# 报告：基于Q-learning算法的倒立摆问题求解

目录

[报告：基于Q-learning算法的倒立摆问题求解 1](#_Toc21919)

[1. 题目分析 2](#_Toc13727)

[2. 解题思路 2](#_Toc9685)

[3. 使用的库和方法 3](#_Toc25160)

[4. 从模型建立到实际代码的编写流程 3](#_Toc3231)

[5. 中间结论 5](#_Toc6582)

[6. 最终结果 5](#_Toc3211)

[7. 过程分析和结论 6](#_Toc27249)

[8. 实验中的问题 6](#_Toc24450)

[9. 源代码以及运行结果 8](#_Toc32304)

[总结： 10](#_Toc27123)

## 1. 题目分析

本题是经典的强化学习问题——倒立摆问题（`CartPole-v1`）。

在 `CartPole-v1` 环境中，状态空间是**连续的**，但动作空间是**离散的**。状态空间包括以下四个变量：

- 小车的位置

- 小车的速度

- 杆子的角度

- 杆子的角速度

动作空间则有两个：

1. 向左施力

2. 向右施力

我们的目标是通过强化学习算法来找到一个策略，使得在给定的最大步数限制下，小车能够保持杆子不倒下，即保持小车和杆的系统稳定。

## 解题思路

我们使用 Q-learning算法来解决这一强化学习问题。Q-learning 是一种基于表格的强化学习算法，它通过估算每个**状态-动作对**的 Q 值来学习最优策略。Q 值代表在特定状态下执行某个动作所能获得的预期总回报。

主要流程：

1. 将连续的状态空间离散化，这样可以在 Q 表中存储不同状态下的动作值。

2. 通过 Q-learning 公式不断更新状态-动作对的 Q 值。

3. 在训练过程中使用epsilon-greedy策略，保证探索与利用的平衡。

4. 最终，通过学习到的 Q 值，在不同状态下选择最优动作，保持杆子的平衡。

具体步骤：

1. 导入库：使用 `gym` 提供的 `CartPole-v1` 环境和 `numpy` 进行数值处理。

2. 状态离散化：通过 `np.digitize` 函数将连续的状态空间离散化为有限的区间。

3. 初始化 Q 表：Q 表存储了所有状态-动作对的 Q 值，初始值设为 0。

4. Q-learning 训练：通过探索与利用更新 Q 值，使得智能体逐步学会最优策略。

5. 测试训练结果：在训练完成后，对模型进行测试，验证智能体是否学会了如何在给定条件下保持平衡。

## 3. 使用的库和方法

主要库：

- `gym`：用于创建和运行强化学习环境 `CartPole-v1`。

- `numpy`：用于数值计算，尤其是处理状态离散化和 Q 表更新。

使用的强化学习方法：

- Q-learning：

- 是一种值迭代的强化学习算法。

- 通过在不同状态下执行动作，获取即时奖励并更新 Q 值，使得 Q 值逐渐逼近最优策略。

## 4. 从模型建立到实际代码的编写流程

### （1）初始化环境

首先，我们使用 `gym` 库创建 `CartPole-v1` 环境，并设定一些环境参数，如最大步数和杆子的角度限制。

(**其实也可以不改，因为 `CartPole-v1` 环境的默认参数就是老师给出的题目的参数**)

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

env.env.theta\_threshold\_radians = 12 \* 2 \* 3.14159 / 360

env.env.x\_threshold = 2.4

env.env.max\_episode\_steps = 500

### （2）状态离散化

由于 Q-learning 是基于离散状态的算法，我们需要将连续状态离散化为有限的区间。通过 `np.digitize` 将小车的位置、速度、杆子的角度和角速度分成 10 个区间。

state\_bins = [

np.linspace(-2.4, 2.4, 10), # 车位置

np.linspace(-3.0, 3.0, 10), # 车速度

np.linspace(-0.2095, 0.2095, 10), # 杆角度

np.linspace(-3.5, 3.5, 10) # 杆顶端的速度

]

def discretize\_state(state):

discretized = []

for i, s in enumerate(state):

discretized.append(np.digitize(s, state\_bins[i]) - 1)

return tuple(discretized)

