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

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

«RL_DQN»

Выполнил:
студент группы ИУ5и-24М
Аунг Пьио Нанда

Москва — 2024 г.

1. Цель лабораторной работы

Ознакомление с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.

2. 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).

В случае реализации среды на основе сверточной архитектуры нейронной сети +1 балл за экзамен.

3. Ход выполнения работы

Импорт библиотек

```
import gym
import numpy as np
import random
import torch
import torch.nn as nn
import torch.optim as optim
from collections import deque
```

Определение нейронной сети Q-network

```
[8] class QNetwork(nn.Module):
        def __init__(self, state_size, action_size):
            super(QNetwork, self).__init__()
            self.fc1 = nn.Linear(state_size, 64)
            self.fc2 = nn.Linear(64, 64)
            self.fc3 = nn.Linear(64, action_size)

        def forward(self, x):
            x = torch.relu(self.fc1(x))
            x = torch.relu(self.fc2(x))
            x = self.fc3(x)
            return x
```

Реализация буфера воспроизведения

```python
class ReplayBuffer:
    def __init__(self, buffer_size, batch_size):
        self.memory = deque(maxlen=buffer_size)
        self.batch_size = batch_size

    def add(self, experience):
        self.memory.append(experience)

    def sample(self):
        experiences = random.sample(self.memory, k=self.batch_size)
        states, actions, rewards, next_states, dones = zip(*experiences)
        states = torch.tensor(states, dtype=torch.float32)
        actions = torch.tensor(actions, dtype=torch.int64)
        rewards = torch.tensor(rewards, dtype=torch.float32)
        next_states = torch.tensor(next_states, dtype=torch.float32)
        dones = torch.tensor(dones, dtype=torch.float32)
        return (states, actions, rewards, next_states, dones)

    def __len__(self):
        return len(self.memory)
```

Функция выбора действия с ε-жадной стратегией

```python
def epsilon_greedy_action(state, q_network, epsilon, action_size):
    if random.random() > epsilon:
        with torch.no_grad():
            state = torch.tensor(state, dtype=torch.float32).unsqueeze(0)
            q_values = q_network(state)
            action = q_values.max(1)[1].item()
    else:
        action = random.choice(np.arange(action_size))
    return action
```

Реализация DQN алгоритма

```python
def dqn(env, n_episodes=1000, max_t=200, gamma=0.99, epsilon_start=1.0, epsilon_end=0.01, epsilon_decay=0.995, buffer_size=10000, batch_size=64, learning_rate=0.001, update_every=4):
    state_size = env.observation_space.shape[0]
    action_size = env.action_space.n

    q_network = QNetwork(state_size, action_size)
    target_network = QNetwork(state_size, action_size)
    target_network.load_state_dict(q_network.state_dict())
    target_network.eval()

    optimizer = optim.Adam(q_network.parameters(), lr=learning_rate)
    memory = ReplayBuffer(buffer_size, batch_size)

    epsilon = epsilon_start
    timestep = 0

    for episode in range(1, n_episodes+1):
        result = env.reset()
        if isinstance(result, tuple):
            state = result[0]
        else:
            state = result
        total_reward = 0

        for t in range(max_t):
            action = epsilon_greedy_action(state, q_network, epsilon, action_size)
            next_state, reward, done, _, _ = env.step(action)
            memory.add((state, action, reward, next_state, done))
            state = next_state
            total_reward += reward
```

```
            timestep += 1
            if timestep % update_every == 0 and len(memory) > batch_size:
                experiences = memory.sample()
                states, actions, rewards, next_states, dones = experiences

                q_targets_next = target_network(next_states).detach().max(1)[0].unsqueeze(1)
                q_targets = rewards.unsqueeze(1) + (gamma * q_targets_next * (1 - dones.unsqueeze(1)))

                q_expected = q_network(states).gather(1, actions.unsqueeze(1))
                loss = nn.MSELoss()(q_expected, q_targets)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            if done:
                break

        epsilon = max(epsilon_end, epsilon_decay*epsilon)
        target_network.load_state_dict(q_network.state_dict())

        print(f"Episode {episode}/{n_episodes}, Total Reward: {total_reward}, Epsilon: {epsilon}")

    return q_network
```

Тренировка модели и оценка результатов

```
[14] env = gym.make('CartPole-v1', new_step_api=True)
     trained_q_network = dqn(env)
```

```
Episode 944/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 945/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 946/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 947/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 948/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 949/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 950/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 951/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 952/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 953/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 954/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 955/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 956/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 957/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 958/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 959/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 960/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 961/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 962/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 963/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 964/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 965/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 966/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 967/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 968/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 969/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 970/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 971/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 972/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 973/1000, Total Reward: 187.0, Epsilon: 0.01
Episode 974/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 975/1000, Total Reward: 147.0, Epsilon: 0.01
Episode 976/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 977/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 978/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 979/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 980/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 981/1000, Total Reward: 200.0, Epsilon: 0.01
Episode 982/1000, Total Reward: 200.0, Epsilon: 0.01
```

Нейронная сеть:

Q-network состоит из двух скрытых слоев с функцией активации ReLU, прогнозирующих значения Q для каждого действия.

Буфер воспроизведения:

Используется для хранения и выборки прошлых состояний, действий, наград, следующих состояний и флагов завершения эпизодов.

ε-жадная стратегия:

Используется для баланса между исследованием новых действий и использованием уже известных хороших действий.

Алгоритм DQN:

На каждой итерации действия выбираются с использованием ε-жадной стратегии, агент взаимодействует со средой и сохраняет опыт в буфере воспроизведения. Периодически из буфера выбираются случайные мини-батчи для обновления нейронной сети.

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_PG.