

Защищено:
Гапанюк Ю.Е.

Демонстрация:
Гапанюк Ю.Е.

"__" _____ 2024 г.

"__" _____ 2024 г.

Отчет по лабораторной работе № 5 по курсу Методы машинного обучения

Тема работы: " Обучение на основе временны'х различий "

11
(количество листов)
Вариант № 15

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-22М

Чиварзин А.Е.

(подпись)

"__" _____ 2024 г.

Задание:

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- SARSA
- Q-обучение
- Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

Ход выполнения работы

In [3]:

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

In [4]:

```
# ***** 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 в 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
In [5]:
# ***** 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=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)

```

In [6]:

```

# ***** 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=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 = []
    # 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)

```

In [7]:

```

# ***** Двойное Q-обучение *****

```

```

class DoubleQLearning_Agent(BasicAgent):
    """
    Реализация алгоритма Double Q-Learning
    """
    # Наименование алгоритма
    ALGO_NAME = 'Двойное 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)
        # Вторая матрица
        self.Q2 = np.zeros((self.nS, self.nA))
        # 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 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 learn(self):
    """
    Обучение на основе алгоритма Double Q-Learning
    """
    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

        # 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)

            if np.random.rand() < 0.5:
                # Обновление первой таблицы
                self.Q[state][action] = self.Q[state][action] + self.lr * \
                    (rew + self.gamma * self.Q2[next_state][np.argmax(self.Q[next_state])] - self.Q[state][action])
            else:
                # Обновление второй таблицы
                self.Q2[state][action] = self.Q2[state][action] + self.lr * \
                    (rew + self.gamma * self.Q[next_state][np.argmax(self.Q2[next_state])] - self.Q2[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('CliffWalking-v0', 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():
    env = gym.make('CliffWalking-v0')
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

```

```

def run_q_learning():
    env = gym.make('CliffWalking-v0')
    agent = QLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

```

```

def run_double_q_learning():
    env = gym.make('CliffWalking-v0')
    agent = DoubleQLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

```

Запустим код

In [10]:

```
run_sarsa()
```

100%  20000/20000 [00:07<00:00, 2855.25it/s]

