Previous Lecture
- **MDPs**: Value function, optimal quantities, algorithms for solving MDPs
- **RL**: No rewards and transitions (RL), function approximation (deep RL).

Today
- **RL Algorithms**
  - Overview, types of RL algorithms
  - Deep-Q Learning: A value based RL algorithm
  - Policy gradients: A policy-based RL algorithm
Recursive Bellman expansion (from definition of Q)

\[
Q^*(s, a) = \mathbb{E}_{\begin{array}{c}
    a_t \sim \pi^*(\cdot | s_t) \\
    s_{t+1} \sim p(\cdot | s_t, a_t)
  \end{array}} \left[ \sum_{t \geq 0} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]
\]

(Expected) return from \( t = 0 \)

(Reward at \( t = 0 \)) + \( \gamma \) \times (Return from expected state at \( t=1 \))
RL setting:

- $P(s, a, s')$ unknown, how actions affect the environment
- $R(s, a, s')$ unknown, what/when are the good actions?

Deep RL setting:

- Large or continuous state space
- Use deep neural networks to learn state representations
Value-based RL
- (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network (DQN)

Policy-based RL
- Directly approximate optimal policy $\pi^*$ with a parametrized policy $\pi^*_\theta$

Model-based RL
- Approximate transition function $T(s', a, s)$ and reward function $R(s, a)$
- Plan by looking ahead in the (approx.) future!
Deep Q-Learning
Intuition: Learn a **parametrized Q-function** from data $\{(s, a, s', r)\}_{i=1}^{N}$

Q-Learning with linear function approximators

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

Deep Q-Learning: Fit a deep Q-Network $Q(s, a; \theta)$

- Works well in practice
- Q-Network can take RGB images

Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Objective: Complete the game with the highest score
State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Atari Games

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Assume we have collected a dataset:
\[ \{(s, a, s', r)_i\}_{i=1}^{N} \]

We want a Q-function that satisfies bellman optimality (Q-value):

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

Loss for a single data point:

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

- Predicted Q-Value
- Target Q-Value
Minibatch of \( \{(s, a, s', r)\}_{i=1}^{B} \)

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

- **Compute loss:**
  - \( \theta_{new} \)

- **Backward pass:**
  - \( \frac{\partial Loss}{\partial \theta_{new}} \)
In practice, for stability:

- Freeze $Q_{old}$ and update $Q_{new}$ parameters
- Set $Q_{old} \leftarrow Q_{new}$ at regular intervals
Assuming a fixed dataset, the MSE Loss can be optimized. This is known as the **Fitted Q-Iteration** algorithm.

However…

How to gather experience or “data”? 

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

This is why RL is hard.
How to gather experience?

Environment → Data

{((s_i, a_i, s'_i, r_i))_{i=1}^N}

Train

Update

\(\pi_{\text{gather}}\) → \(\pi_{\text{trained}}\)

Challenge 1: Exploration vs Exploitation

Challenge 2: Non iid, highly correlated data
What should \( \pi_{\text{gather}} \) be?

- Greedy? \( \rightarrow \) Local minima, no exploration
  \[
  \arg \max_a Q(s, a; \theta)
  \]

- An exploration strategy:
  
  - \( \epsilon \)-greedy
    
    \[
    a_t = \begin{cases} 
    \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\
    \text{random action} & \text{with probability } \epsilon
    \end{cases}
    \]
    
    \( 0.5 \rightarrow 0.1 \)
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 right, training samples will be dominated by samples going right, may fall into local minima
Correlated data: addressed by 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

- Larger the buffer, lower the correlation
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $D$ 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 $\epsilon$ 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 $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    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

Policy Gradients, Actor-Critic
Class of policies defined by parameters $\theta$

$$\pi_\theta(a|s) : S \rightarrow A$$

Eg: $\theta$ can be parameters of linear transformation, deep network, etc.

