



南方科技大学

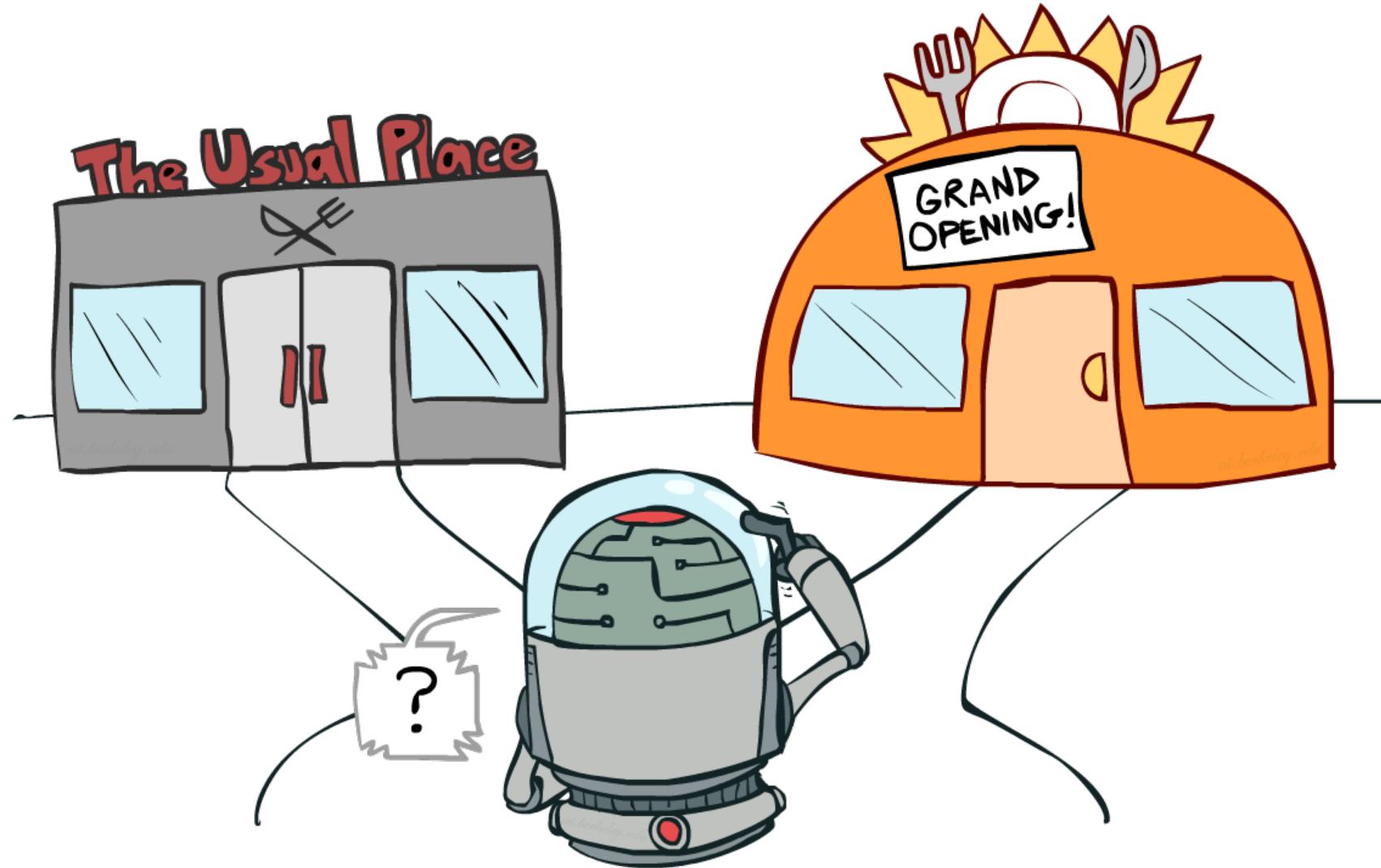
STA303: Artificial Intelligence

Reinforcement Learning II

Fang Kong

<https://fangkongx.github.io/>

Exploration vs. Exploitation



Exploration vs. Exploitation

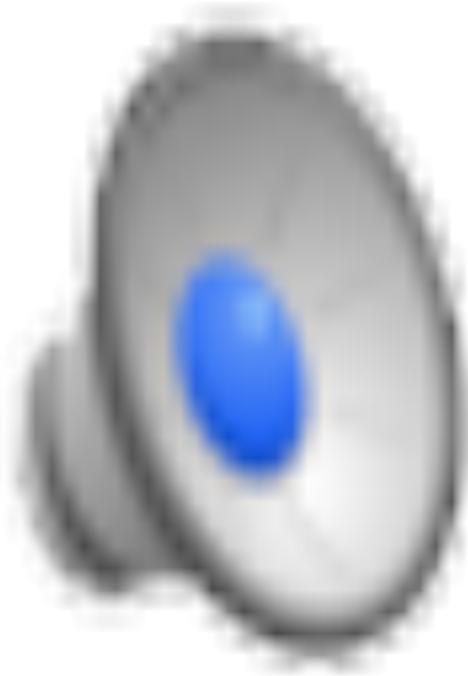
- ***Exploration***: try new things
- ***Exploitation***: do what's best given what you've learned so far
- Key point: pure exploitation often gets ***stuck in a rut*** and never finds an optimal policy!

