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


# Лабораторная работа № 5

# По курсу «методы машинного обучения в АСОИУ»

## «Обучение на основе временных различий»

**Выполнил:**

студент ИУ5-23М
Семенов И.А.

**Проверил:**

Балашов А.М.

Подпись:

29.04.2024

Москва, 2024

**Описание задания**

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).

**Ход работы:**

**Алгоритм SARSA** (State-Action-Reward-State-Action) — это метод обучения с подкреплением, который использует функции действия и состояния для выбора действий и обновления значений на основе полученной награды. SARSA — это метод табличного обучения, который обновляет оценку ценности состояния и действия (Q-значение) на основе наблюдений.

**Параметры:**

alpha (темп обучения): Контролирует скорость обучения. Большое значение alpha приводит к более быстрым изменениям в оценках Q-значений, в то время как низкое значение alpha позволяет оценкам Q-значений сходиться более постепенно.

gamma (коэффициент дисконтирования): Определяет, насколько агент уделяет внимание будущим наградам по сравнению с текущей наградой.

epsilon (эпсилон): Параметр для эпсилон-жадной стратегии. Определяет вероятность выбора случайного действия для обеспечения исследовательского поведения.

**Алгоритм работы:**

Инициализация: Создание матрицы Q-значений для всех состояний и действий и инициализация случайной политики.

Повторение для каждого эпизода:

Инициализация: Агент начинает в начальном состоянии среды.

Цикл: Пока эпизод не завершен:

Агент выбирает действие, используя эпсилон-жадную стратегию, основываясь на Q-значениях текущего состояния.

Агент выполняет действие, получает награду и наблюдает новое состояние.

Агент выбирает новое действие, используя эпсилон-жадную стратегию, основываясь на Q-значениях нового состояния.

Агент обновляет Q-значения текущего состояния и действия, используя формулу:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$$

Агент переходит в новое состояние и продолжает цикл.
Конец эпизода: Повторять шаг 2 до тех пор, пока эпизод не завершится.

**Среда выполнения**

**Да, в среде `Taxi-v3`, которую мы используем для алгоритма SARSA, агент по-прежнему выполняет задачу перемещения такси, подбирая и доставляя пассажиров к месту назначения. Это одна из основных целей среды `Taxi-v3` — такси должно перемещаться по городу, подбирать пассажиров из их начальных местоположений и доставлять их к местам назначения.**

**Вот что агент делает в среде `Taxi-v3`:**

1. Перемещается по городу: Агент перемещается в четырех направлениях (вверх, вниз, влево и вправо) по сетке размером 5x5.
2. Подбирает пассажиров: Агент должен найти пассажира в определенной позиции на сетке и подобрать его.
3. Доставляет пассажиров: После того как агент подбирает пассажира, он должен отвезти его к определенному месту назначения.
4. Получает награду: Агент получает награду за успешное выполнение действий, например, за подъем пассажира и успешную доставку его к месту назначения. Он также может получить отрицательную награду за неудачные действия, например, попытку двигаться за пределы карты.

Алгоритм SARSA позволяет агенту обучаться тому, как эффективно выполнять эту задачу, максимизируя награду за время работы. Агент будет изучать оптимальные действия в каждом состоянии и обновлять свои оценки (Q-значения) на основе опыта, чтобы достичь цели более эффективно.

## Текст программы и экранные формы с примерами выполнения программы

✓ 59 сек.

