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 _____ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)_____

ОТЧЕТ

Рубежный контроль №2

«Методы обучения с подкреплением»

по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: Нищук Р.С.
ФИО
группа ИУ5-21М _____
подпись
"__"_____2023 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.
ФИО
_____
подпись
"__"_____2023 г.

Москва - 2023

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.

Выполнение

Для подбора гиперпараметров был выбран алгоритм Q-learning. Для реализации алгоритма была выбрана среда Taxi из библиотеки Gym. Агент может находится в 25 позициях, пассажир может находится в 5 позициях, и 4 позиции для места назначения = 25*5*4= 500 состояний системы.

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():
        '''
        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=100):
        # 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 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 learn(self):
        '''
        Обучение на основе алгоритма Q-Learning
        '''
        self.episodes_reward = []
        # Цикл по эпизодам
        for ep in list(range(self.num_episodes)):
            # Начальное состояние среды
            state = self.get_state(self.env.reset())
            # Флаг штатного завершения эпизода
```

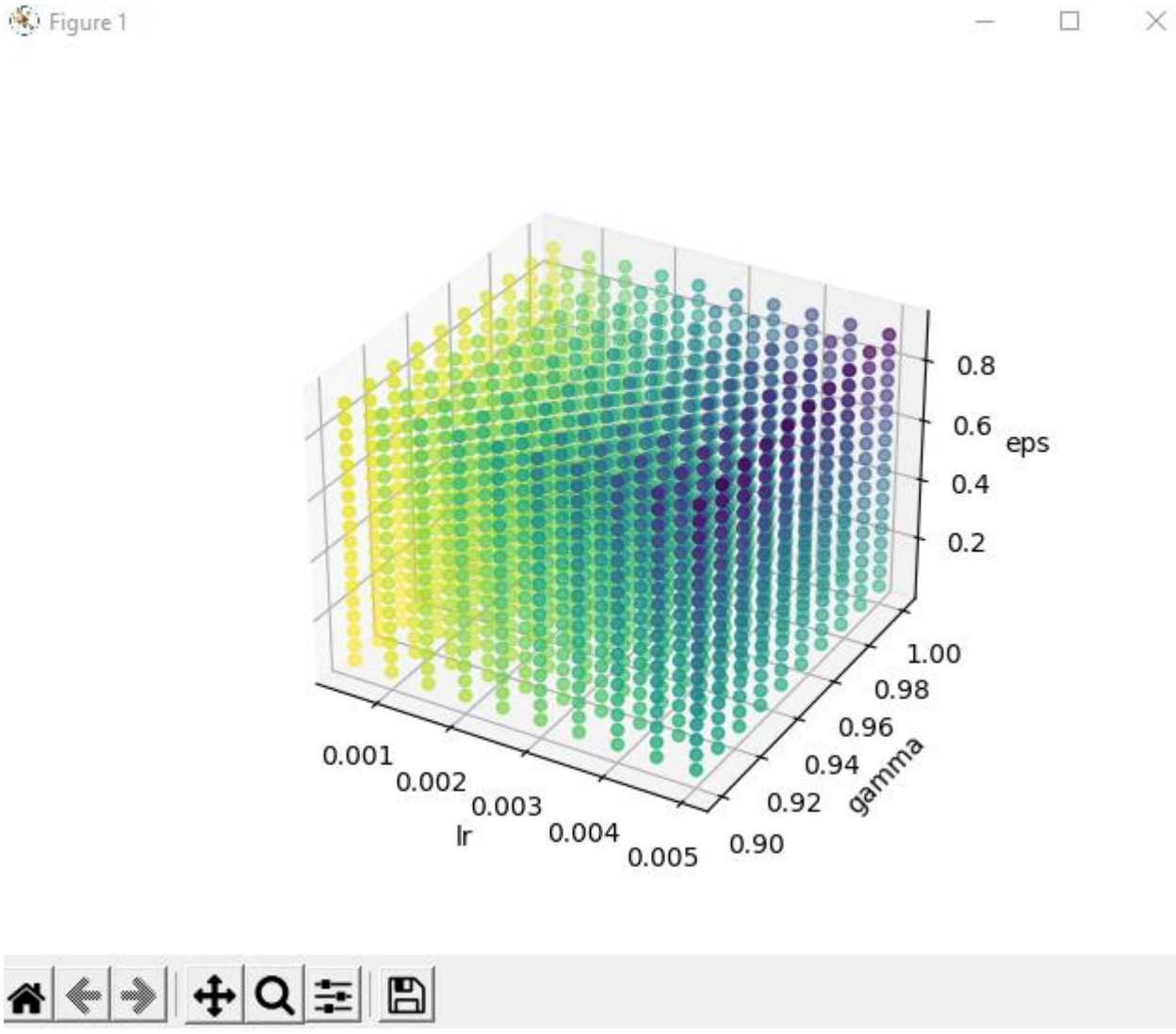
```
            done = False
            # Флаг нештатного завершения эпизода
            truncated = False
            # Суммарная награда по эпизоду
            tot_rew = 0