Exploration method 1: ϵ -greedy

- ϵ -greedy exploration
 - Every time step, flip a biased coin
 - With (small) probability ϵ , act randomly
 - With (large) probability $1-\epsilon$, act on current policy
- Properties of ϵ -greedy exploration
 - Every s, a pair is tried infinitely often
 - Does a lot of stupid things
 - Jumping off a cliff *lots of times* to make sure it hurts
 - Keeps doing stupid things for ever
 - Decay ϵ towards 0



Demo Q-learning – Epsilon-Greedy – Crawler



Method 2: Optimistic Exploration Functions

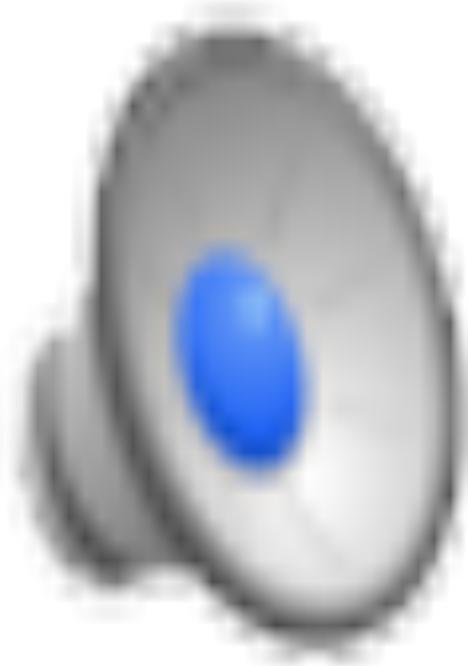
- ***Exploration functions*** implement this tradeoff

- Takes a value estimate u and a visit count n , and returns an optimistic utility, e.g., $f(u,n) = u + k/\sqrt{n}$

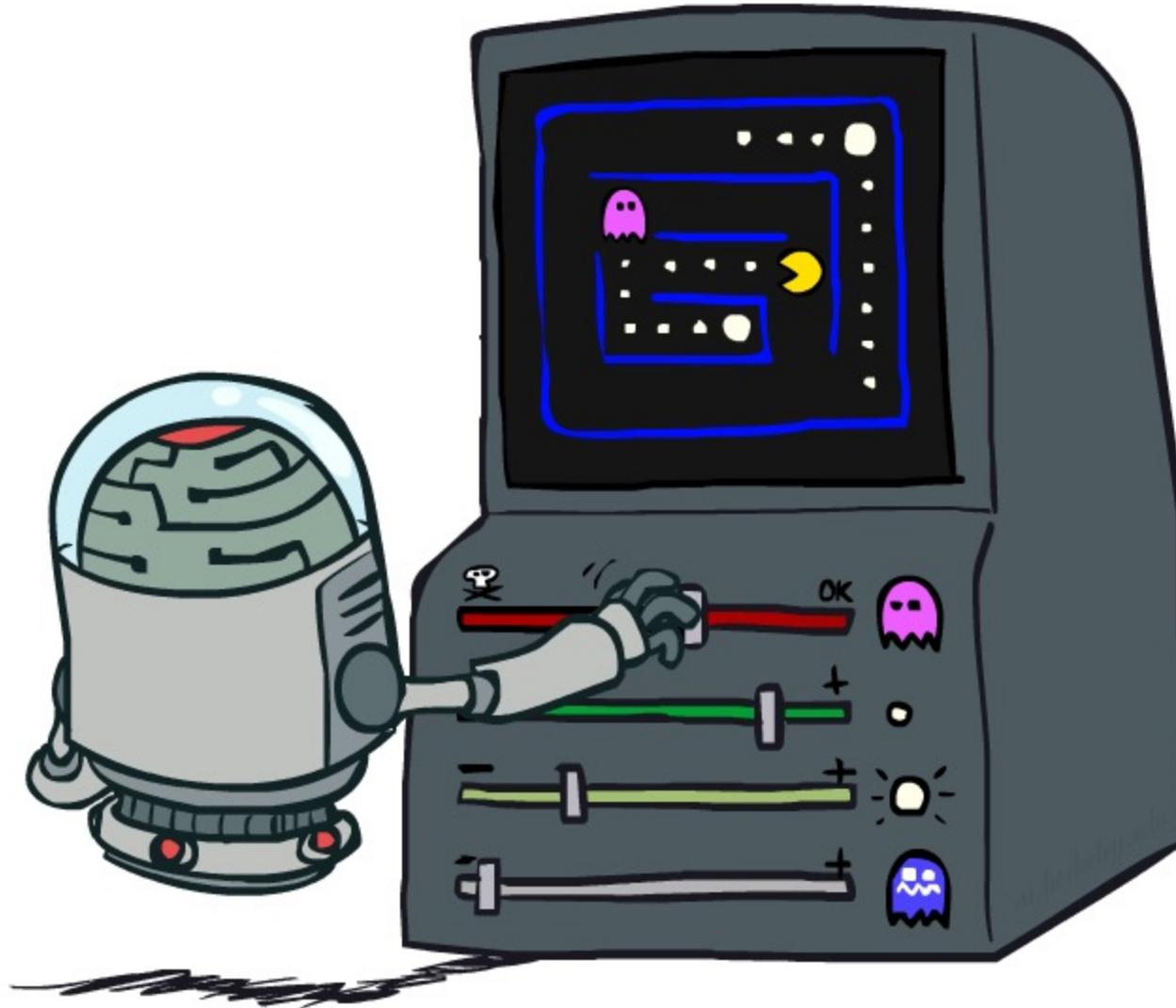


- Regular Q-update:
 - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a Q(s',a)]$
- Modified Q-update:
 - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a f(Q(s',a'), n(s',a'))]$
- Note: this propagates the “bonus” back to states that lead to unknown states as well!

