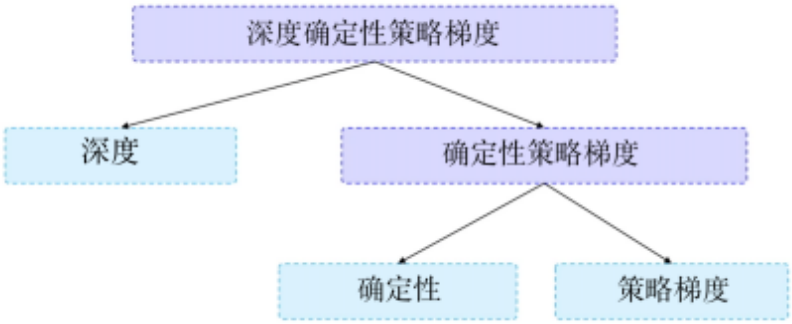
强化学习实验报告十一——DDPG深度确定性策略算法在CartPole游戏上的应用

学号:2013365 姓名:颜铭

1 实验导言

对于之前介绍过的连续奖励任务情景的Pendulum倒立摆模型，在介绍DPG的连续方法中提到过较为经典的强化学习算法是**深度确定性策略梯度算法**(DDPG)



DDPG是对深度Q网络的一个推广，具体是在AC算法的基础上研究经验回放池如何异步地结合策略梯度更新演员和评论员网络。虽然DDPG的提出是为了让深度Q网络可以推广到连续工作空间 ，但此外在离散的游戏任务中也可以学习到定向的确定离散化策略以明显得到较好的适应控制效果，本实验基于的正是倒立摆游戏环境CartPole,目标是尽量延长直立的时间

其算法伪代码如下:

|初始化评论员网络$Q(s,a|\theta_Q)$和演员网络$\mu(s|\theta_\mu)$,其权重分别为θ_Q和θ_u

初始化目标Q'和μ'，并复制权重为$\theta_{Q'} \leftarrow \theta_Q$,$\theta_{\mu'} \leftarrow \theta_\mu$#重点

初始化回放缓冲区(replay)D(Sample可选)

执行M个回合循环，对每个回合

初始化动作探索的随机过程(多为高斯过程)，即噪声$\mathcal{N}$

初始化状态s_t

循环T个时间步长，对于每个时间步长t

根据当前的策略和噪声选择动作$a_t = \mu(s_t|\theta_t)$

环境根据a_t反馈奖励r_t和下一个状态s_{t+1}

存储经验即(s_t, a_t, r_t, s_{t+1})到Sample中

更新策略执行:

从Sample中随机采样N个小批量的转移

计算实际的Q值$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta_\mu)|\theta_{Q'})$

对损失函数L(MSELoss or Entropy)进行关于参数θ的随机

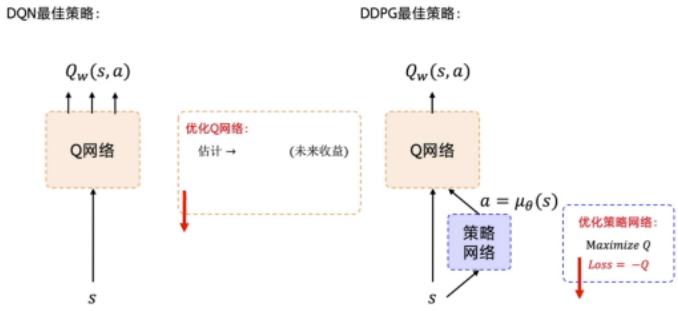
梯度下降更新评论员网络

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta_Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta_\mu)|_{s_i}$$

软更新目标网络(重点)

$$\theta^{Q'} \leftarrow \tau\theta_Q + (1-\tau)\theta_{Q'}, \theta_{\mu'} \leftarrow \tau\theta_\mu + (1-\tau)\theta_{\mu'}$$

