第五讲

简介：

本讲接着上一次 Energy-based Model 的取样问题开始，首先介绍了给定任何一个概率分布$p(x)$之后进行取样的一些算法。接下来，话题自然转到一类特殊的模型——Normalizing Flow Models，它们设计就是为了 sampling 的过程变得简单。由此延申，继续讨论了 Autoregressive Flow，并介绍一系列这样的模型的应用。

目录：

Sampling Methods: 引入

我们希望能够从一个任意的分布$P(x)$中进行取样。对于一般的分布，这一操作是困难的；但是对于简单的情况，我们能找到一些采样方法。作为一个最基础的例子，考虑 **Categorical Distribution**，也就是变量x是离散分布的。对于这样的分布，我们总可以通过取一个均匀分布的随机数并划分大小正比于概率的区间来完成取样。可以发现，对于连续的分布，如果我们能写出它的 Cumulative Density Function（CDF），那么我们也可以使用一个类似的方法。换句话说，给定一个从均匀分布中取出的随机数u，我们找到对应的$x = F^{-1}(u)$，其中F是 CDF，而F^{-1}被称为 quantile function。因此，如果 CDF 具有 closed form，我们就可用这个方法实现取样。

而对于 CDF 没有 closed form 的情况，最常见的就是高斯分布。但是我们可以用 Box-Muller transform 实现高斯分布的取样：如果u_1, u_2在$(0,1)$均匀分布，我们就可取

$$x = \sqrt{-2\ln u_1}\cos(2\pi u_2), y = \sqrt{-2\ln u_1}\sin(2\pi u_2)$$

它们均为符合高斯分布的变量。这是因为我们可以计算：

$$f(x,y) = \left|\frac{\partial(x,y)}{\partial(u_1,u_2)}\right|^{-1} = \left[\sqrt{-2\ln u_1}\cdot 2\pi\cdot\frac{1}{2\sqrt{-2\ln u_1}}\cdot\left(\frac{2}{u_1}\right)\right]^{-1} = \frac{u_1}{2\pi} = \frac{1}{2\pi}e^{-\frac{x^2+y^2}{2}}$$

因此，这里就给出了一个十分高效的从高斯分布取样的方法。我们进而可以考虑从可以取样的均匀分布和高斯分布开始进行数学变形，得到更复杂的概率分布。

另外一种思路是考虑进行近似，希望通过比较小的计算得到近似的取样分布。一个很重要的取样方法是 **Importance Sampling**。对于目标的概率分布$p(x)$，我们使用一个容易取样的分布$q(x)$（也称为 **proposal distribution**），再给取样得到的结果赋以不同的权重。这一结果建立在

$$E_{x\sim p}[f(x)] = E_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$$

的基础上。但是对于方差等更高阶的分布信息，我们这样的取样方法得到的分布不一定能和$p(x)$完全一致。实际上，可以想象，$q(x)$和需要的$p(x)$越接近，这一近似就越精确。也就是说，这样的方法要求我们必须对需要取样的分布给出一个不错的近似，同时。在其他情况下，我们可能需要其他的方法。

Sampling Methods: Markov Chain Monte-Carlo

我们引入 Markov Chain 的概念：它包括一个 state space S和 transition probability $P(s_j|s_i) = T(s_i \to s_j) = T_{ij}$。一开始给定一个分布后，之后的每一轮按照 transition probability 进行演化，也就是说 state s_i有$T(s_i \to s_j)$的概率到达s_j。如果用π_i代表第i轮演化后的结果，那么我们最后可以得到一个 stationary distribution $\pi_\infty = T^\infty \pi_0$。

可以看出计算 Markov Chain 的演化是一个时间开销相对较小的过程，因此如果能构造一个 Markov Chain 使得$\pi_\infty = p(s)$就是我们想要的分布，我们就有了一个有效的 Sampling Method。因此，我们自然要讨论 Markov Chain 的 stationary distribution 满足的性质。理论上可以证明，π是 stationary distribution 的一个充分条件是所谓 **detailed balance**：

$$\pi(s)T(s \to s') = \pi(s')T(s' \to s)$$

同时，为了保证一个唯一的 stationary distribution，我们要求 Markov Chain 是 **ergodic** 的，也就是从任何一个状态都有概率到达其他任何一个状态。这对应的数学表述是

