实验 4：强化学习

2024 年 6 月 3 日

1 实验简介

本实验主要介绍强化学习的基本原理与实现方法。

通过本实验，你将学习利用 OpenAI-Gym、PyTorch 等开源工具训练一个深度强化学习模型，掌握以下技能：

- 利用 OpenAI-Gym 搭建强化学习算法的训练、测试环境；
- 了解 DQL（Deep Q-Network）算法的基本原理；
- 通过 -贪心策略训练模型，判定训练的终止条件。

2 实验准备

为了顺利完成本实验的操作内容，你需要首先安装 `OpenAI-Gym`、`PyTorch` 等工具包及其依赖。为了加快安装速度，请确保 `pip` 包管理器已切换到国内源。本实验的依赖包主要包括：

- `OpenAI-Gym`：用于开发和评估强化学习算法的工具包，提供了一系列强化学习模拟环境；
- `PyTorch`：开源的 Python 机器学习库；
- `numpy`：Python 数值计算库。

[2]:
```
import pkgutil

# 检查 PyTorch 库是否已安装
if pkgutil.find_loader('torch'):
    print('PyTorch is available.')
else:
```

```
    !pip install torch -i https://pypi.tuna.tsinghua.edu.cn/simple/

# 检查 OpenAI-Gym 库是否已安装
if pkgutil.find_loader('gym'):
    print('OpenAI-Gym is available.')
else:
    !pip install gym -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