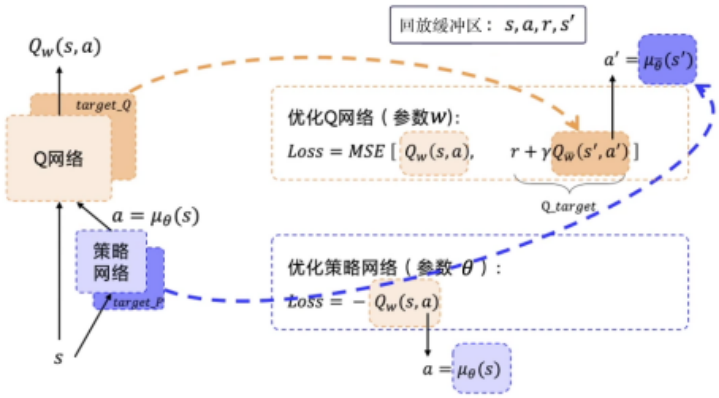
一般流程是，最开始训练的时候，这两个神经网络的参数是随机的。所以评论员最开始是随机打分的，演员也随机输出动作。但是由于有环境反馈的奖励存在，因此评论员的评分会越来越准确，所评判的演员的表现也会 越来越好。既然演员是一个神经网络，是我们希望训练好的策略网络，我们就需要计算梯度来更新优化它 里面的参数 θ 。简单来说，我们希望调整演员的网络参数，使得评委打分尽可能高。注意，这里的演员是 不关注观众的，它只关注评委，它只迎合评委的打分。



下一步结合深度Q网络的训练思想,如何与DDPG联系借助于AC算法。深度 Q 网络的最佳策略是想要学出一个很好的 Q 网 络，学出这个网络之后，我们希望选取的那个动作使 Q 值最大。DDPG 的目的也是求解让 Q 值最大的那个动作。演员只是为了迎合评委的打分而已，所以优化策略网络的梯度就是要最大化这个 Q 值，所以构 造的损失函数就是让 Q 取一个负号。我们写代码的时候把这个损失函数放入优化器里面，它就会自动最 小化损失，也就是最大化 Q。 这里要注意，除了策略网络要做优化，DDPG 还有一个 Q 网络也要优化。评论员一开始也不知道怎么 评分，它也是在一步一步的学习当中，慢慢地给出准确的分数。我们优化 Q 网络的方法其实与深度 Q 网 络优化 Q 网络的方法是一样的，我们用真实的奖励 r 和下一步的 Q 即 Q′ 来拟合未来的奖励 Q_target。 然后让 Q 网络的输出逼近 Q_target。所以构造的损失函数就是直接求这两个值的均方差。构造好损失函 数后，我们将其放到优化器中，让它自动最小化损失。

DDPG 有 4 个网络，策略网络的目标网络和 Q 网络的目标网络是颜 色比较深的这两个，它们只是为了让计算 Q_{target}更加稳定。因为这两个网络也是固定一段时间的参数之 后再与评估网络同步最新的参数。

这里训练需要用到的数据是四元组s, a, r, s'的四元组。用回放缓冲区把这些 数据存起来，然后采样进行训练。经验回放的技巧与深度 Q 网络中的是一样的。注意，因为 DDPG 使用 了经验回放技巧，所以 DDPG 是一个异策略的算法。



