Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»


Лабораторная работа №7
по дисциплине
# «Методы машинного обучения»
на тему

# «RL_TD»

Выполнил:
студент группы ИУ5и-24М
Аунг Пьио Нанда

Москва — 2024 г.

# 1. Цель лабораторной работы

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

# 2. 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).

# 3. Ход выполнения работы

Импорт необходимых библиотек

```
!pip install gym numpy matplotlib
```

```
Requirement already satisfied: gym in /usr/local/lib/python3.10/dist-packages (0.25.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym) (0.0.8)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.52.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
[9] import gym
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[10] # Определение среды
     env = gym.make('CartPole-v1')
```

```
[11] # Дискретизация пространства состояний
     n_bins = (6, 12, 6, 12)  # количество бинов для каждого измерения
     lower_bounds = [env.observation_space.low[0], -0.5, env.observation_space.low[2], -np.radians(50)]
     upper_bounds = [env.observation_space.high[0], 0.5, env.observation_space.high[2], np.radians(50)]
```

```
[12] def discretize_state(state, n_bins, lower_bounds, upper_bounds):
         ratios = [(state[i] - lower_bounds[i]) / (upper_bounds[i] - lower_bounds[i]) for i in range(len(state))]
         new_state = [int(round((n_bins[i] - 1) * ratios[i])) for i in range(len(state))]
         new_state = [min(n_bins[i] - 1, max(0, new_state[i])) for i in range(len(state))]
         return tuple(new_state)
```

Алгоритм SARSA обновляет Q-значения на основе текущего состояния, действия, следующего состояния и следующего действия.

```python
def sarsa(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds):
    q_table = np.zeros(n_bins + (env.action_space.n,))

    def choose_action(state, epsilon):
        if np.random.random() < epsilon:
            return env.action_space.sample()
        else:
            return np.argmax(q_table[state])

    for episode in range(episodes):
        state = discretize_state(env.reset(), n_bins, lower_bounds, upper_bounds)
        action = choose_action(state, epsilon)

        done = False
        while not done:
            next_state_raw, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state_raw, n_bins, lower_bounds, upper_bounds)
            next_action = choose_action(next_state, epsilon)

            q_table[state + (action,)] += alpha * (reward + gamma * q_table[next_state + (next_action,)] - q_table[state + (action,)])

            state = next_state
            action = next_action

    return q_table
```

Q-обучение обновляет Q-значения на основе текущего состояния и действия, выбирая максимальное Q-значение для следующего состояния.

```python
[14] def q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds):
        q_table = np.zeros(n_bins + (env.action_space.n,))

        def choose_action(state, epsilon):
            if np.random.random() < epsilon:
                return env.action_space.sample()
            else:
                return np.argmax(q_table[state])

        for episode in range(episodes):
            state = discretize_state(env.reset(), n_bins, lower_bounds, upper_bounds)

            done = False
            while not done:
                action = choose_action(state, epsilon)
                next_state_raw, reward, done, _ = env.step(action)
                next_state = discretize_state(next_state_raw, n_bins, lower_bounds, upper_bounds)

                q_table[state + (action,)] += alpha * (reward + gamma * np.max(q_table[next_state]) - q_table[state + (action,)])

                state = next_state

        return q_table
```

В Двойном Q-обучении используются две Q-таблицы для уменьшения смещения.

```
def double_q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds):
    q_table1 = np.zeros(n_bins + (env.action_space.n,))
    q_table2 = np.zeros(n_bins + (env.action_space.n,))

    def choose_action(state, epsilon):
        if np.random.random() < epsilon:
            return env.action_space.sample()
        else:
            return np.argmax(q_table1[state] + q_table2[state])

    for episode in range(episodes):
        state = discretize_state(env.reset(), n_bins, lower_bounds, upper_bounds)

        done = False
        while not done:
            action = choose_action(state, epsilon)
            next_state_raw, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state_raw, n_bins, lower_bounds, upper_bounds)

            if np.random.random() < 0.5:
                best_next_action = np.argmax(q_table1[next_state])
                q_table1[state + (action,)] += alpha * (reward + gamma * q_table2[next_state + (best_next_action,)] - q_table1[state + (action,)])
            else:
                best_next_action = np.argmax(q_table2[next_state])
                q_table2[state + (action,)] += alpha * (reward + gamma * q_table1[next_state + (best_next_action,)] - q_table2[state + (action,)])

            state = next_state

    return q_table1 + q_table2
```

