

AI Programming

Lab 07: Tabular Q Learning

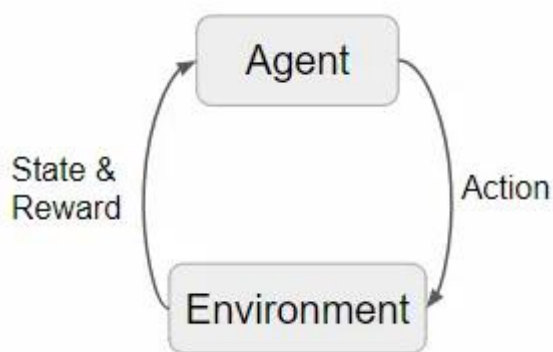
소속: 컴퓨터정보공학부

학번: 2019202103

이름: 이은비

<Lab Objective and Whole simulation program flow>

실습에 앞서서 Tabular Q learning이 무엇인지 보면 먼저 Tabular Q learning을 알기 전 reinforce learning를 생각할 수 있는데 강화 학습의 기본 아이디어는 매우 간단합니다. 즉, 행동의 결과를 관찰하고 학습함으로써 학습합니다. 이 개념은 인간이 많은 경우 학습하는 방식에 의해 동기가 부여됩니다. 우리가 무언가 action을 취하고 그것이 긍정적인 결과를 가져왔다면 우리는 그것을 다시 할 가능성이 더 높지만, 부정적인 결과가 있었다면 우리는 일반적으로 그것을 다시 할 가능성이 적습니다. 이 기본적인 메커니즘을 토대로 agent가 action을 취하고 그에 따른 environment에서의 reward를 받음으로써 agent가 다시 다음 행동을 취할 수 있도록 합니다. (아래 그림 참고)



다시 environment에서의 reward를 얻고 그 다음 행동을 결정하는 과정을 다시 생각해보면 각 단계에서 agent는 현재 environment를 잘 살펴보고 주어진 상황에서 최선의 행동이라고 생각하는 것이 무엇인지 추측합니다. 이전에 선택한 작업 즉, action을 실행하고 해당 작업이 environment를 어떻게 변경했는지 관찰합니다. 또한 특정 state에서 자신의 행동이 얼마나 좋았는지 또는 나빴는지에 대한 피드백을 받습니다. 좋은 action을 선택하면 긍정적인 reward를 받고, 나쁜 action을 선택하면 부정적인 reward를 받게 됩니다. 그리고 이러한 해당 피드백을 기반으로 학습된 추측 프로세스를 조정할 수 있습니다.

그러나 이러한 과정 속에 각 상황에서 에이전트가 행동을 선택한 후 받는 피드백/보상은 결정적이지 않고 게임의 마지막 action에 대해서만 긍정적이거나 부정적인 reward만 return할 것입니다. agent가 이기면 마지막 이동은 긍정적인 reward를 받고, 패배하면 부정적인 reward를 받으며, 무승부의 경우 보상을 받지 못합니다. 따라서 과거에 어떤 action이 발생한 기간이 길수록 그것이 최종 결과에 중요한 역할을 했을 가능성이 낮다는 가정에 기초하여 discount factor를 할당합니다. 따라서 최종 결정에 있어서 시간적 차이가 클수록 discount factor에 의해 결과에 기여한 것이 적다는 것을 알 수 있습니다. 그리고

Tabular Q learning를 단순히 Q function을 이용하여서 설명하면 다음과 같습니다. 이때 Q는 Quality를 나타내고 Q 함수는 State-Action 쌍에 품질 점수를 할당하는 함수입니다. 즉, state S와 action A가 주어지면 함수 $Q(S, A) \rightarrow \mathbb{R}$ 은 state S에서 이 action A를 수행하는 quality을 반영하는 실수를 반환합니다. 우리는 이 함수가 존재하고 deterministic이라고 가정하지만, 상태와 동작을 품질 값에 구체적으로 매핑하는 것이 반드시 알려진 것은 아닙니다. Q-learning은 이 mapping과 함수 Q를 learning하는 것입니다.

