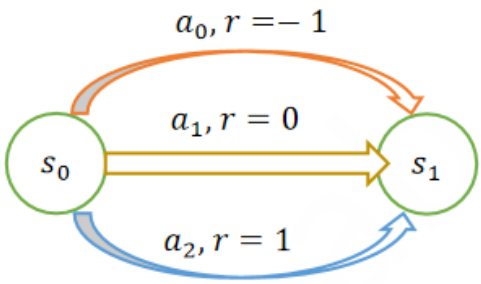
PPO × Family 第二讲习题作业答案

算法理论题

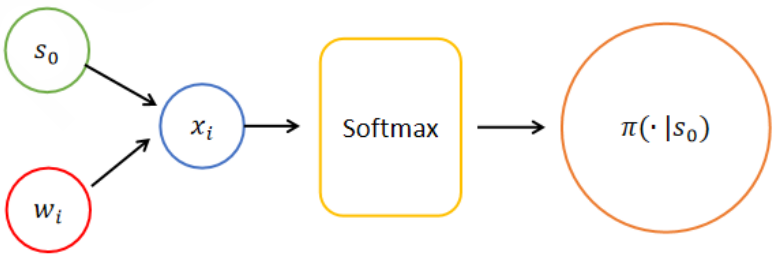
题目1（离散动作空间及动作掩码）

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，

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



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：$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1} G_t\left(\tau^n\right)\nabla \log \pi\left(a_t^n \mid s_t^n\right)$

- Softmax 函数的表达式为：

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

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。

题解：

1.

首先根据 MDP 的定义和累计回报的定义公式，计算得到：

$$G_0([s_0 = 1, a_0, s_1]) = -1$$

$$G_0([s_0 = 1, a_1, s_1]) = 0$$

$$G_0([s_0 = 1, a_2, s_1]) = 1$$

然后计算选择三种类型动作时，策略分布的对数的梯度：

$$\begin{aligned}
\nabla_w \log \pi(a_0|s_0) &= \frac{1}{\pi(a_0|s_0)} \times [\frac{\partial}{\partial w_0}\frac{e^{w0s0}}{\sum e^{wjs0}}, \frac{\partial}{\partial w_1}\frac{e^{w0s0}}{\sum e^{wjs0}}, \frac{\partial}{\partial w_2}\frac{e^{w0s0}}{\sum e^{wjs0}}] \\
&= \frac{\sum e^{wjs0}}{e^{w0s0}} \times [\frac{e^{w0s0} \times (e^{w1s0} + e^{w2s0}) \times s_0}{(\sum e^{wjs0})^2}, \frac{-e^{w0s0} \times e^{w1s0} \times s_0}{(\sum e^{wjs0})^2}, \frac{-e^{w0s0} \times e^{w2s0} \times s_0}{(\sum e^{wjs0})^2}] \\
&= [\frac{(e^{w1s0} + e^{w2s0}) \times s_0}{\sum e^{wjs0}}, \frac{-e^{w1s0} \times s_0}{\sum e^{wjs0}}, \frac{-e^{w2s0} \times s_0}{\sum e^{wjs0}}]
\end{aligned}$$

$$\nabla_w \log \pi(a_1|s_0) = [\frac{-e^{w0s0} \times s_0}{\sum e^{wjs0}}, \frac{(e^{w0s0} + e^{w2s0}) \times s_0}{\sum e^{wjs0}}, \frac{-e^{w2s0} \times s_0}{\sum e^{wjs0}}]$$

$$\nabla_w \log \pi(a_2|s_0) = [\frac{-e^{w0s0} \times s_0}{\sum e^{wjs0}}, \frac{-e^{w1s0} \times s_0}{\sum e^{wjs0}}, \frac{(e^{w0s0} + e^{w1s0}) \times s_0}{\sum e^{wjs0}}]$$

