机器学习中的概率与统计

庞龙刚@CCNU

第十届华大 QCD 讲习班，2022

学习目标

- 通过 Python 学习机器学习中的概率与统计
- 深度学习常用的KL divergence，交叉熵，2分类与多分类
- 与贝叶斯分析相关的马尔科夫链蒙特卡洛采样

学习内容

- 随机变量，概率分布函数
- 方差、涨落、关联
- 熵，联合概率，条件概率，独立
- KL divergence，互信息
- Cross Entropy Loss
- 二分类与多分类任务
- 中心极限定理
- 马尔科夫链蒙特卡洛法采样

In [1]:
```
import numpy as np
from tqdm import tqdm
from scipy.stats import norm
import matplotlib.pyplot as plt

import mplhep as hep
hep.styles.use("ATLAS")
```

随机变量与概率密度分布

随机变量是取值随机会而定的变量。比如，可以用 p(X=true) 表示随机变量 X 取值为真的概率，用 p(X=false) 表示 X 取值为假的概率。举例：X 表示 “明天会下雨”。

p(X) 满足 $0 \leq p(X) \leq 1$， $\sum_x p(X=x) = 1$

$p(X=x)$ 可以简写为 $p(x)$, 记为概率密度分布。

X 的可能取值形成一个可数(或连续无限)的集合（**态空间** χ），如果态空间可数且离散，比如 $\chi = [1,2,3,4,5]$, 则 X 被称为离散型随机变量，满足

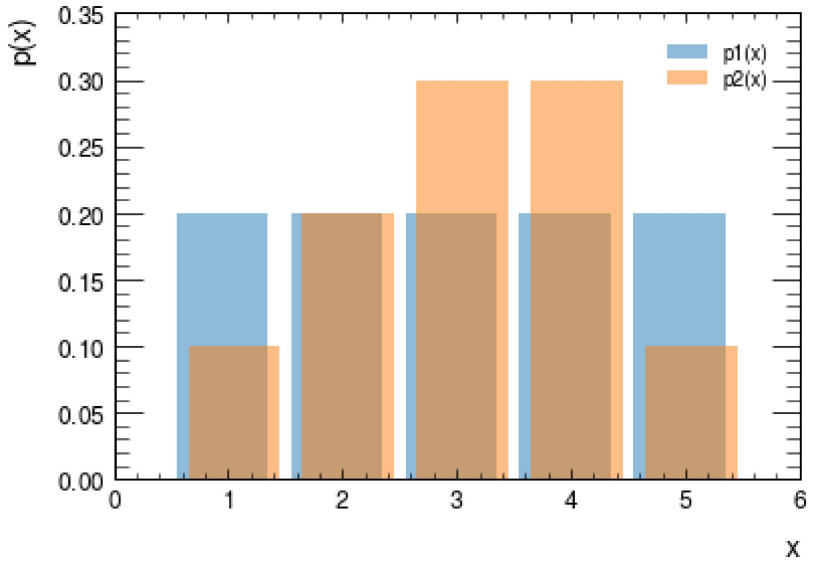
$$\sum_{x \in \chi} p(x) = 1$$

如果 X 的可能取值是连续无限的，则 X 是连续型随机变量，满足

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

In [2]:
```
x = np.array([1, 2, 3, 4, 5])                    # 态空间
p1 = np.array([0.2, 0.2, 0.2, 0.2, 0.2])         # 概率密度分布 1
p2 = np.array([0.1, 0.2, 0.3, 0.3, 0.1])         # 概率密度分布 2
plt.bar(x-0.05, p1, alpha=0.5, label="p1(x)")
plt.bar(x+0.05, p2, alpha=0.5, label="p2(x)")
plt.xlabel("x")
plt.ylabel("p(x)")
plt.legend(loc='best')
```

Out[2]:
```
<matplotlib.legend.Legend at 0x2b5fea07610>
```



期望值，方差与涨落

随机变量 X 的期望值（均值），

$E[X] = \langle x \rangle = \int xp(x)dx$

随机变量的函数也是随机变量，比如，$f(x)$ 的期望值定义为：

$E_{x\sim p}[f] = \int f(x)p(x)dx$

x 平方的均值：$E\left[x^2\right] = \langle x^2 \rangle = \int x^2 p(x)dx$

它经常被用来计算系统的涨落，即对某个 f(x)，多次测量，偏离平均值 $\langle f(x) \rangle$ 的度量，

var = $\langle \delta f^2 \rangle = \langle (f - \langle f \rangle)^2 \rangle = \langle f^2 \rangle - \langle f \rangle^2$

