



**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)**

Курс «Методы машинного обучения»

Отчет по лабораторной работе №5:  
«Обучение на основе временных различий»

Выполнила:  
студентка группы ИУ5-24М  
Мащенко Е. И.

Проверил:  
Балашов А.М.

## **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 работы


Для реализации алгоритмов SARSA, Q-обучение и двойное Q-обучение была выбрана среда обучения Cliff Walking из библиотеки gym. Согласно документации, агент может находиться в 48 состояниях и осуществлять 4 действия.

### Лабораторная работа №5

B [1]:  `pip install gymnasium`

```
Requirement already satisfied: gymnasium in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (0.28.1)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (4.6.3)
Requirement already satisfied: jax-jumpy>=1.0.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (1.0.0)
Requirement already satisfied: numpy>=1.21.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (1.24.3)
Requirement already satisfied: importlib-metadata>=4.8.0; python_version < "3.10" in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (6.6.0)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (1.3.0)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from gymnasium) (0.0.4)
Requirement already satisfied: zipp>=0.5 in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (from importlib-metadata>=4.8.0; python_version < "3.10"->gymnasium) (3.15.0)
```

```
WARNING: You are using pip version 20.1.1; however, version 23.1.2 is available.
You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgrade pip' command.
```

B [14]:  `import gymnasium as gym
import numpy as np
from pprint import pprint
import matplotlib.pyplot as plt
from tqdm import tqdm`

B [3]: # \*\*\*\*\* 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
```

B [4]: # \*\*\*\*\* 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)
```

B [5]:  # \*\*\*\*\* 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

            # Прогрессирование 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)
```

B [6]:  # \*\*\*\*\* 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)
```

B [7]: `!pip install pygame`

Requirement already satisfied: pygame in c:\users\user\appdata\local\programs\python\python38\lib\site-packages (2.4.0)

WARNING: You are using pip version 20.1.1; however, version 23.1.2 is available.

You should consider upgrading via the 'c:\users\user\appdata\local\programs\python\python38\python.exe -m pip install --upgrade pip' command.

B [9]: `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
```

B [11]: `def run_sarsa():`

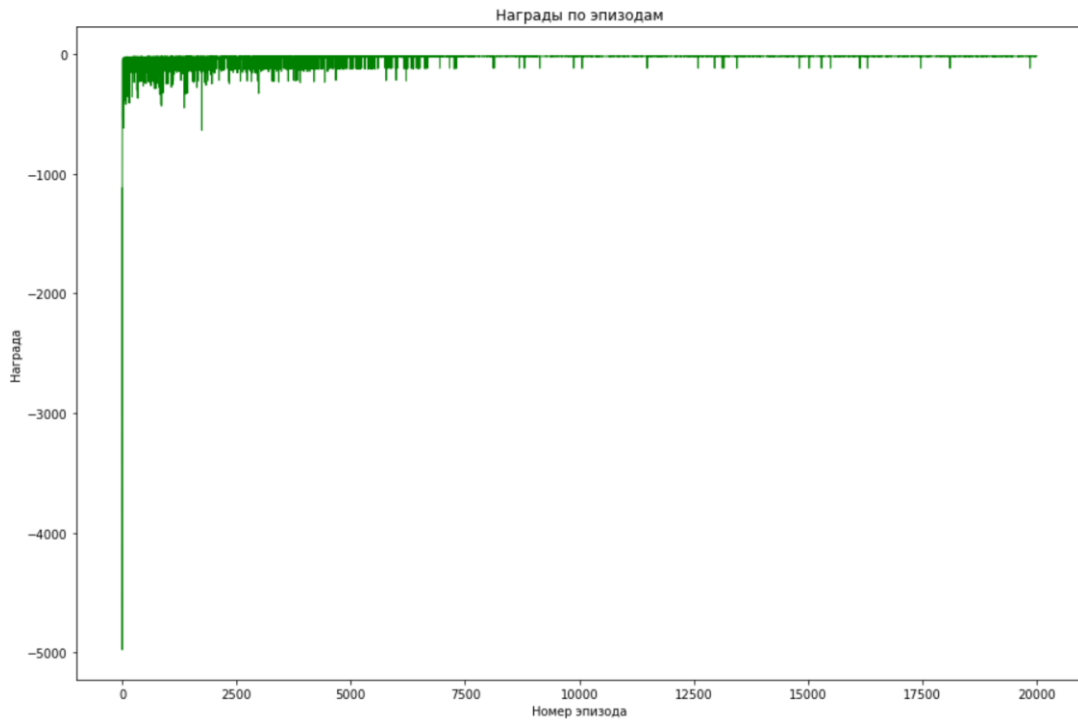
```
    env = gym.make('CliffWalking-v0')
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

