CS 4803 / 7643: Deep Learning

Topics:

- Dynamic Programming (Q-Value Iteration)
- Reinforcement Learning (Intro, Q-Learning, DQNs)

Nirbhay Modhe

Georgia Tech

Topics we'll cover

- Overview of RL
  - RL vs other forms of learning
  - RL "API"
  - Applications
- Framework: Markov Decision Processes (MDP's)
  - Definitions and notations
  - Policies and Value Functions
  - Solving MDP's
    - Value Iteration (recap)
    - Q-Value Iteration (new)
    - Policy Iteration
- Reinforcement learning
  - Value-based RL (Q-learning, Deep-Q Learning)
  - Policy-based RL (Policy gradients)

Topics we'll cover

- Overview of RL
  - RL vs other forms of learning
  - RL "API"
  - Applications
- Framework: Markov Decision Processes (MDP's)
  - Definitions and notations
  - Policies and Value Functions
  - Solving MDP's
    - Value Iteration (recap)
    - Q-Value Iteration (new)
    - Policy Iteration
- Reinforcement learning
  - Value-based RL (Q-learning, Deep-Q Learning)
  - Policy-based RL (Policy gradients)

Recap

Recap

- Markov Decision Process (MDP)
  - Defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$

$\mathcal{S}$: set of possible states [start state = s_0, optional terminal / absorbing state]
$\mathcal{A}$: set of possible actions
$\mathcal{R}(s, a, s')$: distribution of reward given (state, action, next state) tuple
$\mathbb{T}(s, a, s')$: transition probability distribution, also written as $p(s'|s, a)$
γ : discount factor

Recap

- Markov Decision Process (MDP)
  - Defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$

$\mathcal{S}$: set of possible states [start state = s_0, optional terminal / absorbing state]
$\mathcal{A}$: set of possible actions
$\mathcal{R}(s, a, s')$: distribution of reward given (state, action, next state) tuple
$\mathbb{T}(s, a, s')$: transition probability distribution, also written as $p(s'|s, a)$
γ : discount factor

- Value functions, optimal quantities, bellman equation

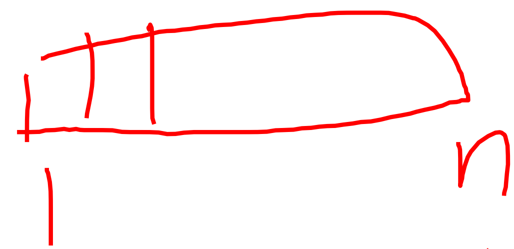
- Algorithms for solving MDP's
  - Value Iteration

Value Function

Following policy π that produces sample trajectories s_0, a_0, r_0, s_1, a_1, …

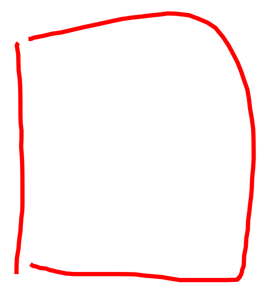
$|S| = n$

$s \to p(a|s)$



$|A| = m$

$\pi \quad n \times m$



Value Function

Following policy π that produces sample trajectories s_0, a_0, r_0, s_1, a_1, …

How good is a state?

The **value function** at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$n = |S|$

s'

$$V^\pi(s) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, \pi\right]$$

$p(s'|s, a)$ $p(r|s, a)$

$\pi(a|s)$

Value Function

Following policy π that produces sample trajectories s_0, a_0, r_0, s_1, a_1, …

How good is a state?
The **value function** at state s, is the expected cumulative reward from state s (and following the policy thereafter):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

How good is a state-action pair?
The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s (and following the policy thereafter):

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Optimal Quantities

Given *optimal* policy π^* that produces sample trajectories s_0, a_0, r_0, s_1, a_1, …

n 1

How good is a state?

