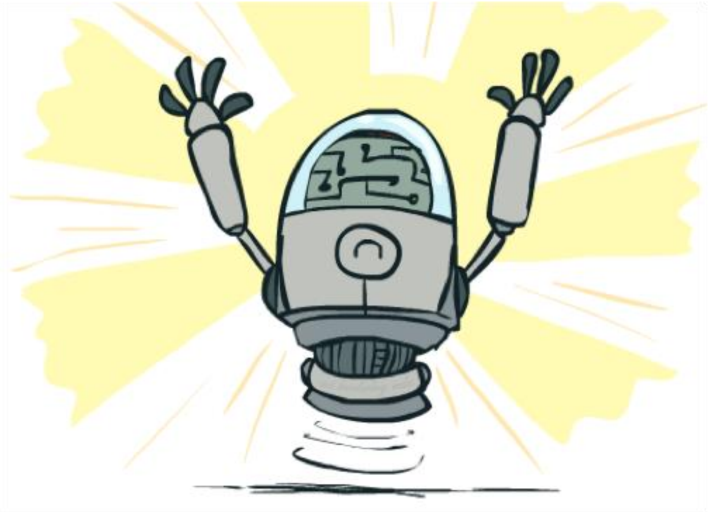
[Demo: Q-learning – auto – cliff grid (L11D1)]

Q-Learning Properties

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

This is called off-policy learning: you learn the value of the optimal policy while your behavior policy (how you act) is a different policy

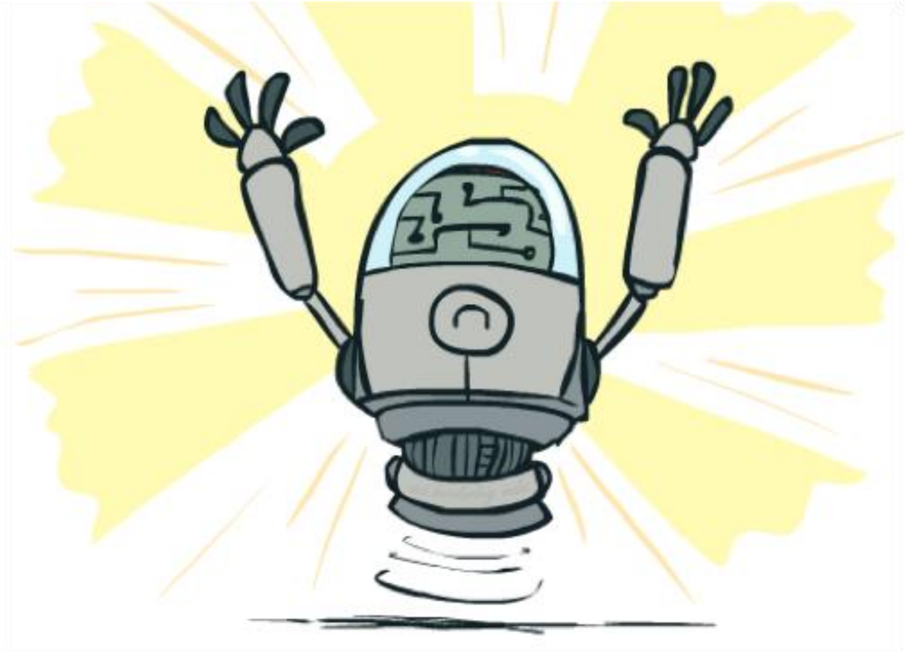
In contrast, on-policy learning (e.g., TD value learning) estimates the value of a policy while acting according to it



Q-Learning Properties

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 (!)



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

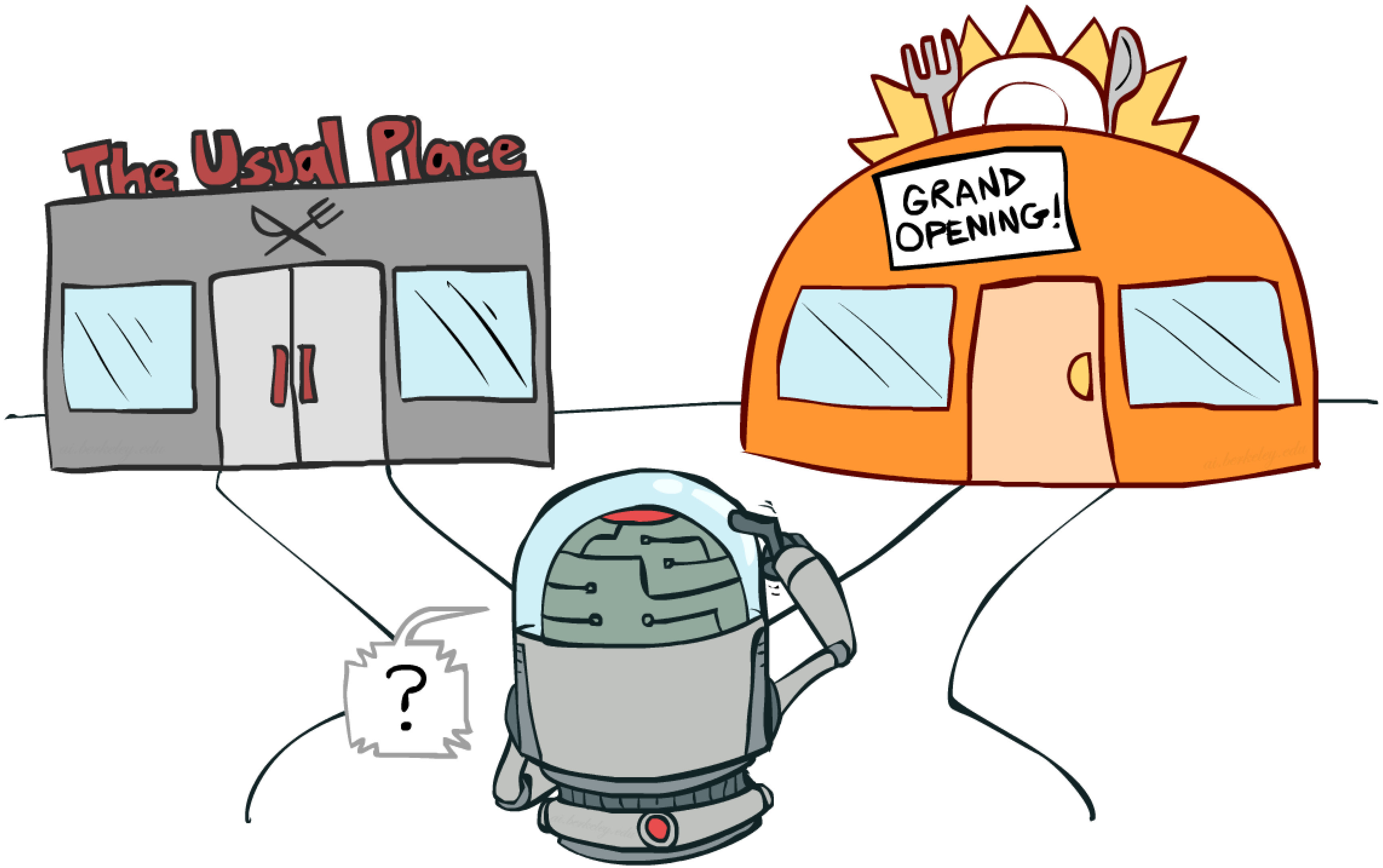
Unknown MDP: Model-Based

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.

