**程序报告**

学号：22375080 姓名：杨佳宇轩

1. **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

1. **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

1. **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()

1. **实验结果**

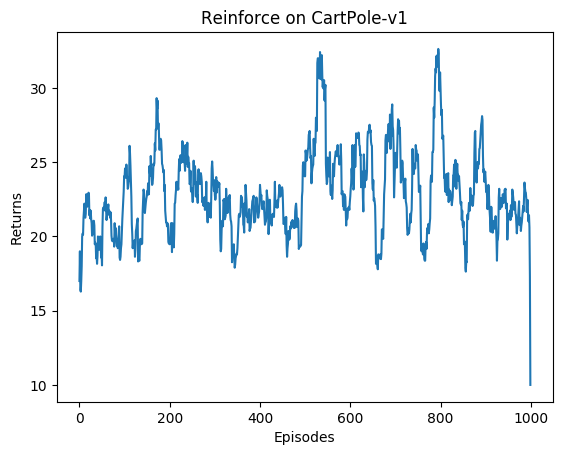
（实验结果，给出训练结果曲线，对比不同方法之间的优劣，必填）

====================================================================

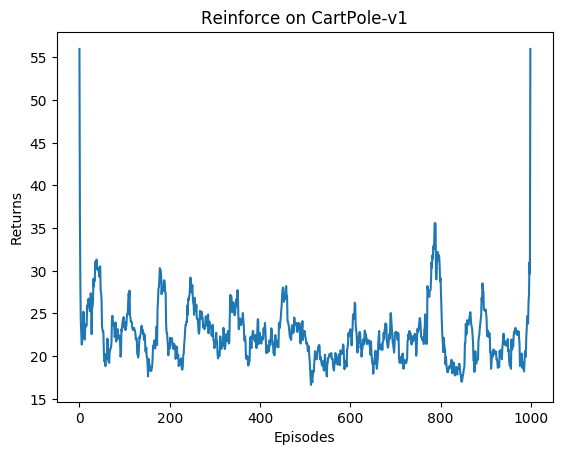
对比：

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

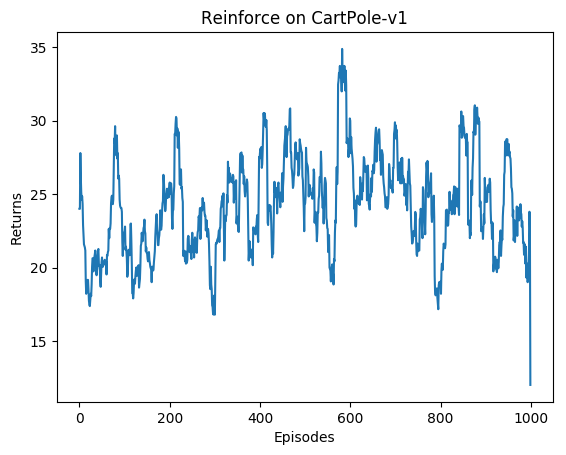
Reinforce version 0



Reinforce version 1

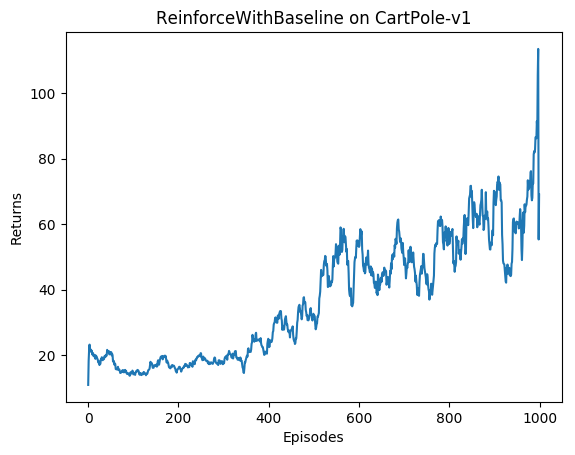


Reinforce version 2



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

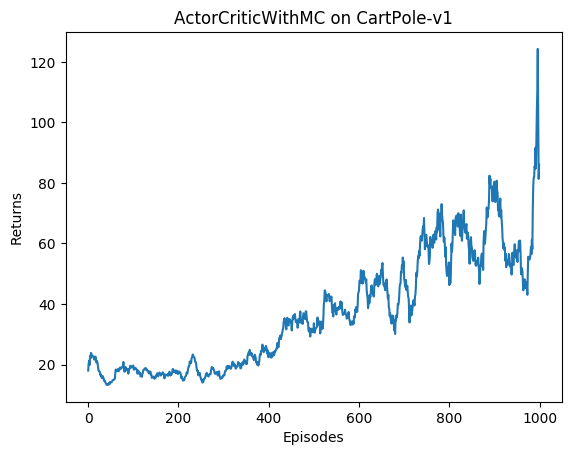


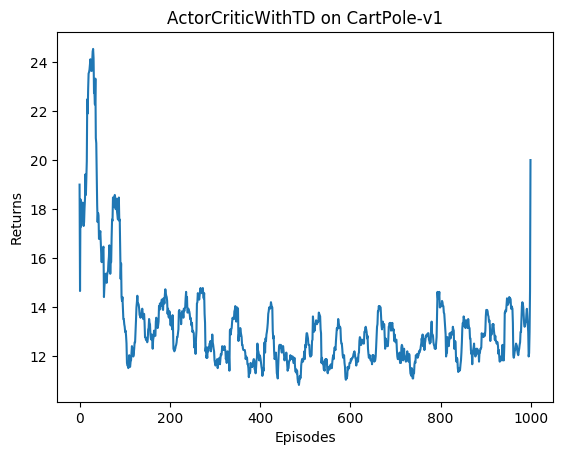
Actor-Critic with 蒙特卡洛&时序差分

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

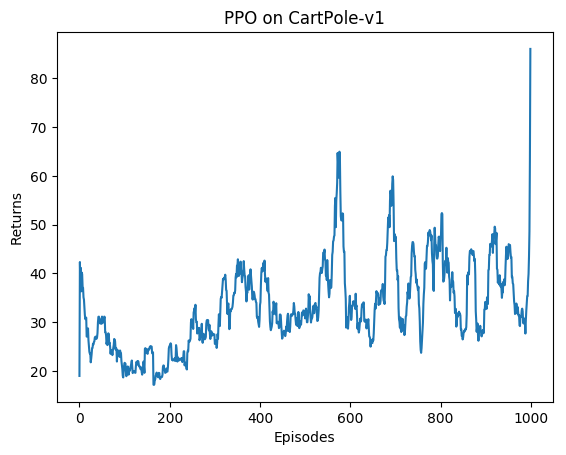
对于TD在线学习，效率高，且适合连续控制，但价值网络误差可能导致偏差





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



1. **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