

Blackjack with Q-Learning

```
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
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()
```

```
# the table of policy to access with the three indices
Q = np.random.uniform(size=(10, 10, 2, 2))
```

```
# 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
    elif player[1][1] == 1:
        player_score = 11 + player[0][1]
        has_ace = 1
    else:
```

Blackjack with Q-Learning

```

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 }

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

# 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 - retrieve 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

        # examine dealer's hands
        if dealer[0][1] == 1 and dealer[1][1] == 1:

```

Blackjack with Q-Learning

```

        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 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
else:
    return { 'observation': (player_score, dealer_open, has_ace),
            'reward': 0., 'step_type': STEPTYPE_MID }

```

```

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

```

```

test = generate_episode()
test

```

```

([{'observation': (14, 6, 0), 'reward': 0.0, 'step_type': 0},
 {'observation': (14, 20, 0), 'reward': -1.0, 'step_type': 2}],
 [1])

```

```

# 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

```

Blackjack with Q-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

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

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] = 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] = 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

```

```

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

```

wo_ace

	1	2	3	4	5	6	7	8	9	10
21	1	1	1	1	1	1	1	1	1	1

Blackjack with Q-Learning

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

```
# without ace
w_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, 1, 0] < Q[row, col, 1, 1] else 0
        w_ace.loc[row + 12, col + 1] = v
```

w_ace

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