Want to maximize:

$$J(\pi) = \mathbb{E} \left[ \sum_{t=1}^{T} R(s_t, a_t) \right]$$

In other words,

$$\pi^* = \arg \max_{\pi:S \rightarrow A} \mathbb{E} \left[ \sum_{t=1}^{T} R(s_t, a_t) \right] \quad \Rightarrow \quad \theta^* = \arg \max_{\theta} \mathbb{E} \left[ \sum_{t=1}^{T} R(s_t, a_t) \right]$$
Pong from Pixels
Policy Gradient: Loss Function

Forward pass:
- Image
- Block of differentiable compute (e.g., neural net)

Backward pass:
- Log probabilities
  - Correct action
    - Label = 0
    - Correct action
  - Sampled action
    - Sampled action = 1
- Eventual reward
  - Reward = -1.0

Supervised Learning (correct label is provided):
- Correct action
  - Log probabilities
    - (-1.2, -0.36)
  - Gradients
    - (1.0, 0)

Reinforcement Learning:
- Sampled action
  - Log probabilities
    - (-1.2, -0.36)
  - Gradients
    - (0, -1.0)
Slightly re-writing the notation

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

$$\pi_\theta(\tau) = p_\theta(\tau) = p_\theta(s_0, a_0, \ldots s_T, a_T)$$

$$= p(s_0) \prod_{t=0}^{T} p_\theta(a_t \mid s_t) \cdot p(s_{t+1} \mid s_t, a_t)$$

$$\arg \max_{\theta} \mathbb{E}_{\tau \sim p_\theta(\tau)} [\mathcal{R}(\tau)]$$
How to gather data?

- We already have a policy: \( \pi_\theta \)
- Sample \( N \) trajectories \( \{\tau_i\}_{i=1}^N \) by acting according to \( \pi_\theta \)

\[
J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\mathcal{R}(\tau)] \\
= \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)} \left[ \sum_{t=0}^{T} \mathcal{R}(s_t, a_t) \right]
\]

\[
\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} r(s_t^i, a_t^i)
\]
Sample trajectories \( \tau_i = \{s_1, a_1, \ldots, s_T, a_T\} \) by acting according to \( \pi_\theta \)

Compute policy gradient as

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \left[ \sum_{t=1}^T \nabla\log\pi_\theta (a_t^i | s_t^i) \cdot \sum_{t=1}^T R(s_t^i | a_t^i) \right]
\]

Update policy parameters:

\[
\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)
\]

Run the policy and sample trajectories

Compute policy gradient

Update policy

The REINFORCE Algorithm
\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i}^{N} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta (a^i_t \mid s^i_t) \cdot \sum_{t=1}^{T} R (s^i_t \mid a^i_t) \right] \]
\( \nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] \)

\[ = \nabla_\theta \int \pi_\theta(\tau) R(\tau) d\tau \]

\[ = \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau \]

\[ = \int \left( \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} \right) R(\tau) d\tau \]

\[ = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau \]

\[ = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log \pi_\theta(\tau) R(\tau) \right] \]

Expectation as integral

Exchange integral and gradient

\( \nabla_\theta \log \pi(\tau) = \frac{\nabla_\theta \pi(\tau)}{\pi(\tau)} \)
Deriving The Policy Gradient

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \nabla_\theta \log \pi_\theta(\tau) R(\tau) \right] \]

\[ \nabla_\theta \left[ \log p(s_0) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1} | s_t, a_t) \right] \]

\[ = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t=1}^{T} R(s_t, a_t) \right] \]

Doesn’t depend on Transition probabilities!

\[ s_t \quad \pi_\theta(a_t|s_t) \quad a_t \]
Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to $\pi \theta$

Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \left[ \sum_{t=1}^T \nabla_\theta \log \pi_\theta (a_t^i | s_t^i) \cdot \sum_{t=1}^T R (s_t^i | a_t^i) \right]$$

Update policy parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Run the policy and sample trajectories

Compute policy gradient

Update policy

The REINFORCE Algorithm

Slide credit: Sergey Levine
Drawbacks of Policy Gradients

Slide credit: Dhruv Batra
Credit assignment is hard!
- Which specific action led to increase in reward
- Suffers from **high variance**, leading to unstable training

How to reduce the variance?
- Subtract an action independent baseline from the reward

\[
\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[ \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \cdot \sum_{t=1}^{T} (R(s_t, a_t) - b(s_t)) \right]
\]

Why does it work?
- What is the best choice of \( b \)?

**Drawbacks of Policy Gradients**
- REINFORCE, use raw reward values

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) R(s, a)]$$

- Actor-critic, use Q-values (learnt from data)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)]$$

- Advantage actor-critic, use Q minus V values (i.e. Advantage)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) (Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s))]$$

Policy Gradient Variants