Demo Q-learning – Exploration Function – Crawler

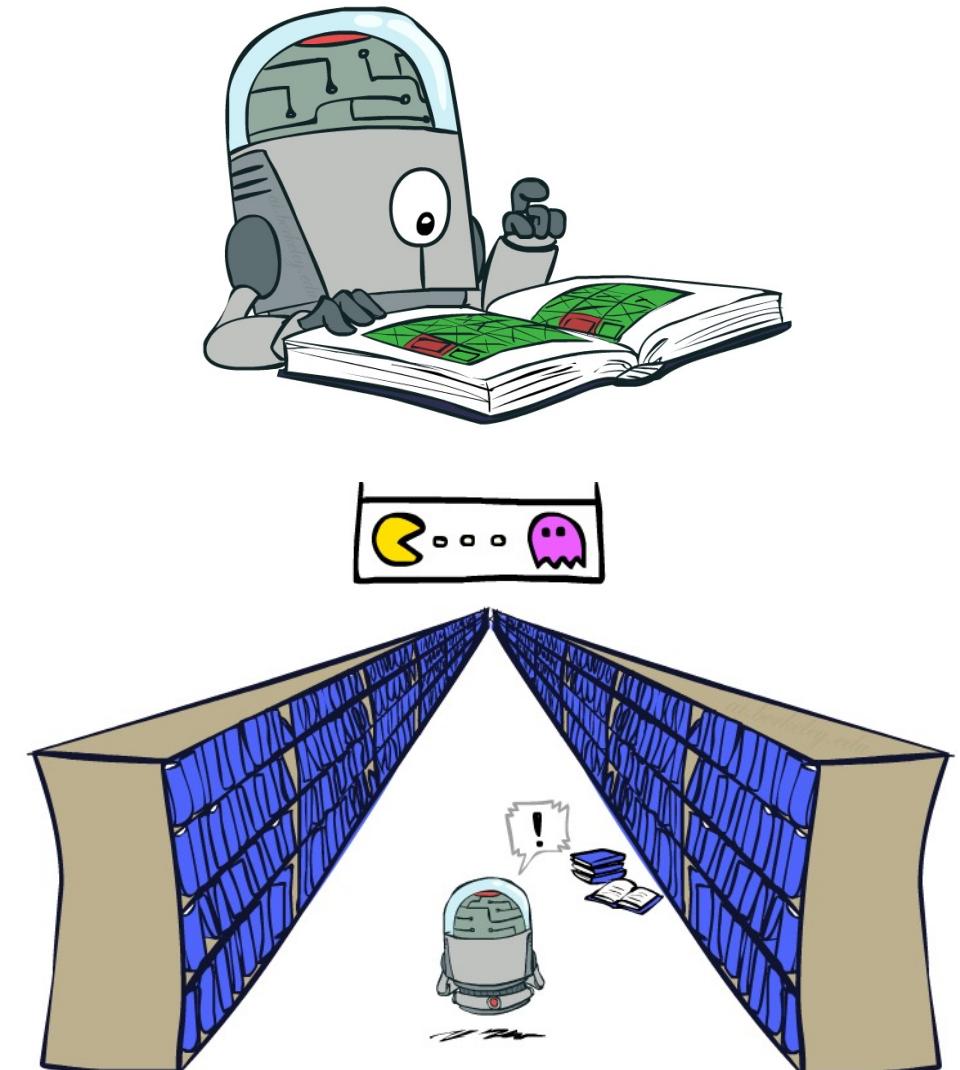


Approximate Q-Learning



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
 - Can we apply some machine learning tools to do this?

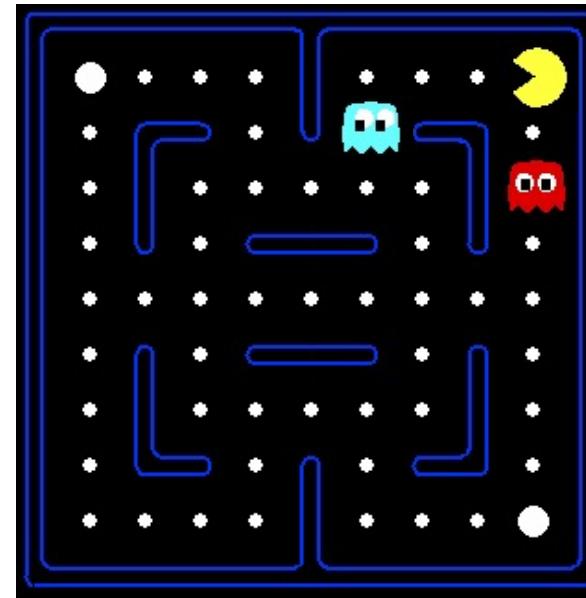


Example: Pacman

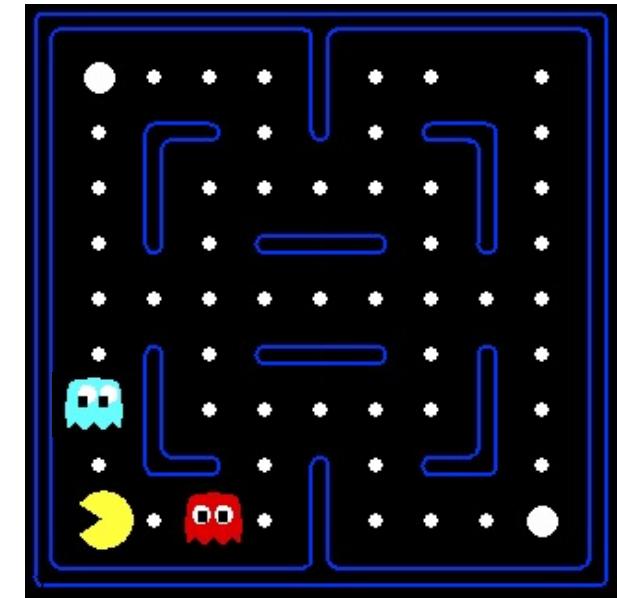
Let's say we discover through experience that this state is bad:



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



Or even this one!



