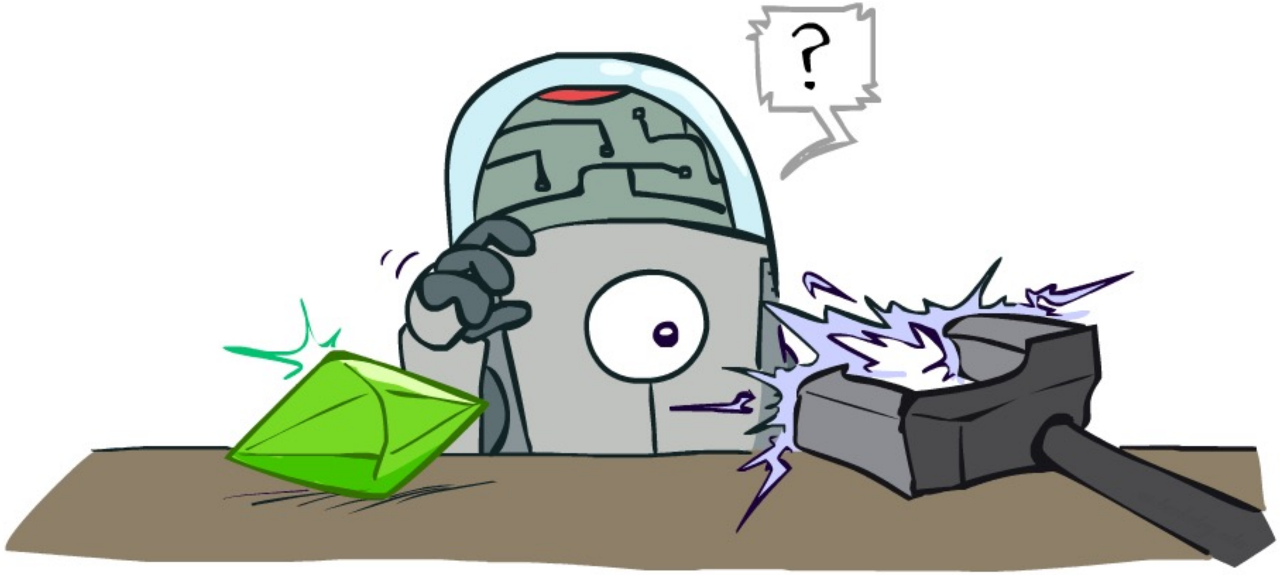
Artificial Intelligence - INFOF311

Reinforcement Learning part 2

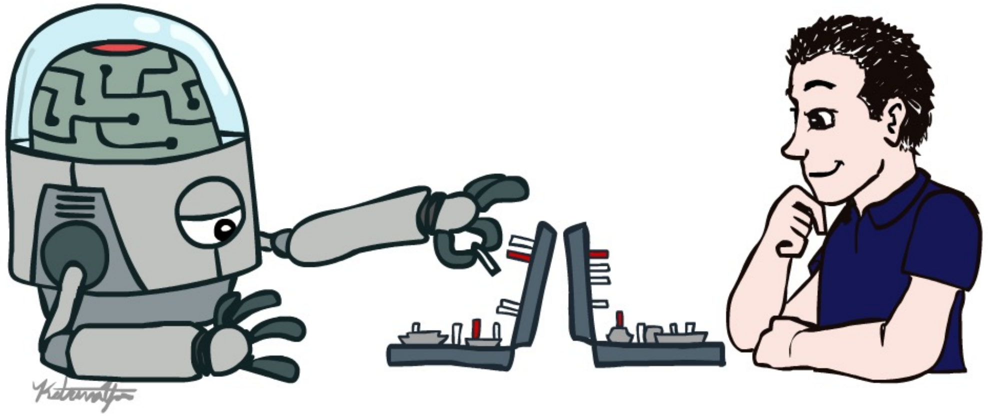


Instructor : Tom Lenaerts

Acknowledgement

We thank Stuart Russell for his generosity in allowing us to use the slide set of the UC Berkeley Course CS188, Introduction to Artificial Intelligence. These slides were created by Dan Klein, Pieter Abbeel and Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.

Center for
Human-Compatible
Artificial
Intelligence



The slides for INFOF311 are slightly modified versions of the slides of the spring and summer CS188 sessions in 2021 and 2022

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states s ∈ 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 π(s)

Cool

Warm

Overheated

- New twist: don't know T or R, so try out actions
- Big Idea : Compute all averages over T using sample outcomes

The story so far …

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

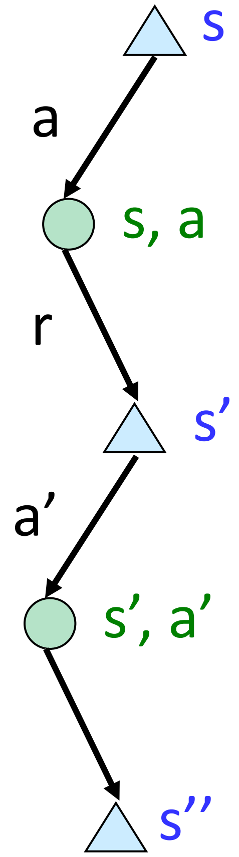
Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	Value Learning

Temporal Difference Value 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



Q-Learning

- Q-Learning: sample-based Q-value iteration

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

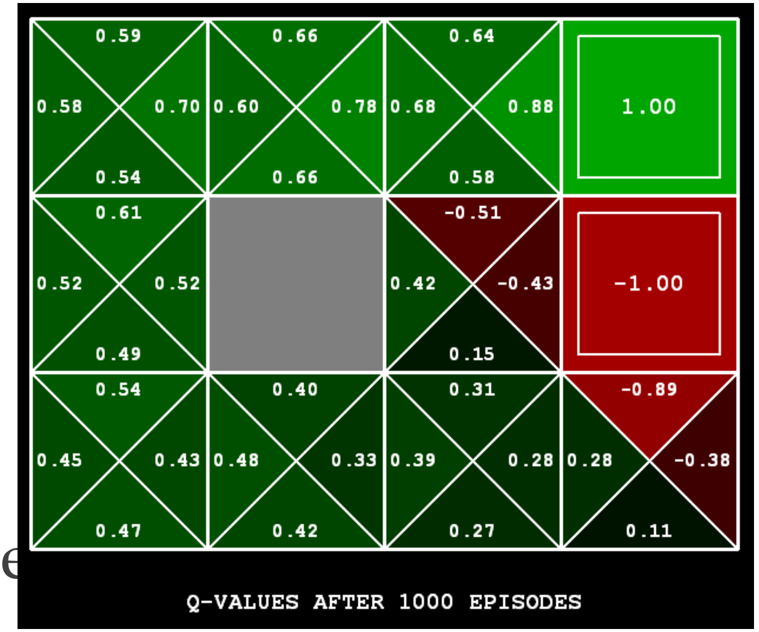
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s’,r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

no longer policy evaluation!

