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

Рубежный контроль №2

«Методы обучения с подкреплением»

Студент группы ИУ5-23М

Уткин Дмитрий Юрьевич

Москва, 2023 г.

Задание

Для алгоритма временных различий Q-обучения осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

Описание выполнения

Алгоритм Q-обучения испытывается в среде Taxi-v3, в котором необходимо найти кратчайший безопасный маршрут от стоянки до пассажира и от пассажира до его дома.

Код

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


#            ********************************            БАЗОВЫЙ            АГЕНТ
****************************************

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)

    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


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=20000):
        # Вызов конструктора верхнего уровня
        super().__init__(env, eps)
        # Learning rate
        self.lr = lr
        # Коэффициент дисконтирования
        self.gamma = gamma
        # Количество эпизодов
```

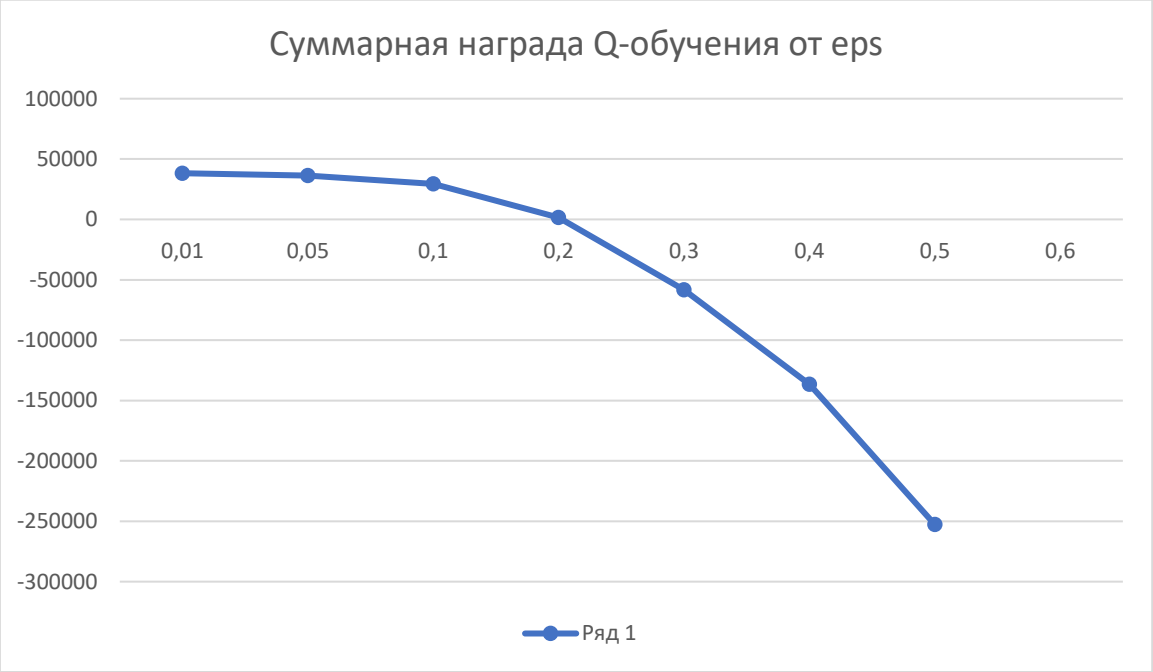
```
        self.num_episodes = num_episodes
        # Постепенное уменьшение eps
        self.eps_decay = 0.00005
        self.eps_threshold = 0.01

    def learn(self):
        '''
        Обучение на основе алгоритма Q-Learning
        '''
        self.episodes_reward = []
        sum_rew = 0
        # 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)
            sum_rew += tot_rew
        print('Суммарная награда = ', sum_rew)


def play_agent(agent):
    '''
    Проигрывание сессии для обученного агента
    '''
    env2 = gym.make('Taxi-v3', render_mode='human')
```

