Generative Models

COMS 4995 - Neural Network and Deep Learning

Overview

There seems to be 3 types of generative models prevalent nowadays :

- GANs
- Reversible Architecture
- VAE & Diffusion models

But what is the idea that is **common** in all of them?

There is some z vector that is drawn from some know distribution like normal distribution $z \sim \mathcal{N}(0,1)$ and that is used to map the data distribution represented as p_x

This mapping $p_z \to p_x$ is done by some function f_θ that is usually a neural network parameterized by θ.

So you end up with $f_\theta(z) = x'$ that forms a distribution $p_{x'}$

What is the loss here ? You want the distribution $p_{x'}$ to be as close to the true distribution of the data p_x. So we measure the loss as the discrepancy or how much does $p_{x'}$ differs from p_x

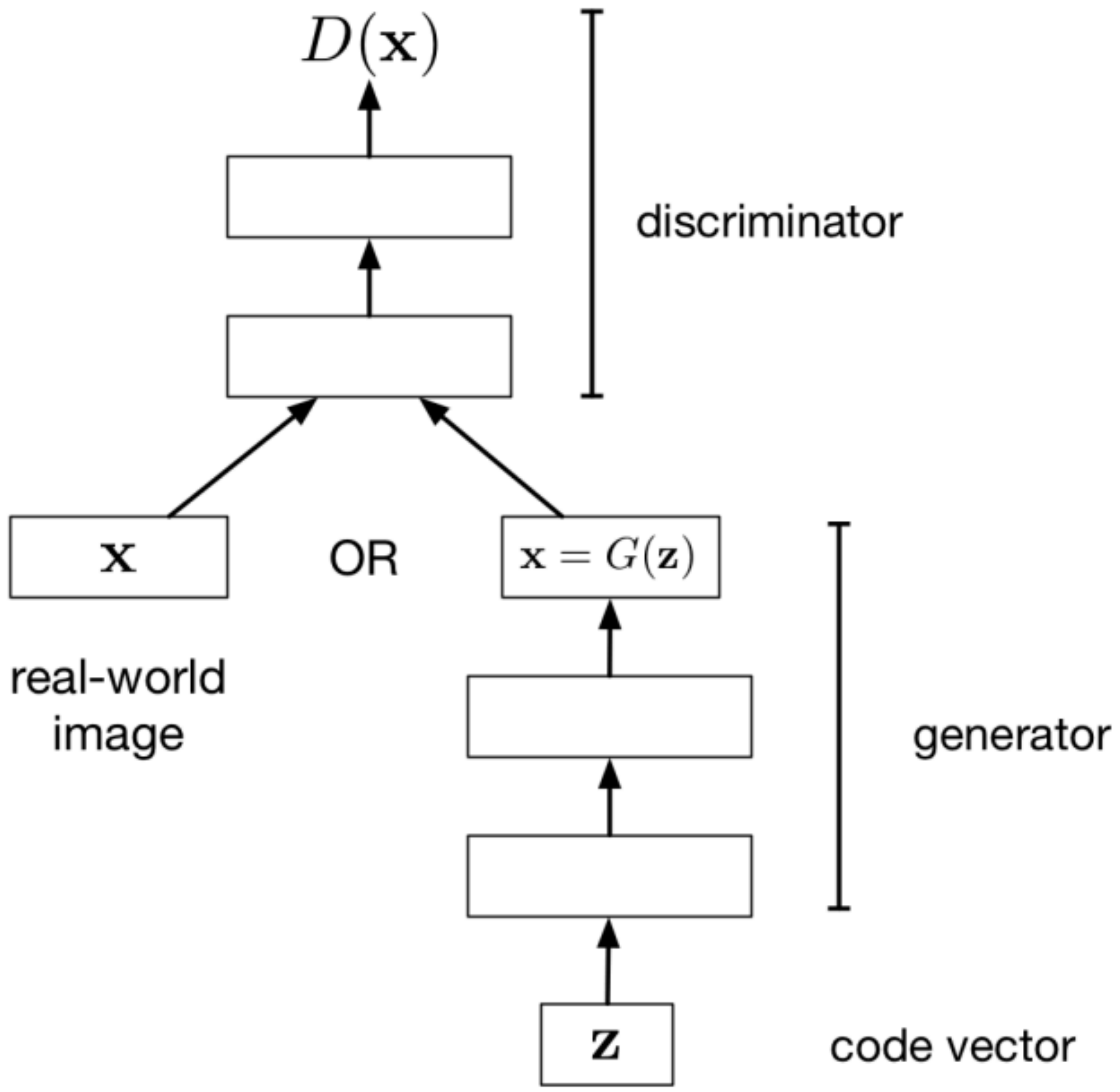
GANs

There is something known as Generator, $G(z)$ that takes the input $z \sim \mathcal{N}(0,1)$ and outputs a vector, x' that is supposed to mimic the actual distribution.

How do we measure if this value x' is close to a sample drawn from the actual distribution $x \sim p_x$

Therefore we have a discriminator, D(x, x') that takes on the actual sample, x and the 'fake' sample x' and outputs a score to each of them telling which one seems to be from the actual distribution.