## Пример использования

```
[19] # Параметры обучения
episodes = 1000
alpha = 0.1
gamma = 0.99
epsilon = 0.1
```

```
[20] # Обучение
q_table_sarsa = sarsa(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
q_table_q_learning = q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
q_table_double_q = double_q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
```

```
# Оценка политики
def evaluate_policy(env, q_table, episodes=100):
    total_rewards = []
    for _ in range(episodes):
        state = discretize_state(env.reset(), n_bins, lower_bounds, upper_bounds)
        done = False
        total_reward = 0
        while not done:
            action = np.argmax(q_table[state])
            next_state_raw, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state_raw, n_bins, lower_bounds, upper_bounds)
            total_reward += reward
            state = next_state
        total_rewards.append(total_reward)
    return np.mean(total_rewards)
```

# Вывод результатов

```
[19] # Параметры обучения
     episodes = 1000
     alpha = 0.1
     gamma = 0.99
     epsilon = 0.1
```

```
[20] # Обучение
     q_table_sarsa = sarsa(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
     q_table_q_learning = q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
     q_table_double_q = double_q_learning(env, episodes, alpha, gamma, epsilon, n_bins, lower_bounds, upper_bounds)
```

```
     # Оценка политики
     def evaluate_policy(env, q_table, episodes=100):
         total_rewards = []
         for _ in range(episodes):
             state = discretize_state(env.reset(), n_bins, lower_bounds, upper_bounds)
             done = False
             total_reward = 0
             while not done:
                 action = np.argmax(q_table[state])
                 next_state_raw, reward, done, _ = env.step(action)
                 next_state = discretize_state(next_state_raw, n_bins, lower_bounds, upper_bounds)
                 total_reward += reward
                 state = next_state
             total_rewards.append(total_reward)
         return np.mean(total_rewards)
```

```
[22] mean_reward_sarsa = evaluate_policy(env, q_table_sarsa)
     mean_reward_q_learning = evaluate_policy(env, q_table_q_learning)
     mean_reward_double_q = evaluate_policy(env, q_table_double_q)
```

```
[23] print(f"SARSA mean reward: {mean_reward_sarsa}")
```

```
SARSA mean reward: 9.63
```

```
[26] print(f"Q-Learning mean reward: {mean_reward_q_learning}")
```

```
Q-Learning mean reward: 10.85
```

```
     print(f"Double Q-Learning mean reward: {mean_reward_double_q}")
```

```
Double Q-Learning mean reward: 10.92
```

SARSA (State-Action-Reward-State-Action):

Это метод временных различий, который использует текущую политику для выбора действий. Обновление Q-значений основывается на действиях, выбранных текущей политикой.

Q-обучение:

Это метод временных различий, который использует наилучшие возможные действия для обновления Q-значений. Использует жадное обновление на основе максимальных Q-значений для следующего состояния.

Двойное Q-обучение:

Использует две отдельные Q-таблицы для уменьшения смещения.Обновляет значения попеременно используя обе Q-таблицы, выбирая максимальные значения из одной и обновляя значения в другой.

# Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Разведочный анализ данных. Исследование и визуализация данных» [Электронный ресурс] // GitHub. — 2024. — Режим доступа: https://github.com/ugapanyuk/courses_current/wiki/LAB_MMO___ RL_TD.