Deep Learning

Angshuman Paul

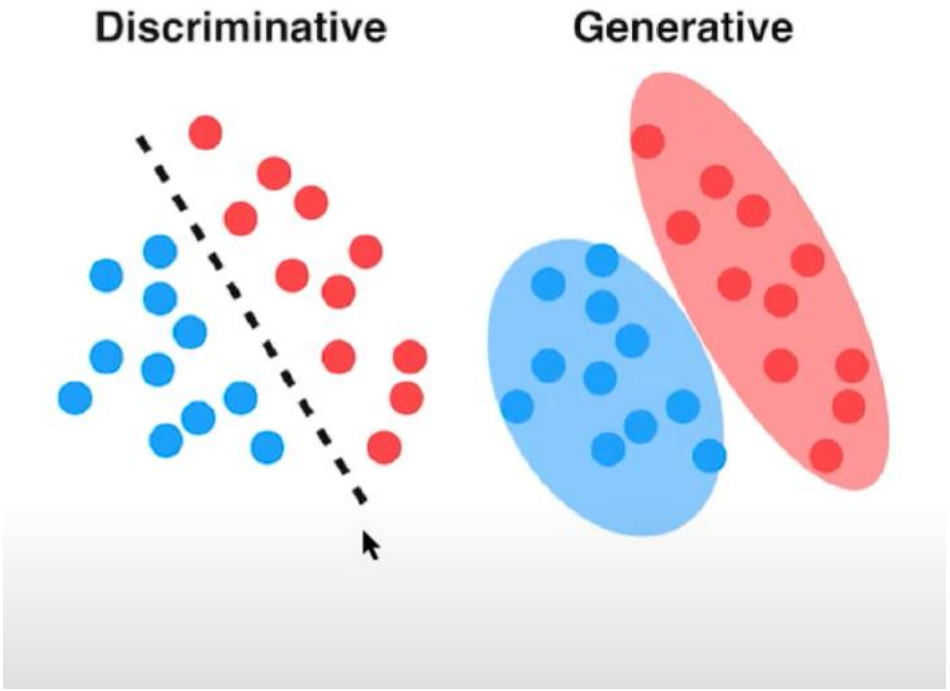
Assistant Professor

Department of Computer Science & Engineering

Deep Generative Models

What are Generative Models?

- Models that can generate new data
- **Discriminative models** aim to learn and utilize differentiating features among classes or groups (such as clusters) in a dataset
- **Generative models** aim to learn the underlying distribution of a dataset or the distribution of different classes in a dataset



https://www.youtube.com/watch?v=v_ksUIpToGk

What are Generative Models?

- Generative models: The basic philosophy
  - **If you understand, you can create**

https://www.youtube.com/watch?v=v_ksUIpToGk

What are Generative Models?

- Generative models: The basic philosophy
  - **If you understand, you can create (loosely)**

- Example: If you understand English grammar, and if you understand cricket, you can write a new paragraph on the cricketing brilliance of Sachin Tendulkar

https://www.youtube.com/watch?v=v_ksUIpToGk

Statistical Generative Models

- Learns using
  - Data
  - Prior knowledge

- There is always a trade-off between how much data and how-much prior knowledge are to be used

Statistical Generative Models

- Let's say I want to design a generative models for synthesizing a particular type of data
  - Image
  - Text
  - Video, etc.
- Let the type of data be denoted by x

- A statistical generative model is a probability distribution $p(x)$

Statistical Generative Models

- A statistical generative model is a probability distribution $p(x)$

- This model will be learnt using
  - Data: samples (e.g., images of animals)
  - Prior knowledge: parametric form (e.g., Gaussian), loss function (e.g., maximum likelihood), optimization algorithm, etc.

https://www.youtube.com/playlist?list=PLoROMvodv4rPOWA-omMM6STXaWW4FvJT8

Statistical Generative Models

- A statistical generative model is a probability distribution $p(x)$

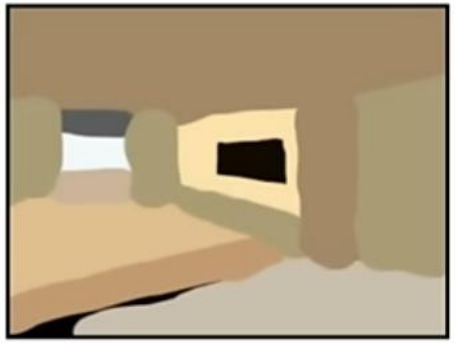
- This model generates through two steps
  - **Learning the distribution**: From the input training data x, learn the probability distribution $p(x)$
    - Can be implicitly or explicitly learnt

  - **Generating new data:** Sample from the distribution to generate new data
    - When your model is good, the generated data will look like the training data

What can Generative Models Do?

- Generate new samples
- Improve quality of data (e.g., image super-resolution)
- Cross-modality synthesis (e.g., Text to image generation)
- May be used for discriminative tasks as well
- May learn good representation of data

What can Generative Models Do?



Generate

Generative model
of realistic images

Stroke paintings to realistic images

[Meng, He, Song, et al., ICLR 2022]

What can Generative Models Do?

"a hedgehog using a calculator"

"a corgi wearing a red bowtie and a purple party hat"

"robots meditating in a vipassana retreat"

"a fall landscape with a small cottage next to a lake"

"a surrealist dream-like oil painting by salvador dalí of a cat playing checkers"

"a professional photo of a sunset behind the grand canyon"

"a high-quality oil painting of a psychedelic hamster dragon"

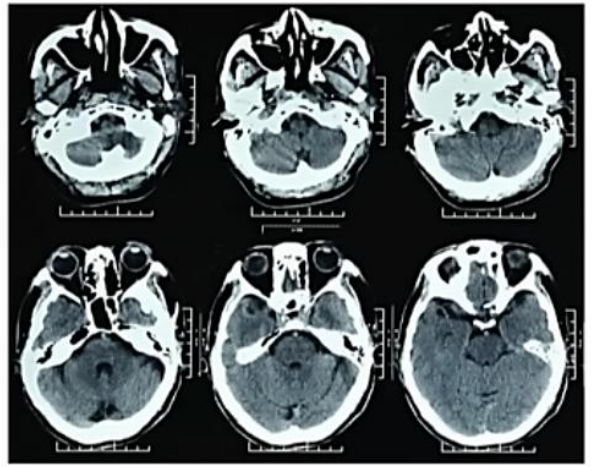
"an illustration of albert einstein wearing a superhero costume"

https://arxiv.org/pdf/2112.10741

What can Generative Models Do?
Solving Inverse Problems

Generative model
of medical images

Generate



Medical image reconstruction

[Song et al., ICLR 2022]

Progress in Generative Modelling

2014

2015

2016

2017

2018

Progress in Inverse Problems

P(high resolution | low resolution)

Menon et al, 2020

P(full image| mask)

Liu al, 2018

P(color image| greyscale)

Antic, 2020

Learning a Generative Model

- We are given a set of training examples $x_1, x_2, \ldots$ (say various images of dogs)

Learning a Generative Model

- **The assumption:** All the training examples $x_1, x_2, \ldots$ (say various images of dogs) come from some true underlying data generating process
  - Each training data x_i is obtained from the distribution P_D
  - P_D is unknown to us

Learning a Generative Model

- **The assumption:** All the training examples $x_1, x_2, \ldots$ (say various images of dogs) come from some true underlying data generating process
  - Each training data x_i is obtained from the distribution P_D
  - P_D is unknown to us

- If we can learn P_D, we may sample from P_D and generate new data points

Learning a Generative Model

- **The assumption:** All the training examples $x_1, x_2, \ldots$ (say various images of dogs) come from some true underlying data generating process
  - Each training data x_i is obtained from the distribution P_D
  - P_D is unknown to us

- If we can learn P_D, we may sample from P_D and generate new data points

- **Our Goal:** **Learning the data generating distribution P_D**

Quick Refresher

- **What is sampling from a distribution and how to do it?**

Inverse Transform Sampling

- This method is used to sample from a distribution when its cumulative distribution function (CDF) is known.

- Generate a random uniform variable U in the interval ($[0, 1]$).
- Find the value X such that $F_X(X) = U$, where F_X is the CDF of the distribution.

- For an exponential distribution with rate λ:

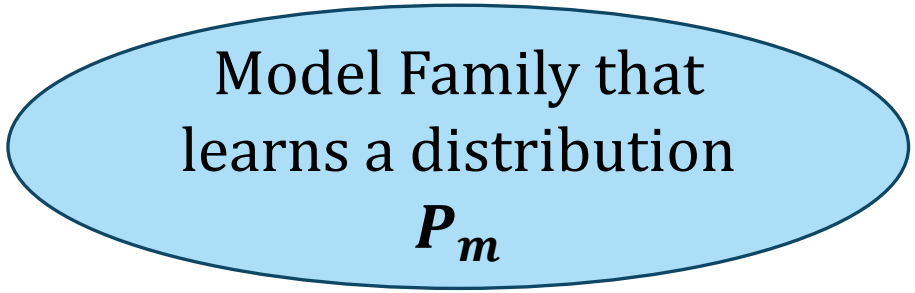
$$F_X(x) = 1 - \exp\{-\lambda x\}$$

Let ($U \sim \text{Uniform}(0, 1)$). Set ($F_X(X) = U$):

$$U = 1 - \exp\{-\lambda x\} \Rightarrow X = -\frac{1}{\lambda}\ln(1 - U)$$

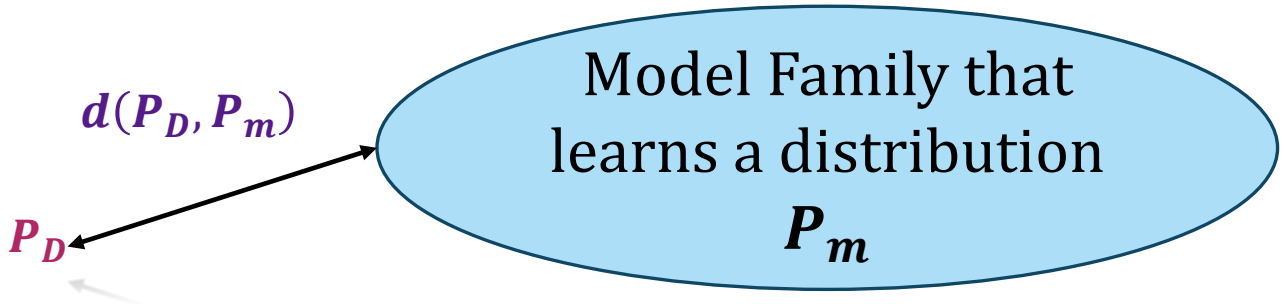
Learning a Generative Model

- **Our Goal:** **Learning the data generating distribution P_D**



Learning a Generative Model

- **Our Goal: Learning the data generating distribution P_D**



Let d be the distance between two distributions

We want to minimize $d(P_D, P_m)$

Learning a Generative Model: Challenges

- Which model family is to be used?
- How to define the distance between the two distributions?

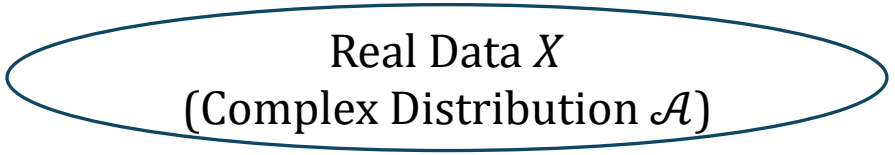
Our First Generative Approach

Our First Generative Approach

- Distribution of real data
  - Complex
  - Difficult to model

- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data

Our First Generative Approach



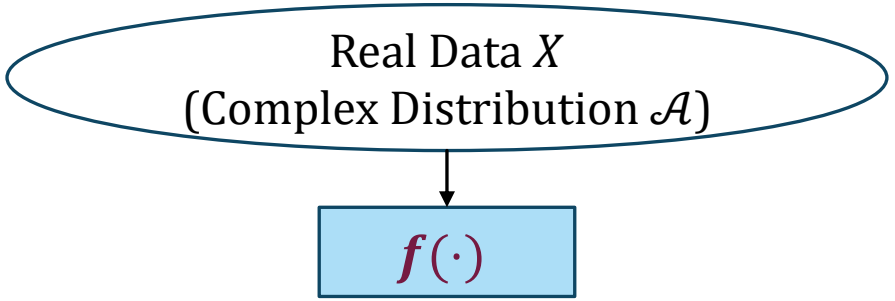
- Distribution of real data
  - Complex
  - Difficult to model

- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data

Our First Generative Approach

- Distribution of real data
  - Complex
  - Difficult to model

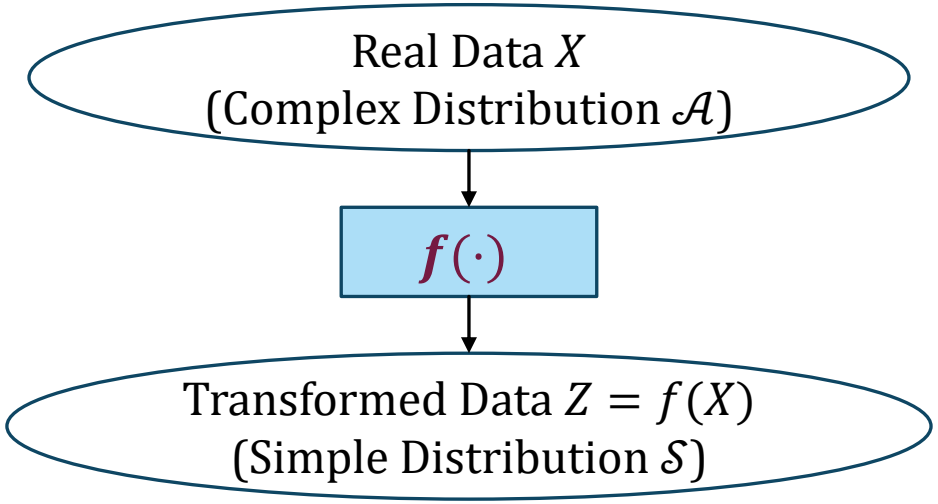
- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data



Our First Generative Approach

- Distribution of real data
  - Complex
  - Difficult to model

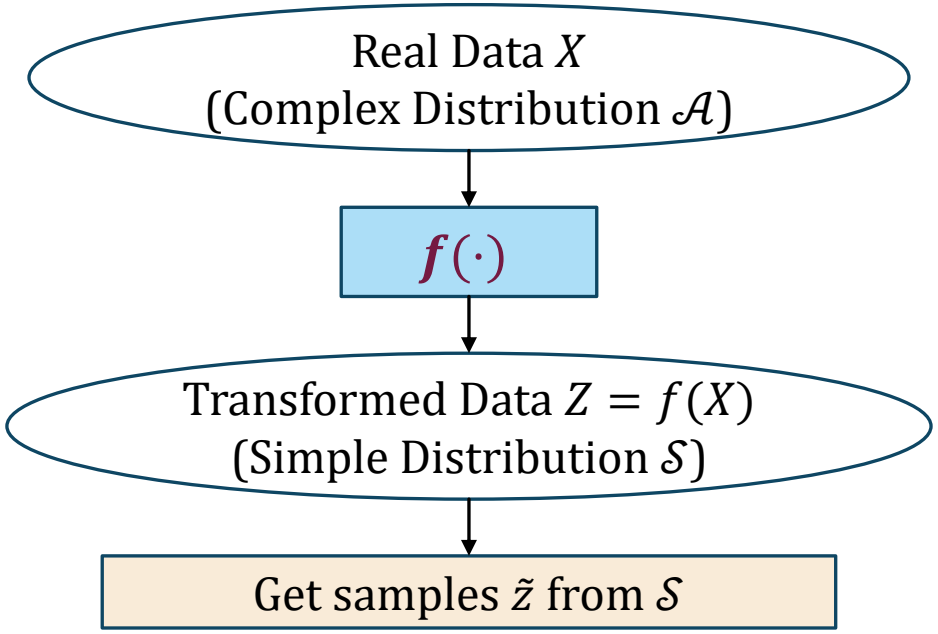
- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data



Our First Generative Approach

- Distribution of real data
  - Complex
  - Difficult to model

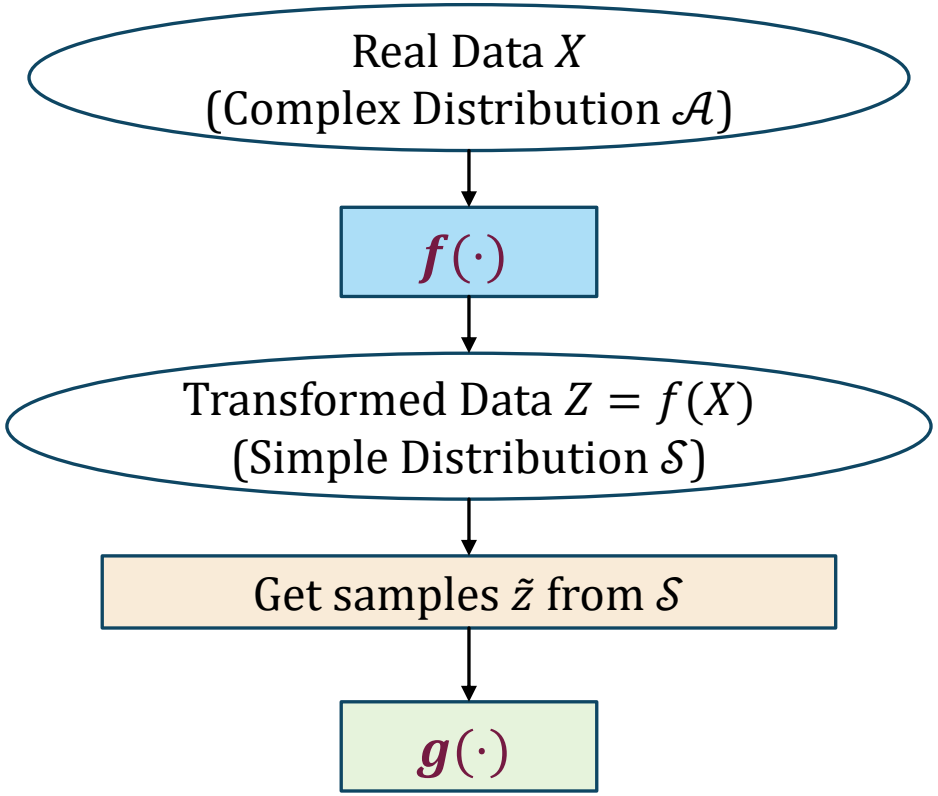
- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data



Our First Generative Approach

- Distribution of real data
  - Complex
  - Difficult to model

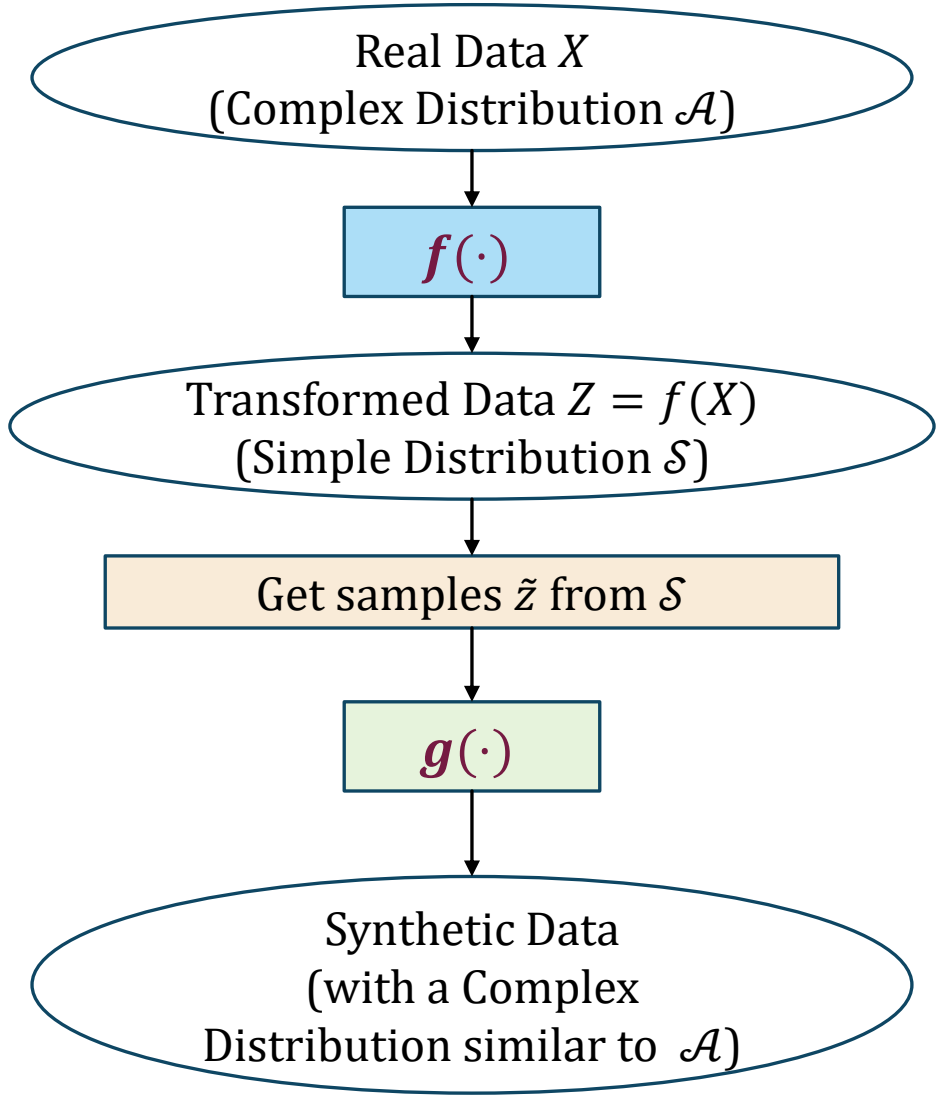
- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data



Our First Generative Approach

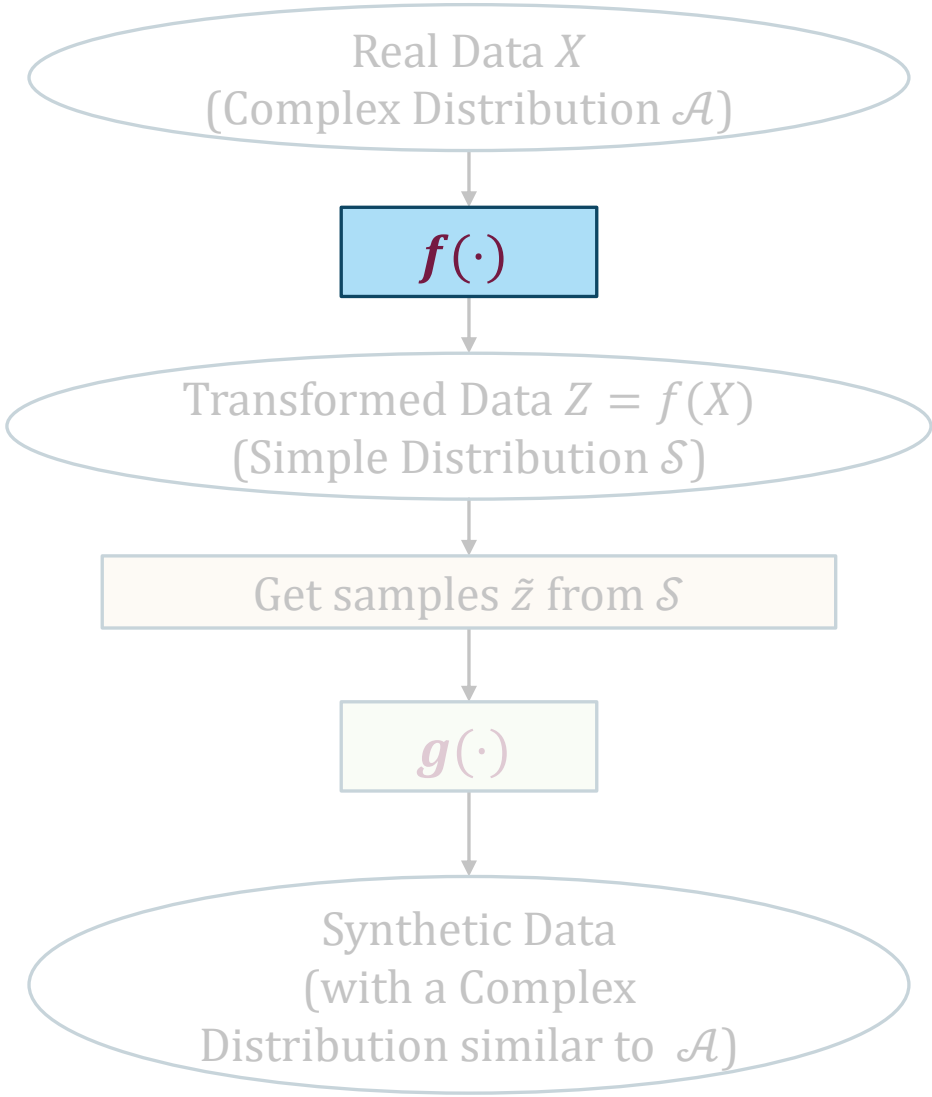
- Distribution of real data
  - Complex
  - Difficult to model

- Can we design a function that will transform the complex distribution of a real data into a simple distribution?
  - If we can, we may be able to sample from the simple distribution
  - Subsequently, if we can implement a (kind of) inverse function, the function can take the sample as input and produce a synthetic data



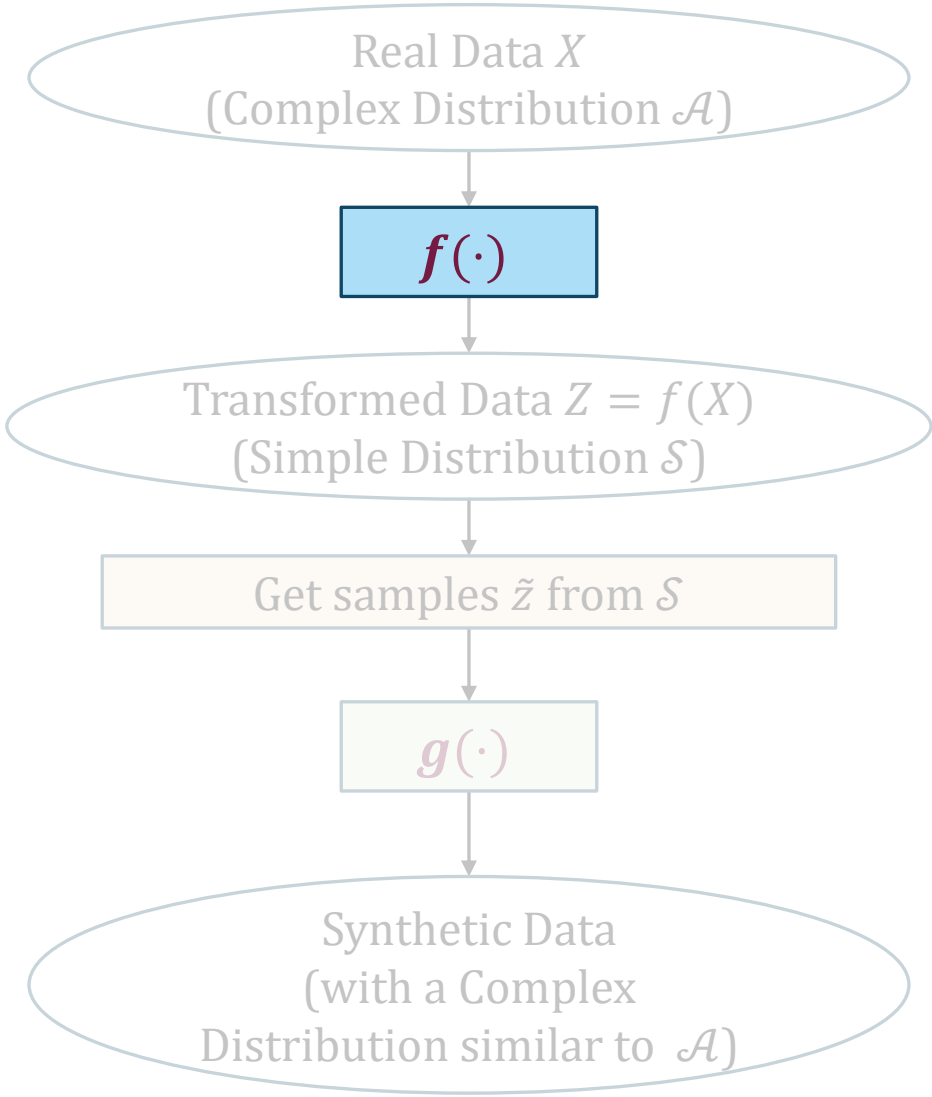
Our First Generative Approach

- Step 1: How to get $f(\cdot)$?



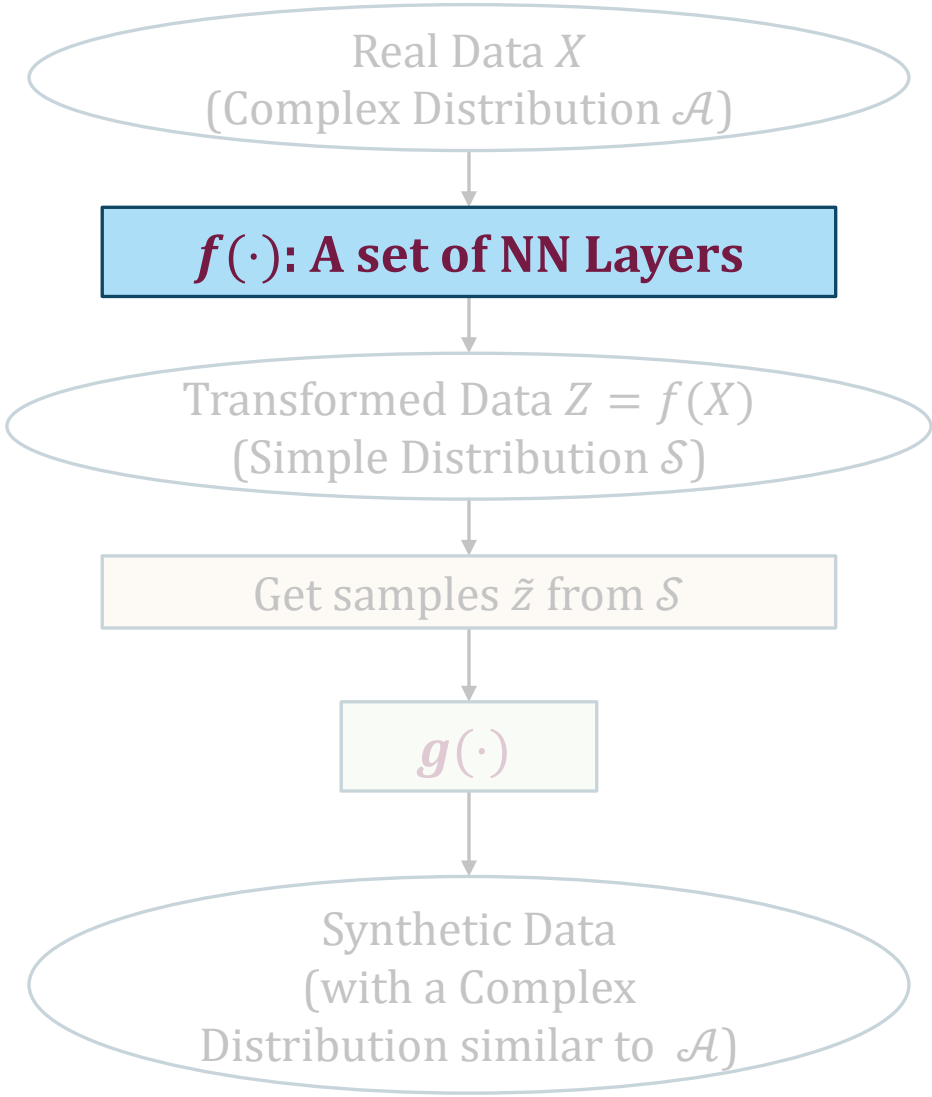
Our First Generative Approach

- Step 1: How to get $f(\cdot)$?
  - May be a set of neural network layers



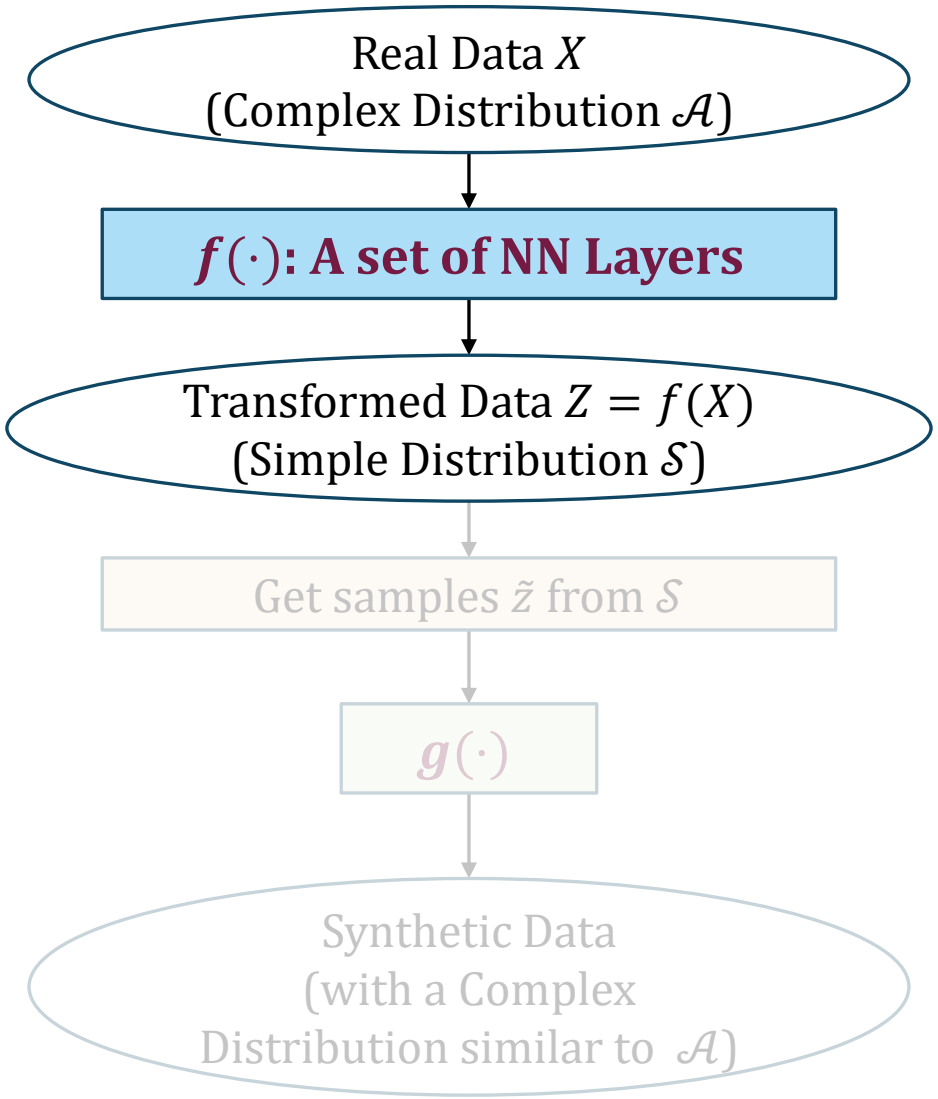
Our First Generative Approach

- Step 1: How to get $f(\cdot)$?
  - May be a set of neural network layers



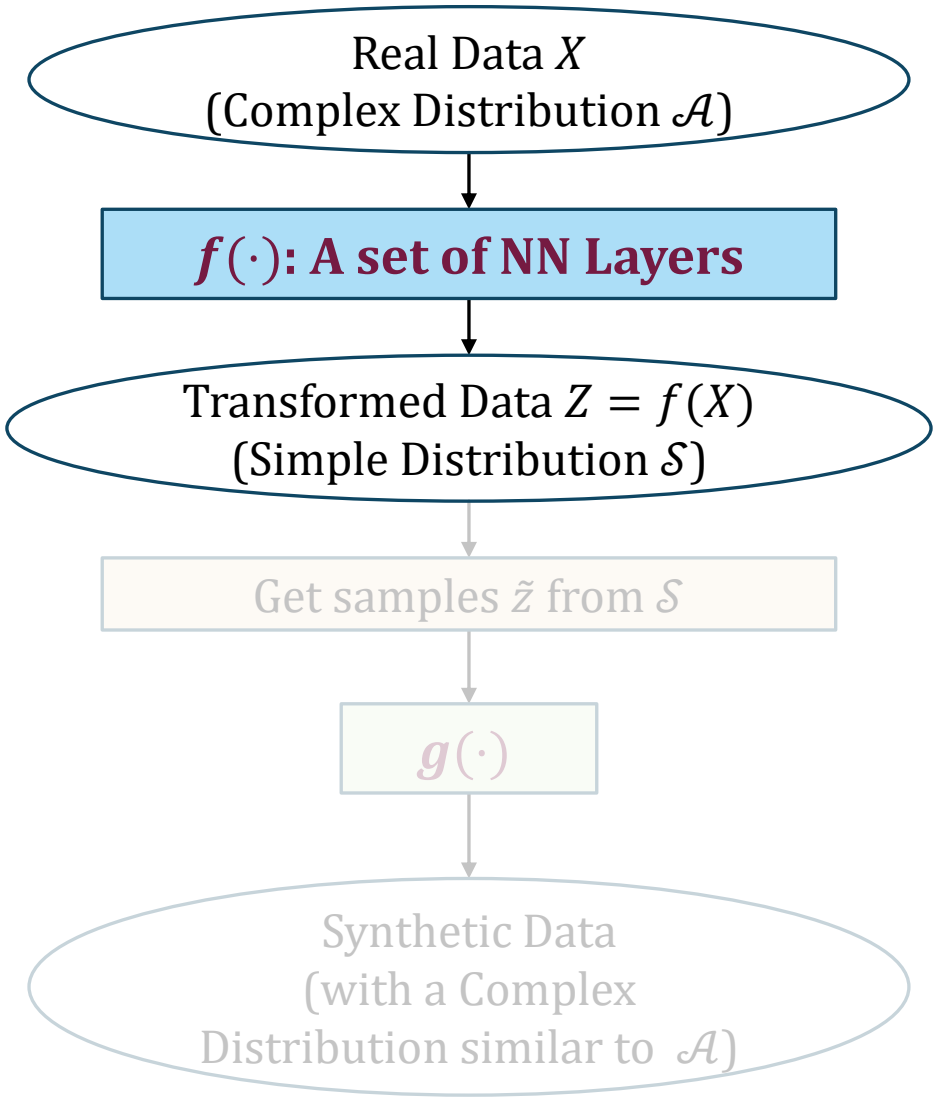
Our First Generative Approach

- Step 2: How do I know that $Z = f(X)$ follows my desired simple distribution?



Our First Generative Approach

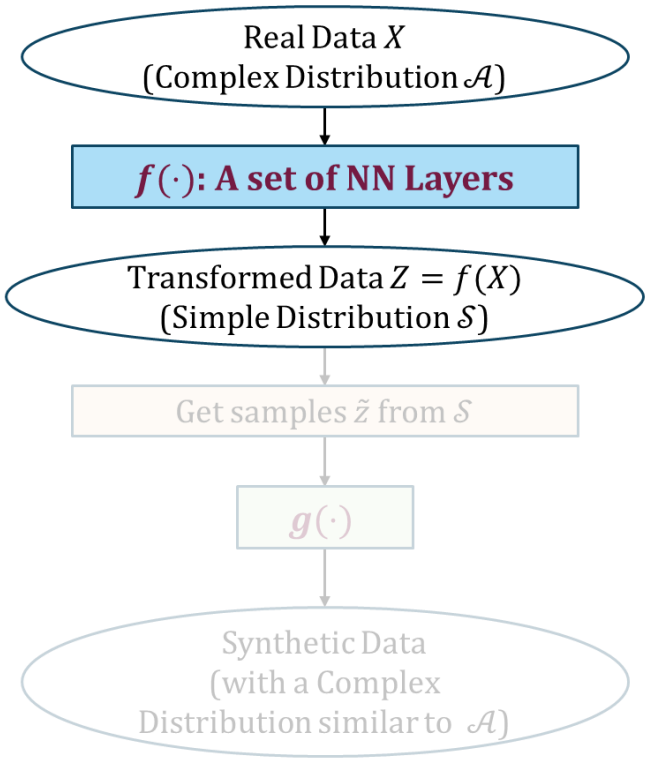
- Step 2: How do I know that $Z = f(X)$ follows my desired simple distribution?
  - Let the desired simple distribution be $\mathcal{N}(0,1)$
    - Reference distribution
  - Calculate the distance between the distribution of Z and $\mathcal{N}(0,1)$
    - KL divergence
    - Assume that Z has a normal distribution with mean μ and standard deviation σ



Our First Generative Approach

- Step 2: How do I know that $Z = f(X)$ follows my desired simple distribution?
  - Calculate the distance between the distribution of Z and $\mathcal{N}(0,1)$
    - KL divergence
    - Assume that Z has a normal distribution with mean μ and standard deviation σ (and the distribution is a multivariate diagonal distribution of dimension k)

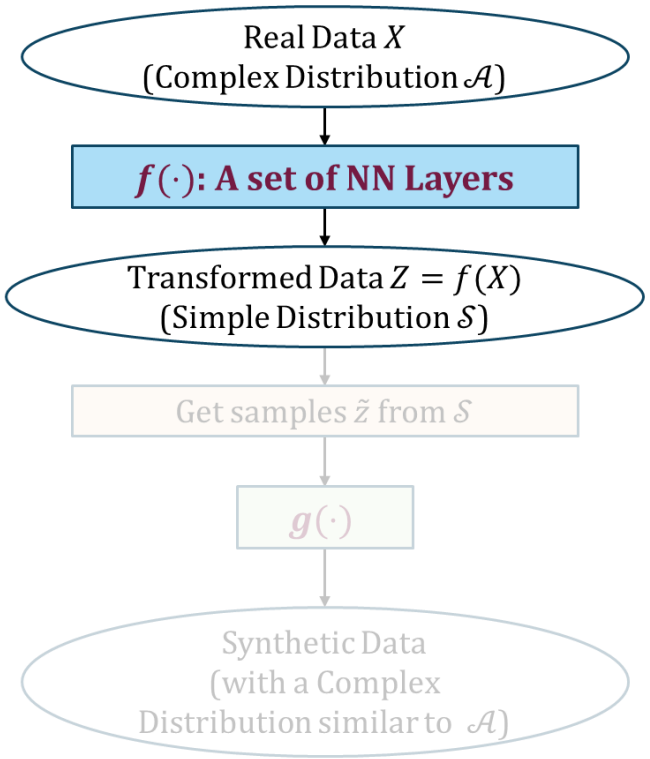
$$KL(\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \ln\sigma_i^2)$$



Our First Generative Approach

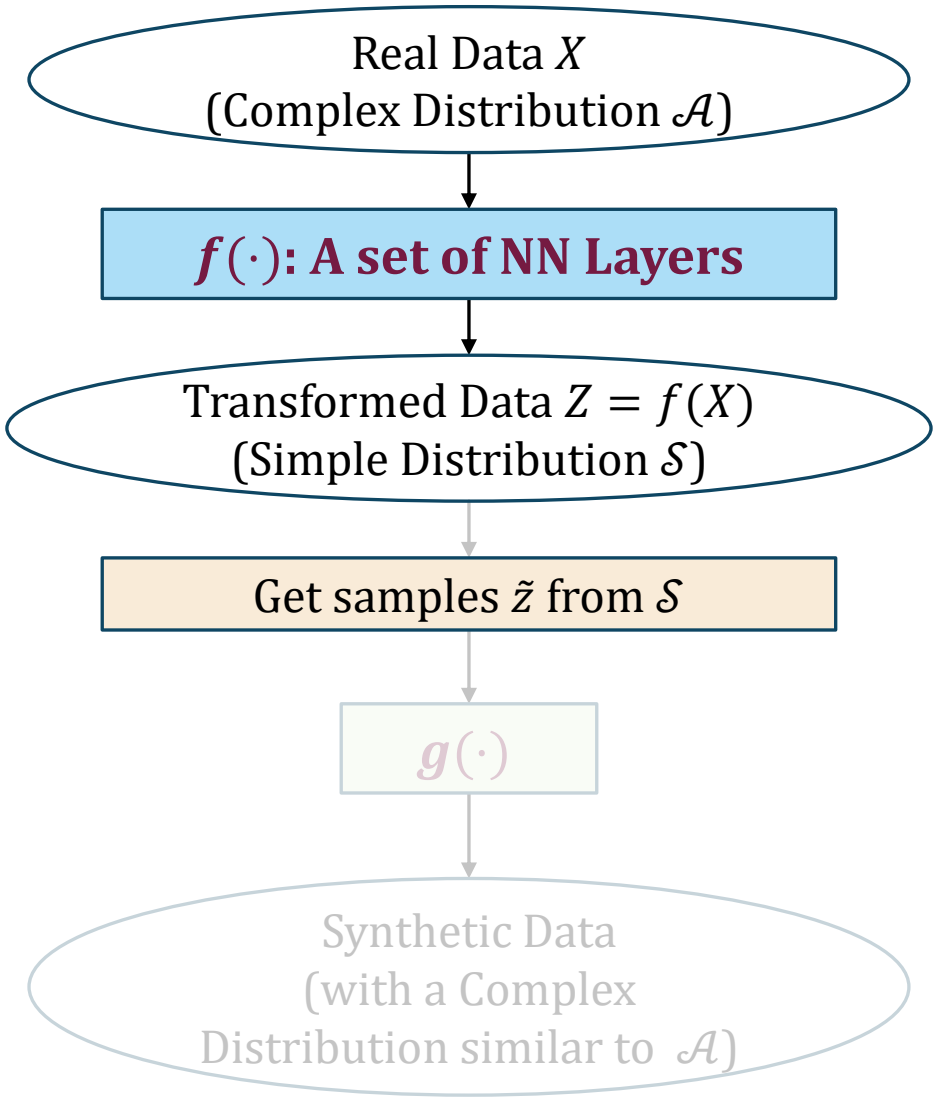
- Step 2: How do I know that $Z = f(X)$ follows my desired simple distribution?
  - Calculate the distance between the distribution of Z and $\mathcal{N}(0,1)$
    - KL divergence
    - Assume that Z has a normal distribution with mean μ and standard deviation σ (and the distribution is a multivariate diagonal distribution of dimension k)
      - No correlation between the dimensions of z
      - The non-diagonal terms of the covariance matrix are zero

$$KL(\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \ln\sigma_i^2)$$



Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?