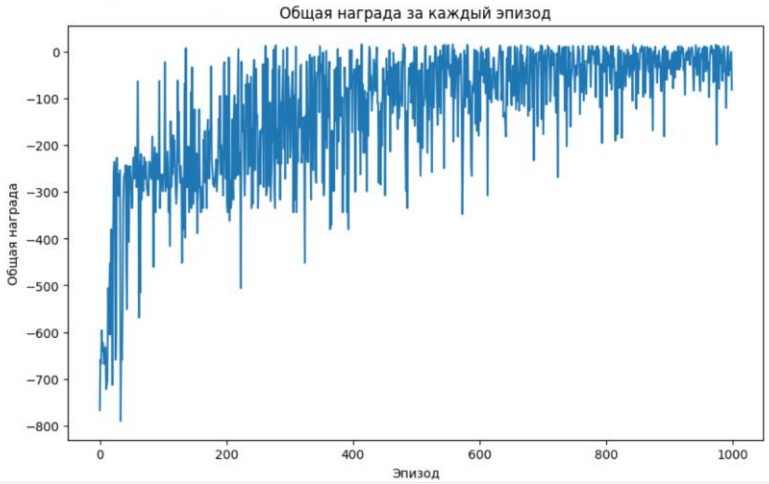
```python
import gym
import numpy as np
import matplotlib.pyplot as plt

def sarsa(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1):
    """
    Алгоритм SARSA.

    Параметры:
    env (gym.Env): Среда обучения с подкреплением.
    num_episodes (int): Количество эпизодов для обучения.
    alpha (float): Темп обучения.
    gamma (float): Коэффициент дисконтирования.
    epsilon (float): Параметр эпсилон-жадной стратегии.

    Возвращает:
    Q (np.array): Матрица Q-значений для всех состояний и действий.
    rewards (list): Список общей награды за каждый эпизод.
    """
    # Инициализация Q-матрицы (значения изначально нулевые)
    Q = np.zeros((env.observation_space.n, env.action_space.n))

    # Список для хранения награды за каждый эпизод
    rewards = []

    # Повторение для каждого эпизода
    for episode in range(num_episodes):
        # Инициализация начального состояния
        state = env.reset()
        done = False
        total_reward = 0

        # Выбор начального действия
        if np.random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])

        # Цикл внутри эпизода
        while not done:
            # Выполнение действия и получение награды и нового состояния
            next_state, reward, done, _ = env.step(action)

            # Выбор следующего действия
            if np.random.uniform(0, 1) < epsilon:
                next_action = env.action_space.sample()
```

59 сек.

```
            else:
                next_action = np.argmax(Q[next_state])

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

            # Переход к новому состоянию и действию
            state = next_state
            action = next_action

            # Увеличение общей награды эпизода
            total_reward += reward

        # Сохранение общей награды эпизода
        rewards.append(total_reward)

    return Q, rewards

# Главный код для запуска алгоритма
if __name__ == "__main__":
    # Создание среды Taxi-v3
    env = gym.make('Taxi-v3')

    # Количество эпизодов для обучения
    num_episodes = 1000

    # Запуск алгоритма SARSA
    Q, rewards = sarsa(env, num_episodes)

    # График общей награды за каждый эпизод
    plt.figure(figsize=(10, 6))
    plt.plot(rewards)
    plt.xlabel('Эпизод')
    plt.ylabel('Общая награда')
    plt.title('Общая награда за каждый эпизод')
    plt.show()

    # Тестирование обученной модели
    state = env.reset()
    total_reward = 0
    done = False

    while not done:
        # Выбор действия, основываясь на Q-значениях
        action = np.argmax(Q[state])
```

```
        # Выполнение действия
        state, reward, done, _ = env.step(action)

        # Увеличение общей награды
        total_reward += reward

        # Отображение среды
        env.render()

    print(f"Общая награда: {total_reward}")
    env.close()
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initializing wrapper in old step API which returns one bool instead of two. It is recomme
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning: WARN: Initializing environment in old step API which returns one bool
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool_`.  (Deprecated NumPy 1.24)
  if not isinstance(terminated, (bool, np.bool8)):
```




## Алгоритм Q-обучение

**Алгоритм Q-обучения** — это метод обучения с подкреплением, который позволяет агенту учиться принимать оптимальные решения, максимизируя награду в среде. Алгоритм основывается на использовании функции ценности (Q-значений) для состояний и действий.

**Параметры**:

alpha (темп обучения): Скорость, с которой агент обновляет свои Q-значения на основе нового опыта. Более высокое значение alpha приводит к более быстрым изменениям в оценках Q-значений, а более низкое значение позволяет оценкам сходиться более постепенно.

gamma (коэффициент дисконтирования): Определяет, насколько агент обращает внимание на будущие награды по сравнению с текущей наградой.

epsilon (эпсилон): Параметр для эпсилон-жадной стратегии. Определяет вероятность выбора случайного действия для обеспечения исследовательского поведения.

**Алгоритм работы:**

Инициализация: Создание матрицы Q-значений для всех состояний и действий, и инициализация случайной политики.

Повторение для каждого эпизода:

Инициализация: Агент начинает в начальном состоянии среды.

