



Московский Государственный Технический Университет имени Н.Э. Баумана  
Факультет Информатика и системы управления

**Кафедра ИУ-5 «Системы обработки информации и управления»**

**Отчёт по рубежному контролю № 2**

**По дисциплине**  
**«Методы Машинного Обучения»**

Выполнил студент: Лю Ань  
Группа ИУ5И-22М

**Москва 2023г**

## Тема: Методы обучения с подкреплением.

Для одного из алгоритмов временных различий, реализованных Вами в соответствующей лабораторной работе:

SARSA

Q-обучение

Двойное Q-обучение

осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

**Я выбрал алгоритм Q-обучение, у которого гиперпараметры: Learning rate, Коэффициент дисконтирования, Количество эпизодов**

Исходный код:

```
###
import numpy as np
import matplotlib.pyplot as plt
import gym
from tqdm import tqdm

# ***** БАЗОВЫЙ АГЕНТ *****
all_reward=[]
parameter=[]
class BasicAgent:
    '''
    Базовый агент, от которого наследуются стратегии обучения
    '''

    # Наименование алгоритма
    ALGO_NAME = '---'

    def __init__(self, env, eps=0.1):
        # Среда
        self.env = env
        # Размерности Q-матрицы
        self.nA = env.action_space.n
        self.nS = env.observation_space.n
        #и сама матрица
```

```

self.Q = np.zeros((self.nS, self.nA))
# Значения коэффициентов
# Порог выбора случайного действия
self.eps=eps
# Награды по эпизодам
self.episodes_reward = []

def print_q(self):
    # print('Вывод Q-матрицы для алгоритма ', self.ALGO_NAME)
    # print(self.Q)
    all_reward.append(np.sum(self.Q))
    print('Суммарная награда:', np.sum(self.Q))

def get_state(self, state):
    """
    Возвращает правильное начальное состояние
    """
    if type(state) is tuple:
        # Если состояние вернулось с виде кортежа, то вернуть только номер
        # состояния
        return state[0]
    else:
        return state

def greedy(self, state):
    """
    <<Жадное>> текущее действие
    Возвращает действие, соответствующее максимальному Q-значению
    для состояния state
    """
    return np.argmax(self.Q[state])

def make_action(self, state):
    """
    Выбор действия агентом
    """
    if np.random.uniform(0,1) < self.eps:
        # Если вероятность меньше eps
        # то выбирается случайное действие
        return self.env.action_space.sample()
    else:

```

```

        # иначе действие, соответствующее максимальному Q-значению
        return self.greedy(state)

def draw_episodes_reward(self):
    # Построение графика наград по эпизодам
    fig, ax = plt.subplots(figsize = (15,10))
    y = self.episodes_reward
    x = list(range(1, len(y)+1))
    plt.plot(x, y, '-', linewidth=1, color='green')
    plt.title('Награды по эпизодам')
    plt.xlabel('Номер эпизода')
    plt.ylabel('Награда')
    plt.show()

def learn():
    '''
    Реализация алгоритма обучения
    '''
    pass

# ***** Q-обучение
# *****

class QLearning_Agent(BasicAgent):
    '''
    Реализация алгоритма Q-Learning
    '''
    # Наименование алгоритма
    ALGO_NAME = 'Q-обучение'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=1000):
        # Вызов конструктора верхнего уровня
        super().__init__(env, eps)
        # Learning rate
        self.lr=lr
        # Коэффициент дисконтирования
        self.gamma = gamma
        # Количество эпизодов
        self.num_episodes=num_episodes
        # Постепенное уменьшение eps
        self.eps_decay=0.00005
        self.eps_threshold=0.01

```

```

def learn(self):
    """
    Обучение на основе алгоритма Q-Learning
    """
    self.episodes_reward = []
    # Цикл по эпизодам
    for ep in tqdm(list(range(self.num_episodes))):
        # Начальное состояние среды
        state = self.get_state(self.env.reset())
        # Флаг штатного завершения эпизода
        done = False
        # Флаг нештатного завершения эпизода
        truncated = False
        # Суммарная награда по эпизоду
        tot_rew = 0

        # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора
        # действия
        if self.eps > self.eps_threshold:
            self.eps -= self.eps_decay

        # Проигрывание одного эпизода до финального состояния
        while not (done or truncated):

            # Выбор действия
            # В SARSA следующее действие выбиралось после шага в среде
            action = self.make_action(state)

            # Выполняем шаг в среде
            next_state, rew, done, truncated, _ = self.env.step(action)

            # Правило обновления Q для SARSA (для сравнения)
            # self.Q[state][action] = self.Q[state][action] + self.lr * \
            #     (rew + self.gamma * self.Q[next_state][next_action] -
            self.Q[state][action])

            # Правило обновления для Q-обучения
            self.Q[state][action] = self.Q[state][action] + self.lr * \
                (rew + self.gamma * np.max(self.Q[next_state]) -
            self.Q[state][action])

            # Следующее состояние считаем текущим
            state = next_state
            # Суммарная награда за эпизод
            tot_rew += rew

