

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Рубежный контроль № 2
по дисциплине «Методы машинного обучения»

Тема: «Методы обучения с подкреплением»

ИСПОЛНИТЕЛЬ: Подопригорова Н.С.
группа ИУ5-24М _____

"__" _____ 2023 г.

ПРЕПОДАВАТЕЛЬ: _____

"__" _____ 2023 г.

Москва - 2023

Задание:

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.

Текст программы.

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

ENV_NAME = 'Taxi-v3'

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 SARSA_Agent(BasicAgent):
    """
    Реализация алгоритма SARSA
    """
    # Наименование алгоритма
    ALGO_NAME = 'SARSA'

    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 SARSA
        """
        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

        # Выбор действия
        action = self.make_action(state)

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

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

            # Выполняем следующее действие
            next_action = self.make_action(next_state)

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

            # Следующее состояние считаем текущим
            state = next_state
            action = next_action
            # Суммарная награда за эпизод
            tot_rew += rew
            if (done or truncated):
                self.episodes_reward.append(tot_rew)

    def learn(self):
        return np.max(episodes_reward)

    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 print_q(self):
        print('Вывод Q-матриц для алгоритма ', self.ALGO_NAME)
        print('Q1')
        print(self.Q)
        print('Q2')
        print(self.Q2)

def play_agent(agent):
    """
    Проигрывание сессии для обученного агента
    """
    env2 = gym.make(ENV_NAME, 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(tot_rew)

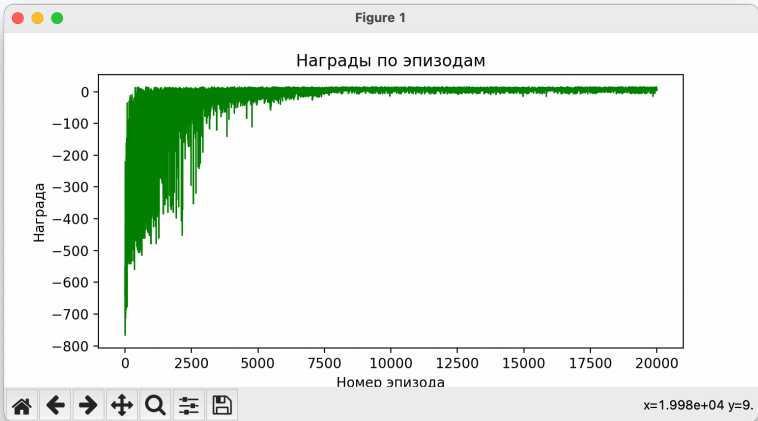
def run_sarsa():
    env = gym.make(ENV_NAME)
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)


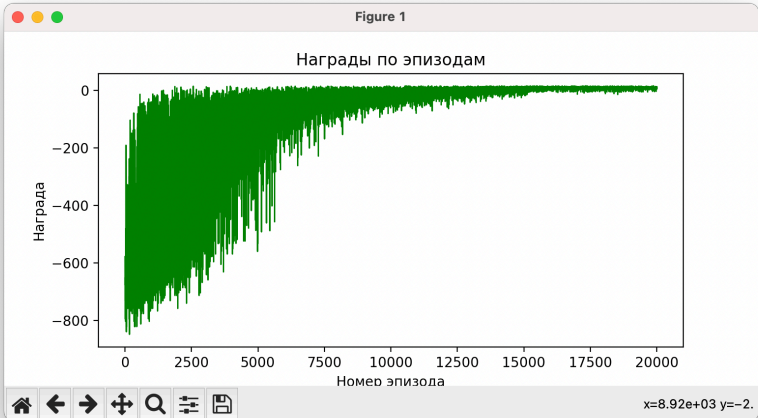
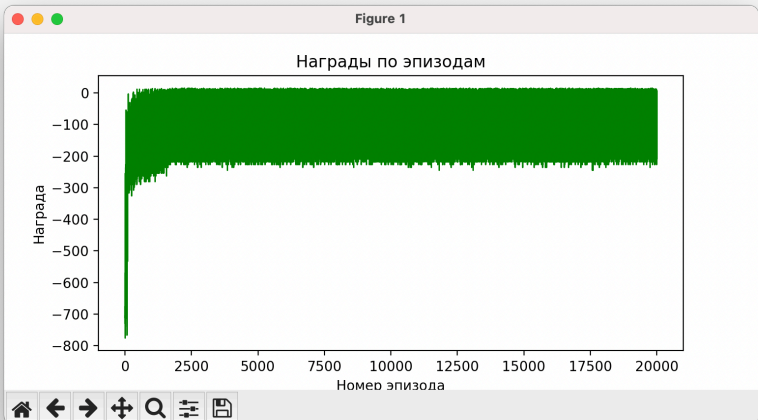
def main():
    run_sarsa()
    print( np.max(rewards) )