### （3）初始化 Q 表

Q 表是一个 5 维的数组，前 4 维表示状态变量的离散化索引，第 5 维表示动作。初始时 Q 表的所有元素都设置为 0。

q\_table = np.zeros([10, 10, 10, 10, 2])

### （4）Q-learning 训练

在每次回合中，智能体选择动作，获取奖励，并根据 Q-learning 更新公式更新 Q 值。整个过程包含探索（以 `epsilon` 概率随机动作）和利用（选择当前状态下的最优动作）两部分。

for episode in range(episodes):

observation, info = env.reset()

state = discretize\_state(observation)

done = False

total\_reward = 0

while not done:

action = choose\_action(state)

next\_observation, reward, terminated, truncated, info = env.step(action)

done = terminated or truncated

next\_state = discretize\_state(next\_observation)

best\_next\_action = np.argmax(q\_table[next\_state])

q\_value = q\_table[state + (action,)]

next\_q\_value = q\_table[next\_state + (best\_next\_action,)]

q\_table[state + (action,)] += alpha \* (reward + gamma \* next\_q\_value - q\_value)

state = next\_state

total\_reward += reward

if episode % 1000 == 0:

print(f"Episode {episode}, Total Reward: {total\_reward}")

### （5）测试模型

在训练完成后，利用训练好的 Q 表进行测试。智能体会根据当前状态的 Q 值选择最优动作，并通过测试回合输出总奖励。

for \_ in range(5):

observation, info = test\_env.reset()

state = discretize\_state(observation)

done = False

total\_reward = 0

while not done:

action = np.argmax(q\_table[state])

next\_observation, reward, terminated, truncated, info = test\_env.step(action)

done = terminated or truncated

next\_state = discretize\_state(next\_observation)

state = next\_state

total\_reward += reward

print(f"Test Total Reward: {total\_reward}")

## 5. 中间结论

- 探索与利用的平衡：在训练的早期，通过 `epsilon-greedy` 策略，智能体以高概率进行探索，以便学习到更多环境信息。随着回合的增加，智能体逐渐减少探索，更多依赖学习到的策略。

- Q 值的逐步更新：每次状态转移后，Q 表会更新。随着训练的进行，Q 值逐渐趋于稳定，表明智能体已经学会了在不同状态下选择最优动作。

## 6. 最终结果

经过训练后，测试结果表明智能体能够在 `CartPole-v1` 环境中保持平衡达到最大步数（500 步）。以下是部分测试结果输出：

```

Test Total Reward: 500.0

Test Total Reward: 500.0

Test Total Reward: 500.0

Test Total Reward: 500.0

Test Total Reward: 500.0

```

## 7. 过程分析和结论

- 状态离散化的影响：状态离散化将连续状态映射到有限的区间，使得 Q-learning 能够适用，但也会导致信息的损失。如果离散化不够细致，智能体可能无法学到最优策略。

- Q 表的收敛：随着训练回合的增加，Q 值逐渐收敛，表明智能体学习到了如何在不同状态下选择合适的动作。最终，智能体能够通过选择最优动作在测试回合中保持杆子平衡，达到最大步数。

- 结论：通过 Q-learning 算法，我们成功地解决了 `CartPole-v1` 问题。智能体通过不断更新 Q 值，逐渐学会如何保持平衡，最终在测试中获得了最优表现（最大总奖励）。

## 实验中的问题

在解决 `CartPole-v1` 强化学习问题的过程中，我遇到了很多的问题，下面是我对这些问题的分析和总结：

### (1) 环境配置和依赖问题

问题：

- 我在使用 `gym` 环境时，遇到了与 `numpy` 版本不兼容的问题，提示 `np.bool8` 不再被支持。

解决方案：

- 这是由于 `numpy` 和 `gym` 库的版本问题，特定版本的 `numpy` 已经弃用了 `np.bool8` 类型。在这里，我尝试过降级 `numpy`，降级`gym` ，手动修改相关源码以解决兼容性问题，但是很多次尝试都没啥用，最终我通过降级 `numpy` 为1.26版本成功运行了程序。

