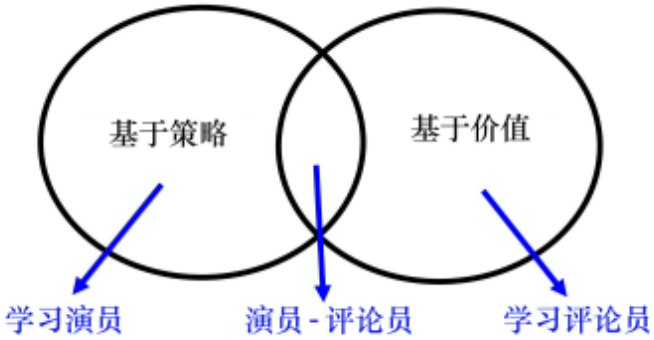
Actor-Critor同轨策略算法解决CartPole倒立摆问题

学号:2013365 姓名:颜铭

1 实验导言

在学习两种当今强化学习领域的主流策略梯度方法和深度Q网络并进行了基于强化学习游戏环境进行相关实验中，我们定性地可以通过可视化图表以及数值模型方法分析出两者的优缺点。深度Q网络较为稳定，如果没有近端策略优化在一些即时策略游戏中其实不应该引入关于非连续动作策略梯度的算法。但是由于策略梯度的数理模型更加完备而不是只停留于关于时序差分的离散近似的思想中，因此在物理控制中的优化和应用效果弱于策略梯度。那么对于倒立摆物理仿真游戏这一类强化学习环境，我们是否可以设置对照实验，引进策略梯度和深度Q网络结合的算法。事实上，演员-评论员算法就是这样一类被提出用来解决针对连续任务场景非Q学习网络的算法。



对于Actor-Critor算法，演员指的是策略函数$V_\theta(a|s)$，也即学习一个策略以得到尽可能高的回报。评论员指的是价值函数$V_\pi(s)$,对当前策略的价值函数进行估计，实现对演员好坏的评估。借助于引入价值函数，演员-评论员算法可以进行到单步参数更新，不需要等到回合结束就可以更新。本实验基于CartPole离散倒立摆环境使用的是同轨策略-的优势演员-评论员算法。其算法解读如下:

设随机变量G的期望值正好为Q值，也即有:

$$\mathbb{E}[G_t^n] = Q_{\pi_\theta}(s_t^n, a_t^n) \tag{1}$$

这对照着Q函数的定义，其实是在某一状态s采取某一个动作a。假设策略是π的情况下所得到的累积奖励的期望值也即G的期望值。因此可以假设$\mathbb{E}[G_t^n]$来代表$\sum_{t'=t}^{T}\gamma^{(t-t')}r_{t'}^n$。那么进一步可以根据式子(1)得到用$Q_{\pi_\theta}(s_t^n, a_t^n)$代表的$\sum_{t'=t}^{T_n}\gamma^{(t'-t)}r_{t'}^n$。有了上述分析我们就可以把演员与评论员两个方法结合起来。

对于基线的表示方法，常见的方法是使用价值函数$V_{\pi_\theta}(s_t^n)$来表示基线。对于价值函数定义为假设策略π,与其在某个状态s一直与环境交互直到游戏结束，期望奖励有多大。如果$V_{\pi_\theta}(s_t^n)$是$Q_{\pi_\theta}(s_t^n, a_t^n)$的期望值。$Q_{\pi_\theta}(s_t^n, a_t^n) - V_{\pi_\theta}(s_t^n)$都会有正负的零点变换。所以对于项$\sum_{t'=t}^{T_n}\gamma^{(t'-t)}r_{t'}^n - b$也同样保持正负一致。因此两者可以替换也就是实现了深度Q网络和策略梯度算法的结合。对于Q网络和V网络分开估计不准的学习鲁棒性改进，就可以使用优势的AC算法，只估计网络V,再利用网络V的值来表示网络Q。我们有:

$$Q_\pi(s_t^n, a_t^n) = \mathbb{E}[r_t^n + V_\pi(s_{t+1}^n)] \tag{2}$$

在状态s采取动作a，我们可以得到奖励r，同时进入状态s_{t+1}。如何克服或者利用这样的随机性，一般来说需要对奖励估计部分的V网络取期望值来等于Q函数的值，但在AC算法中把期望值去掉后就不需要估计Q了，只需要估计V，这和我们前提和先验知识相关的模型是一致的。由于随机变量r相较于累积的奖励G是一个较小的值，G是基于这一步的对于未来可能得到的奖励总和且方差较大。这样从概率分布加数值优化的角度上看把较大的G换成方差较小的r也