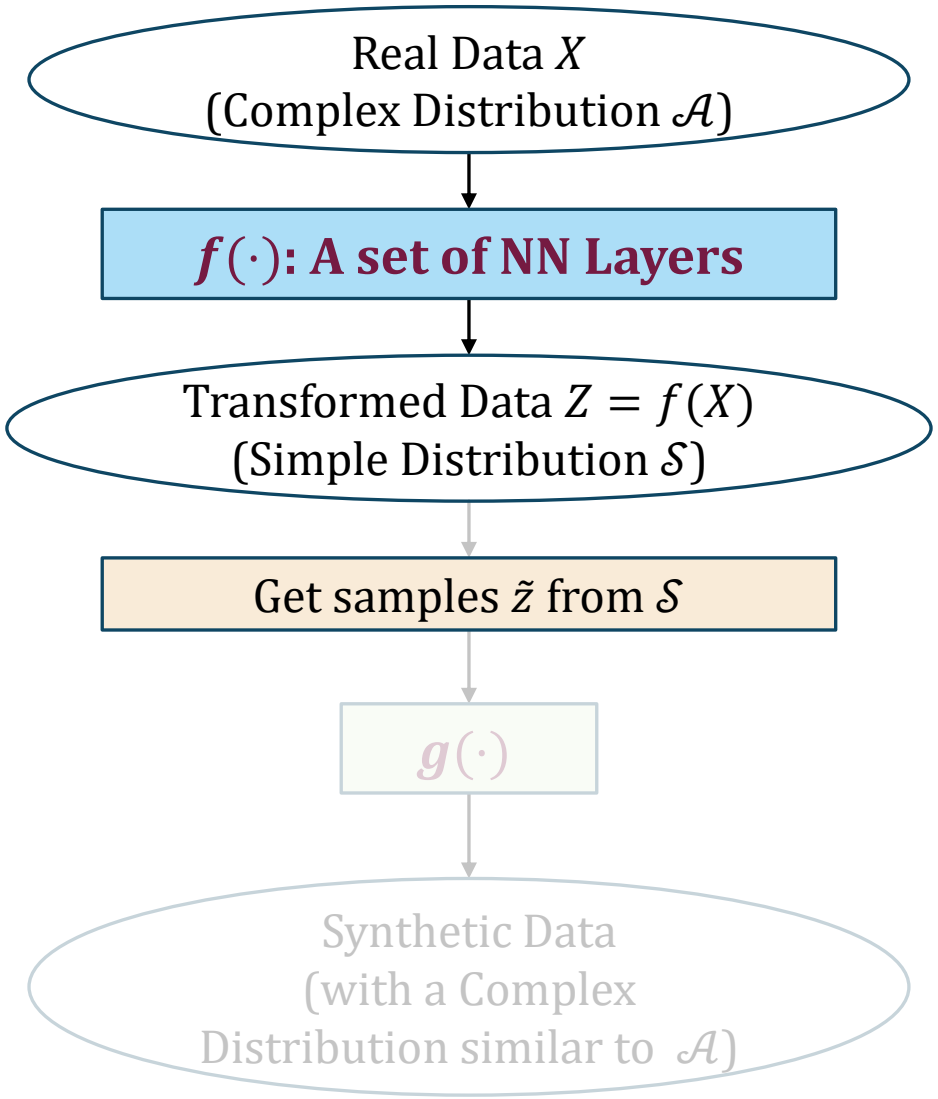


Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?
  - Suppose, I have already got μ and σ of $\mathcal{S}$ through the NN layers
  - Then, sample

$$\tilde{z} = \mu + \epsilon\sigma$$

ϵ is a random number sampled from a standard normal distribution



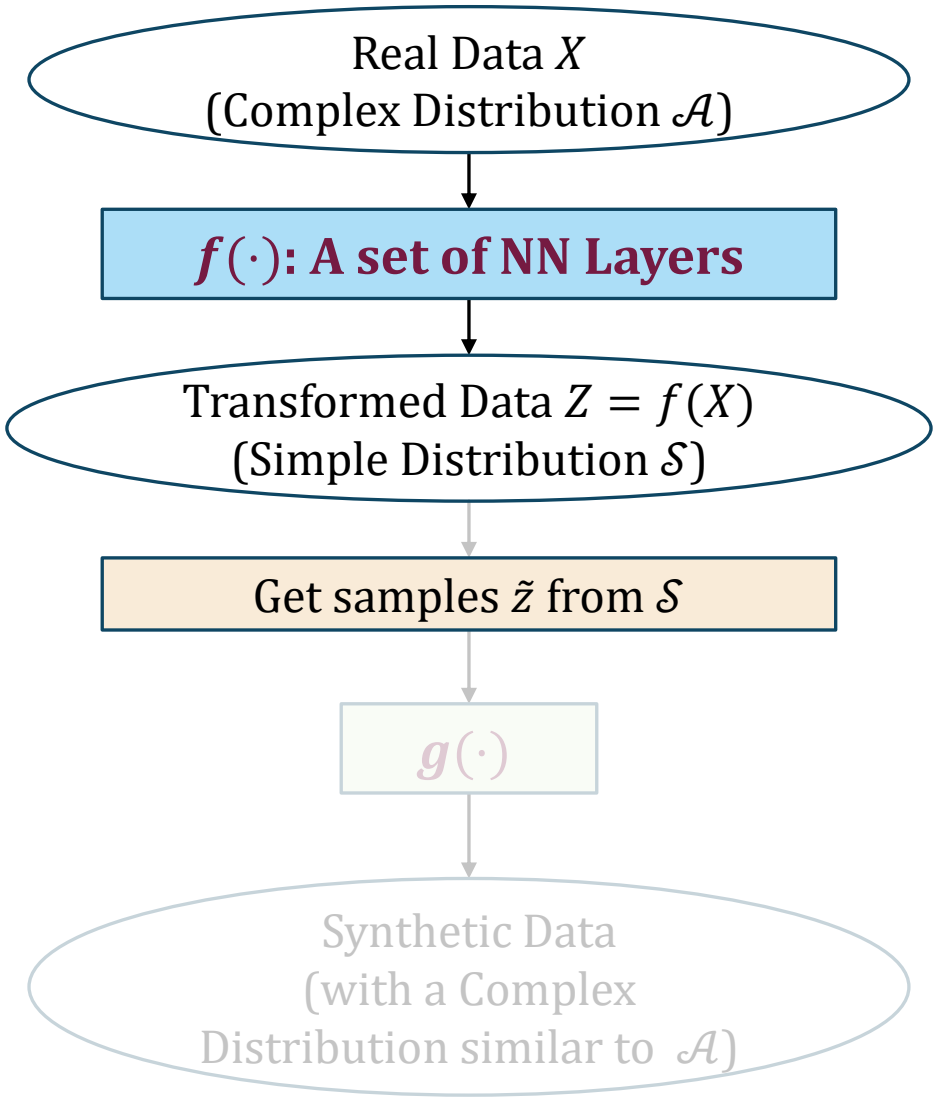
Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?
  - Then, sample

$$\tilde{z} = \mu + \epsilon\sigma$$

ϵ is a random number sampled from a standard normal distribution

  - **But, sampling is not differentiable. So, what to do?**



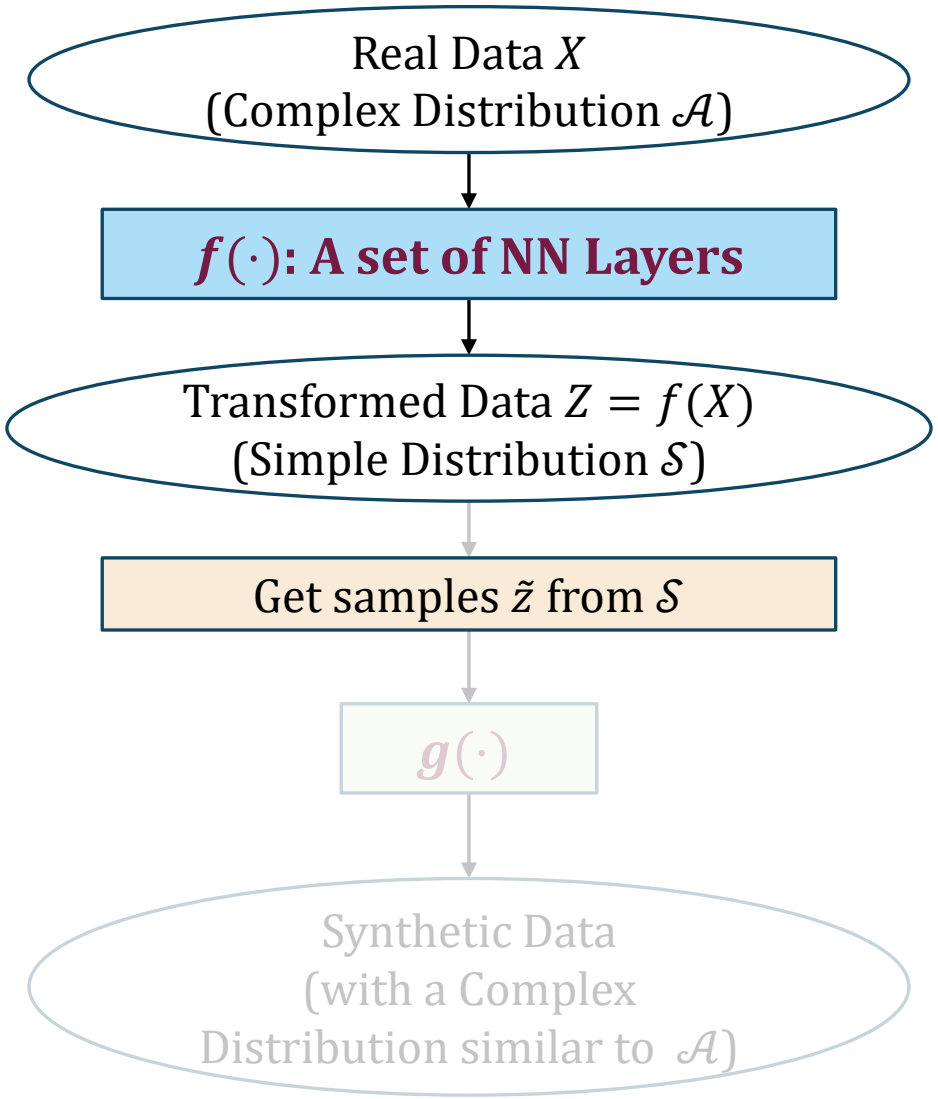
Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?
  - Then, sample

$$\tilde{z} = \mu + \epsilon\sigma$$

ϵ is a random number sampled from a standard normal distribution

  - **But, sampling is not differentiable. We need differentiability for training neural nets. So, what to do?**
    - **Reparameterization**



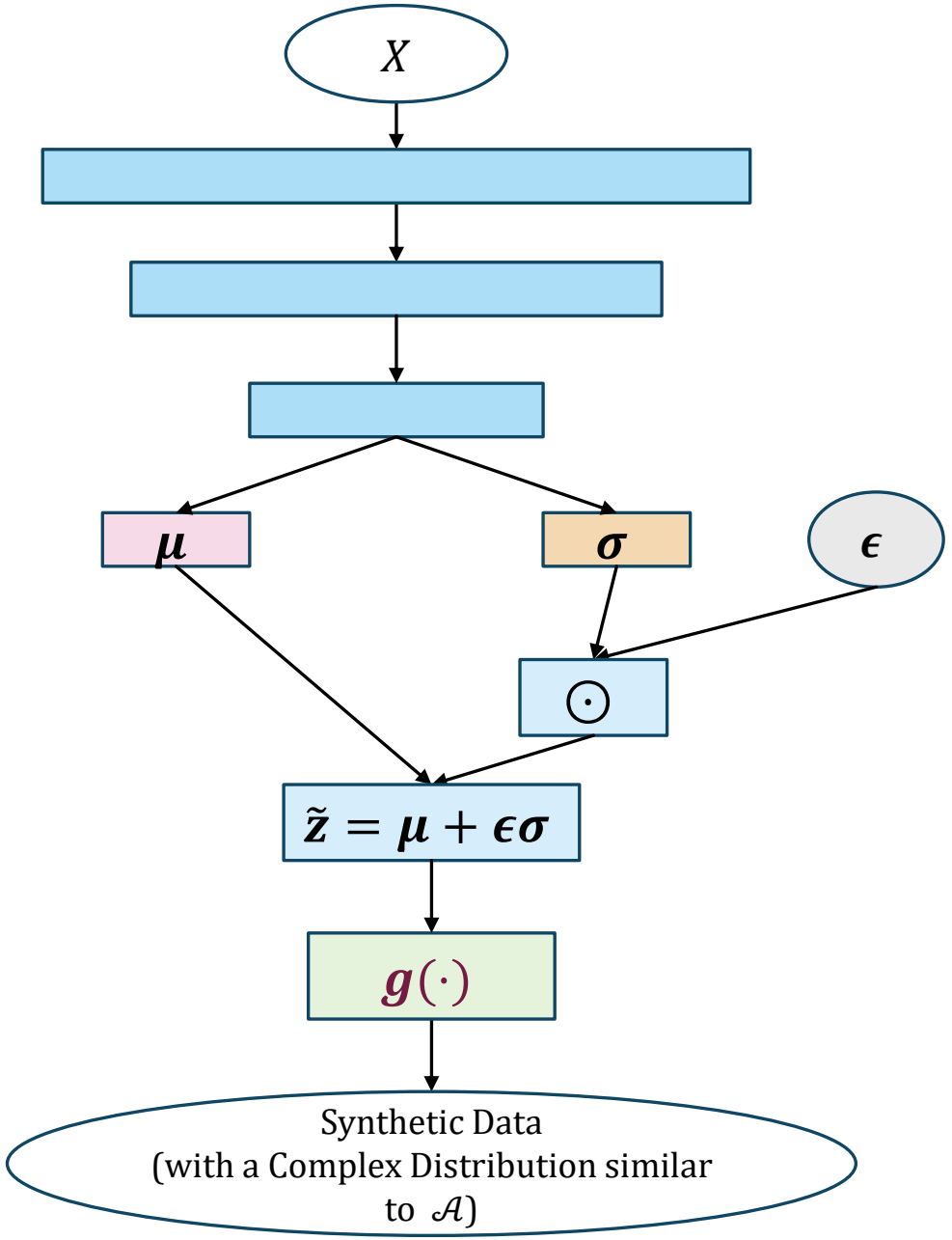
Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?
  - Then, sample

$$\tilde{z} = \mu + \epsilon\sigma$$

ϵ is a random number sampled from a standard normal distribution

- **But, sampling is not differentiable. We need differentiability for training neural nets. So, what to do?**
  - **Reparameterization**



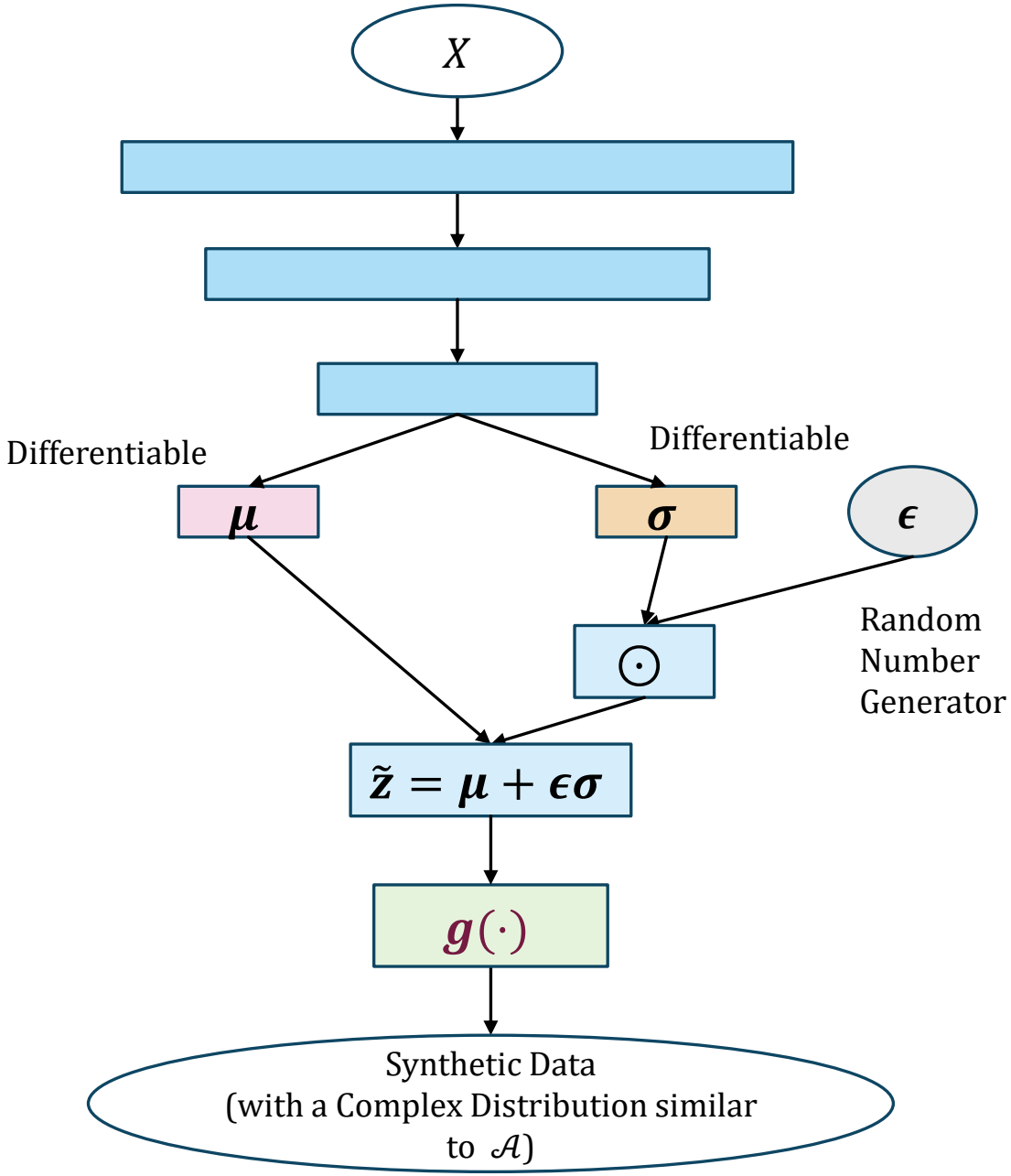
Our First Generative Approach

- Step 3: How to get samples from the simple distribution $\mathcal{S}$?
  - Then, sample

$$\tilde{z} = \mu + \epsilon\sigma$$

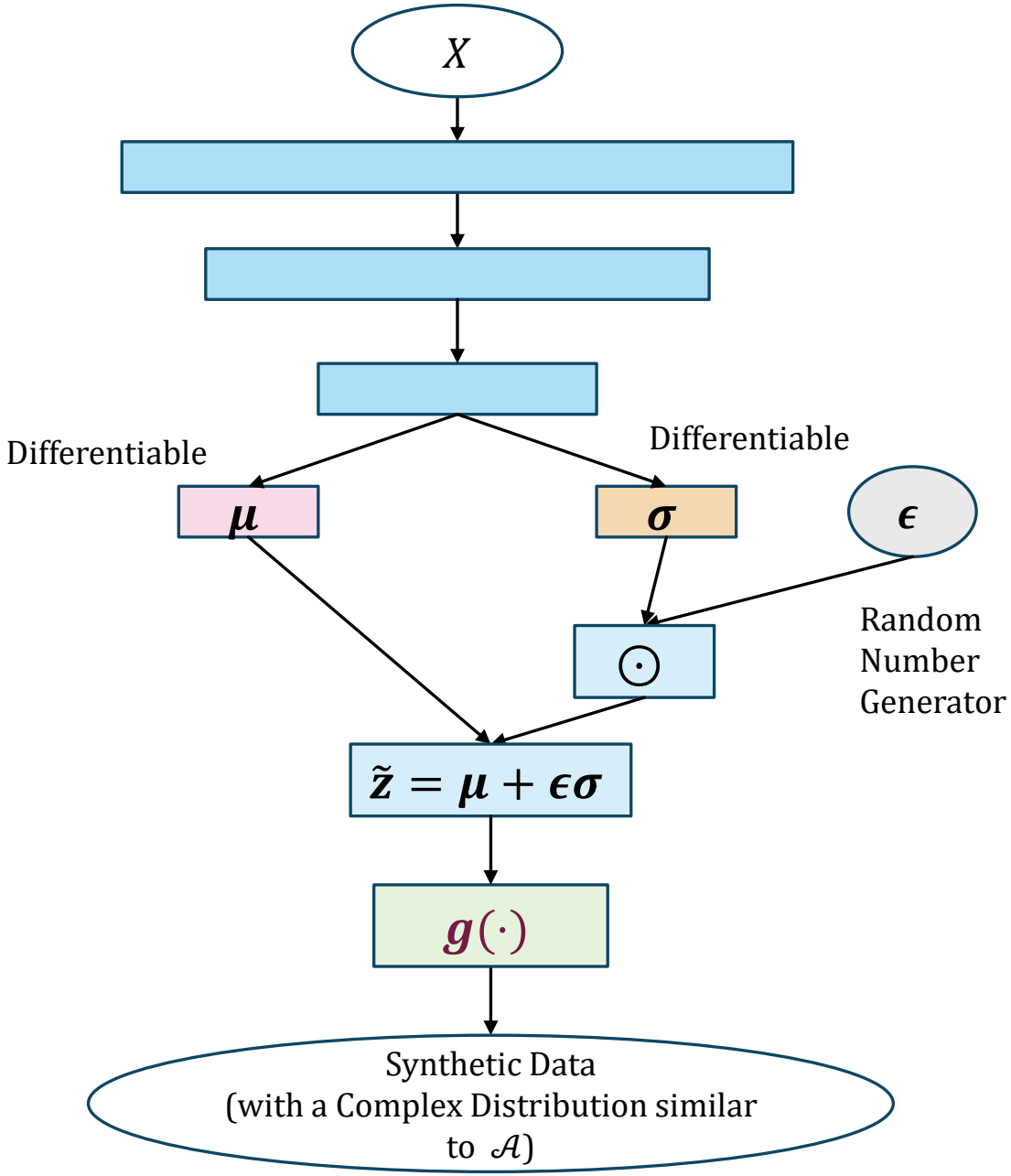
ϵ is a random number sampled from a standard normal distribution

- **But, sampling is not differentiable. We need differentiability for training neural nets. So, what to do?**
  - **Reparameterization**



Our First Generative Approach

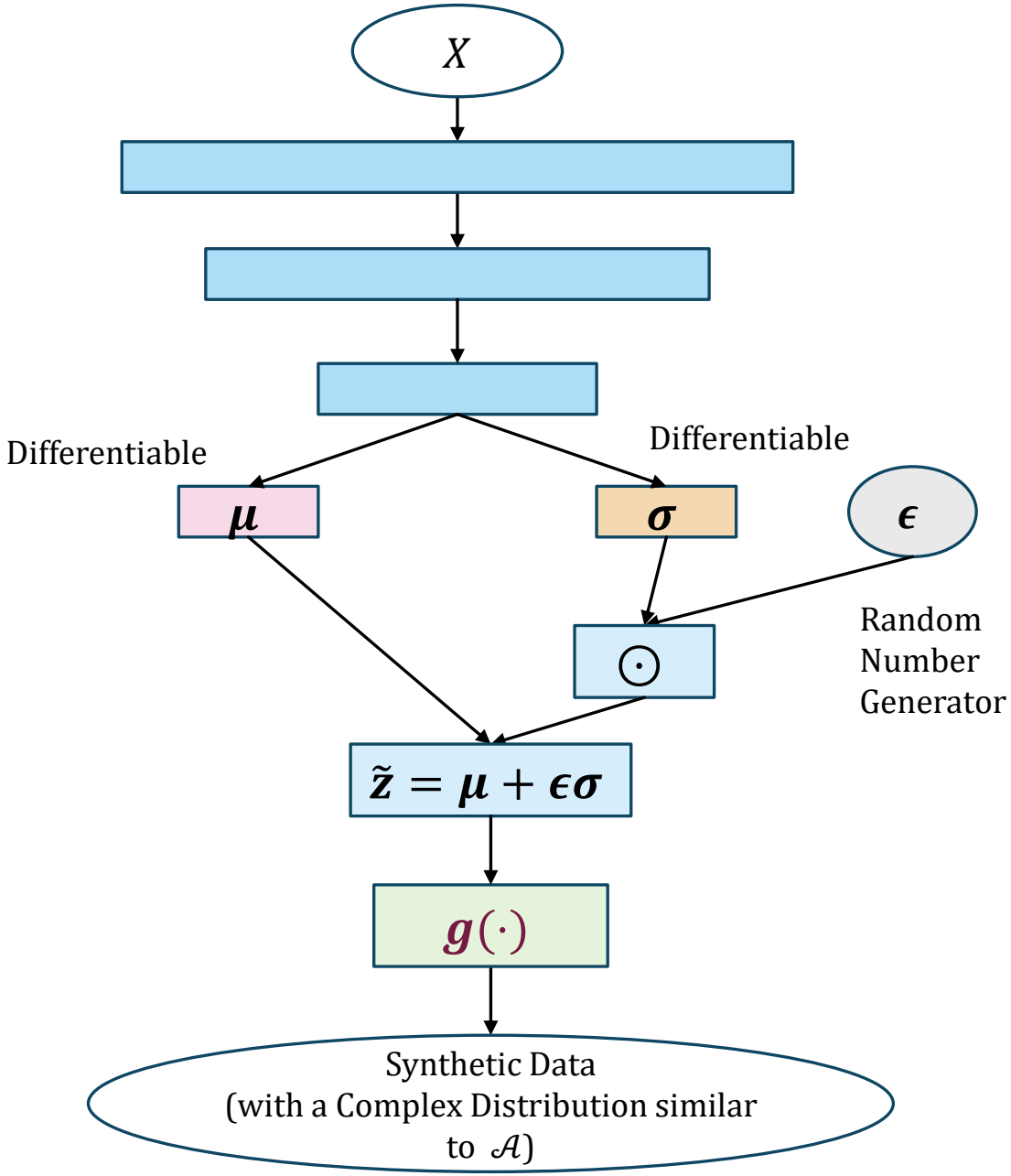
- Step 4: How to ensure that μ and σ and consequently the sample $\tilde{z} = \mu + \epsilon\sigma$ correctly represent a simple transformed distribution of the input data X?



Our First Generative Approach

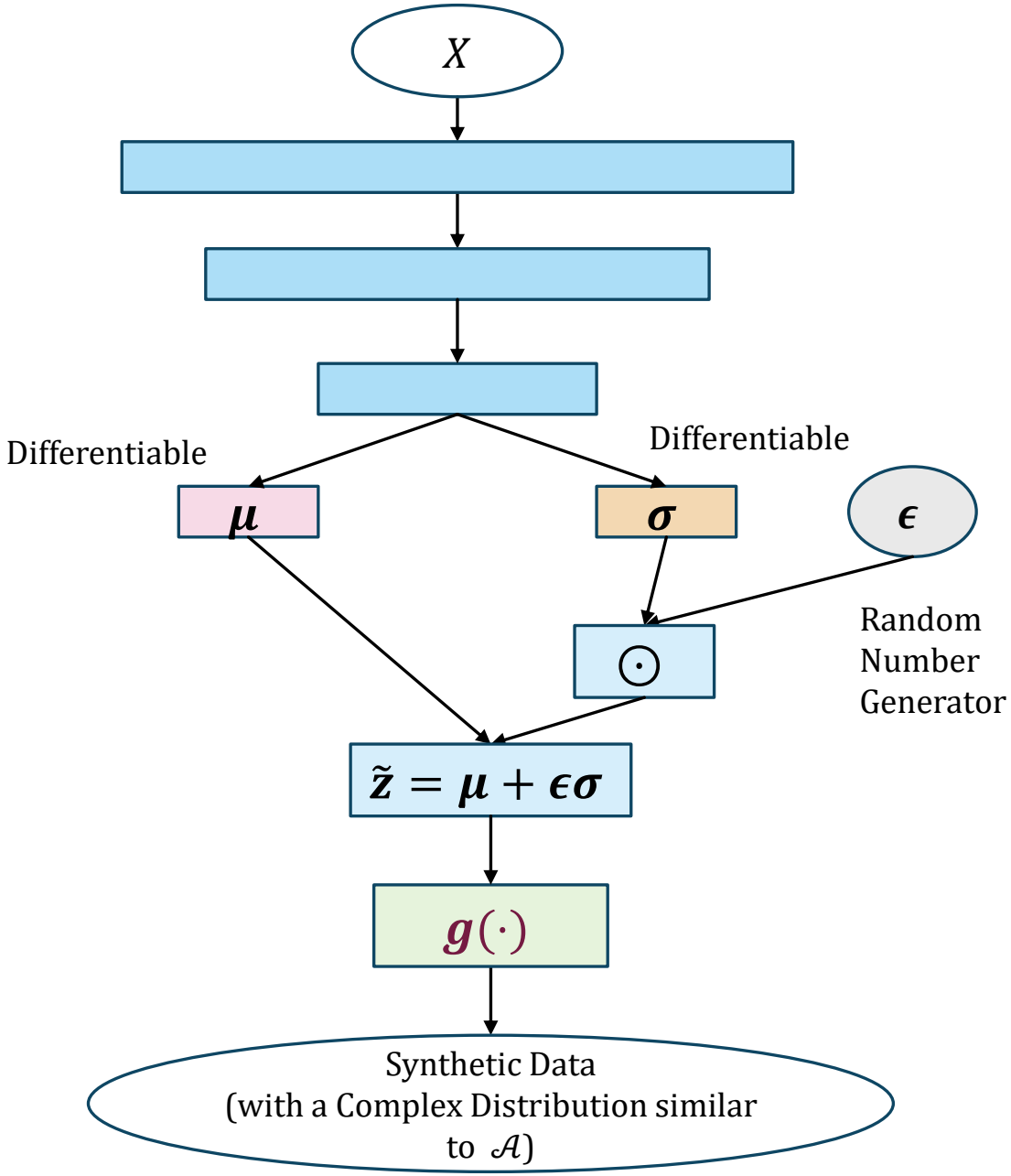
- Step 4: How to ensure that μ and σ and consequently the sample $\tilde{z} = \mu + \epsilon\sigma$ correctly represent a simple transformed distribution of the input data X?

- Design $g(\cdot)$ such that it reconstructs the input data X from $\tilde{z}$
  - Let $\hat{X} = g(\tilde{z})$
  - We want to minimize $\|\hat{X} - X\|$



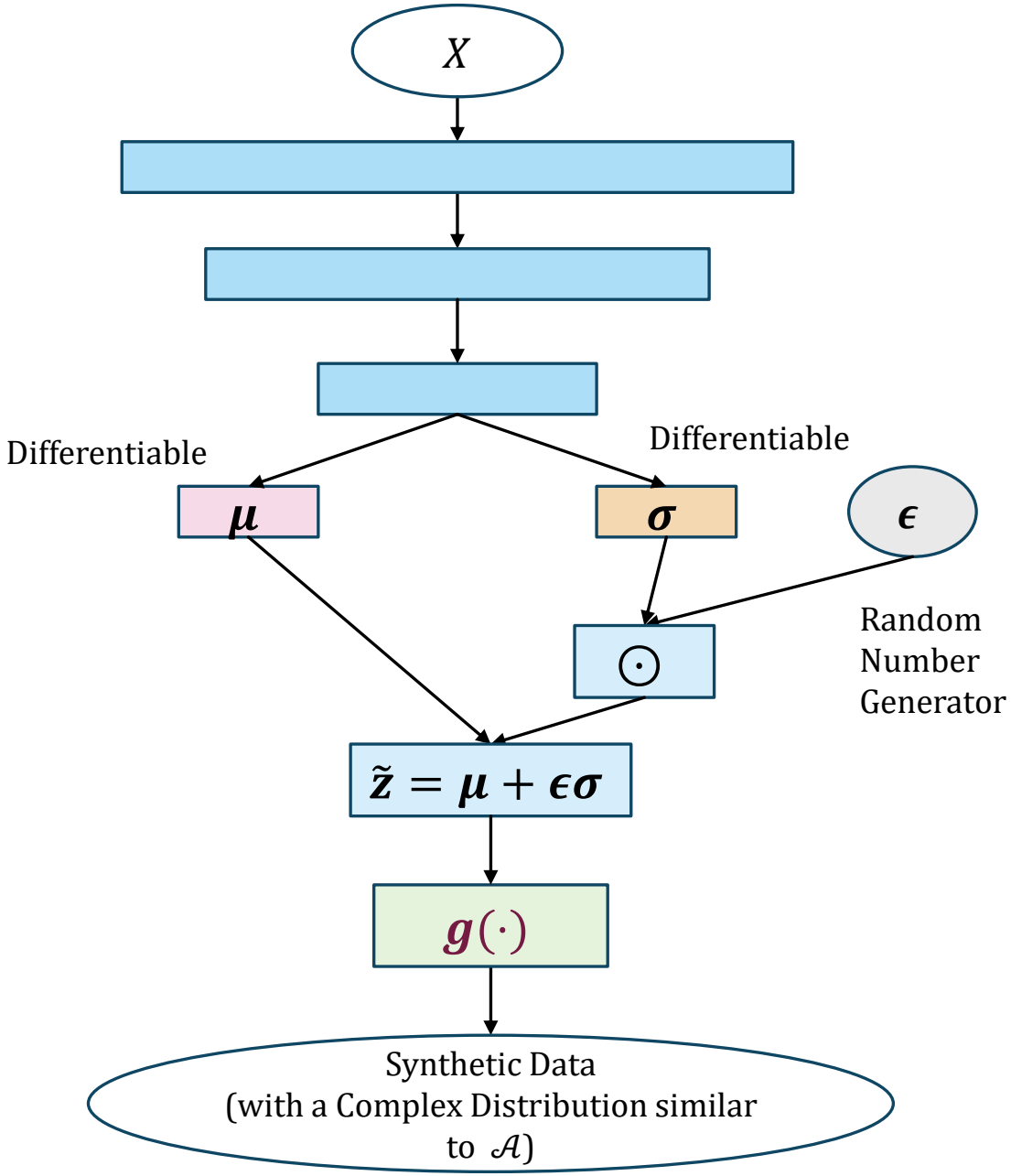
Our First Generative Approach

- Design $g(\cdot)$ such that it reconstructs the input data X from $\tilde{z}$
  - Let $\hat{X} = g(\tilde{z})$
  - We want to minimize $\|\hat{X} - X\|$
- Autoencoder objective function



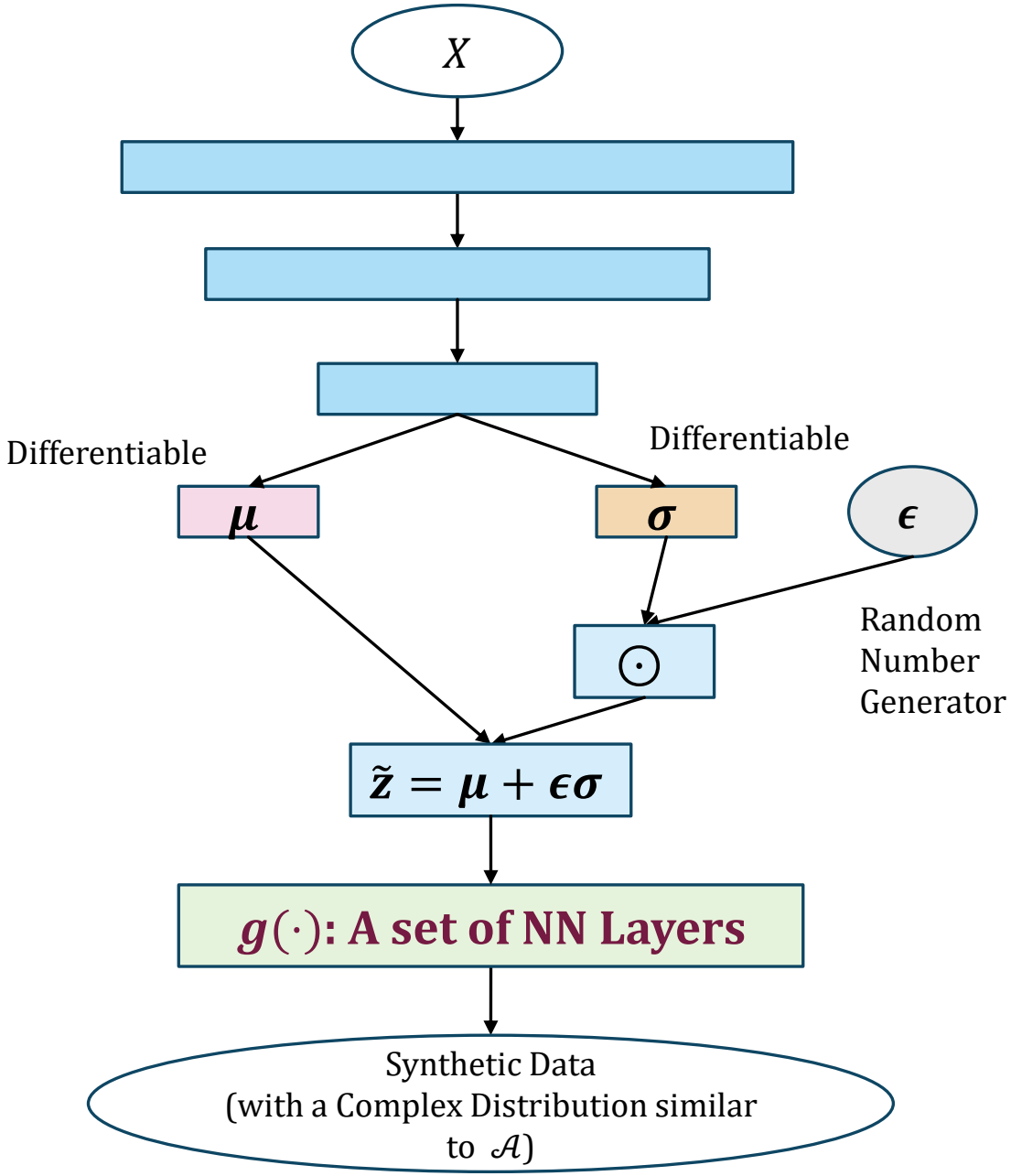
Our First Generative Approach

- Design $g(\cdot)$ such that it reconstructs the input data X from $\tilde{z}$
  - Let $\hat{X} = g(\tilde{z})$
  - We want to minimize $\|\hat{X} - X\|$
- Autoencoder objective function
  - So, $g(\cdot)$ can be a set of NN layers



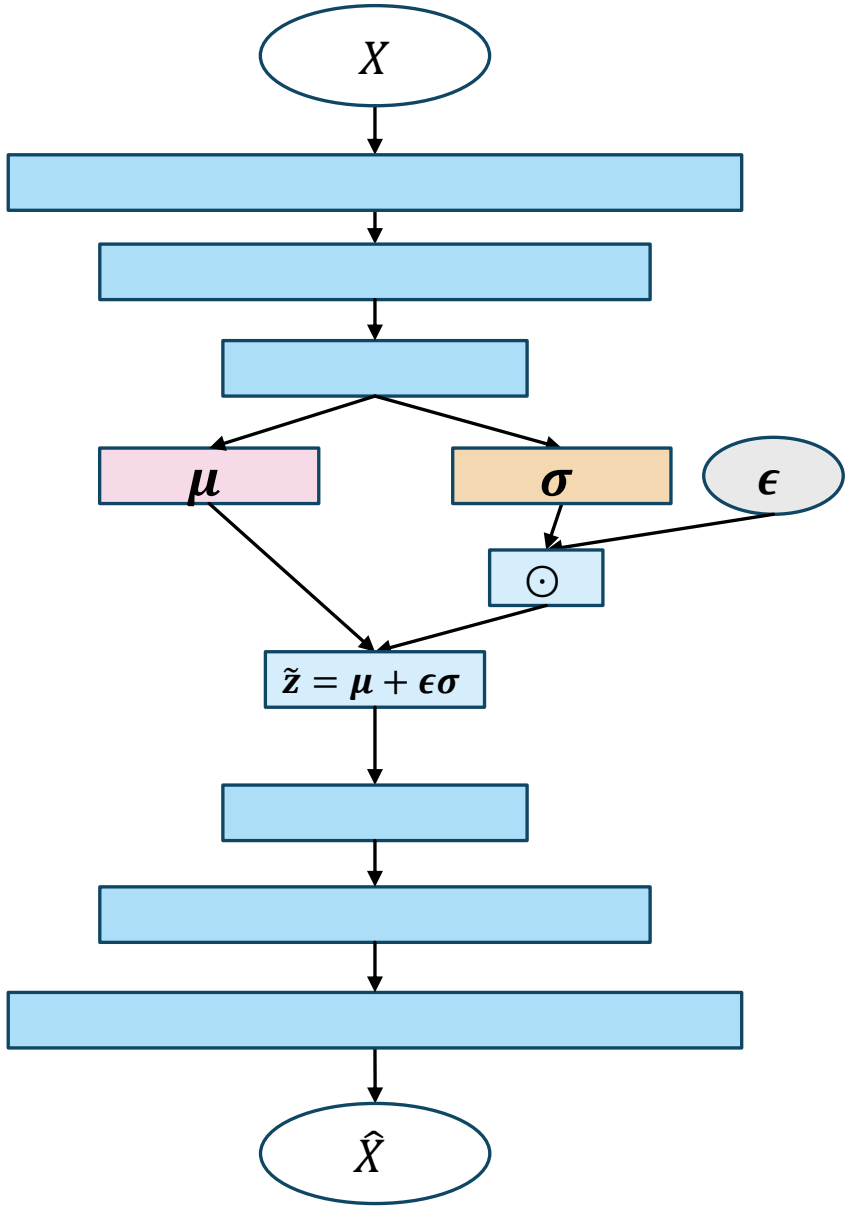
Our First Generative Approach

- Design $g(\cdot)$ such that it reconstructs the input data X from $\tilde{z}$
  - Let $\hat{X} = g(\tilde{z})$
  - We want to minimize $\|\hat{X} - X\|$
- Autoencoder objective function
  - So, $g(\cdot)$ can be a set of NN layers



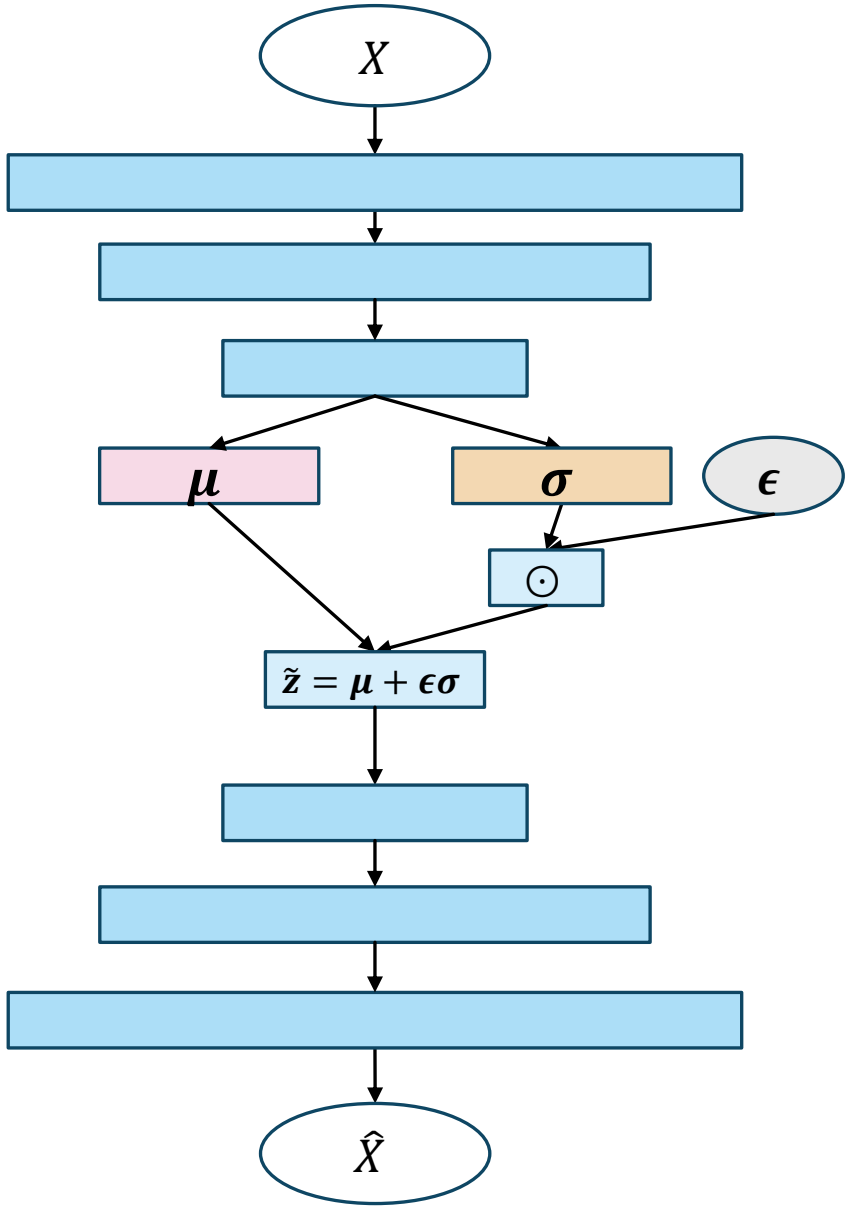
Our First Generative Approach

- Design $g(\cdot)$ such that it reconstructs the input data X from $\tilde{z}$
  - Let $\hat{X} = g(\tilde{z})$
  - We want to minimize $\|\hat{X}\|$
- Autoencoder objective function
  - So, $g(\cdot)$ can be a set of NN layers