总结：

- 依赖库兼容性：在使用 Python 库时，不同版本的库可能存在不兼容的问题。通过确保依赖库的版本匹配或降级某些库，可以解决类似的问题。

### Gym 环境参数理解

问题：

- 我通过网上查阅资料了解了 `CartPole-v1` 环境中定义的参数以及这些参数如何影响智能体的行为，比如最大步数、杆的角度限制、小车的位置限制等。

解决方案：

- 通过代码了解了如何修改环境参数，如最大步数、杆角度限制和小车位置边界。我通过直接修改 `env.env` 的属性来更改这些参数，理解如何修改`CartPole-v1` 环境中的各项参数。

总结：

- 环境配置：理解和调整强化学习环境的参数（如最大步数和物理限制）对提升算法性能和定制需求非常重要。了解环境的细节参数能够帮助我更好地控制问题。

### Q-learning 算法及其参数

问题：

- 我在网上查阅 Q-learning 的公式、训练过程、Q 表中存储的内容以及如何根据 Q-learning 算法更新 Q 值。

解决方案：

- Q-learning 是基于状态-动作对的预期回报更新的值迭代算法。通过探索与利用相结合的策略，智能体能够通过训练逐渐学到最优策略。我通过离散化状态空间来使用 Q-learning，并在 Q 表中存储每个状态-动作对的 Q 值。

总结：

- 算法理解：Q-learning 算法的核心是通过不断更新 Q 值来优化智能体的策略。在这个过程中，理解 Q-learning 的公式和 Q 表的更新机制是非常关键的。

### 状态离散化问题

问题：

- 了解 `discretize\_state` 函数的工作原理以及如何将连续的状态空间映射为离散状态。

解决方案：

- 通过将连续状态变量（小车位置、速度、杆的角度和角速度）离散化为固定区间，智能体可以更容易地在离散空间中学习最优策略。使用了 `np.digitize` 来实现状态离散化，并成功在 Q 表中存储和更新这些离散状态。

总结：

- 状态离散化：Q-learning 通常用于离散状态空间，因此需要将连续状态空间离散化。离散化的区间数量影响算法的性能，区间太少可能导致精度不够，而区间太多则可能导致维度灾难。

## 源代码以及运行结果

源码：

import gym

import numpy as np

# 状态空间离散化的边界

state\_bins = [

    np.linspace(-2.4, 2.4, 10),   # 车位置

    np.linspace(-3.0, 3.0, 10),   # 车速度

    np.linspace(-0.2095, 0.2095, 10),  # 杆角度 (-12度到12度)

    np.linspace(-3.5, 3.5, 10)    # 杆顶端的速度

]

# 将连续状态离散化为表格

def discretize\_state(state):

    discretized = []

    #enumrate返回索引和对应的值

    for i, s in enumerate(state):

        discretized.append(np.digitize(s, state\_bins[i]) - 1)

    return tuple(discretized)

# 初始化 Q 表

q\_table = np.zeros([10, 10, 10, 10, 2])

# Q-learning 参数

alpha = 0.1   # 学习率

gamma = 0.99  # 折扣因子

epsilon = 0.1  # 探索率

episodes = 10000  # 训练回合数

# 创建环境

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

# 修改环境参数

env.env.theta\_threshold\_radians = 12 \* 2 \* 3.14159 / 360  # 修改杆角度限制为 ±12 度

env.env.x\_threshold = 2.4  # 修改小车的位置限制为 ±2.4

env.env.max\_episode\_steps=1000 #修改最大步数

def choose\_action(state):

    if np.random.uniform(0, 1) < epsilon:

        return env.action\_space.sample()  # 随机探索

    else:

        return np.argmax(q\_table[state])  # 利用当前策略选择最优动作

# Q-learning 训练过程

for episode in range(episodes):

    observation, info = env.reset()  # 获取重置后的 observation 和 info

    state = discretize\_state(observation)  # 对 observation 进行离散化

    done = False

    total\_reward = 0

    while not done:

        action = choose\_action(state)

        next\_observation, reward, terminated, truncated, info = env.step(action)

        done = terminated or truncated

        next\_state = discretize\_state(next\_observation)