There various loss that has been defined for estimating D(x,x') :



cross_entropy_loss

$$\mathcal{J}_D = \mathbf{E}_{x \sim p_x}[-log(D(x))] + \mathbf{E}_z[-log(1 - D(G(z)))]$$

$$\mathcal{J}_G = -\mathcal{J}_D$$

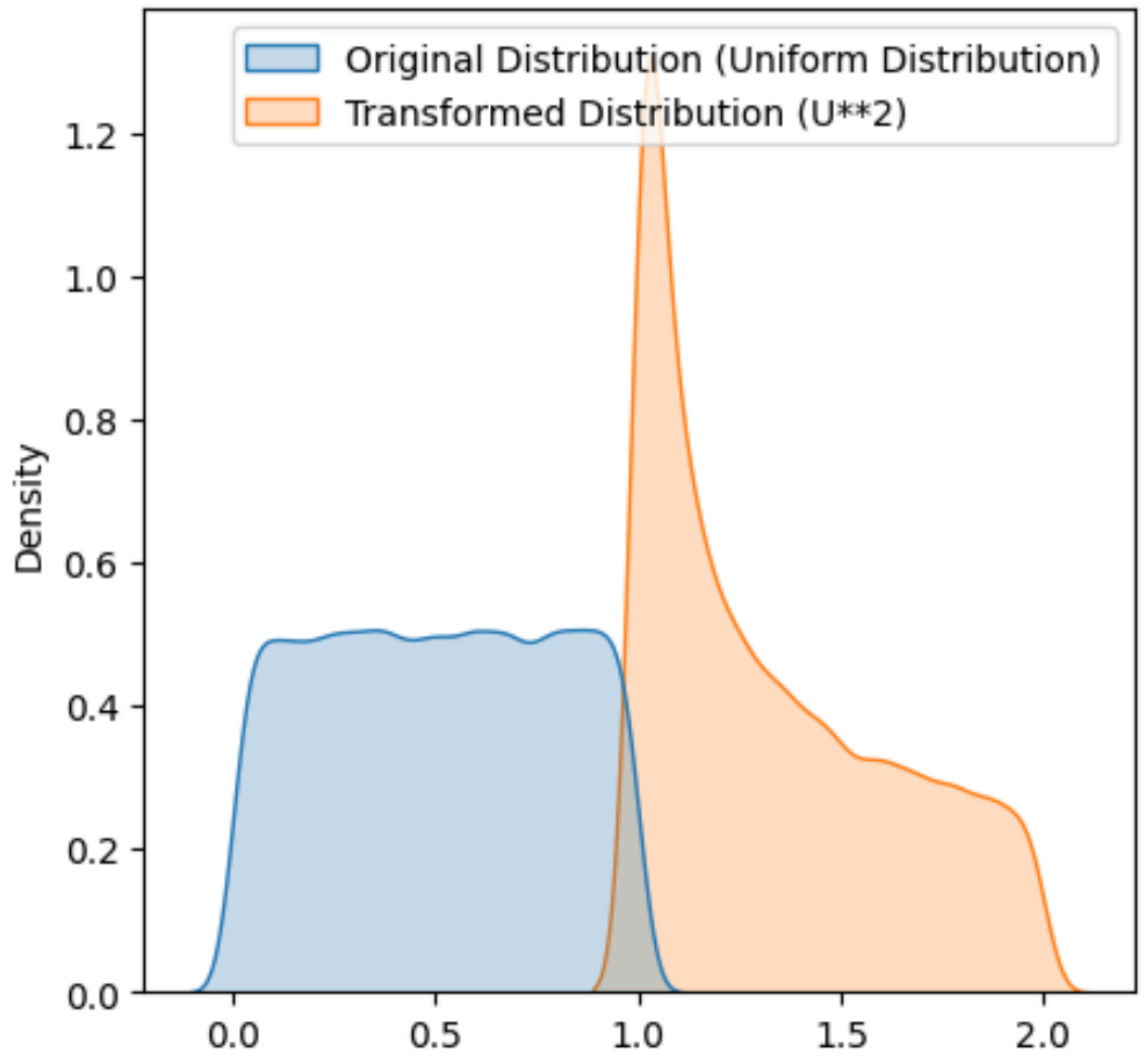
And then we use this to formulate the minimax problem