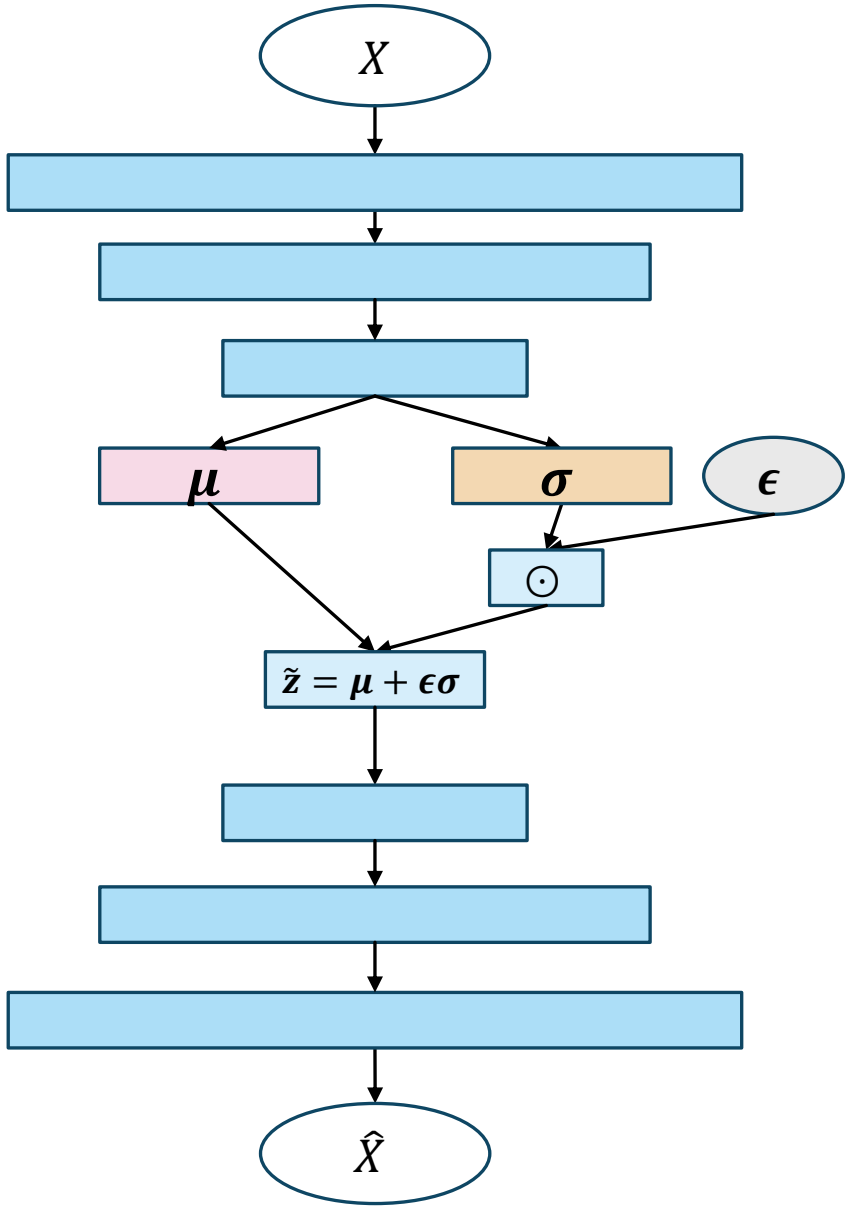
Our First Generative Approach

- An encoder decoder architecture



Variational Autoencoder

- An encoder decoder architecture
- **Variational Autoencoder**
  - Variational inference: a technique to approximate complex distributions

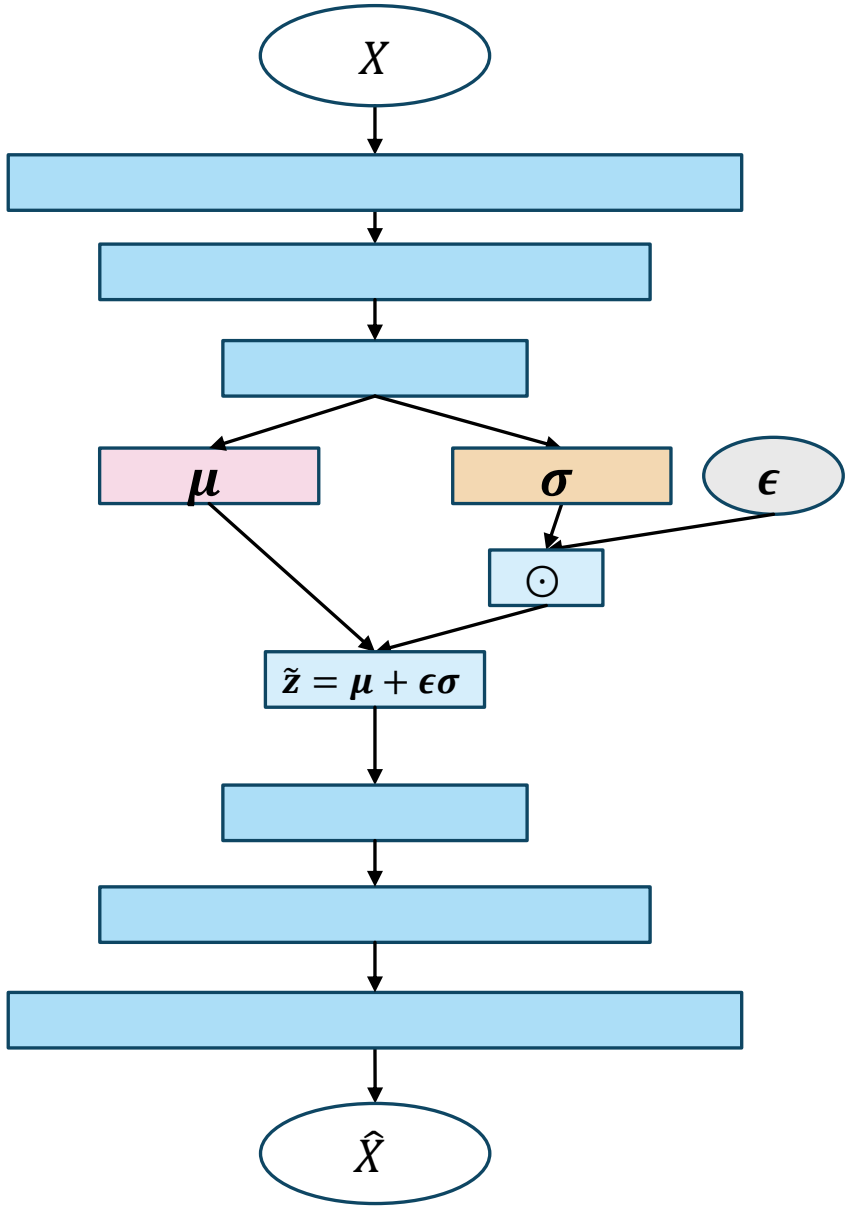


Variational Autoencoder: Loss Function

- Reconstruction loss: $MSE(X, \hat{X})$

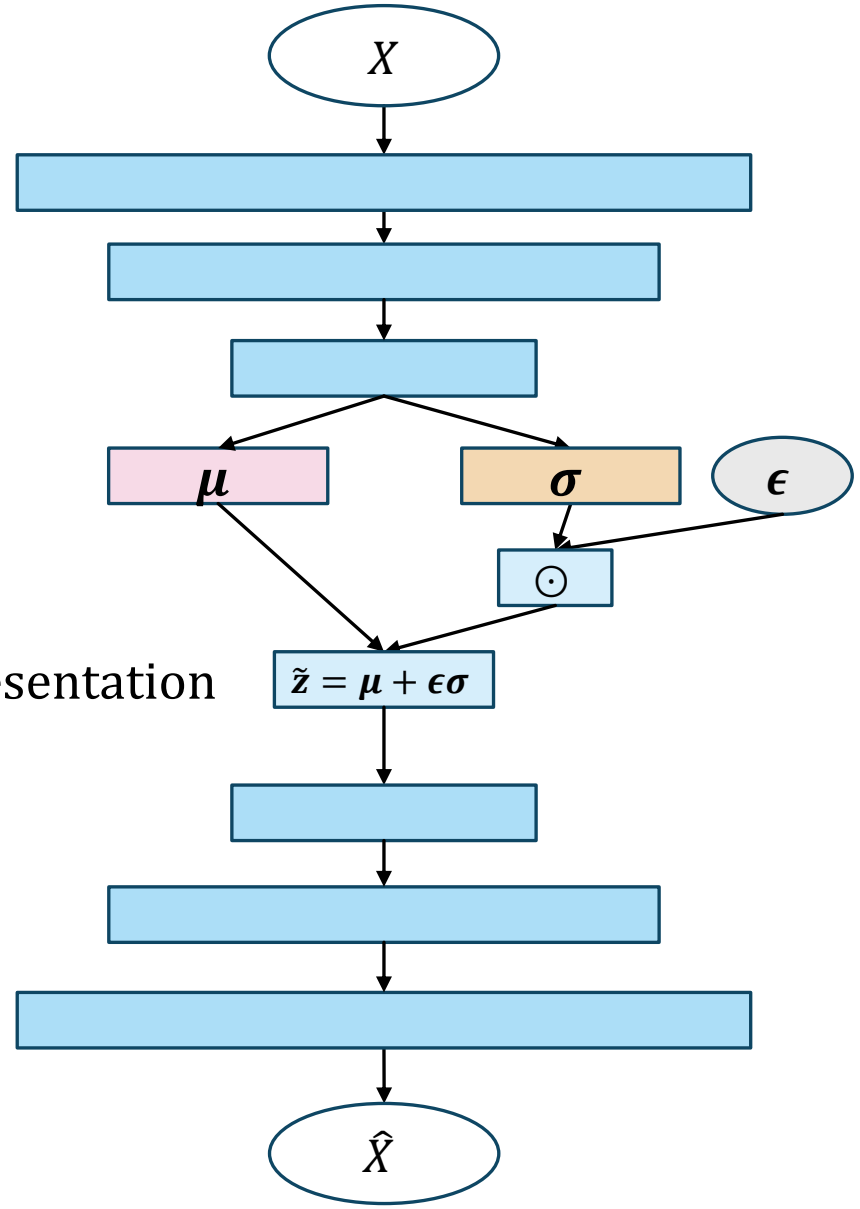
- KL divergence loss

$$KL(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \ln \sigma_i^2)$$

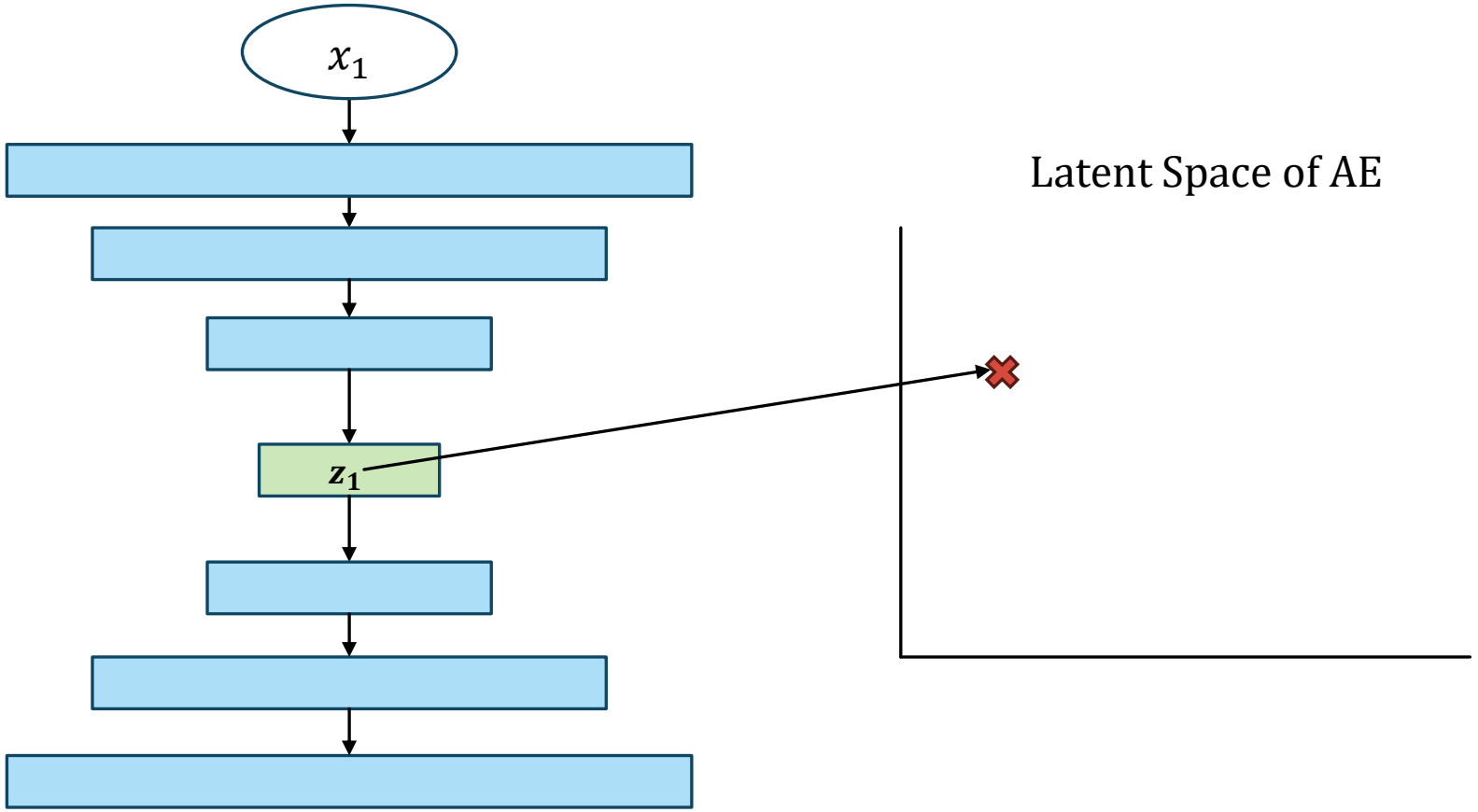


Variational Autoencoder: Latent Space

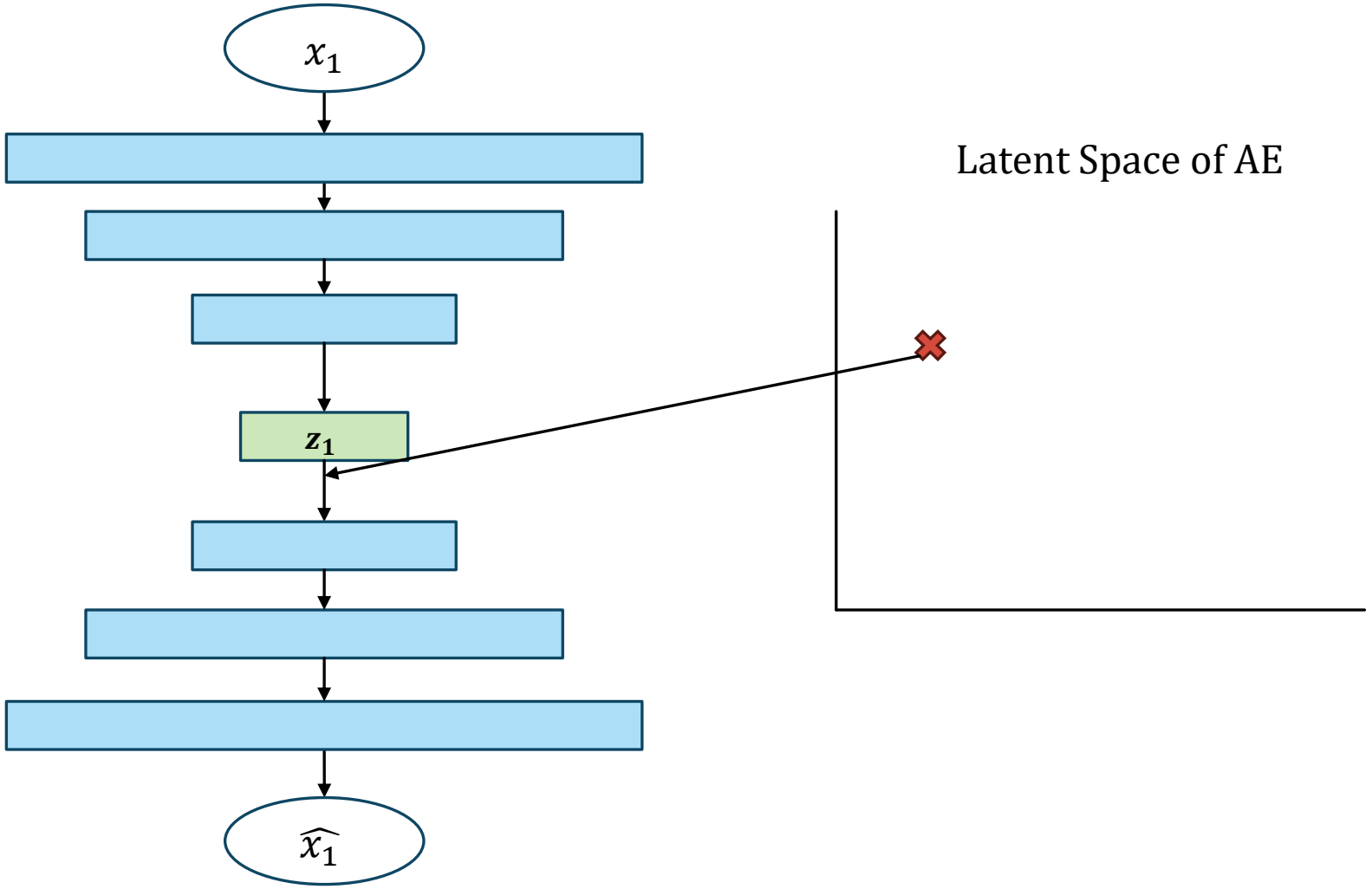
Latent space representation



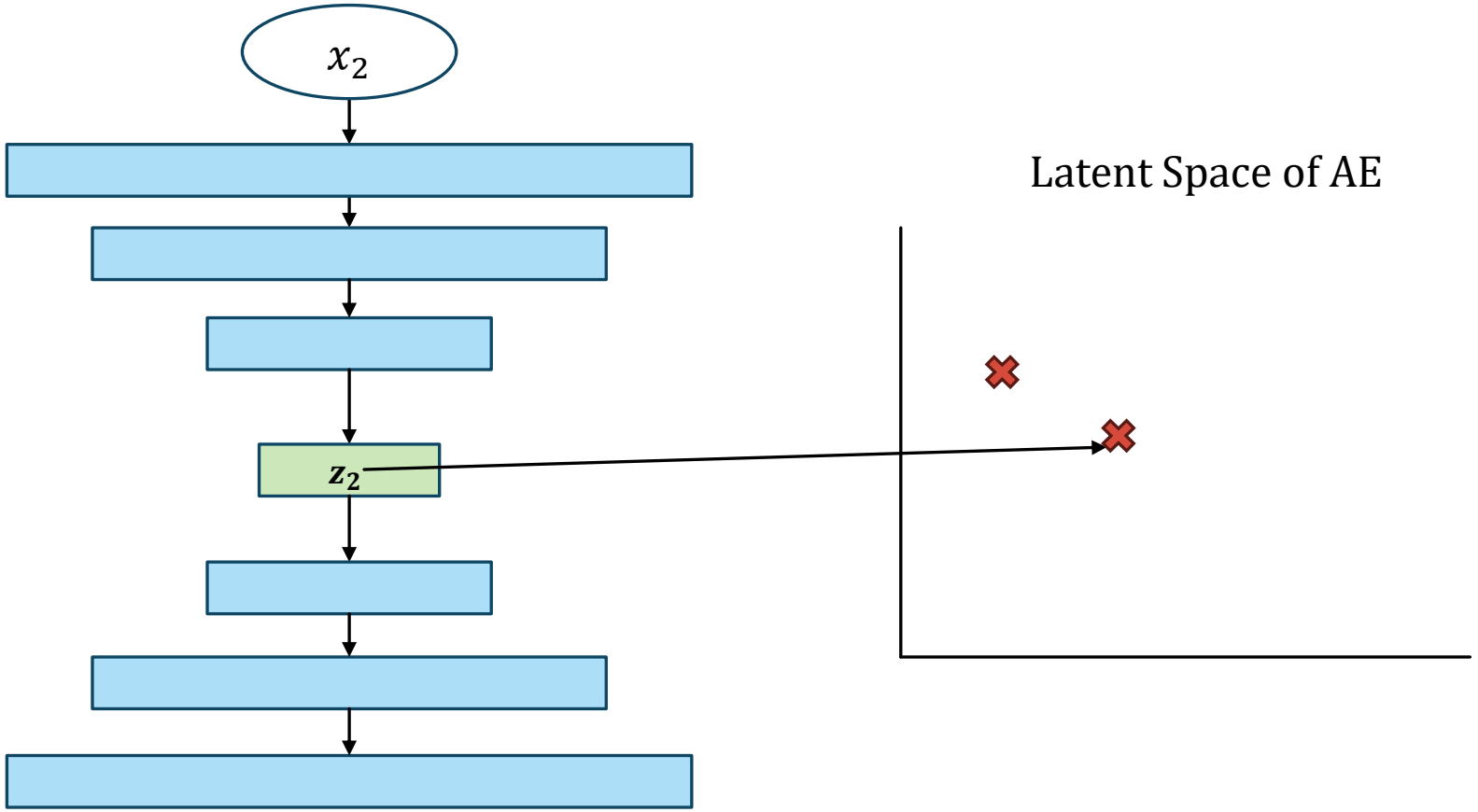
Latent Spaces: AE vs VAE



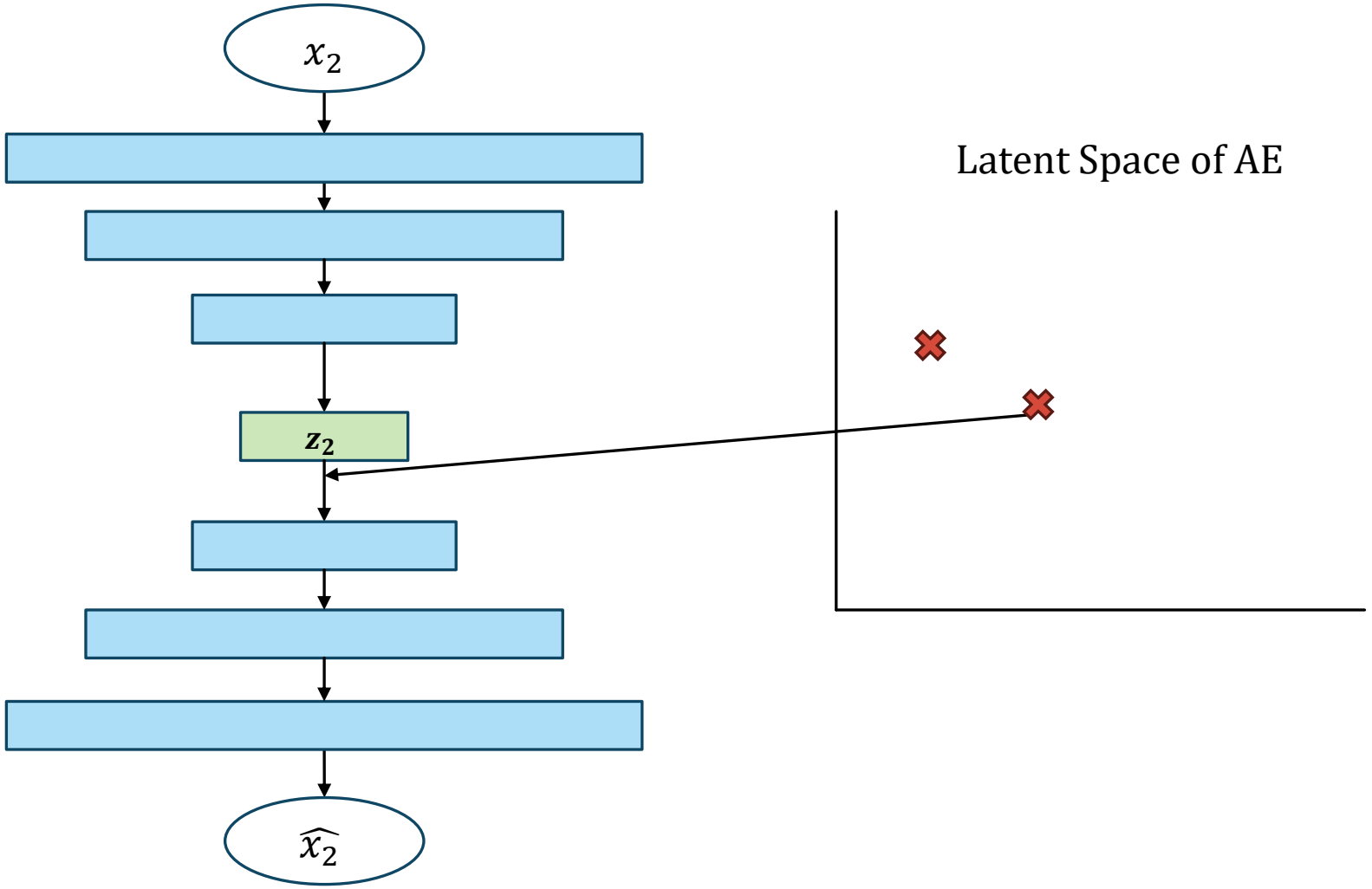
Latent Spaces: AE vs VAE



Latent Spaces: AE vs VAE

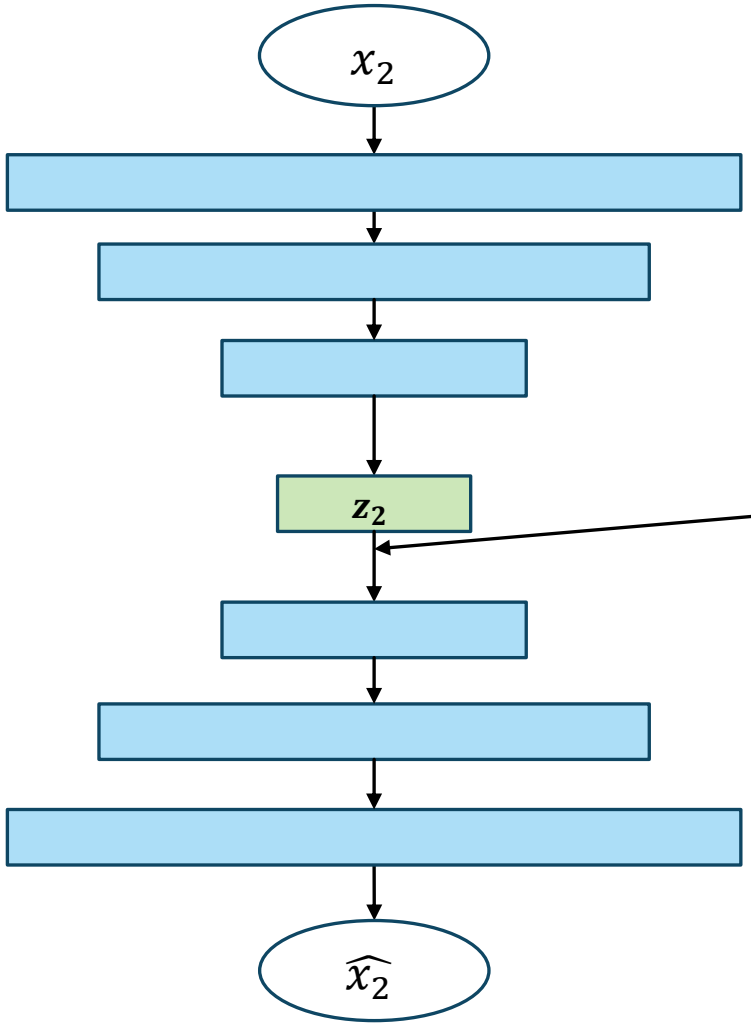


Latent Spaces: AE vs VAE

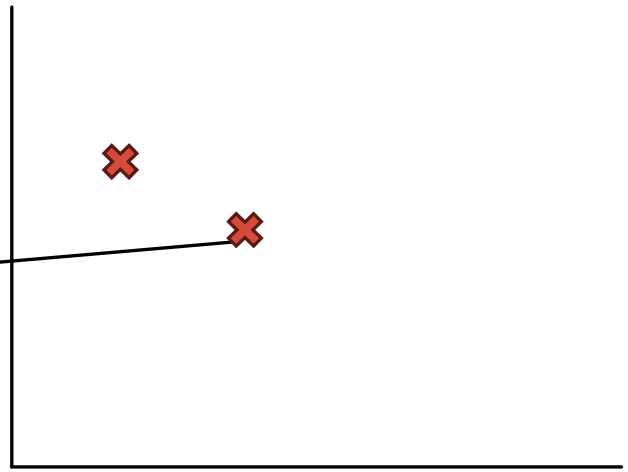


Latent Space of AE

Latent Spaces: AE vs VAE

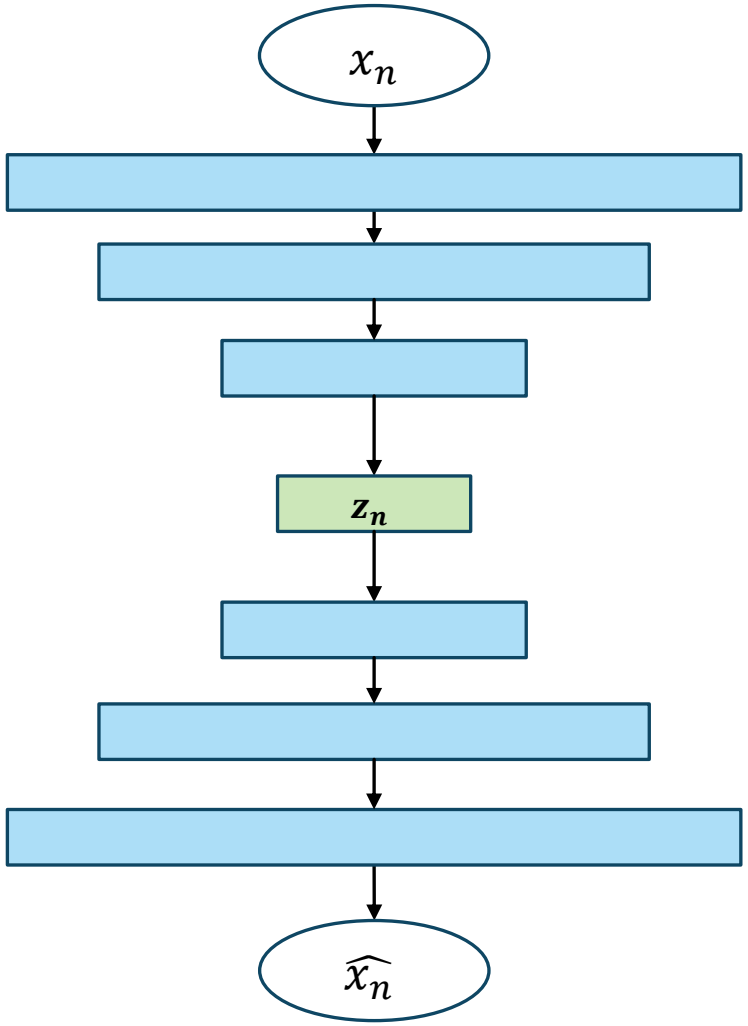


Latent Space of AE

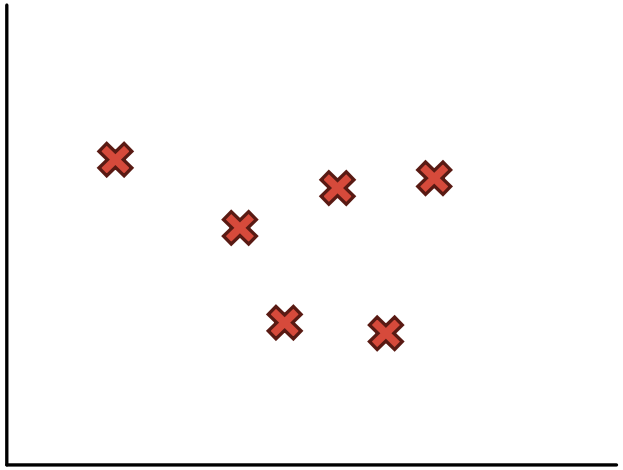


And so on …

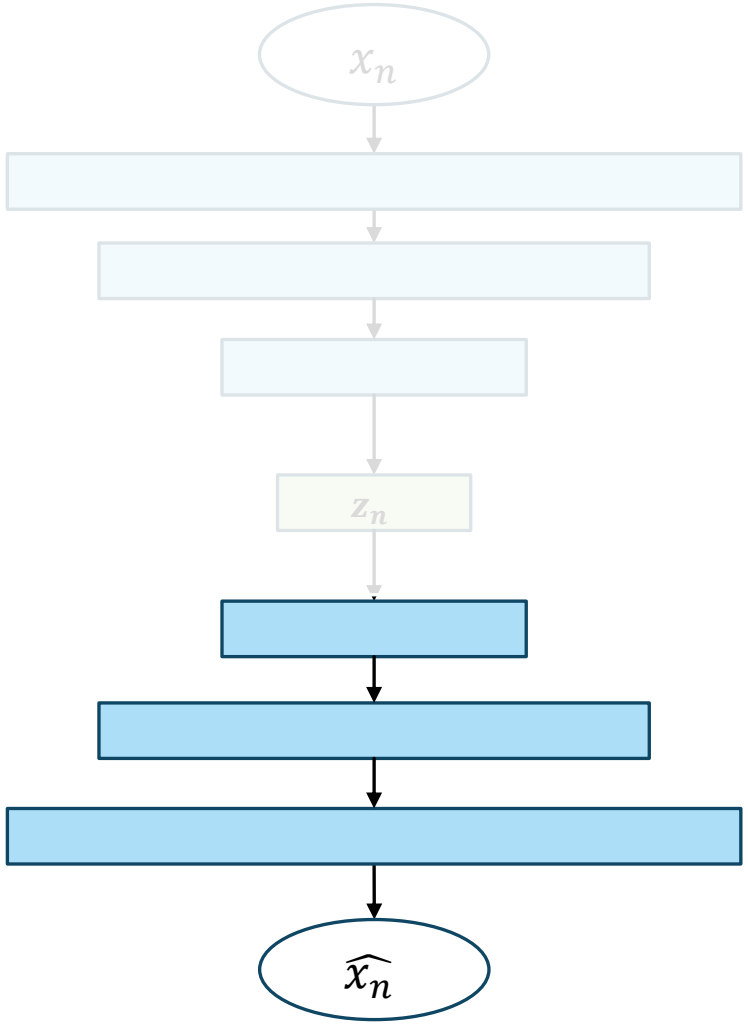
Latent Spaces: AE vs VAE



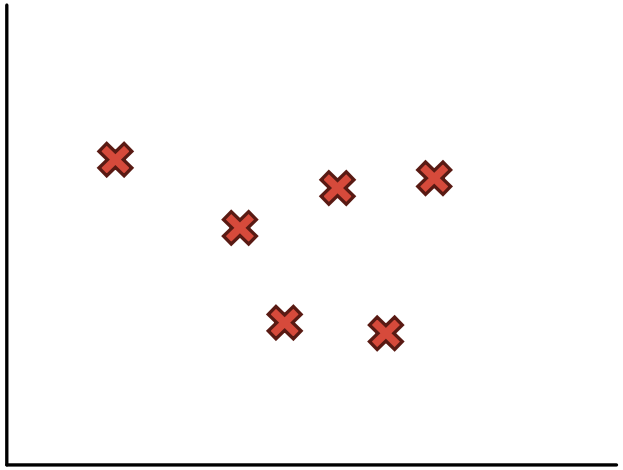
Latent Space of AE



Latent Spaces: AE vs VAE

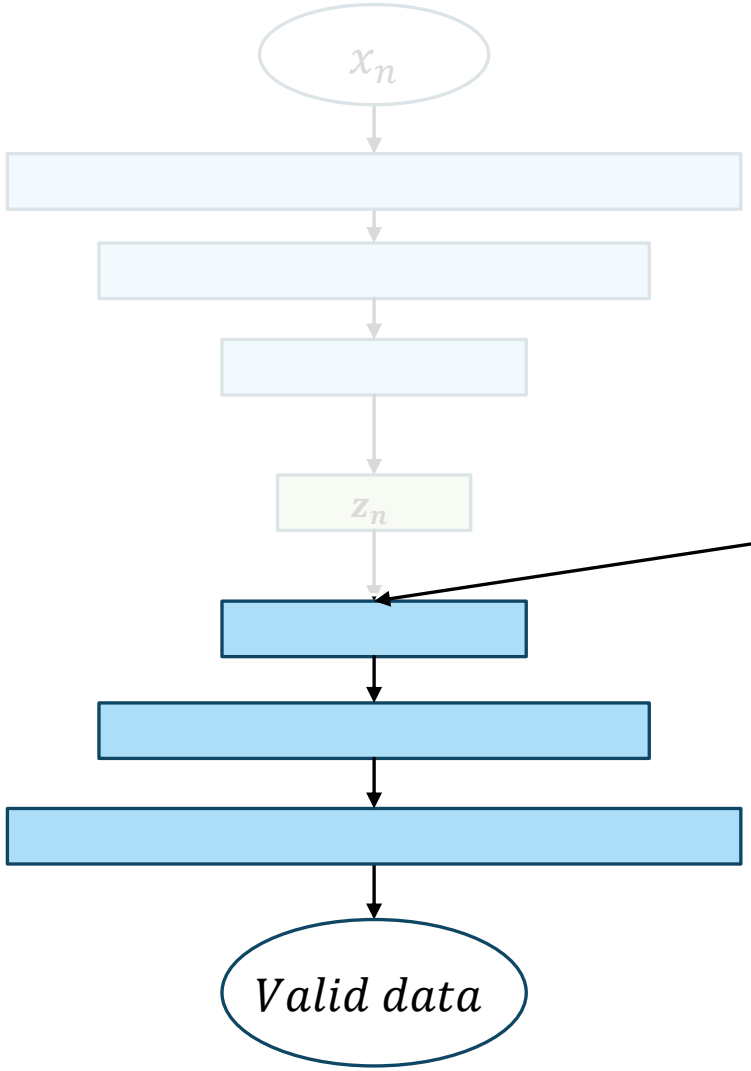


Latent Space of AE

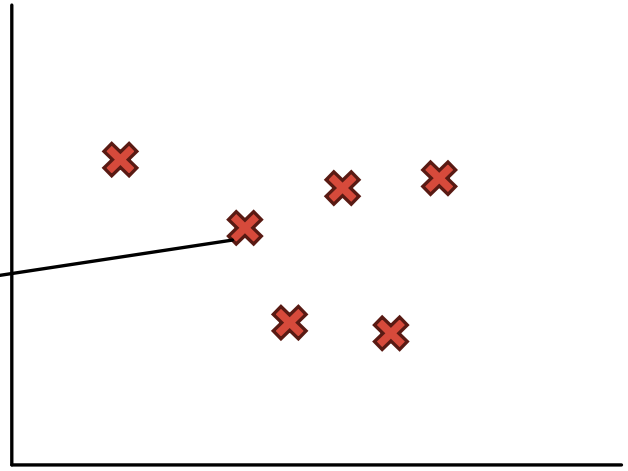


Now consider only the decoder after training

Latent Spaces: AE vs VAE



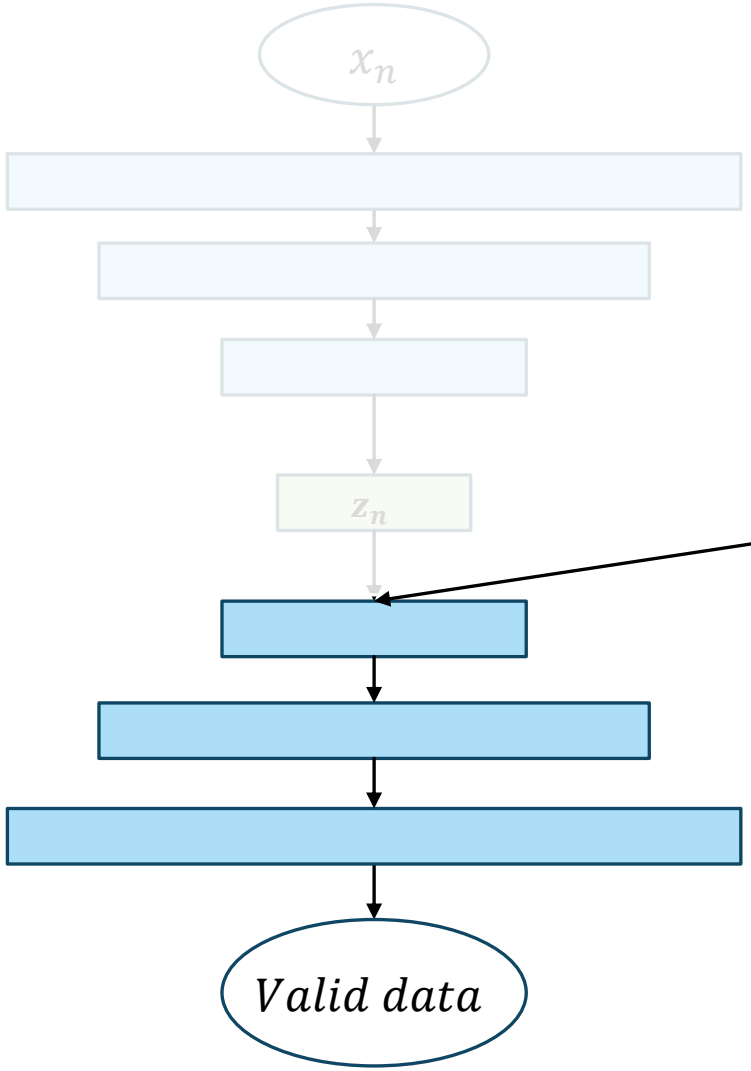
Latent Space of AE



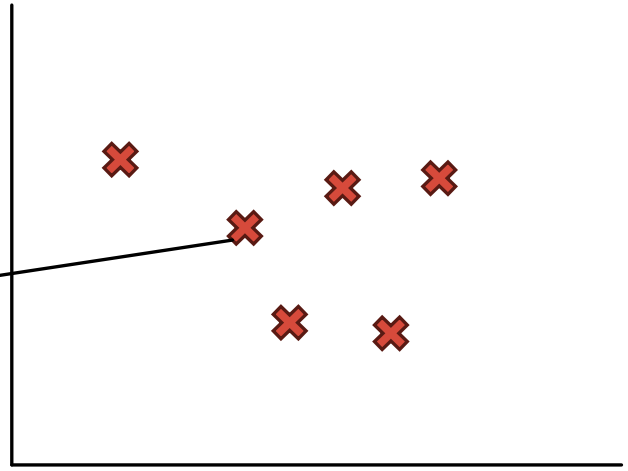
If I apply this latent space representation to the trained decoder, I will get a valid data at the output

Why?

Latent Spaces: AE vs VAE



Latent Space of AE

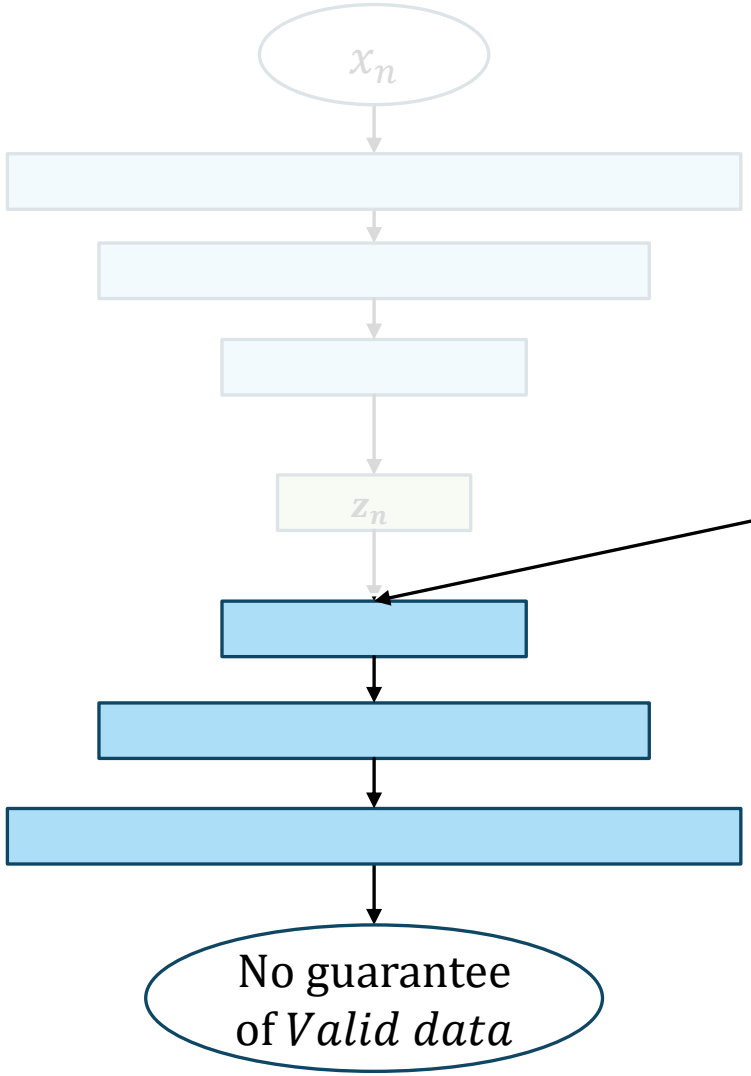


If I apply this latent space representation to the trained decoder, I will get a valid data at the output

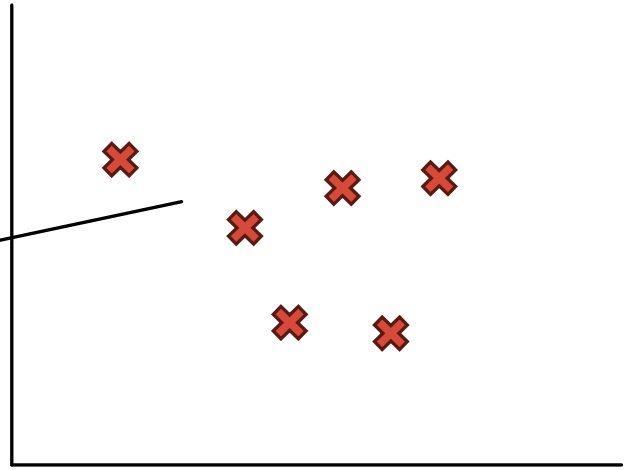
Why?

