

인공지능프로그래밍

Lab7

Tabular Q Learning

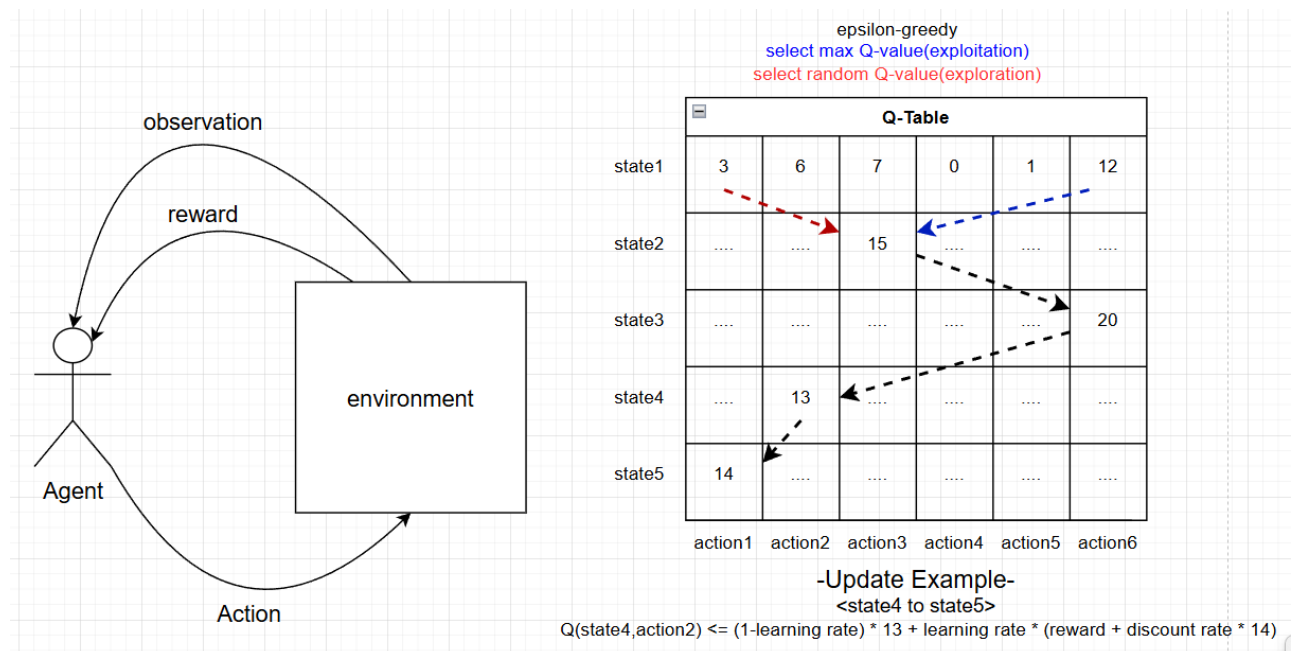
2024/11/25

2019202050 이강현

Lab Objective

Tabular Q-learning을 직접 구현해보며 off-policy의 TD-control이 어떻게 동작하는지 이해한다. Behavior policy와 target policy의 차이를 이해하고 importance sampling을 사용하지 않아도 되는 이유를 이해한다. 다양한 environment에서 Q-learning이 적용되는 것을 확인하고 그 유용함과 한계점을 확인할 수 있다.

Program Flow



강화학습의 흐름과 더불어 Q-learning 핵심을 Q-table을 통해 표현하였다. 왼쪽은 environment와 상호작용하는 Agent의 모습을 그린 것이고 action을 취하면 reward와 observation을 받게 되는 것을 확인한다. 오른쪽은 state와 action에 따른 Q value와 매 episode마다 table이 어떻게 update되는지 확인할 수 있다.

Tabular Q-learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$


Tabular Q-learning에 대한 내용을 Program flow의 Q-table을 이용하여 설명한다. 먼저 state1부터 state 5로의 순서로 state가 고정적으로 변한다고 가정한다. state 1에서 파란색 점선에 해당하는 부분은 epsilon-greedy 중 exploitation에 해당하여 큰 값을 사용하는 것을 표현하였고 빨간색 점선에 해당하는 부분은 exploration에 해당하여 랜덤한 값을 사용하고 그중 3을 택한 것을 표현한 것이다. 두 방법은 behavior policy에 대한 내용이고 target policy는 behavior policy로 인해 선택된 다음 state인 state2에서 가장 큰 Q-value인 15를 공통적으로 선택하게 된다. 이때는 greedy 방식을 채택하기 때문이다. 이후 state 2,3,4,5에서는 epsilon-greedy에서 max값만 채택했다고 가정하면 그림처럼 policy를 선택하며 내려오게 된다. Table 아래는 Q-value를 어떻게 update하는지에 대한 수식을 state4에서 state5로 가는 경우를 예시로 설명하였다. 위처럼 off-policy TD control algorithm 중 같은 분포에서 뽑아낸 policy라는 측면에서 Q-learning은 importance sampling이 필요하지 않게 된다. 또 table을 통해 모든 state, action에 따른 value를 계산, 기록하므로 Tabular Q-learning이라고 한다.