Min-Max 알고리즘에 대한 이전 부분을 다시 생각해 보면 각 상태에 대해 계산한 값이 Q 함수가 반환하는 값과 매우 유사한 것을 알 수 있는데 Min-Max 알고리즘에서는 state에 quality 값을 할당하는 반면, 여기서는 주어진 state에서 선택할 수 있는 작업에 quality 값을 할당합니다. 항상 최선의 움직임을 보이는 agent는 결정론적 환경에서 특정 상태의 특정 행동의 가치는 실제로 다음 상태의 가치를 밀접하게 반영합니다. 그러나 random한 agent는 우리의 행동에 반응하여 그러한 상대가 어떤 움직임을 선택하는지에 따라 우리는 다른 Q 값을 가진 다른 다음 상태로 결정됩니다. 따라서 행동의 Q 값은 가능한 모든 다음 상태의 값과 각각의 발생 확률의 조합이 됩니다. 이로 인해 Q function이 상대방에 종속됩니다. 상대가 항상 최선의 움직임을 선택하는 이상적인 agent라면 Q function은 Min Max 점수를 다시 밀접하게 반영합니다. 그와 반대로 Random한 움직임을 선택하는 agent라면 Min Max와는 멀어지는 것을 알 수 있습니다. 테이블 형식 Q 함수란 무엇입니까? 그리고 앞서서 Q함수의 내용을 참고한 이 맥락에서 테이블 형식은 단순히 Q 함수를 look up table에 저장한다는 의미입니다. 즉, 가능한 각 state와 action(moving)에 대한 Q 값을 저장하는 table을 만듭니다. Q 함수를 알고 있다고 가정하면 현재 상황에서 가능한 모든 동작에 대한 Q 값을 찾아 가장 높은 값을 선택하는 방식으로 진행될 수 있습니다. 그리고 만약 동일한 최고 값을 가진 가능한 이동이 두 개 이상인 경우 그 중에서 무작위로 선택하도록 했습니다. 그리고 앞서서 Q-function에서 말했듯이 가장 높은 값을 갖는다는 것은 이 움직임이 이 상황에서 가장 좋은 움직임이라는 것을 의미합니다. 그리고 이와 같은 전략을 지칭하는 것을 policy라고도 합니다.

. Pessimistic Initialization 혹은 Optimistic Initialization의 방식으로 먼저 table을 초기화하고 이후의 과정을 식을 통해 진행하는데, 먼저 각각의 개념을 간단히 설명하면 pessimistic Initialization은 모든 값을 0으로 초기화합니다. 즉, 모든 움직임이 손실로 이어진다고 가정하고 손실보다 나은 것에 만족할 가능성이 높습니다. 따라서 첫 승리 전에는 무승부로 이어지는 이동의 값이 증가합니다. Optimistic

initialization은 모든 값을 1로 초기화하고, 모든 움직임이 승리로 이어질 것이라고 기대합니다. 따라서 플레이어는 가능한 최상의 결과로 무승부에 만족할 가능성이 낮으며 먼저 다른 옵션을 적극적으로 탐색할 것입니다. 그러나 이와 같은 탐색과정 때문에 학습시간이 기어질 수 있습니다. 이후에 쓰이는 Q-function에 대한 식은 $Q(S,A) = Q(S,A) + \alpha(\gamma \max_a Q(S',a) - Q(S,A))$ 이를 다시 작성하면

$Q(S,A) = (1-\alpha)Q(S,A) + \alpha\gamma \max_a Q(S',a)$ 이며, S는 현재 state, A는 현재 action, S'는 A(현재상태의 action)를 수행한 후 state, α 는 학습률(learning rate), γ 는 discount factor, $\max_a Q(S',a)$ 는 다음 동작의 가장 높은 Q 값입니다. state S', 즉 다음 상태에서 가장 좋은 움직임의 Q 값입니다. 위 식을 정리한다면 α 를 0으로 선택하면 아무것도 변경되지 않고, α 를 1로 선택하면 이전 값을 새 값으로 완전히 대체하며, α 를 0.5로 선택하면 이전 값과 이전 값 사이의 평균을 얻습니다.

이번 과제에서도 위에서 언급한 Tabular Q learning을 이용한 3개의 종류의 게임을 실행 해보고 그에 대한 결과를 plot한 그래프와 동영상까지 확인해보는 과정을 거칩니다.