将它们带入策略梯度定理的定义，并对所有数据代入数值进行计算，从而得到：

$$\begin{aligned}
\nabla_w J &= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1}[\gamma^t G_t(\tau^n) \nabla \log \pi(a_t^n \mid s_t^n)] \\
&= \frac{1}{3} \times \{G_0([s_0, a_0, s_1]) \nabla \log \pi(a_0 \mid s_0) + G_1([s_0, a_1, s_1]) \nabla \log \pi(a_1 \mid s_0) + G_2([s_0, a_2, s_1]) \nabla \log \pi(a_2 \mid s_0)\} \\
&= \frac{1}{3} \times \{-1 \times [\frac{e^{w1s0} + e^{w2s0}}{\sum e^{wjs0}}, \frac{-e^{w1s0}}{\sum e^{wjs0}}, \frac{-e^{w2s0}}{\sum e^{wjs0}}] + 0 \times [\frac{-e^{w0s0}}{\sum e^{wjs0}}, \frac{e^{w0s0} + e^{w2s0}}{\sum e^{wjs0}}, \frac{-e^{w2s0}}{\sum e^{wjs0}}] + 1 \times [\frac{-e^{w0s0}}{\sum e^{wjs0}}, \frac{-e^{w1s0}}{\sum e^{wjs0}}, \frac{e^{w0s0} + e^{w1s0}}{\sum e^{wjs0}}]\} \\
&= \frac{1}{3} \times \{-1 \times [\frac{2}{3}, \frac{-1}{3}, \frac{-1}{3}] + 0 \times [\frac{-1}{3}, \frac{2}{3}, \frac{-1}{3}] + 1 \times [\frac{-1}{3}, \frac{-1}{3}, \frac{2}{3}]\} \\
&= [\frac{-1}{3}, 0, \frac{1}{3}]
\end{aligned}$$

2.

由于添加动作掩码（action mask）之后，输出值变为：

$$\mathrm{ActionMask}([x_0, x_1, x_2]) = ([w_0\, s_0, w_1\, s_0 - \inf, w_2\, s_0])$$

这会使得 a_1 动作不再被“激活”，故重新计算梯度可得：

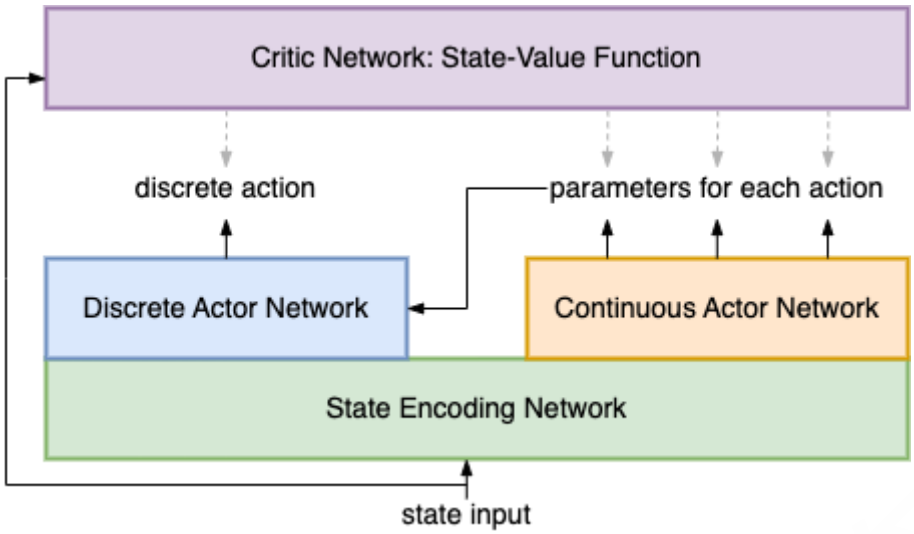
$$\begin{aligned}
\nabla_w J_{\mathrm{ActionMask}} &= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1}\left[\gamma^t G_t\left(\tau^n\right)\nabla\log\pi\left(a_t^n \mid s_t^n\right)\right] \\
&= \frac{1}{2}\times\left\{G_0\left([s_0, a_0, s_1]\right)\nabla\log\pi\left(a_0 \mid s_0\right) + G_2\left([s_0, a_2, s_1]\right)\nabla\log\pi\left(a_2 \mid s_0\right)\right\} \\
&= \frac{1}{2}\times\left\{-1\times\left[\frac{e^{w1\,s0}+e^{w2\,s0}}{\sum e^{wj\,s0}}, \frac{-e^{w1\,s0}}{\sum e^{wj\,s0}}, \frac{-e^{w2\,s0}}{\sum e^{wj\,s0}}\right] + 1\times\left[\frac{-e^{w0\,s0}}{\sum e^{wj\,s0}}, \frac{-e^{w1\,s0}}{\sum e^{wj\,s0}}, \frac{e^{w0\,s0}+e^{w1\,s0}}{\sum e^{wj\,s0}}\right]\right\} \\
&= \frac{1}{2}\times\left\{-1\times\left[\frac{1}{2}, 0, \frac{-1}{2}\right] + 1\times\left[\frac{-1}{2}, 0, \frac{1}{2}\right]\right\} \\
&= \left[\frac{-1}{2}, 0, \frac{1}{2}\right]
\end{aligned}$$

