Temporal-Difference RL

Younghoon Kim

nongaussian@hanyang.ac.kr

Keywords

- Temporal-difference learning ★★★
- Target value ★★★★★
- SARSA ★★★★
- Q-learning ★★★★★
- How to use Gym environments ★★★★★

Recall The MC Prediction Algorithm

```
maxiter = 1000000
gamma = 1
epsilon = 0.4

N = np.zeros((10, 10, 2, 2), dtype='float32')
SUM = np.zeros((10, 10, 2, 2), dtype='float32')
Q = np.random.uniform(size=(10, 10, 2, 2))

for _ in range(maxiter):
    episode = generate_episode()
    G = 0.
    last_step = episode[0].pop()
    while len(episode[0]) > 0:
        G = gamma * G + last_step['reward']
        last_step = episode[0].pop()
        last_action = episode[1].pop()
        # exploring-start estimation: if the state appears for the first time
        # in the episode, update Q value
        observ = last_step['observation']
        idx = (observ[0] - 12, observ[1] - 1, observ[2], last_action)
        if not in_episode(episode, observ, last_action):
            N[idx] += 1.
            SUM[idx] += G
            Q[idx] = SUM[idx] / N[idx]
```

Successful Episode By The Estimated Policy

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
```

Temporal-Difference (TD) Learning

- With a stream of state and return pairs (S_t, G_t),
  - A simple way to compute the expectation of $V(S_t)$ is *exponential moving average*

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t$$

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$$V(S_t) \leftarrow V(S_t) + \alpha[\boxed{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

Called *target* value for expectation

Tabular TD(0): For Evaluating Given Policy

- Given a discount γ and a policy π as inputs
  - Randomly initialize $V(s_i), \forall s_i \in S$
  - For terminal states $s_i, V(s_i) = 0$
- Repeat
  - Sample S_0 randomly and choose an action A_0 from S_0 w.r.t Q with π
  - For each t from 0 to T-1
    - Take the action A_t with π and observe R_{t+1} and S_{t+1}
    - $V(S_t) \leftarrow V(S_t\ ) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

MC vs. TD

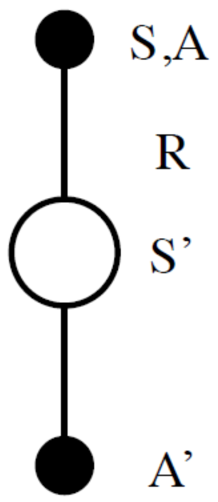
Monte Carlo learning

- Requires **a full sequence of state**, action and reward (i.e., episode)
- Need to examine to the end of episode to compute G_t

Temporal difference learning

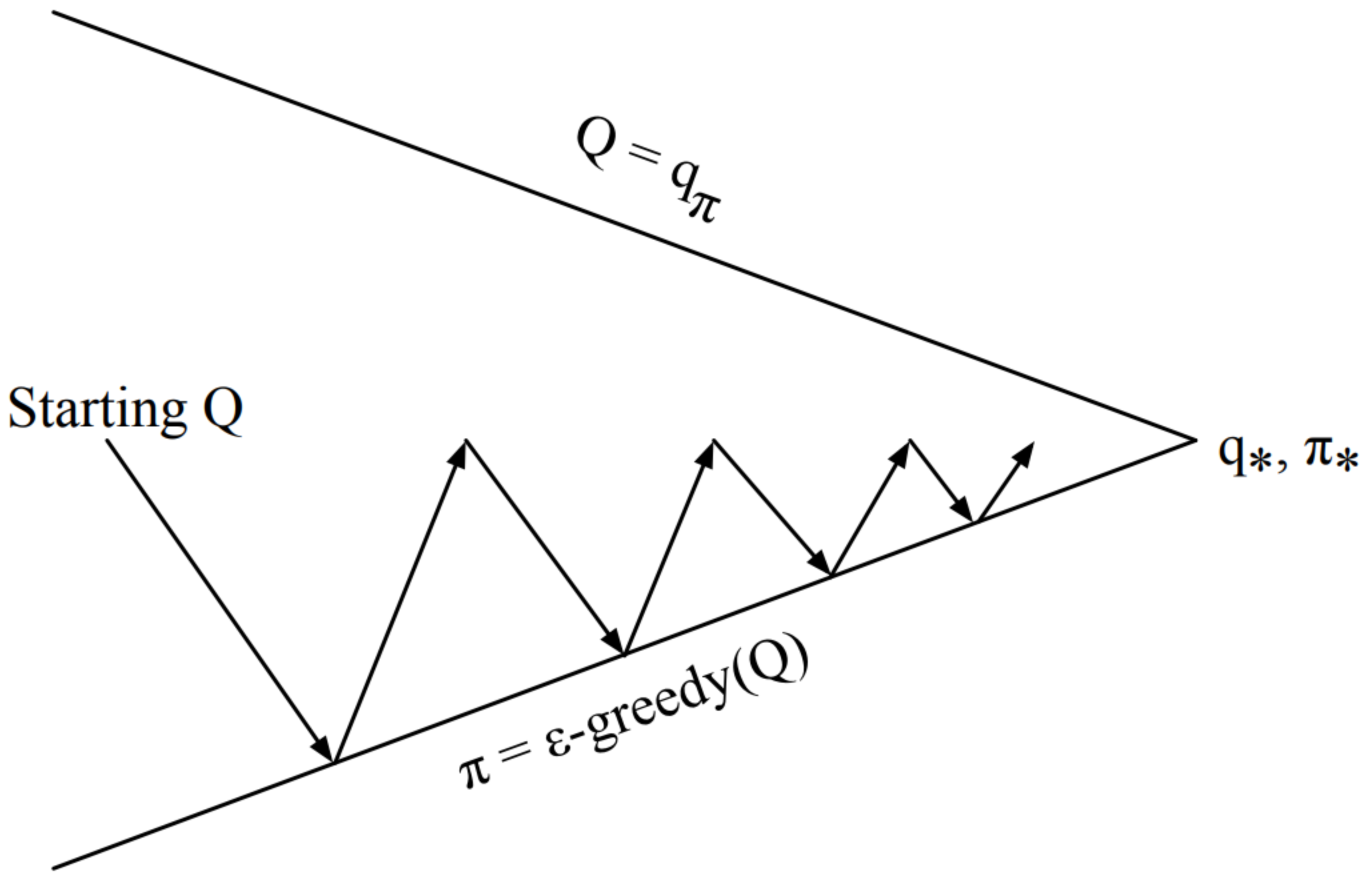
- Learn **based on only a step** (or n steps) of experience
- Lower variance than MC

On-Policy Control With SARSA



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A)\right)$$

On-Policy Control With SARSA

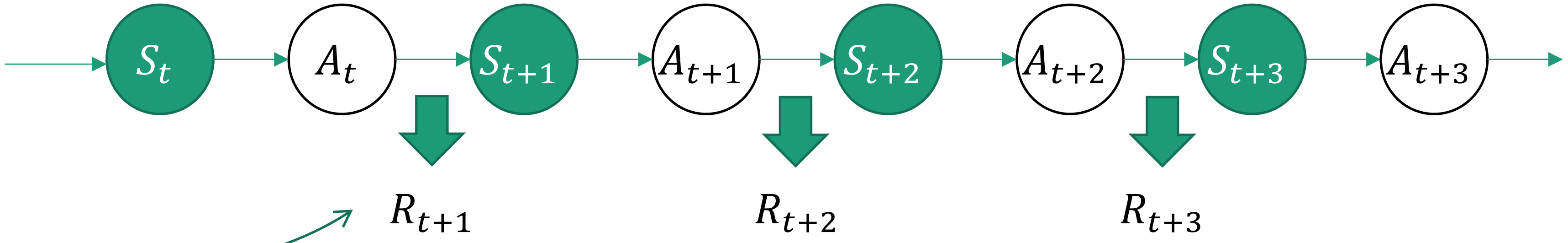


SARSA

- $V(s_i)$대신 action-value function $Q(s_i, a_k)$ 을 바로 학습
  - Update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- S_t에서 시작하는 에피소드를 보자면,



Uses every element of quintuple of events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

SARSA: On-Policy Control

$$\pi(a, s_i) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, if\ a = \arg\max_{a \in A} q(s_i, a) \\ \frac{\epsilon}{|A|} \end{cases}$$

- Repeat
  - 첫 상태 S_0을 임의로 샘플링 후 ϵ-soft greedy policy 에 따라 행동 A_0을 선택
  - For each t from 0 to T-1
    - With the action A_t, observe R_{t+1} and S_{t+1}
    - ϵ-soft greedy policy 에 따라 행동 A_{t+1}을 선택
    - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\boxed{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})} - Q(S_t, A_t)]$

Target

Learning To Play Blackjack With SARSA