熵 Entropy

对于满足分布 p(x) 的离散型随机变量 X，假设它的态空间大小为 K，则其熵定义为，

$$H(X) = -\sum_{k}^{K} p(X=k)\log_2 p(X=k) = -E\left[\log_2 p(x)\right]$$

对于连续型随机变量 X，其熵为，

$$H(X) = -\int p(x)\log_2 p(x)dx$$

熵测量了 X 不确定度的期望值。它近似等于 X 取特定值时，我们能够获取的平均信息量。log 的不同基底只提供一个常数因子，不重要。

In [3]:
```
def discrete_entropy(p_arr):
    '''计算离散型概率密度分布的信息熵
    :p_arr: numpy 数组，X 取 K 个不同值时的概率
    :return: 信息熵, -sum_k p(X=k) log_2 p(X=k)'''
    return - np.sum(p_arr * np.log2(p_arr))

p1_entropy = discrete_entropy(p1)
p2_entropy = discrete_entropy(p2)
print("H(X1)=", p1_entropy)
print("H(X2)=", p2_entropy)
```

```
H(X1)= 2.321928094887362
H(X2)= 2.1709505944546685
```

In [4]:
```
from scipy.integrate import quad

# quad 提供了连续函数的数值积分
def entropy(f, xmin=-10, xmax=10):
    flogf = lambda x: -f(x) * np.log(f(x))
    return quad(flogf, -10, 10)[0]

entropy(norm.pdf, -10, 10)
```

Out[4]:
```
1.418938533204673
```

In [5]:
```
# scipy.stats 中的正态分布函数 norm 内建的熵计算
norm.entropy()

# 结果与 -f(x) log f(x) 积分一致。

# 尝试：对于不同宽度的正态分布，计算它们的熵并对比
```

Out[5]:
```
array(1.41893853)
```

In [6]:
```
def gauss(x, mu=0, sigma=1):
    coef = 1/(np.sqrt(2*np.pi) * sigma)
    return coef * np.exp(-(x - mu)**2 / (2.0 * sigma**2))

# 更宽的正态分布，熵更大
gauss2 = lambda x: gauss(x, mu=0, sigma=2)
gauss3 = lambda x: gauss(x, mu=0, sigma=3)
print("sigma=2, entopy=", entropy(gauss2))
print("sigma=3, entopy=", entropy(gauss3))
```

```
sigma=2, entopy= 2.1120770693016646
sigma=3, entopy= 2.5102495294962
```

基本概率公式

对于两个随机变量，概率密度分布为 2 维函数，

1. A and B 的概率 (联合概率，joint probabilities)

$$p(A,B) = p(A \wedge B) = p(A|B)p(B)$$

1. A or B 的概率

$$p(A \vee B) = p(A) + p(B) - p(A \wedge B)$$

如果 A 与 B 是互斥事件，

$$p(A \vee B) = p(A) + p(B)$$

1. 边际分布

$$p(B) = \sum_a p(B|A=a)p(A=a)$$

1. 条件概率

$$p(A|B) = \frac{p(A,B)}{p(B)}$$

独立事件

两个事件 A 与 B 无条件独立，可以写为，

$$p(A,B) = p(A)p(B)$$

协方差与关联

有了涨落的定义，就可以进一步定义两个随机变量之间的关联。即两者取值同时增大，或同时变小的概率（忽略归一化）。

先定义协方差 (Covariance)

$$\begin{aligned} cov(f_1,f_2) &\equiv \langle \delta f_1 \delta f_2 \rangle & (1)\\ &= \langle (f_1 - \langle f_1 \rangle)(f_2 - \langle f_2 \rangle) \rangle & (2)\\ &= \langle f_1 f_2 \rangle - \langle f_1 \rangle \langle f_2 \rangle & (3) \end{aligned}$$

再定义关联，

$$corr(f_1,f_2) \equiv \frac{cov(f_1,f_2)}{\sqrt{var(f_1)var(f_2)}} \quad (4)$$

寻找依赖关系

如果两个事件独立，则

$$cov(A,B) = 0$$

$$corr(A,B)=0$$

反过来不成立，如果 $corr(A,B)=0$, 不意味 A 独立于 B。

举例：$X\sim U(-1,1)$, 且 $Y=X^2$, 此时 Y 明显依赖 X，但 $cov(X,Y)=\langle X^3\rangle-\langle X\rangle\langle X^2\rangle=0$。

