Reinforcement Learning

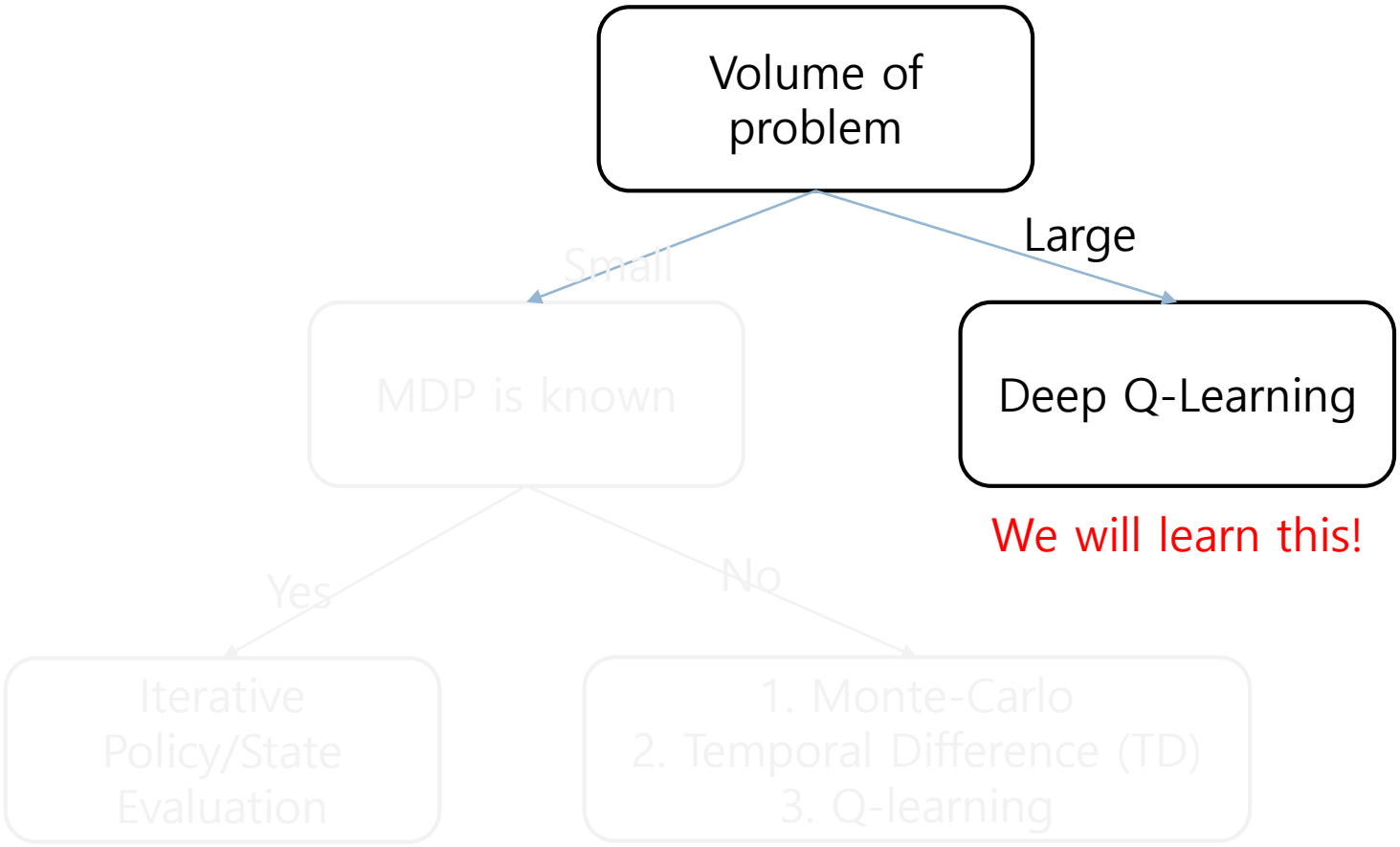
[ch.6 DQN: Deep Q-Learning]