Цикл: Пока эпизод не завершен:

Агент выбирает действие, используя эпсилон-жадную стратегию (выбирает случайное действие с вероятностью epsilon или наилучшее известное

действие на основе Q-значений).
Агент выполняет действие и получает награду и новое состояние.
Агент обновляет Q-значение для текущего состояния и действия, используя формулу:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max Q(s',a') - Q(s,a)\right)$$

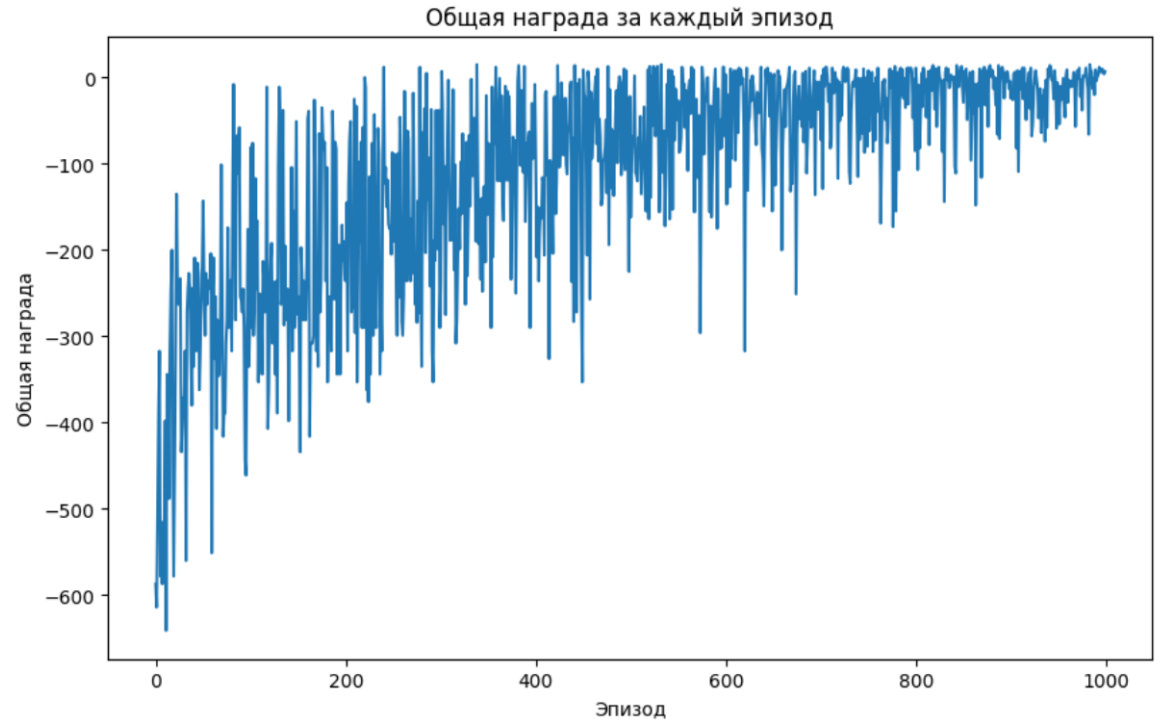
Агент переходит в новое состояние.
Конец эпизода: Повторяет до тех пор, пока эпизод не завершится.

```python
import gym
import numpy as np
import matplotlib.pyplot as plt

def q_learning(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1):
    """
    Алгоритм Q-обучения.

    Параметры:
    env (gym.Env): Среда обучения с подкреплением.
    num_episodes (int): Количество эпизодов для обучения.
    alpha (float): Темп обучения.
    gamma (float): Коэффициент дисконтирования.
    epsilon (float): Параметр эпсилон-жадной стратегии.

    Возвращает:
    Q (np.array): Матрица Q-значений для всех состояний и действий.
    rewards (list): Список общей награды за каждый эпизод.
    """
    # Инициализация Q-матрицы (начальное Q-значение нулевое)
    Q = np.zeros((env.observation_space.n, env.action_space.n))

    # Список для хранения награды за каждый эпизод
    rewards = []

    # Повторение для каждого эпизода
    for episode in range(num_episodes):
        # Инициализация начального состояния
        state = env.reset()
        done = False
        total_reward = 0

        # Цикл внутри эпизода
        while not done:
            # Эпсилон-жадная стратегия для выбора действия
            if np.random.uniform(0, 1) < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(Q[state])

            # Выполнение действия и получение награды и нового состояния
            next_state, reward, done, _ = env.step(action)

            # Обновление Q-значения
            Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state, action])
```

