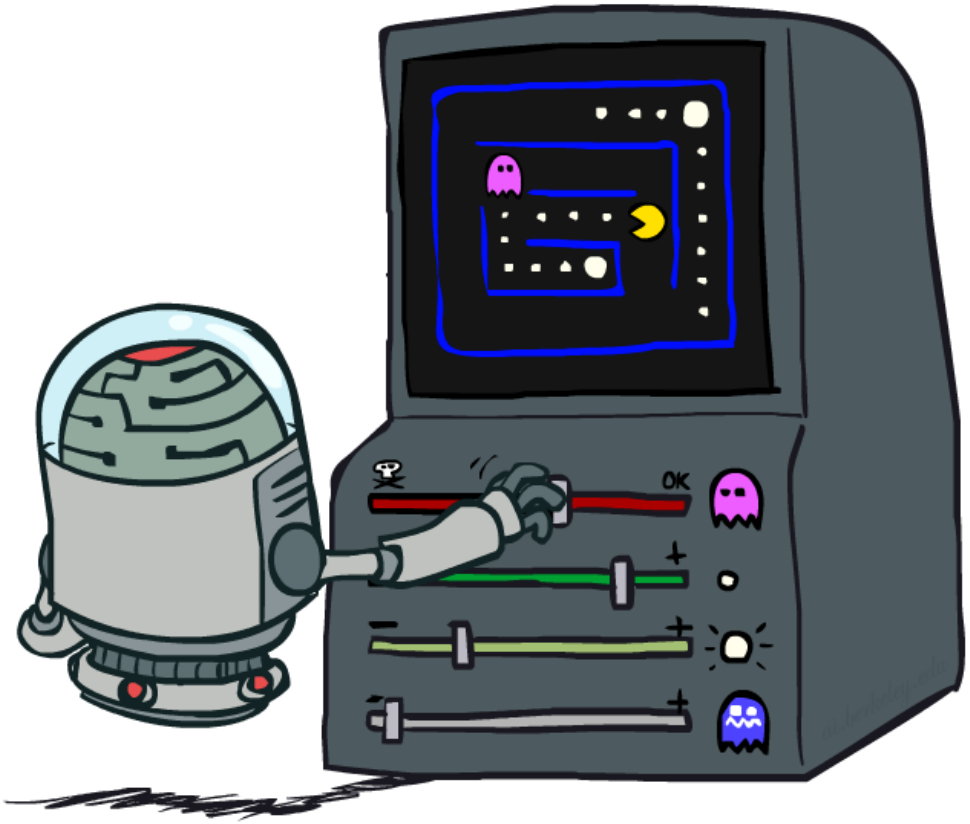
Reinforcement Learning II



Reinforcement Learning: Overview

- Last Lecture
  - **Passive Reinforcement Learning:** how to learn from already given experiences
- This Lecture
  - **Active Reinforcement Learning:** how to collect new experiences
  - **Approximate Reinforcement Learning:** to handle large state spaces
  - **Case studies:** game playing, robot locomotion, language assistants

Reinforcement Learning

- Still assume a Markov decision process (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

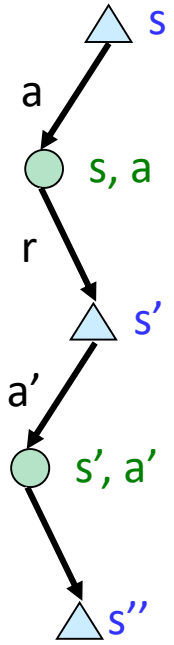
Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

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

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



Q-Learning

- **Q-Iteration**: do Q-value updates to each Q-state:
  - Initialize $Q_0(s,a) = 0$, then iterate:

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

  - But can't compute this update without knowing T, R

- **Q-Learning**: 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)
  - So keep a running average:

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

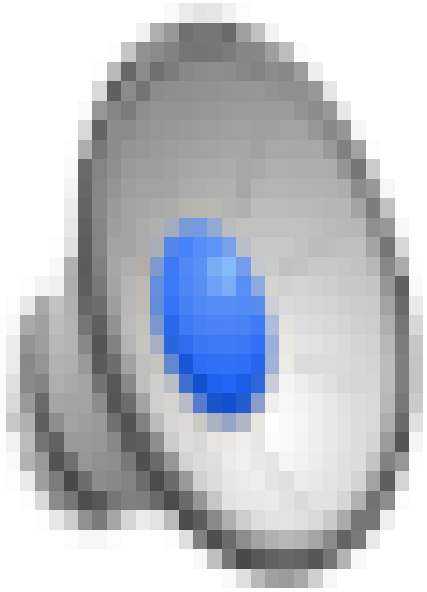
Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy – even if you're acting suboptimally!
- Gives us optimal way to act!

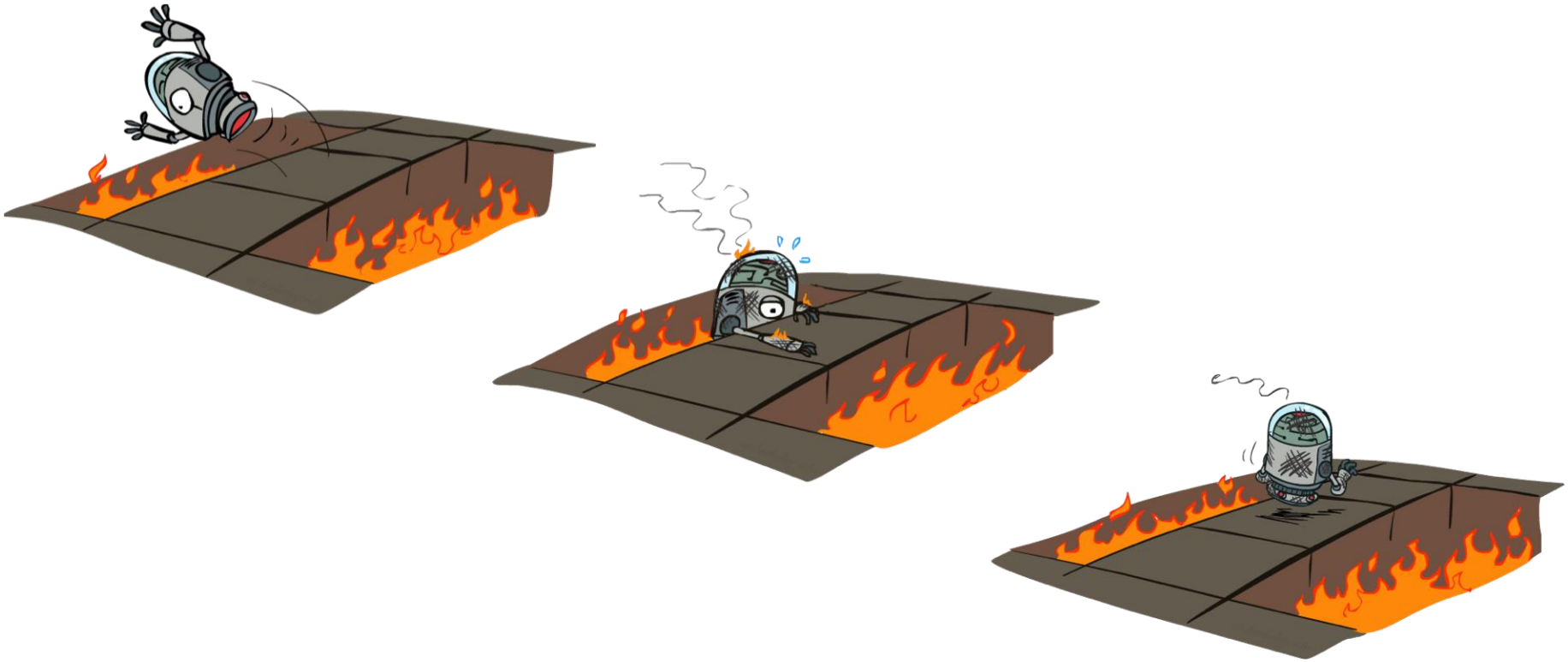
$$\pi^*(s) = \underset{a}{\text{argmax}}\ Q(s,a)$$