```
maxiter = 1000000
gamma = 1
epsilon = 0.3
lr_rate = 0.8

Q = np.random.uniform(size=(10, 10, 2, 2))

for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_eps_soft_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            Q[idx1] += lr_rate * (next_step['reward'] - Q[idx1])
            done = True
```

Generate the first step by following ϵ-soft policy

```python
for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_eps_soft_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            Q[idx1] += lr_rate * (next_step['reward'] - Q[idx1])
            done = True
        else:
            next_action = get_eps_soft_action(next_step)

            state = step['observation']
            next_state = next_step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            idx2 = (next_state[0] - 12, next_state[1] - 1, next_state[2], next_action)
            Q[idx1] += lr_rate * ((next_step['reward'] + gamma * Q[idx2]) - Q[idx1])

            step = next_step
            action = next_action
```

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} - Q(S_t, A_t)]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Policy Found By SARSA

<Without Ace>

Dealer / Player	2	3	4	5	6	7	8	9	10	A
21	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	0
17	1	1	1	0	1	0	0	1	1	1
16	1	1	1	1	0	0	0	1	1	0
15	1	1	0	1	1	0	1	0	0	0
14	1	0	1	0	1	0	0	0	1	0
13	1	0	0	1	0	1	0	1	0	0
12	0	1	1	0	1	0	0	0	1	0

<With Ace>

Dealer / Player	2	3	4	5	6	7	8	9	10	A
21	1	1	1	1	0	1	1	1	1	0
20	0	1	1	0	1	1	1	0	1	0
19	1	1	1	1	1	1	1	1	0	1
18	1	0	1	1	1	1	1	0	0	0
17	0	1	0	0	0	1	1	0	0	0
16	0	0	1	0	0	1	0	0	0	0
15	0	0	1	1	0	0	1	1	0	0
14	0	0	0	0	0	1	0	0	0	1
13	0	0	1	0	0	0	0	0	0	1
12	1	1	0	0	1	0	0	0	0	0

Q-Learning: Off-Policy TD Control

- Repeat
  - 첫 상태 S_0을 임의로 샘플링 후 ϵ-soft greedy policy 에 따라 행동 A_0을 선택
  - For each t from 0 to T-1
    - Take the action A_t and observe R_{t+1} and S_{t+1} w.r.t Q with an ϵ-soft policy
    - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\boxed{R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a)} - Q(S_t, A_t)\right]$
    - ϵ-soft greedy policy 에 따라 행동 A_{t+1}을 선택

Target

Blackjack By Q-Learning

```
maxiter = 1000000
gamma = 1
epsilon = 0.3
lr_rate = 0.8

Q = np.random.uniform(size=(10, 10, 2, 2))

for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_random_action(step)
    done = False
    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            Q[idx1] += lr_rate * (next_step['reward'] - Q[idx1])
            done = True
```