<Simulation results (screenshots)>

[taxi]

Environment를 setup한 결과를 출력합니다. Requirement already satisfied로 이미 설치된 것을 알 수 있습니다.

```
# Environment Setup
# Depending on the execution environment (Colab), it sets the environment variable RunningInCOLAB's packages
if RunningInCOLAB: # if the environment is set, installs specific packages (gymnasium, pygame, moviepy)
    !pip install gymnasium
    !pip install pygame
    !pip install 'moviepy>=1.0.3'
    from tqdm.notebook import tqdm
else: # if the environment is not set
    from tqdm import tqdm

Requirement already satisfied: gymnasium in /usr/local/lib/python3.10/dist-packages (0.29.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (1.23.5)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (4.5.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: pygame in /usr/local/lib/python3.10/dist-packages (2.5.2)
Requirement already satisfied: moviepy>=1.0.3 in /usr/local/lib/python3.10/dist-packages (1.0.3)
Requirement already satisfied: decorator<5.0,>=4.0.2 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (4.66.1)
Requirement already satisfied: requests<3.0,>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (2.31.0)
Requirement already satisfied: proglog<=1.0.0 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (0.1.10)
Requirement already satisfied: numpy<=1.17.3 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (1.23.5)
Requirement already satisfied: imageio<3.0,>=2.5 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (2.31.6)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from moviepy>=1.0.3) (0.4.9)
Requirement already satisfied: pillow<10.1.0,>=8.3.2 in /usr/local/lib/python3.10/dist-packages (from imageio<3.0,>=2.5->moviepy>=1.0.3) (9.4.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from imageio-ffmpeg>=0.2.0->moviepy>=1.0.3) (67.7.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy>=1.0.3) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy>=1.0.3) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy>=1.0.3) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.8.1->moviepy>=1.0.3) (2023.7.22)
```

Gym.__version을 출력한 결과 입니다.

```
# import the libraries(by using gym.)
gym.__version__

'0.29.1'
```

taxi예제에 맞는 정보를 print out 한 결과입니다.

```
] # print out the information.
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 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)
Vallue shape (1,)
Value dimensions 1
```

Q_table을 define하고 Q_table의 shape를 출력한 결과입니다.

```
] # define q-table
### START CODE HERE ###
# the number of states is 500, and the number of actions is 6
Q_table = np.zeros((num_states,action_range)) # define q-table

### END CODE HERE ###

print(Q_table.shape) # print out the Q_table's shape

(500, 6)
```