对比第一问和第二问的梯度计算结果，我们可以发现使用动作掩码后，梯度更新会更为针对和集中至未被 mask 的部分，被 mask 的部分则不会受到本次更新的影响。实际使用中，动作掩码不仅能保证策略网络始终输出合法的动作，并且还有一定程度上加速训练收敛的作用。

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：

$$a_t = (k_t, [x_1, x_2, ..., x_K])$$
$$[x_1, \cdots, x_K] \sim \pi_{\theta_c}(x_k|s_t)$$
$$k_t \sim \pi_{\theta_d}(k_t|s_t)$$

H-PPO算法变种的模型输出为：

$$a_t = (k_t, [x_1, x_2, ..., x_K])$$
$$[x_1, \cdots, x_K] \sim \pi_{\theta_c}(x_k|s_t)$$
$$k_t \sim \pi_{\theta_d}(k_t|s_t, x_1(\theta_c), ..., x_K(\theta_c))$$

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

题解：

由于使用 H-PPO 的变种算法，离散动作类型网络引入了来自连续动作参数网络的输出 $[x_1, \cdots, x_K]$ 作为输入，因此离散动作的目标函数需要分别计算其对两部分参数 (θ_d, θ_c) 的梯度，有：

$$\begin{aligned}\nabla_{\theta_d} L_{\text{Discrete}}(\theta_d, \theta_c) &= \mathbb{E}_t[\nabla_\theta \min(r_t^d(\theta_d, \theta_c)\hat{A}_t, \text{clip}(r_t^d(\theta_d, \theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ &= \mathbb{E}_t[\frac{\partial \min((\cdot), \text{clip}(\cdot))}{\partial(\cdot)} \times \frac{\partial r_t^d}{\partial \pi_{\text{Discrete}}} \times \nabla_{\theta_d} \pi_{\text{Discrete}}]\end{aligned}$$

$$\begin{aligned}\nabla_{\theta_c} L_{\text{Discrete}}(\theta_d, \theta_c) &= \mathbb{E}_t[\nabla_\theta \min(r_t^d(\theta_d, \theta_c)\hat{A}_t, \text{clip}(r_t^d(\theta_d, \theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ &= \mathbb{E}_t[\frac{\partial \min((\cdot), \text{clip}(\cdot))}{\partial(\cdot)} \times \frac{\partial r_t^d}{\partial \pi_{\text{Discrete}}} \times (\frac{\partial \pi_{\text{Discrete}}}{\partial x_1}\nabla_{\theta_c} x_1 + \ldots + \frac{\partial \pi_{\text{Discrete}}}{\partial x_K}\nabla_{\theta_c} x_K)]\end{aligned}$$