更好的判据是**互信息（Mutual information）**。

In [7]:
```
x = np.random.uniform(-1, 1, size=10000000)
y = x**2
np.mean((x - x.mean() * (y - y.mean())))
```

Out[7]:
```
-0.0002452501029298282
```

KL Divergence

KL divergence 量化两个概率密度分布之间的差别。

对于离散型随机变量，KL divergence 定义为，

$$KL(p||q)\equiv\sum_{k=1}^{K}p_k\log\frac{p_k}{q_k}$$

对于连续型随机变量，上面的求和变为积分。

In [8]:
```
def kl_divergence(p, q):
    return np.sum(p * np.log(p / q))

kl_divergence(p1, p2)
```

Out[8]:
```
0.11507282898071239
```

In [9]:
```
kl_divergence(p2, p1)  # 非对称
```

Out[9]:
```
0.10464962875290947
```

In [10]:
```
kl_divergence(p1, p1)
```

Out[10]:
```
0.0
```

Cross Entropy （交叉熵）

KL divergence 可以分开写为，

$$KL(p||q)\equiv\sum_{k=1}^{K}p_k\log\frac{p_k}{q_k}=\sum_k(p_k\log p_k-p_k\log q_k)=-H(p)+H(p,q)$$

其中 $H(p,q)\equiv-\sum_k p_k\log q_k$ 被定义为两个分布 p 和 q 的交叉熵。

The cross entropy is the average number of bits needed to encode data coming from a source with distribution p when we use model q.

In [11]:
```
def cross_entropy(p, q):
    return - np.sum(p * np.log(q))

cross_entropy(p1, p2)
```

Out[11]:
```
1.7245107414148126
```

In [12]:
```
cross_entropy(p2, p1)
```

Out[12]:
```
1.6094379124341003
```

在二分类任务中，经常使用 Cross Entropy Loss，

$$L(y,p) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log p_i + (1-y_i)\log(1-p_i)]$$

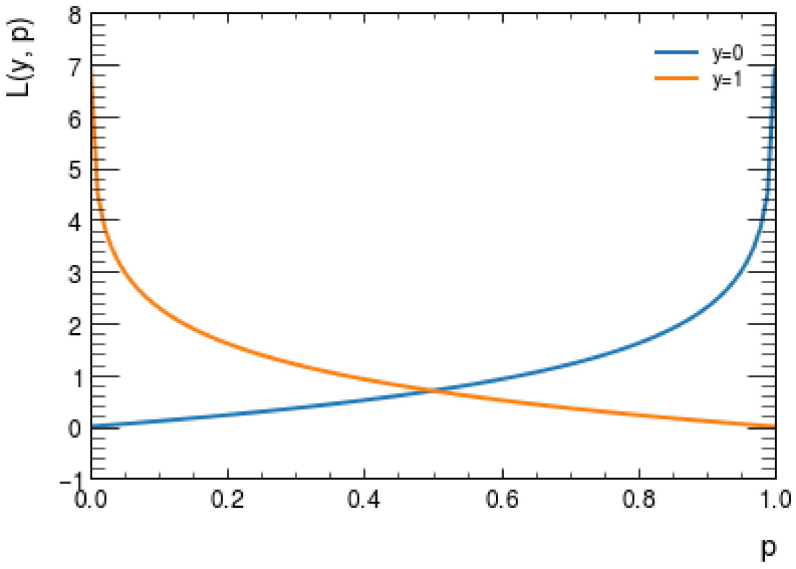
其中

- y_i 是二分类中第 i 个样本的真实标签，取值 0 或 1。
- p_i 是深度神经网络预测的分类概率，$0 < p_i < 1$。
- m 表示一个批次的训练样本个数。
- 如果 $y_i = 0$, 则只有第 2 项贡献，如果 $y_i = 1$, 则只有第一项贡献。

接下来画图对比真实标签 y=0 和 y=1 时 Cross Entropy Loss 随预测结果 p 的变化，

In [13]:
```
p_i = np.linspace(0.001, 0.999, 100)
plt.plot(p_i, -np.log(1 - p_i), label="y=0")
plt.plot(p_i, - np.log(p_i), label="y=1")
plt.legend(loc='best')
plt.xlabel("p")
plt.ylabel("L(y, p)")
# Cross Entropy Loss 加大对 (y=0, p=1) 以及 (y=1, p=0) 的惩罚
# 即加大惩罚“错误又自信的预测”
```

