**Технологии машинного обучения. Рубежный контроль №2.**

**Студент: Халимонов А.М.  
Группа: ИУ5-22М**

## Тема: Методы обучения с подкреплением.

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

* **SARSA (выбранный вариант)**
* 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 statistics import mean

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SARSA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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=10000):

        # 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 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 play\_agent(agent):

    '''

    Проигрывание сессии для обученного агента

    '''

    env2 = gym.make('Taxi-v3', render\_mode='human')

    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\_sarsa():

    # Default eps=0.4, lr=0.1, gamma=0.98, num\_episodes=20000

    epsArr = [0.2, 0.3, 0.4, 0.5, 0.6]

    epsRew = []

    for eps in epsArr:

        env = gym.make('Taxi-v3')

        agent = SARSA\_Agent(env, eps=eps)

        agent.learn()

        epsRew.append(mean(agent.episodes\_reward[-10:]))

        print(f'Награда при eps = {eps}: {epsRew[-1]}')

    bestEps = epsArr[epsRew.index(max(epsRew))]

    print('='\*30)

    lrArr = [0.2, 0.15, 0.1, 0.07, 0.05]

    lrRew = []

    for lr in lrArr:

        env = gym.make('Taxi-v3')

        agent = SARSA\_Agent(env, lr=lr)

        agent.learn()

        lrRew.append(mean(agent.episodes\_reward[-10:]))

        print(f'Награда при lr = {lr}: {lrRew[-1]}')

    bestLr = lrArr[lrRew.index(max(lrRew))]

    print('='\*30)

    gammaArr = [0.9, 0.95, 0.98, 0.99]

    gammaRew = []

    for gamma in gammaArr:

        env = gym.make('Taxi-v3')

        agent = SARSA\_Agent(env, gamma=gamma)

        agent.learn()

        gammaRew.append(mean(agent.episodes\_reward[-10:]))

        print(f'Награда при gamma = {gamma}: {gammaRew[-1]}')

    bestGamma = gammaArr[gammaRew.index(max(gammaRew))]

    print('='\*30)

    env = gym.make('Taxi-v3')

    agent = SARSA\_Agent(env,

        eps=bestEps,

        lr=bestLr,

        gamma=bestGamma,

    )

    agent.learn()

    print(f'Награда при лучших гиперпараметрах (eps = {bestEps}, lr = {bestLr}, gamme = {bestGamma}): {mean(agent.episodes\_reward[-10:])}')

def main():

    run\_sarsa()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Результаты:**

