贝叶斯神经网络

请注意区分贝叶斯网络和贝叶斯神经网络两个概念。贝叶斯神经网络指的是网络的权重为随机变量的神经网络。对于随机变量，我们往往关注随机变量的数学期望。因此我们先回顾数学期望的定义。

数学期望

假设现在有一个随机变量$x \sim p_x(x)$，其中p表示“概率分布”，p_x表示x的概率分布。

如果x是连续变量，则它的数学期望是：

$$\mathbb{E}_x\{x\} = \int x \cdot p_x(x)\ \mathrm{d}x$$

如果x是离散变量，则它的数学期望是：

$$\mathbb{E}_x\{x\} = \sum_{X_i \in X} X_i \cdot p_x(X_i)$$

其中X是所有的可能的值，X_i是其中的某一个值。

如果x只是一个自变量，那么因变量$f(x)$的数学期望是：

$$\mathbb{E}_x\{f(x)\} = \int f(x) \cdot p_x(x)\ \mathrm{d}x$$

或者

$$\mathbb{E}_x\{f(x)\} = \sum_{X_i \in X} f(X_i) \cdot p_x(X_i)$$

对于数学期望的近似估计

概率的学派

概率主要分为经典学派和贝叶斯学派，有的地方又叫经典学派为频率学派。两者的区别在于是否通过重复实验来得到概率。比如：

频率学派:

- 骰子每个面朝上的概率为1/6，因为我们可以扔60000次骰子，其中每一面朝上的次数大概都是10000次。
- 硬币每个面朝上的概率为1/2，因为我们可以扔20000次骰子，其中每一面朝上的次数大概都是10000次。

这其实很好理解， 也是最直观的概率/频率。但是有的时候，很多所谓的概率是无法用实验或频率证实的，比如：

- 下一班飞机晚点的概率是90%
- 火星上有生命的概率是2%
- 太阳明天从东方升起的概率是100%
- 每天喝酒会增加20%的癌症几率

类似的表达也是生活中常见的概率，尽管他们无法用重复的实验/频率来验证，但这些概率仍然有重要而且实际的意义，这些概率就是**贝叶斯概率/贝叶斯学派**。

所以，生活中有些人常说的：“有些事情，发生了就是100%，不发生就是0%”，这就是典型的频率学派的一家之词。

在生活中这两种概率都经常出现，而且都有重要的应用以及成功的案例。因此不应当有高低之分，但是盲目否认另外一种就是片面的。

当我们用贝叶斯概率来看待问题，只要一组数符合概率公理，也就是：非负性，归一化，可加性，它们就可以**被当作概率来看待**，这样的话，就可以运用许多公式来对它们进行处理、推导、换算等等。尽管在换算之前或换算的过程中他们失去了“频率”的意义，他们仍然具有正确的含义。

用频率近似概率（大数定理）

在估计数学期望时，我们常用的方法就是用频率代替概率，也就是说所谓的大数定理，我们进行大量（N次）采样$(x_1, x_2, \ldots, x_N)$，然后观察x_n的值，并且通过统计的方法近似估计它们的概率$p_x(X_i) \approx \frac{N_i}{N}$，其中$N_i$就是$N$个采样$x_n$，$n = 1, \ldots, N$中有几个$x_n = X_i$

> 举例，对于扔10次硬币来说, $X = \{0, 1\}$，其中$X_1 = 0$, $X_2 = 1$分别表示反面和正面。一共有10个采样（也就是10次实验）$x_1, \ldots, x_{10}$，每一个采样x_n都是一次投掷的结果，比如$x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$, $x_5 = 1$, $x_6 = 1$, $x_7 = 0$, $x_8 = 0$, $x_9 = 1$, $x_{10} = 0$。那么N_1就是有几个x_n等于X_1，所以$N_1 = 6$，类似，$N_2 = 4$。所以估计出来的$p_x(X_1) \approx N_1/N = 0.6$，以及$p_x(X_2) \approx N_2/N = 0.4$。