```
PyTorch is available.
OpenAI-Gym is available.
```

[3]:
```
# 忽略 Warning 输出
import warnings
warnings.filterwarnings("ignore")
```

3 DQN 算法原理

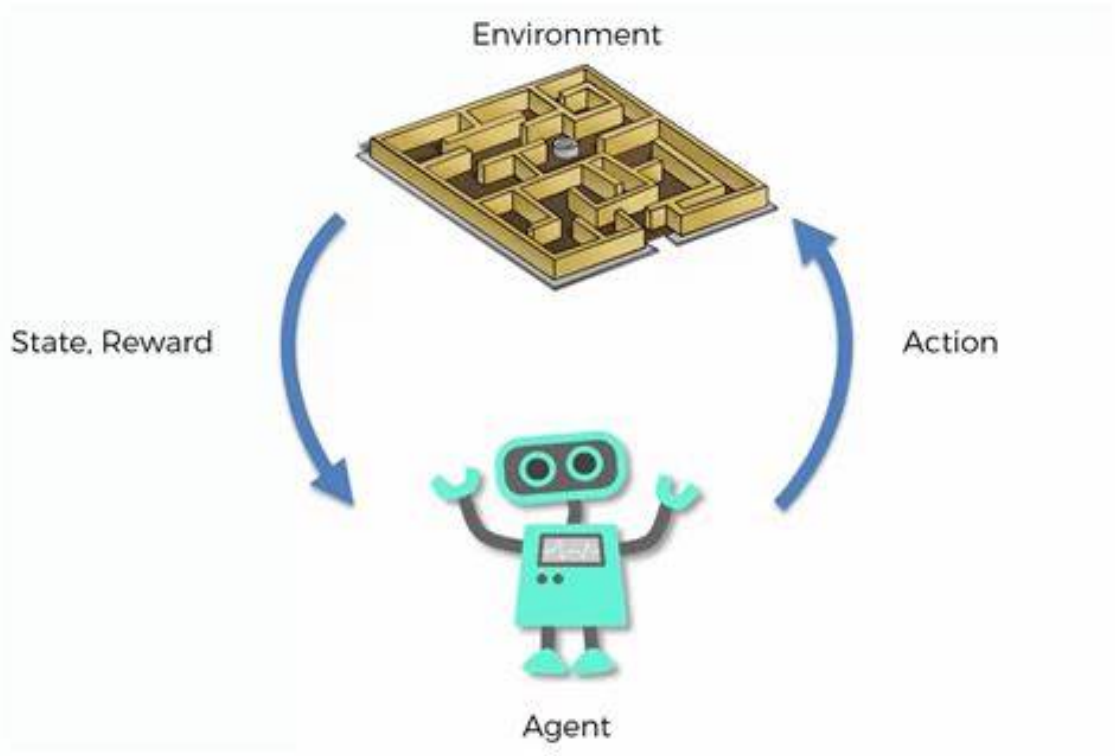
3.1 强化学习

强化学习（Reinforcement Learning）是人工智能领域行为主义学派最具代表性的方法，主要研究智能体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为。强化学习的主要特点是智能体和环境之间不断进行交互，智能体为了获得更多的累计奖励而不断搜索和试错。

强化学习系统主要由以下部分组成，分别是智能体（Agent）、环境（Environment）、状态（State）、动作（Action）和奖励（Reward），而行动是通过策略（Policy）决定的。对上述各概念的简短解释如下：

- 智能体：作为学习器与决策者，参与与环境的交互；
- 环境：智能体之外的一切与之交互的事物；
- 动作：智能体的行为表征；
- 状态：智能体从环境获取的信息；
- 奖励：环境对于动作的反馈；
- 策略：智能体根据状态进行下一步动作的函数。

强化学习的训练过程涉及两个重要概念：探索（Exploration）和开发（Exploitation）。其中，探索是指选择之前未执行过的动作，从而探索更多的可能性；开发（或称“利用”、“深化”）是指选择已执行过的动作，从而对现有模型进行完善。



3.2 Deep Q-Network

Deep Q-Network（DQN）算法是一种基于深度神经网络的强化学习算法，旨在解决 Q-learning 算法在高维状态空间中面临的问题。DQN 算法的流程如下：

- 定义输入状态 s、动作 a、输出 Q 值的深度神经网络 $Q(s, a, \theta)$，其中 θ 为神经网络参数；
- 初始化经验回放（Experience replay）缓冲区 D，用于存储过去的经验；
- 对于每个时间步 t，执行以下步骤：
  - 以 ϵ-贪心策略从网络中获取动作 a_t；
  - 在环境中执行动作 a_t，得到下一个状态 s_{t+1} 和奖励 r_t；
  - 将 (s_t, a_t, r_t, s_{t+1}) 存储到经验回放缓冲区 D 中；
  - 从 D 中随机抽取一个小批量经验 (s_j, a_j, r_j, s_{j+1})，并计算目标 Q 值；
  - 通过反向传播算法更新神经网络参数 θ
- 每 C 步将当前网络参数 θ 复制给目标网络参数 θ^-。

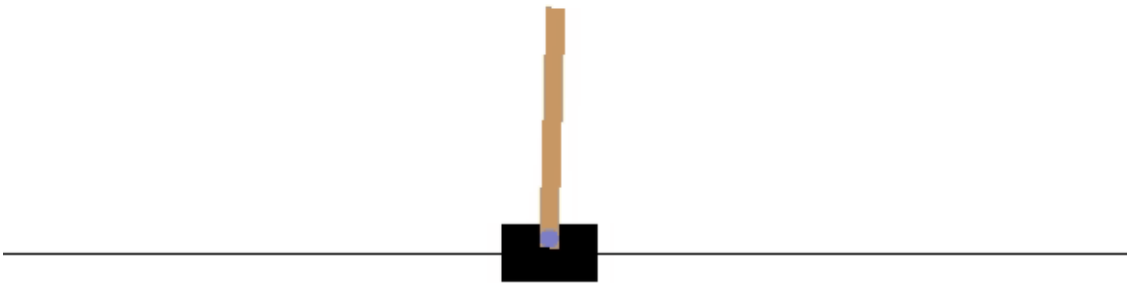
如果你希望进一步了解算法细节，参见[Link]。

4 强化学习测试环境

4.1 Cartpole 平衡杆环境

在 Cartpole 环境中，存在一根倒立杆，它被连接到一个可在水平轨道上移动的小车上。我们的目标是通过移动小车来平衡倒立的杆，防止它倒下。这个问题是研究和测试不同强化学习算法和控制策略的一个经典基准问题。

该问题中系统的状态可以通过四个连续变量来描述，包括小车的位置、小车的速度、杆的角度和杆的角速度。控制信号是对小车施加向左或向右的力，通过改变车的位置来影响杆的平衡状态。



4.2 OpenAI-Gym 中的环境说明

在 OpenAI-Gym 中已经预置了 Cartpole 环境，其说明如下：