The **optimal value function** at state s, and acting optimally thereafter

$$V^*(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi^*\right]$$

V n 1

Optimal Quantities

Given *optimal* policy π^* that produces sample trajectories s_0, a_0, r_0, s_1, a_1, …

How good is a state?
The **optimal value function** at state s, and acting optimally thereafter

$$V^*(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi^*\right]$$

How good is a state-action pair?
The **optimal Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and acting optimally thereafter

$$Q^*(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi^*\right]$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

$p(s'|s,a)$

- Recursive optimality equations:

$$Q^*(s,a) =$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

- Recursive optimality equations:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim p(s'|s,a)} \left[r(s,a) + \gamma V^*(s') \right]$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

- Recursive optimality equations:

$$\begin{aligned} Q^*(s,a) &= \mathop{\mathbb{E}}_{s' \sim p(s'|s,a)} \left[r(s,a) + \gamma V^*(s) \right] \\ &= \sum_{s'} \underline{p(s'|s,a)} \left[r(s,a) + \gamma V^*(s) \right] \end{aligned}$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

- Recursive optimality equations:

$$\begin{aligned}
Q^*(s,a) &= \mathop{\mathbb{E}}_{s' \sim p(s'|s,a)} \left[r(s,a) + \gamma V^*(s) \right] \\
&= \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^*(s) \right] \\
&= \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma \max_a Q^*(s',a') \right]
\end{aligned}$$

Bellman Optimality Equations

- Relations:

$$\boxed{V^*(s) = \max_a Q^*(s,a)} \qquad \boxed{\pi^*(s) = \arg\max_a Q^*(s,a)}$$

- Recursive optimality equations:

$$\begin{aligned} Q^*(s,a) &= \mathop{\mathbb{E}}_{s'\sim p(s'|s,a)} \left[r(s,a) + \gamma V^*(s) \right] \\ &= \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^*(s) \right] \\ &= \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma \max_a Q^*(s',a') \right] \end{aligned}$$

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^*(s') \right]$$

Q

|A|

m

n

|S|

Value Iteration (VI)

- Based on the bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^*(s')\right]$$

Value Iteration (VI)

- Based on the bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} p\left(s'|s,a\right)\left[r(s,a) + \gamma V^*\left(s'\right)\right]$$

- Algorithm
  - Initialize values of all states $V^0(s) = 0$
  - While not converged:
    - For each state:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]$$

  - Repeat until convergence (no change in values)

Homework

V_0

0 0 0 0 0 0

S^n

V^1

$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \quad \cdots \rightarrow V^i \rightarrow \ldots \quad \rightarrow V^*$$

Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

Q-Value Iteration

- Value Iteration Update:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^i(s') \right]$$

- Q-Value Iteration Update:

$$Q^{i+1}(s,a) \leftarrow$$

Q-Value Iteration

- Value Iteration Update:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[r(s,a) + \gamma V^i(s') \right]$$

- Q-Value Iteration Update:

$$Q^{i+1}(s,a) \leftarrow \sum_{s'} p\left(s'|s,a\right) \left[r\left(s,a\right) + \gamma \max_{a'} Q^i(s',a') \right]$$

The algorithm is same as value iteration,
but it loops over actions as well as states

Policy Iteration

Policy Iteration

- Policy iteration: Start with arbitrary π_0 and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

Policy Iteration

- Policy iteration: Start with arbitrary π_0 and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

- Involves repeating two steps:
  - Policy Evaluation: Compute V^π (similar to VI)
  - Policy Refinement: Greedily change actions as per V^π

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \quad \ldots \quad \longrightarrow \pi^* \longrightarrow V^{\pi^*}$$

Policy Iteration

- Policy iteration: Start with arbitrary π_0 and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

- Involves repeating two steps:
  - Policy Evaluation: Compute V^π (similar to VI)
  - Policy Refinement: Greedily change actions as per V^π

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \quad \ldots \quad \longrightarrow \pi^* \longrightarrow V^{\pi^*}$$

- Why do policy iteration?
  - π_i often converges to π^* much sooner than V^{π_i}

Summary

- Value Iteration
  - Bellman update to state value estimates
- Q-Value Iteration
  - Bellman update to (state, action) value estimates
- Policy Iteration
  - Policy evaluation + refinement

Learning Based Methods

Learning Based Methods

- Typically, we don't know the environment
  - $\mathbb{T}(s, a, s')$ unknown, how actions affect the environment.
  - $\mathcal{R}(s, a, s')$ unknown, what/when are the good actions?