DAEHAN AHN

AI Convergence

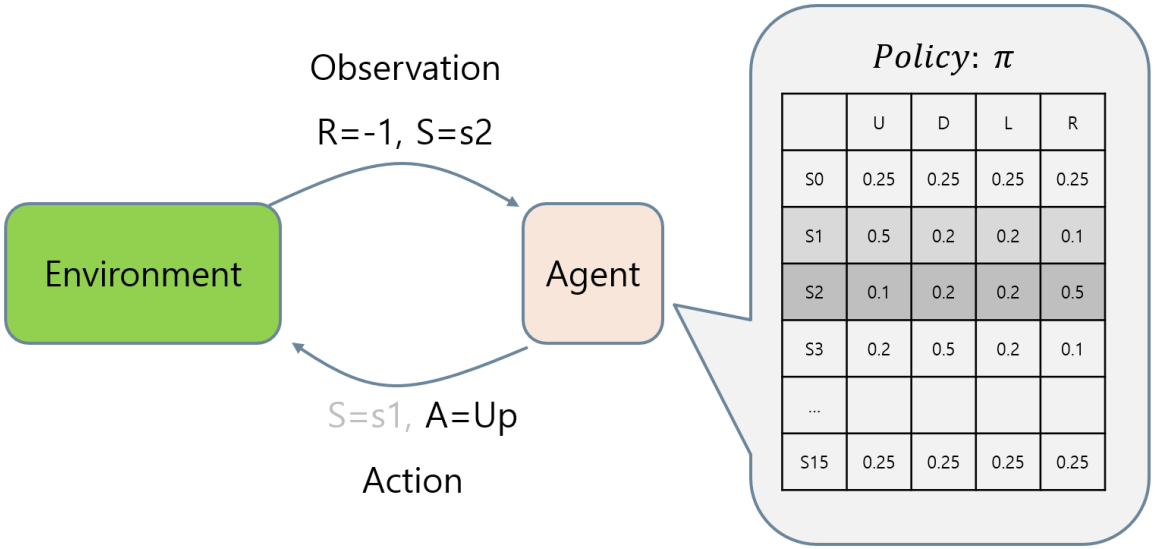
울산대학교
UNIVERSITY OF ULSAN

How to solve MDP problem



Limitation of Tabular Method

- **Previous models use tabular method for planning and learning**
  - They require a memory to store the policies for each state
  - But our real problem is more huge and complex
    - 큰 테이블을 사용하는 것은 비효율적
    - 실제 문제에서는 학습되지 않은 칸이 발생할 수 있음



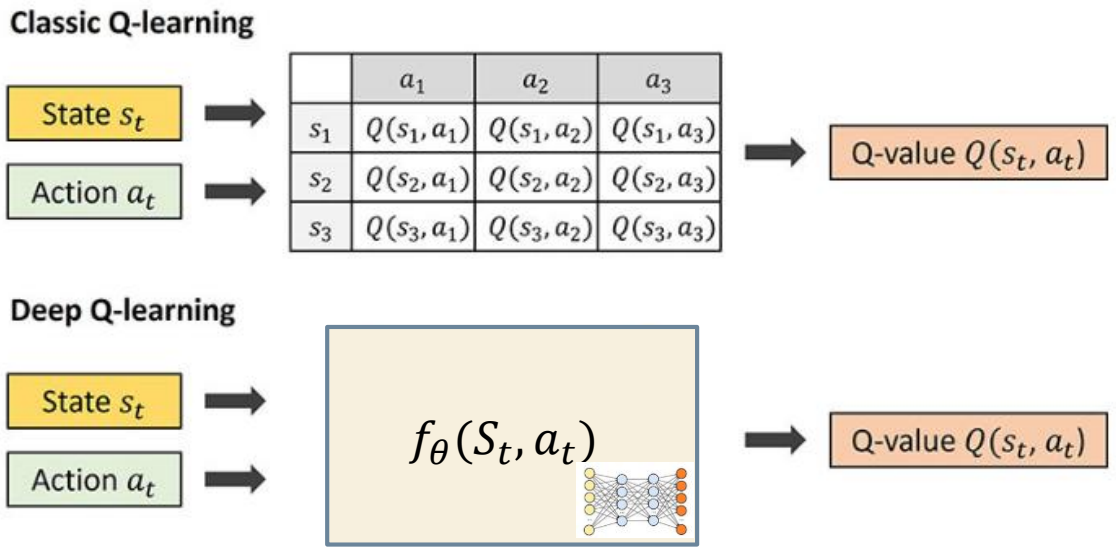
Backgammon: 10^{20} states
Go: 10^{170} states
Helicopter: continuous state space

Approximate Methods

- **Solution for large MDPs:**
  - Estimate value function with *function approximation*

