Understanding VAE and Normalizing Flows

Rongfan Li

Contents

1 Introduction

Simple distributions (e.g., Gaussian) are often used as likelihood distributions. However, the true distribution is often far from this simple distribution and this results in issues such as blurry reconstructions in the case of images. Latent variable models such as VAEs often set the prior distribution $p(\mathbf{z})$ to a factorial multivariate Gaussian distribution. Such a simplistic assumption hampers the model in multiple ways. For instance, this does not allow a multi-modal latent space distribution. Normalizing Flows allow transformation of samples from a simple distribution (subsequently denoted by q_0) to samples from a complex distribution by applying a series of invertible flows.

1.1 Preliminaries and Notations

- Uppercase X denotes a random variable
- Uppercase $P(X)$ denotes the probability distribution over that variable
- Lowercase $x \sim P(X)$ denotes a value x sampled ($\sim$) from the probability distribution $P(X)$ via some generative process.
- Lowercase $p(X)$ is the density function of the distribution of X. It is a scalar function over the measure space of X.
- $p(X = x)$ (shorthand $p(x)$) denotes the density function evaluated at a particular value x.
- $p(Z|X)$ is the **posterior probability**: "given the image, what is the probability that this is of a cat?" If we can sample from $z \sim P(Z|X)$, we can use this to make a cat classifier that tells us whether a given image is a cat or not.
- $p(X|Z)$ is the **likelihood**: "given a value of Z this computes how "probable" this image X is under that category ("is-a-cat" / "is-not-a-cat"). If we can sample from $x \sim P(X|Z)$, then we generate images of cats and images of non-cats just as easily as we can generate random numbers. Likelihood is just what generative models want.
- $p(Z)$ is the **prior probability**. This captures any prior information we know about Z - for example, if we think that 1/3 of all images in existence are of cats, then $p(Z = 1) = 1/3$ and $p(Z = 0) = 2/3$.

1.2 Distribution of a Simple Transformation of a Random Value

Before jumping into normalizing flows, let's consider a simple univariate distribution $p(x) = 2x$ with support $x \in [0, 1]$. Define a function $y = f(x) = x^2$. Note that $f(x)$ is monotonically increasing in $[0, 1]$. What is the PDF of the variable y? (see Probability density function wikipedia for details.)

First, we have some basic knowledges:

$$
\begin{aligned}
F_X(a) &= P(X \leq a) = \int_{-\infty}^{a} f_X(x)dx \\
f_X(x) &= \frac{\partial F_X(x)}{\partial x}
\end{aligned}
\tag{1}
$$

We can compute $p(y)$ using the CDFs as follows.

$$
\begin{aligned}
F_Y(y) &= P(Y \leq y) \\
&= P(X^2 \leq y) \\
&= P(X \leq \sqrt{y}) \\
&= F_X(\sqrt{y})
\end{aligned} \tag{2}
$$

Now, $p(y) = F'_Y(y) = \frac{dF_X(\sqrt{y})}{dy}$ where

$$
F_X(\sqrt{y}) = \int_0^{\sqrt{y}} p(x)dx \tag{3}
$$

$$
= \left[2\frac{x^2}{2}\right]_0^{\sqrt{y}} \tag{4}
$$

$$
= y \tag{5}
$$

differentiating w.r.t. y we get $\frac{d(y)}{dy} = 1$ which means that $p(y) = \mathcal{U}(0, 1)$.

1.3 Change of Variables

See section 3 for arbitrary function situations and multivariate distributions.

The method described above can be extended to multivariate distributions $q_0(\mathbf{z})$ and smooth invertible mappings $f : \mathbb{R}^d \Rightarrow \mathbb{R}^d$. Samples $\mathbf{z} \sim q_0(\mathbf{z})$ can be transformed using f to give $\mathbf{y} = f(\mathbf{z})$. The PDF of $\mathbf{y}$ is given by

$$
q_1(\mathbf{y}) = q_0(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{y}} \right| = q_0(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1} \tag{6}
$$

where the second equality comes from the inverse-function theorem.

Rezende et. al. proposed two different families of invertible transformations: planar flow and radial flow.

Improving inference, sampling, and density estimation in deep generative models (DGMs) are important research problems. Simple distributions (e.g., Gaussian) are often used as likelihood distributions in DGMs. However, the true distribution is often far from this simple distribution, which results in issues such as blurry reconstructions in the case of images. Latent variable models, such as VAEs, often set the variational posterior distribution $q(\mathbf{z}|\mathbf{x})$ to a factorial multivariate Gaussian distribution. Such a simplistic assumption hampers the model in multiple ways. For instance, this does not allow a multi-modal latent space distribution. Normalizing Flows allow transformation of samples from a simple distribution into samples from a complex distribution, whose density can be evaluated analytically, by applying a series of invertible transformations. In this report, we discuss a number of recent works that introduce techniques for improved variational inference and/or density estimation in deep generative models using normalizing flows.

1.4 Visualizing of PDF

Now, let's compute $\mathbf{y} = f(\mathbf{z})$ and the density $q_1(\mathbf{y})$. Note that we not inverting f. Instead, we are first setting a $\mathbf{z}$ and then plotting the density $q_1(\mathbf{y})$ at corresponding $\mathbf{y}$s.

概率密度和频次有直接的联系，因为 $q_1(y)d_y$ 就是某点的概率，下面证明

$$F_X(a) - F_X(b) = P(b < X \leq a) = \int_b^a f_X(x)d_x \tag{7}$$

当 ab 无限接近的时候，有 $\int_b^a f_X(x)dx = (a-b)f_X(a) = f_X(a)dx$。

在频次图中，我们又知道，某值的频次占据总次数的比例就是概率，当频次进行归一化后，使得总频次为 1，某值的频次就是某值的概率（100 次出现 20 次和 1 次出现 0.2 次是一样的，概率是 1/5），我们可以用频次来模拟概率

综上，在直方图中，频次的累积就代表了某点的概率，也表示某点的概率密度函数的大小，反之亦然，我们可以用概率密度函数的大小来表示该点的频次，或者可以用密度函数大小来对这个点进行着色

下面开始举例说明。首先生成 1000000 个均匀的随机变量。注意这里是均匀的，所以其实代表的是无限次采样的期望，最终的图像会很平滑。如果需要真正的随机，用下面一段生成方式即可，见子图 d 和 e，其边缘是比较模糊的。

Listing 1: Initialize the random variable

```
r = np.linspace(-3, 3, 1000)
z = np.array(np.meshgrid(r, r)).transpose(1, 2, 0)
z = np.reshape(z, [z.shape[0] * z.shape[1], -1])

# randomly sample
z = np.random.normal(size=(int(1e6),2))
```

准备从随机变量中生成多元随机分布

Listing 2: Generate multivariate Gaussian distribution

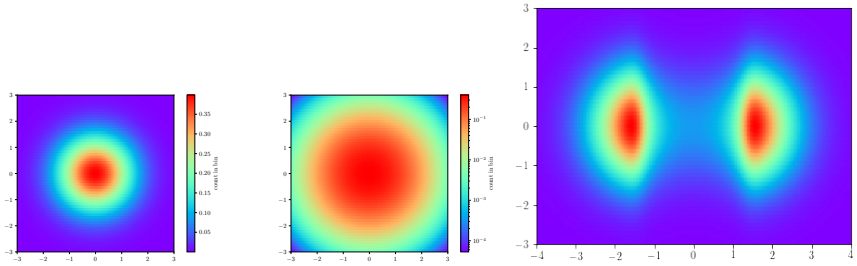
```
    def mvn_pdf(X, mu=np.array([[0, 0]]), sig=np.eye(2)):
        sqrt_det_2pi_sig = np.sqrt(2 * np.pi * LA.det(sig)) # determinat
        sig_inv = LA.inv(sig) # inversion
        X = X[:, None, :] - mu[None, :, :]
        return np.exp(-np.matmul(np.matmul(X, np.expand_dims(sig_inv, 0)), (X.transpose(0, 2,
            1)))/2)/sqrt_det_2pi_sig
```

生成 PDF 后画图，见1子图 a。hist2d 和 hexbin 都能累积频次，频次越高，则对应位置的颜色越深，refference，用另一个颜色 map 方式，就可以反映这种变化，见子图 b，通过调整 bin 参数，原来较小的频次就显得更红了。hexbin 的前两个参数是位置，C 表示对参数进行累积。如果不用 PDF 进行累积，那么画出来的图形将是纯色。而如果是真正随机采样的，便不需要指定 C，因为 z 自己就分布不均匀。

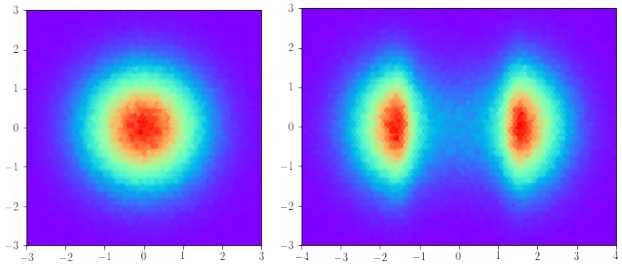
Listing 3: Draw multivariate Gaussian distribution

```
q0 = mvn_pdf(z)
plt.hexbin(z [:,0], z [:,1], C=q0.squeeze(), cmap='rainbow')
plt.gca().set_aspect('equal', adjustable='box')

# another way to plot
plt.hexbin(z [:,0], z [:,1], C=q0.squeeze(), cmap='rainbow',bins="log")
```



(a) Initial Density (b) Initial Density Variant (c) Planar Flow



(d) Initial Density (e) Planar Flow

Figure 1: Change in standard normal density on application of length 1 planar and radial flows.

已知，$q_1(\mathbf{y}) = q_0(\mathbf{z}) \left|\det \frac{\partial f}{\partial \mathbf{z}}\right|^{-1}$ 用雅可比矩阵得到 q1 的概率密度后，可以看到其分布如子图 c 所示。

Listing 4: Draw transformed distribution

```
q1 = q0.squeeze()/det_J(z).squeeze()
y = f(z)
plt.hexbin(y[:,0], y[:,1], C=q1.squeeze(), cmap='rainbow')
plt.gca().set_aspect('equal', adjustable='box')
```

2 VAEs

2.1 Autoencoder

An Autoencoder (AE) [5] is comprised of two neural networks:
Encoder The encoder network f_ϕ which transforms an input data-point $\mathbf{x} \in \mathbb{R}^d$ into a representation $\mathbf{z} \in \mathbb{R}^p$ where $p < d$ generally.
Decoder The decoder network g_θ which attempts to reconstruct $\mathbf{x}$ as $\widehat{\mathbf{x}} \in \mathbb{R}^d$ from the representation $\mathbf{z}$.

Training proceeds by minimizing the reconstruction error between $\mathbf{x}$ and $\widehat{\mathbf{x}}$ and the training

objective is given in Eq. (8)

$$\phi, \theta = \arg\min_{\phi,\theta} \mathcal{L}(\widehat{\mathbf{x}}, \mathbf{x}) \tag{8}$$

where $\mathcal{L}$ is a suitable reconstruction error which can be the squared error $\|\widehat{\mathbf{x}} - \mathbf{x}\|_2$ for real-valued observations or binary cross-entropy $\sum_{i=1}^{d} -\mathbf{x}|_i \log(\widehat{\mathbf{x}}|_i) - (1-\mathbf{x}|_i)\log(1-\widehat{\mathbf{x}}|_i)$ for binary observations.

2.2 Reverse Kullback-Leibler Divergence and ELBO

We usually have *Reverse Kullback-Leibler Divergence* to measure the distance between two distributions. We wish to minimize this quantity with respect to ϕ. Note that, minimizing $KL(Q||P)$ is to disort Q to approximate P, and it has a contrary meaning to optimal a reverse form with respect to forward form.

$$KL(Q_\phi(Z|X)||P(Z|X)) = \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z|x)} dz \quad \text{Continuous form} \tag{9}$$

$$KL(Q_\phi(Z|X)||P(Z|X)) = \sum_{z \in Z} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z|x)} \quad \text{Discrete form} \tag{10}$$

It is crucial that KL divergence can be written as *expectation form*, so that we can use *Monte Carlo EM* algorithm to sample from the posterior.

There is still some chaos in the use of $\int$ and $\sum$, but it does not disturb the derivation.

By definition of a conditional distribution, $p(z|x) = \frac{p(x,z)}{p(x)}$, we substitute this expression into our original KL expression, and then distribute:

$$\begin{aligned}
KL(Q||P) &= \sum_{z \in Z} q_\phi(z|x) \log \frac{q_\phi(z|x)p(x)}{p(z,x)} \\
&= \sum_{z \in Z} q_\phi(z|x) \big(\log \frac{q_\phi(z|x)}{p(z,x)} + \log p(x) \big) \\
&= \Big(\sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z,x)} \Big) + \Big(\sum_z \log p(x) q_\phi(z|x) \Big) \\
&= \Big(\sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z,x)} \Big) + \Big(\log p(x) \sum_z q_\phi(z|x) \Big) \quad \text{note: } \sum_z q(z) = 1 \\
&= \log p(x) + \Big(\sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z,x)} \Big)
\end{aligned} \tag{11}$$

To minimize KL with respect to variational parameters ϕ, we just have to minimize the second part $\sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z,x)}$, since $\log p(x)$ is fixed with respect to ϕ. See expectation for details of how to compute expectation of a random variable.

$$\begin{aligned}\sum_z q_\phi(z|x)\log\frac{q_\phi(z|x)}{p(z,x)} &= \mathbb{E}_{z\sim Q_\phi(Z|X)}\big[\log\frac{q_\phi(z|x)}{p(z,x)}\big]\\ &= \mathbb{E}_Q\big[\log q_\phi(z|x) - \log p(x,z)\big]\\ &= \mathbb{E}_Q\big[\log q_\phi(z|x) - (\log p(x|z) + \log(p(z)))\big]\\ &= \mathbb{E}_Q\big[\log q_\phi(z|x) - \log p(x|z) - \log(p(z))\big]\end{aligned} \tag{12}$$

And minimizing it is equivalent to maximizing the negation of this function:

$$\begin{aligned}\text{maximize } \mathcal{L}_{\text{ELBO}} &= -\sum_z q_\phi(z|x)\log\frac{q_\phi(z|x)}{p(z,x)}\\ &= \mathbb{E}_Q\big[-\log q_\phi(z|x) + \log p(x|z) + \log(p(z))\big]\\ &= \mathbb{E}_Q\big[\log p(x|z) + \log\frac{p(z)}{q_\phi(z|x)}\big]\end{aligned} \tag{13}$$

$\mathcal{L}$ is known as the *evidence lower bound* (ELBO), and is computationally tractable if we can evaluate $p(x|z), p(z), q(z|x)$. We can further re-arrange terms in a way that yields an intuitive formula:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_Q\big[\log p(x|z) + \log\frac{p(z)}{q_\phi(z|x)}\big]\\ &= \mathbb{E}_Q\big[\log p(x|z)\big] + \sum_Q q_\phi(z|x)\log\frac{p(z)}{q_\phi(z|x)} \quad \text{Definition of expectation}\\ &= \mathbb{E}_Q\big[\log p(x|z)\big] - \sum_Q q_\phi(z|x)\log\frac{q_\phi(z|x)}{p(z)}\\ &= \mathbb{E}_Q\big[\log p(x|z)\big] - KL(Q(Z|X)||P(Z)) \quad \text{Definition of KL divergence}\end{aligned} \tag{14}$$

第一项，是给定 $Q(Z|X)$ 的时候，求得 p 使得期望最大，即在 encoder 输出的时候，decoder 要尽力使解码能力增强，这使得 encoder 和 decoder 要配合。第二项，在 $Q(Z|X)$ 得到的 Z 的分布要尽量和原来的 Z 保持一致，否则这个就不是 noise 了，x 到 z 再回到 x，如果 z 没有 noise 的加入，在神经网络中便多此一举。

Now back to (11), we substitute the complicated expressions with ELBO.

$$\begin{aligned}KL(Q||P) &= \log p(x) - \mathcal{L}\\ \mathcal{L} &= \log p(x) - KL(Q||P)\\ \text{w.r.t}\quad \mathcal{L} &= \mathbb{E}_Q\big[\log p(x|z)\big] - KL(Q(Z|X)||P(Z))\end{aligned} \tag{15}$$

In (15), $p(x)$, the log-likelihood of a data point x under the true distribution, equals $\mathcal{L} + KL$, and the KL term can be treat as an error that capture the distance between Q and P. Since $KL(Q||P) \geq 0$, $\log p(x) \geq \mathcal{L}$, which means $\mathcal{L}$ is the *evidence lower bound* of $\log p(x)$. 这里的 likelihood 是指，如何重建这个 $p(x)$ 使得这些 x 出现的概率最大，其实这里指的是 $p(x|\theta)$，此时最大化 likelihood 和最大化 ELBO 是等价的。

Note that $\mathcal{L}$ itself contains a KL divergence term between the approximate posterior and the prior, so there are two KL terms in total in $\log p(x)$.

2.3 Variational Autoencoder

注意 VAE 这里的符号来自于原文，很多参数使用了 boldsymbol。

The Variational Autoencoder (VAE) [8] combines an inference network f_ϕ (analogous to the encoder in AE) with a generative model g_θ (analogous to the decoder in AE). There is a complete introduction to VAE in Chinese by Jianlin Su.

In short, VAE wants to optimize the parameters so that our proposed posterior q_ϕ (an easy, parametric distribution $Q_\phi(Z|X)$, like a Gaussian, for which we know how to do posterior inference) can approximate the true posterior p.

Recall (15), VAE maximizes the ELBO $\mathcal{L} = \mathbb{E}_Q\big[\log p(x|z)\big] - KL(Q(Z|X)||P(Z))$ to get the approximation of the true posterior. The first part is reconstruction error of decoder(generative model), and the second is error of encoder(inference model).

2.3.1 KL in ELBO

在 VAE 中，假设 prior $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ 是多元高斯分布，p 的参数是未知的，其实这里是什么都无所谓，反正是要靠 posterior 去模拟。令 $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ 也是一个多元高斯分布，其参数是通过一个 MLP 算出来的。True posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ is intractable in this case. 然后我们有关键的假设，我们使用一个各向同性的高斯分布来做 approximate posterior:

$$\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)}) = \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\mathbf{I}) \tag{16}$$

又因为随机采样是无法求梯度的，所以，再通过重参数化技巧 (reparameterization trick)，使得梯度可以传递回参数上。We sample from the posterior $\mathbf{z}^{(i,l)} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ using $g_{\boldsymbol{\phi}}\left(\mathbf{x}^{(i)}, \boldsymbol{\epsilon}^{(l)}\right) = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$ where $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are outputs of the encoder, i.e. nonlinear functions of inputs $\mathbf{x}^{(i)}$ and variational parameters $\boldsymbol{\phi}$, $\odot$ is element-wise product。其中，$\boldsymbol{\mu}^{(i)}$ 和 $\boldsymbol{\sigma}^{(i)}$ 都是 encoder 的输出，这个 encoder 用 MLP 来实现，输入就是不同的 $\mathbf{x}^{(i)}$，参数是 $\boldsymbol{\phi}$。从而我们得到了 $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ 的采样，从而去判断这个 posterior 是否和先验分布接近 $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$。

下面推导一元正态分布的情况。

$$\begin{aligned}
&KL\Big(N(\mu, \sigma^2)\Big\|N(0,1)\Big) \\
=&\int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \left(\log \frac{e^{-(x-\mu)^2/2\sigma^2}/\sqrt{2\pi\sigma^2}}{e^{-x^2/2}/\sqrt{2\pi}}\right) dx \\
=&\int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \log\left\{\frac{1}{\sqrt{\sigma^2}} \exp\left\{\frac{1}{2}\left[x^2 - (x-\mu)^2/\sigma^2\right]\right\}\right\} dx \\
=&\frac{1}{2}\int \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}\Big[-\log\sigma^2 + x^2 - (x-\mu)^2/\sigma^2\Big] dx
\end{aligned} \tag{17}$$

整个结果分为三项积分，第一项实际上就是 $-\log\sigma^2$ 乘以概率密度的积分（也就是 1），所以结果是 $-\log\sigma^2$；第二项实际是正态分布的二阶矩，熟悉正态分布的朋友应该都清楚正态分布的二阶矩为 $\mu^2 + \sigma^2$；而根据定义，第三项实际上就是 “-方差除以方差 =-1”。所以总结果就是

$$KL\Big(N(\mu, \sigma^2)\Big\|N(0,1)\Big) = -\frac{1}{2}\Big(\log\sigma^2 - \mu^2 - \sigma^2 + 1\Big) \tag{18}$$

对应的多元的情况如下，其中 j 是 j-th element of these vectors of input $\mathbf{x}^{(i)}$.

$$-\frac{1}{2}\sum_{j=1}^{J}\left(1+\log\left(\left(\sigma_j^{(i)}\right)^2\right)-\left(\mu_j^{(i)}\right)^2-\left(\sigma_j^{(i)}\right)^2\right) \tag{19}$$

所以也能看出，我们的核心是 posterior 是正态分布，这样才方便重参数化和采样，而 prior 假设为正态分布是无奈之举。其实 prior 可以不假设成正态分布，重写 KL 即可，反正都能算一个解析解出来，同时，posterior 只是去逼近 prior，无论 prior 是什么，都没有期待他完全地拟合。

下面是两个多元正态分布的 KL 散度求解。(符号不规范) 其中 J 是维数。

$$\begin{aligned}
D_{KL}(q(z)\|p(z)) &= \int q(z)\log\frac{q(z)}{p(z)}dz \\
&= \int q(z)\{\log q(z)-\log p(z)\}dz \\
&= \int q(z)\log q(z)dz-\int q(z)\log p(z)dz \\
&= \left\{-\frac{J}{2}\log(2\pi)-\frac{1}{2}\sum_{j=1}^{J}\left(\log\sigma_{1,j}^2+1\right)\right\}- \\
&\left\{-\frac{J}{2}\log(2\pi)-\frac{1}{2}\sum_{j=1}^{J}\log\sigma_{2j}^2-\frac{1}{2}\sum_{j=1}^{J}\left[\frac{\sigma_{1,j}^2}{\sigma_{2,j}^2}+\frac{\left(\mu_{1,j}-\mu_{2,j}\right)^2}{\sigma_{2,j}^2}\right]\right\} \\
&= -\frac{1}{2}\sum_{j=1}^{J}\left[\log\frac{\sigma_{1,j}^2}{\sigma_{2,j}^2}-\frac{\sigma_{1,j}^2}{\sigma_{2,j}^2}-\frac{\left(\mu_{1,j}-\mu_{2,j}\right)^2}{\sigma_{2,j}^2}+1\right]
\end{aligned} \tag{20}$$

2.3.2 Reconstruction error in ELBO

ELBO 中还有一项，就是 $\mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p(x|z)\big]$，可以用蒙特卡洛法来解决。注意，上一节我们已经可以得到 samples from the posterior $\mathbf{z}^{(i,l)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})$ 了。

一般情况下，蒙特卡洛法可以写成如下形式

$$\mathbb{E}_{x\sim p(x)}[f(x)]=\int f(x)p(x)dx\simeq\frac{1}{n}\sum_{i=1}^{n}f(x_i),\quad x_i\sim p(x) \tag{21}$$

并且随着采样的数量增加，期望的计算越精确。现在有 L 个随机采样，$\boldsymbol{\epsilon}^{(l)}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$，生成了 $\mathbf{z}^{(i,l)}$。原文公式 8 下面指出，只要 x 的 minibatch M 取得够大，那么采样的数量 L 可以取 1。

$$\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)})}\big[\log p(\mathbf{x}^{(i)}|\mathbf{z})\big]\simeq\frac{1}{L}\sum_{l=1}^{L}\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})\quad\text{where }\mathbf{z}^{(i,l)}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}^{(i)}) \tag{22}$$

$p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$ 是一个概率密度函数，由 decoder 定义。这里 VAE 给出了两种分布，一个是伯努利分布，一个是正态分布，视数据的类型而定。有了 numerical form，就可以用神经网络计算参数，如下 $\tilde{\mu}(z)$ $\tilde{\sigma}(z)$，D 是维数。

$$
\begin{aligned}
p(x|z) &= \frac{1}{\prod_{k=1}^{D}\sqrt{2\pi\tilde{\sigma}_{(k)}^2(z)}}\exp\left(-\frac{1}{2}\left\|\frac{x-\tilde{\mu}(z)}{\tilde{\sigma}(z)}\right\|^2\right) \\
\log p(x|z) &= -\Big(\frac{1}{2}\left\|\frac{x-\tilde{\mu}(z)}{\tilde{\sigma}(z)}\right\|^2 + \frac{D}{2}\ln 2\pi + \frac{1}{2}\sum_{k=1}^{D}\ln\tilde{\sigma}_{(k)}^2(z)\Big)
\end{aligned} \tag{23}
$$

2.3.3 Numerical ELBO

不规范

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}^{(i)}) &= \mathbb{E}_Q\big[\log p(x|z)\big] - KL(Q(Z|X)||P(Z)) \\
&= \frac{1}{L}\sum_{l=1}^{L} -\Big(\frac{1}{2}\left\|\frac{x-\tilde{\mu}(z)}{\tilde{\sigma}(z)}\right\|^2 + \frac{D}{2}\ln 2\pi + \frac{1}{2}\sum_{k=1}^{D}\ln\tilde{\sigma}_{(k)}^2(z)\Big) \\
&\quad + \frac{1}{2}\sum_{j=1}^{J}\left(1+\log\left(\left(\sigma_j^{(i)}\right)^2\right) - \left(\mu_j^{(i)}\right)^2 - \left(\sigma_j^{(i)}\right)^2\right)
\end{aligned} \tag{24}
$$

如果认为 σ 是常数 (实践里一般也不考虑)，那么可以进一步简化

$$
\mathcal{L} \simeq -\frac{1}{2L\sigma}\sum_{l=1}^{L}\|x-\tilde{\mu}(z)\|^2 + \frac{1}{2}\sum_{j=1}^{J}\left(1+\log\left(\left(\sigma_j^{(i)}\right)^2\right) - \left(\mu_j^{(i)}\right)^2 - \left(\sigma_j^{(i)}\right)^2\right) \tag{25}
$$

第一项显然是一个 MSE，完全地定义了一个 reconstruction error。这个 x 就是输入的 $x^{(i)}$，即在假定 σ 常数的时候，我们只需要一个网络的输出即可。直观的理解可以理解成，高斯分布最高的地方，也就是 μ 所在的地方，输出 x 概率最高的地方和 x 接近即可。

最大化 $\mathcal{L}$ 就可以完成优化，相当于最小化 reconstruction error，并最大化第二项。或者更通常的，为了利用随机梯度下降法，令

$$
Loss = -\mathcal{L} \tag{26}
$$

2.4 Assumptions in VAE

VAEs 的关键假设是 [8] 的公式 9，此处为 16。相关描述是 “In this case, we can let the variational approximate posterior be a multivariate Gaussian with a diagonal covariance structure. Note that this is just a (simplifying) choice, and not a limitation of our method. ” (参考多元正态分布 wiki)。

VAE 考虑的是，各分量相互独立的多元正态分布，然而这种简化的设置有极大的局限性。

1. 隐藏空间的变量 $\mathbf{z}$ 并不是相互独立的，而是有相互的联系的
2. $\mathbf{z}$ 的分布并不是正态分布

这些缺点正是 normalizing flow 的发挥之处。NF can modify the simple normal distribution into a richer posterior that can better model the true posterior.

VAE 还假设了 prior 是多元标准正态分布 (centered isotropic multivariate Gaussian. Isotropic means its covariance matrix is represented by $\Sigma = \sigma^2\mathbf{I}$ and all dimensions are independent and the

variance is the same.)，从而计算出了 KL 散度。但是这个假设并不重要，因为 prior 作为多元正态分布是合理的，至于标准，和各向同性，就并不算很准确，但是拿 posterior 来模拟 prior 并不需要非常精确。

最后 VAE 假设 $p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})$ 是正态分布。至于为什么可以直接看成正态分布，是因为混合高斯分布: 在一个变换下，多个高斯分布 $p(z)$ 可以组合成任意复杂的混合高斯分布 $P(x) = \sum_z p(z)p(x|z)$，而这里我们假设这个条件分布是正态分布，在众多分布叠加后，理论上 x 的分布可以完全被学习到。

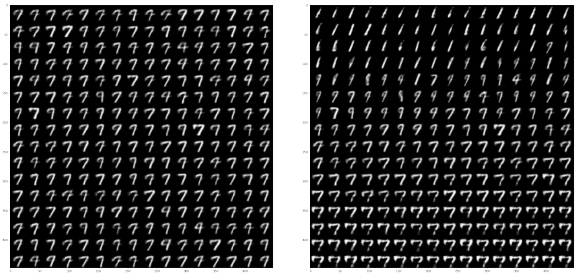
2.5 What VAE learned?

Fig 2a and fig 2b shows the latent space of VAE somehow. We use encoder to encode an image, and get it's exclusive μ and σ. Then we have two different methods to add bias.

Normal distribution. Sample ϵ from normal distribution, combine with μ and σ, then we get the result that VAE would get. As fig 2a shows, the numbers keep it's shape among biases, which seem like 7 or 9. Therefore, 我们知道，encoder 可以捕捉对应数字的对应的联合分布的参数，将这个联合分布的采样解码就是对应的同形的 256 个不同的数字。

Linear spaced sequence. We generated some bias, which does not obbey any distribution but consist of a linear spaced sequence, which from -3 to 3. 前一种方式生成的分布有 5-dimension, and we add 3 of which with the sequence like the code shows. As fig 2b shows, 这种方式生成的分布与上次的分布相比，在 3 个维度上进行了更大程度的，规律性的偏移，可以看出数字从 1 到 4 或 9 到 7 到粗体的 7 转变。可以看出，在 latent space，编码的不同维度组成的联合分布影响了生成的图形，并且这种连续的分布生成的图形也是连续的，通过连续修改其中的某些维度，图形也有渐变。

```
sample_d = 3
b = tf.linspace(-3., 3., 256)
bias = tf.reshape(a, [256, 1])
bias = tf.tile(bias, [1, sample_d])
epsilon_prime = tf.concat([z[:, :sample_d]+bias, z[:, sample_d:]], axis=1)
```



(a) Random epsilon (b) Add sequence bias

上面的两个案例说明，VAE 学习到了如何将同一个数字 7 映射到隐藏空间（想象一个 2 维空间）中，并且无论数字 7 的形状如何改变，他们都会靠近彼此，在 fig 2a中出现形态最多的 7 就是频率最高的 7，也就是 7 对应的 μ（正态分布中 μ 是在波峰，概率最大），说明 VAE 学习到了如何根据图像生成隐藏空间的变量。其次，这个空间是连续的，7 代表的一团空间中的附近，是有其他数字的分布存在的。如果两个分布在某点 z 重叠的概率密度越多，则在对 z 解码后，生成的图像越是处于两者之间—谁覆盖得越多，则生成的图像就越像谁。

可以推理得到，如果在 μ 的附近采样，那么得到的多半也是 μ 相同的数字。

3 Normalizing Flows

Before defining normalizing flows, let's consider a univariate distribution with density function $p(x)$. Define a continuous, differentiable, and increasing function f. Define $y = f(x)$ where $x \sim p(x)$. Recall 1.2 and the density function of the random variable Y can then be derived analytically using the Cumulative Distribution Function (CDF) as follows.

$$\begin{aligned} F_Y(y) &= P(Y \leq y) \\ &= P(f(X) \leq y) \\ &= P(X \leq f^{-1}(y)) = F_X(f^{-1}(y)) \end{aligned} \tag{27}$$

And when f is decreasing, $F_Y(y) = P(X \geq f^{-1}(y))$.

We end up with the CDF of the random variable X at the point $f^{-1}(y)$. Now, $p(y) = F'_Y(y)$ by definition, where

$$F_Y(y) = F_X(f^{-1}(y)) = \int_{-\infty}^{f^{-1}(y)} p(x)dx \tag{28}$$

Differentiating Eq. (28) with respect to y (using the Fundamental Theorem of Calculus given $F(x) = \int_a^x f(t)dt$ then $F'(x) = f(x)$ and the chain rule) we get PDF of Y:

$$\begin{aligned} q(y) &= \frac{dF_Y(y)}{dy} \\ &= \frac{d\int_{-\infty}^{f^{-1}(y)} p(x)dx}{dy} \\ &= \frac{d\int_{-\infty}^{f^{-1}(y)} p(f^{-1}(y))df^{-1}(y)}{dy} \quad \text{Substitute } x \\ &= \frac{d\int_{-\infty}^{f^{-1}(y)} p(f^{-1}(y))df^{-1}(y)}{df^{-1}(y)} \cdot \frac{df^{-1}(y)}{dy} \\ &= p(f^{-1}(y)) \cdot \frac{df^{-1}}{dy} \end{aligned} \tag{29}$$

When f is a decreasing function, we get $q(y) = -p(f^{-1}(y)) \cdot \frac{df^{-1}}{dy}$. For an invertible function in general, which don't need to be monotone, Eq. (29) can be written as

$$q(y) = p(f^{-1}(y)) \cdot \left| \frac{df^{-1}}{dy} \right| \tag{30}$$

Eq. (30) can be extended to the multivariate case where the derivative is replaced by the determinant of the Jacobian matrix

$$q(\mathbf{y}) = p(f^{-1}(\mathbf{y})) \cdot \left| \det \frac{\partial f^{-1}}{\partial \mathbf{y}} \right| = p(f^{-1}(\mathbf{y})) \cdot \left| \det \frac{\partial f}{\partial f^{-1}(\mathbf{y})} \right|^{-1} = p(f^{-1}(\mathbf{y})) \cdot \left| \det \frac{\partial f}{\partial \mathbf{x}} \right|^{-1} \tag{31}$$

$$p(\mathbf{x}) = q(\mathbf{y}) \cdot \left|\det \frac{\partial f}{\partial \mathbf{x}}\right| = q(f(\mathbf{x})) \cdot \left|\det \frac{\partial f}{\partial \mathbf{x}}\right| \tag{32}$$

In the above equation, the second equality comes from the inverse function theorem. Successive applications of such smooth, invertible transformation on a random variable with known density is called a *normalizing flow*.

Computation of the probability density of the transformed random variable **requires the computation of the determinant of the Jacobian matrix which is computationally expensive** as it scales with $O(d^3)$ where d is the dimensionality of the random variable. Developing transformations with cheap determinant computation has been the primary focus of many recent works.

假设已知 $y \sim q(y)$ and $x \sim N(0,1)$，并且也求出了对应的 $x = f^{-1}(y)$ and $y = f(x)$。对应的采样用下标 s 表示。

如果想得到 $y_s \sim q(y)$，则需要先从标准正态中采样 x_s, 然后 $y_s = f(x_s)$。从 x 到 y，正态到一般分布的过程往往称为 forward。

如果有一堆 y_s，想知道对应的概率密度（比如常见的，用分布均匀的 y，通过高亮大概率的 y 部分来显示出 q 的形状），则需要 $x_s = f^{-1}(y_s)$，然后得到 $N(x_s)$，最后用 $\log q(y_s) = \log N(x_s) - \log|\det \frac{\partial f}{\partial \mathbf{x}}|$ 来计算 $q(y_s)$。这个过程往往称为 inverse。

4 Applications

Literature on normalizing flows can be broadly classified into two parts: ones using normalizing flows for improved variational inference and ones using normalizing flows for density estimation.

注意以下的单个变量都是 1-dimension 的，只需要考虑 1-dimension 但是是 multivariate 的分布即可，即随机变量 $\mathbf{V} \in \mathbb{R}^1$，但是多个 $\mathbf{V}$ 组成了一个联合分布，所以下面的变量大小都是 $\mathbf{x} \in \mathbb{R}^{n\times 1}$。

4.1 Variational Inference

Variational methods perform inference by approximating the true posterior $p(\mathbf{z}|\mathbf{x})$ using a simpler variational family $q_\phi(\mathbf{z}|\mathbf{x})$. Recent works have focused on improving the variational posterior used in the VAE which is generally set to a multivariate normal distribution with diagonal covariance matrix $\mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I})$. It is clear that such a simplistic, unimodal choice for the posterior can be arbitrarily far away from the true posterior which can be a complex multi-modal distribution.

Recent works seek to convert samples from a simple variational posterior (such as the multivariate normal distribution) into a richer distribution by applying a series of smooth, invertible transformations or a flow. Let $\mathbf{z}_0$ be a sample from a simple distribution $q_0(\mathbf{z}_0)$ and $\mathbf{z}_K$ be a sample obtained by applying a flow of length K on $\mathbf{z}_0$, i.e., $\mathbf{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0)$. Using Eq. (32), the density function $q_K(\mathbf{z}_K)$ is given by

$$\begin{aligned} q_K(\mathbf{z}_K) &= q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left|\det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|^{-1} \\ \log q_K(\mathbf{z}_K) &= \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left|\det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right| \end{aligned} \tag{33}$$

The variational lower bound in VAEs (Eq. 13) can now be modified by setting $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = q_K(\mathbf{z}_K|\mathbf{x})$

$$
\begin{aligned}
\mathcal{L} &= \mathbb{E}_{q_\phi}\big[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\big] \\
&= \mathbb{E}_{q_K(\mathbf{z}_K|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z}_K) - \log q_K(\mathbf{z}_K|\mathbf{x})\right] \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z}_K) - \log q_K(\mathbf{z}_K|\mathbf{x})\right]
\end{aligned} \tag{34}
$$

where $q_0(\mathbf{z}_0|\mathbf{x})$ is the simple initial density. Note the change of the density of expectation. Plugging in Eq. (33) into Eq. (34), we get a modified bound for flow-based VAEs

$$
\begin{aligned}
\mathcal{L} &= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log p(\mathbf{x},\mathbf{z}_K) - \log q_0(\mathbf{z}_0|\mathbf{x}) + \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z}_K)}{q_0(\mathbf{z}_0|\mathbf{x})} + \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{z}_K)}{q_0(\mathbf{z}_0|\mathbf{x})}\right] + \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right|\right] \quad \text{first term is similar to (13)} \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log \frac{p(\mathbf{x}|\mathbf{z}_K)p(\mathbf{z}_K)}{q_0(\mathbf{z}_0|\mathbf{x})}\right] + \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K}\log|\det|\right] \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z}_K)\right] - \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log \frac{q_0(\mathbf{z}_0|\mathbf{x})}{p(\mathbf{z}_K)}\right] + \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K}\log|\det|\right] \\
&= \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z}_K)\right] - KL(q_0(\mathbf{z}_0|\mathbf{x})\|p(\mathbf{z}_K)) + \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K}\log|\det|\right]
\end{aligned} \tag{35}
$$

或者直接从原来的项中分裂出来，见(14)

$$
\begin{aligned}
KL(Q(\mathbf{z}_K|\mathbf{x})||P(\mathbf{z}_K)) &= \mathbb{E}_{q_K}[\log \frac{q_K(\mathbf{z}_K|\mathbf{x})}{p(\mathbf{z}_K)}] = \mathbb{E}_{q_0}[\log \frac{q_K(\mathbf{z}_K|\mathbf{x})}{p(\mathbf{z}_K)}] \\
&= \sum q_0 \log \frac{q_K(\mathbf{z}_K|\mathbf{x})}{p(\mathbf{z}_K)} = \sum q_0(\log q_K - \log p) \\
&= \sum q_0(\log q_0(\mathbf{z}_0|\mathbf{x}) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right| - \log p) \\
&= \sum q_0 \log \frac{q_0(\mathbf{z}_0|\mathbf{x})}{p(\mathbf{z}_K)} - \sum q_0 \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right| \\
&= KL(q_0(\mathbf{z}_0|\mathbf{x})\|p(\mathbf{z}_K)) - \mathbb{E}_{q_0(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K}\log|\det|\right]
\end{aligned} \tag{36}
$$

Now, given the bound, our attention shold be paid to the computation of the determinant.

4.1.1 Planar and Radial Flow

Planar and Radial Flows [12] are one of the earliest flows proposed in the context of variational inference. Planar flows apply transformations perpendicular to a plane while radial flows apply them around a point.

Planar flows use functions of the form

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b) \tag{37}$$

where $\mathbf{u}, \mathbf{w} \in \mathbb{R}^d$, both are column vectors, $b \in \mathbb{R}$, and h is an element-wise non-linearity such as tanh. Therefore, h is a scalar and f is a column vector of $\mathbf{z}$ shape. During the computation of derivative, we keep the result of numerator layout. Refer to matrix calculus for help.

$$\begin{aligned}
\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \frac{\partial \mathbf{z}}{\partial \mathbf{z}} + \frac{\partial \mathbf{u}h}{\partial \mathbf{z}} \\
&= \mathbf{I} + h\frac{\partial \mathbf{u}}{\partial \mathbf{z}} + \mathbf{u}\frac{\partial h}{\partial \mathbf{z}} && = \mathbf{I} + \mathbf{u}\frac{\partial h}{\partial \mathbf{z}} \\
&= \mathbf{I} + \mathbf{u}\frac{\partial h}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{z}} && = \mathbf{I} + \mathbf{u}\frac{\partial h}{\partial \mathbf{y}}\frac{\partial (\mathbf{w}^\top \mathbf{z} + b)}{\partial \mathbf{z}} \\
&= \mathbf{I} + \mathbf{u}h'(\frac{\partial \mathbf{w}^\top \mathbf{z}}{\partial \mathbf{z}} + \frac{\partial b}{\partial \mathbf{z}}) \\
&= \mathbf{I} + \mathbf{u}h'\mathbf{w}^\top \\
&= \mathbf{I} + \mathbf{u}\big(h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}\big)^\top
\end{aligned} \tag{38}$$

which can be computed in $O(d)$ time. Temporal variable $\mathbf{y} = \mathbf{w}^\top \mathbf{z} + b$ is used for simplify the process. Mark $h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}$ as $\psi(\mathbf{z})$. The Jacobian determinant is then given by

$$\begin{aligned}
&|\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}| = |\det \left(\mathbf{I} + \mathbf{u}\psi(\mathbf{z})^\top\right)| = |1 + \mathbf{u}^\top \psi(\mathbf{z})| \\
&\text{note, for any } \mathbf{uv},\ \det \left(\mathbf{I} + \mathbf{u}\mathbf{v}^T\right) = 1 + \mathbf{u}^T\mathbf{v}
\end{aligned} \tag{39}$$

Recall (33), with (39) we now have a quick method to compute the ELBO.

在实践中，参数的初始化非常重要参考讨论。seed=2,3 无法完成 u_z1分布的分裂，而初始化为 1 就可以。

Radial flows use functions of the form

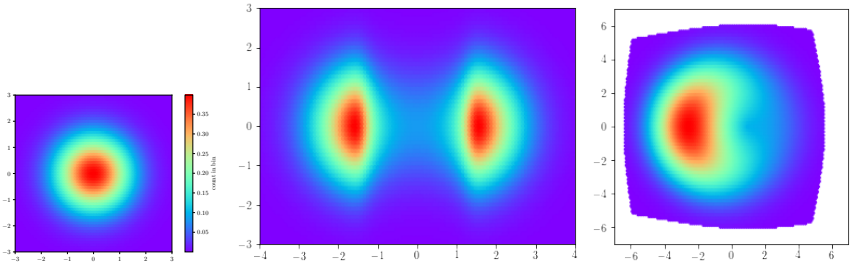
$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_0) \tag{40}$$

where $\alpha \in \mathbb{R}^+$, $\beta \in \mathbb{R}$, $h(\alpha, r) = (\alpha + r)^{-1}$ and $r = ||\mathbf{z} - \mathbf{z}_0||$.

The Jacobian determinant is then given by

$$|\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}| = [1 + \beta h(\alpha, r)]^{d-1}\,[1 + \beta h(\alpha, r) + \beta h'(\alpha, r)r] \tag{41}$$

Detailed computation is shown below. But not all functions of the form (37) or (40) will be invertible. [12] discuss the conditions for invertibility and how to satisfy them in a numerically stable way in the appendix. When using $h(x) = \tanh(x)$, a sufficient condition for $f(z)$ in (37) to be invertible is that $w^\top u \geq -1$.



(a) Initial Density (b) Planar Flow (c) Radial Flow

Figure 2: Change in standard normal density on application of length 1 planar and radial flows.

Fig. 2 shows how planar and radial flows change a standard normal density.

$$\begin{aligned}\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \mathbf{I} + \beta \left((\mathbf{z} - \mathbf{z}_0)\, h'(\alpha, r) \frac{\partial r}{\partial \mathbf{z}} + h(\alpha, r)\mathbf{I} \right) \\ &= (1 + \beta h(\alpha, r))\mathbf{I} + \beta h'(\alpha, r)\, (\mathbf{z} - \mathbf{z}_0)\, \frac{(\mathbf{z} - \mathbf{z}_0)^\top}{\|\mathbf{z} - \mathbf{z}_0\|}\end{aligned} \tag{42}$$

Let $\gamma = (1 + \beta h(\alpha, r))$. Using the matrix determinant lemma

$$\begin{aligned}\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \left(1 + \beta h'(\alpha, r) \frac{(\mathbf{z} - \mathbf{z}_0)^\top}{\|\mathbf{z} - \mathbf{z}_0\|} \frac{\mathbf{I}}{\gamma} (\mathbf{z} - \mathbf{z}_0) \right) \det(\gamma \mathbf{I}) \\ &= \left(\frac{1 + \beta h(\alpha, r) + \beta h'(\alpha, r)\, \|\mathbf{z} - \mathbf{z}_0\|}{(1 + \beta h(\alpha, r))} \right) (1 + \beta h(\alpha, r))^d \\ &= (1 + \beta h(\alpha, r) + \beta h'(\alpha, r) r)\, (1 + \beta h(\alpha, r))^{d-1}\end{aligned} \tag{43}$$

todo, 上面的全部都 check 过了。

4.1.2 Inverse Autoregressive Flow

Planar and radial flows provide a simple invertible transformation shown to be effective in a low-dimensional latent spaces (up to hundred dimensions). The transformation in planar flows (Eq. 37) can be seen as a Multilayer Perceptron (MLP) with a single-unit bottleneck hidden layer with a skip connection. Since a single-unit hidden layer isn't very expressive, a long chain of transformations is needed to model a high-dimensional distribution.

Autoregressive flows [4] is a normalizing flow that scales to high-dimensional latent space by exploiting the ordering f the variables. In autoregressive flow, given a sequence of variable $\mathbf{y} = y_{i_{i=0}}^{D}$, each variable is only dependent only on variables from the previous index. The distribution is then given by

$$p(\mathbf{y}) = \prod_{i=0}^{D} p(x_i | x_0, \ldots, x_{i-1}) \tag{44}$$

Kingma et. al. [7] proposed a Gaussian version of an autoregressive flow on a noise vector $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ given as follow:

$$y_0 = \mu_0 + \sigma_0 \epsilon_0 \tag{45}$$

$$y_i = \mu_i(\mathbf{y}_{0:i-1}) + \sigma(\mathbf{y}_{0:i-1})\epsilon_i \tag{46}$$

This flow is invertible and the noise ϵ is given by:

$$\epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{0:i-1})}{\sigma(\mathbf{y}_{0:i-1})} \tag{47}$$

Note that, *epsilon* is independent to each other so the calculation of Equation 47 can be vectorized as follow:

$$\boldsymbol{\epsilon} = \frac{\mathbf{y} - \boldsymbol{\mu}(\mathbf{y})}{\boldsymbol{\sigma}(\mathbf{y})} \tag{48}$$

This enables an efficient computation with GPU.

Due to the autoregressive structure, the transformation has a lower triangular Jacobian where diagonal is σ_i. For calculation of normalizing flows, we are interested in log-determinant of the Jacobian which is just a product of the diagonal given as:

$$\log \det \left| \frac{d\boldsymbol{\epsilon}}{d\mathbf{y}} \right| = \sum_{i=0}^{D} - \log \sigma_i(\mathbf{y}) \tag{49}$$

To apply Inverse Autoregressive Flows (IAF) for variational inference in VAE, we add IAF transforms after the latent variables $\mathbf{z}$ and modify the likelihood to account for IAF transforms. Figure 3 shows the process of applying IAF to Variational Autoencoder.

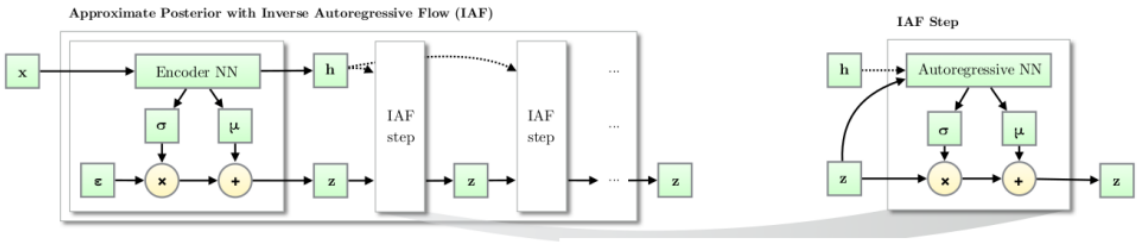


Figure 3: The process of Inverse Autoregressive Flows in Variational Autoencoder [7].

The flow consists of a chain of T following transformations:

$$\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \mathbf{z}_{t-1} \tag{50}$$

Kingma et.al. [7] proposed a more stable update based on LSTM-type update. LSTM is a type of recurrent neural network that applies autoregressive technique to the neural network. The update is given as follows:

$$\mathbf{s}_t = 1/\sigma_t \text{ and } \mathbf{m}_t = -\mu_t/\sigma_t \text{ and } \sigma_t = \frac{1}{1 + e^{-\mathbf{s}_t}} \text{ and } \mathbf{z}_t = \sigma_t \mathbf{z}_{t-1} + (1 - \sigma_t)\mathbf{m}_t \tag{51}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ learnt from an Autoregressive neural network given in [4] receiving input $\mathbf{z}$ and $\mathbf{h}$ from the VAE.

4.1.3 Sylvester Normalizing Flow

As explained earlier, planar flows suffer from the single-unit bottleneck problem. Sylvester Normalizing Flows (SNF) [13] attempt to solve this problem by modifying the transformation function which then behaves as an MLP with M units instead of 1. SNF uses a transformation function of the following form

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{A}h(\mathbf{B}\mathbf{z} + \mathbf{b}) \tag{52}$$

where $\mathbf{A} \in \mathbb{R}^{d\times m}, \mathbf{B} \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$ with $m \leq d$, and h is an element-wise non-linearity such as tanh. Using Sylvester's determinant identity, we can convert the computation of the determinant of a $d \times d$ matrix into the computation of the determinant of an $m \times m$ matrix.

$$\det(\mathbf{I}_d + \mathbf{A}\mathbf{B}) = \det(\mathbf{I}_m + \mathbf{B}\mathbf{A}) \tag{53}$$

Matrices $\mathbf{A}$ and $\mathbf{B}$ are further parameterized as $\mathbf{A} = \mathbf{Q}\mathbf{R}$ and $\mathbf{B} = \mathbf{R}\mathbf{Q}^\top$ where $\mathbf{R}$ and $\mathbf{R}$ are $m \times m$ upper-triangular matrices and $\mathbf{Q}$ is $d\times m$ matrix with orthonormal column vectors. The determinant of the Jacobian can then be written as

$$\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = \det\left(\mathbf{I}_m + \operatorname{diag}\left(h'(\mathbf{R}\mathbf{Q}^\top\mathbf{z} + \mathbf{b})\right)\mathbf{R}\mathbf{R}\right) \tag{54}$$

which can be computed in $O(m)$ time. Please refer to the original paper [13] for details.

4.2 Density Estimation

Density Estimation techniques take a different approach from Variational Inference (VI) methods to model the complex data distribution. Unlike VI, these methods aim for exact inference, sampling and log-likelihood evaluation. The primary goal in this regime is to find a bijective function $h = f(x)$, $x \in \mathcal{X}$ to map complex data-distribution $p_X(x)$ to density $p_H(f(x))$. Given that $p_H()$ has a simpler density whose likelihood function is analytically known, the overall log-likelihood of the data can be easily calculated. The complex log-likelihood of the data can now be calculated using the change of variables as follows:

$$\log p_X(x) = \log p_H(f(x)) + \log\left|\det\frac{\partial f(x)}{\partial x}\right| \tag{55}$$

4.2.1 Non-linear Independent Components Estimation

Non-linear Independent Components Estimation (NICE) [2] is one of the early works adopting normalizing flows in density estimation. This work focuses on transformations $h = f(x)$ that maps the data into a factorized distribution, i.e., the components of h_d are independent. Consequently, the log-likelihood in eq. (55) can be written as:

$$\log(p_X(x)) = \left[\sum_{d=1}^{D}\log p_{H_d}(f_d(x))\right] + \log\left|\det\frac{\partial f(x)}{\partial x}\right| \tag{56}$$

where $f(x) = (f_d(x))_{d\leq D}$.

This work targets invertible functions whose Jacobians have triangular structure so that calculating the determinant is tractable. In particular, it proposes the family of *coupling layers* that we define below.

Coupling layer: The coupling layer serves a building block of the transformation proposed in this work. The general coupling layer comprises of two partitions I_1, I_2 of the input dimensions $[1, D]$, such that $d = |I_1|$. The transformation is then defined as:

$$y_{I_1} = x_{I_1} \tag{57}$$

$$y_{I_2} = g(x_{I_2}; m(x_{I_1})) \tag{58}$$

where $g : \mathbb{R}^{D-d} \times m(\mathbb{R}^d) \to \mathbb{R}^{D-d}$ is an invertible function. Considering $I_1 = [1, d]$ and $I_2 = [d, D]$, the Jacobian of this function is:

$$\frac{\partial y}{\partial x} = \begin{bmatrix} I_d & 0 \\ \frac{\partial y_{I_2}}{\partial x_{I_1}} & \frac{\partial y_{I_2}}{\partial x_{I_2}} \end{bmatrix}$$

Where I_d is the identity matrix of size d. That means that $\det \frac{\partial y}{\partial x} = \det \frac{\partial y_{I_2}}{\partial x_{I_2}}$ which evaluates to 1. The inverse of this transformation can be expressed as $x_{I_1} = y_{I_1}$ and $x_{I_2} = g^{-1}(y_{I_2}; m(y_{I_1}))$:

It is important to notice that the inverse of coupling function $m(.)$ is not required, thus allowing it to be modeled as complex non-linear functions. NICE adopts an *additive coupling law* which defines the function $g(.)$ as $g(a, b) = a + b$. Also, reLU is chosen as the coupling function $m(.)$.

A single coupling layer leaves part of the input unchanged. This is problematic, since modification of every dimension is desired. To achieve this, roles of partitions are interachanged in arjacent layers to ensure proper mixing.

4.2.2 Real-valued Non-Volume Preserving

This subsequent work (RealNVP) [3] is an extension to the previous work (NICE) to enable multi-scalable architecture. The model uses *affine coupling law* as its primary constituent. This is defined as:

$$y_{1:d} = x_{1:d} \tag{59}$$

$$y_{d+1:D} = x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}) \tag{60}$$

where $I_1 = [1, d]$ and $I_2 = [(d + 1), D]$. Also, $\exp(s(.))$ represents the *scale* and $t(.)$ represents the *shift* operation respectively. The Jacobian of this transformation is also a triangular matrix (derivation has been provided in appendix).

Apart from the affine transformation, the model also proposes partitioning schemes across channels of input images and checkerboard pattern-based partitioning for exploiting pixel correlations. Further, to enable deep networks, the work proposes using techiniques such as Gaussianization and invertible *batch-normalization*. It achieves to get competitive scores against state-of-the variants, with some advantages that are discussed in the appendix.

4.2.3 Masked Autoregressive Flow

The section on Inverse Autoregressive Flow (IAF) has already described how Autoregressive models such as MADE [4] can be used as normalizing flows. Masked Autoregressive Flow (MAF) [11] uses the transformation function in Eq. (46) contrary to IAF which uses the inverse function. Since the inverse function can be parallelized, MAF is suitable for computation of the density $p(\mathbf{x})$ of an externally provided data point $\mathbf{x}$. Sampling a new point from MAF requires D sequential passes which cannot be parallelized and hence sampling is slow. However, in the case of IAF, as the inverse function is used, sampling is fast but density estimation is slow and requires D sequential passes.

5 NF in Probabilistic Programming Languages

Normalizing flows have been implemented in two recent deep probabilistic programming languages Pyro (based on Pytorch) and Tensorflow Probability (TFP, based on Tensorflow). They provide a construct called the `Bijector` for the implementation of an invertible transformation required for construction of a normalizing flow. New transformation functions can be defined by extending the `Bijector` class. TFP also provides a construct called the `TransformedDistribution` which takes in a base distribution (which is generally a simple distribution) and a `Bijector`. A `TransformedDistribution` represents the distribution obtained after applying the `Bijector` on the base distribution. A number of functions commonly used in deep learning (e.g., Affine, Sigmoid, RealNVP) have been implemented as `Bijectors` in TFP. Another bijector called `Chain` is provided in TFP that converts a list of bijectors into a single bijector which represents a flow. The complete documentation of normalizing flows in TFP is given in [1].

Defining a custom transformation requires extending the `Bijector` class and implementing three functions: 1) `_forward` – receives samples x from the base distribution as an input, evaluates the function $y = f(x)$, and returns the variable y. 2) `_inverse` – receives y as the input and outputs x corresponding to the inverse of the function, i.e., $x = f^{-1}(y)$. 3) `_inverse_log_det_jacobian` – receives y as the input and computes the inverse log-determinant of the Jacobian, i.e., $\log\det\left|\frac{\partial f^{-1}}{\partial y}\right|$.

Let's define the three functions for an example function, the Parametrized ReLU (PReLU) function.

$$y = f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise, where } \alpha \in [0, 1] \end{cases} \tag{61}$$

$$x = f^{-1}(y) = \begin{cases} y & \text{if } y \geq 0 \\ \alpha^{-1} y & \text{otherwise} \end{cases} \tag{62}$$

$$\left.\frac{\partial f^{-1}}{\partial y}\right|_{ii} = \begin{cases} 1 & \text{if } y_i \geq 0 \\ \frac{1}{\alpha} & \text{otherwise} \end{cases} \tag{63}$$

An example bijector implementation for the PReLU function is given in the appendix.

6 Recent Advances

In this section, we discuss several recent works that used normalizing flows. We only explain the higher-level concept of each work. Interested readers should refer to the original paper given in the reference.

6.1 Pixel Recurrent Neural Network

Pixel Recurrent Neural Network (PixelRNN) [10] is an auto-regressive image generative model where the joint distribution over the image pixel is factorized into a product of conditional distribution. This means that the probability of pixel at position i is given as: $p(x_i|x_1, \ldots, x_{i-1})$ where $x_1, \ldots, x_{i-1}$ is the previously generated pixels. This is a strong and counter-intuitive assumption for an image generation where pixel mostly conditioned on their neighbors. However, PixelRNN is

proven to work well for image completion and generation task. In image completion, the occluded pixel is generated by conditioned upon the non-occluded pixels.

Oord et. al. [10] proposed three methods model an auto-regressive image generation. First, Row LSTM 1D convolution is used to generate image row by row from top to bottom. However, Row LSTM cannot capture the whole previously generated pixels since there is a coarse-graining in the convolution method. Second, Bidirectional LSTM is used to capture all the generated pixels as the context. Every pixel is conditioned upon their neighboring pixels except the one that has not been generated. However, LSTM training is known to be expensive so the authors proposed another model based on CNN. Third, Pixel CNN used a masked convolution by setting the filter of the pixels that has not been generated as zero.

6.2 Wavenet

Wavenet [14] used the idea of Pixel CNN from [10] for raw audio generation. It used one-dimensional convolution Pixel CNN to generate raw audio. Wavenet has been proven to be the state-of-the-art model for text-to-speech model. Google Assistant in Android use wavenet to generate the audio of the assistant.

Similar to Pixel CNN, the joint probability of an input audio is modelled by a stack of causal convolutional layers. Causal convolution has similar idea to masked convolution by shifting the output of a normal convolution by a few timesteps so it does not violate the autoregressive requirement. One of the problems with causal convolution is that it requires deep layers to capture long range dependency. The authors proposed to use dilated convolution where the convolution filter is applied over an area larger than its length. This is done by skipping input values with a certain step.

6.3 Glow

Glow [6] extended and simplified the NICE and RealNVP model explained in Section 4.2.2 with three main extensions. First, the authors used activation normalization (act-norm) instead of batch normalization in RealNVP. Act-norm performs a channel-wise normalization and is faster than batch normalization. Second, the authors used 1×1 convolution for the channel-wise permutation. NICE and RealNVP proposed a flow containing the equivalent of a permutation that reverses the ordering of the channels. This is changed with a 1×1 convolution so that the permutation can be learned. Third, they extend RealNVP so that it can work faster by changing it into a multi-scale architecture. The authors splitted each step of the flow so it can be parallelized.

All of the additions mentioned before can be inverted and log-determinant of the Jacobian can be computed easily. Please refer to the paper for the details of the inversion and calculation of the Jacobian matrix. Glow has similar advantage as RealNVP that it can generate a high quality images. Moreover, the latent space learned by Glow is meaningful. This means that we can control the output of the generated image by tuning the latent space.

7 Conclusion

In this project, we have discussed how to transform a simple base distribution into a complex distribution by applying a series of invertible transformations called normalizing flows. We also discussed how normalizing flows can be applied to various representation learning regimes. Firstly,

normalizing flows can be used for richer latent-posterior proposals for inference in the Variational Autoencoder. We discussed Planar and Radial Flows [12], Inverse Autoregressive Flow [7], and Sylvester Normalizing Flow [13] in this regime. Secondly, normalizing flows can be used to estimate density when the exact likelihood is not known. We discussed NICE [2], RealNVP [3], and Masked Autoregressive Flow [11] in this regime. We also discussed how normalizing flows can be implemented in deep probabilistic programming languages specifically Tensorflow Probability. Finally, we discussed several recent advances in normalizing flows used for image and audio generation.

In present day trends, applications of normalizing flows are being heavily used in methods involving variation inference. These are scalable methods which provide robust generative solutions. In contrast, auto-regressive methods suffer from slow sampling as the forward calculation of an autoregressive flow is not parallelizable. Density estimation methods ameliorate problems present in the mentioned methods, however their popularity remains constrained for historical reasons. Nevertheless, benchmark performance by models such as Glow seem to reverse this trend. From the PPL perspective, Tensorflow Probability provides the highest range of bijector families, thus making it the ideal choice for flow-based implementations.

Finally, 提供参考的 blog

1. Adam Kosiorek's Blog
2. Jianlin Su's Blog
3. Eric Jang's Blog
4. Brian Keng's Blog

8 Missing Flow

8.1 Challenges

Normalizing flows(NF) have richer posterior than normal VAE frames, eg. HI-VAE [9], which improvement is the basical advantage of NF. For details, refer to (35) and (15).

$$
\begin{aligned}
ELBO &= \log p(x) - KL(Q_\phi(Z|X)||P(Z|X)) \\
ELBO &= \mathbb{E}_{q_0(\mathbf{z_0}|\mathbf{x})}\left[\log p(\mathbf{x}|\mathbf{z}_K)\right] - KL(q_0(\mathbf{z}_0|\mathbf{x})\|p(\mathbf{z}_K)) + \mathbb{E}_{q_0(\mathbf{z_0}|\mathbf{x})}\left[\sum_{k=1}^{K} \log|\det|\right]
\end{aligned} \tag{64}
$$

1. 更好地建立不同 samples 之间的联系
2. 能不能指定性地进行生成，生成单个元素肯定比生成多个准确（proof）
3. 如何判定生成的质量
4. 能否不假设先验是正态分布? 先用采样的方式逼近一次? 不仅在学习参数, 还在同时修改分布, 将先验的参数也设定成可学习的值 (proof: 其不会无限歪曲，而是会收敛)

8.2 Improvement

Make some contributions.

1. 改进 HI-VAE [9] 生成的过程，(先深入理解为什么输出不能直接依赖于周围的值)，引入 MADE [4] 和 MAF [11] 的 mask 概念，使得条件分布能更好地得到表示。
2. 相比 HI-VAE，加强对邻居的重视程度，而不是对全部的点都一视同仁。(参考 GAIN) 具体实施表现在
   (a) 首先选取邻居 set。基于假设，邻居的相似度在不同的变换中会得到保留。nf 的每次变换都是有输出的，使得每一次的 flow 后，都有一个 loss，来保持隐藏空间中的邻居的相似度。
   (b) 将全部点生成的对应参数结果和邻居生成结果进行结合. Parameters in every flow are computed by former layer.
   (c) 用三个 flow 来从三个角度增强学习力度, 分别是 user，item，interaction。三种邻居的集合
   (d) 用 boost 来替换平均的输出
3. 想想其他的提升，用 flow 针对性地对 imputation 进行优化，最好有公式

其他调优

1. 对预测错误的惩罚加大，尤其是分类问题，保证预测的结果有更高的 confidence。提供在 confidence 和 missing imputation 补全率之间的权衡的方案。

8.3 todo list

1. 选择一个 nf 基本模型来带入 HI-VAE 中进行初步实验。

References

[1] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[2] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[4] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

[5] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[6] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[7] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Alfredo Nazábal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using VAEs. *Pattern Recognition*, 107, 2020.

[10] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[11] George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[12] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

[13] Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

[14] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.