Learning Based Methods

- Typically, we don't know the environment
  - $\mathbb{T}(s, a, s')$ unknown, how actions affect the environment.
  - $\mathcal{R}(s, a, s')$ unknown, what/when are the good actions?
- But, we can learn by trial and error.
  - Gather experience (data) by performing actions.

$$\{s, a, s', r\}_{i=1}^{N}$$

  - Approximate unknown quantities from data.

Reinforcement Learning

Learning Based Methods

- Old Dynamic Programming Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

- RL Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Reinforcement Learning

(Deep) Learning Based Methods

(Deep) Learning Based Methods

- In addition to not knowing the environment, sometimes the state space is too large.

(Deep) Learning Based Methods

- In addition to not knowing the environment, sometimes the state space is too large.

- A value iteration updates takes $O(|\mathcal{S}|^2|\mathcal{A}|)$
  - Not scalable to high dimensional states e.g.: RGB images.

(Deep) Learning Based Methods

- In addition to not knowing the environment, sometimes the state space is too large.

- A value iteration updates takes $O(|\mathcal{S}|^2|\mathcal{A}|)$
  - Not scalable to high dimensional states e.g.: RGB images.

- Solution: Deep Learning!
  - Use deep neural networks to learn low-dimensional representations.

$\boxed{\text{Deep Reinforcement Learning}}$

Reinforcement Learning

Reinforcement Learning

- Value-based RL
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

$Q(s, a, \theta)$

Reinforcement Learning

- Value-based RL
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

- Policy-based RL
  - Directly approximate optimal policy π^* with a parametrized policy π_θ^*

Reinforcement Learning

- Value-based RL
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network

- Policy-based RL
  - Directly approximate optimal policy π^* with a parametrized policy π^*_θ

- Model-based RL
  - Approximate transition function $T(s', a, s)$ and reward function $\mathcal{R}(s, a)$
  - Plan by looking ahead in the (approx.) future!

Reinforcement Learning

- Value-based RL
  - (Deep) Q-Learning, approximating $Q^*(s,a)$ with a deep Q-network
- Policy-based RL
  - Directly approximate optimal policy π^* with a parametrized policy π^*_θ
- Model-based RL
  - Approximate transition function $T(s', a, s)$ and reward function $\mathcal{R}(s,a)$
  - Plan by looking ahead in the (approx.) future!

Homework!

Value-based Reinforcement Learning

Deep Q-Learning

Deep Q-Learning

- Q-Learning with linear function approximators

$$Q(s, a; w, b) = w_a^\top s + b_a$$

多 d

  - Has some theoretical guarantees

Deep Q-Learning

HxW

S

$\theta \to f_{\theta}(\ )$

n

- Q-Learning with linear function approximators

$$Q(s,a;w,b) = w_a^\top s + b_a$$

  - Has some theoretical guarantees

- Deep Q-Learning: Fit a deep Q-Network $Q(s,a;\theta)$
  - Works well in practice
  - Q-Network can take RGB images

10 256

256 → (0,1)

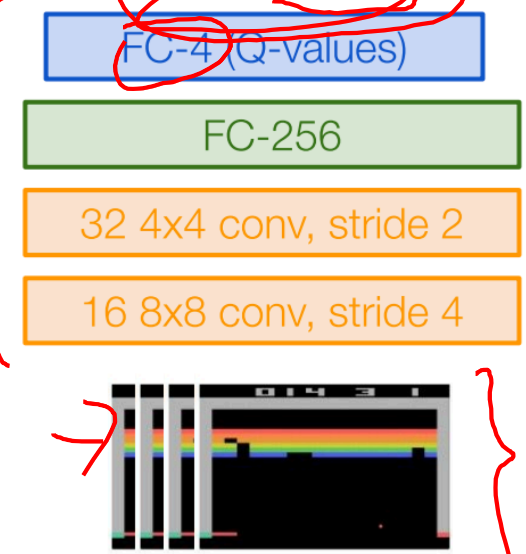


Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Deep Q-Learning

Deep Q-Learning

- Assume we have collected a dataset

$$\{(s, a, s', r)_i\}_{i=1}^{N}$$

- We want a Q-function that satisfies:

Q-Value Bellman Optimality

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim p(s'|s,a)} \left[r(s,a) + \gamma \max_{a'} Q^*(s', a') \right]$$

- Loss for a single data point:

$$\text{MSE Loss} := \Big(\underbrace{Q_{new}(s,a)}_{\text{Predicted Q-Value}} - \underbrace{(r + \max_{a} Q_{old}(s', a))}_{\text{Target Q-Value}} \Big)^2$$

Deep Q-Learning

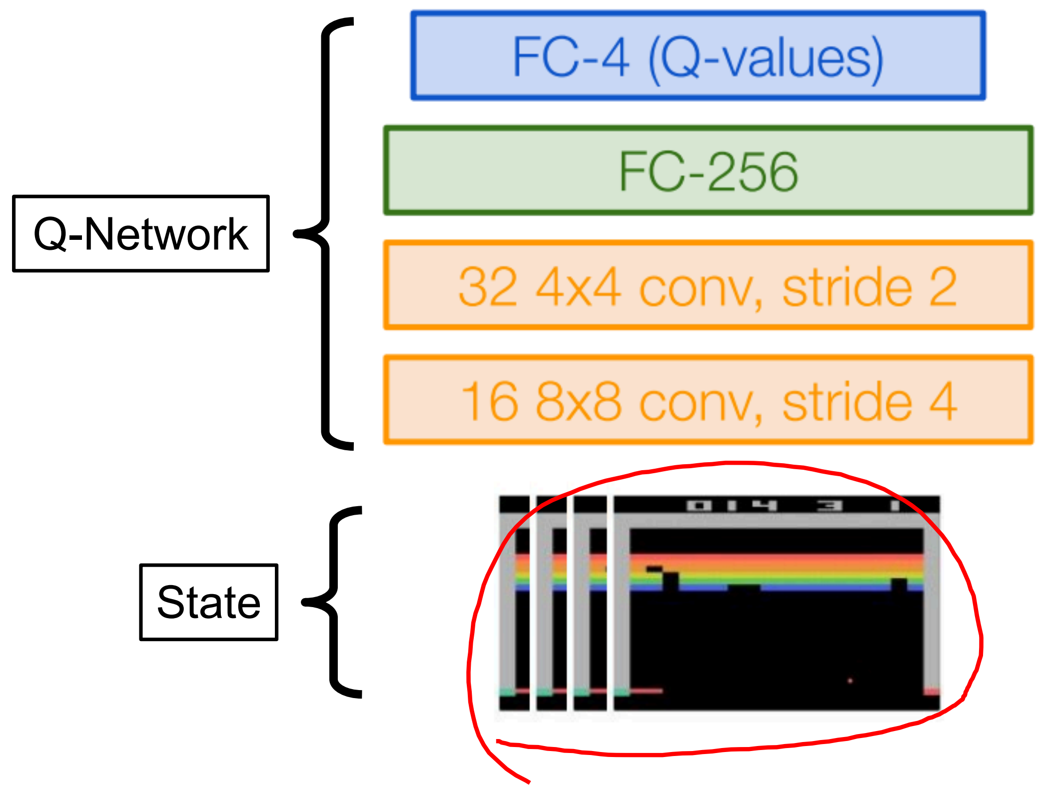
- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$

- Forward pass: State ($B \times D$) → Q-Network → Q-Values per action ($B \times n_{actions}$)

Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$
- Forward pass: State $\rightarrow$ Q-Network $\rightarrow$ Q-Values per action