因此，离散变量的数学期望就可以被表示为

$$\begin{aligned}\mathbb{E}_x\{f(x)\} &= \sum_{X_i \in X} f(X_i) \cdot p_x(X_i) \\ &\approx \frac{1}{N} \sum_{X_i \in X} f(X_i) \cdot N_i\end{aligned}$$

我们再把N_i表示成$x_n = X_i$的个数，也就是$\sum_{x_n = X_i} 1$（表示把所有x_n中等于X_i的采样挑出来）。对于被挑出来的这些x_n，显然$X_i = x_n$，那么$f(X_i)$就可以写成$f(x_n)$，所以上式就成了

$$\begin{aligned}\mathbb{E}_x\{f(x)\} &\approx \frac{1}{N} \sum_{X_i \in X} \sum_{x_n = X_i} f(x_n) \\ &= \frac{1}{N} \left(\sum_{x_n = X_1} f(x_n) + \sum_{x_n = X_2} f(x_n) + \sum_{x_n = X_3} f(x_n) + \cdots \right)\end{aligned}$$

这里把$x_n = X_1$, $x_n = X_2$, $x_n = X_3$,...全都加起来，相当于既不重复也不遗漏的把每一个x_n，$n = 1, \ldots, N$加了一遍，也就是

$$\mathbb{E}_x\{f(x)\} \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n)$$

其中采样x_i时，它隐含了x的概率分布$p_x(\cdot)$。相当于$x = X_i$的概率大小体现在了N_i上面。因此我们需要把采样的概率分布也标记上：

$$\mathbb{E}_x\{f(x)\} \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n), \quad x_n \sim p_x(x_n)$$

可以看出，数学期望的数学形式其实就是对采样值求平均值。

连续变量

大数定理同样适用于连续变量，也就是说

$$\mathbb{E}_x\{f(x)\} \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n), \quad x_n \sim p_x(x_n)$$

蒙特卡洛积分

通过大数定理在连续变量上的应用，我们可以推导出一个有用的工具，也就是蒙特卡洛积分。让我们观察下面这个公式的左右两端：

$$\int f(x) \cdot p_x(x)\ \mathrm{d}x = \mathbb{E}_x\{f(x)\} \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n), \quad x_n \sim p_x(x_n)$$

可以得到下面这个公式

$$\int f(x) \cdot p_x(x)\ \mathrm{d}x \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n), \quad x_n \sim p_x(x_n)$$

那么，如果抛开公式含义不谈，我们可以得到下面这个公式：

$$\int f(x)\ \mathrm{d}x = \int f(x) \cdot \frac{p_x(x)}{p_x(x)}\ \mathrm{d}x = \int \frac{f(x)}{p_x(x)} \cdot p_x(x)\ \mathrm{d}x$$

等式的最右端一项等价于$\mathbb{E}_x\left\{\frac{f(x)}{p_x(x)}\right\}$，所以可以借助于大数定理：

$$\int f(x)\ \mathrm{d}x \approx \frac{1}{N} \sum_{n=1}^{N} \frac{f(x)}{p_x(x)}, \quad x_n \sim p_x(x_n)$$

等式左边现在是任意的被积分的函数，同时右边的$p_x(x)$也是我们可以随意定义的分布了，对它的要求是要满足概率公理以及和$f(x)$具有同样的定义域。

> 举例：如果我们要计算$\sin(x)$在$[0, 2]$上的积分，我们可以用解析解求出：
>
> $$\int_0^2 \sin(x)\mathrm{d}x = -\cos(x)\big|_0^2 = 1 - \cos(2) \approx 1.41614$$
>
> 我们可以用蒙特卡洛积分来计算：我们选取$p_x(x)$是$[0, 2]$之间的均匀分布，所以$p_x(x) = 0.5$，$x \in [0, 2]$。下面是代码：