Result


```
✓ 5초 # Check if this code runs in Colab
RunningInCOLAB = 'google.colab' in str(get_ipython())
# if in colab, Use reinforcement learning library gymnasium and tqdm
if RunningInCOLAB:
    !pip install gymnasium
    from tqdm.notebook import tqdm
else:
    from tqdm import tqdm
```

Collecting gymnasium
Downloading gymnasium-1.0.0-py3-none-any.whl.metadata (9.5 kB)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (1.26.4)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (3.1.0)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (4.12.2)
Collecting farama-notifications>=0.0.1 (from gymnasium)
Downloading Farama_Notifications-0.0.4-py3-none-any.whl.metadata (558 bytes)
Downloading gymnasium-1.0.0-py3-none-any.whl (958 kB)
958.1/958.1 kB 11.7 MB/s eta 0:00:00
Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Installing collected packages: farama-notifications, gymnasium
Successfully installed farama-notifications-0.0.4 gymnasium-1.0.0

먼저 필요한 라이브러리를 import하는 부분이다. colab에서 사용하는 경우를 고려하여 tqdm을 notebook용으로 머신러닝 툴을 이용하기 위한 gymnasium 라이브러리를 가져오는 것을 확인한다.


```
✓ 1초  #import numpy, matplotlib, gymnasium
import numpy as np
import matplotlib.pyplot as plt

import gymnasium as gym # name gymnasium as gym
from gymnasium import wrappers # for making video
```

```
✓ 1초  gym.__version__ #gymnasium library version

→ '1.0.0'
```


Q-table과 랜덤 변수 생성을 위한 numpy와 plot을 위한 matplotlib, Q-learning environment를 제공하는 gymnasium, 결과를 영상으로 볼 수 있게 하기 위한 wrappers 모듈까지 import한다.

```
✓ 0초 [4]  # select environment
# Text Game: 0 for FrozenLake 4x4, 1 for Taxi, 2 for Blackjack

SELECT_ENV = 0
```

```
✓ 0초  # select environment
# Text Game: 0 for FrozenLake 4x4, 1 for Taxi, 2 for Blackjack

SELECT_ENV = 1
```

```
 # select environment
# Text Game: 0 for FrozenLake 4x4, 1 for Taxi, 2 for Blackjack

SELECT_ENV = 2
```

위처럼 environment를 선택하여 아래 코드를 실행한다. 주석을 통해 알 수 있듯이 이번 실습에서는 0일 때 frozenlake, 1일 때 taxi, 2일 때 blackjack

environment를 사용할 예정이다.

```
0.5 [6] #classification environments
if SELECT_ENV == 0: # Case FrozenLake
    env_name, res_prefix = 'FrozenLake-v1', 'lak4' # set environment and save prefix
    #set hyperparameters in FrozenLake learning
    max_episodes, max_ep_steps, goal_score = 3000, 500, 0.8
    val_ep_num, lr_rate, discount_rate = 20, 0.8, 0.95
elif SELECT_ENV == 1: # Case Taxi
    env_name, res_prefix = 'Taxi-v3', 'taxi' # set environment and save prefix
    #set hyperparameters in Taxi learning
    max_episodes, max_ep_steps, goal_score = 1000, 500, 8.0
    val_ep_num, lr_rate, discount_rate = 20, 0.8, 0.95
elif SELECT_ENV == 2: # Case Blackjack
    env_name, res_prefix = 'Blackjack-v1', 'blkj' # set environment and save prefix
    #set hyperparameters in Blackjack learning
    max_episodes, max_ep_steps, goal_score = 5000, 10, -0.05
    val_ep_num, lr_rate, discount_rate = 1000, 0.1, 0.95
else: assert False, 'environment selection error' # If it is not a case of 0, 1, 2
# create environment
def create_env():
    if SELECT_ENV == 0: # Forzen Lake
        env = gym.make(env_name, desc=None, map_name='4x4', is_slippery=True, render_mode='rgb_array') #make FrozenLake environment
    elif SELECT_ENV == 1: # Taxi
        env = gym.make(env_name, render_mode='rgb_array') #make Taxi environment
    elif SELECT_ENV == 2: # Blackjack
        env = gym.make(env_name, natural=False, sab=False, render_mode='rgb_array') #make Blackjack environment
    else: pass
    return env
```

위는 각각의 enviroment에 따른 파라미터를 조정하는 것이다. Environment에 따라 적절한 수렴을 위한 변수가 다양하기에 이처럼 설정해준다. Create_env 함수는 각 environment를 gymnasium 라이브러리를 통해 만드는 함수이다. Frozen lake는 미끄러짐 정도, 맵의 크기등을 조절할 수 있고 blackjack은 21로 시작할 때 추가보상을 줄지 여부등 environment를 만들 때 인자를 통해 다양하게 커스텀할 수 있음을 확인한다.

```
0.5 [6] #define environment function
def env_reset(env): # initialize observation
    observation = env.reset() #environment reset
    obs = observation[0] if type(observation)==tuple else observation # observation space is tuple or one value
    if SELECT_ENV == 2: # if blackjack
        state = obs[0] + 32 * obs[2] + 64 * obs[1] #observation space has 3-tuple, player's current sum, dealer's one showing card, whether player holds usable ace
    else: state = obs # else just select one value
    return state

def env_step(env, action):
    observation = env.step(action) # Observation according to action, order -> observation, reward, terminate, truncated, info
    if SELECT_ENV == 2:
        state = observation[0][0] + 32 * observation[0][2] + 64 * observation[0][1] #blackjack state has 3-tuple
    else: state = observation[0] # reward
    reward = observation[1] # reward
    done = observation[2] or observation[3] if len(observation)>4 else observation[2] # Generally, only the terminate argument is given, but consider the case where there is an additional argument indicating the termination status.
    return state, reward, done

0.5 [7] # Create variables based on defined functions
env = create_env()
state = env_reset(env)
state, reward, done = env_step(env, env.action_space.sample())
```

Env_reset함수는 초기 state를 반환하는 함수이다. Frozen_lake와 taxi는 state가 단일 값으로 존재하기에 observation[0]으로 받고 blackjack은 state가 3개의 int값으로 이루어진 tuple형태로 존재하기 때문에 다르게 받는 것을 확인한다.