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

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)]

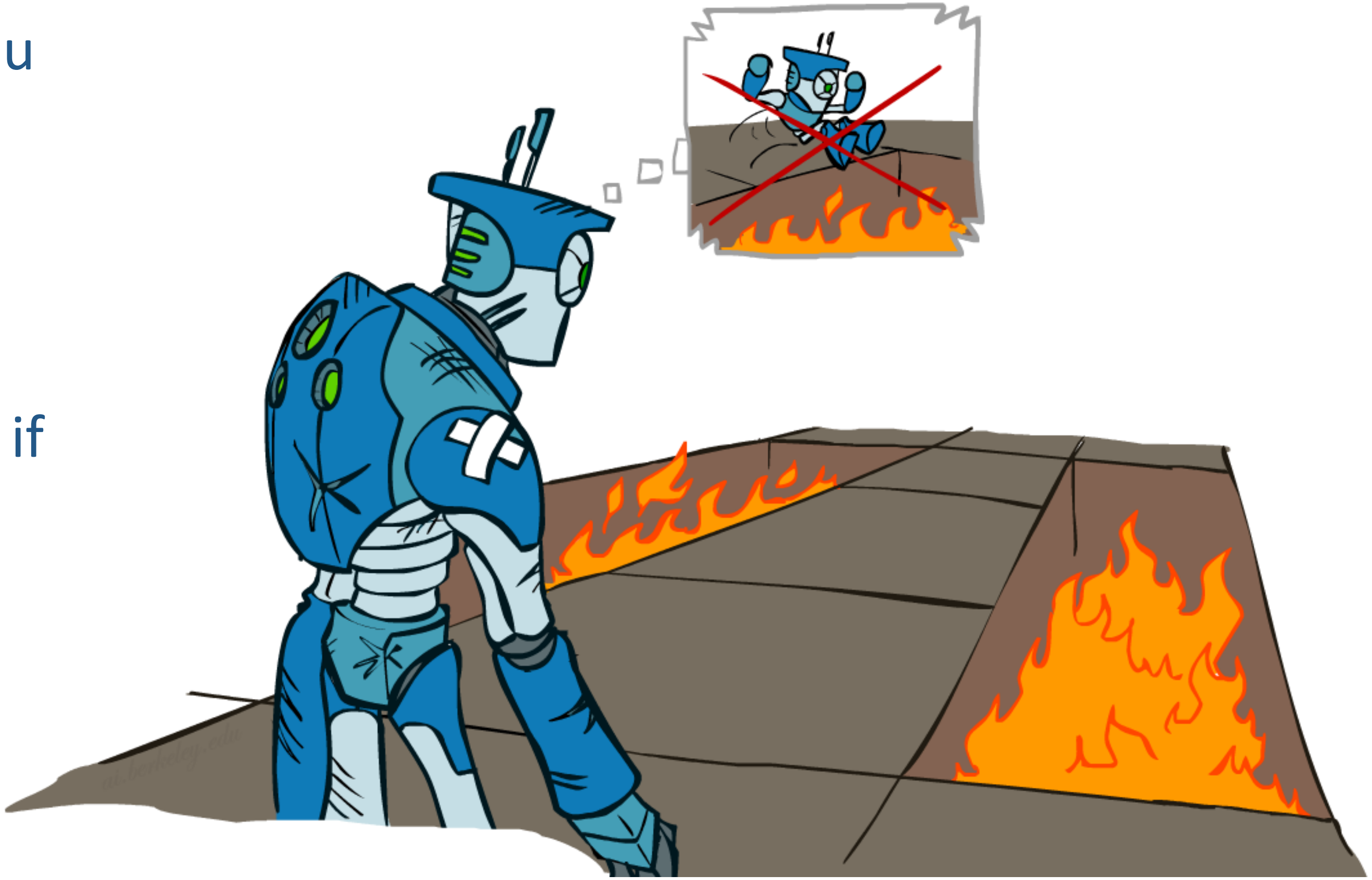
Demo Q-learning – Exploration Function – Crawler

Regret

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

Regret: the difference between your (expected) rewards, including youthful suboptimality, and (expected) rewards if you use an optimal policy in hindsight

Minimizing regret requires optimally learning to be optimal



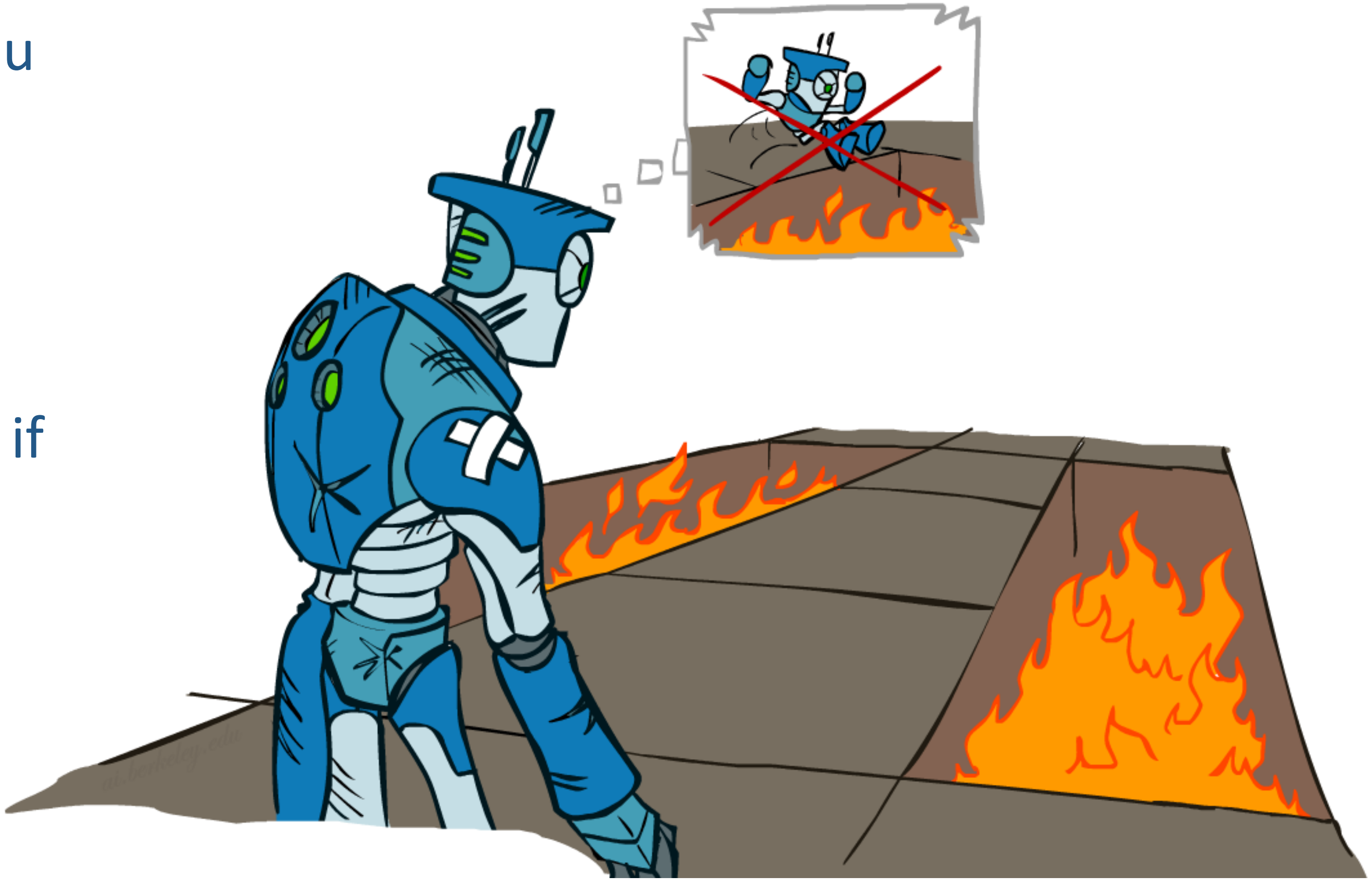
Which one has higher regret: random exploration or using exploration function $f(u,n) = u + \frac{k}{n}$?