            # 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('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:
```

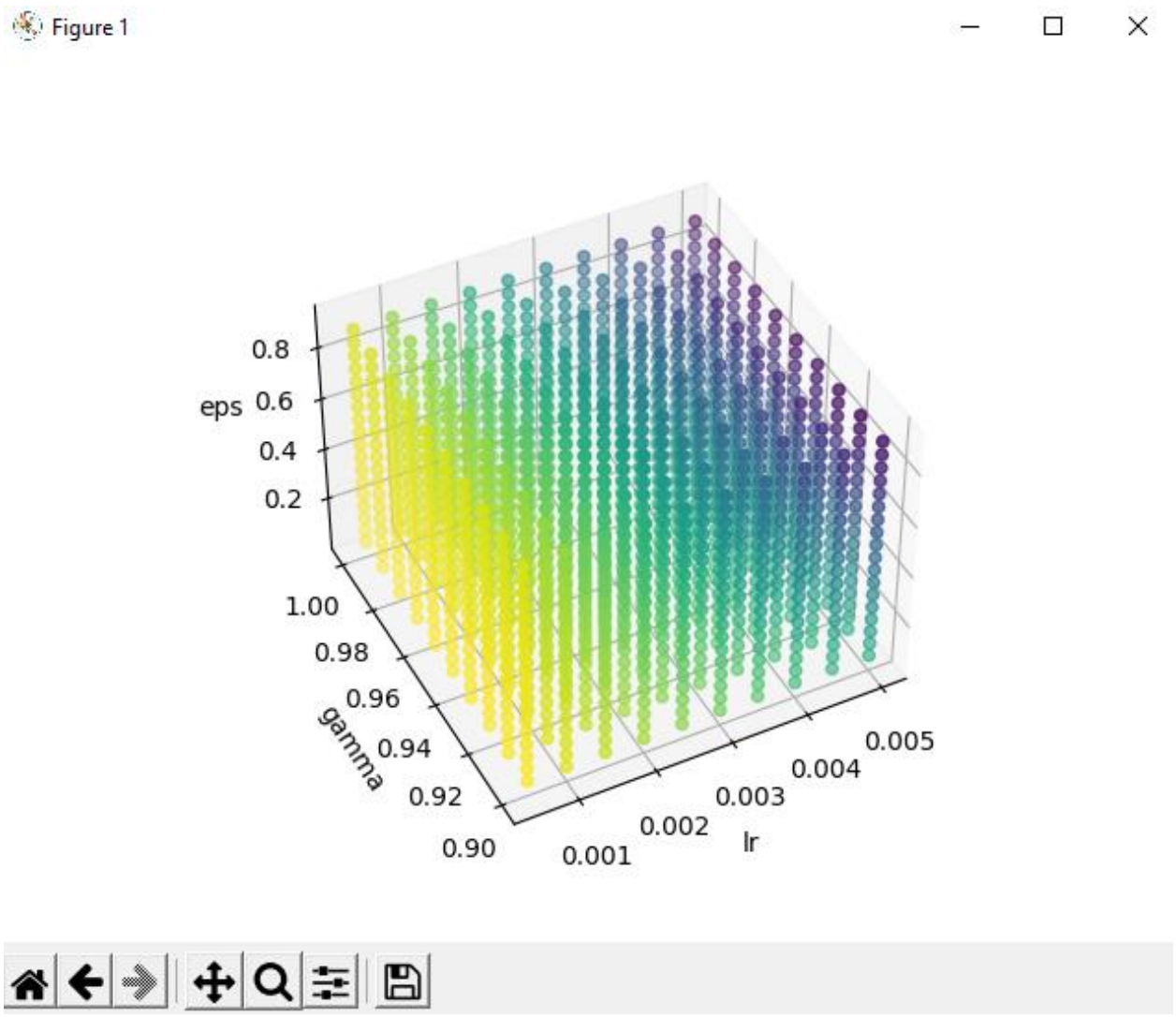
```
            done = True

