Monte-Carlo RL

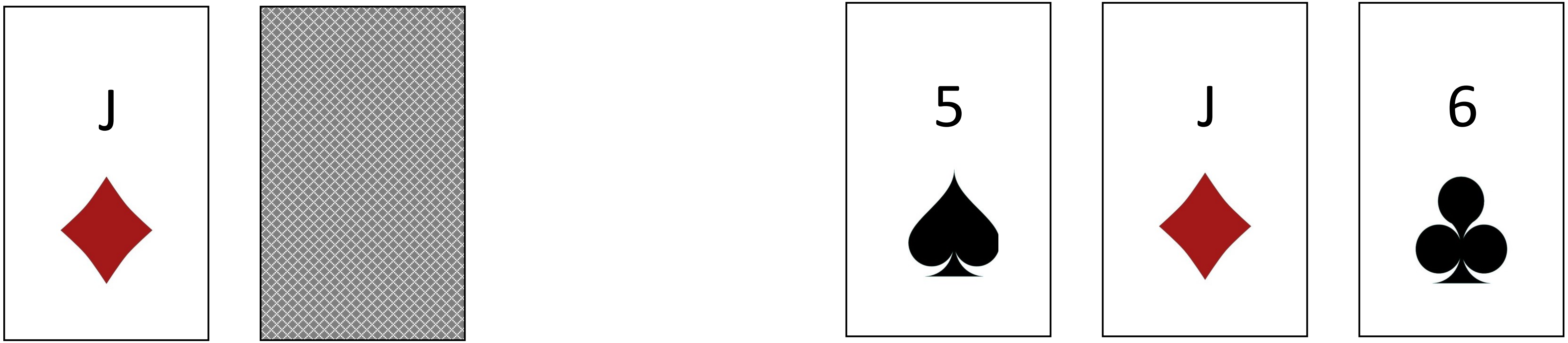
Younghoon Kim
nongaussian@hanyang.ac.kr

Keywords

- Monte-Carlo policy evaluation ★★
- Monte-Carlo policy learning ★★★★
- Epsilon-greedy policy ★★★★★
- On-policy learning vs. off-policy learning ★★★★★
- Behavior policy vs. estimation policy ★★★★

Today's Practice: Blackjack

- Rules
  - Randomly retrieve poker cards to make the sum of scores be closer to 21, without going over 21



<Dealer> <Player>

MDPs

Known MDP	Unknown MDP
전체 환경이 알려져 있음	Unknown
모든 상태에 대한 상태 전환 (state transition) 및 보상 (reward)가 알려져 있어야	실험 혹은 시뮬레이션(환경과의 상호작용으로 인한 일련의 상태, 행동 및 보상을 하나씩 관측가능)이 필요
Model dependent	Model free

Monte-Carlo Policy Evaluation

- Learn the value function v_π from episodes under a given policy π
- Return
  - The sum of discounted rewards in the future
  - i.e., $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{t+T-1} R_T$
- Value function
  - The expected return
  - i.e., $v_\pi(s_i) = E[G_t | S_t = s_i]$

리턴의 기대값을 계산하기 위해
Monte-Carlo 추정법을 사용

Monte-Carlo Estimation

- Goal
  - 주어진 분포 p(x)에서 샘플링하기
  - 기대값 $\sum_{x \in S} p(x) f(x)$을 근사 추정하기
- How to compute
  - 확률 분포 $p(x)$를 따르는 샘플 x 을 뽑을 수 있다면,

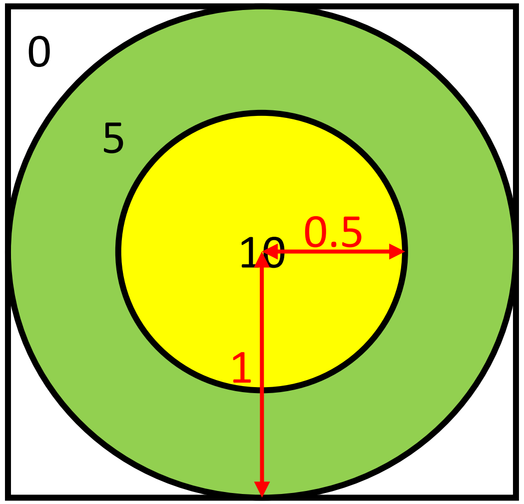
$$\frac{1}{N} \sum_{i=1 \wedge x^{(i)} \in S}^{N} f\left(x^{(i)}\right) \xrightarrow{a.s} \sum_{x \in S} p(x) f(x)$$

An Example: Region Estimate

- Suppose
  - Uniformly throw arrows on the dart board
  - How can you estimate the expected (mean) score?
    - That is, (sum of scores) / (# of throws)
- Analytically,
  - $\sum_{x \in S} p(x) f(x)$
  - $= 10 \cdot \Pr(yellow) + 5 \cdot \Pr(green) + 0 \cdot \Pr(white)$
  - $= 10 \cdot \frac{0.25\pi}{4} + 5 \cdot \frac{(1-0.25)\pi}{4} + 0 \cdot \frac{4-\pi}{4}$

The probability that an arrow hits the yellow circle

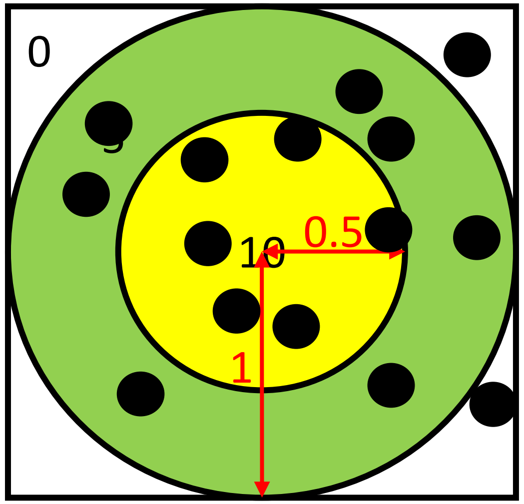
  - $= 1.5625\pi$



An Example: Region Estimate

- Empirically,
  - $\frac{1}{N}\sum_{i=1 \wedge x^{(i)} \in S}^{N} f\left(x^{(i)}\right)$
  - $= \frac{1}{15}(5 + 10 + 10 + 5 + 10 + 5 + 5 + 10 + 10 + 0 + 10 + 5 + 5 + 0 + 5)$
  - $= \frac{1}{15}(6 \cdot 10 + 7 \cdot 5 + 2 \cdot 0)$
  - $= 6.33$
  - $\cong 1.5625\pi$