Regret

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

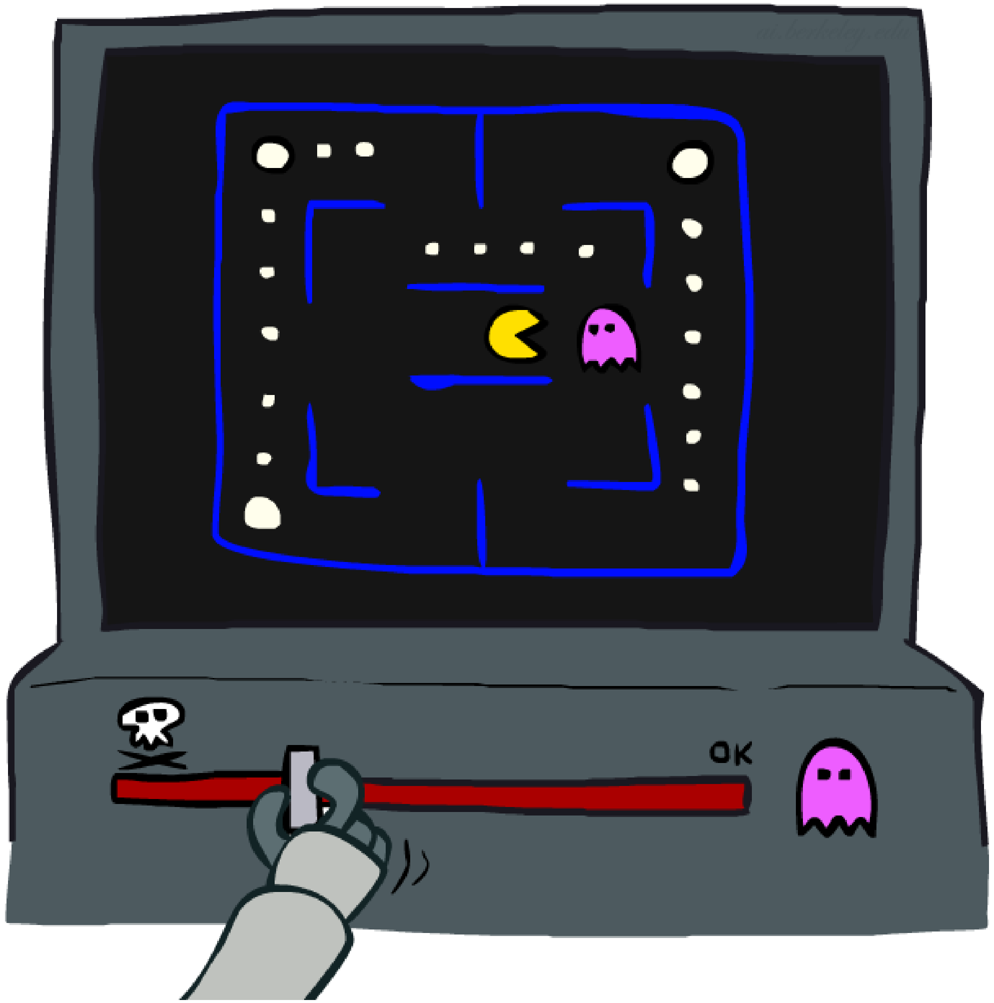
Regret: the difference between your (expected) rewards, including youthful suboptimality, and (expected) rewards if you use an optimal policy in hindsight

Minimizing regret requires optimally learning to be optimal



Which one has higher regret: random exploration or using exploration function $f(u,n) = u + \frac{k}{n}$? The former

Approximate Q-Learning



Generalizing Across States

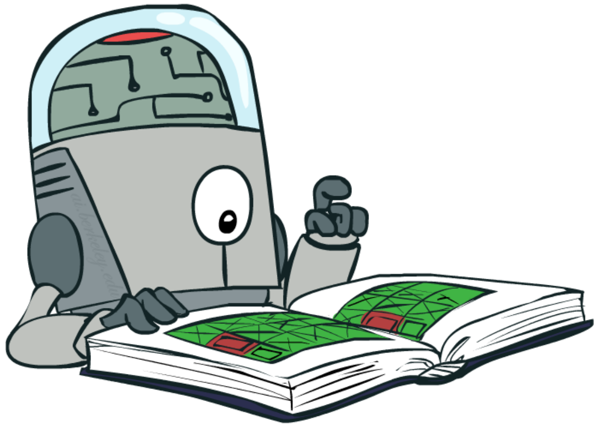
Basic Q-Learning keeps a table of all q-values

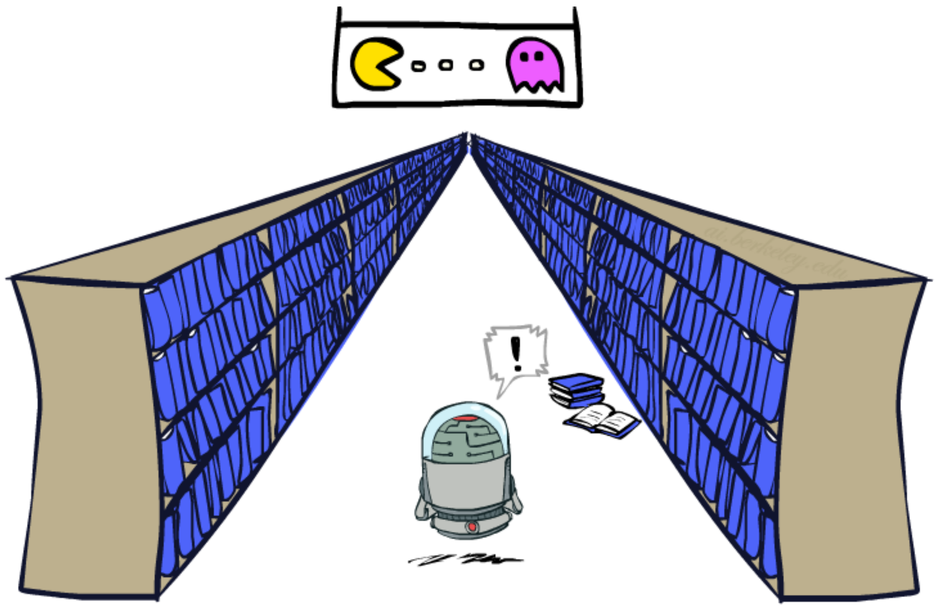
In realistic situations, we cannot possibly learn about every single state!