$$\max_G \min_D \mathcal{J}_D$$

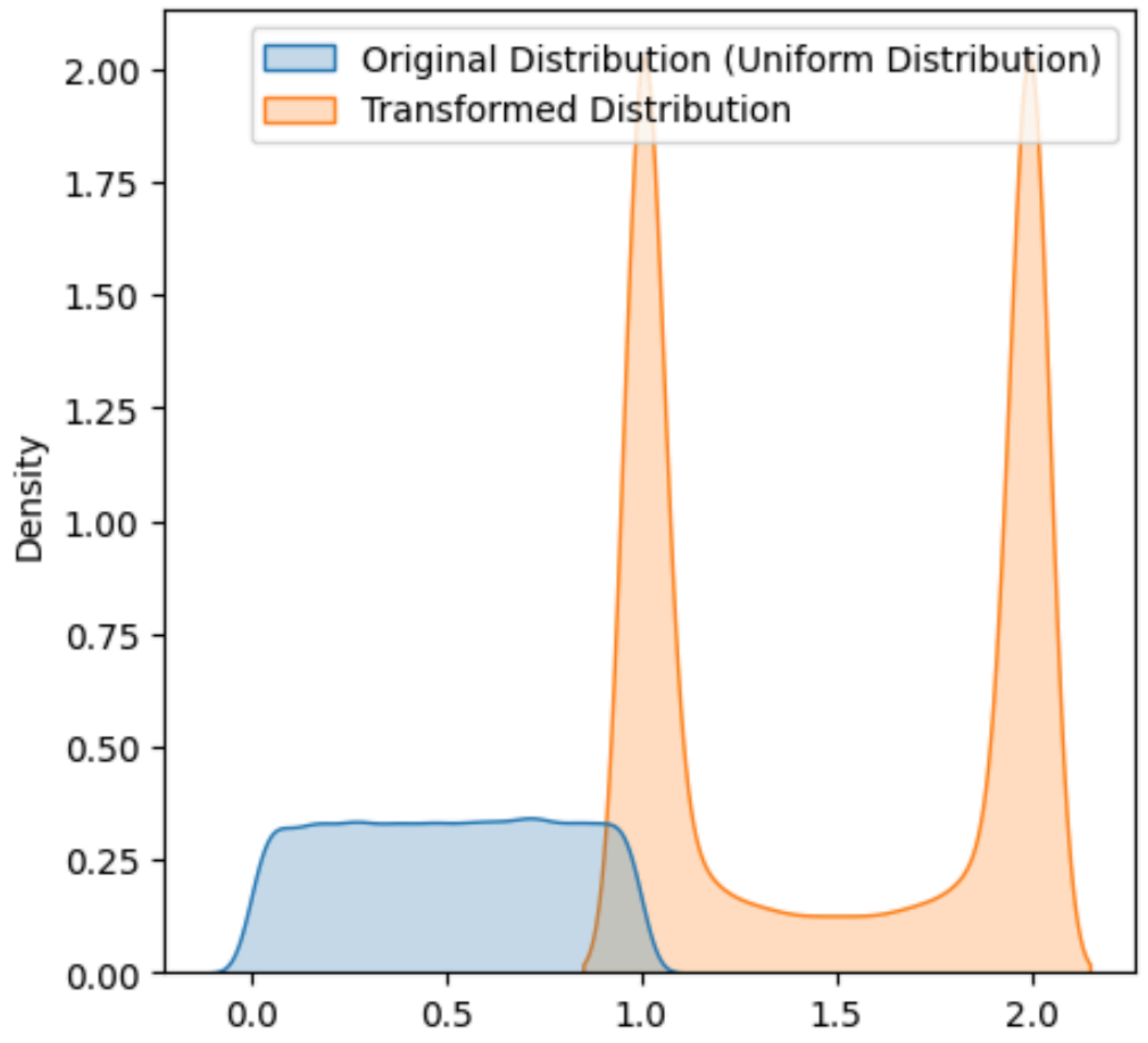
Some Examples and Codes

This things really work?

Transformed Distribution : p_x



Single Mode



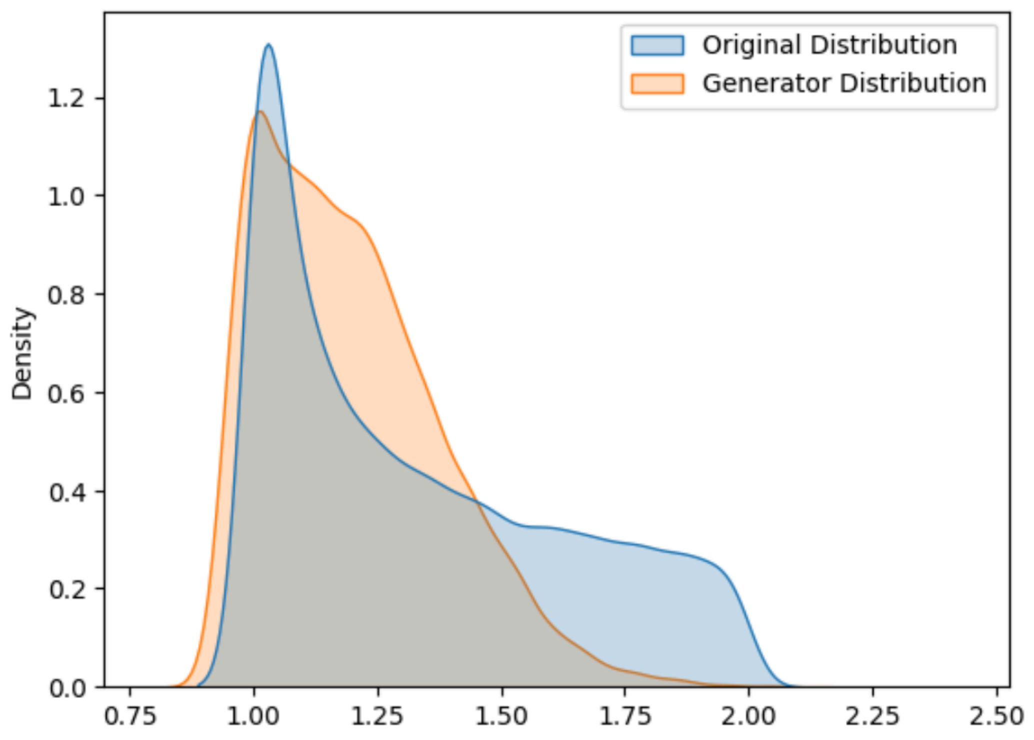
Bimodal

Recall : What are we trying to learn ????