As increasing N, the estimated expectation will get close to 1.5625π



Monte-Carlo Policy Evaluation

- Our goal of using Monte Carlo estimation is to compute the expected returns

$$E_\pi[G_t|S_t = s_i]$$

$$\simeq \frac{1}{N(s_i)} \sum_{i=1 \wedge S_t^{(\ell)}=s_i}^{N(s_i)} G_t$$

The return at t when $S_t = s_i$

The return G_t is added from a sample episode $A_t, S_{t+1}, \dots, S_T$

The sample $S_t^{(\ell)} = s_i$ is drawn by the probability of $\Pr(A_t, S_{t+1}, \dots, S_T | S_t = s_i)$ in the episode

Monte-Carlo Policy Evaluation

- 상태 s에서의 기대 리턴을 평가하려면,
- 한 에피소드에서 상태 s에 처음 방문한 시간: t
- 카운터 증가: $N(s_t) \leftarrow N(s_t) + 1$
- 총 리턴 증가: $S(s) \leftarrow S(s) + G_t$
- 리턴의 평균을 추정: $V(s) = S(s) / N(s)$

Monte Carlo ES (Exploring Starts) Policy Learning

- Given a discount γ and a policy π as inputs
  - Randomly initialize $q(s_i, a_k), \forall s_i \in S, \forall a_k \in A$
- Repeat
  - Choose S_0 and A_0 randomly
  - Generate an episode starting from S_0 and A_0 following π
  - G ← 0
  - For each t from T-1 to 0
    - $G \leftarrow \gamma G + R_{t+1}$
    - If (S_t, A_t) does not appear in $(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})$
      - $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1, SUM(S_t, A_t) \leftarrow SUM(S_t, A_t) + G$
      - $q(S_t, A_t) = \frac{SUM(S_t)}{N(S_t)}$
  - $\forall s_i \in S, \pi(s_i) = \arg\max_{a \in A} q(s_i, a)$

$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{t+T-1} R_T$

에피소드에서 처음 나오는 샘플 (St, At)만을 통계 값에 사용하기 때문에, ***exploring starts*** 라고 부름

Monte Carlo ES With ϵ-soft Policy (ϵ-greedy)

- Given a discount γ and a policy π as inputs
  - Randomly initialize $q(s_i, a_k), \forall s_i \in S, \forall a_k \in A$
- Repeat
  - Choose S_0 and A_0 randomly and generate an episode starting from S_0 and A_0 following π
  - G ← 0
  - For each t from T-1 to 0
    - $G \leftarrow \gamma G + R_{t+1}$
    - If (S_t, A_t) does not appear in $(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})$
      - $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1, SUM(S_t, A_t) \leftarrow SUM(S_t, A_t) + G$
      - $q(S_t, A_t) = \frac{SUM(S_t)}{N(S_t)}$
  - For all $s_i \in S$
    - $\pi(a, s_i) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(\mathrm{s_i})|} & if\ a = \arg\max_{a \in A} q(\mathrm{s_i}, a) \\ \frac{\epsilon}{|A(s_i)|} & \text{otherwise} \end{cases}$

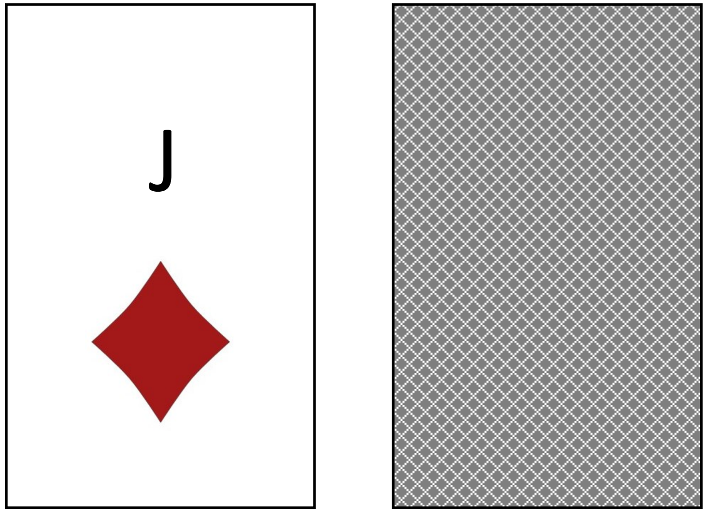
ϵ의 확률로 액션 들 중
하나를 동일 확률로
랜덤하게 선택하기도 함

Blackjack

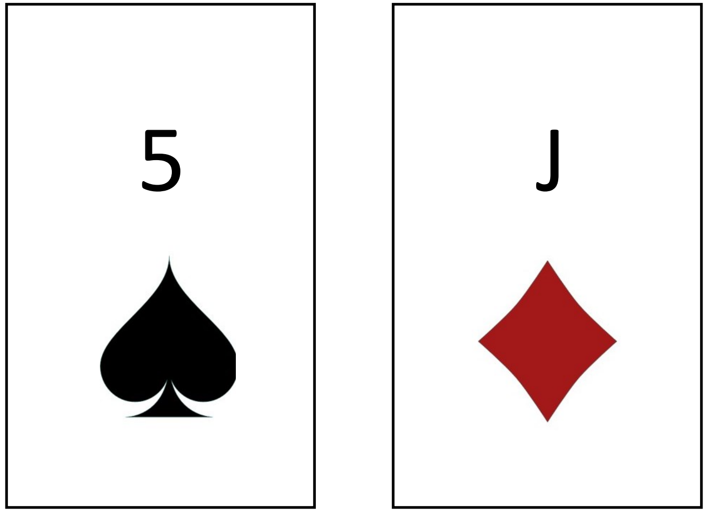
- Object of game
  - 무작위로 포커 카드를 가져와 점수의 합이 21을 넘지 않고 21에 가까워지도록 (카드 추가 요청 가능)
- Card scores
  - 2 ~ 10: 카드 액면 그대로
  - All face cards (J, Q, K): 10으로 친다 (called ten-cards)
  - Ace: 1 또는 11로 계산 가능 (유리하게 선택)

Blackjack

- Rules
  - 1) 게임은 딜러와 플레이어 모두에게 두 장의 카드를 주는 것으로 시작됩니다. 딜러의 카드 한 장은 앞면이 위로 향하게 하고 다른 한 장은 뒷면이 아래로 향하게 합니다.