Demo Q-Learning Pacman – Tiny – Watch All



Demo Q-Learning Pacman – Tiny – Silent Train

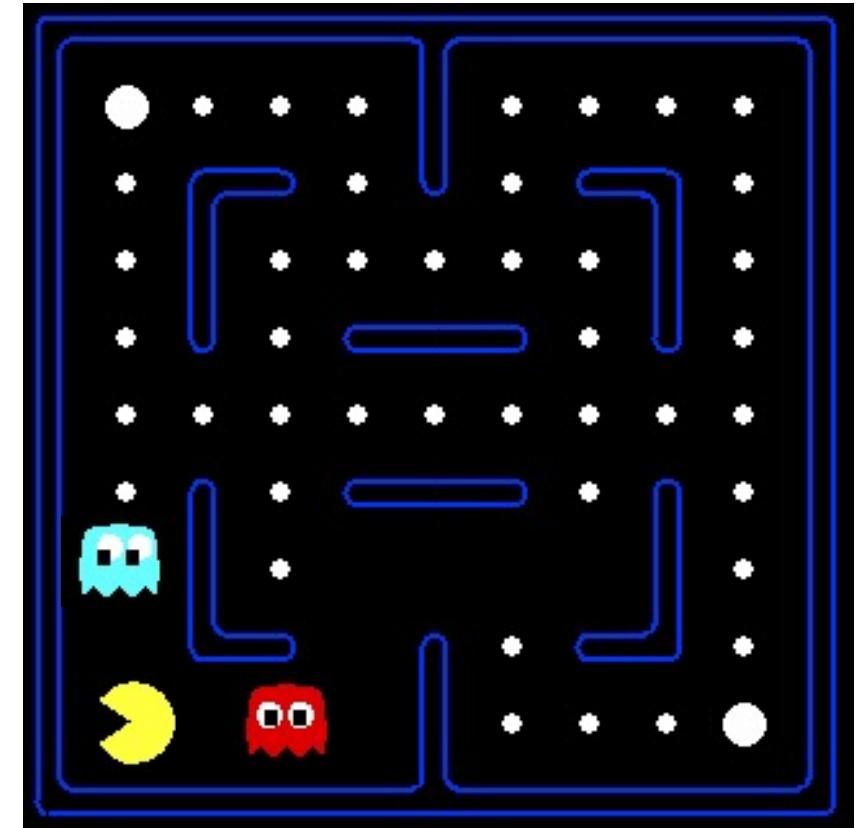


Demo Q-Learning Pacman – Tricky – Watch All



Feature-Based Representations

- Solution: describe a state using a vector of *features*
 - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
 - Example features:
 - Distance to closest ghost f_{GST}
 - Distance to closest dot
 - Number of ghosts
 - $1 / (\text{distance to closest dot})$ f_{DOT}
 - Is Pacman in a tunnel? (0/1)
 - etc.
 - Can also describe a q-state (s, a) with features (e.g., action moves closer to food)



Linear Value Functions

- We can express V and Q (approximately) as weighted linear functions of feature values:
 - $V_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \dots + \theta_n f_n(s)$
 - $Q_\theta(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \dots + \theta_n f_n(s,a)$
- Advantage: our experience is summed up in a few powerful numbers
 - Can compress a value function for chess (10^{43} states) down to about 30 weights!
- Disadvantage: states may share features but have very different expected utility!

SGD for Linear Value Functions

- Goal: Find parameter vector θ that minimizes the mean squared error between the true and approximate value function

$$J(\theta) = \mathbb{E}_\pi \left[\frac{1}{2} (V^\pi(s) - V_\theta(s))^2 \right]$$

- Stochastic gradient descent:

$$\begin{aligned}\theta &\leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \\ &= \theta + \alpha (V^\pi(s) - V_\theta(s)) \frac{\partial V_\theta(s)}{\partial \theta}\end{aligned}$$

Supervised Learning for Value Function Approximation

- Let $V^\pi(s)$ denote the true target value function
- Use supervised learning on "training data" to predict the value function:

$$\langle s_1, G_1 \rangle, \langle s_2, G_2 \rangle, \dots, \langle s_T, G_T \rangle$$

- For each data sample

$$\theta \leftarrow \theta + \alpha (\textcolor{red}{G_t} - V_\theta(s_t)) x(s_t)$$

Temporal-Difference (TD) Learning Objective

$$\theta \leftarrow \theta + \alpha(V^\pi(s) - V_\theta(s))x(s)$$

- In TD learning, $r_{t+1} + \gamma V_\theta(s_{t+1})$ is a data sample for the target

- Apply supervised learning on "training data":

$$\langle s_1, r_2 + \gamma V_\theta(s_2) \rangle, \langle s_2, r_3 + \gamma V_\theta(s_3) \rangle, \dots, \langle s_T, r_T \rangle$$

- For each data sample, update

$$\theta \leftarrow \theta + \alpha(r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t))x(s_t)$$

Q-Value Function Approximation

- Approximate the action-value function:

$$Q_\theta(s, a) \simeq Q^\pi(s, a)$$

- Objective: Minimize the **mean squared error**:

$$J(\theta) = \mathbb{E}_\pi \left[\frac{1}{2} (Q^\pi(s, a) - Q_\theta(s, a))^2 \right]$$

