强化学习实验指导

一、实验背景

强化学习（Reinforcement Learning, RL）是机器学习的一个重要分支，其核心思想是智能体（Agent）通过与环境交互，不断调整策略以最大化累积奖励。本实验围绕四种典型的策略梯度方法：

- **REINFORCE**；
- **Baseline**：引入基线的**REINFORCE**；
- **Actor-Critic**：引入价值函数减小方差；
- **PPO（Proximal Policy Optimization）**：稳定性更强的近端策略优化方法。

通过对比这几种算法在经典控制环境 `CartPole-v1` 中的表现，理解策略梯度方法的训练过程、稳定性与收敛性能。

二、实验目标

1. 理解五种算法中的 REINFORCE、Actor-Critic 和 PPO 三种核心算法的**基本原理。**
2. 掌握 `PaddlePaddle` 框架中**策略网络（`PolicyNet`）**与**价值网络（`ValueNet`）**的构建方法。
3. 实现并训练五种策略梯度算法，评估其在 `CartPole-v1` 环境中的性能。
4. 比较五种方法的**收敛速度与最终得分**，分析其优劣。

三、本地环境配置说明

本次实验算法流程简单，非常建议直接线上运行。

四、实验内容

任务总览

按顺序完成从基础策略梯度到近端策略优化算法实现。通过本项目将理解：

- 策略函数如何决定动作
- 梯度上升优化动作选择策略
- 状态值函数降低方差的作用
- 策略函数与价值函数的协同训练
- PPO的探索与稳定更新机制

任务中常见符号的解释

符号	含义简述
θ	**策略网络的参数**
L_{actor}	**策略网络的损失函数,用于更新 `PolicyNet`**
s_t	**Agent第 t 步的状态（State）**
a_t	**Agent第 t 步选择的动作（Action）**

符号	含义简述
r_t	**Agent第 t 步获得的奖励（Reward）**
γ	**折扣因子（0～1），控制未来奖励的重要性**
G_t	**第 t步开始的总奖励（回报）**
A_t	**优势函数，表示某动作比平均值好多少**
$V(s_t)$	**价值网络输出的当前状态 s_t 的估计回报**
L_{critic}	**价值网络的损失函数,用于更新 `ValueNet`**
$r_t(\theta)$	**PPO 中旧策略和新策略的动作概率比值**
ϵ	**PPO 中用来限制策略更新的超参数**
$L^{CLIP}(\theta)$	**PPO 的总损失函数**

任务中通常会用到的函数：

`paddle.mean()`:计算输入张量（tensor）所有元素的平均值

`paddle.nn.functional.mse_loss()`:计算均方误差（MSE）损失

任务1：实现标准 REINFORCE 算法（REINFORCE v2）

核心思想:
REINFORCE 是一种基于策略梯度的算法，目标是直接优化策略网络，使其产生的动作能最大化未来累积奖励。关键是用**折扣累计回报** G_t 衡量每个动作的好坏。

具体步骤：

1. **采集完整轨迹**：智能体与环境交互，执行动作直到终止，记录每一步的状态 s_t、动作 a_t、奖励 r_t 。
2. **计算折扣累计回报**:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

   其中，$\gamma \in (0, 1]$ 是折扣因子，用于降低远期奖励的权重。

3. **计算损失并更新策略**:
   损失函数为:

$$L_{\text{actor}} = -\frac{1}{T}\sum_{t=0}^{T-1} \log \pi(a_t|s_t) \cdot G_t$$

   这里的 $\log \pi(a_t|s_t)$ 是策略网络输出的动作概率对数，乘以累计回报 G_t 指明奖励对该动作的影响。用负号是为了做梯度下降优化最大化期望回报。

这个算法简单直观，但在实践中可能因为直接使用 G_t 导致梯度方差较大，训练不稳定。

在我们的代码中，这一部分的实现在`Reinforce`类

任务2：引入基线函数(Baseline)的 REINFORCE

核心思想：
直接用 G_t 作为梯度权重可能导致梯度估计方差很大，影响训练效率。我们引入一个**基线（Baseline）**，一般是一个常数或状态值的估计，来减小方差。