        # 更新 Q 值

        best\_next\_action = np.argmax(q\_table[next\_state])

        q\_value = q\_table[state + (action,)]

        next\_q\_value = q\_table[next\_state + (best\_next\_action,)]

        q\_table[state + (action,)] += alpha \* (reward + gamma \* next\_q\_value - q\_value)

        state = next\_state

        total\_reward += reward

    if episode % 1000 == 0:

        print(f"Episode {episode}, Total Reward: {total\_reward}")

env.close()

# 测试训练好的策略

# 创建测试环境，指定渲染模式

test\_env = gym.make('CartPole-v1', render\_mode='human')

for \_ in range(5):

    observation, info = test\_env.reset()

    state = discretize\_state(observation)

    done = False

    total\_reward = 0

    while not done:

        action = np.argmax(q\_table[state])

        next\_observation, reward, terminated, truncated, info = test\_env.step(action)

        done = terminated or truncated  # 新版 Gym 处理终止条件

        next\_state = discretize\_state(next\_observation)

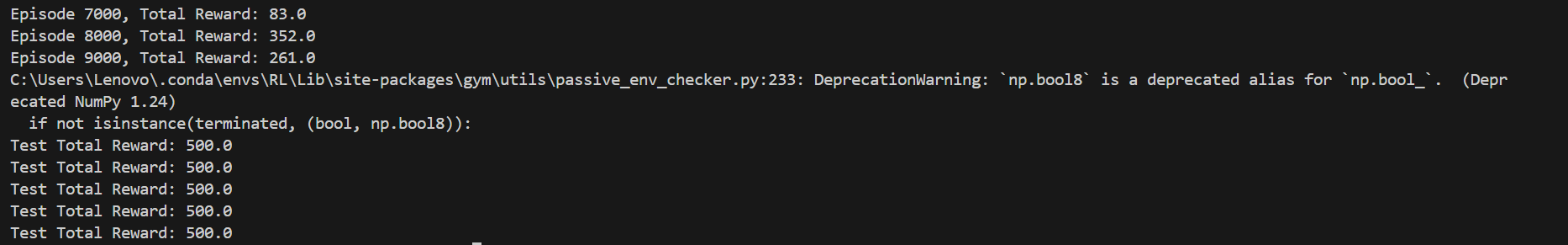
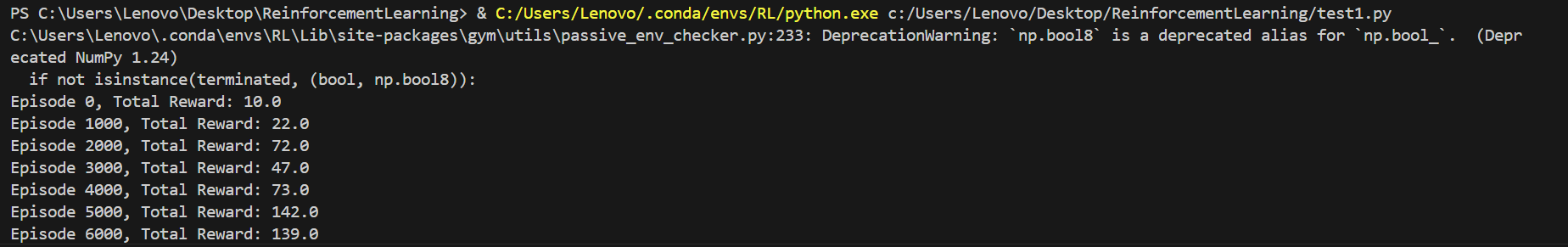
        state = next\_state

        total\_reward += reward

    print(f"Test Total Reward: {total\_reward}")

test\_env.close()

运行结果：



## 总结：

Q-learning 通过探索与利用相结合的方式，使得智能体能够逐步学习如何解决 `CartPole-v1` 的控制问题。最终，智能体能够在 500 步内保持杆子的平衡(虽然不是每次训练都能达到500)，达到了预期的效果。