Because, at the time of training the AE learnt to reconstruct this latent space representation

Latent Spaces: AE vs VAE



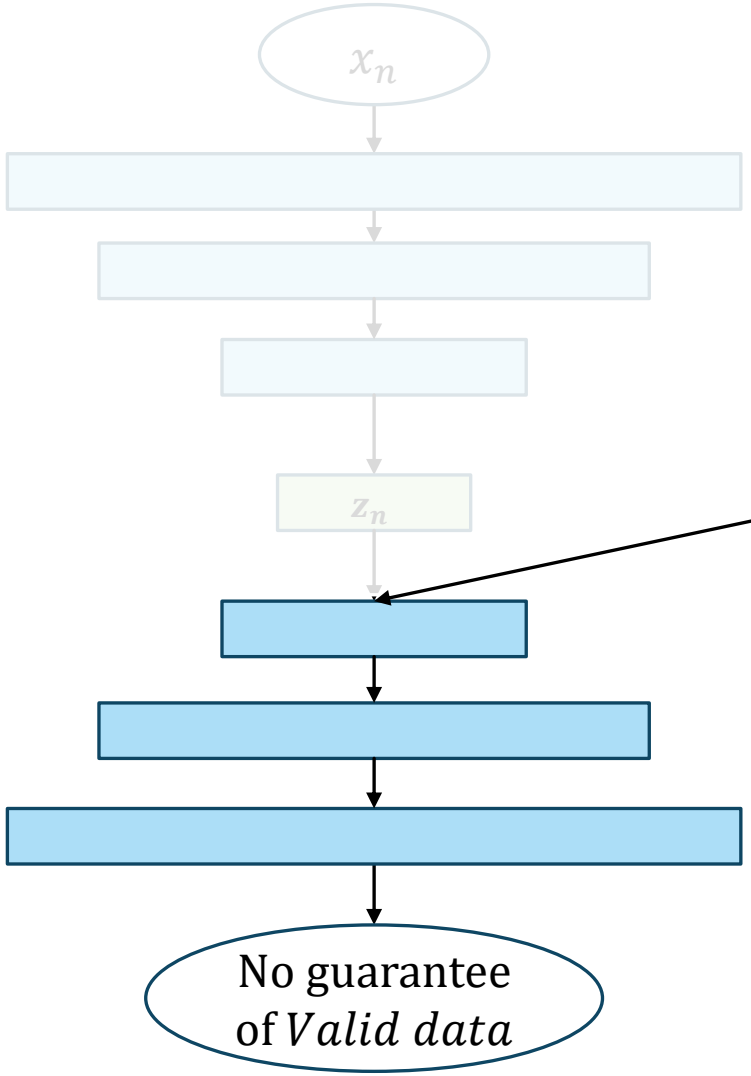
Latent Space of AE



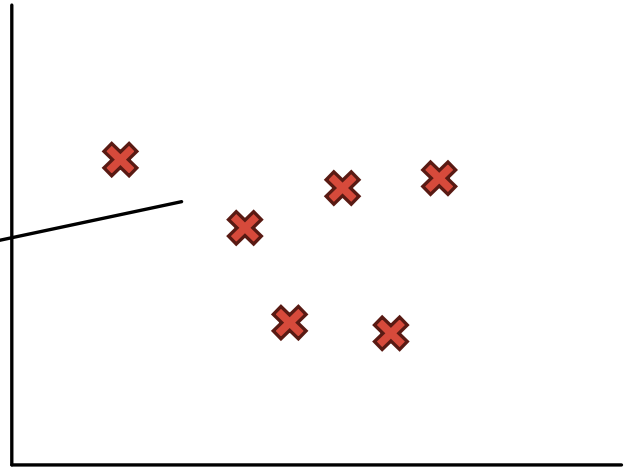
However, If I apply this point from the latent space representation to the trained decoder, I am not guaranteed to get a valid data at the output

Why?

Latent Spaces: AE vs VAE



Latent Space of AE

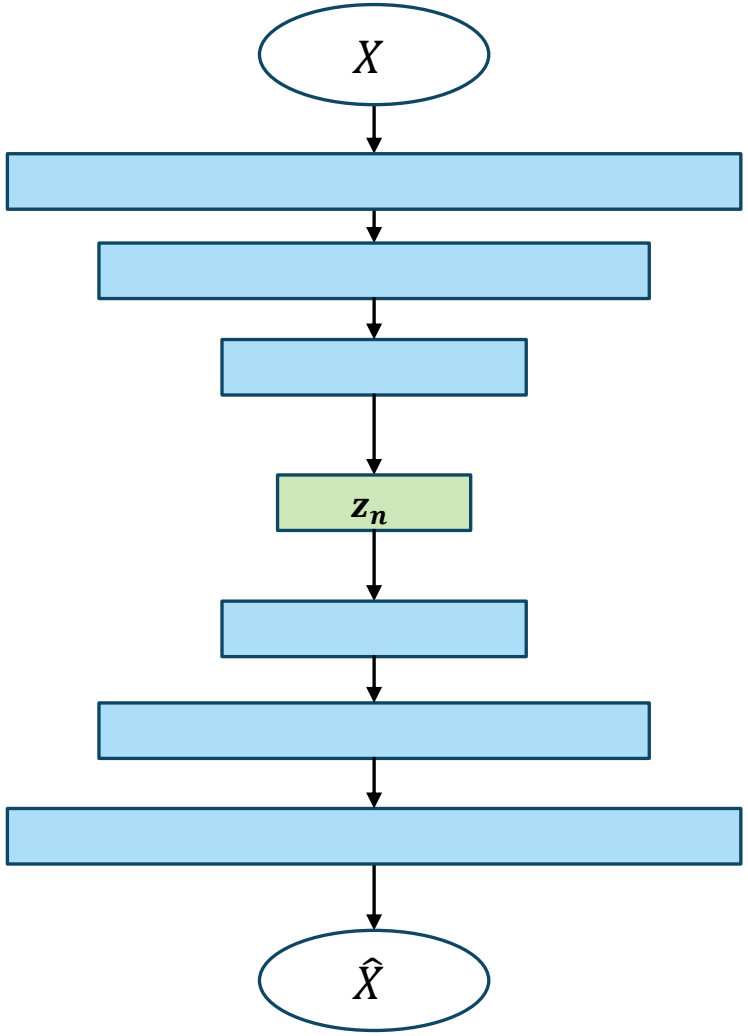


However, If I apply this point from the latent space representation to the trained decoder, I am not guaranteed to get a valid data at the output

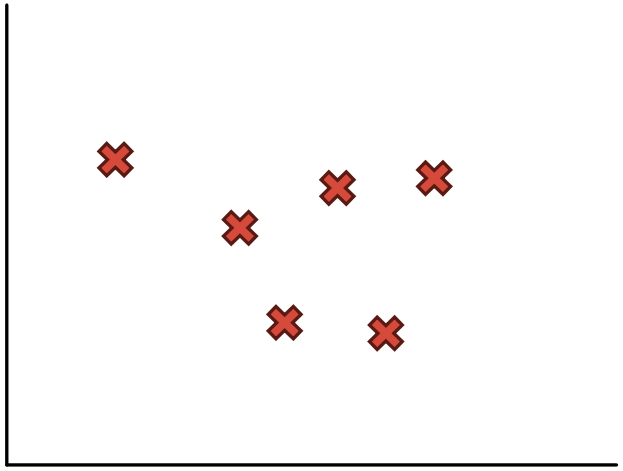
Why?

Because, the AE did not learn to reconstruct this point at the time of training

Latent Spaces: AE vs VAE

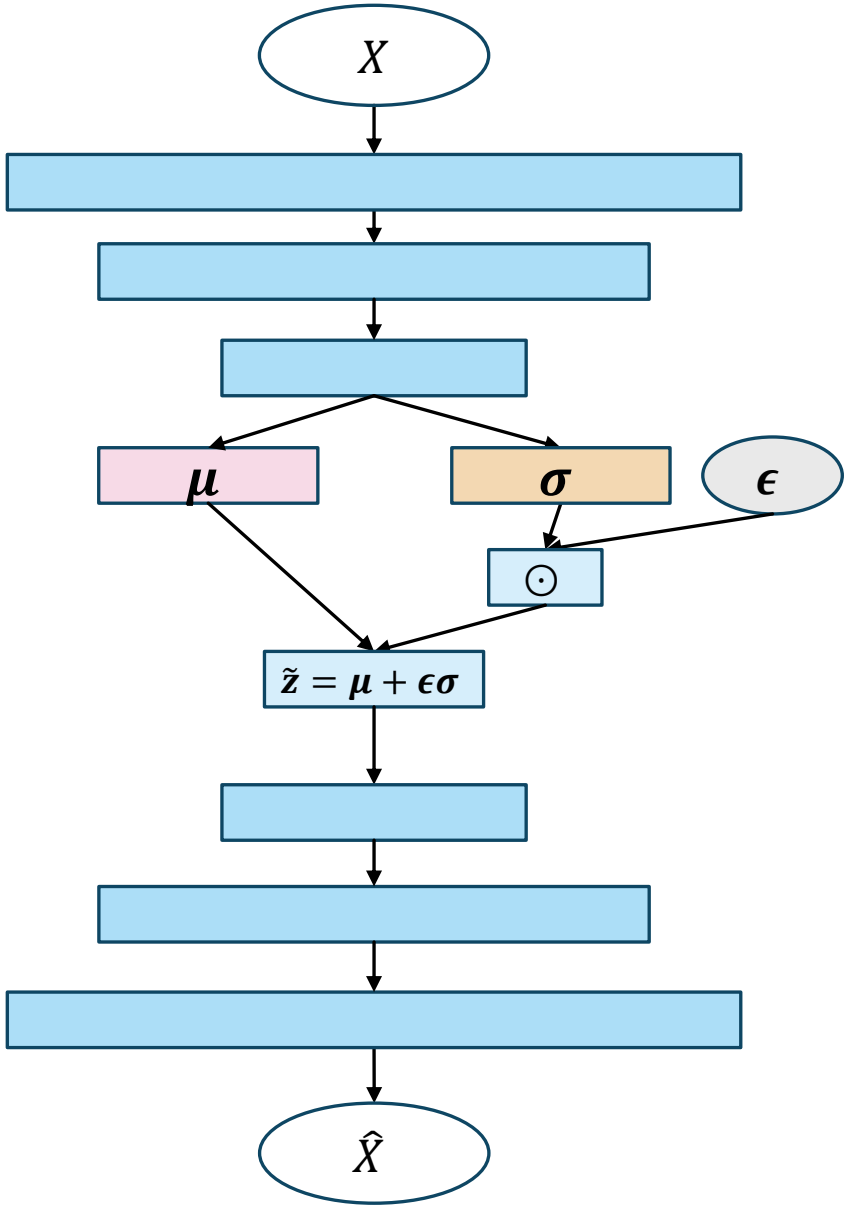


Latent Space of AE

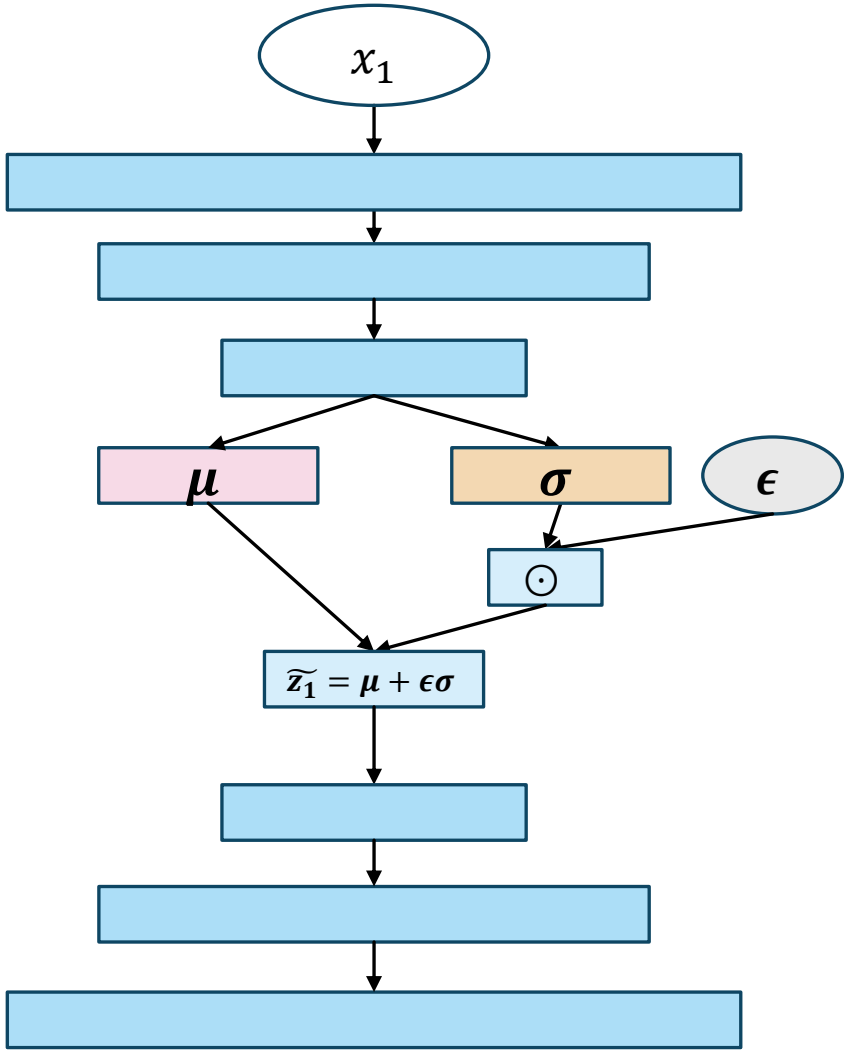


The latent space of the AE is discrete

Latent Spaces: AE vs VAE

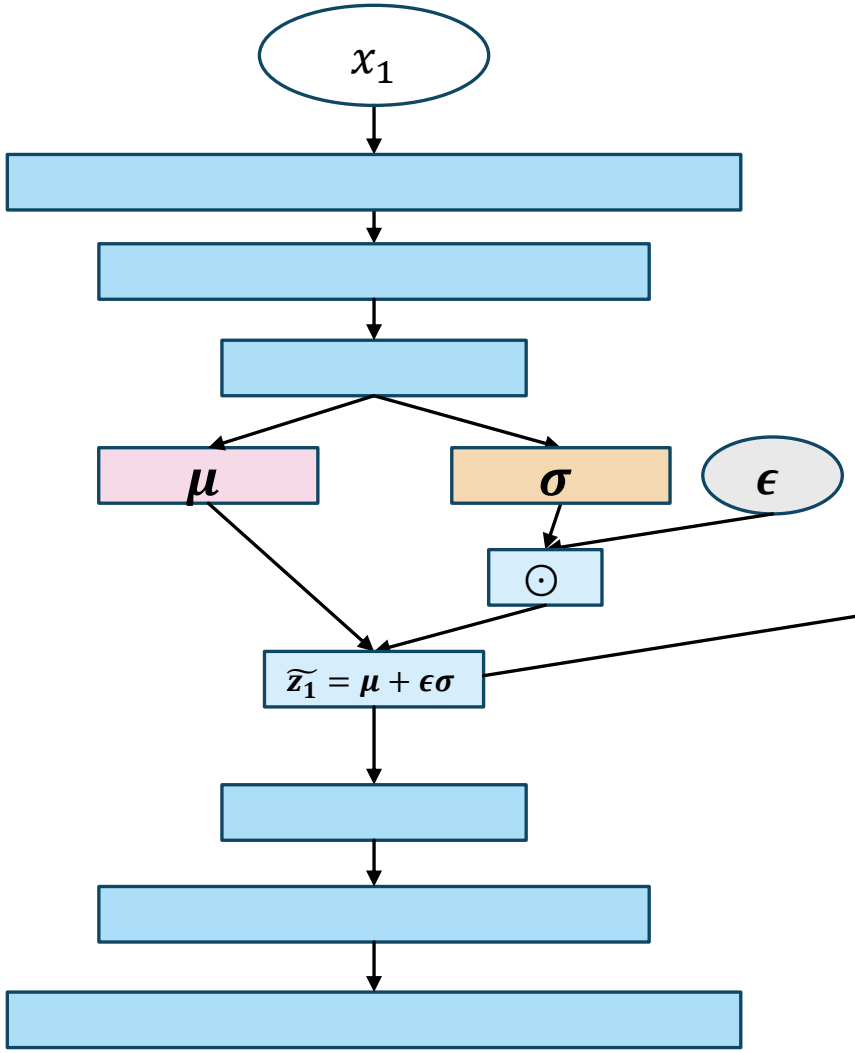


Latent Spaces: AE vs VAE

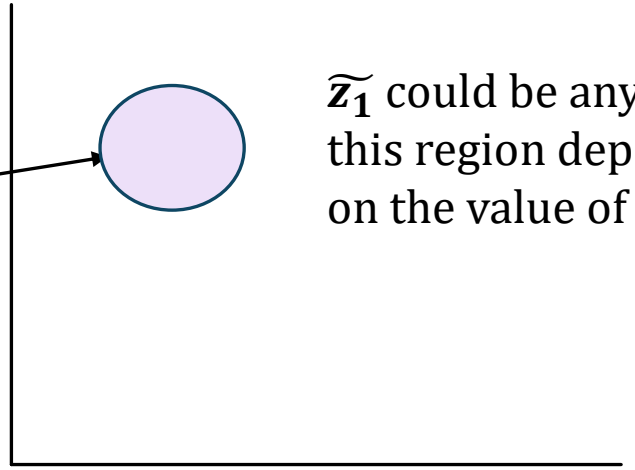


Latent Space of VAE

Latent Spaces: AE vs VAE

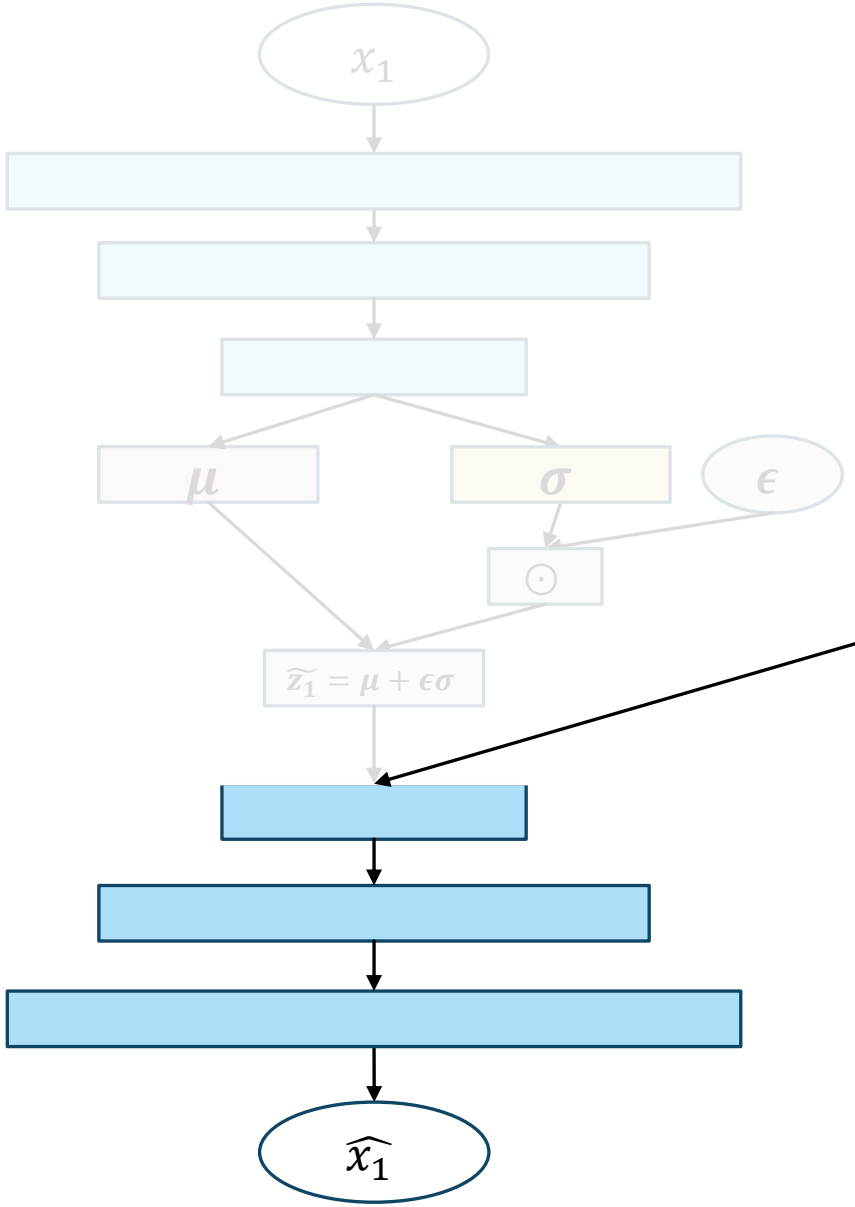


Latent Space of VAE

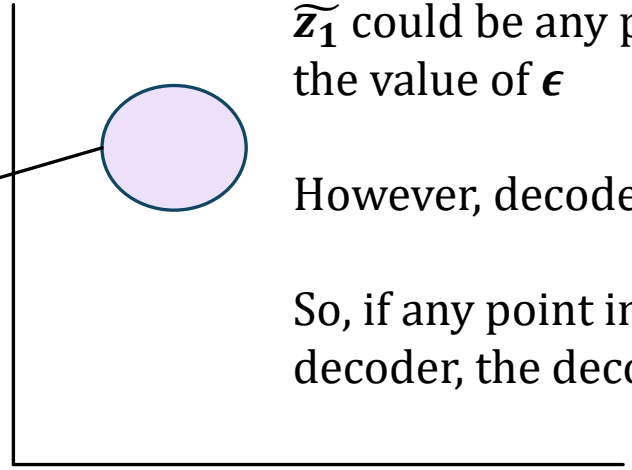


$\widetilde{z_1}$ could be any point in this region depending on the value of ϵ

Latent Spaces: AE vs VAE



Latent Space of VAE

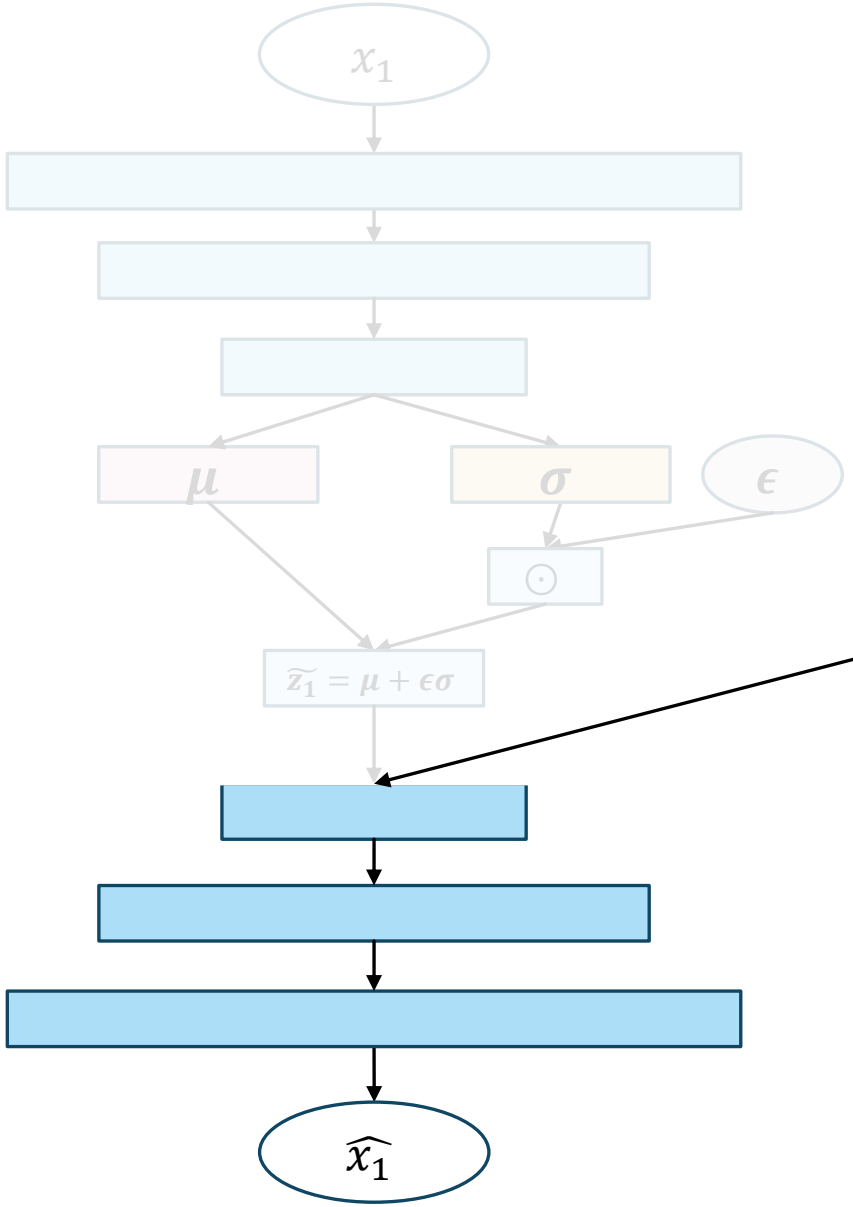


$\widetilde{z_1}$ could be any point in this region depending on the value of ϵ

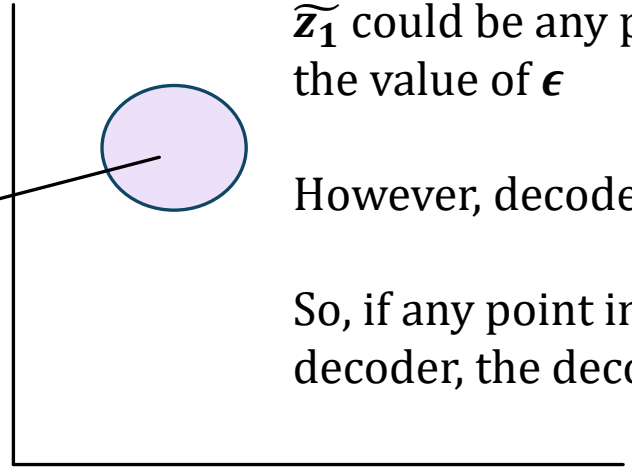
However, decoder output must be $\widehat{x_1}$

So, if any point in this region is applied to the decoder, the decoder should produce $\widehat{x_1}$ as output

Latent Spaces: AE vs VAE



Latent Space of VAE

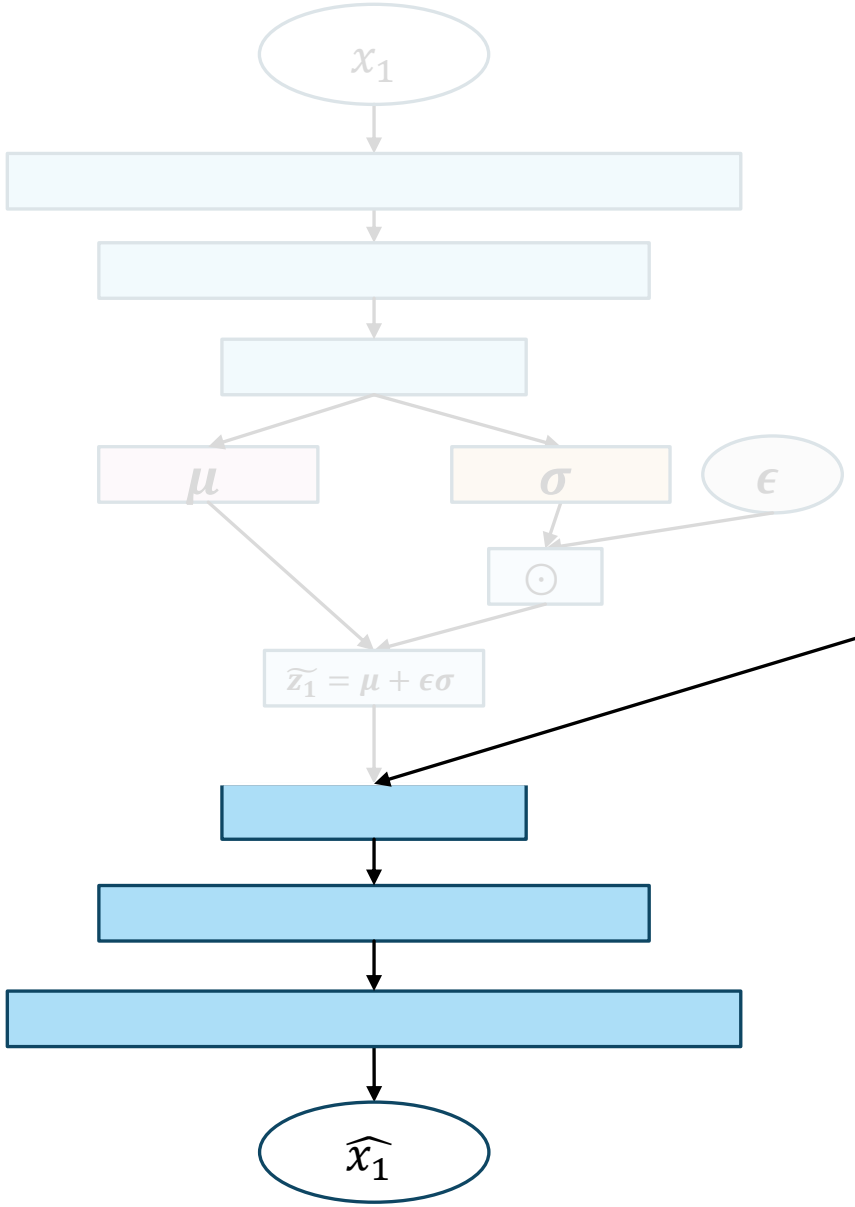


$\widetilde{z_1}$ could be any point in this region depending on the value of ϵ

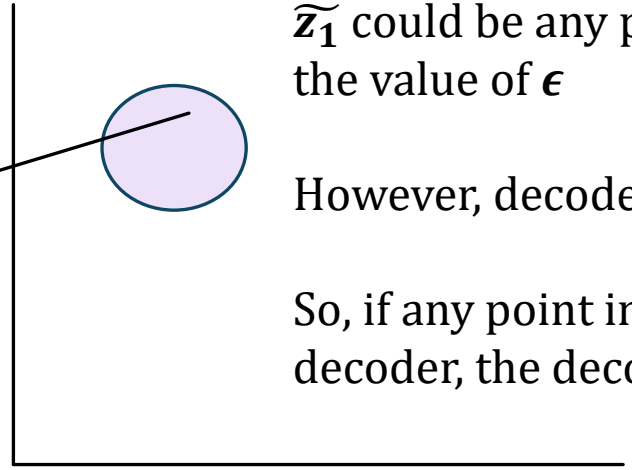
However, decoder output must be $\widehat{x_1}$

So, if any point in this region is applied to the decoder, the decoder should produce $\widehat{x_1}$ as output

Latent Spaces: AE vs VAE



Latent Space of VAE

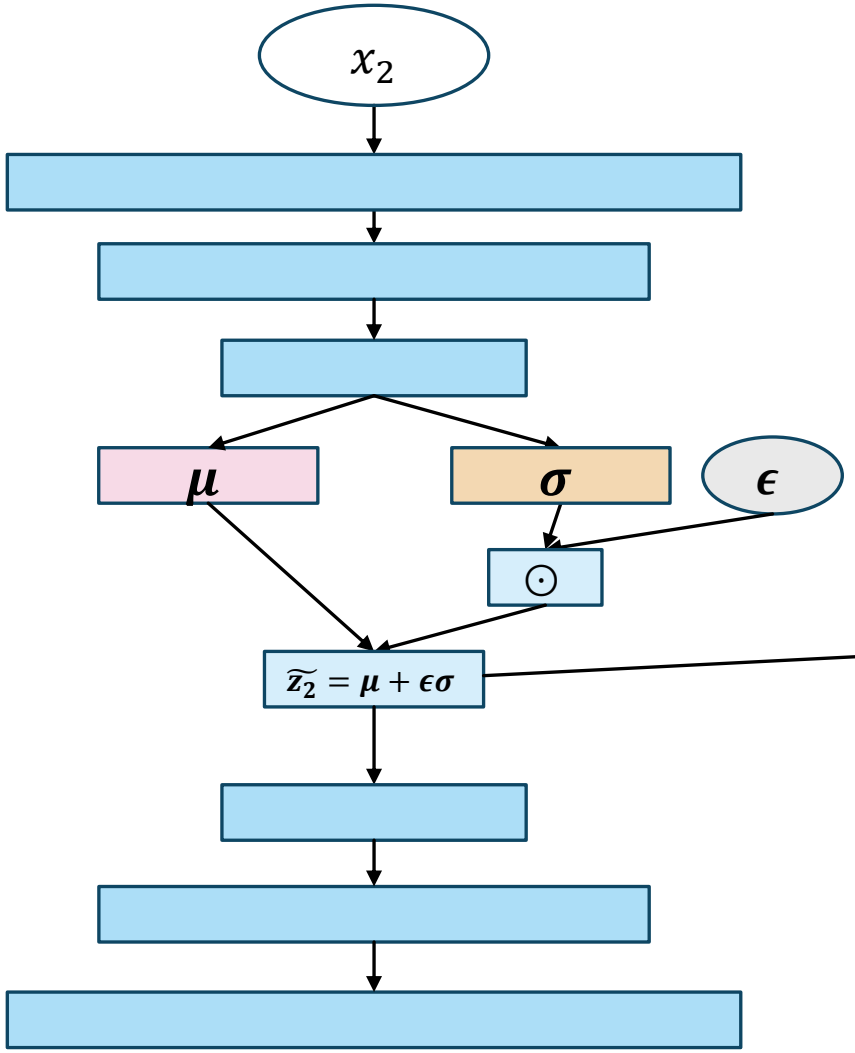


$\widetilde{z_1}$ could be any point in this region depending on the value of ϵ

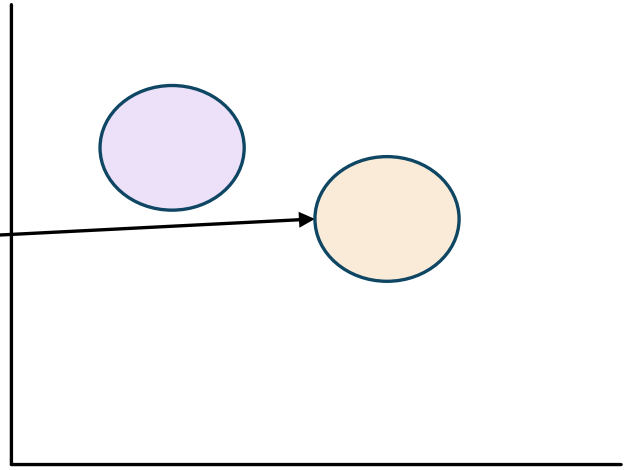
However, decoder output must be $\widehat{x_1}$

So, if any point in this region is applied to the decoder, the decoder should produce $\widehat{x_1}$ as output

Latent Spaces: AE vs VAE

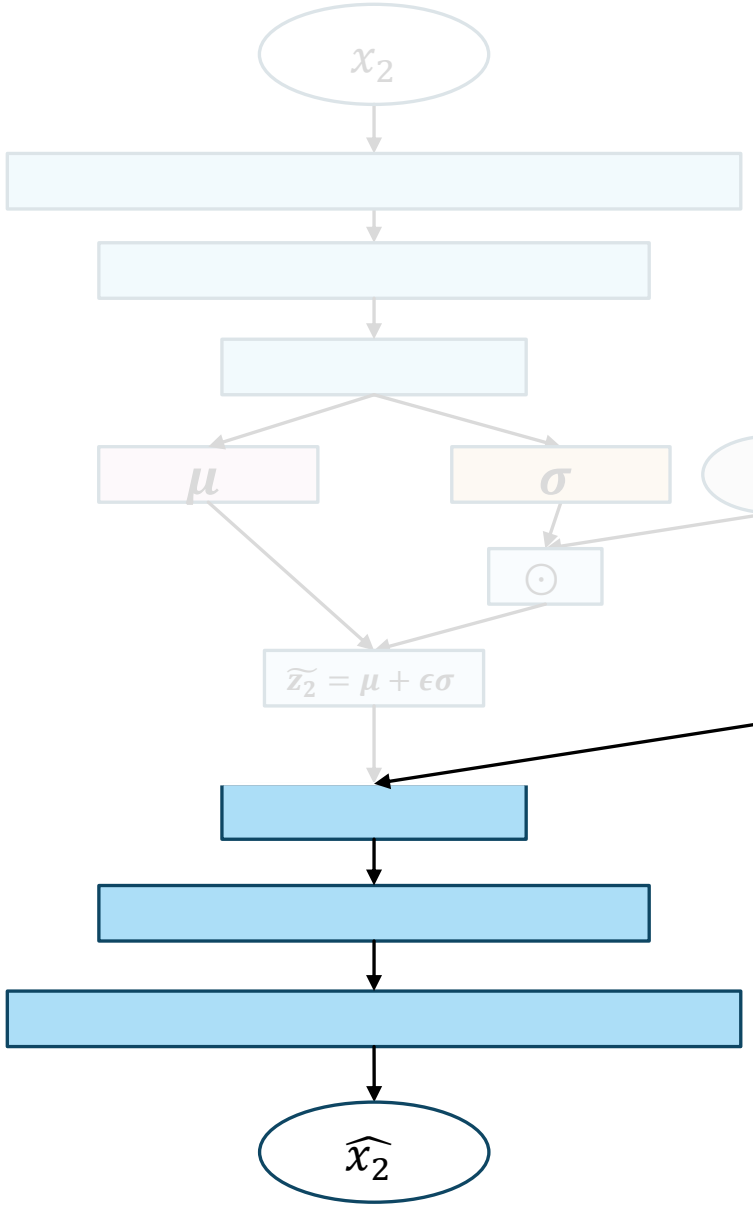


Latent Space of VAE

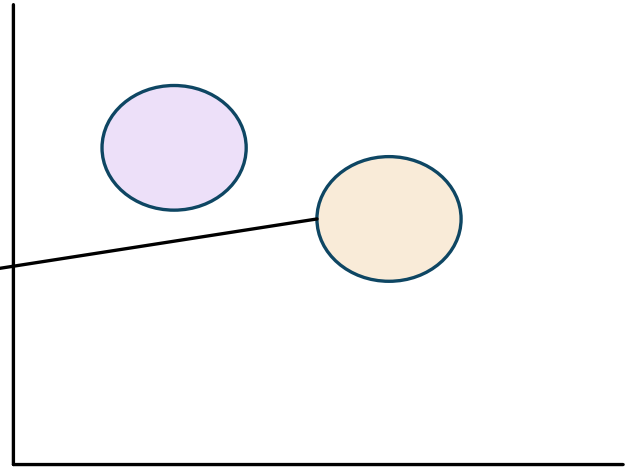


Similarly for each of the other data points, we will get such a region in the latent space

Latent Spaces: AE vs VAE

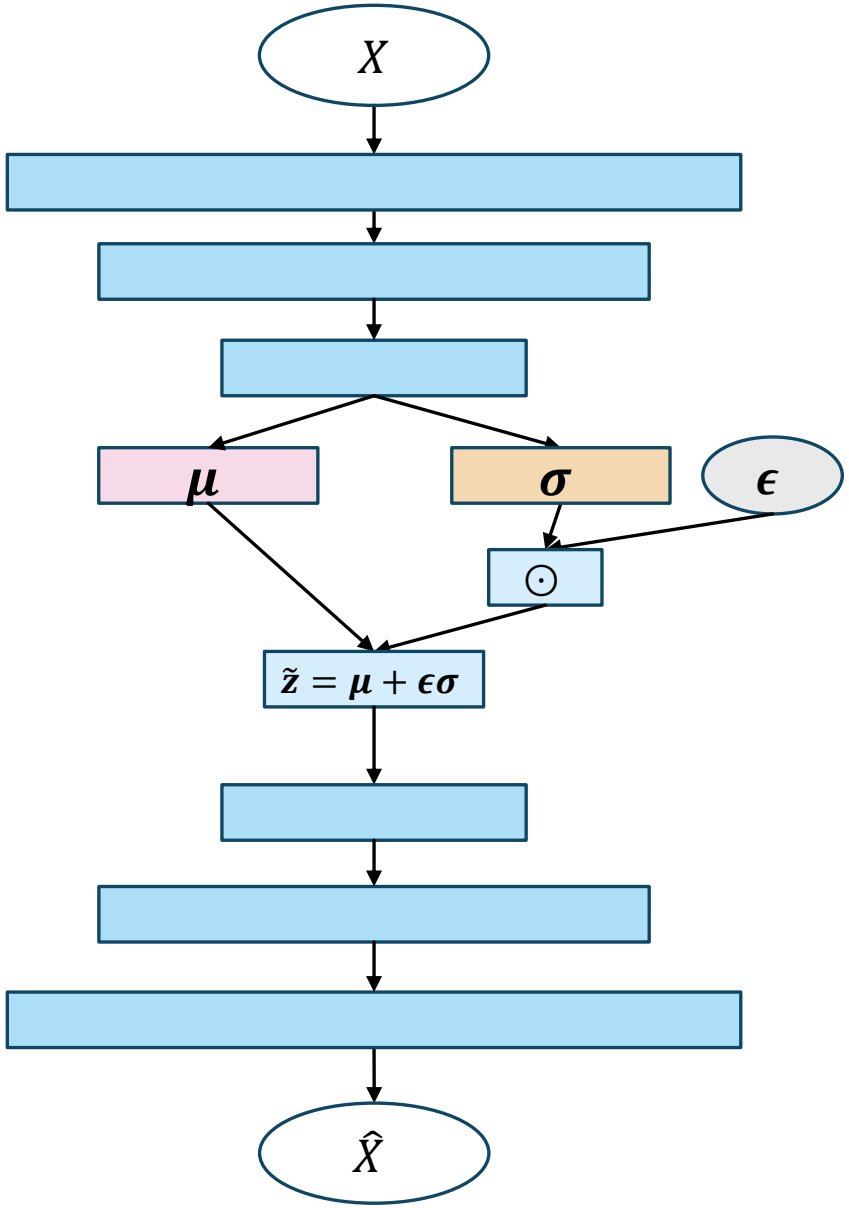


Latent Space of VAE

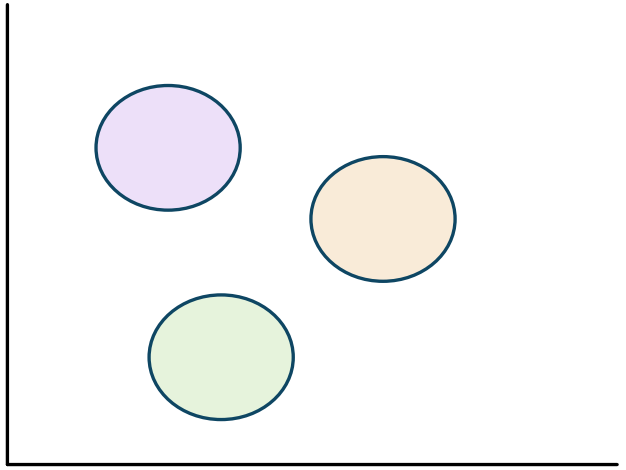


Similarly for each of the other data points, we will get such a region in the latent space

Latent Spaces: AE vs VAE

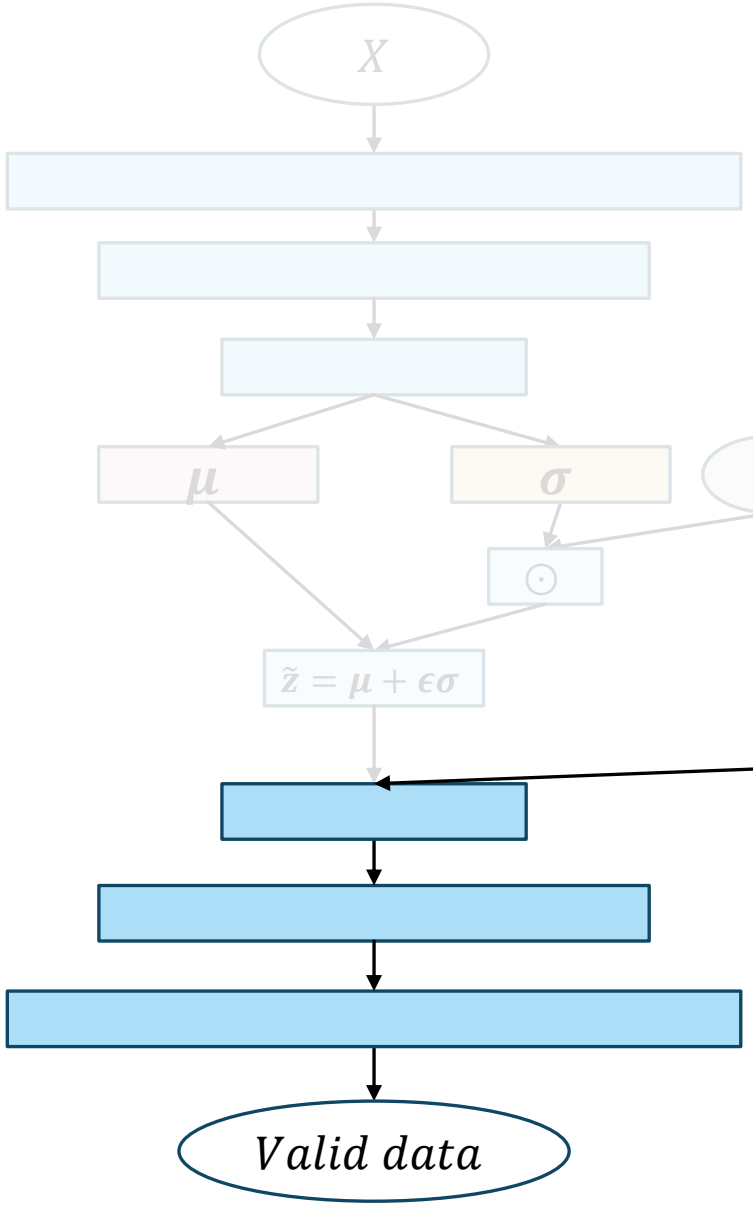


Latent Space of VAE

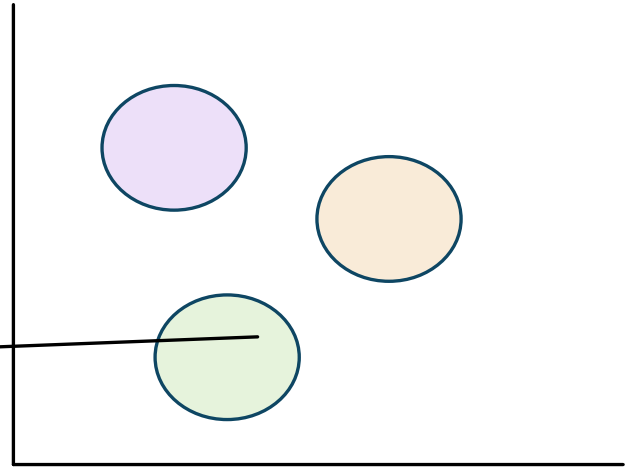


Once the training is complete, we will get many such regions in the latent space

Latent Spaces: AE vs VAE

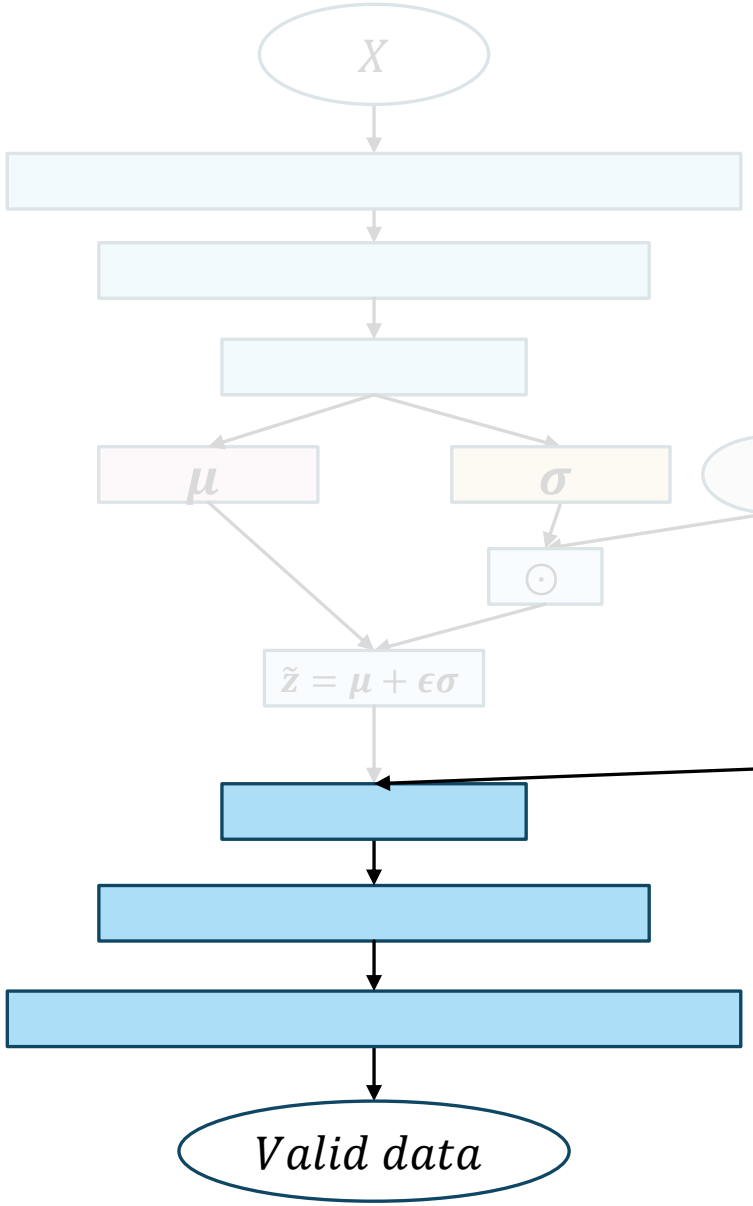


Latent Space of VAE

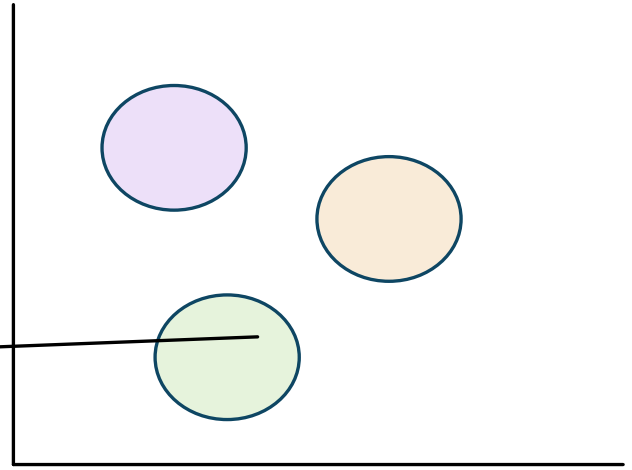


If we apply any points from those regions to the trained decoder, the decoder will produce valid data