Print out the result of the Tabular-Q-learning in case that is taxi

```
# Update Q-table based on Q-learning algorithm
Q_table[state, action] = (1 - learning_rate) * Q_table[state, action] + learning_rate * (reward + gamma * np.max(Q_table[next_state, :]))

### END CODE HERE ###

epis_rewards += reward # accumulate rewards for the episode
state = next_state # update state for the next step

if epis_steps > max_steps:
    break # If the episode runs too long, terminate it
else:
    epis_steps += 1

history['rewards'].append(epis_rewards) # Store episode rewards in history
pbar.set_postfix({'reward': epis_rewards, 'steps': epis_steps}) # Update progress bar with reward and steps

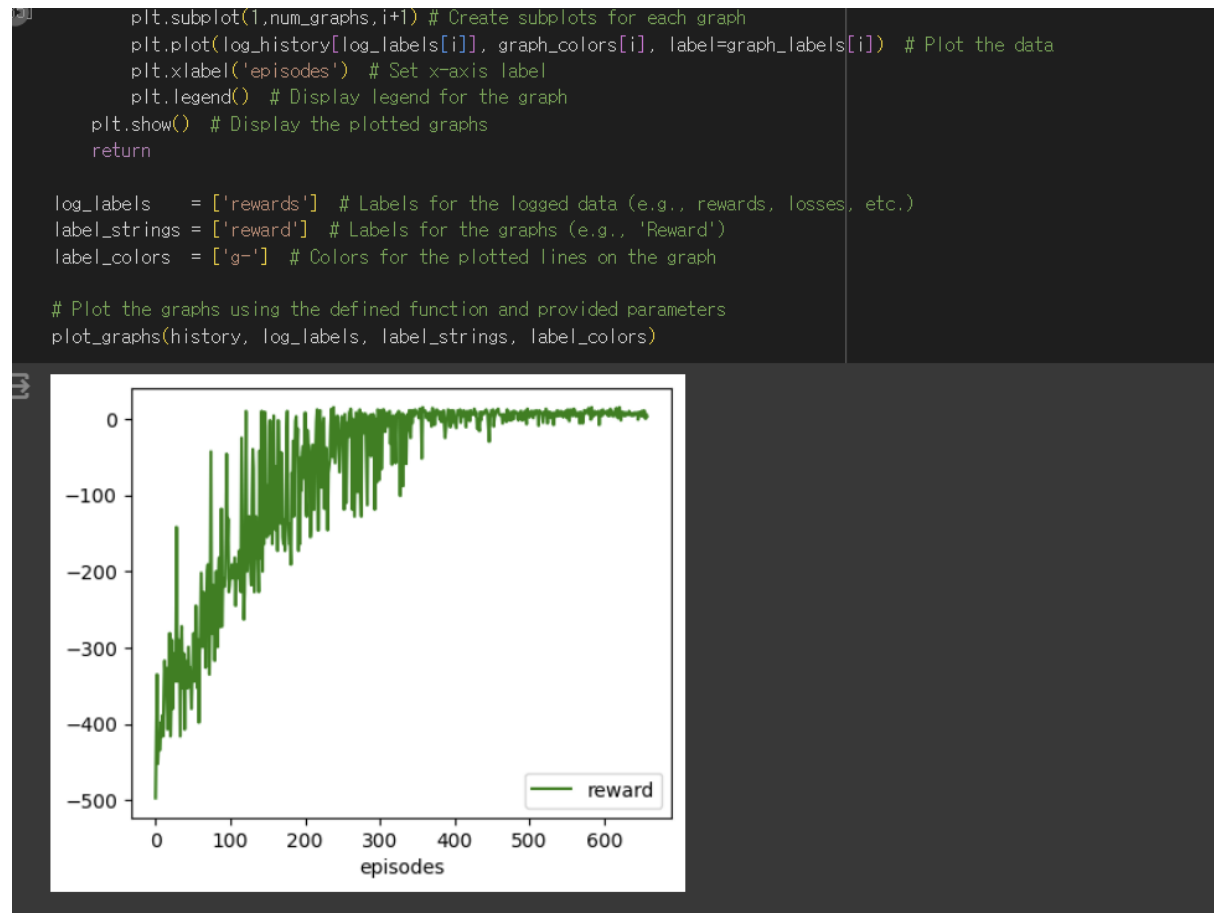
eval_reward = evaluate_policy(env, Q_table, val_episodes) # evaluate the policy over validation episodes

if eval_reward > goal_score:
    break # Exit loop if the goal score is achieved

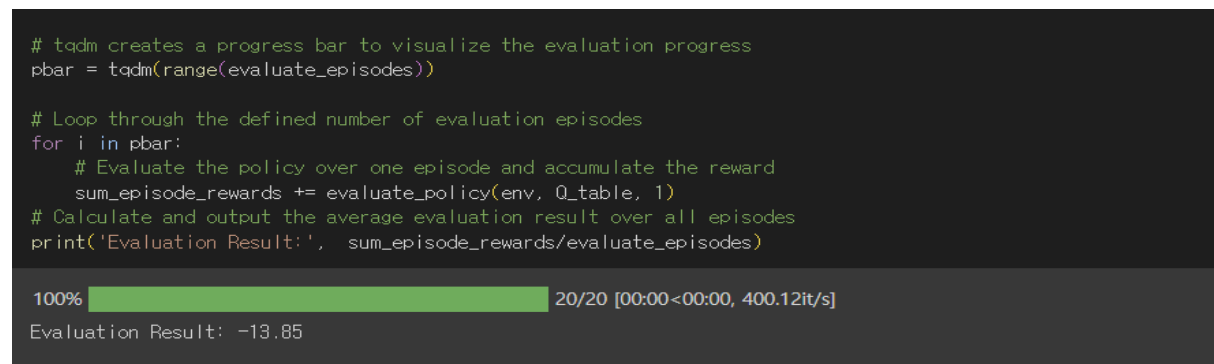
# Output the final number of episodes and the achieved final reward
print('episodes: {0:5d}, final_reward {1:4.2f}'.format(episodes + 1, eval_reward))

33% | 657/2000 [01:00<00:45, 29.70it/s, reward=3, steps=18]
episodes: 658, final_reward 8.65
```

Episode가 진행됨에 따라 얻는 reward를 graph로 plot한 결과를 출력합니다.



얻는 reward를 통해 policy를 평가하고 다시 책정하는 과정이후 evaluation result를 출력합니다.




Taxi case에 맞는 비디오를 생성하고 그에 대한 정보를 print out합니다.

```
eval_reward = evaluate_policy(env, Q_table, 1)

# Output the total reward achieved in the sample evaluation episode
print('Sample Total Reward: ', eval_reward)

# Close the environment after recording the video
env.close()
```

 /usr/local/lib/python3.10/dist-packages/gymnasium/wrappers/record_video.py:94: UserWarning: WARN: Overwriting existing videos
logger.warn(
Moviepy - Building video /content/gym-results/taxi-episode-0.mp4.
Moviepy - Writing video /content/gym-results/taxi-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/gym-results/taxi-episode-0.mp4
Sample Total Reward: 8.0

생성된 비디오를 display하도록 합니다.