  - Incorporate the new estimate into a running average

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



Q-VALUES AFTER 1000 EPISODES

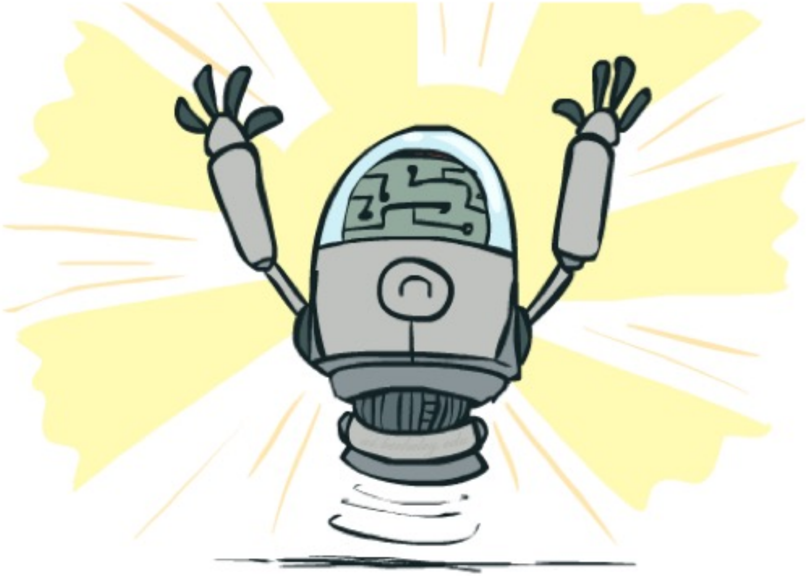
[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

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 (!)

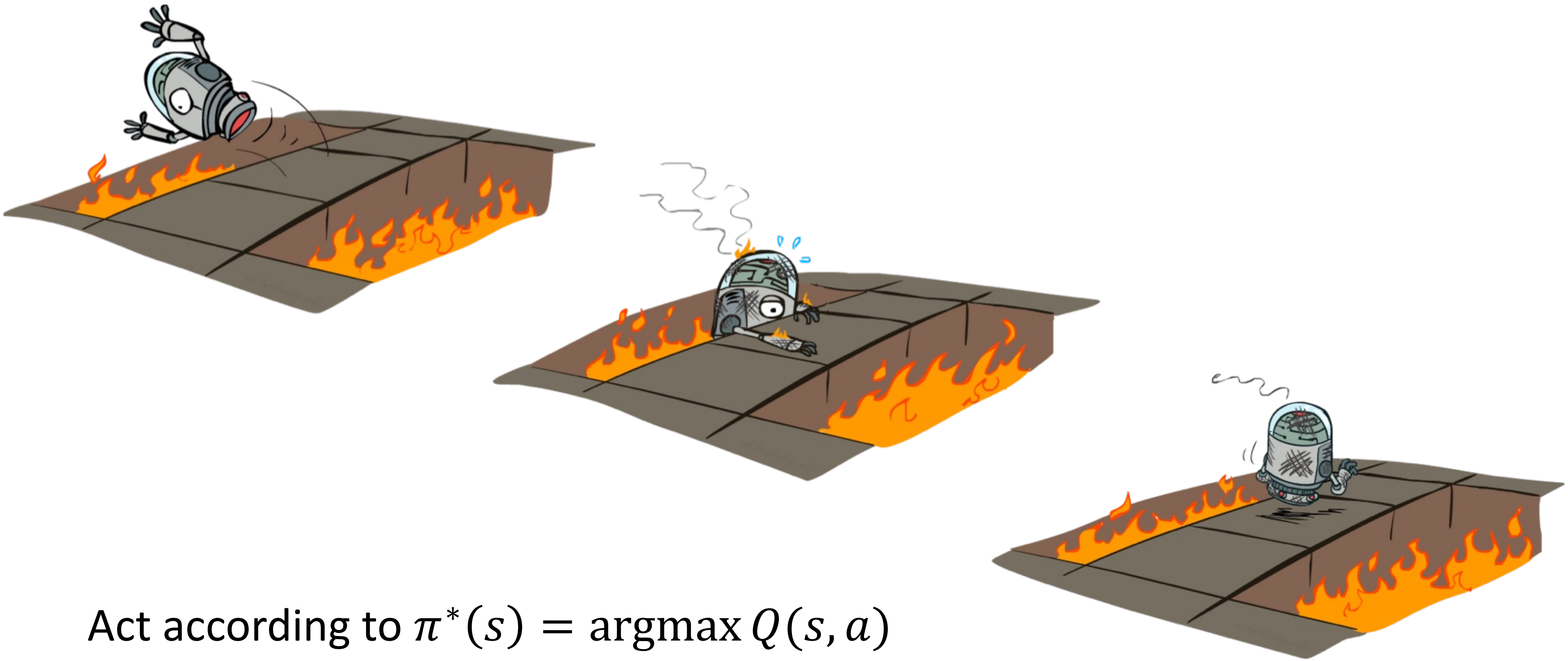


Video of Demo Q-Learning -- Crawler

Reinforcement Learning -- Overview

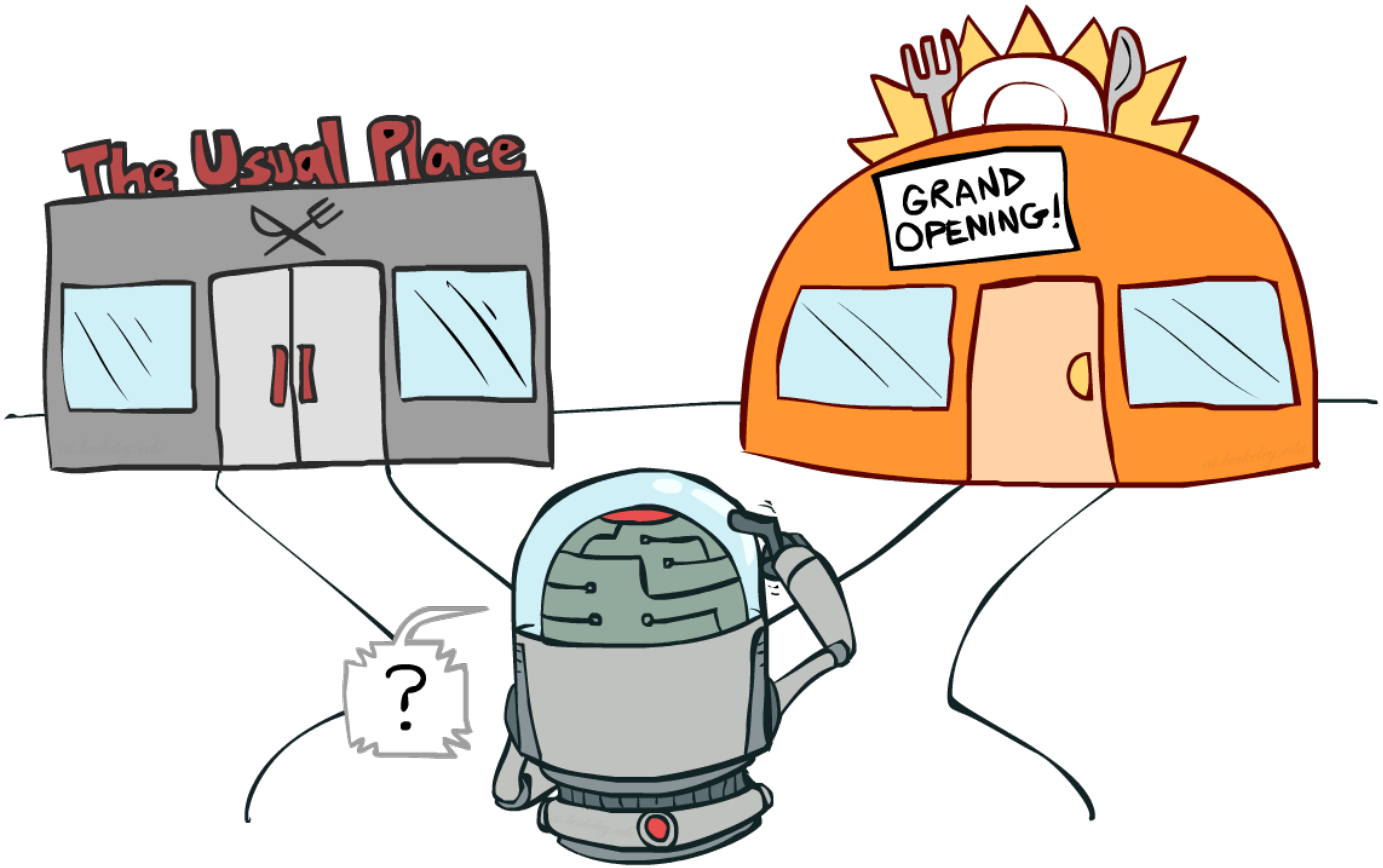
- Passive Reinforcement Learning (= how to learn from experiences)
  - Model-based Passive RL
    - Learn the MDP model from experiences, then solve the MDP
  - Model-free Passive RL
    - Forego learning the MDP model, directly learn V or Q:
      - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
      - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- **Active Reinforcement Learning (= agent also needs to decide how to collect experiences)**
  - Key challenges:
    - How to efficiently explore?
    - How to trade off exploration <> exploitation
  - Applies to both model-based and model-free. We'll cover only in context of Q-learning