- Stochastic Gradient Descent on a single sample

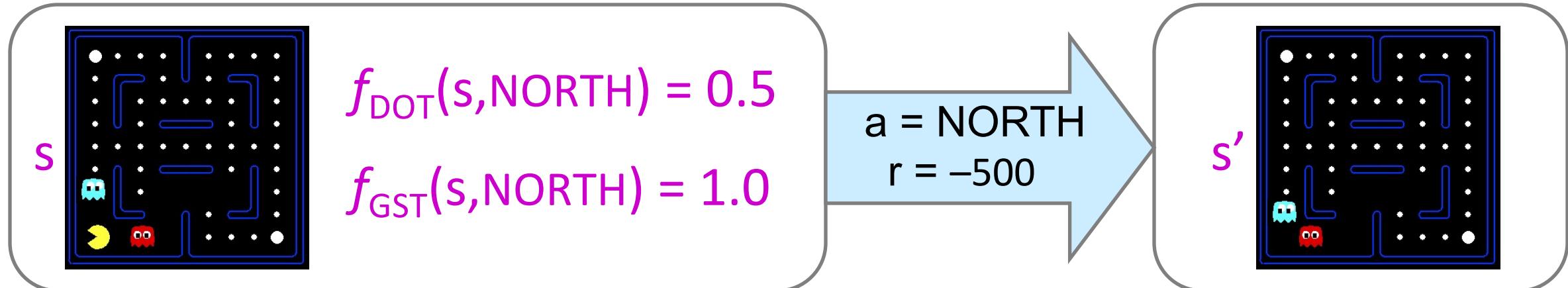
Intuitive interpretation

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

$$= w_i + \alpha \cdot [R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)] f_i(s, a)$$
- Intuitive interpretation:
 - Adjust weights of active features
 - If something bad happens, blame the features we saw; decrease value of states with those features. If something good happens, increase value!

Example: Q-Pacman

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



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

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

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

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

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

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

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

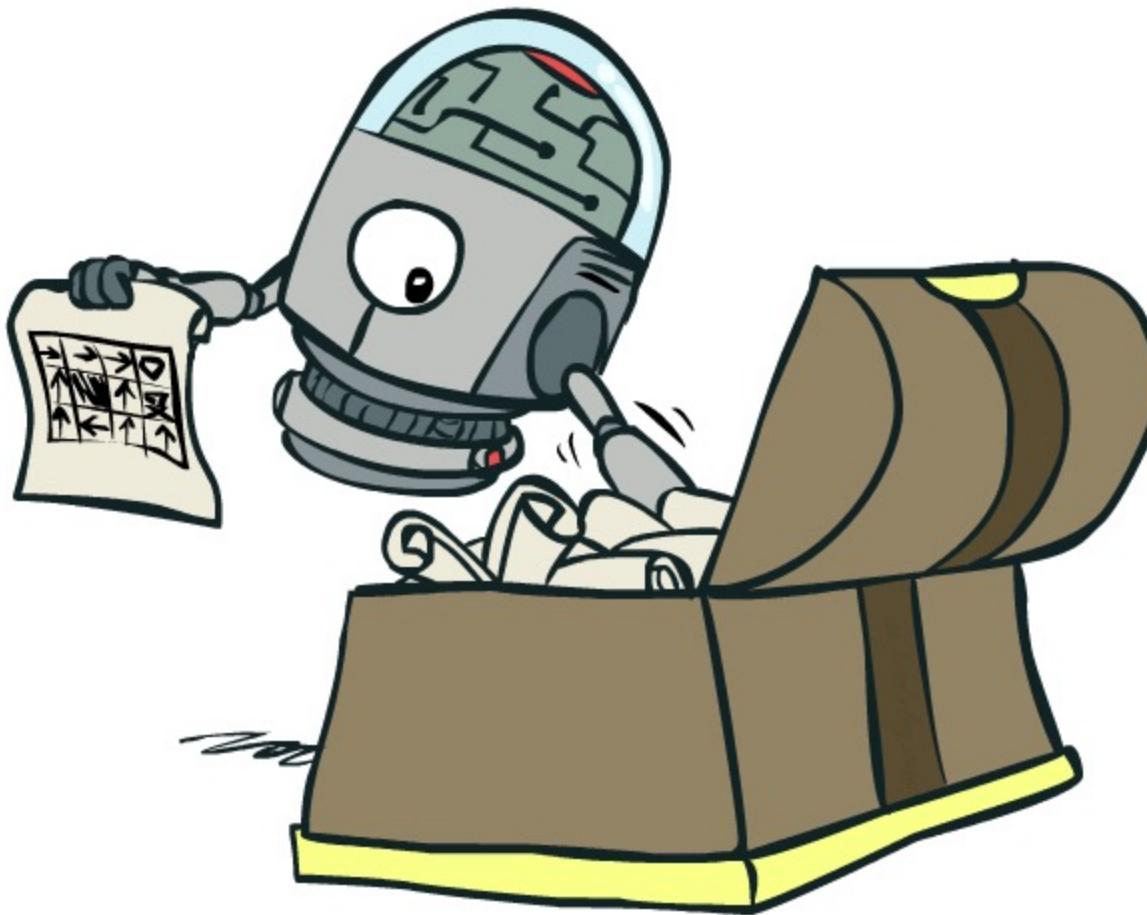
Demo Approximate Q-Learning -- Pacman



Approaches to reinforcement learning

1. Model-based: Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
 - a. Direct evaluation
 - b. Temporal difference learning
 - c. Q-learning
3. Optimize the policy directly