Out[13]:
```
Text(0, 1, 'L(y, p)')
```



可以看到，真实标签 $y = 0$ 时，如果预测结果 p<0.5, 则 Loss 比较小，如果预测错误 $p > 0.5$,

则 Loss 加速增大，如果预测值 p=1, 即预测错误且非常自信，则损失函数在这一点发散。

Cross Entropy Loss 加大对 (y=0, p=1) 以及 (y=1, p=0) 的惩罚，即加大惩罚"错误又自信的预测"。

多分类任务

在 n 个种类的分类任务中，数据的标签用 One Hot 表示，即相应的位为 1， 其他位为 0 的数组。

比如，在 MNIST 手写数字识别任务中，分类目标是识别 0 到 9 共 10 个手写数字。

则每个数字的真实标签设置为：

$$
\begin{aligned}
0 &= [1,0,0,0,0,0,0,0,0,0] && (5)\\
1 &= [0,1,0,0,0,0,0,0,0,0] && (6)\\
2 &= [0,0,1,0,0,0,0,0,0,0] && (7)\\
&\cdots && (8)\\
9 &= [0,0,0,0,0,0,0,0,0,1] && (9)
\end{aligned}
$$

每个数字的预测概率用 10 个数字输入 softmax 激活函数计算，

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{10}$$

z_i 一般被称作 logits, softmax 激活函数将一般的 logits 数组转化为归一化的概率。

In [28]:
```
def softmax(zvector):
    '''多分类的 softmax 激活函数
    :zvector: 每个类别的 logits
    :return: 一个样本属于每个类别的概率
    '''
    exp_zi = np.exp(zvector)
    return exp_zi / exp_zi.sum()

# 产生 3 个随机数，表示 3 分类的 logits
zi = np.random.uniform(-1, 1, size=3)
print("zi=", zi)
```

```
zi= [ 0.59444965 -0.01124788 -0.76823288]
```

In [29]:
```
print("softmax(zi)=", softmax(zi))
# softmax 将数组转化为归一化的概率
```

```
softmax(zi)= [0.55504157 0.30288266 0.14207577]
```

Mutual Information (互信息)

互信息定义为 $p(x,y)$ 和 $p(x)p(y)$ 之间的 KL divergence。

$$I(x;\ y) \equiv KL\left(p(x,y)||p(x)p(y)\right) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = E\left[\log \frac{p(x,y)}{p(x)p(y)}\right]$$

根据 $p(x|y) = p(x,y)/p(y)$, 可以证明，

$$I(x;\ y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

其中 $H(X|Y)$ 和 $H(Y|X)$ 称作条件熵，定义为

$$H(Y|X) = -\sum_{x,y} p(x,y)\log p(y|x) = -E\left[\log p(y|x)\right]$$

互信息可以解释为观测到 Y 之后，X 不确定性的减小量，也等于观测到 X 后，Y 不确定性的减小量。

蒙特卡洛方法（打靶问题）

假设想对比不同奥运选手的打靶成绩，我们不是先通过大量实验统计出每个选手子弹落在不同位置 x 的概率 $p(x)$, 再乘以不同位置对应的分值函数 $f(x)$ 并做积分来计算，而是直接让选手打靶，统计 S 次打靶的平均分，

$$E\left[f(X)\right] = \int f(x)p(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x_s)$$

这对应着蒙特卡洛方法，子弹的每个落点相当于从 $p(x)$ 函数中采样的一个样本， $f(X)$ 的期望值可以用在这些样本上 $f(X = x_s)$ 的均值来近似。