Active Reinforcement Learning



Act according to $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$

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 (L10D5)]
[Demo: Q-learning – epsilon-greedy -- crawler (L10D3)]

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

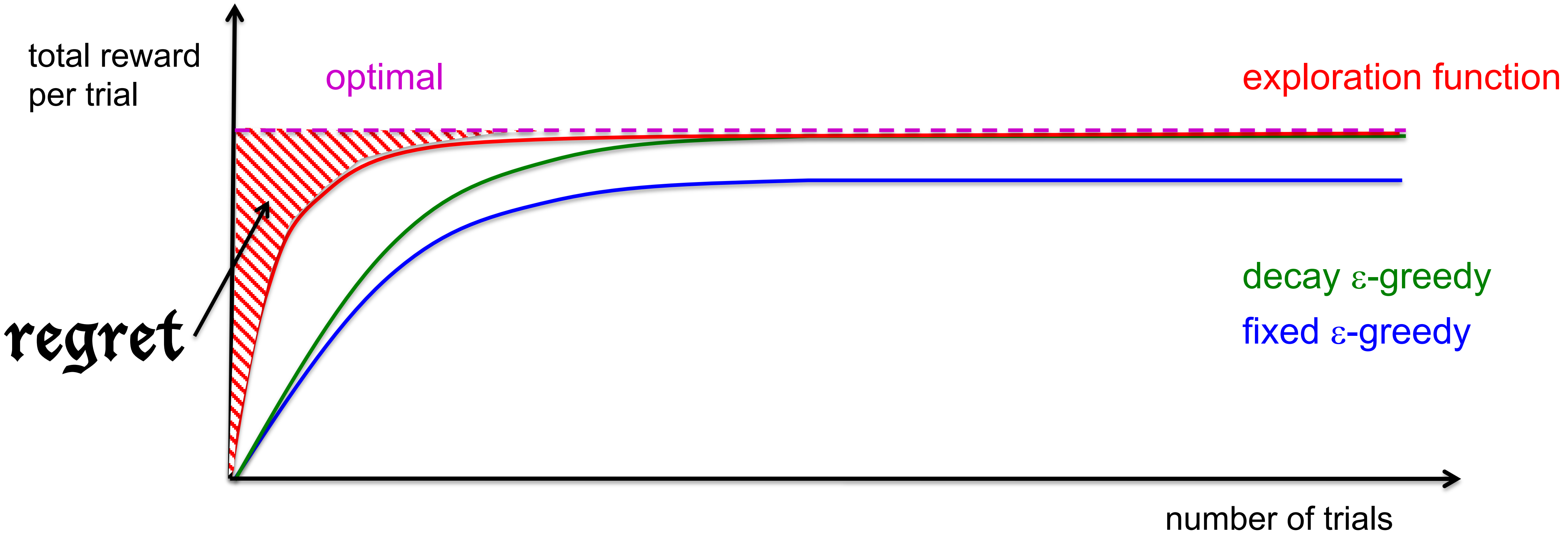
Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

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

[Demo: exploration – Q-learning – crawler – exploration function (L10D4)]

