Diffusion Models

📅Date: 12-30-2022
🕐Time: 20:39

Bipin Koirala

Table of Contents

Preliminary 🔍

Discriminant Models Vs Generative Models

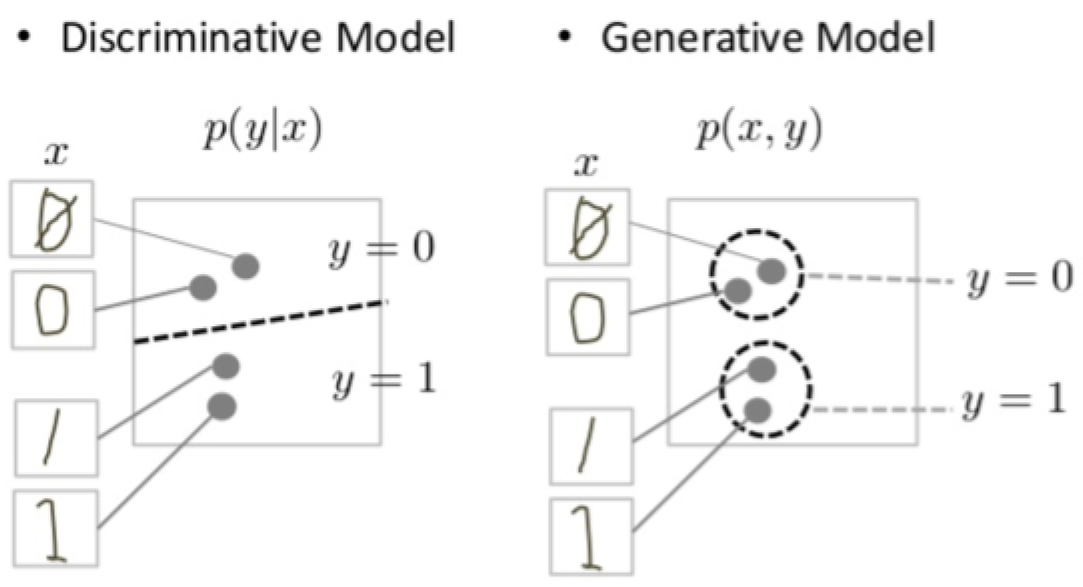
Most of the time we are concerned with predicting a label given an instance of a dataset. Statistical models that operate on this notion are #discriminant models. Unlike these models there exists a different paradigm where we want to learn the joint probability distribution between the data and its label (holds true even if the data has no label). These models are called #generative models and can generate new data instances.

For a set of data instances X and set of labels Y, we have the following:

$$\text{Discriminant Model} : \mathbb{P}(label|data) = \mathbb{P}(Y|X)$$

$$\text{Generative Model} : \mathbb{P}(data, label) = \mathbb{P}(X, Y) \;\; \text{or} \;\; \mathbb{P}(X) \;\; \text{if no labels}$$

For example, a discriminant model could tell a picture of a bird from a horse but generative model could generate a new pictures of animals that look like real animals.
For data $x \in \mathcal{D}$, discriminant model aims to draw a boundary in $\mathcal{D}$ whereas generative model aims to model how a data is placed throughout the space $\mathcal{D}$.



Some types of generative models are:

- Gaussian Mixture Model (GMM)
- Bayesian Network (Naive Bayes, Auto-regressive models)
- Boltzmann Machine
- Generative Adversarial Network (GAN)
- Variational Auto-encoder (VAE)
- Diffusion Models

- Energy Based Models (EBM) etc.

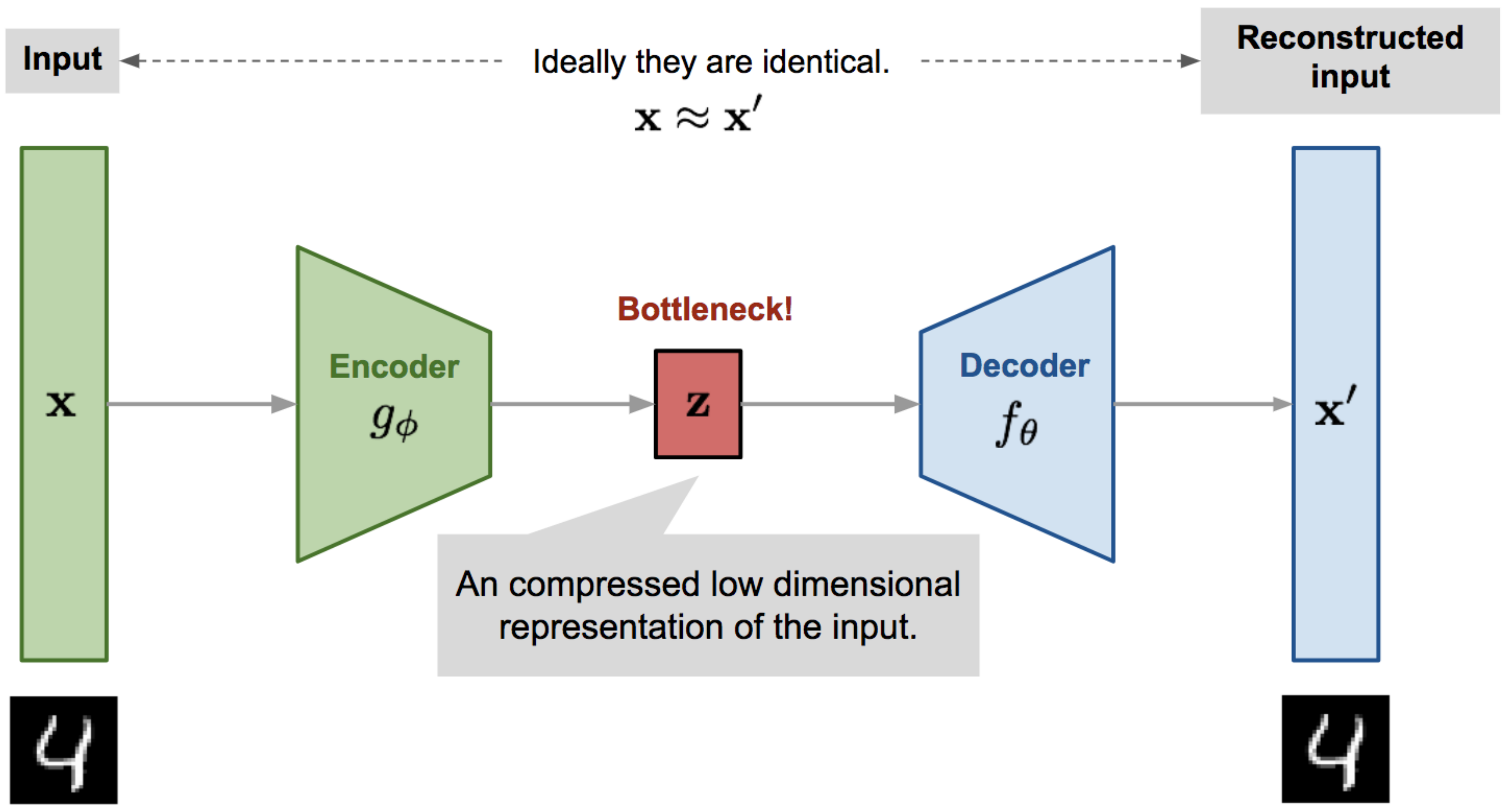
Below are summary of few Generative-Models[1].

GANs

These are primarily used to replicate real-world contents such as images, languages, and musics. Two agents, #generator and #discriminator play a min-max game to attain equilibrium. It is difficult to train GAN because of training instability and failure to converge. #Wasserstein GAN (WGAN) provides improved results over traditional GAN.

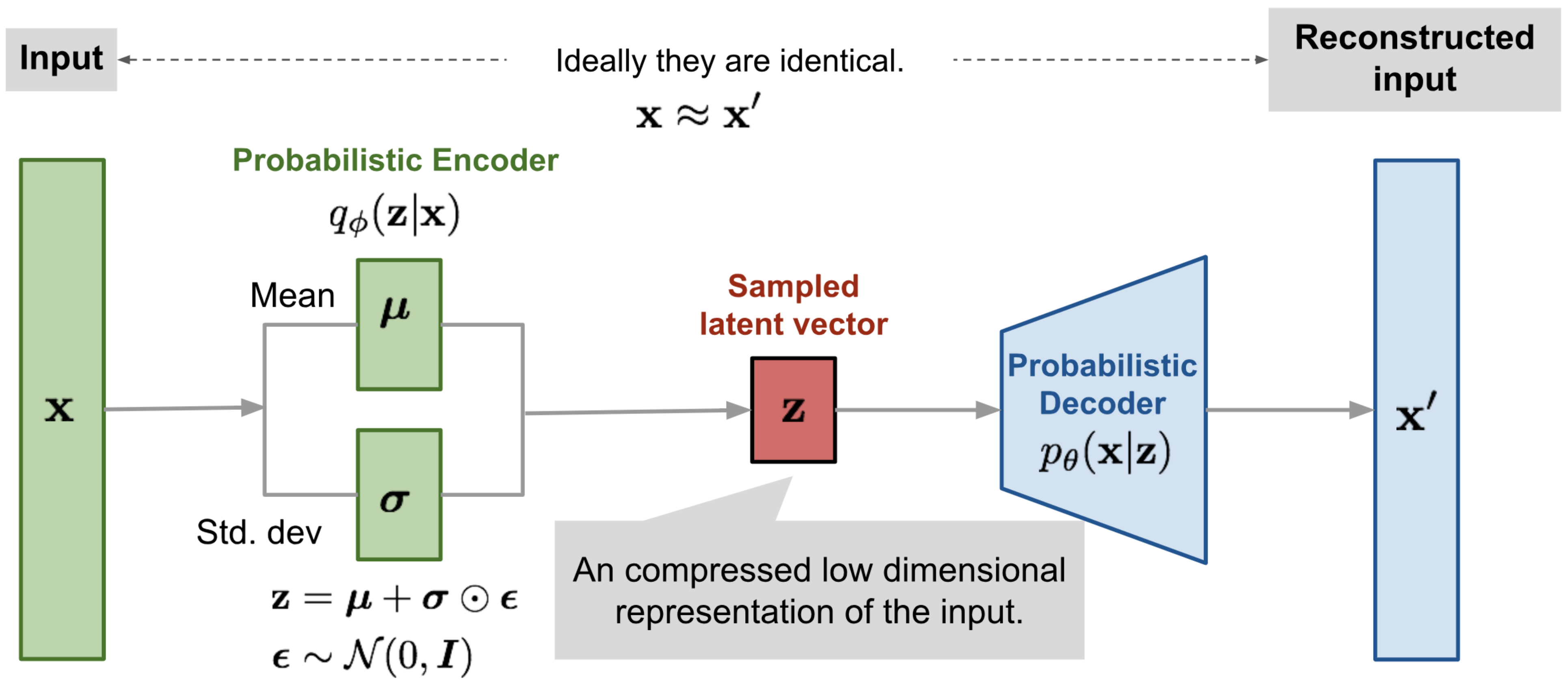
VAE

#Auto-encoder is a neural network which attempts to reconstruct a given data via compressing the input in the process so as to discover a latent/sparse representation. This latent representation of data can later be used in various downstream tasks.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

The latent space in auto-encoders are primarily discrete and does not allow for an easy interpolation. The generative part or the decoder of the auto-encoder works by randomly sampling points from the latent space and it can be challenging if the latent space is itself discontinuous or has gaps.

#Variational-Auto-Encoder (VAE) solves this issue because its latent space is continuous in nature which makes VAE powerful in generating new data instances. Instead of generating a latent vector $z \in \mathbb{R}^N$, VAE generates two vectors i.e. mean (μ) vector and standard deviation (σ) vector followed by decoder sampling from this distribution.


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

Some Remarks:

> $p(z)$: Prior
> $p(x|z)$: Likelihood (Generation)
> $p(x, z) = p(x|z)p(z)$: Joint Distribution

$p(x)$: Marginal
$p(z|x)$: Posterior (Inference)

For Generation: sample $z \sim p(z)$ then sample $x \sim p_\theta(x|z)$
For Inference: sample $x \sim p(x)$ then sample $z \sim q_\phi(z|x)$ which acts as a proxy for $p_\theta(z|x)$

Ideally we would like to get;

$$p_\theta(x) = \int p_\theta(x|z)\, p_\theta(z)\, dz$$

However the integral is intractable because it needs to be evaluate over all possible values of $z \sim p_\theta(z)$. These kinds of latent variable models are often trained with Maximum Likelihood Estimation (M.L.E)

$$\theta_{MLE} = \arg\max_\theta \sum_{i=1}^{N} log\, p_\theta(x_i)$$

Since, $p_\theta(x)$ does not have a closed form solution we find its lower bound called the #Variational-Lower-Bound (V.L.E) or #Evidence-Lower-Bound (E.L.B.O)

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

Since $D_{KL}\Big(q_\phi(z|x)\ \ ||\ \ p_\theta(z|x)\Big) \geq 0$;

$$\log p_\theta(x) \geq L_{\theta,\phi}(x)$$

Therefore, we optimize (maximize) over this #Variational-Lower-Bound.

Optimizing Variational Lower Bound

One possibility is to sample x_i and get the best $q_\phi(z|x_i)$ using multiple gradient steps in ϕ. Then gradient ascend in θ. However, this is an expensive inference process because we would need to learn individual distribution for each x_i.

Instead we #Amortize the inference costs by learning an inference neural network that presumes $q_\phi(z|x) \sim \mathcal{N}(z\,;\ \mu(x), I\sigma(x))$.

👉 Stochastic Gradient Optimization of #Variational-Lower-Bound
The want to solve the following optimization problem

$$\max_{\theta,\,\phi} \sum_{x_i\,\in\,\mathcal{D}} L_{\theta,\phi}(x_i)$$

Computing $\nabla_{\theta,\phi}\ L_{\theta,\phi}(x_i)$ is intractable as we would need to sample across entire $q_\phi(z|x)$ and take its gradient. However, there are unbiased estimators for this.

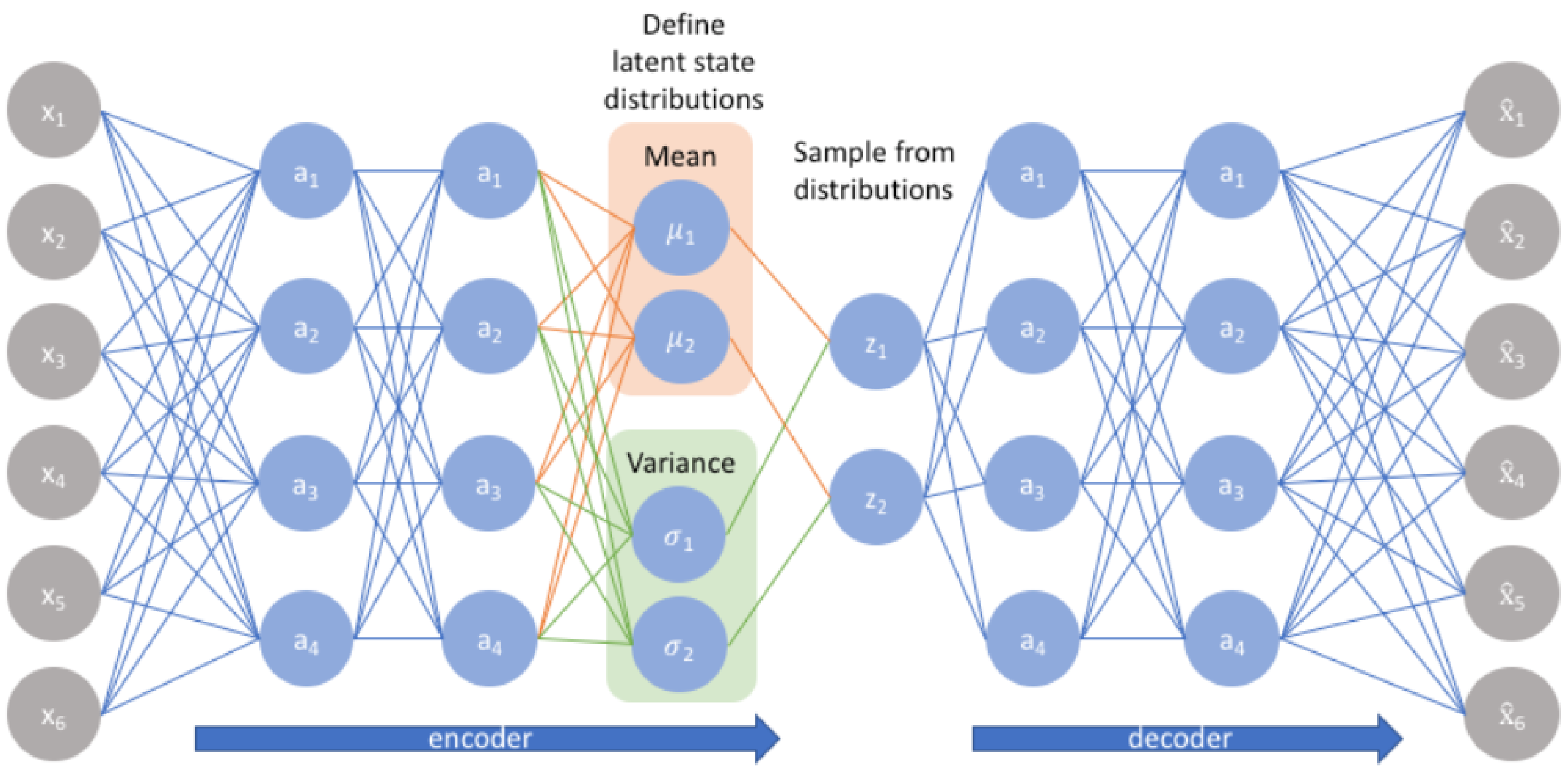
Recall, $q_\phi(z|x) \sim \mathcal{N}(z\,;\ \mu(x), I\sigma(x)) = \mu(x) + \sigma(x)\epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$

Now,

$$\begin{aligned} L_{\theta,\phi}(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}[log\, p_\theta(x,z) - log\, q_\phi(z|x)] \\ \hat{L}_{\theta,\phi}(x) &= \mathbb{E}_{\epsilon\sim p(\epsilon)}[log\, p_\theta(x,z) - log\, q_\phi(z|x)] \qquad \text{(unbiased estimator)} \end{aligned}$$

🔥Note: $\mathbb{E}_{\epsilon\sim p(\epsilon)}\hat{L}_{\theta,\phi}(x) = L_{\theta,\phi}(x)$ so now need to compute $\nabla_\phi \hat{L}_{\theta,\phi}(x)$

Neural Network architecture of VAE showing how decoder samples from latent vectors.



Flow-Models

At its core, #Flow-Models make use of **Normalizing Flow (NF)**, a technique used to build a complex probability distributions by transforming simple distributions.
Let, $z \sim \mathbb{P}_\theta(z)$ and $z \in Z$ be a probability distribution, generally taken something simple like $\mathcal{N}(z; \mu, \sigma)$. The key idea here is to transform this simple distribution to a complex distribution $x = f(z)$, where f is a bijective map. We formulate f as a composition of sequence of invertible transformations.

$$x = f_K \circ f_{K-1} \circ \ \dots \ f_2 \circ f_1(z)$$

Now,

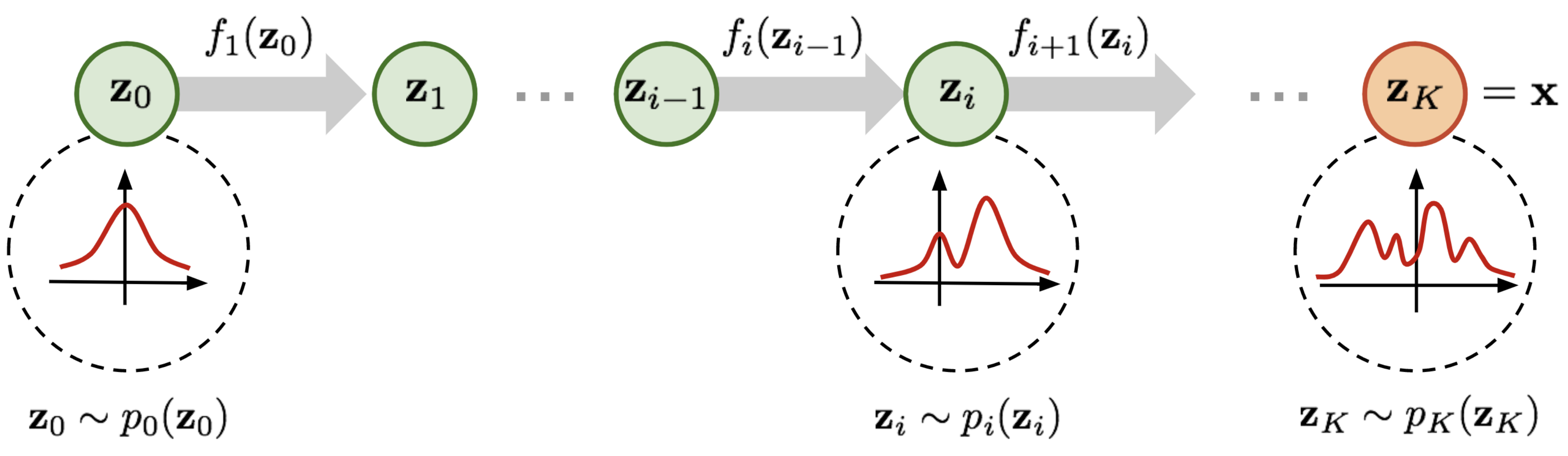
$$\int p_\theta(x)dx = \int p_\theta(z)dz = 1$$

$$p_\theta(x)dx = p_\theta(f^{-1}(x))dz$$

$$p_\theta(x) = p_\theta(f^{-1}(x))\Big|\frac{dz}{dx}\Big| = p_\theta(f^{-1}(x))\Big|\frac{df^{-1}}{dx}\Big|$$

Multivariable formulation of the above expression gives us;

$$p_\theta(\mathbf{x}) = p_\theta(f^{-1}(\mathbf{x}))\Big|det\Big(\frac{df^{-1}}{d\mathbf{x}}\Big)\Big|$$


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

Diffusion Models

Non-equilibrium thermodynamics deals with the study of time-dependent thermodynamic systems, irreversible transformations and open systems. #Diffusion-Models are heavily inspired by non-equilibrium thermodynamics. They define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. Unlike VAE or Flow-Models, diffusion models are learned with a fixed procedure and the latent space has high dimensionality (same as the original data)[2].

> Much of the notes in the followings sections on the #Diffusion-Model are based on *Deep Unsupervised Learning using Non-equilibrium Thermodynamics*.

#Diffusion-Model allows us to rapidly learn, sample from, and evaluate probabilities in deep generative models with thousands of layers/time-steps, as well as to compute conditional and posterior probabilities under the learned model. Diffusion process exists for any smooth target distribution, this method can capture data distributions of arbitrary form.

Algorithm

The goal is to define a forward (or inference) diffusion process which converts any complex data distribution into a simple, tractable, distribution. Then learn a finite-time reversal of this diffusion process which defines the generative model distribution.

Forward Trajectory

👉 Initial data distribution: $q(x^{(0)})$

👉 This is gradually converted to a well-behaved (analytically tractable) distribution $\pi(y)$ by repeated application of a Markov diffusion kernel $T_\pi(y \mid y^{'}; \beta)$ where β is diffusion rate

$$\pi(y) = \int T_\pi(y \mid y^{'}; \beta) \;\; \pi(y^{'}) \;\; dy^{'} \tag{1}$$
$$q\left(x^t | x^{(t-1)}\right) = T_\pi\left(x^{(t)} \mid x^{(t-1)}; \beta_t\right) \tag{2}$$

Here; forward diffusion kernel is $T_\pi\left(x^{(t)} \mid x^{(t-1)}; \beta_t\right) = \mathcal{N}\left(x^{(t)}; x^{(t-1)}\sqrt{1-\beta_t},\; I\beta_t\right)$ i.e. Gaussian but can be other distribution as well for example a Binomial distribution.

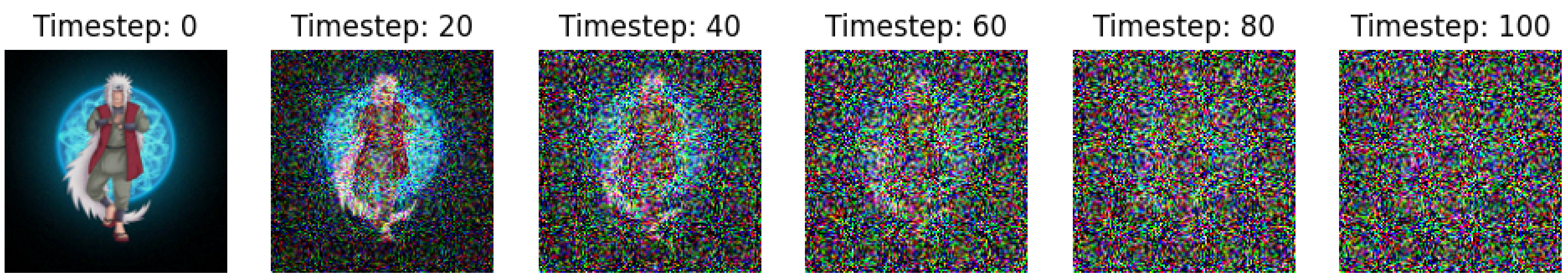
The forward trajectory, corresponding to starting at the initial data distribution and performing T steps of diffusion is given by,

$$q\left(x^{(0....T)}\right) = q(x^{(0)}) \prod_{t=1}^{T} q\left(x^{(t)} \mid x^{(t-1)}\right) \tag{3}$$

The above process allows for the sampling of x_t at any arbitrary time step t in a closed form using reparameterization trick.

$$x^{(t)} = \sqrt{1-\beta_t} \;\; x^{(t-1)} \; + \; \sqrt{\beta_t}\,\mathcal{N}(0, I)$$

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

img = Image.open('Jiraya.jpg')
img = img.resize(size = (128,128))
current_img = np.asarray(img)/255.0

def forward_diffusion(previous_img, beta, t):
        beta_t = beta[t]
        mean = previous_img * np.sqrt(1.0 - beta_t)
        sigma = np.sqrt(beta_t) # variance = beta
        # Generate sample from N(0,1) of prev img size and scale to new distribution.
        xt = mean + sigma * np.random.randn(*previous_img.shape)
        return xt

time_steps = 100
beta_start = 0.0001
beta_end = 0.05
beta = np.linspace(beta_start, beta_end, time_steps)

samples = []

for i in range(time_steps):
        current_img = forward_diffusion(previous_img = current_img, beta = beta, t = i)

        if i%20 == 0 or i == time_steps - 1:
                # convert to integer for display
                sample = (current_img.clip(0,1)*255.0).astype(np.uint8)
                samples.append(sample)


plt.figure(figsize = (12,5))
for i in range(len(samples)):
        plt.subplot(1, len(samples), i+1)
        plt.imshow(samples[i])
        plt.title(f'Timestep: {i*20}')
        plt.axis('off')

plt.show()
```



Reverse Trajectory

Generative distribution is trained to describe the above (forward) trajectory, but in the reverse direction,

$$p\left(x^{(T)}\right) = \pi\left(x^{(T)}\right) \tag{4}$$

$$p\left(x^{(0 \ldots T)}\right) = p\left(x^{(T)}\right) \prod_{t=1}^{T} p\left(x^{(t-1)} \mid x^{(t)}\right) \tag{5}$$

If we take β_t to be small enough then, $q\left(x^{(t-1)}|x^{(t)}\right)$ will also be a Gaussian distribution. The longer the trajectory the smaller the diffusion rate β can be made. During learning only the mean and the covariance for a Gaussian diffusion kernel need to be estimated. The functions defining the mean and the covariance of the reverse process are:

$$f_\mu(x^{(t)}, t) \quad \& \quad f_\Sigma(x^{(t)}, t)$$

Practically speaking, we don't know $q\left(x^{t-1}|x^{(t)}\right)$ as it is intractable since the statistical estimate requires computations involving the data distribution. Therefore, we approximate $q\left(x^{t-1}|x^{(t)}\right)$ with a parameterized model p_θ (e.g. a neural network). With the parameterized model, we have

$$p_\theta\left(x^{(0 \ldots T)}\right) = p_\theta\left(x^{(T)}\right) \prod_{t=1}^{T} p_\theta\left(x^{(t-1)} \mid x^{(t)}\right)$$

i.e. starting with the pure Gaussian noise, the model learns the joint distribution $p_\theta(x^{(0 \ldots T)})$. Conditioning the model on time-step t, it will learn to predict the Gaussian parameters, $f_\mu(x^{(t)}, t)$ and $f_\Sigma(x^{(t)}, t)$, for each time-step.

Here;

$$p_\theta(x^{(t-1)}|x^{(t)}) = \mathcal{N}\left(x^{(t-1)}; \;\; \mu_\theta(x^{(t)}, t), \;\; \Sigma_\theta(x^{(t)}, t)\right)$$

Training

The probability the generative model assigns to the data is

$$p(x^{(0)}) = \int p(x^{(0 \ldots T)}) \, dx^{(1 \ldots T)} \tag{6}$$

This tells us that if we were to calculate $p(x^{(0)})$ we need to marginalize over all the possible trajectories to arrive at the initial distribution starting from the noise sample...which is intractable in practice. However, we can maximize a lower bound.

In #Diffusion-Model , the forward process is fixed and only reverse process needs to be trained i.e. only a single network is trained unlike #Variational-Auto-Encoder .

#Diffusion-Models are trained by finding the reverse Markov transitions that maximize the likelihood of the training data. Similar to #Variational-Auto-Encoder training is based on minimizing the #Variational-Lower-Bound . Therefore, we optimize the negative log-likelihood.

$$\begin{aligned}
-\log p_\theta(x_0) &\leq -\log p_\theta(x_0) + D_{KL}\Big(\; q(x_{1:T}|x_0) \;\; || \;\; p_\theta(x_{1:T}|x_0) \; \Big) \\
&= -\log p_\theta(x_0) + \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)} \left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)} \right] \\
&= -\log p_\theta(x_0) + \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)} \left[\log \frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_{1:T}) \, p_\theta(x_0|x_{1:T})}{p_\theta(x_0)}} \right] \\
&= \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)} \left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:1:T})} \right]
\end{aligned}$$

Here, #Variational-Lower-Bound is $\mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)}$ and

$$-\log p_\theta(x_0) \leq \mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)} \left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:1:T})} \right] \tag{7}$$

Now,

$$
\begin{aligned}
\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} &= \log \frac{\prod_{t=1}^{T} q\,(x_t|x_{t-1})}{p(x_T)\prod_{t=1}^{T}\ p_\theta(x_{t-1}|x_t)} \\
&= -\log\,(p(x_T)) + \log \frac{\prod_{t=1}^{T} q\,(x_t|x_{t-1})}{\prod_{t=1}^{T}\ p_\theta(x_{t-1}|x_t)} \\
&= -\log\,(p(x_T)) + \sum_{t=1}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} \\
&= -\log\,(p(x_T)) + \sum_{t=2}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} \\
&= -\log\,(p(x_T)) + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)\;q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)\;q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} \\
&\because \text{Baye's Rule and conditoning on } x_0 \text{ to avoid high variance} \\
&= -\log\,(p(x_T)) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} \\
&= -\log\,(p(x_T)) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_T|x_0)}{q(x_1|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} \\
&= \log \frac{q(x_T|x_0)}{p(x_T)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_0|x_1) \\
&\because \text{ Firt terms has no learnable parameter --- drop it}
\end{aligned}
$$

From (7);

$$
\begin{aligned}
-\log p_\theta(x_0) &\le \mathbb{E}_{x_{1:T}\sim q(x_{1:T}|x_0)}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:1:T})}\right] \\
&= \mathbb{E}_{x_{1:T}\sim q(x_{1:T}|x_0)}\left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_0|x_1)\right] \\
&= \mathbb{E}_{x_{1:T}\sim q(x_{1:T}|x_0)}\left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t,x_0)}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_0|x_1)\right] \\
&= \sum_{t=2}^{T} D_{KL}\Big(q(x_{t-1}|x_t,x_0)\;||\;p_\theta(x_{t-1}|x_t)\Big) - \log p_\theta(x_0|x_1)
\end{aligned}
$$

We can further simplify the first term above because;

$$
p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\mu_\theta(x_t,t),\beta_t I) \quad \& \quad q(x_{t-1}|x_t,x_0) = \mathcal{N}(x_{t-1};\ \tilde{\mu}_t(x_t,x_0),\tilde{\beta}_t I)
$$

And,

$$
\begin{aligned}
\tilde{\mu}_t(x_t,x_0) &:= \frac{\sqrt{\overline{\alpha}_{t-1}}\,\beta_t}{1-\overline{\alpha}_t}\,x_0 + \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}\,x_t \\
\tilde{\beta}_t &:= \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t
\end{aligned}
$$

where $\alpha_t = 1-\beta_t$ and $\overline{\alpha}_t = \prod_{t=s}^{t}\alpha_s$ and further simplification as shown in *Denoising Diffusion Probabilistic Models [2020]* yields the loss function

$$
L_t \sim ||\epsilon - \epsilon_\theta(x_t,t)||_2^2
$$

where, $\epsilon := \mathcal{N}(0,I)$ and $(x_t,t) = \left(\sqrt{\overline{\alpha}_t}\,x_0 + \sqrt{1-\overline{\alpha}_t}\,\epsilon,\ t\right)$

Notes 📝

Sources 📚

[1]. Generative Models
[2]. Diffusion Models