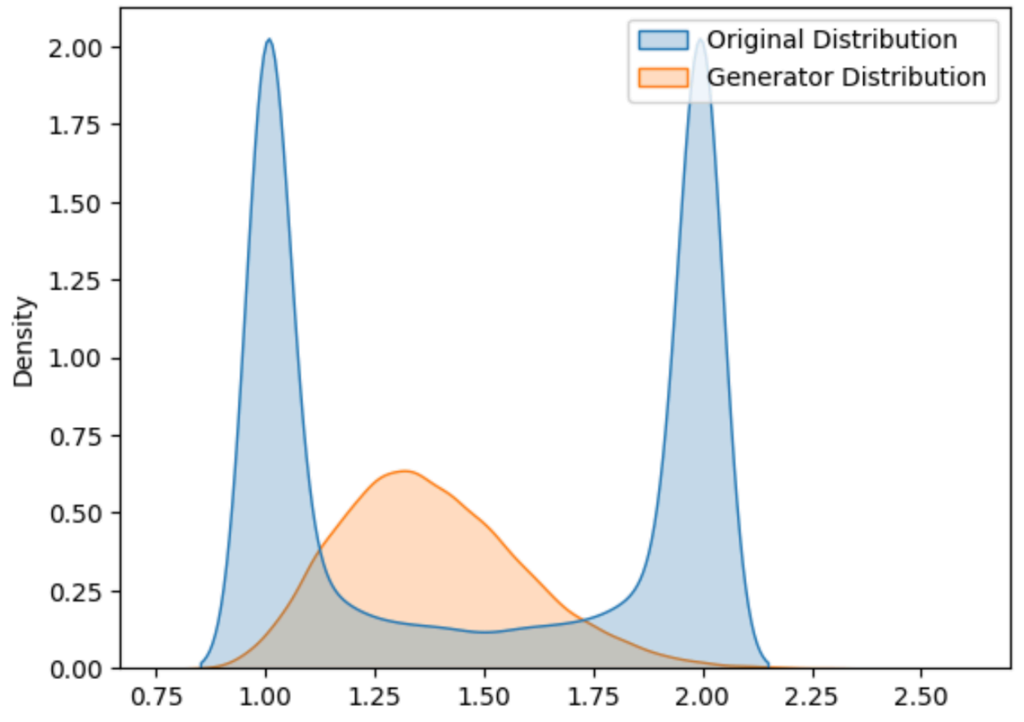
A function that takes in a value $z \sim N(0,1)$ and gives a sample x' that is close to the true distribution p_x

Loss Function and Mode Collapse

Did Cross Entropy Loss work in all cases ?



Yeah probably for single mode



Not quite for bimodal

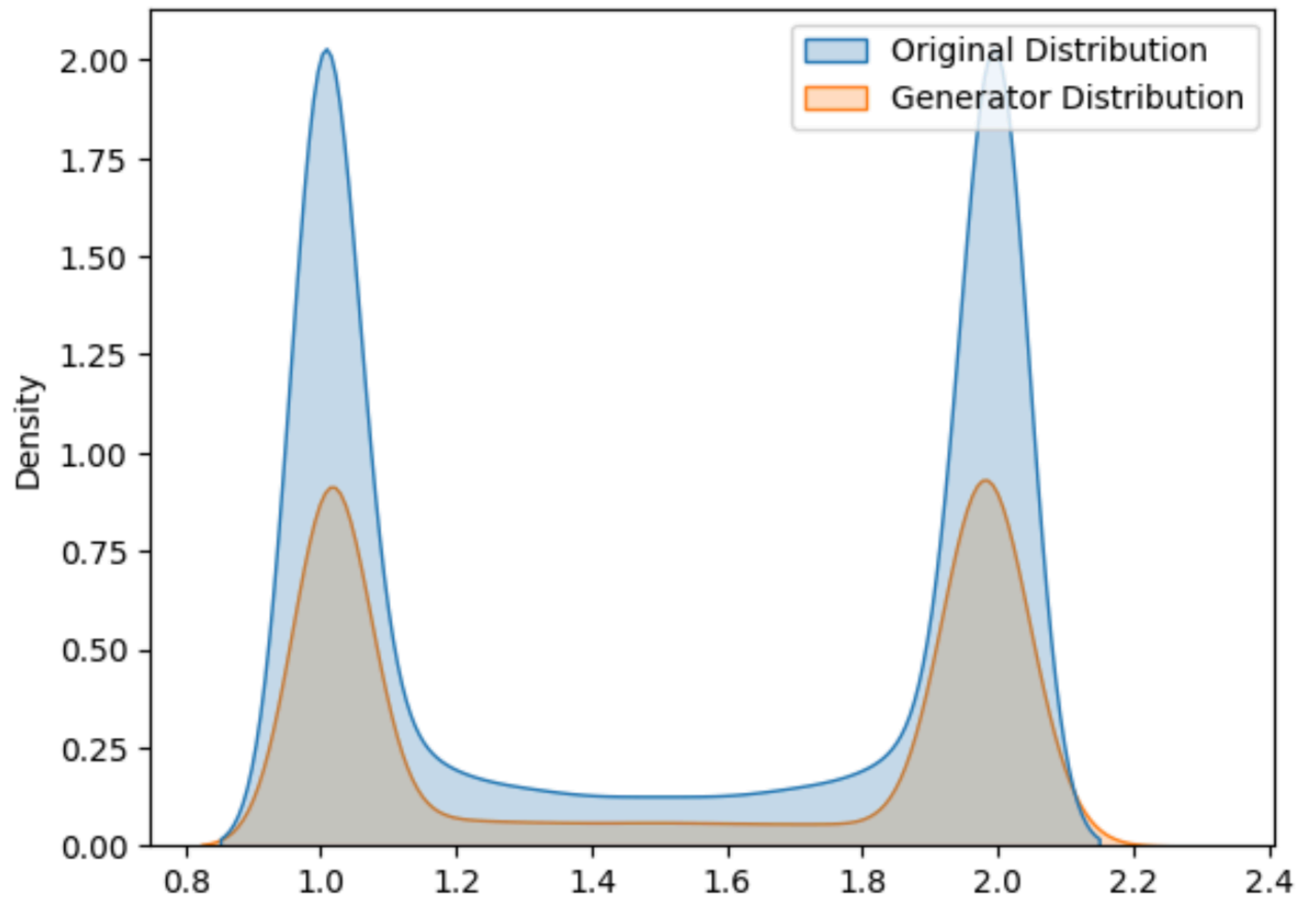
Mode Collapse !!