Env_step함수는 학습시 매 단계마다 state, reward, done을 반환받기 위해 정의되

는 함수이다. 초기에는 reward가 없지만 추후 state에 따른 action을 하게 되면 얻게되는 reward를 Q-value를 update할 때 사용해야 하므로 함수화하는 것을 확인한다. 또한 done을 통해 종료 조건이 성립되었는지를 확인하여 반복을 종료하는데 사용한다.

```

0초
▶ # Data processing according to action space
action_shape = env.action_space.shape
action_space_type = type(env.action_space)

if action_space_type==gym.spaces.discrete.Discrete: # FrozenLake, Taxi, Blackjack
    actn_space = 'DISCRETE'
    action_shape = (1,)
    action_dims = 1
    action_range = env.action_space.n
    num_actions = action_range # number of actions is action range for DISCRETE actions
    action_batch_shape = (None, action_range)
elif action_space_type==gym.spaces.box.Box: # Additionally, for action space in box2d format.
    actn_space = 'CONTINUOUS'
    action_dims = action_shape[0]
    actn_uppr_bound = env.action_space.high[0]
    actn_lowr_bound = env.action_space.low[0]
    action_range = (actn_uppr_bound - actn_lowr_bound) # x0.5 for tanh output
    action_batch_shape = tuple([None]+[x for x in action_shape])
    num_actions = action_dims # number of actions is action dimension for CONTINUOUS actions
else: assert False, 'other action space type are not supported'

# Data processing according to observation space
observation_space_type = type(env.observation_space)
observation_shape = env.observation_space.shape

if observation_space_type==gym.spaces.discrete.Discrete: # FrozenLake, Taxi
    observation_shape = (1,)
    num_states = env.observation_space.n
elif observation_space_type==gym.spaces.box.Box: # Additionally, for observation space in box2d format.
    num_states = observation_shape[0]
elif observation_space_type==gym.spaces.tuple.Tuple: #Blackjack
    observation_shape = tuple([x.n for x in env.observation_space])
    num_states = np.prod(observation_shape) #product
else: print('observation space type error')

if SELECT_ENW == 2: # blackjack state has just one value, player's current sum
    state_shape = (1,)
    state_batch_shape = (None,1)
else:
    state_shape = observation_shape
    state_batch_shape = tuple([None]+[x for x in observation_shape])

value_shape = (1,)
num_values = 1

```

위는 action과 observation에 대한 값들을 확인하는 부분이다. Action은 이번 실습에 주어진 environment에서는 모두 DISCRETE로 분류되어 1차원 값들의 모음으로 처리할 수 있으나 2d-box 형태의 CONTINUOUS한 다른 environment의 경우에는 다르게 처리해야 하기에 존재하는 코드이다. Observation은 단일 값으로 확인가능

한 경우에는 Discrete하게 처리하고 blackjack과 같이 세개의 int값이 tuple로 존재하고 이들의 값에 따라 state가 정해지는 경우에는 다르게 처리해야 한다.

아래는 각 environment에 따른 변수들을 출력한 모습이다.

✓
0초

```
# Description of current environment
print('Action space ', action_space_type)
print('Action shape ', action_shape)
print('Action dimensions ', action_dims)
print('Action range ', action_range)
if action_space_type==gym.spaces.box.Box:
    print('Max Value of Action ', actn_uppr_bound)
    print('Min Value of Action ', actn_lowr_bound)
else: pass
print('Action batch shape ', action_batch_shape)

print('Observation space ', observation_space_type)
print('Observation shape ', observation_shape)
print('Size of State Space ', num_states)
print('State shape ', state_shape)
print('State batch shape ', state_batch_shape)

print('Vallue shape ', value_shape)
print('Value dimensions ', num_values)
```

```
⇒ Action space <class 'gymnasium.spaces.discrete.Discrete'>
Action shape (1,)
Action dimensions 1
Action range 4
Action batch shape (None, 4)
Observation space <class 'gymnasium.spaces.discrete.Discrete'>
Observation shape (1,)
Size of State Space 16
State shape (1,)
State batch shape (None, 1)
Vallue shape (1,)
Value dimensions 1
```

위는 FrozenLake에 사용되는 변수들의 모습인데 핵심적인 부분을 살펴보면 action이 상하좌우로 이동하는 경우이기 때문에 4개 있는 것을 확인한다. 또 observation은 4x4 크기의 map을 이동하는 형태이기 때문에 16으로 정해진다.

```
Action space <class 'gymnasium.spaces.discrete.Discrete'>
Action shape (1,)
Action dimensions 1
Action range 6
Action batch shape (None, 6)
Observation space <class 'gymnasium.spaces.discrete.Discrete'>
Observation shape (1,)
Size of State Space 500
State shape (1,)
State batch shape (None, 1)
Value shape (1,)
Value dimensions 1
```

위는 taxi의 경우이다. Action은 승객 픽업,하차 두 가지와 상하좌우 이동 4가지를 합쳐 총 6개의 action을 가진다. State는 목적지 4개 x 승객이 있는 곳(목적지 4개 + 택시 안) 5개 x 25개의 택시 위치로 총 500개의 state를 가질 수 있기에 500의 크기를 가진다.

```
Action space <class 'gymnasium.spaces.discrete.Discrete'>
Action shape (1,)
Action dimensions 1
Action range 2
Action batch shape (None, 2)
Observation space <class 'gymnasium.spaces.tuple.Tuple'>
Observation shape (32, 11, 2)
Size of State Space 704
State shape (1,)
State batch shape (None, 1)
Value shape (1,)
Value dimensions 1
```

위는 blackjack의 경우이다. Action은 21이라는 값에 더 가까워지기 위해 카드를 더 뽑는 경우와 멈추는 경우로 두 가지를 가진다.

State 중 플레이어의 현재 합은 21보다 큰 경우 bust를 하지만 10 10을 뽑고 에이스를 뽑는 경우로 31까지 고려하여 0부터 31까지로 32로 설정되어 있다.

딜러의 값은 1부터 10까지로 0부터 10까지 11로 설정되어 있다.

에이스의 유무는 0과 1까지 2로 설정되어 있다. 카드의 값은 최소가 1부터 시작하므로 0은 빼야하지만 디버깅이나 다양한 목적으로 범위가 설정된 것으로 예상

한다.

```
✓ 0초 ▶ #define q-table for current environment  
      ### START CODE HERE ###  
  
      Q_table = np.zeros((num_states, action_range)) # define q-table  
  
      ### END CODE HERE ###  
  
      print(Q_table.shape)
```

⇒ (16, 4)

(500, 6)

(704, 2)

순서대로 frozenlake, taxi, blackjack의 Q-table의 shape을 출력한 것이다.

