

**腾讯开悟平台实验手册**

Enhanced Learning Case Experiment Manual

第一版

宋麟

二〇二四年六月

实验目录

[一、环境搭建 3](#_Toc13986)

[二、项目创建 6](#_Toc9347)

[三、实验案例 7](#_Toc21451)

1. **环境搭建**

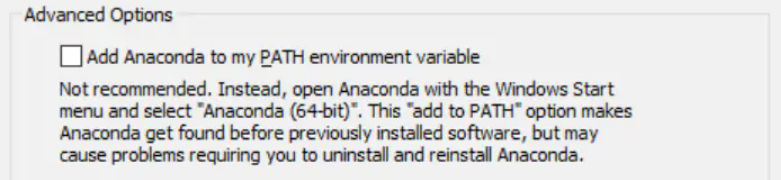
（本环境将使用Anaconda管理工具来搭建Python环境）

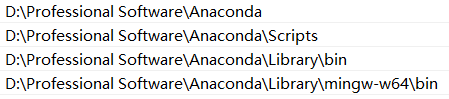
1. Anaconda安装

注意事项：

（1）注意安装目录文件夹要为全英文、无空格，不要加中文

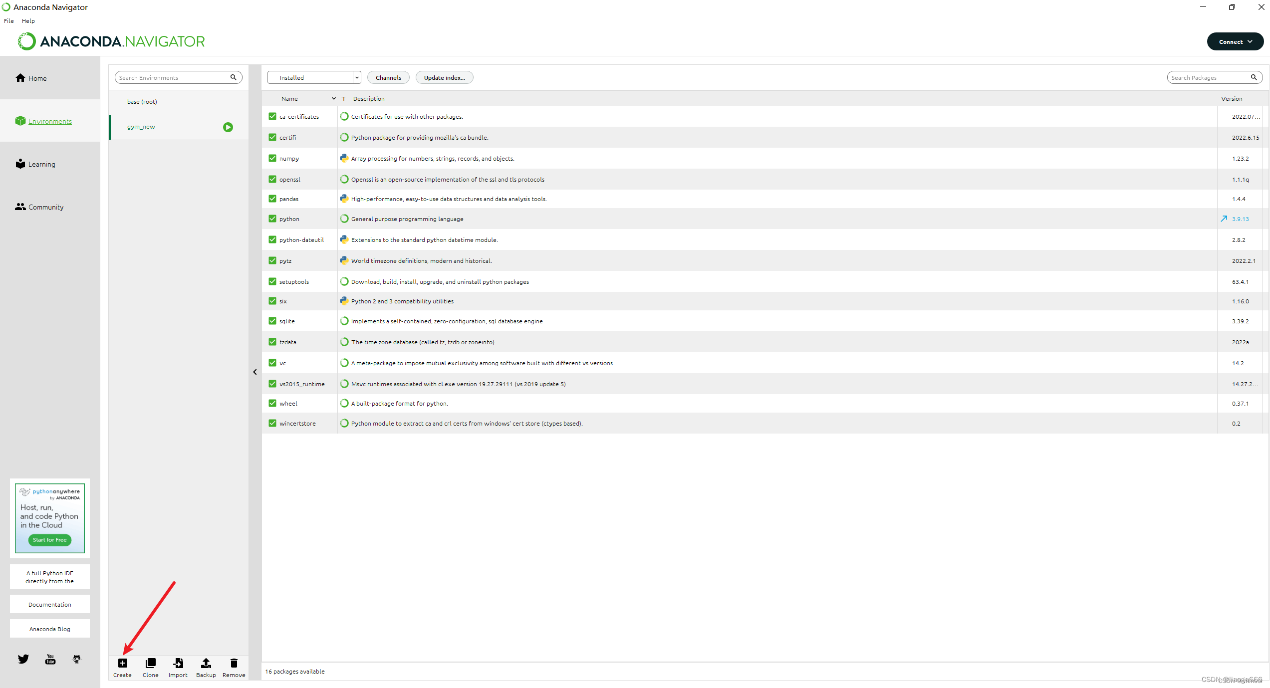
（2）将路径添加到系统环境变量中（这里安装时可以打勾）



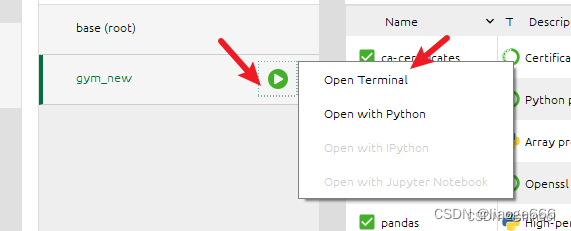
1. 若已下载好，但未添加到系统变量中，可以按如下操作：
2. 找到安装路径，在路径中找到类似如图的四个路径
3. 编辑系统环境变量，打开该页面，点击环境变量



1. 在系统变量的Path中新建环境变量将步骤一中的四个路径，分别添加进去即可
2. 验证anaconda安装成功：使用命令符cmd，弹出命令运行窗口，输入conda -V，出现版本号，说明安装成功。
