 **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 и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Курс «Методы машинного обучения»

Отчет по домашнему заданию

Выполнила:

студент группы ИУ5-24М

Пименов Г.Ю.

Москва, 2023 г.

**Цель работы:** ознакомление с методом обучения с подкреплением на основе временных различий Double SARSA.

**Задание:**

Реализовать алгоритм Double SARSA для среды обучения с подкрепления

CliffWalking из библиотеки Gym

**Ход работы**

В среде CliffWalking агент может находиться в 48 состояниях и осуществлять 4 действия. Одно состояние начальное, одно конечное, а также есть «обрыв» из 10 клеток внизу карты (рис.1).

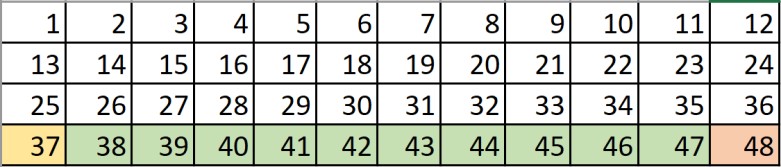


Рис.1. Карта состояний Действия: 1 – вверх, 2 – вправо, 3 – вниз, 4 – влево.

Вознаграждение равно –1 для всех переходов, кроме ведущих в «обрыв». За вход в эту область начисляется вознаграждение –100, после чего агент мгновенно переносится в начальную позицию [2].

В алгоритме Double SARSA, как и в Double Q-learning, используется эпсилонжадная стратегия, определяющая действие на основе среднего по сумме значений двух таблиц 𝑄1 и 𝑄2.

Обновление происходит по правилу [1]:

𝑄𝑡1+1(𝑠, 𝑎) ← 𝑄𝑡1(𝑠, 𝑎) + 𝛼[𝑟 + 𝛾𝑄𝑡2(𝑠′, 𝑎′) − 𝑄𝑡1(𝑠, 𝑎)]

Алгоритм Double SARSA отличается от SARSA наличием дополнительной таблицы, а от Double Q-learning тем, что в конце 𝑄1 и 𝑄2 не обновляются с вероятностью

0.5, а обновляется лишь одна таблица, а затем с вероятностью 0.5 они меняются местами.

**Алгоритм 1. Double SARSA**

1. Инициализировать 𝑄1(𝑠, 𝑎) и 𝑄2(𝑠, 𝑎) произвольным образом
2. **Повторять** для каждого эпизода:
3. Инициализировать s
4. Выбрать a в состоянии s, следуя произвольной стратегии
5. **Повторять** для каждого шага эпизода:
6. Предпринять действие, наблюдать r, s’
7. Выбрать a’ из состояния s’, следуя ε-жадной стратегии относительно среднего 𝑄1 + 𝑄2
8. 𝑄1(𝑠, 𝑎) ← 𝑄1(𝑠, 𝑎) + 𝛼[𝑟 + 𝛾𝑄2(𝑠′, 𝑎′) − 𝑄1(𝑠, 𝑎)]
9. 𝑠 ← 𝑠′; 𝑎 ← 𝑎′;
10. **С вероятностью** 0.5:
11. Поменять местами 𝑄1и 𝑄2

**Текст программы**

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

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class DoubleSARSA\_Agent(BasicAgent):

'''

Реализация алгоритма Double SARSA

'''

# Наименование алгоритма

ALGO\_NAME = 'Double SARSA'

def \_\_init\_\_(self, env, eps=0.015, lr=0.1, gamma=0.97, num\_episodes=20000):

#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 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)

self.Q[state][action] = self.Q[state][action] + self.lr \* \ (rew + self.gamma \* self.Q2[next\_state][next\_action] - self.Q[state][action])

# Следующее состояние считаем текущим state = next\_state action = next\_action if np.random.rand() < 0.5: swap\_list = self.Q self.Q = self.Q2 self.Q2 = swap\_list # Суммарная награда за эпизод tot\_rew += rew if (done or truncated):

self.episodes\_reward.append(tot\_rew) def sum\_rewards(self): # Суммарная награда sum\_rewards = sum(self.episodes\_reward) print('Cуммарная награда Double SARSA: ', sum\_rewards)

def play\_agent(agent):