Array ← $v_\pi(s) \approx \hat{v}(s;\boldsymbol{\theta})$ or $q_\pi(s,a) \approx \hat{q}(s,a;\boldsymbol{\theta})$ → Neural Net

  - Generalize from seen states to unseen states
  - Update parameter $\boldsymbol{\theta}$ using MC or TD learning



Deep Q Learning

- **'Playing Atari with Deep Reinforcement Learning', NIPS 2013**
- **'Human-level control through deep reinforcement learning', Nature 2015**
  - **Author: David Silver, et al.**

Deep Q Learning

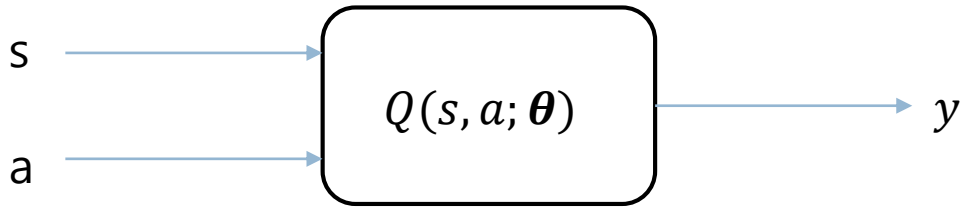
Definition: Deep Q Learning

- **기존의 Q Learning을 딥러닝을 확장한 방법**
- **State가 입력되면 행동의 가치를 예측**

$$Q_\pi(s,a) \approx Q(s,a;\boldsymbol{\theta})$$

- **상태가치 *V*는 사용하지 않음 (행동가치 *Q*만 사용)**
  - **행동 비교 불가:** 상태 S에서 여러 행동이 있을 때, 어떤 행동이 더 좋은지 알 수 없음
  - **정책 개선 불가:** 주어진 정책이 얼마나 좋은지만 측정할 수 있고, 새로운 정책으로 업데이트 어려움

  - -> Q를 사용하는 이유는 샘플링으로 부터 즉시 정책개선에 대한 학습이 가능



Deep Q Learning

- **학습 방법**

알고리즘	업데이트 방식	핵심 오차 개념
TD(0)	$\mathrm{V(s)} = \mathrm{V(s)} + \alpha[R + \gamma V(s') - V(s)]$	TD error
SARSA	$\mathrm{Q(s,a)} = \mathrm{Q(s,a)} + \alpha[R + \gamma Q(s', a') - Q(s, a)]$	TD error
Q-learning	$\mathrm{Q(s,a)} = \mathrm{Q(s,a)} + \alpha\left[R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$	TD error
DQN	$\mathbf{L(\theta)} = \left[\left(\boldsymbol{R} + \boldsymbol{\gamma} \max_{\boldsymbol{a'}} \boldsymbol{Q}(\boldsymbol{s'}, \boldsymbol{a'}; \boldsymbol{\theta}^{-})\right) - \boldsymbol{Q}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})\right]^{\mathbf{2}}$	Loss (target | prediction)

Problems

- **Q-learning with value function approximation can be unstable and diverge (do not converges to the optimal q*(s,a))**
  - **P1) highly correlated state**
    - In a continuous environment, the states at t, t+1, t+2, … are almost same
    - It makes generalization difficult
  - **P2) Target oscillation problem**
    - $R + \gamma \max_{a'} q(s', a') - q(s, a)$
    - if the learning model $q(s, a)$ is updated, the learning target $R + \gamma \max_{a'} q(s', a')$ is changed
    - Thus, even though the model has not adequately learned, the target value (true value) change

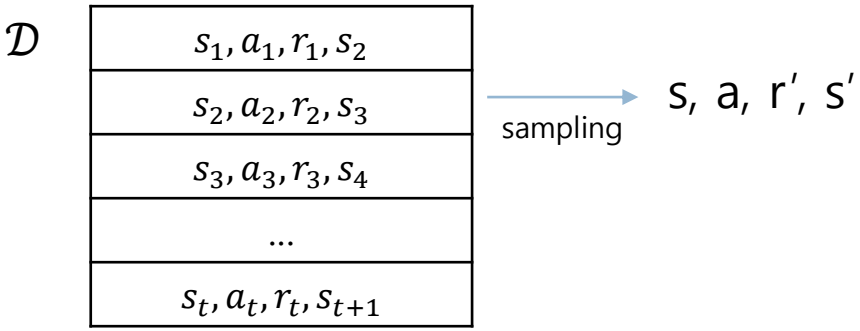
DQN: contributions

- **Deep Q-learning (DQN) addresses both of these challenges by**
  - **1) Experience Replay**
    - It stores several transition data (s, a, r, s′) in a buffer and trains a model via uniform random sampling
    - It can solve the correlation problem
  - **2) Fixed Q-Targets**
    - It uses an additional neural network to learn the target value and update the target network for Q function
    - It can solve the target oscillation problem