```
class CartPoleEnv(gym.Env):
    """
    Description:
        A pole is attached by an un-actuated joint to a cart, which moves along
        a frictionless track. The pendulum starts upright, and the goal is to
        prevent it from falling over by increasing and reducing the cart's
```

```
    velocity.
Source:
    This environment corresponds to the version of the cart-pole problem
    described by Barto, Sutton, and Anderson
Observation:
    Type: Box(4)
    Num     Observation               Min                     Max
    0       Cart Position             -4.8                    4.8
    1       Cart Velocity             -Inf                    Inf
    2       Pole Angle                -0.418 rad (-24 deg)    0.418 rad (24 deg)
    3       Pole Angular Velocity     -Inf                    Inf
Actions:
    Type: Discrete(2)
    Num   Action
    0     Push cart to the left
    1     Push cart to the right
    Note: The amount the velocity that is reduced or increased is not
    fixed; it depends on the angle the pole is pointing. This is because
    the center of gravity of the pole increases the amount of energy needed
    to move the cart underneath it
Reward:
    Reward is 1 for every step taken, including the termination step
Starting State:
    All observations are assigned a uniform random value in [-0.05..0.05]
Episode Termination:
    Pole Angle is more than 12 degrees.
    Cart Position is more than 2.4 (center of the cart reaches the edge of
    the display).
    Episode length is greater than 200.
    Solved Requirements:
    Considered solved when the average return is greater than or equal to
    195.0 over 100 consecutive trials.
"""
```

5 环境与模型搭建

首先，导入实验所需的包，并定义 CartPole-v1 环境。

[12]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import gym
import matplotlib.pyplot as plt

# 定义 CartPole-v1 环境
env = gym.make("CartPole-v1", render_mode="human")
N_ACTIONS = 2
N_STATES = env.observation_space.shape[0]

# 定义超参数
BATCH_SIZE = 32
LR = 0.01
EPSILON = 0.5
GAMMA = 0.95
TARGET_REPLACE_ITER = 100  # C 值，即目标网络更新间隔
MEMORY_CAPACITY = 2000

# 有可用的 GPU 时将使用 GPU 计算，否则使用 CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

定义一个简单的神经网络，作为 Q 值的评估函数。

[7]:
```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.fc1 = nn.Linear(N_STATES, 128)
        self.fc1.weight.data.normal_(0, 0.1)  # 网络权重参数初始化
        self.fc2 = nn.Linear(128, 128)
        self.out = nn.Linear(128, N_ACTIONS)
        self.out.weight.data.normal_(0, 0.1)
```

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

接下来，定义 DQN 智能体。

- `__init__`：定义了主网络和目标网络，同时定义了经验回放缓存、优化器、损失函数等；
- `choose_action`：动作选择函数，其输入为智能体当前状态，通过 ϵ-贪心策略实现探索与开发的平衡；
- `store_transition`：经验回放缓存函数；
- `learn`：定义了 DQN 学习算法，完成神经网络参数更新。

[8]:
```
class DQN:
    def __init__(self):
        self.main_net, self.target_net = MyNet().to(device), MyNet().to(device)␣
 ↪ # device 来自于 torch.device()
        self.target_net.load_state_dict(self.main_net.state_dict())
        self.target_net.eval()

        self.learn_step_counter = 0  # 已训练的次数计数器
        self.memory_counter = 0  # 已保存的经验样本数量计数器
        self.memory = np.zeros((MEMORY_CAPACITY, N_STATES * 2 + 2))  # 经验样本
缓存池
        self.optimizer = torch.optim.Adam(self.main_net.parameters(), lr=LR)
        self.loss_func = nn.MSELoss()

