



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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА _____СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)_____

ОТЧЕТ

Лабораторная работа №5

«Обучение на основе временных различий»

по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ:

группа ИУ5-21М

Рабцевич К.Р.

ФИО

подпись

"__" _____ 2023 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

ФИО

подпись

"__" _____ 2023 г.

Цель работы

Ознакомление с базовыми методами обучения с подкреплением на основе временных различий.

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).

Выполнение

Для реализации алгоритмов была выбрана среда Taxi из библиотеки Gym. Агент может находиться в 25 позициях, пассажир может находиться в 5 позициях, и 4 позиции для места назначения = $25 \cdot 5 \cdot 4 = 500$ состояний системы.

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

# ***** 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)

# ***** 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)

# ***** 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('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():  
    env = gym.make('Taxi-v3')  
    agent = SARSA_Agent(env)  
    agent.learn()  
    agent.print_q()  
    agent.draw_episodes_reward()  
    play_agent(agent)
```

```
def run_q_learning():  
    env = gym.make('Taxi-v3')  
    agent = QLearning_Agent(env)  
    agent.learn()  
    agent.print_q()  
    agent.draw_episodes_reward()  
    play_agent(agent)
```

```
def run_double_q_learning():  
    env = gym.make('Taxi-v3')  
    agent = DoubleQLearning_Agent(env)  
    agent.learn()  
    agent.print_q()  
    agent.draw_episodes_reward()  
    play_agent(agent)
```

```
def main():  
    run_sarsa()  
    #run_q_learning()  
    #run_double_q_learning()
```

```
if __name__ == '__main__':
    main()
```

Результат выполнения

Алгоритм SARSA

```
Вывод Q-матрицы для алгоритма SARSA
[[ 0.          0.          0.          0.          0.
  0.          ]
 [ -7.21253293 -5.52965781 -8.32184702 -7.81722066  8.23591415
 -10.79235364]
 [  2.85385617  3.01086911  1.17921886  3.283334    13.1100137
 -3.48551645]
 ...
 [ -2.29815054  7.23607832 -2.53886278 -2.6828966  -5.91462164
 -5.6040925 ]
 [ -7.95087932 -7.86176175 -7.91306024  1.23651618 -13.64943576
 -13.56327907]
 [  3.25578131  1.11399395  2.5149983   18.36665666 -1.53488857
  0.88146156]]
```

Алгоритм Q-обучение

```
Вывод Q-матрицы для алгоритма Q-обучение
[[ 0.          0.          0.          0.          0.          0.
  0.          ]
 [  4.90946932  6.44412692  3.11497671  5.92595067  8.36234335 -2.94449909]
 [  9.31821858 10.70734193  8.84470896 10.74984506 13.27445578  2.65545226]
 ...
 [  0.92354858 12.40292008 -0.09307987  1.16382017 -5.28920988 -6.02338487]
 [-2.21381144 -1.77072531 -2.95799901  7.54164964 -9.36308927 -5.95207207]
 [  4.43845598  2.728849    8.81646326 18.59794943  0.8545733   2.53106673]]
```

Алгоритм двойное Q-обучение

```

Вывод Q-матриц для алгоритма Двойное Q-обучение
Q1
[[ 0.          0.          0.          0.          0.
  0.          ]
 [ 0.03879444  2.03511462 -2.52967576  0.66739653  8.36234335
 -5.04727032]
 [ 5.47475381  8.77669818  3.02390899  7.3278276  13.27445578
 -1.16827727]
 ...
 [ 6.76930641 14.5657712  4.5083121  5.45931852  2.66374495
 2.21038481]
 [ -3.46186265  2.26457269 -3.38031277 -2.60041054 -10.54557724
 -5.15309731]
 [ 3.72224538  4.42100656  1.87083552 18.17822734  0.48799718
 -1.96400176]]

Q2
[[ 0.          0.          0.          0.          0.          0.          ]
 [ 1.25301708  3.65355552 -3.24639715  2.62026556  8.36234335 -7.49386278]
 [ 4.8311577  4.06482313  2.93888801  6.84052853 13.27445578 -1.90779817]
 ...
 [ 9.6762051 14.5657712  7.45221976  5.15329804  1.87985424  1.41236416]
 [-3.97515029 3.83260412 -4.12713239 -0.91614667 -8.2192125  -8.15621667]
 [ 2.38757838 2.59278084  2.98911653 18.15425271 -0.1895167  -2.92785128]]

```

Вывод

В ходе выполнения лабораторной работы мы ознакомились с базовыми методами обучения с подкреплением на основе временных различий, а именно алгоритмами SARSA, Q-Learning, Double Q-Learning.