```python
for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_random_action(step)
    done = False
    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            Q[idx1] += lr_rate * (next_step['reward'] - Q[idx1])
            done = True
        else:
            best_action = get_greedy_action(next_step)

            state = step['observation']
            next_state = next_step['observation']
            idx1 = (state[0] - 12, state[1] - 1, state[2], action)
            idx2 = (next_state[0] - 12, next_state[1] - 1, next_state[2], best_action)
            Q[idx1] += lr_rate * ((next_step['reward'] + gamma * Q[idx2]) - Q[idx1])

            next_action = get_eps_soft_action(step)
            step = next_step
            action = next_action
```

$$\underset{a \in A}{\operatorname{argmax}}\, Q\left(S_{t+1}, a\right)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Practice

- Q-learning를 이용해 블랙잭 게임을 학습하고 최적의 정책 테이블을 출력하세요.
- 다음 상수값을 바꾸어가며 실험해보세요.
  - maxiter
  - gamma
  - epsilon
  - lr_rate

Policy By Q-Learning

<Without Ace>

Dealer Player	2	3	4	5	6	7	8	9	10	A
21	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1
19	1	1	1	1	1	1	1	1	0	1
18	1	1	1	1	1	1	1	1	0	1
17	1	0	1	1	1	1	0	0	1	0
16	1	1	0	1	0	0	0	0	1	0
15	1	0	1	0	0	0	1	0	0	1
14	1	0	0	1	1	1	1	0	0	1
13	0	1	0	1	1	1	0	1	1	0
12	1	0	0	0	1	0	0	1	1	0

<With Ace>

Dealer Player	2	3	4	5	6	7	8	9	10	A
21	1	1	1	1	0	1	1	0	1	1
20	1	1	1	0	1	0	0	1	0	1
19	1	0	0	1	1	1	0	1	0	1
18	0	0	0	0	1	1	0	1	0	0
17	0	0	0	1	1	0	1	0	0	0
16	0	0	1	1	0	1	0	1	0	0
15	0	0	0	0	0	0	0	0	1	0
14	0	1	0	1	0	0	0	1	0	0
13	0	1	0	0	1	0	0	0	0	0
12	0	0	1	0	0	0	0	0	0	0

Gym- Introduce

- "학습 알고리즘과 환경 간의 통신을 위한 표준 API를 제공하여 강화 학습 알고리즘을 개발하고 비교하기 위한 오픈 소스 Python 라이브러리 입니다."

  -- https://github.com/**openai**/gym
- 비디오 게임(Atari) 및 로봇용 물리적 세계 (e.g., PoleCart, Box2D)와 같은 환경이 포함

Gym’s APIs For Interacting with Environment

- `env.reset()`: initialize the state and return the first state of an episode
  - Observation, info: see below
- `env.step(action)`: go ahead with the given action and returns
  - Observation: the resulting state
  - Reward
  - Done: True if the episode is over
  - Truncated: if the episode is truncated
  - Info: a dictionary including additional information such as performance and latency for debugging purpose
- `env.render()`: render ANSI text or image to visualize each step
  - With mode=‘rgb_array’, it returns a numpy array with three channel

Use Blackjack Env In Gym

```
import gym
env = gym.make('Blackjack-v0')

step = env.reset()
print(step)
```

(Integer 1~31, Integer 1~10, [True|False])

```
(21, 10, True)
```

```
env.step(0)
```

Input: an action (0 = stay, 1 = hit)

```
((21, 10, True), 1.0, True, {})
```

Output: (step, reward, done, info)

Wrapping Gym’s APIs For Our Code

```
# wrapper for gym's blackjack environment
def generate_start_step():
    return { 'observation': env.reset(),
             'reward': 0., 'step_type': STEPTYPE_FIRST }


def generate_next_step(step, action):
    obs, reward, done, info = env.step(action)
    step_type = STEPTYPE_LAST if done else STEPTYPE_MID
    return { 'observation': obs,
             'reward': reward, 'step_type': step_type }
```

Q-Learning With Blackjack Env In Gym

```python
import gym
import numpy as np

env = gym.make('Blackjack-v0')

maxiter = 1000000
gamma = 1
epsilon = 0.3
lr_rate = 0.8

Q = np.random.uniform(size=(32, 11, 2, 2))

for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_random_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)
```