3. 打开安装好的Anaconda Navigator，选择第二项environment，然后在左下角点击新建create



1. 命名新环境选择Python版本
2. 点击新环境右侧的按钮，选择open terminal进入指令窗口



具体操作如下：

（1）升级pip

python -m pip install --upgrade pip

（2）永久更换为清华的pip源，（这个下载速度会比官网快的多），也可以不换

pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pandas

（3）安装gym0.26.2（最新版本会更新的，这里只是参考版本）版本包和pyglet包，

安装gym：

pip install gym

安装pyglet：

pip install pyglet

（4）安装pygame（调用程序中的动画窗口）

pip install pygame

1. 降级pip版本

python -m pip install pip==21.3.1 setuptools==59.5.0

1. 降级gym版本

pip install gym==0.21.0

1. 降级box2d-py版本

pip install box2d-py==2.3.5

(8) 安装gym的实验环境

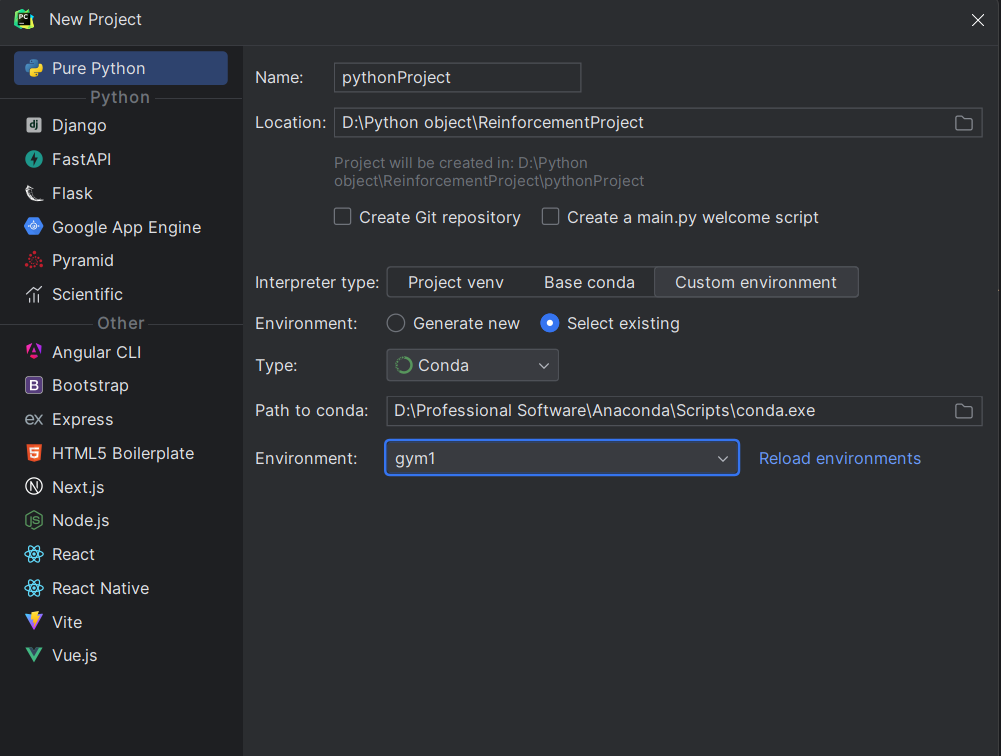
pip install gym[all]