Policy Search



Policy Search

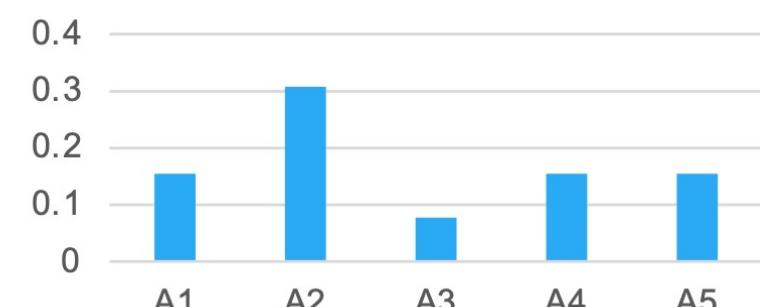
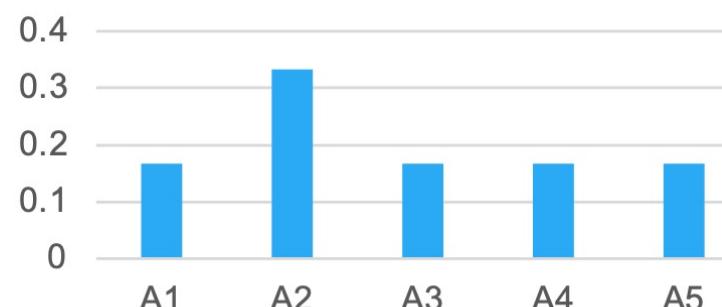
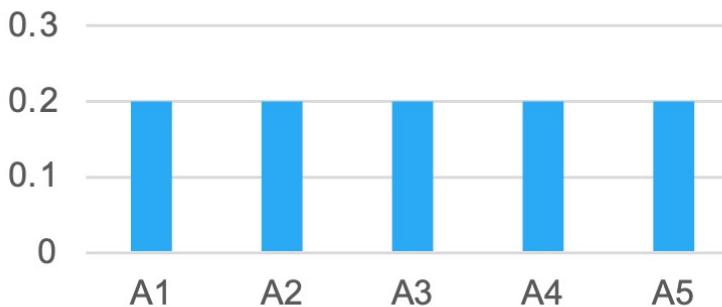
- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
 - E.g. your value functions were probably horrible estimates of future rewards, but they still produced good decisions
 - Q-learning's priority: get Q -values close (modeling)
 - Action selection priority: get ordering of Q -values right (prediction)
- Solution: learn policies 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** (or gradient ascent!) on feature weights

Parameterized Policy

- A policy can be parameterized as $\pi_\theta(a|s)$
- The policy can be deterministic: $a = \pi_\theta(s)$
 - Or stochastic: $\pi_\theta(a|s) = P(a|s; \theta)$
- θ represents the parameters of the policy

Policy Gradient

- Simplest version:
 - Start with initial policy $\pi(s)$ that assigns probability to each action
 - Sample actions according to policy π
 - Update policy:
 - If an episode led to high utility, make sampled actions more likely
 - If an episode led to low utility, make sampled actions less likely



Policy Gradient in a Single-Step MDP

- Consider a simple single-step Markov Decision Process (MDP)
 - The initial state is drawn from a distribution: $s \sim d(s)$
 - The process terminates after one action, yielding a reward r_{sa}
- Expected Value of the Policy

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$

Likelihood Ratio Trick

- Use the identity:

$$\begin{aligned}\frac{\partial \pi_\theta(a|s)}{\partial \theta} &= \pi_\theta(a|s) \frac{1}{\pi_\theta(a|s)} \frac{\partial \pi_\theta(a|s)}{\partial \theta} \\ &= \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta}\end{aligned}$$

- The gradient of the expected return can be written as:

$$\begin{aligned}J(\theta) &= \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa} \\ \frac{\partial J(\theta)}{\partial \theta} &= \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa} \\ &= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa} \\ &= \mathbb{E}_{\pi_\theta} \left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa} \right]\end{aligned}$$