2 实验分析

我们首先从`DDPG`类中对应算法部分入手，首先是网络初始化创建四个网络:

```
 def __init__(self, state_dim, hidden_size, action_dim, max_action, env:gym.Env, episodes, batch_size, buffer_size, lr=0.01, gamma=0.9, tau=0.02):
        self.env = env
        self.episodes = episodes
        self.replay_buffer = Replay_Buffer(buffer_size, batch_size)
        self.gamma = gamma
        self.state_dim = state_dim
        self.action_dim = action_dim
        self.actor = Actor(state_dim, hidden_size, action_dim, max_action)
        self.actor_target = Actor(state_dim, hidden_size, action_dim, max_action)
        self.actor_target.load_state_dict(self.actor.state_dict())
        self.critic = Critic(state_dim, hidden_size, action_dim)
        self.critic_target = Critic(state_dim, hidden_size, action_dim)
        self.critic_target.load_state_dict(self.critic.state_dict())
        self.optim_for_actor = torch.optim.Adam(self.actor.parameters(), lr=lr)
        self.optim_for_critic = torch.optim.Adam(self.critic.parameters(), lr=lr)
        self.tau = tau
```

我们可以定位到算法主体的网络混合训练。此外软更新函数实现也同论文一致，作为拆开两个演员和评论员网络的后处理手段。

```
def soft_update(self, network: nn.Module, target_network: nn.Module):
       for param, target_param in zip(network.parameters(), target_network.parameters()):
           target_param.data.copy_(self.tau*param.data + (1-self.tau)*target_param.data)


self.soft_update(self.actor, self.actor_target)
self.soft_update(self.critic, self.critic_target)
```

根据算法主体，不难预见到第一个选择到的动作由正态分布权重初始化全连接层的演员网络拟合的

对应的期望深度Q网络评估值为$Q(s_i,a_i|\theta_Q)$也即`q_pre`为评论员预测输出

```
q_pre = self.critic(train_s,train_a)
```

对应计算实际Q值公式有$\gamma Q'(s_{i+1},\mu'(s_{i+1}|\theta_\mu)|\theta_{Q'})$。注意这里使用的均是上标为$Q',\mu'$的目标网络副本，因此不难根据下一个状态`train_s_`得到结合深度奖励的实际Q值:

```
q_target = self.critic_target(train_s_,self.actor_target(train_s))
q_target = train_r + self.gamma*q_target*(1-train_done)
```

注意到异步策略复现中使用均方误差函数进行评论员网络训练。注意算法中是关于参数θ作随机(这里的参数θ其实就是指的网络权重参数`parameter`),因此使用`detach()`函数从计算图中撤销，不计算耦合的奖励目标网络的梯度部分，

```
loss_for_critic = torch.nn.functional.mse_loss(q_pre,q_target.detach())
```

算法主体中更新演员网络使用的是采样梯度，定义为关于演员网络参数的评论员期望Q值:

$$\nabla_{\theta^\mu}J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta_Q)|_{s=s_i,a=\mu(s_i)}\nabla_{\theta^\mu}\mu(s|\theta_\mu)|_{s_i} \tag{1}$$

注意到我们求的是最大值，因此必须取负号:

```
loss_for_actor = -(self.critic(train_s,self.actor(train_s))).mean()
```

