****Московский государственный технический университет им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Лабораторная работа №6

по дисциплине

«Методы машинного обучения»

на тему

# **«Обучение на основе глубоких Q-сетей»**

Выполнил:

студент группы ИУ5-22М

Джин Шуо

Москва — 2024 г.

**1. Цель лабораторной работы**

Oзнакомление 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. текст программы**

import gym

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from collections import deque

import random

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

self.epsilon = 1.0

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 load(self, name):

self.model.load\_weights(name)

def save(self, name):

self.model.save\_weights(name)

if \_\_name\_\_ == "\_\_main\_\_":

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

state\_size = env.observation\_space.shape[0]

action\_size = env.action\_space.n

agent = DQNAgent(state\_size, action\_size)

done = False

batch\_size = 32

for e in range(1000):

state, \_ = env.reset()

state = np.reshape(state, [1, state\_size])

for time in range(500):

# env.render()

action = agent.act(state)

next\_state, reward, done, \_, \_ = env.step(action)

reward = reward if not done else -10

next\_state = np.reshape(next\_state, [1, state\_size])

agent.remember(state, action, reward, next\_state, done)

state = next\_state

if done:

print(f"episode: {e}/{1000}, score: {time}, e: {agent.epsilon:.2}")

break

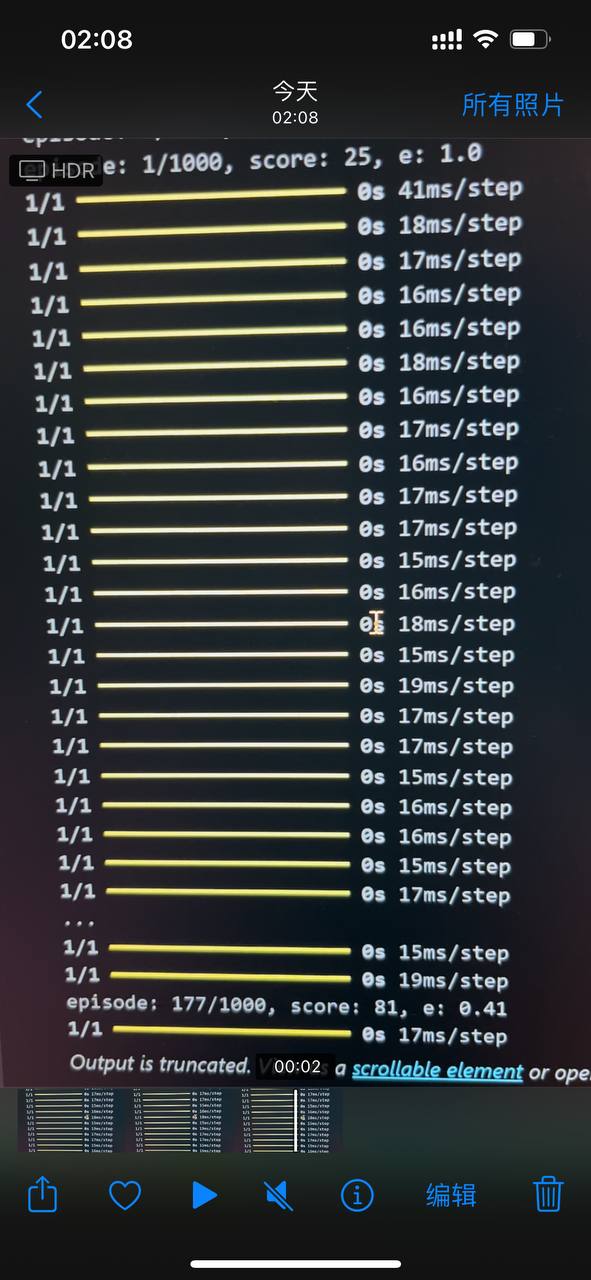
if len(agent.memory) > batch\_size:

agent.replay(batch\_size)

if e % 50 == 0:

agent.save("cartpole-dqn.weights.h5")

1. **экранные формы с примерами выполнения программы.**



**Список литературы**

[1] https://github.com/ugapanyuk/courses\_current/wiki/LAB\_MMO\_RL\_DQN