Технологии машинного обучения. Рубежный контроль №2.

Студент: Перлин Л. В.
Группа: ИУ5-24М

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

```
Награда при eps = 0.2: 6.3
Награда при eps = 0.3: 8.1
Награда при eps = 0.4: 7.3
Награда при eps = 0.5: 8.6
Награда при eps = 0.6: 0.4
==============================
Награда при lr = 0.2: 6.9
Награда при lr = 0.15: 7.8
Награда при lr = 0.1: 7.4
Награда при lr = 0.07: 7.1
Награда при lr = 0.05: 6.4
==============================
Награда при gamma = 0.9: 8.4
Награда при gamma = 0.95: 7.4
Награда при gamma = 0.98: 7.7
Награда при gamma = 0.99: 8.1
==============================
Награда при лучших гиперпараметрах (eps = 0.5, lr = 0.15, gamme = 0.9): 7.5
```