- 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

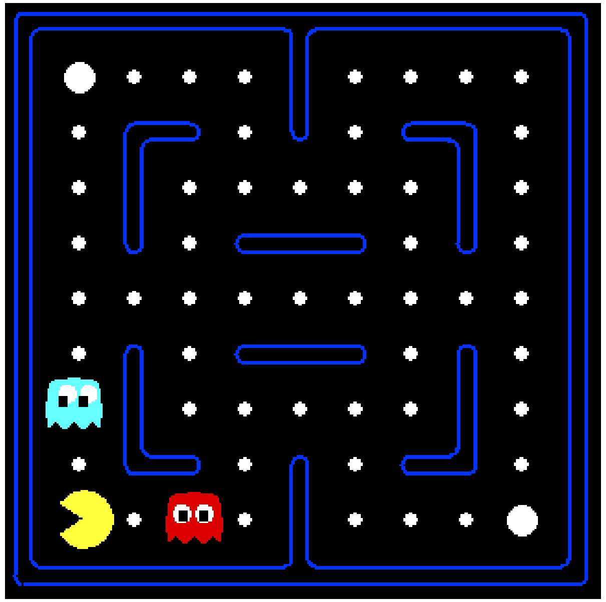




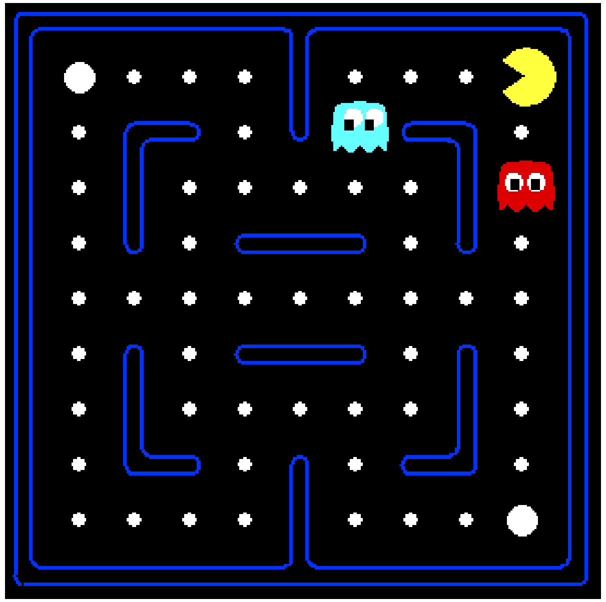
[demo – RL pacman]

Example: Pacman

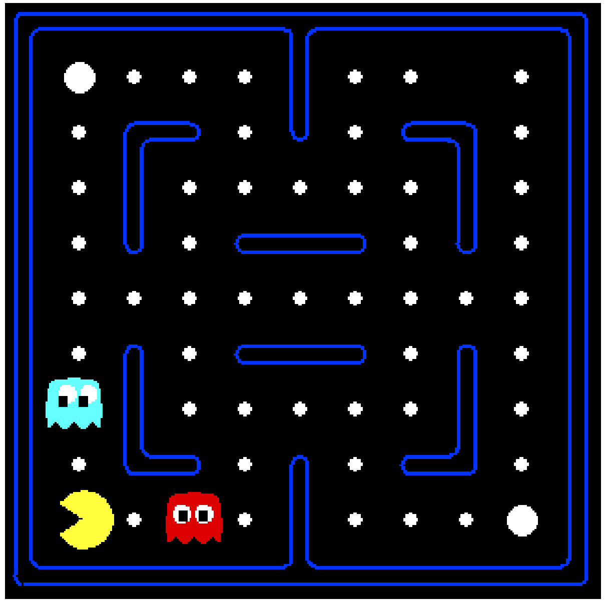
Let's say we discover through experience that this state is bad:



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

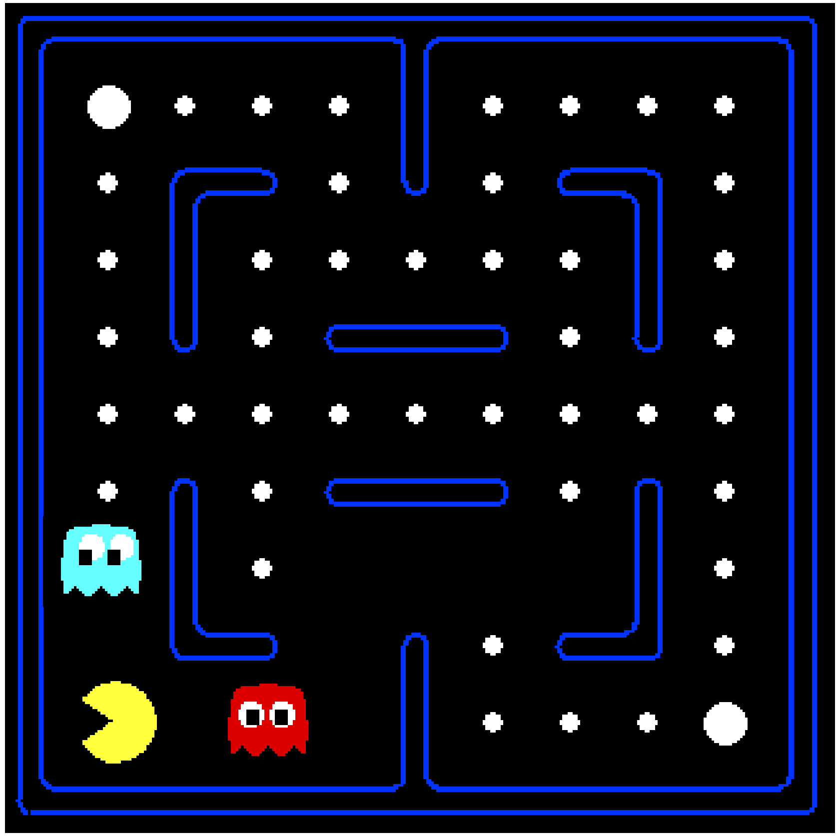
Demo Q-Learning Pacman – Tiny – Silent Train

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-value function (or state value function) to approximate the Q-value (or state value) for any state using a few weights:

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

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

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

Approximate Q-Learning: Q-Learning with Q-value function (a.k.a. Q-function)

Piazza Poll 2

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

Piazza Poll 2

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)

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

Latest sample — Previous estimate

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:

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

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

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

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

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

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:

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

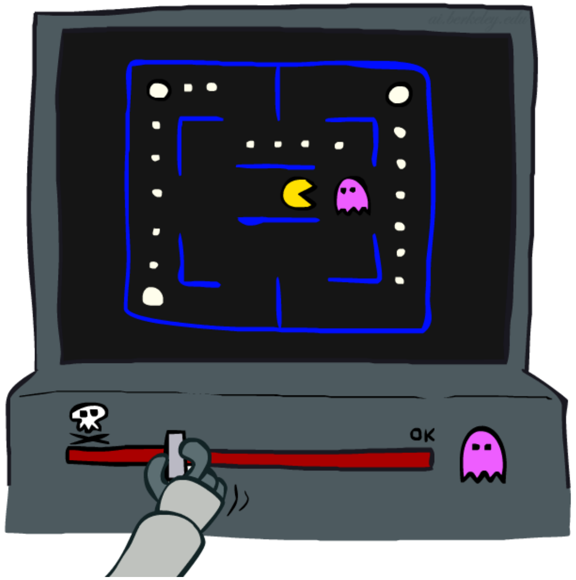
Approximate Q-Learning

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:

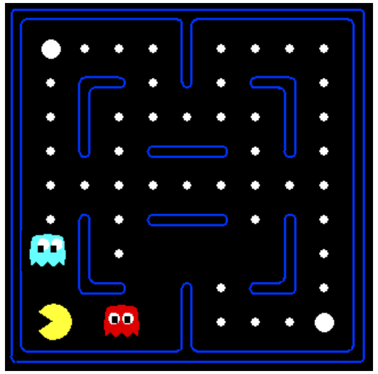
- Pleasant surprise: increase weights on +valued features, decrease on – ones
  - As a result, Q_w increased for states with the same (similar) features too. Will now prefer all states with that state's features.
- Unpleasant surprise: decrease weights on +valued features, increase on – ones
  - Disprefer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



Example: Q-Pacman

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

s

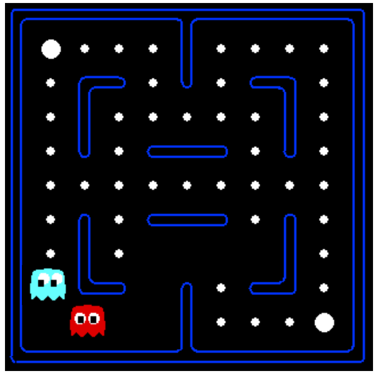


$f_{DOT}(s, \text{NORTH}) = 0.5$

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

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

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

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 $Q_w(s,a)$:

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?

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

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

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

What if non-linear value function?

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

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

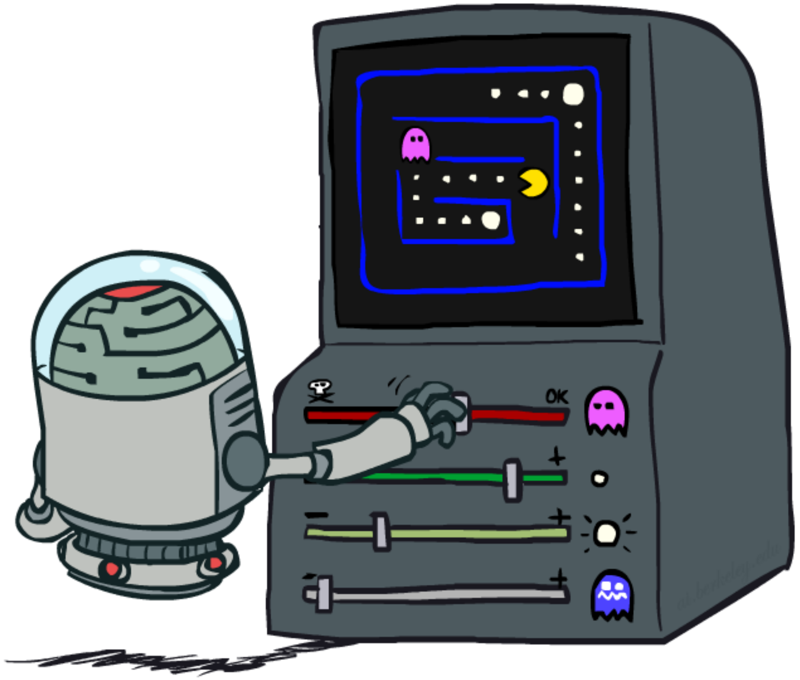
Example: $Q_w(s,a) = \exp(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a))$

$$\frac{\partial Q_w(s,a)}{\partial w_i} = \exp\left(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)\right) f_i(s,a)$$
$$= Q_w(s,a) f_i(s,a)$$

Update Rule:

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

Recent Reinforcement Learning Milestones



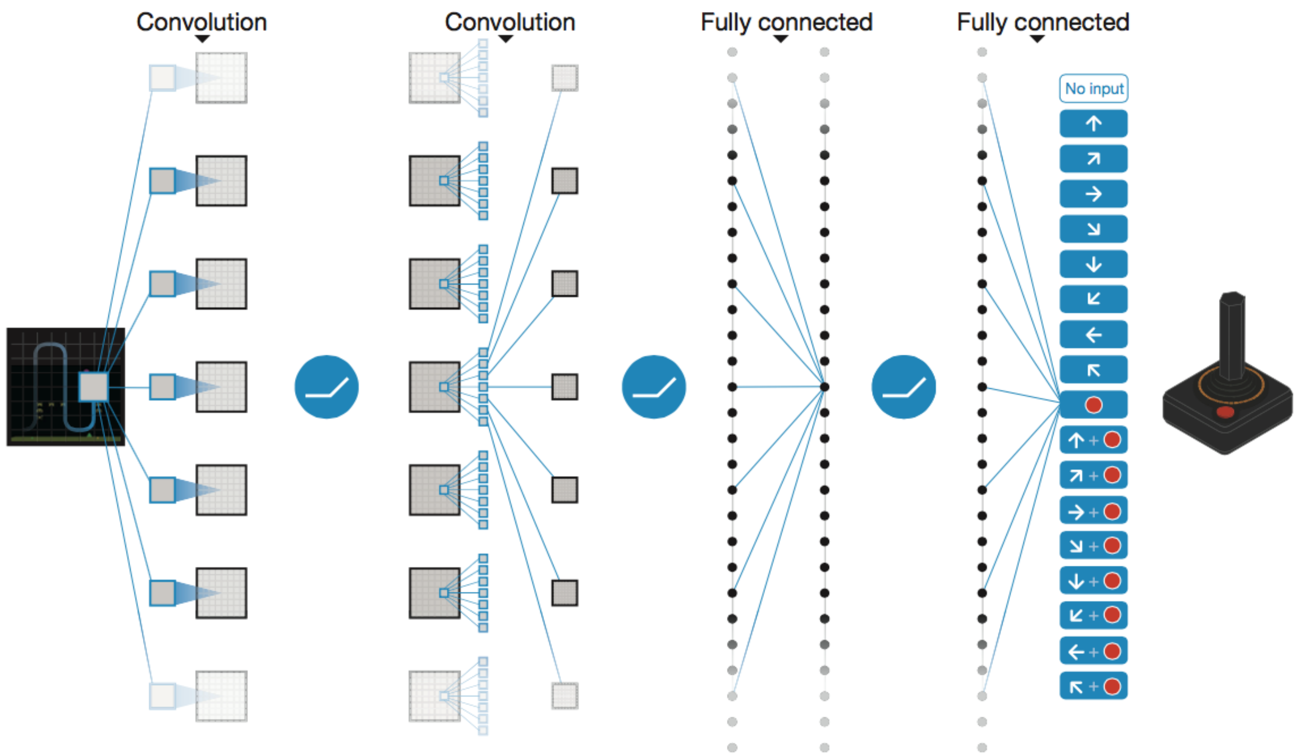
Deep Q-Networks

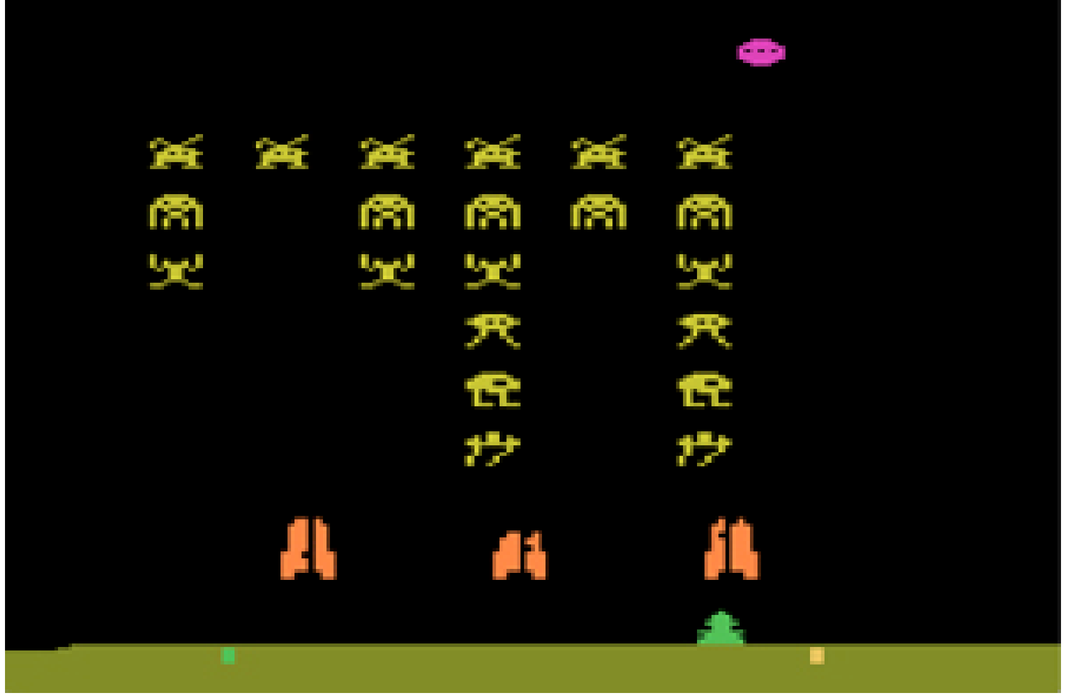
Deep Mind, 2015

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






0
OXYGEN
ACTIVISION


0000025
01
180
142 KM/H

OpenAI Gym

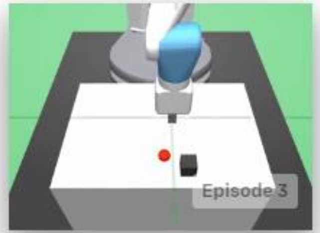
2016+

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.

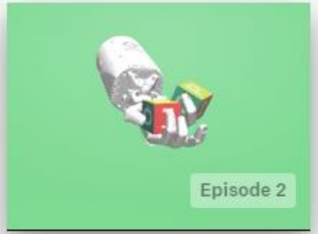


FetchPush-v0
Push a block to a goal position.

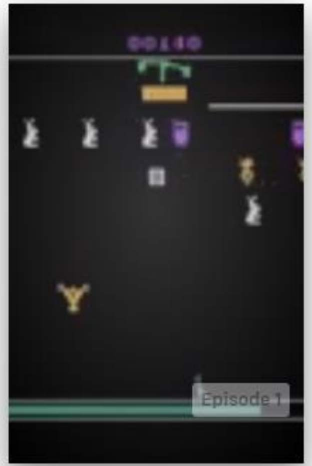
Breakout-ram-v0
Maximize score in the game Breakout, with RAM as input

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

TDGammon

1992 by Gerald Tesauro, IBM

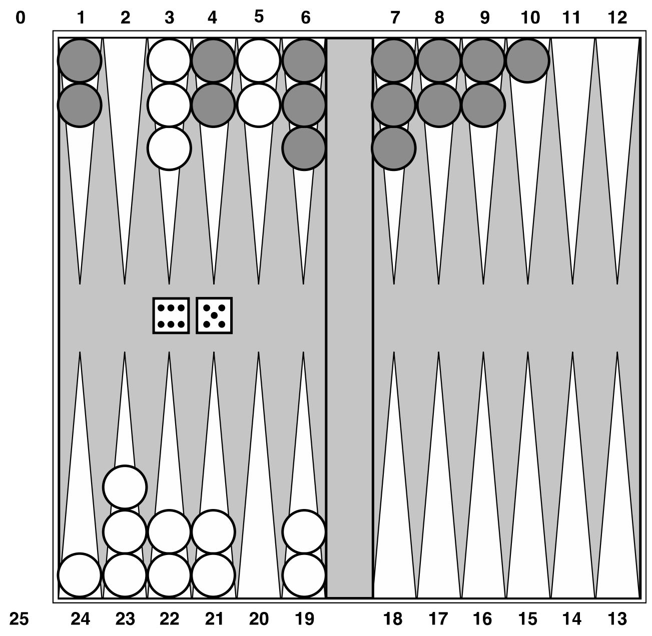
4-ply lookahead using V(s) trained from 1,500,000 games of self-play

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



AlphaGo, AlphaZero

Deep Mind, 2016+

(Deep) Q-Learning for Combating (Naïve) Poacher

Wildlife protection

- Rangers make flexible decisions instead of sticking to a fixed patrol route
- Rangers and poachers may leave traces as they move