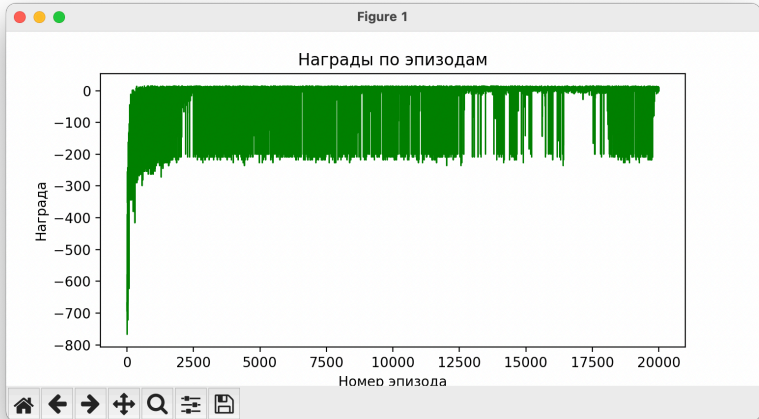
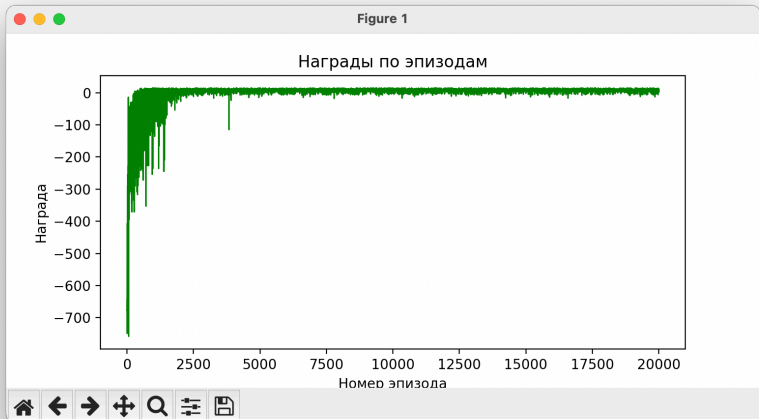
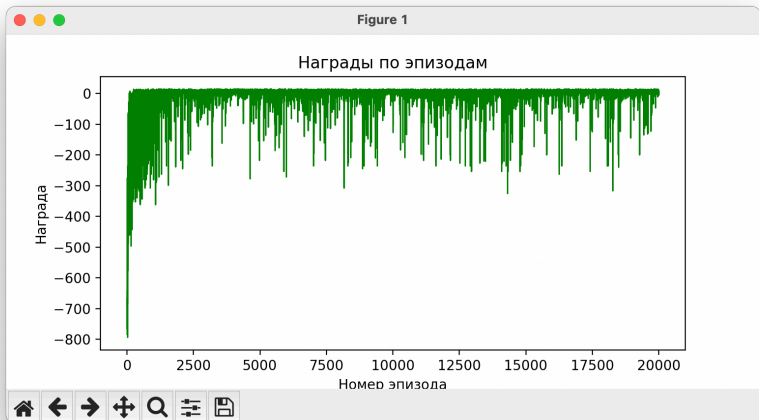
if __name__ == '__main__':
    main()

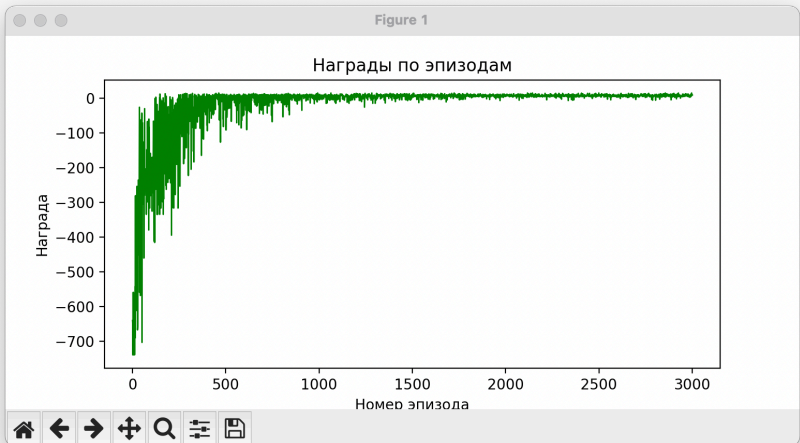
```

Подбор гиперпараметров для SARSA

		Максимальная награда
eps=0.4, lr=0.1, gamma=0.9 8, num_episodes=20000		12

		Максимальная награда
$\text{eps}=0.1,$ $\text{lr}=0.1,$ $\text{gamma}=0.98,$ $\text{num_episodes}=20000$		4
$\text{eps}=0.8,$ $\text{lr}=0.1,$ $\text{gamma}=0.98,$ $\text{num_episodes}=20000$		8
$\text{eps}=0.1,$ $\text{lr}=0.1,$ $\text{gamma}=0.2,$ $\text{num_episodes}=20000$		-223

		Максимальная награда
$\text{eps}=0.1$, $\text{lr}=0.1$, $\text{gamma}=0.7$, $\text{num_episodes}=20000$		9
$\text{eps}=0.1$, $\text{lr}=0.1$, $\text{gamma}=0.8$, $\text{num_episodes}=20000$		11
$\text{eps}=0.1$, $\text{lr}=0.8$, $\text{gamma}=0.98$, $\text{num_episodes}=20000$		16

		Максимальная награда
$\text{eps}=0.1$, $\text{lr}=0.1$, $\text{gamma}=0.98$, $\text{num_episodes}=3000$		

Выводы

Q-обучение с параметрами по умолчанию сходится быстрее остальных алгоритмов.

Для SARSA-алгоритма уменьшение eps позволяет ускорить сходимость.

Слишком большой lr (0,8) увеличивает разброс результатов (некоторые проходы успешны, другие сильно нет). Уменьшение gamma не позволяет достичь наилучшей ценности (агент застревает надолго в одних ячейках).

Наилучшие параметры: $\text{eps}=0.1$, $\text{lr}=0.1$, $\text{gamma}=0.98$, $\text{num_episodes}=20000$