Московский государственный технический университет им. Н.Э. Баумана
Факультет «Головной учебно-исследовательский и методический центр
профессиональной реабилитации лиц с ограниченными возможностями здоровья (инвалидов)»
Кафедра «Системы обработки информации и управления»


# Лабораторная работа 6

## по дисциплине «Методы машинного обучения в АСОИУ»

## " Обучение на основе глубоких Q-сетей"

СТУДЕНТ:
студент группы ИУ5Ц-21М
Москалик А.А.

ПРЕПОДАВАТЕЛЬ:
Гапанюк Ю.Е.

Москва, 2024

**Цель лабораторной работы:** ознакомление с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.

**Задание:**

- На основе рассмотренных на лекции примеров реализуйте алгоритм DQN.
- В качестве среды можно использовать классические среды.
- В качестве среды можно использовать игры Atari.

## Ход работы

### 1. Импорт библиотек и настройка среды

```
import gym
import random
import numpy as np
from collections import deque
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import os

# Настройка среды CartPole
env = gym.make('CartPole-v1')

# Параметры для DQN
state_size = env.observation_space.shape[0]
action_size = env.action_space.n
batch_size = 32
n_episodes = 10
output_dir = 'model_output/cartpole/'

if not os.path.exists(output_dir):
    os.makedirs(output_register_dir)
```

В данном блоке кода задаются основные параметры и настройки, необходимые для обучения агента с использованием глубоких Q-сетей (DQN) на примере игры CartPole. Вот описание ключевых параметров и их значений:

`env = gym.make('CartPole-v1')`: Эта строка создаёт экземпляр игровой среды CartPole версии 1 из библиотеки OpenAI Gym. В этой игре цель состоит в том, чтобы удерживать шест, который стоит на движущейся тележке, в вертикальном положении как можно дольше.

`state_size = env.observation_space.shape[0]`: `state_size` определяет размерность вектора состояний среды. Для CartPole это обычно 4, что соответствует таким параметрам среды, как позиция тележки, скорость тележки, угол наклона шеста и скорость вращения шеста.

`action_size = env.action_space.n`: `action_size` определяет количество возможных действий, которые агент может выбрать в данной среде. Для CartPole это 2: двигать тележку влево или вправо.

`batch_size = 32`: Этот параметр указывает размер мини-пакета (batch), который используется для обучения нейронной сети. Значение 32 является стандартным выбором, обеспечивая хороший баланс между временем обучения и качеством модели.

n_episodes = 10`: Указывает количество эпизодов, для которых будет обучаться агент. Эпизод заканчивается, когда шест упадёт или будет достигнут предел времени эпизода. В этом случае установлено значение 10, что является довольно малым количеством для демонстрации или начального тестирования.

output_dir = 'model_output/cartpole/'`: Путь, где будут сохраняться файлы весов модели после обучения. Это позволяет в дальнейшем использовать обученную модель без необходимости повторного обучения.

`os.makedirs(output_dir)`: Этот вызов создаёт каталог `output_dir`, если он не существует. Это гарантирует, что программа не столкнётся с ошибкой при попытке сохранить данные в несуществующую директорию.

## 2. Определение класса DQNAgent

Класс DQNAgent представляет собой реализацию агента, который использует глубокие Q-сети (Deep Q-Networks, DQN) для обучения и принятия решений в задаче обучения с подкреплением. В данном случае агент обучается управлять тележкой таким образом, чтобы шест, установленный на ней, оставался в вертикальном положении как можно дольше.

```python
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95  # коэффициент дисконтирования
        self.epsilon = 1.0  # вероятность исследования
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.995
        self.learning_rate = 0.001
        self.model = self._build_model()

    def _build_model(self):
        model = Sequential()
        model.add(Dense(24, input_dim=self.state_size, activation='relu'))
        model.add(Dense(24, activation='relu'))
        model.add(Dense(self.action_size, activation='linear'))
        model.compile(loss='mse', optimizer=Adam(learning_rate=self.learning_rate))
        return model

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        act_values = self.model.predict(state)
        return np.argmax(act_values[0])

    def replay(self, batch_size):
        minibatch = random.sample(self.memory, batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = reward
            if not done:
                target = (reward + self.gamma * np.amax(self.model.predict(next_state)[0]))
            target_f = self.model.predict(state)
            target_f[0][action] = target
            self.model.fit(state, target_f, epochs=1, verbose=0)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

    def save(self, name):
        self.model.save_weights(name)
```

### 3. Создание и тренировка агента

Этот этап лабораторной работы включает инициализацию и обучение агента с использованием глубокой Q-сети (DQN) на задаче CartPole. Основная цель состоит в том, чтобы агент научился поддерживать шест в вертикальном положении на движущейся тележке максимально долго.

```python
agent = DQNAgent(state_size, action_size)

for e in range(n_episodes):
    state = env.reset()
    if isinstance(state, tuple):  # Проверяем, является ли состояние кортежем
        state = state[0]  # Используем только массив состояния
    state = np.reshape(state, [1, state_size])
    for time in range(500):
        action = agent.act(state)
        output = env.step(action)
        next_state = output[0]
        reward = output[1]
        done = output[2]
        if isinstance(next_state, tuple):
            next_state = next_state[0]
        next_state = np.reshape(next_state, [1, state_size])
        agent.remember(state, action, reward, next_state, done)
        state = next_state
        if done:
            print(f"эпизод: {e}/{n_episodes}, счёт: {time}, e: {agent.epsilon:.2f}")
            break
        if len(agent.memory) > batch_size:
            agent.replay(batch_size)
    if e % 10 == 0:
        agent.save(output_dir + f"weights_{e:04d}.weights.h5")
```

```
эпизод: 1/10, счёт: 16, e: 0.98
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

```
эпизод: 0/10, счёт: 19, e: 1.00
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

Эти строки вывода предоставляют информацию о процессе обучения вашего агента DQN в симуляции среды CartPole.

Эпизоды и счёт:

эпизод: 0/10, счёт: 11, e: 1.00 - Это сообщение говорит о том, что первый эпизод закончился, и за это время агент смог удержать шест в вертикальном положении на тележке 11 временных шагов. Значение e: 1.00 относится к параметру ε в стратегии ε-greedy, который используется для баланса между исследованием новых действий и использованием уже известных знаний. Значение 1.00 означает, что агент на этом этапе полностью фокусируется на исследовании.

Логи обучения модели:

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step и последующие - Эти строки показывают, как происходит процесс обучения нейронной сети на данных, собранных за эпизод. 1/1 означает, что производится одна итерация обучения (одна эпоха) за каждый шаг обучения. Значения типа 0s 63ms/step показывают временные затраты на каждый шаг обучения.

**Вывод:**

В ходе выполнения данной лабораторной работы была реализована модель глубокой Q-сети (DQN) для решения задачи балансировки шеста на движущейся тележке в симулированной среде CartPole из библиотеки OpenAI Gym. Эта задача демонстрирует основные принципы обучения с подкреплением, где агент должен научиться принимать решения на основе окружающего состояния для максимизации получаемого вознаграждения.

Реализация и анализ глубокой Q-сети показали значительные возможности методов обучения с подкреплением в контролируемых условиях. Понимание и умение применять эти методы открывает широкие перспективы для исследований и разработок в области искусственного интеллекта и машинного обучения.