Lab 6: 基于强化学习的Cart Pole游戏

231880101 孙俊晖

1 引言

1.1 背景

Cart Pole游戏是一个经典的强化学习问题，在这个游戏中，一个垂直的杆被放置在一个可以左右移动的小车上，玩家需根据当前状态，如**小车的位置、杆子的倾斜角度、小车的速度以及杆子的角速度**等，来决定施加向左或向右的力，以防止杆子倒下。这个问题被广泛用于测试强化学习算法的性能，因为它包含了控制、平衡和规划等多个方面的挑战。

1.2 目标

利用强化学习算法训练一个智能体，使其能够成功地在Cart Pole游戏中保持杆的平衡，并尽可能长时间地防止杆倒下。

2 方法

2.1 技术栈

开发环境：Visual Studio Code 1.93.0

编程语言：Python 3.12.3

2.2 选择的强化学习算法

在本次实验中，我选择使用**REINFORCE算法**作为强化学习算法。REINFORCE是一种基于策略梯度的算法，它直接优化策略参数以最大化期望回报。与DQN等基于价值的算法不同，REINFORCE算法通过计算每个采取动作的对数概率和相应回报的乘积来更新策略参数。

2.3 导入必要的库

为了构建和训练策略网络，我首先导入了实验中所需的关键库。gym库用于创建和测试强化学习环境，torch库及其子模块则用于构建和训练神经网络，time库用于记录测试过程中的时间消耗，而pygame则用于测试部分的可视化。

```
import gym
import torch
import time
import pygame
import torch.nn as nn
from torch import optim
from torch.distributions import Categorical
```

2.4 定义策略网络

为了近似策略函数，我设计了一个神经网络模型。该模型接收当前游戏状态作为输入，并输出每个可能动作的概率。在本实验中，我采用了一个包含两层全连接隐藏层的神经网络架构。第一层隐藏层包含128个神经元，用于从输入的4个状态特征（小车位置、杆子倾斜角度、小车速度和杆子角速度）中提取高层次的特征表示。由于Cart Pole游戏仅包含两个可能的动作，即向左移动（**action=0**）和向右移动（**action=1**），因此输出层包含两个神经元，分别对应这两个动作的概率。为了减轻过拟合的风险，我在第一层隐藏层之后添加了一个Dropout层，设置丢弃率为60%。这意味着在每次前向传播过程中，随机选择60%的神经元不参与计算，从而增强模型的泛化能力。最后，我使用softmax激活函数对输出层的两个神经元进行激活。softmax函数能够将任意实数值映射到(0,1)区间内，并且保证所有输出值的和为1，从而形成一个有效的概率分布。这样，就可以根据输出的概率来选择动作了。

以下是神经网络模型的代码实现：

```
#定义神经网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #第一层全连接层，输入4个特征，输出128个特征
        self.fc1 = nn.Linear(in_features=4, out_features=128)
        #第二层全连接层，输入128个特征，输出2个动作的概率
        self.fc2 = nn.Linear(128, 2)
        #Dropout层，用于减少过拟合，丢弃60%的神经元
        self.drop = nn.Dropout(p=0.6)

    def forward(self, x):
        #通过第一层全连接层
        x = self.fc1(x)
        #应用Dropout
        x = self.drop(x)
        #应用ReLU激活函数
        x = nn.functional.relu(x)
        #通过第二层全连接层
        x = self.fc2(x)
        #使用softmax函数将输出转换为概率分布
        return nn.functional.softmax(x, dim=1)
```

2.5 定义损失函数

REINFORCE算法的关键在于计算每个采取动作的对数概率和相应回报(奖励)的乘积，并据此更新策略参数。在本实验中，我定义了一个损失函数accumulate_loss，该函数根据动作的对数概率列表和奖励序列计算总损失。为了使杆子尽可能长时间维持平衡状态，杆子应该尽可能在初期处于平衡状态。所以一开始杆子处于平衡状态获得的奖励应尽可能多，越往后处于平衡状态得到的奖励越少。基于这种想法，我创建了一个递减的奖励序列，杆子在第i轮处于平衡状态的奖励为

$$reward_i = n - i + 1$$

对奖励序列进行标准化处理后，计算总损失

$$loss = -\sum_{i=1}^{n} p_i * reward_i$$

以下是损失函数的代码实现：

```
#基于REINFORCE算法的损失函数
def accumulate_loss(n, log_prob):
    #创建一个递减的奖励序列
    reward = torch.arange(n, 0, -1).float()
    #对奖励序列进行标准化处理
    reward = (reward - reward.mean()) / reward.std()
    loss = 0
    #基于每一步的log概率和对应的奖励计算总损失
    for p, r in zip(log_prob, reward):
        loss -= p * r
    return loss
```