从上式可见，在 L_{Discrete} 相对于参数 θ_c 的梯度更新公式中，梯度信息会回传给所有 K 个离散动作的连续动作参数。

但是，理想情况下，每个离散类型的动作数据，应该**仅对自身对应的连续参数 x_k 回传梯度**。即每个连续动作参数对应的网络参数，应该只受到相对应的离散动作类型的梯度影响，其他离散动作是好是坏对它应该不存在影响。

即对于数据 $a_k = (k, x_k)$，将有：

$$\begin{aligned}\nabla_{\theta_c} L_{\text{Discrete}}(\theta_d, \theta_c) &= \mathbb{E}_t[\nabla_\theta \min(r_t^d(\theta_d, \theta_c)\hat{A}_t, \text{clip}(r_t^d(\theta_d, \theta_c), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ &= \mathbb{E}_t[\frac{\partial \min((\cdot), \text{clip}(\cdot))}{\partial(\cdot)} \times \frac{\partial r_t^d}{\partial \pi_{\text{Discrete}}} \times \frac{\partial \pi_{\text{Discrete}}}{\partial x_k}\nabla_{\theta_c} x_k]\end{aligned}$$

从上下两个公式的区别中，我们可以看出这种变种算法在梯度反向传播更新参数的时候，可能存在冗余性和干扰，当前离散动作更新时会影响到不相关的连续参数对应的神经网络。另外，从前向计算图角度来看，某个离散动作的输出不仅仅受到跟自己相关的连续的参数的影响，还受到其他连续参数的干扰。

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。

题解：

不使用重参数化时，梯度为：

$$\begin{aligned}\nabla_\mu\mathcal{L}(\mu)&=\nabla_\mu\int x^2q(x)\mathrm{d}x=\int(\nabla_\mu\ x^2)q(x)\mathrm{d}x+\int x^2(\nabla_\mu q(x))\mathrm{d}x\\&=0+\int x^2(\nabla_\mu\frac{\exp\left(-\frac{1}{2}(x-\mu)^2\right)}{\sqrt{2\pi\times1}})\mathrm{d}x\\&=\int x^2(x-\mu)(\frac{\exp\left(-\frac{1}{2}(x-\mu)^2\right)}{\sqrt{2\pi\times1}})\mathrm{d}x\\&=\int x^2(x-\mu)q(x)\mathrm{d}x=\mathbb{E}_q(x^2(x-\mu))\end{aligned}$$

使用重参数化时，梯度为：

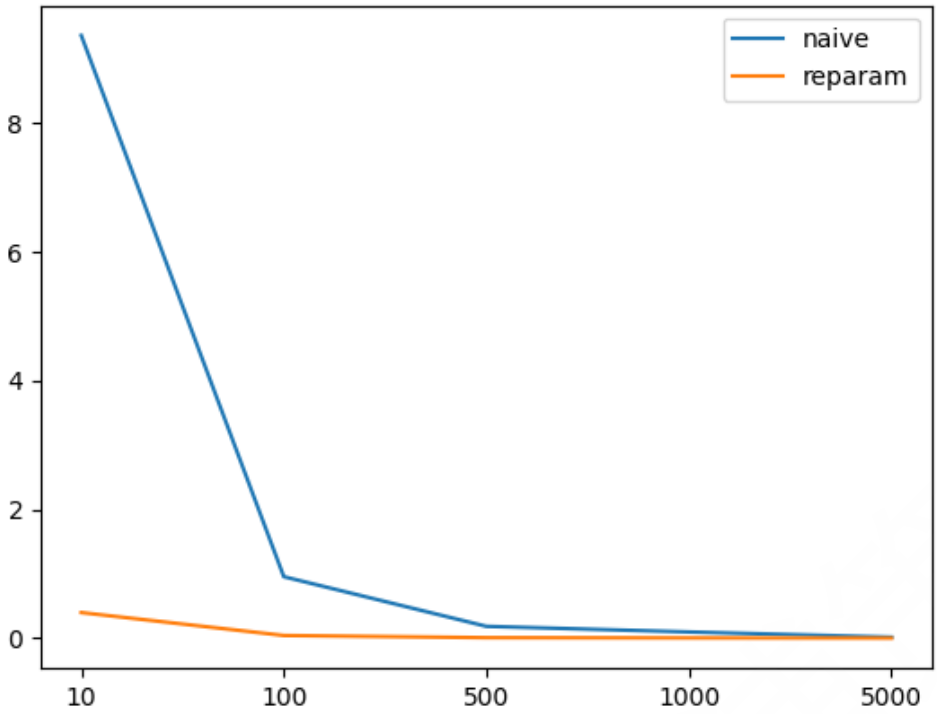
$$\nabla_\mu\mathcal{L}(\mu)=\nabla_\mu\int x^2q(x)\mathrm{d}x=\nabla_\mu\int x^2q(\epsilon)\mathrm{d}\epsilon=\int 2(\epsilon+\mu)q(\epsilon)\mathrm{d}\epsilon=\mathbb{E}_{q_\epsilon}(2(\epsilon+\mu))$$

