Flow-based Deep Generative Models

Jiarui Xu
University of California San Diego
jix026@ucsd.edu

Hao-Wen Dong
University of California San Diego
hwdong@ucsd.edu

Abstract

In this report, we investigate the flow-based deep generative models. We first compare different generative models, especially generative adversarial networks (GANs), variational autoencoders (VAEs) and flow-based generative models. We then survey different normalizing flow models, including non-linear independent components estimation (NICE), real-valued non-volume preserving (RealNVP) transformations, generative flow with invertible 1×1 convolutions (Glow), masked autoregressive flow (MAF) and inverse autoregressive flow (IAF). Finally, we conduct experiments on generating MNIST handwritten digits using NICE and RealNVP to examine the effectiveness of flow-based models. Source code is available at https://github.com/salu133445/flows.

1 Background

Recent year have witnessed a progress on deep generative models. Figure 1 shows the taxonomy of these models [1]. On the left branch of this taxonomic tree, an explicit likelihood can be maximized by constructing an explicit density. However, the density may be computationally intractable in some cases, where we have to approximate the density with variational methods or Markov chain Monte Carlo (MCMC). On the right branch of the tree, the model does not explicitly represent a probability distribution over the space where the data lies. Instead, the model provides some way of interacting less directly with the data distribution.

In the following sections, we will briefly introduce generative adversarial networks (GANs) and variational autoencoder (VAEs) and compare them with the flow-based models.

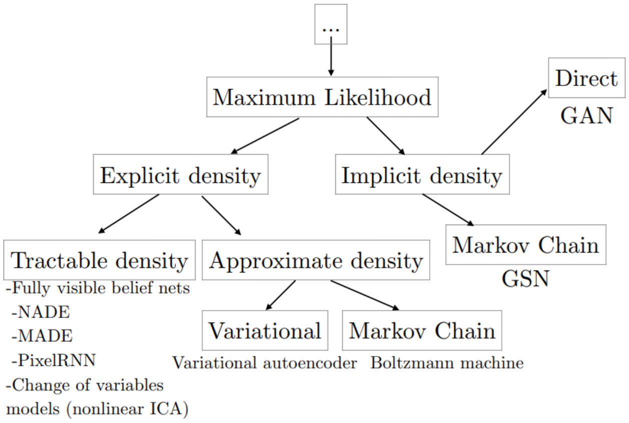


Figure 1: Taxonomy of generative models (Source: [1])

Final project for “Probabilistic Approaches to Unsupervised Learning” (UCSD CSE 291, Fall 2020)

1.1 Generative adversarial networks (GANs)

Generative adversarial networks (GANs) [2] have demonstrated great success in different fields in replicating real-world, rich content such as images, human language, and music. Originally inspired by game theory, a GAN is essentially a game between two players—a generator and a discriminator. The game goes as follows.

- The discriminator D estimates the probability of a given sample coming from the real data distribution. It works as a critic and is optimized to tell the fake samples from the real ones.
- The generator G outputs synthetic samples given a noise variable input. It is trained to capture the real data distribution so that the generated samples are authentic enough to trick the discriminator into assigning them with high probability of being real samples.

The two models compete against each other during the training—the generator aims at fooling the discriminator, whereas the discriminator aims not to be cheated. The zero-sum game between them motivates both to improve their performance. Figure 2 illustrates the pipeline of a GAN.

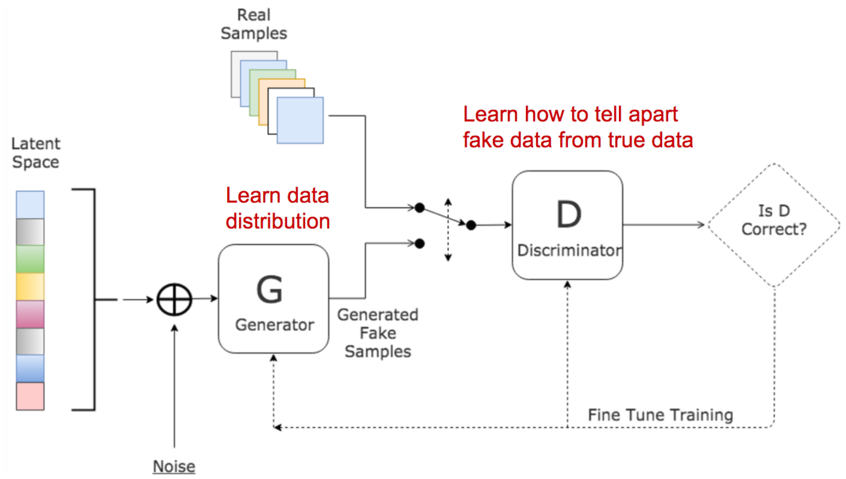


Figure 2: Pipeline of a GAN (Source: [3])

Formally, we have the data distribution $p_{\text{data}}(x)$ and a prior distribution $p(1)$ on the latent variable z (usually modeled by a uniform distribution). Moreover, we have a generator G with parameter θ_g and a discriminator D with parameter θ_d.

