Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика, искусственный интеллект и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Рубежный контроль №2

по дисциплине «Методы машинного обучения»

по теме «Методы обучения с подкреплением»

Выполнил:
студент группы № ИУ5-21М
Камалов М.Р.
подпись, дата

Проверил:
Гапанюк Ю.Е.
подпись, дата

2023 г.

Задание:

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

- SARSA
- Q-обучение
- **Двойное Q-обучение**

осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

Текст программы

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

# ********************************** 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 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(self):
        '''
        Реализация алгоритма обучения
        '''
        pass


# ********************************** Двойное Q-обучение
*****************************************

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

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=100):
        # Вызов конструктора верхнего уровня
        super().__init__(env, eps)
        # Learning rate
        self.lr = lr
        # Вторая матрица
        self.Q2 = np.zeros((self.nS, self.nA))
        # 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 print_q(self):
        all_reward.append(np.sum(self.Q))
        print('Суммарная награда:', np.sum(self.Q), f"lr = {self.lr:.3f} gamma =
{self.gamma:.3f} eps = {self.eps:.3f}")

    def greedy(self, state):
        '''
        <<Жадное>> текущее действие
        Возвращает действие, соответствующее максимальному Q-значению
        для состояния state
        '''
        temp_q = self.Q[state] + self.Q2[state]
        return np.argmax(temp_q)

    def learn(self):
        '''
        Обучение на основе алгоритма Double 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

            # 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)

                if np.random.rand() < 0.5:
                    # Обновление первой таблицы
                    self.Q[state][action] = self.Q[state][action] + self.lr * \
                                            (rew + self.gamma *
self.Q2[next_state][np.argmax(self.Q[next_state])] -
                                             self.Q[state][action])
                else:
                    # Обновление второй таблицы
                    self.Q2[state][action] = self.Q2[state][action] + self.lr * \
                                             (rew + self.gamma *
self.Q[next_state][np.argmax(self.Q2[next_state])] -
                                              self.Q2[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('Taxi-v3')
    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


def run_q_learning():
    env = gym.make('Taxi-v3')
    lr_list = np.linspace(0.0005, 0.005, num=5)
    gamma_list = np.linspace(0.9, 1, num=5)
    eps_list = np.linspace(0.05, 0.9, num=9)
    for l in tqdm(lr_list):
        for g in gamma_list:
            for e in eps_list:
                agent = DoubleQLearning_Agent(env, lr=l, gamma=g, eps=e)
                agent.learn()
                agent.print_q()
                parameter.append([l, g, e])


def main():
    run_q_learning()


if __name__ == '__main__':
    st = time.time()
```

```
    main()
    print(all_reward)
    print('Максимальная награда:', np.max(all_reward), 'Значения гиперпараметров(lr, gamma,
eps):',
          parameter[np.argmax(np.max(all_reward))])
    all_time = time.time() - st
    print(f"Закончено за {all_time:.3f} сек")
    parameter = np.asarray(parameter)
    print(parameter.shape)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.scatter(parameter[:, 0], parameter[:, 1], parameter[:, 2], c=all_reward,
cmap='viridis')
    ax.set_xlabel('lr')
    ax.set_ylabel('gamma')
    ax.set_zlabel('eps')

    plt.show()
```

Подбор гиперпараметров

Перебор параметров:
lr от 0.0005 до 0.005 – 5 значений с равным шагом
Gamma от 0.9 до 1 – 5 значений с равным шагом
Eps от 0.05 до 0.9 – 9 значений с равным шагом

Всего 225 комбинаций.

Суммарная награда: -8.113470913926768 lr = 0.001 gamma = 0.900 eps = 0.050
Суммарная награда: -8.637901455723094 lr = 0.001 gamma = 0.900 eps = 0.156
Суммарная награда: -15.370377804869172 lr = 0.001 gamma = 0.900 eps = 0.688
Суммарная награда: -10.023106007504664 lr = 0.001 gamma = 0.925 eps = 0.263
Суммарная награда: -16.576836682149832 lr = 0.001 gamma = 0.925 eps = 0.794
Суммарная награда: -15.114266986996064 lr = 0.001 gamma = 0.950 eps = 0.688
Суммарная награда: -9.695509814383982 lr = 0.001 gamma = 0.975 eps = 0.263
Суммарная награда: -17.986032977730584 lr = 0.001 gamma = 0.975 eps = 0.900
Суммарная награда: -11.51856954548113 lr = 0.001 gamma = 1.000 eps = 0.369

Максимальная награда: -7.672707545879871
Значения гиперпараметров(lr, gamma, eps): [0.0005, 0.9, 0.05]
Закончено за 75.834 сек