|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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:

группа ИУ5-23М

Гаврилов Л.Я.

ФИО

подпись

" " 2023 г.

ПРЕПОДАВАТЕЛЬ:

Москва - 2023

Гапанюк Ю.Е.

ФИО

подпись

" " 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.

**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()

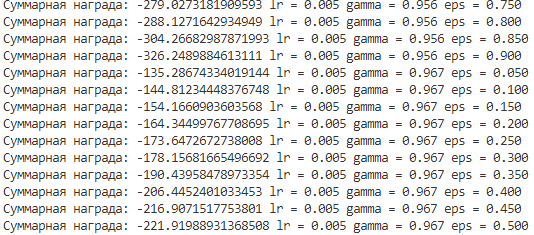
**Результат выполнения**

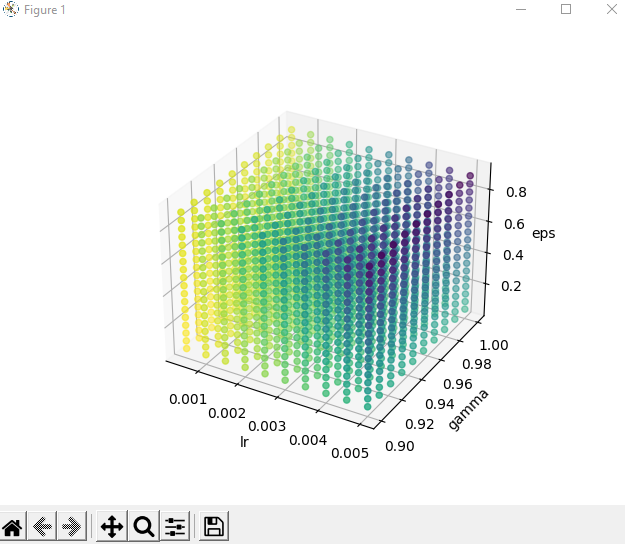
**Перебор параметров:  
lr** от 0.0005 до 0.005 – 10 значений с равным шагом

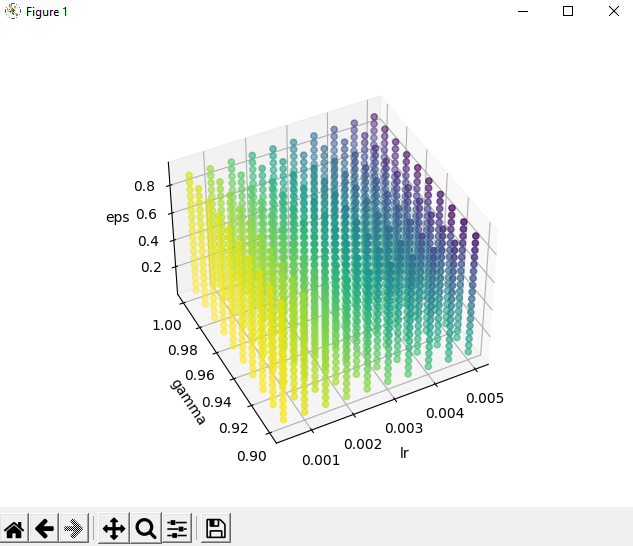
**Gamma** от 0.9 до 1 – 10 значений с равным шагом

**Eps** от 0.05 до 0.9 – 18 значений с равным шагом

Всего 1800 комбинаций.







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.