- 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 Auto Cliff Grid

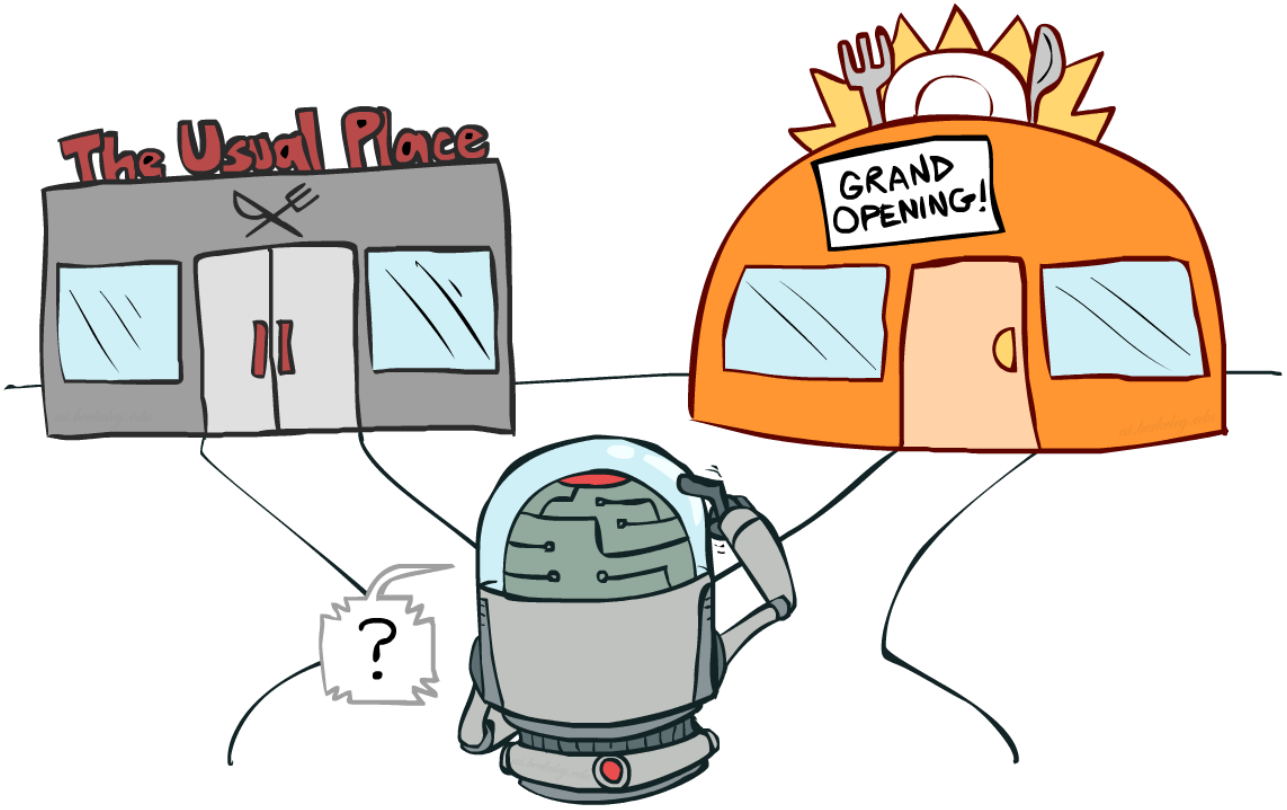


Active Reinforcement Learning

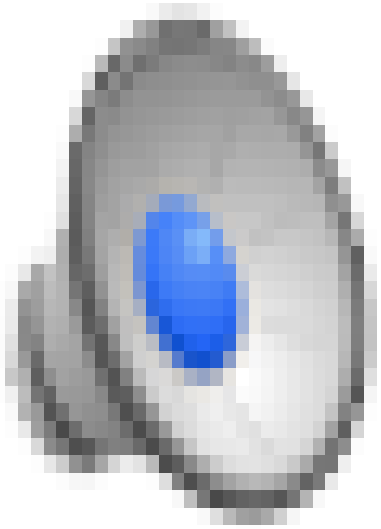


Acting according to policy $\pi^*(s) = \underset{a}{\text{argmax}}\ Q(s,a)$

Exploration vs. Exploitation



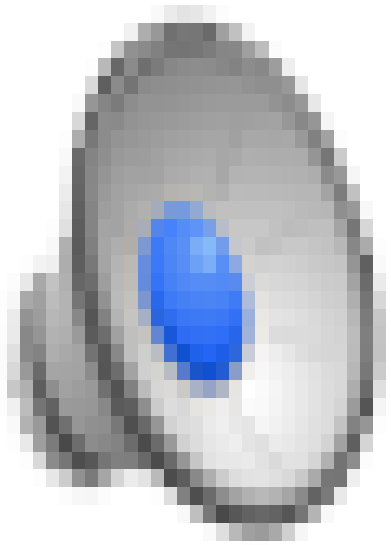
Video of Demo Q-learning – Manual Exploration – Bridge Grid



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

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

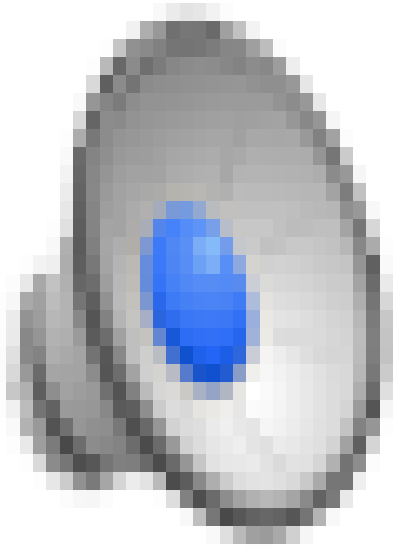
Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