✓ 9 сек.

```
            # Переход в новое состояние
            state = next_state

            # Увеличение общей награды эпизода
            total_reward += reward

        # Сохранение общей награды эпизода
        rewards.append(total_reward)

    return Q, rewards

# Главный код для запуска алгоритма
if __name__ == "__main__":
    # Создание среды Taxi-v3
    env = gym.make('Taxi-v3')

    # Количество эпизодов для обучения
    num_episodes = 1000

    # Запуск алгоритма Q-обучения
    Q, rewards = q_learning(env, num_episodes)

    # График общей награды за каждый эпизод
    plt.figure(figsize=(10, 6))
    plt.plot(rewards)
    plt.xlabel('Эпизод')
    plt.ylabel('Общая награда')
    plt.title('Общая награда за каждый эпизод')
    plt.show()

    # Тестирование обученной модели
    state = env.reset()
    total_reward = 0
    done = False

    while not done:
        # Выбор действия, основываясь на Q-значениях
        action = np.argmax(Q[state])

        # Выполнение действия
        state, reward, done, _ = env.step(action)

        # Увеличение общей награды
        total_reward += reward

        # Отображение среды
        env.render()
```

```
print(f"Общая награда: {total_reward}")
env.close()
```




```
Общая награда: 11
```

## Двойное Q-обучение

Двойное Q-обучение — это вариант стандартного алгоритма Q-обучения, направленный на уменьшение переоценки значений Q-функции. В стандартном Q-обучении агент использует текущие Q-значения для оценки как текущего состояния, так и нового состояния, что может привести к переоценке будущих наград.

В Двойном Q-обучении используется два набора Q-значений (`Q1` и `Q2`) для более точной оценки будущих наград:

- Q1 и Q2: Два массива Q-значений для всех состояний и действий.

- Выбор действия: Агент выбирает действие на основе случайного выбора между `Q1` и `Q2`.

- Обновление Q-значений: Если действие было выбрано из `Q1`, обновляется `Q2` с использованием лучшего действия из `Q1` в новом состоянии, и наоборот. Это помогает снизить смещение при оценке Q-значений.

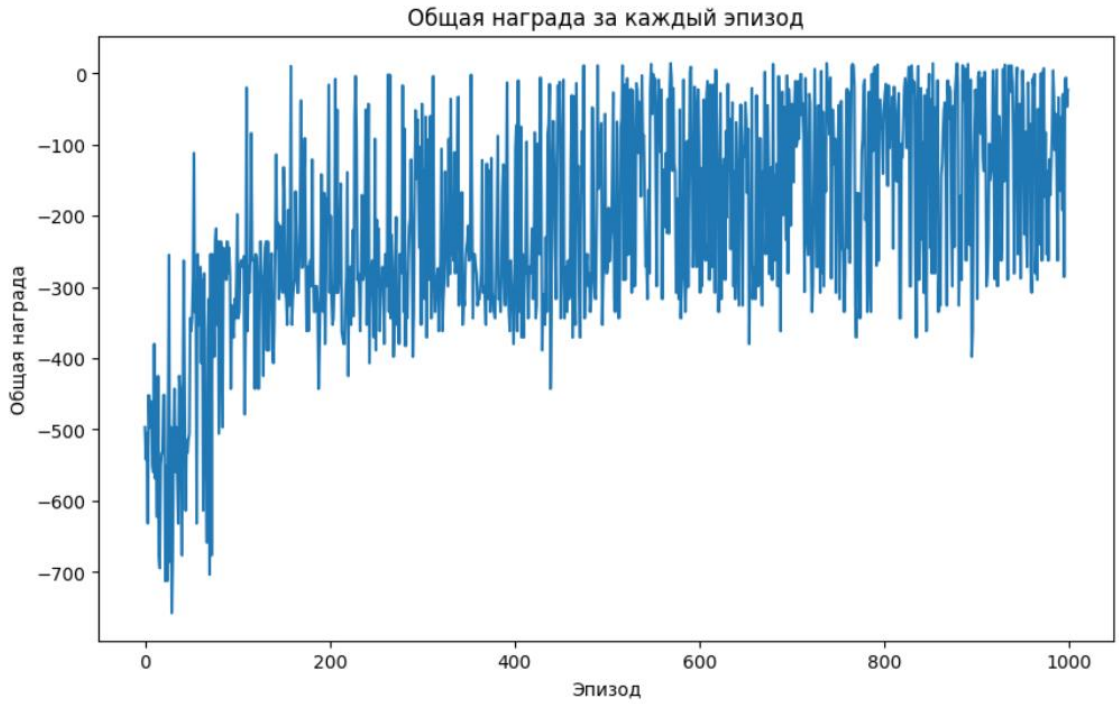
- Параметры: В остальном, параметры (alpha, gamma, epsilon) остаются такими же, как в стандартном Q-обучении.