(state,action)의 형태로 존재한다.

```
✓ 0초 [11] #hyperparameter for train  
      total_episodes = max_episodes # Total episodes  
      max_steps = max_ep_steps # Max steps per episode  
      learning_rate = lr_rate # Learning rate  
      gamma = discount_rate # Discounting rate  
      val_episodes = val_ep_num # number of validation episodes
```

Environment를 설정하는 부분에서 정의해둔 하이퍼파라미터를 명시하는 부분이다.

```

0초
▶ # Exploration parameters for epsilon greedy strategy
class Epsilon:
    def __init__(self, max_steps):
        self.explore_start = 1.0          # exploration probability at start
        self.explore_stop = 0.01         # minimum exploration probability
        self.decay_rate = 20.0/max_steps  # exponential decay rate for exploration prob (4.6 ~ max_step = 0.01)
        self.steps = 0

    def get_epsilon(self):
        eps = (self.explore_stop
              + (self.explore_start - self.explore_stop) * np.exp(-self.decay_rate * self.steps))
        self.steps += 1
        return eps
    # The epsilon value gradually decreases over time, reducing the exploration rate.

```

위는 step에 따라 달라지는 epsilon값을 얻어내기 위한 함수이다. 처음에는 최적화되어 있지 않다는 점을 고려하여 높은 epsilon 값으로 랜덤한 policy가 자주 선택되게 하며 이후에는 점점 epsilon값을 낮추어 최적화된 policy를 사용하게끔 하는 접근 방식을 택하고 있다.

```

0초
▶ #For policy evaluate
def evaluate_policy(env, qtable, num_average, images=None):

    total_reward = 0.0
    total_steps = 0
    episodes_to_play = num_average # Works as well as validation episode
    for _ in range(episodes_to_play): # Play n episode and take the average
        state = env.reset(env)
        if images!=None: images.append(env.render()) #if image is not None, appending to list
        done = False
        episode_reward = 0.0
        while not done:

            ### START CODE HERE ###

            action = np.argmax(qtable[state,:]) # get an action from q-table
            next_state, reward, done = env_step(env, action) # take action and observe outcomes

            ### END CODE HERE ###

            if images!=None: images.append(env.render())
            state = next_state # update state
            episode_reward += reward
            total_steps += 1
        total_reward += episode_reward
    average_reward = total_reward / episodes_to_play # take the average reward of evaluations
    average_steps = total_steps / episodes_to_play # take the average steps of evaluations
    #calculate reward
    return average_reward, average_steps, images

```

위는 policy를 검증하는 함수이다. 훈련 이후 사용될 것이므로 최적화된 qtable을 입력인자로 받는 것을 확인한다. Action은 state에 따른 action을 qtable을 근거로 선택해 사용하며 이로 인해 얻어지는 observation을 다시 사용하는 것으로 매

step이 구성된다. done flag가 활성화될때까지 반복되며 reward와 step을 누적하여 반환하는 것을 확인할 수 있다.

```

#training
history = {'rewards': []} # logs of rewards
epsF = Epsilon(total_episodes) # generate epsilon object

pbar = tqdm(range(total_episodes), bar_format='{l_bar}{bar:10}{r_bar}{bar:-10b}') # tqdm initializing

for episodes in pbar:

    # initialize variables for a new episode
    epis_rewards = 0 # episode reward
    epis_steps = 0 # steps for an episode
    done = False # episode end flag

    state = env.reset(env) # Reset the environment

    epsilon = epsF.get_epsilon() # get new epsilon value

    while not done: # simulate to the terminal state of the episode

        ### START CODE HERE ###

        random_number = np.random.uniform(0,1) # get a random number for exploration decision
        if random_number > epsilon: # If this number > greater than epsilon --> exploitation
            action = np.argmax(Q_table[state,:]) # find an index of the biggest Q value for this state
        else: # Else doing a random choice --> exploration
            action = np.random.randint(0,action_range) # get a random integer in [0,action_range)
        next_state, reward, done = env_step(env,action) # take the action to environment and observe outcomes
        Q_table[state, action] = (1-learning_rate) * Q_table[state, action] + learning_rate * (reward + gamma + np.max(Q_table[next_state,:])) # update q-table

        ### END CODE HERE ###

        epis_rewards += reward # accumulate rewards to calculate the episode reward
        state = next_state # set state for the next state

        if epis_steps > max_steps: break # if run too much : finish episode
        else: epis_steps += 1

    # evaluate the policy
    eval_reward, eval_steps, _ = evaluate_policy(env, Q_table, val_episodes) # evaluate the policy

    history['rewards'].append(eval_reward) #save reward tendency for plotting

    pbar.set_postfix({'reward':eval_reward, 'steps':eval_steps})

    if eval_reward>goal_score: break # exit if convergence

    print('episodes:{0:5d}, final_reward {1:4.2f}'.format(episodes, eval_reward))

```

위는 정의한 함수들을 적절히 사용하여 훈련과 검증을 진행하는 부분이다. 핵심적인 부분을 살펴보면 위에서 정의한 epsilon함수를 이용하여 epsilon-greedy를 구현하는 모습이다. 랜덤한 값이 epsilon보다 작으면 랜덤한 action을 선택하여 진행하고 그렇지 않으면 현재 state에서 가장 Q-value가 큰 action을 선택하는 것을 확인한다. 그렇게 얻어낸 action이 behavior policy가 되고 이를 통해 다음 state,reward를 얻어낸다. 다음 action은 Q-value가 가장 큰 것을 택할 것으로 greedy하게 결정하고 이것이 target policy가 된다. 이 두 가지 Q-value를 이용하여 현재의 Q값을 update하는 것을 확인할 수 있다. 이와 같은 방식을 매 step 반복하여 최적의 Q-table을 얻어낸다. 아래는 훈련과정을 보여준다.