Footprints

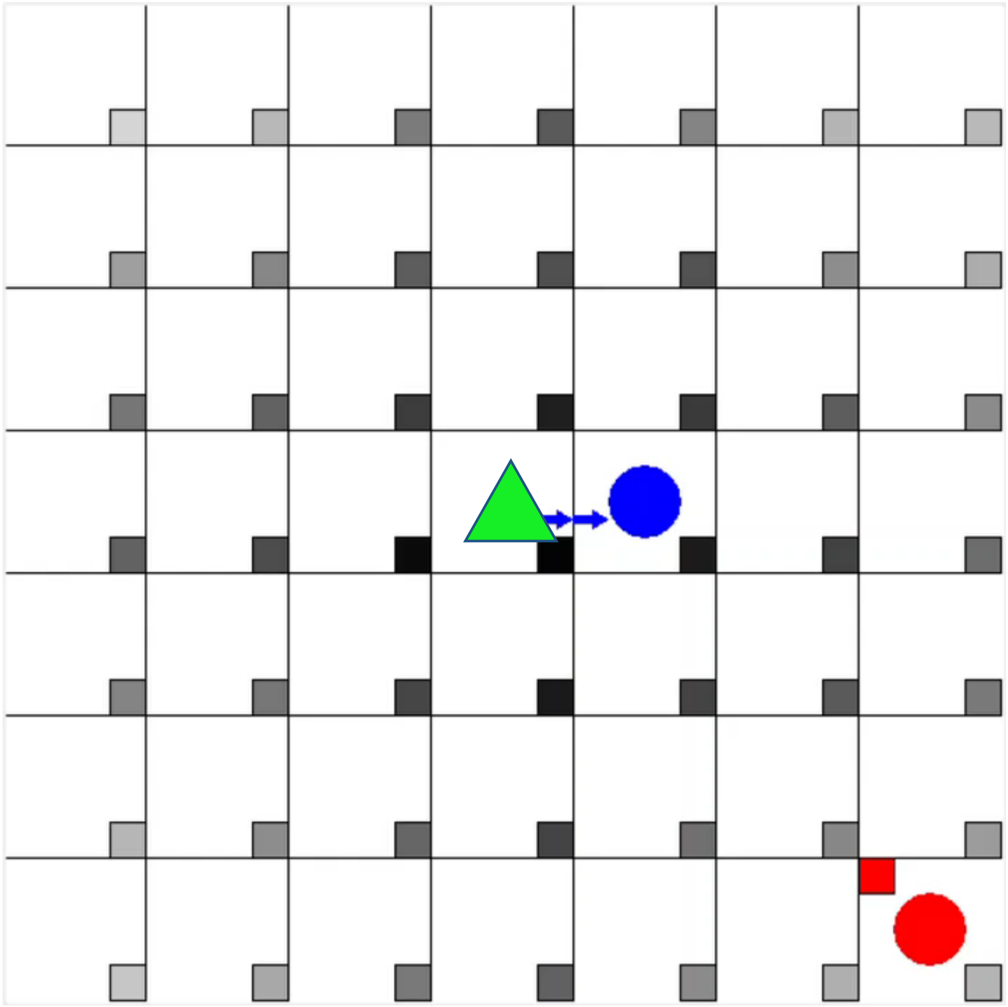
Lighters

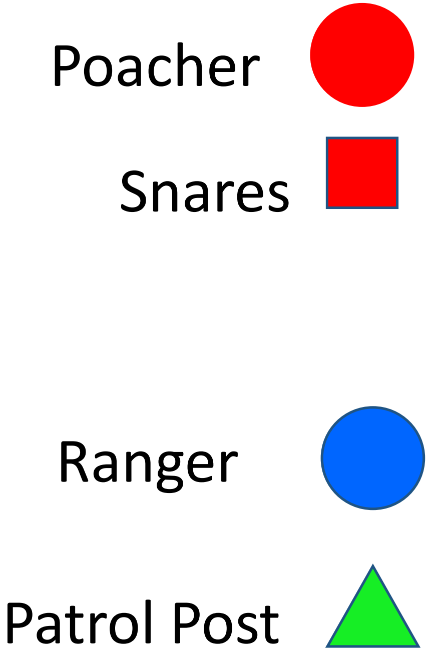
Old poacher camp

Tree marking

Deep Reinforcement Learning for Green Security Games with Real-Time Information Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, Fei Fang In AAAI-19

(Deep) Q-Learning for Combating (Naïve) Poacher

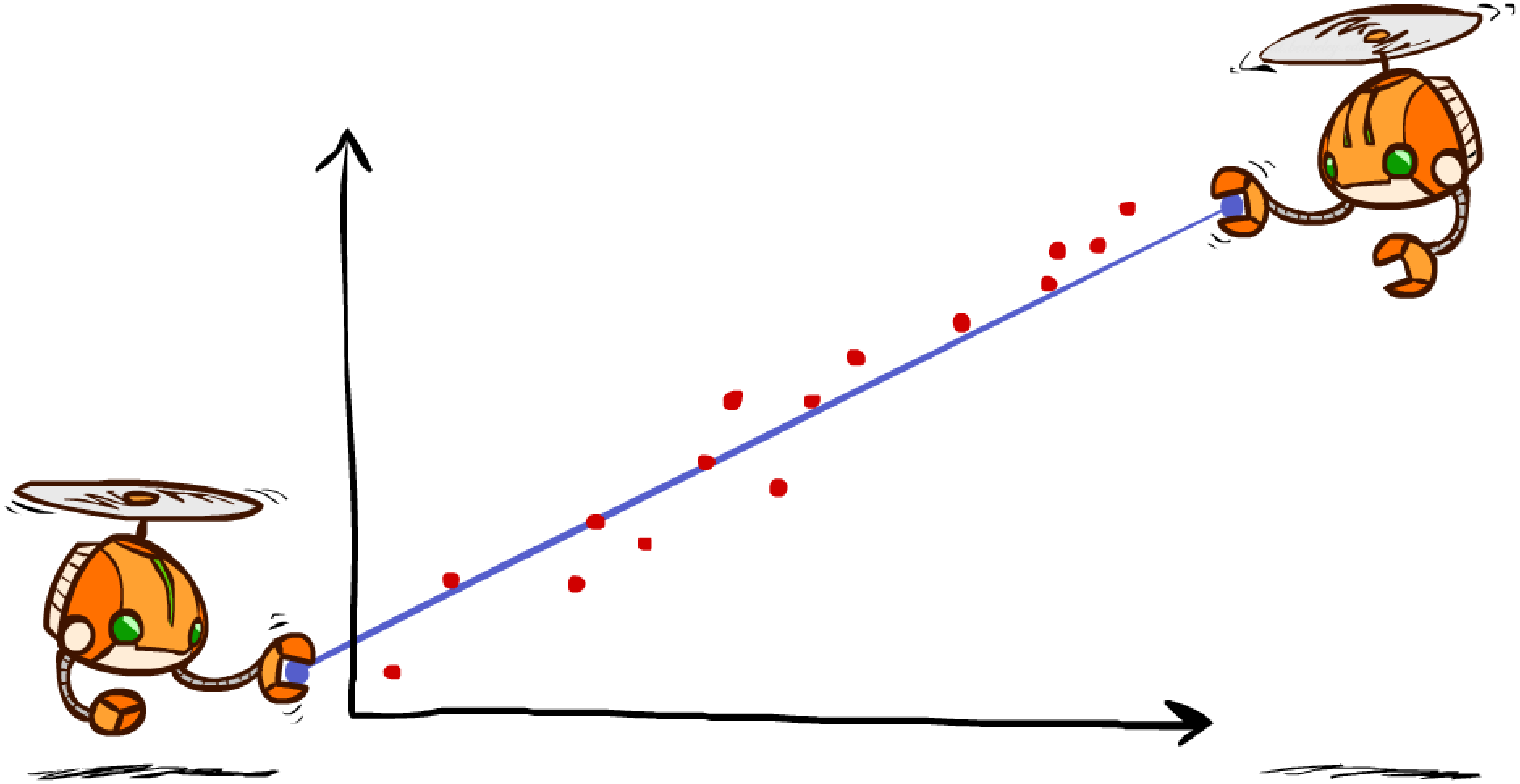




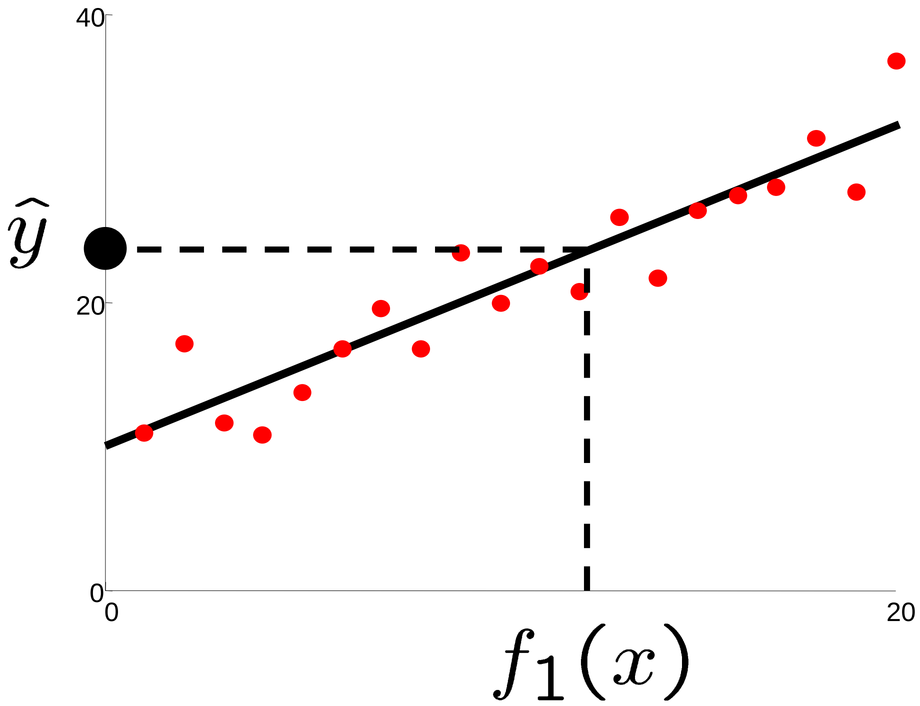
Autonomous Vehicles?