Latent Spaces: AE vs VAE

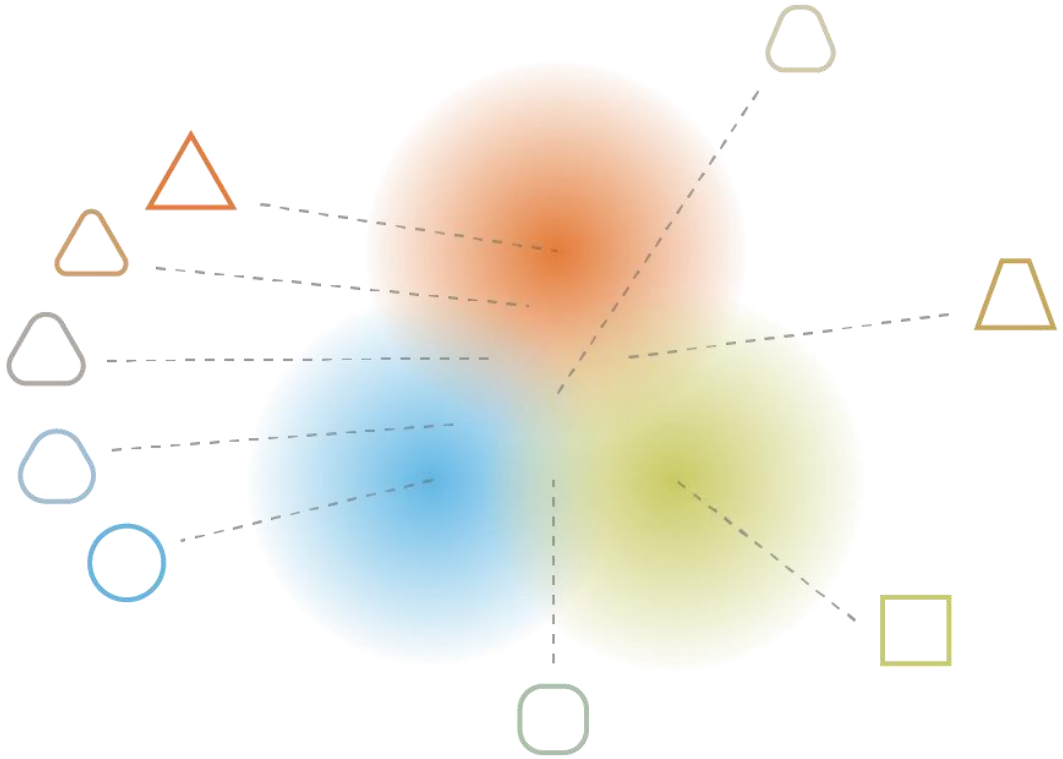


Latent Space of VAE



Thus, the latent space of the VAE is more continuous than that of AE

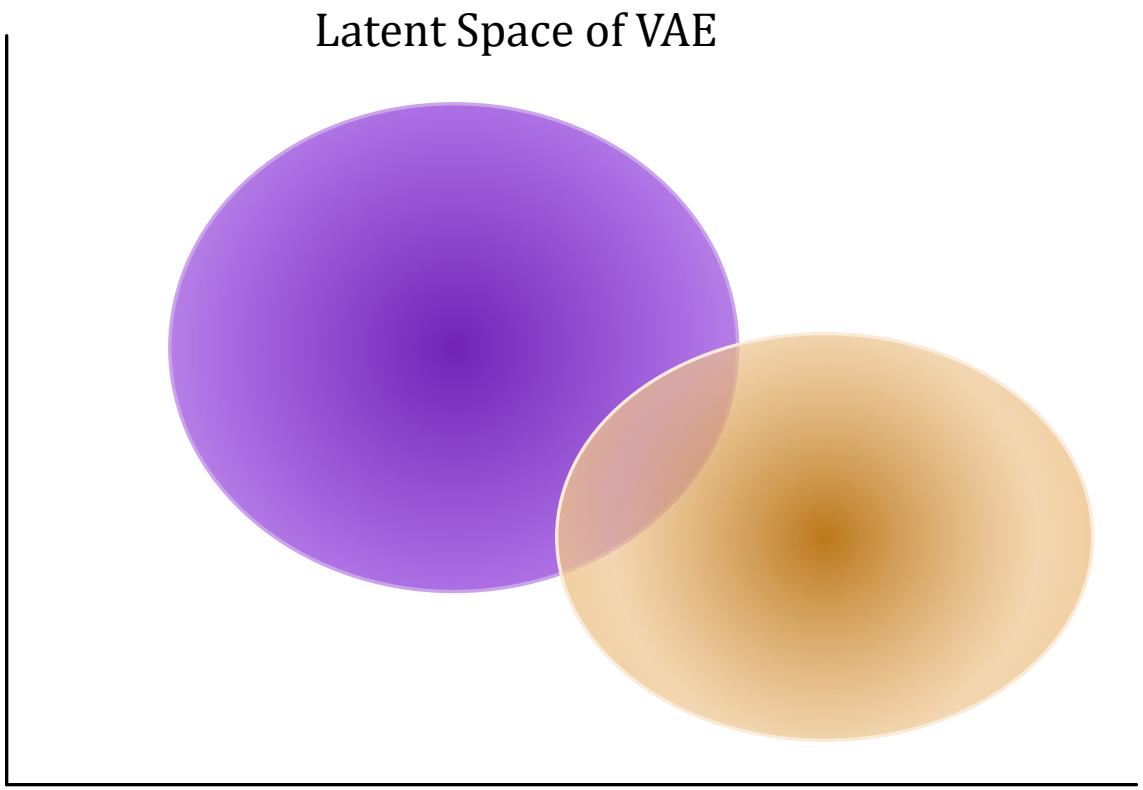
Latent Space of VAE



we can observe that continuity and completeness obtained with regularization **tend to create a "gradient" over the information encoded in the latent space**. For example, a point of the latent space that would be halfway between the means of two encoded distributions coming from different training data should be decoded in something that is somewhere between the data that gave the first distribution and the data that gave the second distribution as it may be sampled by the autoencoder in both cases.

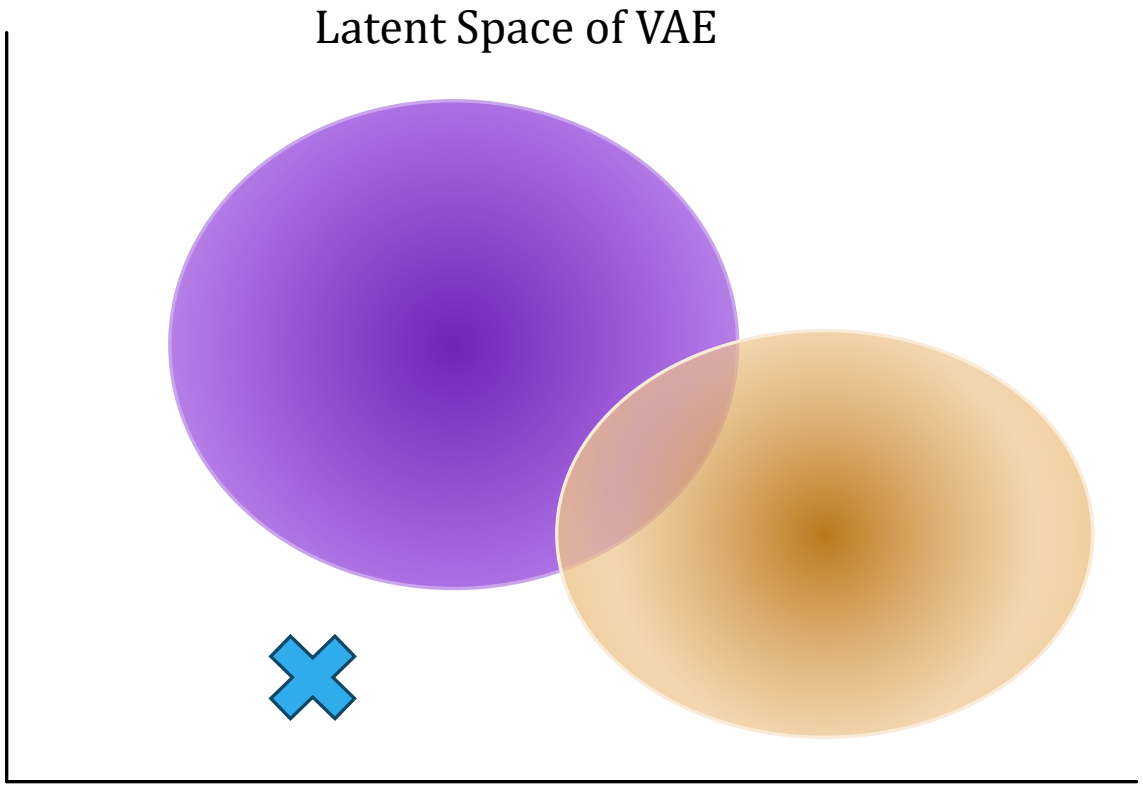
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Drawback of VAE



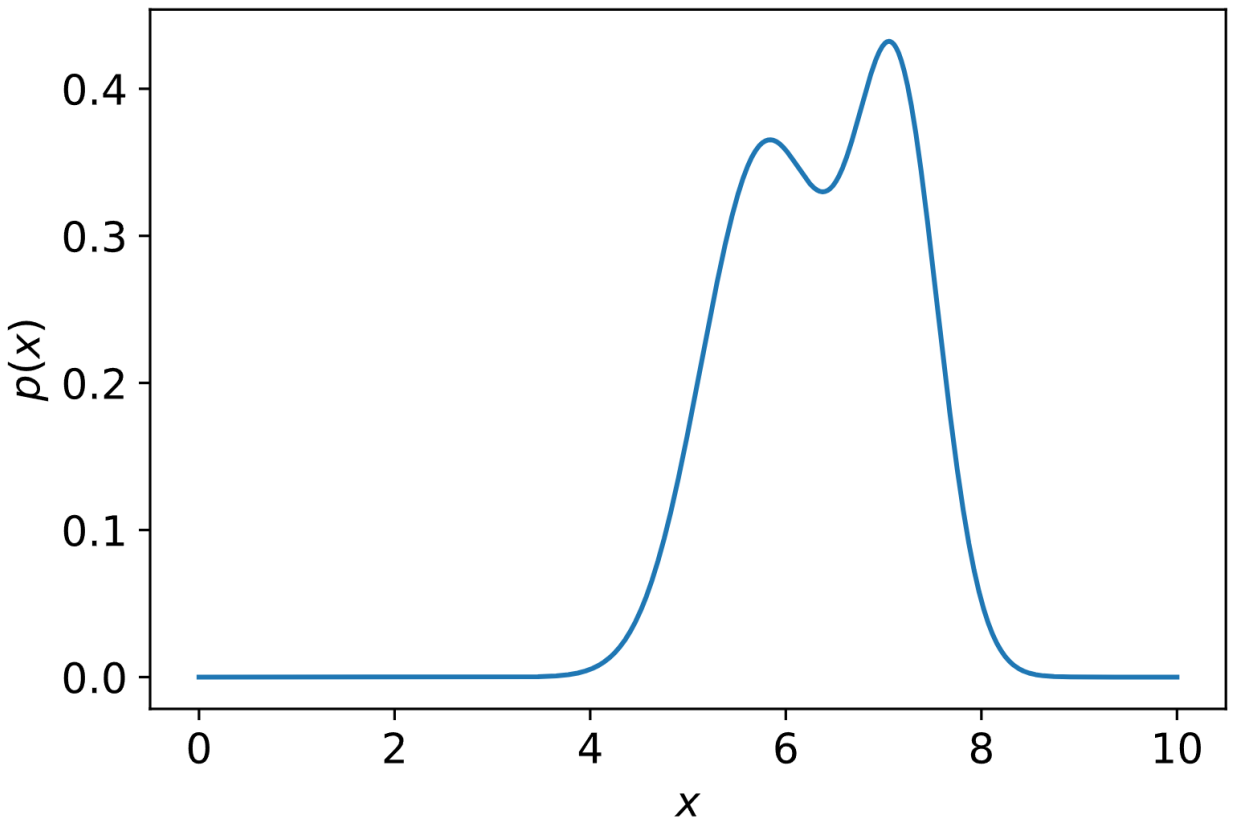
Drawback of VAE

- At the time of sampling, we may sample from any point in the latent space
- Such a point may not lie close to any mean value
- Such a point may give rise to unrealistic or blurry images

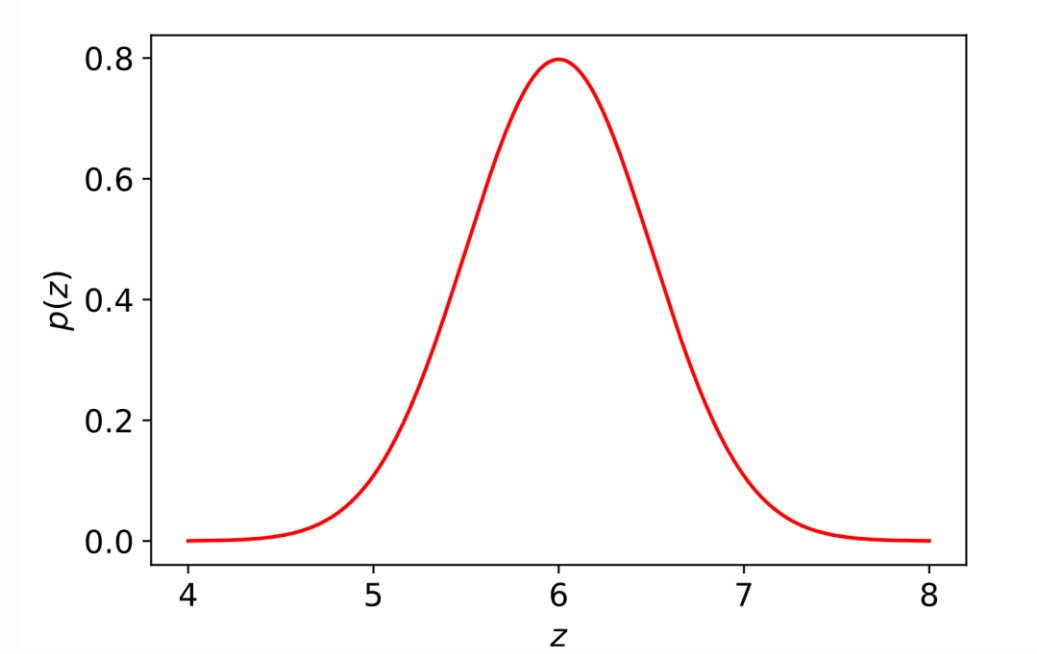


Let's Dig a Bit Deeper: Variational Inference

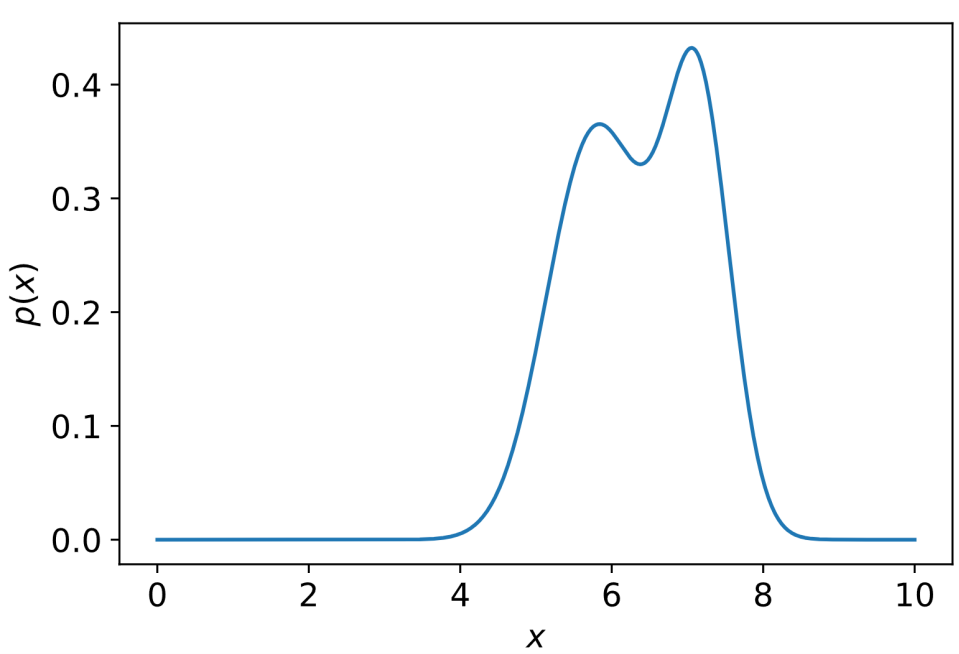
- Consider this pdf
- It's a complex pdf (more like real data)

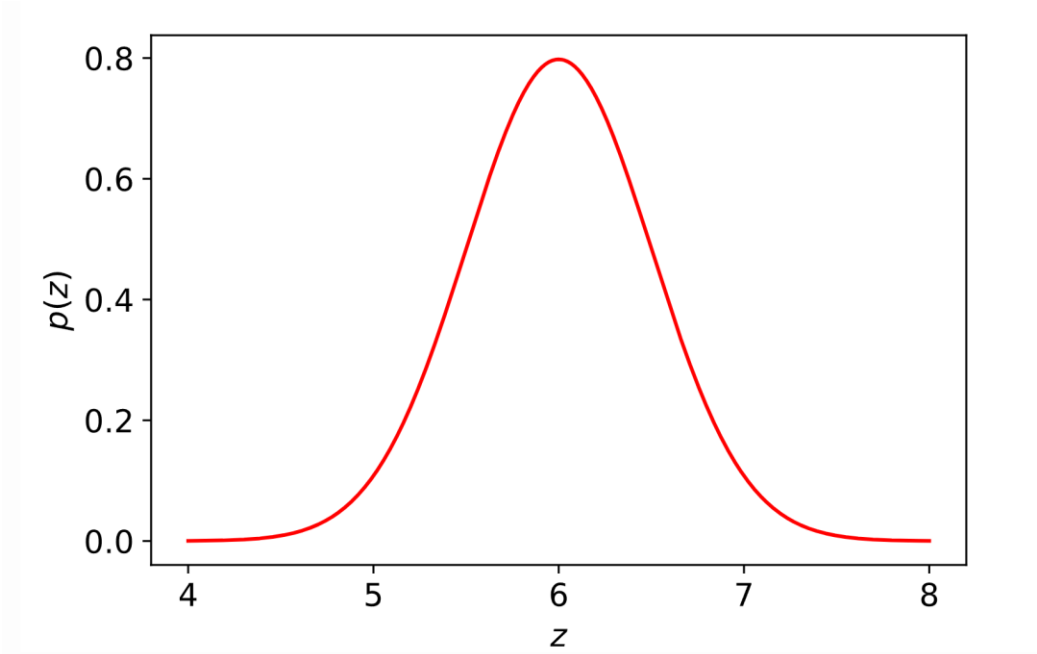


https://yunfanj.com/blog/2021/01/11/ELBO.html?form=MG0AV3

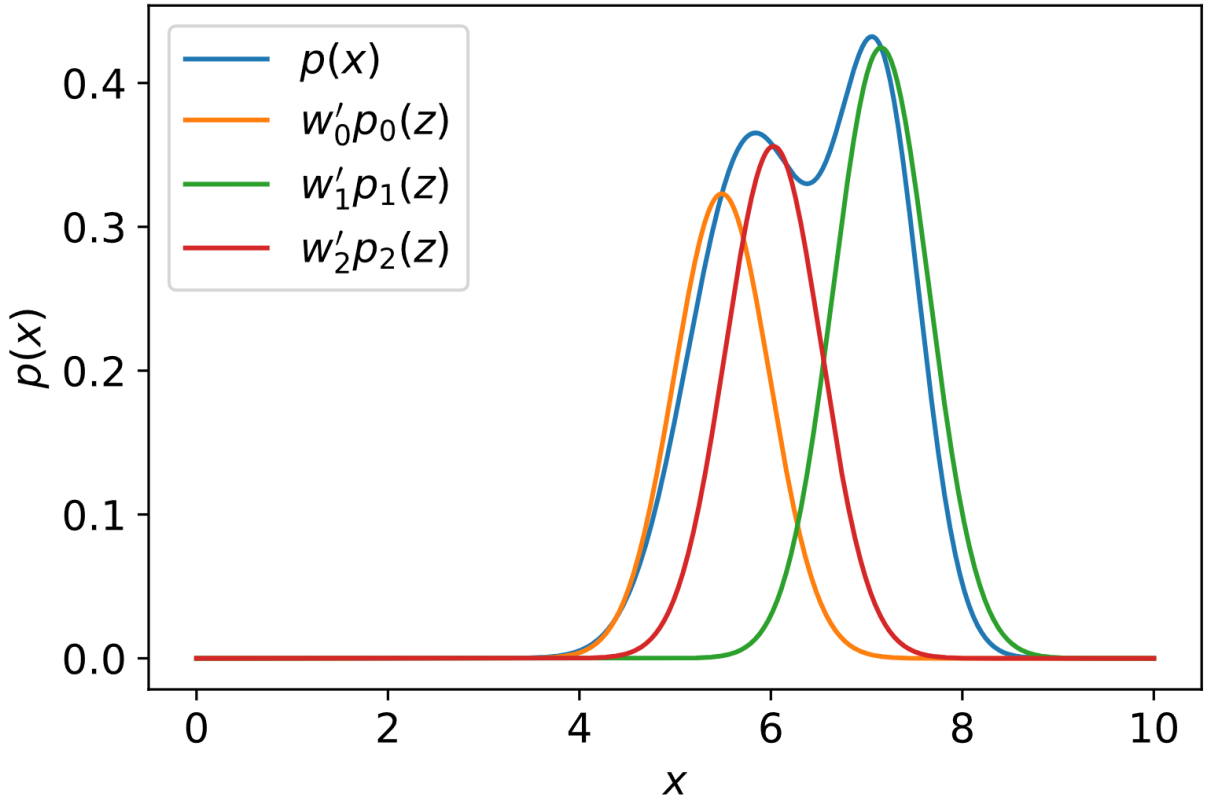


- Can you model this complex distribution $\left(p(x)\right)$ with the help of simpler distribution, such as a Gaussian $\left(p(z)\right)$?



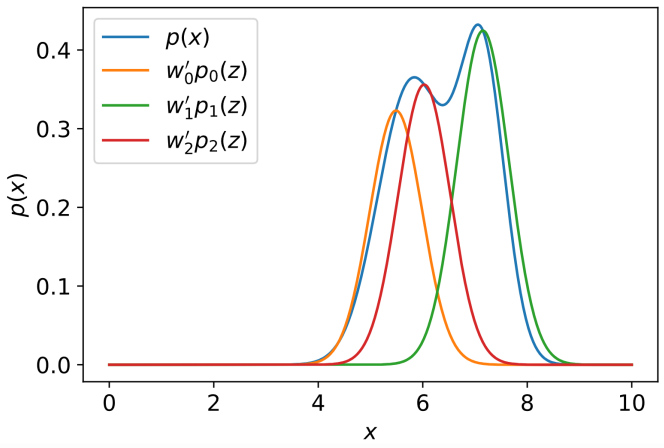


- Can you model this complex distribution $\left(p(x)\right)$ with the help of simpler distribution, such as a Gaussian $\left(p(z)\right)$?
  - Use some transformation $f(\cdot)$



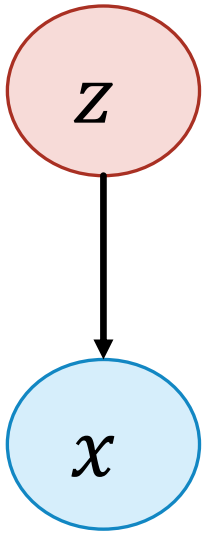
- We take several shifted copies of $p(z)$
  - $p_0(z), p_1(z), p_2(z)$
- Scale them with suitable scaling factors
  - w_0', w_1', w_2'
- Add the scaled and shifted Gaussians

Latent Variable Model

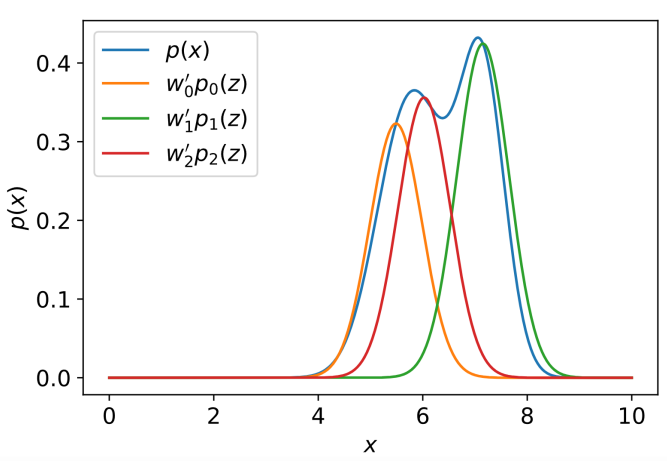


- Consider observation x (a random variable)
  - For example, fever

- Let's say, random variable z indicates the causes of fever
  - z_1: Viral infection, z_2: Bacterial infection, z_3: Other causes



Latent Variable Model

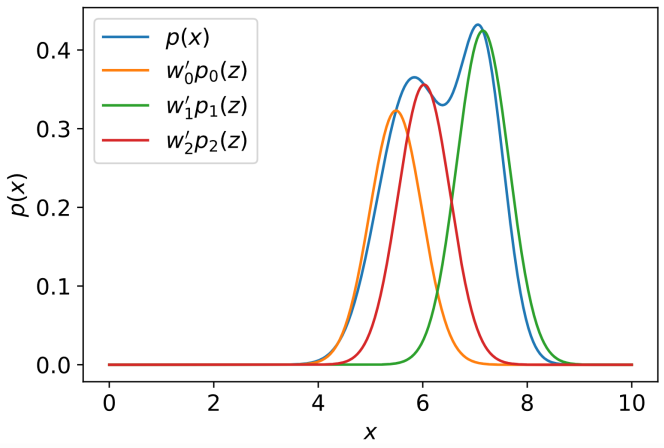


- Consider observation x (a random variable)
  - For example, fever

- Let's say, random variable z indicates the causes of fever
  - z_1: Viral infection, z_2: Bacterial infection, z_3: Other causes

$$p(x) = p(x, z_1) + p(x, z_2) + p(x, z_3)$$

Latent Variable Model



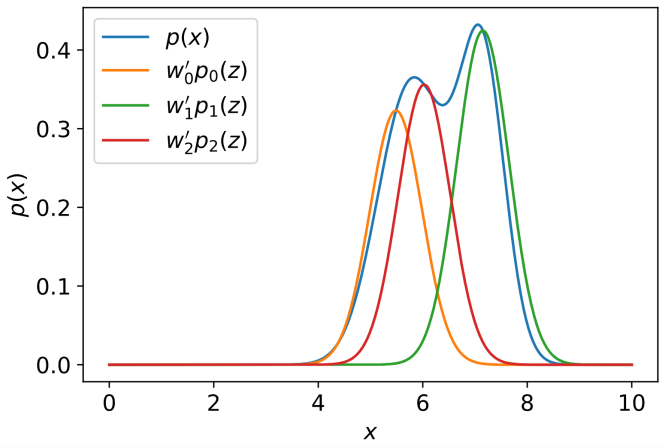
- Consider observation x (a random variable)
  - For example, fever

- Let's say, random variable z indicates the causes of fever
  - z_1: Viral infection, z_2: Bacterial infection, z_3: Other causes

$$p(x) = p(x, z_1) + p(x, z_2) + p(x, z_3)$$

$$= p(x|z_1)p(z_1) + p(x|z_2)p(z_2) + p(x|z_3)p(z_3)$$

Latent Variable Model



- Consider observation x (a random variable)
  - For example, fever

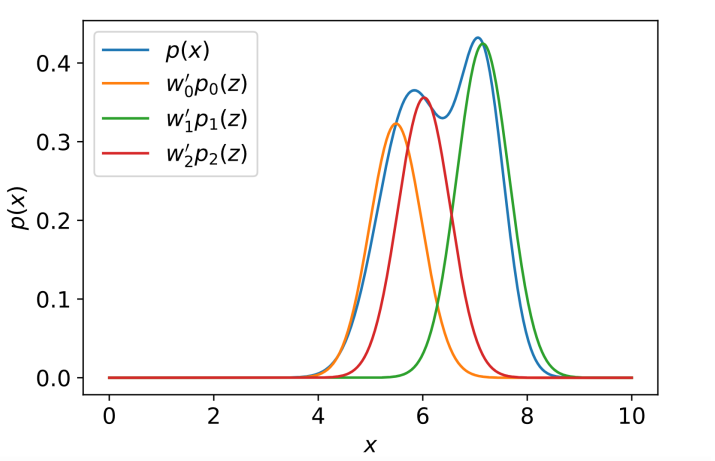
- Let's say, random variable z indicates the causes of fever
  - z_1: Viral infection, z_2: Bacterial infection, z_3: Other causes

$$p(x) = p(x, z_1) + p(x, z_2) + p(x, z_3)$$

$$= p(x|z_1)p(z_1) + p(x|z_2)p(z_2) + p(x|z_3)p(z_3)$$

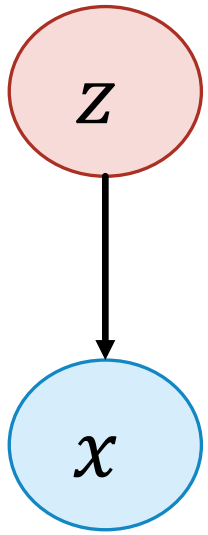
$$= \sum_{i=1}^{3} p(x|z_i)p(z_i)$$

Latent Variable Model

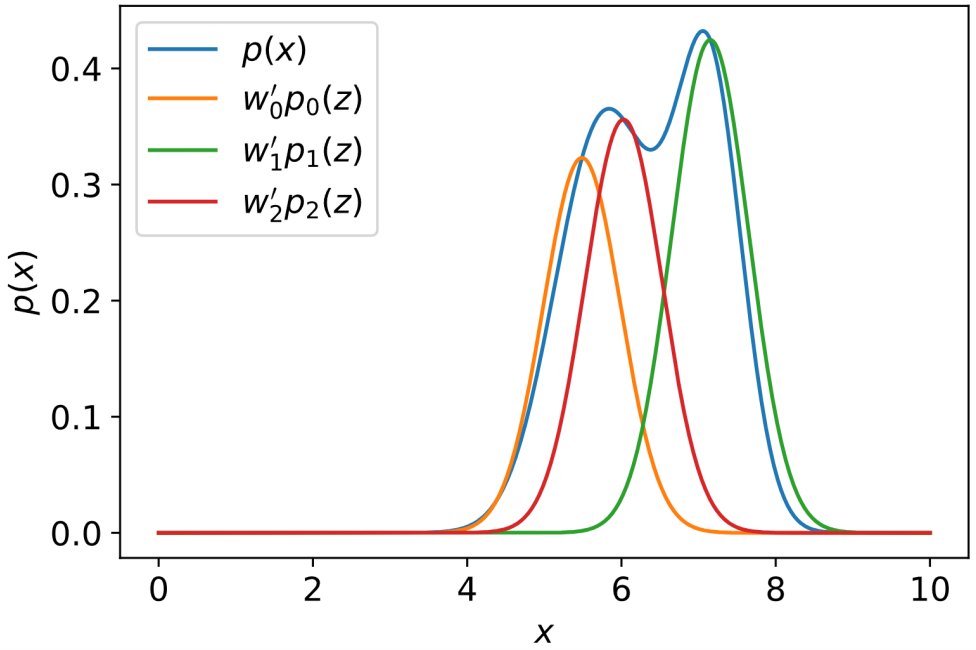


- For continuous random variables

$$p(x) = \int_z p(x|z)p(z)dz$$



Latent Variable Model



$$p(x) = \int_z p(x|z)\, p(z)dz$$

Scaling

Shifted pdf

KL Divergence

The KL-divergence between $p(x)$ and $q(x)$ is

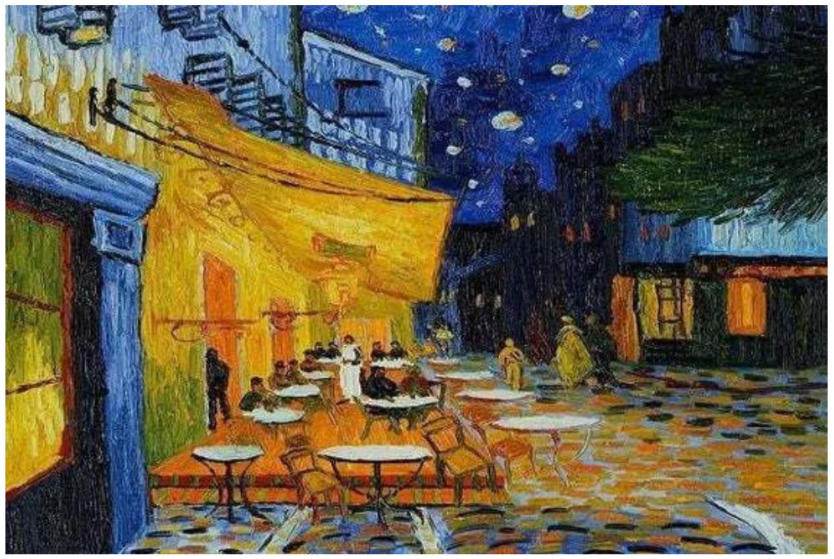
$$D_{KL} = \sum p(x) \log \frac{p(x)}{q(x)}$$

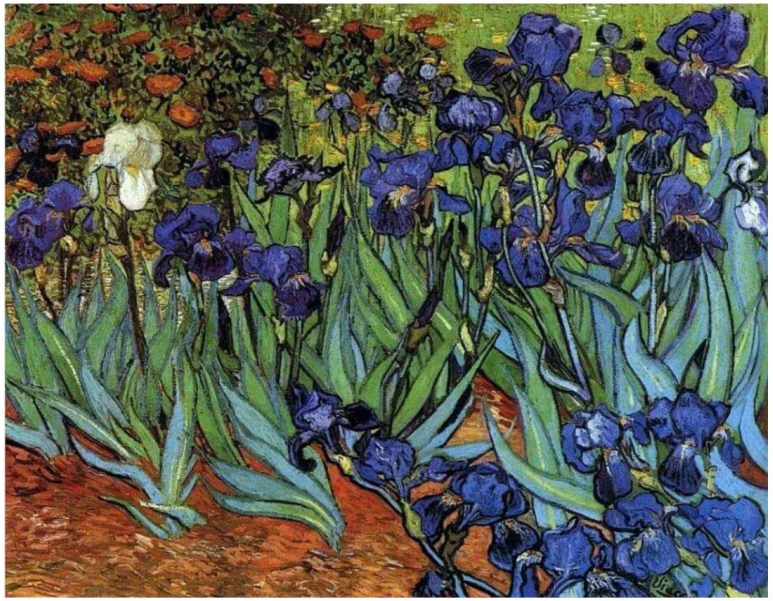
$$D_{KL} \geq 0$$

$$D_{KL}(p||q) \neq D_{KL}(q||p)$$

Latent Variable Model

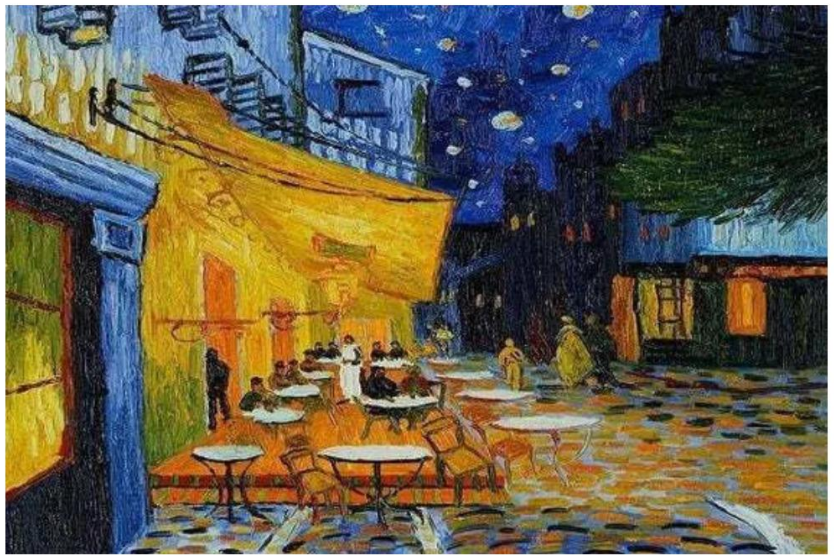
- Consider observation x (a random variable)
  - For example, paintings of Van Gogh

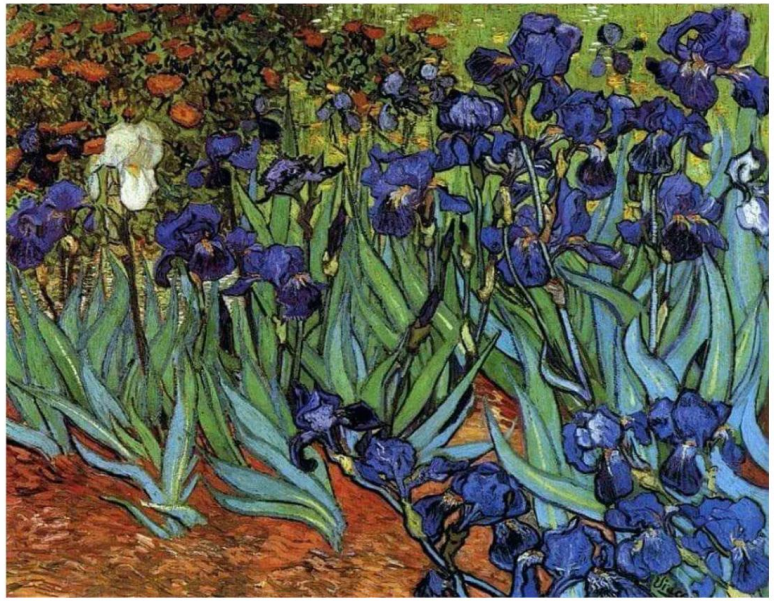




Latent Variable Model

- Consider observation x (a random variable)
  - $p(x)$ is a very complicated distribution
  - So, I want to convert $p(x)$ to a simple distribution $p(z|x)$ and sample from $p(z|x)$





Latent Variable Model

- Consider observation x (a random variable)
  - $p(x)$ is a very complicated distribution
  - So, I want to convert $p(x)$ to a simple distribution $p(z|x)$

- $p(z|x) = \frac{p(x,z)}{p(x)}$

- $p(x) = \int_z p(x|z)p(z)dz$

- Finding $p(x)$ is very difficult especially when z is of high-dimension
- Therefore, finding $p(z|x)$ is very difficult

Latent Variable Model

- $p(z|x) = \frac{p(x,z)}{p(x)}$

- Therefore, we approximate $p(z|x)$ by a tractable distribution $q(z|x)$
  - So, we try to make $q(z)$ as similar to $p(z|x)$ as possible

Latent Variable Model

- $p(z|x) = \frac{p(x,z)}{p(x)}$

- Therefore, we approximate $p(z|x)$ by a tractable distribution $q(z)$
  - So, we try to make $q(z)$ as similar to $p(z|x)$ as possible

- $D_{KL}\big(q(z)||p(z|x)\big) = -\sum_z q(z) \log \frac{p(x,z)}{q(z)} + \log p(x)$

Latent Variable Model

- $D_{KL}\left(q(z)||p(z|x)\right) = -\sum_z q(z)\log\frac{p(x,z)}{q(z)} + \log p(x)$

- I want to minimize $D_{KL}\left(q(z)||p(z|x)\right)$

Latent Variable Model

- $D_{KL}\left(q(z)||p(z|x)\right) = -\sum_z q(z)\log\frac{p(x,z)}{q(z)} + \log p(x)$
- I want to minimize $D_{KL}\left(q(z)||p(z|x)\right)$
- x is my observation (training data) which is given
  - So, $p(x)$ is constant

Latent Variable Model

- $D_{KL}\big(q(z)||p(z|x)\big) = -\sum_z q(z)\log\frac{p(x,z)}{q(z)} + \log p(x)$
- I want to minimize $D_{KL}\big(q(z)||p(z|x)\big)$
- x is my observation (training data) which is given
  - So, $p(x)$ is constant
- So, minimizing $D_{KL}\big(q(z)||p(z|x)\big)$ is equivalent to maximizing $\sum_z q(z)\log\frac{p(x,z)}{q(z)}$

Latent Variable Model

- $\sum_z q(z) \log \frac{p(x,z)}{q(z)}$ is called variational lower bound or evidence lower bound (ELBO)

Latent Variable Model

- $\sum_z q(z) \log \frac{p(x,z)}{q(z)}$ is called variational lower bound or evidence lower bound (ELBO)

- $D_{KL}\big(q(z)||p(z|x)\big) = -\sum_z q(z) \log \frac{p(x,z)}{q(z)} + \log p(x)$
- Since $D_{KL}\big(q(z)||p(z|x)\big) \geq 0$

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} \leq \log p(x)$$

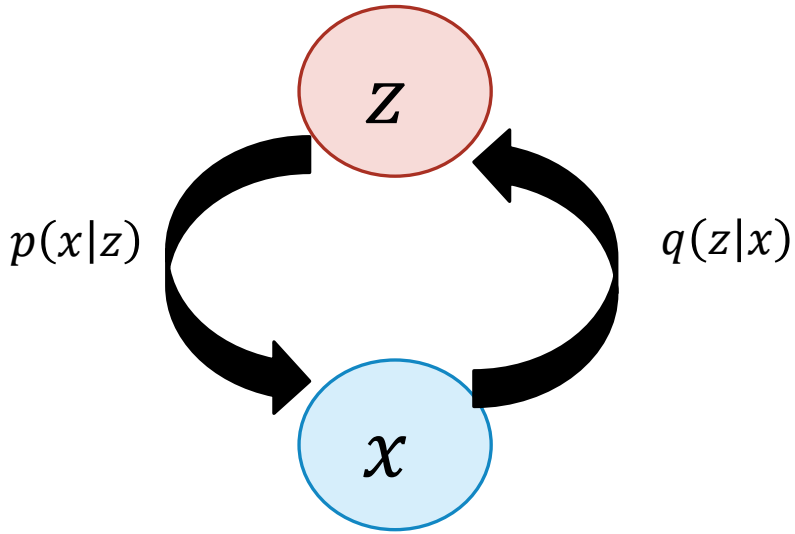
That's why $\sum_z q(z) \log \frac{p(x,z)}{q(z)}$ is called lower bound

Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \sum_z q(z) \log p(z|x) - D_{KL}\big(q(z)||p(z)\big)$$
$$= \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