 11% | 333/3000 [00:10<01:40, 26.43it/s, reward=0.85, steps=39.2]
 episodes: 333, final_reward 0.85



45% | 447/1000 [00:43<00:22, 24.76it/s, reward=8.5, steps=12.5]
 episodes: 447, final_reward 8.50



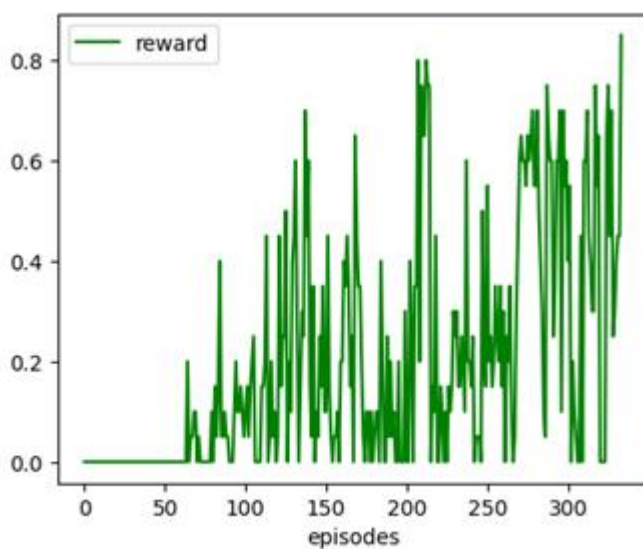
23% | 1161/5000 [03:11<09:02, 7.07it/s, reward=-0.048, steps=1.56]
 episodes: 1161, final_reward -0.05

위에서부터 순서대로 frozenlake, taxi, blackjack이다. 랜덤한 경우가 존재하기에 수렴을 위한 episode 수는 매 훈련마다 다르지만 위에서 설정한 각각의 goal_score(frozenlake = 0.8, taxi = 8.0, blackjack = -0.05)에 도달하면 멈추는 것을 확인할 수 있다.

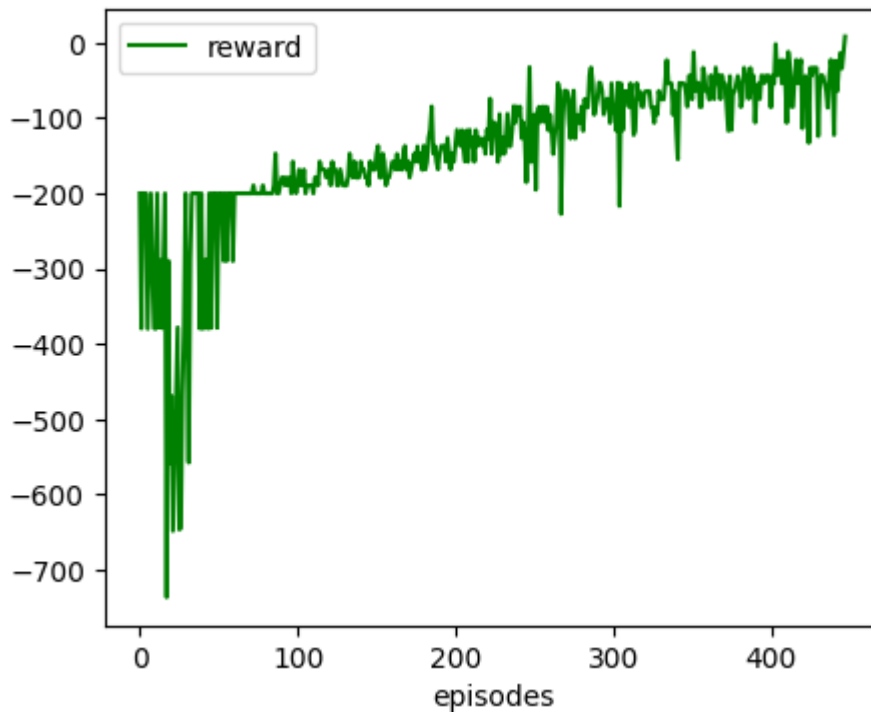
아래는 위에서부터 frozenlake, taxi, blackjack의 reward 그래프이다. 매 step마다 저장해둔 reward를 이용하여 시각화한 모습이다.

```
# plot loss and accuracy
def plot_graphs(log_history, log_labels, graph_labels, graph_colors=['b-', 'g-']):
    num_graphs = len(log_labels)
    plt.figure(figsize=(5*num_graphs, 4))
    for i in range(num_graphs):
        plt.subplot(1, num_graphs, i+1)
        plt.plot(log_history[log_labels[i]], graph_colors[i], label=graph_labels[i])
        plt.xlabel('episodes')
        plt.legend()
    plt.show()
    return

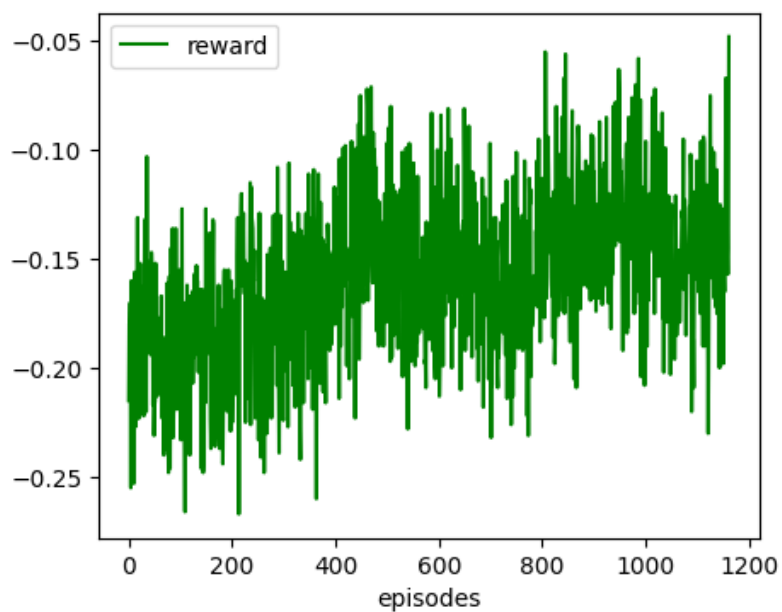
# Change in reward according to episode Plot to see the completeness of learning
log_labels = ['rewards']
label_strings = ['reward']
label_colors = ['g-']
plot_graphs(history, log_labels, label_strings, label_colors)
```



위는 frozenlake의 reward이다. 점차 목표치 reward에 가까워지고 있으나 다소 편차가 큰 것을 확인한다.