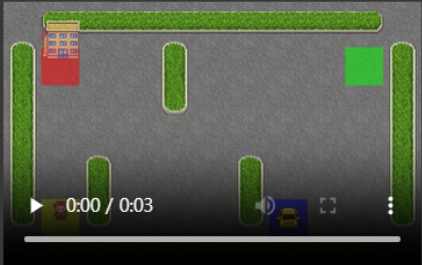


231128_taxi.mp4

```
# play the generated video.
from IPython.display import HTML # Importing necessary modules for displaying HTML content
from base64 import b64encode # Module to encode video file into base64

# Function to display a video file in the Jupyter Notebook
def show_video(video_path, video_width = 320):
    # Read the video file and encode it into base64
    video_file = open(video_path, "r+b").read()
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
    # Return HTML code to display the video with the specified width and controls for playback
    return HTML(f'<video width={video_width} controls><source src="{video_url}"></video>')

# Display the video located at the specified path
show_video('./gym-results/' + res_prefix + '-episode-0.mp4')
```



[lake]

아래 출력 결과들은 위의 설명과 동일하며 case만 lake로 바뀐 것 입니다.

```

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('Value 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)
Value shape (1,)
Value dimensions 1

```

```

# define q-table
### START CODE HERE ###
# the number of states is 500, and the number of actions is 6
Q_table = np.zeros((num_states,action_range))

### END CODE HERE ###

print(Q_table.shape) # print out the Q_table's shape

```

(16, 4)

```

# Update Q-table based on Q-learning algorithm
Q_table[state, action] = (1 - learning_rate) * Q_table[state, action] + learning_rate * (reward + gamma * np.max(Q_table[next_state, :]))

### END CODE HERE ###

epis_rewards += reward # accumulate rewards for the episode
state = next_state # update state for the next step

if epis_steps > max_steps:
    break # If the episode runs too long, terminate it
else:
    epis_steps += 1

history['rewards'].append(epis_rewards) # Store episode rewards in history

pbar.set_postfix({'reward': epis_rewards, 'steps': epis_steps}) # Update progress bar with reward and steps

eval_reward = evaluate_policy(env, Q_table, val_episodes) # evaluate the policy over validation episodes

if eval_reward > goal_score:
    break # Exit loop if the goal score is achieved

# Output the final number of episodes and the achieved final reward
print('episodes:{0:5d}, final_reward {1:4.2f}'.format(episodes + 1, eval_reward))

```

3

67% | 1341/2000 [00:24<00:20, 32.95it/s, reward=1, steps=40]

episodes: 1342, final_reward 0.85

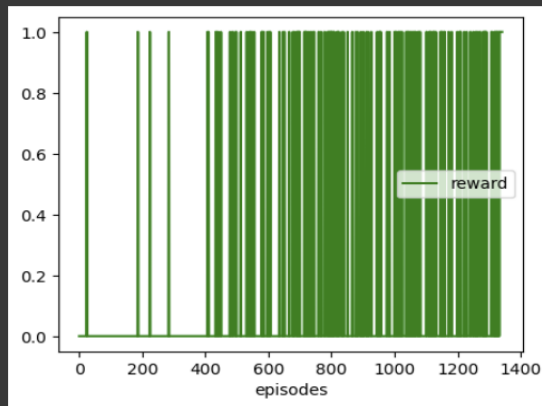

```

plt.plot(history['loss'], log_labels[1], label='loss', color='r'); # Plot the loss
plt.xlabel('episodes') # Set x-axis label
plt.legend() # Display legend for the graph
plt.show() # Display the plotted graphs
return

log_labels = ['rewards'] # Labels for the logged data (e.g., rewards, losses, etc.)
label_strings = ['reward'] # Labels for the graphs (e.g., 'Reward')
label_colors = ['g-'] # Colors for the plotted lines on the graph

# Plot the graphs using the defined function and provided parameters
plot_graphs(history, log_labels, label_strings, label_colors)

