Deep Generative Models

Flow-based Deep Generative Models

Hamid Beigy

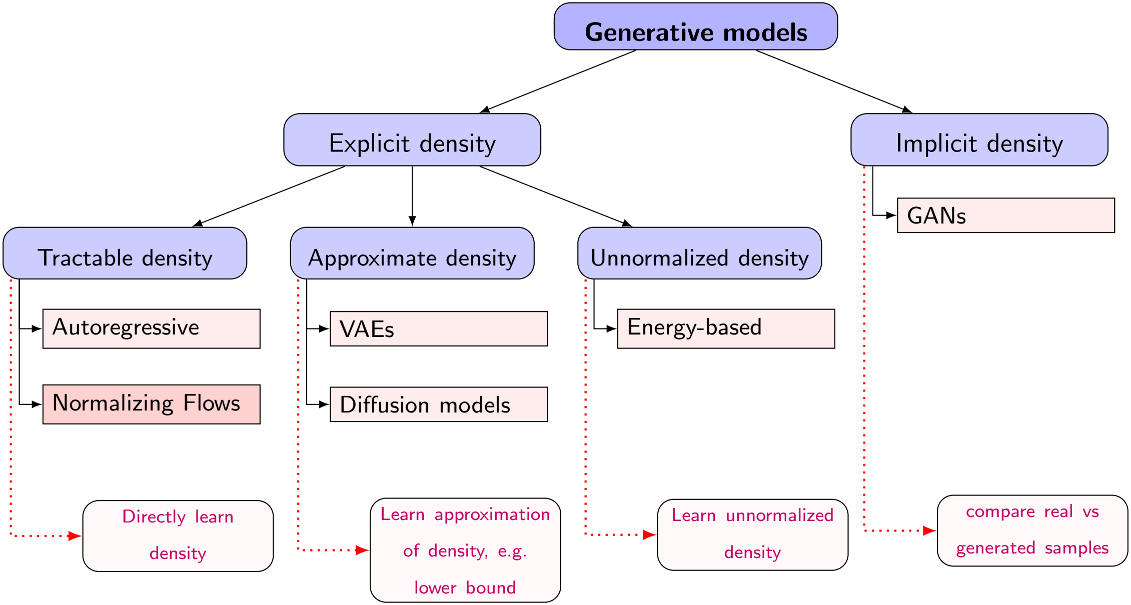
Sharif University of Technology

April 20, 2025

Table of contents

Introduction

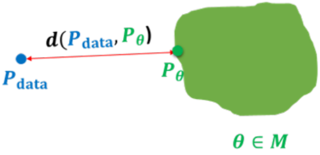
Generative models categories



Generative models

1. Assume that the observed variable $\mathbf{x}$ is a random sample from an underlying process, whose true distribution $p_{\mathrm{d}}(\mathbf{x})$ is unknown.

$\mathbf{x}_i \sim P_{\mathrm{data}}$
$i = 1, 2, \dots, n$



Model family

2. We attempt to approximate this process with a chosen model, $p_\theta(\mathbf{x})$, with parameters θ such that $\mathbf{x} \sim p_\theta(\mathbf{x})$.

3. Learning is the process of searching for the parameter θ such that $p_\theta(\mathbf{x})$ well approximates $p_{\mathrm{d}}(\mathbf{x})$ for any observed $\mathbf{x}$, i.e.

$$p_\theta(\mathbf{x}) \approx p_{\mathrm{d}}(\mathbf{x})$$

4. We wish $p_\theta(\mathbf{x})$ to be sufficiently flexible to be able to adapt to the data for obtaining sufficiently accurate model and to be able to incorporate prior knowledge.

Credit: Aditya Grover

Likelihood-based models

Autoregressive models

1. Tractable density
2. Density is estimated as

$$p_\theta(\mathbf{x}) = \prod_{j=1}^{m} p_\theta(\mathbf{x}_j \mid \mathbf{x}_{<j})$$

3. Tractable likelihood
4. No inferred latent factors

Latent variable models

1. Approximated density
2. Density is estimated as

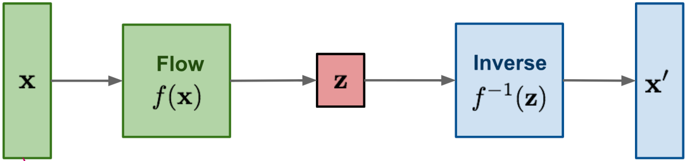
$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) dz$$

3. Intractable likelihood
4. Latent feature representation

Autoregressive models are powerful density estimators and can learn long-rang statistics.

How to build model with latent variables and tractable likelihood?

1. Flow-based generative models:
   - Constructed by a sequence of invertible transformations.
   - Learns data distribution where loss function is simply the negative log-likelihood.



Flow-based Models

Jacobian Matrix and Determinant

1. 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 Jacobian matrix, $\mathbf{J}$, is

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

2. The determinant of a $n \times n$ matrix $\mathbf{M}$ is

$$\det(\mathbf{M}) = \det\left(\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}\right) = \sum_{j_1 j_2 \ldots j_n} (-1)^{\tau(j_1 j_2 \ldots j_n)} a_{1j_1} a_{2j_2} \ldots a_{nj_n}$$

$\tau(.)$ indicates the signature of a permutation.

Some slides of this section are adopted from Lilian Weng's page

Change of Variable Theorem

1. Given a random variable $\mathbf{z}$ and its known probability density function $\mathbf{z} \sim p(\mathbf{z})$, we would like to construct a new random variable using a one-one mapping function $\mathbf{x} = f(\mathbf{z})$.
2. The function f is invertible, so $\mathbf{z} = f^{-1}(\mathbf{x})$.
3. The question is how to infer the unknown probability density function of the new variable, $p(\mathbf{x})$?

$$\int_{\mathbf{x}} p(\mathbf{x})d\mathbf{x} = \int_{\mathbf{z}} p(\mathbf{z})d\mathbf{z} = 1$$

Definition of probability distribution.

$$\begin{aligned} p(\mathbf{x}) &= p(\mathbf{z})\left|\frac{d\mathbf{z}}{d\mathbf{x}}\right| \\ &= p(f^{-1}(\mathbf{x}))\left|\frac{df^{-1}}{d\mathbf{x}}\right| = p(f^{-1}(\mathbf{x}))\left|(f^{-1})'(\mathbf{x})\right| \end{aligned}$$

4. By definition, the integral $\int_{\mathbf{z}} p(\mathbf{z})d\mathbf{z}$ is the sum of an infinite number of rectangles of infinitesimal width $\Delta\mathbf{z}$.
5. The height of such a rectangle at position $\mathbf{z}$ is the value of the density function $p(\mathbf{z})$.

Change of Variable Theorem

1. When we substitute the variable, $\mathbf{z} = f^{-1}(\mathbf{x})$ yields $\frac{\Delta \mathbf{z}}{\Delta \mathbf{x}} = \left(f^{-1}(\mathbf{x})\right)'$ and $\Delta \mathbf{z} = \left(f^{-1}(\mathbf{x})\right)' \Delta \mathbf{x}$.
2. Here $\left|f^{-1}(x)'\right|$ indicates the ratio between the area of rectangles defined in two different coordinate of variables z and x, respectively.
3. The multivariable version has a similar format:

$$
\begin{aligned}
\mathbf{z} &\sim p(\mathbf{z}) \\
\mathbf{x} &= f(\mathbf{z}) \\
\mathbf{z} &= f^{-1}(\mathbf{x}) \\
p(\mathbf{x}) &= p(\mathbf{z}) \left|\det\left(\frac{d\mathbf{z}}{d\mathbf{x}}\right)\right| \\
&= p(f^{-1}(\mathbf{x})) \left|\det\left(\frac{df^{-1}}{d\mathbf{x}}\right)\right|
\end{aligned}
$$

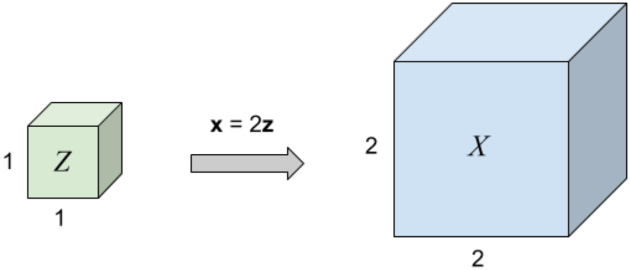
where $\det\left(\frac{\partial f}{\partial \mathbf{z}}\right)$ is the Jacobian determinant of the function f.

Change of Variable Theorem (example)

1. Consider a random variable Z that is uniformly distributed over the unit cube $\mathbf{z} \in [0,1]^3$.
2. We can scale Z by a factor of 2 to get a new random variable X,

$$\mathbf{x} = f(\mathbf{z}) = \mathbf{A}\mathbf{z} = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix} \mathbf{z}$$

where X is uniform over a cube with side length 2.



Change of Variable Theorem (example)

1. How is the density $p(\mathbf{x})$ related to $p(\mathbf{z})$?
2. Since every distribution sums to 1 and the unit cube has volume $V_Z = 1$.

   $$p(\mathbf{z})V_Z = 1$$

   and $p(\mathbf{z}) = 1$ for all $\mathbf{z}$ in the unit cube.
3. The volume of the larger cube is easy to compute: $V_X = 2^3 = 8$.
4. The total probability mass must be conserved, so we can solve for the density of X.

   $$p(\mathbf{x}) = \frac{p(\mathbf{z})V_Z}{V_X} = \frac{1}{8}$$

5. The new density is equal to the original density multiplied by the ratio of the volumes.

Change of Variable Theorem (example)

1. The change of variables formula allows us to tractably compute normalized probability densities when we apply an invertible transformation f.

$$\begin{aligned} p(\mathbf{x}) =& \; p(\mathbf{z})\left|\det\left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right| \\ =& \; p(\mathbf{z})\left|\det\left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right)^{-1}\right| \end{aligned}$$

2. The invertible function is just multiplication by a scaling matrix, so the determinant of the Jacobian matrix is easy to compute:

$$\det\left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right) = \det\left(\mathbf{A}\right) = 8.$$

Change of Variable Theorem (example)

1. Let $z \in \mathbb{R}$ and $p(z) = \mathcal{N}(0, 1)$.
2. Let x be a new random variable after applying a linear transformation to z.
3. Let $x = 0.75z + 1$, what is $p(x)$?
4. Using the change of variable theorem,

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

5. Let $f(z) = 0.75z + 1$, and hence

$$\begin{aligned} f^{-1}(x) &= \frac{x-1}{0.75} \\ \left| \frac{\partial f^{-1}(x)}{\partial x} \right| &= \frac{4}{3} \end{aligned}$$

6. Putting all together yields

$$\begin{aligned} p(x) &= p(z = \frac{x-1}{0.75})\frac{4}{3} \\ &= \mathcal{N}(1, 0.75) \end{aligned}$$

Change of Variable Theorem (example)

1. The change of variables formula allows us to tractably compute normalized probability densities when we apply an invertible transformation f.

$$
\begin{aligned}
p(\mathbf{x}) &= p(\mathbf{z}) \left| \det \left(\frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \\
&= p(\mathbf{z}) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right)^{-1} \right|
\end{aligned}
$$

2. The invertible function is just multiplication by a scaling matrix, so the determinant of the Jacobian matrix is easy to compute:

$$
\det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) = \det(\mathbf{A}) = 8.
$$

Inverse Function Theorem

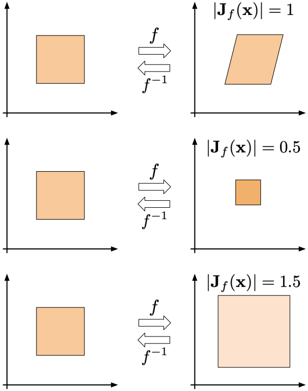
1. Theses examples show that we can calculate a new distribution of a continuous random variable by applying a known bijective transformation f to a random variable with a known distribution $\mathbf{z} \sim p(\mathbf{z})$.

$$p(\mathbf{x}) = p(\mathbf{z} = f^{-1}(\mathbf{x}))\left|\det\left(\mathbf{f}^{-1}(\mathbf{x})\right)\right|$$

2. From **inverse function theorem**, we have

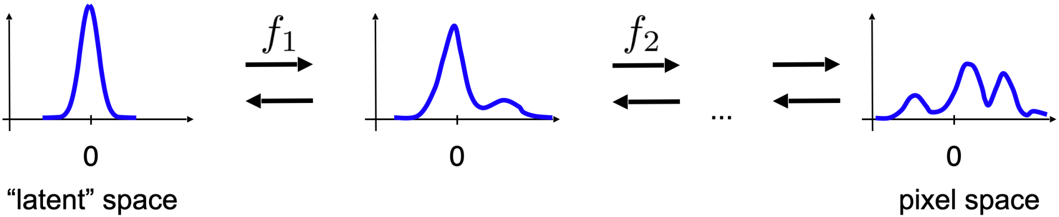
$$\left|\det\left(\mathbf{f}^{-1}(\mathbf{x})\right)\right| = \left|\det\left(\mathbf{f}(\mathbf{x})\right)^{-1}\right|$$

1. $\mathbf{x}$ and $\mathbf{z}$ have the same dimensionality $(\mathbb{R}^n)$.
2. f could be parametric function as $\mathbf{f}_\theta(\mathbf{x})$.
3. Determinant of Jacobian matrix $\mathbf{J} = \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}}$ shows how the volume changes under the transformation


f
f^{-1}
$|\mathbf{J}_f(\mathbf{x})| = 1$
f
f^{-1}
$|\mathbf{J}_f(\mathbf{x})| = 0.5$
f
f^{-1}
$|\mathbf{J}_f(\mathbf{x})| = 1.5$

What is Normalizing Flows?

1. Density estimation has several important applications in many machine learning problems.
2. In deep learning models, the embedded probability distribution is expected to be simple enough to calculate the derivative easily and efficiently.
3. This is why Gaussian distribution is often used in latent variable generative models.
4. Normalizing Flow (NF) models are used for better and more powerful distribution approximation (Rezende and Mohamed 2015).
5. A normalizing flow transforms a simple distribution into a complex one by applying a sequence of invertible transformation functions.

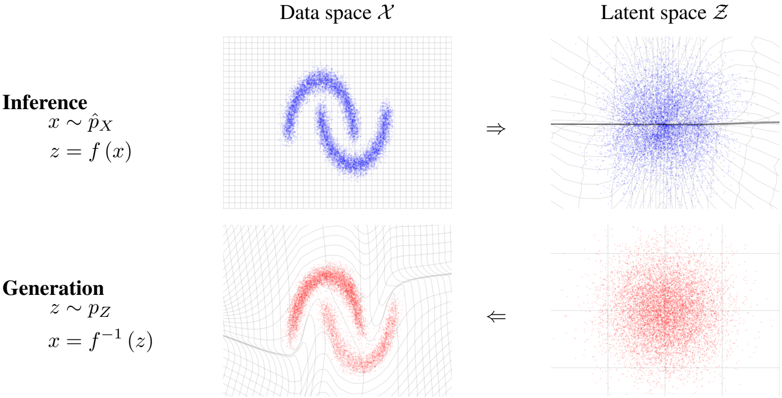


Fitting normalizing flows

1. **MLE problem**

$$
\begin{aligned}
p_\theta(\mathbf{x}) &= p(\mathbf{z})\left|\det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right)\right| \\
&= p(\mathbf{f}_\theta(\mathbf{x}))\left|\det\left(\frac{\partial \mathbf{f}_\theta(\mathbf{x})}{\partial \mathbf{x}}\right)\right| \\
\log\ p_\theta(\mathbf{x}) &= \log\ p(\mathbf{f}_\theta(\mathbf{x})) + \log\left|\det\left(\mathbf{J}_f\right)\right|
\end{aligned}
$$

2. We want to find θ that maximizes $\log\ p_\theta(\mathbf{x})$.



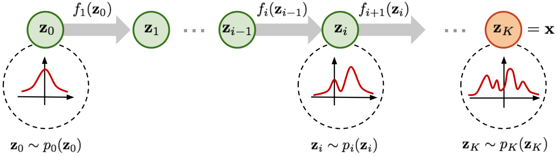
Composition of normalizing flows

Theorem

Diffeomorphisms[a] *are* **composable** *if* $\{f_k\}_{k=1}^{K}$ *satisfy conditions of the change of variable theorem, then* $\mathbf{z} = f(\mathbf{x}) = f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{x})$ *also satisfies it.*

[a] It is an invertible function that maps one differentiable manifold to another such that both the function and its inverse are continuously differentiable.

$$
\begin{aligned}
p(\mathbf{x}) = \; & p(f(\mathbf{x}))\left|\det\left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right)\right| = p(f(\mathbf{x}))\left|\det\left(\frac{\partial f_K}{\partial f_{K-1}} \cdots \frac{\partial f_1}{\partial \mathbf{x}}\right)\right| \\
= \; & p(f(\mathbf{x}))\left|\det\left(\frac{\partial f_k}{\partial f_{k-1}}\right)\right| \\
= \; & p(f(\mathbf{x}))\prod_{k=1}^{K}\left|\det\left(\mathbf{J}_{f_k}\right)\right|
\end{aligned}
$$



Normalizing Flows

1. We want to find θ that maximizes $\log\ p_\theta(\mathbf{x})$.

$$\log\ p_\theta(\mathbf{x}) = \log\ p(\mathbf{f}_\theta(\mathbf{x})) + \log\left|\det\left(\mathbf{J}_f\right)\right|$$

Definition (Normalizing flows)

Normalizing flow is a differentiable, invertible mapping from data $\mathbf{x}$ to the noise $\mathbf{z}$.

- **Normalizing** means that NF takes samples from $p(\mathbf{x})$ and normalizes them into samples from the density $p(\mathbf{z})$.
- **Flow** refers to the trajectory followed by samples from $p(\mathbf{x})$ as they are transformed by the sequence of transformations

$$\begin{aligned}
\mathbf{z} &= f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{x}) \\
\mathbf{x} &= f_K^{-1} \circ f_{K-1}^{-1} \circ \ldots \circ f_1^{-1}(\mathbf{z}) \\
&= g_1 \circ g_2 \circ \ldots \circ g_K(\mathbf{z})
\end{aligned}$$

Normalizing Flows

1. From the previous slide, we have

$$\begin{aligned} \mathbf{z}_{i-1} &\sim p_{i-1}(\mathbf{z}_{i-1}) \\ \mathbf{z}_i &= f_i(\mathbf{z}_{i-1}), \text{ thus } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i) \\ p_i(\mathbf{z}_i) &= p_{i-1}(f_i^{-1}(\mathbf{z}_i))\left|\det\left(\frac{df_i^{-1}}{d\mathbf{z}_i}\right)\right| \end{aligned}$$

2. Repeating above, we can do inference using base distribution.

$$\begin{aligned} p_i(\mathbf{z}_i) &= p_{i-1}(\mathbf{z}_{i-1})(f_i^{-1}(\mathbf{z}_i)) \\ &= p_{i-1}(\mathbf{z}_{i-1})\left|\det\left(\left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)^{-1}\right)\right| \text{According to the inverse function theorem.} \\ &= p_{i-1}(\mathbf{z}_{i-1})\left|\det\left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)\right|^{-1} \quad \text{Using property of Jacobians of invertible funcion} \\ \log\, p_i(\mathbf{z}_i) &= \log\, p_{i-1}(\mathbf{z}_{i-1}) - \log\left|\det\left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)\right| \end{aligned}$$

Normalizing Flows

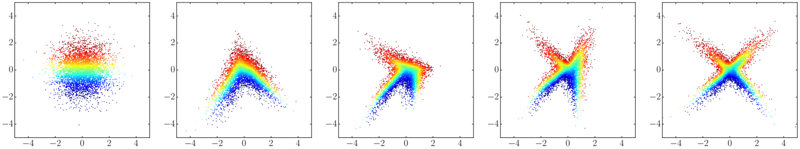
1. Given chain of pdfs, we can expand the equation of the output $\mathbf{x}$ step by step until tracing back to the initial distribution $\mathbf{z}_0$.

$$\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \\
\log\ p(\mathbf{x}) &= \log\ p_K(\mathbf{z}_K) \\
&= \log\ p_{K-1}(\mathbf{z}_{K-1}) - \log\left|\det\left(\frac{df_K}{d\mathbf{z}_{K-1}}\right)\right| \\
&= \log\ p_{K-2}(\mathbf{z}_{K-2}) - \log\left|\det\left(\frac{df_{K-1}}{d\mathbf{z}_{K-2}}\right)\right| \\
&- \log\left|\det\left(\frac{df_K}{d\mathbf{z}_{K-1}}\right)\right| \\
&= \ldots \\
&= \log\ p_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log\left|\det\left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)\right|
\end{aligned}$$

Normalizing Flows

1. The path traversed by the random variables $\mathbf{z}_i = f_i(\mathbf{z}_{i-1})$ is the flow.
2. The full chain formed by the successive distributions $p_i(\mathbf{z}_i)$ is called a normalizing flow.
3. For computation of equation, a transformation function f_i should satisfy two properties:
   - It is easily invertible.
   - Its Jacobian determinant is easy to compute.
4. With normalizing flows, the exact log-likelihood of input data $\log p(\mathbf{x})$ becomes tractable.
5. The training criterion of flow-based generative model is simply the negative log-likelihood (NLL) over the training dataset S.

$$\mathcal{L}(S) = -\frac{1}{|S|} \sum_{\mathbf{x} \in S} \log p(\mathbf{x})$$



Applications of Normalizing Flows

1. How to generate a sample?
   - Sample $\mathbf{z}$ from $p_{\mathbf{z}}(\mathbf{z})$: $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$.
   - Compute function $\mathbf{x} = f(\mathbf{z})$.
2. How to compute the density $p_{\mathbf{x}}(\mathbf{x})$. Let $g = f^{-1}$.

$$\begin{aligned} p_{\mathbf{x}}(\mathbf{x}) &= p_{\mathbf{z}}(g(\mathbf{x}))|\det(\mathbf{J}_g(\mathbf{x}))| \\ &= p_{\mathbf{z}}(\mathbf{z})|\det(\mathbf{J}_f(\mathbf{z}))|^{-1} \end{aligned}$$

3. How flexible are the densities $p_{\mathbf{x}}(\mathbf{x})$ obtained by transforming random variables sampled from simple $p_{\mathbf{z}}(\mathbf{z})$?
4. We can use normalizing flows method to approximate any smooth distribution.

How to train a flow model?

1. There are two common applications of normalizing flow models.
   - **Density estimation**, i.e. fitting $p_\theta(\mathbf{x})$ to the data.

     The objective is to maximize the following function

     $$p_\mathbf{x}(\mathbf{x}) = p_\mathbf{z}(\mathbf{z})\left|\det\left(\mathbf{J}_f(\mathbf{z})\right)\right|^{-1}$$

     This requires that we can evaluate $f(\mathbf{z})^{-1}$ and $\det\left(\mathbf{J}_f(\mathbf{z})\right)^{-1}$ efficiently.
   - **Variational inference**, sampling and evaluating from a variational posterior $q_\theta(\mathbf{z} \mid \mathbf{x})$ parameterized by a flow model.

     The variational parameters are trained by maximizing the evidence lower bound (ELBO),

     $$\mathcal{L}(\theta) = \mathbb{E}_{q_\theta(\mathbf{z} \mid \mathbf{x})}[\log\, p(\mathbf{x} \mid \mathbf{z}) + \log\, p_\mathbf{z}(\mathbf{z}) - \log\, q_\theta(\mathbf{z} \mid \mathbf{x})]$$

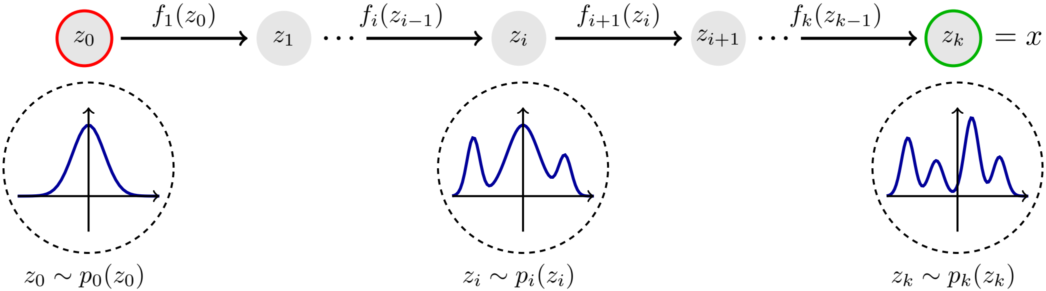
     To optimize this objective, we need to be able to efficiently sample from $q_\theta(\mathbf{z} \mid \mathbf{x})$ and evaluate the probability density of these samples during optimization.

Constructing flows

Introduction

1. The density $p_k(\mathbf{z}_k)$ obtained by successively transforming a random variable $\mathbf{x}_0$ with distribution $p_0(\mathbf{z})$ through a chain of K transformations f_k defined as

$$\mathbf{x}_K = f_K \circ \ldots \circ f_2 \circ f_1(\mathbf{z}_0)$$



Affine flows

1. A simple choice is to use an affine transformation $\mathbf{x} = f(\mathbf{z}) = \mathbf{Az} + \mathbf{b}$.
2. This is a bijection if and only if $\mathbf{A}$ is an invertible square matrix.
3. The inverse is $\mathbf{z} = f^{-1}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b})$.
4. Affine flows are limited in their expressive power.
   - Let the base distribution be Gaussian $p_{\mathbf{z}}(\mathbf{z}) = \mathcal{N}(\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}})$.
   - The distribution after an affine bijection is still Gaussian $p_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{A}\mu_{\mathbf{z}} + \mathbf{b}, \mathbf{A}\Sigma_{\mathbf{z}}\mathbf{A}^{\top})$.
5. However, they are useful building blocks when composed with the non-affine bijections.
6. We need to ensure Jacobian determinant and inverse of flow are fast to compute.
   - In general, computing $\det(\mathbf{A})$ and $\det\left(\mathbf{A}^{-1}\right)$ requires $O(D^3)$ time.
   - If $\mathbf{A}$ is diagonal, the cost becomes $O(D)$.
   - If $\mathbf{A}$ is triangular, Jacobian determinant is product of its diagonal elements taking $O(D)$.
   - Inverting flow requires solving triangular system $\mathbf{Az} = \mathbf{x} - \mathbf{b}$ using back-substitution in $O(D^2)$ time.

Elementwise flows

1. Let $h : \mathbb{R} \mapsto \mathbb{R}$ be a scalar-valued bijection.
2. We can create a vector-valued bijection $f : \mathbb{R}^D \mapsto \mathbb{R}^D$ by applying h element-wise: $f(\mathbf{z}) = (h(z_1), h(z_2), \ldots, h(z_D))$.
3. Function f is invertible and its Jacobian determinant is $\prod_{i=1}^{D} \frac{\partial h}{\partial z_i}$.
4. Element-wise flows are limited, since they do not model dependencies between the elements.
5. However, they are useful building blocks for more complex flows.
6. Examples of element-wise flows are
   - Affine scalar bijection
   - Higher-order perturbations
   - Combinations of strictly monotonic scalar functions
   - Scalar bijections from integration
   - Splines

Elementwise flows

1. **Affine scalar bijection**
   - An affine scalar bijection has the form $h(z;\theta) = az + b$, where $\theta = (a, b) \in \mathbb{R}^2$.
   - Its derivative equals to a.
   - It is invertible if and only if $a \neq 0$.
2. **Higher-order perturbations**
   - The affine scalar bijection is simple to use, but limited.
   - We can make it more flexible by adding higher-order perturbations, under the constraint that invertibility is preserved.

     $$h(z;\theta) = az + b + \frac{c}{1 + (dz + e)^2}$$

     where $\theta = (a, b, c, d, e) \in \mathbb{R}^5$.
   - Under the constraints $a > \frac{9}{8\sqrt{3}}cd$ and $d > 0$, the function becomes invertible.

Elementwise flows

3. **Combinations of strictly monotonic scalar functions**

- A strictly monotonic scalar function is one that is always increasing or always decreasing.
- These functions are **invertible**.
- Many activation functions, such as $\sigma(z) = \frac{1}{1+e^{-z}}$, are strictly monotonic.
- Using such activation functions as a starting point, we can build more flexible monotonic functions via conical combination (linear combination with positive coefficients) and function composition.
- Suppose $h_1, \ldots, h_K$ are strictly increasing; then the following are also strictly increasing

$$\sum_{k=1}^{K} a_k h_k + b \quad \text{with} \quad a_k > 0 \qquad \text{conical combination with a bias}$$

$$h_1 \circ \ldots \circ h_K \qquad \text{function composition}$$

- By repeating the above two constructions, we can build arbitrarily complex increasing functions.
- For example, a composition of conical combinations of logistic sigmoids is just an MLP where all weights are positive.

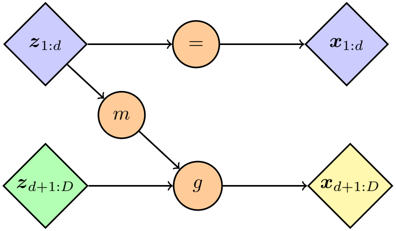
Coupling flows

1. **Coupling flows** allow us to model dependencies between dimensions using arbitrary non-linear functions
2. Consider a partition of the input $\mathbf{z} \in \mathbb{R}^D$ into two sub-spaces: $(\mathbf{z}^A, \mathbf{z}^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$, where d is an integer in interval $[1, D-1]$.
3. Assume a bijection $\widehat{f}(.;\theta) : \mathbb{R}^d \mapsto \mathbb{R}^d$ parameterized by θ.
4. We define the function $f : \mathbb{R}^D \mapsto \mathbb{R}^D$ as

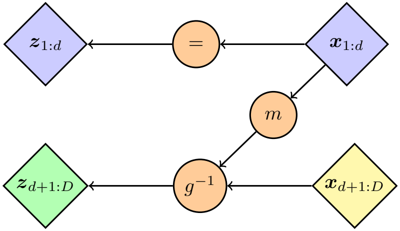
$$\begin{aligned} \mathbf{x}^A &= \widehat{f}(\mathbf{z}^A; \Theta(\mathbf{z}^B)) \\ \mathbf{x}^B &= \mathbf{z}^B \end{aligned}$$

   or vice versa.

5. Θ is an arbitrary function called the **conditioner**.



forward pass



inverse pass

Coupling flows

1. The coupling layer f is invertible, and its inverse is

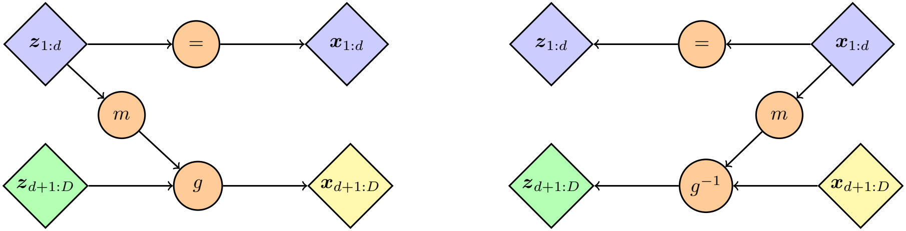
$$\mathbf{z}^A = \widehat{f}^{-1}(\mathbf{x}^A; \Theta(\mathbf{x}^B))$$
$$\mathbf{z}^B = \mathbf{x}^B$$

   or vice versa.

2. The Jacobian of f is block triangular:

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial \mathbf{x}^A}{\partial \mathbf{z}^A} & \frac{\partial \mathbf{x}^A}{\partial \mathbf{z}^B} \\ \frac{\partial \mathbf{x}^B}{\partial \mathbf{z}^A} & \frac{\partial \mathbf{x}^B}{\partial \mathbf{z}^B} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_f & \frac{\partial \mathbf{x}^A}{\partial \mathbf{z}^B} \\ 0 & 1 \end{bmatrix}$$

3. We often define $\widehat{f}$ to be an elementwise bijection, so that $\widehat{f}^{-1}$ and $\det\left(J_{\widehat{f}}\right)$ are easy to compute.



forward pass

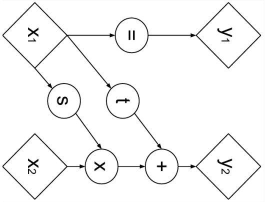
inverse pass

Real-valued Non-Volume Preserving (RealNVP)

1. The RealNVP model implements a normalizing flow by stacking a sequence of invertible bijective transformation functions (Dinh, Sohl-Dickstein, and S. Bengio 2017).
2. In each bijection $f : \mathbf{x} \mapsto \mathbf{y}$, the input dimensions are split into two parts:
   - The first d dimensions stay same (x_1);
   - The second part, $d + 1$ to D dimensions (x_2) transformed using

     $$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{aligned}$$

     where $s(.)$ and $t(.)$ are scale and translation functions and both map $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$. The $\odot$ operation is the element-wise product.
3. This network has
   - Stack many invertible coupling layers.
   - Each has simple inverse and determinant



Real-valued Non-Volume Preserving (RealNVP)

1. This transformation satisfy two properties of flow transformations.
   - It is easily invertible.

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

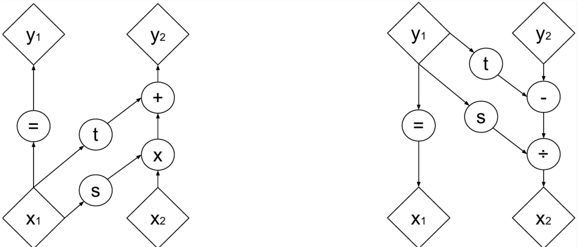
   - Its Jacobian determinant is easy to compute. The Jacobian is a lower triangular matrix.

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix}$$

   Hence, the determinant is simply the product of terms on the diagonal.

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

2. The inverse transformation



Real-valued Non-Volume Preserving (RealNVP)

Inference

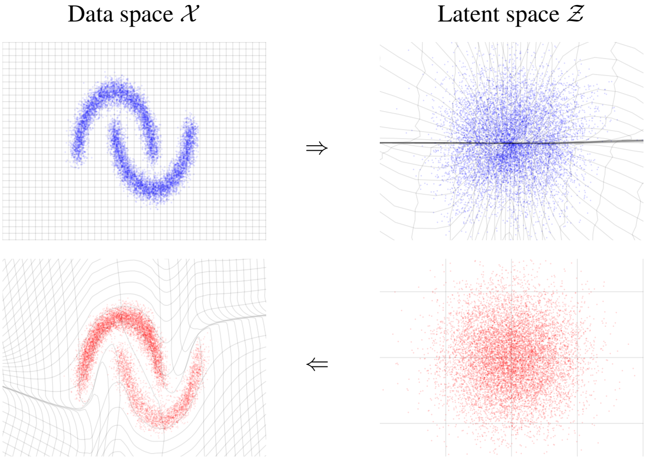
$$x \sim \hat{p}_X$$

$$z = f\left(x\right)$$

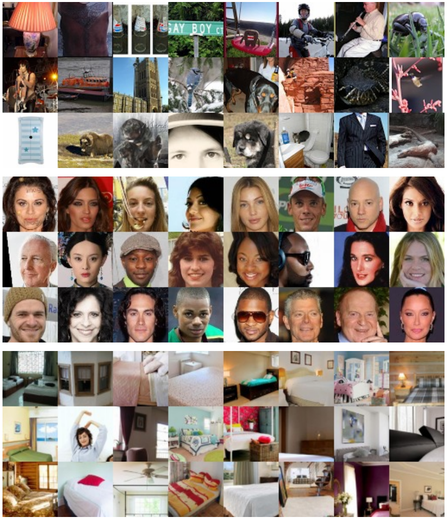
Generation

$$z \sim p_Z$$

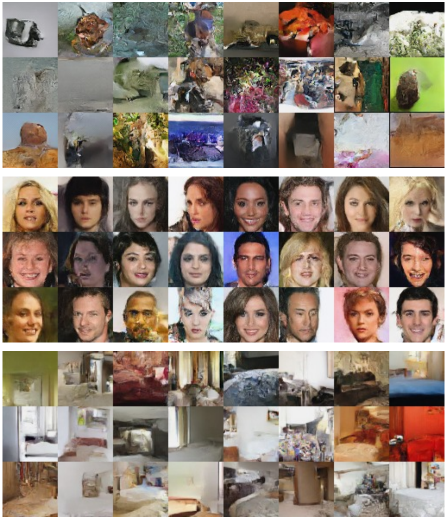
$$x = f^{-1}\left(z\right)$$



Real-valued Non-Volume Preserving (RealNVP)



Dataset



samples from model

Non-linear Independent Component Estimation (NICE)

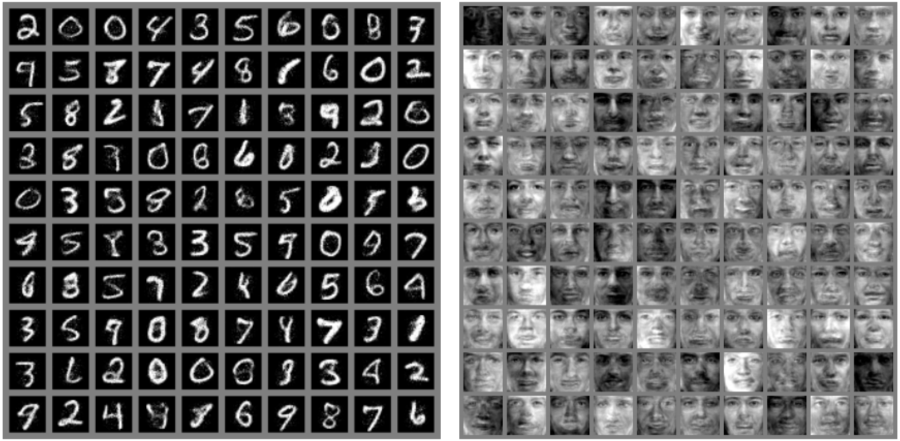
1. The NICE model is a predecessor of Real NVP (Dinh, Krueger, and Y. Bengio 2015).
2. The transformation in NICE is the affine coupling layer without the scale term, known as additive coupling layer.

$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} + m(\mathbf{x}_{1:d}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = \mathbf{y}_{d+1:D} - m(\mathbf{y}_{1:d}) \end{cases}$$

   where m is an arbitrarily complex function, in this case a ReLU MLP.
3. Additive layers have unit Jacobian determinant, and their composition will necessarily have unit Jacobian determinant too.
4. NICE includes a diagonal scaling matrix S as the top layer.
5. Final layer of NICE applies a rescaling transformation $x_i = s_i z_i$ and inverse mapping $z_i = \frac{x_i}{s_i}$.
6. Jacobian of forward mapping:

$$\mathbf{J} = \text{diag}(\mathbf{S})$$
$$\det(\mathbf{J}) = \prod_i s_i.$$

Samples generated by NICE

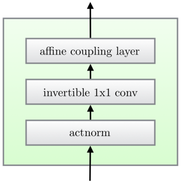


(a) Model trained on MNIST

(b) Model trained on TFD

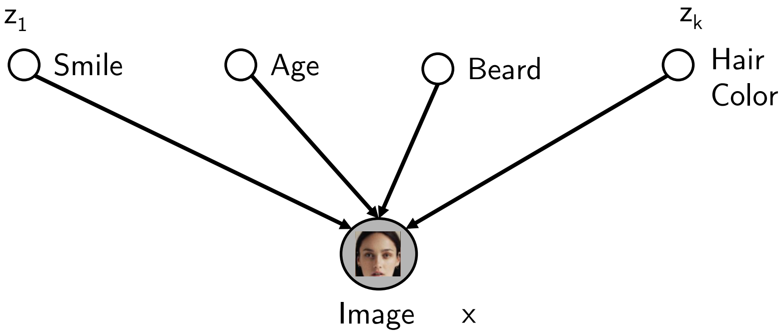
Glow model

1. Glow extends NICE and RealNVP by replacing the reverse permutation operation on the channel ordering with invertible 1×1 convolutions (Kingma and Dhariwal 2018).
2. There are three substeps in one step of flow in Glow.
   - Activation normalization (short for actnorm): (similar to batch normalization with mini-batch size 1 and trainable parameters)
     - Actnorm layer performs an affine transformation of the activations using a scale and bias parameter per channel.
     - These parameters are initialized such that the post-actnorm activations per-channel have zero mean and unit variance.
     - After initialization, the scale and bias are treated as regular trainable parameters that are independent of the data.
   - Invertible 1×1 convolution: Instead of fixed ordering, 1×1 convolution is used.
   - Affine coupling layer (Same as in RealNVP)



Glow model

1. Latent factors



z_1

Smile

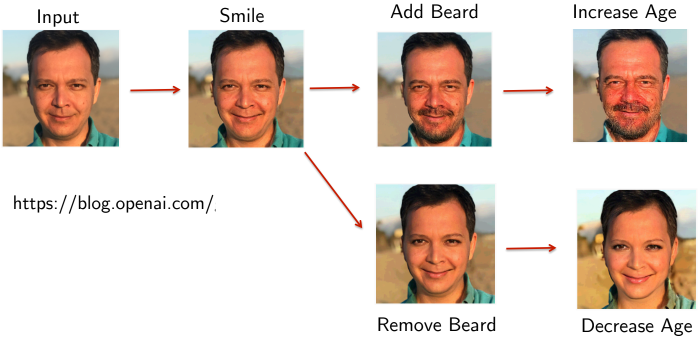
Age

Beard

z_k

Hair Color

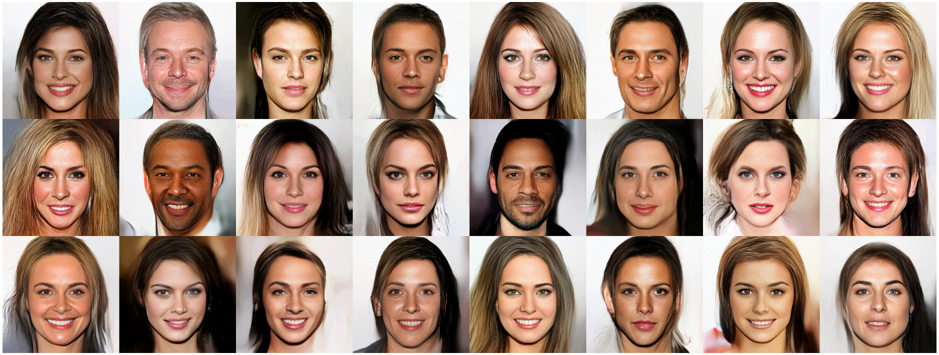
Image x



https://blog.openai.com/

Glow Results

Synthetic celebrities sampled from Glow model



Random samples from the Glow model.

See also https://openai.com/blog/glow/.

Autoregressive flows

1. Autoregressive flows are flows composed of autoregressive bijections.
2. An autoregressive bijection $f : \mathbb{R}^D \mapsto \mathbb{R}^D$ is defined as

$$x_i = h(z_i; \Theta_i(x_{<i})) \qquad\qquad i = 1, \ldots, D$$

3. Since h is invertible, f is also invertible, and its inverse is given by:

$$z_i = h^{-1}(x_i; \Theta_i(x_{<i})) \qquad\qquad i = 1, \ldots, D$$

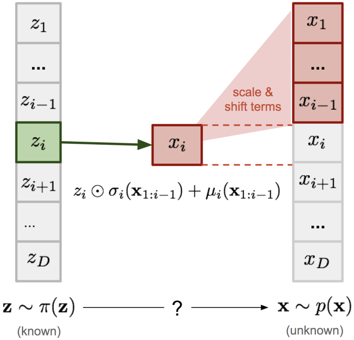
4. An important property of f is that each output x_i depends only on $z_{<i}$, which results

$$\det(\mathbf{J}_f) = \prod_{i=1}^{D} \frac{\partial h}{\partial z_i}$$

5. Hence, $\mathbf{J}_f$ can be computed efficiently in $O(D)$ time.
6. Although invertible, autoregressive bijections are computationally asymmetric:
   - Evaluating f is inherently sequential. That is because we need $x_1, \ldots, x_{i-1}$ to compute x_i.
   - Evaluating f^{-1} is inherently parallel. That is because z does not appear on the right-hand.

Autoregressive flows (Masked autoregressive flows)

1. The conditioners Θ_i can be arbitrary non-linear functions.
2. The most straightforward way to parameterize them is separately for each i, for example by using D separate neural networks (Papamakarios, Murray, and Pavlakou 2017).
3. However, this can be parameter-inefficient for large D.
4. In practice, we often share parameters between conditioners by combining them into a single model Θ that takes in x and outputs $(\theta_1, \ldots, \theta_D)$.
5. For the bijection to remain autoregressive, we must constrain Θ so that θ_i depends only on $x_1, \ldots, x_{i-1}$ and not on $x_i, \ldots, x_D$.


z_1
...
z_{i-1}
z_i
z_{i+1}
...
z_D
x_1
...
x_{i-1}
x_i
x_{i+1}
...
x_D
scale & shift terms
x_i
$z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1})$
$\mathbf{z} \sim \pi(\mathbf{z})$
(known)
?
$\mathbf{x} \sim p(\mathbf{x})$
(unknown)

Autoregressive flows (Masked autoregressive flows)

1. Given two random variables $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ and $\mathbf{x} \sim p_{\mathbf{x}}(\mathbf{x})$, and the probability density function $p_{\mathbf{z}}(\mathbf{z})$ is known, MAF aims to learn $p_{\mathbf{x}}(\mathbf{x})$.
2. MAF generates each x_i, conditioned on the past dimensions $\mathbf{x}_{1:i-1}$.
   - Data generation, producing a new $\mathbf{x}$.

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

     where

     $$\begin{aligned} p(x_i \mid \mathbf{x}_{1:i-1}) &= \mathcal{N}\left(x_i | \mu_i, (\exp \alpha_i)^2\right) \\ \mu_i &= f_{\mu_i}(\mathbf{x}_{1:i-1}) \\ \alpha_i &= f_{\alpha_i}(\mathbf{x}_{1:i-1}) \\ z_i &\sim \mathcal{N}(0, 1) \end{aligned}$$

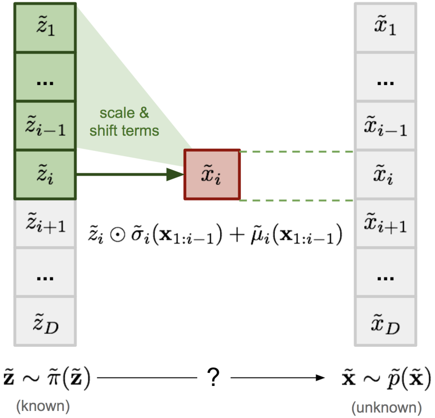
   - Density estimation, given a known $\mathbf{x}$.

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

Autoregressive flows (Inverse autoregressive flows)

1. IAF models the conditional probability of the target variable as an autoregressive model too, but with a reversed flow (for efficient sampling process) (Kingma, Salimans, and Welling 2016).
2. IAF makes θ_i a function of the previous inputs $z_1, \ldots, z_{i-1}$ (Papamakarios, Murray, and Pavlakou 2017).
3. This leads to the following bijection, which is known as inverse autoregressive:

$$x_i = h(z_i; \Theta_i(\mathbf{z}_{<i})) \qquad\qquad i = 1, \ldots, D$$



Autoregressive flows (Inverse autoregressive flows)

1. In IAF, the nonlinear shift/scale statistics are computed using the previous noise variates $\mathbf{z}_{1:i-1}$, instead of the data samples:

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

$$\mu_i = f_{\mu_i}(\mathbf{z}_{1:i-1}) \qquad \alpha_i = f_{\alpha_i}(\mathbf{z}_{1:i-1})$$

2. The reverse transformation in MAF is

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

3. If we consider

$$\tilde{\mathbf{x}} = \mathbf{z},\ \tilde{p}(.) = p(.),\ \tilde{\mathbf{x}} \sim \tilde{p}(\tilde{\mathbf{x}})$$

$$\tilde{\mathbf{z}} = \mathbf{x},\ \tilde{p}(.) = p(.),\ \tilde{\mathbf{z}} \sim \tilde{p}(\tilde{\mathbf{z}})$$

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

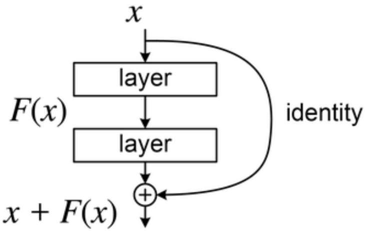
$$\tilde{\sigma}(\tilde{\mathbf{z}}_{1:i-1}) = \tilde{\sigma}(\mathbf{x}_{1:i-1}) = \frac{1}{\sigma_i(\mathbf{x}_{1:i-1})}$$

4. Then, $\tilde{x}_i \sim p(\tilde{x}_i|\tilde{\mathbf{z}}_{1:i}) = \tilde{z}_i \odot \tilde{\sigma}_i(\tilde{\mathbf{z}}_{1:i-1}) + \tilde{\mu}_i(\tilde{\mathbf{z}}_{1:i-1})$, where $\tilde{\mathbf{z}} \sim \tilde{p}(\tilde{\mathbf{z}})$

5. IAF intends to estimate the probability density function of $\tilde{\mathbf{x}}$ given that $\tilde{p}(\tilde{\mathbf{z}})$ is already known.

Residual flows

1. A residual network is a composition of residual connections, which are functions of the form $f(\mathbf{z}) = \mathbf{z} + F(\mathbf{z})$.
2. The function $F : \mathbb{R}^D \mapsto \mathbb{R}^D$ is called the residual block.
3. Under certain conditions on F, the residual connection f becomes invertible.



4. Flows composed of invertible residual connections are referred as residual flows.

Residual flows (Contractive residual blocks)

1. One way to ensure the residual connection is invertible is to choose the residual block to be a contraction.
2. A function F whose Lipschitz constant is less than 1 is called contraction,

   there exists $0 \leq L \leq 1$ such that for all $\mathbf{z}_1$ and $\mathbf{z}_2$ we have:

   $$\|F(\mathbf{z}_1) - F(\mathbf{z}_2)\| \leq L\|\mathbf{z}_1 - \mathbf{z}_2\|$$

3. The invertibility of $f(\mathbf{z}) = \mathbf{z} + F(\mathbf{z})$ can be shown as:
   - Consider the mapping $g(\mathbf{z}) = \mathbf{x} - F(\mathbf{z})$.
   - Because F is a contraction, g is also a contraction.
   - Using Banach's fixed-point theorem, g has has a unique fixed point $\mathbf{z}_*$. Hence,

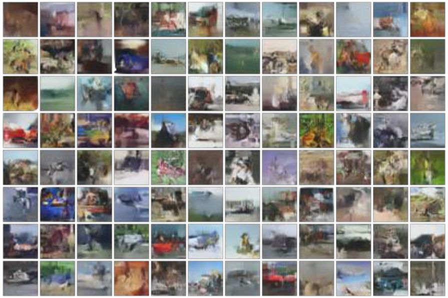
   $$\begin{aligned} \mathbf{z}_* &= \mathbf{x} - F(\mathbf{z}_*) \\ \Longrightarrow& \mathbf{z}_* + F(\mathbf{z}_*) = \mathbf{x} \\ \Longrightarrow& f(\mathbf{z}_*) = \mathbf{x} \end{aligned}$$

   - Since $\mathbf{z}_*$ is **unique**, it follows that $\mathbf{z}_* = f^{-1}(\mathbf{z}_*)$

Residual flows (Contractive residual blocks)

1. The iResNet uses convolutional neural networks as residual blocks, that is, compositions of convolutional layers with non-linear activation functions (Behrmann et al. 2019).
2. Since the Lipschitz constant of a composition is less or equal to the product of the Lipschitz constants of the individual functions, it is enough to ensure the convolutions are contractive, and to use increasing activation functions with slope less or equal to 1.
3. The iResNet model ensures the convolutions are contractive by applying spectral normalization to their weights.
4. there is no analytical expression for the Jacobian determinant, whose exact computation costs $O(D^3)$.
5. However, there is a computationally efficient stochastic estimator of the log Jacobian determinant.
6. The idea is to express the log Jacobian determinant as a power series.

Residual flows (Contractive residual blocks)



CIFAR10 samples.

MNIST samples.

Residual flows (Residual blocks with low-rank Jacobian)

1. There is an efficient way of computing the determinant of a matrix which is a low-rank perturbation of an identity matrix.
2. Let $\mathbf{A}$ and $\mathbf{B}$ be two $D \times M$ and $M \times D$ matrices, respectively. We have

$$\det\left(\mathbf{I}_D + \mathbf{AB}\right) = \det\left(\mathbf{I}_M + \mathbf{BA}\right)$$

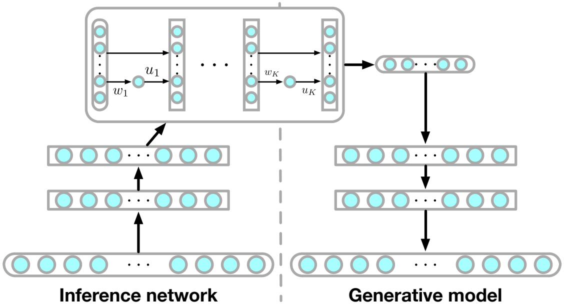
3. The significance of this formula is that it turns a $D \times D$ determinant that costs $O(D^3)$ into an $M \times M$ determinant that costs $O(M^3)$.
4. If $M \ll D$, this saves computation.
5. With some restrictions on the residual block $F : \mathbb{R}^D \mapsto \mathbb{R}^D$, we can apply this formula to compute the determinant of a residual connection efficiently.
6. The trick is to create a bottleneck inside F.

Residual flows (Residual blocks with low-rank Jacobian)

1. The trick is to create a bottleneck inside F.
2. We do that by defining $F = F_2 \circ F_1$, where $F_1 : \mathbb{R}^D \mapsto \mathbb{R}^M$, $F_2 : \mathbb{R}^M \mapsto \mathbb{R}^D$ and $M \ll D$.
3. The chain rule gives $\mathbf{J}_F = \mathbf{J}_{F_2}\mathbf{J}_{F_1}$. Now, we have

$$\det\left(\mathbf{J}_f\right) = \det\left(\mathbf{I}_D + \mathbf{J}_F\right) = \det\left(\mathbf{I}_D + \mathbf{J}_{F_2}\mathbf{J}_{F_1}\right) = \det\left(\mathbf{I}_M + \mathbf{J}_{F_1}\mathbf{J}_{F_2}\right)$$

4. In planar flow, each residual block is an MLP with one hidden layer and one hidden unit (Rezende and Mohamed 2015).



5. The flow is $f(\mathbf{z}) = \mathbf{z} + \mathbf{v}\sigma(\mathbf{w}^\top\mathbf{z} + b)$ and hence $\det\left(\mathbf{J}_f\right) = 1 + \mathbf{w}^\top\mathbf{v}\sigma'(\mathbf{w}^\top\mathbf{z} + b)$

References

Reading

1. Paper Normalizing Flows: An Introduction and Review of Current Methods (Kobyzev, Prince, and Brubaker 2021).
2. Paper Normalizing Flows for Probabilistic Modeling and Inference (Papamakarios, Nalisnick, et al. 2021).
3. Chapter 23 of Probabilistic Machine Learning: Advanced Topics (Murphy 2023).
4. Chapter 3 of Deep Generative Modeling (Tomczak 2022).

References i

- Behrmann, Jens et al. (2019). "Invertible Residual Networks". In: *International Conference on Machine Learning*. Vol. 97, pp. 573–582.
- Dinh, Laurent, David Krueger, and Yoshua Bengio (2015). "NICE: Non-linear Independent Components Estimation". In: *International Conference on Learning Representations*.
- Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density estimation using Real NVP". In: *International Conference on Learning Representations*.
- Kingma, Diederik P. and Prafulla Dhariwal (2018). "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *Advances in Neural Information Processing Systems*, pp. 10236–10245.
- Kingma, Diederik P., Tim Salimans, and Max Welling (2016). "Improving Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems*.
- Kobyzev, Ivan, Simon J. D. Prince, and Marcus A. Brubaker (2021). "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Trans. Pattern Analyses and Machine Intelligence* 43.11, pp. 3964–3979.
- Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics*. The MIT Press.
- Papamakarios, George, Iain Murray, and Theo Pavlakou (2017). "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems*, pp. 2338–2347.
- Papamakarios, George, Eric T. Nalisnick, et al. (2021). "Normalizing Flows for Probabilistic Modeling and Inference". In: *Journal of Machine Learning Research* 22, 57:1–57:64.

References ii

Rezende, Danilo Jimenez and Shakir Mohamed (2015). “Variational Inference with Normalizing Flows”. In: *International Conference on Machine Learning*. Vol. 37, pp. 1530–1538.

Tomczak, Jakub M. (2022). *Deep Generative Modeling*. Springer.

Questions?