程序报告

学号：22375080　　　　　　姓名：杨佳宇轩

一、问题重述

（简单描述对问题的理解，从问题中抓住主干，必填）

==

强化学习通过智能体与环境交互优化策略，本实验通过策略梯度方法，如 REINFORCE、Actor-Critic、PPO 等，在 CartPole-v1 环境中对比算法的收敛性与稳定性。

通过不同策略梯度算法使得智能体在 CarPole-v1 中维持平衡，并最大化累计奖励

二、设计思想

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

==

REINFORCE 算法：用累计回报衡量动作价值，但存在梯度方差大，训练不稳定等问题

带基线的 REINFORCE：引入相对回报作为优势函数

Actor-Critic 算法：分为蒙特卡洛版本以及时序差分版本，更适合在线学习

PPO 算法：通过裁剪目标函数限制策略更新幅度，提升模型训练稳定性

三、代码内容

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

==

TODO1：

```
elif self.version == 2:
    R = 0
    for r in reversed(rewards):
        R = r + self.gamma * R
        A.insert(0, R)
```

TODO2：

```
returns = []
R = 0
for r, d in zip(reversed(rewards), reversed(dones)):
    if d:
        R = 0
    R = r + self.gamma * R
    returns.insert(0, R)
returns = paddle.to_tensor(returns, dtype='float32').reshape((-1, 1))
values = self.critic(states)
advantages = returns - values
```

TODO3：

```
    # 1. 计算折扣累计回报 Gt
    returns = []
    G = 0
    for r, d in zip(reversed(rewards), reversed(dones)):
        if d:
            G = 0
        G = r + self.gamma * G
        returns.insert(0, G)
    returns = paddle.to_tensor(returns, dtype='float32').reshape((-1, 1))

    # 2. 使用 critic 网络估计当前状态价值 V(s)
    values = self.critic(states)

    # 3. 计算 advantage = Gt - V(s)
    advantages = returns - values

    # 4. 策略网络输出动作概率，并选择对应动作的概率
    probs = self.actor(states)
    action_probs = paddle.gather(probs, axis=1, index=actions)
    log_probs = paddle.log(action_probs)

    # 5. 计算 actor loss
    actor_loss = paddle.mean(-log_probs * advantages.detach())

    # 6. 计算 critic loss
    critic_loss = F.mse_loss(values, returns)

    # 7. 清空梯度，反向传播并更新 actor 和 critic 网络
    self.actor_optimizer.clear_grad()
    actor_loss.backward()
    self.actor_optimizer.step()

    self.critic_optimizer.clear_grad()
    critic_loss.backward()
    self.critic_optimizer.step()
```

TODO4：

```
    dones = paddle.to_tensor(transition_dict['dones'], dtype='float32').reshape((-1, 1))

    # 1. 判断 episode 是否结束，done=1 时未来价值不计入
    #  已在输入参数中处理，dones 为 0/1 张量

    # 2. 计算 TD target：r + γ * V(s') * (1 - done)
    next_values = self.critic(next_states)
    td_targets = rewards + self.gamma * next_values * (1 - dones)
```

```
        # 3. 计算 TD delta：δ = TD target - V(s)
        values = self.critic(states)
        td_deltas = td_targets - values

        # 4. 计算策略网络输出的动作概率 probs，并选中 actions 对应的概率
        probs = self.actor(states)
        action_probs = paddle.gather(probs, axis=1, index=actions)
        log_probs = paddle.log(action_probs)

        # 5. 计算 actor loss
        actor_loss = paddle.mean(-log_probs * td_deltas.detach())

        # 6. 计算 critic loss
        critic_loss = F.mse_loss(values, td_targets)

        # 7. 清空梯度，反向传播，更新 actor 与 critic
        self.actor_optimizer.clear_grad()
        actor_loss.backward()
        self.actor_optimizer.step()

        self.critic_optimizer.clear_grad()
        critic_loss.backward()
        self.critic_optimizer.step()
```

TOOD5：

```
        # 1. 计算  TD target：r + γ * V(s') * (1 - done)
        next_values = self.critic(next_states)
        td_target = rewards + self.gamma * next_values * (1 - dones)

        # 2. 计算  TD delta：δ = TD target - V(s)
        values = self.critic(states)
        td_delta = td_target - values

        # 3. 调用  compute_advantage  函数计算 advantage
        advantage = self.compute_advantage(self.gamma, self.lmbda, td_delta)
        probs = self.actor(states)
            old_log_probs = paddle.log(paddle.take_along_axis(probs, actions, axis=1) +
1e-8).detach()
```