```



```

# set the parameters
evaluate_episodes = 20 # Number of episodes for evaluation
sum_episode_rewards = 0.0 # Initialize the sum of episode rewards

# tqdm creates a progress bar to visualize the evaluation progress
pbar = tqdm(range(evaluate_episodes))

# Loop through the defined number of evaluation episodes
for i in pbar:
    # Evaluate the policy over one episode and accumulate the reward
    sum_episode_rewards += evaluate_policy(env, Q_table, 1)
# Calculate and output the average evaluation result over all episodes
print('Evaluation Result:', sum_episode_rewards/evaluate_episodes)

```

100%  20/20 [00:00<00:00, 405.31it/s]
Evaluation Result: 0.75

```

# Read the video file and encode it into base64
video_file = open(video_path, "r+b").read()
video_url = f"data:video/mp4;base64,{base64.encode(video_file).decode()}"
# Return HTML code to display the video with the specified width and controls for playback
return HTML(f'<video width={video_width} controls><source src="{video_url}"></video>')

# Display the video located at the specified path
show_video('./gym-results/' + res_prefix + '-episode=0.mp4')

```





231128_lake.mp4

[blackjack]

아래 출력 결과들은 위의 설명과 동일하며 case만 blackjack로 바뀐 것 입니다.

```
# print out the information.
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 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)
Vallue shape (1,)
Value dimensions 1
```

```
# define q-table
### START CODE HERE ###
# the number of states is 500, and the number of actions is 6
Q_table = np.zeros((num_states,action_range)) # define q-table

### END CODE HERE ###

print(Q_table.shape) # print out the Q_table's shape

(704, 2)
```

```

next_state, reward, done = env_step(env, action) # take the action and observe outcomes

# Update Q-table based on Q-learning algorithm
Q_table[state, action] = (1 - learning_rate) * Q_table[state, action] + learning_rate * (reward + gamma * np.max(Q_table[next_state, :]))

### END CODE HERE ###

epis_rewards += reward # accumulate rewards for the episode
state = next_state # update state for the next step

if epis_steps > max_steps:
    break # If the episode runs too long, terminate it
else:
    epis_steps += 1

history['rewards'].append(epis_rewards) # Store episode rewards in history

pbar.set_postfix({'reward': epis_rewards, 'steps': epis_steps}) # Update progress bar with reward and steps

eval_reward = evaluate_policy(env, Q_table, val_episodes) # evaluate the policy over validation episodes

if eval_reward > goal_score:
    break # Exit loop if the goal score is achieved

# Output the final number of episodes and the achieved final reward
print('episodes: {0:5d}, final_reward {1:4.2f}'.format(episodes + 1, eval_reward))

```



100% 2000/2000 [00:12<00:00, 160.57it/s, reward=1, steps=4]
episodes: 2000, final_reward -0.45

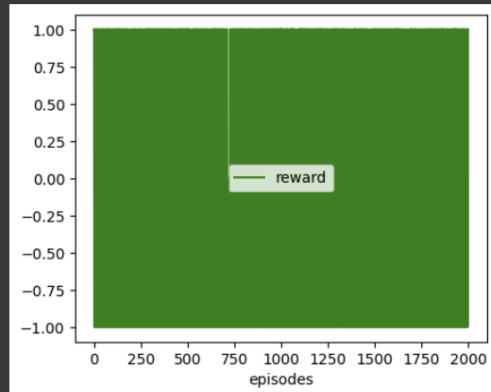
```

plt.subplot(1, num_graphs, i+1) # Create subplots for each graph
plt.plot(log_history[log_labels[i]], graph_colors[i], label=graph_labels[i]) # Plot the data
plt.xlabel('episodes') # Set x-axis label
plt.legend() # Display legend for the graph
plt.show() # Display the plotted graphs
return

log_labels = ['rewards'] # Labels for the logged data (e.g., rewards, losses, etc.)
label_strings = ['reward'] # Labels for the graphs (e.g., 'Reward')
label_colors = ['g-'] # Colors for the plotted lines on the graph

# Plot the graphs using the defined function and provided parameters
plot_graphs(history, log_labels, label_strings, label_colors)

```



```
# set the parameters
evaluate_episodes = 20 # Number of episodes for evaluation
sum_episode_rewards = 0.0 # Initialize the sum of episode rewards

# tqdm creates a progress bar to visualize the evaluation progress
pbar = tqdm(range(evaluate_episodes))

# Loop through the defined number of evaluation episodes
for i in pbar:
    # Evaluate the policy over one episode and accumulate the reward
    sum_episode_rewards += evaluate_policy(env, Q_table, 1)
# Calculate and output the average evaluation result over all episodes
print('Evaluation Result:', sum_episode_rewards/evaluate_episodes)
```

