


PPO × Family 第二讲习题

 本讲习题共包含两部分：分别是算法理论题和代码实践题。同学们可以选择其一项完成并提交。（当然也欢迎大家将两部分全部完成，这样能够加深对课程的理解）

- 算法理论题提交方式：

发送邮件至 opendilab@pjlab.org.cn

请同学们严格按照下方格式命名邮箱主题/标题：

【PPO × Family】+ 学生名 + vol.1 (第几节课) + 作业提交日期

示例：【PPO × Family】+ 喵小DI + vol.1 + 20221207

- 代码实践题提交方式：

PPO × Family 官方GitHub 上发起 Pull Request

- 地址： [PP0xFamily/chapter2_action/hw_submission](https://github.com/PP0xFamily/chapter2_action/hw_submission)
- PR示例： <https://github.com/opendilab/PP0xFamily/pull/5/files>
- 命名规范：

hw_submission(学生名称): add hw2 (第几节课) + 作业提交日期

示例：hw_submission(nyz): add hw2_20230104

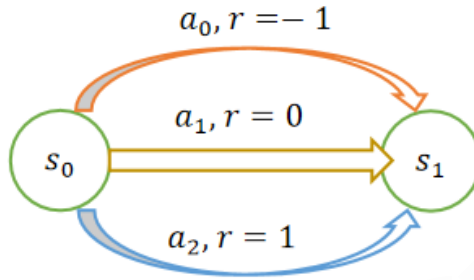
提交截止时间为 2023.1.17 23:59 (GMT +8)，逾期作业将不会计入证书考量。

如果其他问题请添加官方课程小助手微信 (vx: OpenDILab)，备注「课程」，小助手将邀请您加入官方课程微信交流群；或发送邮件至 opendilab@pjlab.org.cn

算法理论题

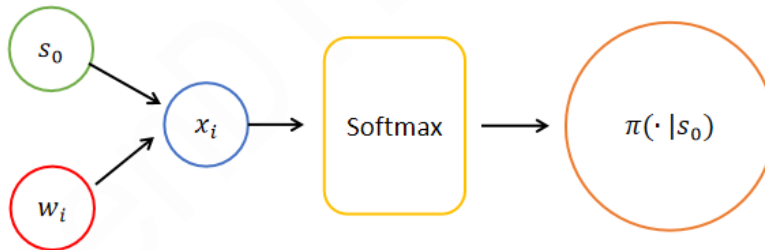
题目1（离散动作空间及动作掩码）

量化交易市场中，常常需要根据某些市场特征进行多种类型的决策，选择做多、做空或按兵不动（可参考[博客](#)）。我们可以简单抽象为如下的一个**单步**马尔可夫决策过程，每个时间步的 transition 为 $[s, a, s']$ ，如下图所示，初始状态为 s_0 ，在本问题的设定中是价格信息（一维）， s_1 为终态，三种决策动作 $[a_0, a_1, a_2]$ 对应的单次回报分别为 $[-1, 0, 1]$ 。



如下图所示，假设其策略函数 π 由一层全连接层构成，即简化为三个参数 $[w_0, w_1, w_2]$ ，状态信息和策略参数线性运算后得到激活值 $[x_0, x_1, x_2] = [w_0 s_0, w_1 s_0, w_2 s_0]$ ，激活值经过 Softmax 函数后得到选择三种决策动作的概率，即，

$$\pi(\cdot | s_0) = \text{Softmax}([x_0, x_1, x_2]) = \text{Softmax}([w_0 s_0, w_1 s_0, w_2 s_0])$$



1. 假设折扣因子 $\gamma = 1$ ，3条训练数据的轨迹分别为

$[[s_0 = 1, a_0, s_1], [s_0 = 1, a_1, s_1], [s_0 = 1, a_2, s_1]]$ ，参数初始值为 $[w_0, w_1, w_2] = [1, 1, 1]$ ，基于上述设定，请使用**策略梯度算法**求出参数 $[w_0, w_1, w_2]$ 在当前一次梯度计算过程中的梯度。

提示：

◦ 策略梯度定理的更新公式为：
$$\frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} G_t(\tau^n) \nabla \log \pi(a_t^n | s_t^n)$$