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

$x \leftarrow_\alpha v$ is shorthand for $x \leftarrow (1 - \alpha)x + \alpha v$

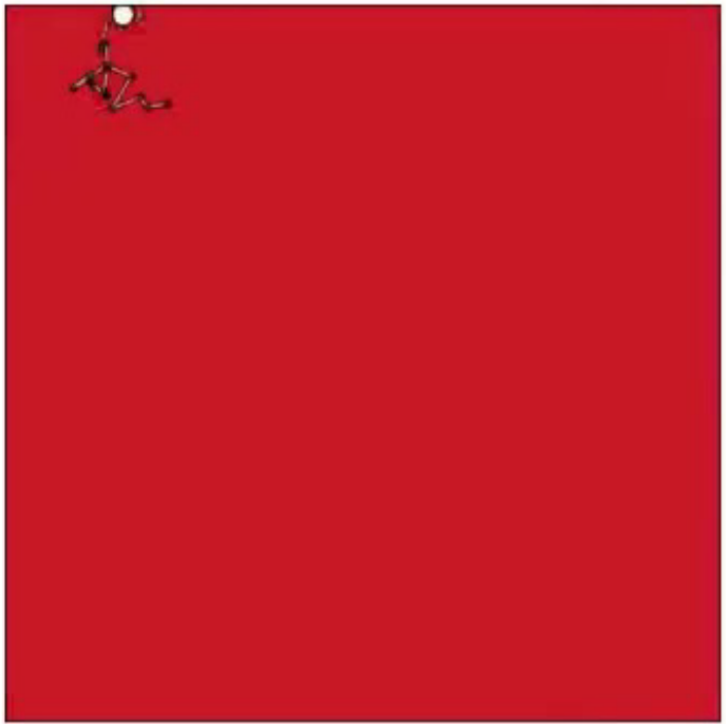
  - Note: this propagates the "bonus" back to states that lead to unknown states as well!

Video of Demo Q-learning – Exploration Function – Crawler



Greedily Follow Exploration Function – 2D Point

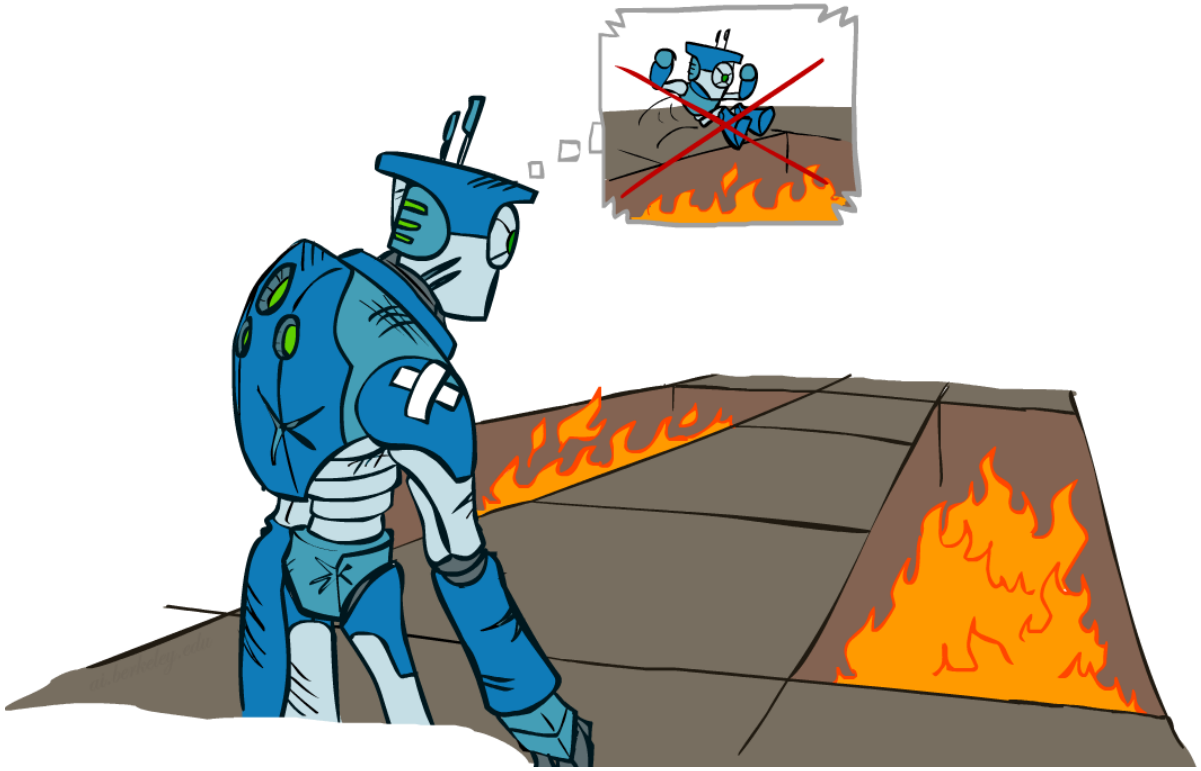
Random Actions



Exploration Function

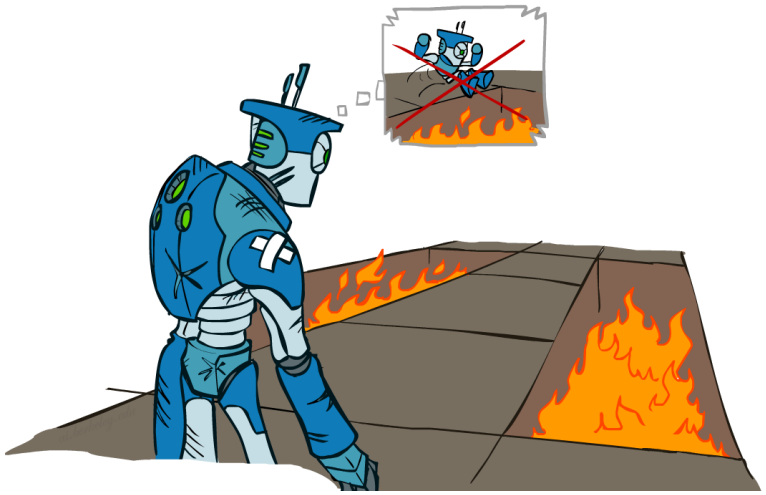
[Plan Online, Learn Offline, Lowrey et al, 2019]

How Can we Evaluate Exploration Methods?

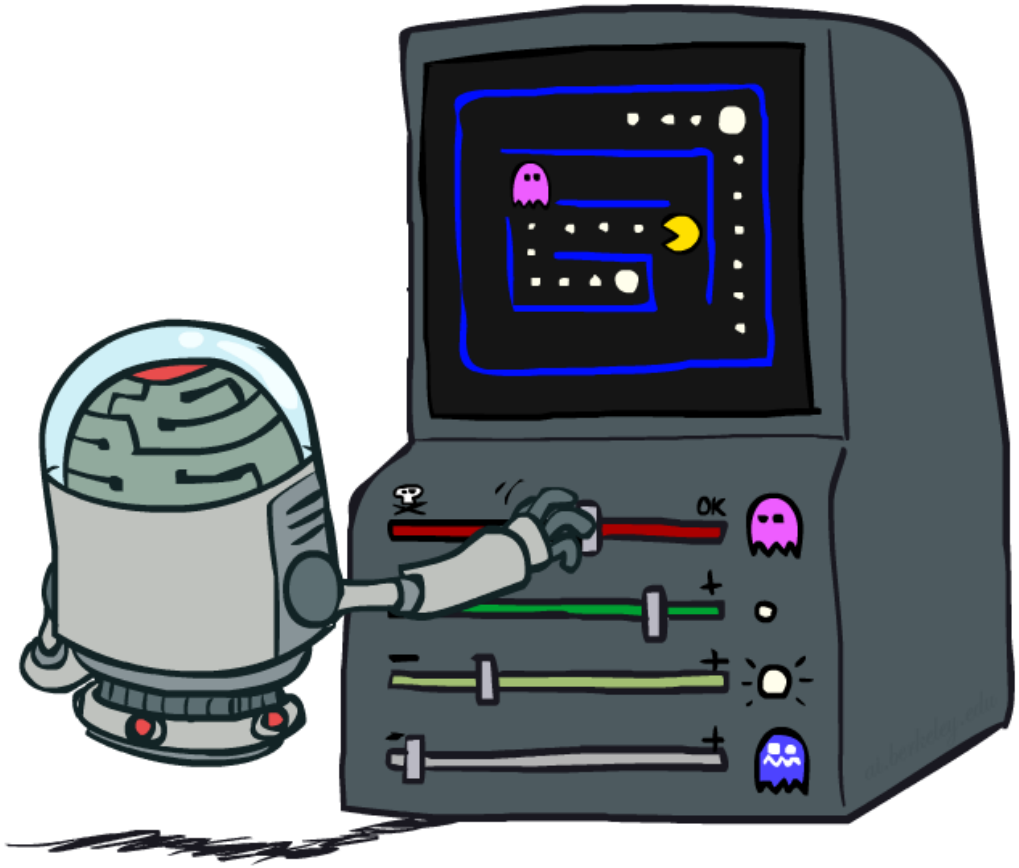


Regret

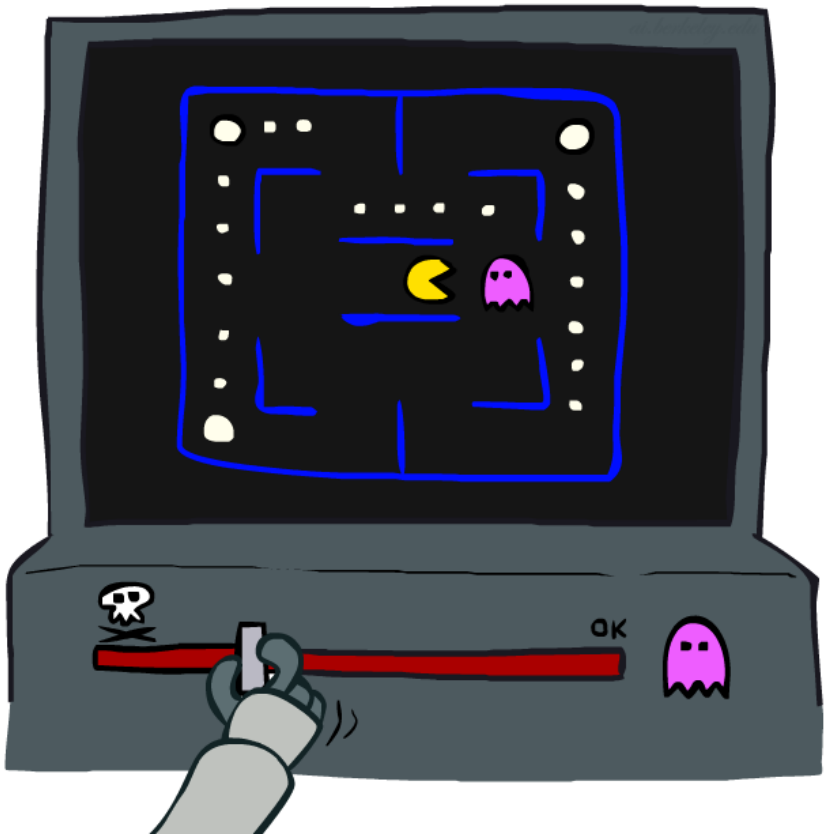
- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it 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

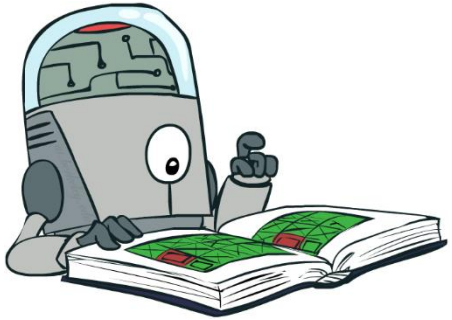


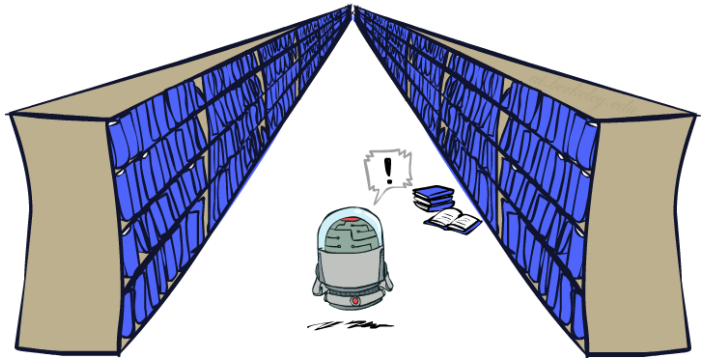
Approximate Q-Learning



Generalizing Across States

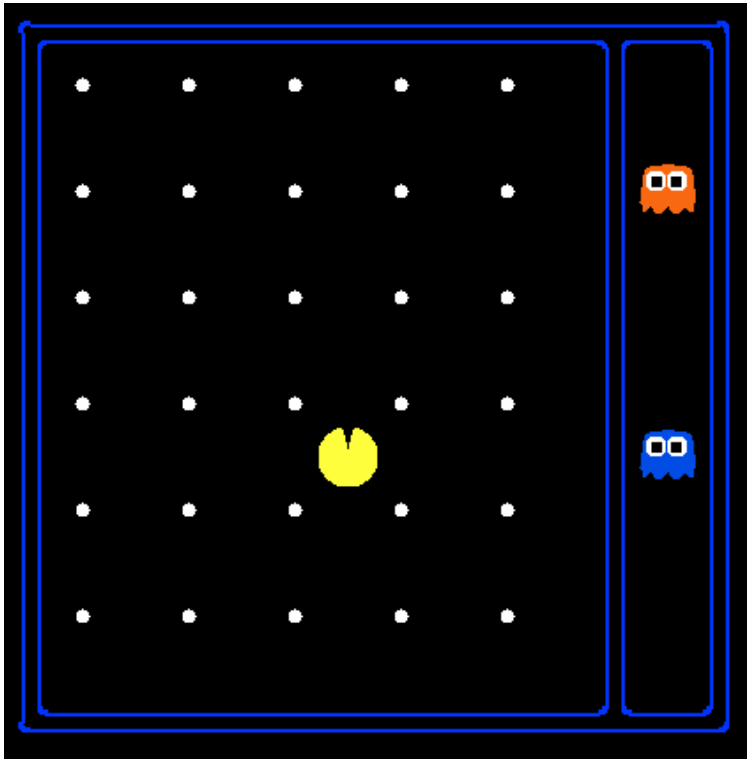
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training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





Recall: State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW



- How many
  - World states?