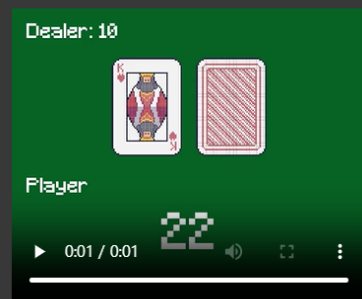
100% 20/20 [00:00<00:00, 862.71it/s]

Evaluation Result: -0.3

```
# play the generated video.
from IPython.display import HTML # Importing necessary modules for displaying HTML content
from base64 import b64encode # Module to encode video file into base64

# Function to display a video file in the Jupyter Notebook
def show_video(video_path, video_width = 320):
    # Read the video file and encode it into base64
    video_file = open(video_path, "r+b").read()
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
    # Return HTML code to display the video with the specified width and controls for playback
    return HTML(f'<video width={video_width} controls><source src="{video_url}"></video>')

# Display the video located at the specified path
show_video('./gym-results/' + res_prefix + '-episode=0.mp4')
```

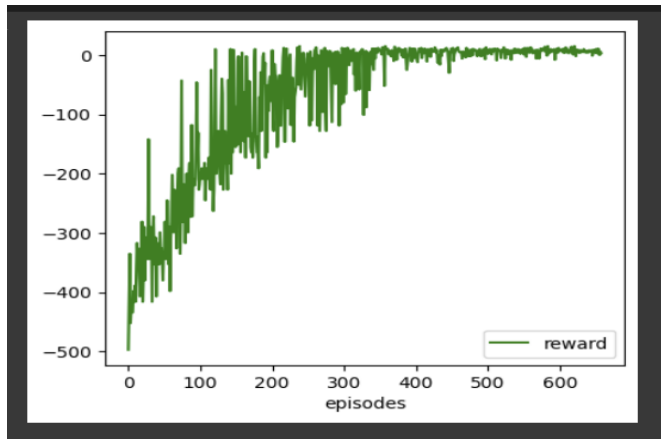


231128_black.mp4

< Discussion (Describe the 3 graphs and 3 videos - Without comparing each other)>

먼저 각각의 graph 와 video 를 첨부하면 아래와 같습니다.

Taxi 예제의 그래프는 다음과 같으며, 비디오는 다음과 같습니다.



231128_taxi.mp4

기본적인 정보는 다음과 같으며

Action Space Discrete(6)

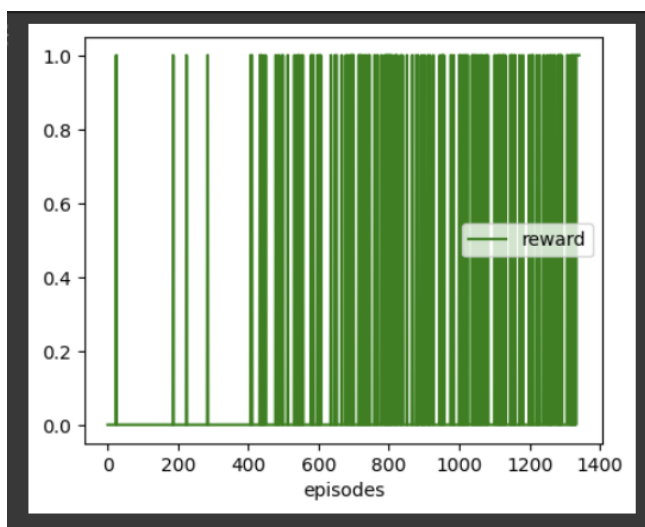
Observation Space Discrete(500)

```
import gymnasium.make("Taxi-v3")
```

각 단계마다 -1의 페널티가 발생하여 agent가 효율적으로 목표에 도달할 수 있도록 유도합니다.

차가 올바른 목적지로 배달하면 agent는 +20의 reward를 받고 잘못된 장소에서 픽업 또는 하차를 실행하면 -10의 reward가 부과됩니다. 이 정보를 염두에 두고 훈련 루프 내의 'epis_rewards' 변수는 각 episodes에서 얻은 누적 reward를 추적합니다. 이 누적 reward는 에이전트가 에피소드를 성공적으로 완료하면 받는 positive reward(+20)와 함께 발생하는 penalty(단계당 1개, 불법 행위 시 -10개)를 원리로 과정을 진행합니다. 에이전트가 학습하고 취한 과정을 기반으로 Q-테이블이 업데이트됨에 따라 에피소드당 누적 reward는 agent의 학습 진행 상황과 환경을 얼마나 잘 탐색하는지를 반영해야 하며, optimized policy를 학습하여 total reward를 극대화하는 것을 목표로 합니다. 따라서 그래프도 accumulate하는 그래프의 모양이 되는 것을 알 수 있습니다.