◦ Softmax 函数的表达式为：

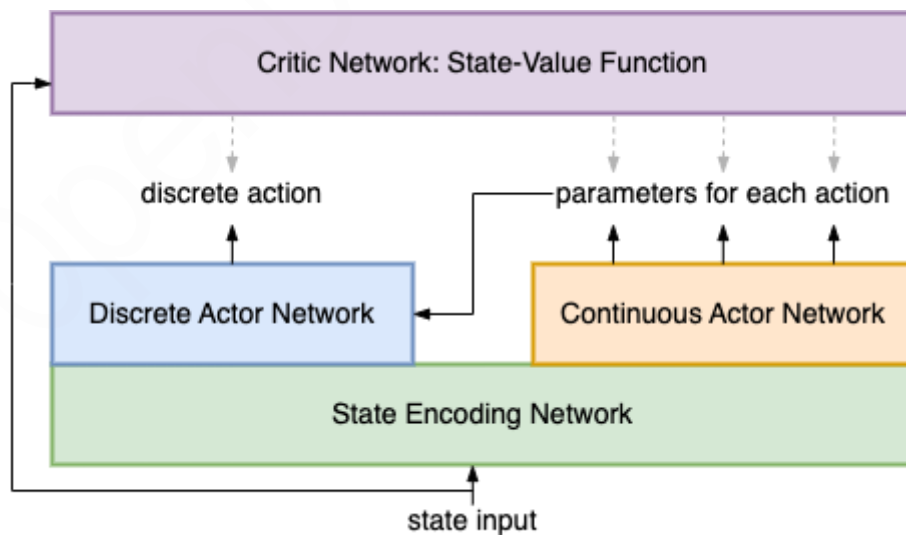
$$\text{Softmax}([w_0 s_0, w_1 s_0, w_2 s_0]) = \left[\frac{e^{w_0 s_0}}{\sum_j e^{w_j s_0}}, \frac{e^{w_1 s_0}}{\sum_j e^{w_j s_0}}, \frac{e^{w_2 s_0}}{\sum_j e^{w_j s_0}} \right]$$

2. 在实际的决策问题中，我们往往会面对的非常大的离散动作空间。但是对于特定的时间步，大部分的动作类型或选择是无效的。此时，我们会对策略函数施加掩码（mask），提前屏蔽一些无效动作，让这些动作输出的概率变为零（代码实现中，是将需要屏蔽的对应动作的激活值在进入 Softmax 之前减去一个无穷大的浮点数，使得最终该项动作输出概率接近于零）。
- 基于第一问，我们额外假定 a_1 是一个在 s_0 状态下不会产生实际意义的动作，并在策略中使用 mask 将其屏蔽掉。假如使用掩码后的策略和环境交互，收集到的数据为 $[[s_0 = 1, a_0, s_1], [s_0 = 1, a_2, s_1]]$ ，请计算此时使用策略梯度算法对应得到的参数 $[w_0, w_1, w_2]$ 在当前一次梯度计算过程中的梯度。

题目2（混合动作空间动作依赖）

大多数的深度强化学习算法只能处理单一类型动作的决策问题。而对于混合动作空间的马尔可夫决策问题，比如参数化动作空间，就需要特殊的算法设计。在参数化动作空间中，对于每一个状态 s_t ，都可以选择一个混合动作 $a(s_t) = (k(s_t), x_k(s_t))$ ，其中包含 k 维离散动作类型，以及离散动作类型对应的一个连续动作参数 x_k 。

混合动作空间 PPO（H-PPO）[1] 算法给出了一种处理混合动作决策问题的策略梯度算法的解决方案，即分别使用离散动作策略模型和连续动作策略模型，各自生成离散的动作类型与连续的动作参数，并将其输出组合为整体策略模型的输出。但是原生的 H-PPO 并没有考虑混合动作之间的依赖关系，这里我们参考在混合动作问题中性能表现较好的 Parameterized Deep Q-Learning（P-DQN）[2] 算法思路，设计了 H-PPO 算法的一种变种。如下图所示，将连续的动作参数的输出，作为离散动作类型模型的输入的一部分，从而让离散动作类型网络在输出类型时，也能考虑到连续动作参数的输出情况。



已知 H-PPO 算法的连续动作参数模型的目标函数：

$$L_{\text{Continuous}}(\theta_c) = \mathbb{E}_t[\min(r_t^c(\theta_c)\hat{A}_t, \text{clip}(r_t^c(\theta_c), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

H-PPO算法的离散动作类型模型的目标函数：

$$L_{\text{Discrete}}(\theta_d) = \mathbb{E}_t[\min(r_t^d(\theta_d)\hat{A}_t, \text{clip}(r_t^d(\theta_d), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