Wasserstein Loss

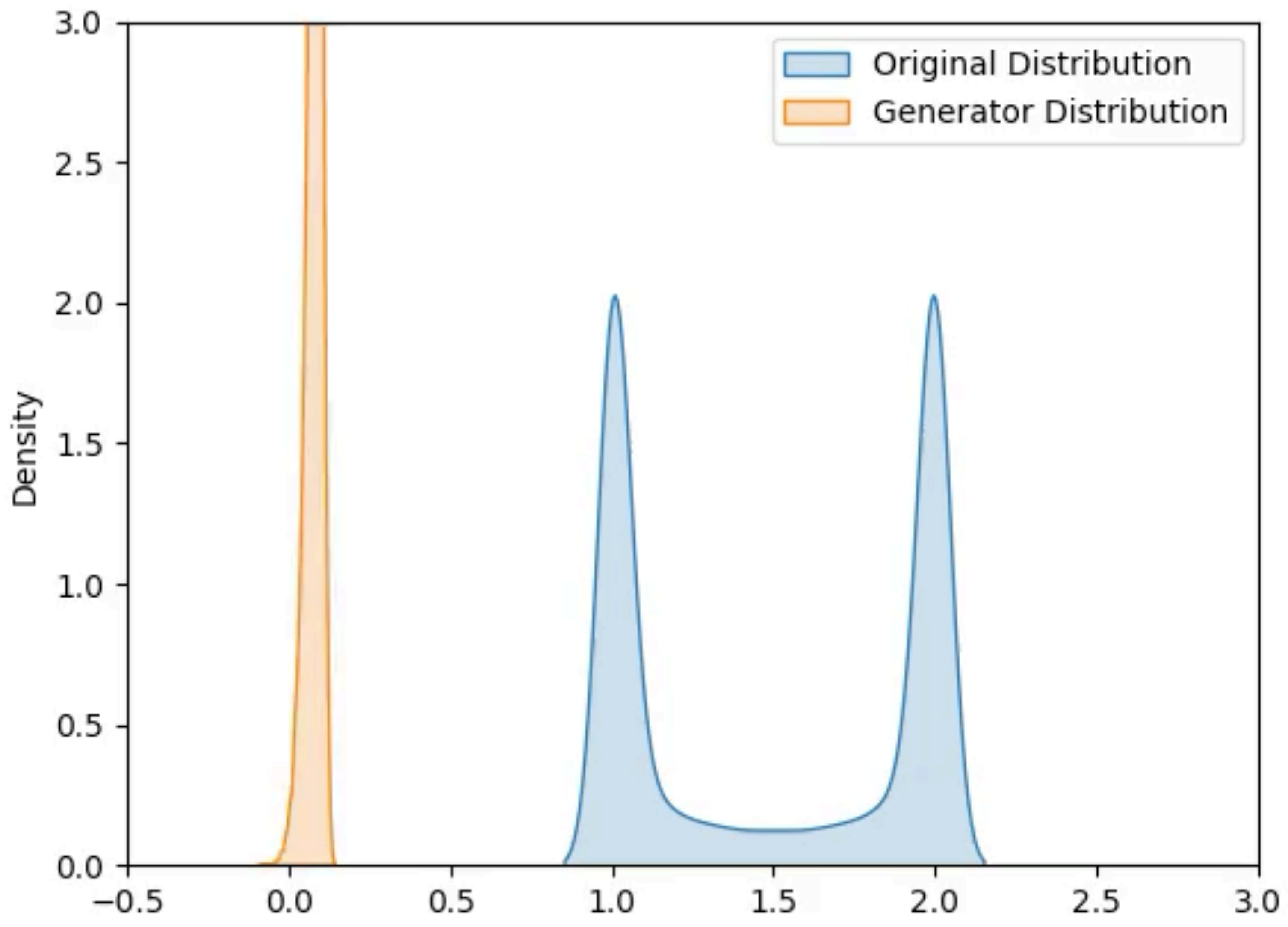
The output of the discriminator is a score

Discriminator loss : $D(x) - D(G(z))$

Generator loss : $D(G(z))$



Earth Mover distance



What all to be aware of ?

Do not loss your sleep over this

Training GAN is hard. Its an understatement. Its really hard to get a stable training

The **learning rate**, yes even the **epsilon values of Adam** needs to be tuned for this

Loss function, like **Wasserstein Loss** can help preventing in mode collapse

Unrolling of GAN : Can help in speed up

Clip the gradients at each updates

Normalizing Flow

Also called as reversible models. X is broken into $[x_1, x_2]$

$$p_X(x) = p_Z(f(x)) \,|\, \det(\frac{\partial f(x)}{\partial x^T}) \,|$$

In this you have something called as forward propagation which is the f that maps $x \rightarrow z$ that is being learnt.