✓ 1 мин.

```python
import gym
import numpy as np
import matplotlib.pyplot as plt

def double_q_learning(env, num_episodes, alpha=0.1, gamma=0.99, epsilon=0.1):
    """
    Алгоритм Двойного Q-обучения.

    Параметры:
    env (gym.Env): Среда обучения с подкреплением.
    num_episodes (int): Количество эпизодов для обучения.
    alpha (float): Темп обучения.
    gamma (float): Коэффициент дисконтирования.
    epsilon (float): Параметр эпсилон-жадной стратегии.

    Возвращает:
    Q1 (np.array): Первый набор Q-значений для всех состояний и действий.
    Q2 (np.array): Второй набор Q-значений для всех состояний и действий.
    rewards (list): Список общей награды за каждый эпизод.
    """
    # Инициализация Q1 и Q2-матриц (начальные Q-значения нулевые)
    Q1 = np.zeros((env.observation_space.n, env.action_space.n))
    Q2 = np.zeros((env.observation_space.n, env.action_space.n))

    # Список для хранения награды за каждый эпизод
    rewards = []

    # Повторение для каждого эпизода
    for episode in range(num_episodes):
        # Инициализация начального состояния
        state = env.reset()
        done = False
        total_reward = 0

        # Цикл внутри эпизода
        while not done:
            # Эпсилон-жадная стратегия для выбора действия
            if np.random.uniform(0, 1) < epsilon:
                action = env.action_space.sample()
            else:
                if np.random.uniform(0, 1) < 0.5:
                    action = np.argmax(Q1[state])
                else:
                    action = np.argmax(Q2[state])

            # Выполнение действия и получение награды и нового состояния
            next_state, reward, done, _ = env.step(action)
```

```
            # Обновление Q1 и Q2-значений
            if np.random.uniform(0, 1) < 0.5:
                # Обновление Q1-значений
                best_next_action = np.argmax(Q1[next_state])
                target = reward + gamma * Q2[next_state, best_next_action]
                Q1[state, action] += alpha * (target - Q1[state, action])
            else:
                # Обновление Q2-значений
                best_next_action = np.argmax(Q2[next_state])
                target = reward + gamma * Q1[next_state, best_next_action]
                Q2[state, action] += alpha * (target - Q2[state, action])

            # Переход в новое состояние
            state = next_state

            # Увеличение общей награды эпизода
            total_reward += reward

        # Сохранение общей награды эпизода
        rewards.append(total_reward)

    return Q1, Q2, rewards

# Главный код для запуска алгоритма
if __name__ == "__main__":
    # Создание среды Taxi-v3
    env = gym.make('Taxi-v3')

    # Количество эпизодов для обучения
    num_episodes = 1000

    # Запуск алгоритма Двойного Q-обучения
    Q1, Q2, rewards = double_q_learning(env, num_episodes)

    # График общей награды за каждый эпизод
    plt.figure(figsize=(10, 6))
    plt.plot(rewards)
    plt.xlabel('Эпизод')
    plt.ylabel('Общая награда')
    plt.title('Общая награда за каждый эпизод')
    plt.show()

    # Тестирование обученной модели
    state = env.reset()
    total_reward = 0
    done = False
```

```python
# Тестирование обученной модели
state = env.reset()
total_reward = 0
done = False

while not done:
    # Выбор действия на основе среднего значения Q1 и Q2
    action = np.argmax(Q1[state] + Q2[state])

    # Выполнение действия
    state, reward, done, _ = env.step(action)

    # Увеличение общей награды
    total_reward += reward

    # Отображение среды
    env.render()

print(f"Общая награда: {total_reward}")
env.close()
```




```
Общая награда: -200
```

В этом коде реализуется алгоритм Двойного Q-обучения в среде `Taxi-v3`. Алгоритм использует два набора Q-значений (`Q1` и `Q2`) для более точной оценки Q-значений и уменьшения риска переоценки будущих наград. Код визуализирует общую награду за каждый эпизод обучения и позволяет агенту более эффективно выполнять задачу перемещения такси в среде.

**Вывод**

Все три модели (Policy Iteration, SARSA и Q-обучение, включая его двойной вариант) успешно решают задачу перемещения такси в среде `Taxi-v3`, где агент должен подбирать пассажиров и доставлять их к месту назначения. Policy Iteration сосредоточен на создании и улучшении политики агента для максимизации награды, в то время как SARSA и Q-обучение используют оценку Q-функции для оптимизации действий агента. Двойное Q-обучение представляет собой улучшенный вариант стандартного Q-обучения, который снижает переоценку будущих наград, используя два набора Q-значений. В

целом, все модели продемонстрировали эффективность в обучении агента в среде `Taxi-v3`, помогая ему научиться выбирать действия, которые приводят к максимальной награде.