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


Лабораторная работа №5
по дисциплине
# «Методы машинного обучения»

Выполнил:
студент группы ИУ5-22М
Лю Ченхао

Москва — 2024 г.

## Цель лабораторной работы:

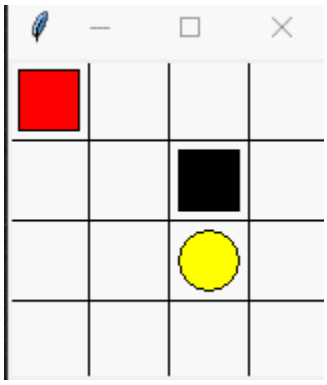
ознакомление с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.

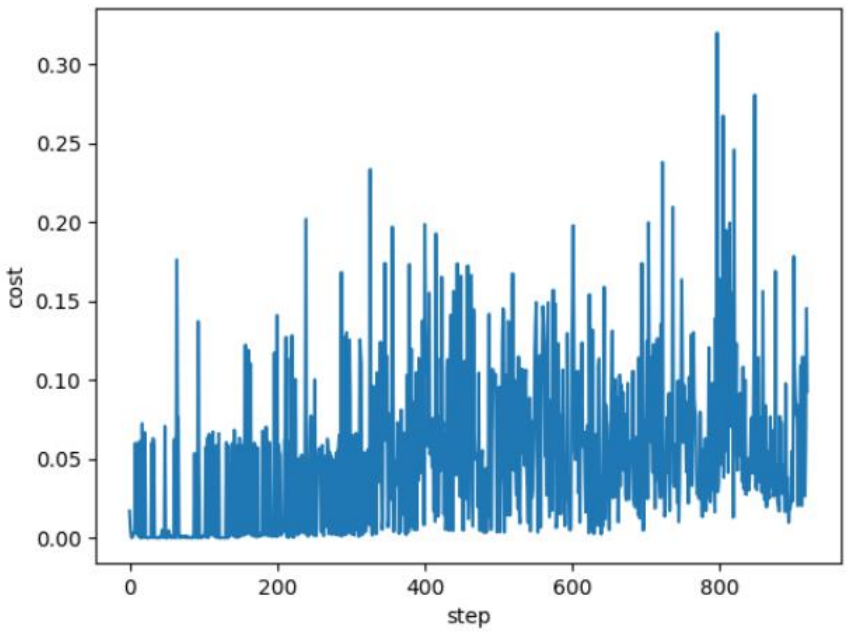
## Задание:

На основе рассмотренных на лекции примеров реализуйте алгоритм DQN.

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

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





main.py：

```python
from dqn.maze_env import Maze
from dqn.RL_brain import DQN
import time


1 usage
def run_maze():
    print("====Game Start====")
    step = 0
    max_episode = 500
    for episode in range(max_episode):
        state = env.reset()  # 重置智能体位置
        step_every_episode = 0
        epsilon = episode / max_episode  # 动态变化随机值
        while True:
            if episode < 10:
                time.sleep(0.1)
            if episode > 480:
                time.sleep(0.5)
            env.render()  # 显示新位置
            action = model.choose_action(state, epsilon)  # 根据状态选择行为
            # 环境根据行为给出下一个状态，奖励，是否结束。
            next_state, reward, terminal = env.step(action)
            model.store_transition(state, action, reward, next_state)  # 模型存储经历
            # 控制学习起始时间(先积累记忆再学习)和控制学习的频率(积累多少步经验学习一次)
            if step > 200 and step % 5 == 0:
                model.learn()
            # 进入下一步
            state = next_state
            if terminal:
                print("episode=", episode, end=",")
                print("step=", step_every_episode)
                break
            step += 1
            step_every_episode += 1
    # 游戏环境结束
    print("====Game Over====")
    env.destroy()


if __name__ == "__main__":
    env = Maze()  # 环境
    model = DQN(
        n_states=env.n_states,
        n_actions=env.n_actions
    )  # 算法模型
    run_maze()
    env.mainloop()
    model.plot_cost()  # 误差曲线
```