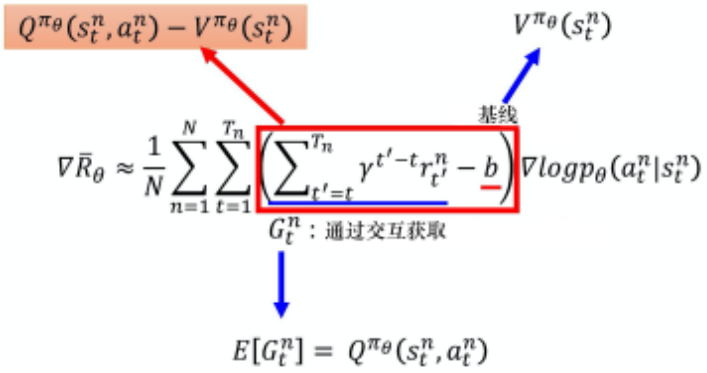
是较为合理的。

这样，我们有优势函数

$$r_t^n + V_\pi(s_{t+1}^n) - V_\pi(s_t^n) \tag{3}$$

以及使用梯度回归对基于奖励R的策略π的更新如下:

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(r_t^n + V_\pi(s_{t+1}^n) - V_\pi(s_t^n))\nabla \log p_\theta(a_n^t|s_t^n) \tag{4}$$



优势演员-动作网络算法

2 实验分析

关于代码补全，我们首要利用的信息就是算法主体外根据观察状态-动作对到反馈奖励和定义的演员网络和评论员网络。

其中演员网络主体为三个线性层，传播输出为一个基于多分类预测动作概率分布的条件分布函数。而评论员网络为选定一个奇异动作并输出估计的动作价值，这相当于Q网络迁移估计。

同时我们注意到对评论员的监督网络选择的动作倾向的系列动作值是实现优势函数$Q-V$的关键。这与`compute_return`函数是对应的。因为函数返回了前向差分的关于单步奖励的推广奖励(AC算法的即时预测同步)，也就是融合了基于Actor演员网络生成概率分布选择动作的单步奖励与评论员修正的奖励期望值。

因此在训练函数主体中我们可以看到

```
optimizerA = optim.Adam(actor.parameters())
    optimizerC = optim.Adam(critic.parameters())
    for iter in range(n_iters):
        state = env.reset()
        log_probs = []
        values = []
        rewards = []
        masks = []
        entropy = 0
        env.reset()

```

我们根据英文变量的规范命名可以得出我们确定的价值函数集合是由V网络也即评论员网络完成的工作，对数概率明显对应动作员网络生成的概率分布。从状态入手，我们需要基于这样概率分布采样的动作。而这特征是基于四元组观察状态在网络中学习得到的。掩码`masks`在上个函数中用到主要是评定参数θ关于策略π的选择是否有效(0-1)

entropy则是网络学习使用到的损失函数相关的熵可以作为状态平稳学习的一个系统性统计指标

对于网络学习损失函数的选择，类似于深度Q网络实现的Actor网络使用的融合了策略梯度对数概率的奖励期望均值，而V网络对于的评论员网络使用的是框架一致交换的，策略梯度学习和深度Q网络同样使用的标准均方误差损失函数。

3 实验代码补全

有了上述分析，着手对代码进行补充

第一步在未达到平衡终止条件前积累需要的数值量的交互循环中:

```
    env.render()
            # -------补充代码--------
            state = torch.FloatTensor(state).to(device)
            dist = actor(state) #概率分布
            val = critic(state) #评论员建议值

            action = dist.sample() #动作采样
            action = action.cpu().numpy() #注意数据类型的转换
            observation_, reward, done, info = env.step(action)  #交互
            action = torch.from_numpy(action).to(device)
            log_prob = dist.log_prob(action).unsqueeze(0) #对数概率分布
            entropy+=dist.entropy().mean()
            mask = 1-done

            reward = torch.FloatTensor([reward]).to(device)
            mask  = torch.FloatTensor([mask]).to(device)


            values.append(val) #累积估计价值
            rewards.append(reward) #累积奖励
            log_probs.append(log_prob)
            masks.append(mask)


            state = observation_ #下一个状态
```