위는 taxi의 reward이다. 초기에는 매우 낮은 reward를 경험하나 어느 순간을 기점으로 방향성을 찾고 목표치로 도달하는 모습을 확인한다.



위는 blackjack의 reward이다. 처음부터 끝까지 비슷한 분산을 가지지만 전반적으

로 reward가 높아지는 그래프를 형성한다.

✓
0초

```
#agent evaluation
evaluate_episodes = 20
sum_episode_rewards = 0.0
pbar = tqdm(range(evaluate_episodes))

for i in pbar:
    rewards, _, _ = evaluate_policy(env, Q_table, 1) # just one episode
    sum_episode_rewards += rewards

env.close()

print('Evaluation Result:', sum_episode_rewards/evaluate_episodes)
```



100% 20/20 [00:00<00:00, 430.50it/s]

Evaluation Result: 0.6

100% 20/20 [00:00<00:00, 414.67it/s]

Evaluation Result: -32.6

100% 20/20 [00:00<00:00, 502.61it/s]

Evaluation Result: 0.05

위는 20번의 episode를 통해 검증과정을 거쳤을 때 평균 보상을 확인하는 부분이다.

✓
0초

```
[17] #Measuring reward through actions for each state in the optimized Q-table
env = create_env()

if SELECT_ENV != 2: #for Frozen Lake and Taxi
    env = wrappers.RecordVideo(env, video_folder='./gym-results/', name_prefix=res_prefix) #record video
    eval_reward, _, _ = evaluate_policy(env, Q_table, 1)
else: #for blackjack
    ims = []
    eval_reward, _, ims = evaluate_policy(env, Q_table, 1, ims) # save image

print('Sample Total Reward:', eval_reward)

env.close()
```

/usr/local/lib/python3.10/dist-packages/gymnasium/wrappers/rendering.py:283: UserWarning: WARN: Overwriting existing videos at /content/gym-results folder (try specifying a different 'video_folder' for the 'RecordVideo' wrapper if this is not desired)
logger.warn()
Sample Total Reward: 1.0

/usr/local/lib/python3.10/dist-packages/gymnasium/wrappers/rendering.py:283: UserWarning: WARN: Overwriting existing videos at /content/gym-results folder (try specifying a different 'video_folder' for the 'RecordVideo' wrapper if this is not desired)
logger.warn()
Sample Total Reward: 10.0

Sample Total Reward: -1.0

위는 최적화된 Q-table을 사용하여 영상 또는 이미지로 시각화 하기 위한 부분이다. 추가로 전체 reward도 함께 출력한다. 영상의 노란색 부분은 이미 영상이 존재하여 덮어씌워질 수 있다는 경고문구였다.

```
✓ 0초 ▶ #display agent video
from IPython.display import HTML, display
from base64 import b64encode

def show_video(video_path, video_width = 320):
    video_file = open(video_path, "r+b").read() #read saved video
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}" #encoding decoding
    return HTML(f"<video width={video_width} controls><source src='{video_url}'></video>") # create link

if SELECT_ENV==2: #create image
    for i in range(len(ims)):
        plt.figure(figsize=(3,3))
        plt.imshow(ims[i])
        plt.show()
    else:
        display(show_video('./gym-results/' + res_prefix + '-episode-0.mp4')) #show
```

위는 저장한 영상이나 이미지를 확인할 수 있도록 하는 코드이다.

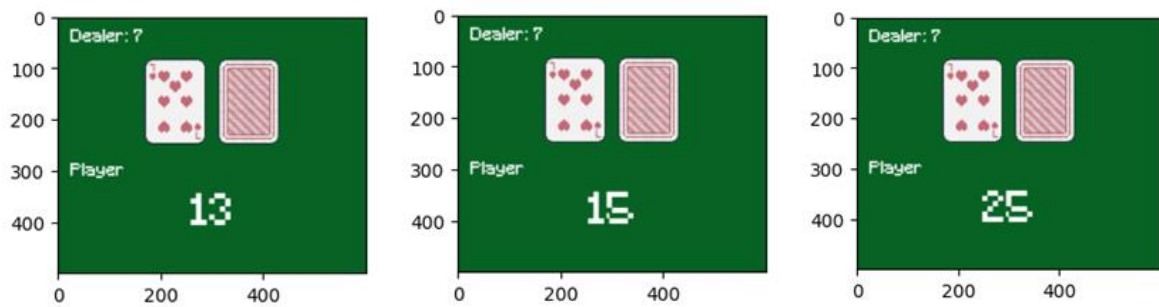


영상의 한 부분인 위 사진을 보면 frozenlake는 왔던 길을 다시 돌아가는 경우가 많았는데 결과적으로는 출발지로부터 웅덩이를 잘 피해 오른쪽 아래 선물 상자를 목표로 잘 도달하는 것을 확인할 수 있었다.



Taxi는 무작위 초기 상태에서 가장 빠른 길로 이동해 승객을 태우고 가장 빠른 길로 목적지로 향해 가는 것을 확인할 수 있었다.