In [1]:

```
import torch
torch.manual_seed(0)
N = 10
x = torch.rand(N) * 2
y = 1 - torch.cos(torch.tensor(2))
yhat = torch.sin(x).sum().item() * 2 / N
square_error = (y - yhat) ** 2
print(f'real value is {y:.5f}, estimation from MC is {yhat:.5f}, squared error is {square_error:.5f}.')
```

```
real value is 1.41615, estimation from MC is 1.47101, squared error is 0.00301.
```

这种估计方法类似于矩形近似法，我们来看一个矩形近似法(Riemann Sum)的代码：

In [2]:

```
x = torch.linspace(0,2,N)
yhat = torch.sin(x).sum().item() * 2 / N
square_error = (y - yhat) ** 2
print(f'real value is {y:.5f}, estimation from RS is {yhat:.5f}, squared error is {square_error:.5f}.')
```

```
real value is 1.41615, estimation from RS is 1.36021, squared error is 0.00313.
```

> 在这个例子中，我们看似矩形近似法相差不大。其实这是因为函数过于平滑。当函数变化剧烈时，结果就不同了。我们假设要计算$\sin(kx)$在$[0, 2]$上的积分，我们可以看出，当k增大时，函数变得快速波动，我们测试k从1到10000:
> 真实值为：
>
> $$\int_0^2 \sin(kx)\mathrm{d}x = -\frac{1}{k}\cos(kx)\big|_0^2 = \frac{1}{k}(1 - \cos(2k))$$

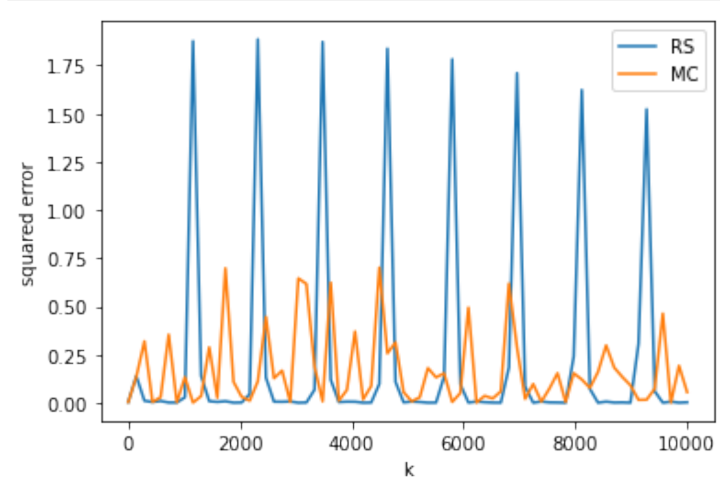
In [3]:

```
import torch
torch.manual_seed(0)
N = 10
x_MC = torch.rand(N) * 2
x_RS  = torch.linspace(0,2,N)
x_real  = torch.linspace(0,2,N*100)

error_MC = []
error_RS = []
K = torch.linspace(1,10000,70)
for k in K:
    real = 1/k*(1-torch.cos(torch.tensor(2)*k))
    MC = torch.sin(k*x_MC).sum().item() * 2 / N
    RS = torch.sin(k*x_RS).sum().item() * 2 / N
    error_MC.append((MC-real)**2)
    error_RS.append((RS-real)**2)
```

In [4]:

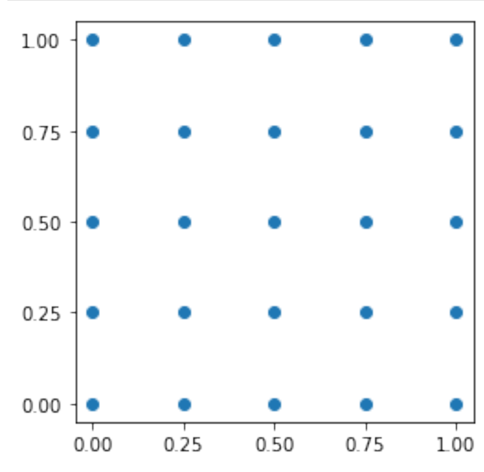
```
import matplotlib.pyplot as plt
plt.plot(K.numpy(), error_RS, label='RS')
plt.plot(K.numpy(), error_MC, label='MC')
plt.xlabel('k')
plt.ylabel('squared error')
plt.legend();
```