上述二式中，参数的 c 与 d 后缀分别指代连续(Continuous)与离散(Discrete)， r_t^c 与 r_t^d 分别为H-PPO算法连续动作参数和离散动作类型模型各自对应的重要性采样系数， θ_c 与 θ_d 分别为各自模型参数：

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

H-PPO算法的模型输出为：

$$x_k \sim \pi_{\theta_c}(x_k|s_t, k)$$

$$k_t \sim \pi_{\theta_d}(k_t|s_t)$$

H-PPO算法变种的模型输出为：

$$x_k \sim \pi_{\theta_c}(x_k|s_t, k)$$

$$k_t \sim \pi_{\theta_d}(k_t|s_t, x_1(\theta_c), \dots, x_K(\theta_c))$$

在这种修改下，离散动作类型网络的目标函数相比较于原版发生了改变，即：

$$L_{\text{Discrete}}(\theta_d, \theta_c) = \mathbb{E}_t[\min(r_t^d(\theta_d, \theta_c)\hat{A}_t, \text{clip}(r_t^d(\theta_d, \theta_c), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

对于上面所叙述的 H-PPO 变种算法，请计算并分析离散动作类型网络的目标函数的梯度更新公式 $\nabla_{\theta} L_{\text{Discrete}}(\theta_d, \theta_c)$ ，并分析可能存在什么问题。

提示：考虑连续参数对于该梯度的影响

代码实践题

题目1（重参数化）

考虑如下的优化问题：

$$\min \mathbb{E}_q[x^2], x \sim N(\mu, 1)$$

如果使用重参数化，则转化为：

$$\min \mathbb{E}_q[x^2], x = \epsilon + \mu, \epsilon \sim N(0, 1)$$

本题需要计算对比是否使用重参数化下，优化目标对 μ 的梯度的方差（即验证重参数化可减小方差）
请结合 [重参数化补充材料](#)，填补完成下方给出的代码实现，对比使用重参数化前后的梯度方差，并画出相关对比曲线。最终提交需上传完整代码和所得曲线图。

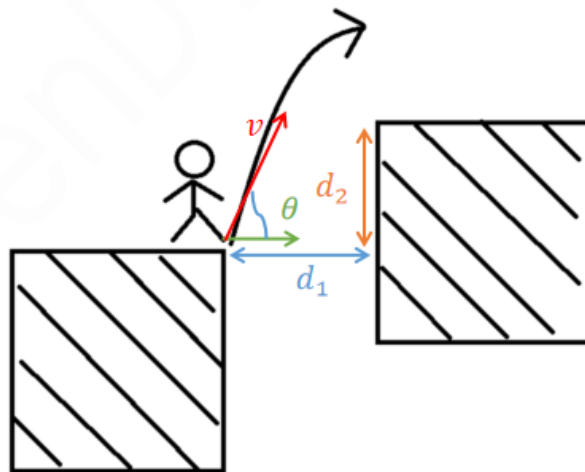
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 def naive_grad(x, mu):
6     # the naive gradient to mu
7     # \nabla_{\mu} \mathbb{E}_q[x^2] = \mathbb{E}_q[x^2(x-\mu)]
8     return np.mean(x ** 2 * (x - mu))
9
10
11 def reparam_grad(eps, mu):
12     ##### You need to finish the reparameterization gradient to mu here #####
13
14
15 def main():
16     data_size_list = [10, 100, 500, 1000, 5000]
17     sample_num = 100
18     mu, sigma = 2.0, 1.0
19
20     # variance of the gradient to mu
21     var1 = np.zeros(len(data_size_list))
22     var2 = np.zeros(len(data_size_list))
23
24     for i, data_size in enumerate(data_size_list):
25         estimation1 = np.zeros(sample_num)
26         estimation2 = np.zeros(sample_num)
27         for n in range(sample_num):
28             # 1. naive method
29             x = np.random.normal(mu, sigma, size=(data_size,))
30             estimation1[n] = naive_grad(x, mu)
```