    @torch.no_grad()
    def choose_action(self, x):
        if np.random.uniform(0, 1) < EPSILON:
            action = np.random.randint(0, N_ACTIONS)
        else:
            x = torch.unsqueeze(torch.FloatTensor(x), 0)  # 在 x 的第 0 维添加一
个维度，达到升维的效果
            actions_value = self.main_net.forward(x)
```

```
            action = torch.max(actions_value, 1)[1].data.cpu().numpy()  # 在 actions_value 的第 1 维上计算最大值
            # 最大值返回的是一个元组，分别表示最大值及其下标，并以 numpy 数组的形式返回
            action = action[0]

        return action

    def store_transition(self, s, a, r, s_prime):
        transition = np.hstack((s, [a, r], s_prime))  # 将参数以水平方式堆叠成一个一维 numpy 数组
        index = self.memory_counter % MEMORY_CAPACITY
        self.memory[index, :] = transition
        self.memory_counter += 1

    def learn(self):
        if self.learn_step_counter % TARGET_REPLACE_ITER == 0:
            self.target_net.load_state_dict(self.main_net.state_dict())
        self.learn_step_counter += 1

        sample_index = np.random.choice(MEMORY_CAPACITY, BATCH_SIZE)
        batch_memory = self.memory[sample_index, :]
        batch_s = torch.FloatTensor(batch_memory[:, :N_STATES])
        batch_a = torch.LongTensor(batch_memory[:, N_STATES:N_STATES + 1].
 ↪astype(int))
        batch_r = torch.FloatTensor(batch_memory[:, N_STATES + 1:N_STATES + 2])
        batch_s_prime = torch.FloatTensor(batch_memory[:, -N_STATES:])

        # 主网络对输入的一批状态，计算每个状态下所有的行为价值，并从第 1 维中取出一批动作的价值；
        # 返回一个 1 维张量 q，包含了这批状态-动作对的行为价值
        q = self.main_net(batch_s).gather(1, batch_a)

        # 在目标网络中计算后续状态的所有可能的行为价值，detach 函数将其从计算图中分离出来，避免在反向传播过程中产生新的梯度；
        # 返回一个新的张量，但与原始张量共享底层存储
```

```
        q_target = self.target_net(batch_s_prime).detach()

        # q_target.max(1) 取出每个状态下最大的行为价值及其索引，q_target.max(1)[0]
取出所有的状态下的最大行为价值；
        # 最后调用 view 方法来将张量重塑为可计算的形状
        y = batch_r + GAMMA * q_target.max(1)[0].view(BATCH_SIZE, 1)

        loss = self.loss_func(q, y)

        self.optimizer.zero_grad()  # 将所有可学习的参数的梯度清零，否则梯度会累加
        loss.backward()  # 计算梯度
        for param in self.main_net.parameters():
            param.grad.data.clamp_(-1, 1)
        self.optimizer.step()  # 更新参数
```

6 模型训练

[1]:
```
dqn = DQN()
rounds = 220  # 实验表明，经过约 220 轮训练，模型就可以达到较好的水平

for i in range(rounds):
    state, _ = env.reset()
    episode_reward = 0

    if i % 100 == 0:
        print("Process training: {}%".format(i / rounds * 100))
    while True:
        action = dqn.choose_action(state)
        s_prime, r, done, _, _ = env.step(action)

