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

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему

«Обучение на основе временны’х различий»

Выполнила:
студентка Цзян Юхуэй

группы ИУ5И-23М

Москва — 2024 г.

Цель лабораторной 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).

Часть 1. SARSA

```
def sarsa(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1, num_bins=10):
    bins = create_bins(num_bins)
    Q = defaultdict(lambda: np.zeros(env.action_space.n))

    def epsilon_greedy_policy(state):
        if np.random.rand() < epsilon:
            return env.action_space.sample()
        else:
            return np.argmax(Q[state])

    for episode in range(num_episodes):
        state = discretize_state(env.reset(), bins)
        action = epsilon_greedy_policy(state)

        done = False
        while not done:
            next_state, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state, bins)
            next_action = epsilon_greedy_policy(next_state)

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

            state, action = next_state, next_action

    return Q
```

```
env = gym.make('CartPole-v1')

num_episodes = 500

Q_sarsa = sarsa(env, num_episodes)
```

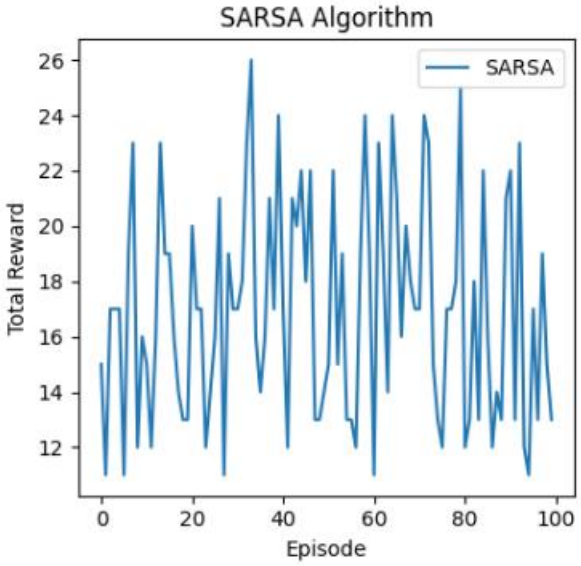

Рис 1. SARSA Algorithm.

Часть 2. Q-обучение

```
def q_learning(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1, num_bins=10):
    bins = create_bins(num_bins)
    Q = defaultdict(lambda: np.zeros(env.action_space.n))

    def epsilon_greedy_policy(state):
        if np.random.rand() < epsilon:
            return env.action_space.sample()
        else:
            return np.argmax(Q[state])

    for episode in range(num_episodes):
        state = discretize_state(env.reset(), bins)

        done = False
        while not done:
            action = epsilon_greedy_policy(state)
            next_state, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state, bins)

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

            state = next_state

    return Q
```

```
Q_q_learning = q_learning(env, num_episodes)
```

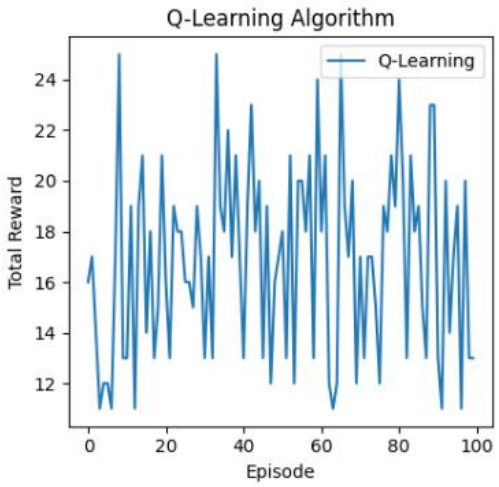

Рис 2. Q-learning Algorithm.

Часть 3. Двойное Q-обучение

```
def double_q_learning(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1, num_bins=10):
    bins = create_bins(num_bins)
    Q1 = defaultdict(lambda: np.zeros(env.action_space.n))
    Q2 = defaultdict(lambda: np.zeros(env.action_space.n))

    def epsilon_greedy_policy(state):
        if np.random.rand() < epsilon:
            return env.action_space.sample()
        else:
            return np.argmax(Q1[state] + Q2[state])

    for episode in range(num_episodes):
        state = discretize_state(env.reset(), bins)

        done = False
        while not done:
            action = epsilon_greedy_policy(state)
            next_state, reward, done, _ = env.step(action)
            next_state = discretize_state(next_state, bins)

            if np.random.rand() < 0.5:
                Q1[state][action] += alpha * (reward + gamma * Q2[next_state][np.argmax(Q1[next_state])] - Q1[state][action])
            else:
                Q2[state][action] += alpha * (reward + gamma * Q1[next_state][np.argmax(Q2[next_state])] - Q2[state][action])
```

```
            state = next_state

    return {state: Q1[state] + Q2[state] for state in Q1}

Q_double_q_learning = double_q_learning(env, num_episodes)
```

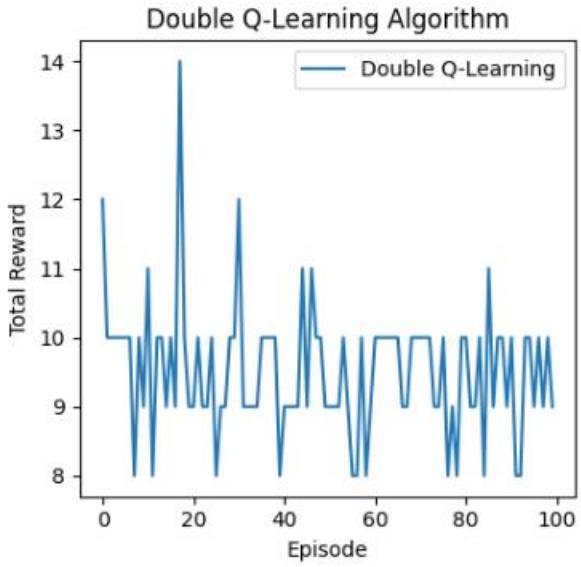

Рис 3. Double Q-Learning Algorithm.