Lake 예제의 그래프는 다음과 같으며, 비디오는 다음과 같습니다.



231128_lake.mp4

기본적인 정보는 다음과 같으며

Action Space Discrete(4)

Observation Space Discrete(16)

```
import gymnasium.make("FrozenLake-v1")
```

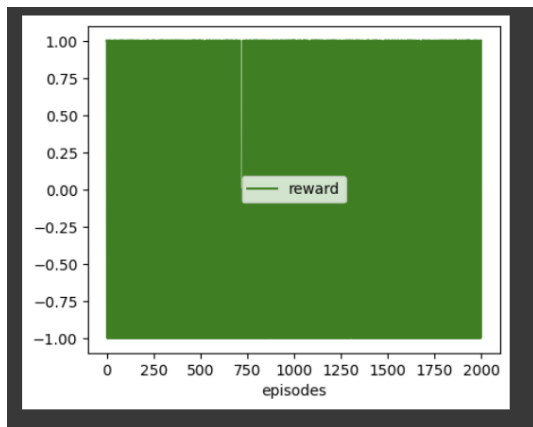
Reach goal: +1

Reach Hole or Frozen: 0

reward 은 1 아님 0 인 이진법으로, goal 에 도달하면 +1 의 reward 가 주어지며 episode 의 성공적인 완료를 의미합니다. 그러나 구멍에 빠지거나 얼어붙은 타일에 남아 있으면 0 의 reward 가 발생합니다. episode 전반에 걸쳐 agent 의 goal 은 구멍이나 얼어붙은 타일을 피하면서 goal 에 도달함으로써 누적 reward 을 최대화하는 것입니다. goal 에 도달한 성공적인 episode 마다 +1 의 보상이 주어지고, 다른 결과(구멍에 떨어지거나 얼어붙은 타일에 남아 있음)는 0 의 reward 가 주어집니다.

episode 당 누적 reward 은 이러한 reward 구조를 반영하며, goal 상태를 달성하면 긍정적인 reward 을 제공하고, 중립적인 보상을 제공하는 다른 상태나 행동을 반영합니다. 따라서 그래프의 모양도 마치 spike 처럼 binary 한 결과가 plot 되는 것을 알 수 있습니다.

blackjack 예제의 그래프는 다음과 같으며, 비디오는 다음과 같습니다.



231128_black.mp4

기본적인 정보는 다음과 같습니다.

Action Space Discrete(2)

Observation Space Tuple(Discrete(32), Discrete(11), Discrete(2))

```
import gymnasium.make("Blackjack-v1")
```

아래 룰을 따르며 경우에 따른 reward 를 받는데

Win Game: +1 (if the player wins the game)

Lose Game: -1 (if the player loses the game)

Draw Game: 0 (if the game results in a draw)

Win Game with Natural Blackjack: +1.5 (if natural is True) or +1 (if natural is False)

간단하게 설명하면 다음과 같습니다. 보상은 각 게임 에피소드의 결과에 따라 할당됩니다:

player 가 게임에서 승리하면 +1 의 reward 을 받고,게임에서 지면 -1 의 보상을 받습니다.

player 와 dealer 가 모두 승리하지 못한 무승부의 경우 reward 은 0 입니다. 추가적으로 플레이어가 내추럴 블랙잭으로 시작하여 '내추럴'을 True 로 설정하면 +0.5 의 추가 보상을 받게 됩니다(전체적으로 +1.5 까지). 내추럴을 False 로 설정하면 내추럴 블랙잭의 보상은 +1 이 됩니다.

이러한 policy 를 갖고 게임 결과에 따른 결과를 반영하여 에이전트가 이를 초과하지 않고 총 21 개에 가까운 승리를 목표로 하도록 유도하여 궁극적으로 여러 에피소드에 걸쳐 총 누적 보상을 극대화합니다. 근데 사실 아직까지 black jack 의 결과 그래프와 비디오가 맞는지 모르겠습니다. 정확히 문제가 이해가 안된 것인지 경우에 따른 reward 값이 존재하므로 그에 맞게 plot 되어야 할 것 같은데 하는 의문이 아직 남아있는 것 같습니다.