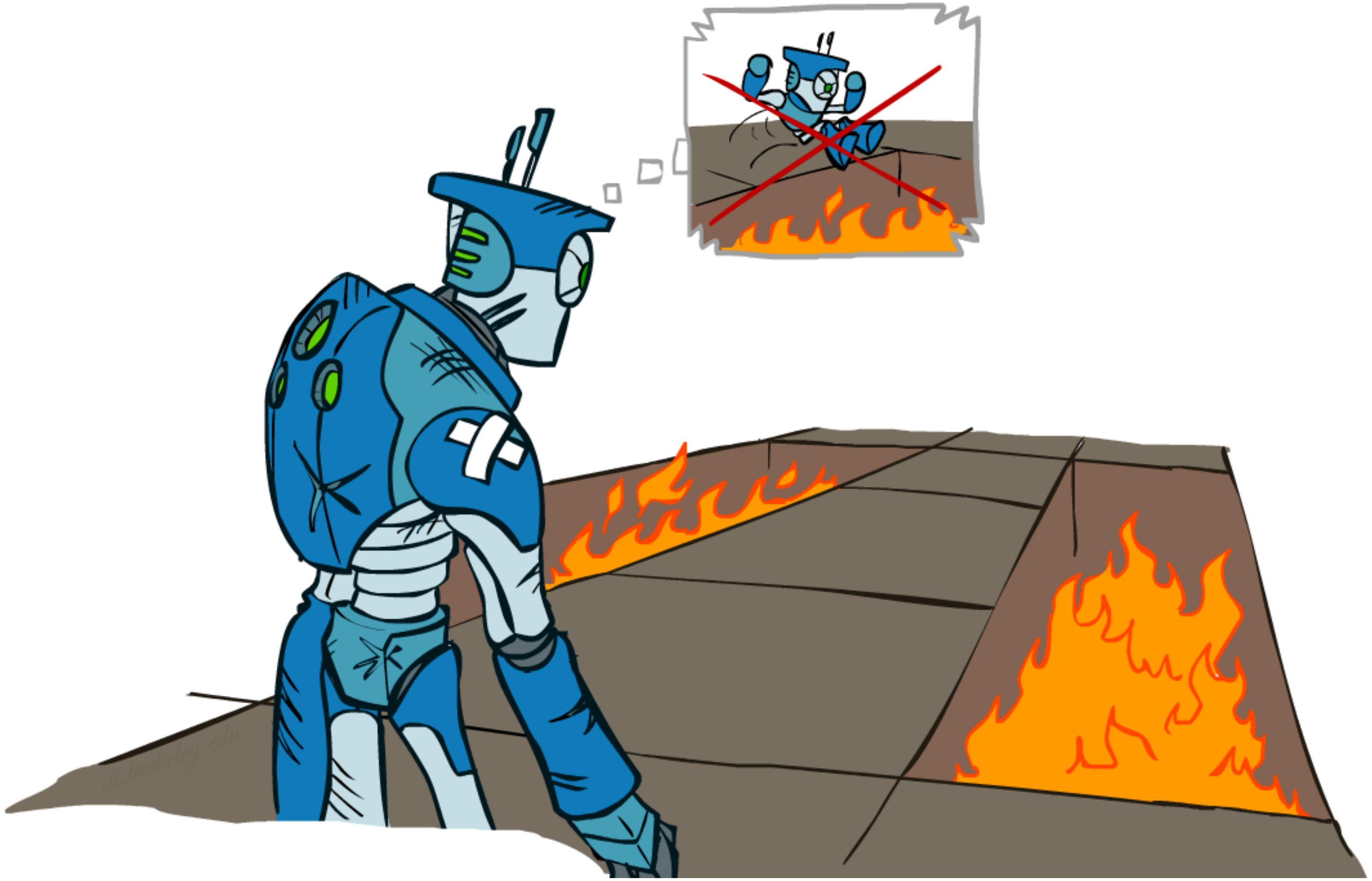
Video of Demo Q-learning – Exploration Function – Crawler

Optimality and exploration



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



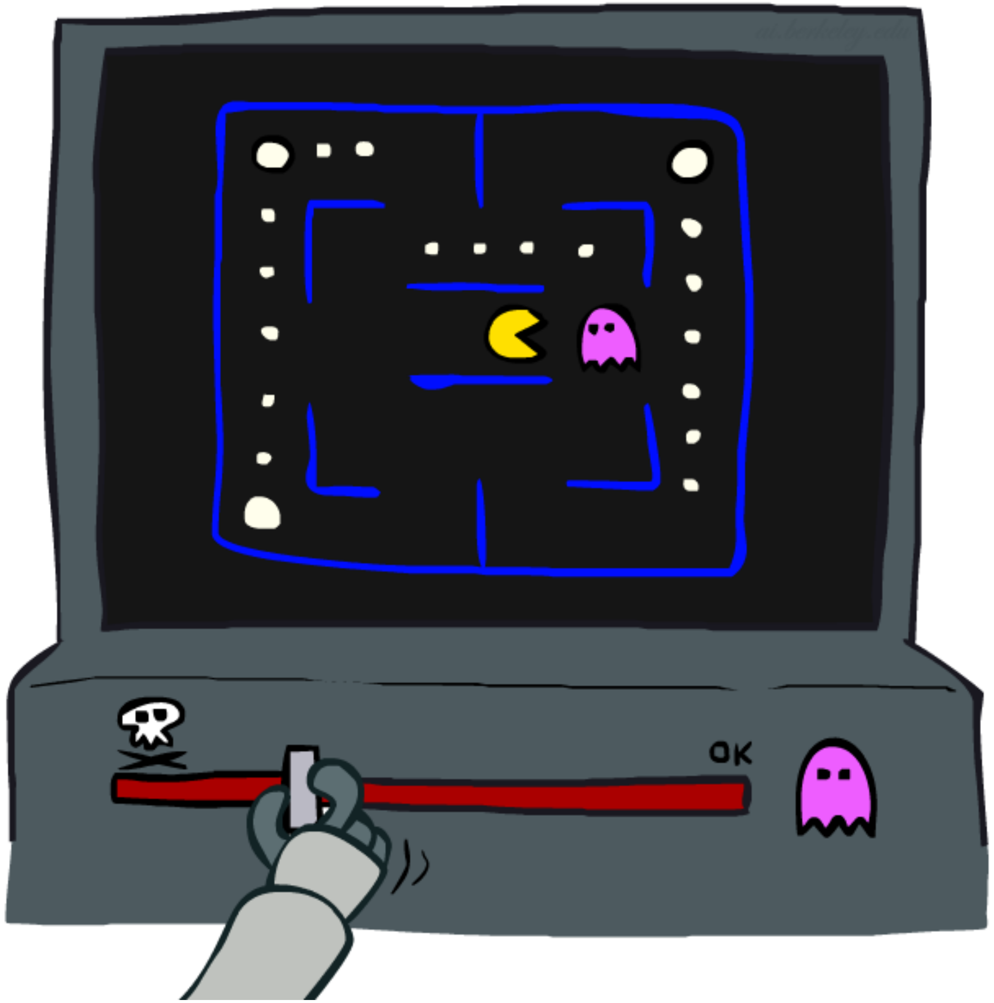
Reinforcement Learning -- Overview

- Passive Reinforcement Learning (= how to learn from experiences)
  - Model-based Passive RL
    - Learn the MDP model from experiences, then solve the MDP
  - Model-free Passive RL
    - Forego learning the MDP model, directly learn V or Q:
      - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
      - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
  - Key challenges:
    - How to efficiently explore?
    - How to trade off exploration <> exploitation
  - Applies to both model-based and model-free. We’ll cover only in context of Q-learning

Reinforcement Learning -- Overview

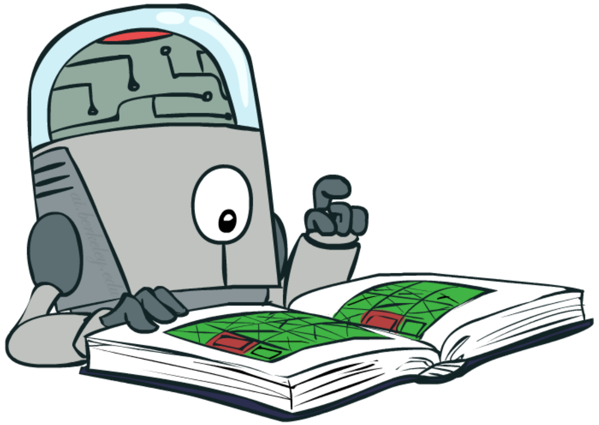
- Passive Reinforcement Learning (= how to learn from experiences)
  - Model-based Passive RL
    - Learn the MDP model from experiences, then solve the MDP
  - Model-free Passive RL
    - Forego learning the MDP model, directly learn V or Q:
      - Value learning – learns value of a fixed policy; 2 approaches: Direct Evaluation & TD Learning
      - Q learning – learns Q values of the optimal policy (uses a Q version of TD Learning)
- Active Reinforcement Learning (= agent also needs to decide how to collect experiences)
  - Key challenges:
    - How to efficiently explore?
    - How to trade off exploration <> exploitation
  - Applies to both model-based and model-free. In CS188 we'll cover only in context of Q-learning
- **Approximate Reinforcement Learning (= to handle large state spaces)**
  - **Approximate Q-Learning**
  - **Policy Search**