```python
for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_random_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state[0], state[1], int(state[2]), action)
            Q[idx1] = Q[idx1] + lr_rate * (next_step['reward'] - Q[idx1])
            done = True
        else:
            best_action = get_greedy_action(next_step)

            state = step['observation']
            next_state = next_step['observation']
            idx1 = (state[0], state[1], int(state[2]), action)
            idx2 = (next_state[0], next_state[1], int(next_state[2]), best_action)
            Q[idx1] += lr_rate * ((next_step['reward'] + gamma * Q[idx2]) - Q[idx1])

            action = get_eps_soft_action(step)
            step = next_step
```

약간 변경이 필요한데, 코딩을 더 잘 한다면 wrapper만 바꾸고 본 알고리즘은 바꿀 필요가 없겠지요?

Policy By Q-Learning

<Without Ace>

Dealer / Player	2	3	4	5	6	7	8	9	10	A
21	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
18	0	0	1	0	0	1	0	1	1	0
17	0	0	1	0	0	0	1	1	0	0
16	1	0	0	0	0	0	0	0	0	1
15	1	1	0	0	0	0	1	0	0	0
14	1	1	0	0	1	0	1	1	0	1
13	0	1	0	1	0	1	1	0	0	1
12	1	0	0	1	0	1	1	1	1	1

<With Ace>

Dealer / Player	2	3	4	5	6	7	8	9	10	A
21	1	0	0	0	0	0	0	0	0	0
20	0	0	0	1	0	0	0	0	1	0
19	1	0	0	0	1	0	1	0	0	1
18	0	0	0	0	0	0	0	1	1	0
17	0	1	1	1	0	0	1	0	0	1
16	0	1	0	0	1	1	1	1	1	0
15	0	0	1	0	0	1	1	0	0	1
14	0	1	0	1	1	0	1	1	0	1
13	1	1	0	1	1	1	1	0	1	0
12	1	1	1	1	1	1	1	1	1	1

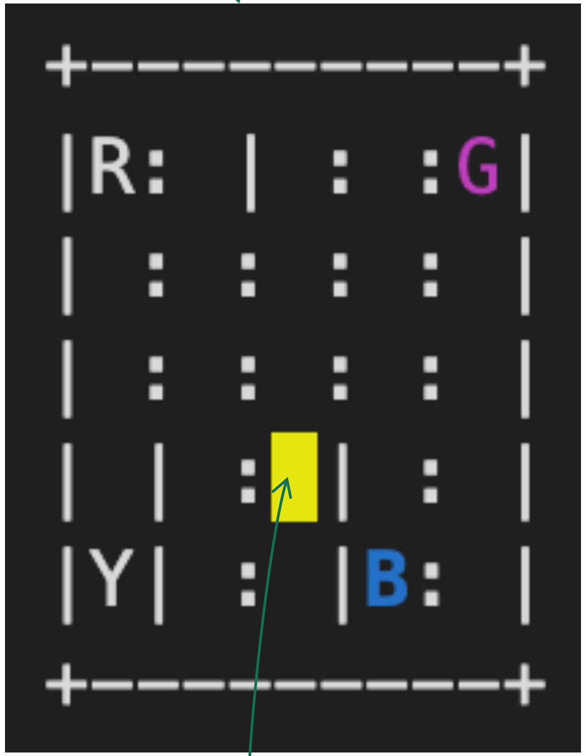
Gym 환경에서는 게임에 단일 카드 덱을 사용하기 때문에 최적의 정책이 저희와 는 많이 다름

Practice: Taxi Problem

- <u>택시는 임의의 광장에서 출발하고 승객은 임의의 위치에 대기</u>
- Goal of control
  - 1) 택시가 승객의 위치로 이동
  - 2) 승객을 태움
  - 3) 승객의 목적지(지정된 네 곳 중 다른 한 곳)까지 주행한 다음
  - 4) 승객을 하차
  - 5) 승객을 내려주면 에피소드가 종료

승객의 목적지는 R, G, B, Y 중에서 무작위로 선택

R, G, B, Y의 위치는 고정



택시는 처음에 각 에피소드에서 무작위로 배치

Gym’s Toy Text

- pip install gym[toy_text]==0.25.2

Taxi Problem: Actions

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Taxi Problem: Observation

Passenger locations:
- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)
- 4: in taxi

Destinations:
- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)

- 500 combinations of
  - 25 택시위치
  - 5 승객 위치 (including the case when the passenger is in the taxi)
  - 4 목적지 위치
- Each state space can be represented by the tuple: (taxi_row, taxi_col, passenger_location, destination)
  - Instead, the env in gym represents it using a single integer value ranged from 0 to 499
  - Note that you need not to know the meaning of a state

Taxi Problem: Rewards

- 직접 보상을 디자인해봅시다 (1 min)
- Rewards
  - +20 승객을 목적지에 성공적으로 하차
  - -10 불법적으로 " 픽업" 및 "하차" 행위를 실행하는 경우(예: 승객 없이 광장에서 픽업하고 목적지가 아닌 광장에서 하차하는 경우)
  - -1 이 외, 매 스텝마다 (시간 지연 패널티)

Taxi Environment In Gym