TODO6：

```
        critic_loss=F.mse_loss(values, td_target)
```

TODO7：

```
class MyAgent:
    def __init__(self, state_dim, action_dim):
        """
```

```
    初始化强化学习方法
    默认超参数设置:
        hidden_dim = 128
        actor_lr = 1e-3
        critic_lr = 1e-2
        gamma = 0.98
        lmbda = 0.95 (GAE 参数)
        epochs = 10 (每个批次数据的更新次数)
        eps = 0.2 (PPO 裁剪参数)
    """
    hidden_dim = 128
    actor_lr = 1e-3
    critic_lr = 1e-2
    gamma = 0.98
    lmbda = 0.95
    epochs = 10
    eps = 0.2

    self.actor = PolicyNet(state_dim, hidden_dim, action_dim)
    self.critic = ValueNet(state_dim, hidden_dim)

    self.actor_optimizer = paddle.optimizer.Adam(
        parameters=self.actor.parameters(),
        learning_rate=actor_lr
    )
    self.critic_optimizer = paddle.optimizer.Adam(
        parameters=self.critic.parameters(),
        learning_rate=critic_lr
    )

    self.gamma = gamma
    self.lmbda = lmbda
    self.epochs = epochs
    self.eps = eps

def take_action(self, state):
    state = paddle.to_tensor(np.array([state]), dtype='float32')
    probs = self.actor(state)
    action_dist = paddle.distribution.Categorical(probs)
    action = action_dist.sample([1]).numpy()[0]
    return action.item()

def compute_advantage(self, gamma, lmbda, td_delta):
    """计算广义优势估计(GAE)"""
```

```
        td_delta = td_delta.detach().numpy()
        advantage_list = []
        advantage = 0.0
        for delta in td_delta[::-1]:
            advantage = gamma * lmbda * advantage + delta
            advantage_list.append(advantage)
        advantage_list.reverse()
        advantage = paddle.to_tensor(advantage_list, dtype='float32')
        return advantage

    def update(self, transition_dict):
        states = paddle.to_tensor(transition_dict['states'], dtype='float32')
        actions = paddle.to_tensor(transition_dict['actions']).reshape((-1, 1))
        rewards = paddle.to_tensor(transition_dict['rewards'], dtype='float32').reshape((-1, 1))
        next_states = paddle.to_tensor(transition_dict['next_states'], dtype='float32')
        dones = paddle.to_tensor(transition_dict['dones'], dtype='float32').reshape((-1, 1))

        # 计算 TD 目标和 TD 误差
        next_values = self.critic(next_states)
        td_target = rewards + self.gamma * next_values * (1 - dones)
        values = self.critic(states)
        td_delta = td_target - values

        # 计算广义优势估计(GAE)
        advantage = self.compute_advantage(self.gamma, self.lmbda, td_delta)

        # 保存旧策略的动作概率
        probs = self.actor(states)
                old_log_probs = paddle.log(paddle.take_along_axis(probs, actions, axis=1) +
1e-8).detach()

        # 多次更新策略和价值网络
        for _ in range(self.epochs):
            probs = self.actor(states)
            log_probs = paddle.log(paddle.take_along_axis(probs, actions, axis=1) + 1e-8)
            ratio = paddle.exp(log_probs - old_log_probs)

            # PPO 裁剪目标函数
            surr1 = ratio * advantage
            surr2 = paddle.clip(ratio, 1 - self.eps, 1 + self.eps) * advantage
            actor_loss = paddle.mean(-paddle.minimum(surr1, surr2))

            # 价值网络损失
            critic_loss = F.mse_loss(values, td_target)
```

```
# 更新网络参数
self.actor_optimizer.clear_grad()
self.critic_optimizer.clear_grad()
actor_loss.backward()
critic_loss.backward()
self.actor_optimizer.step()
self.critic_optimizer.step()
```