特别注意的是向计算设备兼容的张量类型的转换以及向量大小一致上需要保持$n \times 1$中1维度的存在

进而根据上述分析，在主体的损失函数后向传播进行求导的过程中需要计算优势函数并计算基于累积奖励的损失函数:

```
# ------更新网络------
        state = torch.tensor(state, dtype=torch.float32, device=device)

        get_val = critic(state)
```

```
5
6         returns = compute_returns(get_val,rewards,masks)#Q网络计算值
7
8         log_probs = torch.cat(log_probs)
9         values = torch.cat(values)
10        returns = torch.cat(returns).detach()
11
12        optim_supervised = returns - values #优势函数计算
13
14        actor_loss  = -(log_probs*optim_supervised.detach()).mean() #演员网络损失函数
15
16        critic_loss = torch.pow(optim_supervised,2).mean()#评论员网络损失函数
17
18
19        optimizerA.zero_grad()
20        optimizerC.zero_grad()
21
22        actor_loss.backward()
23        critic_loss.backward()
24
25        optimizerA.step()
26        optimizerC.step()
27
```

4 实验结果

4.0.1 渲染可视化图

4.0.2 训练日志输出

```
Actor Model loaded
Critic Model loaded
Iteration: 0, Score: 150
Iteration: 1, Score: 55
Iteration: 2, Score: 118
Iteration: 3, Score: 131
Iteration: 4, Score: 148
Iteration: 5, Score: 113
Iteration: 6, Score: 189
Iteration: 7, Score: 114
Iteration: 8, Score: 198
Iteration: 9, Score: 77
Iteration: 10, Score: 216
Iteration: 11, Score: 465
Iteration: 12, Score: 123
Iteration: 13, Score: 220
Iteration: 14, Score: 205
Iteration: 15, Score: 138
Iteration: 16, Score: 421
Iteration: 17, Score: 168
Iteration: 18, Score: 147
Iteration: 19, Score: 222
Iteration: 20, Score: 387
Iteration: 21, Score: 227
Iteration: 22, Score: 467
Iteration: 23, Score: 366
Iteration: 24, Score: 303
Iteration: 25, Score: 129
Iteration: 26, Score: 411
Iteration: 27, Score: 223
```

Iteration: 28, Score: 168
Iteration: 29, Score: 181
Iteration: 30, Score: 312
Iteration: 31, Score: 502
Iteration: 32, Score: 173
Iteration: 33, Score: 159
Iteration: 34, Score: 186
Iteration: 35, Score: 183
Iteration: 36, Score: 272
Iteration: 37, Score: 281
Iteration: 38, Score: 52
Iteration: 39, Score: 158
Iteration: 40, Score: 716
Iteration: 41, Score: 140
Iteration: 42, Score: 208
Iteration: 43, Score: 373
Iteration: 44, Score: 517
Iteration: 45, Score: 174
Iteration: 46, Score: 219
Iteration: 47, Score: 218
Iteration: 48, Score: 208
Iteration: 49, Score: 174
Iteration: 50, Score: 474
Iteration: 51, Score: 361
Iteration: 52, Score: 927
Iteration: 53, Score: 497
Iteration: 54, Score: 1133
Iteration: 55, Score: 310
Iteration: 56, Score: 262
Iteration: 57, Score: 353
Iteration: 58, Score: 286
Iteration: 59, Score: 210
Iteration: 60, Score: 554
Iteration: 61, Score: 447
Iteration: 62, Score: 204
Iteration: 63, Score: 218
Iteration: 64, Score: 341
Iteration: 65, Score: 350
Iteration: 66, Score: 242
Iteration: 67, Score: 366
Iteration: 68, Score: 507
Iteration: 69, Score: 580
Iteration: 70, Score: 261
Iteration: 71, Score: 447
Iteration: 72, Score: 453
Iteration: 73, Score: 545
Iteration: 74, Score: 318
Iteration: 75, Score: 485

Iteration: 76, Score: 648
Iteration: 77, Score: 862
Iteration: 78, Score: 404
Iteration: 79, Score: 485
Iteration: 80, Score: 684
Iteration: 81, Score: 161
Iteration: 82, Score: 773
Iteration: 83, Score: 264
Iteration: 84, Score: 904
Iteration: 85, Score: 518
Iteration: 86, Score: 299
Iteration: 87, Score: 496
Iteration: 88, Score: 520
Iteration: 89, Score: 732
Iteration: 90, Score: 350
Iteration: 91, Score: 397
Iteration: 92, Score: 385
Iteration: 93, Score: 397
Iteration: 94, Score: 243
Iteration: 95, Score: 277
Iteration: 96, Score: 451
Iteration: 97, Score: 192
Iteration: 98, Score: 304
Iteration: 99, Score: 221</body>