        x, x_dot, theta, theta_dot = s_prime
        r1 = (env.x_threshold - abs(x)) / env.x_threshold - 0.8
        r2 = (env.theta_threshold_radians - abs(theta)) / env.
 ↪theta_threshold_radians - 0.5
        r = r1 + r2
```

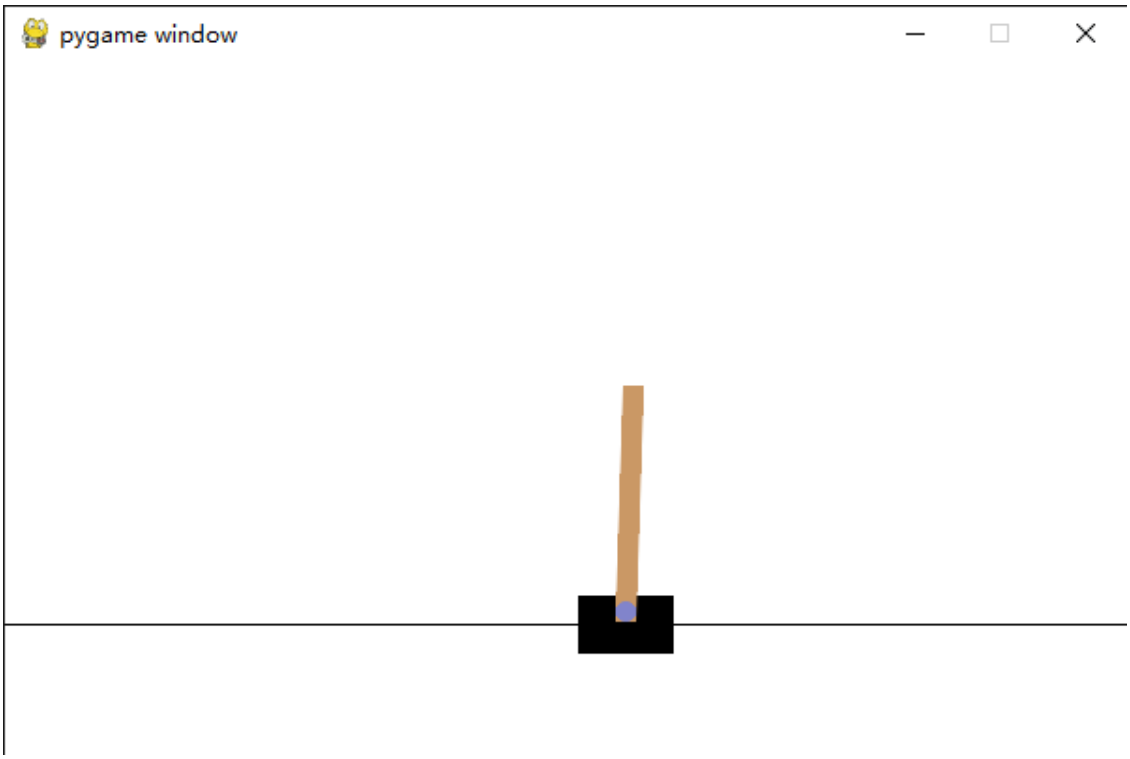
```
        dqn.store_transition(state, action, r, s_prime)

        episode_reward += r
        if dqn.memory_counter > MEMORY_CAPACITY:
            dqn.learn()
            if done:
                print('Episode: ', i, '| Episode_reward: ',␣
↪round(episode_reward, 2))

        if done:
            break
        state = s_prime

    if EPSILON > 0.05:
        EPSILON *= 0.99
```

运行上面的代码后，你将看到如下图所示的窗口：



在训练刚开始时，平衡杆很容易就会倒下，此时的 Episode 奖励值很低；一段时间后，平衡杆的表现

将不断提升，并可能在滑轨上维持很长时间不倒下，每轮的持续时间将显著增长，获得的 Episode 奖励值也将提高。一般而言，在 200 个 Episode 后，奖励值就可以达到较高的水平。