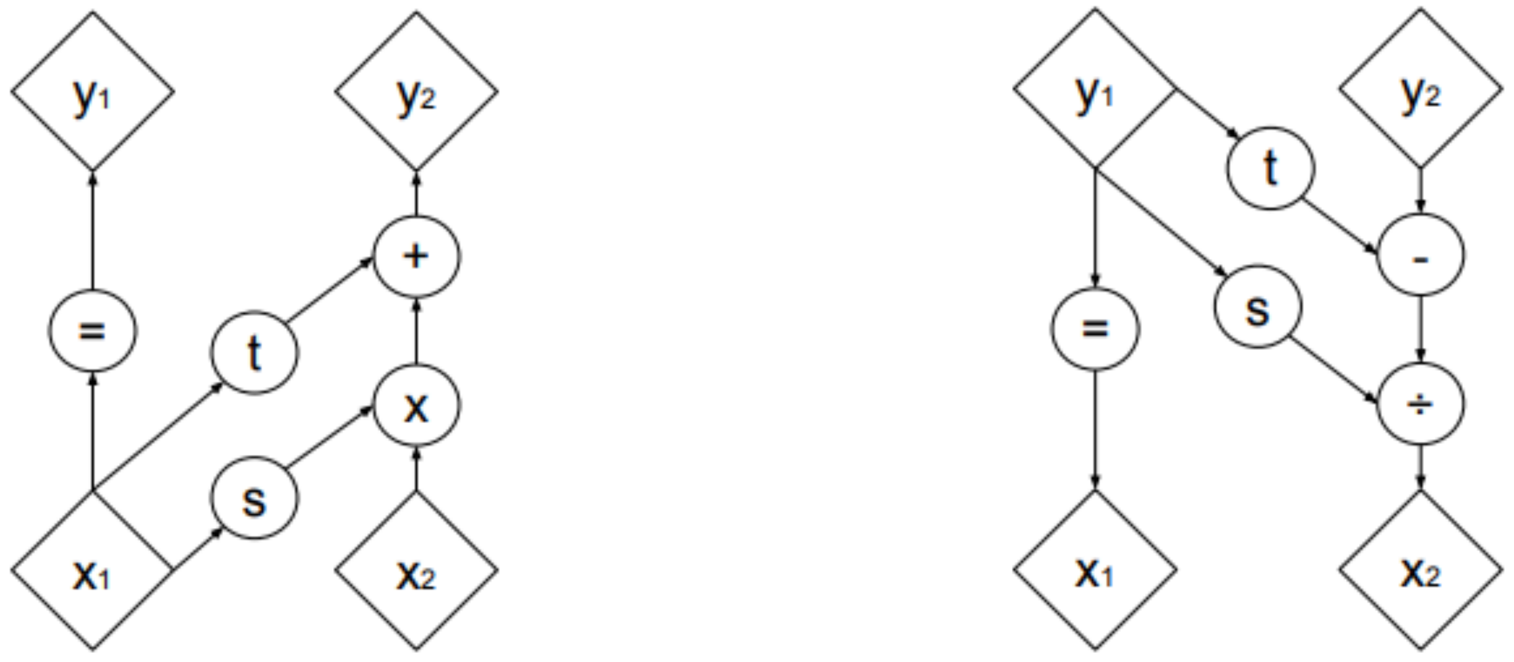
This f **is invertible**. So how do design **neural networks that are invertible** ??

So now our basic neural network is learning the transformations s **and** t.

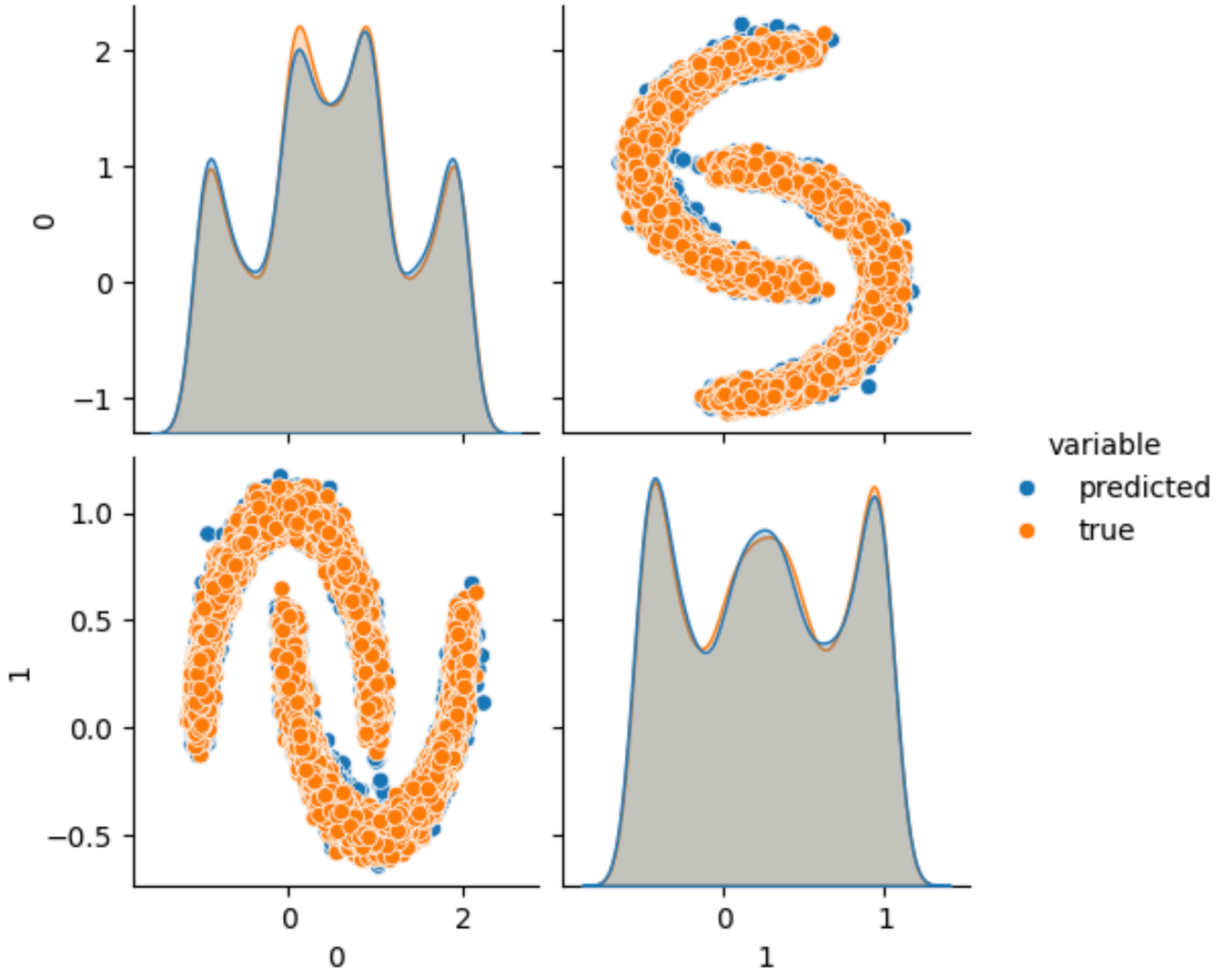
These transformations forms one block. We can now have one more block over this and so on t