四、实验结果

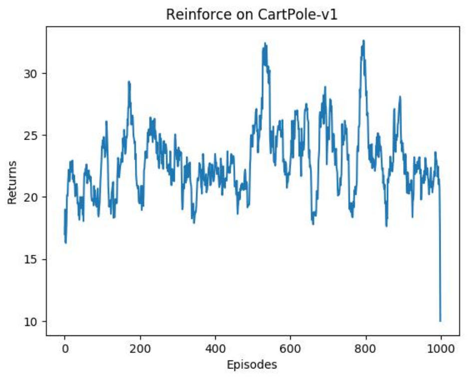
（实验结果，给出训练结果曲线，对比不同方法之间的优劣，必填）

==

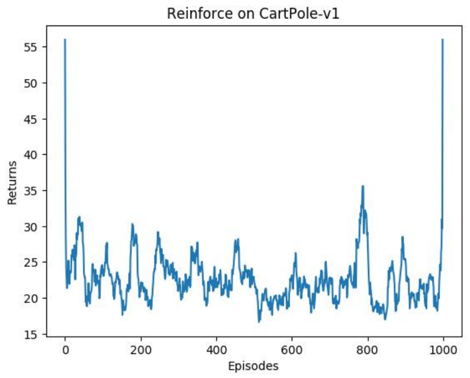
对比：

Reinforce0 - 2 版本原理简单，实现容易，但方差极大，训练结果不稳定，收敛速度慢

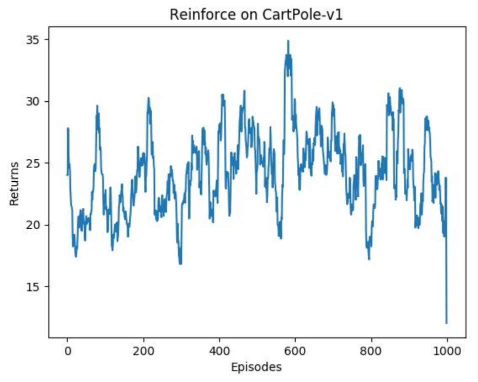
Reinforce version 0



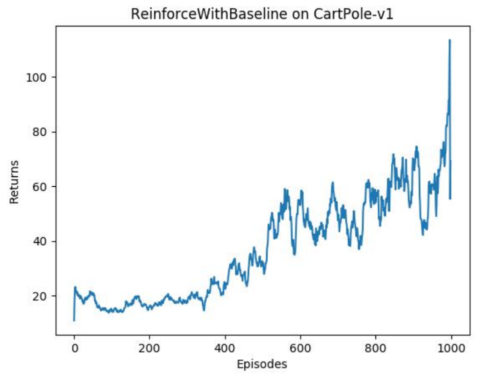
Reinforce version 1



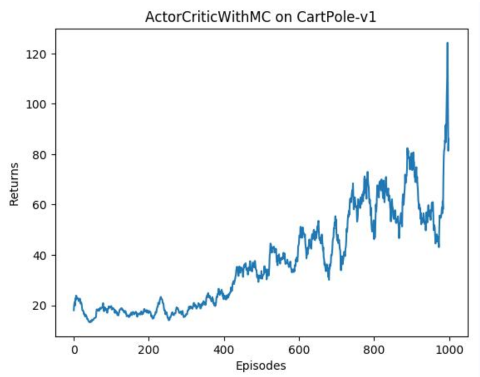
Reinforce version 2

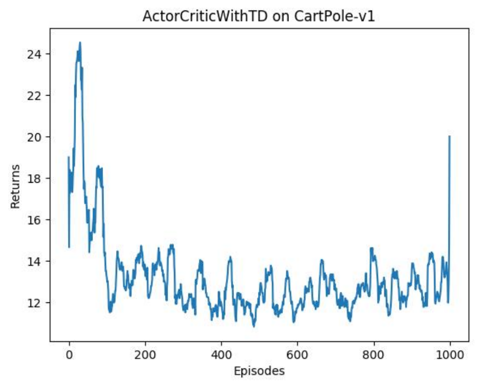


Reinforce with Baseline
方差降低，收敛速度提升，但仍需完整轨迹，样本利用率低



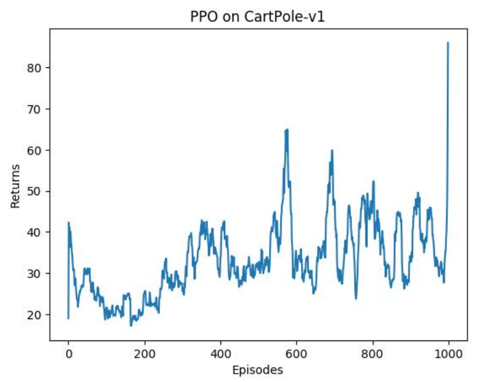
Actor-Critic with 蒙特卡洛&时序差分
蒙特卡洛和时序差分有着明显的上升趋势，强化学习效果明显
对于 MC 结合价值网络，方差进一步降低，但需要完整轨迹，计算量略增
对于 TD 在线学习，效率高，且适合连续控制，但价值网络误差可能导致偏差





PPO

稳定性较强，样本利用率高，但查参数调优重要且计算量较大



五、总结

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

===

通过超参数调优以及模型完善，在比较好的情况下，可以达到 1000 次以内 200 的优秀回报，但仍然无法得到更加优秀的结果，遇到的问题其一就是 REINFORCE 在训练时回报波动极大，难以收敛。可以优化的方向包括奖励函数的设计（对于实际问题进行建模，设计适合的奖励机制）、尝试双层 MLP 提升策略表达能力等