  State: $B \times D$; Q-Values per action: $B \times n_{actions}$



Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$
- Forward pass: $\underset{B \times D}{\boxed{\text{State}}} \rightarrow \boxed{\text{Q-Network}} \rightarrow \underset{B \times n_{actions}}{\boxed{\text{Q-Values per action}}}$
- Compute loss: $$\left(\underbrace{Q_{new}(s, a)}_{\theta_{new}} - \left(r + \max_{a} \underbrace{Q_{old}(s', a)}_{\theta_{old}}\right)\right)^2$$

Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$
- Forward pass: State ($B \times D$) → Q-Network → Q-Values per action ($B \times n_{actions}$)
- Compute loss: $$\Big(\underbrace{Q_{new}(s,a)}_{\theta_{new}} - (r + \max_a \underbrace{Q_{old}(s',a)}_{\theta_{old}})\Big)^2$$

Deep Q-Learning

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$

- Forward pass: State → Q-Network → Q-Values per action

  State: $B \times D$ → $H \times W \times C$

  Q-Values per action: $B \times n_{actions}$

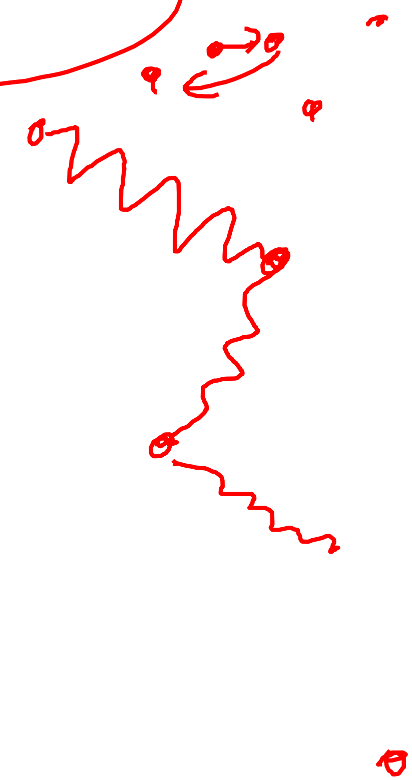
- Compute loss: $$\Big(\underbrace{Q_{new}(s,a)}_{\theta_{new}} - \big(r + \gamma \max_a \underbrace{Q_{old}(s',a)}_{\theta_{old}}\big)\Big)^2$$

- Backward pass: $$\frac{\partial Loss}{\partial \theta_{new}}$$

Deep Q-Learning

$$\text{MSE Loss} := \left(Q_{new}(s,a) - \left(r + \gamma \max_a \underline{Q_{old}(s',a)}\right)\right)^2$$

- In practice, for stability:
  - Freeze Q_{old} and update Q_{new} parameters
  - Set $Q_{old} \leftarrow Q_{new}$ at regular intervals