可以看出，相比于矩形近似，蒙特卡洛积分往往可以避免极端情况下出现的巨大误差。这是因为有很多关键的位置被跳过了。这在高维度的积分时更加明显。比如说，对于二维函数进行积分时，采用矩形积分，会有以下的采样点：

In [5]:

```
torch.manual_seed(0)
N = 5
X = torch.linspace(0,1,N)
Y = torch.linspace(0,1,N)
sampling_QS = torch.zeros(N**2,2)
k = 0
for x in X:
    for y in Y:
        sampling_QS[k,0], sampling_QS[k,1] = x, y
        k += 1
plt.scatter(sampling_QS[:,0].numpy(), sampling_QS[:,1].numpy());
ax = plt.gca()
ax.set_aspect(1)
plt.xticks(X.numpy());
plt.yticks(Y.numpy());
```

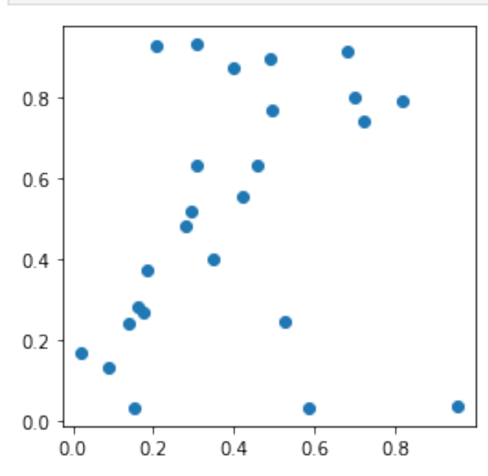


可以发现，无论单独观察横轴还是纵轴，都只取了$[0, 0.25, 0.5, 0.75, 1]$这5个值。如果函数很平坦，倒也无关紧要，但是如果函数变化剧烈，那么这些采样点就有可能无法捕捉这些信息。

但是对于蒙特卡洛采样：

In [6]:

```
sampling_MC = torch.rand(N**2,2)
plt.scatter(sampling_MC[:,0].numpy(), sampling_MC[:,1].numpy());
ax = plt.gca();
ax.set_aspect(1);
```

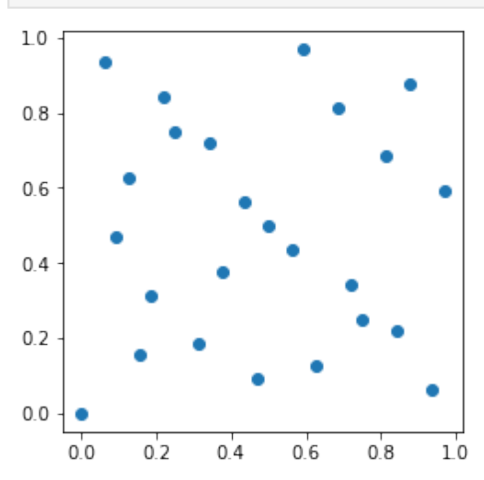


我们可以看出，无论是虽然在空间中点的个数一样，但是有25个不同的x值和不同y值被采样到。初此之外，我们这里采用的是均匀分布的随机采样，但是这个采样的分布在蒙特卡洛积分中并没有被限制，人们可以根据先验知识（如果有的话）设计不同的采样，在关键的地方增加采样概率。

而且还有**拟蒙特卡洛**采样等其他方法，在保证边缘信息的情况下，在空间中分布的更加均匀，这样就兼具了在空间均匀性和对于每一个轴的边缘采样点数量。

In [7]:

```
QuasiMC =  torch.quasirandom.SobolEngine(2).draw(N**2)
plt.scatter(QuasiMC[:,0].numpy(), QuasiMC[:,1].numpy());
ax = plt.gca();
ax.set_aspect(1);
```



总结

蒙特卡洛积分在数值计算中有重要应用。而且它还有非常多的变形，大家可以自己查阅资料。