```
    state = env2.reset()[0]
    done = False
    tot_rew = 0
    while not done:
        action = agent.greedy(state)
        next_state, reward, terminated, truncated, _ = env2.step(action)
        env2.render()
        state = next_state
        tot_rew += reward
        if terminated or truncated:
            done = True
            print('Суммарная награда прогона = ',tot_rew)

def run_q_learning():
    env = gym.make('Taxi-v3')
    agent = QLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def main():
    run_q_learning()

if __name__ == '__main__':
    main()
```

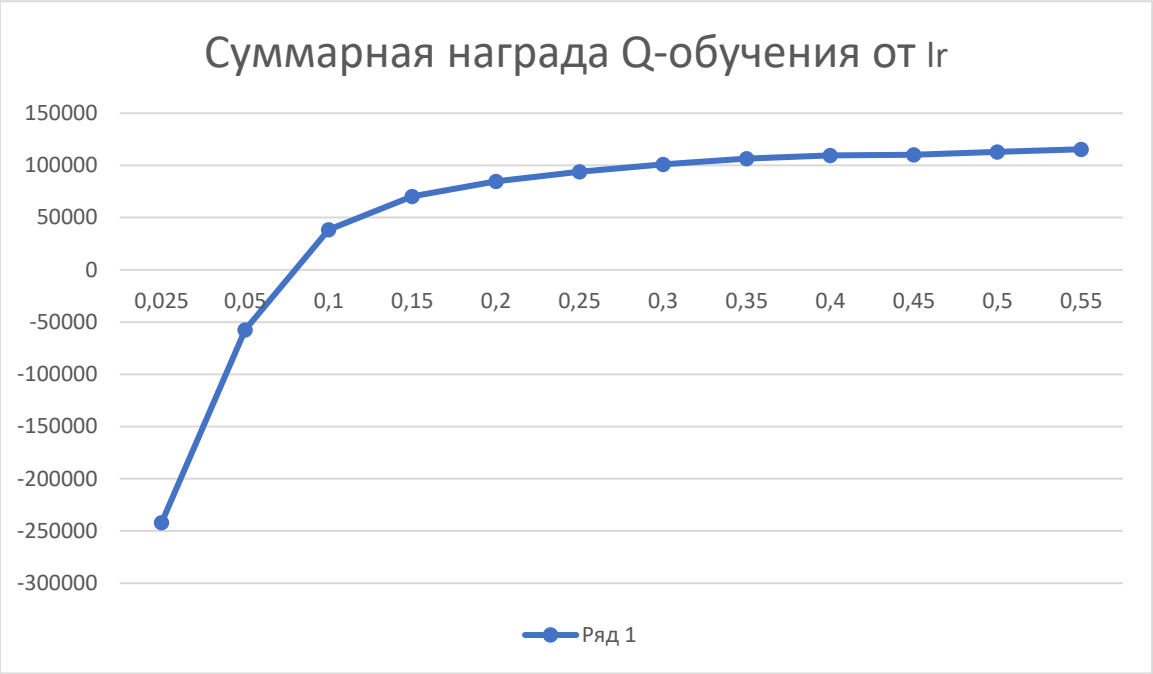
Подбор параметров

Начальные гиперпараметры: eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

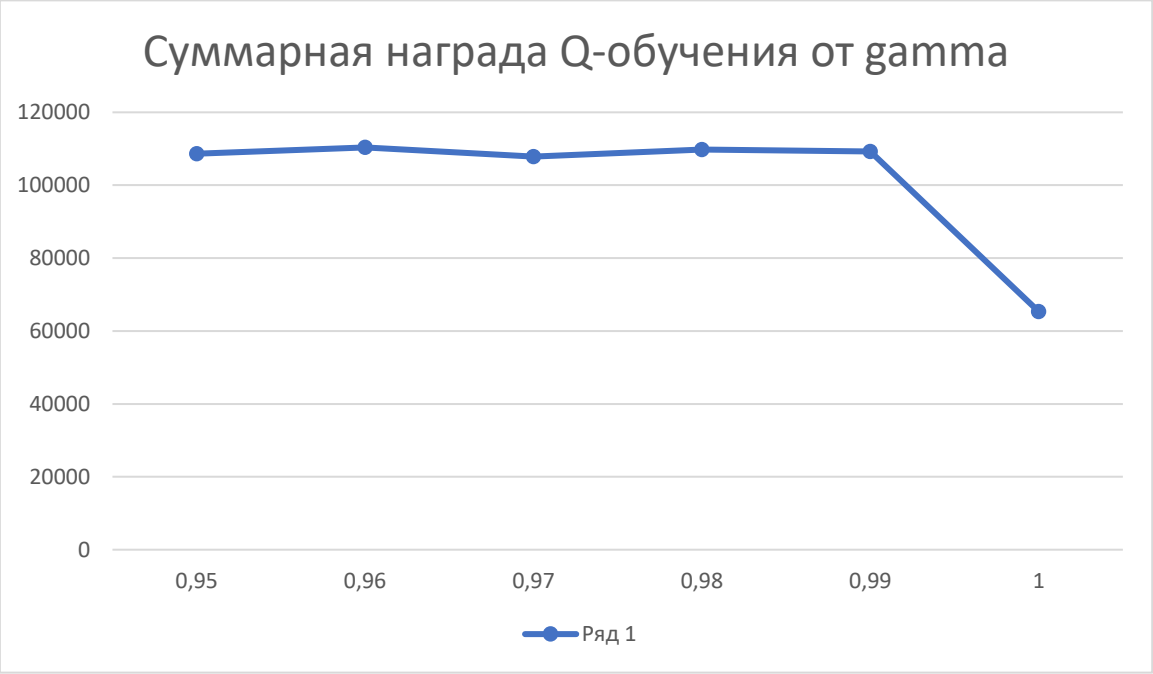
Изменим параметр eps и посмотрим зависимость суммарной награды от eps:



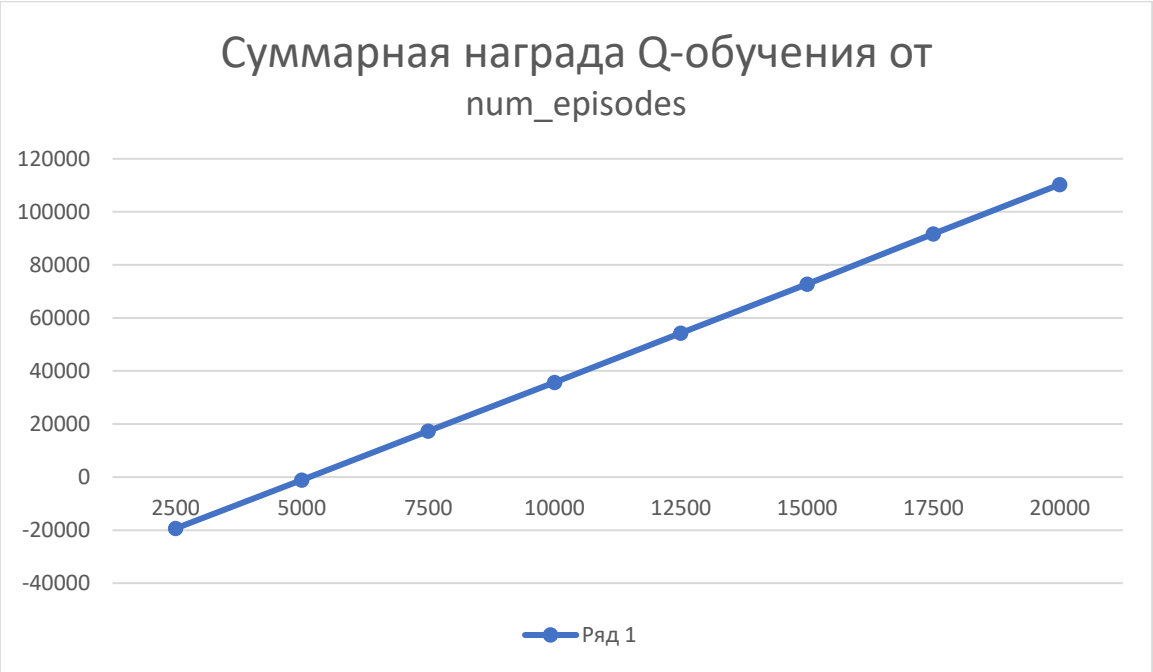
Изменим параметр lr и посмотрим зависимость суммарной награды от lr:



Изменим параметр gamma и посмотрим зависимость суммарной награды от gamma:



Изменим параметр num_episodes и посмотрим зависимость суммарной награды от num_episodes:



Вывод

В ходе многочисленных экспериментов были выведены отдельные этапы, на которых алгоритм с определенными гиперпараметрами выдавал различную суммарную награду. Эту награду удалось увеличить за счет сильного снижения параметра eps (эпсилон), на основе которого алгоритм выбирал действие (он стал чаще выбирать наилучшее действие), небольшого увеличения скорости обучения (lr) и небольшого уменьшения параметра гамма, которое отвечает за ценность долгосрочной награды. Также было сильно сокращено количество эпизодов, так как алгоритм достаточно быстро обучался в районе четверти от исходного количества эпизодов, следовательно, дальнейшее обучения было неэффективно и более времязатратно.

Наилучшие значения:

eps=0.01, lr=0.4, gamma=0.96, num_episodes=7500