maze_env.py:

```python
import tkinter as tk
import sys
import numpy as np

UNIT = 40  # pixels
MAZE_H = 4  # grid height
MAZE_W = 4  # grid width


3 usages
class Maze(tk.Tk, object):
    def __init__(self):
        print("<env init>")
        super(Maze, self).__init__()
        # 动作空间(定义智能体可选的行为),action=0-3
        self.action_space = ['u', 'd', 'l', 'r']
        # 使用变量
        self.n_actions = len(self.action_space)
        self.n_states = 2
        # 配置信息
        self.title('maze')
        self.geometry("160x160")
        # 初始化操作
        self.__build_maze()

    def render(self):
        # time.sleep(0.1)
        self.update()
```

```python
def reset(self):
    # 智能体回到初始位置
    # time.sleep(0.1)
    self.update()
    self.canvas.delete(self.rect)
    origin = np.array([20, 20])
    self.rect = self.canvas.create_rectangle(
        origin[0] - 15, origin[1] - 15,
        origin[0] + 15, origin[1] + 15,
        fill='red')
    # return observation
    return (np.array(self.canvas.coords(self.rect)[:2]) - np.array(self.canvas.coords(self.oval)[:2])) / (MAZE_H * UNIT)

def step(self, action):
    # 智能体向前移动一步：返回next_state,reward,terminal
    s = self.canvas.coords(self.rect)
    base_action = np.array([0, 0])
    if action == 0:  # up
        if s[1] > UNIT:
            base_action[1] -= UNIT
    elif action == 1:  # down
        if s[1] < (MAZE_H - 1) * UNIT:
            base_action[1] += UNIT
    elif action == 2:  # right
        if s[0] < (MAZE_W - 1) * UNIT:
            base_action[0] += UNIT
    elif action == 3:  # left
        if s[0] > UNIT:
```

```
    elif action == 2:  # right
        if s[0] < (MAZE_W - 1) * UNIT:
            base_action[0] += UNIT
    elif action == 3:  # left
        if s[0] > UNIT:
            base_action[0] -= UNIT

    self.canvas.move( *args: self.rect, base_action[0], base_action[1])  # move agent

    next_coords = self.canvas.coords(self.rect)  # next state

    # reward function
    if next_coords == self.canvas.coords(self.oval):
        reward = 1
        print("victory")
        done = True
    elif next_coords in [self.canvas.coords(self.hell1)]:
        reward = -1
        print("defeat")
        done = True
    else:
        reward = 0
        done = False
    s_ = (np.array(next_coords[:2]) - np.array(self.canvas.coords(self.oval)[:2])) / (MAZE_H * UNIT)
    return s_, reward, done
```

```
def __build_maze(self):
    self.canvas = tk.Canvas(self, bg='white',
                            height=MAZE_H * UNIT,
                            width=MAZE_W * UNIT)

    # create grids
    for c in range(0, MAZE_W * UNIT, UNIT):
        x0, y0, x1, y1 = c, 0, c, MAZE_H * UNIT
        self.canvas.create_line(x0, y0, x1, y1)
    for r in range(0, MAZE_H * UNIT, UNIT):
        x0, y0, x1, y1 = 0, r, MAZE_W * UNIT, r
        self.canvas.create_line(x0, y0, x1, y1)
    origin = np.array([20, 20])
    hell1_center = origin + np.array([UNIT * 2, UNIT])
    self.hell1 = self.canvas.create_rectangle(
        hell1_center[0] - 15, hell1_center[1] - 15,
        hell1_center[0] + 15, hell1_center[1] + 15,
        fill='black')
    oval_center = origin + UNIT * 2
    self.oval = self.canvas.create_oval(
        oval_center[0] - 15, oval_center[1] - 15,
        oval_center[0] + 15, oval_center[1] + 15,
        fill='yellow')
    self.rect = self.canvas.create_rectangle(
        origin[0] - 15, origin[1] - 15,
        origin[0] + 15, origin[1] + 15,
        fill='red')
    self.canvas.pack()
```

RL_brain.py:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt

3 usages
class Net(nn.Module):
    def __init__(self, n_states, n_actions):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_states, out_features: 10)
        self.fc2 = nn.Linear( in_features: 10, n_actions)
        self.fc1.weight.data.normal_( mean: 0, std: 0.1)
        self.fc2.weight.data.normal_( mean: 0, std: 0.1)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        out = self.fc2(x)
        return out
```

