МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа № 6
по дисциплине «Методы машинного обучения»

Тема: « Обучение на основе глубоких Q-сетей.»

ИСПОЛНИТЕЛЬ: Ли Яцзинь
ФИО
группа ИУ5И-21М
подпись"

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю .Е

Москва - 2024

Задание:

На основе рассмотренных на лекции примеров реализуйте алгоритм DQN.

В качестве среды можно использовать классические среды (в этом случае используется полносвязная архитектура нейронной сети).

В качестве среды можно использовать игры Atari (в этом случае используется сверточная архитектура нейронной сети).

1. **Создание среды**:

игровая среда с непрерывным пространством состояний и дискретным пространством действий была создана с использованием среды CartPole-v1 в OpenAI Gym.

```
import numpy as np
import tensorflow as tf
import gym
import matplotlib.pyplot as plt

# 创建环境
env = gym.make('CartPole-v1')
state_size = env.observation_space.shape[0]
action_size = env.action_space.n
```

Рисунок 1- Код для импорта библиотеки и создания среды

2. Определение модели DQN.

Для аппроксимации функции Q была создана модель нейронной сети, содержащая три полностью связанных слоя. Эта модель принимает состояния в качестве входных данных и выводит значение Q для каждого действия.

```python
# 定义DQN模型
class DQN(tf.keras.Model):
    def __init__(self, state_size, action_size):
        super(DQN, self).__init__()
        self.fc1 = tf.keras.layers.Dense(24, activation='relu')
        self.fc2 = tf.keras.layers.Dense(24, activation='relu')
        self.fc3 = tf.keras.layers.Dense(action_size)

    def call(self, inputs):
        x = self.fc1(inputs)
        x = self.fc2(x)
        return self.fc3(x)

# 初始化DQN模型和优化器
```

Рисунок 2-Определить код модели DQN

3. **Определите эпсилон-жадную стратегию**.

Эпсилон-жадная стратегия определена для выбора действий во время обучения. На каждом временном шаге выбирайте случайное действие, основанное на вероятности эпсилона исследовать окружающую среду, в противном случае выбирайте оптимальное действие на основе текущего значения Q.

```python
# 初始化DQN模型和优化器
model = DQN(state_size, action_size)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# 定义epsilon-greedy策略
def epsilon_greedy_policy(state, epsilon=0):
    if np.random.rand() < epsilon:
        return np.random.randint(action_size)
    else:
        q_values = model.predict(state[np.newaxis])
        return np.argmax(q_values[0])
```

Рисунок 3- Определить эпсилон-жадную стратегию

4. Обучение модели DQN.

Функция train_dqn предназначена для обучения модели DQN. В каждом эпизоде выполните следующие действия:

Сбросьте окружение и получите исходное состояние.

На каждом временном шаге действие выбирается на основе эпсилон-жадной политики и выполняется.

Рассчитайте целевое значение Q на основе состояния и немедленного вознаграждения после выполнения действия.

Рассчитайте потерю, используя значение Q текущего состояния и целевое значение Q, и обновите параметры модели нейронной сети.

Обновите значение эпсилона, чтобы постепенно снизить вероятность исследования.

Запишите общую награду текущего эпизода.

После каждого эпизода строятся кривые оценок, чтобы визуализировать изменения вознаграждения во время тренировки.