Experience Replay

Definition: Experience Replay

- **에이전트가 환경과 상호작용하면서 얻은 경험을 메모리에 저장해두고, 나중에 무작위로 샘플링하여 학습하는 기법**



- **To perform experience replay, repeat the following:**
  - (s, a, r, s′) ~ $\mathcal{D}$: sample an experience tuple from the dataset
  - Compute the Q-learning loss from the samples

$$\mathrm{L}(\theta) = \left[\left(R + \gamma \max_{a'} Q(s', a'; \theta^{-})\right) - Q(s, a; \theta)\right]^2$$

Fixed Q-Targets

Definition: Fixed Q-Targets

- **예측 네트워크 Q(s,a;θ)가 바뀌면 목표 값 $R+\gamma max Q(s,a;\theta)$ 도 바뀌면서 불안정한 학습이 되는 현상을 방지하기 위해 별도의 목표 네트워크$Q(s,a;\theta^{-})$를 두고 즉시 업데이트가 아닌 일정 주기마다 업데이트 시키는 학습방법**

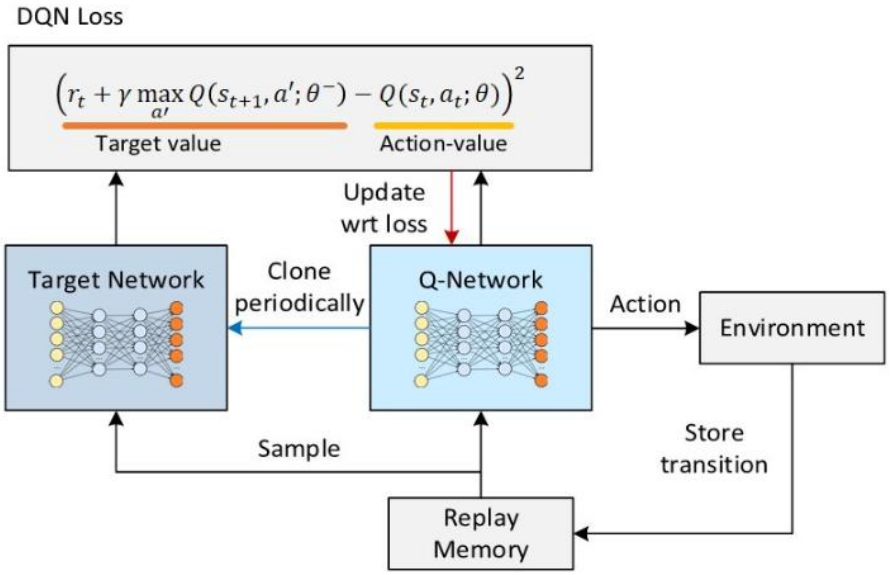
- Let parameters $\boldsymbol{\theta}^{-}$ be the set of weights used in the target, and $\boldsymbol{\theta}$ be the weights that are being updated
- Slight change to computation of target value:
  - (s, a, r, s′) ~ $\mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value by the target network $\boldsymbol{\theta}^{-}$: $r+\gamma max_{a'}\hat{Q}(s',a';\boldsymbol{\theta}^{-})$
  - Compute the action-value $\hat{Q}(s,a;\boldsymbol{\theta})$ by main Q-network $\boldsymbol{\theta}$

Target network w^{-} Main Q-network w

$$L(\theta)=\left(\underbrace{R+\gamma\max_{a'}Q(s',a';w^{-})}_{\text{Target value}}-\underbrace{Q(s,a;w)}_{\text{Action-value}}\right)^{2}$$

Training DQN

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{\theta}^-$
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent



Gymnasium

- **Introduction**
  - OpenAI Gym is an open-source platform developed by OpenAI that provides a collection of environments for training and testing reinforcement learning agents.
  - It's a toolkit designed to help researchers and developers in the field of machine learning and artificial intelligence build and experiment with reinforcement learning algorithms.
  - You can visit the official website:
    - https://gymnasium.farama.org/index.html