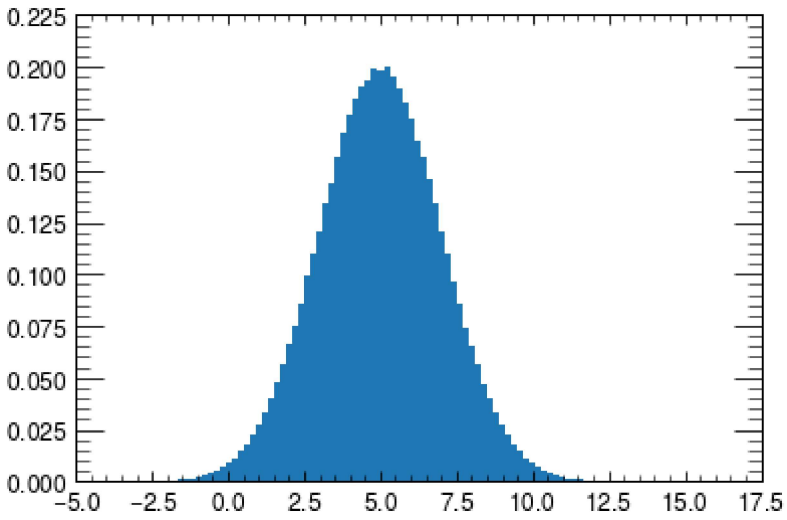
通过改变函数 f, 可以得到很多感兴趣的统计量，比如

$$\bar{x} = \frac{1}{S}\sum_{s=1}^{S} x_s \to E\left[x\right] \tag{11}$$

$$\frac{1}{S}\sum_{s=1}^{S}(x_s - \bar{x})^2 \to \mathrm{var}\left[x\right] \tag{12}$$

$$\frac{1}{S}\#\{x_s \le c\} \to P(x \le c) \tag{13}$$

In [16]:
```
#（举例）从正态分布采样
ri = np.random.normal(loc=5, scale=2, size=1000000)
# 对采样得到的样本进行密度估计
_ = plt.hist(ri, density=True, bins=100)
```



In [17]:
```
# 通过计算采样点的均值，获得 x 的期望值
np.mean(ri)
```

Out[17]:
```
4.999600737957644
```

In [18]:
```
np.mean((ri - ri.mean())**2)
```

Out[18]:
```
4.008172022028907
```

In [19]:
```
np.var(ri)
```

Out[19]:
```
4.008172022028907
```

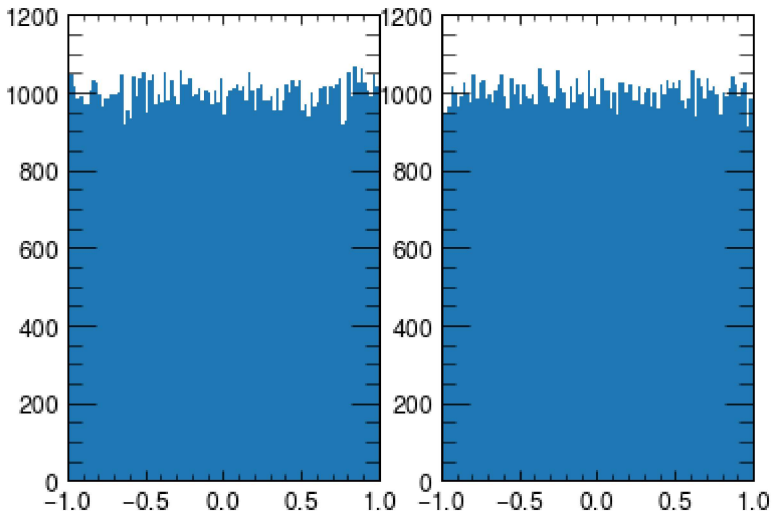
使用蒙特卡洛方法验证中心极限定理

N 个独立同分布的随机变量(不一定满足正态分布)，假设它们的均值都为 μ, 方差都为 σ^2,

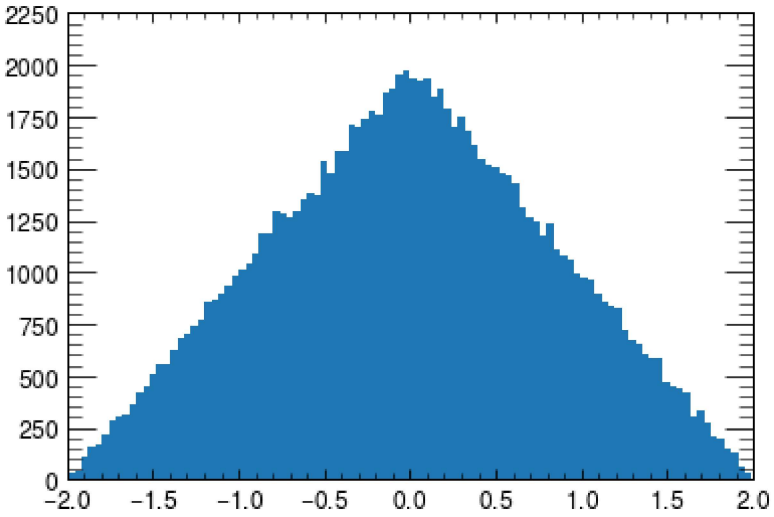
则它们的和 $S = \sum_{i=1}^{N} X_i$ 随 N 的增大，趋近于正态分布，

$$P(S = s) = \frac{1}{\sqrt{2\pi N\sigma^2}} \exp\left(\frac{-(s - N\mu)^2}{2N\sigma^2}\right)$$