```

```

        if (done or truncated):
            self.episodes_reward.append(tot_rew)

def play_agent(agent):
    '''
    Проигрывание сессии для обученного агента
    '''
    env2 = gym.make('CliffWalking-v0', render_mode='human')
    state = env2.reset()[0]
    done = False
    while not done:
        action = agent.greedy(state)
        next_state, reward, terminated, truncated, _ = env2.step(action)
        env2.render()
        state = next_state
        if terminated or truncated:
            done = True

#%% Цикл ,

def run_q_learning():
    env = gym.make('CliffWalking-v0')
    for i in np.arange(0.01,0.2,0.02):
        for j in np.arange(0.95,1,0.1):
            for n in np.arange(100,2001,200):
                agent = QLearning_Agent(env,lr=i, gamma=j, num_episodes=n)
                agent.learn()
                agent.print_q()
                #agent.draw_episodes_reward()
                parameter.append([i,j,n])

def main():
    run_q_learning()
    print(all_reward)
    print('Максимальная награда:',np.max(all_reward),'Значения гиперпараметров(lr, gamma, num_episodes):',parameter[np.argmax(np.max(all_reward))])
    #play_agent(agent)

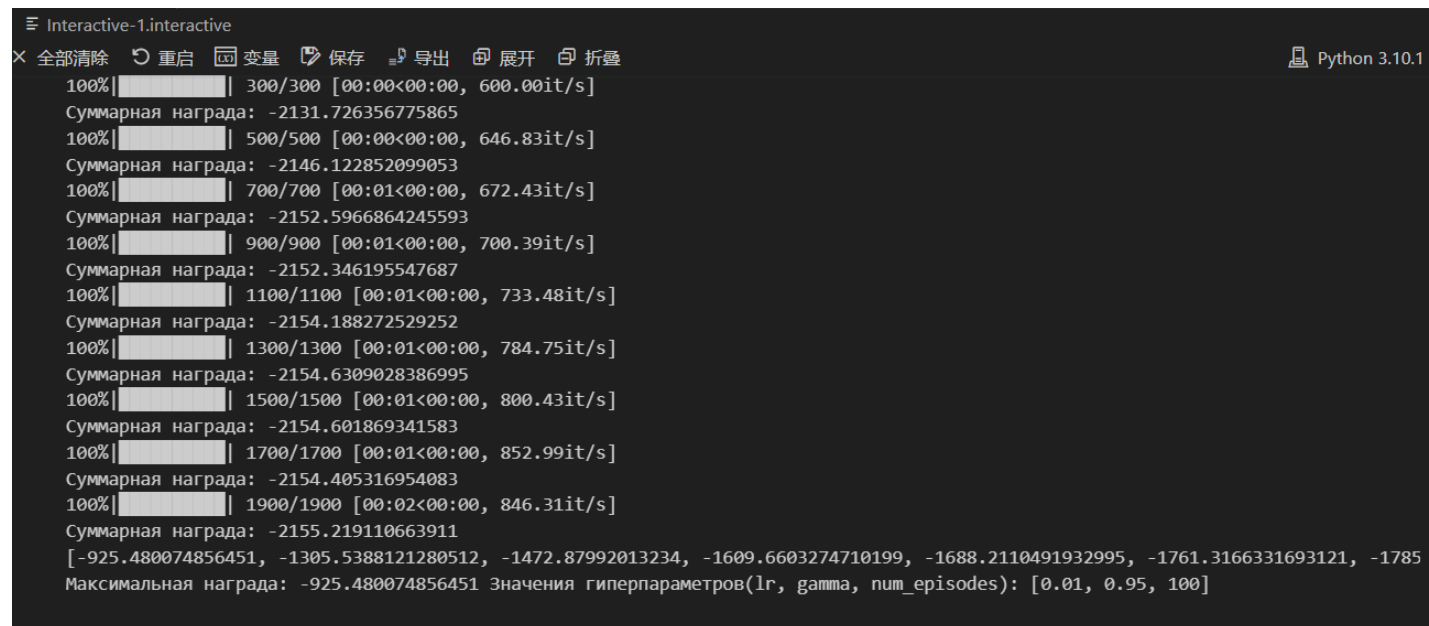
if __name__ == '__main__':
    main()
# %%

```

Результат:

```
def run_q_learning():
    env = gym.make('CliffWalking-v0')
    for i in np.arange(0.01,0.2,0.02):
        for j in np.arange(0.95,1,0.1):
            for n in np.arange(100,2001,200):
                agent = QLearning_Agent(env,lr=i, gamma=j, num_episodes=n)
                agent.learn()
                agent.print_q()
                #agent.draw_episodes_reward()
                parameter.append([i,j,n])
```

Всего 500 разных комбинаций значений гиперпараметров.



```
Interactive-1.interactive
x 全部清除 重启 变量 保存 导出 展开 折叠 Python 3.10.1
100%|██████████| 300/300 [00:00<00:00, 600.00it/s]
Суммарная награда: -2131.726356775865
100%|██████████| 500/500 [00:00<00:00, 646.83it/s]
Суммарная награда: -2146.122852099053
100%|██████████| 700/700 [00:01<00:00, 672.43it/s]
Суммарная награда: -2152.5966864245593
100%|██████████| 900/900 [00:01<00:00, 700.39it/s]
Суммарная награда: -2152.346195547687
100%|██████████| 1100/1100 [00:01<00:00, 733.48it/s]
Суммарная награда: -2154.188272529252
100%|██████████| 1300/1300 [00:01<00:00, 784.75it/s]
Суммарная награда: -2154.6309028386995
100%|██████████| 1500/1500 [00:01<00:00, 800.43it/s]
Суммарная награда: -2154.601869341583
100%|██████████| 1700/1700 [00:01<00:00, 852.99it/s]
Суммарная награда: -2154.405316954083
100%|██████████| 1900/1900 [00:02<00:00, 846.31it/s]
Суммарная награда: -2155.219110663911
[-925.480074856451, -1305.5388121280512, -1472.87992013234, -1609.6603274710199, -1688.2110491932995, -1761.3166331693121, -1785.480074856451]
Максимальная награда: -925.480074856451 Значения гиперпараметров(lr, gamma, num_episodes): [0.01, 0.95, 100]
```

Оптимальное из них:

Максимальная суммарная награда: -925.480074856451

Значения гиперпараметров (lr, gamma, num\_episodes): [0.01, 0.95, 100]