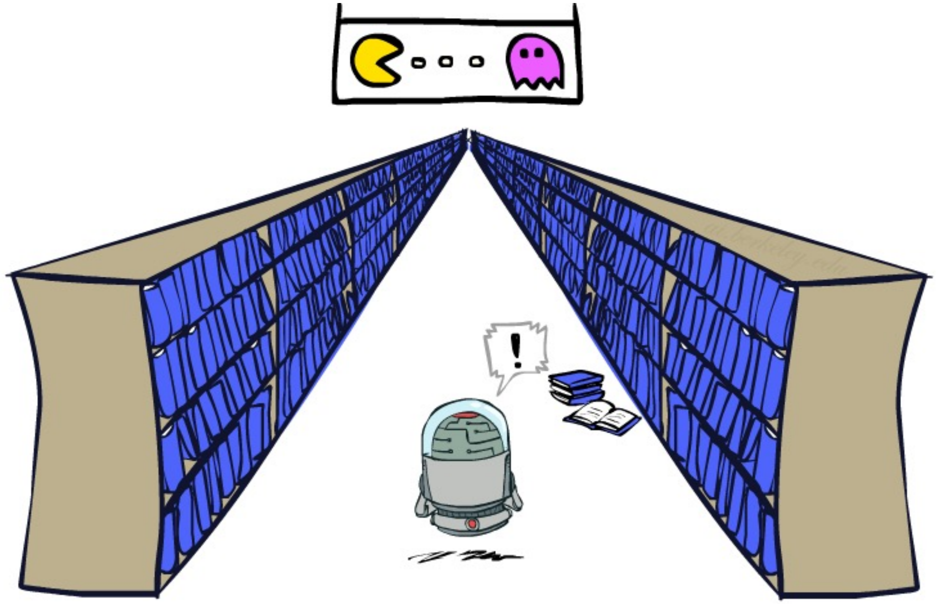
Approximate Q-Learning



Generalizing Across States

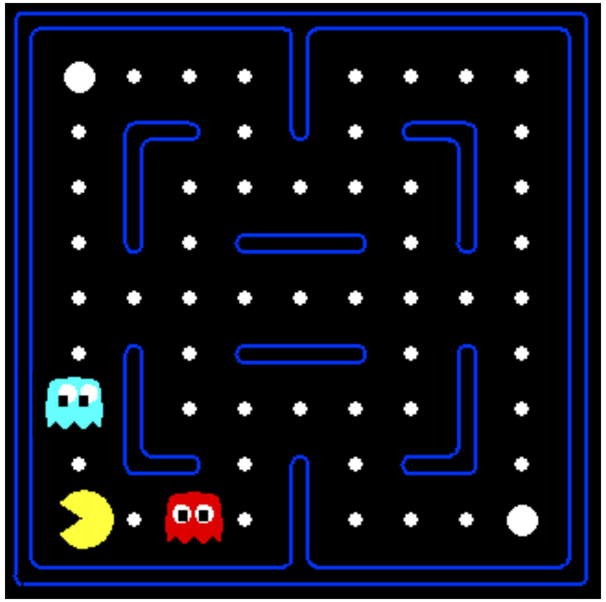
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training 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



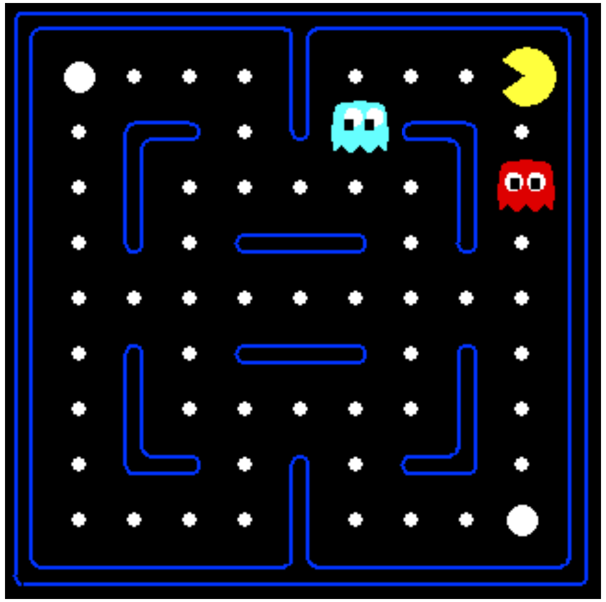


Example: Pacman

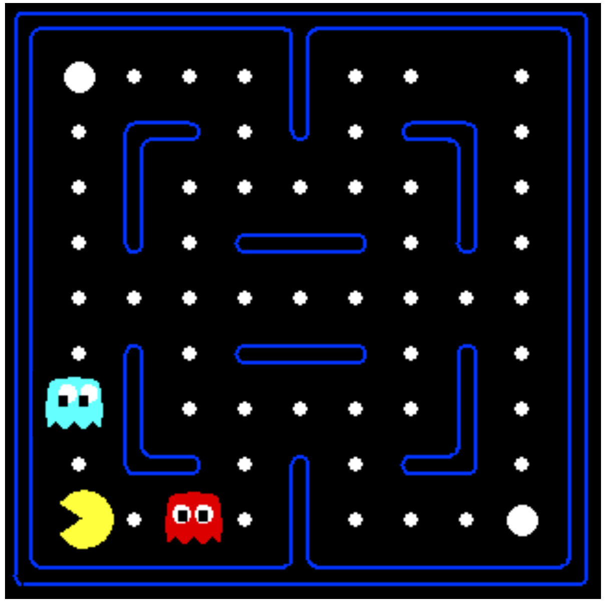
Let's say we discover through experience that this state is bad:



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



Or even this one!



[Demo: Q-learning – pacman – tiny – watch all (L11D4)]
[Demo: Q-learning – pacman – tiny – silent train (L11D6)]
[Demo: Q-learning – pacman – tricky – watch all (L11D5)]