$$\min_{z} \min_{z'} \frac{T(z \to z')}{\pi(z')} = \delta > 0$$

同时满足 detailed balance 和 ergodic 性质的 Markov Chain 被称为 **valid** 的。

那么应该如何构造这样的 Markov Chain 呢？**Metropolis Hastings Algorithm** 给出了一个方案：

Metropolis Hastings Algorithm

1. 给出一个 proposal distribution $q(s'|s)$

2. 重复：

（1）随机取$s' \sim q(s'|s)$

（2）设$\alpha = \min\left(1, \frac{p(s')q(s' \to s)}{p(s)q(s \to s')}\right)$为 **acceptance ratio**。以$\alpha$的概率跳跃到$s'$；在剩下的情况里，保留在$s$。这两种操作也分别称为 accept 和 reject。

可以证明，这样的方案的确给出了一个 valid 的 Markov Chain 的演化，并且其 stationary distribution 就是p。

一个重要的情况是在 Metropolis Hastings Algorithm 中取$q(s' \to s)$为某种仅依赖于$|s' - s|$的分布。这样，我们就可以化简出

$$\alpha(s \to s') = \min\left(1, \frac{p(s')}{p(s)}\right)$$

对于 energy-based model，我们有$p(s) = \frac{1}{z}\exp(-E(s))$。此时，Metropolis Hastings Algorithm 的这个修改版本就可也被表述为上一讲提到的

从 Generic Energy-based Model 中取样

1. 随机初始化x^0

2. 重复：

（1）为x^t加一些噪声，得到x'（更具体地，$x'-x^t$遵循高斯分布或者均匀分布）；

（2）（i）如果$E(x')<E(x^t)$，那么取$x^{t+1}=x'$；

（ii）如果$E(x')\geq E(x^t)$，那么以$\exp\big(E(x^t)-E(x')\big)$的概率取$x^{t+1}=x'$；

作为一个总结，Metropolis Hastings Algorithm 是一个对于给定的概率分布$p(x)$高效构造一个 valid Markov Chain 的方式。和 Importance Sampling 方法一样，它里面也具有一个 proposal q；但是这一方法相比于 Importance Sampling 的优势在于（大致的描述）q并不是全局的“盲猜”$q(x)\approx p(x)$，而是局部的猜测$q(x'|x)$。这使得它的 sampling 更加精确。

但是 Metropolis Hasting Algorithm 有一个问题：如果 acceptance rate 比较低，我们可能会浪费很多次计算机会。我们接下来考虑如何解决这个问题。

Sampling Methods: Other methods

Gibbs Sampling 是一个在前面的 Metropolis Hasting Algorithm 的基础上选取一种$q(s \to s')$以实现 **acceptance rate 始终为 1** 的 Sampling Algorithm。类似于之前 Energy model 的 Gibbs Sampling，我们取

$$q(s \to s') = \begin{cases} 0, s\text{和}s'\text{相差多于 2 位} \\ p(s_i' | s_{j \neq i}), \text{otherwise} \end{cases}$$

可以验证，这样的选取可以保证$\alpha(s \to s')$在$q(s \to s')$非零的时候总是 1。当然，只有在 posterior $p(s_i' | s_{j \neq i})$很容易 sample 的时候这一方法才可以具有很好的计算效率。不过对于很大一类函数（Exponential Family）而言，这一方法都表现很优秀。

虽然如此，整个 MCMC（Markov Chain Monte Carlo）方法体系都具有一个公共的问题：因为它们都使用 sampling 的方式，它们具有**较慢的 convergence rate**。因而，人们希望把 gradient 的信息加入，这就有了 Stochastic Gradient MCMC。

除此之外，人们还提出了许多 Non-sampling methods——既然 sample 这样的困难，我们 sample 复杂概率分布的原始目的是为了给 generative model 的概率分布计算期望，那为什么我们不能直接设计一个模型，它本身就很好 sample 呢？这样就产生了之后的 Normalizing Flow。换句话说，我们从原先**构造一个很有表现力的模型并研究该如何 sample 它**转化到了**构造一个比较容易 sample 的模型并研究如何让它更有表现力（Easy to sample & have tractable likelihood）**。

Normalizing Flow：引入

如何构造一个容易 sample 的模型呢？我们已经知道比较容易 sample 的概率分布有均匀分布和高斯分布；同时，如果x容易sample，那么$z=f(x)$也容易sample。具体地，根据多元微积分的知识，我们有

$$p_z(z)=p_x(x)\left|\det\left(\frac{\partial f^{-1}(z)}{\partial z}\right)\right|=p_x(x)\left|\det\left(\frac{\partial f(x)}{\partial x}\right)\right|^{-1}$$

人们据此提出了 Normalizing Flow 模型：首先从一个基本的分布（比如高斯分布）取出x_0，再经过K个双射：

$$x_0\overset{f_1}{\to}x_1\overset{f_2}{\to}x_2\cdots\overset{f_K}{\to}x_K\overset{f_{K+1}}{\longrightarrow}z$$

这样，最后的概率分布就可也写为

$$p(z)=p(x_0)\prod_i\left|\det\left(\frac{\partial f_i(x_{i-1})}{\partial x_{i-1}}\right)\right|^{-1}$$

为了训练最终的概率分布达到目标，我们需要 log-likelihood，它恰好就是

$$\log p(z)=\log p(x_0)-\sum_i\log\left|\det\left(\frac{\partial f_i}{\partial x_{i-1}}\right)\right|$$

这样，我们就可以给出这一类 generation model 的训练和生成：假设$z=f(x)$是我们的整个变换：

1. **Forward pass**，即通过x计算z，对应**生成**，也被称为 **inference**（把x从比较简单的分布中随机取样，就可以得到一个需要的z分布）；

2. **Backward pass**，即通过z计算x，对应**训练**，也被称为 **sampling**（这是因为我们只有得到了x才能计算 log-likelihood，进而使用上一次的方法进行训练）。

但是可以注意到，这样的计算依然存在困难：对于大小为d的输入，雅可比矩阵$\frac{\partial f_i}{\partial x_{i-1}}$是$d\times d$的，因此计算的行列式是$O(d^3)$的。如何解决这一问题？我们发现，我们必须有一个 **structured Jacobian**，也就是说我们可以快速地计算这个具有结构的行列式，而不是只能按照定义计算。

Planar Flow 是一个早期的尝试，它使用所谓的 Matrix Determinant Lemma 进行数学上的操作，通过选取特殊的f形式保证每一次行列式的计算在$O(d)$时间内进行。在$K=10$层的情况下，该模型已经取得了很不错的结果。但是一个更普适的尝试是**把 Jacobian 变成上三角/下三角矩阵**，这很容易实现：如果记$f(x)=(f_1(x),\cdots,f_d(x))$，那么我们只需要让$f_i(x)$只依赖于$x_{\le i}$就可以让$\frac{\partial f}{\partial x}$这个矩阵变成下三角。

NICE（Nonlinear Independent Components Estimation）就是使用这个方法：它的每一层计算是$z=f(x)$，其中每个向量被分为两个部分，分别有

$$\begin{cases}z_{1:m}=x_{1:m}\\ z_{m+1:d}=x_{m+1:d}+\mu_\theta(x_{1:m})\end{cases}$$

其中μ_θ代表一个带参数θ的 neural network。对于这样的一个网络，我们始终有$\det J = 1$。这也被称为 Volume-preserving transformation。与此对应，存在 **NVP**（即 Non-Volume Preserving）的方法，它对前面的公式做了一定的改动：

$$\begin{cases} z_{1:m} = x_{1:m} \\ z_{m+1:d} = \exp\big(\alpha_\theta(x_{1:m})\big) * x_{m+1:d} + \mu_\theta(x_{1:m}) \end{cases}$$

其中添加的α_θ是一个 neuron network，它具有$m+1 \sim d$指标，和后面的$x_{m+1:d}$对应，而$*$代表分量乘。这个α的加入使得 Jacobian 的行列式不再为 1，而是

$$\det J = \prod_{i=m+1}^{d} \exp(\alpha_\theta(x_{1:m})_i)$$

另外一个简化行列式计算的思路是 **GLOW**（Generative Flow with Invertible 1x1 Convolutions）：如果输入的维度d比较大导致计算$O(d^3)$的行列式开销比较大，我们可以反过来考虑**对不同的 channel 进行卷积，这样卷积核的大小就是$c \times c$，计算行列式的开销就是$O(c^3)$。**这种方法对应的 forward pass 计算方式是

$$z_{ij:} = W x_{ij:} + b$$

其中i, j代表某个 pixel 的x, y坐标，而第三个坐标:代表对于不同的 channel 之间存在线性组合。对于 backward，只需要计算W的逆矩阵就可以了。这样可以发现，对于一个i, j而言，$\frac{\partial z_{ij}}{\partial x_{ij}}$都是矩阵$W$。因此，对于全部的输入$x$、输出$z$的 Jacobian 就是$w \times h$个$W$放在对角线上，因此

$$\log|\det J| = hw \log|\det W|$$

当然，只用若干这样的卷积叠加起来就像是 Linear 之间忘记加 ReLU，并不能达到预期的目的。我们还需要一些 normalizing layer 来加入非线性性。

也是基于这个思路，一个重要的模型——Autoregressive Flow 诞生了。我们会把它作为一个完整的话题介绍。

Autoregressive Flow：引入

Autoregressive（AR） Flow 也是基于把 Jacobian 变为上三角或者下三角的思路，但是和之前的思路不同，我们使用 sequential generation，即取

$$p(x) = \prod_{i=1}^{d} p(x_i|x_{<i})$$

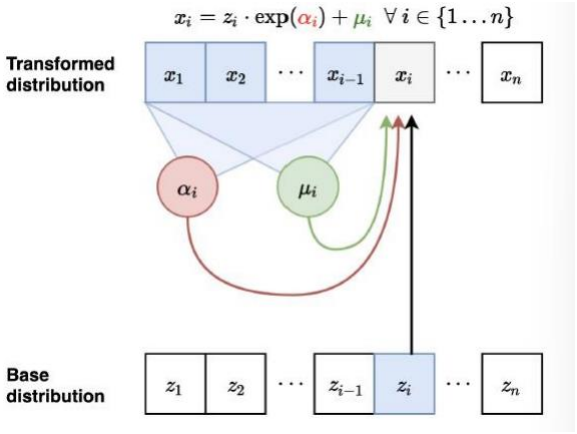
这样log-likelihood就比较容易可以得到。一个例子是Gaussian Autoregressive Model，即

$$p(x_i|x_{<i}) = N\big(\mu_i(x_{<i}), \exp\big(\alpha_i(x_{<i})\big)\big)$$

这样，forward pass 就可以写为：

1. 取d个随机数$z_i \sim N(0,1)$
2. 重复：
   （1）用$x_{<i}$计算$\mu_i(x_{<i}), \alpha(x_{<i})$；
   （2）计算

$$x_i = \exp(\alpha_i) z_i + \mu_i$$



这一过程也可以用右边的图示比较形象地给出。我们可以发现，因为每一次计算新的x_i需要借助前面的$x_{<i}$来给出α和μ，**forward 是不可以并行的**，最少需要$O(d)$的时间才可以完成。然而，我们可以考虑 backward 过程：

$$z_i = \exp(-\alpha_i)\,(x - \mu_i)$$

因为x_i已经全部已知，我们可以并行计算出α_i, μ_i；同时，也可以并行计算出z_i。因此**backward 过程可以以比 forward 快很多的速度进行**。

Autoregressive Flow：Quantization

Autoregressive Flow 遇到的一个比较细微的问题是 **Quantization**：按照前面方法生成的x都是一个连续的概率分布，那么对于离散的变量值（比如图片的 pixel 值只能是 0 到 255），该如何处理呢？对于 inference（也就是生成）计算，我们可以用上下取整的方式；但是在训练的时候，给定的数据点并不能给出x不是整数的时候的概率分布的限制，因此我们需要给出一种新的训练方法。

我们自然想到使用一段$[x, x+1)$上进行训练。换句话说，给定$\hat{p}$这个原始的连续分布，我们给出一个对应的离散分布

$$p(x) = \int_{[0,1)^d} \hat{p}(x+u)du$$

并按照$p(x)$进行训练。原来离散的的目标 loss 函数是

$$L_0(\theta) = E_{x=\text{data}}[\log p(x)]$$

我们需要用$\hat{p}(x)$给出这个 loss 的近似值。实际上，我们取

$$\begin{aligned} L(\theta) = E_{x'\approx\text{data}}[\log \hat{p}(x')] &= \sum_x p_{\text{data}}(x) \int_{[0,1)^d} \log \hat{p}(x+u)\, du \\ &\leq \sum_x p_{\text{data}}(x) \log \int_{[0,1)^d} \hat{p}(x+u)du = L_0(\theta) \end{aligned}$$

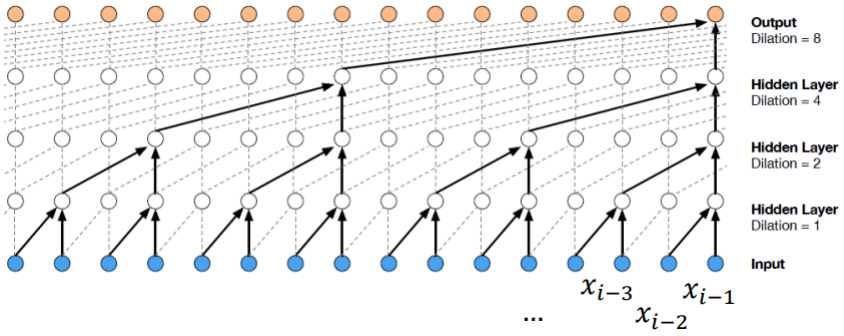
因此我们的这个 loss function $L(\theta)$就是真实损失函数$L_0(\theta)$的下界的近似。用这个方法，我们就解决了这个问题，进而可以在离散的数据上对$E_{x'}[\log \hat{p}(x')]$进行训练即可。

Autoregressive Flow：例子

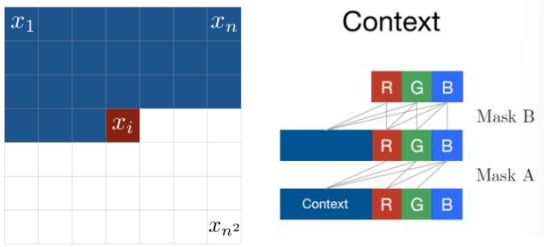
Autoregressive Flow 的一个例子是 **WaveNet**，它给出

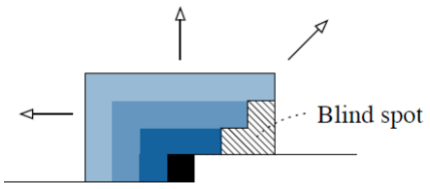
$$p(x) = \prod_i p(x_i|x_1, \dots, x_{i-1})$$

的一个实现方法。如果按照一般的方式，需要N层；但是这一方法使用如下的结构，使得只需要$O(\log N)$层。这样，generation 需要$O(N)$的计算，而 likelihood 的计算（即训练）则是和之前一样可以完全 parallel 进行的。当然，这只是一个简单的模型，实际的实现更加复杂。



另外一个例子是 **PixelCNN**，它通过 Convolution 实现 Autoregressive 的生成。实现这一点的重要思路是 **Masked Convolution**，即把卷积核的中心右下的部分全部设置为 0 并进行多次卷积。对于反向的 likelihood 计算，可以证明这个卷积操作可以高度平行地进行。当然，对于更细节的讨论，需要提到这里必须**使用两种不同的卷积核 Mask A 和 Mask B**，分别对应着卷积核中间是否为 1：Mask A（中心为 0）保证变换的行列式是 1；Mask B（中心为 1）保证多层卷积后结果不会因为大小原因变成 0。即便如此，卷积的依赖关系依然会存在“Blind Spot”，即右图中黑色的块不会依赖于阴影部分的结果。





为了解决这一问题，可以把卷积的结构变得更加复杂一些，这就有了 Gated PixelCNN。这一方法解决了 Blind Spot 的问题，进而有着更好的 performance。

Autoregressive Flow：Inversed Autoregressive Flow（IAF）

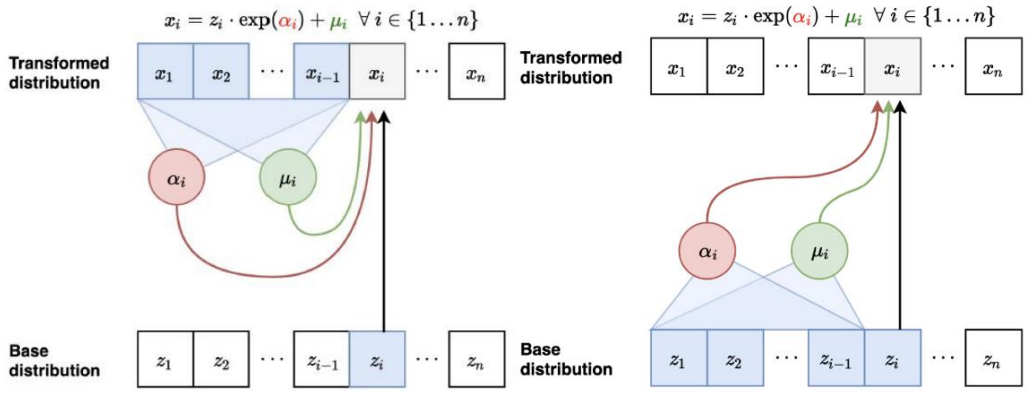
让我们回到之前介绍的 Gaussian Autoregressive Flow：

$$x_i = \exp\big(\alpha_i(x_{<i})\big)z_i + \mu_i(x_{<i})$$

之前提到，这样的方法在 forward pass 的时候只能顺序计算，因此比较慢；但是在 backward pass 却可以并行处理。这产生应用上的一些问题：对于大批量的使用，我们希望生成应该快一些，哪怕以牺牲训练速度为代价。为了实现这样的操作，我们可以**考虑让α_i, μ_i依赖于$z_{<i}$而非$x_{<i}$**。这样就有了 Inversed Autoregressive Flow（IAF）：

$$x_i = \exp\big(\alpha_i(z_{<i})\big)z_i + \mu_i(z_{<i})$$

可以把这两种方法对比如下：左边是 AF，而右边是 IAF。



可以立刻料想，IAF 的生成很快，因为可以根据z并行计算x；但是训练则相对较慢，因为x得到z必须通过顺序地解方程实现。同时，两个模型的表现力应该是类似的，因为结构本身是相似的。

我们也自然会想：是否有办法使得我们可以充分利用两者的性质，得到一个训练和生成都相对快的网络呢？这就有了 **Parallel WaveNet**，其关键想法是 **Teacher-Student Framework**：**老师**是一个标准的 AF，可以很快的实现训练，得到$p_T(x_i|x_{<i})$；而**学生**是一个 IAF，它的训练虽然比较慢，但是可以很快的生成。完成老师的训练之后，我们只需要最小化学生的$p_S(x)$和$p_T(x)$之间的差别就可以了。但这一方法的精妙之处在于，有了老师的结果后，学生**不再需要对于数据x反推z**，因为这一工作可以交给老师来进行。这就解决了学生训练速度慢的瓶颈。

更具体地，学生网络的训练过程称为 **Imitation Learning**：我们最小化学生分布和老师分布的 **KL Divergence**

$$L(\theta) = KL(p_S||p_T) = E_{x\sim p_S}[\log p_S(x;\theta) - \log p_T(x)]$$

注意到这里**取样是从学生网络进行（$x \sim p_S$）**，因此十分高效。具体的方法为：

1. 随机取样$z \sim N(0, I)$，并取样$x \sim p_S(x|z)$ //（这步是学生取样，高效）；

2. 计算$p_S(x|z)$ //（本来不高效，但是因为z已知，所以高效）；

3. 计算$p_T(x)$ //（这步是老师训练，高效）.

换一种理解方式，原来学生不高效的方式是因为我们要对给定的$x \in$ data进行训练，而这些对应的z是未知的；但现在高效的原因是我们训练的 **loss 函数改变了**，不再是对于$x \in$ data的 log-likelihood，而是**对于任何的x，只要$p_S(x)$接近于$p_T(x)$即可**。

总结

回顾一下，Normalizing Flow 的**基本思想**是我们得到一个容易 sample 的模型（换句话说，有一个 tractable likelihood），使得它适用于 MLE（Maximize Likelihood Estimation）的训练方式。它的实现有很多方式，一个重要的思路就是通过上三角/下三角化给出 **Structured Jacobian**。

Autoregressive Flow 是一个特别的例子，它通过直接分解

$$p(x_1, \dots, x_d) = \prod_{i=1}^{d} p(x_i | x_{<i})$$

使得每一个 Jacobian 都比较容易得到。据此有了 AF 和 IAF 两种不同的网络结构，并且它们的优势可以被结合。

会想起上一讲的 Energy-based model，我们发现它是另外一个极端：极度的 Flexibility 带来的代价是 sample 的困难，这就是为什么本讲的开始需要介绍 Sampling Method。Metropolis Hastings Algorithm 的形式已经给出了很不错的 sampling 效率和结果，但是 Energy-based model 依然很难训练。与此相反，设计即是为了 sample 高效的 Normalizing Flow 很容易进行训练，但是却具有一个限制的 structure 种类。

在最后，我们不妨拔高视角：这一切的困难都来自于上一讲提出的训练方式，也就是 Maximize Likelihood Estimation。他真的是必须的吗？在接下来的几讲我们会继续讨论这一点。