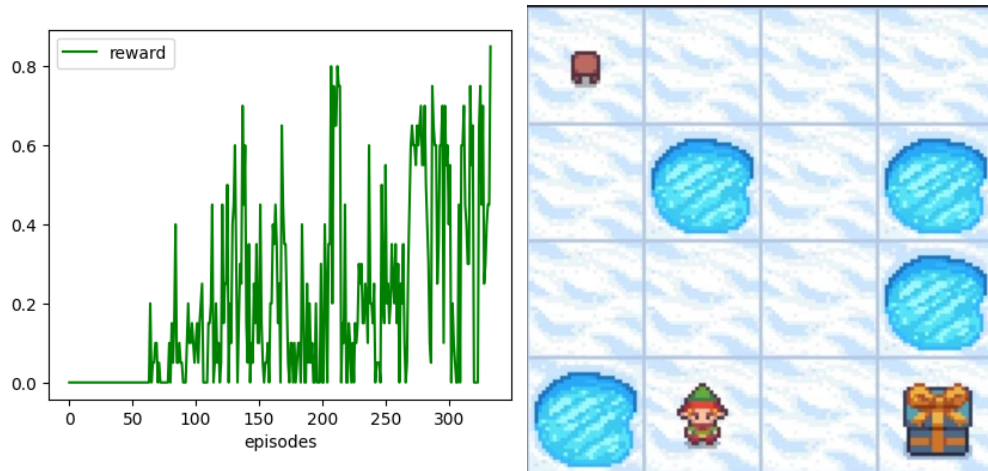
Blackjack은 영상이 아니라 사진으로 얻을 수 있었다. 따라서 아래와 같은 결과를 얻었다.



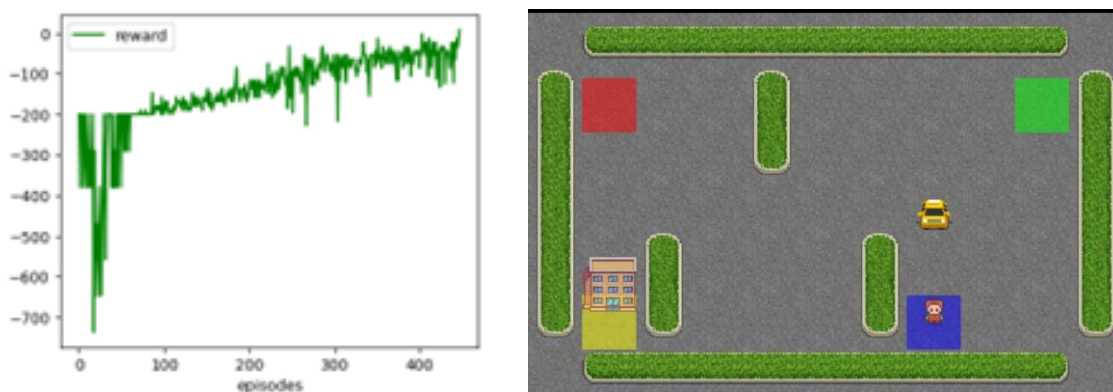
왼쪽부터 처음에 13이라는 값을 얻었으나 딜러의 7과 숨겨진 값을 더했을 때 13보다 클 것을 우려해 한번 더 진행했고 15라는 값을 얻었으나 더 적다고 판단해 한번 더 진행하는 것을 확인한다. 결과적으로는 25라는 값을 얻어 bust되어 패배하는 모습이다.

이후 discussion에서 영상과 이미지 그리고 3개의 environment에 대한 결과에 대해 자세히 논의한다.

Discussion

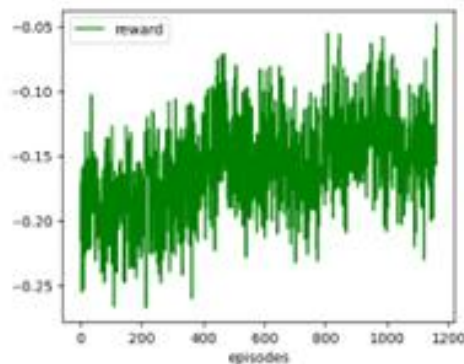


먼저 그래프에 대한 논의이다. frozenlake는 reward에 있어서 약간의 조정이 필요할 것으로 보였다. Reward 그래프를 보면 결국 목표 reward로 도달하긴 하였으나 도달할 때쯤 episode reward도 편차가 매우 큼을 확인할 수 있다. 이는 최적의 Q-value이 아닌 랜덤하게 움직여도 목표치까지 갈 수 있을 것 같다는 생각이 들게한다. 목표에 도달함이 +1, 그렇지 않으면 모두 0이라는 reward 설정이 방향성을 잡는데 도움이 되지 못했다. 이는 영상에서도 드러나는데 영상에서 agent는 왔던 길을 다시 돌아가는 action을 취하는 경우가 많다. 그 이유는 왔던 길을 반복하여 간 후 목적지로 도달하나 가장 빠른길로 도달하나 reward의 차이가 없기 때문에 그 우열을 가릴 수 없기 때문이었다고 생각한다.