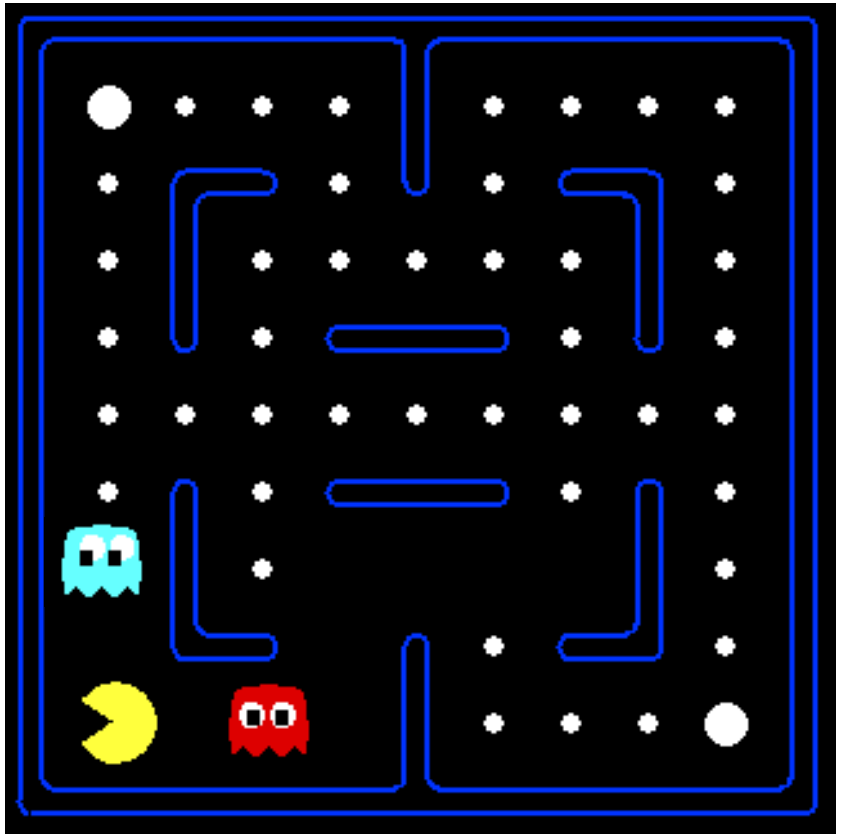
Video of Demo Q-Learning Pacman – Tiny – Watch All

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 V 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!

Approximate Q-Learning

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

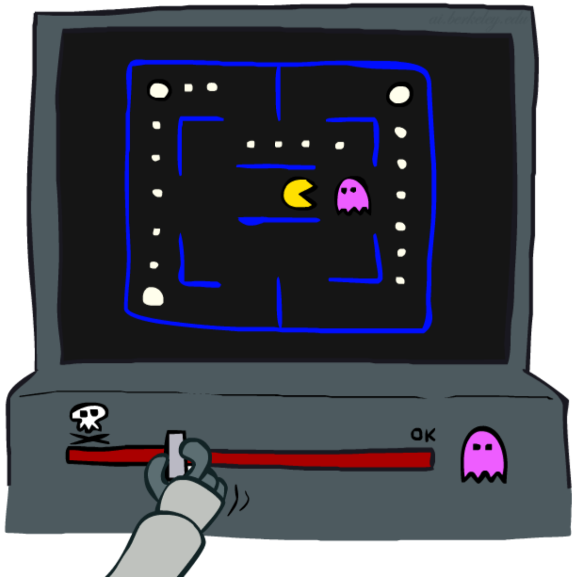
- Q-learning with linear Q-functions:

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

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

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

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



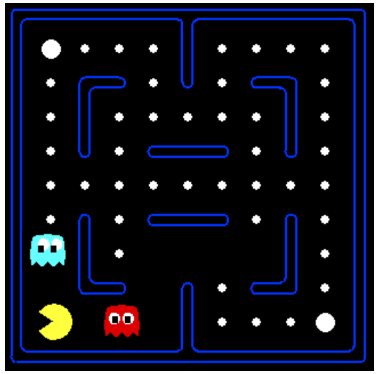
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Example: Q-Pacman

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

s



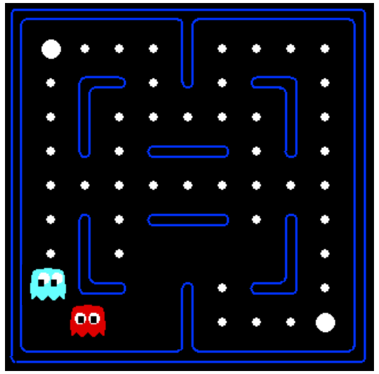
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

difference $= -501$

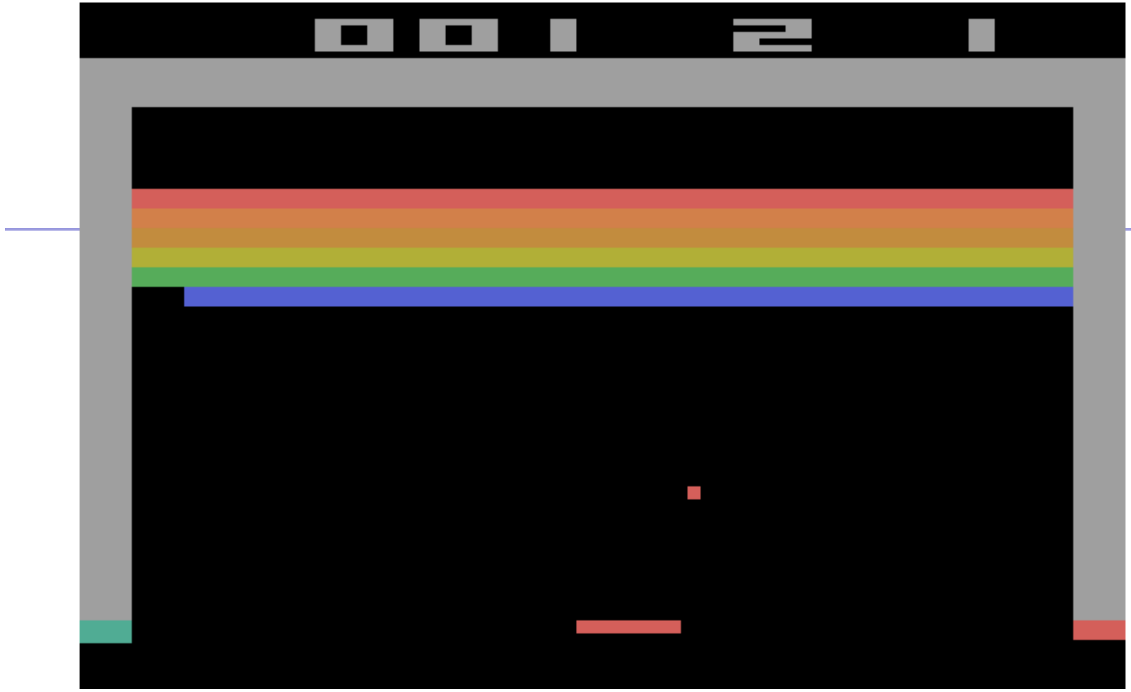
$w_{DOT} \leftarrow 4.0 + \alpha \left[-501\right] 0.5$