Таким образом, модель DQN изучает оптимальную политику посредством взаимодействия с окружающей средой, чтобы максимизировать совокупное вознаграждение. Изменения в кривой оценок могут отражать изменения в производительности, полученные моделью во время обучения.

```python
def train_dqn(env, model, optimizer, epsilon_decay=0.995, min_epsilon=0.01,
              gamma=0.99, max_episodes=270, max_steps_per_episode=1000, batch_size=64):
    epsilon = 1.0
    episode_rewards = []  # 用于存储每个episode的得分
    for episode in range(max_episodes):
        state = env.reset()
        episode_reward = 0
        for step in range(max_steps_per_episode):
            action = epsilon_greedy_policy(state, epsilon)
            next_state, reward, done, _ = env.step(action)
            episode_reward += reward

            # 计算target Q值
            target = reward + (1 - done) * gamma * np.max(model.predict(next_state[np.newaxis]))

            # 计算当前状态的Q值
            with tf.GradientTape() as tape:
                q_values = model(state[np.newaxis])
                action_mask = tf.one_hot(action, action_size)
                current_q = tf.reduce_sum(tf.multiply(q_values, action_mask), axis=1)
                loss = tf.reduce_mean(tf.square(target - current_q))

            # 计算梯度并更新模型
            grads = tape.gradient(loss, model.trainable_variables)
            optimizer.apply_gradients(zip(grads, model.trainable_variables))

            if done:
                break

            state = next_state

        # 更新epsilon
```

Рисунок 4- Код модели обучения DQN

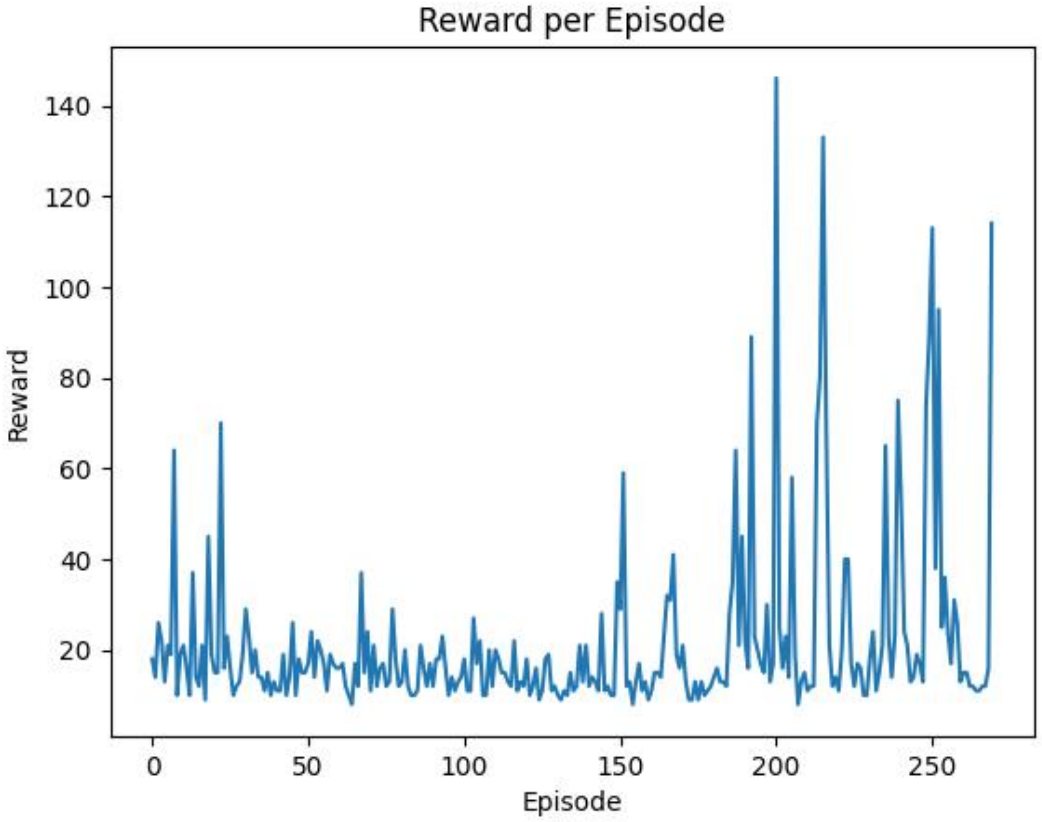

Рисунок 5-кривая оценки