故代码实现如下（即在11行补充相应实现）：

```python
import numpy as np
import matplotlib.pyplot as plt

def naive_grad(x, mu):
    # the naive gradient to mu
    # \nabla_\mu \mathbb{E}_q[x^2] = \mathbb{E}_q[x^2(x-\mu)]
    return np.mean(x ** 2 * (x - mu))

def reparam_grad(eps, mu):
    #### You need to finish the reparameterization gradient to mu here ####
```

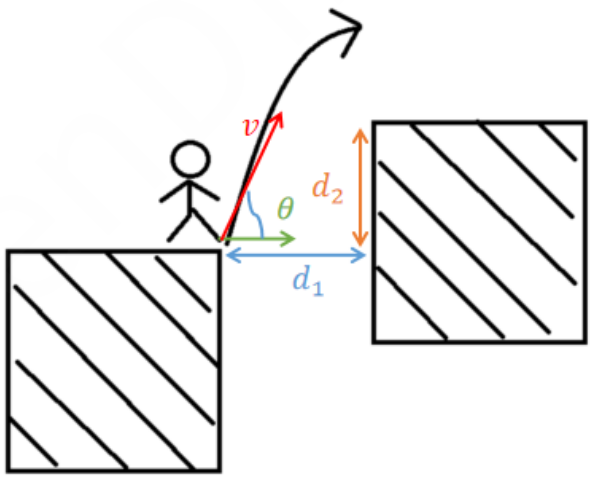
```python
    return np.mean(2 * (eps + mu))

def main():
    data_size_list = [10, 100, 500, 1000, 5000]
    sample_num = 100
    mu, sigma = 2.0, 1.0

    # variance of the gradient to mu
    var1 = np.zeros(len(data_size_list))
    var2 = np.zeros(len(data_size_list))

    for i, data_size in enumerate(data_size_list):
        estimation1 = np.zeros(sample_num)
        estimation2 = np.zeros(sample_num)
        for n in range(sample_num):
            # 1.naive method
            x = np.random.normal(mu, sigma, size=(data_size, ))
            estimation1[n] = naive_grad(x, mu)
            # 2.reparameterization method
            eps = np.random.normal(0.0, 1.0, size=(data_size, ))
            x = eps * sigma + mu
            estimation2[n] = reparam_grad(eps, mu)

        var1[i] = np.var(estimation1)
        var2[i] = np.var(estimation2)

    print('naive grad variance: {}'.format(var1))
    print('reparameterization grad variance: {}'.format(var2))
    # plot figure
    index = [_ for _ in range(len(data_size_list))]
    plt.plot(index, var1)
    plt.plot(index, var2)
    plt.xticks(index, data_size_list)
    plt.legend(['naive', 'reparam'])
    plt.savefig('reparam.png')
    plt.show()

if __name__ == "__main__":
    main()
```