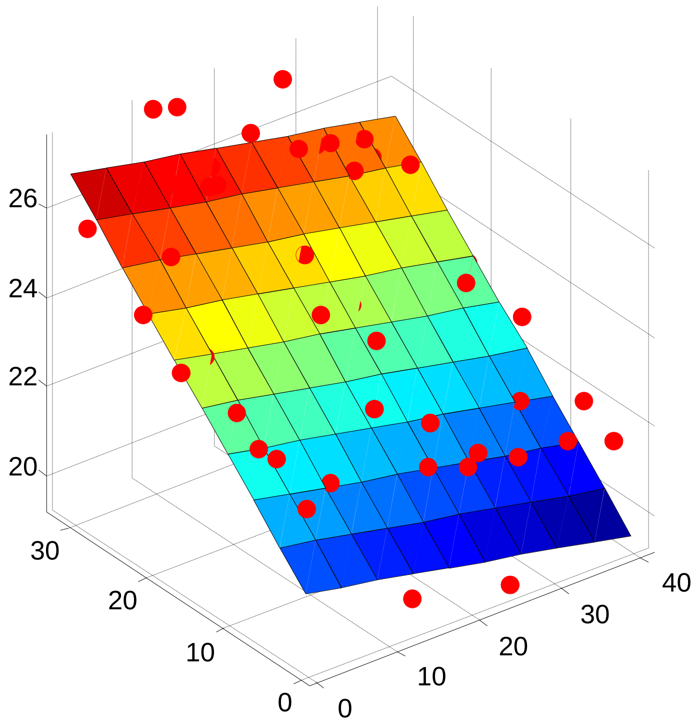
Backup Slides

Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

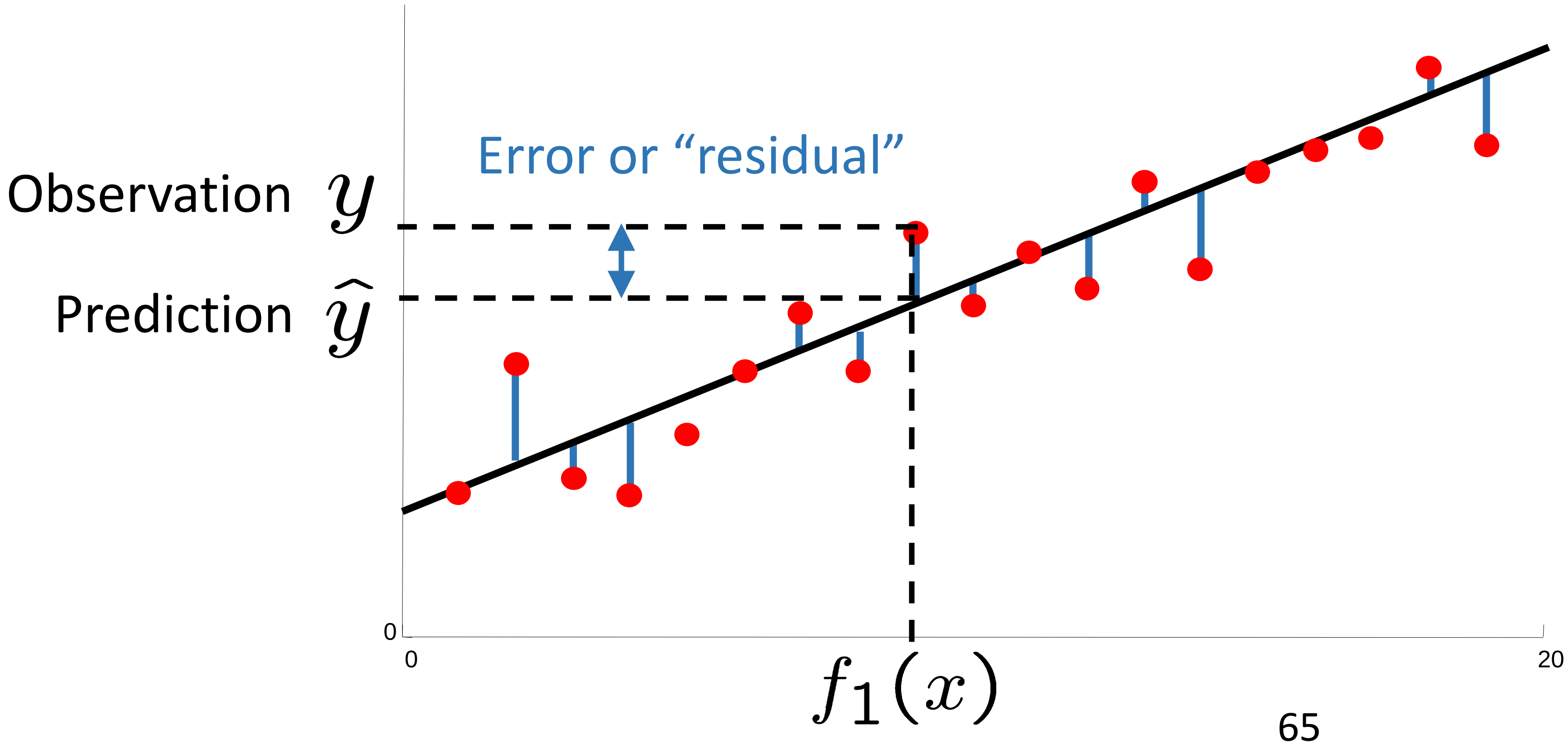
$$\hat{y} = w_0 + w_1 f_1(x)$$

Prediction:

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

Optimization: Least Squares

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



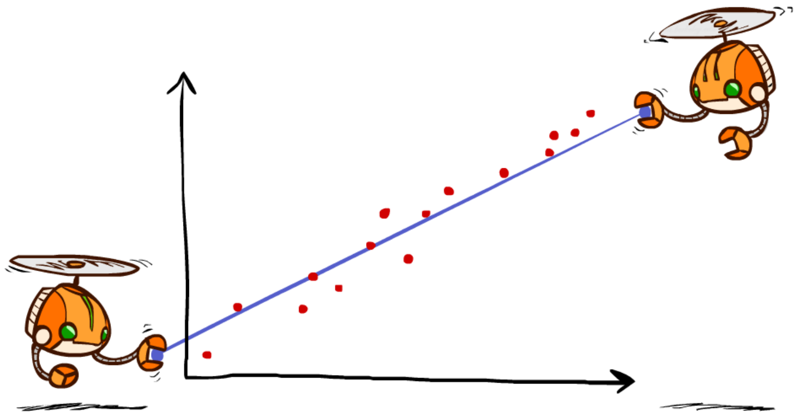
Minimizing Error

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"