Gymnasium

- **Installation**
  - Type the command on your prompt:

```
> pip install gymnasium
```

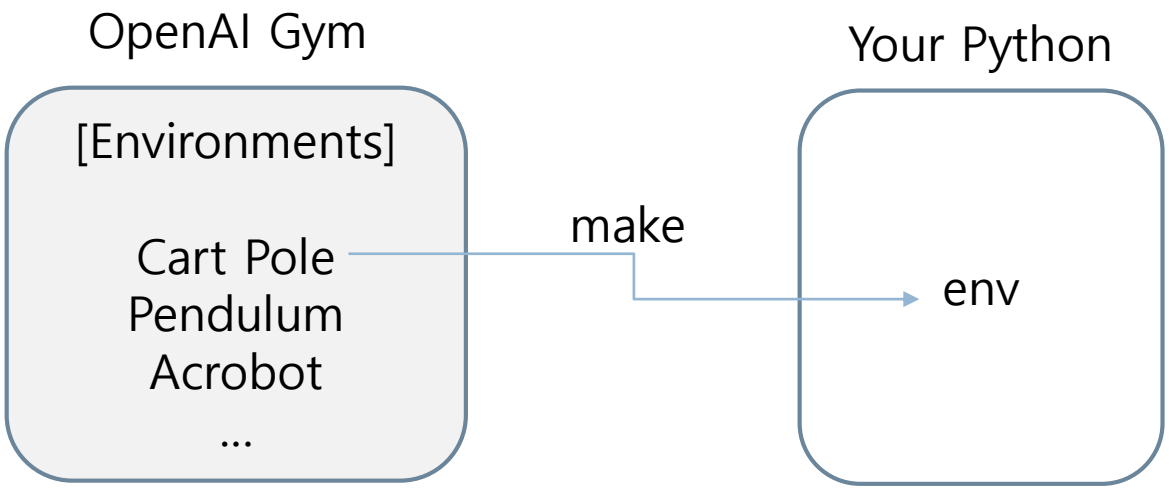
- **4 key functions**
  - make, reset, step and render

Gymnasium

- **1) Make**
  - **Load an environment**

```
import gymnasium as gym
env = gym.make('CartPole-v1')
```

-> Env. Name



Gymnasium

- **2) Reset**
  - **Initialize the environment**
    - **It provides an initial observation (state)**

```
observation, info = env.reset()
```

    - **Example**
      - In Cart Pole example, the observation includes [Cart position, Velocity, Pole angle, Pole angular velocity]
      - info is additional information of environment (if there is no information, it will be Null value)

Gymnasium

- **3) Step**
  - **Interaction between an agent and environment**

```
observation, reward, terminated, truncated, info = env.step(action)
```

Value	Description
Action	(ActType) An action value decided by the agent
Observation	(Object) this will be an element of the environment's
Reward	(Float) The reward returned as a result of taking the action
Terminated	(Bool) End of episode (True) or False
Truncated	(Bool) End of episode by unexpected error (True) or False
Info	(dictionary) Additional information (Null in the typical case)

Gymnasium

- **4) Render (mode)**
  - **Compute the render frames as specified by render_mode**

Option	Description
None (default)	no render is computed
Human	Visualization mode (render return None)
rgb_array	return a single frame representing the current state of the environment

Gymnasium

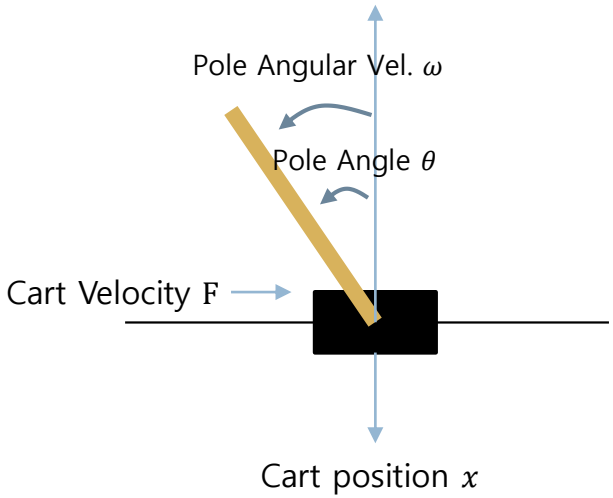
- **4) Render (mode)**
  - **Example (human model)**

```
import gymnasium as gym
import time

env = gym.make("CartPole-v1", render_mode='human')
state, info = env.reset()

for i in range(300):
    action = 1
    next_state, reward, terminated, truncated, info = env.step(action)

    # Render the env
    env.render()
    time.sleep(0.01)
    state = next_state

    if terminated:
        state, info = env.reset()

env.close()
```