(9)降级pyglet渲染器

pip install pyglet==1.5.27

**项目创建**

1. 下载Pycharm
2. 打开Pycharm创建新项目，并选择刚刚配好的环境（我的环境是gym1）



1. 编写代码，运行
2. **实验案例**
3. 月球登陆器

**（1）实验介绍**

本实验通过训练智能体在OpenAI Gym中的LunarLander环境中进行决策，以实现成功着陆。实验使用深度强化学习中的近端策略优化（PPO）算法，开发者可以修改源代码中的各个模块，以了解和优化强化学习算法的实际应用。

**（2）代码结构**

main.py: 包含训练主循环和与环境的交互逻辑。

ppo.py: 包含PPO算法的具体实现，包括策略网络、价值网络的定义和更新。

**（3）代码理解与修改（若对PPO算法已有理解可以修改PPO算法内部的代码）：**

训练主循环（main.py）：

修改训练参数，如最大迭代次数、学习率等。

控制训练过程中的日志输出和模型保存逻辑。



**（4）训练过程**

启动训练过程，观察训练进展和日志输出。

根据需要调整参数和模型结构，优化训练效果。

**（5）模型评估**

在训练完成后，对智能体进行评估，查看其在LunarLander环境中的表现。

评估过程中可以调整模型的超参数，并通过多轮测试记录其着陆成功率、奖励值等指标。

**（6）结果分析**

根据训练和评估结果，分析智能体的表现，找出模型的优点和不足。

通过调整奖励函数、超参数等，进一步优化智能体的策略，提高其在任务中的表现。

以及强化学习的理论原理，以下是这两个程序代码文档执行的流程概述：

执行流程

1. 环境初始化与状态获取

代码文件：main.txt

具体参数和语句：

env = gym.make(env\_name)：初始化环境，env\_name是环境名称（如"LunarLander-v2"）。

state = env.reset()：重置环境，获取初始状态state。

2. 行动选择与执行

代码文件：main.txt 和 PPO.txt

具体参数和语句：

在main.txt中：

action = ppo.policy\_old.act(state, memory)：使用旧策略网络policy\_old根据当前状态state选择行动action。

在PPO.txt的ActorCritic类中：

state = torch.from\_numpy(state).float().to(device)：将状态state转换为张量，并移动到合适的设备（GPU或CPU）。

action\_probs = self.action\_layer(state)：计算行动概率。

dist = Categorical(action\_probs)：将行动概率转换为类别分布。

action = dist.sample()：从类别分布中采样得到行动action。

回到main.txt：

state, reward, done, \_ = env.step(action)：执行行动action，环境返回新的状态state、奖励reward、是否终止done等信息。

3. 环境反馈与记忆体更新

代码文件：main.txt

具体参数和语句：

memory.rewards.append(reward)：将奖励reward添加到记忆体memory中。

memory.is\_terminals.append(done)：将是否终止done信息添加到记忆体memory中。

（虽然代码中未显式更新行动和状态到记忆体，但在act方法中，memory.states.append(state)和memory.actions.append(action)已经被调用）