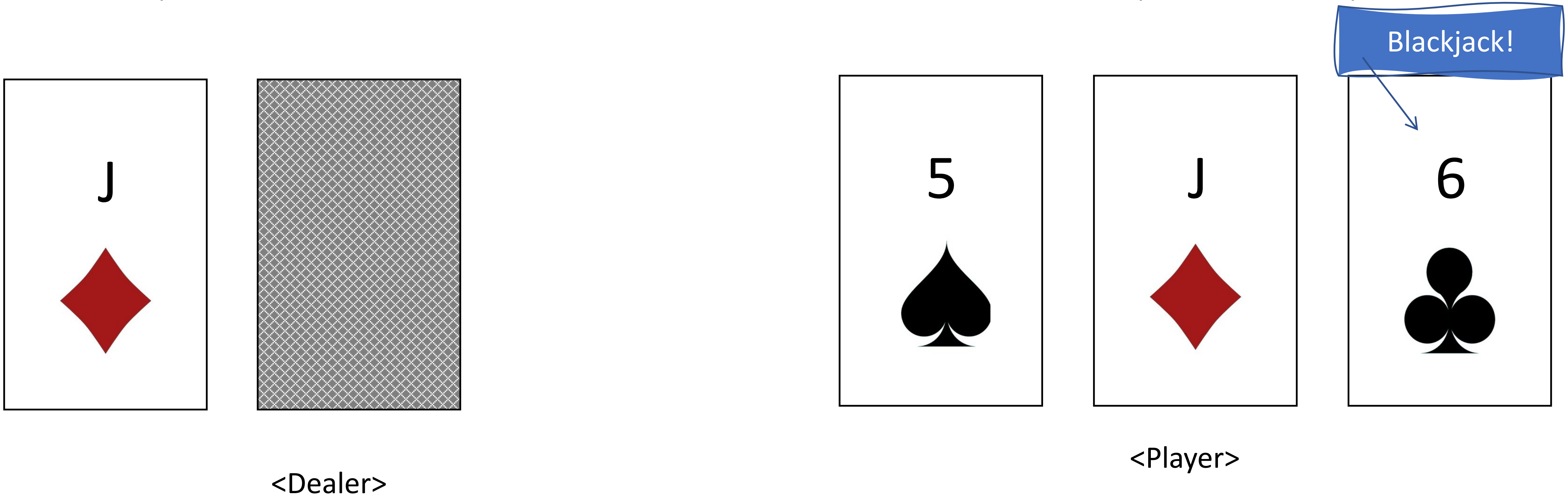
<Dealer>



<Player>

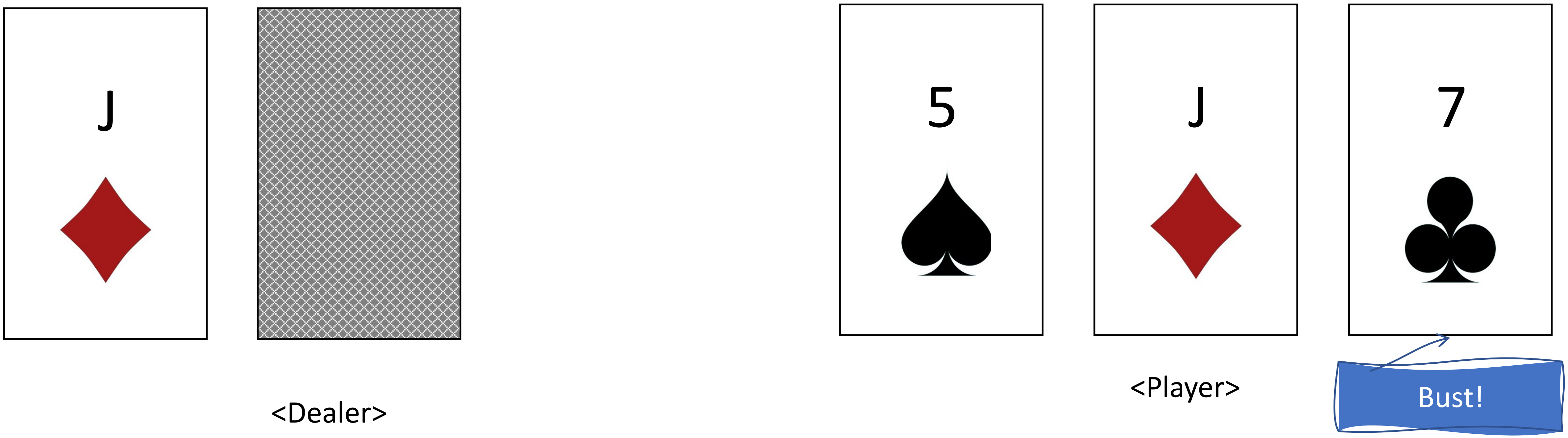
Blackjack

- Rules
  - 2) 플레이어가 21을 가지고 있다면, 딜러의 패를 열어 플레이어가 이기는지 확인합니다.
  - 3) 그렇지 않은 경우, 플레이어는 추가 카드를 요청할 수 있습니다 (‘hit’라고 부름).