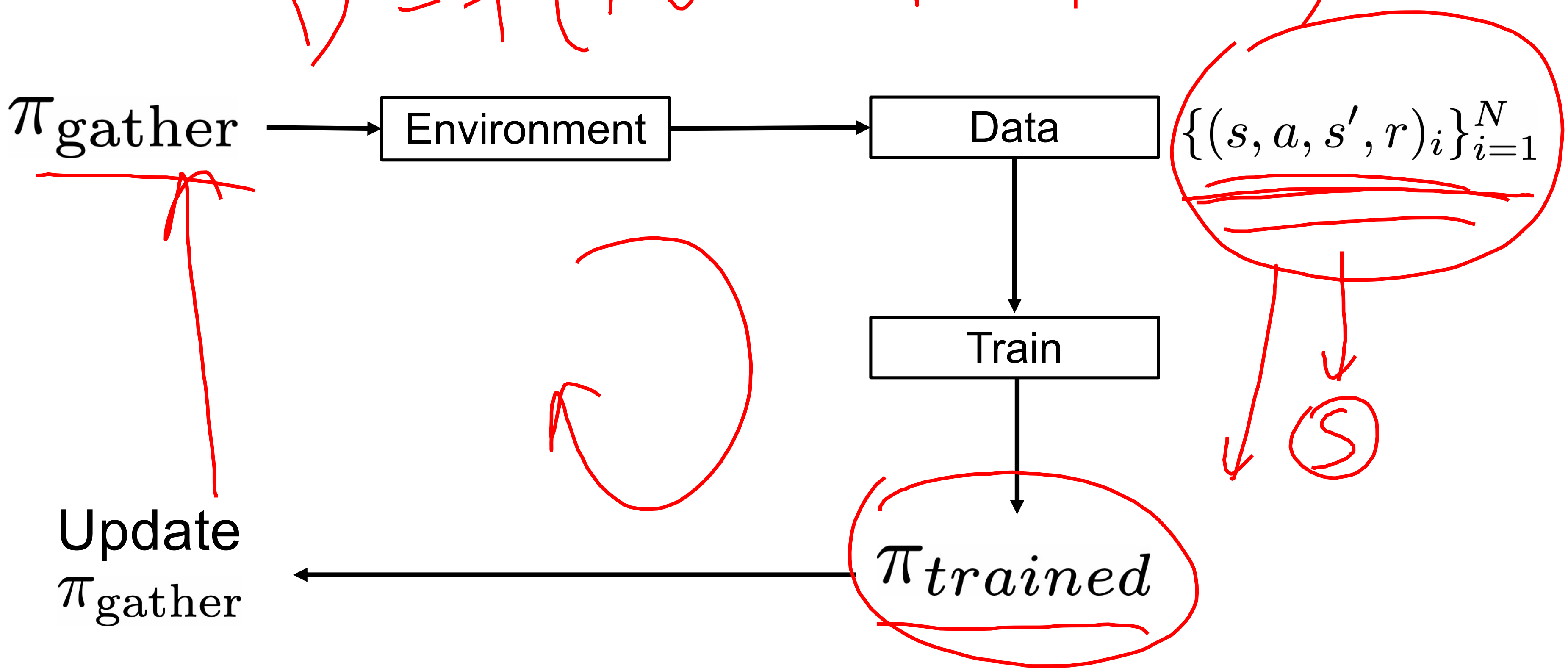
How to gather experience?