4. 策略更新与梯度计算

代码文件：PPO.txt

具体参数和语句：

old\_states = torch.stack(memory.states).to(device).detach()：从记忆体中提取状态，转换为张量，并移动到合适的设备。

old\_actions = torch.stack(memory.actions).to(device).detach()：从记忆体中提取行动，转换为张量，并移动到合适的设备。

old\_logprobs = torch.stack(memory.logprobs).to(device).detach()：从记忆体中提取行动概率对数，转换为张量，并移动到合适的设备。

logprobs, state\_values, dist\_entropy = self.policy.evaluate(old\_states, old\_actions)：使用当前策略网络policy评估状态，计算行动概率、评判值和行动概率熵。

ratios = torch.exp(logprobs - old\_logprobs.detach())：计算比率。

advantages = rewards - state\_values.detach()：计算优势。

loss = -torch.min(surr1, surr2) + 0.5 \* self.MseLoss(state\_values, rewards) - 0.01 \* dist\_entropy：计算损失函数。

self.optimizer.zero\_grad()：梯度清零。

loss.mean().backward()：反向传播计算梯度。

self.optimizer.step()：更新策略网络权重。

5. 权重复制与策略迭代

代码文件：PPO.txt

具体参数和语句：

self.policy\_old.load\_state\_dict(self.policy.state\_dict())：将更新后的策略网络policy的权重复制给旧策略网络policy\_old。

环境初始化：

在main.txt中，通过gym.make(env\_name)初始化游戏环境（如LunarLander-v2），并设置相关的超参数，如状态维度（state\_dim）、行动维度（action\_dim）、最大迭代次数（max\_episodes）等。

初始化环境后，通过env.reset()重置环境，获取初始状态。

策略网络和价值网络初始化：

在PPO.txt中，定义了ActorCritic类，该类包含了策略网络（行动部分）和价值网络（评判部分）。

在main.txt中，通过实例化PPO类，创建了两个ActorCritic网络实例：policy（当前策略网络）和policy\_old（旧策略网络）。这两个网络具有相同的结构，但权重不同。policy\_old的权重在初始化时从policy复制。

训练循环：

在main.txt中，通过外层循环（for i\_episode in range(1, max\_episodes + 1)）控制训练的总迭代次数。

在每次迭代开始时，通过env.reset()重置环境，获取新的初始状态。

数据收集：

在内层循环（for t in range(max\_timesteps)）中，使用旧策略网络policy\_old根据当前状态选择行动（action = ppo.policy\_old.act(state, memory)）。

执行行动后，环境返回新的状态、奖励、是否终止等信息。这些信息被用于更新记忆体memory（包括行动、状态、奖励、是否终止等）。

策略更新：

当达到一定的时间步数（update\_timestep）后，使用记忆体memory中的数据更新策略网络。

在PPO.txt的update方法中，首先计算每个时间步的折扣奖励（discounted\_reward），并对奖励进行标准化处理。

然后，使用当前策略网络policy对记忆体中的状态进行评估，计算行动概率、评判值等。

接着，计算比率（ratios）和优势（advantages），并根据PPO的损失函数计算总损失。

最后，通过反向传播更新策略网络的权重，并将更新后的权重复制给旧策略网络policy\_old。

可视化与日志记录：

在训练过程中，如果设置了render=True，则会在每次行动后调用env.render()进行可视化。

每隔一定的迭代次数（log\_interval），打印平均游戏步长和平均奖励等信息，用于监控训练进度。

模型保存与训练终止：

如果在某次迭代中，总奖励达到了预设的停止条件（solved\_reward），则打印“Solved!”信息，并保存当前策略网络的权重到文件。

当达到最大迭代次数（max\_episodes）后，训练终止。

逻辑与强化学习理论原理对应

环境初始化：对应强化学习中的环境设置，包括状态空间和行动空间的定义。

策略网络和价值网络初始化：对应强化学习中的策略和价值函数的表示，通常使用神经网络进行建模。

训练循环：对应强化学习中的迭代训练过程，通过不断与环境交互来优化策略。

数据收集：对应强化学习中的经验收集过程，通过执行策略来获取状态、奖励等信息。

策略更新：对应强化学习中的策略优化过程，通过计算梯度来更新策略网络的权重。

可视化与日志记录：对应强化学习中的监控和调试过程，有助于了解训练进度和性能。

模型保存与训练终止：对应强化学习中的模型保存和训练终止条件设置，有助于保存训练结果和结束训练过程。