In order to generate new samples from $z \sim N(0,1)$, we run through the inverse of these functions through the inverse propagation.

This forces each block to be of the same dimension as the input dimension



(a) Forward propagation (b) Inverse propagation



Variational Auto-Encoder

The overall idea is quite similar.

We take a sample $x \sim p_x$ and then get a representation of it z. However, we do not directly predict z, rather the distribution parameters from which z is sampled from μ, σ.

Then we use the re-parameterization trick to get a sample z:

$$z = \mu + \sigma \odot \epsilon, \epsilon \sim N(0,1)$$

Then we pass through the decoder to produce a sample x'.

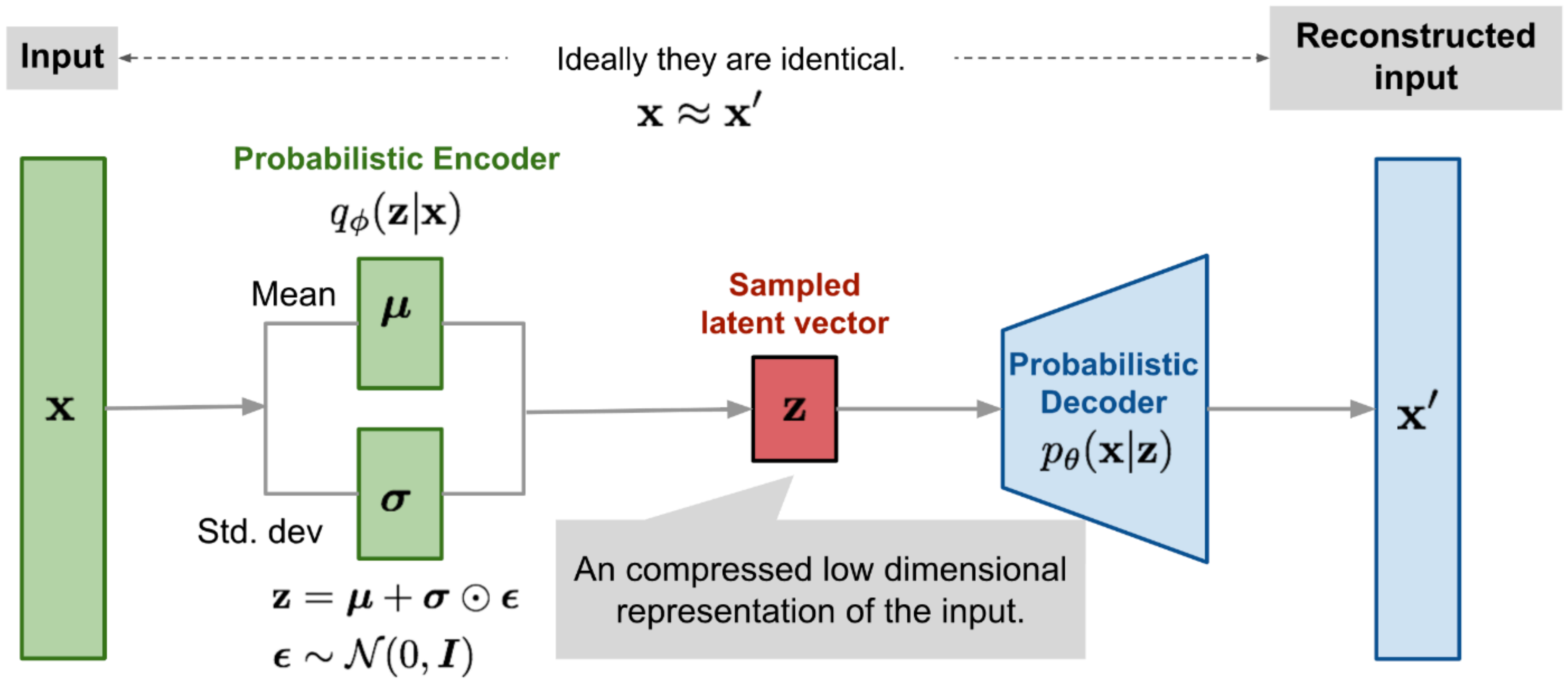
$$\mathbf{E_q}[\log \frac{p(z)}{q(z)}] + \mathbf{E_q}[\log p(x \mid z)]$$