```python
class DQN:
    def __init__(self, n_states, n_actions):
        print("<DQN init>")
        # DQN有两个net:target net和eval net,具有选动作，存经历，学习三个基本功能
        self.eval_net, self.target_net = Net(n_states, n_actions), Net(n_states, n_actions)
        self.loss = nn.MSELoss()
        self.optimizer = torch.optim.Adam(self.eval_net.parameters(), lr=0.01)
        self.n_actions = n_actions
        self.n_states = n_states
        # 使用变量
        self.learn_step_counter = 0  # target网络学习计数
        self.memory_counter = 0  # 记忆计数
        self.memory = np.zeros((2000, 2 * 2 + 2))  # 2*2(state和next_state,每个x,y坐标确定)+2(action和reward),存储2000个记忆体
        self.cost = []  # 记录损失值

    1 usage
    def choose_action(self, x, epsilon):
        # print("<choose_action>")
        x = torch.unsqueeze(torch.FloatTensor(x), dim: 0)  # (1,2)
        if np.random.uniform() < epsilon:
            action_value = self.eval_net.forward(x)
            action = torch.max(action_value, 1)[1].data.numpy()[0]
        else:
            action = np.random.randint( low: 0, self.n_actions)
        # print("action=", action)
        return action
```

```python
def store_transition(self, state, action, reward, next_state):
    # print("<store_transition>")
    transition = np.hstack((state, [action, reward], next_state))
    index = self.memory_counter % 2000  # 满了就覆盖旧的
    self.memory[index, :] = transition
    self.memory_counter += 1

def learn(self):
    # print("<learn>")
    # target net 更新频率,用于预测，不会及时更新参数
    if self.learn_step_counter % 100 == 0:
        self.target_net.load_state_dict((self.eval_net.state_dict()))
    self.learn_step_counter += 1

    # 使用记忆库中批量数据
    sample_index = np.random.choice( a: 2000,  size: 16)  # 2000个中随机抽取16个作为batch_size
    memory = self.memory[sample_index, :]  # 抽取的记忆单元，并逐个提取
    state = torch.FloatTensor(memory[:, :2])
    action = torch.LongTensor(memory[:, 2:3])
    reward = torch.LongTensor(memory[:, 3:4])
    next_state = torch.FloatTensor(memory[:, 4:6])

    # 计算loss,q_eval:所采取动作的预测value,q_target:所采取动作的实际value
    q_eval = self.eval_net(state).gather(1, action)# eval_net->(16,4)->按照action索引提取出q_value
    q_next = self.target_net(next_state).detach()
    # torch.max->[values=[],indices=[]] max(1)[0]->values=[]
```

```python
    # 计算loss,q_eval:所采取动作的预测value,q_target:所采取动作的实际value
    q_eval = self.eval_net(state).gather(1, action)# eval_net->(16,4)->按照action索引提取出q_value
    q_next = self.target_net(next_state).detach()
    # torch.max->[values=[],indices=[]] max(1)[0]->values=[]
    q_target = reward + 0.9 * q_next.max(1)[0].unsqueeze(1)# label
    loss = self.loss(q_eval, q_target)
    self.cost.append(loss.item())  # 注意：这里是添加损失值
    # 反向传播更新
    self.optimizer.zero_grad()  # 梯度重置
    loss.backward()  # 反向求导
    self.optimizer.step()  # 更新模型参数

1 usage
def plot_cost(self):
    plt.plot( *args: np.arange(len(self.cost)), self.cost)
    plt.xlabel("step")
    plt.ylabel("cost")
    plt.show()
```

## Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Подготовка обучающей и тестовой выборки,кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»[Электронный ресурс] // GitHub.. — Режим доступа: https://github.com/ugapanyuk/ml_course/wiki/LAB_KNN

[2] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource] //Read the Docs. — Access mode: https://ipython.readthedocs.io/en/stable/

[3] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData.Access mode: https://seaborn.pydata.org/

[4] pandas 0.24.1 documentation [Electronic resource] // PyData. — Access mode:http://pandas.pydata.org/pandas-docs/stable/