Can be approximated by sampling s from $d(s)$ and a from π_θ

Extension to Multi-step MDP

- Replace the instantaneous reward $r(s,a)$ with the Q-value

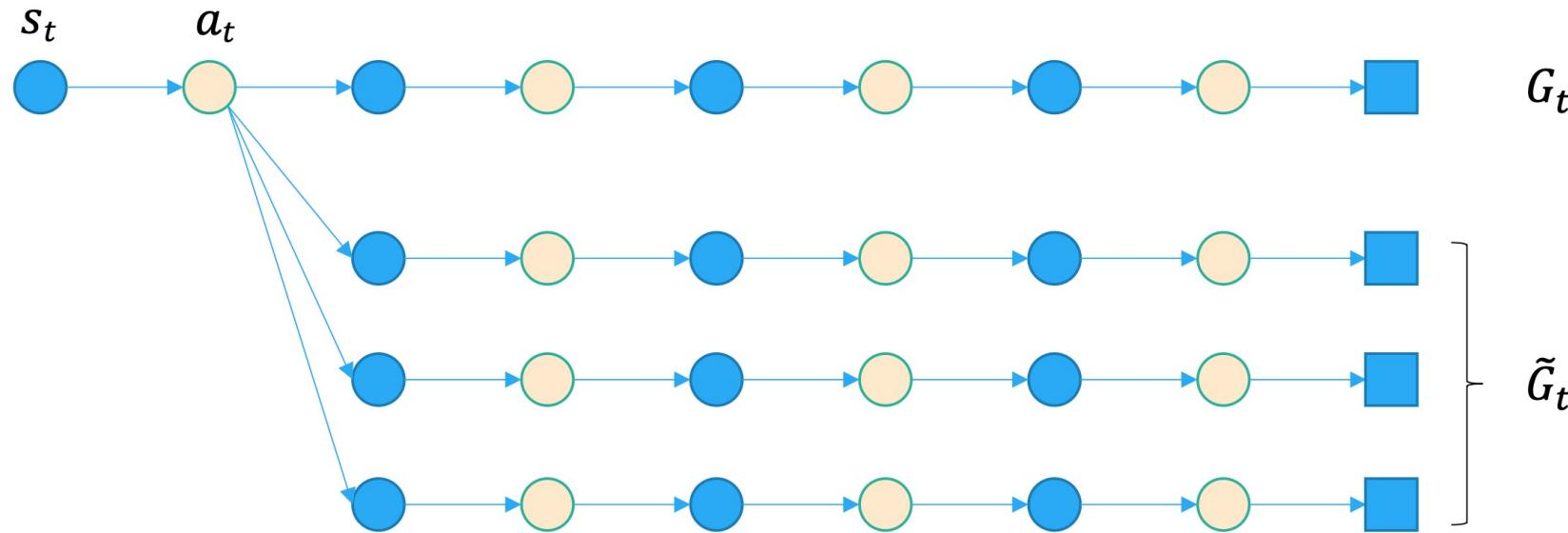
$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta} \left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s, a) \right]$$

REINFORCE Algorithm

- Use the cumulative reward G_t as an estimator for $Q^{\pi_\theta}(s, a)$
- initialize θ arbitrarily
for each episode $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
 for $t = 1$ to $T - 1$ do
 $\theta \leftarrow \theta + \alpha \frac{\partial}{\partial \theta} \log \pi_\theta(a_t | s_t) G_t$
 end for
end for
return θ

REINFORCE Algorithm 2

- Can average multiple roll-out returns



$$\tilde{G}_t = \frac{1}{N} \sum_{i=1}^n G_t^{(i)}$$

Limitations of the REINFORCE Algorithm

- **Episodic data requirement**
 - REINFORCE typically requires tasks to terminate in order to compute the full return G_t
- **Low data efficiency**
 - In practice, REINFORCE needs a large amount of training data to achieve stable learning
- **High variance in training (most critical issue)**
 - The estimated returns from sampled trajectories can have very high variance, making gradient estimates noisy and unstable

Actor-Critic

- Intuition
 - REINFORCE estimates the policy gradient using Monte Carlo returns G_t to approximate $Q(s_t, a_t)$
 - Why not learn a trainable value function $Q_\phi(s, a)$ to estimate $Q^\pi(s, a)$ directly?

- Actor and critic



Training of the Actor-Critic Algorithm

- Critic: $Q_\phi(s, a)$
 - Learns to accurately estimate the action-value under the current actor policy

$$Q_\Phi(s, a) \simeq r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a), a' \sim \pi_\theta(a'|s')} [Q_\Phi(s', a')]$$

- Actor: $\pi_\theta(a|s)$
 - Learns to take actions that maximize the critic's estimated value

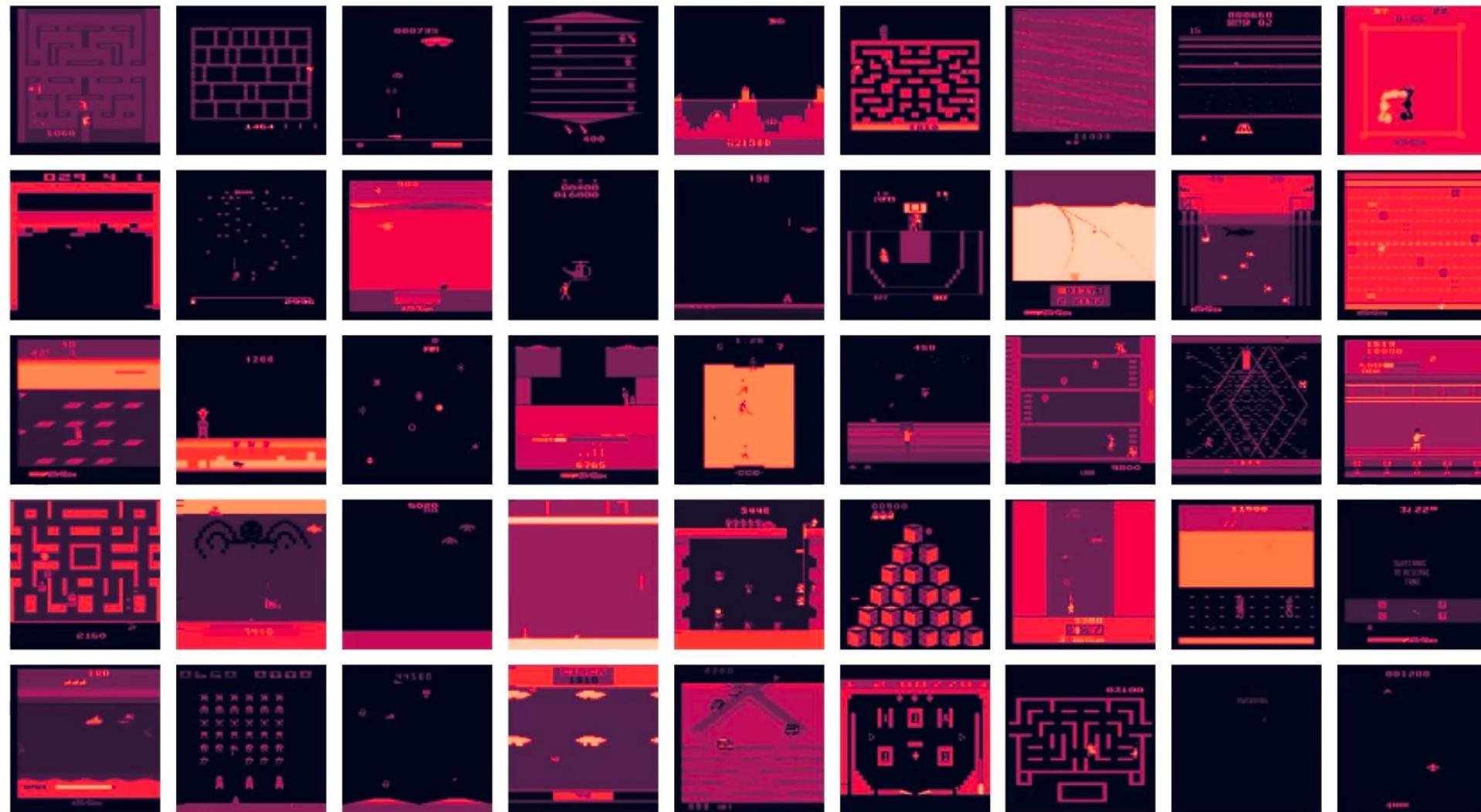
$$J(\theta) = \mathbb{E}_{s \sim p, \pi_\theta} [\pi_\theta(a|s) Q_\Phi(s, a)]$$