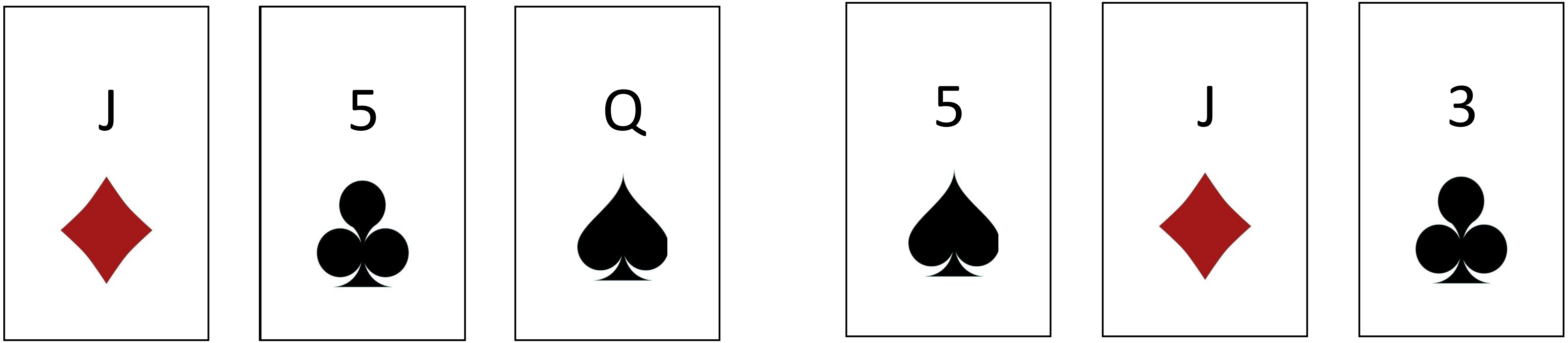
Blackjack

- 2) 플레이어가 추가 카드로 21을 얻으면 블랙잭!
- 3) 그렇지 않은 경우 플레이어는 추가 카드를 다시 요청할 수 있습니다.
  - 합이 21보다 크면 플레이어는 딜러의 패 확인없이 즉시 패배 (버스트).
  - 플레이어는 더 이상 카드를 받고 싶지 않다면 'stay'라고 말합니다.



Blackjack

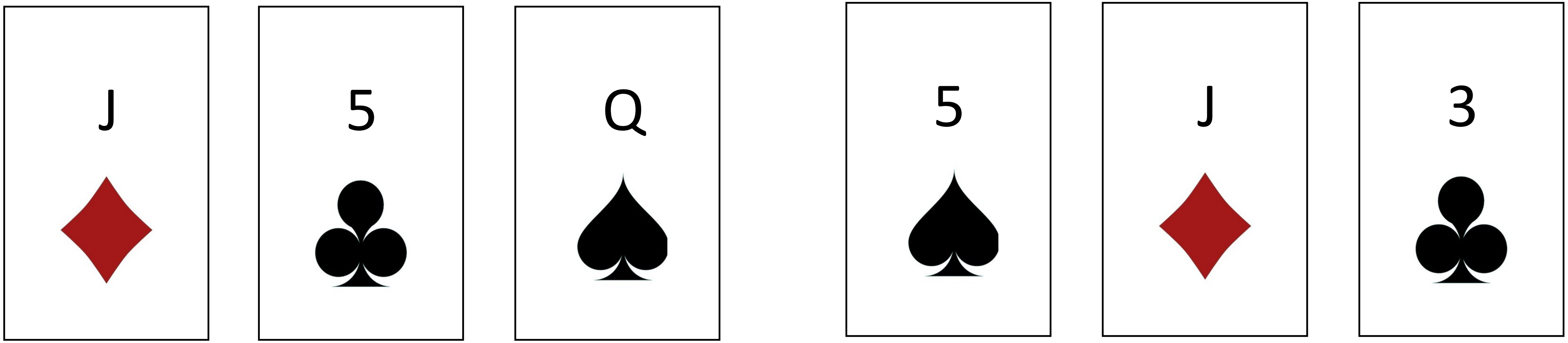
- 4) 플레이어가 ‘stay’하면 딜러의 차례입니다.
  - 앞면이 아래로 향한 카드를 열고,
  - 딜러는 합이 17 이상이 될 때까지 카드를 추가해야 합니다.
  - 딜러는 플레이어를 이기기 위해 ‘hit’하거나 ‘stay’할 선택권이 없습니다.



<Dealer> <Player>

Blackjack

- 5) If the dealer bust, the player wins
  - If both do not exceed 21, **the one closer to 21 wins**



<Dealer> <Player>

Represent As A Finite MDP

- Each game of blackjack is an episode
  - $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
    - S_i: state (how to define a state?)
    - A_i: an action in {Hit, Stay}
    - R_i: reward
      - +1: winning
      - -1: losing
      - 0: tie or within a game

Do not allow split, double and surrender

가정: 카드는 무한한 카드덱에서 샘플링된다고 가정하면 과거 공개된 카드는 아무 상관이 없으며(즉, 카운팅 불가), 따라서 상태를 정의하기에는 현재 점수 합계만 중요합니다.

- State
  - 처음에 플레이어의 점수가 최대 11 점이면 딜러를 이길 방법이 없다!
    - 딜러의 초기 점수도 11이면 딜러는 카드를 한 장 더 요청하여 확실히 승리
    - 따라서 **플레이어의 가능한 점수는** 최소 12점 이상 21점 이하
  - **딜러가 보여주는 카드 한 장의 점수**: 에이스부터 10까지
  - 플레이어가 사용 가능한 **에이스를 보유하고 있는지 여부**
  - ➔ 200가지 상태

Optimal Policy With The Basic Rule

Player hand	Dealer's face-up card									
	2	3	4	5	6	7	8	9	10	A
Hard totals (excluding pairs)										
18–21	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	Us
16	S	S	S	S	S	H	H	Uh	Uh	Uh
15	S	S	S	S	S	H	H	H	Uh	Uh
13–14	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H

Soft totals										
	2	3	4	5	6	7	8	9	10	A
A,9	S	S	S	S	S	S	S	S	S	S
A,8	S	S	S	S	Ds	S	S	S	S	S
A,7	Ds	Ds	Ds	Ds	Ds	S	S	H	H	H
A,6	H	Dh	Dh	Dh	Dh	H	H	H	H	H
A,4–A,5	H	H	Dh	Dh	Dh	H	H	H	H	H
A,2–A,3	H	H	H	Dh	Dh	H	H	H	H	H