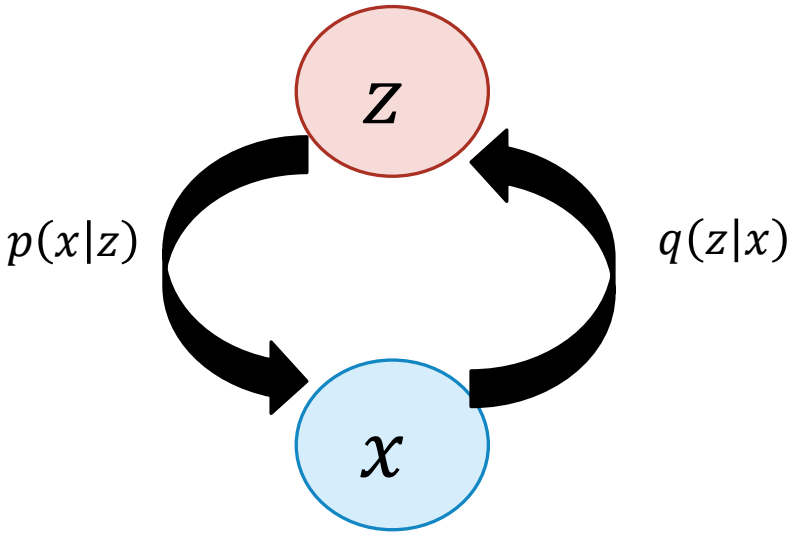
Variational Autoencoder

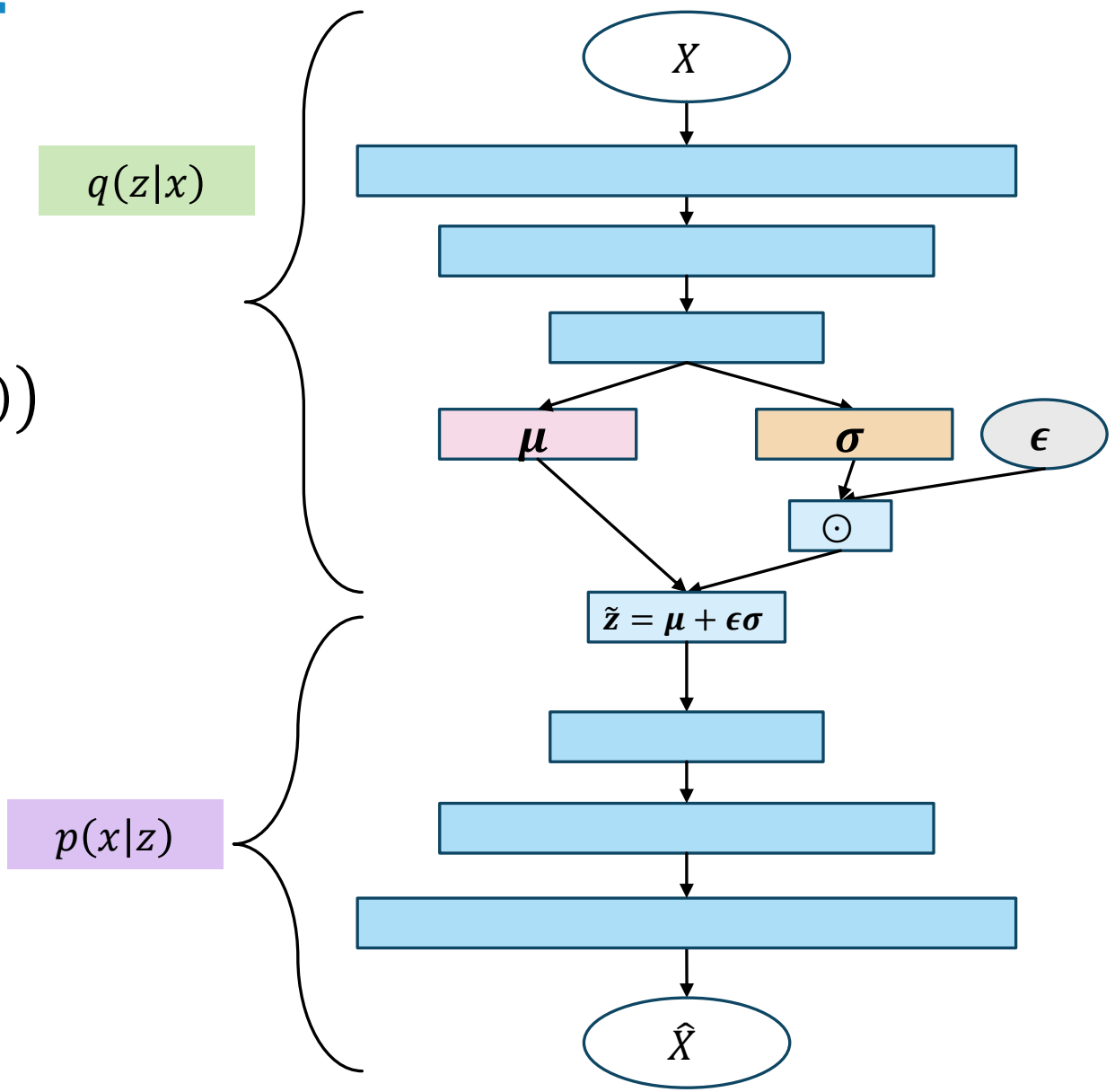
$$\sum_z q(z) \log \frac{p(x,z)}{q(z)}$$
$$= \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$



Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)}$$
$$= \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

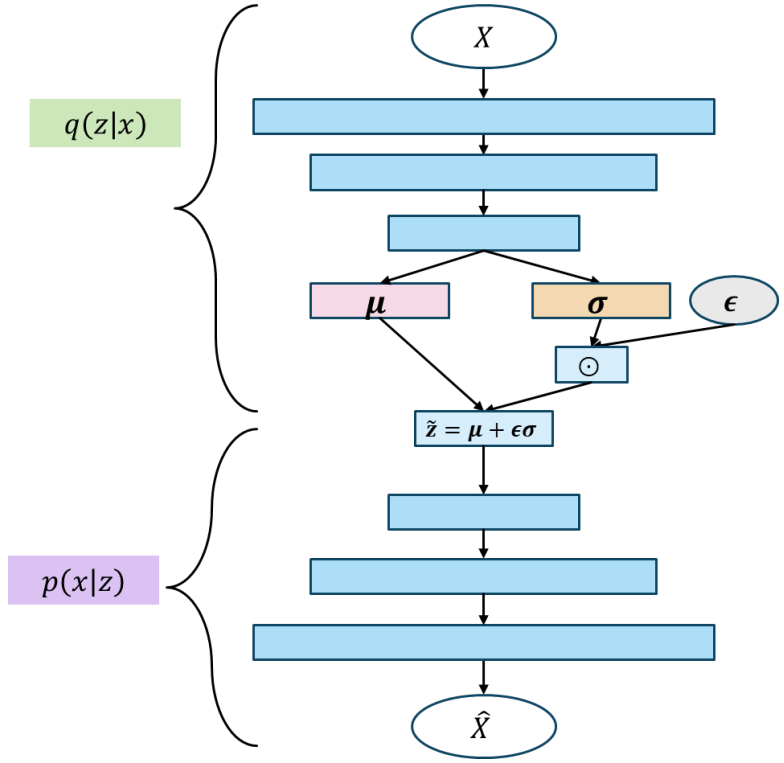




Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

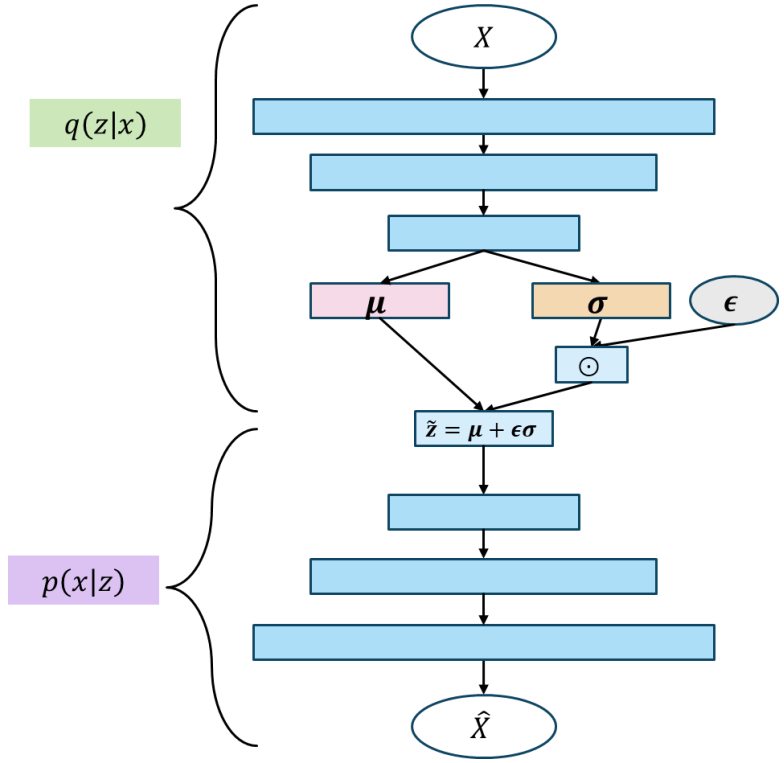
Maximize



Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Maximize Maximize Minimize



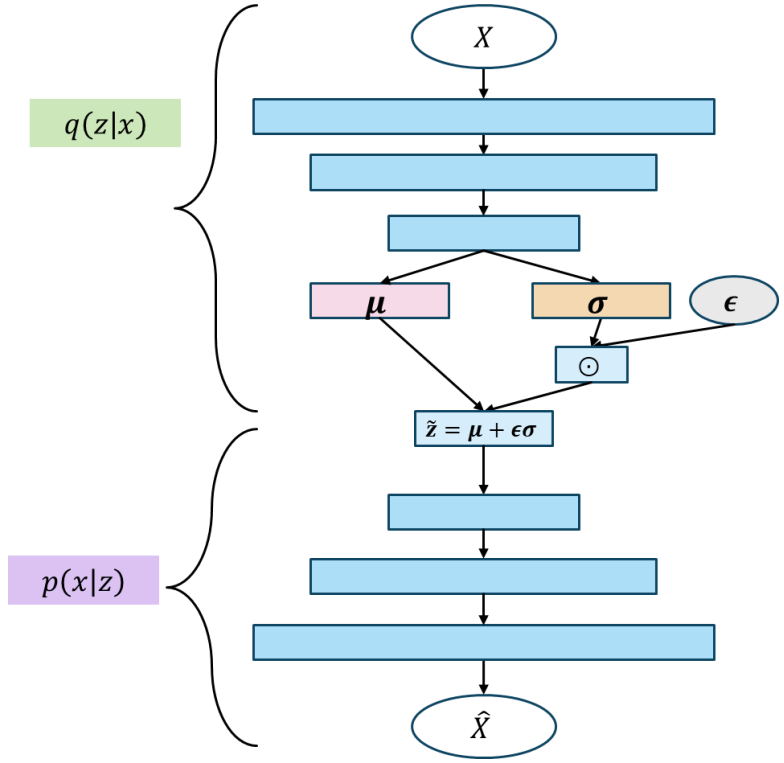
Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Maximize Maximize Minimize

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(\hat{x}|x)\big) - D_{KL}\big(q(z)||p(z)\big)$$

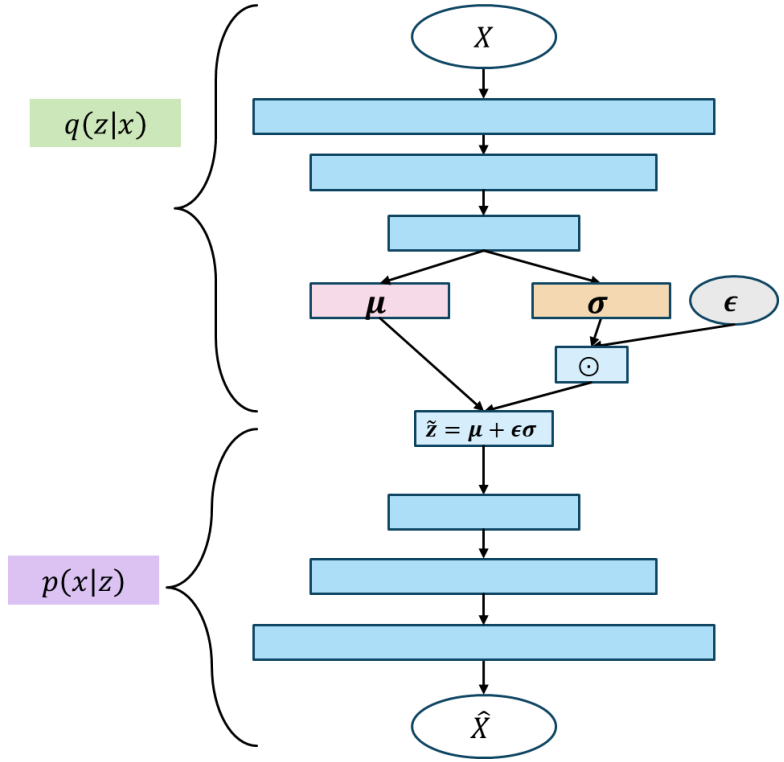
Reconstruction error



Variational Autoencoder

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(\hat{x}|x)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Reconstruction error



For Gaussian distribution, $\mathbb{E}_{q(z)} \log\big(p(\hat{x}|x)\big)$ will have the form of

$$\exp\big[-\big((\hat{x} - x)^2\big)\big]$$

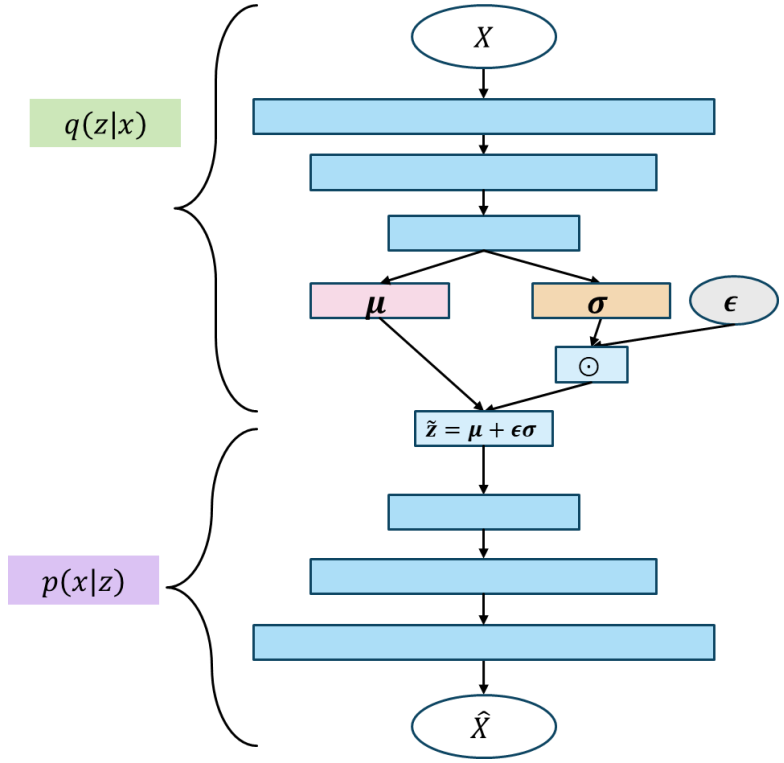
$$\sum_z q(z)\log\frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)}\log\big(p(x|z)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Maximize Maximize Minimize

$$\sum_z q(z)\log\frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)}\log\big(p(\hat{x}|x)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Reconstruction error

Maximize Maximize Minimize



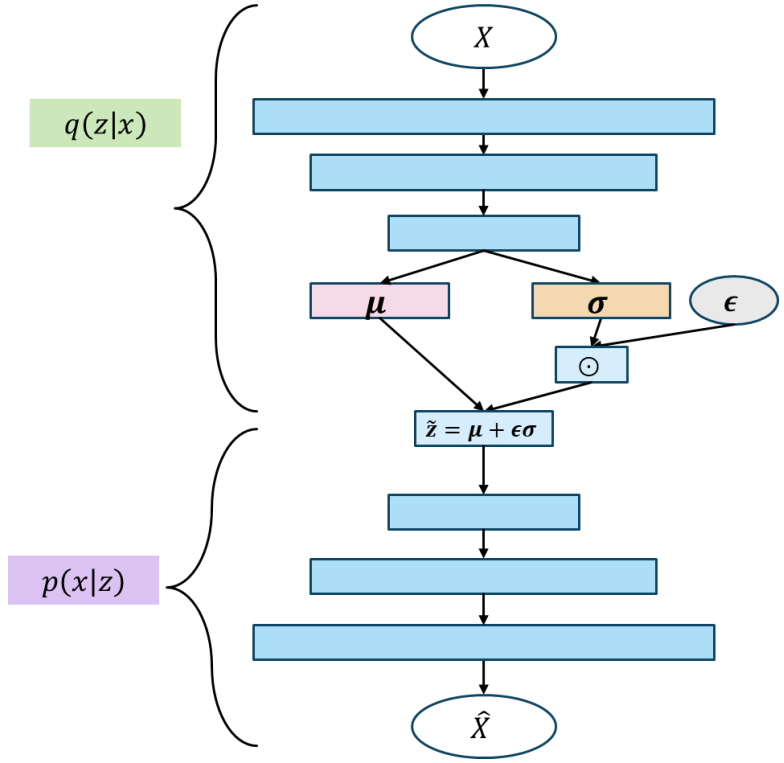
For Gaussian distribution, $\mathbb{E}_{q(z)}\log\big(p(\hat{x}|x)\big)$ will have the form of

$$\exp\big[-\big((\hat{x}-x)^2\big)\big]$$

$$\sum_z q(z) \log \frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)} \log\big(p(\hat{x}|x)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Reconstruction error

Maximize | Maximize | Minimize



For Gaussian distribution, $\mathbb{E}_{q(z)} \log\big(p(\hat{x}|x)\big)$ will have the form of

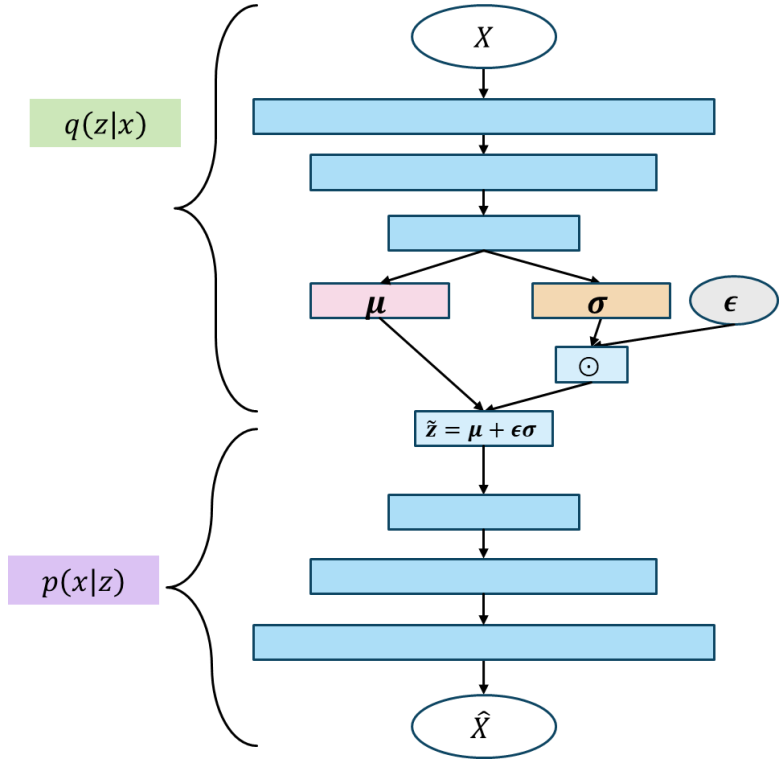
$$\exp\big[-\big((\hat{x} - x)^2\big)\big]$$

To maximize reconstruction error, we have to minimize $\big((\hat{x} - x)^2\big)$

$$\sum_z q(z)\log\frac{p(x,z)}{q(z)} = \mathbb{E}_{q(z)}\log\big(p(\hat{x}|x)\big) - D_{KL}\big(q(z)||p(z)\big)$$

Reconstruction error

Maximize | Maximize | Minimize



To maximize reconstruction error, we have to minimize $\big((\hat{x}-x)^2\big)$

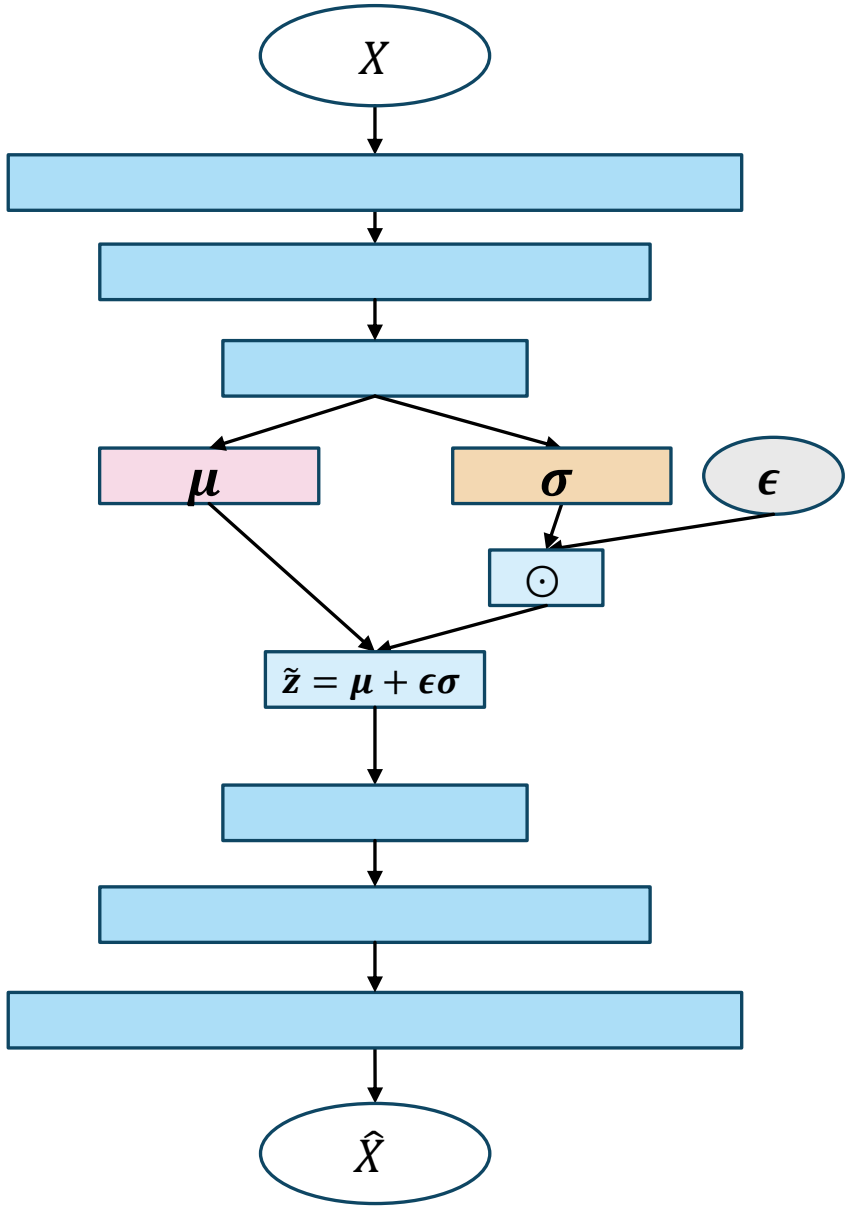
We also have to minimize $D_{KL}\big(q(z)||p(z)\big)$

Variational Autoencoder: Loss Function

- Reconstruction loss: $MSE(X, \hat{X})$

- KL divergence loss

$$KL(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0,1)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_i^2 + \mu_i^2 - 1 - \ln \sigma_i^2)$$



Generative Adversarial Network (GAN)

GAN: Basic Philosophy

- Take a simple distribution
- Sample from the distribution and get a sample z
- Feed sample z to a neural network
  - The neural network produces a desired synthetic data $\tilde{x}$

GAN: Basic Philosophy

- Take a simple distribution
  - Why?
    - Easy to sample
- Sample from the distribution and get a sample z
- Feed sample z to a neural network
  - The neural network produces a desired synthetic data $\tilde{x}$

GAN: Basic Philosophy

- Take a simple distribution
  - Why?
    - Easy to sample
- Sample from the distribution and get a sample z
- Feed sample z to a neural network
  - The neural network produces a desired synthetic data $\tilde{x}$
    - How to know if $\tilde{x}$ is good enough?

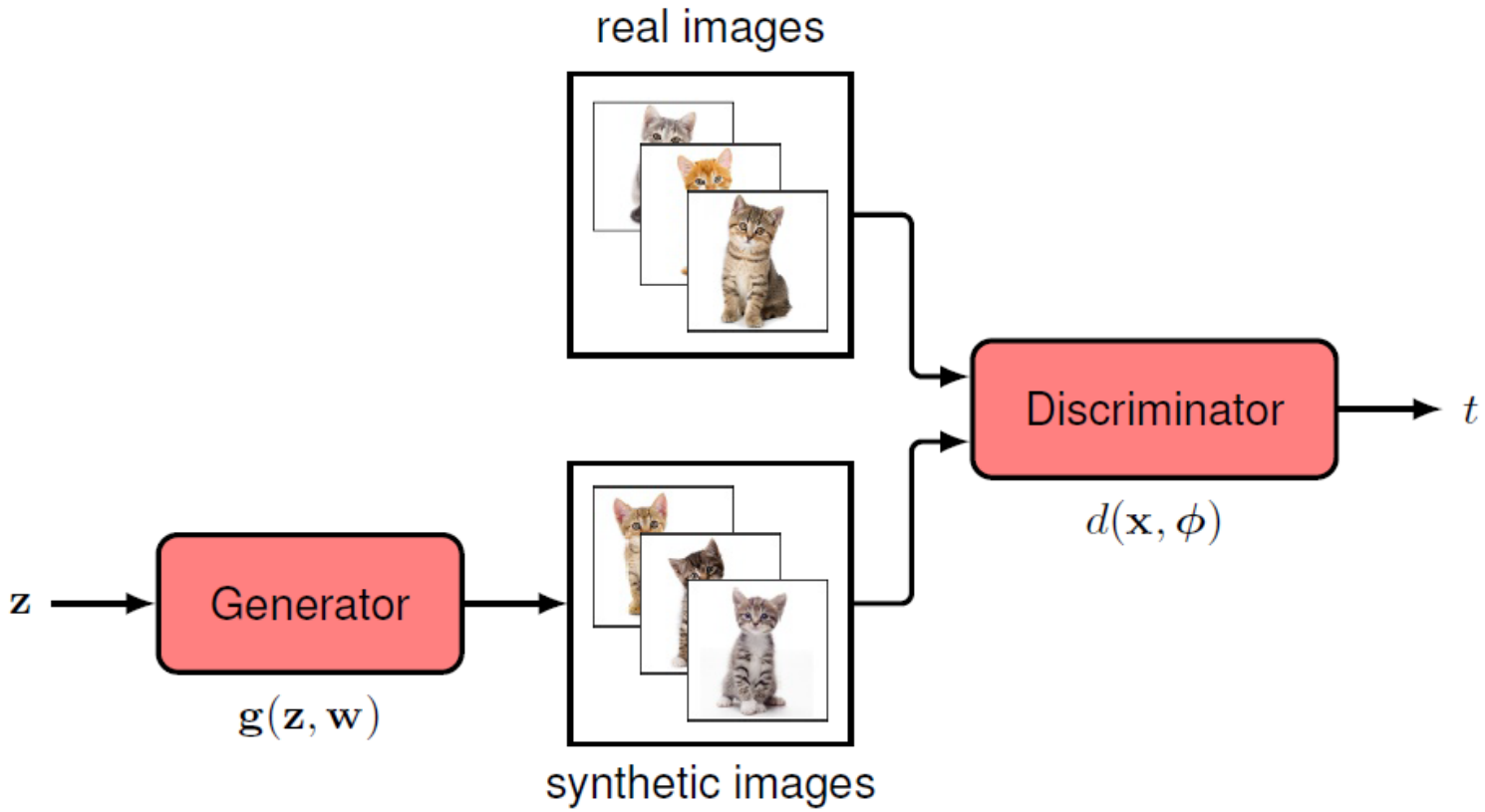
GAN: Basic Philosophy

- Take a simple distribution
  - Why?
    - Easy to sample
- Sample from the distribution and get a sample z
- Feed sample z to a neural network
  - The neural network produces a desired synthetic data $\tilde{x}$
    - How to know if $\tilde{x}$ is good enough?
      - Use an automated expert to judge the quality of $\tilde{x}$ (discriminator)

Generative Adversarial Networks

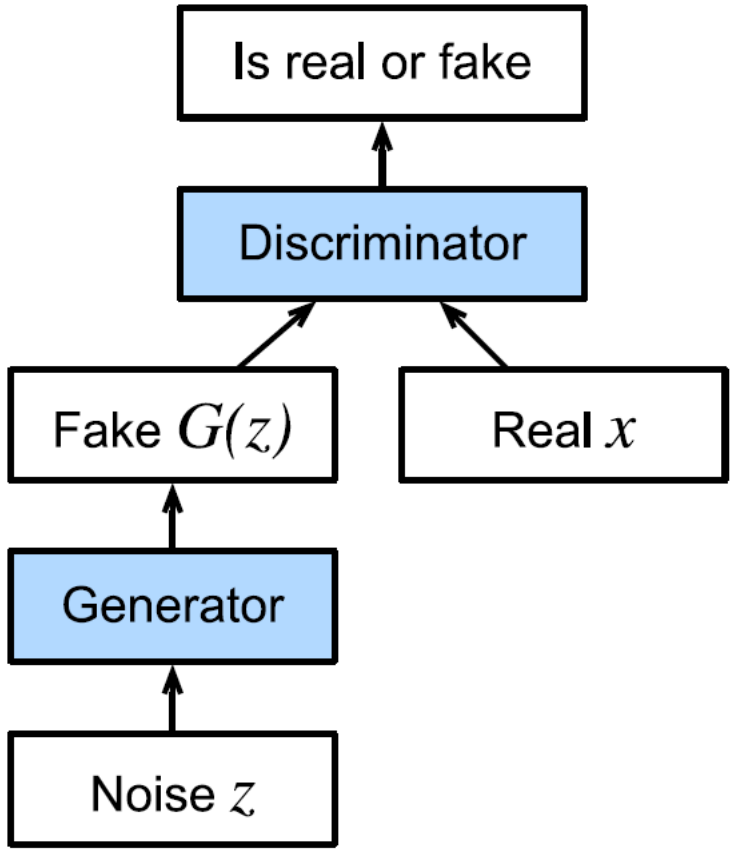
- Distribution of data: $p_D(x)$
- Goal of GAN: Design a sampler to generate samples with a distribution $p_m(x)$ such that $p_m(x)$ is similar to $p_D(x)$
- Use of discriminative model to get good generative model

Generative Adversarial Networks



Generative Adversarial Networks

- Apply noise to the generator (a neural network)
- Ask it to generate a data (synthetic)
- The generated data should be good enough to convince the discriminator (another neural network) that it was a real data
- The main idea: the generator and the discriminator competes with each other



The Discriminator

- Train with labeled real data (class 1) and synthetic (fake) data (class 0)

- Consider data x
  - Predicted class probability by the discriminator is $D(x)$
    - $D(x)$ is a number between 0 and 1
      - $D(x) = 0$ the discriminator is absolutely confident that data x is fake
      - $D(x) = 1$ the discriminator is absolutely confident that data x is real
  - Ground truth label of data x is y
    - $y = 0$ the data x is actually fake
    - $y = 1$ the data x is actually real

The Discriminator

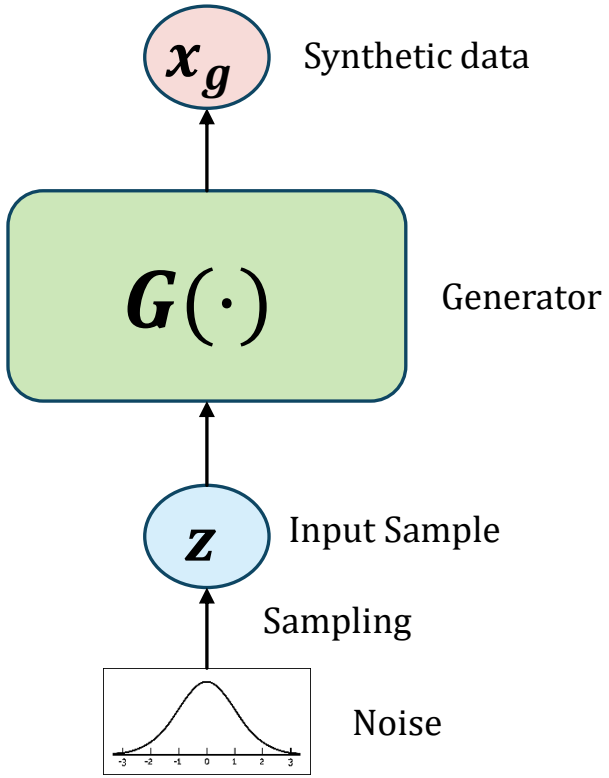
- Consider data x
  - Predicted class probability by the discriminator is $D(x)$
    - $D(x)$ is a number between 0 and 1
      - $D(x) = 0$ the discriminator is absolutely confident that data x is fake
      - $D(x) = 1$ the discriminator is absolutely confident that data x is real
  - Ground truth label of data x is y

- The discriminator will try to minimize the cross-entropy loss for classification into real and fake classes

$$L_{ce} = -y \log D(x) - (1 - y) \log(1 - D(x))$$

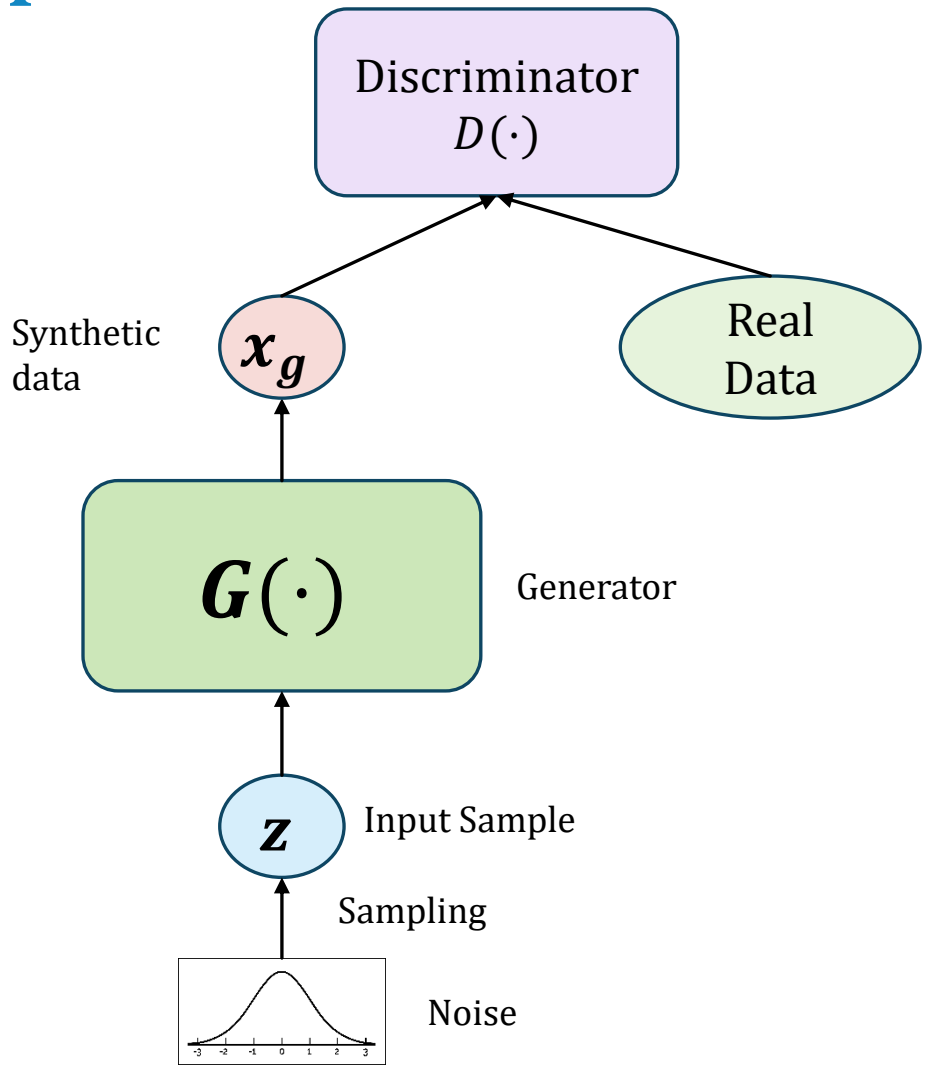
The Generator

- We start with a simple distribution, such $\mathcal{N}(0,1)$ (often called noise in the context of GAN)
- Sample z from the above distribution
- The generator neural network implements a function $G(\cdot)$
- When z is applied to the generator, it produces an output $x_g = G(z)$
  - Synthetic data



The Generator

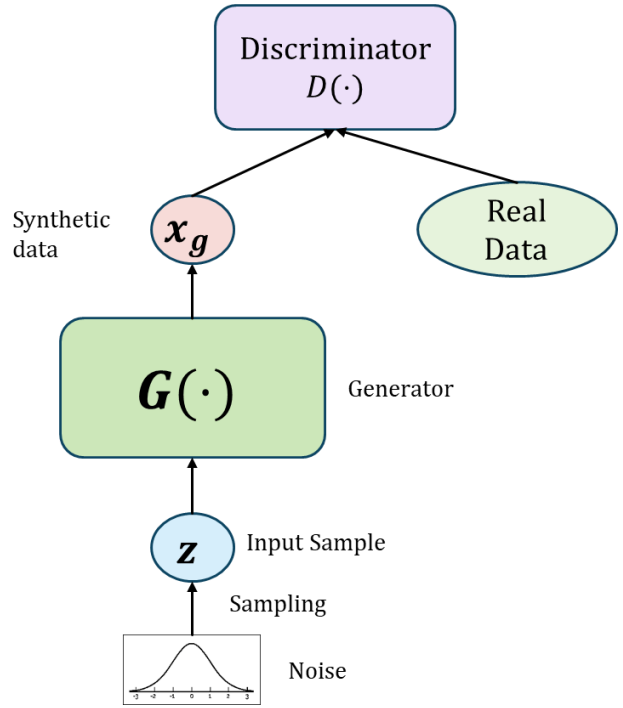
- When z is applied to the generator, it produces an output $x_g = G(z)$
  - Synthetic data
- Since x_g is synthetic, it's fake data
  - Ground truth class label of x_g is $y_g = 0$
  - The output class probability by the discriminator is $D(x_g)$
- But the generator tries to fool the discriminator
- So, the generator wants $D(x_g)$ to be equal to 1
- The discriminator wants $D(x_g)$ to be equal to 0



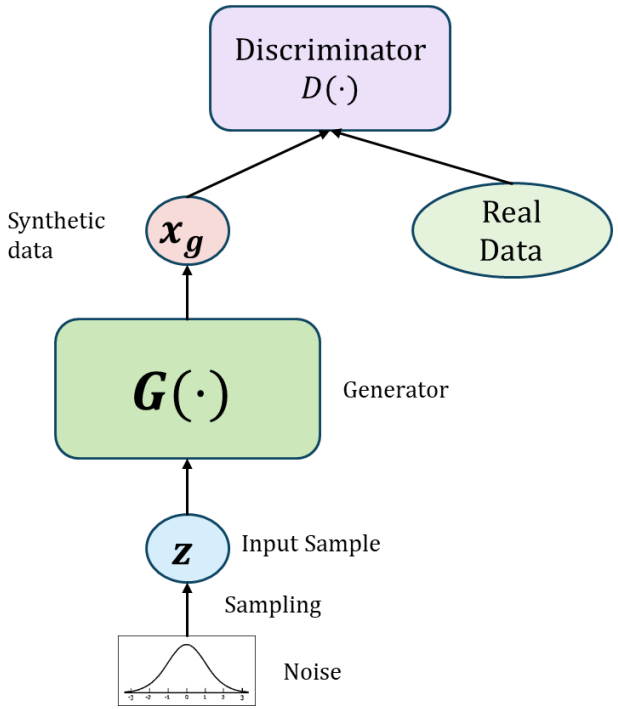
The Generator

- $\boldsymbol{L_{ce} = -y_g \log D(x_g) - (1 - y_g) \log(1 - D(x_g))}$
  - $\boldsymbol{y_g}$ **is the ground truth label of** $\boldsymbol{x_g}$
  - **So,** $\boldsymbol{y_g = 0}$

- $\boldsymbol{L_{ce} = -\log(1 - D(x_g))}$
- The generator wants $D(x_g)$ to be equal to 1
- The discriminator wants $D(x_g)$ to be equal to 0
- Since the generator tries to fool the discriminator
  - The generator wants $\boldsymbol{L_{ce}}$ **to be maximized**



The Generator

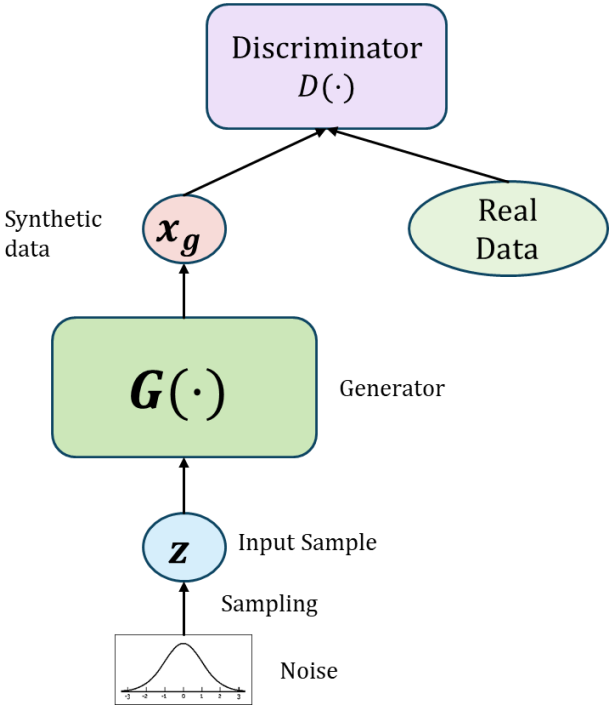


- So, the generator wants $\boldsymbol{L_{ce} = -\log(1 - D(x_g))}$ **to be maximized**
- So, the generator wants $\boldsymbol{L_{ce} = -\log(1 - D(G(z)))}$ **to be maximized**

The Generator

- So, the generator wants $\boldsymbol{L_{ce}} = \boldsymbol{-\log(1 - D(G(z)))}$ **to be maximized**

- **Equivalently, we want** $\boldsymbol{L_G = -\log(D(G(z)))}$ **to be minimized**



The Generator

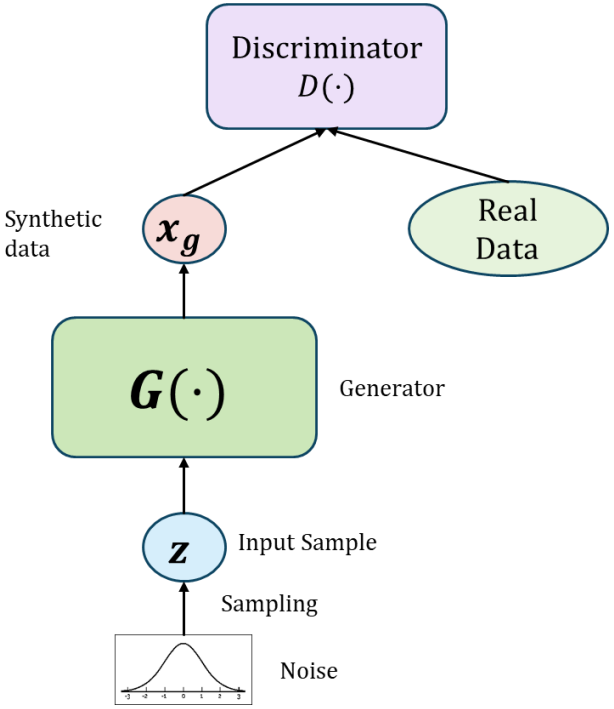
- In generator, we want to maximize

$$-\log(1 - D(G(z)))$$

- Considering all the synthetic data, we want to maximize

$$-\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

p_z is the distribution of noise



The Discriminator

- In discriminator, we want to minimize

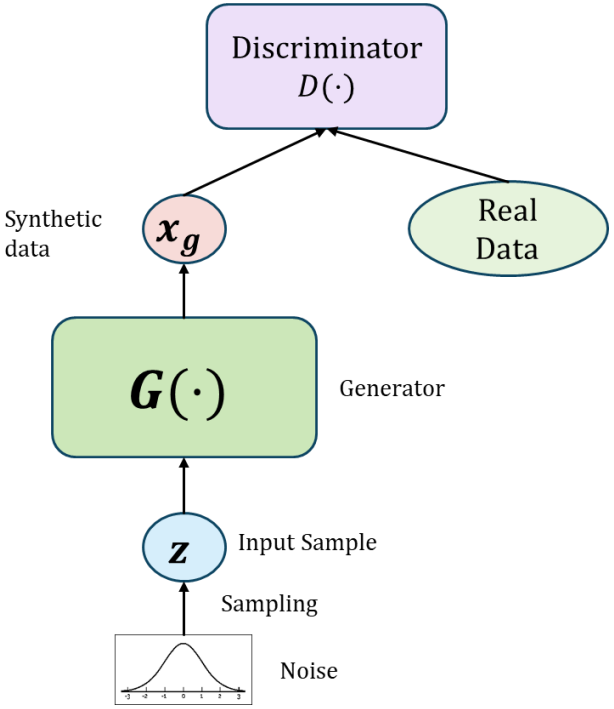
  $-y_x \log D(x) - (1 - y_x) \log(1 - D(x))$

  - y_x is the ground truth class label for x

- For real samples, $y_x = 1$

- So in the discriminator, we want to minimize

$$-\log D(x)$$



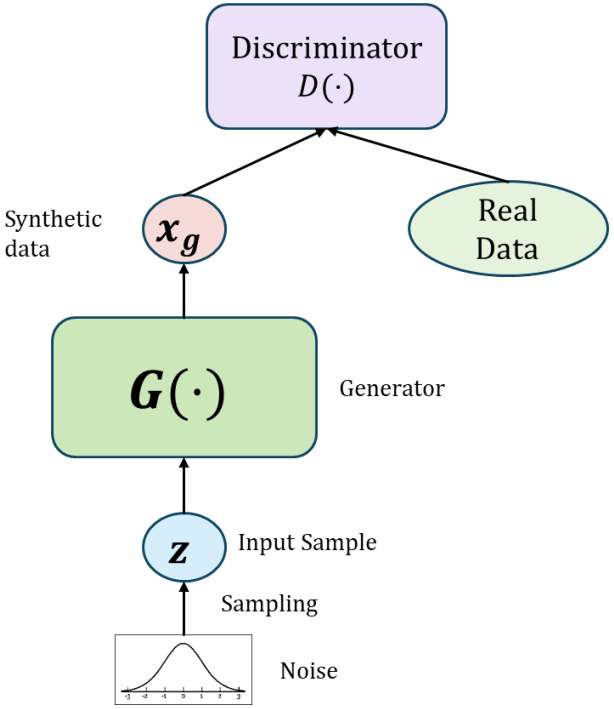
The Discriminator

- Considering all real samples

- In the discriminator, we want to minimize

$$-\mathbb{E}_{x\sim p_D} \log D(x)$$

p_D is the distribution of real data



The Generator & Discriminator

- Considering all real samples, in the discriminator, we want to minimize

$$-\mathbb{E}_{x\sim p_D} \log D(x)$$

p_D is the distribution of real data

- w.r.t. generator, considering all the synthetic data, we want to maximize

$$-\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

p_z is the distribution of noise

The Generator & Discriminator

- Considering all real samples, in the discriminator, we want to minimize

$$-\mathbb{E}_{x\sim p_D} \log D(x)$$

- w.r.t generator, considering all the synthetic data, we want to maximize

$$-\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

p_D is the distribution of real data

p_z is the distribution of noise

$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{z\sim p_z} \log(1 - D(G(z))))$$