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta} \left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q_\Phi(s, a) \right]$$

Case Studies of Reinforcement Learning!

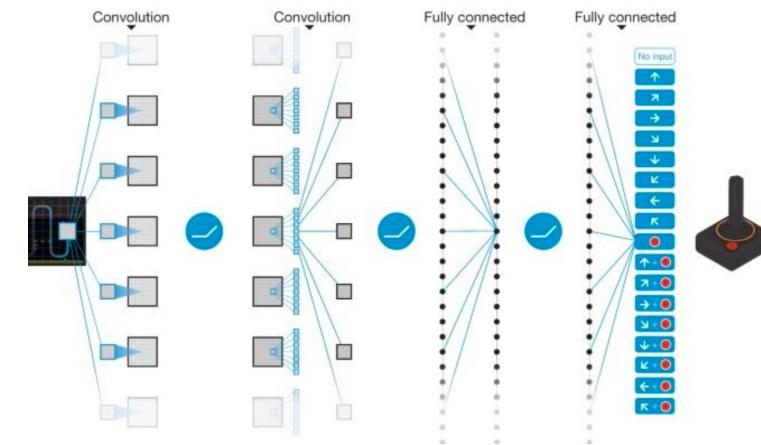
- Atari game playing
- Robot Locomotion
- Language assistants

Case Studies: Atari Game Playing



Case Studies: 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 our 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]

Case Studies: Robot Locomotion

- <https://www.youtube.com/watch?v=cqvAgcQl6s4>

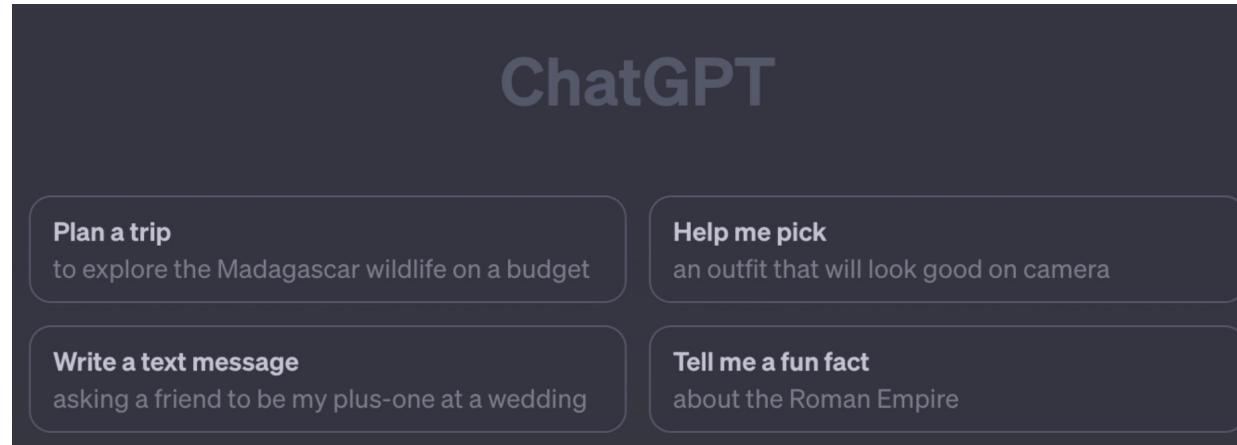
Case Studies: Robot Locomotion

- **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 $\max Q(s', a)$ 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



[Extreme Parkour with Legged Robots, Cheng et al, 2023]

Case Studies: Language Assistants



我是 DeepSeek, 很高兴见到你!

我可以帮你写代码、读文件、写作各种创意内容, 请把你的任务交给我吧~

给 DeepSeek 发送消息

深度思考 (R1)

联网搜索



Case Studies: 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

Case Studies: 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 $\max Q(s', a)$ 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)
- Often use policy gradient (Proximal Policy Optimization)

Summary

- Exploration in Q-learning
 - Epsilon greedy; optimistic function
- Scaling up with feature representations and approximation
- Policy gradient
 - REINFORCE; Actor-Critic
- Some case studies