S = Stand
H = Hit
Dh = Double (if not allowed, then hit)
Ds = Double (if not allowed, then stand)
SP = Split
Uh = Surrender (if not allowed, then hit)
Us = Surrender (if not allowed, then stand)
Usp = Surrender (if not allowed, then split)

* https://en.wikipedia.org/wiki/Blackjack

Optimal Policy Found By Monte-Carlo ES

<Without Ace>

Player \ Dealer	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	1
17	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	0
15	1	1	1	1	1	1	1	1	1	0
14	1	1	1	1	1	0	0	0	1	0
13	1	1	1	1	1	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0	0

<With Ace>

Player \ Dealer	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	1
17	0	1	0	1	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0

MC Learning For Blackjack: Setting

```
import numpy as np
import random
import itertools

# blackjack

# observation (=state):
# triple ( integer, integer, integer )
# 1. integer: the player's score (12 ~ 21)
# 2. integer: the dealer's card score of upside (1 ~ 10)
# 3. integer: 1 if the player has at least an ace, and 0 otherwise

# action
# 0: hit
# 1: stay
# doesn't allow double down, surrender and split

# step types
STEPTYPE_FIRST = 0
STEPTYPE_MID = 1
```

- Observation is represented by three integers
- Action is 0 (hit) or 1 (stay)

```
# 3. integer: 1 if the player has at least an ace, and 0 otherwise

# action
# 0: hit
# 1: stay
# doesn't allow double down, surrender and split

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

cardset = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10 ]
deck = None

def shuffle_deck():
        global deck
        # card deck (we don't care the suite, but, for gui game in future) - 3 sets
        deck = \
          list(itertools.product(range(4), cardset)) \
          + list(itertools.product(range(4), cardset)) \
          + list(itertools.product(range(4), cardset))

    random.shuffle(deck)
shuffle_deck()
```

Three types of steps (1st, mid, last)

Shuffle with three sets of 52 cards

MC Learning For Blackjack: Algorithm

```
# monte-carlo policy learning
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)
```

Values for MC estimation, which are the tensors shaped by (10, 10, 2, 2)

Generate an episode (we'll see in the next slides)

```python
# monte-carlo policy learning
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]
```

Eploring-start (ES) 전략에 따라 상태와
행동의 쌍이 에피소드에서 처음
나타나면 통계를 업데이트

Environment: Generating An Episode

Github 홈페이지에서 다운로드: example-mc-black-env.ipynb

Reset the env and return the first step

Initialize the card deck

```
# environment parameters
dealer = None # dealer's hands
player = None # player's hands

# reset the environment
def generate_start_step():
    global dealer, player
    shuffle_deck()
    dealer = [ deck.pop(), deck.pop() ]
    player = [ deck.pop(), deck.pop() ]
    dealer_score = dealer[0][1]

    if player[0][1] == 1 and player[1][1] == 1:
        # if player gets double ace, the second one is counted as 1
        player_score = 12
        has_ace = 1
    elif player[0][1] == 1:
        player_score = 11 + player[1][1]
        has_ace = 1
```

```
dealer = [ deck.pop(), deck.pop() ]
player = [ deck.pop(), deck.pop() ]
dealer_score = dealer[0][1]

if player[0][1] == 1 and player[1][1] == 1:
    # if player gets double ace, the second one is counted as 1
    player_score = 12
    has_ace = 1
elif player[0][1] == 1:
    player_score = 11 + player[1][1]
    has_ace = 1
elif player[1][1] == 1:
    player_score = 11 + player[0][1]
    has_ace = 1
else:
    player_score = player[0][1] + player[1][1]
    while player_score < 12:
        player.append(deck.pop())
        player_score += player[-1][1]
    has_ace = 0

# 1st step
return { 'observation': (player_score, dealer_score, has_ace),
    'reward': 0., 'step_type': STEPTYPE_FIRST }
```

When at least a card is ace,

If the score is under 12, draw cards until getting at least 12

Environment: Generating An Episode

Given the current step and the action chosen, return the next step

```
# returns a step, which is a dictionary { 'observation', 'reward', 'step_type' }
def generate_next_step(step, action):
        global player, dealer
        player_score, dealer_open, has_ace = step['observation']
        # has_ace is used to check if the player has
        # the option to count an ace as 1
        game_stop = False
        busted = False

        # with hit, get a card
        if action == 0:
                # hit - take an additional card
                player.append(deck.pop())
                # note that an additional ace should be counted as 1
                player_score += player[-1][1]

                # if blackjack or bust, the game stops
                if player_score == 21:
                        game_stop = True
                elif player_score > 21:
                        # if busted but has an ace, the ace is counted as 1
                        # and has_ace becomes false since already used
```

With action 0 (=hit), take an additional card

```python
# with hit, get a card
if action == 0:
        # hit - take an additional card
        player.append(deck.pop())
        # note that an additional ace should be counted as 1
        player_score += player[-1][1]

        # if blackjack or bust, the game stops
        if player_score == 21:
                game_stop = True
        elif player_score > 21:
                # if busted but has an ace, the ace is counted as 1
                # and has_ace becomes false since already used
                if has_ace == 1:
                        player_score -= 10
                        has_ace = 0
                else:
                        game_stop = True
                        busted = True
# with stay, game_stop
else:
        game_stop = True

# if busted, immediately the player loses
if busted:
        return { 'observation': (player_score, dealer_open, has_ace),
              'reward': -1., 'step_type': STEPTYPE_LAST }

# now, if game_stop, it's dealer's turn & game stop
if game_stop:
        dealer_has_ace = False
        dealer_busted = False
```

With the score sum, determine if game is done

If the player has ace, it can be counted as 1 if busted

If busted, return the last step with reward -1 without checking the dealer's hand

```python
# now, if game_stop, it's dealer's turn & game stop
if game_stop:
    dealer_has_ace = False
    dealer_busted = False

    # examine dealer's hands
    if dealer[0][1] == 1 and dealer[1][1] == 1:
        dealer_score = 12.
        dealer_has_ace = True
    elif dealer[0][1] == 1:
        dealer_score = 11. + dealer[1][1]
        dealer_has_ace = True
    elif dealer[1][1] == 1:
        dealer_score = 11. + dealer[0][1]
        dealer_has_ace = True
    else:
        dealer_score = dealer[0][1] + dealer[1][1]
        dealer_has_ace = False

    # the dealer takes cards until the score is at least 17
    while dealer_score < 17:
        dealer.append(deck.pop())
        dealer_score += dealer[-1][1]

        # if busted but has an ace, the ace is counted as 1
        if dealer_score > 21:
            if dealer_has_ace:
                dealer_score -= 10
                dealer_has_ace = False
            else:
                dealer_busted = True
```