```

31         # 2.reparameterization method
32         eps = np.random.normal(0.0, 1.0, size=(data_size, ))
33         x = eps * sigma + mu
34         estimation2[n] = reparam_grad(eps, mu)
35
36         var1[i] = np.var(estimation1)
37         var2[i] = np.var(estimation2)
38
39     print('naive grad variance: {}'.format(var1))
40     print('reparameterization grad variance: {}'.format(var2))
41     # plot figure
42     index = [_ for _ in range(len(data_size_list))]
43     plt.plot(index, var1)
44     plt.plot(index, var2)
45     plt.xticks(index, data_size_list)
46     plt.legend(['naive', 'reparam'])
47     plt.savefig('reparam.png')
48     plt.show()
49
50
51 if __name__ == "__main__":
52     main()

```

题目2（连续动作空间实践）



在一个重力场中，智能体小人需要跳跃翻过一个悬崖障碍

- 观测信息（state）为悬崖两壁距离 d_1 与悬崖两边的高度差 d_2 两维信息，且每次场景中的距离与高度差都不完全相同。
- 动作空间是跳跃该障碍所需要的跳跃角度 θ 和速度 v ，空中轨迹符合动力学定理。

- 完成任务将得到奖励 $r = 100 - v^2$ ，未完成将得到奖励 $r = 0$ 。即翻越障碍的速度越小，越节省能量，得分越高。
- $\theta \in [0, \frac{\pi}{2}]$ ， $v \in [0, 10]$ ， $d_1, d_2 \in [0, 1]$ ， N 指 batch size

策略函数可以参数化定义为

$$\begin{aligned}\pi(a|s) &= \mathcal{N}([\mu_\theta, \mu_v], [[\sigma_\mu^2, 0], [0, \sigma_v^2]]) \\ x &= \tanh(\text{linear}(d_1, d_2)), \quad x.\text{shape} = (N, 2) \\ \mu_\theta(d_1, d_2) &= \frac{1}{2}(x_0 + 1) \times \frac{\pi}{2} \\ \mu_v(d_1, d_2) &= \frac{1}{2}(x_1 + 1) \times 10 \\ \sigma_\theta(d_1, d_2) &= \exp(\text{param}_{\sigma_\theta}) \times \frac{\pi}{2} \\ \sigma_v(d_1, d_2) &= \exp(\text{param}_{\sigma_v}) \times 10\end{aligned}$$

上述的 $\tanh()$ ， $\exp()$ 与 $\text{linear}()$ 可借助 PyTorch 中的相关函数实现，其中 $\text{linear}()$ 函数拥有可学习的权重参数 w 与偏置参数 b ，其完整表达式为：

$$\text{linear}(x) := \sum_i w_i x_i + b$$

上述的 param_σ 为 PyTorch 定义参数对象，来自类 [torch.nn.Parameter](#)。

请使用 Python 代码实现上述策略函数，并计算当观测信息 $d_1 = 0.2$ ， $d_2 = 0.2$ 时，策略函数的参数在一次策略梯度算法更新时的梯度的数值大小。

提示：

- 策略梯度定理的更新公式为：
$$\frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{T_n-1} G_t(\tau^n) \nabla \log \pi(a_t^n | s_t^n)$$
- 可根据动力学公式判断智能体小人是否可以按照该速度和角度越过障碍

$$\begin{aligned}v_x &= v \cos(\theta) \\ v_y &= v \sin(\theta) \\ t &= \frac{d_1}{v_x} = \frac{d_1}{v \cos(\theta)} \\ \Delta y &= v_y t - \frac{1}{2} g t^2 = \frac{d_1 \sin(\theta)}{\cos(\theta)} - \frac{g d_1^2}{2 v^2 \cos^2(\theta)} \geq d_2\end{aligned}$$

- 需要计算梯度的参数为策略函数内定义的所有参数，具体包括 $\text{linear}()$ 函数的参数与 param_σ 参数。

题目3（应用实践）

在课程第二讲（解构复杂动作空间）几个应用中任选一个

- 火箭回收（离散动作空间）
- 无人机姿态控制（连续动作空间）
- 导航控制（混合动作空间）

根据课程组给出的[示例代码](#)，训练得到相应的智能体。最终提交需要上传相关训练代码、日志截图或最终所得的智能体效果视频（replay），具体样式可以参考第二讲的[示例 ISSUE](#)。

参考文献

- [1] Fan Z, Su R, Zhang W, et al. Hybrid actor-critic reinforcement learning in parameterized action space[J]. arXiv preprint arXiv:1903.01344, 2019.
- [2] Xiong J, Wang Q, Yang Z, et al. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space[J]. arXiv preprint arXiv:1810.06394, 2018.