$$\{(s, a, s', r)_i\}_{i=1}^{N}$$

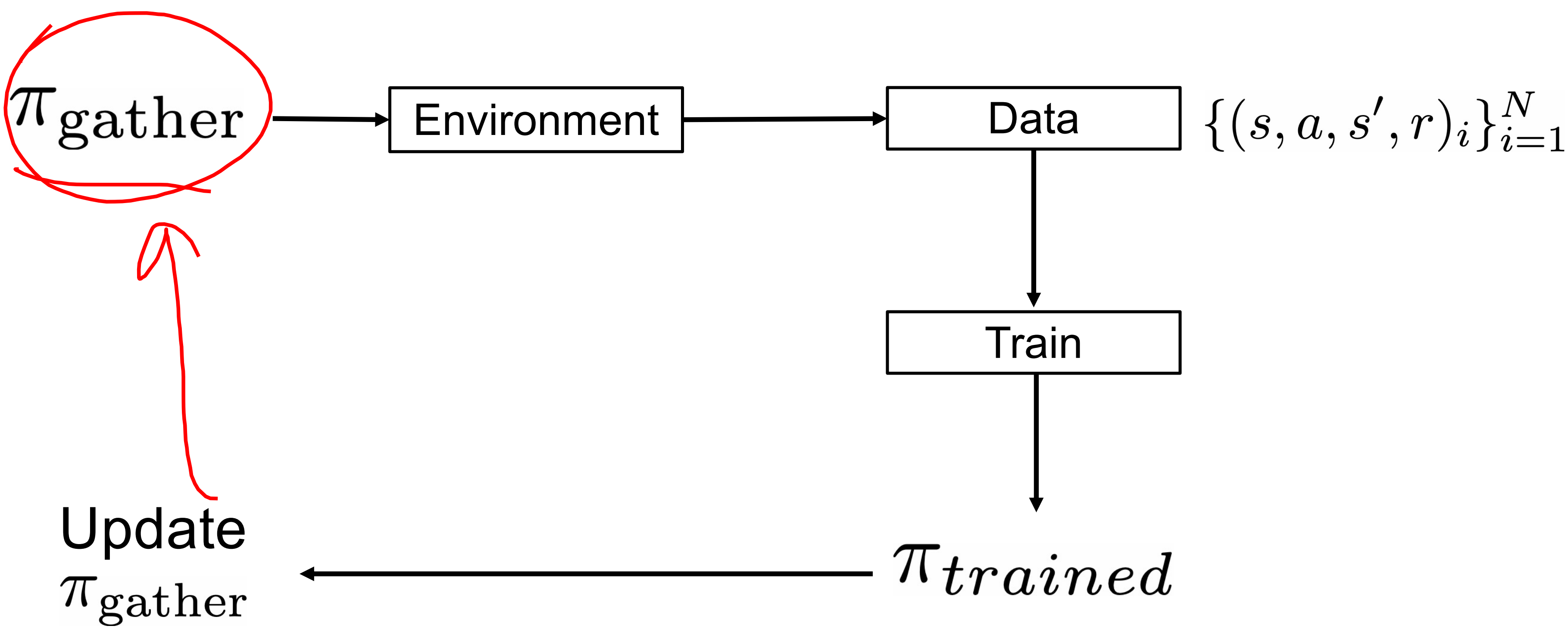
This is why RL is hard

How To Gather Experience?

$D = f(\pi_{trained})$ $\quad P = f(D)$



How To Gather Experience?



Challenge 1: Exploration vs Exploitation

Challenge 2: Non iid, highly correlated data

Exploration Problem

- What should π_{gather} be?
  - Greedy? -> Local minimas, no exploration

$$\arg\max_a Q(s, a; \theta)$$

Exploration Problem

- What should π_{gather} be?
  - Greedy? -> Local minimas, no exploration

$$\arg\max_a Q(s, a; \theta)$$

- An exploration strategy:
  - ϵ-greedy

$$a_t = \begin{cases} \arg\max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Correlated Data Problem

- Samples are correlated => high variance gradients => inefficient learning
- Current Q-network parameters determines next training samples => can lead to bad feedback loops
  - e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size.

Experience Replay

- Address this problem using experience replay
  - A replay buffer stores transitions (s, a, s', r)

Experience Replay

- Address this problem using experience replay
  - A replay buffer stores transitions (s, a, s', r)
  - Continually update replay buffer as game (experience) episodes are played, older samples discarded

Experience Replay

- Address this problem using experience replay
  - A replay buffer stores transitions (s, a, s', r)
  - Continually update replay buffer as game (experience) episodes are played, older samples discarded
  - Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Q-Learning Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Experience Replay