Compute the dealer’s score

Dealers have no choice but to get cards until score sum is at least 17

```python
            # if busted but has an ace, the ace is counted as 1
            if dealer_score > 21:
                if dealer_has_ace:
                    dealer_score -= 10
                    dealer_has_ace = False
                else:
                    dealer_busted = True

        # compute the reward
        if dealer_busted:
            reward = 1.
        else:
            if player_score > dealer_score:
                reward = 1.
            elif player_score < dealer_score:
                reward = -1.
            else:
                reward = 0.

        return { 'observation': (player_score, dealer_score, has_ace),
              'reward': reward, 'step_type': STEPTYPE_LAST }

    # continue (i.e., not game_stop)
    else:
        return { 'observation': (player_score, dealer_open, has_ace),
              'reward': 0., 'step_type': STEPTYPE_MID }
```

Compute the reward & return the last step

Return the step for continuing the game

ϵ-Soft Greedy Policy

```python
import random

epsilon = 0.01

def get_eps_soft_action(step):
    assert(step['observation'][0] >= 12 and step['observation'][0] <= 21)
    observ = step['observation']
    idx = (observ[0] - 12, observ[1] - 1, observ[2])

    # epsilon-soft greedy policy
    if random.random() < epsilon:
        return 1 if Q[idx][0] > Q[idx][1] else 0
    else:
        return 1 if Q[idx][0] < Q[idx][1] else 0
```

Generating An Episode

```
def generate_episode():
    episode = list()
    actions = list()
    step = generate_start_step()
    episode.append(step)

    while step['step_type'] != STEPTYPE_LAST:
        action = get_eps_soft_action(step)
        step = generate_next_step(step, action)
        episode.append(step)
        actions.append(action)

    return episode, actions
```

Misc

```
# return true if (observ, action) exists in epi
def in_episode(epi, observ, action):
    for s, a in zip(*epi):
        if s['observation'] == observ and a == action:
            return True
    return False
```

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]
```

Print Out The Policy

```
import pandas as pd

# without ace
wo_ace = pd.DataFrame(np.zeros((10, 10)),
            columns = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            index = [21, 20, 19, 18, 17, 16, 15, 14, 13, 12], dtype='int32')