def run_q_learning():
    env = gym.make('Taxi-v3')
    lr_list = np.linspace(0.0005, 0.005, num=10)
    gamma_list = np.linspace(0.9, 1, num=10)
    eps_list = np.linspace(0.05, 0.9, num=18)
    for l in tqdm(lr_list):
        for g in gamma_list:
            for e in eps_list:
                agent = QLearning_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()
```

Результат выполнения

Перебор параметров:
lr от 0.0005 до 0.005 – 10 значений с равным шагом
Gamma от 0.9 до 1 – 10 значений с равным шагом
Eps от 0.05 до 0.9 – 18 значений с равным шагом
Всего 1800 комбинаций.

```
Суммарная награда: -279.0273181909593 lr = 0.005 gamma = 0.956 eps = 0.750
Суммарная награда: -288.1271642934949 lr = 0.005 gamma = 0.956 eps = 0.800
Суммарная награда: -304.26682987871993 lr = 0.005 gamma = 0.956 eps = 0.850
Суммарная награда: -326.2489884613111 lr = 0.005 gamma = 0.956 eps = 0.900
Суммарная награда: -135.28674334019144 lr = 0.005 gamma = 0.967 eps = 0.050
Суммарная награда: -144.81234448376748 lr = 0.005 gamma = 0.967 eps = 0.100
Суммарная награда: -154.1660903603568 lr = 0.005 gamma = 0.967 eps = 0.150
Суммарная награда: -164.34499767708695 lr = 0.005 gamma = 0.967 eps = 0.200
Суммарная награда: -173.6472672738008 lr = 0.005 gamma = 0.967 eps = 0.250
Суммарная награда: -178.15681665496692 lr = 0.005 gamma = 0.967 eps = 0.300
Суммарная награда: -190.43958478973354 lr = 0.005 gamma = 0.967 eps = 0.350
Суммарная награда: -206.4452401033453 lr = 0.005 gamma = 0.967 eps = 0.400
Суммарная награда: -216.9071517753801 lr = 0.005 gamma = 0.967 eps = 0.450
Суммарная награда: -221.91988931368508 lr = 0.005 gamma = 0.967 eps = 0.500
```





По графику можно заметить, что чем меньше lr, тем значения лучше (более светлее). Лучше значения наблюдаются при меньшей lr, средней gamma и меньшей eps из проверяемых значений.

Максимальная награда: -14.927047489544622
Значения гиперпараметров(lr, gamma, eps): [0.0005, 0.9, 0.05]
Закончено за 1836.790 сек

Вывод

В ходе выполнения лабораторной работы мы ознакомились с базовыми методами обучения с подкреплением и осуществили подбор гиперпараметров для алгоритма q-learning, где критерием оптимизации являлась суммарная награда.