The Generator & Discriminator

- Considering all real samples, in the discriminator, we want to minimize

$$-\mathbb{E}_{x\sim p_D} \log D(x)$$

p_D is the distribution of real data

- w.r.t generator, considering all the synthetic data, we want to maximize

$$-\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

p_z is the distribution of noise

$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{z\sim p_z} \log(1 - D(G(z))))$$

Zero-sum game

The Generator & Discriminator

- Considering all real samples, in the discriminator, we want to minimize

$$-\mathbb{E}_{x\sim p_D} \log D(x)$$

i.e., maximize

$$\mathbb{E}_{x\sim p_D} \log D(x)$$

Considering synthetic samples, the discriminator wants to maximize

$$\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

So, overall, the discriminator wants to maximize

$$J_1 = \mathbb{E}_{x\sim p_D} \log D(x) + \mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

First objective (discriminator objective)

$\mathbb{E}$ indicates that we maximize over a minibatch of samples

- In generator, considering all the synthetic data, we want to maximize

$$-\mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

i.e. minimize

$$J_2 = \mathbb{E}_{z\sim p_z} \log(1 - D(G(z)))$$

Generator objective

Training the GAN

- What if I fully train the discriminator first and then train the generator
- Initial generated samples are of poor quality
- Discriminator easily identifies all of them to be fake, $D\big(G(z)\big) = 0$

$$J_2 = \mathbb{E}_{z \sim p_z} \log(1 - D(G(z)))$$

- Generator gets no gradient to train

Training the GAN: Solution

- Alternate between the generator and the discriminator

Training the GAN: Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training the GAN: Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

Discriminator Objective

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training the GAN: Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

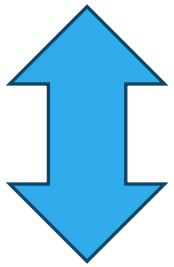
Generator Objective

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

The Generator & Discriminator

$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{z\sim p_z} \log(1 - D(G(z))))$$

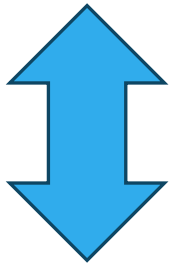


$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{x\sim p_z} \log(1 - D(G(x))))$$

Equivalent expression with change of variable

The Generator & Discriminator

$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{z\sim p_z} \log(1 - D(G(z))))$$

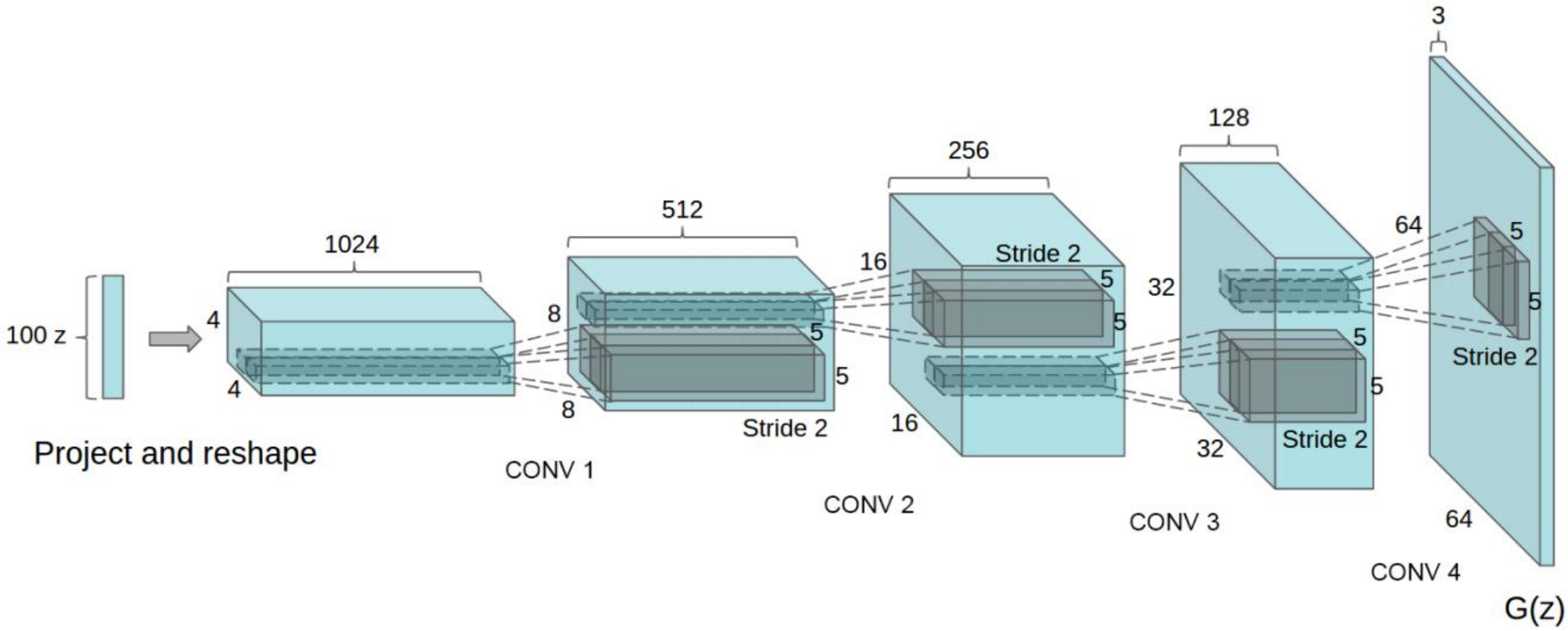


$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x) - \mathbb{E}_{x\sim p_z} \log(1 - D(G(x))))$$

This x indicates a sample from real data distribution

This x indicates a sample from noise distribution

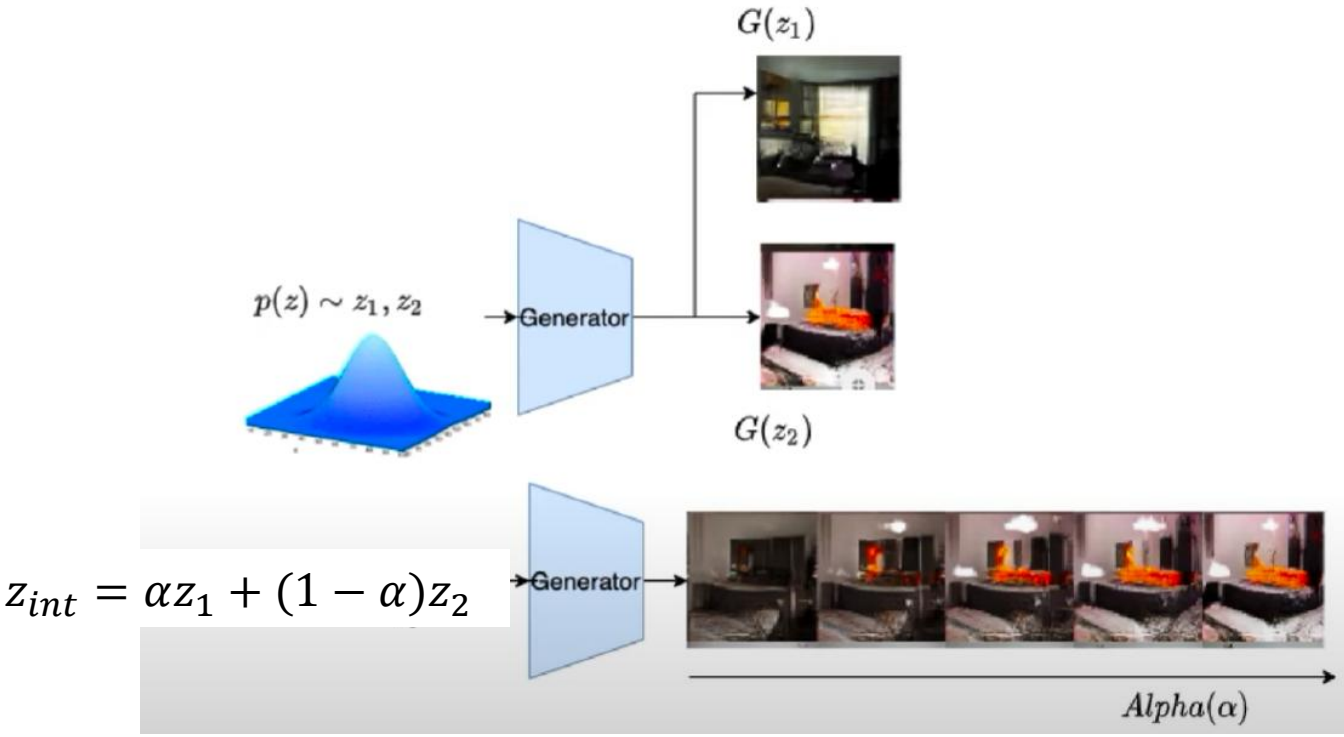
Deep Convolutional GAN



Generator for DC GAN

Deep Convolutional GAN

- Suppose two samples from the noise are z_1 and z_2
- We can interpolate between z_1 and z_2
- If we take $z_{int} = \alpha z_1 + (1 - \alpha) z_2$
  - We can get different smoothly changing outputs by inputting z_{int} with changing values of α



https://www.youtube.com/watch?v=X3SJ2mRodF0&list=PLyqSpQzTE6M_PI-rIz4O1jEgffhJU9GgG&index=64

Deep Convolutional GAN

Generator for DC GAN

https://carpedm20.github.io/faces/

How to Evaluate Generative Models

How to Evaluate Generative Models

- Human Judgement
  - Recognizable objects
  - Semantic diversity
- Prediction power: How a pre-trained CNN (e.g., Inception Net V3) perform on the generated images
- Popular performance metric: Inception score, Fréchet inception distance
- Still an open area of discussion and research

Inception Score (IS)

- Suppose, we use an InceptionV3 model to classify our generated images
  - $p(y|x)$ is the predicted label class probabilities generated image x is applied to InceptionV3
  - $p(y)$ is the distribution of the predicted class labels by InceptionV3
    - Empirical estimation of $p(y)$ is

$$\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_i)$$

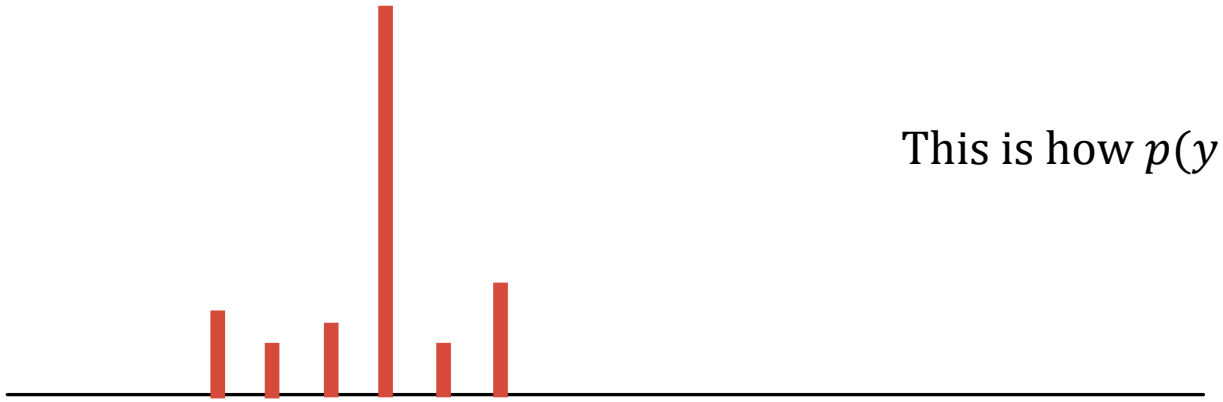
https://arxiv.org/pdf/1801.01973

Inception Score (IS)

- What do we want about the generated images
  - InceptionV3 should predict only one class label with high probability given a generated image x
  - All other class labels should be predicted with low probabilities

Inception Score (IS)

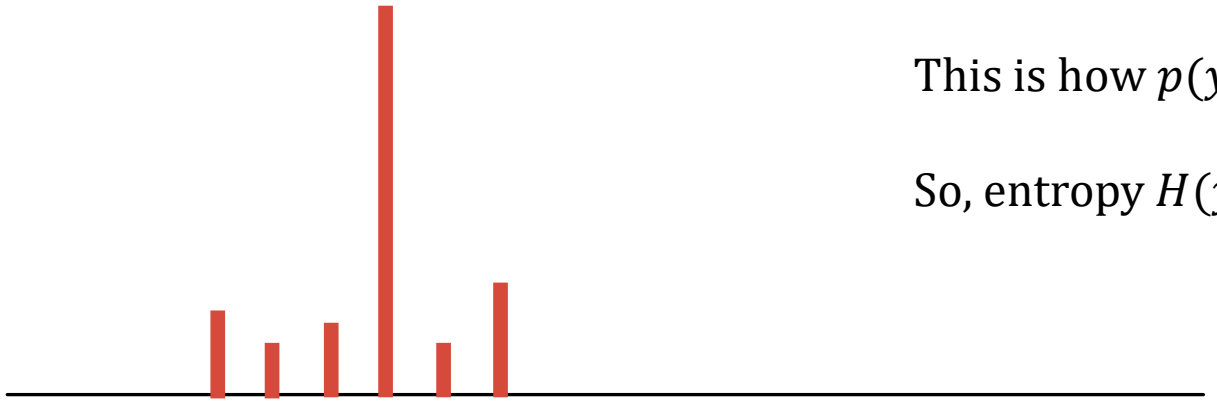
- What do we want about the generated images
  - InceptionV3 should predict only one class label with high probability given a generated image x
  - All other class labels should be predicted with low probabilities



This is how $p(y|x)$ should look like

Inception Score (IS)

- What do we want about the generated images
  - InceptionV3 should predict only one class label with high probability given a generated image x
  - All other class labels should be predicted with low probabilities



This is how $p(y|x)$ should look like

So, entropy $H(y|x)$ should be low

Inception Score (IS)

- What do we want about the generated images
  - The generated images should be diverse

Inception Score (IS)

- What do we want about the generated images
  - The generated images should be diverse
  - So the distribution of the class labels predicted by InceptionV3 should be close to uniform

This is how $p(y)$ should look like

Inception Score (IS)

- What do we want about the generated images
  - The generated images should be diverse
  - So the distribution of the class labels predicted by InceptionV3 should be close to uniform

This is how $p(y)$ should look like

So, entropy $H(y)$ should be high

Inception Score (IS)

- $IS = exp\{H(y) - H(y|x)\}$

- Higher IS indicates better generator

Inception Score (IS): Limitation

- It does not consider the distribution of the real data at all

Fréchet Inception Distance

- Intuition: Calculate the distance between the features of real data and generated data

- How to get the features?
  - Pass the real data and the generated data through an InceptionV3 (pre-trained)
  - Take the features from the Pool3 layer

- Calculate Fréchet distance between the features of the real data and generated data

Fréchet Inception Distance

- Fréchet distance

$$FID = \left\| m_r - m_g \right\|_2 + Tr\left(C_r + C_g - 2\left(C_r C_g \right)^{\frac{1}{2}} \right)$$

m_r, C_r: mean and covariance matrix of the multi-dimensional features from real data

m_g, C_g: mean and covariance matrix of the multi-dimensional features from generated data

$Tr(\cdot)$: Trace of matrix

Fréchet Inception Distance

- Fréchet distance

$$FID = \|m_r - m_g\|_2 + Tr\left(C_r + C_g - 2(C_r C_g)^{\frac{1}{2}}\right)$$

- If the generated data is more realistic, m_g and m_r would have closer values and C_g and C_r would have closer values
  - So, FID will get close to 0

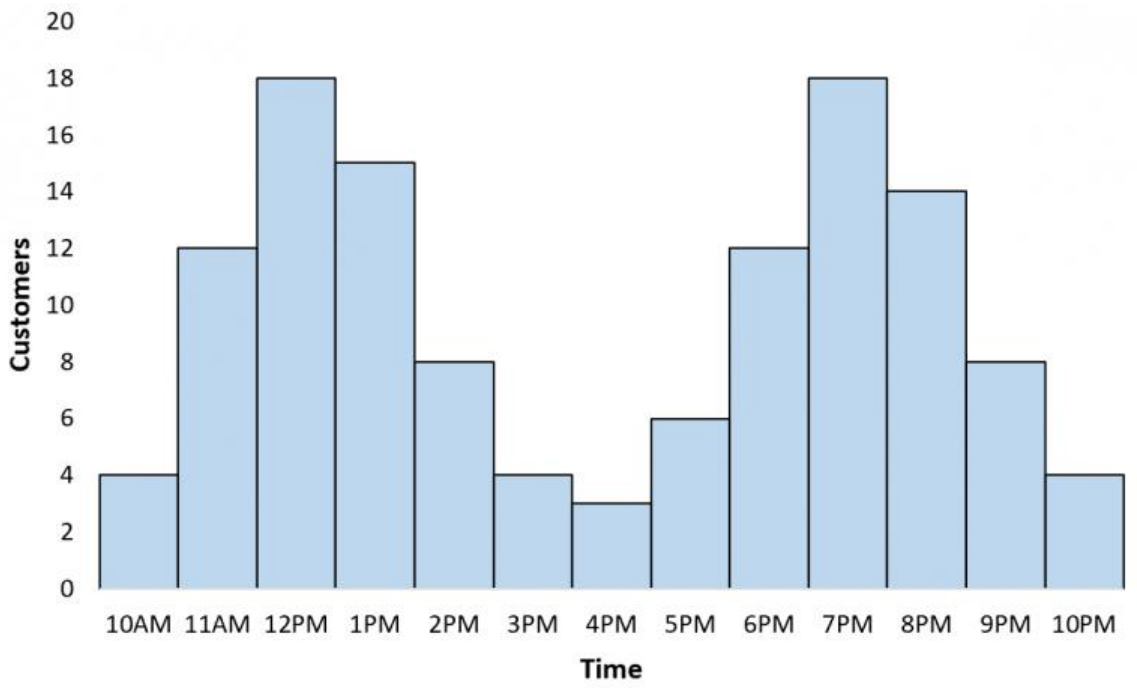
- So, lower FID indicates better generator

Challenges in GAN

- GAN generates data
- Generation (creativity) is more challenging compared to discrimination
  - Identifying an a Mozart music is much easier compared to composing one

Mode in Data

- Real life data is multi modal
- An example: number of customers in a restaurant
- Mode: the value that occurs most frequently in the distribution



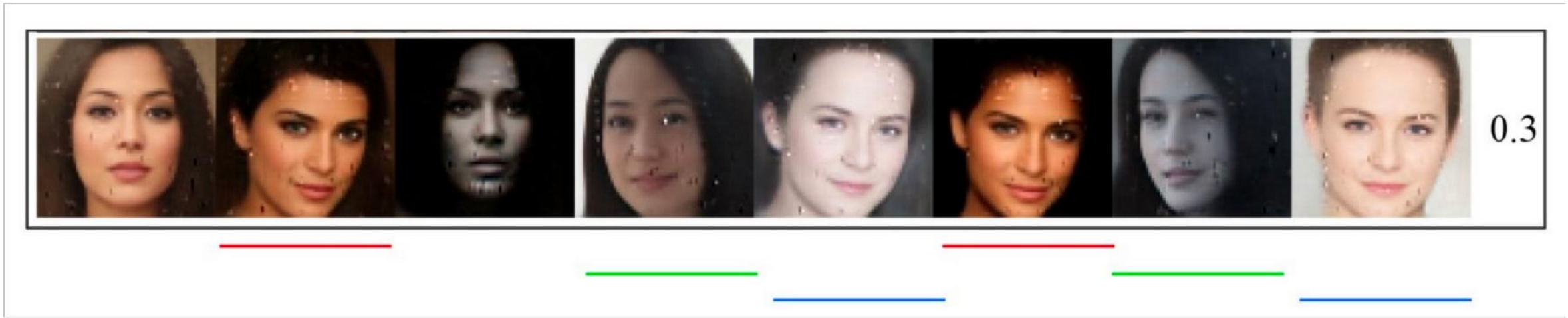
https://www.statology.org/multimodal-distribution/#:~:text=Multimodal%20distributions%20can%20occur%20when,placed%20on%20the%20same%20histogram.

Mode Collapse

- Generator tends to generate data that has value close to the mode
- Limited variety in samples
- Complete collapse (unlikely)
- Partial collapse (more common)

Mode Collapse

- Generator tends to generate data that has value close to the mode
- Limited variety in samples
- Complete collapse (unlikely)
- Partial collapse (more common)



Why Does Mode Collapse Happen?

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

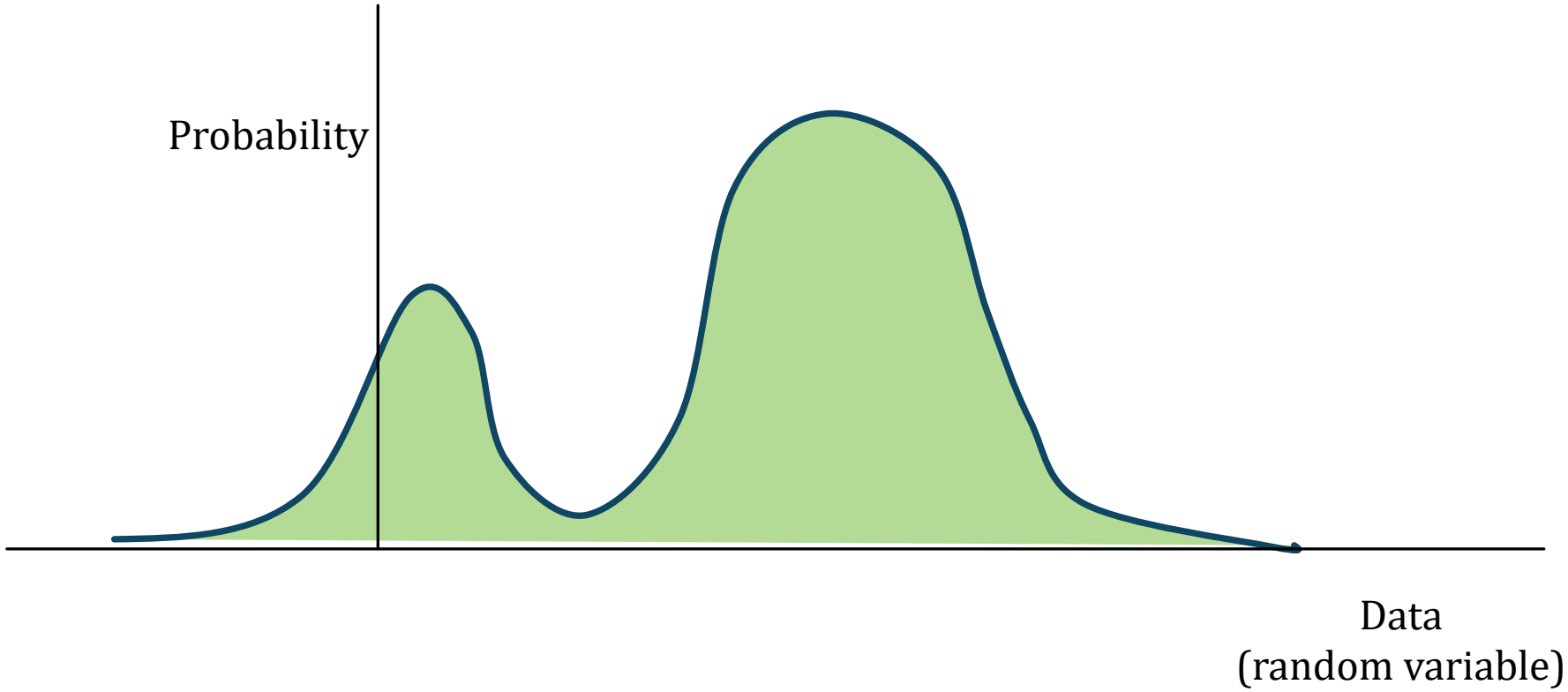
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

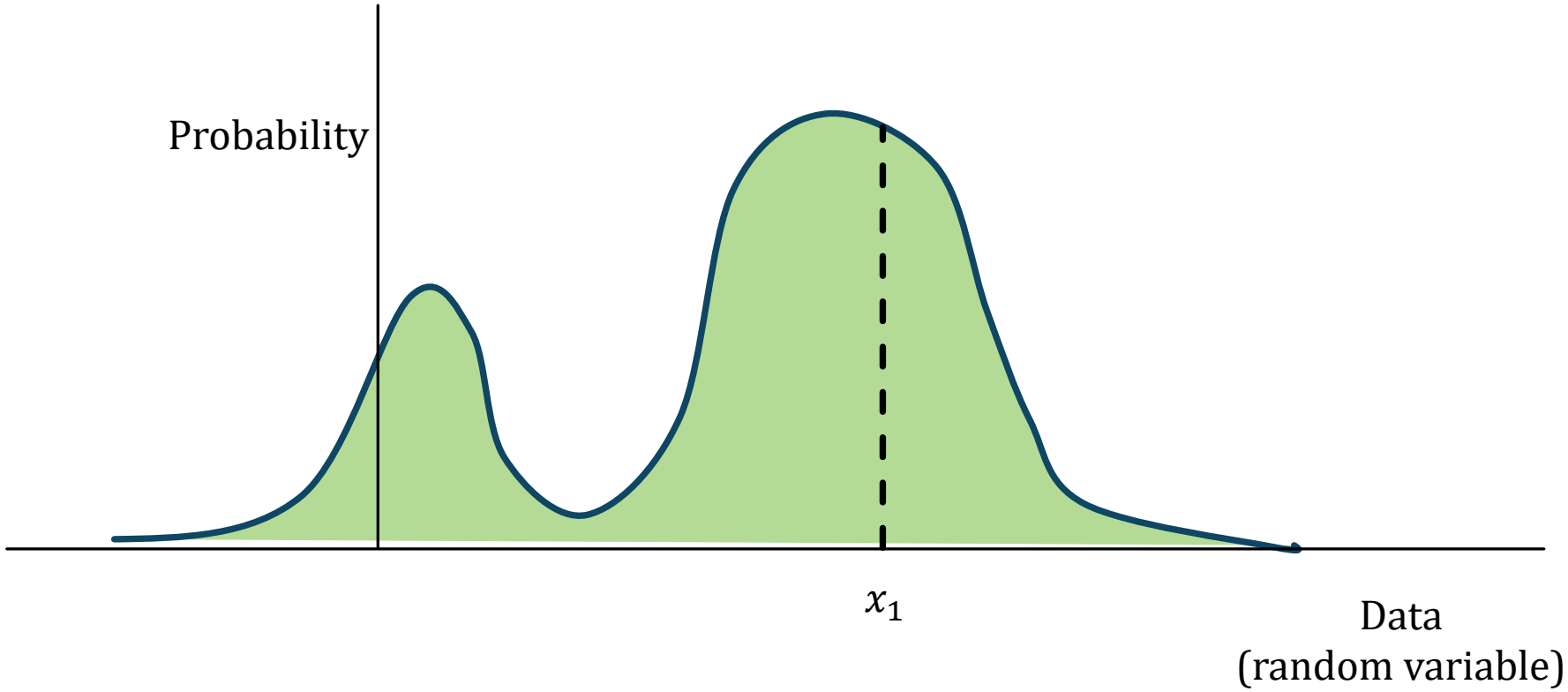
Why Does Mode Collapse Happen?

- Generator tends to generate data that fooled the discriminator the most
  - Data close to a mode; because such data is more realistic



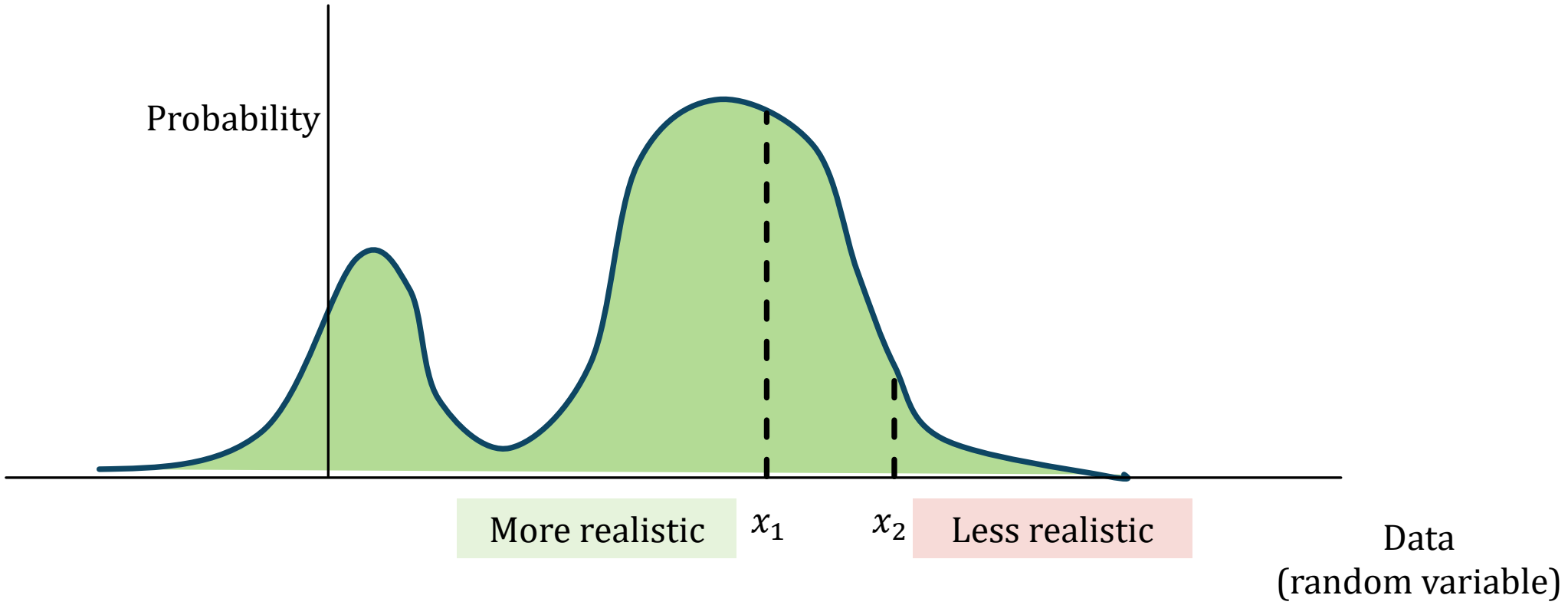
Why Does Mode Collapse Happen?

- Generator tends to generate data that fooled the discriminator the most
  - Data close to a mode; because such data is more realistic



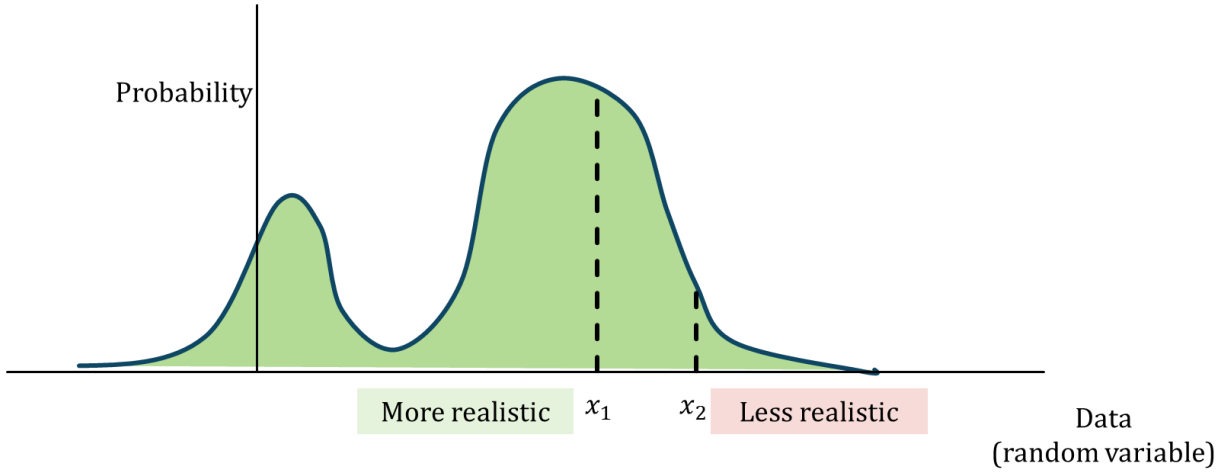
Why Does Mode Collapse Happen?

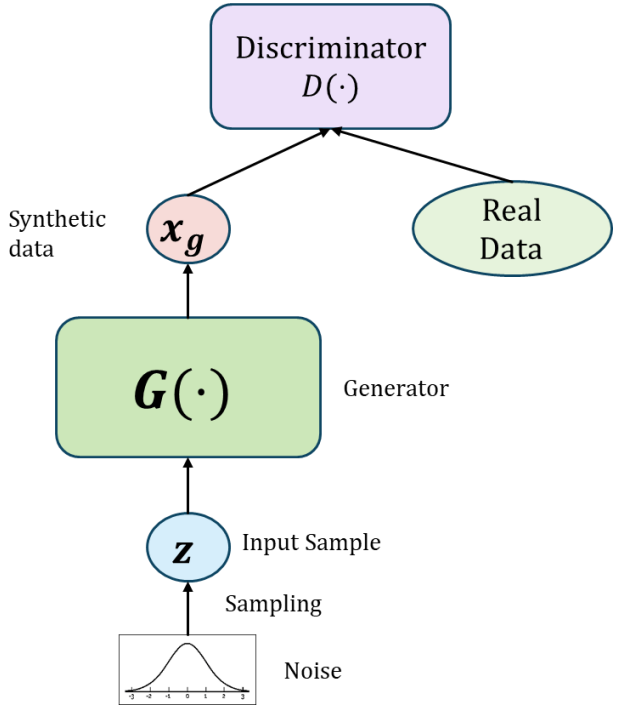
- Generator tends to generate data that fooled the discriminator the most
  - Data close to a mode; because such data is more realistic



Why Does Mode Collapse Happen?

- Generator tends to generate data that fooled the discriminator the most
  - Data close to a mode; because such data is more realistic

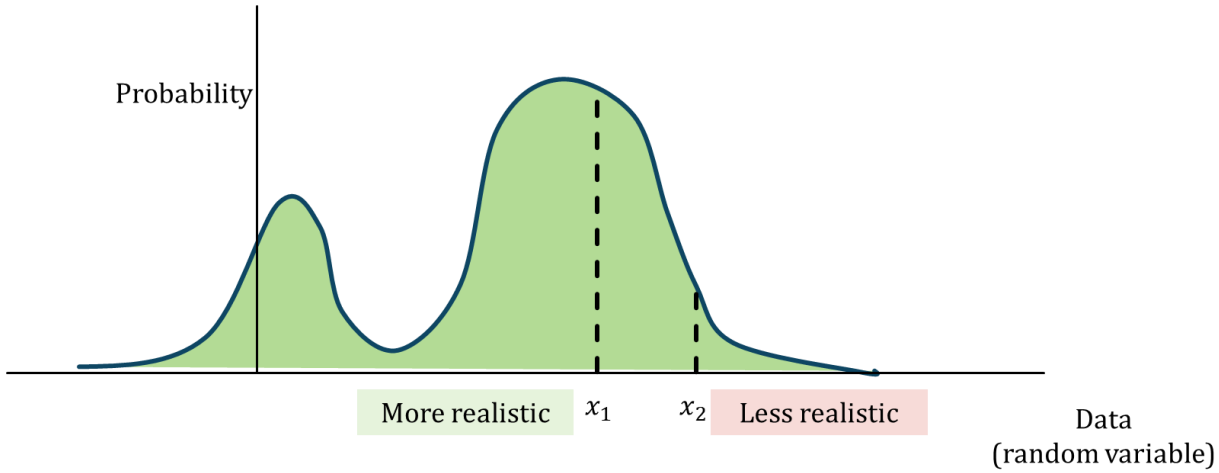


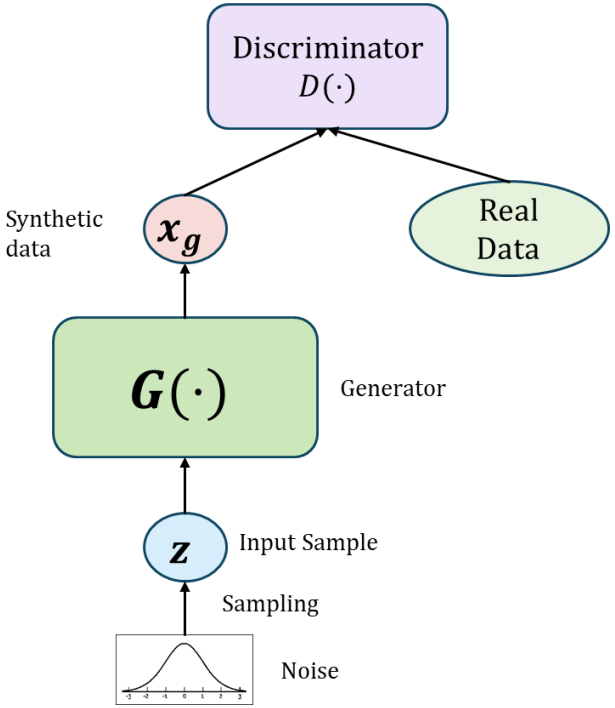


- At the extreme situation generated data x_g becomes independent of z
  - Training of generator kind of stops

Why Does Mode Collapse Happen?

- Generator tends to generate data that fooled the discriminator the most
  - Data close to a mode; because such data is more realistic

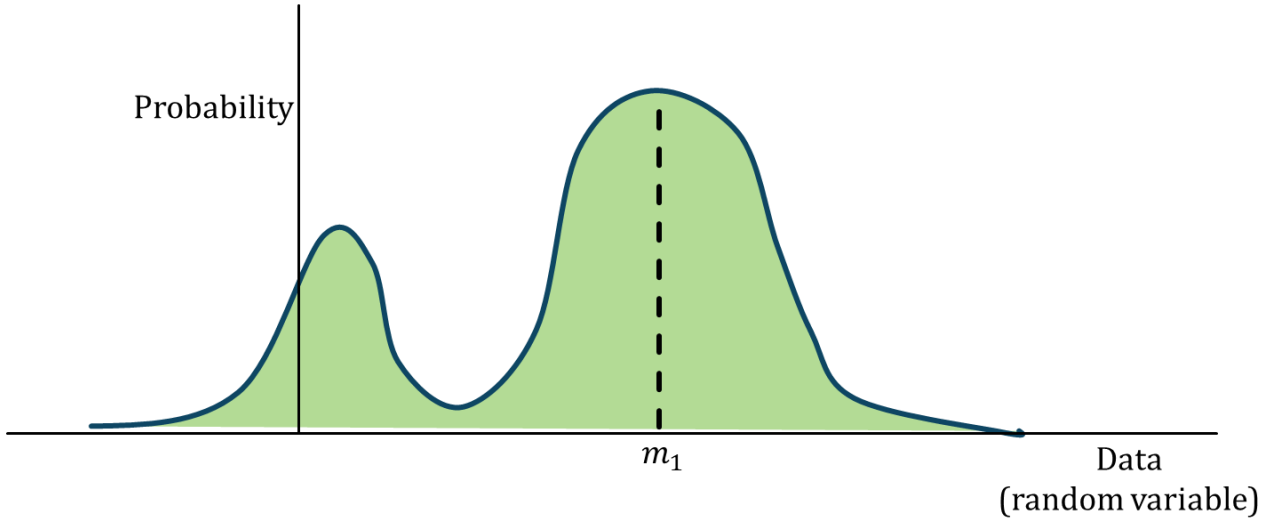


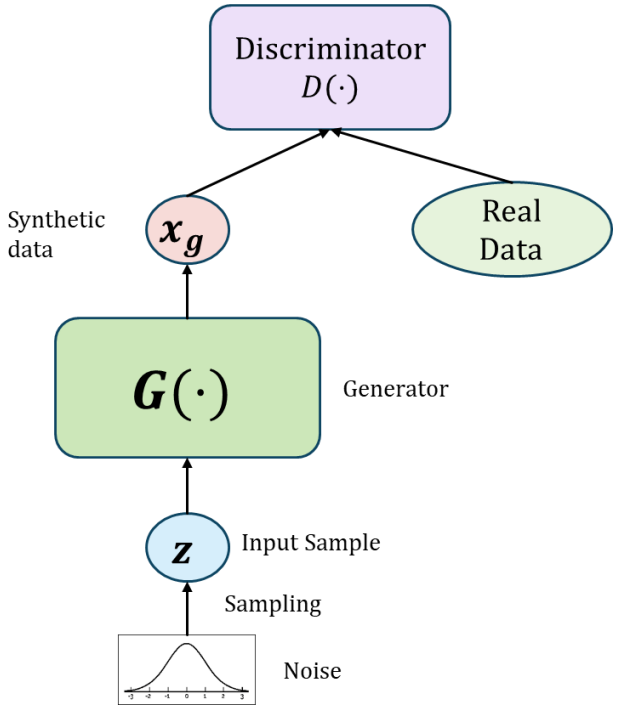


- At the extreme situation generated data x_g becomes independent of z
  - Training of generator kind of stops
  - Most generated data looks similar to x_1

Why Does Mode Collapse Happen?

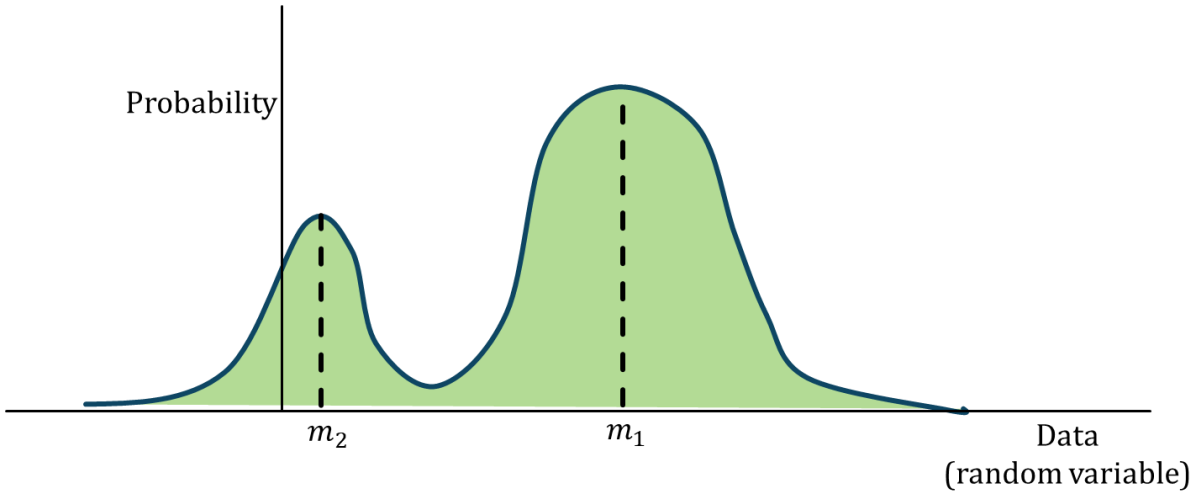
- When we restart the training of discriminator D, the most effective way to catch the generated data is to catch all the data around mode m_1

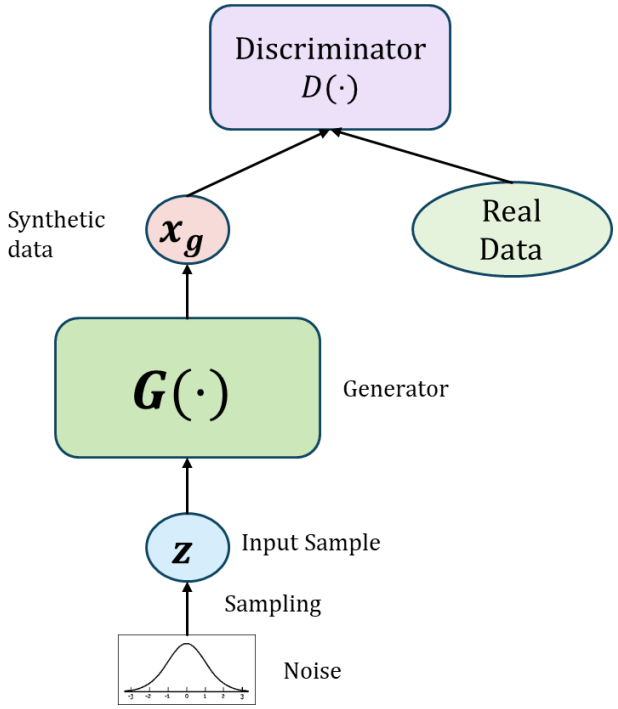




Why Does Mode Collapse Happen?

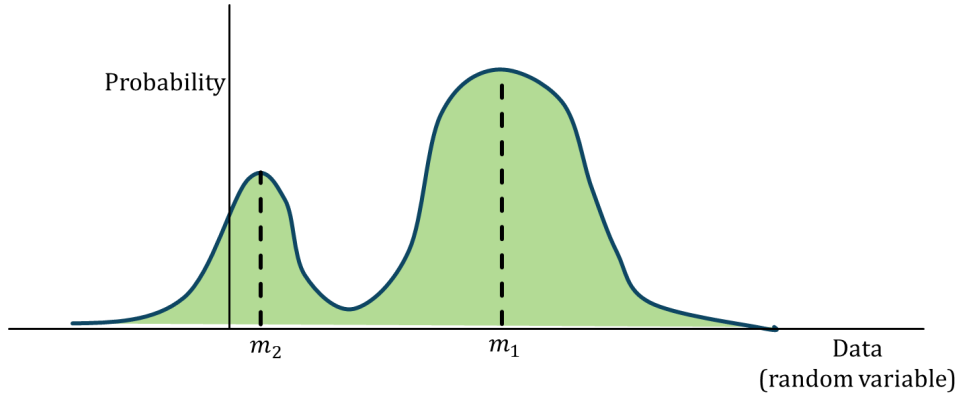
- Then, the gradient from discriminator is likely to push the generated data towards another mode (the next most likely data)



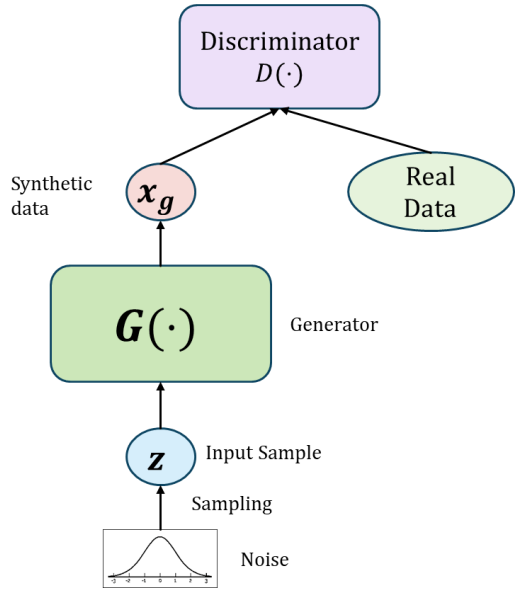


Why Does Mode Collapse Happen?