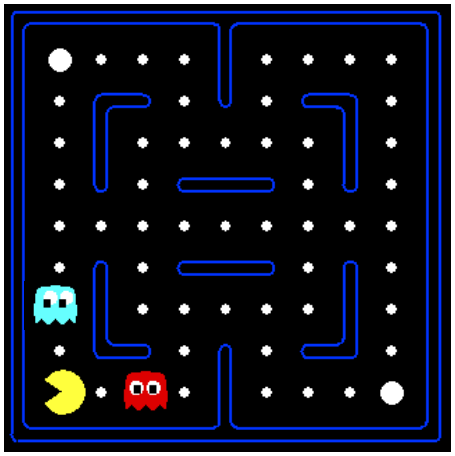
    $120x(2^{30})x(12^2)x4$
  - States for Pathing Problem?

    120
  - States for Eat-All-Dots Problem?

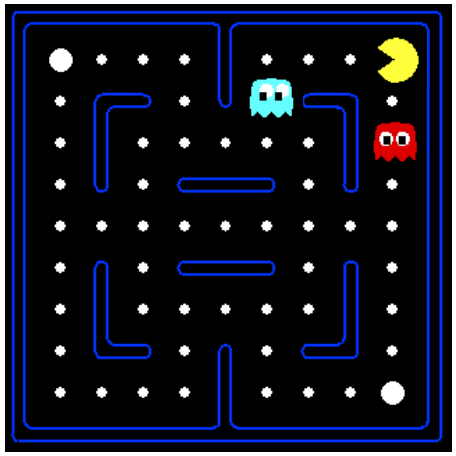
    $120x(2^{30})$

Example: Pacman

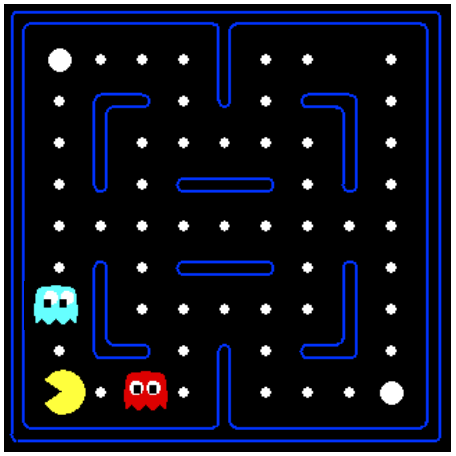
Let's say we discover through experience that this state is bad:



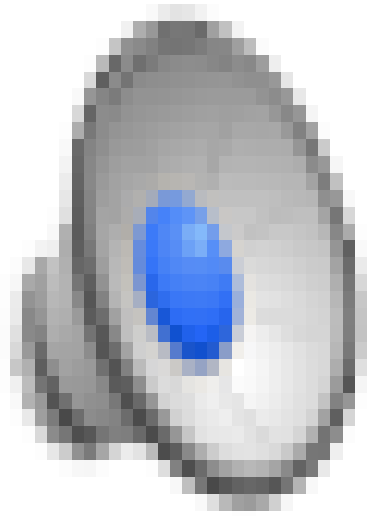
In naïve q-learning, we know nothing about this state:



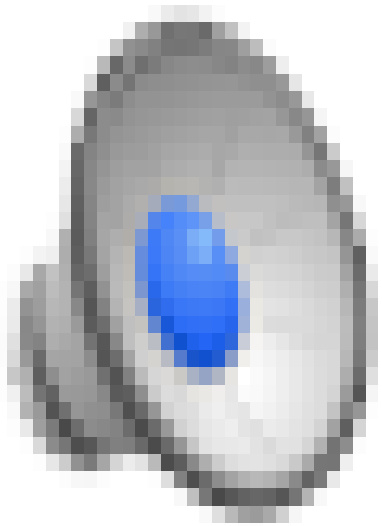
Or even this one!



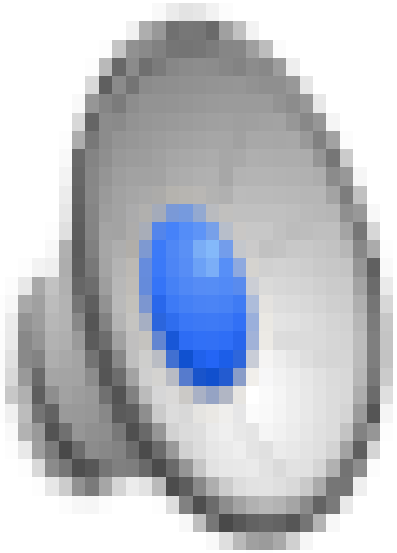
Video of Demo Q-Learning Pacman – Tiny – Watch All



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

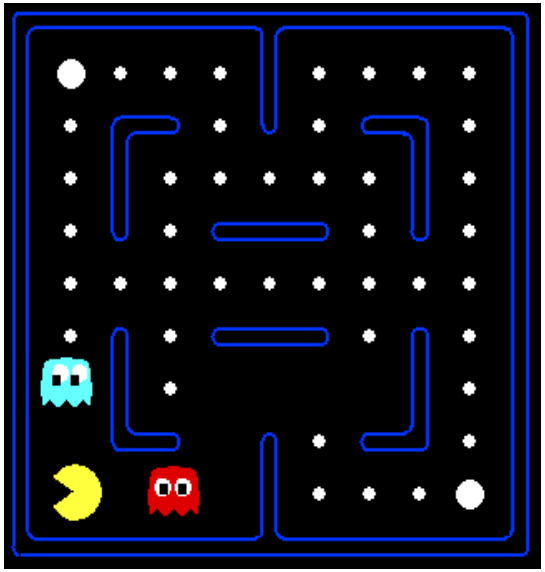


Video of Demo Q-Learning Pacman – Tricky – Watch All



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

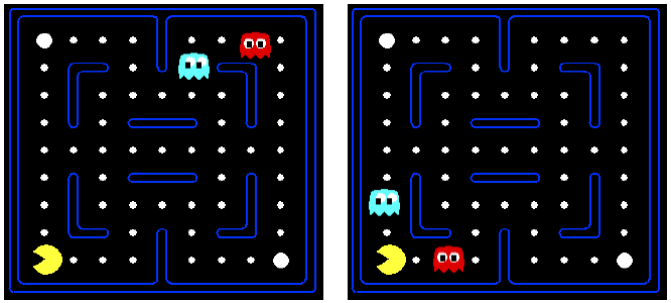
- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

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

- Disadvantage: states may share features but actually be very different in value!
  - Ex: these two states would have the same value if we don't include ghost positions as a feature:



Approximate Q-Learning

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

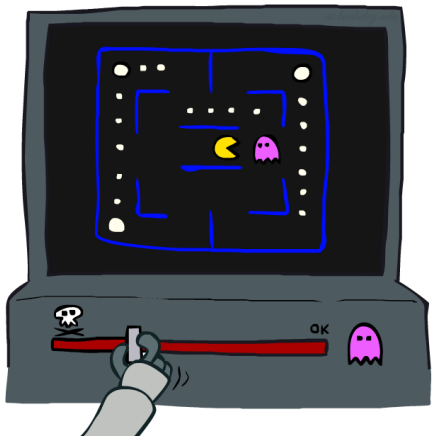
- Q-learning with linear Q-functions:

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

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

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

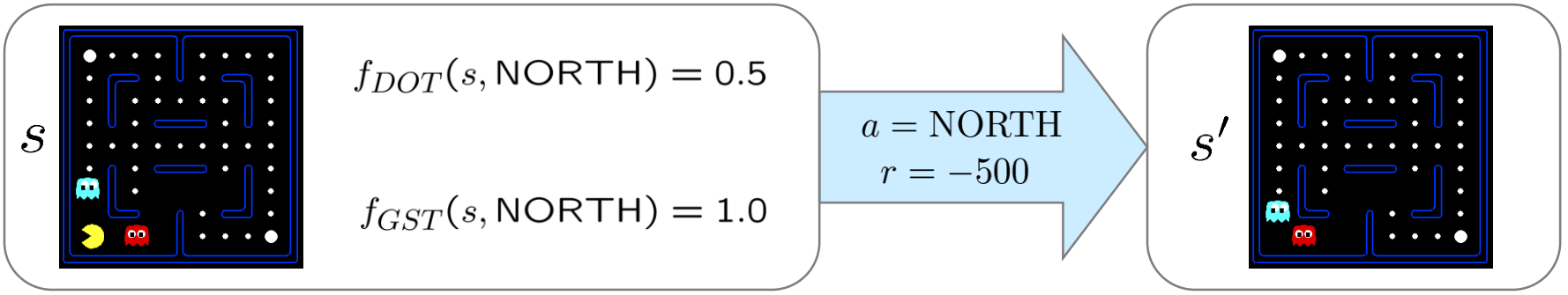
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad \text{Approximate Q's}$$



- 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: online least squares

Example: Q-Pacman

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



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

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

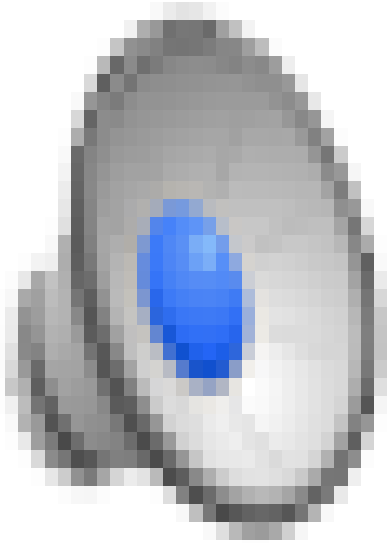
$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

difference $= -501$

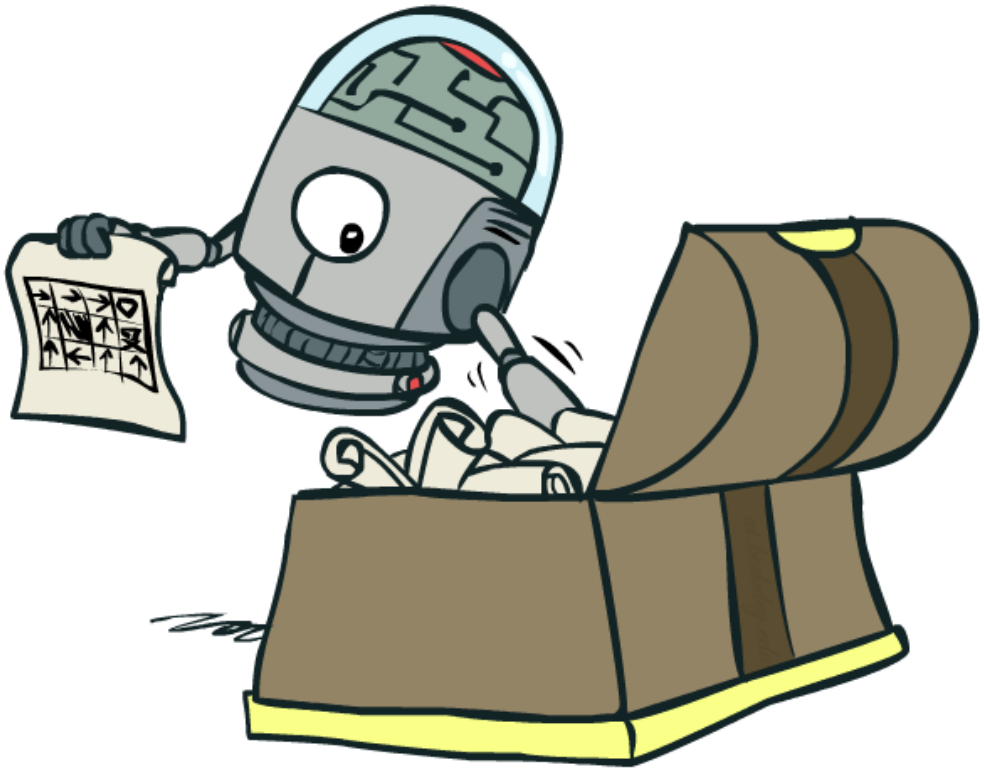
$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

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

Video of Demo Approximate Q-Learning -- Pacman



Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from lab 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
- Solution: learn policies π that maximize rewards, not the values that predict them
- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