'''

Проигрывание сессии для обученного агента

''' env2 = gym.make('CliffWalking-v0', render\_mode='human') #FrozenLake-v1 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\_double\_sarsa():

env = gym.make('CliffWalking-v0') agent = DoubleSARSA\_Agent(env) agent.learn() agent.sum\_rewards() agent.print\_q() agent.draw\_episodes\_reward() play\_agent(agent)

def main():

run\_double\_sarsa()

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

main()

**Результаты выполнения программы**

Обучение проводилось со следующими гиперпараметрами: eps=0.015, lr=0.1, gamma=0.97, num\_episodes=20000. Награды по эпизодам представлены на рис.1.

Суммарная награда -333627.

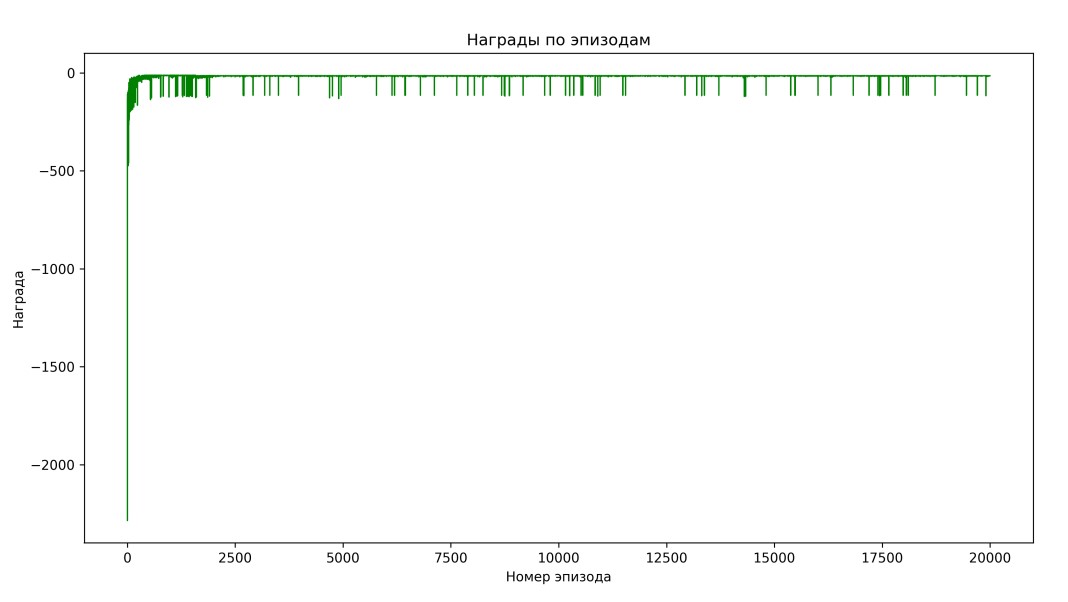


Рис.1. Награды по эпизодам

Q-матрицы для алгоритма Double SARSA представлены на рис.2 – рис.3.