Cart Pole Example

- **Cart Pole Example**
  - https://gymnasium.farama.org/environments/classic_control/cart_pole/
- **Action space:**
  - 0: Push cart to the left
  - 1: Push cart to the right
- **Observation Space**
  - it returns 4 values with shape (4,)

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf



- **Rewards**
  - a reward of +1 for every step taken

Cart Pole Example

- **1. 라이브러리 불러오기**

```
import gymnasium as gym
import numpy as np
import random
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
```

- **2. 하이퍼파리미터 설정**

```
env = gym.make("CartPole-v1")

STATE_SIZE = env.observation_space.shape[0]
ACTION_SIZE = env.action_space.n

# 하이퍼파라미터
LR = 1e-3           # learning rate
GAMMA = 0.99        # discount factor
EPSILON = 1.0       # initial exploration rate
EPSILON_MIN = 0.05  # minimum epsilon
EPSILON_DECAY = 0.995
BATCH_SIZE = 64
MEMORY_SIZE = 10000
TARGET_UPDATE = 100  # target network 갱신 주기
NUM_EPISODES = 300

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('device:{}'.format(device))
```

Cart Pole Example

- **3. Q-Network 정의**

```
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

    def sample_action(self, state, eps):
        if random.random() < eps:
            return random.randint(0,1)
        else:
            state = torch.FloatTensor(state).to(device)
            out = self.forward(state)
            return out.argmax().item()
```

Cart Pole Example

- **4. Replay Buffer**

```
class ReplayBuffer:
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)

    def store(self, state, action, reward, next_state, done):
        self.buffer.append((state, action, reward, next_state, done))

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        states, actions, rewards, next_states, dones = zip(*batch)
        return (np.vstack(states),
                np.array(actions),
                np.array(rewards),
                np.vstack(next_states),
                np.array(dones))

    def size(self):
        return len(self.buffer)
```

Cart Pole Example

- **5. 학습용 함수 정의**

```
# Q 네트워크 두 개 정의: online / target
q_net = QNetwork(STATE_SIZE, ACTION_SIZE).to(device)
target_net = QNetwork(STATE_SIZE, ACTION_SIZE).to(device)
target_net.load_state_dict(q_net.state_dict())  # 동일하게 초기화


optimizer = optim.Adam(q_net.parameters(), lr=LR)
memory = ReplayBuffer(MEMORY_SIZE)
```

Cart Pole Example

- **6. 학습 함수**

```
def train_step():
    if memory.size() < BATCH_SIZE:
        return

    states, actions, rewards, next_states, dones = memory.sample(BATCH_SIZE)

    states = torch.FloatTensor(states).to(device)
    actions = torch.LongTensor(actions).unsqueeze(1).to(device)
    rewards = torch.FloatTensor(rewards).unsqueeze(1).to(device)
    next_states = torch.FloatTensor(next_states).to(device)
    dones = torch.FloatTensor(dones).unsqueeze(1).to(device)

    # 1 Q(s,a) 예측값
    q_values = q_net(states).gather(1, actions)

    # 2 target 값 계산 (Fixed Q-Target)
    with torch.no_grad():
        next_q_values = target_net(next_states).max(1, keepdim=True)[0]
        target = rewards + (1 - dones) * GAMMA * next_q_values

    # 3 손실 계산
    loss = nn.MSELoss()(q_values, target)

    # 4 경사하강법으로 업데이트
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Cart Pole Example

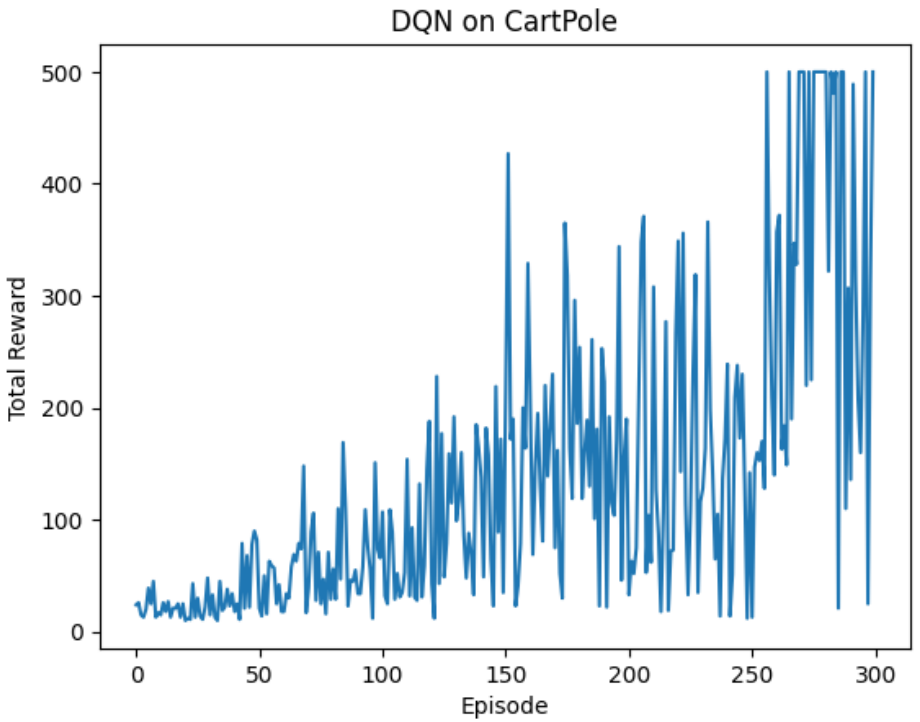
- **7. 학습 루프**