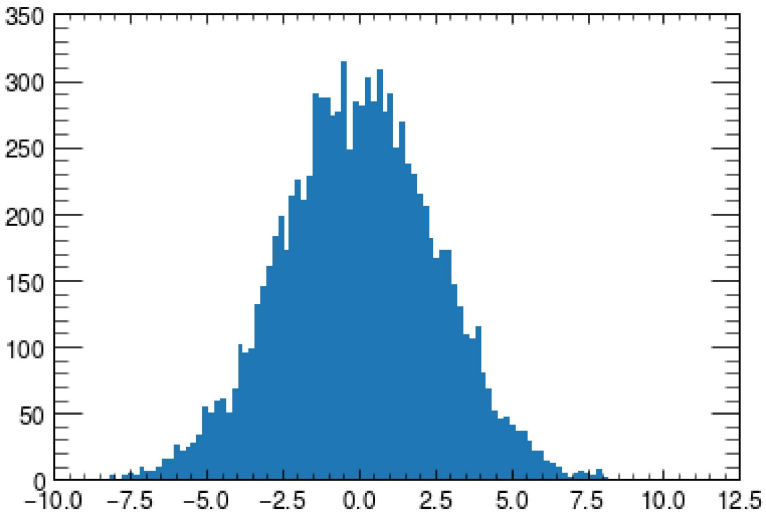
In [20]:
```
# 两个满足 U(-1, 1) 的随机变量 R1 与 R2 之和，满足什么分布？
r1 = np.random.uniform(-1, 1, size=100000)
r2 = np.random.uniform(-1, 1, size=100000)

plt.subplot(121)
_ = plt.hist(r1, bins=100)

plt.subplot(122)
_ = plt.hist(r2, bins=100)
```



In [21]:
```
_ = plt.hist(r1 + r2, bins=100)
```



In [22]:
```
# 20 个满足 [-1, 1] 均匀分布的随机变量之和，满足的分布
rn = np.random.uniform(-1, 1, size=(10000, 20))
rn_sum = np.sum(rn, axis=1)

_ = plt.hist(rn_sum, bins=100)
```



重要采样法

$$E\left[f(x)\right]=\int f(x)p(x)dx\approx\frac{1}{S}\sum_{s=1}^{S}f(x_s)$$

如果 $p(x)$ 不容易采样，可以使用与 $p(x)$ 接近的 $q(x)$ 采样，然后计算 $f(x_s)\frac{p(x_s)}{q(x_s)}$ 在这些采样点上的均值，即

$$E\left[f(x)\right]=\int f(x)\frac{p(x)}{q(x)}q(x)dx\approx\frac{1}{S}\sum_{s=1}^{S}f(x_s)\frac{p(x_s)}{q(x_s)}=E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

注意：

$$\mathrm{Var}\left[f(x)\right]\neq\mathrm{Var}_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

马尔科夫链蒙特卡洛方法采样

一些概率密度分布很难用传统方法采样，比如多变量的概率密度分布，此时可以用马尔科夫链蒙特卡洛（MCMC）方法。

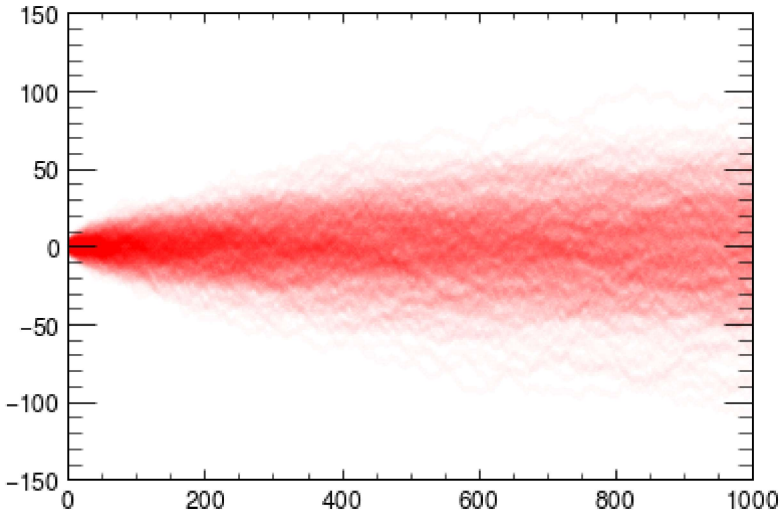
MCMC 方法的基本思路是在态空间χ中构造一个马尔科夫链，使得其稳定分布为待采样的概率密度分布 $p(x)$。