2.6 训练模型

首先，我创建了一个CartPole环境，并实例化刚才定义的策略网络。然后，我选择了Adam优化器作为我训练模型的优化器，设置学习率为0.01。

```
#创建一个CartPole环境
env = gym.make("CartPole-v1")
#实例化神经网络模型
net = Net()
#优化器
optimizer = optim.Adam(net.parameters(), lr=0.01)
```

做好了准备工作之后，就可以开始训练了。我设置了最大训练次数为1000次，如果在某个训练回合中模型坚持了5000步以上，则提前结束训练。在训练过程中，模型通过构建的策略网络计算动作概率，根据概率分布选择一个动作执行，执行完动作后再获取下一步的状态。每训练完一个回合之后，我都会使用梯度下降算法更新策略参数，且每10个回合打印一次在该回合模型坚持的步数，以监测模型的训练情况。

```
#训练循环
for episod in range(1, 1001):  #训练1000个回合
   train_state, _ = env.reset()  #重置环境，获取初始状态
   train_step = 0  #记录当前回合的步数
   log_prob = []  #存储每一步的log概率
   for train_step in range(1, 10001):  #每个回合最多10000步
      train_state = torch.from_numpy(train_state).float().unsqueeze(0)
      probs = net(train_state)  #通过神经网络计算动作概率
      m = Categorical(probs)  #创建一个概率分布
      action = m.sample()  #根据概率分布采样一个动作
      train_state, _, done, _, _ = env.step(action.item())  #执行动作，获取下一步的状态
      if done:  #如果回合结束，跳出循环
        break
      log_prob.append(m.log_prob(action))  #存储当前动作的log概率
   if train_step > 5000:  # 如果某个回合超过5000步，则结束训练
       print(f"Last episode {episod} Run steps {train_step}")
       break
   #梯度下降算法
   optimizer.zero_grad()  #清空梯度
   loss = accumulate_loss(train_step, log_prob)  #计算损失
   loss.backward()  #反向传播
   optimizer.step()  #更新模型参数
```

```
    if episod % 10 == 0:  #每10个回合打印一次信息
        print(f"Episode {episod} Run step {train_step}")
print("Finish training!")
```

我的模型在训练了137个回合后结束了训练，在第137回合坚持了8921步：

```
Episode 10 Run step 51
Episode 20 Run step 29
Episode 30 Run step 40
Episode 40 Run step 48
Episode 50 Run step 84
Episode 60 Run step 107
Episode 70 Run step 87
Episode 80 Run step 154
Episode 90 Run step 109
Episode 100 Run step 114
Episode 110 Run step 113
Episode 120 Run step 307
Episode 130 Run step 112
Last episode 137 Run steps 8921
Finish training!
```

2.7 测试模型

与训练模型时不同，为了直观地感受模型的性能，需要将Cart Pole游戏可视化。这需要使用pygame库进行图形渲染，并且创建的环境必须得支持图形渲染。

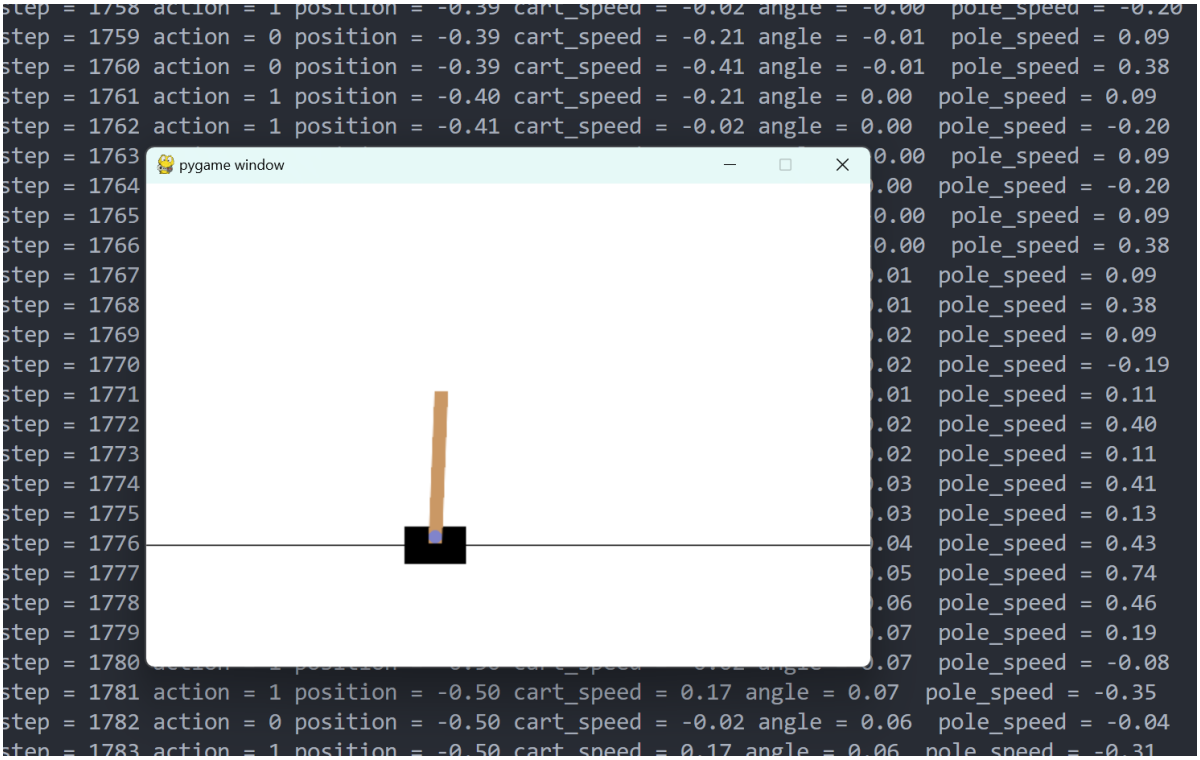
```
#初始化pygame
pygame.init()
#创建一个支持图形渲染的环境
env = gym.make('CartPole-v1', render_mode="human")
```