最终运行该代码得到的曲线图为：



从图中可以看出，重参数化方法相比原始方法能够显著地减小梯度的方差。不过当样本数量（data size）足够多时，梯度方差本身也就变得足够小了。

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

$$\pi(a|s) = \mathcal{N}([\mu_\theta, \mu_v], [[\sigma_\mu^2, 0], [0, \sigma_v^2]])$$

$$x = \tanh(\text{linear}(d_1, d_2)),\ x.shape = (N,\ 2)$$

$$\mu_\theta(d_1, d_2) = \frac{1}{2}(x_0 + 1) \times \frac{\pi}{2}$$

$$\mu_v(d_1, d_2) = \frac{1}{2}(x_1 + 1) \times 10$$

$$\sigma_\theta(d_1, d_2) = \exp(\text{param}_{\sigma_\theta}) \times \frac{\pi}{2}$$

$$\sigma_v(d_1, d_2) = \exp(\text{param}_{\sigma_v}) \times 10$$

上述的 tanh( )，exp( ) 与 linear( ) 可借助 PyTorch 中的相关函数实现，其中 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定义的参数对象，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：$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T_n-1} G_t\left(\tau^n\right)\nabla\log\pi\left(a_t^n \mid s_t^n\right)$
- 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

$$v_x = v\cos(\theta)$$

$$v_y = v\sin(\theta)$$

$$t = \frac{d_1}{v_x} = \frac{d_1}{v\cos(\theta)}$$

$$\Delta\ y = v_y\ t - \frac{1}{2}gt^2 = \frac{d_1\sin(\theta)}{\cos(\theta)} - \frac{gd_1^2}{2v^2\cos^2(\theta)} \geq d_2$$

- 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 linear( ) 函数的参数与 param_σ 参数

题解：

完整的代码实现如下，策略梯度部分可以参考课程第一讲和第二讲提供的代码样例（即课程文件夹下的py文件）

```
import torch
from torch import nn
from torch.distributions import Normal, Independent
```

```
4
5  def is_success(theta, v, d1, d2):
6      # if the following conditions hold True, it is a success action.
   Otherwise, it would fail.
7      return d1 * torch.tan(theta) - 9.8 * d1 * d1 / (2.0 * v * v *
   torch.cos(theta) * torch.cos(theta)) - d2 >= 0
8
9  def compute_reward(v):
10     return 100.0 - v ** 2
11
12 class Policy(nn.Module):
13
14     def __init__(self):
15         super().__init__()
16         self.tanh = nn.Tanh()
17         self.n1 = nn.Linear(2, 2)
18         self.log_sigma = nn.Parameter(torch.ones(2))
19
20     def forward(self, d1, d2):
21         d = torch.concat([d1, d2])
22         mu = (self.tanh(self.n1(d)) + 1.0) / 2.0 * torch.tensor([torch.pi /
   2.0, 10.0])
23         sigma = torch.exp(self.log_sigma) * torch.tensor([torch.pi / 2.0,
   10.0])
24         return mu, sigma
25
26     def compute_objective(self, d1, d2):
27         mu, sigma = self.forward(d1, d2)
28         p = Independent(Normal(loc=mu, scale=sigma),
   reinterpreted_batch_ndims=1)
29         x = p.sample()
30         with torch.no_grad():
31             if is_success(x[0], x[1], d1, d2):
32                 r = compute_reward(x[1])
33             else:
34                 r = 0.0
35         obj = r * p.log_prob(x)
36         return obj
37
38 def main():
39     policy = Policy()
40     d1 = torch.FloatTensor([0.2])
41     d2 = torch.FloatTensor([0.2])
42
43     obj = policy.compute_objective(d1, d2)
44     obj.backward()
45     print(policy.n1.weight.grad, policy.n1.bias.grad, policy.log_sigma.grad)
```

```
46
47 if __name__ == "__main__":
48     main()
```

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.