具体做法是在态空间中做随机行走（每一步只依赖于前一步的位置），根据细致平衡条件，调整访问不同状态的频次。

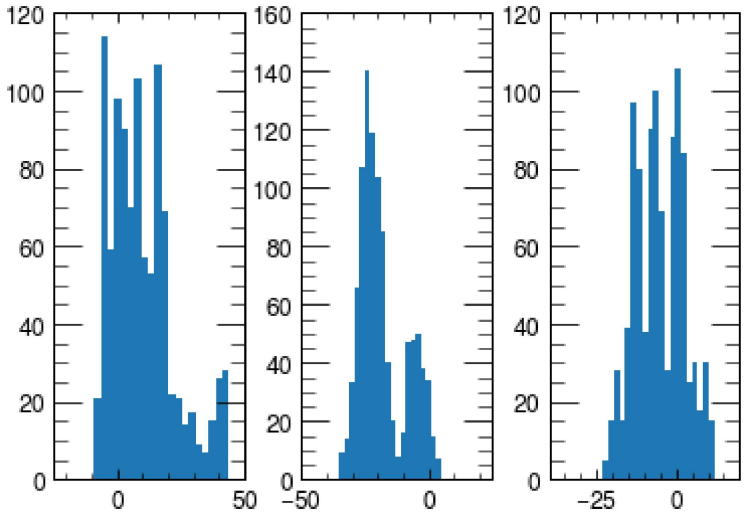
随机行走的轨迹（落脚点）构成一个集合，这个集合是概率密度分布的一个采样。

MCMC 方法最大的优点：可以从非归一化的高维概率密度分布采样

In [23]:
```
# 简单的随机行走举例 （步长为 1）
def random_walk(nsteps=1000, ntracks=10000):
    # 产生 ntracks 条随机行走轨迹，每条轨迹行走 nsteps 步
    r = np.random.uniform(-1, 1, size=(nsteps, ntracks))
    walk = np.where(r > 0, 1, -1)
    x = np.cumsum(walk, axis=0)
    return x

x = random_walk()
_ = plt.plot(x[:, :500], color='r', alpha=0.02)
```



In [24]:
```
plt.subplot(131)  # 第 0 条随机行走轨迹满足的分布
_ = plt.hist(x[:, 0], bins=20)
plt.subplot(132)  # 第 1 条随机行走轨迹满足的分布
_ = plt.hist(x[:, 1], bins=20)
plt.subplot(133)  # 第 2 条随机行走轨迹满足的分布
_ = plt.hist(x[:, 2], bins=20)
plt.subplots_adjust(wspace=0.3)
```



上面的随机行走例子中，下一步的步长要么是 +1， 要么是 -1。

Metropolis Hastings 算法使用 proposal function，来采样下一步到达的位置，

$$q(x_{i+1}|x_i) = \mathcal{N}(x_i, \sigma^2) \tag{14}$$

这个分布以当前样本 x_i 为中心，以方差为 σ^2 的分布抽样，得到下一个样本 x_{i+1}。

σ 是一个超参数，可以手动设置或动态调节。此处可以看到，x_{i+1} 只依赖于 x_i 的位置。

上面这个过程也可以写为，

$$x_{i+1} = x_i + \epsilon$$

其中 ϵ 从 $\mathcal{N}(0, \sigma^2)$ 采样。步长 ϵ 也可从均匀分布 $U(-\sigma, \sigma)$采样。

为了使采样得到的点 x_i 的分布满足待采样的概率密度函数，还需要细致平衡条件，

设置舍选率

$$r = \min(1, \alpha) \tag{15}$$

$$\alpha = \frac{p(x_{i+1})q(x_i|x_{i+1})}{p(x_i)q(x_{i+1}|x_i)} \tag{16}$$

如果 proposal function 对称，则

$$\alpha = \frac{p(x_{i+1})}{p(x_i)}$$

当 $p(x_{i+1}) > p(x_i)$ 时，百分百接受此次采样。否则，以 α 的概率接受此次采样。

Metroplis-Hastings 算法使用对称的 proposal function， Hastings 判据简化为 $r = \min\left(1, \frac{p(x_{i+1})}{p(x_i)}\right)$,