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)

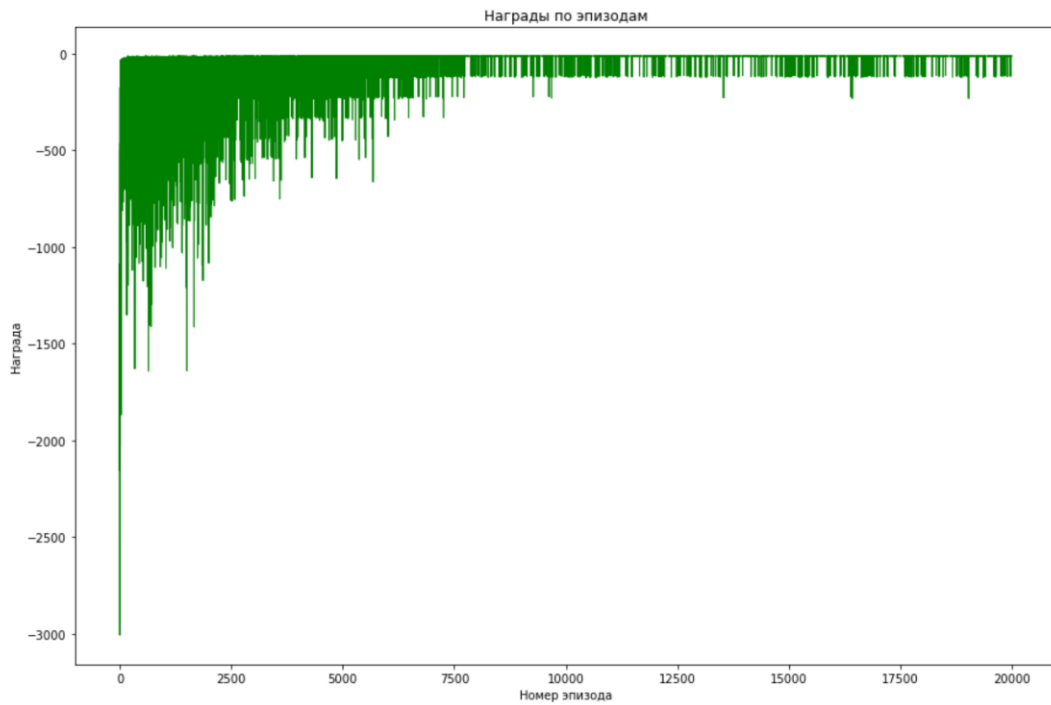
def run_double_q_learning():
    env = gym.make('CliffWalking-v0')
    agent = DoubleQLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)
```



### Результаты выполнения алгоритма SARSA:

[illegible]

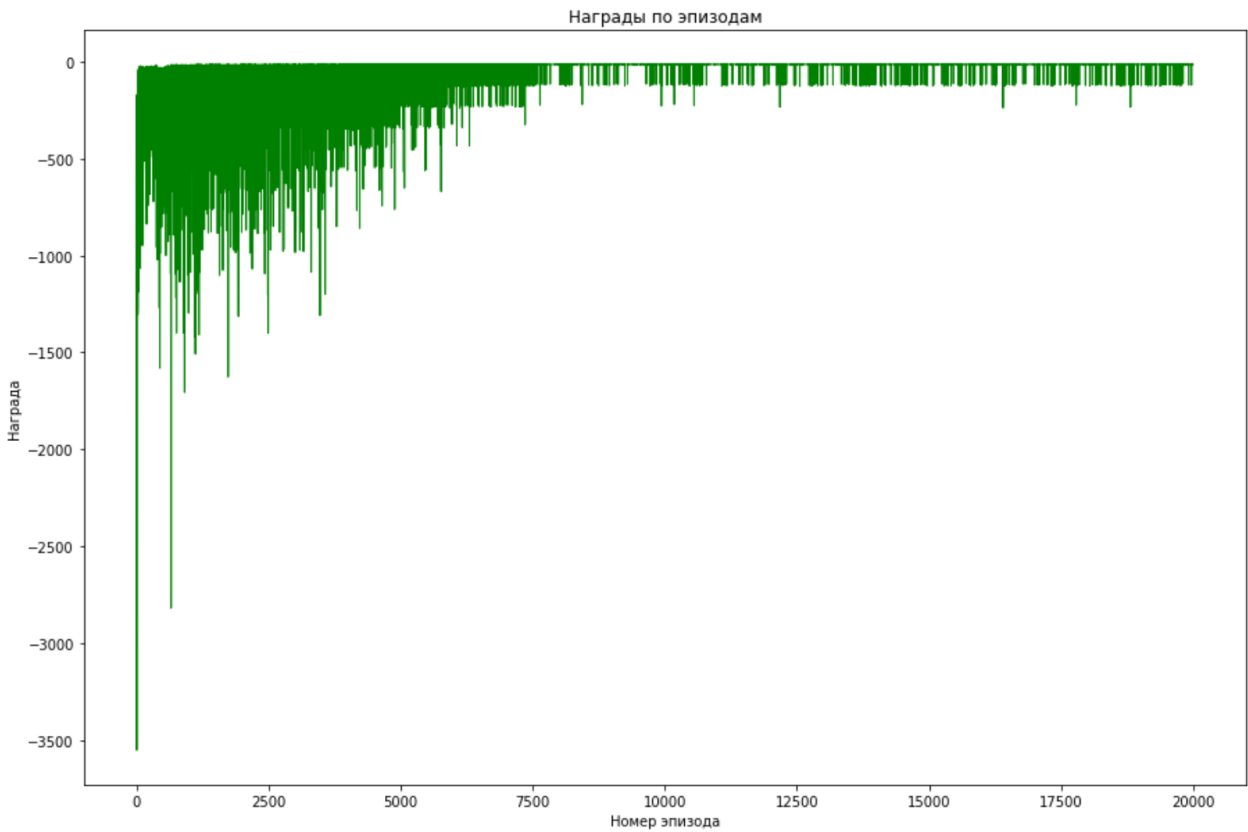
### Результаты выполнения алгоритма Q-обучение:

[illegible]

Результаты выполнения алгоритма двойное Q-обучение:

Вывод Q-матриц для алгоритма Двойное Q-обучение				
Q1				
[	[	-14.72368332	-14.44139918	-12.32201971
[	-13.84014705	-13.88274759	-11.61397923	-14.4834763
[	-13.01118045	-11.69981379	-10.76479727	-12.89794643
[	-11.2196401	-12.03426601	-10.00903399	-12.75786237
[	-10.36020818	-9.13769096	-8.09187603	-11.1847185
[	-8.89588977	-8.31365248	-8.56975067	-10.16000227
[	-8.08165511	-7.46310595	-7.46186547	-8.99673211
[	-7.04636956	-6.75166673	-6.59372208	-8.05680844
[	-6.28626435	-5.69877612	-5.7078722	-7.02035336
[	-5.01945141	-4.93186079	-4.93115864	-5.44802543
[	-4.31967822	-3.87330027	-3.72845749	-4.28719339
[	-3.62824656	-3.66345184	-2.94043034	-3.8814804
[	-13.17169285	-11.58379149	-11.54888054	-12.32203364
