An Introduction to Deep Generative Models

Victor Dheur

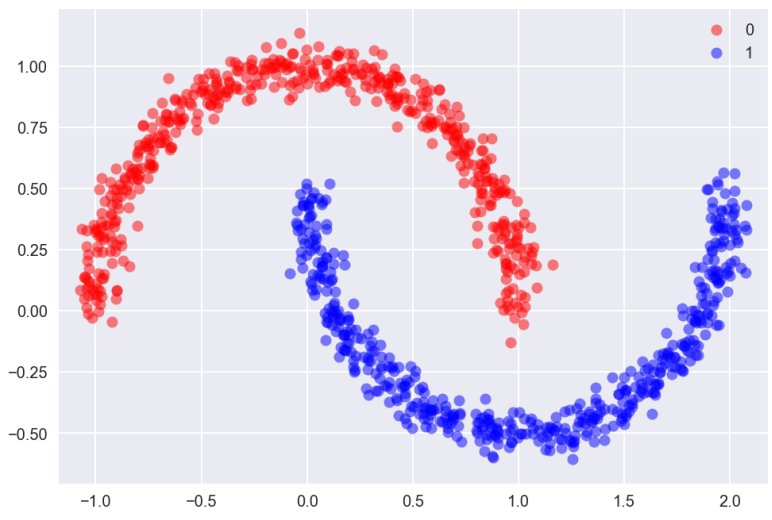
16 March, 2023

Université de Mons

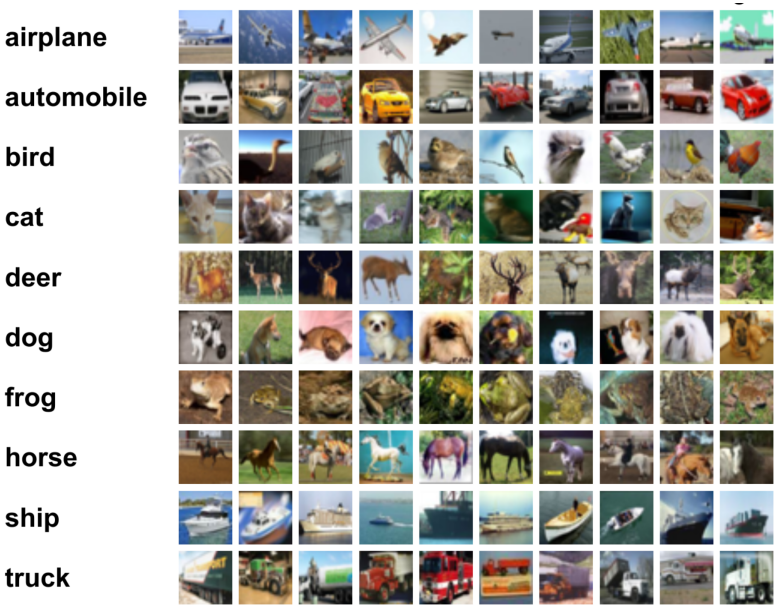
Introduction

A **generative model** is a statistical model of the **probability distribution** $p(\boldsymbol{x})$ over a domain $\mathcal{X} \subseteq \mathbb{R}^D$.

If it is conditioned on inputs $\boldsymbol{c} \in \mathcal{C}$, it becomes a **conditional generative model** $p(\boldsymbol{x} \mid \boldsymbol{c})$.



(a) Two moons dataset ($\mathcal{X} \subseteq \mathbb{R}^2$)



(b) CIFAR-10 dataset ($\mathcal{X} \subseteq \mathbb{R}^{32\times32\times3}$)

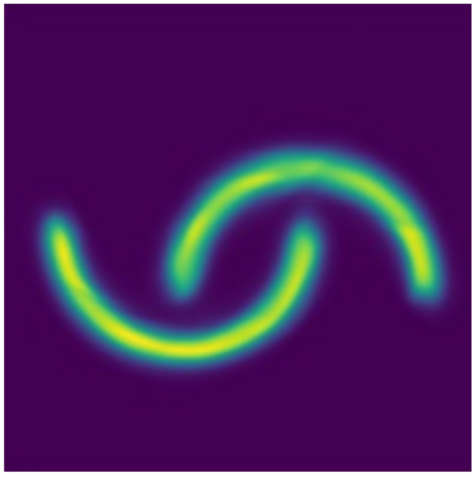
Deep generative models

Deep generative models use a **deep neural network** to learn the probability distribution $p(\boldsymbol{x})$.

In contrast, **shallow generative models** include naives Bayes, linear discriminant analysis or Gaussian mixture models.

Typical applications:

- Data generation
- Density estimation
- Unsupervised representation learning

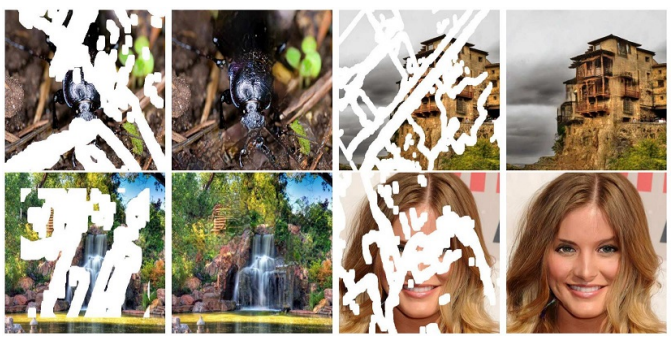


Density estimation using normalizing flows

(Source: [Du+22])

Examples of data generation applications

c	x	Examples of applications
Text	Text	Conversational bots (e.g. ChatGPT), translation, summarization
Text	Image	Digital creation assistant tools
Image	Text	Image search engines, captioning
Image	Image	Inpainting, colorization, style transfer
Text	Speech	Voice assistants
Protein sequence	3D structure	Drug discovery



(a) Inpainting (Source: [Liu+18])

(b) Colorization (Source: [Ant])

Examples of data generation applications

V

Can you give me a structure for a lecture about deep generative models? I would talk about maximum likelihood estimation, autoregressive model, normalizing flows and variational autoencoders.

Sure, here's a possible structure for a lecture on deep generative models, covering maximum likelihood estimation, autoregressive models, normalizing flows, and variational autoencoders:

I. Introduction to Deep Generative Models

- Definition and motivation
- Applications and examples

II. Maximum Likelihood Estimation

- Definition and formulation
- Optimization using gradient descent
- Issues with high-dimensional data and solutions

III. Autoregressive Models

- Definition and intuition
- Example models: PixelCNN

Stop generating

(a) Text-to-text sample (ChatGPT)

(b) Text-to-image sample (Midjourney V4). Prompt: `Belgian super hero black and red and yellow black wings beer lion powers full body.`

Maximum Likelihood Estimation for Deep Generative Models

Setting

A dataset of n observations from $p(\boldsymbol{x})$ is given:

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \right\} \subseteq \mathcal{X}. \tag{1}$$

The standard assumption is that the observations are **independent and identically distributed (i.i.d.)**.

We want to select parameters θ such that the distribution $p_\theta(\boldsymbol{x})$ is a good approximation to $p(\boldsymbol{x})$.

Maximum likelihood estimation

The maximum likelihood estimator is given by:

$$
\begin{aligned}
\arg\max_\theta \mathcal{L}(\theta) &= \arg\max_\theta p_\theta(\mathcal{D}) && \text{(Definition of likelihood)} \\
&= \arg\max_\theta \prod_{i=1}^{n} p_\theta(\boldsymbol{x}^{(i)}) && \text{(Independence)} \\
&= \arg\max_\theta \log \prod_{i=1}^{n} p_\theta(\boldsymbol{x}^{(i)}) && \text{(log is increasing)} \\
&= \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\boldsymbol{x}^{(i)}) && \text{(log of product)} \\
&= \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(\boldsymbol{x}^{(i)}) && (x \mapsto -x/n \text{ is decreasing})
\end{aligned}
$$

MLE as KL divergence minimization

By the weak law of large numbers:

$$\frac{1}{n}\sum_{i=1}^{n} -\log p_\theta(\boldsymbol{x}^{(i)}) \xrightarrow{p} \underbrace{\mathbb{E}_{p(\boldsymbol{x})}[-\log p_\theta(\boldsymbol{x})]}_{\text{Expected NLL}} \quad \text{when } n \to \infty.$$

We can think of NLL minimization as trying to minimize the **KL divergence** between the true distribution $p(\boldsymbol{x})$ and the estimated distribution $p_\theta(\boldsymbol{x})$:

$$\begin{aligned}
\arg\min_\theta \mathbb{E}_{p(\boldsymbol{x})}[-\log p_\theta(\boldsymbol{x})] &= \arg\min_\theta \mathbb{E}_{p(\boldsymbol{x})}[\log p(\boldsymbol{x}) - \log p_\theta(\boldsymbol{x})] \\
&= \arg\min_\theta \mathbb{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right] \\
&= \arg\min_\theta D_{\mathrm{KL}}(p(\boldsymbol{x}) \parallel p_\theta(\boldsymbol{x}))).
\end{aligned}$$

More data ($n \to \infty$) leads to a better approximation.

Properties of KL divergence

Basic properties of the **Kullback-Leibler (KL) divergence**:

$$D_{\mathrm{KL}}(p \parallel q) = \mathbb{E}_{p(x)}\left[\log \frac{p(x)}{q(x)}\right] = \int p(x) \log \frac{p(x)}{q(x)} dx.$$

Note that $D_{\mathrm{KL}}(p \parallel q) \geq 0$ for any p, q:

$$\begin{aligned}
D_{\mathrm{KL}}(p \parallel q) &= -\mathbb{E}_{p(x)}\left[\log \frac{q(x)}{p(x)}\right] \\
&\geq -\log \mathbb{E}_{p(x)}\left[\frac{q(x)}{p(x)}\right] && \text{(Jensen's inequality)} \\
&= -\log \int p(x) \frac{q(x)}{p(x)} dx \\
&= -\log 1 && \text{(integral of PDF)} \\
&= 0.
\end{aligned}$$

If $p = q$, $D_{\mathrm{KL}}(p \parallel q) = 0$.

Conditional MLE

In many cases, the probability distribution $p(\boldsymbol{x})$ is conditioned on some input $\boldsymbol{c} \in \mathcal{C}$.

Then, the dataset consists in n pairs of observations from $p(\boldsymbol{x}, \boldsymbol{c})$:

$$\mathcal{D} = \left\{ (\boldsymbol{x}^{(1)}, \boldsymbol{c}^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, \boldsymbol{c}^{(n)}) \right\}.$$

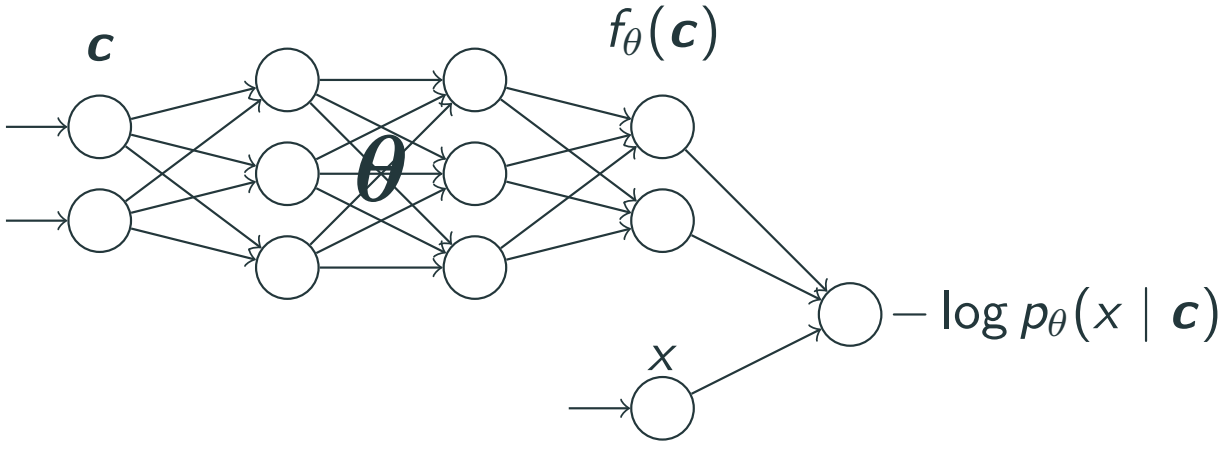
The standard assumption is that $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$ are **i.i.d. conditionally to** $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(n)}$.

Using this assumption, the maximum likelihood estimator is given by:

$$\begin{aligned} \arg\max_{\theta} \mathcal{L}(\theta) &= \arg\max_{\theta} p_\theta(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} \mid \boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(n)}) \\ &= \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{c}^{(i)}). \end{aligned}$$

MLE with neural networks

In practice, how can we perform MLE with neural networks when $D = 1$?



Optimization using gradient descent

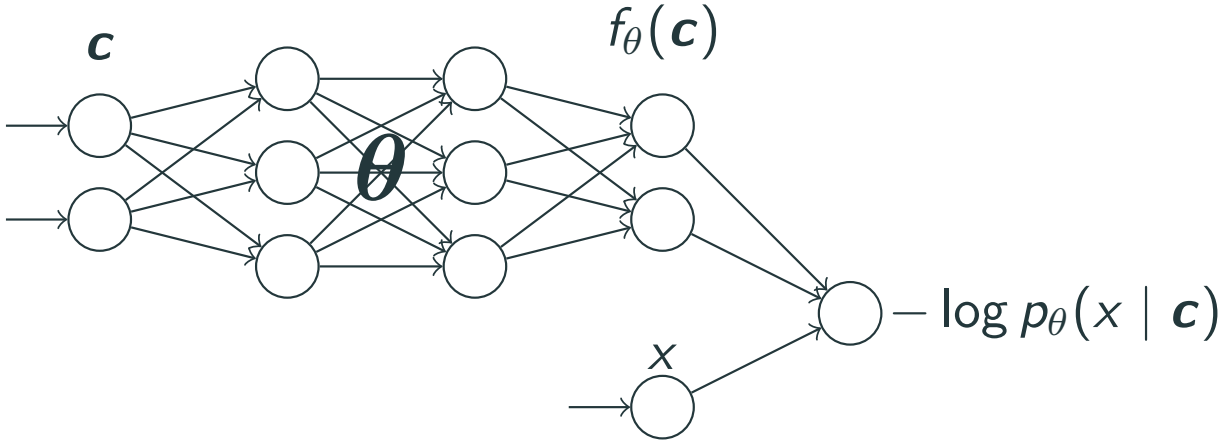
Successively update the parameters:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(x^{(i)} \mid \boldsymbol{c}^{(i)}),$$

where $\eta > 0$ is the learning rate.

MLE with neural networks

In practice, how can we perform MLE with neural networks when $D = 1$?



MLE for a categorical distribution

Suppose that $f_\theta(\boldsymbol{c}) = \boldsymbol{l} \in \mathbb{R}^K$ (we omit c and θ for ease of notation).

We can create a categorical distribution $\boldsymbol{w}$ using the softmax function:

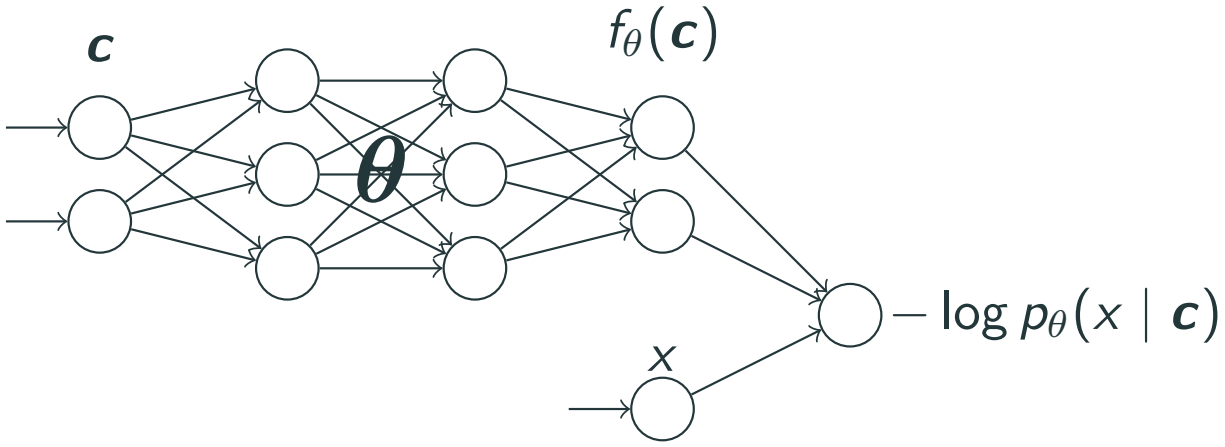
$$\boldsymbol{w}_i = \frac{e^{l_i}}{\sum_{j=1}^{K} e^{l_j}} \quad \text{for } i = 1, \ldots, K.$$

You can check that $\boldsymbol{w}_i \geq 0$ and $\sum_{i=1}^{K} \boldsymbol{w}_i = 1$.

Then: $-\log p_\theta(x \mid \boldsymbol{c}) = -\log \boldsymbol{w}_x$.

MLE with neural networks

In practice, how can we perform MLE with neural networks when $D = 1$?



MLE for a normal distribution

Suppose that $f_\theta(\boldsymbol{c}) = (\mu, \rho)$ where $\mu \in \mathbb{R}$ and $\rho \in \mathbb{R}$ (we omit c and θ for ease of notation).

We can create a positive standard deviation σ using the softplus function:

$$\sigma = \log(1 + e^\rho) > 0.$$

Then: $-\log p_\theta(x \mid \boldsymbol{c}) = -\log \mathcal{N}(x; \mu, \sigma^2)$.

Auto-regressive models (ARMs)

Autoregressive models

How to represent the density of a **high dimensional** random variable $\boldsymbol{x} \in \mathbb{R}^D$ where $D \gg 1$?

We can use the **product rule**:

$$\begin{aligned} p(\boldsymbol{x}) &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2, x_1)\ldots \\ &= \prod_{d=1}^{D} p(x_d \mid x_1, \ldots, x_{d-1}) \\ -\log p(\boldsymbol{x}) &= \sum_{d=1}^{D} -\log p(x_d \mid x_1, \ldots, x_{d-1}). \end{aligned}$$

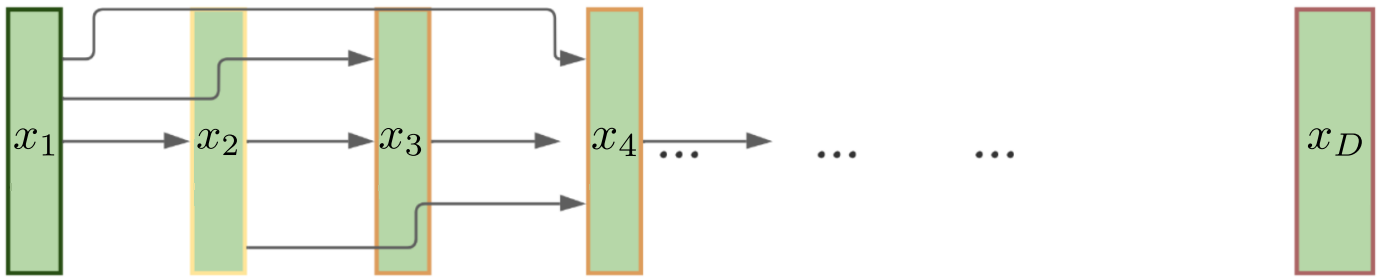
This is the idea behind **autoregressive models**.

Autoregressive neural networks

For a given dimension d, we modelize the conditional density using a **neural network** $p_\theta(x_d \mid x_1, \ldots, x_{d-1})$.

$p_\theta(x_d \mid x_1, \ldots, x_{d-1})$ could be, for example, a categorical distribution or a normal distribution.

By doing so, we reduce the multidimensional estimation problem to a **unidimensional** one, which does not suffer from the curse of dimensionality.



Adapted from [Mur23]

Naive density evaluation and sampling

Given an auto-regressive model with parameters θ, how to evaluate its density $p_\theta(\boldsymbol{x})$ and sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$?

Density evaluation:

- Evaluate $p_\theta(x_1)$
- Evaluate $p_\theta(x_2 \mid x_1)$
- ...
- Evaluate $p_\theta(x_D \mid x_1, \ldots, x_{D-1})$

Sampling:

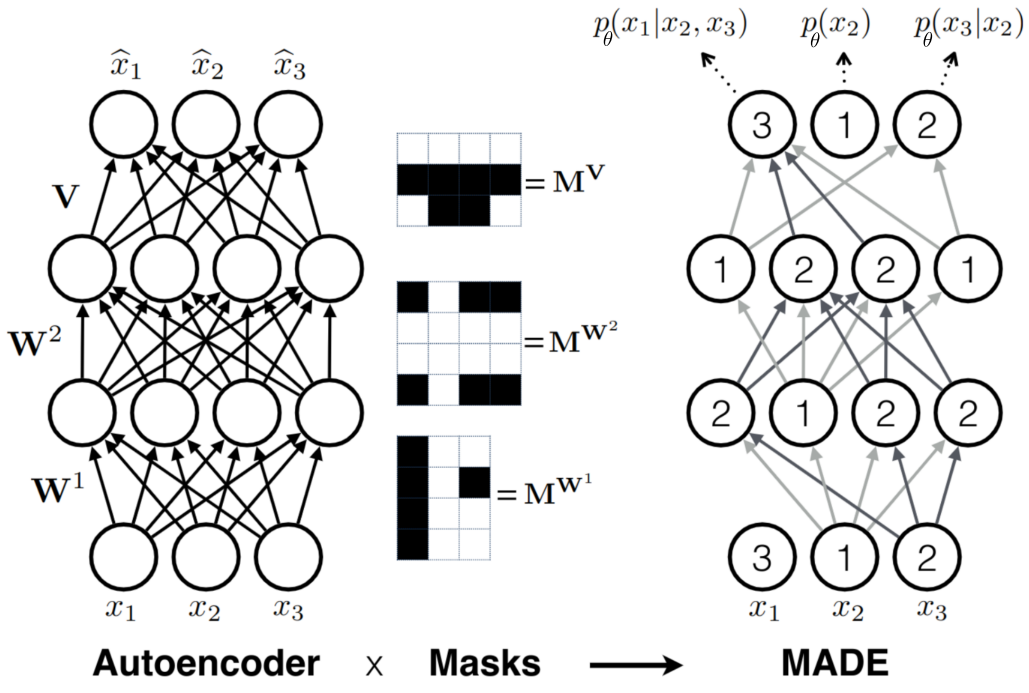
- Sample $x_1 \sim p_\theta(x_1)$
- Sample $x_2 \sim p_\theta(x_2 \mid x_1)$
- ...
- Sample $x_D \sim p_\theta(x_D \mid x_1, \ldots, x_{D-1})$

Both operations require D neural network forward passes. :-(

Masked Autoencoder for Distribution Estimation (MADE)

MADE adapts autoencoders for distribution estimation:

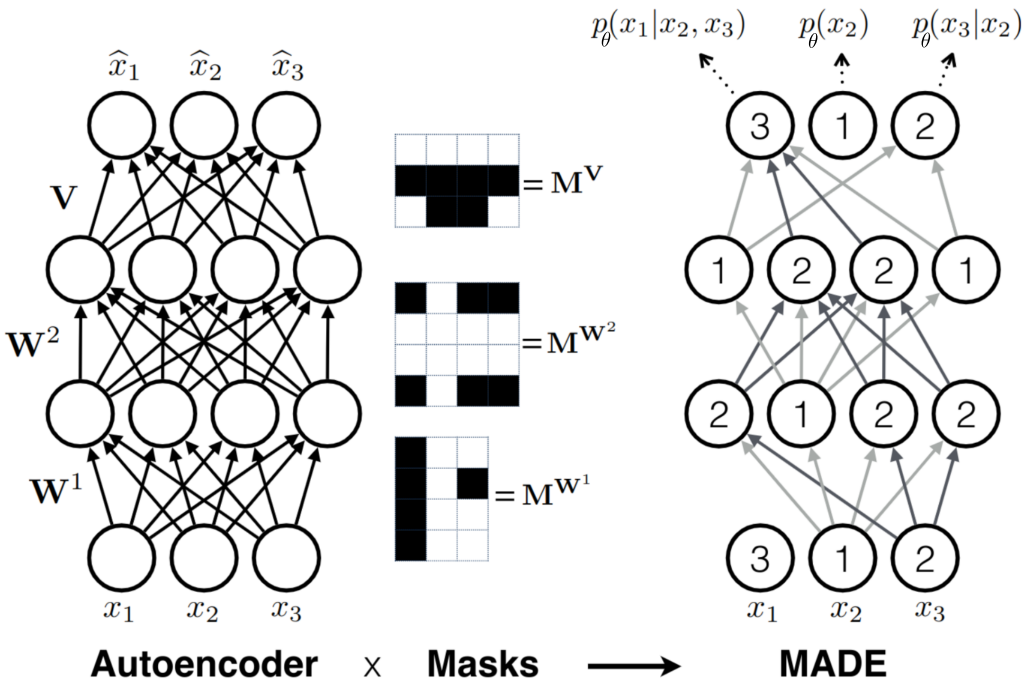
- Autoencoders normally output a reconstruction $\hat{\boldsymbol{x}}$ given an input $\boldsymbol{x}$.
- MADE instead outputs **conditional distributions** $p_\theta(x_d \mid x_1, \ldots, x_{d-1})$.
- MADE allows different orderings, e.g., x_2, x_3, x_1.



Adapted from [Ger+15]

Masked Autoencoder for Distribution Estimation (MADE)

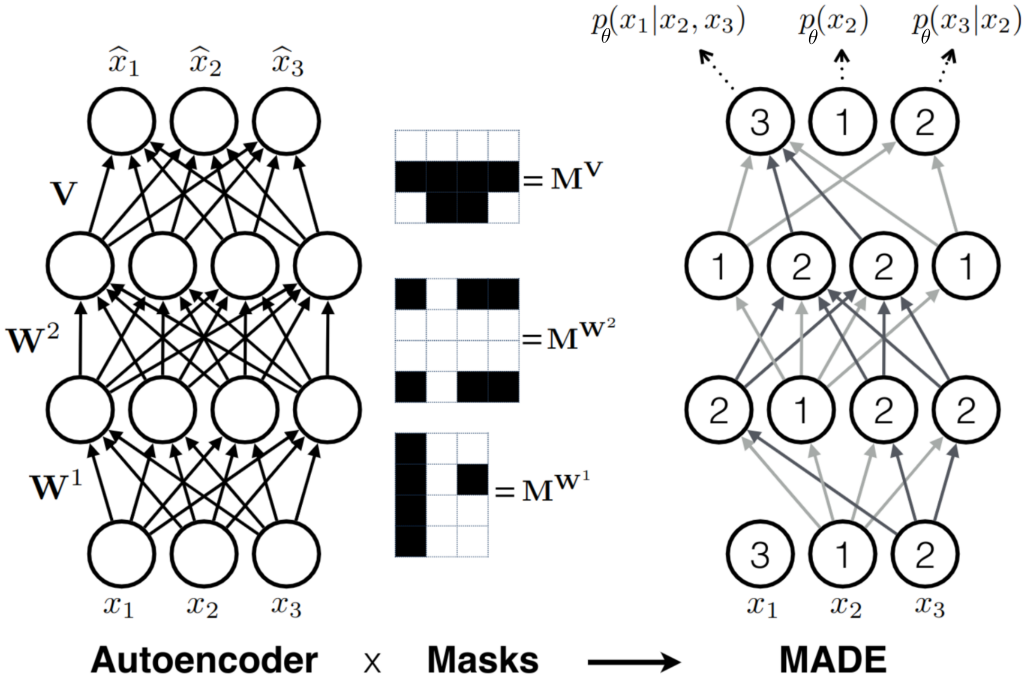
- A random assignment $1 \leq m(k) < D$ is given to each hidden unit k.
- The unit k is only allowed to depend on the first $m(k)$ inputs (according to the ordering).
- **Masked connections** allow to preserve this invariant: each hidden unit k is connected to all units k' in the previous layer such that $m(k) \geq m(k')$.
- In the final layer, it is required that $m(k) > m(k')$.



Adapted from [Ger+15]

Masked Autoencoder for Distribution Estimation (MADE)

- **Only one forward pass** suffices to evaluate $p_\theta(x_1)$, $p_\theta(x_2 \mid x_1)$, ..., and $p_\theta(x_D \mid x_1, \ldots, x_{D-1})$. :-)
  - Training becomes a lot more efficient.
- Sampling still requires D forward passes.

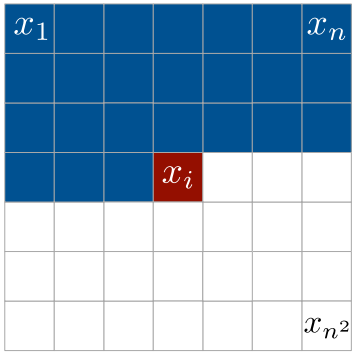


Adapted from [Ger+15]

Further readings: PixelCNN

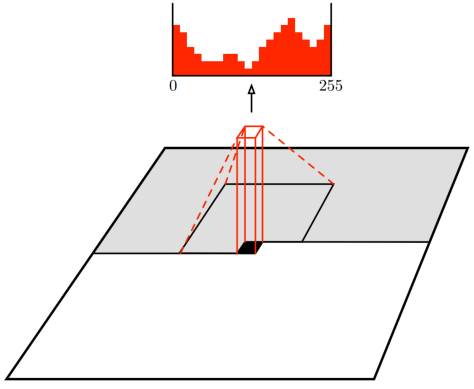
Autoencoders based on CNNs can also perform distribution estimation efficiently.

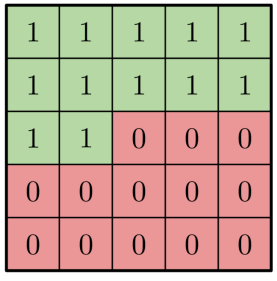
Masked convolutions allow to ensure proper conditionals.



ARMs require to define an order on the pixels.

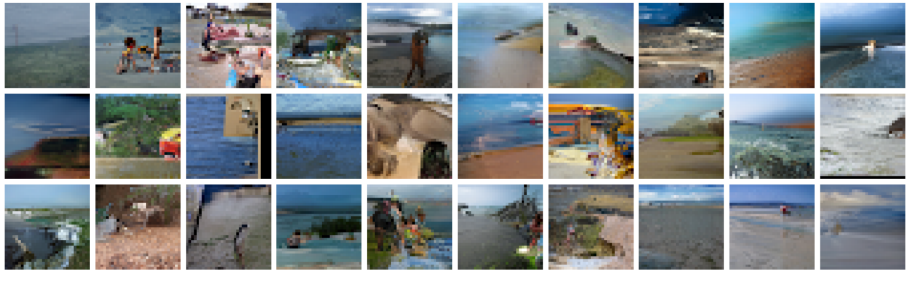
Source: [OKK16]





PixelCNN produces a distribution over the next pixel.

(Source: [Oor+16])



Samples from a PixelCNN. (Source: [Oor+16])

Pros and cons

All sorts of other architectures allow to create proper conditionals:

- **Recurrent neural networks**
- **Causal convolutions** in 1D convolutional neural networks
- **Masked self-attention** in transformers
  - At the core of most Large Language Models (e.g., ChatGPT)

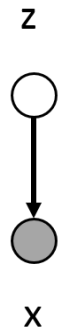
Pros	Cons
• Explicit likelihood $p(\boldsymbol{x})$	• Slow sampling
• Stable training	• No natural way to do unsupervised learning
• Good samples	

Variational autoencoders

Latent variable models

We assume that the training data $\left\{ \boldsymbol{x}^{(i)} \right\}_{i=1}^{n}$ is generated from an unobserved (latent) representation $\boldsymbol{z} \in \mathcal{Z}$ using the following data generating process:

1. Sample from the prior: $\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z})$
2. Sample from the conditional: $\boldsymbol{x}^{(i)} \sim p(\boldsymbol{x} \mid \boldsymbol{z}^{(i)})$



$\mathcal{Z}$ is called the latent space and is often of smaller dimension than $\mathcal{X}$.

Intuitively, if $\mathcal{X} = \mathbb{R}^{128 \times 128 \times 3}$ is a space with images of faces, it can be represented by simple attributes such as the orientation or degree of smile in $\mathcal{Z} = \mathbb{R}^{50}$.

Latent variable models

The prior $p(\boldsymbol{z})$ is known (e.g., $p(\boldsymbol{z}) = \mathcal{N}(0, I)$), and the conditional is estimated using a decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$.

What is the (marginal) likelihood?

$$\begin{aligned} p_\theta(\boldsymbol{x}) &= \int p(\boldsymbol{z}) p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) d\boldsymbol{z} \\ &= \mathbb{E}_{p(\boldsymbol{z})}[p_\theta(\boldsymbol{x} \mid \boldsymbol{z})] \end{aligned}$$

Problem: the integral is intractable!

We can use Monte Carlo estimation:

$$p_\theta(\boldsymbol{x}) \approx \frac{1}{K} \sum_{j=1}^{K} p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(j)}) \quad \text{where } \boldsymbol{z}^{(j)} \sim p(\boldsymbol{z}).$$

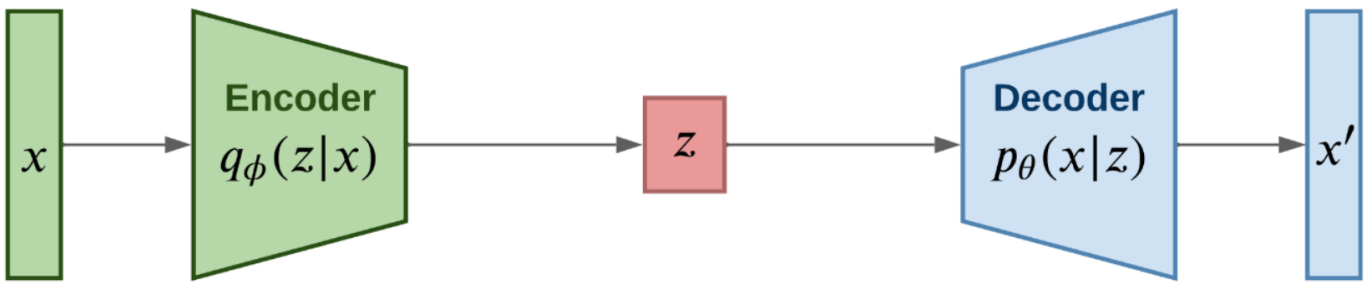
However, if $\boldsymbol{z}$ is high-dimensional, a large sample is required due to the curse of dimensionality.

Variational autoencoders

Inspired by autoencoders, **variational autoencoders** learn to:

1. Map $\boldsymbol{x}$ to a latent distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ using an **encoder** with parameters ϕ.
2. Map $\boldsymbol{z}$ to the posterior distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ using a **decoder** with parameters θ.

The encoder and decoder are typically trained together.



Source: [Mur23]

$q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is called the **variational distribution** and allows to estimate a lower bound on $\log p_\theta(\boldsymbol{x}^{(i)})$.

Evidence Lower BOund (ELBO)

$$
\begin{aligned}
&\log p_\theta(\boldsymbol{x}^{(i)}) \\
&= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_\theta(\boldsymbol{x}^{(i)})\right] \quad (p_\theta(\boldsymbol{x}^{(i)}) \text{ does not depend on } \boldsymbol{z}) \\
&= \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}\right] \quad \text{(Bayes rule)} \\
&= \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}\right] \quad \text{(Multiply by 1)} \\
&= \mathbb{E}_{\boldsymbol{z}}\left[\log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})\right] - \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}{p(\boldsymbol{z})}\right] + \mathbb{E}_{\boldsymbol{z}}\left[\log \frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})}\right] \quad \text{(Logarithms)} \\
&= \underbrace{\mathbb{E}_{\boldsymbol{z}}\left[\log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})\right]}_{\text{-Reconstruction loss}} - \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}) \parallel p(\boldsymbol{z}))}_{\text{Complexity loss } (\geq 0)} + \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}) \parallel p_\theta(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}))}_{\text{Latent loss } (\geq 0) \text{ (intractable)}}
\end{aligned}
$$

Evidence Lower BOund (ELBO)

$$
\begin{aligned}
&\log p_\theta(x^{(i)}) \\
&= \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ does not depend on } z) \\
&= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes rule)} \\
&= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by 1)} \\
&= \mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbb{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)} \\
&= \underbrace{\mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]}_{\text{-Reconstruction loss}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x^{(i)}) \parallel p(z))}_{\text{Complexity loss } (\geq 0)} + \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x^{(i)}) \parallel p_\theta(z \mid x^{(i)}))}_{\text{Latent loss } (\geq 0) \text{ (intractable)}} \\
&\geq \underbrace{\mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{\mathrm{KL}}(q_\phi(z \mid x^{(i)}) \parallel p(z))}_{\text{ELBO (Evidence Lower BOund)}}
\end{aligned}
$$

Variational autoencoders

We minimize the negative ELBO:

$$
\begin{aligned}
& -\log p_\theta(\boldsymbol{x}^{(i)}) \\
&= \underbrace{\underbrace{\mathbb{E}_{\boldsymbol{z}}\Big[-\log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})\Big]}_{\text{Reconstruction loss}} + \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}) \parallel p(\boldsymbol{z}))}_{\text{Complexity loss } (\geq 0)}}_{\text{Negative ELBO}} \underbrace{- D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}) \parallel p_\theta(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}))}_{\text{-Latent loss } (\geq 0) \text{ (intractable)}}
\end{aligned}
$$

The **decoder** optimizes the **reconstruction loss**: it learns parameters θ that reconstruct $\boldsymbol{x}$ from $\boldsymbol{z}$ using the NLL conditional to $\boldsymbol{z}$.

The **encoder** optimizes the **complexity loss**: it learns parameters ϕ so that the variational distribution is close to the prior.

The **encoder** indirectly optimizes the **latent loss**: assuming that θ is fixed, since $-\log p_\theta(\boldsymbol{x}^{(i)})$ does not depend on ϕ, minimizing the negative ELBO with respect to ϕ will minimize the **latent loss**.

Reparameterization trick

We have to be careful when computing the gradient of the reconstruction loss:

$$\nabla_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\Big[-\log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})\Big] \neq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\Big[-\nabla_\phi \log p_\theta(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})\Big]$$

Reparametrization trick

Suppose that the encoder parameterizes a normal distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}) = \mathcal{N}(\mu_\phi(\boldsymbol{x}^{(i)}), \sigma^2_\phi(\boldsymbol{x}^{(i)}))$.
We can sample $\boldsymbol{z}$ using:

1. Sample $\epsilon \sim \mathcal{N}(0, I)$
2. Compute $\boldsymbol{z} = \mu_\phi(\boldsymbol{x}^{(i)}) + \epsilon\sigma_\phi(\boldsymbol{x}^{(i)})$

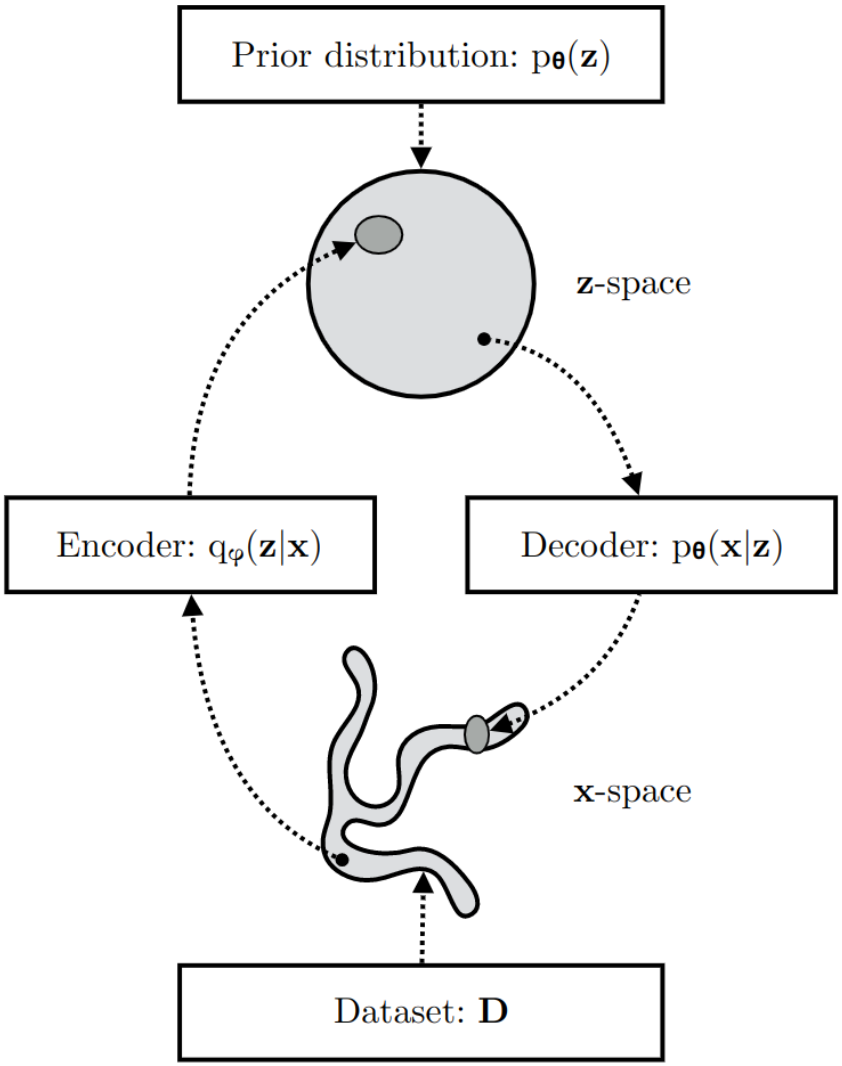
$\Rightarrow$ We can use:

$$\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\Big[-\nabla_\phi \log p_\theta\left(\boldsymbol{x}^{(i)}\middle|\mu_\phi(\boldsymbol{x}^{(i)}) + \epsilon\sigma_\phi(\boldsymbol{x}^{(i)})\right)\Big]$$

Variational autoencoder recap

Once the model is trained, we can sample using the data generating process:

1. Sample from the prior: $\boldsymbol{z} \sim p(\boldsymbol{z})$
2. Sample from the conditional: $\boldsymbol{x} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$



Source: [Kin17]

Examples of predictions on a latent space

Example of interpolations on a 2D latent space ($\mathcal{Z} = \mathbb{R}^2$)

Source: [Kin17]

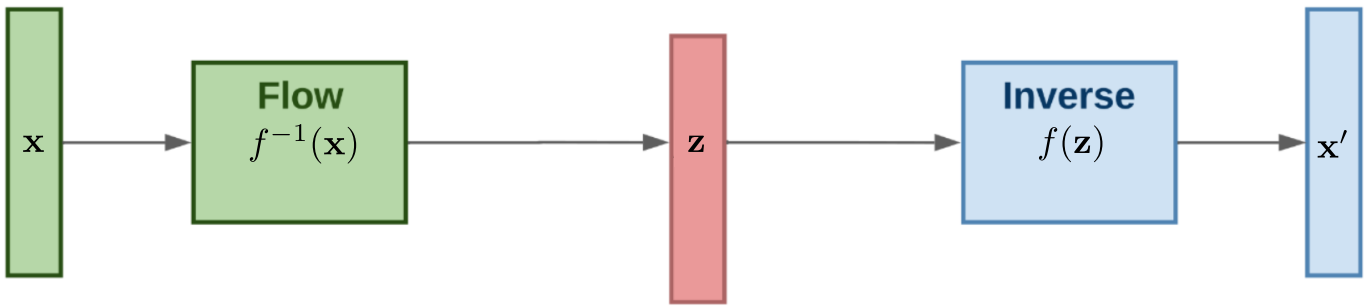
Pros and cons

Pros	Cons
• Principled approach • Interpretable latent space • Allows unsupervised learning • Fast sampling	• Maximizes a lower bound on the likelihood • Samples are more blurry

Normalizing flows

Normalizing flows

Normalizing flows create complex distributions $p(\boldsymbol{x})$ by passing another random variable $\mathrm{z} \in \mathbb{R}^D$ through a nonlinear but **invertible** transformation $f : \mathbb{R}^D \to \mathbb{R}^D$.



Adapted from [Mur23]

As VAEs, they are latent variable models, i.e., $p(\boldsymbol{x})$ is defined by:

$$\boldsymbol{x} = f(\boldsymbol{z}) \quad \text{where } \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}).$$

To determine $p(\boldsymbol{x})$ and compute the likelihood, we have to use the change of variable formula.

Change of variable formula

Let $z \sim \mathcal{U}(0, 2)$ be a uniform random variable between 0 and 2.

What is $p_z(1)$?

The density is constant, we have $\int_0^2 \frac{1}{2} dz' = 1$, so $p_z(1) = \frac{1}{2}$.

Let $x = 4z$. **What is $p_x(4)$?**

We have to use the change of variable formula.

Change of variable formula (1D case)

Let z be a random variable and $x = f(z)$ where f is strictly increasing and differentiable with inverse g.
Then, $p_x(x) = p_z(g(x)) \frac{\partial g(x)}{\partial x}$.

Let $g(x) = \frac{x}{4}$. Then $p_x(4) = p_z(g(4)) \frac{\partial g(x)}{\partial x} = \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}$.

Change of variable formula: proof (1D case)

$$
\begin{aligned}
F_x(y) &= P(x \leq y) \\
&= P(f(z) \leq y) \\
&= P(z \leq g(y)) && (g \text{ strictly increasing}) \\
&= F_z(g(y))
\end{aligned}
$$

Taking the derivative on both sides:

$$
\begin{aligned}
\frac{\partial F_x(y)}{\partial y} &= \frac{\partial F_z(g(y))}{\partial y} \\
p_x(y) &= p_z(g(y)) \frac{\partial g(y)}{\partial y} && (\text{chain rule})
\end{aligned}
$$

Example: log-normal distribution

Let $z \sim \mathcal{N}(\mu, \sigma^2)$ be a **normally distributed** random variable and $f(z) = \exp(z)$.

By definition, $x = f(z)$ has a **log-normal** distribution.

We can deduce:

- $g(x) = f^{-1}(x) = \log(x)$.
- $\frac{\partial g(x)}{\partial x} = \frac{1}{x}$.

Since f is strictly increasing, we can use the change of variable formula:

$$\begin{aligned} p_x(x) &= p_z(g(x)) \frac{\partial g(x)}{\partial x} \\ &= p_z(\log(x))/x. \end{aligned}$$

Change of variable formula (multidimensional case)

Change of variable formula (multidimensional case)

Let $f : \mathbb{R}^D \to \mathbb{R}^D$ be a bijection (i.e., it has an inverse $g = f^{-1}$).
Let $\boldsymbol{z} \in \mathbb{R}^D$ be a random variable and $\boldsymbol{x} = f(\boldsymbol{z}) \in \mathbb{R}^D$. Then,

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(g(\boldsymbol{x})) \left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|.$$

where $\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial g(x_1)}{\partial x_1} & \cdots & \frac{\partial g(x_1)}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial g(x_D)}{\partial x_1} & \cdots & \frac{\partial g(x_D)}{\partial x_D} \end{bmatrix}$ is the Jacobian of g.

The proof is more complicated and we don't show it here.

Intuition behind $\left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|$: the magnitude of "volume change" in $\boldsymbol{x}$.

The density $p_{\boldsymbol{x}}$ can be computed from the density $p_{\boldsymbol{z}}$ and the inverse function g. :-)

Training objective

Since we have the likelihood in closed form, we can optimize the NLL:

$$-\log p_\theta(\boldsymbol{x}^{(i)}) = -\log p_z(g_\theta(\boldsymbol{x}^{(i)})) - \log \left| \det \left(\frac{\partial g_\theta(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right) \right|.$$

Then we can sample $\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$ by:

1. Sample from the latent space: $\boldsymbol{z} \sim p_z(\boldsymbol{z})$
2. Apply the forward transformation: $\boldsymbol{x} = f_\theta(\boldsymbol{z})$

Challenges

We want multiple things at once:

- g_θ must be invertible.
- g_θ and $\det \left(\frac{\partial g_\theta(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} \right)$ should be fast to compute for fast training.
- f_θ should be fast to compute for fast sampling.

Elementwise flows

Elementwise flows are applied independently to each dimension.

Let $f(\boldsymbol{z}) = (f_1(z_1), \ldots, f_D(z_D))$ where each f_d is invertible with inverse $g_d = f_d^{-1}$.

A simple example is the elementwise affine flow, with $f_d(z_d) = a_d z_d + b_d$.

Then $g_d(x_d) = (x_d - b_d)/a_d$.

Jacobian and determinant of reverse mapping are:

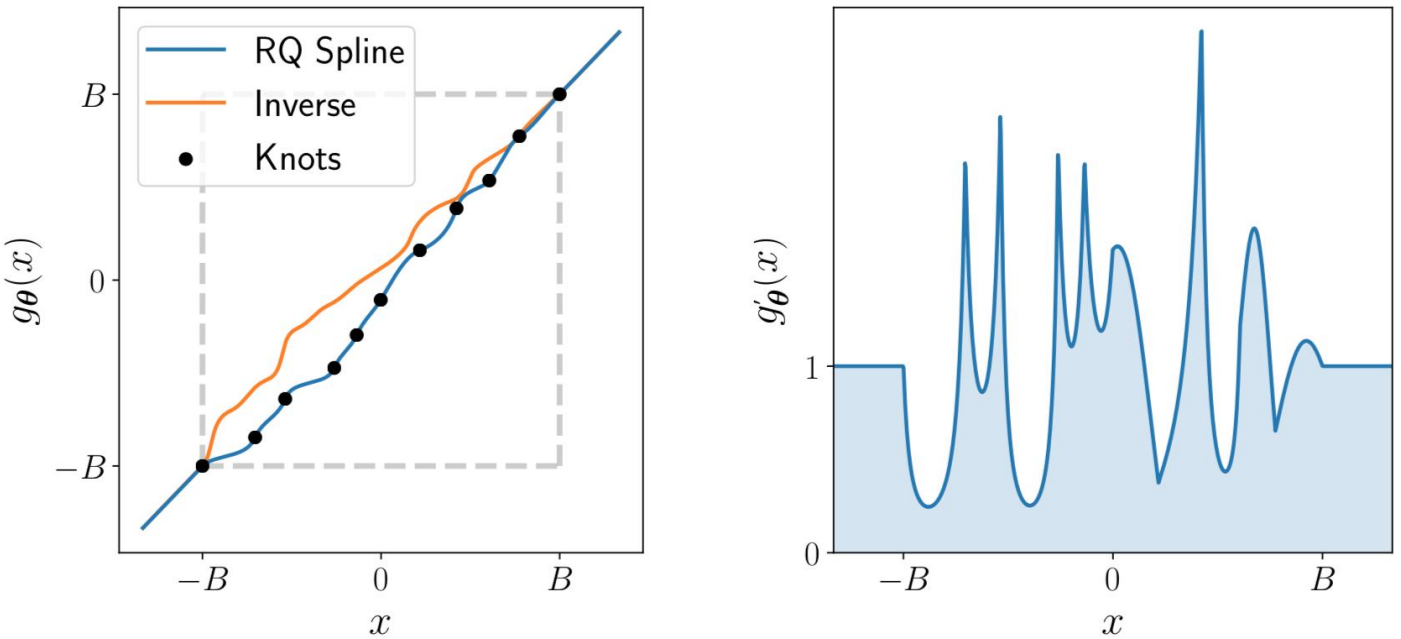
$$\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} 1/a_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 1/a_D \end{bmatrix} \quad \det\left(\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{d=1}^{D} 1/a_d$$

Next, we introduce a flow that allows mixing between dimensions.

Further readings: Neural Spline Flows [DBM+19]

Neural Spline Flows are flexible elementwise flows.

The forward function f_d, inverse function g_d and derivative $\frac{\partial g_d(x_d)}{\partial x_d}$ are all computable in closed form.



Source: [DBM+19]

Coupling flows

Partition $\boldsymbol{z}$ in two subsets $\boldsymbol{z}_{1:d}$ and $\boldsymbol{z}_{d+1:D}$ where $1 \leq d \leq D$.

Forward mapping f_θ:

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \\ \boldsymbol{x}_{d+1:D} = \boldsymbol{z}_{d+1:D} \odot \exp\big(\alpha_\theta(\boldsymbol{z}_{1:d})\big) + \mu_\theta(\boldsymbol{z}_{1:d}) \end{cases}$$

$\mu_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ can be **any neural network**!

Inverse mapping $g_\theta = f_\theta^{-1}$:

$$\begin{cases} \boldsymbol{z}_{1:d} = \\ \boldsymbol{z}_{d+1:D} = \end{cases}$$

Coupling flows

Partition $\boldsymbol{z}$ in two subsets $\boldsymbol{z}_{1:d}$ and $\boldsymbol{z}_{d+1:D}$ where $1 \leq d \leq D$.

Forward mapping f_θ:

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \\ \boldsymbol{x}_{d+1:D} = \boldsymbol{z}_{d+1:D} \odot \exp(\alpha_\theta(\boldsymbol{z}_{1:d})) + \mu_\theta(\boldsymbol{z}_{1:d}) \end{cases}$$

$\mu_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ can be **any neural network**!

Inverse mapping $g_\theta = f_\theta^{-1}$:

$$\begin{cases} \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \\ \boldsymbol{z}_{d+1:D} = (\boldsymbol{x}_{d+1:D} - \mu_\theta(\boldsymbol{x}_{1:d})) \odot \exp(-\alpha_\theta(\boldsymbol{x}_{1:d})) \end{cases}$$

Coupling flows

Partition $\boldsymbol{z}$ in two subsets $\boldsymbol{z}_{1:d}$ and $\boldsymbol{z}_{d+1:D}$ where $1 \leq d \leq D$.

Forward mapping f_θ:

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \\ \boldsymbol{x}_{d+1:D} = \boldsymbol{z}_{d+1:D} \odot \exp(\alpha_\theta(\boldsymbol{z}_{1:d})) + \mu_\theta(\boldsymbol{z}_{1:d}) \end{cases}$$

$\mu_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ can be **any neural network**!

Inverse mapping $g_\theta = f_\theta^{-1}$:

$$\begin{cases} \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \\ \boldsymbol{z}_{d+1:D} = (\boldsymbol{x}_{d+1:D} - \mu_\theta(\boldsymbol{x}_{1:d})) \odot \exp(-\alpha_\theta(\boldsymbol{x}_{1:d})) \end{cases}$$

Jacobian and determinant of reverse mapping:

$$\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} =$$

$$\det \left(\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) =$$

Coupling flows

Partition $\boldsymbol{z}$ in two subsets $\boldsymbol{z}_{1:d}$ and $\boldsymbol{z}_{d+1:D}$ where $1 \leq d \leq D$.

Forward mapping f_θ:

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \\ \boldsymbol{x}_{d+1:D} = \boldsymbol{z}_{d+1:D} \odot \exp(\alpha_\theta(\boldsymbol{z}_{1:d})) + \mu_\theta(\boldsymbol{z}_{1:d}) \end{cases}$$

$\mu_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ can be **any neural network**!

Inverse mapping $g_\theta = f_\theta^{-1}$:

$$\begin{cases} \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \\ \boldsymbol{z}_{d+1:D} = (\boldsymbol{x}_{d+1:D} - \mu_\theta(\boldsymbol{x}_{1:d})) \odot \exp(-\alpha_\theta(\boldsymbol{x}_{1:d})) \end{cases}$$

Jacobian and determinant of reverse mapping:

$$\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} I_d & 0 \\ \frac{\partial \boldsymbol{z}_{d+1:D}}{\partial \boldsymbol{x}_{1:d}} & \operatorname{diag}(\exp(-\alpha_\theta(\boldsymbol{x}_{1:d}))) \end{bmatrix}$$

$$\det\left(\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{i=d+1}^{D} \exp(-\alpha_\theta(\boldsymbol{x}_{1:d}))_i$$

Coupling flows

Partition $\boldsymbol{z}$ in two subsets $\boldsymbol{z}_{1:d}$ and $\boldsymbol{z}_{d+1:D}$ where $1 \leq d \leq D$.

Forward mapping f_θ:

$$\begin{cases} \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \\ \boldsymbol{x}_{d+1:D} = \boldsymbol{z}_{d+1:D} \odot \exp(\alpha_\theta(\boldsymbol{z}_{1:d})) + \mu_\theta(\boldsymbol{z}_{1:d}) \end{cases}$$

$\mu_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^{D-d}$ can be **any neural network**!

Inverse mapping $g_\theta = f_\theta^{-1}$:

$$\begin{cases} \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \\ \boldsymbol{z}_{d+1:D} = (\boldsymbol{x}_{d+1:D} - \mu_\theta(\boldsymbol{x}_{1:d})) \odot \exp(-\alpha_\theta(\boldsymbol{x}_{1:d})) \end{cases}$$

Jacobian and determinant of reverse mapping:

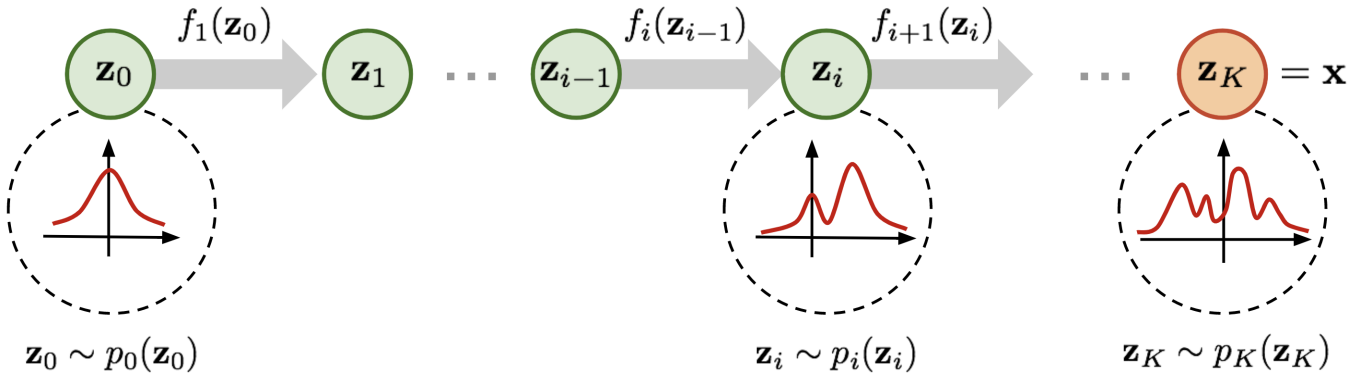
$$\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} I_d & 0 \\ \frac{\partial \boldsymbol{z}_{d+1:D}}{\partial \boldsymbol{x}_{1:d}} & \operatorname{diag}(\exp(-\alpha_\theta(\boldsymbol{x}_{1:d}))) \end{bmatrix}$$

$$\det\left(\frac{\partial g_\theta(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \prod_{i=d+1}^{D} \exp(-\alpha_\theta(\boldsymbol{x}_{1:d}))_i$$

Composition of flows

Let $f_1, \ldots, f_K$ be invertible transformations with inverse $g_1, \ldots, g_K$.

Then the composition $f = f_K \circ \cdots \circ f_1$ is also invertible with inverse $g = g_1 \circ \cdots \circ g_K$.



Source: [Wen18]

$$p_{z_K}(z_K) = p_{z_{K-1}}(z_{K-1}) \left| \det \left(\frac{\partial z_{K-1}}{\partial z_K} \right) \right| \qquad \text{where } z_{K-1} = g_K(z_K)$$

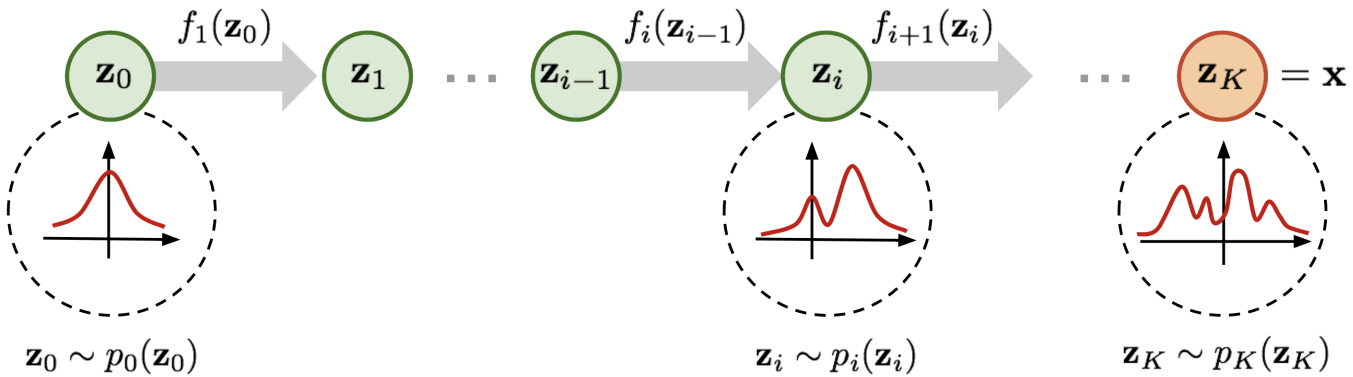
$$= p_{z_{K-2}}(z_{K-2}) \left| \det \left(\frac{\partial z_{K-2}}{\partial z_{K-1}} \right) \right| \left| \det \left(\frac{\partial z_{K-1}}{\partial z_K} \right) \right| \qquad \text{where } z_{K-2} = g_{K-1}(z_{K-1})$$

$$= \ldots$$

Composition of flows

Let $f_1, \ldots, f_K$ be invertible transformations with inverse $g_1, \ldots, g_K$.

Then the composition $f = f_K \circ \cdots \circ f_1$ is also invertible with inverse $g = g_1 \circ \cdots \circ g_K$.



Source: [Wen18]

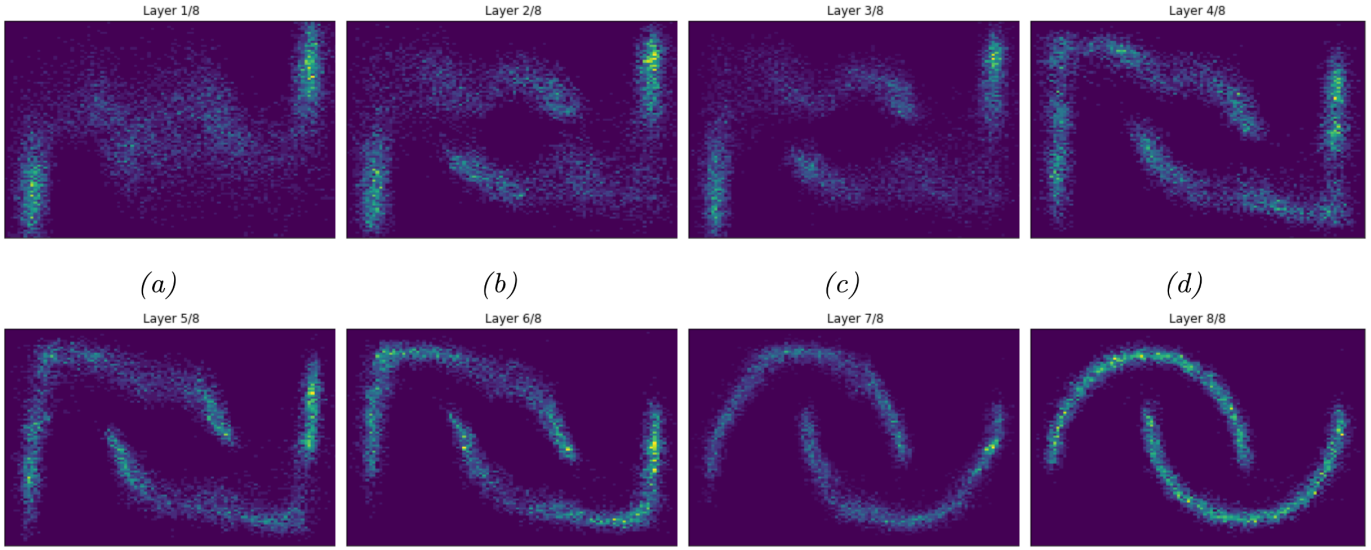
Let $\boldsymbol{z}_k = g_{k+1} \circ \cdots \circ g_K(\boldsymbol{z}_K)$, $\boldsymbol{x} = \boldsymbol{z}_K$ and $\boldsymbol{z} = \boldsymbol{z}_0$.

By induction, we have:

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(g(\boldsymbol{x})) \prod_{k=1}^{K} \left| \det \left(\frac{\partial \boldsymbol{z}_{k-1}}{\partial \boldsymbol{z}_k} \right) \right|$$

$$-\log p_{\boldsymbol{x}}(\boldsymbol{x}) = -\log p_{\boldsymbol{z}}(g(\boldsymbol{x})) - \sum_{k=1}^{K} \left| \det \left(\frac{\partial \boldsymbol{z}_{k-1}}{\partial \boldsymbol{z}_k} \right) \right|$$

Examples of samples



Samples per layer of a normalizing flow based on spline flows and coupling flows (two moons dataset). Source: [Mur23]

Samples of GLOW (CelebA-HQ dataset)

Source: [KD18]

Pros and cons

Pros	Cons
• Explicit likelihood $p(\boldsymbol{x})$	• Restrictions on the architecture due to the need of invertible functions
• Latent space available	
• Fast sampling	• More difficult to train

Summary

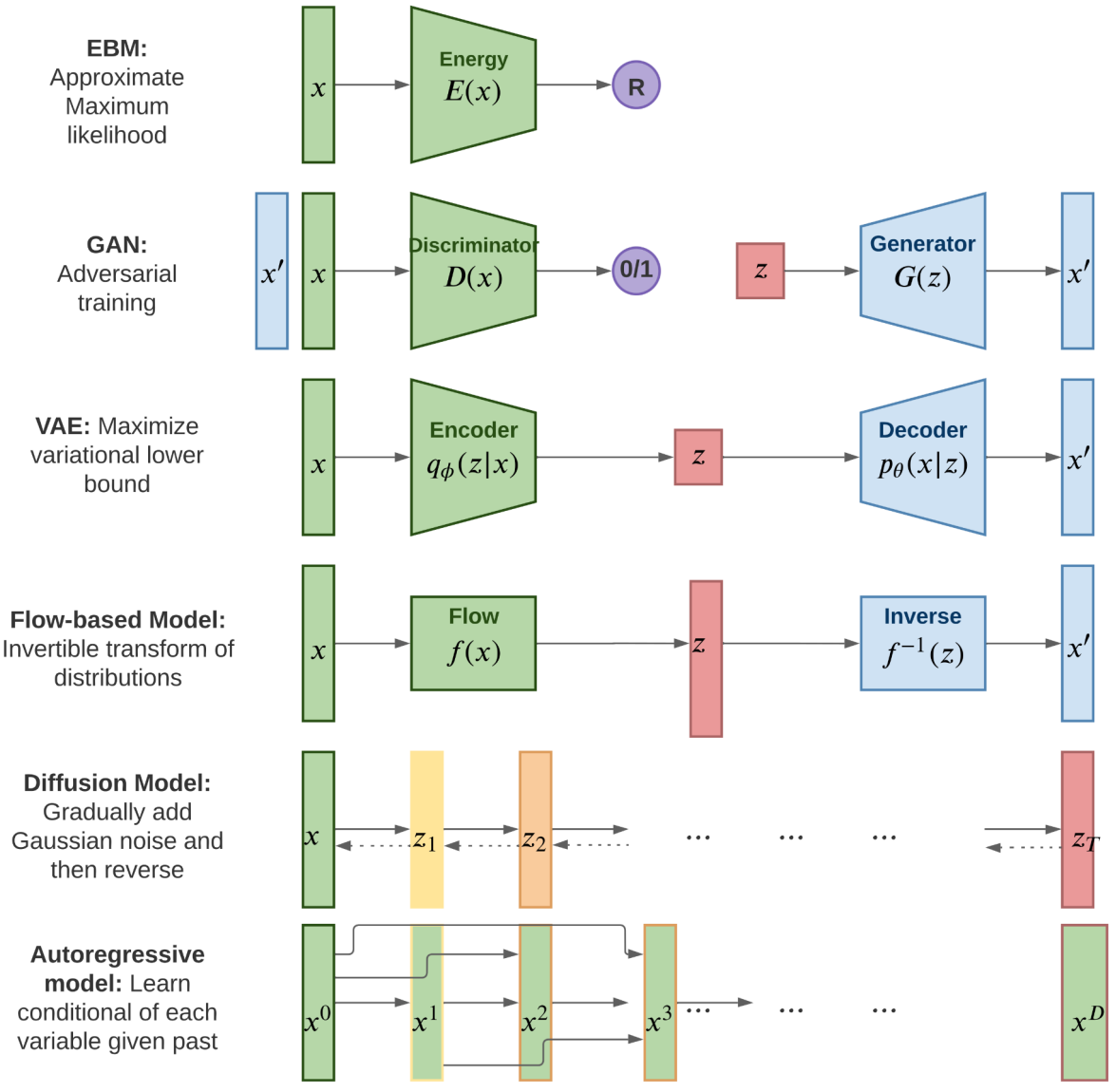
We have seen the fundamental ideas behind:

- Autoregressive models: $p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_d \mid x_1, \ldots, x_{d-1})$
- Normalizing flows: $p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(g(\boldsymbol{x})) \left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|$
- Variational autoencoders: $p(\boldsymbol{x}) = \int p(\boldsymbol{z}) p(\boldsymbol{x} \mid \boldsymbol{z}) d\boldsymbol{z}$

These methods can be combined, notably:

- **Normalizing flows** can be constructed from **autoregressive models** [PP+17].
- The **latent space** of a VAE can become more flexible when parameterized by a **normalizing flow** [RM15].

Overview of deep generative models



Source: [Mur23]

Bibliography i

[Ant] Jason Antic. *DeOldify: A Deep Learning based project for colorizing and restoring old images (and video!)* en.

[DBM+19] Durkan, Bekasov, Murray, et al. "Neural spline flows". In: *Advances in neural information processing systems* (2019). ISSN: 1049-5258.

[Du+22] Shian Du et al. "TO-FLOW: Efficient Continuous Normalizing Flows with Temporal Optimization adjoint with Moving Speed". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12570–12580.

Bibliography ii

[Ger+15] Mathieu Germain et al. "MADE: Masked Autoencoder for Distribution Estimation". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 881–889.

[KD18] Kingma and Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* (2018). ISSN: 1049-5258.

Bibliography iii

[Kin17] D P Kingma. "Variational inference & deep learning". PhD thesis. Univsersity of Amsterdam, 2017.

[Liu+18] Guilin Liu et al. "Image Inpainting for Irregular Holes Using Partial Convolutions". In: (20 4 2018). arXiv: 1804.07723 [cs.CV].

[Mur23] Kevin P Murphy. *Probabilistic machine learning: Advanced topics*. MIT Press, 2023.

[OKK16] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel Recurrent Neural Networks". In: (25 1 2016). arXiv: 1601.06759 [cs.CV].

Bibliography iv

[Oor+16] Aaron van den Oord et al. “Conditional Image Generation with PixelCNN Decoders”. In: (16 6 2016). arXiv: 1606.05328 [cs.CV].

[PP+17] Papamakarios, Pavlakou, et al. “Masked autoregressive flow for density estimation”. In: *Advances in neural information processing systems* (2017). ISSN: 1049-5258.

Bibliography v

[RM15] Danilo Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1530–1538.

[Wen18] Lilian Weng. *Flow-based Deep Generative Models*. `https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html`. Accessed: 2021-12-10. 13 10 2018.