Then, the objective function for the discriminator is

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D\left(G(z)\right)\right)\right] ,$$

which encourages the discriminator to distinguish between real and fake data. The objective function for the generator is

$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log\left(1 - D\left(G(z)\right)\right) ,$$

which pushes the generator to try to fool the discriminator. Finally, combining the two objectives gives a *minimax game* bewteen the generator G and the discriminator D:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\sim p(z)} \log\left(1 - D\left(G(z)\right)\right)\right] .$$

1.2 Variational autoencoders (VAEs)

The idea of variational autoencoder (VAE) [4] is different from a traditional autoencoder. Instead of mapping the input into a fixed vector, VAE map it into a distribution. We sample a z from some prior distribution $p_\theta(z)$. Then, x is generated from a conditional distribution $p_\theta(x \mid z)$. Mathematically, the process can be modeled as

$$p_\theta\left(\mathbf{x}\right) = \int p_\theta\left(\mathbf{x} \mid \mathbf{z}\right) p_\theta(\mathbf{z}) d\mathbf{z}$$

However, it is computationally intractable to check all z for the integral. To narrow down the value space, consider the posterior $p_\theta(z \mid x)$ and approximate it by an inference model $q_\phi(z \mid x)$. Then, we can compute the data likelihood as follows.

$$\begin{aligned}
&\log p_\theta(x) \\
&\quad = \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x\right)\right] \\
&\quad = \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log \frac{p_\theta\left(x \mid z\right) p_\theta(z)}{p_\theta\left(z \mid x\right)}\right] \\
&\quad = \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log \frac{p_\theta\left(x \mid z\right) p_\theta(z)}{p_\theta\left(z \mid x\right)} \frac{q_\phi\left(z \mid x\right)}{q_\phi\left(z \mid x\right)}\right] \\
&\quad = \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x \mid z\right)\right] - \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log \frac{q_\phi\left(z \mid x\right)}{p_\theta(z)}\right] + \mathbf{E}_{z \sim q_\phi(z|x)} \left[\log \frac{q_\phi\left(z \mid x\right)}{p_\theta\left(z \mid x\right)}\right] \\
&\quad = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x \mid z\right)\right] - D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta(z)\right) + D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta\left(z \mid x\right)\right)
\end{aligned}$$

Now, a VAE adopts the variational method to obtain the evidence lower bound (ELBO) of the data likelihood.

$$\begin{aligned}
&\log p_\theta(x) \\
&\quad = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x \mid z\right)\right] - D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta(z)\right) + D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta\left(z \mid x\right)\right) \\
&\quad \geq \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x \mid z\right)\right] - D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta(z)\right) \\
&\quad = \mathrm{ELBO}(x; \theta, \phi)
\end{aligned}$$

Since $\mathrm{ELBO}(x; \theta, \phi)$ is tractable, we maximize the ELBO of the data likelihood rather than maximizing the exact data likelihood. That is, the objective of a VAE is

$$\max_{\theta, \phi} \mathrm{ELBO}(x; \theta, \phi) .$$

Figure 3 illustrates the pipeline of a VAE.

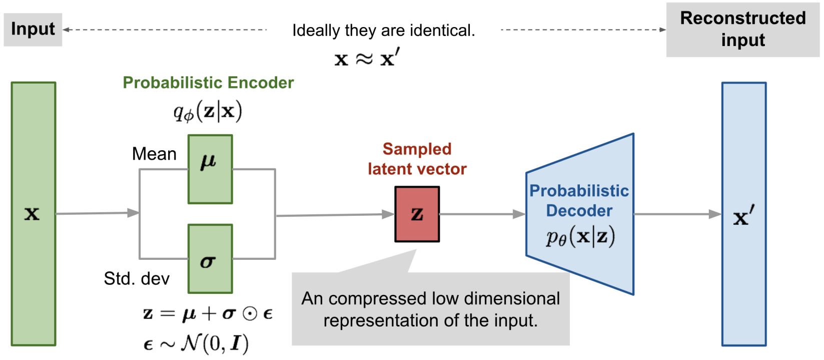

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
$\mathbf{x}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure 3: Pipeline of a VAE (Source: [5])