Epsilon-greedy

Q Update

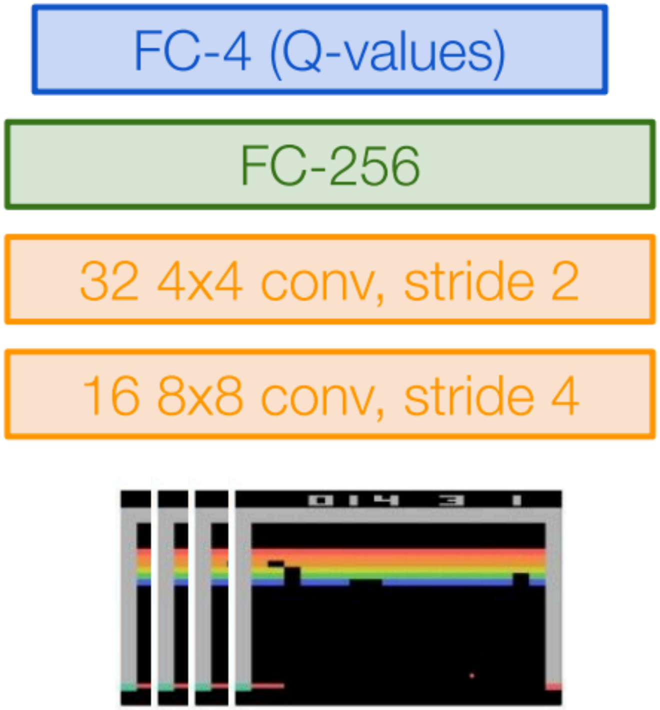
Case study: Playing Atari Games

- Objective: Complete the game with the highest score

- State: Raw pixel inputs from the game state
- Action: Game controls e.g.: Left, Right, Up, Down
- Reward: Score increase/decrease at each time step

Playing Atari Games

- Q-Network architecture

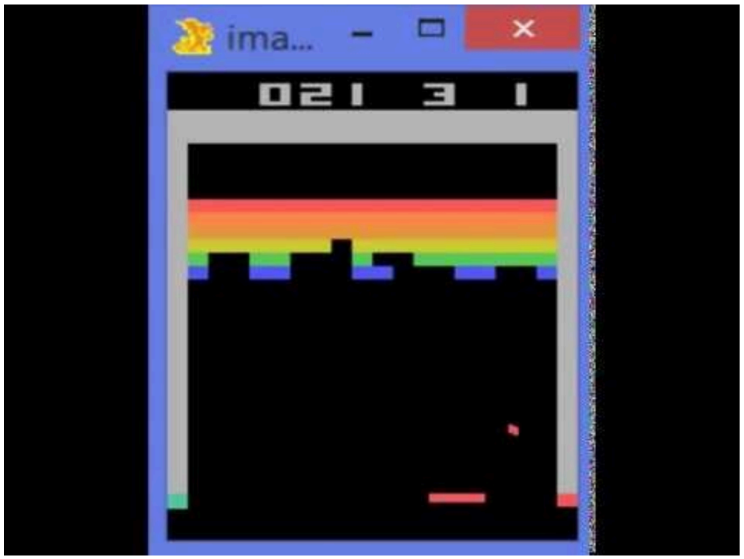


- State:
  - Stack of 4 image frames, grayscale conversion, down-sampling and cropping to (84 x 84 x 4)

- Last FC layer has #(actions) dimensions (predicts Q-values)

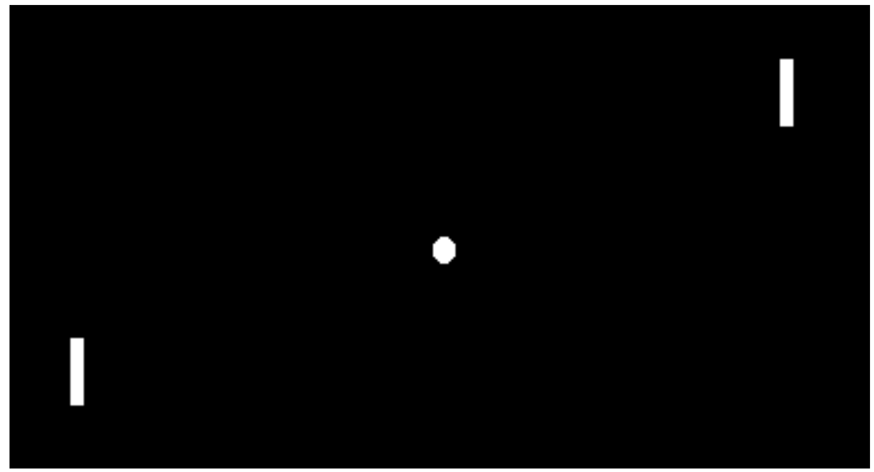
Atari Games

Breakout



https://www.youtube.com/watch?v=V1eYniJ0Rnk

Pong



Summary

In today's class, we looked at

- Dynamic Programming
  - Q-Value Iteration
  - Policy Iteration
- Reinforcement Learning (RL)
  - The challenges of (deep) learning based methods
  - Value-based RL algorithms
    - Deep Q-Learning

Next class:

- Policy-based RL algorithms

Thanks!