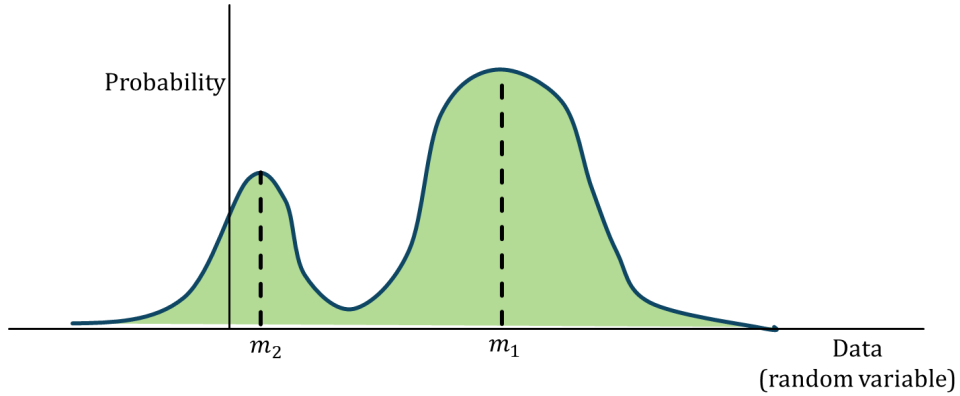
- Then, the gradient from discriminator is likely to push the generated data towards another mode (the next most likely data)



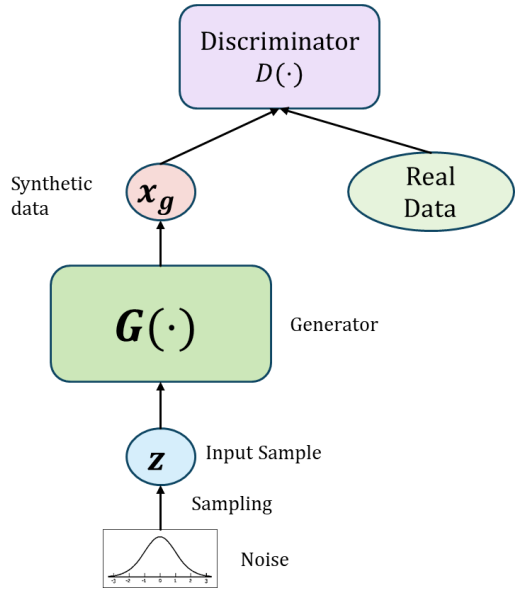
- At this point, both the generator and discriminator are overfitted and they do not converge
  - Both opponents (G and D) exploits the short-term weakness of the other and overfit
- This results in mode collapse



Why Does Mode Collapse Happen?

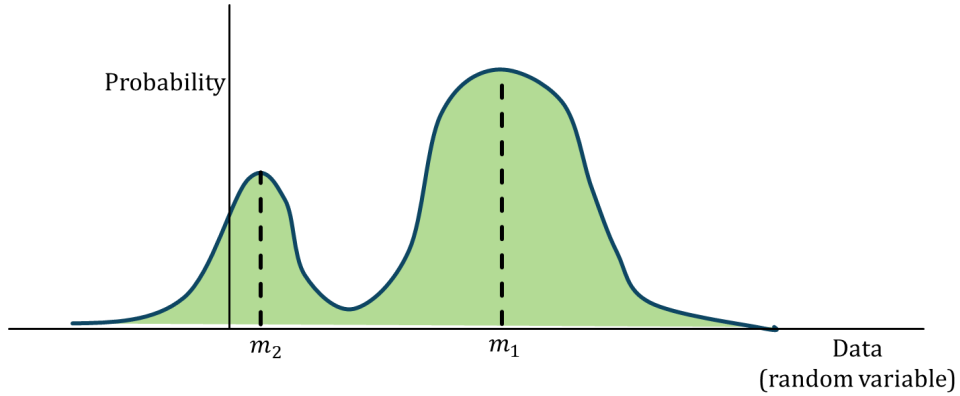


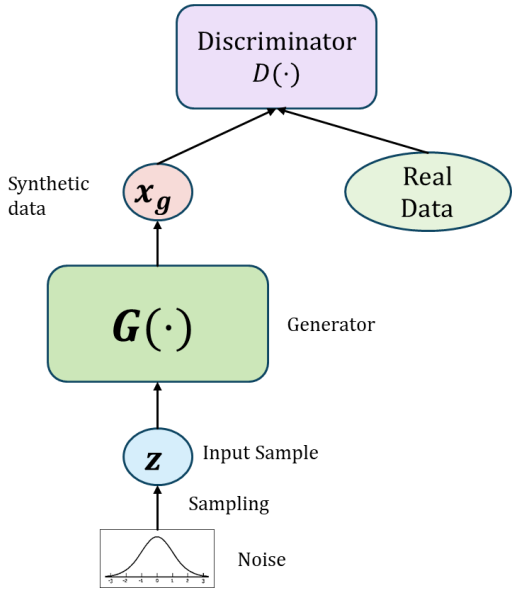
- At this point, discriminator keeps on failing
- Generator is happy to generate data around mode m_2



Why Does Mode Collapse Happen?

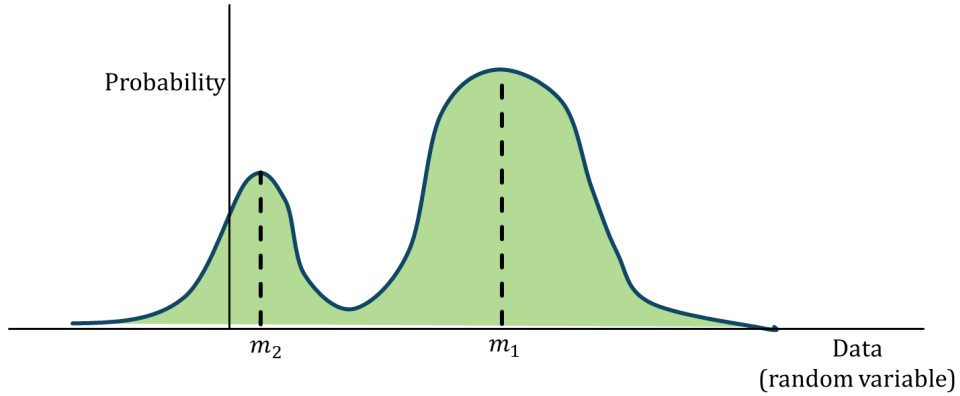
- At this point, discriminator keeps on failing
  - Discriminator loss increases
- Generator is happy to generate data around mode m_2
  - Generator loss does not change

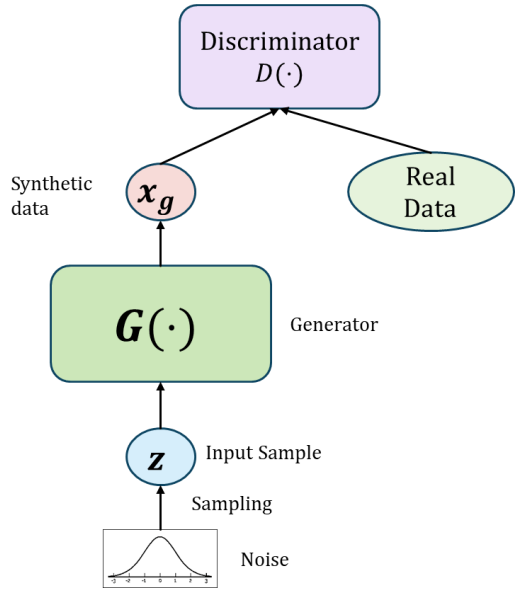




Why Does Mode Collapse Happen?

- At this point, discriminator keeps on failing
  - Discriminator loss increases
- Generator is happy to generate data around mode m_2
  - Generator loss does not change
- Nash equilibrium
  - Action of the one player (failing of discriminator) does not affect the other player (generator)
- Mode collapse





Mode Collapse: A Practical Example

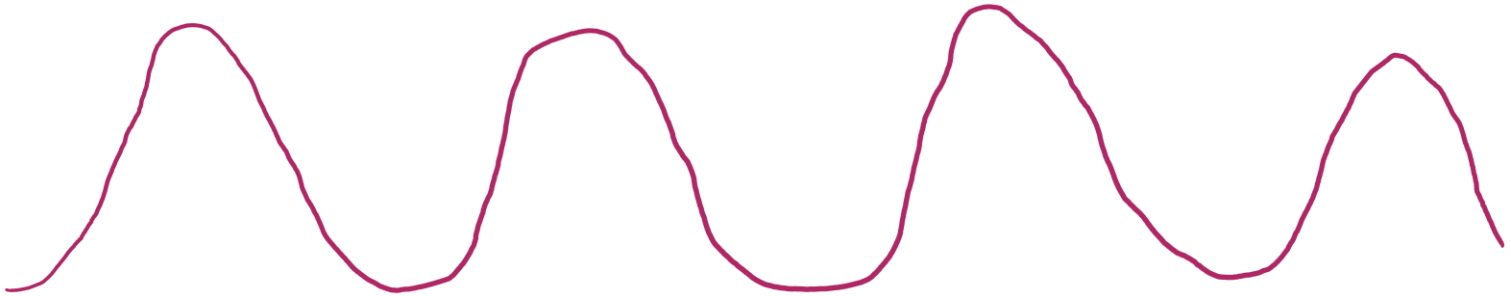
- There was a music director
  - Let's say his name is HR
- Once he composed some music with a specific pattern and it got popularity
- He continued producing similar music and got popularity for some time (fooled the discriminator)
  - He lacked artistic exploration (produced data close to the mode)
  - And gradually lost the capability to produce other music
- After some time, audience got bored
- But because HR lost the capability, he kept on producing same music

Mode Collapse: A Practical Example

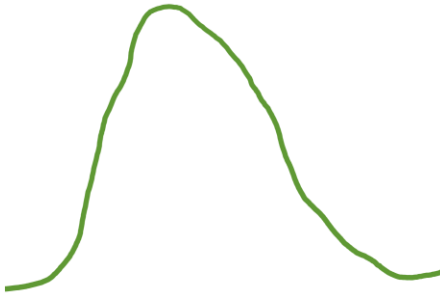
- A generative model for generation of handwritten digits
  - Real data distribution is Decamodal (10 modes)
- Suppose, the generator generates synthetic images of digit 2 with an excellent quality
  - It does not generate the images of any other digit
- Since the generated digit 2 is very good, the discriminator loss is very low
  - So, generator does not update
- However, the discriminator could not catch that the generator is not generating the other digits

Mode Collapse: A Practical Example

Desired Data Distribution

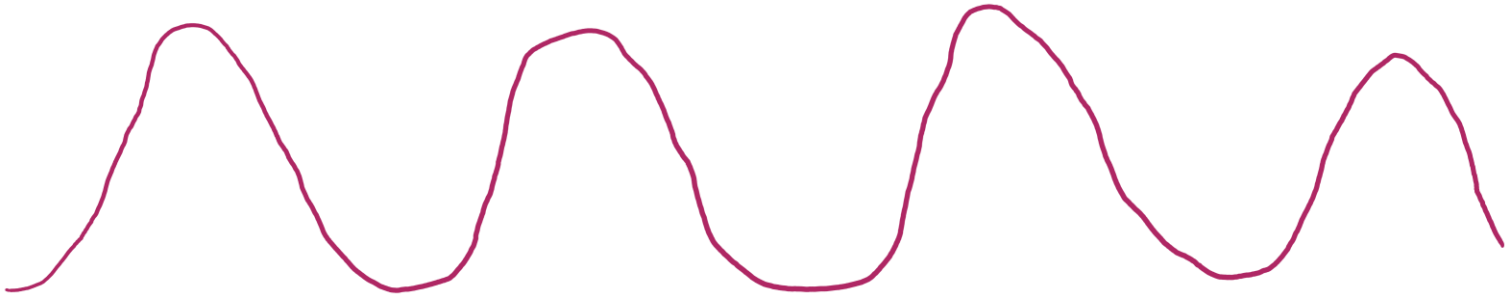


Distribution Generated by the Generator



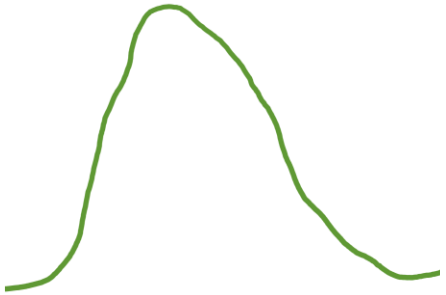
Mode Collapse: A Practical Example

Desired Data Distribution



To address the issue of mode collapse, we have to find the difference (distance) between the red and green distributions

Distribution Generated by the Generator



Need for Earth Mover's Distance

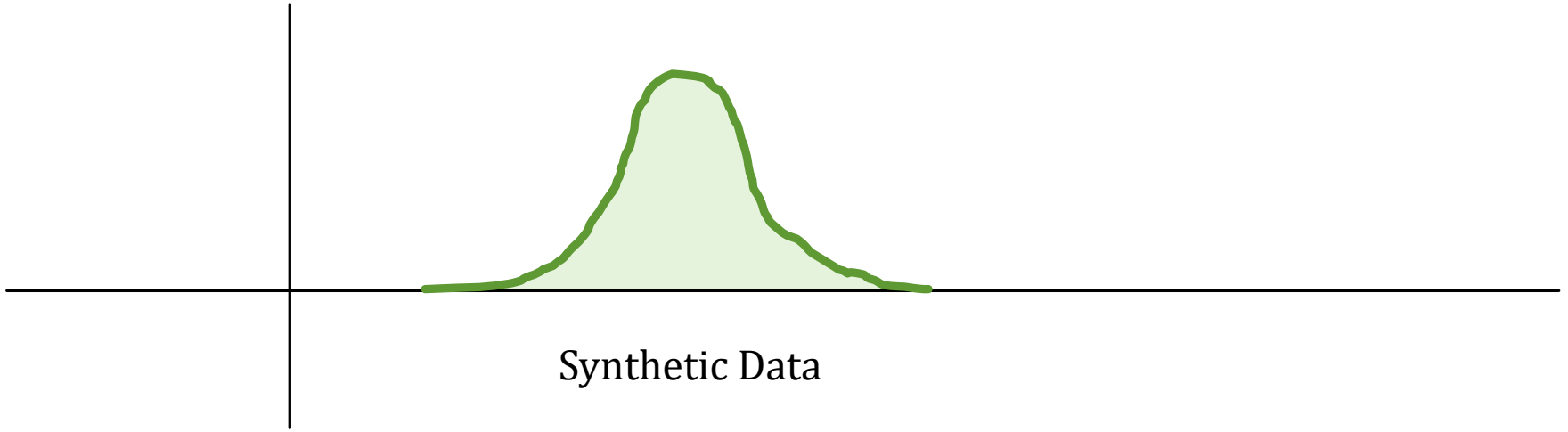
Real Data

Synthetic Data

Situation 1

Need for Earth Mover's Distance

Real Data

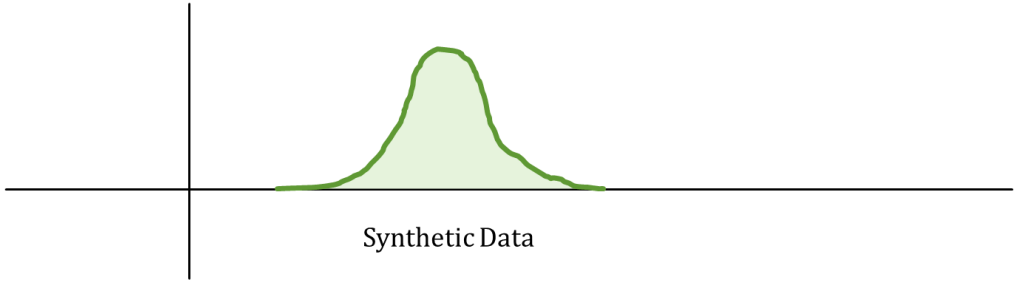


Synthetic Data

Situation 2

Need for Earth Mover's Distance

Situation 1



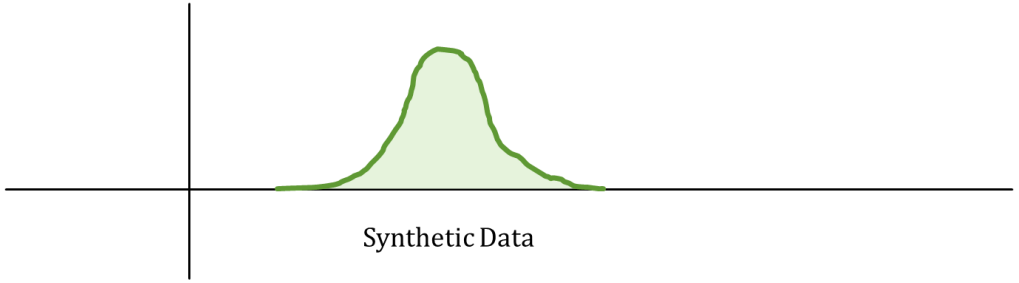
Situation 2

The discriminator of the traditional GAN will treat both the situations equally.

Are the two situations same?

Need for Earth Mover's Distance

Situation 1



Situation 2

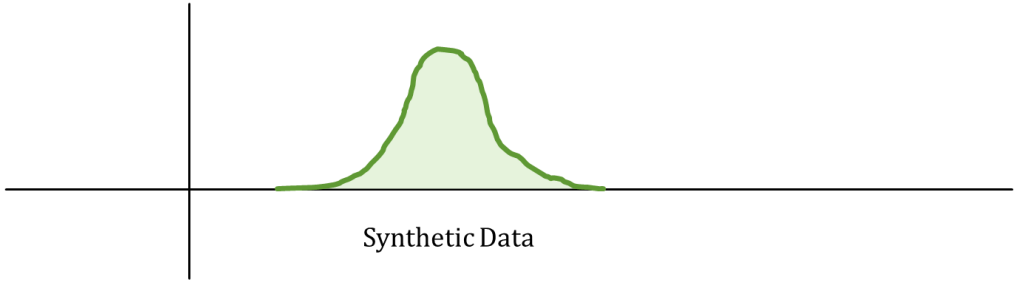
The discriminator of the traditional GAN will treat both the situations equally.

Are the two situations same?

No

Need for Earth Mover's Distance

Situation 1



Situation 2

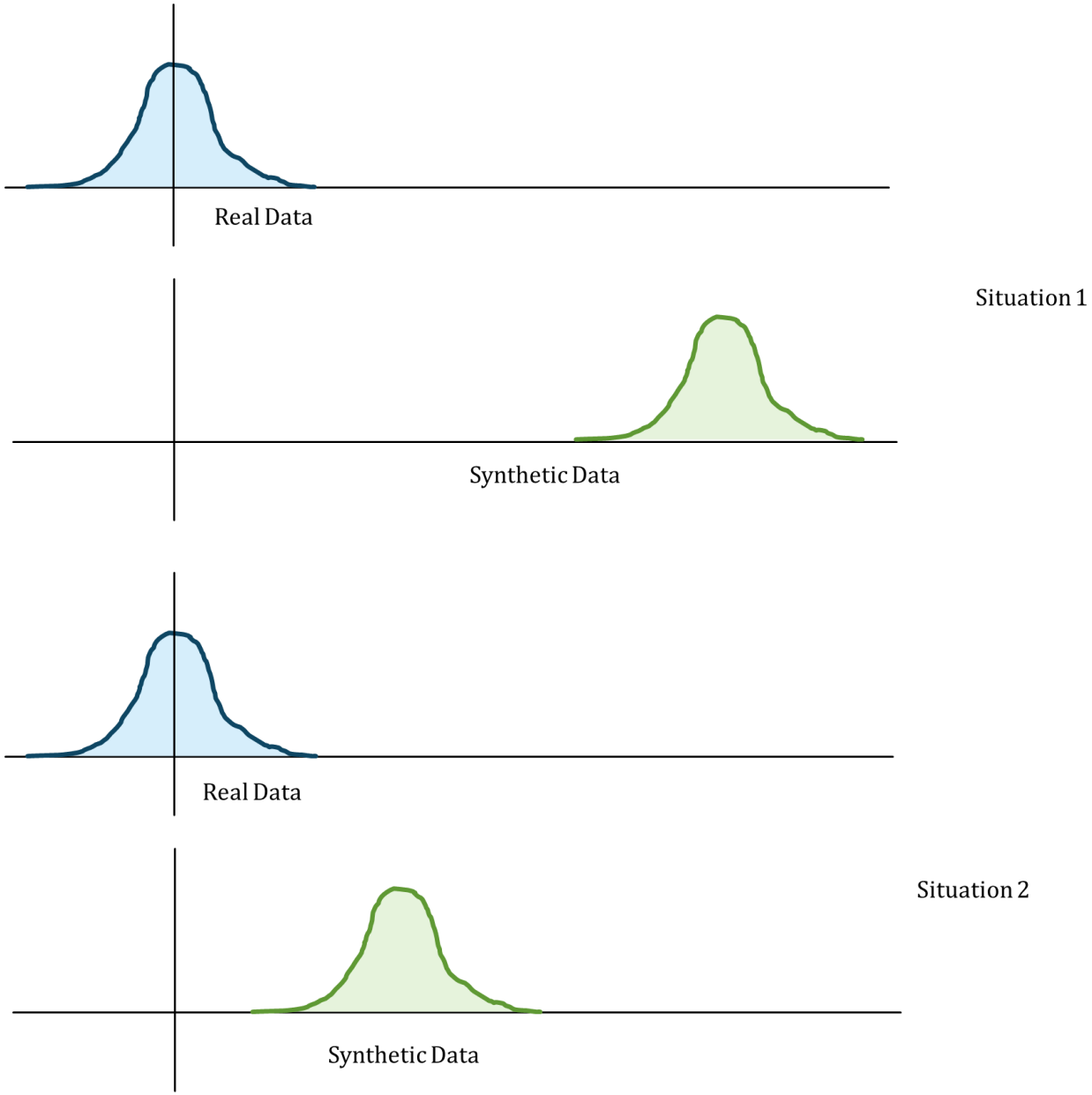
The discriminator of the traditional GAN will treat both the situations equally.

Are the two situations same?

No

How can we differentiate between the two situations?

Need for Earth Mover's Distance



The discriminator of the traditional GAN will treat both the situations equally.

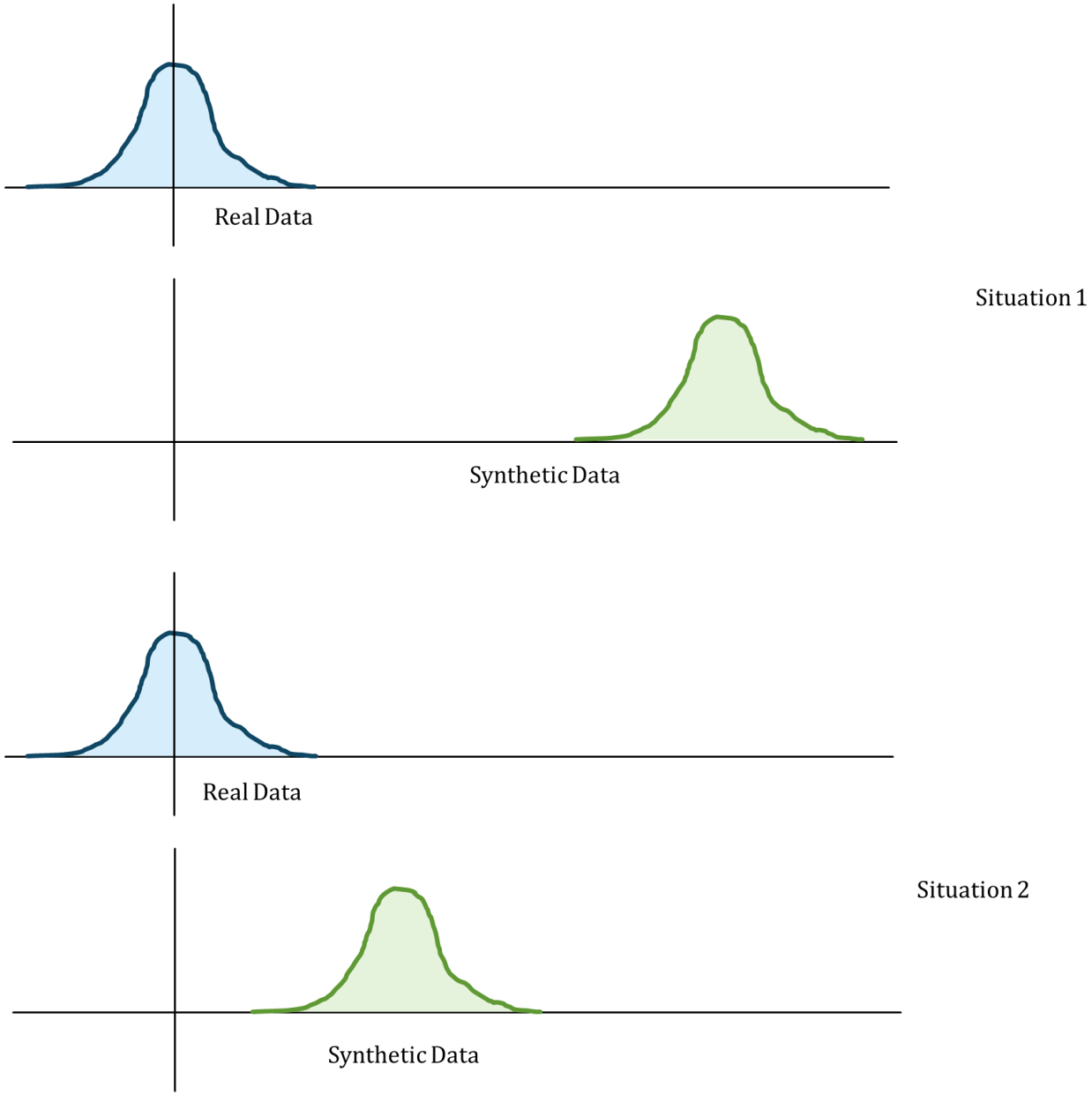
Are the two situations same?

No

How can we differentiate between the two situations?

Earth mover's distance

Need for Earth Mover's Distance

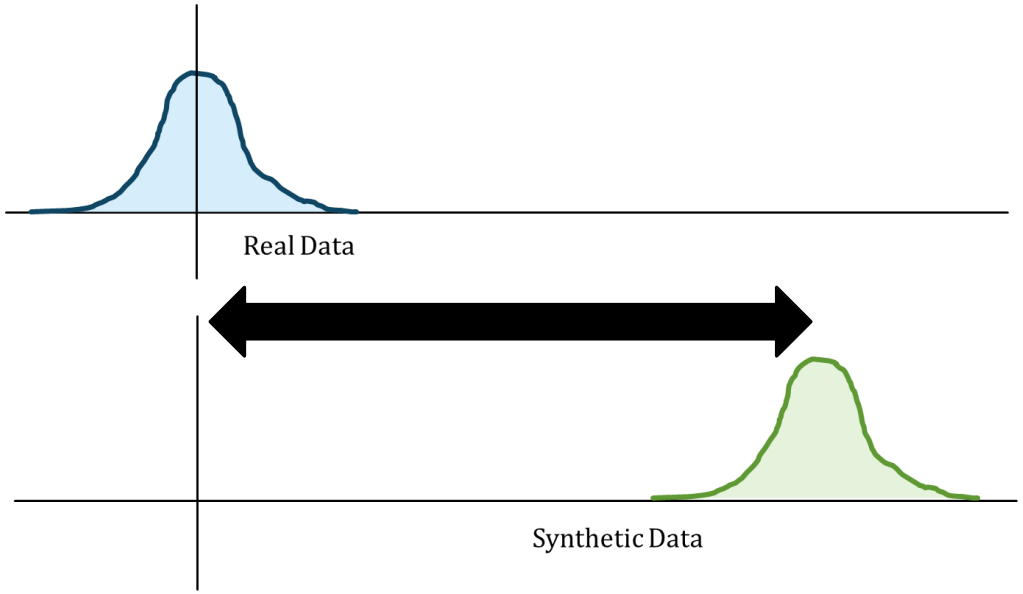


Earth mover's distance

Consider the blue pile and the green pile of stones

Earth mover's distance is the minimum effort required to move the blue pile to the position and the shape of the green pile

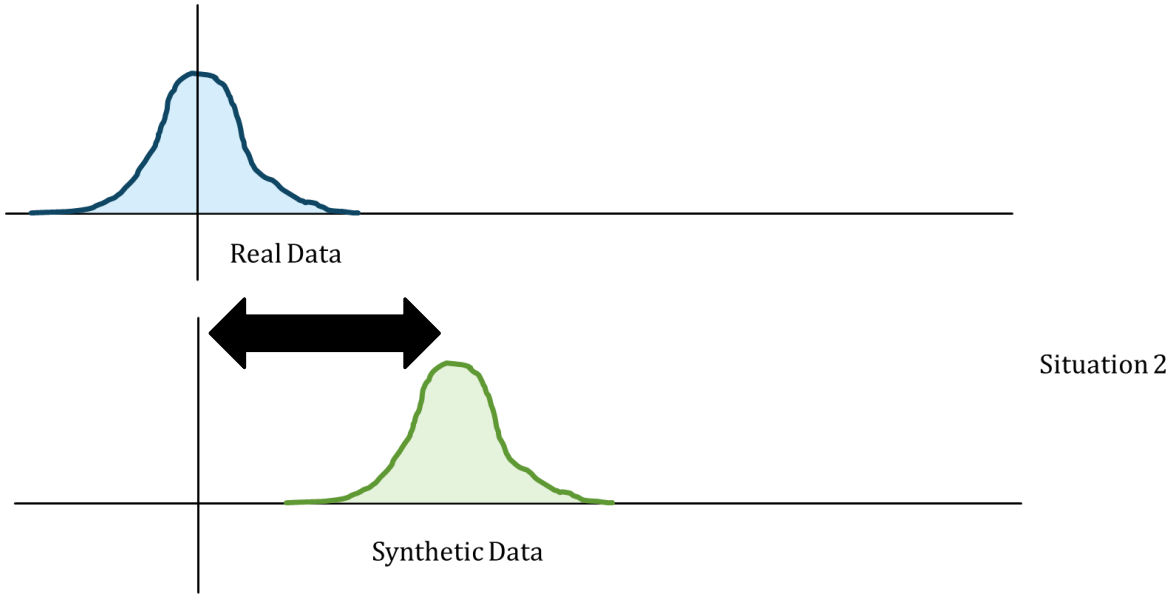
Need for Earth Mover's Distance



Situation 1

Situation 1

More effort required; more earth mover's distance



Situation 2

Situation 2

Less effort required; less earth mover's distance

Wasserstein GAN

- Uses earth mover's distance (Wasserstein distance) to train the discriminator

- Discriminator objective for GAN; maximize

$$J_1 = \mathbb{E}_{x \sim p_D} \log D(x) + \mathbb{E}_{z \sim p_z} \log(1 - D(G(z)))$$

- Discriminator objective for WGAN; minimize

$$J_1 = \mathbb{E}_{x \sim p_D}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))]$$

Wasserstein GAN

- Discriminator (critic) objective for WGAN; minimize

$$J_1 = \mathbb{E}_{x \sim p_D}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))]$$

- Clipping of the discriminator weights in $[-c, c]$

- Generator objective for WGAN; minimize

$$J_2 = \mathbb{E}_{z \sim p_z} \log(1 - D(G(z)))$$

The Lipschitz Constraint

- The Discriminator (critic) is 1-Lipschitz continuous
- Critic function D
- For any two input images x_1 and x_2

$$\frac{|D(x_1) - D(x_2)|}{|x_1 - x_2|} \leq 1$$

- $|x_1 - x_2|$: Average of pixel-wise absolute difference between x_1 and x_2
- Putting a limit on the rate at which predictions can change

The Lipschitz Constraint

- For any two input images x_1 and x_2

$$\frac{|D(x_1) - D(x_2)|}{|x_1 - x_2|} \leq 1$$

- $|x_1 - x_2|$: Average of pixel-wise absolute difference between x_1 and x_2
- Putting a limit on the rate at which predictions can change

- Enforced by weight clipping of critic in WGAN

Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Can WGAN Deal with Mode Collapse

- Critic objective for WGAN; minimize

$$J_1 = \mathbb{E}_{x \sim p_D}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))]$$

- Discriminator does not try to detect real/ fake
  - It tries to align the distribution
- So, discriminator is less likely to focus on a mode
- Hence, the generator is less likely to converge to a mode

Improving WGAN

- Weight clipping may lead to difficulties in convergence
- Don’t clip weights of critic
- Don’t use batch norm as well
- Introduce gradient penalty such that the gradients of critic weights w.r.t. the inputs remains close to 1

WGAN-GP

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

WGAN-GP

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

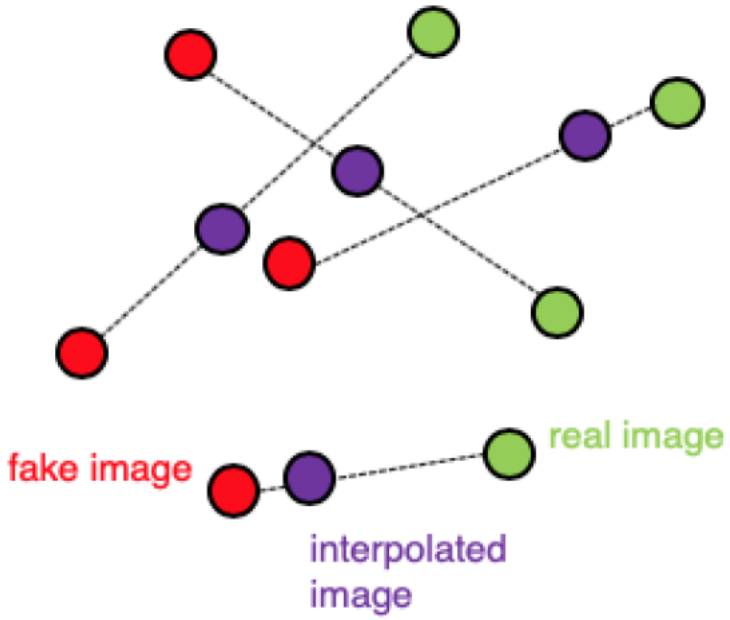
while θ has not converged **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

What is this step?

WGAN-GP

- Calculating gradient everywhere during the training process may be intractable
- So, WGAN-GP evaluates the gradient at only a handful of points
- To ensure a balanced mix, we use a set of interpolated images that lie at randomly chosen points along lines connecting the batch of real images to the batch of fake images pairwise

Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
$\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
$\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
$L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$



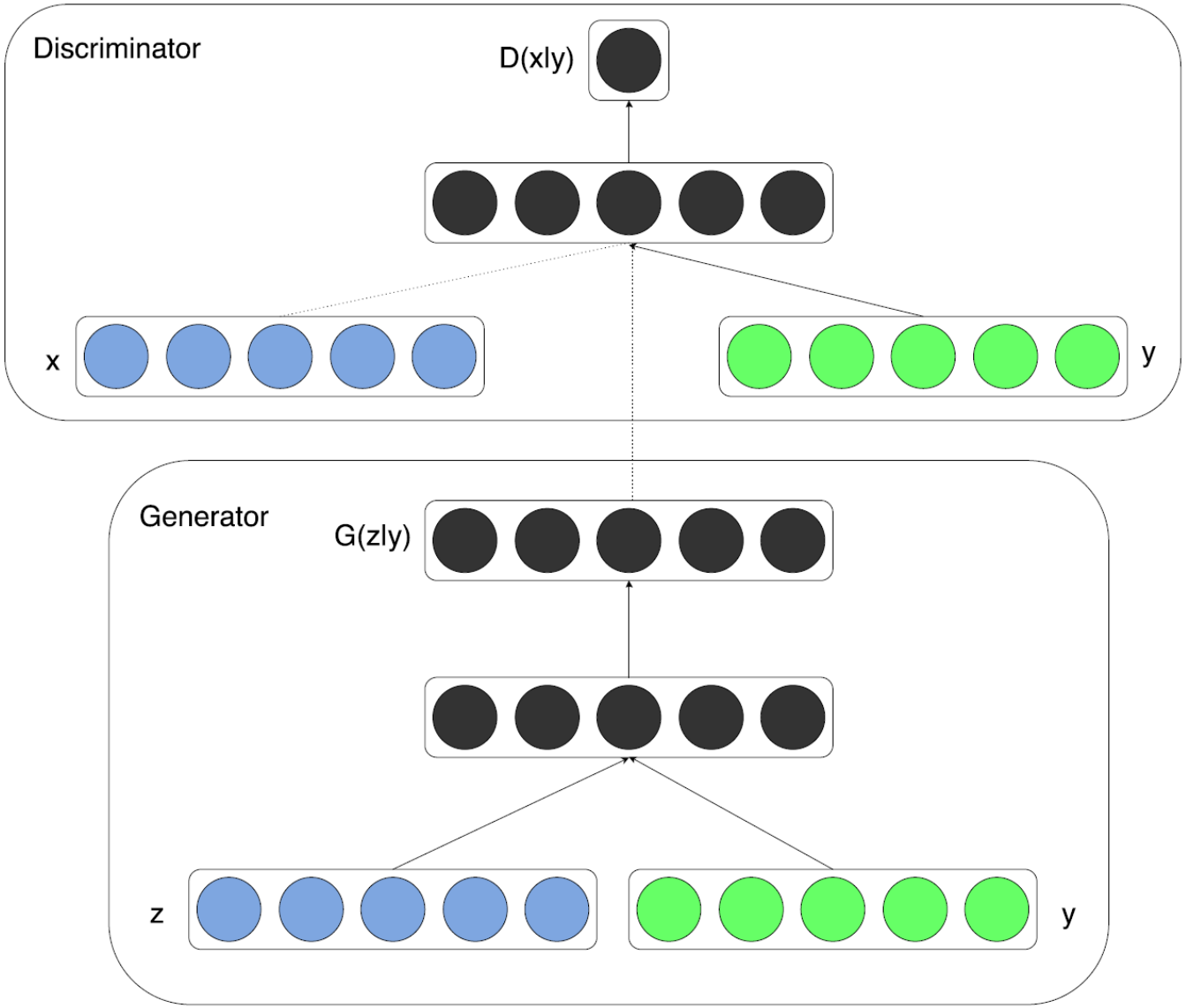
A Problem in the GANs Discussed So Far

- Suppose, I have a training image dataset of different animals
  - Lion, tiger, elephant, horse, etc.
- I train a GAN
- Can I generate an image of a lion?
  - No guarantee

Conditional GAN

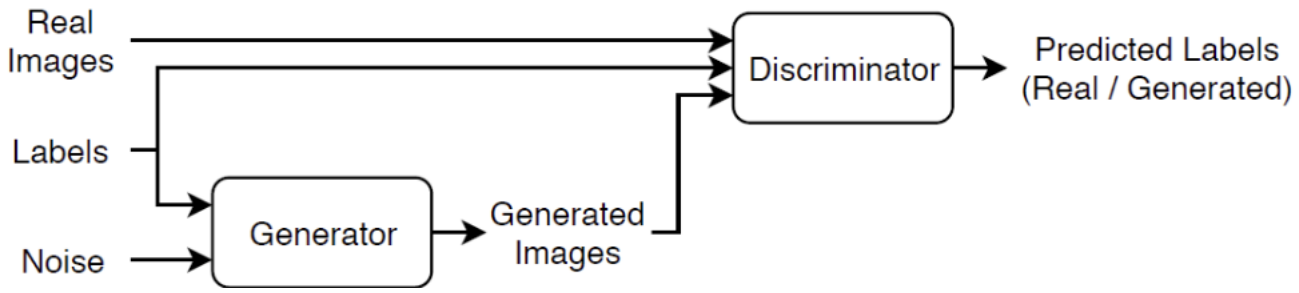
- Can I generate an image of a lion?
  - No guarantee
- How to solve this problem?
- Apply class label as condition in the GAN

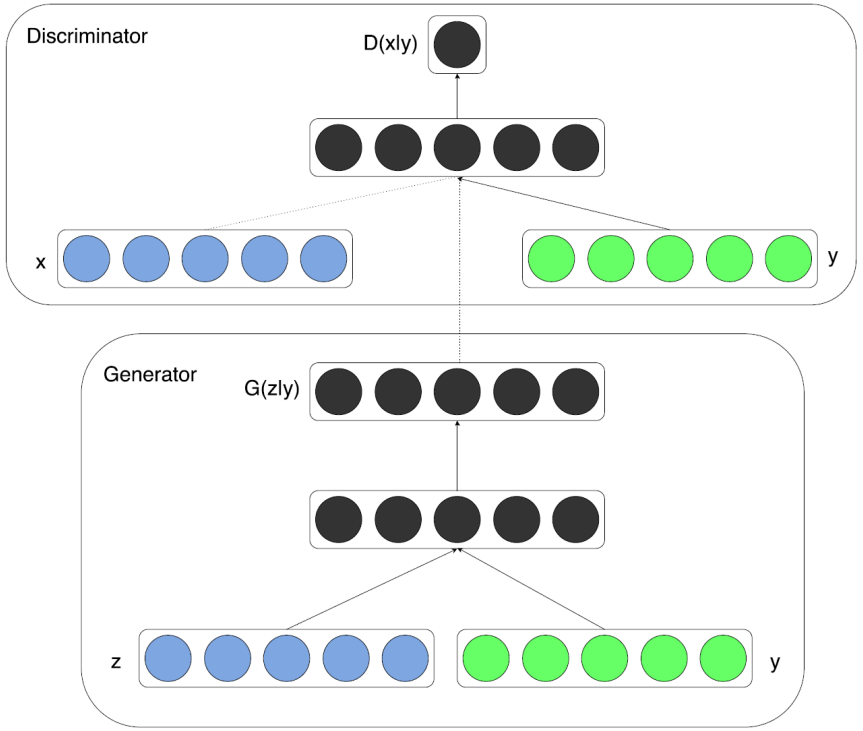
Conditional GAN



Conditional GAN

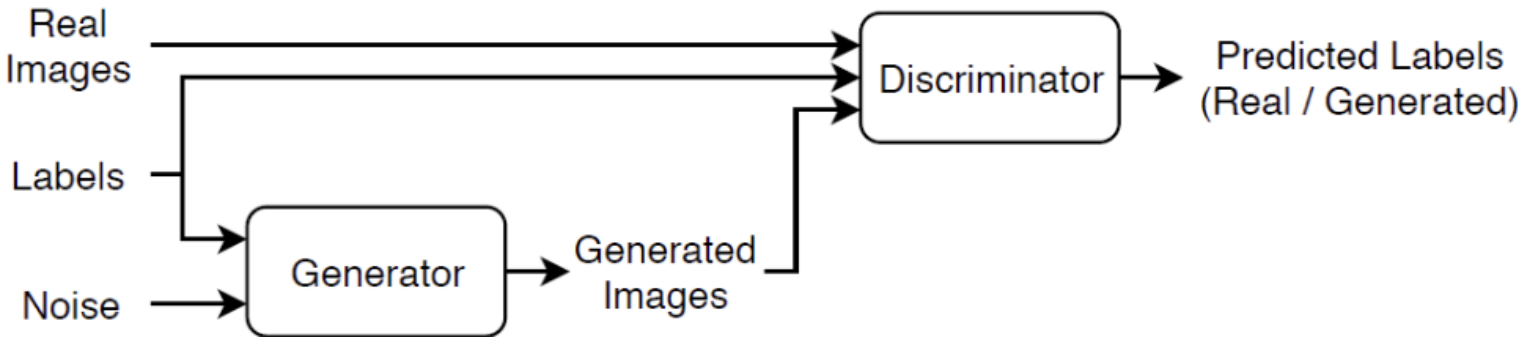
$$\min_D \max_G (-\mathbb{E}_{x\sim p_D} \log D(x|y) - \mathbb{E}_{x\sim p_z} \log(1 - D(G(x|y))))$$





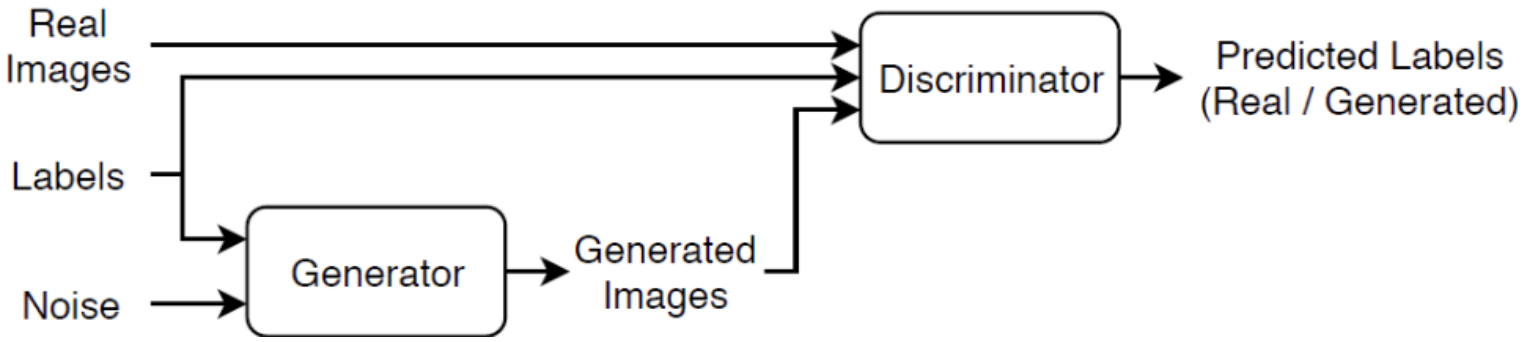
https://www.mathworks.com/help/deeplearning/ug/train-conditional-generative-adversarial-network.html

Conditional GAN



- Can you improve the discriminator?

Conditional GAN



- Can you improve the discriminator?
  - Instead of just real and fake, can you make the discriminator output class labels?
    - e.g., real class 1, fake class 3, etc.