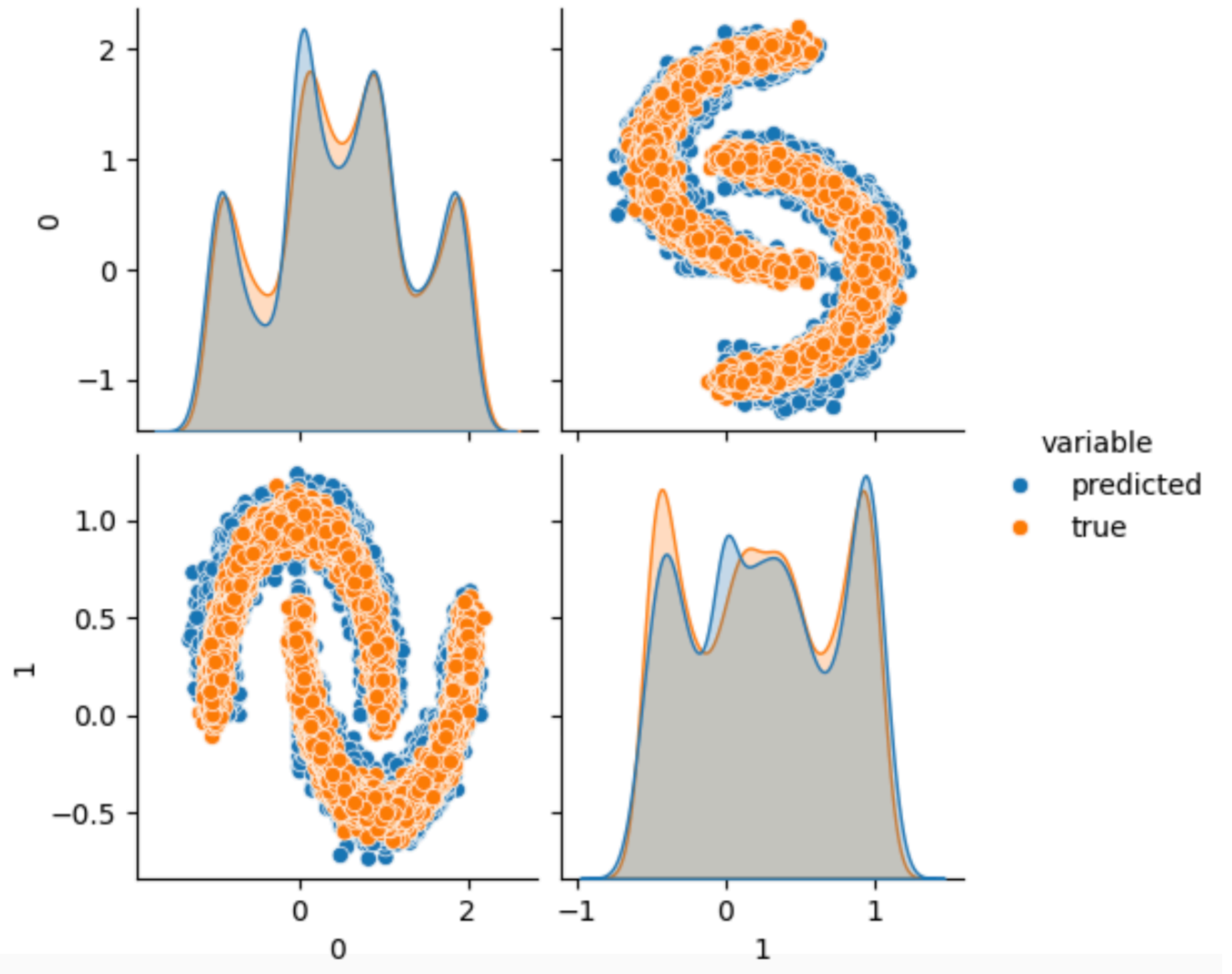
KL Divergence Loss

The derivation of this itself is a maths problem

This is the MSE loss

Reconstruction Loss





Diffusion Model

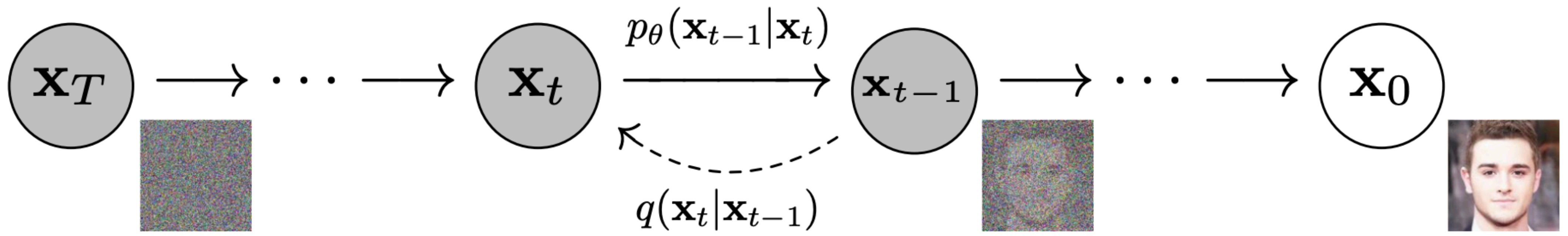
We sample a point $x \sim p_x$ from the real distribution. A forward diffusion process involves adding a small amount of gaussian noise to the sample at each step controlled by variance scheduler β_t. Eventually when $T \to \infty$, x_T resembles a isotropic gaussian ($\Sigma = \sigma^2 I$)

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

So now, we can do the reverse of this as well, called as the reverse diffusion process which takes a isotropic gaussian sample which becomes your code vector z and then gradually form the image or the sample which belongs to the real distribution.

There is a way to formulate x_t in terms of x_0 where $\alpha = 1 - \beta$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \dots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && \text{;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gaussians (*).} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}$$



Post this the maths becomes too complicated.

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
end for
return $\mathbf{x}_0$

Just follow this algorithm