```
returns = []

for episode in range(NUM_EPISODES):
    state, _ = env.reset()
    total_reward = 0

    for t in range(500):
        action = q_net.sample_action(state, EPSILON)
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated  # 에피소드 종료 여부 통합

        memory.store(state, action, reward, next_state, done)   # 경험 저장

        state = next_state
        total_reward += reward

        train_step()  # 한 스텝 학습

        # 타깃 네트워크 동기화
        if t % TARGET_UPDATE == 0:
            target_net.load_state_dict(q_net.state_dict())

        if done:
            break

    EPSILON = max(EPSILON_MIN, EPSILON * EPSILON_DECAY)
    returns.append(total_reward)
    print(f"Episode {episode+1} | Reward: {total_reward:.1f} | Epsilon: {EPSILON:.3f}")
```

Cart Pole Example

- **8. 학습결과 시각화**

```
import matplotlib.pyplot as plt


plt.plot(returns)
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.title("DQN on CartPole")
plt.show()
```

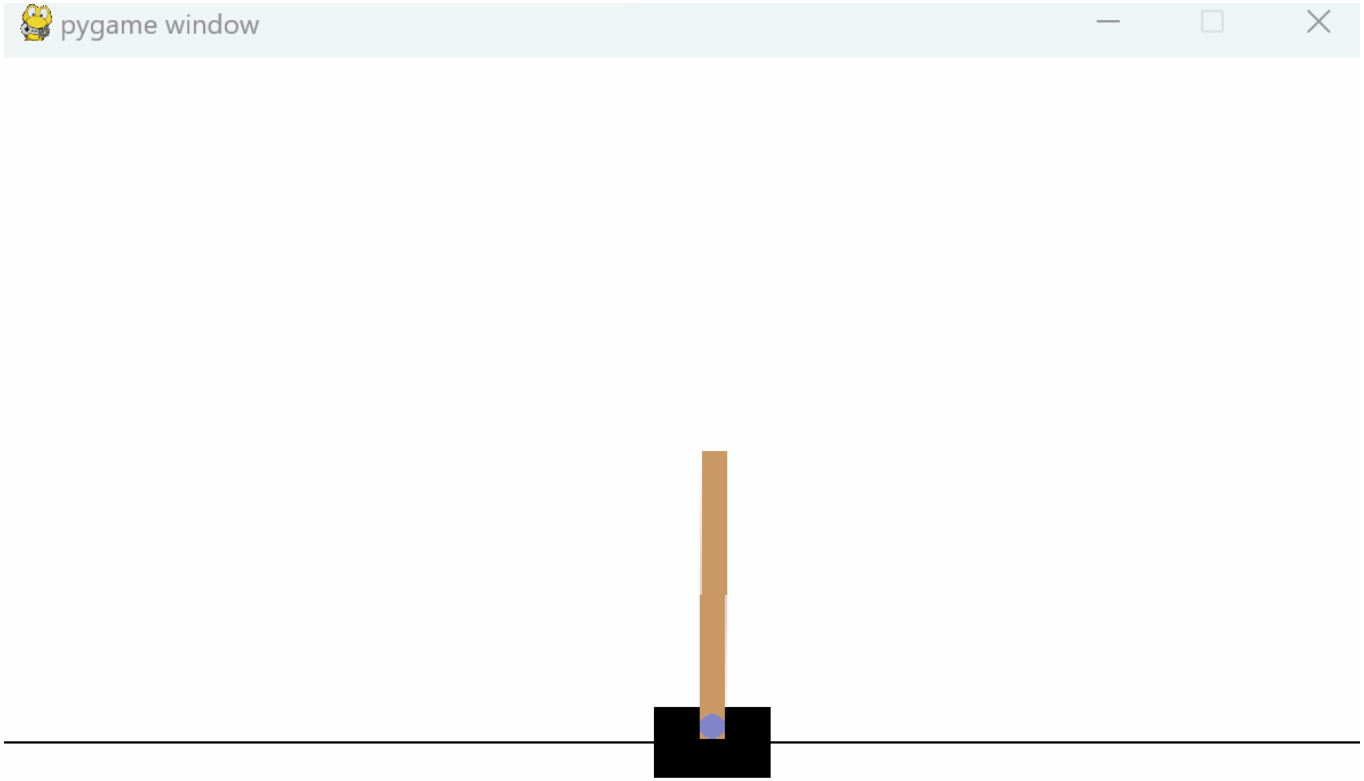


Cart Pole Example

- **9. 시뮬레이션**

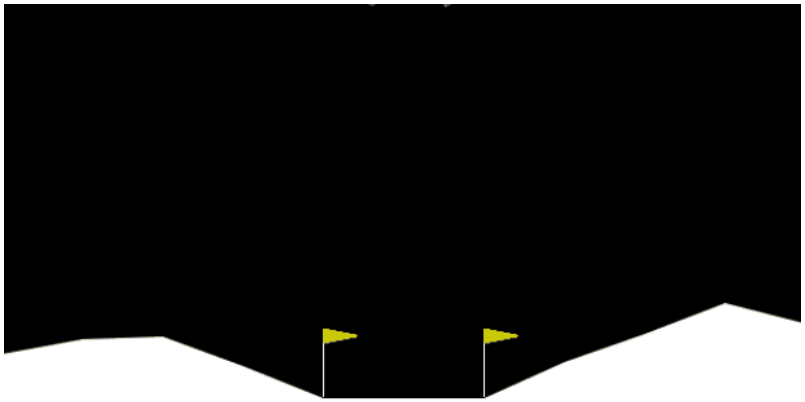
```
import time
env = gym.make("CartPole-v1", render_mode='human')
state, info = env.reset()


for i in range(300):
    action = q_net.sample_action(state, 0)
    next_state, reward, terminated, truncated, info = env.step(action)
    env.render()


    time.sleep(0.01)


    state = next_state
    if terminated:
        state, info = env.reset()
env.close()
```

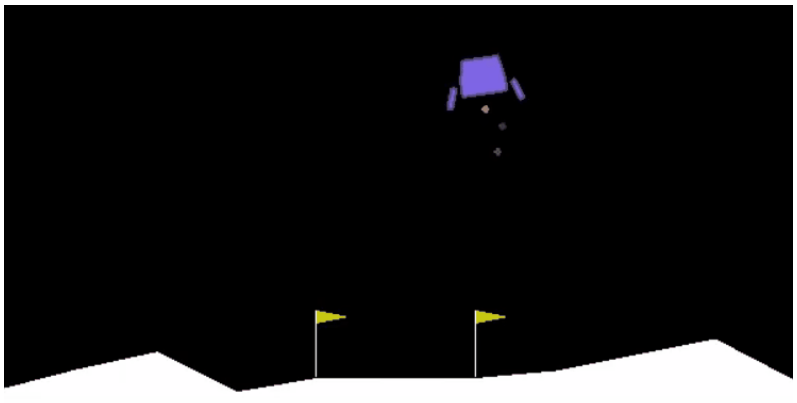
Demo



Assignment

- **Lunar Lander**
  - **Build DQN to solve the Lunar Lander problem**
  - **Environment**
    - **https://gymnasium.farama.org/environments/box2d/lunar_lander/**





Q & A