接下来开始测试模型。我设置了模型的最大测试步数为2000次，在测试过程中，模型通过策略网络计算动作概率并选择概率最高的动作执行，然后打印每一步的状态，包括模型已经坚持的步数、模型执行的动作、小车的位置、杆子的倾斜角度、小车的速度以及杆子的角速度。同时，利用time库记录测试过程中的时间消耗。然后，输出测试完成时杆子维持平衡的时间以及模型坚持的步数。最后，关闭环境。

```
state, _ = env.reset()  #重置环境，获取初始状态
test_step = 0  #记录测试时的步数
start = time.time()  #记录开始时间
for test_step in range(1, 2001):  #测试最多2000步
    state = torch.from_numpy(state).float().unsqueeze(0)
```

```
    probs = net(state)  #通过神经网络计算动作概率
    action = torch.argmax(probs, dim=1).item()  #选择概率最高的动作
    state, _, done, _, _ = env.step(action)  #执行动作，获取下一步的状态
    if done:  #如果游戏结束，跳出循环
      break
    print(f"step = {test_step} action = {action} position = {state[0]:.2f} cart_speed = {state[1]:.2f} angle = {state[2]:.2f}  pole_speed = {state[3]:.2f}")  # 打印当前步骤的信息
end = time.time()  #记录结束时间
print(f"You play {end - start:.2f} seconds, {test_step} steps.")
env.close()  #关闭环境
```

Cart Pole游戏可视化如图所示：



最终**模型成功坚持了2000步，使杆维持平衡长达43.16秒**，说明利用强化学习算法构建的模型能够很好地完成Cart Pole游戏。

```
step = 1994 action = 1 position = -0.42 cart_speed = -0.01 angle = 0.01  pole_speed = -0.28
step = 1995 action = 0 position = -0.42 cart_speed = -0.21 angle = 0.01  pole_speed = 0.01
step = 1996 action = 0 position = -0.42 cart_speed = -0.40 angle = 0.01  pole_speed = 0.31
step = 1997 action = 0 position = -0.43 cart_speed = -0.60 angle = 0.02  pole_speed = 0.61
step = 1998 action = 1 position = -0.44 cart_speed = -0.40 angle = 0.03  pole_speed = 0.32
step = 1999 action = 1 position = -0.45 cart_speed = -0.21 angle = 0.03  pole_speed = 0.03
step = 2000 action = 0 position = -0.45 cart_speed = -0.40 angle = 0.03  pole_speed = 0.34
You play 43.16 seconds, 2000 steps.
```

3 总结

在本次实验中，我深入探索了基于强化学习的Cart Pole游戏控制问题。通过应用REINFORCE算法，我成功训练了一个智能体，使其能够在Cart Pole游戏中维持杆的平衡，并实现了长时间的稳定运行。整个实验过程涵盖了从环境搭建、算法选择、模型设计到训练与测试的完整流程，不仅加深了我对强化学习原理的理解，也锻炼了我的编程实践能力。

4 参考文献

1.https://www.gymlibrary.dev/environments/classic_control/cart_pole/

2.https://blog.csdn.net/jermy00/article/details/132214194

3.https://download.csdn.net/blog/column/11787767/126676660