Вывод Q-матрицы для алгоритма SARSA

```

[[-13.28458418 -12.46793538 -14.22885418 -13.27657094]
 [-12.45936175 -11.66533596 -13.34660858 -13.45906628]
 [-11.71638403 -10.88101524 -13.39477295 -12.65541078]
 [-10.86737962 -10.11829038 -11.93552502 -11.92231622]
 [-10.07705676 -9.37897128 -11.23305926 -11.06916088]
 [-9.2474099 -8.4283109 -10.18159789 -10.29406844]
 [-8.52879815 -7.47575527 -9.32193519 -9.43540966]
 [-7.55275583 -6.59704964 -8.722669 -8.63843327]
 [-6.72157012 -5.70996807 -7.58792355 -7.72135969]
 [-5.75754608 -4.83684192 -5.44219841 -6.86500434]
 [-4.83800566 -4.03257156 -4.27658203 -5.94011482]
 [-3.93797347 -3.91535093 -2.9404 -5.0490357 ]
 [-13.2441641 -13.80960639 -14.87261324 -14.00541517]
 [-12.42772767 -16.72399429 -20.83693558 -17.26277688]
 [-11.87122455 -15.62293863 -19.44586793 -19.62173403]
 [-10.02590105 -14.50795002 -22.49297071 -16.05194529]]

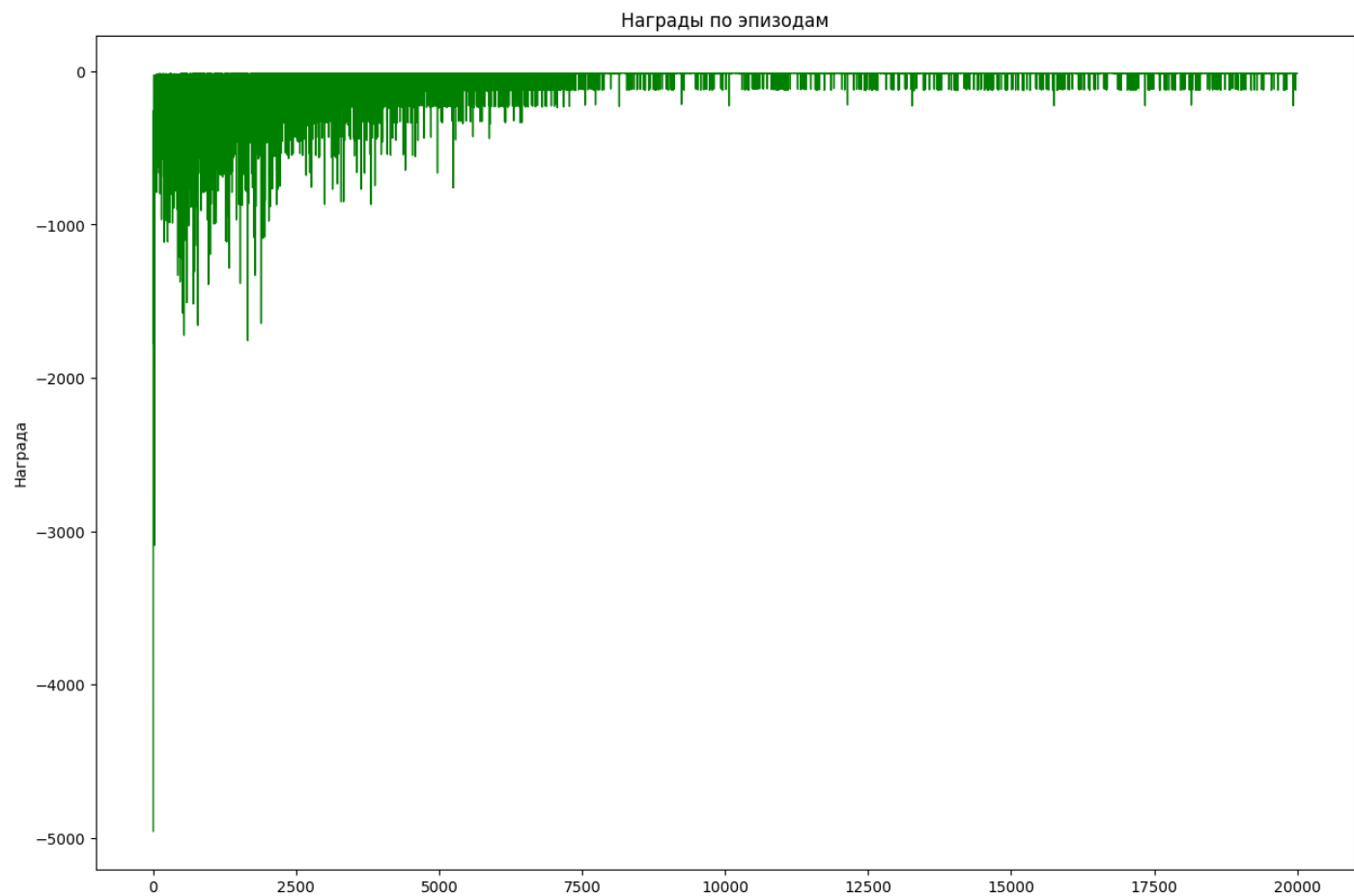
```



```

[ -10.7141511 -9.96342959 -9.96342936 -11.46688772]
[ -9.93713951 -9.14635911 -9.14635906 -10.68894613]
[ -9.11813207 -8.31261184 -8.31261185 -9.94427009]
[ -8.29806773 -7.46184886 -7.46184886 -9.13158981]
[ -7.45170816 -6.59372334 -6.59372334 -8.28755561]
[ -6.59019291 -5.70788096 -5.70788096 -7.45437305]
[ -5.69600763 -4.80396016 -4.80396016 -6.5784164 ]
[ -4.7856999 -3.881592 -3.881592 -5.66666797]
[ -3.87170624 -3.85779722 -2.9404 -4.75994889]
[ -13.05646141 -11.54888054 -11.54888054 -12.31788719]
[ -12.31703959 -10.76416381 -10.76416381 -12.31790068]
[ -11.54871861 -9.96343246 -9.96343246 -11.54888039]
[ -10.76415436 -9.14635966 -9.14635966 -10.76416381]
[ -9.96342579 -8.31261189 -8.31261189 -9.96343241]
[ -9.14635939 -7.46184887 -7.46184887 -9.14635965]
[ -8.31261182 -6.59372334 -6.59372334 -8.31261186]
[ -7.46184883 -5.70788096 -5.70788096 -7.46184671]
[ -6.59372332 -4.80396016 -4.80396016 -6.59372319]
[ -5.70788095 -3.881592 -3.881592 -5.70788095]
[ -4.80396014 -2.9404 -2.9404 -4.80396015]
[ -3.881592 -2.9404 -1.98 -3.88159199]
[ -12.31790293 -10.76416381 -12.31790293 -11.54888054]
[ -11.54888054 -9.96343246 -11.31790293 -11.54888054]
[ -10.76416381 -9.14635966 -11.31790293 -10.76416381]
[ -9.96343246 -8.31261189 -11.31790293 -9.96343246]
[ -9.14635966 -7.46184887 -11.31790293 -9.14635966]
[ -8.31261189 -6.59372334 -11.31790293 -8.31261189]
[ -7.46184887 -5.70788096 -11.31790293 -7.46184887]
[ -6.59372334 -4.80396016 -11.31790293 -6.59372334]
[ -5.70788096 -3.881592 -11.31790293 -5.70788096]
[ -4.80396016 -2.9404 -11.31790293 -4.80396016]
[ -3.881592 -1.98 -11.31790293 -3.881592 ]
[ -2.9404 -1.98 -1. -2.9404 ]
[ -11.54888054 -11.31790293 -12.31790293 -12.31790293]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]