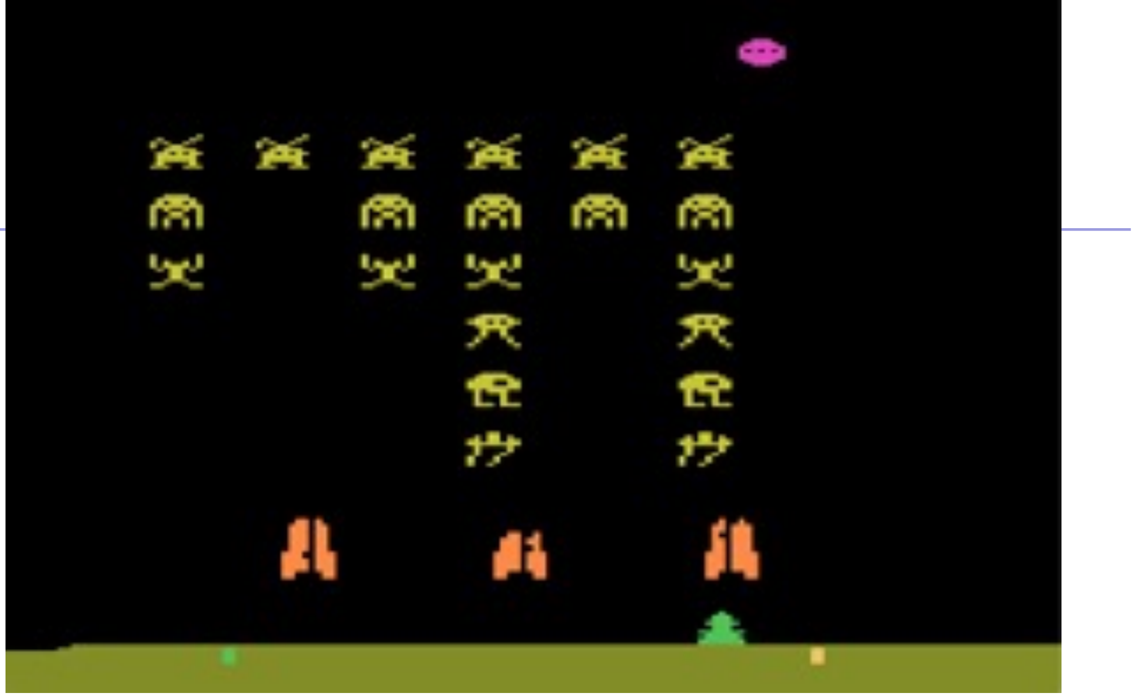
$w_{GST} \leftarrow -1.0 + \alpha \left[-501\right] 1.0$

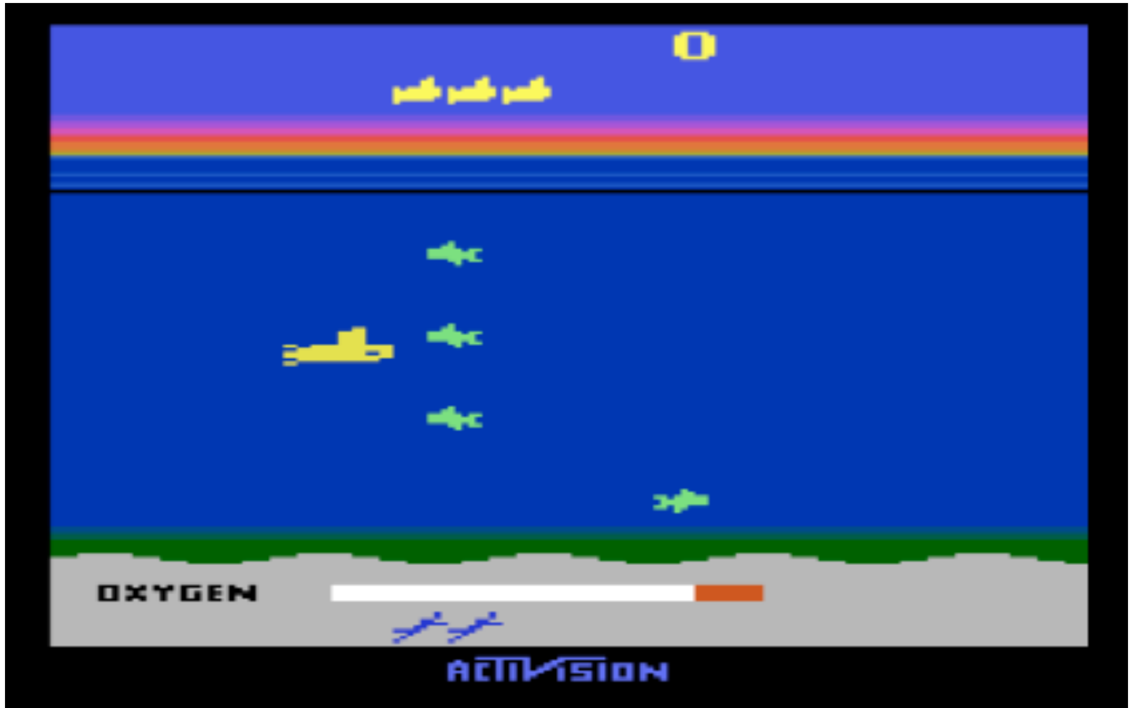
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

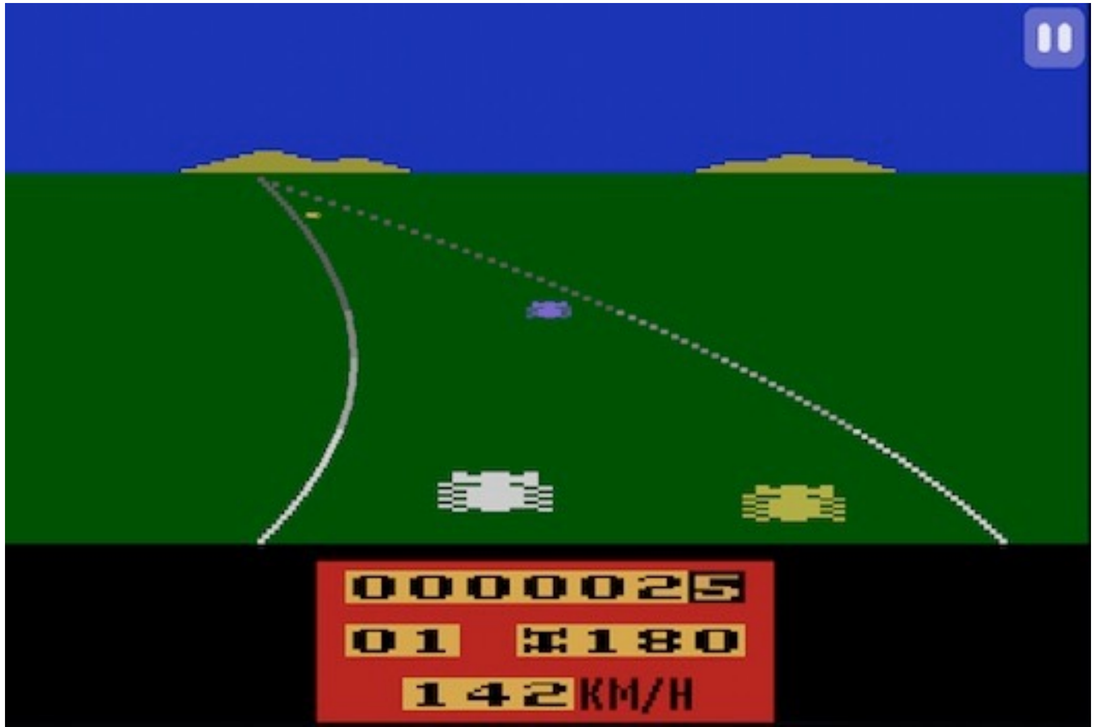
[Demo: approximate Q-learning pacman (L11D8)]