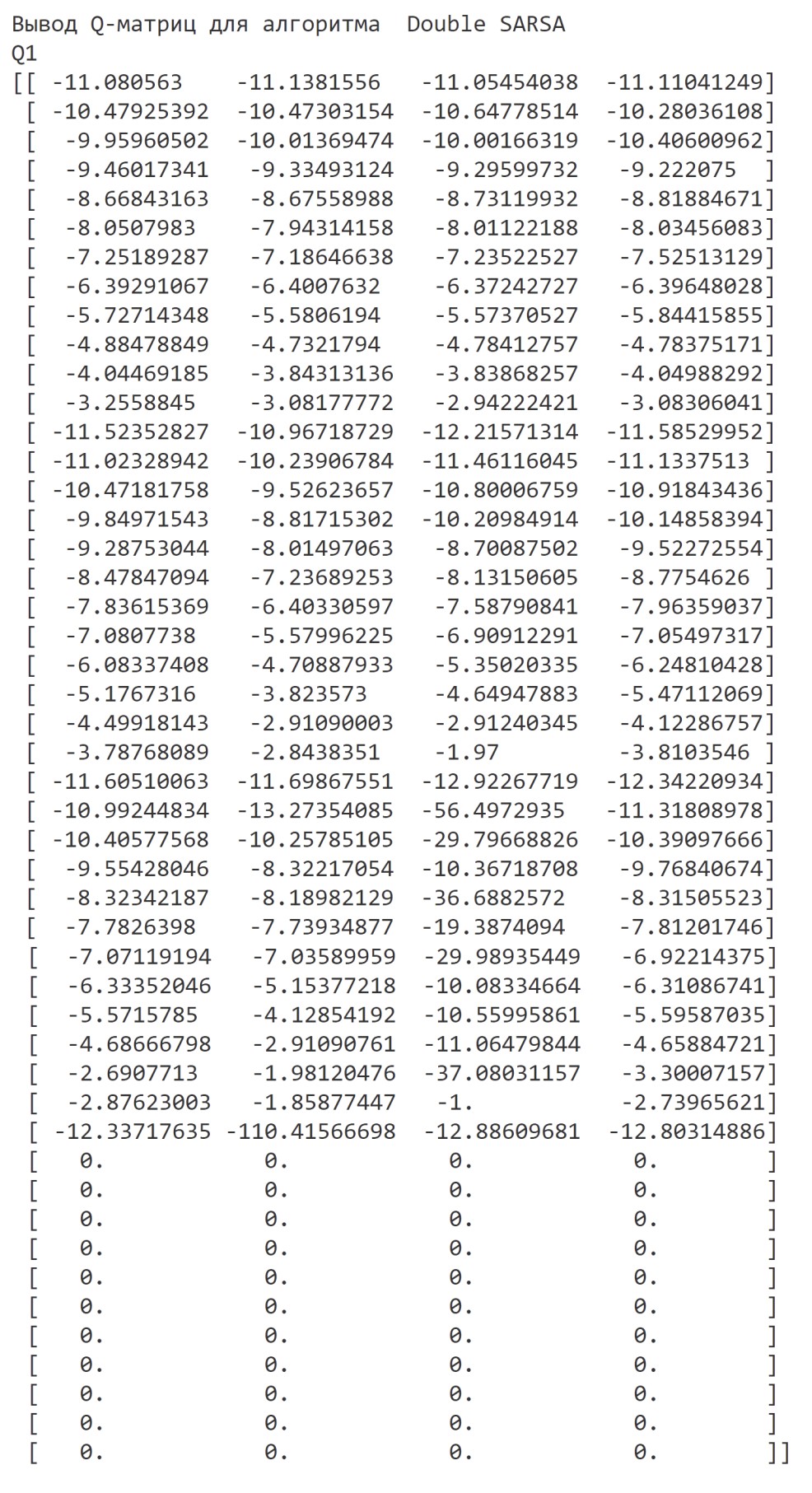


Рис.2. Матрица Q1

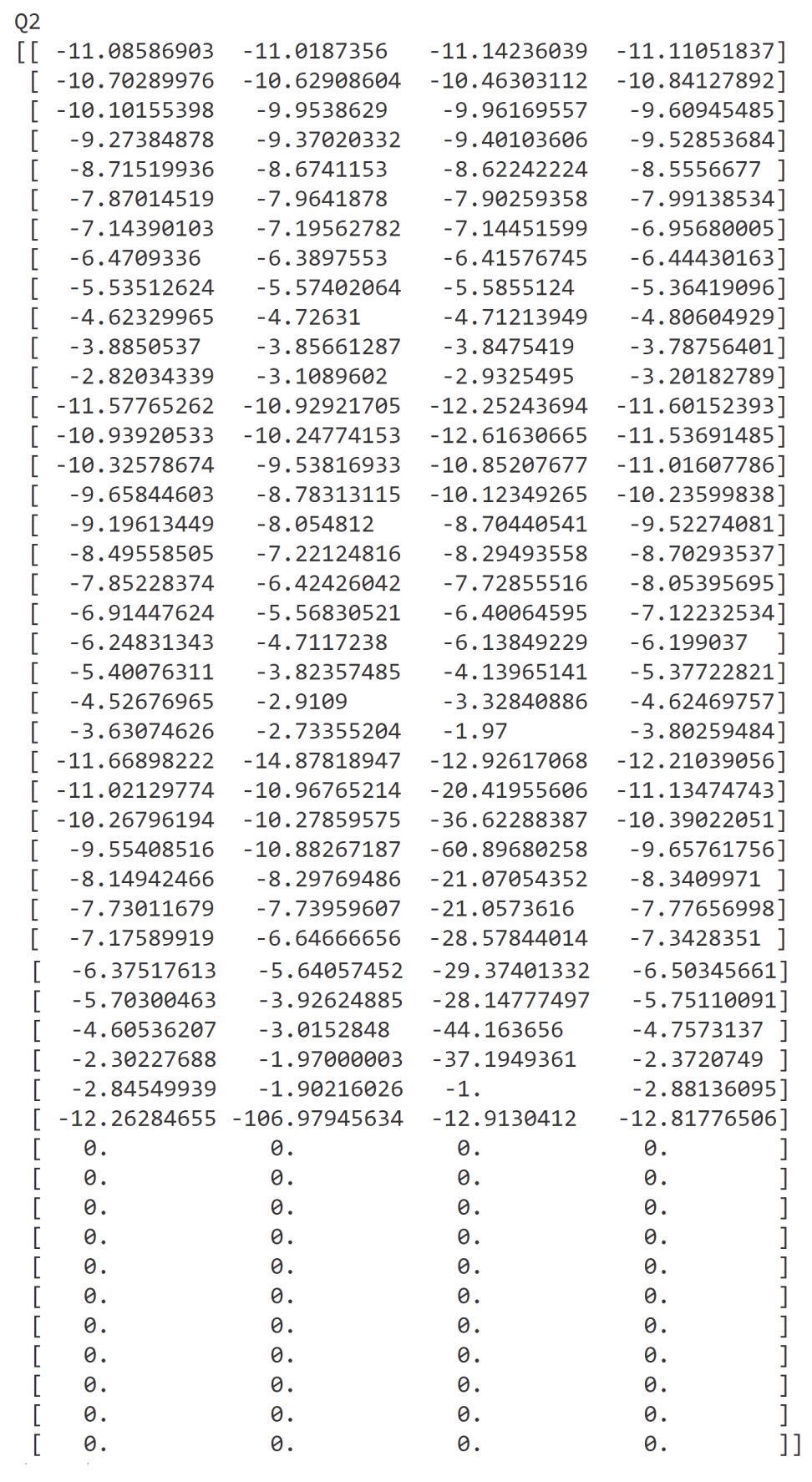


Рис.3. Матрица Q2

По матрицам видно, что для состояний 38 – 48 награды 0 (эти состояния соответствуют «обрыву» и конечному состоянию. Максимальное вознаграждение из 36-го состояния (клетка над конечным состоянием) соответствует исходному -1 за действие «вниз».

Путь обученного агента представлен на рис.4. При значении eps=0.4 путь проходит по максимально «безопасной» траектории, то есть через максимально удаленные клетки от «обрыва». Однако в данном примере eps=0.015, и стратегия дальше приближается к оптимальной при дальнейшем уменьшении этого параметра.

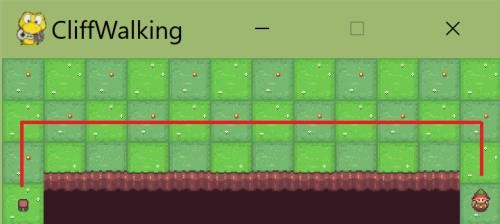


Рис.4. Путь обученного агента

**Вывод:** в ходе выполнения домашнего задания я ознакомилась с методом обучения с подкреплением на основе временных различий Double SARSA.

**Список использованных источников**

1. Ganger, M., Du- ryea, E. and Hu, W. (2016) Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning. Journal of Data Ana- lysis and Information Processing, 4, 159-

176. <http://dx.doi.org/10.4236/jdaip.2016.44014>

2. Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 552 с.