1. 从当前位置 x_0 出发，使用 proposal function $q(x_1|x_0)$ 抽取下一个样本 x_1
2. 如果下一个样本处概率密度 $p(x_1)$ 大于 $p(x_0)$, 则 $r = 1$，百分百接受 x_1
3. 如果下一个样本处概率密度 $p(x_1)$ 小于 $p(x_0)$，也不用放弃，按照 $r = p(x_1)/p(x_0)$ 的概率接受 x_1 即可

- 这需要我们按均匀分布抽取 $u \sim U(0,1)$，如果 $u \leq r$, 接受 x_1; 如果 $u > r$, 复制老的样本, 让 $X = x_0$

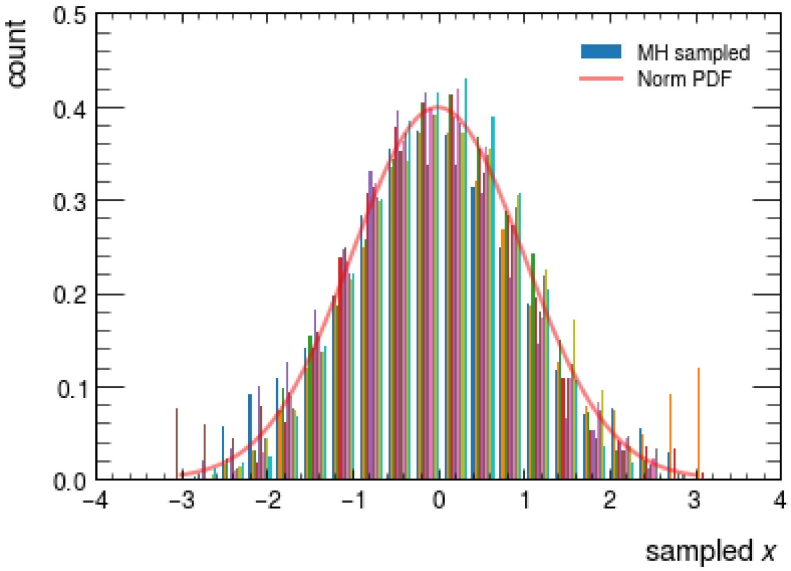
In [25]:
```
def metroplis_hastings(f, x0=0, steps=100000, ntracks=1, sigma=0.1):
    ''' M-H 算法，可以同时抽样多条轨迹 ntracks
    :f: pdf that we are going to sample
    :x0: starting point
    :steps: num of random walks
    :sigma: the std of the proposal normal distribution
    :return: make plot and return trajectories of ntracks (samples {xi})
    '''
    r = np.random.randn(steps, ntracks)
    # x is the history trajectory of the random walk
    x = [np.ones(ntracks) * x0]
    f0 = f(x[0])
    for i in tqdm(range(steps)):
        # 根据 proposal function 做随机行走
        x1 = x[-1] + sigma * r[i]
        f1 = f(x1)
        # 计算 Hasting 判据 alpha
        alpha = np.where(f(x1) > f(x0), 1, f(x1)/f(x0))
        u = np.random.rand(ntracks)
        # 按概率 alpha 接受新的样本，按 1-alpha 概率复制旧的样本
        X = np.where(u <= alpha, x1, x[-1])
        x.append(X)
        f0 = f1

    return np.array(x)
```

In [26]:
```
f = norm.pdf
tracks = metroplis_hastings(f, ntracks=10)
```

```
100%|██████████| 100000/100000 [00:56<00:00, 1768.12it/s]
```

In [27]:
```
# MH 采样结果
_ = plt.hist(tracks, density=True, bins=20, label="MH sampled")
# 待采样的正态分布函数
xrange = np.linspace(-3, 3, 100)
plt.plot(xrange, norm.pdf(xrange), 'r', alpha=0.5, label='Norm PDF')
plt.legend(loc='best')
plt.xlabel(r'sampled $x$')
plt.ylabel("count")
```

Out[27]:
```
Text(0, 1, 'count')
```



贝叶斯分析公式

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\sum_i p(D|\theta_i)p(\theta_i)}$$

其中 θ 是物理模型参数，D 表示实验数据，$p(\theta|D)$ 被称为参数的后验分布。分母上的求和提供归一化系数，需要遍历整个参数空间，非常耗时。

幸运的是，MCMC 方法可以从非归一化的概率密度分布采样 $p(\theta|D) \propto p(D|\theta)p(\theta)$。

使用采样得到的 $\{\theta_i\}$ 集合，就能计算参数的均值，方差，以及使后验分布最大化时 θ 的取值。