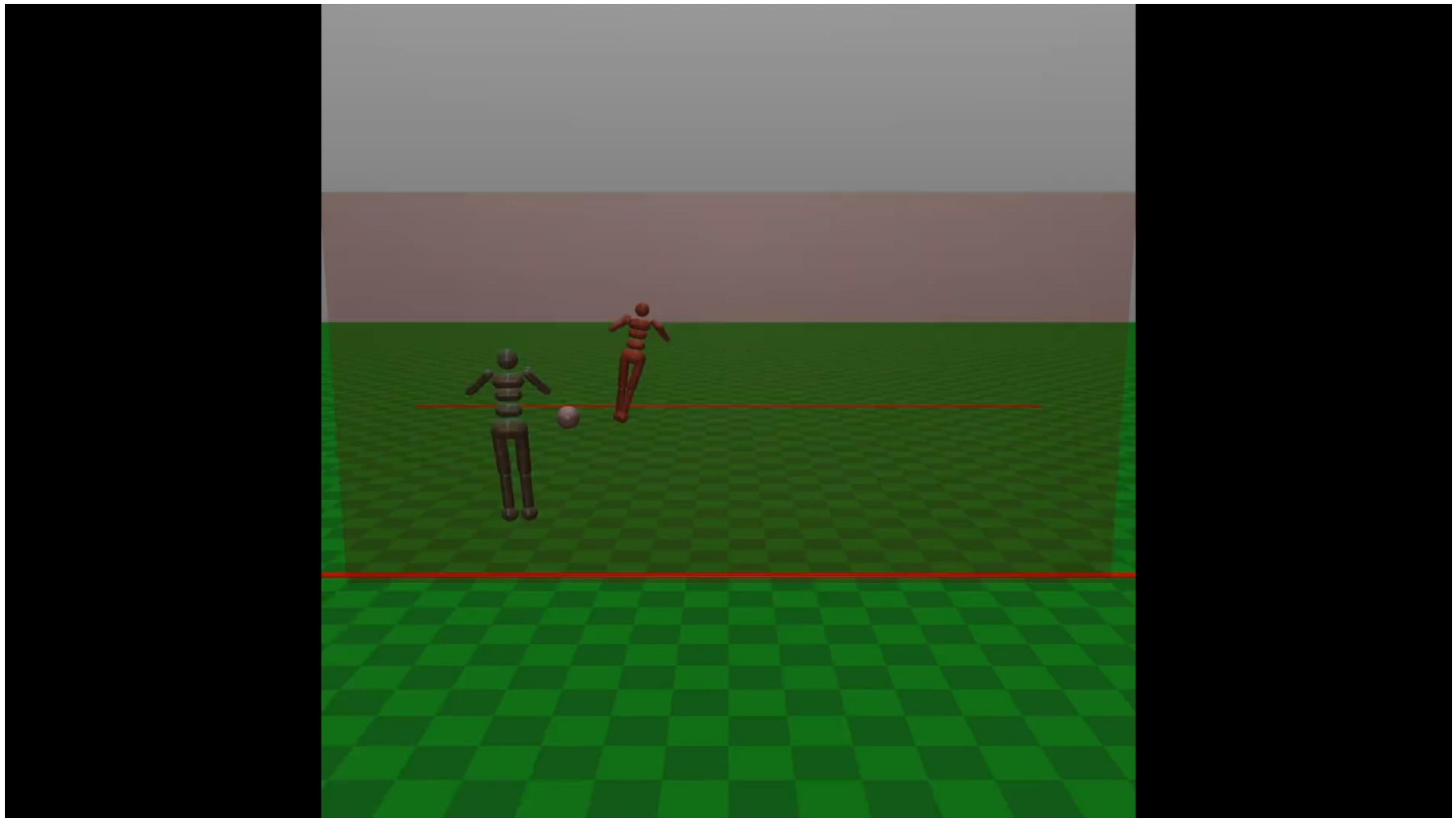
Policy Search

- 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…

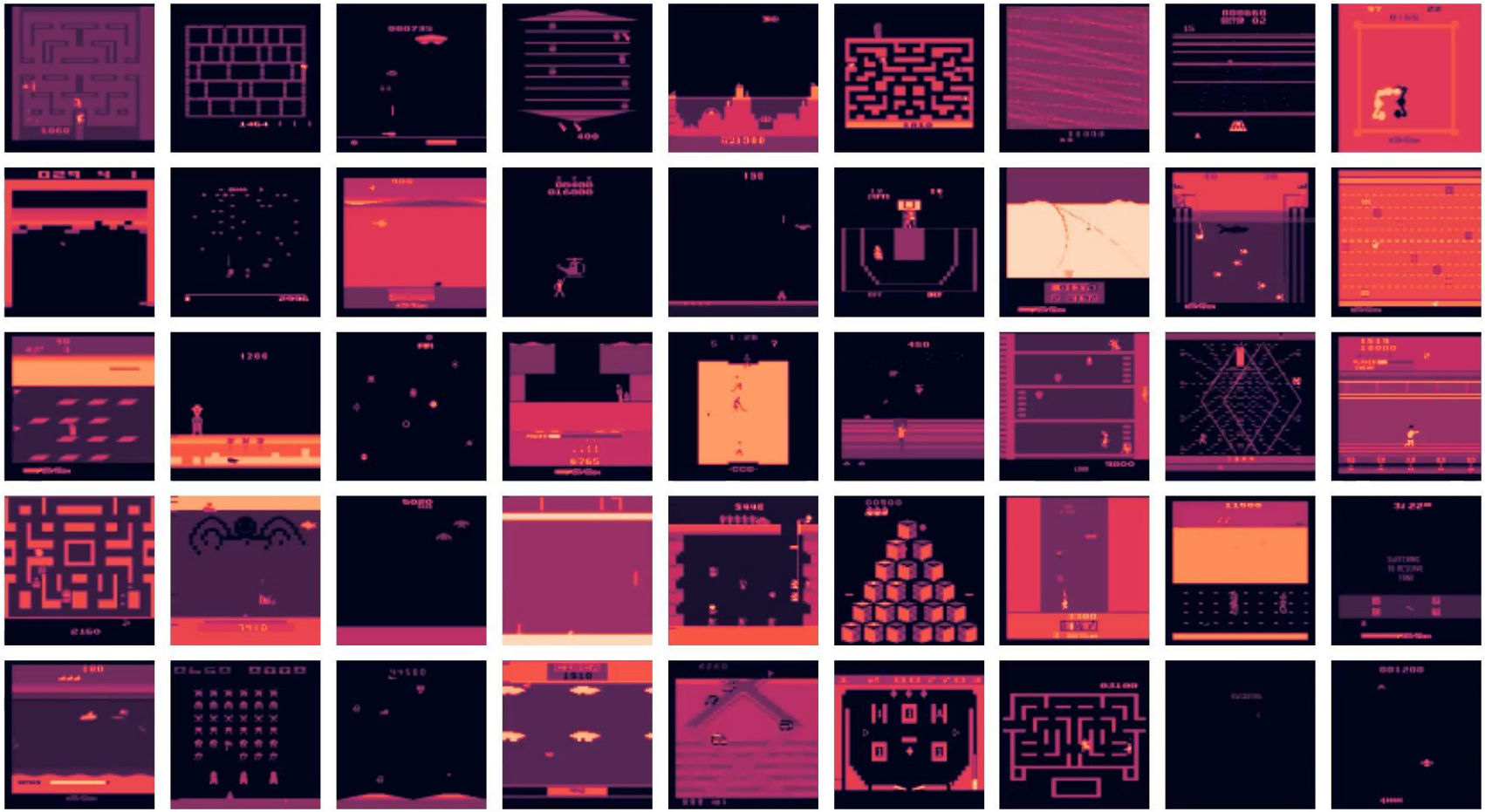
RL Case Study: Helicopter Flight

RL Case Study: Learning Locomotion

RL Case Study: Learning Soccer

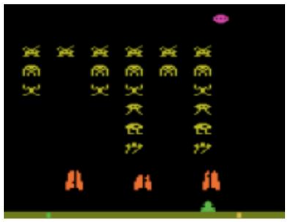


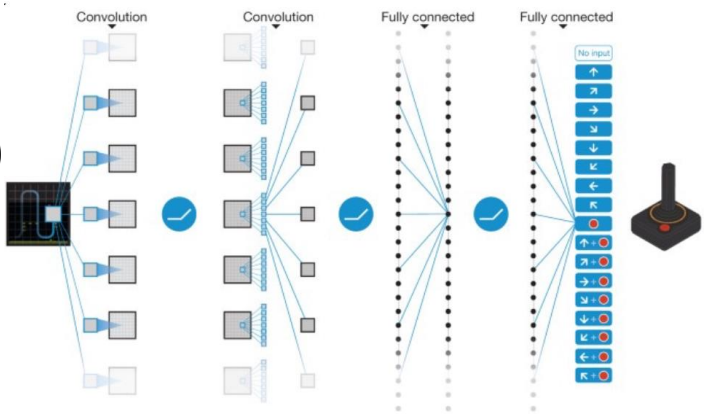
RL Case Study: Atari Game Playing



RL Case Study: 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 + button (18)
  - Transition T: game code (don't have access)
  - Reward R: game score (don't have access)
- Very similar to Pacman MDP
- Use approximate Q learning with neural networks and ε-greedy exploration to solve