```
import gym
env = gym.make('Taxi-v3')

# step types
STEPTYPE_FIRST = 0
STEPTYPE_MID = 1
STEPTYPE_LAST = 2

Q = np.random.uniform(size=(500, 6))
```

Taxi In Gym: Wrapping For Our Code

```python
# wrapper for gym's blackjack environment
def generate_start_step():
    return { 'observation': env.reset(),
             'reward': 0., 'step_type': STEPTYPE_FIRST }


def generate_next_step(step, action):
    obs, reward, done, _, info = env.step(action)
    step_type = STEPTYPE_LAST if done else STEPTYPE_MID
    return { 'observation': obs, 'reward': reward, 'step_type': step_type }
```

Define Behavior And Greedy Policy Functions

```python
def get_greedy_action(step):
    observ = step['observation']
    return np.argmax(Q[observ])


def get_random_action(step):
    return random.randint(0, env.action_space.n-1)


def get_eps_soft_action(step):
```

연습문제: epsilon-greedy 함수를 작성하세요.

Function To Generate An Episode

```python
def generate_episode(policy_func=get_eps_soft_action):
    episode = list()
    actions = list()
    frames = list()

    step = generate_start_step()
    frames.append(env.render(mode='ansi'))
    episode.append(step)

    while step['step_type'] != STEPTYPE_LAST:
        action = policy_func(step)
        step = generate_next_step(step, action)

        frames.append(env.render(mode='ansi'))
        episode.append(step)
        actions.append(action)

    return episode, actions, frames
```

Little change for recording the text-based visualization with every step

Test & Print An Episode

```
from IPython.display import clear_output
from time import sleep

def print_frames(frames):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame)
        sleep(.2)

epi, actions, frames = generate_episode(policy_func=get_random_action)

print_frames(frames)
```

Let us generate an episode using random policy and print every shot by interval of 0.2 secs

Test & Print An Episode

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
```

Q-Learning Algorithm

```
maxiter = 100000
gamma = 1
epsilon = 0.3
lr_rate = 0.8

Q = np.random.uniform(size=(env.observation_space.n, env.action_space.n))

for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_eps_soft_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state, action)
            Q[idx1] += lr_rate * (next_step['reward'] - Q[idx1])
```

```python
for _ in range(maxiter):
    # starting step
    step = generate_start_step()
    action = get_eps_soft_action(step)
    done = False

    while not done:
        next_step = generate_next_step(step, action)

        if next_step['step_type'] == STEPTYPE_LAST:
            state = step['observation']
            idx1 = (state, action)
            Q[idx1] += lr_rate * (next_step['reward']
            done = True
        else:
            best_action = get_greedy_action(next_step)

            state = step['observation']
            next_state = next_step['observation'
            idx1 = (state, action)
            idx2 = (next_state, best_action)
            Q[idx1] += lr_rate * ((next_step['reward'] + gamma * Q[idx2]) - Q[idx1])

            next_action = get_eps_soft_action(step)
            step = next_step
            action = next_action
```

$$\underset{a \in A}{\operatorname{argmax}}\, Q\left(S_{t+1}, a\right)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Successful Episode By The Estimated Policy

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
```

Failed Episode By The Estimated Policy

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
```

연습

- Q-Learning으로 절벽 걷기 (cliffwalking) 문제에 대한 정책 학습
- 위에서 본 택시 예제와 같이 에피소드를 출력해보기