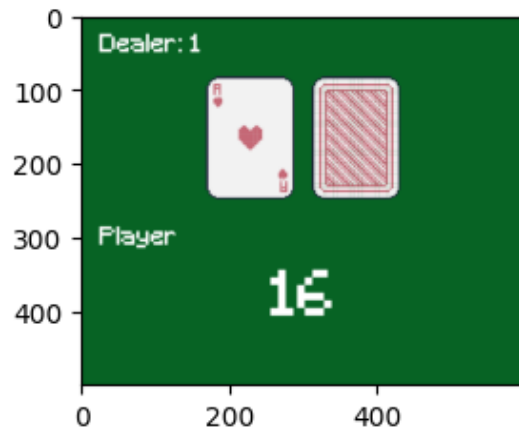
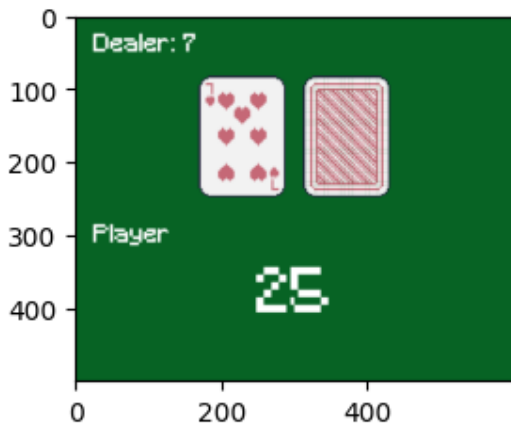
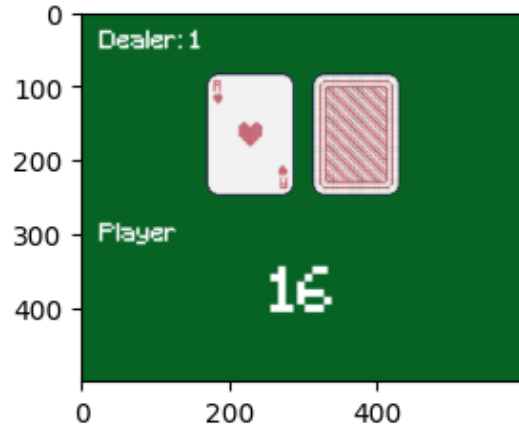
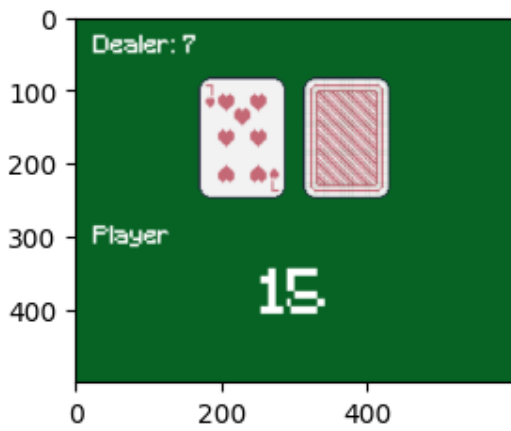
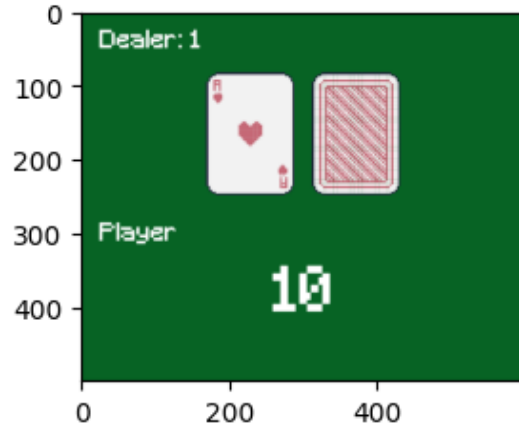
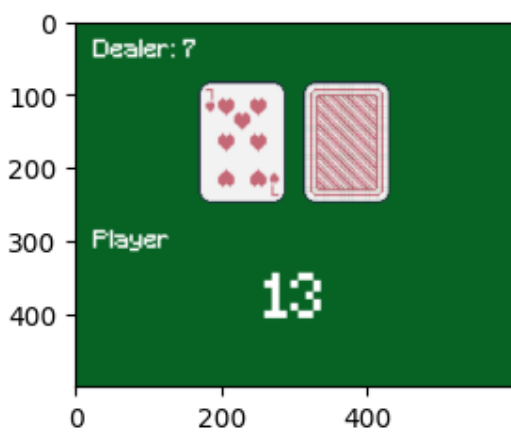


taxi는 초기에는 방향성을 잘 찾지 못해 매우 낮은 reward를 형성하나 추후에는 방향성을 잘 잡고 최적점을 향해 가는 것을 확인한다. 이는 reward 설정에서 그

이유가 드러난다. 한번 이동할 때 -1, 승객을 잘못 내리거나 태우는 경우 -10, 승객을 잘 내려준 경우 20이라는 reward 설정이 최적의 경로로 승객을 목적지까지 내려주는 것이 잘 학습되게 하였다. 영상에서도 이러한 설정이 드러났다. 택시는 승객을 태우러 가는 경로와 내려주러 가는 경로가 모두 가장 빠르고 짧은 경로로 이동하였다.



Blackjack은 딜러의 카드에 따른 랜덤적 요소가 많이 들어가 있는 경우이다. 따라서 같은 state에서 같은 action을 취하더라도 결과는 전혀 다를 수 있다. 따라서 그래프 또한 처음부터 끝까지 reward의 분포는 비슷하게 형성되었으나 평균치가 상향 조정된 것으로 보인다. 아래는 랜덤의 요소가 많이 들어가 있어도 어느정도 학습이 된 것을 확인할 수 있어서 첨부하였다. 여러가지 샘플 중 일부분이므로 모두가 아래와 같은 결과를 가진다고는 장담할 수 없음을 감안해야 한다.



왼쪽은 13을 뽑았고 딜러의 7과 숨겨진 카드의 합이 13을 넘을 수 있을 것을 고려해 추가 패를 요청하는 것을 확인한다. 이후 15가 나오게 되었고 이 값조차 부족하다고 생각하여 추가패를 요청해 21을 넘은 25의 값을 얻어 패한다. 반대로 오른쪽은 10이라는 값을 얻었고 추가 요청해 16이라는 값을 얻는다. 딜러의 1과 어떠한 수조차 더해져도 16을 넘을 수 없을 것을 확인하고 멈추는 것을 확인한다. 비슷한 15와 16인 값을 얻었을 때 딜러의 열린 카드를 보고 추가 패를 얻을

지 말지를 결정하는 것은 학습된 것을 보여주는 부분이다.

이처럼 다양한 environment에서의 Tabular Q-learning을 진행해보았다.

Environment에 적합한 reward와 각종 하이퍼파라미터를 설정하는 것이 중요해보였고 on-policy가 아닌 off-policy 형태로도 강화학습의 목적을 잘 이룰 수 있다는 것을 알 수 있었다.