具体步骤：

1. **计算优势函数**：

$$A_t = G_t - b$$

   其中，b 是基线，常见选择为轨迹的平均回报或者状态值估计 $V(s_t)$，这里我们选择**轨迹的平均回报**作为b。

2. **策略损失函数改写为**：

$$L_{\text{actor}} = -\frac{1}{T}\sum_{t=0}^{T-1}\log \pi(a_t|s_t)\cdot A_t$$

   利用优势函数 A_t 代替原来的 G_t，减少方差。

3. **通过损失函数反向传播**，更新策略网络参数。

基线不会引入偏差，因为优势函数的期望仍然是梯度的无偏估计，但能显著降低训练中的梯度方差，提升收敛速度。

在我们的代码中，这一部分的实现在`ReinforceWithBaseline`类

任务3：实现 Actor-Critic 算法

核心思想：
将策略（Actor）与价值评估（Critic）结合起来，Actor负责选择动作，Critic负责给出状态价值估计辅助Actor优化。

3.1 蒙特卡洛版本

1. **计算优势**：

$$A_t = G_t - V(s_t)$$

   这里，$V(s_t)$ 是价值网络对当前状态价值的预测。

2. **策略网络损失**：

$$L_{\text{actor}} = -\frac{1}{T}\sum_{t}\log \pi(a_t|s_t)\cdot A_t$$

3. **价值网络损失**：

$$L_{critic} = \frac{1}{T}\sum_{t}(V(s_t) - G_t)^2$$

   通过均方误差最小化价值网络预测与实际回报的差距。

3.2 时序差分版本（TD）

1. **计算 TD 误差**：

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

2. **策略损失**：

$$L_{\text{actor}} = -\frac{1}{T}\sum_t \log \pi(a_t|s_t) \cdot \delta_t$$

   注意此处用 TD 误差 δ_t 替代优势函数。

3. **价值网络损失**：

$$L_{critic} = \frac{1}{T}\sum_t \delta_t^2$$

TD版本更高效，无需等待整条轨迹结束，适合在线学习。需要注意在计算梯度时，通常对 δ_t 做 `detach()`，防止其梯度回传影响策略网络的梯度计算。

在我们的代码中，这一部分的实现在 `ActorCriticWithMC` 类 `ActorCriticWithTD` 类

任务4：实现 PPO（Proximal Policy Optimization）算法

核心思想：
PPO 通过限制策略更新幅度，避免策略更新过大导致性能骤降，从而实现稳定高效的策略优化。

具体步骤：

1. **收集批量轨迹**，计算回报 G_t、价值估计 $V(s_t)$、优势函数 $A_t = G_t - V(s_t)$。
2. **计算新旧策略概率比**：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

3. **裁剪目标函数**：

$$L_{\text{actor}} = -\frac{1}{T}\sum_t \min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)A_t\right)$$

   ϵ 是一个超参数（如0.2），用来限制策略变动幅度。

4. **价值网络损失**：

$$L_{critic} = \frac{1}{T}\sum_t (V(s_t) - G_t)^2$$

5. **同一批数据上重复多次训练**，增强训练稳定性。

在我们的代码中，这一部分的实现在 `PPO` 类

任务5：自定义强化学习算法建议

你可以结合自己的想法，对现有算法进行改进或设计新算法，思考以下方向：

- **奖励函数设计**: 设计更符合任务目标的奖励。
- **损失函数设计**：加入熵正则化、KL散度等，稳定训练。
- **策略网络结构**：尝试双层多层感知机（MLP）、卷积网络或残差网络。

- **采样和训练流程**：改进样本利用率，设计经验回放，异步训练等。

在我们的代码中，这一部分的实现在`MyAgent`类

五、作业报告要求

请每位同学提交一份实验报告（PDF 或 Word 格式），内容包括：

1. **实验结果**：提交可视化的曲线结果,根据曲线结果进行对比分析。
2. **实验代码复现**：提交完整的代码文件（`main.ipynb`）。
3. **实验报告**：提交实验报告，回答实验报告中的问题（**Report.pdf**）。
4. **问答题作答**：提交问答题的作答（**QA.pdf**）。
5. **其他任何你想说明的东西**：非重要不提交。