for row in range(10):
        for col in range(10):
            v = 1 if Q[row, col, 0, 0] < Q[row, col, 0, 1] else 0
            wo_ace.loc[row + 12, col + 1] = v
```

Optimal Policy Found By Monte-Carlo ES

<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	1
17	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	0
15	1	1	1	1	1	1	1	1	1	0
14	1	1	1	1	1	0	0	0	1	0
13	1	1	1	1	1	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0	0

<With 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	1
17	0	1	0	1	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0

Off-Policy Learning

On-Policy and Off-Policy

On-Policy Methods

- Evaluate and improve the policy that is used to make decisions

Off-Policy Methods

- Evaluate and improve a policy different from that used to generate the data

On-Policy vs. Off-Policy

- On-policy
  - Learning fast
  - But may miss the best policy in a long run

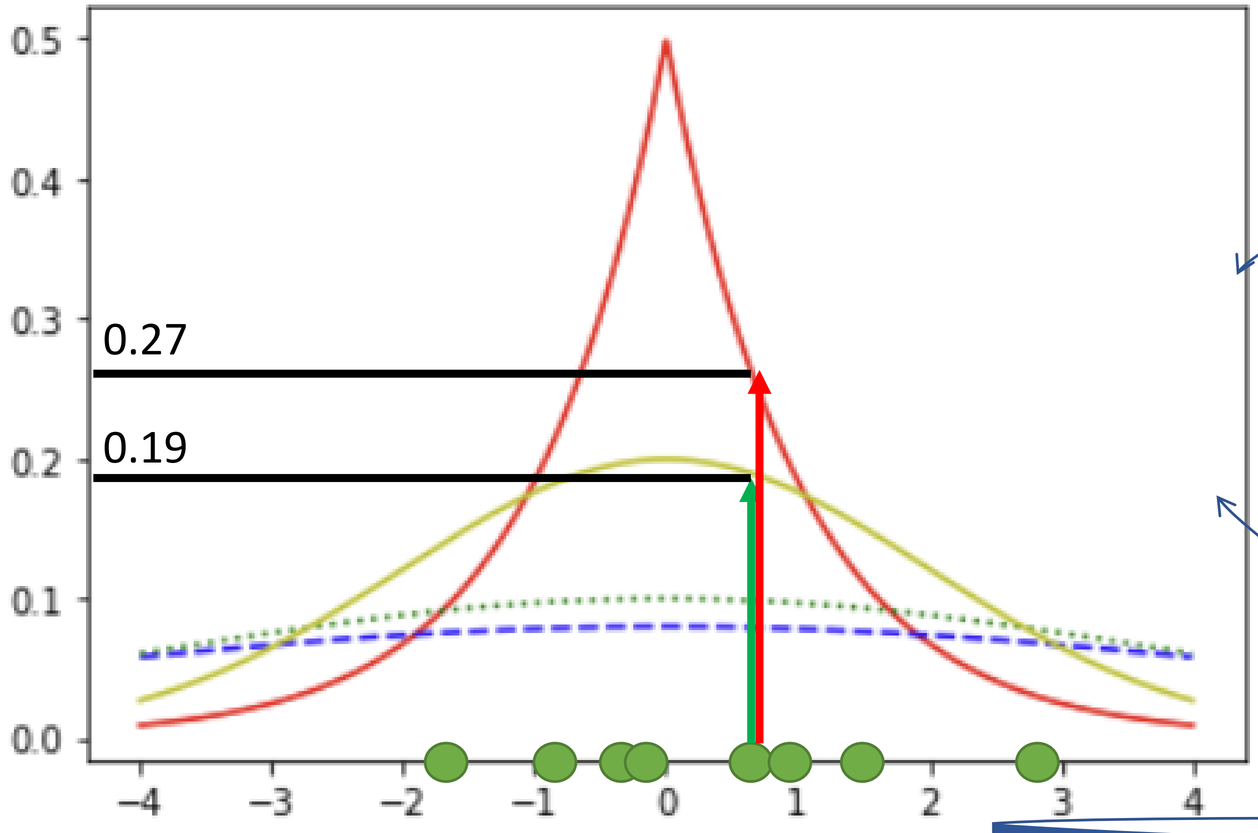
On-Policy vs. Off-Policy

- Off-policy
  - Learning slow
  - Explore diverse actions for finding the best policy

Off-Policy Learning

- Behavior policy
  - 환경을 통해 에피소드를 생성할 때 사용하는 policy distribution
- Estimation policy
  - The policy we want to learn
- Most of off-policy learning uses *important sampling*

Example: $\mathrm{E}[x^2]$ Following Laplace Distribution



<u>Question:</u>
녹색 분포에 따라 x를 샘플링하되(쉬움),
빨간색 분포에 따라 x의 x^2의 평균을 계산
(빨간색 분포로 샘플링은 어려움)

녹색 분포에 따라 0.8를 샘플 뽑았으므로 평균을 계산할 때는 0.64에 $\frac{0.27}{0.19}$의 가중치를 부여

빨간색 분포에 따라 0.8를 샘플링하면 0.64를 합산하여 평균을 구함

Revisit: On-Policy Prediction

- Note that
  - Our goal of using Monte Carlo estimation is to compute the expected returns

$$E_\pi[G_t | S_t = s_i]$$

$$\simeq \frac{1}{N(s_i)} \sum_{i=1 \wedge S_t^{(\ell)} = s_i}^{N(s_i)} G_t$$

The return at t when $S_t = s_i$

Recall that the return G_t is added from a sample episode $A_t, S_{t+1}, \dots, S_T$

The sample $S_t^{(\ell)} = s_i$ which results in G_t is drawn by the probability of $\Pr(A_t, S_{t+1}, \dots, S_T | S_t = s_i)$ in the episode

What if we sample an episode $S_0, A_0, S_1, \dots, S_t, A_t, S_{t+1}, \dots, S_T$ following different policy?

Off-Policy Prediction Using Importance Sampling

- Note that
  - Our goal of using Monte Carlo estimation is to compute the expected returns

$$E_\pi[G_t | S_t = s_i]$$

$$\simeq \sum_{\langle S_t, \dots, S_T \rangle^{(\ell)} \in E \wedge S_t^{(\ell)} = s_i}^{N(s_i)} \frac{Pr_\pi\left(S_t^{(\ell)} = s_i\right)}{Pr_b\left(S_t^{(\ell)} = s_i\right)} G_t$$

The probability we get the (future) return G_t when $S_t = s_i$ in an episode $S_t, A_t, S_{t+1}, \dots, S_T$ generated using a policy π

Among all possible episodes, with the ℓ-th sample episode episode $\langle S_t, A_t, S_{t+1}, \dots, S_T \rangle^{(\ell)}$

The probability we get the (future) return G_t when $S_t = s_i$ in an episode $S_t, A_t, S_{t+1}, \dots, S_T$ generated using a policy b

Off-Policy Prediction Using Importance Sampling

- Let
  - π be a greedy policy and b be a soft policy
- Consider t from T to 0
  - The ratio of probabilities to have the final state $S_T = s_i$

$$\frac{Pr_\pi(S_T = s_i)}{Pr_b(S_T = s_i)} = 1$$

  - At T-1, it is

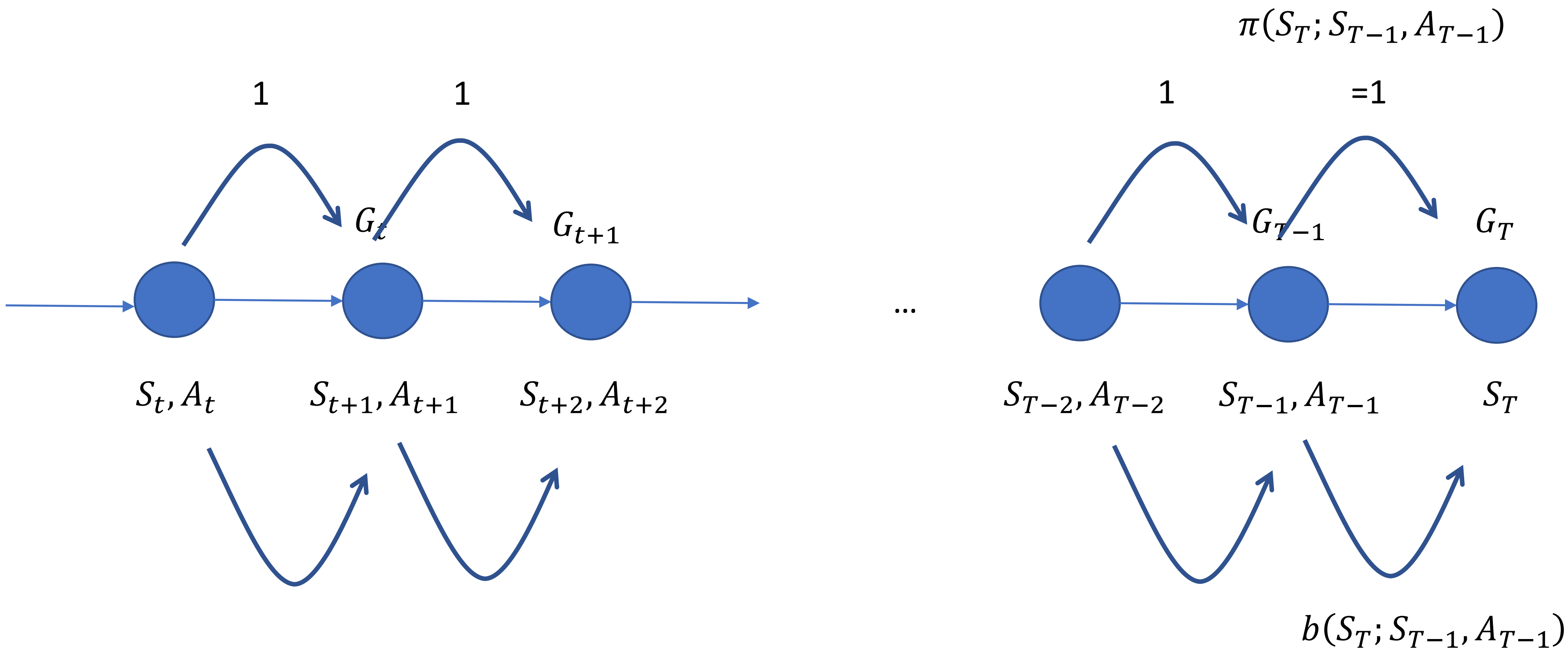
$$\frac{Pr_\pi(S_{T-1} = s_i)}{Pr_b(S_{T-1} = s_i)} = \frac{1}{b(A_{T-1}, S_{T-1})}$$

π is greedy and thus the chance to select the action A_T with $S_{T-1} = s_i$ is simply 1

  - At t, it is

$$\frac{Pr_\pi(S_t = s_i)}{Pr_b(S_t = s_i)} = \frac{1}{\prod_{\ell=t}^{T-1} b(A_\ell, S_\ell)}$$

Off-Policy Prediction Using Importance Sampling



Off-Policy Markov Carlo Control

- Repeat
  - b ← an arbitrary soft policy
  - Choose S_0 and A_0 randomly and generate an episode starting from S_0 and A_0 following b
  - G ← 0, W ← 1
  - For each t from T-1 to 0
    - $G \leftarrow \gamma G + R_{t+1}$
    - If (S_t, A_t) does not appear in $(S_0, A_0), \dots, (S_{t-1}, A_{t-1})$
      - $N(S_t) \leftarrow N(S_t) + \mathrm{W}, SUM(S_t) \leftarrow SUM(S_t) + \mathrm{W} \cdot G$
      - $q(S_t, A_t) = \frac{SUM(S_t)}{N(S_t)}$
    - If $A_t \neq \pi(S_t)$ then, exit the For each loop
    - $W \leftarrow W \cdot \frac{1}{b(A_t, S_t)}$
  - For all $s_i \in S$
    - $\pi(s_i) = \arg\max_{a \in A} q(s_i, a)$

Weighted average

Greedy update

두 정책이 같은 상태, 행동의 열 $S_t, A_t, S_{t+1}, \dots, S_T$ 을 만드는 동안에만 유효

Play Blackjack With Off-Policy Learning