Video of Demo Approximate Q-Learning -- Pacman


001
2
1

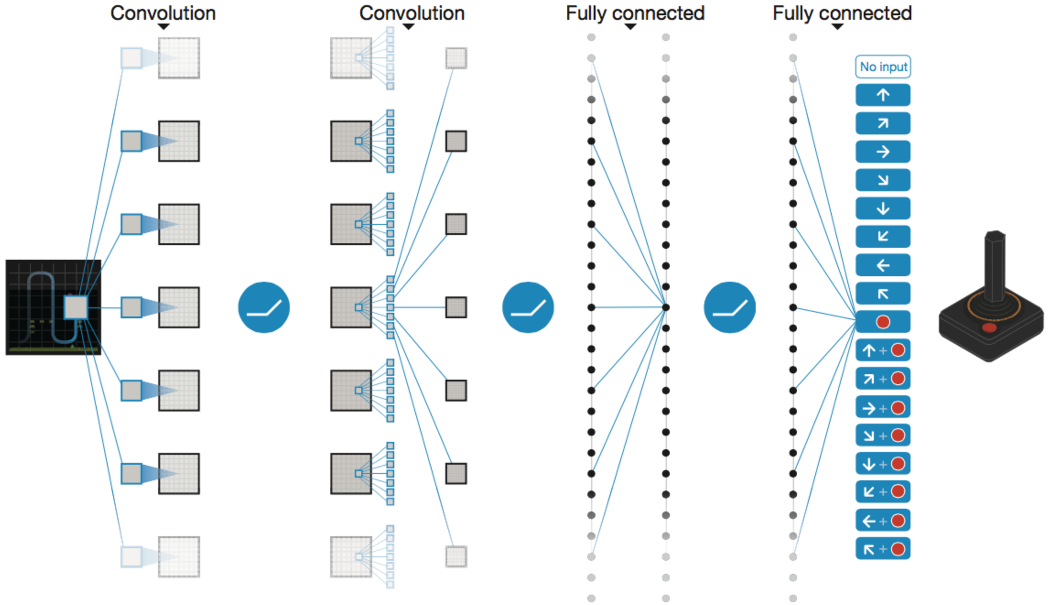



0
OXYGEN
ACTIVISION

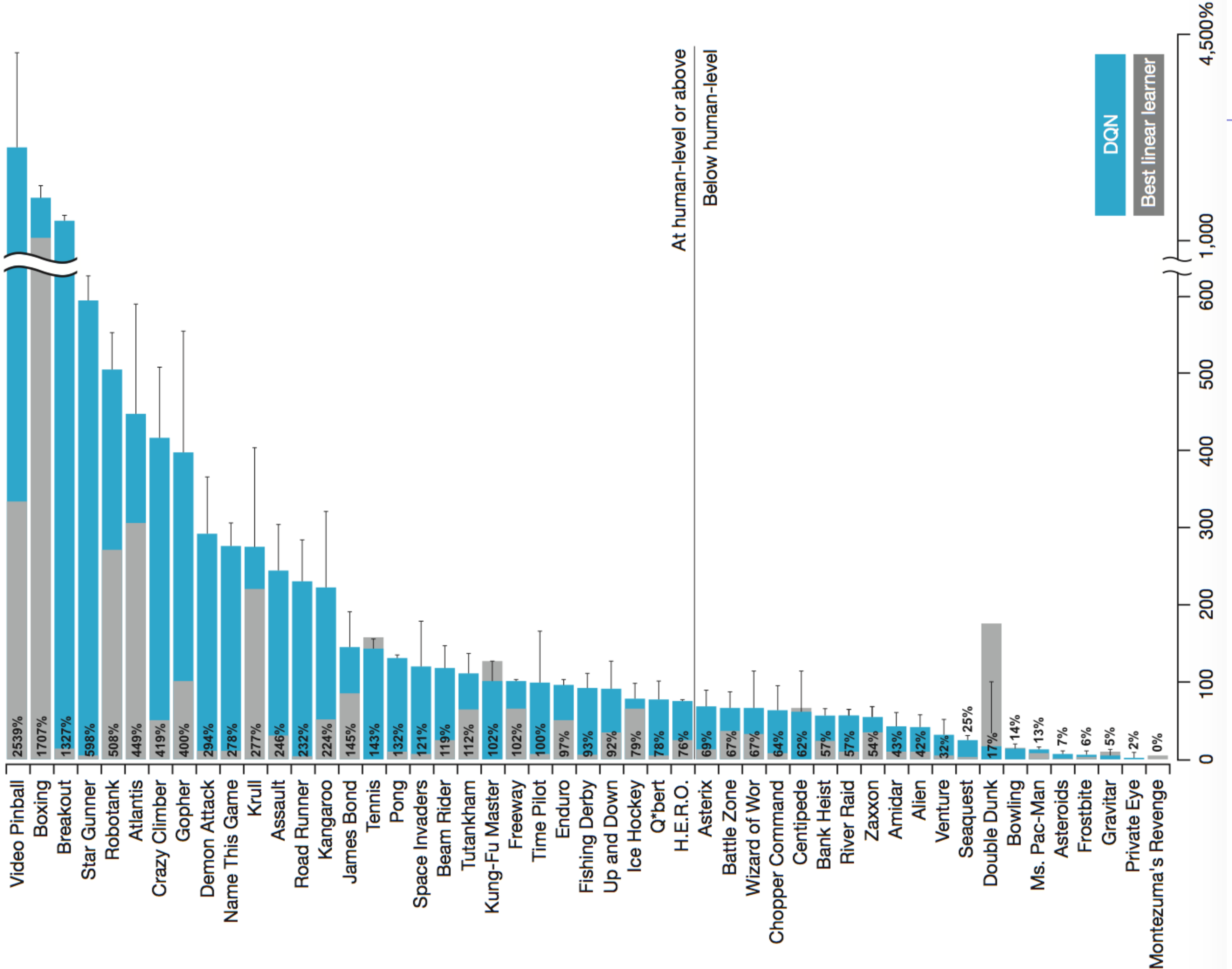

0000025
01
180
142 KM/H

Atari game playing

- MDP
  - State: image of game screen (256^{84*84} possible states)
    - Processed with hand-designed feature vectors or neural networks.
  - Action; combination of arrow keys + buttons (18)
  - Transition T: game code (no access)
  - Reward R: game code (no access)
- Essentially similar to pacman
- Use approximate Q-learning with NN and ε-greedy exploration



Mnih et al, 2015 Human-level control through deep RL


DQN
Best linear learner
At human-level or above
Below human-level
4,500%
1,000
600
500
400
300
200
100
0
Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%

Summary

- Exploration vs. exploitation
  - Exploration guided by unfamiliarity and potential
  - Appropriately designed bonuses tend to minimize regret
- Generalization allows RL to scale up to real problems
  - Represent V or Q with parameterized functions
  - Adjust parameters to reduce sample prediction error

Next Time

- Machine Learning!
  - Learning CPTs in Bayes Nets from data
  - From Perceptron to Neural Networks
  - Optimization