1.3 Comparisons of GANs, VAEs and flow-based models

Mathematically, the objective functions for the three types of models are

(GANs) $$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

(VAEs) $$\max_{\theta, \phi} \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta\left(x \mid z\right)\right] - D_{KL}\left(q_\phi\left(z \mid x\right) \| p_\theta(z)\right)$$

(flow-based models) $$\max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} \log p_\theta(x)$$

Below we summarize the differences between GANS, VAEs and flow-based models.

- Generative adversarial networks: GAN provides a smart solution to model the data generation, an unsupervised learning problem, as a supervised one. The discriminator model learns to distinguish the real data from the fake samples that are produced by the generator model. Two models are trained as they are playing a minimax game.
- Variational autoencoders: VAE implicitly optimizes the log-likelihood of the data by maximizing the evidence lower bound (ELBO).
- Flow-based generative models: A flow-based generative model is constructed by a sequence of invertible transformations. Unlike other two, the model explicitly learns the data distribution $p(\mathbf{x})$ and therefore the loss function is simply the negative log-likelihood. Please refer to Section 3 for detail.

Figure 4 illustrates their computation graphs.

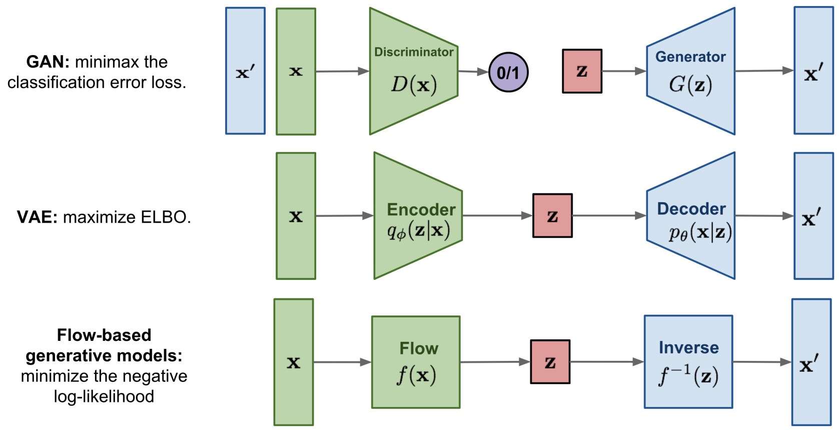


Figure 4: Difference between GAN, VAE and FLOW based model (Source: [6])

2 Linear Algebra Basics

2.1 Jacobian matrix

Given a function of mapping a n-dimensional input vector $\mathbf{x}$ to a m-dimensional output vector, $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$, the matrix of all first-order partial derivatives of this function is called the Jacobian matrix $\mathbf{J}$, where one entry on the i-th row and j-th column is $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

2.2 Change of variable theorem

Given some random variable $z \sim \pi(z)$ and a invertible mapping $x = f(z)$ (i.e., $z = f^{-1}(x) = g(x)$). Then, the distribution of x is

$$p(x) = \pi(z) \left| \frac{dz}{dx} \right| = \pi(g(x)) \left| \frac{dg}{dx} \right| .$$

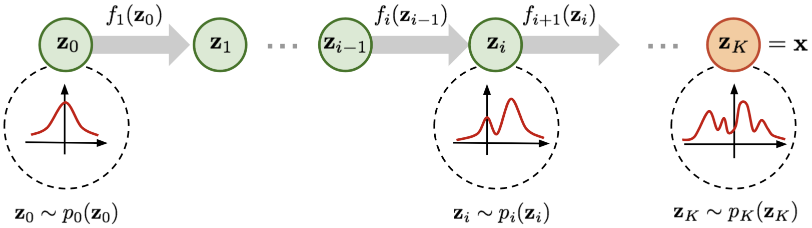
The multivariate version takes the following form:

$$p(\mathbf{x}) = \pi(\mathbf{z}) \left| \det \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \pi(g(\mathbf{x})) \left| \det \frac{dg}{d\mathbf{x}} \right| ,$$

where $\det \frac{dg}{d\mathbf{x}}$ is the *Jacobian determinant* of g.

3 Normalizing Flows

Figure 5 illustrates the core idea behind normalizing flows [6,7]. In each step, we substitute the variable with the new one by change of variables theorem. Eventually, we hope the final distribution we obtain is close enough to the target distribution.



$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$\mathbf{z}_{i-1}$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_i$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Figure 5: Transform a simple distribution into a complex one by applying a sequence of *invertible transformations* (Source: [6])

Mathematically, we have, for each step, $\mathbf{z}_i \sim p_i(\mathbf{z}_i)$, $\mathbf{z}_i = f_i(\mathbf{z}_{i-1})$ and $\mathbf{z}_{i-1} = g_i(\mathbf{z}_i)$. Next, we hope to represent $p_i(\mathbf{z}_i)$ in terms of $p_{i-1}(\mathbf{z}_{i-1})$ and $\mathbf{z}_{i-1}$:

$$\begin{aligned}
p_i(\mathbf{z}_i) &= p_{i-1}(g_i(\mathbf{z}_i)) \left|\det \frac{dg_i(\mathbf{z}_i)}{d\mathbf{z}_i}\right| && \text{(by change of variables theorem)} \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \frac{d\mathbf{z}_{i-1}}{df_i(\mathbf{z}_{i-1})}\right| && \text{(by definition)} \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \left(\frac{df_i(\mathbf{z}_{i-1})}{d\mathbf{z}_{i-1}}\right)^{-1}\right| && \text{(by inverse function theorem)} \\
&= p_{i-1}(\mathbf{z}_{i-1}) \left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|^{-1}. && \text{(by } \det M \det(M^{-1}) = \det I = 1\text{)}
\end{aligned}$$

Taking logarithm of both sides, we obtain $\log p_i(\mathbf{z}_i) = \log p_{i-1}(\mathbf{z}_{i-1}) - \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right|$. Now, recall that $\mathbf{x} = \mathbf{z}_K = f_K \circ f_{K-1} \dots f_1(\mathbf{z_0})$. Thus, we can compute the exact log-likelihood $\log p(\mathbf{x})$ of input data x as follows.

$$\begin{aligned}
\log p(\mathbf{x}) &= \log p_K(\mathbf{z}_K) \\
&= \log p_{K-1}(\mathbf{z}_{K-1}) - \log\left|\det \frac{df_K}{d\mathbf{z}_{K-1}}\right| \\
&= \dots \\
&= \log p_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log\left|\det \frac{df_i}{d\mathbf{z}_{i-1}}\right| .
\end{aligned}$$

To make the computation tractable, it requires

- f_i is easily invertible
- The Jacobian determinant of f_i is easy to compute

Finally, we can train the model by maximizing the log-likelihood over some training dataset $\mathcal{D}$:

$$LL(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}) .$$

In the following sections, we will introduce three representative normalizing flow models—non-linear independent components estimation (NICE) [8], real-valued non-volume preserving (RealNVP) [9] and Glow [10].

3.1 Non-linear independent components estimation (NICE)

The core idea behind non-linear independent components estimation (NICE) [8] is as follows.

1. Split $\mathbf{x} \in \mathbb{R}^D$ into two blocks $\mathbf{x}_1 \in \mathbb{R}^d$ and $\mathbf{x}_2 \in \mathbb{R}^{D-d}$.
2. Apply the following transformation from $(\mathbf{x}_1, \mathbf{x}_2)$ to $(\mathbf{y}_1, \mathbf{y}_2)$:

$$\begin{cases} \mathbf{y}_1 & = \mathbf{x}_1 \\ \mathbf{y}_2 & = \mathbf{x}_2 + m(\mathbf{x}_1) \end{cases},$$

   where $m(\cdot)$ is an arbitrarily function (e.g., a deep neural network).

The transformation is called an *additive coupling layer*. It satisfies the requirements described above.

- First, it is trivially invertible as

$$\begin{cases} \mathbf{x}_1 & = \mathbf{y}_1 \\ \mathbf{x}_2 & = \mathbf{y}_2 - m(\mathbf{y}_1) \end{cases}.$$

- Second, its Jacobian matrix is

$$\mathbf{J} = \begin{bmatrix} \mathbf{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial m(\mathbf{x}_1)}{\partial \mathbf{x}_1} & \mathbf{I}_{D-d} \end{bmatrix},$$

  which has a unit Jacobian determinant $\det(\mathbf{J}) = \mathbf{I}$. Note that NICE is a type of *volume-preserving flows* as it has a unit Jacobian determinant.

However, some dimensions remain unchanged after the transform. Thus, we need to alternate the dimensions being modified in each step, as illustrated in Figure 6. Note that three coupling layers are necessary to allow all dimensions to influence one another [8].

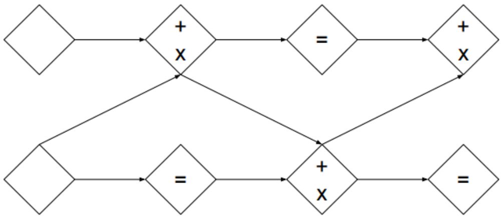


Figure 6: Alternating Pattern (Source: [8])

3.2 Real-valued non-volume preserving (RealNVP) transformations

The core idea behind real-valued non-volume preserving (RealNVP) [9] is as follows.

1. Split $\mathbf{x} \in \mathbb{R}^D$ into two blocks $\mathbf{x}_1 \in \mathbb{R}^d$ and $\mathbf{x}_2 \in \mathbb{R}^{D-d}$
2. Apply the following transformation from $(\mathbf{x}_1, \mathbf{x}_2)$ to $(\mathbf{y}_1, \mathbf{y}_2)$

$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} \odot e^{s(\mathbf{x}_{1:d})} + t(\mathbf{x}_{1:d}) \end{cases},$$

   where $s(\cdot)$ and $t(\cdot)$ are *scale* and *translation* functions that map $\mathbb{R}^d$ to $\mathbb{R}^{D-d}$, and $\odot$ denotes the element-wise product. Note that NICE does not have the scaling term.

The transformation is called an *affine coupling layer*. It satisfies the requirements described above.

- First, it is easily invertible as

$$\begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = (\mathbf{y}_{d+1:D} - t(\mathbf{x}_{1:d})) \odot e^{-s(\mathbf{x}_{1:d})} \end{cases} .$$

  Note that the above computation does not involve computing s^{-1} and t^{-1}.

- Second, its Jacobian matrix is

$$\mathbf{J} = \begin{bmatrix} \mathbf{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \text{diag}\left(e^{s(\mathbf{x}_{1:d})}\right) \end{bmatrix} ,$$

  which has a Jacobian determinant that is easy to compute

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} e^{s(\mathbf{x}_{1:d})_j} = \exp\left(\sum_{j=1}^{D-d} s(\mathbf{x}_{1:d})_j\right) .$$

  Note that the above computation does not involve computing the Jacobian of s and t.

3.3 Generative flow with invertible 1×1 convolutions (Glow)

Glow [10] further extends RealNVP with invertible 1×1 convolutions to improve the modeling capability. Each step in Glow consists of the following operations (see Figure 7).

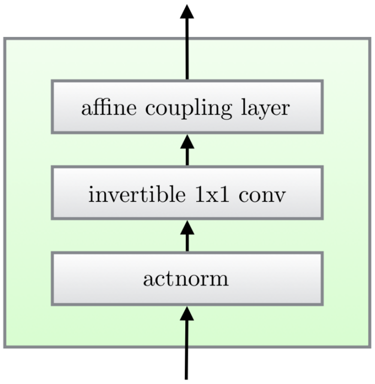


Figure 7: A step of the Glow model (Source: [10])

- **Actnorm**:
  - Forward: $\mathbf{y} = \mathbf{s} \odot \mathbf{x} + \mathbf{b}$
  - Backward: $\mathbf{x} = \mathbf{s} \odot (\mathbf{y} - \mathbf{b})$
  - Log-determinant: $h \cdot w \cdot \sum_i \log|\mathbf{s}_i|$
- **Invertible 1×1 convolution**:
  - Forward: $\mathbf{y} = \mathbf{W}\mathbf{x}$
  - Backward: $\mathbf{x} = \mathbf{W}^{-1}\mathbf{y}$
  - Log-determinant: $h \cdot w \cdot \log|\det \mathbf{W}|$
- **Affine coupling Layer**: same as RealNVP

Figure 8 shows some examples generated by the Glow model on CelebA-HQ dataset [11] with different temperatures.

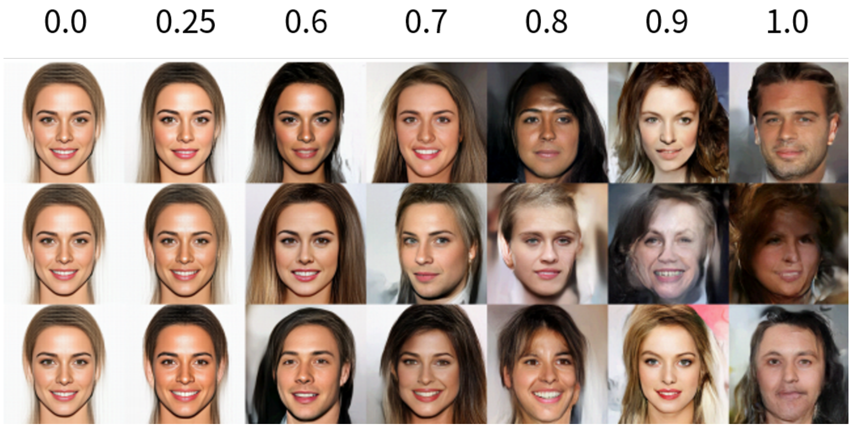


Figure 8: Samples of the Glow model with different temperatures (Source: [10])

4 Autoregressive Flows

The key idea behind an autoregressive flow is to model the transformation in a normalizing flow as an *autoregressive model*. In an autoregressive model, we assume that *the current output depends only on the data observed in the past* and factorize the joint probability $p(x_1, x_2, \dots, x_D)$ into the product of the probability of observing 'x_i' conditioned on the past observations $x_1, x_2, \dots, x_{i-1}$.

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, x_2, \dots, x_D) \\ &= p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_1, x_2) \ \dots\ p(x_D \mid x_1, x_2, \dots, x_{D-1}) \\ &= \prod_{i=1}^{D} p(x_i \mid x_1, x_2, \dots, x_{i-1}) \\ &= \prod_{i=1}^{D} p(x_i \mid \mathbf{x}_{1:i-1})\,. \end{aligned}$$

In the following sections, we will introduce two representative autoregressive flow models—masked autoregressive flow (MAF) [12] and inverse autoregressive flow (IAF) [13].

4.1 Masked autoregressive flow (MAF)

Given two random variables $\mathbf{z} \sim \pi(\mathbf{z})$ and $\mathbf{x} \sim p(\mathbf{x})$ where $\pi(\mathbf{z})$ is known but $p(\mathbf{x})$ is unknown. Masked autoregressive flow (MAF) aims to learn $p(x)$. To sample from the model, we have

$$x_i \sim p(x_i|\mathbf{x}_{1:i-1}) = z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1})\,.$$

Note that this computation is slow as it is sequential and autoregressive to generate the whole sequence $\mathbf{x}$. To estimate the density of a sample $\mathbf{x}$, we have

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i \mid \mathbf{x}_{1:i-1})\,.$$

Note that this computation can be fast if we use the *masking* approach introduced in MADE [14] as it only requires one single pass to the network.

4.2 Inverse autoregressive flow (IAF)

In MAF, we have $x_i = z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1})$. We can reverse it into

$$z_i = x_i \odot \frac{1}{\sigma_i(\mathbf{x}_{1:i-1})} - \frac{\mu_i(\mathbf{x}_{1:i-1})}{\sigma_i(\mathbf{x}_{1:i-1})}\,.$$

Now, if we swap $\mathbf{x}$ and $\mathbf{z}$ by letting $\tilde{\mathbf{z}} = \mathbf{x}$ and $\tilde{\mathbf{x}} = \mathbf{z}$, we get the inverse autoregressive flow (IAF)

$$\begin{aligned}\tilde{x}_i &= \tilde{z}_i \odot \frac{1}{\sigma_i(\tilde{\mathbf{z}}_{1:i-1})} - \frac{\mu_i(\tilde{\mathbf{z}}_{1:i-1})}{\sigma_i(\tilde{\mathbf{z}}_{1:i-1})} \\ &= \tilde{z}_i \odot \tilde{\sigma}_i(\tilde{\mathbf{z}}_{1:i-1}) + \tilde{\mu}_i(\tilde{\mathbf{z}}_{1:i-1}) ,\end{aligned}$$

where

$$\tilde{\sigma}_i(\tilde{\mathbf{z}}_{1:i-1}) = \frac{1}{\sigma_i(\tilde{\mathbf{z}}_{1:i-1})}, \quad \tilde{\mu}_i(\tilde{\mathbf{z}}_{1:i-1}) = -\frac{\mu_i(\tilde{\mathbf{z}}_{1:i-1})}{\sigma_i(\tilde{\mathbf{z}}_{1:i-1})} .$$

Figure 9 illustrates how MAF and IAF work differently.

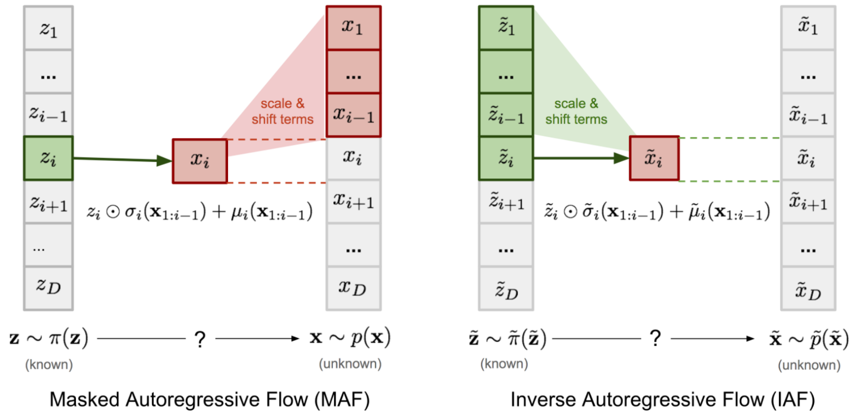

z_1
...
z_{i-1}
z_i
z_{i+1}
...
z_D
x_i
scale & shift terms
x_1
...
x_{i-1}
x_i
x_{i+1}
...
x_D
$z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1})$
$\mathbf{z} \sim \pi(\mathbf{z})$
(known)
?
$\mathbf{x} \sim p(\mathbf{x})$
(unknown)
Masked Autoregressive Flow (MAF)
$\tilde{z}_1$
...
$\tilde{z}_{i-1}$
$\tilde{z}_i$
$\tilde{z}_{i+1}$
...
$\tilde{z}_D$
scale & shift terms
$\tilde{x}_i$
$\tilde{x}_1$
...
$\tilde{x}_{i-1}$
$\tilde{x}_i$
$\tilde{x}_{i+1}$
...
$\tilde{x}_D$
$\tilde{z}_i \odot \tilde{\sigma}_i(\mathbf{x}_{1:i-1}) + \tilde{\mu}_i(\mathbf{x}_{1:i-1})$
$\tilde{\mathbf{z}} \sim \tilde{\pi}(\tilde{\mathbf{z}})$
(known)
?
$\tilde{\mathbf{x}} \sim \tilde{p}(\tilde{\mathbf{x}})$
(unknown)
Inverse Autoregressive Flow (IAF)

Figure 9: Comparison of MAF and IAF. Note that $\tilde{\mathbf{z}} = \mathbf{x}$, $\tilde{\mathbf{x}} = \mathbf{z}$, $\tilde{\pi} = p$ and $\tilde{p} = \pi$. (Source: [6])

Moreover, Table 1 summarizes the differences between MAF and IAF. We can see the computational trade-offs between them [7]—MAF is able to calculate the density of a sample x in single pass through the model, while sampling from it requires D sequential passes, where D is the dimensionality of x. In contrast, IAF can generate samples with single pass, while calculating the density $p(x)$ of a sample x requires D sequential passes.

Table 1: Comparison of MAF and IAF

	MAF	IAF
Base distribution	$\mathbf{z} \sim \pi(\mathbf{z})$	$\mathbf{x} \sim p(\mathbf{x})$
Target distribution	$\tilde{\mathbf{z}} \sim \tilde{\pi}(\tilde{\mathbf{z}})$	$\tilde{\mathbf{x}} \sim \tilde{p}(\tilde{\mathbf{x}})$
Model	$x_i = z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1})$	$\tilde{x}_i = \tilde{z}_i \odot \tilde{\sigma}_i(\tilde{\mathbf{z}}_{1:i-1}) + \tilde{\mu}_i(\tilde{\mathbf{z}}_{1:i-1})$
Sampling	slow (sequential)	fast (single pass)
Density estimation	fast (single pass)	slow (sequential)

5 Experiments

5.1 Toy data

We implement RealNVP on a toy dataset to examine its effectiveness. We consider a simple 2D data and use 5 affine coupling layers in the model. Figure 10 shows the results. We can see how the latent distribution is transformed into the target distribution.

5.2 MNIST dataset

We implement NICE and RealNVP on MNIST handwritten digit database [15] to examine their effectiveness on more complex data. Each MNIST digit is flatten into a vector of 764 dimensions (originally a 28×28 image). For NICE, we use 6 additive coupling layers and Figure 11 shows the

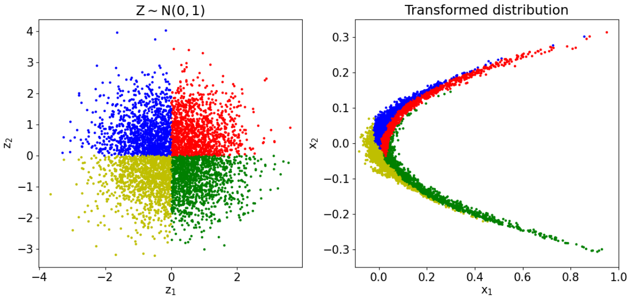


Figure 10: RealNVP on toy dataset

generated MNIST digits. For RealNVP, we use 5 affine coupling layers and Figure 12 shows the generated MNIST digits. We can see that both models are able to capture some characteristics of MNIST digits.

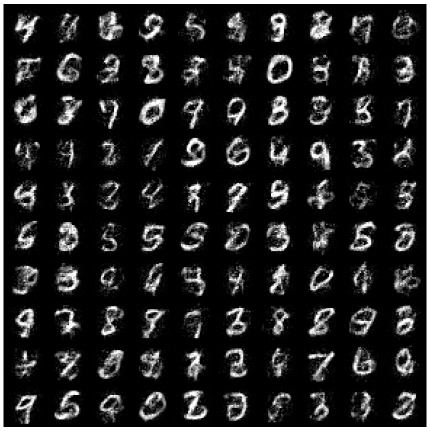


Figure 11: NICE on MNIST

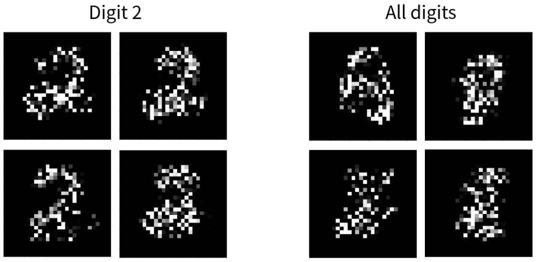


Figure 12: RealNVP on MNIST

6 Summary

In this report, we investigated the flow-based deep generative models, as summarized below.

- We compared different generative models, including GANs, VAEs and flow-based models.
- We surveyed different normalizing flow models, including NICE, RealNVP, Glow, MAF and IAF.
- We conducted experiments on generating MNIST handwritten digits using NICE and RealNVP.

References

[1] Ian Goodfellow. “Generative Adversarial Networks.” In *NeurIPS tutorial*. 2016.

[2] Ian J. Goodfellow et al. “Generative Adversarial Nets.” In *NeurIPS*. 2014.

[3] AI Gharakhanian. *Generative Adversarial Networks*. blog post. 2017. URL: `https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html`.

[4] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes.” In *ICLR*. 2014.

[5] Lilian Weng. *From Autoencoder to Beta-VAE*. blog post. 2018. URL: `https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html`.

[6] Lilian Weng. *Flow-based Deep Generative Models*. blog post. 2018. URL: `https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html`.

[7] Danilo Rezende and Shakir Mohamed. “Variational Inference with Normalizing Flows.” In *ICML*. 2015.

[8] Laurent Dinh, David Krueger, and Yoshua Bengio. “NICE: Non-linear Independent Components Estimation.” In *ICLR*. 2015.

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density Estimation using Real NVP.” In *ICLR*. 2017.

[10] Diederik P. Kingma and Prafulla Dhariwal. “Glow: Generative Flow with Invertible 1×1 Convolutions.” In *NeurIPS*. 2018.

[11] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation.” In *ICLR*. 2018.

[12] George Papamakarios, Theo Pavlakou, and Iain Murray. “Masked Autoregressive Flow for Density Estimation.” In *NeurIPS*. 2017.

[13] Diederik P. Kingma et al. “Improved Variational Inference with Inverse Autoregressive Flow.” In *NeurIPS*. 2016.

[14] Mathieu Germain et al. “MADE: Masked Autoencoder for Distribution Estimation.” In *ICML*. 2015.

[15] Yann LeCun et al. “Gradient-based learning applied to document recognition.” *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.