```
behavior_prob_hit = 0.6

# choose hit(=0) with chance 0.6 or stay(=1) with chance 0.4
def get_random_action(step):
    # epsilon-soft greedy policy
    if random.random() < behavior_prob_hit:
        return 0
    else:
        return 1
```

Define the behavior policy function: simply choose hit (=0) with probability 0.6 for all states

Play Blackjack With Off-Policy Learning

```
def get_greedy_action(step):
    observ = step['observation']
    idx = (observ[0] - 12, observ[1] - 1, observ[2])
    return 0 if Q[idx][0] > Q[idx][1] else 1
```

Greedy policy function that answers
$\arg\max_{a \in A} Q(s_i, a), \forall s_i \in S$

Environment- Generating An Episode

Modify *generate_episode* function

```
def generate_episode(policy_func=get_eps_soft_action):
    episode = list()
    actions = list()
    step = generate_start_step()
    episode.append(step)

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

    return episode, actions
```

The same as the previously defined one except it takes the policy function as an input

Blackjack By Off-Policy Learning- Algorithm

```
# monte-carlo off-policy learning
maxiter = 1000000
gamma = 1

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):
    steps, actions = generate_episode(policy_func=get_random_action)
    G = 0.
    W = 1.
    last_step = steps.pop()

    while len(steps) > 0:
        G = gamma * G + last_step['reward']
        last_step = steps.pop()
        last_action = actions.pop()

        # exploring-start estimation
        observ = last_step['observation']
```

애피소드를 생성할 때는
랜덤 액션 선택 정책을
사용함

```python
W = 1.
last_step = steps.pop()

while len(steps) > 0:
        G = gamma * G + last_step['reward']
        last_step = steps.pop()
        last_action = actions.pop()

        # exploring-start estimation
        observ = last_step['observation']
        idx = (observ[0] - 12, observ[1] - 1, observ[2], last_action)

        if not in_episode((steps, actions), observ, last_action):
                N[idx] += W
                SUM[idx] += W * G
                Q[idx] = SUM[idx] / N[idx]

        if last_action != get_greedy_action(last_step):
                break

        if last_action == 0:
                W = W / behavior_prob_hit
        else:
                W = W / (1. - behavior_prob_hit)
```

리턴 가중 평균

$$W \leftarrow W \cdot \frac{1}{b(A_t, S_t)}$$

Practice

- Monte-Carlo Off-Policy Learning를 이용해 블랙잭 게임을 학습하고 최적의 policy matrix를 출력하세요.

	1	2	3	4	5	6	7	8	9	10
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	1
17	1	1	1	1	1	1	1	1	1	1
16	0	1	1	1	1	1	0	0	0	0
15	0	1	1	1	1	1	0	0	0	0
14	0	1	1	1	1	1	0	0	0	0
13	0	1	0	1	1	1	0	0	0	0
12	0	0	1	1	1	0	0	0	0	0

	1	2	3	4	5	6	7	8	9	10
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	0	0	1	0	1	1	1	0
17	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0

Policy By Monte-Carlo Off-Policy 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	1	1
18	1	1	1	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	0	0	1	0	0
15	1	1	1	1	1	0	0	0	0	0
14	1	1	1	1	1	0	0	0	0	0
13	1	1	1	1	1	0	0	0	0	0
12	1	1	1	1	0	0	0	0	0	0

<With 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	0	1	1	1	1	1	1
17	0	0	0	0	0	1	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0