3 实验代码

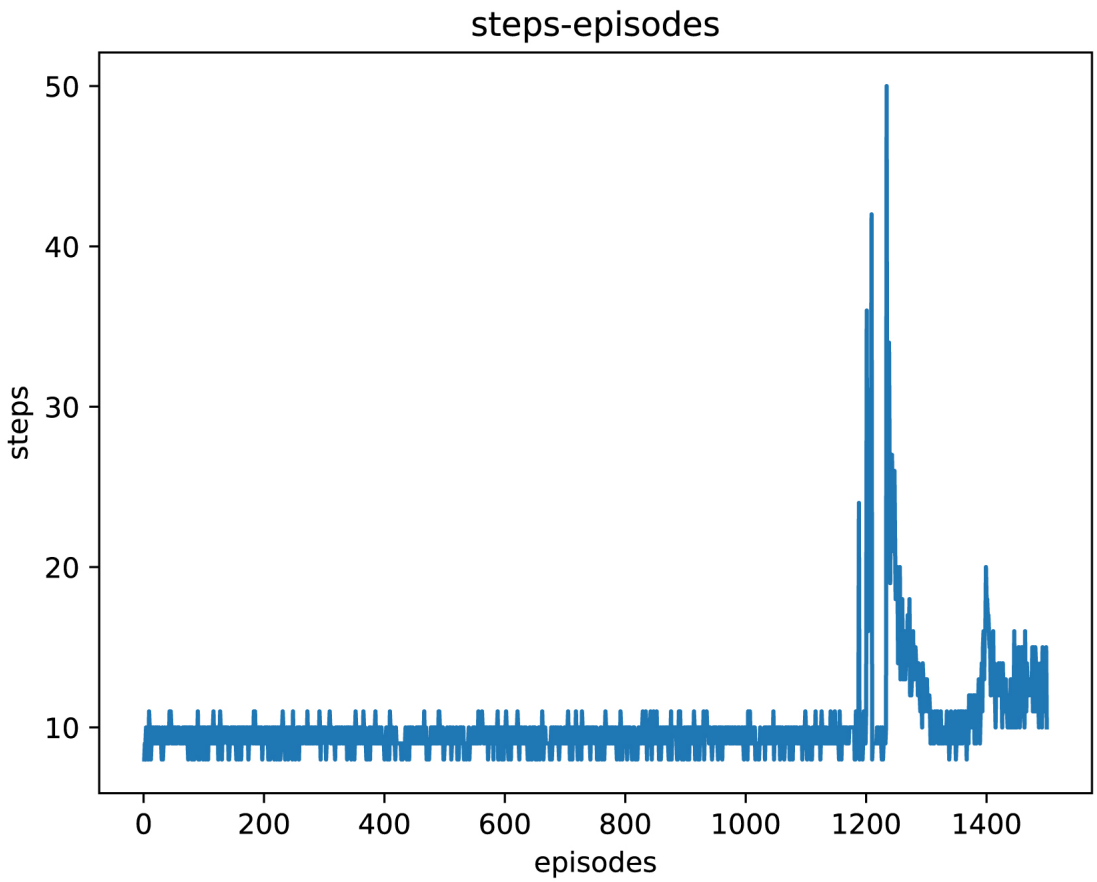
由上述分析可以整理得补充代码为:

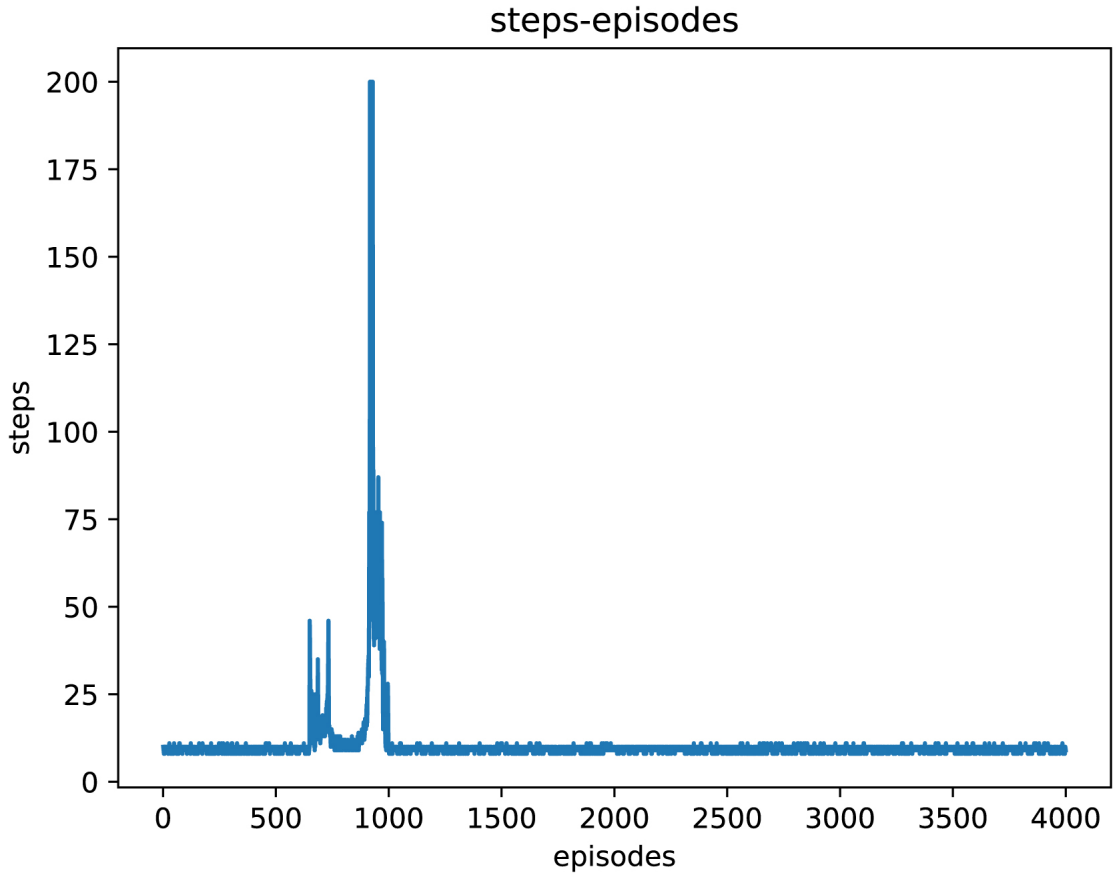
```
    def learn(self):
        counters = []
        for episode in range(self.episodes):
            counter = 0
            done = False
            observation = self.env.reset()
            while not done:
                counter += 1
                action = self.choose_action(observation)
                observation_, reward, done, _ = self.env.step(round(action))
                self.replay_buffer.push(deepcopy(observation), action, reward,
deepcopy(observation_), done)
                observation = observation_
                if not self.replay_buffer.is_full():
                    continue
                train_s, train_a, train_r, train_s_, train_done = self.replay_buffer.sample()
                train_s = torch.Tensor(train_s).view(-1, self.state_dim)
```

```
                train_a = torch.Tensor(train_a).view(-1, self.action_dim)
                train_r = torch.Tensor(train_r).view(-1, 1)
                train_s_ = torch.Tensor(train_s_).view(-1, self.state_dim)
                train_done = torch.Tensor(train_done).view(-1, 1)
                q_pre = self.critic(train_s,train_a)
                q_target = self.critic_target(train_s_,self.actor_target(train_s))
                q_target = train_r + self.gamma*q_target*(1-train_done)
                loss_for_critic = torch.nn.functional.mse_loss(q_pre,q_target.detach())
                loss_for_critic.backward()
                self.optim_for_critic.step()
                loss_for_actor = -(self.critic(train_s,self.actor(train_s))).mean()
                self.optim_for_actor.zero_grad()
                loss_for_actor.backward()
                self.optim_for_actor.step()
                self.soft_update(self.actor, self.actor_target)
                self.soft_update(self.critic, self.critic_target)
            counters.append(counter)
            print("episode {} end, stay for {} steps".format(episode+1, counter))
            if len(counters)>5 and counters[-5]==200 and counters[-4]==200 and
counters[-3]==200 and counters[-2]==200 and counters[-1]==200:
                return counters
        return counters
```

4 实验结果

由于CartPole游戏不是连续任务情景，因此没有数值优化加分析意义下的鲁棒性研究，理想的曲线应该是在动作边际上采取一系列的定向策略在多次试错的随机探索中开发出奖励的突变，因此下述图像表述的性能属性是针对`batch=1024`的采样相关较为理想的用法:


steps-episodes
steps
episodes
50
40
30
20
10
0
200
400
600
800
1000
1200
1400


steps-episodes
steps
episodes
200
175
150
125
100
75
50
25
0
0
500
1000
1500
2000
2500
3000
3500
4000