```



In [11]:

run_double_q_learning()

100% ██████████ 20000/20000 [00:06<00:00, 2969.
40it/s]

Вывод Q-матриц для алгоритма Двойное Q-обучение

Q1

[illegible]

Q2

[-22.11359144	-24.08695095	-12.34648022	-20.58376148]
[-24.3772731	-20.23936943	-11.60015664	-20.68966162]
[-20.78442806	-22.33907093	-11.32435794	-23.79702714]
[-21.40948399	-10.52538298	-11.8607127	-19.4275484]
[-14.41432868	-14.19766013	-9.15410114	-19.89932976]
[-12.73742277	-12.34065824	-8.31784586	-13.03936663]
[-12.21831473	-13.8877325	-7.87461485	-13.22091431]
[-13.02798502	-6.61729505	-9.58935746	-11.18921565]
[-7.78057713	-6.67925808	-10.90617805	-12.56648911]
[-7.20641231	-4.80396014	-4.68174345	-6.94005041]
[-4.69247715	-3.881592	-3.98608298	-7.38218033]
[-3.79611695	-3.86943326	-2.9404	-4.77927439]
[-13.88980285	-11.59500634	-11.54888054	-12.51232918]
[-13.76011821	-11.53302271	-10.76146381	-12.34451383]
[-12.16494067	-10.17350557	-9.96343246	-11.81916679]
[-15.97823657	-9.3125217	-9.14635966	-11.14385817]
[-10.41644216	-8.31261189	-8.33855528	-10.0650769]
[-9.22029759	-7.47603917	-7.46184887	-9.1833926]
[-9.32584615	-6.92893503	-6.59372334	-8.43118825]
[-13.52840395	-6.51039313	-6.03702695	-8.19836957]
[-6.63599923	-4.80396016	-4.79143801	-7.15782243]
[-6.67385549	-3.881592	-3.89020301	-6.16591246]
[-4.80252192	-2.9404	-2.94057964	-4.80173266]
[-3.88155886	-2.94039095	-1.98	-3.88159896]
[-12.31790293	-10.76416381	-12.31790293	-11.54888054]

```
[ -11.54888054 -9.96343246 -111.31790293 -11.54888054 ]
[ -10.76416381 -9.14635966 -111.31790293 -10.76416381 ]
[ -9.96343246 -8.31261189 -111.31790293 -9.96343246 ]
[ -9.14635966 -7.46184887 -111.31790293 -9.14635966 ]
[ -8.31261189 -6.59372334 -111.31790293 -8.31261189 ]
[ -7.46184887 -5.70788096 -111.31790293 -7.46184887 ]
[ -6.59372334 -4.80396016 -111.31790293 -6.59372334 ]
[ -5.81072552 -3.881592 -111.31790293 -5.70788096 ]
[ -4.80396016 -2.9404 -111.31790293 -4.80396016 ]
[ -3.881592 -1.98 -111.31790292 -3.881592 ]
[ -2.9404 -1.98 -1. -2.9404 ]
[ -11.54888054 -111.31790293 -12.31790293 -12.31790293 ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
[ 0. 0. 0. 0. ]
```