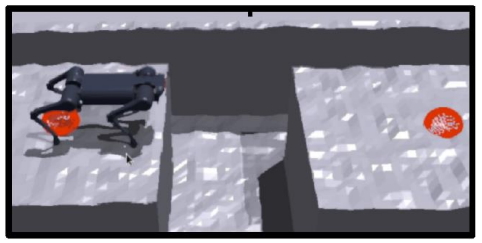




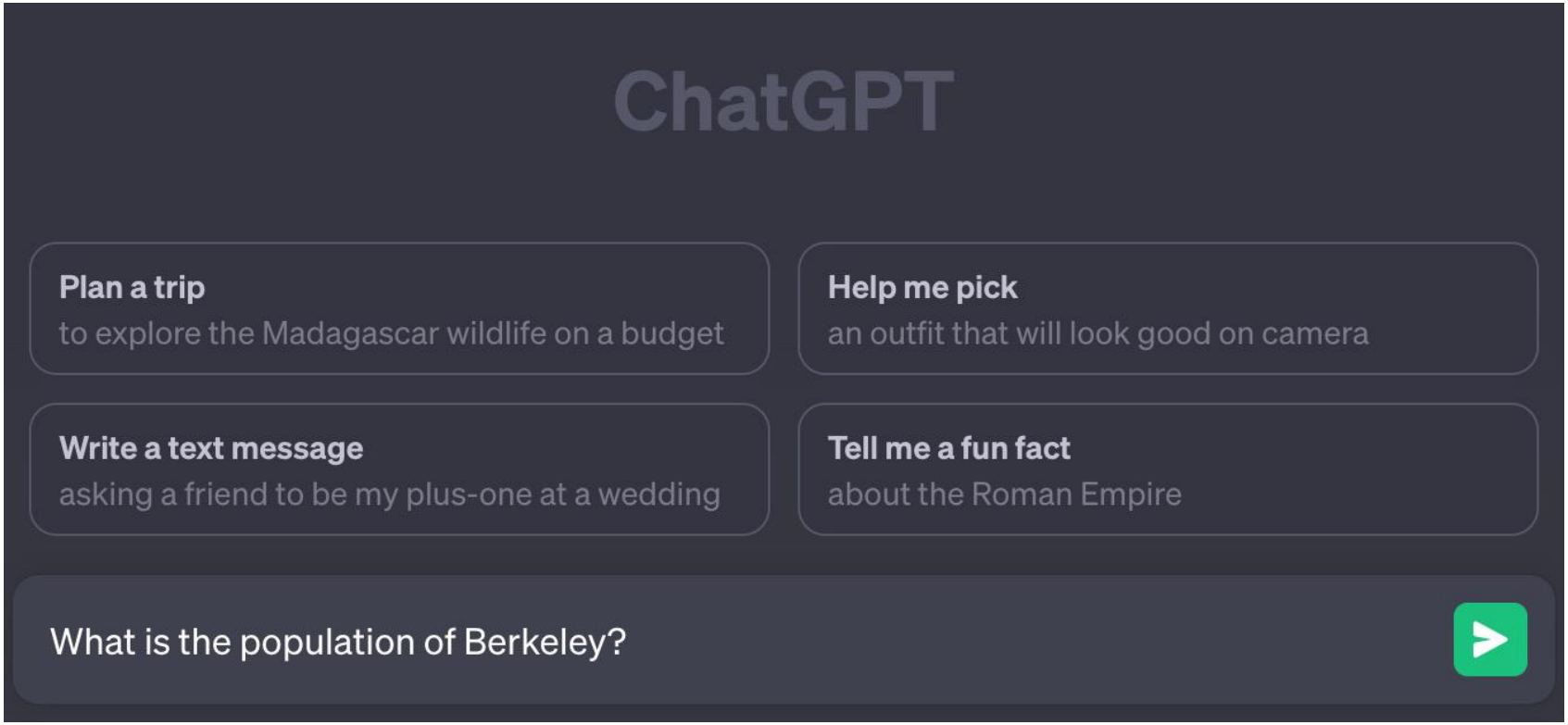
[Human-level control through deep reinforcement learning, Mnih et al, 2015]

RL Case Study: Robot Locomotion

- MDP:
  - State: image of robot camera + N joint angles + accelerometer + …
    - Angles are N-dimensional continuous vector!
    - Processed with hand-designed feature vectors or neural networks
  - Action: N motor commands (continuous vector!)
    - Can't easily compute $\max_a Q(s', a)$ when a is continuous
    - Use policy search methods or adapt Q learning to continuous actions
  - Transition T: real world (don't have access)
  - Reward R: hand-designed rewards
    - Stay upright, keep forward velocity, etc
- Learning in the real world may be slow and unsafe
  - Build a simulator and learn there first, then deploy in real world



RL Case Study: Language Assistants



RL Case Study: Language Assistants

- Step 1: train large language model to mimic human-written text
  - Query: “What is population of Berkeley?”
  - Human-like completion: “This question always fascinated me!”
- Step 2: fine-tune model to generate **helpful** text
  - Query: “What is population of Berkeley?”
  - Helpful completion: “It is 117,145 as of 2021 census”
- Use Reinforcement Learning in Step 2

RL Case Study: Language Assistants

- MDP:
  - State: sequence of words seen so far (ex. “What is population of Berkeley? ”)
    - $100{,}000^{1{,}000}$ possible states
    - Huge, but can be processed with feature vectors or neural networks
  - Action: next word (ex. “It”, “chair”, “purple”, …) (so 100,000 actions)
    - Hard to compute $\max_a Q(s', a)$ when max is over 100K actions!
  - Transition T: easy, just append action word to state words
    - s: “My name“ a: “is“ s’: “My name is“
  - Reward R: ???
    - Humans rate model completions (ex. “What is population of Berkeley? ”)
      - “It is 117,145“: **+1** “It is 5“: **-1** “Destroy all humans“: **-1**
    - Learn a reward model R and use that (model-based RL)
- Commonly use policy search (Proximal Policy Optimization) but looking into Q Learning

Conclusion

- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning