





Diffusion Models

 Date: 12-30-2022

 Time: 20:39

Bipin Koirala

Table of Contents

1. [Preliminary](#) 
 1. [Discriminant Models Vs Generative Models](#)
 1. [GANs](#)
 2. [VAE](#)
 3. [Flow-Models](#)
 2. [Diffusion Models](#)
 1. [Algorithm](#)
 1. [Forward Trajectory](#)
 3. [Notes](#) 
 4. [Sources](#) 
-

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:

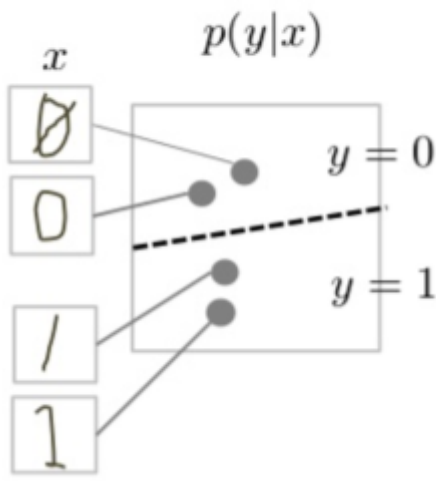
$$\text{Discriminant Model : } \mathbb{P}(\text{label}|\text{data}) = \mathbb{P}(Y|X)$$

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

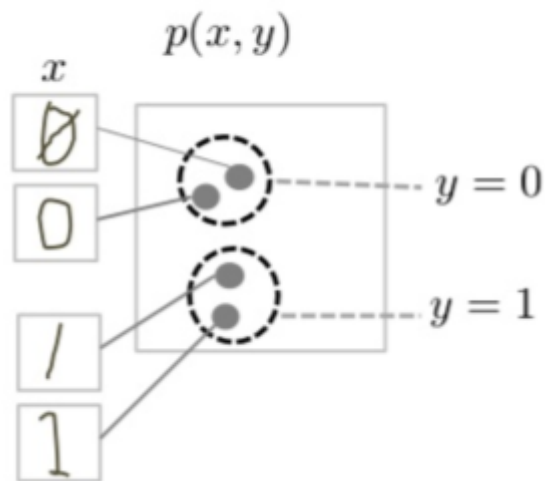
For example, a discriminant model could tell a picture of a bird from a horse but generative model could generate a new pictures of animals that look like real animals.

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} .

- Discriminative Model



- Generative Model



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.

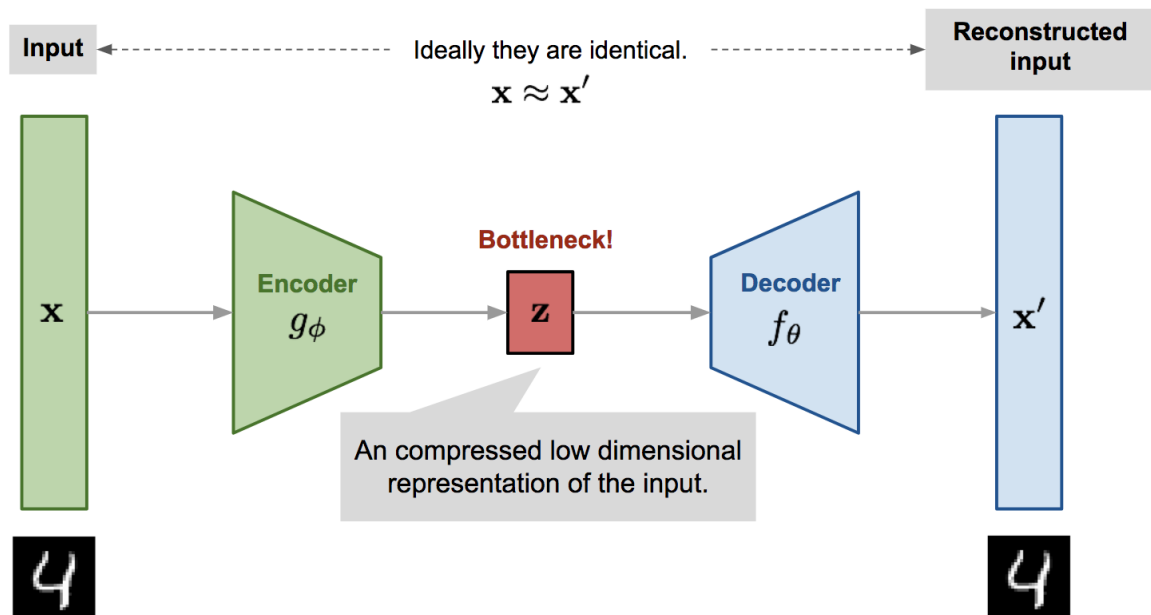
Below are summary of few Generative-Models[1].

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.

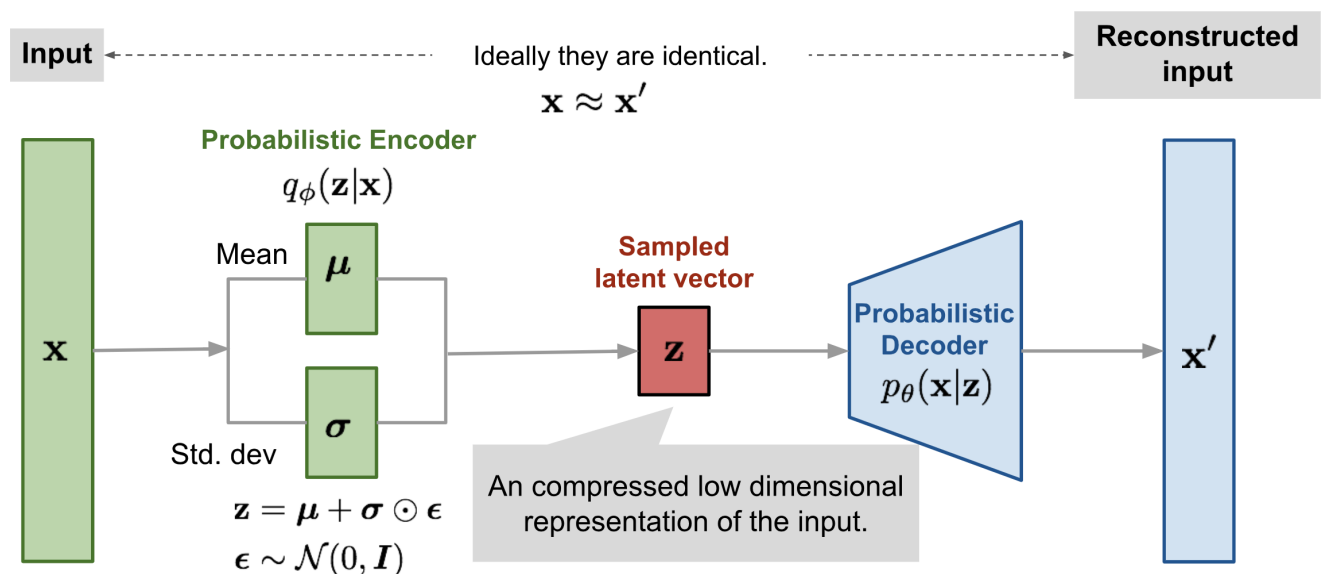
VAE

`#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.

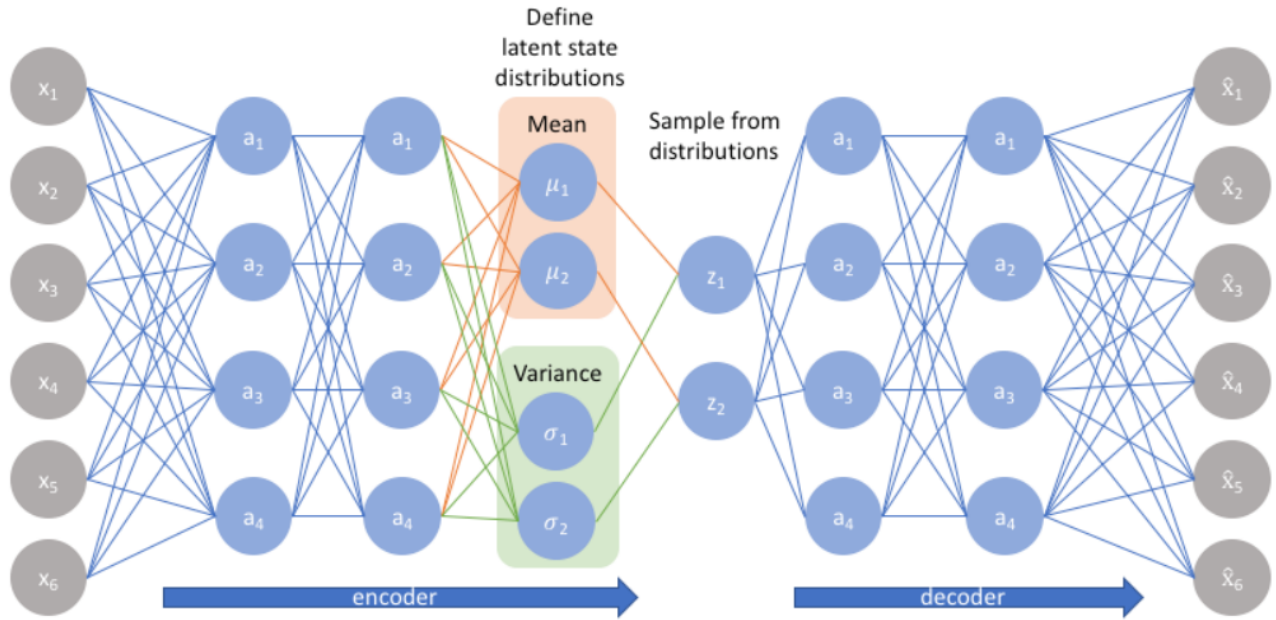


The latent space in auto-encoders are primarily discrete and does not allow for an easy interpolation. The generative part or the decoder of the auto-encoder works by randomly sampling points from the latent space and it can be challenging if the latent space is itself discontinuous or has gaps.

#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.



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.

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

Now,

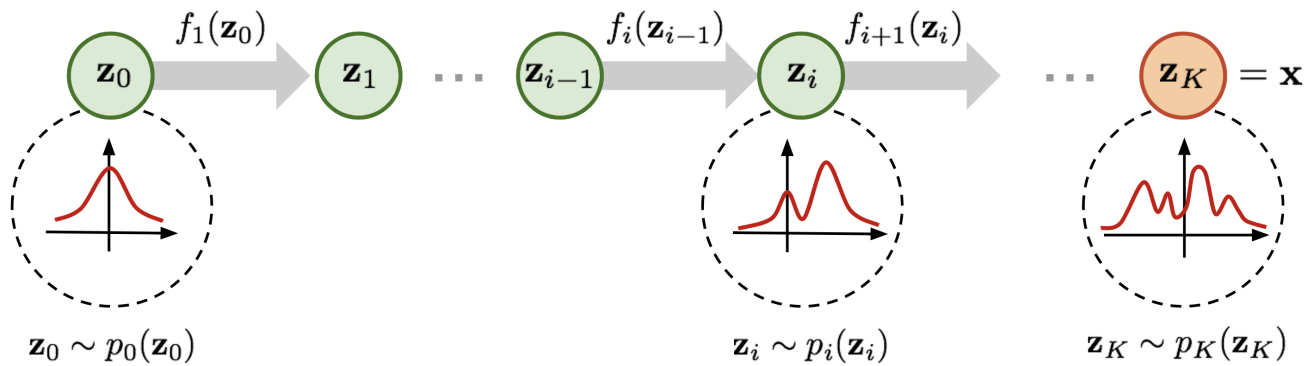
$$\int p_\theta(x) dx = \int p_\theta(z) dz = 1$$

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

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

Multivariable formulation of the above expression gives us;

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



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 $T_\pi(y | y'; \beta)$ where β is diffusion rate

$$\pi(y) = \int T_{\pi}(y | y'; \beta) \pi(y') dy' \quad (1)$$

$$q(x^t | x^{(t-1)}) = T_{\pi}(x^{(t)} | x^{(t-1)}; \beta_t) \quad (2)$$

Here; forward diffusion kernel is $T_{\pi}(x^{(t)} | x^{(t-1)}; \beta_t) = \mathcal{N}(x^{(t)}; x^{(t-1)}\sqrt{1-\beta_t}, I\beta_t)$ 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,

$$q(x^{(0...T)}) = q(x^{(0)}) \prod_{t=1}^T q(x^{(t)} | x^{(t-1)}) \quad (3)$$

```
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()
```

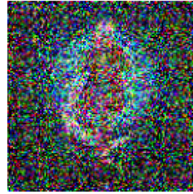
Timestep: 0



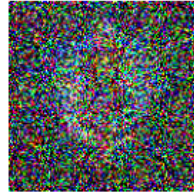
Timestep: 20



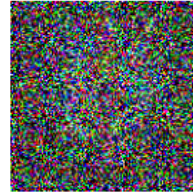
Timestep: 40



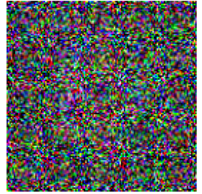
Timestep: 60



Timestep: 80



Timestep: 100



Notes

Sources

- [1]. [Generative Models](#)
- [2]. [Diffusion Models](#)