[	-12.39268404	-10.79588252	-10.76416381	-12.33282328
[	-11.8812196	-10.18572479	-9.96343246	-11.57839974
[	-10.74310069	-9.15323511	-9.14635966	-10.76437316
[	-12.3782129	-8.33887401	-8.40789865	-10.01311137

Q2				
[	[	-14.72077416	-15.0800717	-12.33951287
[	-14.71444435	-13.65796101	-11.55697814	-14.72033705
[	-12.82335539	-13.41573727	-10.86946365	-14.9287179
[	-11.98949528	-10.8853534	-9.91980932	-11.46147513
[	-9.36302299	-9.41417943	-11.41533546	-10.16645376
[	-8.79614702	-8.3127866	-8.24830202	-10.4880784
[	-8.21460256	-7.61348758	-7.46184887	-9.02102836
[	-7.41234814	-6.54418667	-6.59374686	-7.84301672
[	-6.41120013	-5.80258036	-5.70785737	-7.2587489
[	-5.35330181	-4.84773028	-4.79859502	-4.8826891
[	-4.0276935	-3.88011614	-4.03354006	-4.72920931
[	-3.18708222	-3.36695608	-2.9403798	-3.83303216
[	-13.13615202	-11.56699018	-11.54888054	-12.3330435
[	-12.58387206	-10.8054778	-10.76416381	-12.34855159
[	-11.65754981	-9.97855505	-9.96343246	-11.55181744
[	-11.16808309	-9.48967342	-9.14635966	-10.86120421
[	-11.08957629	-8.31261813	-8.28321184	-9.92502949
[	-9.41318239	-7.46184887	-7.4886378	-9.41597598



Пример агента в конечном состоянии:

