Lecture 13 - (Deep) Generative Models

Ertunc Erdil

December 2025


Video Generation

Video Generation

Art Generation

The New York Times

An A.I.-Generated Picture Won an Art Prize. Artists Aren't Happy.

"I won, and I didn't break any rules," the artwork's creator says.

Jason Allon won the digital-art competition at the Colorado State Fair for his piece "Théâtre D'opéra Spatial" that he created using the AI software Midjourney. The US Copyright Office refused to grant him copyright for his piece writing "we have decided that we cannot register this copyright claim because the deposit does not contain any human authorship".

3D Generation

Image-to-3D

Text-to-3D

Tang et al. “LGM: Large Multi-View Gaussian Model for High-Resolution 3D Content Creation”, ECCV 2024

4D Generation

Wu et al. “CAT4D: Create Anything in 4D with Multi-View Video Diffusion Models”, Arxiv 2024

Conditional Generation

Zhang et al. “Adding Conditional Control to Text-to-Image Diffusion Models”, ICCV 2023

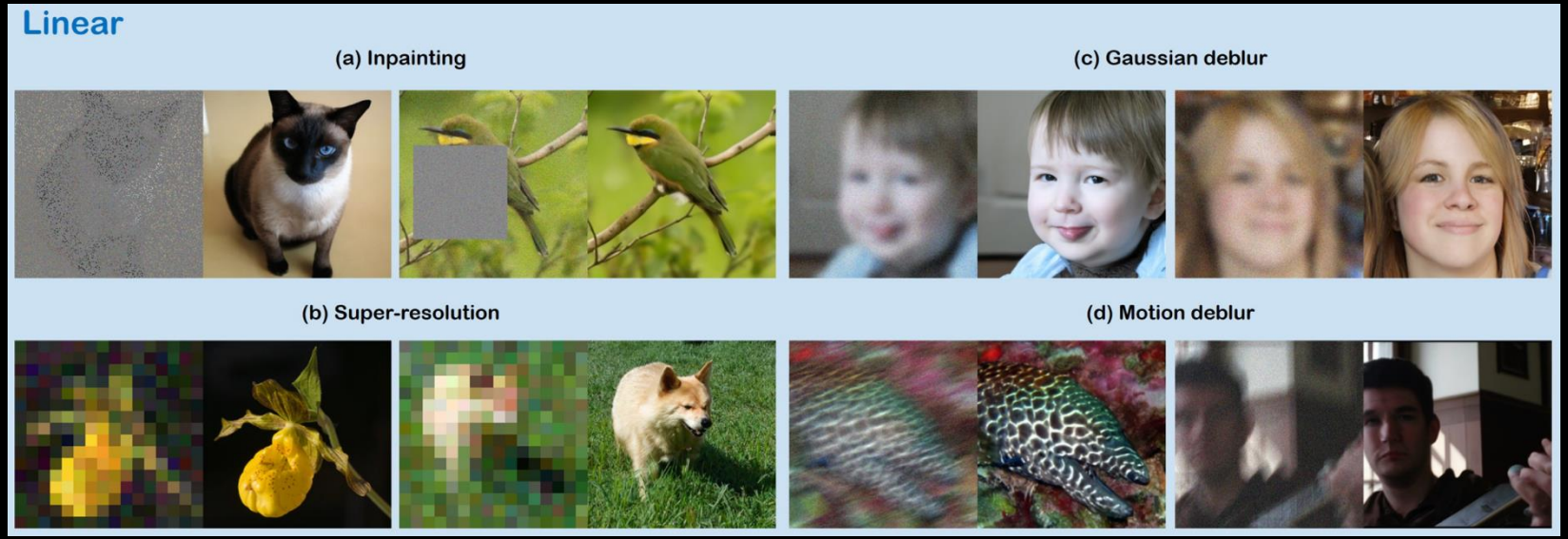
Personalization

Ruiz et al. “DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation”, CVPR 2023

Image Editing w/ Prompt

Hertz et al. "Prompt-to-prompt Image Editing with Cross-Attention Control", Arxiv 2022

Inverse Problems



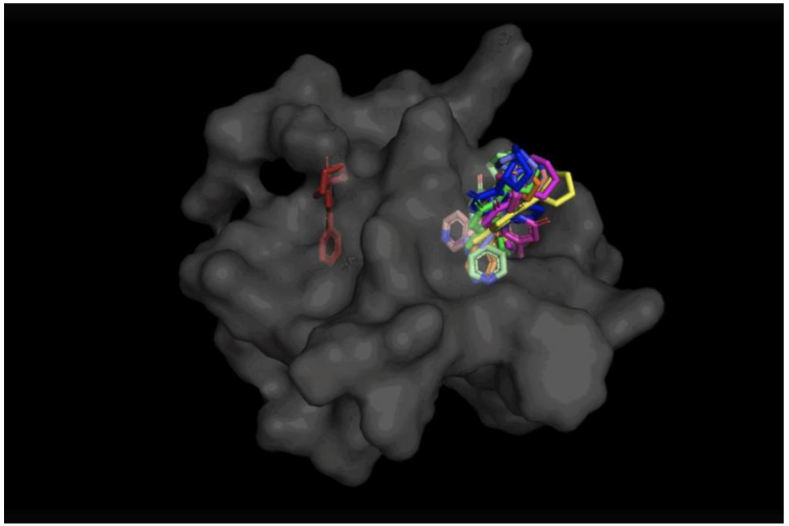
Chung et al. “Diffusion Posterior Sampling for General Noisy Inverse Problems”, ICLR 2023

Drug Discovery

Speeding up drug discovery with diffusion generative models

MIT researchers built DiffDock, a model that may one day be able to find new drugs faster than traditional methods and reduce the potential for adverse side effects.

Alex Ouyang | Abdul Latif Jameel Clinic for Machine Learning in Health
March 31, 2023



With the release of platforms like DALL-E 2 and Midjourney, diffusion generative models have achieved mainstream popularity, owing to their ability to generate a series of absurd, breathtaking, and often meme-worthy images from text prompts like "teddy bears working on new AI research on the moon in the 1980s." But a team of researchers at MIT's Abdul Latif Jameel Clinic for Machine Learning in Health (Jameel Clinic) thinks there could be more to diffusion generative models than just creating surreal images — they could accelerate the development of new drugs and reduce the likelihood of adverse side effects.

https://news.mit.edu/2023/speeding-drug-discovery-with-diffusion-generative-models-diffdock-0331

General Idea in Generative Modelling

Image from: https://pixabay.com/photos/pictures-photos-photo-collection-382009/

General Idea in Generative Modelling

- Learning data distribution $p(x;\theta)$

Image from: https://pixabay.com/photos/pictures-photos-photo-collection-382009/

General Idea in Generative Modelling

- Learning data distribution $p(x;\theta)$
- We can sample from the distribution

$$x' \sim p(x;\theta)$$

Image from: https://pixabay.com/photos/pictures-photos-photo-collection-382009/

General Idea in Generative Modelling

- Learning data distribution $p(x;\theta)$
- We can sample from the distribution

$$x' \sim p(x;\theta)$$

- We can evaluate the likelihood of the data

$$p(x = \text{  }; \theta) = 0.145$$

Image from: https://pixabay.com/photos/pictures-photos-photo-collection-382009/

General Idea in Generative Modelling

- Learning data distribution $p(x;\theta)$
- We can sample from the distribution

$$x' \sim p(x;\theta)$$

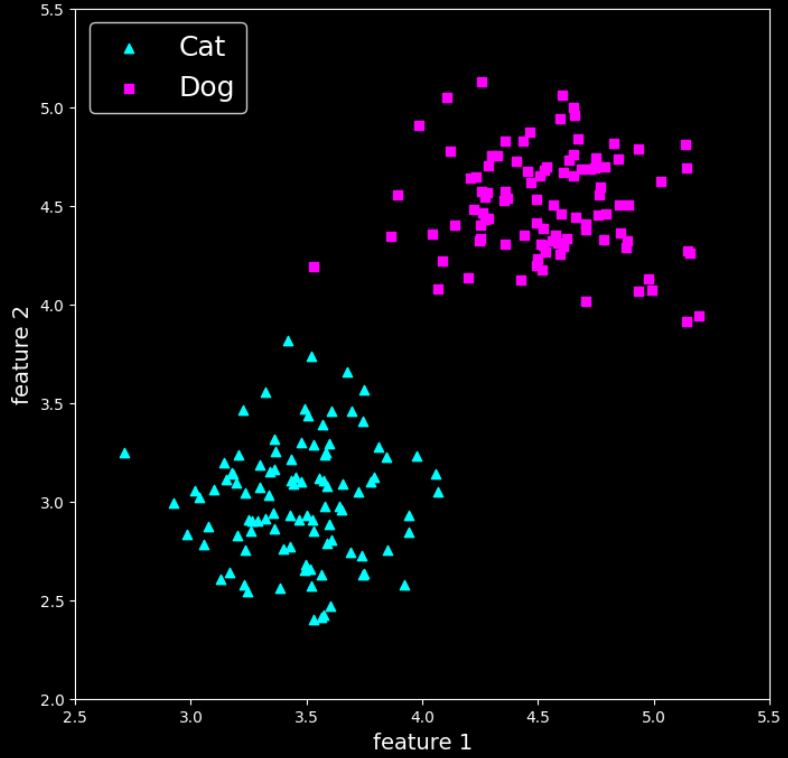
- We can evaluate the likelihood of the data

$$p(x = \text{  }; \theta) = 0.145$$

$$p(x = \text{  }; \theta) = 0.00001$$

Image from: https://pixabay.com/photos/pictures-photos-photo-collection-382009/

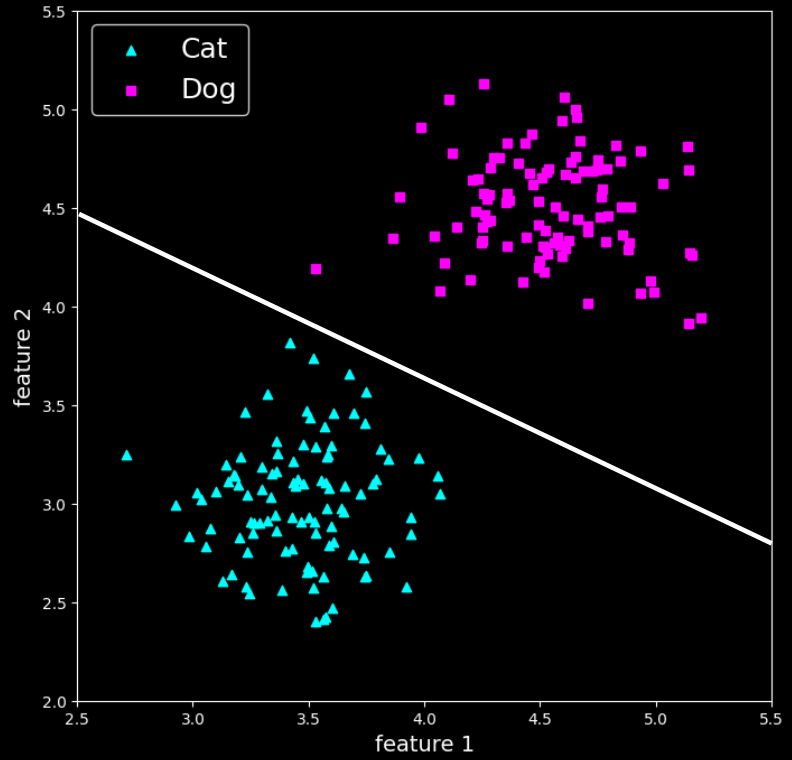
How are generative models useful beyond generating samples?



- Let’s consider a dataset of dog and cat images

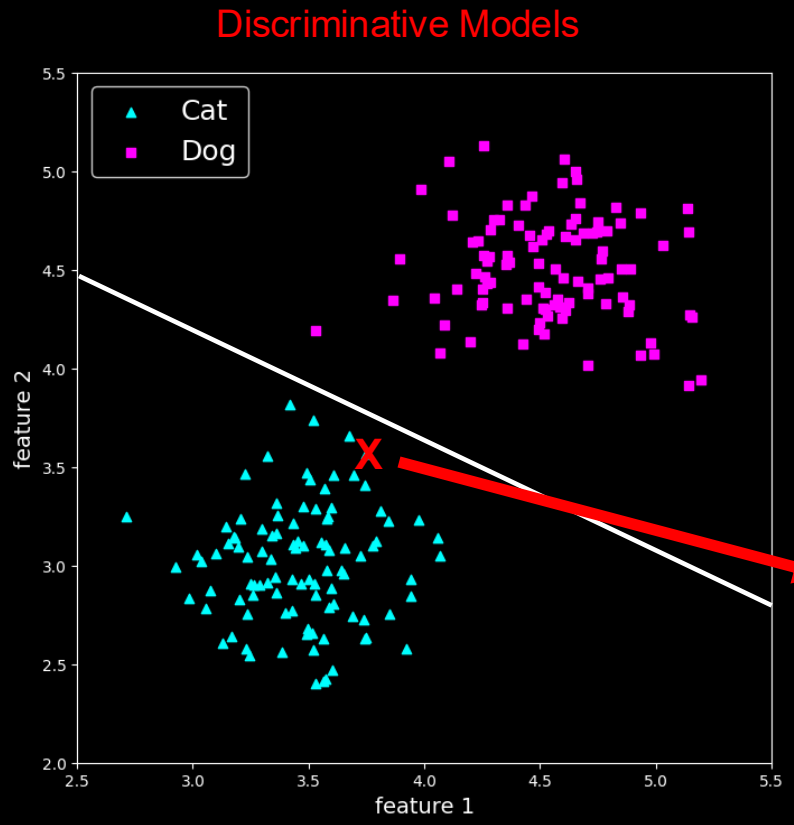
How are generative models useful beyond generating samples?

Discriminative Models



- Let's consider a dataset of dog and cat images
- Discriminative models find a decision boundary to *discriminate* these classes
- This corresponds to modelling the conditional distribution $p(y|x)$

How are generative models useful beyond generating samples?



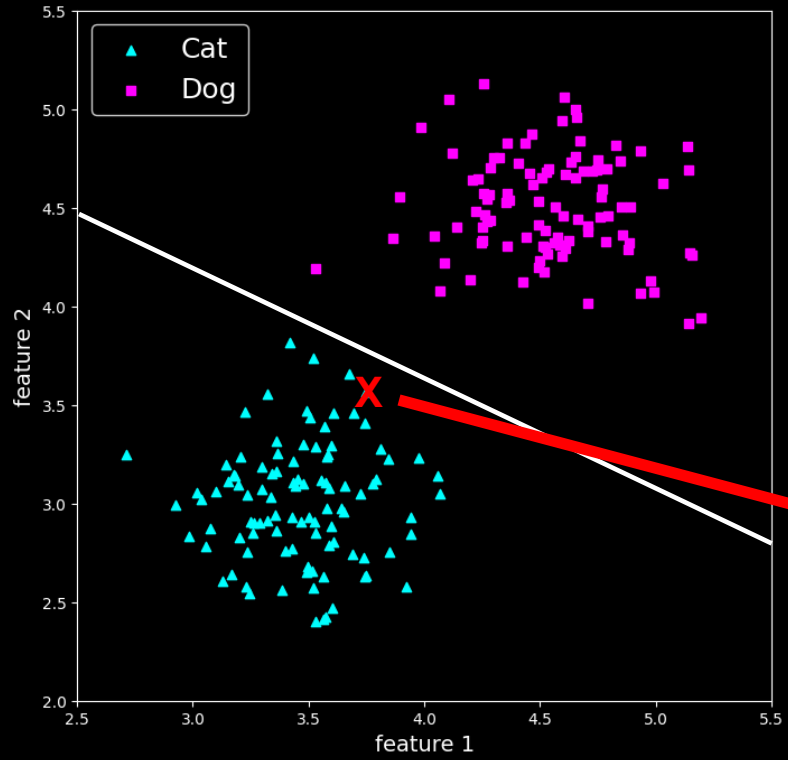
- Let's consider a dataset of dog and cat images
- Discriminative models find a decision boundary to *discriminate* these classes
- This corresponds to modelling the conditional distribution $p(y|x)$

$p(y = cat|x =$  $) = 0.90$

$p(y = dog|x =$  $) = 0.10$

How are generative models useful beyond generating samples?

Discriminative Models



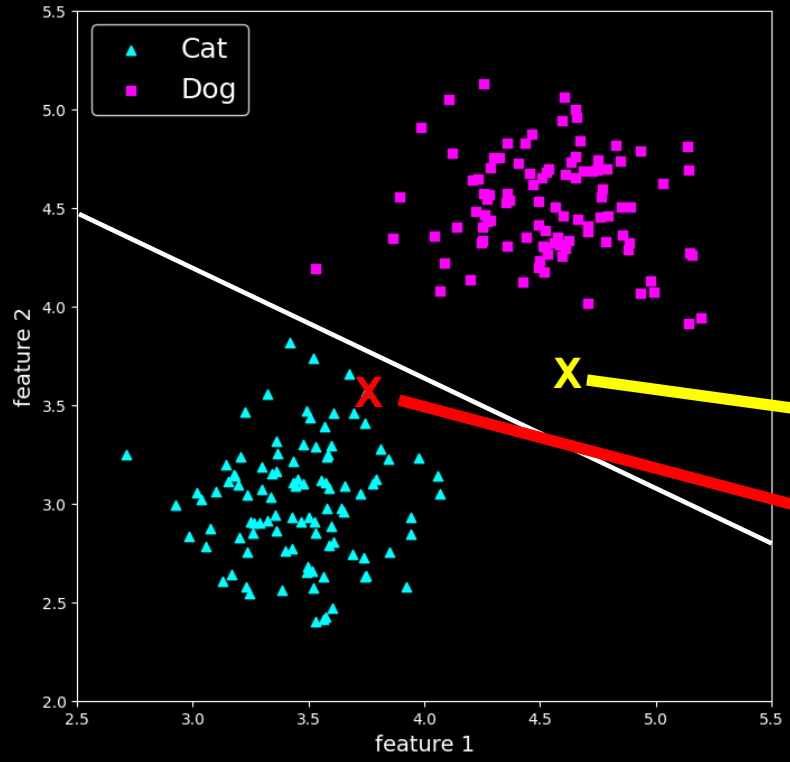
- Let's consider a dataset of dog and cat images
- Discriminative models find a decision boundary to *discriminate* these classes
- This corresponds to modelling the conditional distribution $p(y|x)$

$p(y = cat|x =$  $) = 0.90$

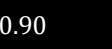
$p(y = dog|x =$  $) = 0.10$

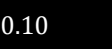
How are generative models useful beyond generating samples?

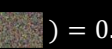
Discriminative Models

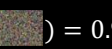


- Let's consider a dataset of dog and cat images
- Discriminative models find a decision boundary to *discriminate* these classes
- This corresponds to modelling the conditional distribution $p(y|x)$

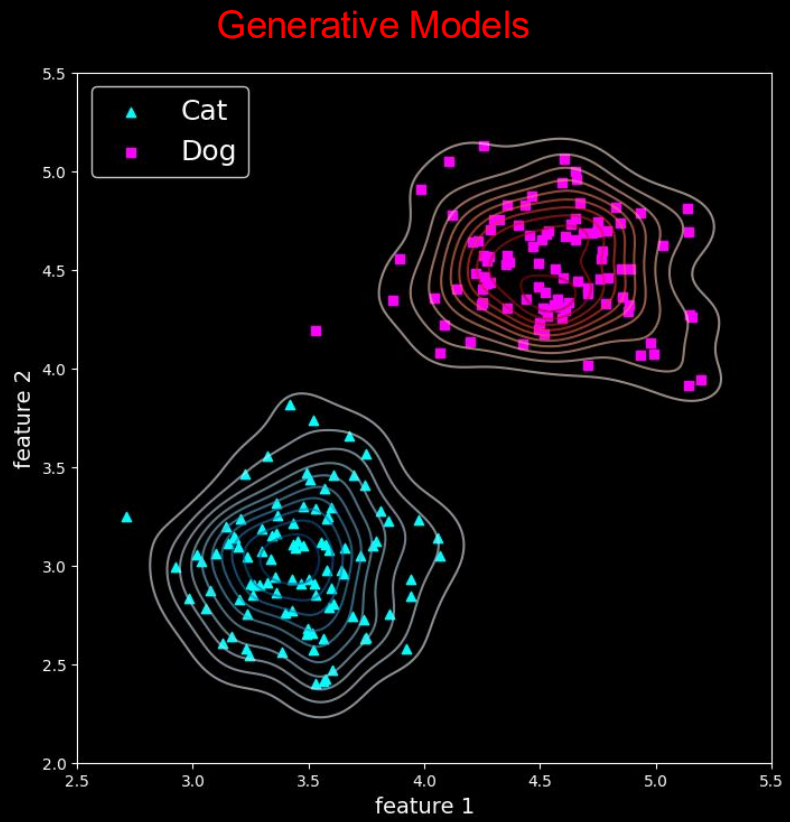
$p(y = cat|x =$  $) = 0.90$

$p(y = dog|x =$  $) = 0.10$

$p(y = cat|x =$  $+$  $) = 0.05$

$p(y = dog|x =$  $+$  $) = 0.95$

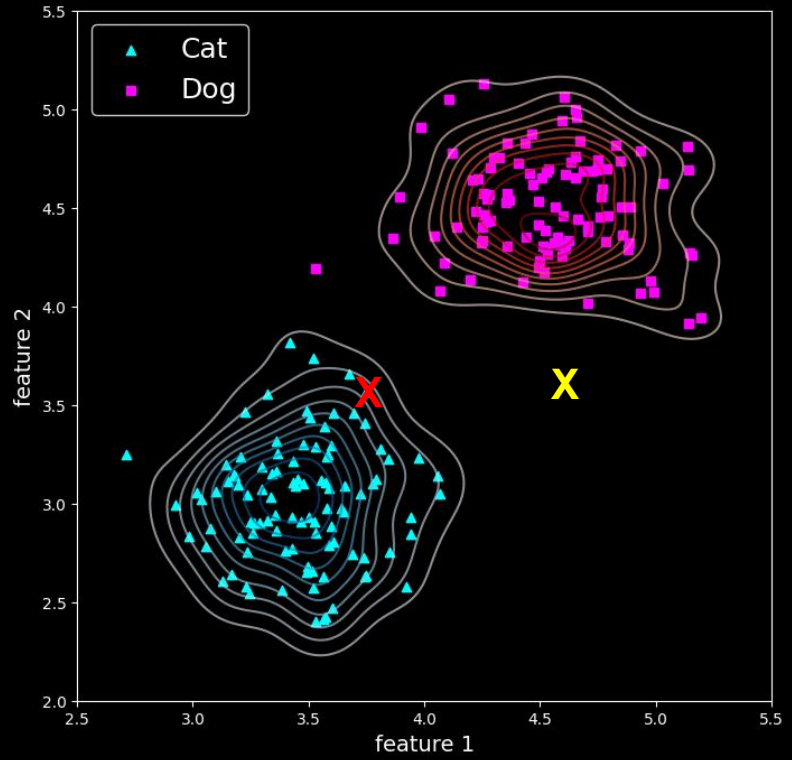
How are generative models useful beyond generating samples?



- Generative models can learn the data distribution
- This corresponds to modelling the distribution $p(x)$

How are generative models useful beyond generating samples?

Generative Models



- Generative models can learn the data distribution
- This corresponds to modelling the distribution $p(x)$
- Discriminative Model: Certain but wrong prediction!

  $p(y = dog|x =$  $) = 0.95$
- Generative Model:

  $p(x =$  $) \gg p(x =$  $)$
- We can model joint distribution $p(x, y) = p(y|x)p(x)$

  $p(x, y) = p(y = dog|x =$  $)$ $p(x =$  $)$

Uncertain prediction!

In this lecture, we will mainly talk about …

- How can we evaluate the likelihood $p(x)$?
- How can we generate samples from $p(x)$?
- How can we learn the data distribution $p(x)$ from observations?

In this lecture, we will mainly talk about …

1. Low-dimensional data

- How can we evaluate the likelihood $p(x)$?
- How can we generate samples from $p(x)$?
- How can we learn the data distribution $p(x)$ from observations?

2. High-dimensional data -> Deep Generative Models

In this lecture, we will mainly talk about …

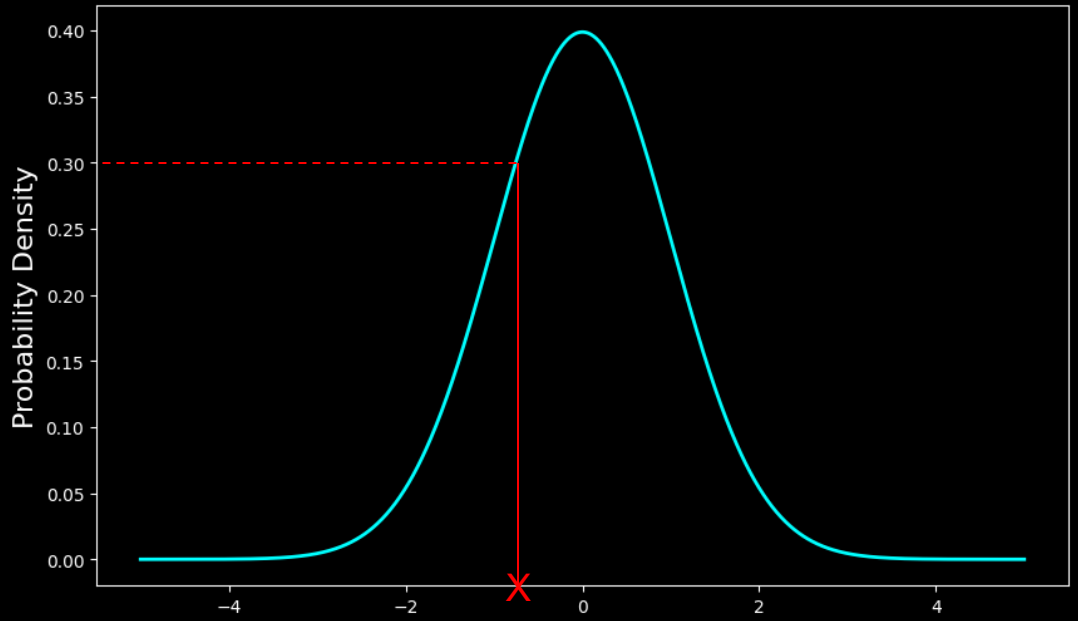
1. Low-dimensional data

- How can we evaluate the likelihood $p(x)$?
- How can we generate samples from $p(x)$?
- How can we learn the data distribution $p(x)$ from observations?

How can we evaluate the likelihood $p(x)$?

How can we evaluate the likelihood $p(x)$?

- Let's consider a simple example of 1D Gaussian function $N(\mu = 0, \sigma = 1)$



$$p(x) = N(\mu = 0, \sigma = 1)$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In this lecture, we will mainly talk about …

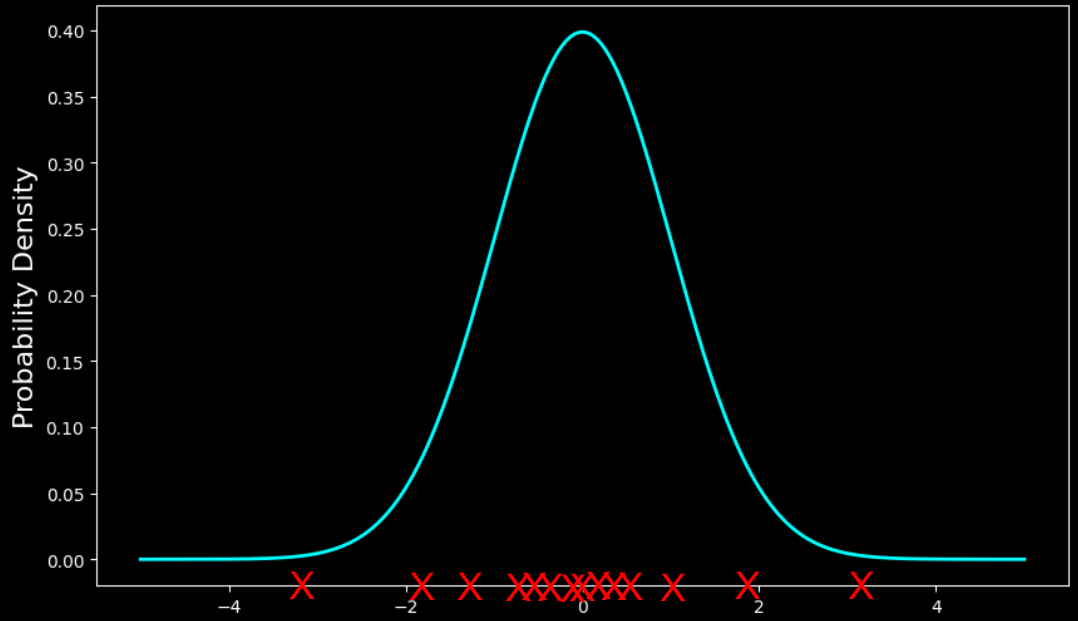
1. Low-dimensional data

- ✓ How can we evaluate the likelihood $p(x)$?
- How can we generate samples from $p(x)$?
- How can we learn the data distribution $p(x)$ from observations?

How can we generate samples from $p(x)$?

How can we generate samples from $p(x)$?

- Let's consider a simple example of 1D Gaussian function $N(\mu = 0, \sigma = 1)$



$$p(x) = N(\mu = 0, \sigma = 1)$$

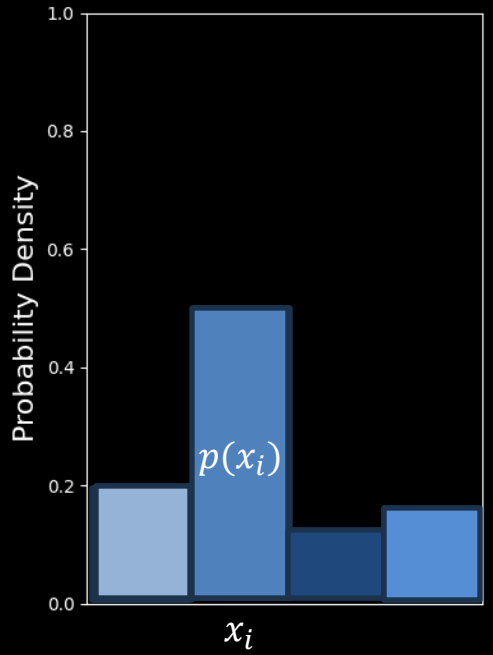
$$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- If the probability density function (PDF) of the distribution is known, we can sample from it directly.

Q: How can we sample from this PDF?

How can we generate samples from $p(x)$?

- First, let's consider Discrete Probability Distributions

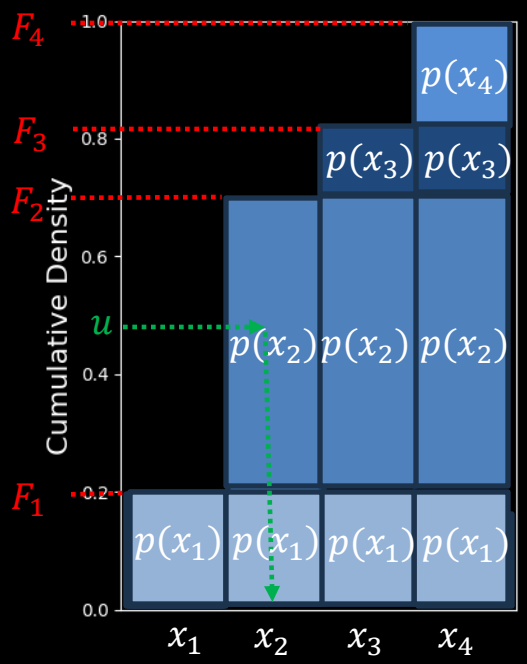


- n discrete values x_i with probability $p(x_i)$.
- Requirements of a PDF:
  - $p(x_i) \geq 0$
  - $\sum_{i=1}^{n} p(x_i) = 1$

Q: How can we sample from this PDF?

How can we generate samples from $p(x)$?

- First, let's consider Discrete Probability Distributions

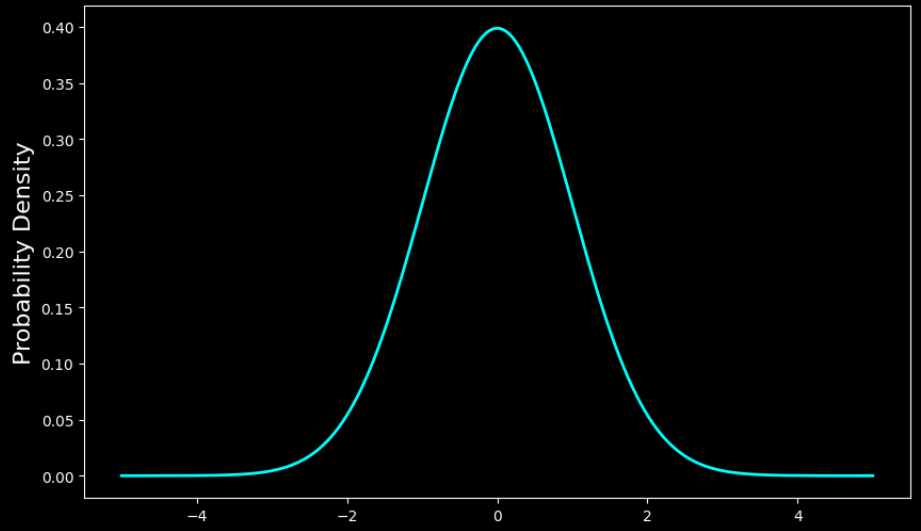


Inverse Transform Sampling

1. Compute Cumulative Density Function (CDF)
   - $F_i = \sum_{j=1}^{i} p(x_j)$ where $0 \leq F_i \leq 1$
     $F_n = 1$
2. Draw a sample from a Uniform Distribution between 0 and 1
   - $u \sim U(0,1)$
3. Take x_i if $F_{i-1} \leq u \leq F_i$

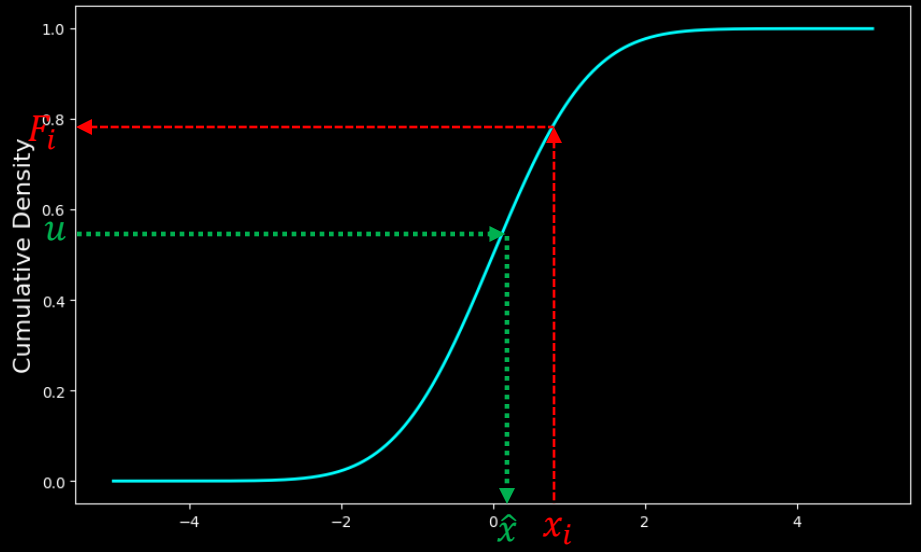
How can we generate samples from $p(x)$?

- We can apply Inverse Transform Sampling on Continuous Probability Distributions
- Let's go back to 1D Gaussian function example $\mathcal{N}(\mu = 0, \sigma = 1)$



How can we generate samples from $p(x)$?

- We can apply Inverse Transform Sampling on Continuous Probability Distributions
- Let's go back to 1D Gaussian function example $\mathcal{N}(\mu = 0, \sigma = 1)$



1. Compute Cumulative Density Function (CDF)
   - $F_i = \int_{-\infty}^{i} p(x_j)\, d_j$ where $0 \leq F_i \leq 1$
     $F_n = 1$
2. Draw a sample from a Uniform Distribution between 0 and 1
   - $u \sim U(0, 1)$
3. Evaluate the inverse CDF at u to generate a sample
   - $\hat{x} \sim F^{-1}(u)$

How can we generate samples from $p(x)$?

- What if we cannot compute the inverse CDF F^{-1}?
- We have other tools:
  - Rejection Sampling
  - MCMC Sampling (e.g. Metropolis-Hastings)

- Most of these methods require having access to $p(x)$!

In this lecture, we will mainly talk about …

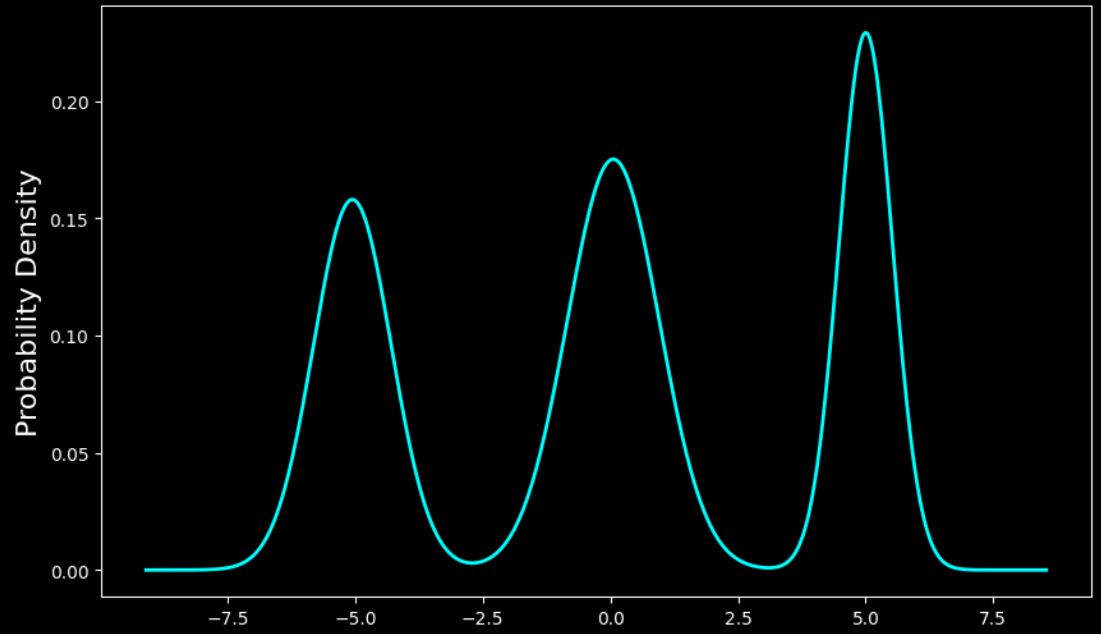
1. Low-dimensional data

- ✓ How can we evaluate the likelihood $p(x)$?
- ✓ How can we generate samples from $p(x)$?
- • How can we learn the data distribution $p(x)$ from observations?

How can we learn the data distribution $p(x)$ from observations?

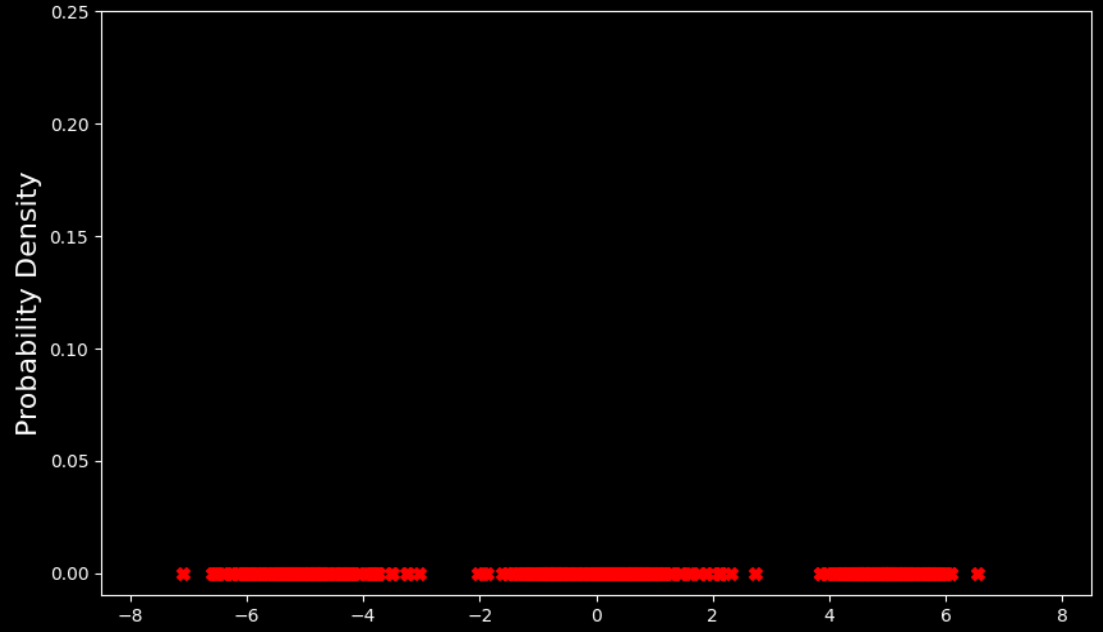
How can we learn the data distribution $p(x)$ from observations?

- In most real applications we do not know $p(x)$



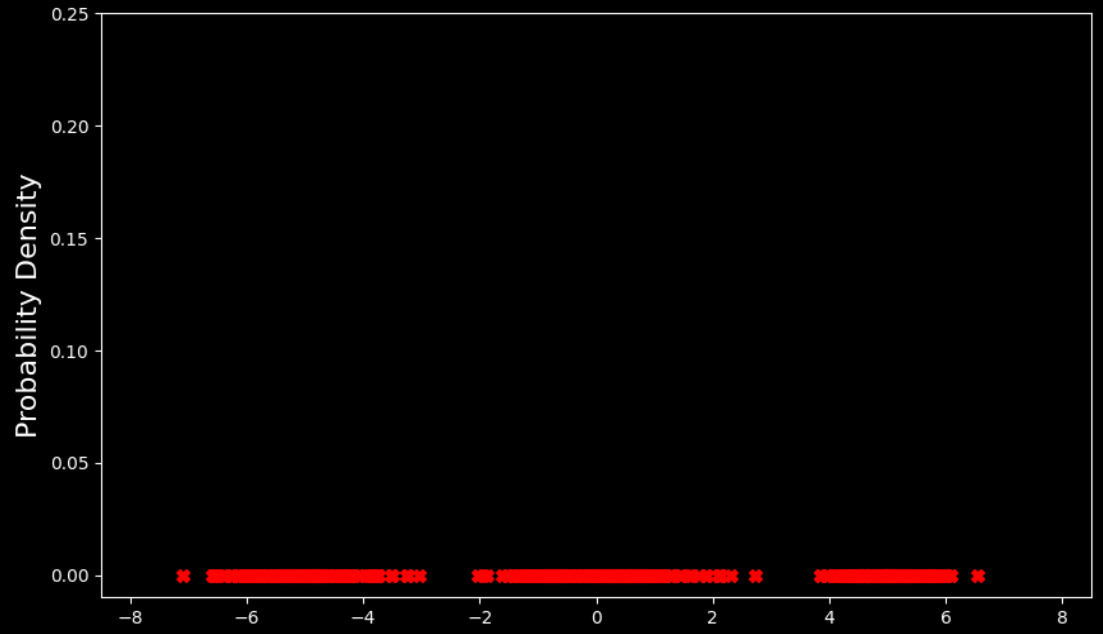
How can we learn the data distribution $p(x)$ from observations?

- In most real applications we do not know $p(x)$
- Instead, we have access to observations, i.e., samples from $p(x)$



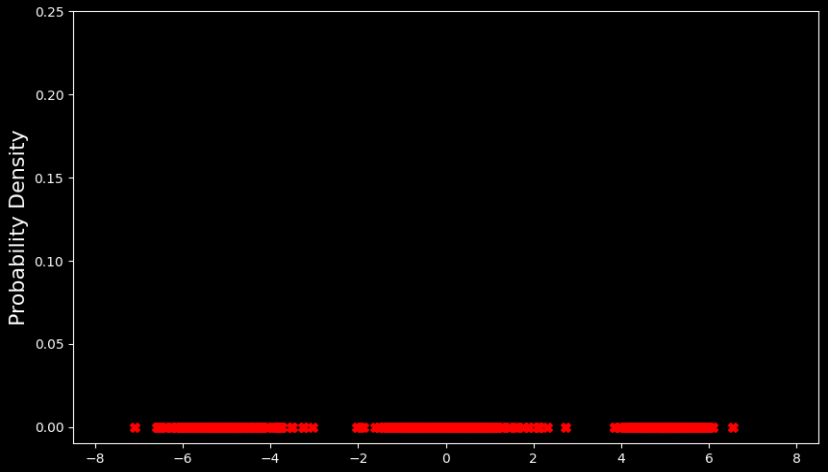
How can we learn the data distribution $p(x)$ from observations?

- We will talk about two methods:
    1. Gaussian Mixture Models (GMMs)
    2. Kernel Density Estimation (KDE)



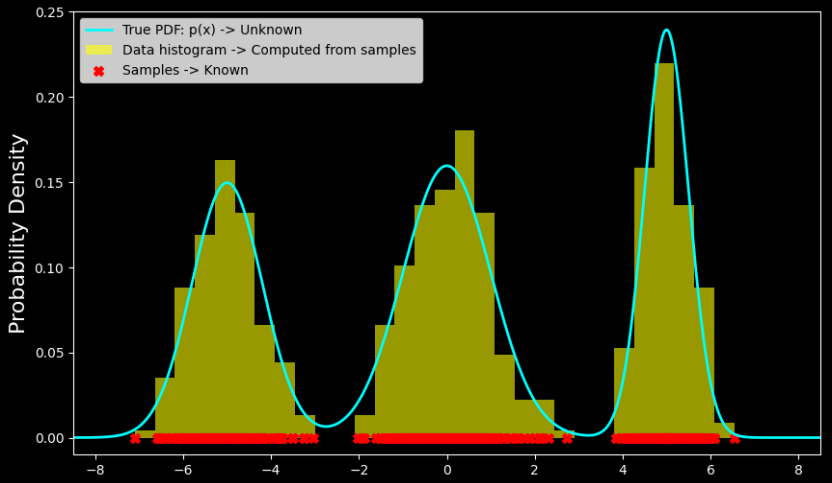
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)

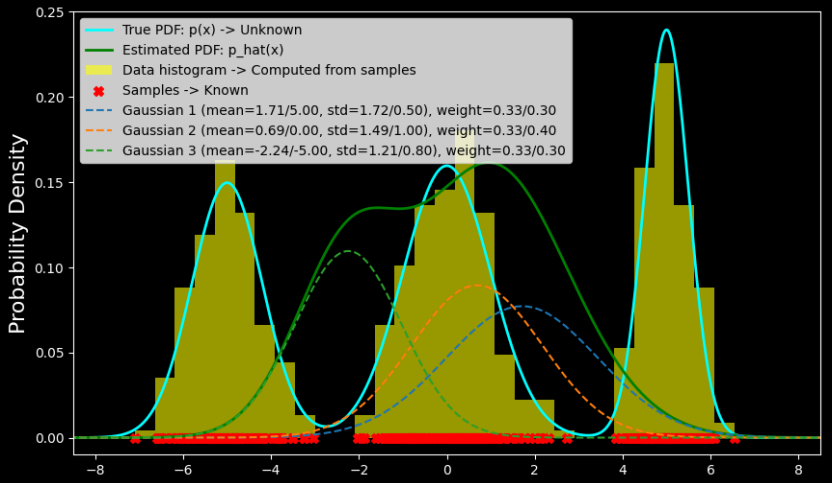


- The data comes from $p(x) = \sum_{i=1}^{3} w_i \, N(\mu_i, \sigma_i)$ where,
- $w = [0.3, 0.4, 0.3]$
- $\mu = [-5, 0, 5]$
- $\sigma = [0.8, 1.0, 0.5]$

w, μ, σ are unknown

How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



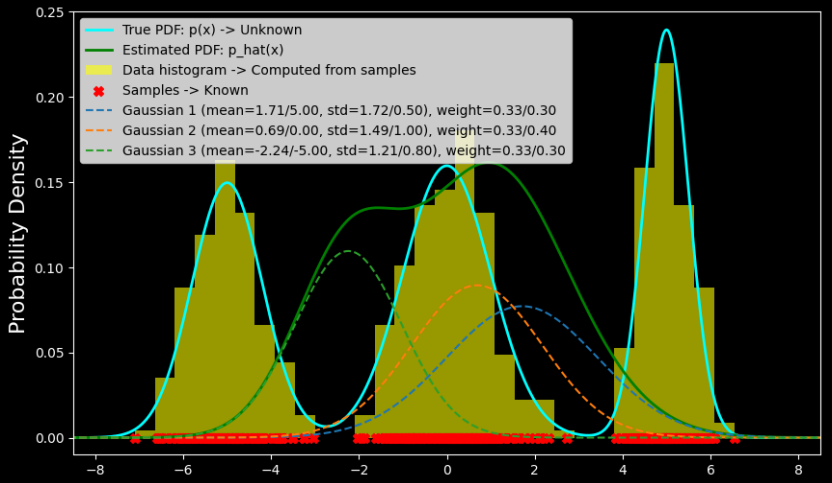
1. Randomly initialize weights and parameters of each Gaussian

- $w^0 = [0.33, 0.33, 0.33]$
- $\mu^0 = [1.71, 0.69, -2.24]$
- $\sigma^0 = [1.72, 1.49, 1.21]$
- Note that number of Gaussians, K, is a parameter and we chose it as $K = 3$ here

need prior knowledge about distribution

How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)

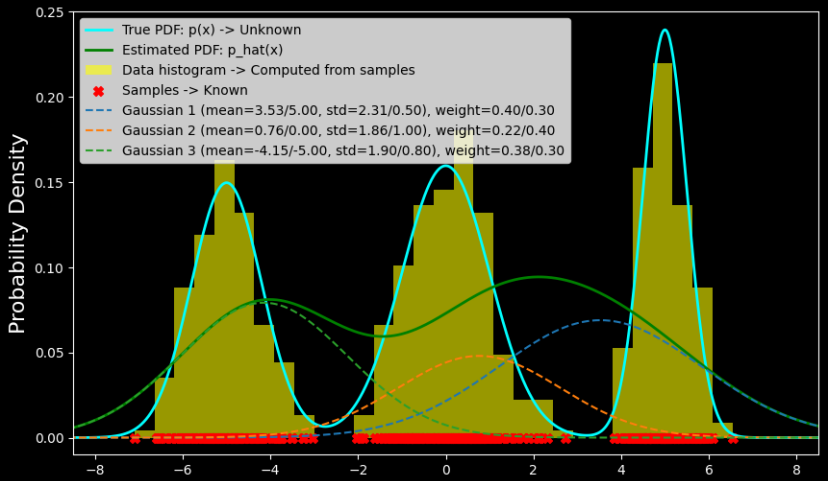


1. **Expectation step**: Compute probability that data points x is in the k^{th} Gaussian using the previous estimates of the weights (w^{t-1}) and parameters (μ^{t-1}, σ^{t-1})

$$p(k|x) = \frac{w_k^{t-1}\, N\left(x\middle|\mu_k^{t-1}, \sigma_k^{t-1}\right)}{\sum_{j=1}^{K} w_j^{t-1}\, N(x|\mu_j^{t-1}, \sigma_j^{t-1})}$$

How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



2. **Maximization step**:

- Compute new weights w^t

$$w_k^t = \frac{1}{N}\sum_{i=1}^{N} p(k|x_i)$$

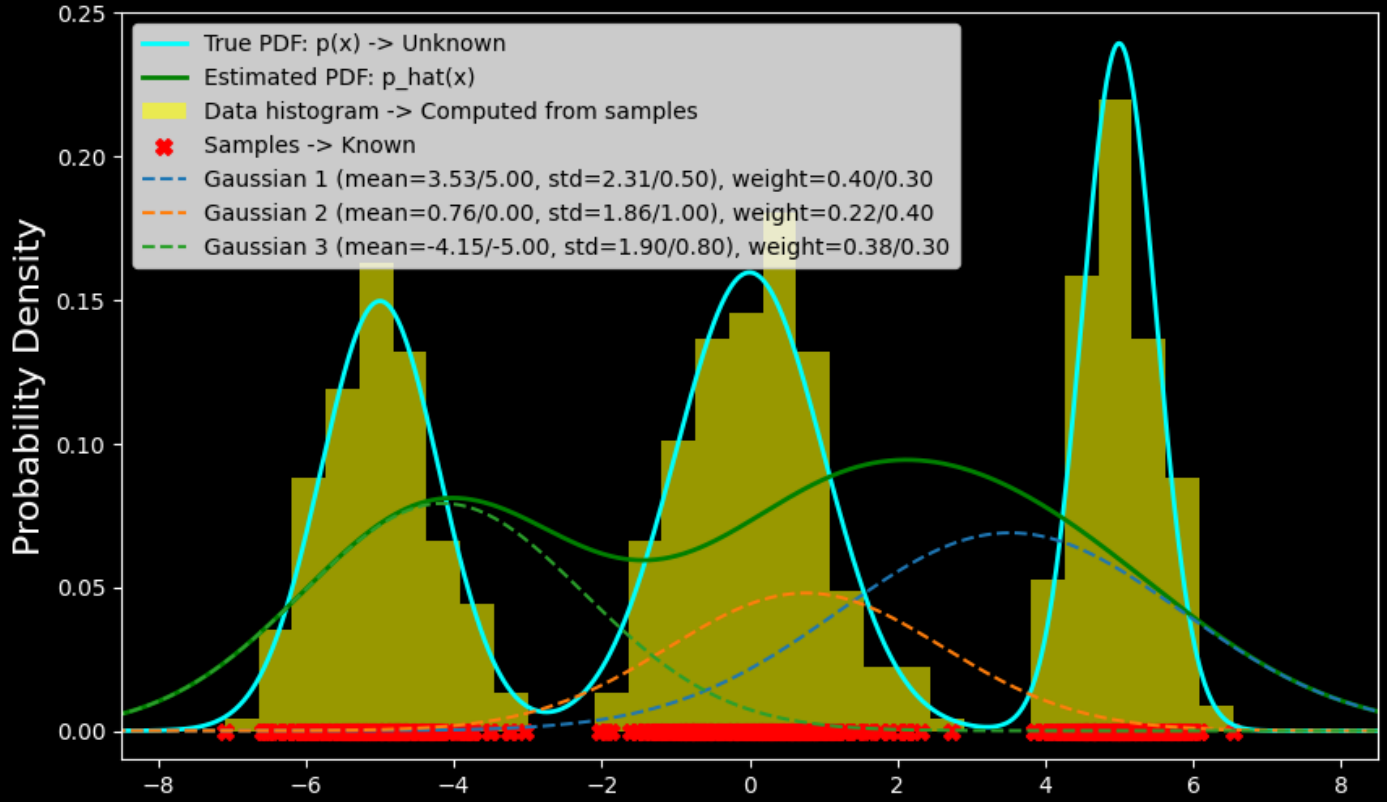
- Compute new Gaussian parameters (μ^t, σ^t)

$$\mu_k^t = \frac{\sum_{i=1}^{N} x_i p(k|x_i)}{\sum_{i=1}^{N} p(k|x_i)}$$

$$\sigma_k^t = \frac{\sum_{i=1}^{N} (x_i - \mu_k^t)^2 p(k|x_i)}{\sum_{i=1}^{N} p(k|x_i)}$$

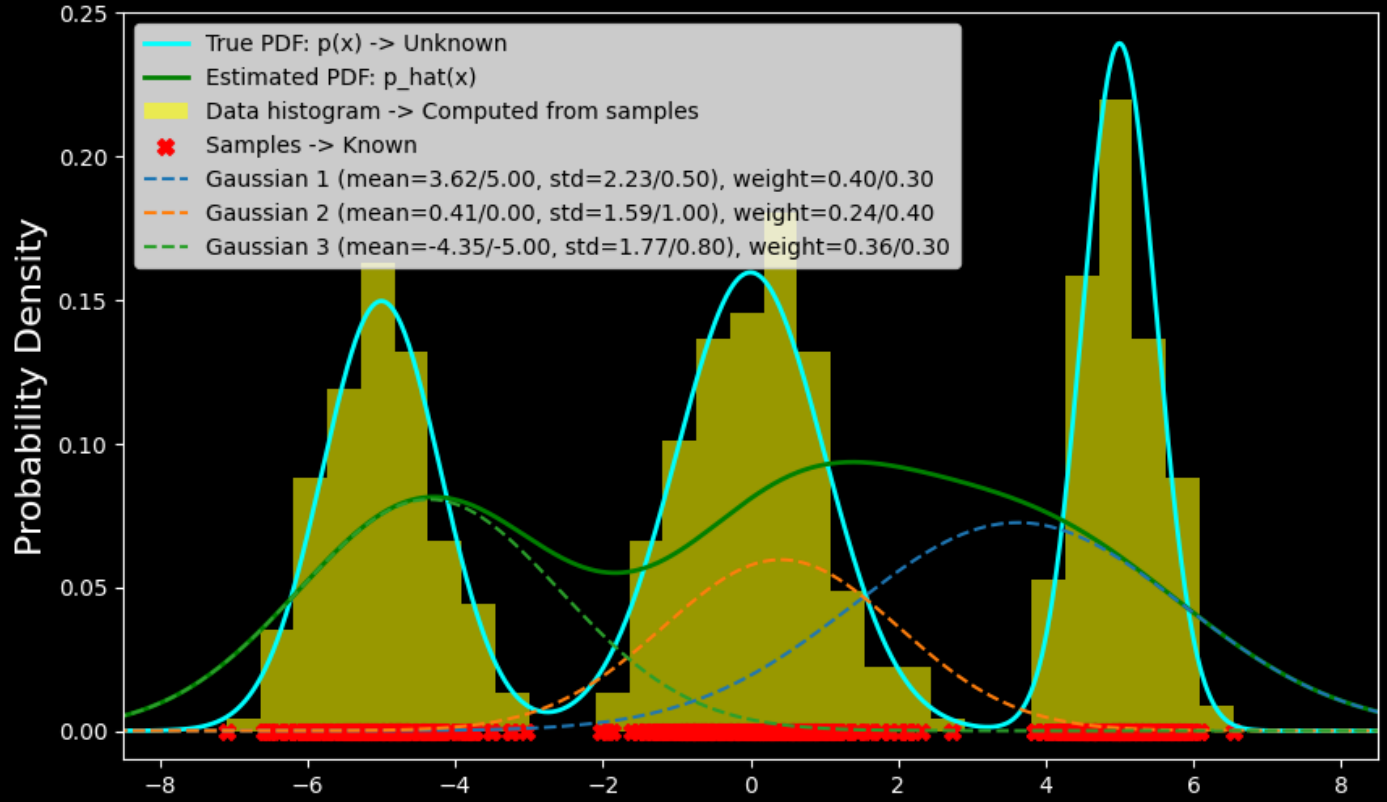
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



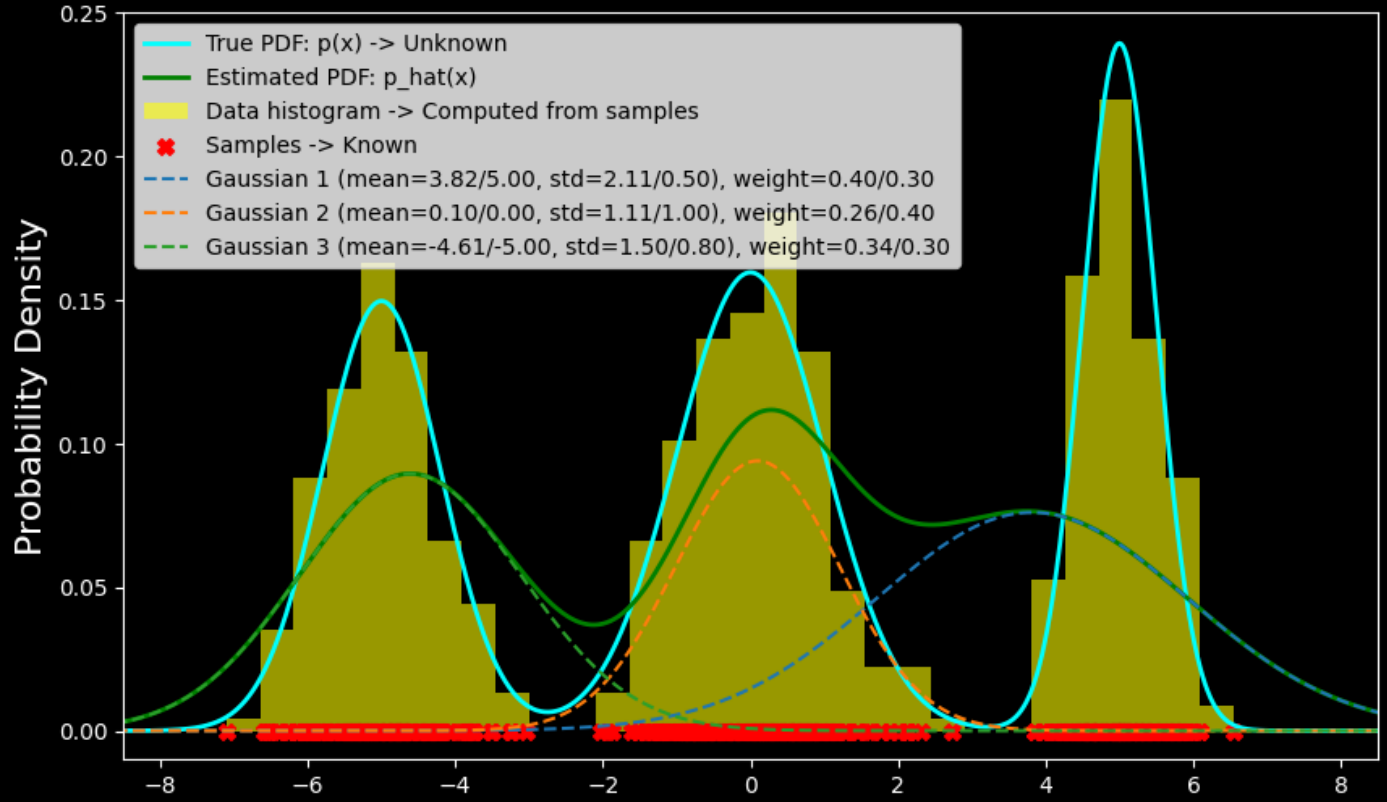
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



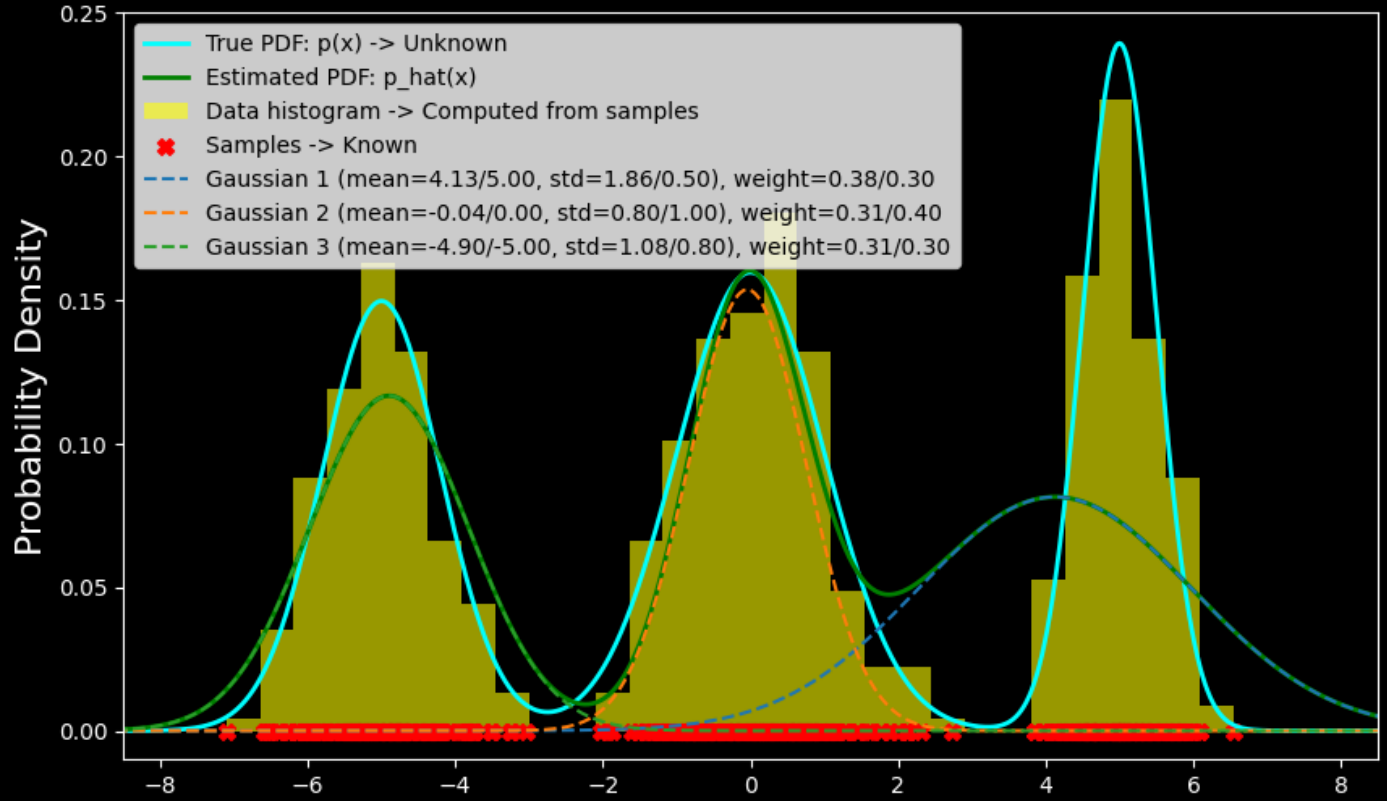
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



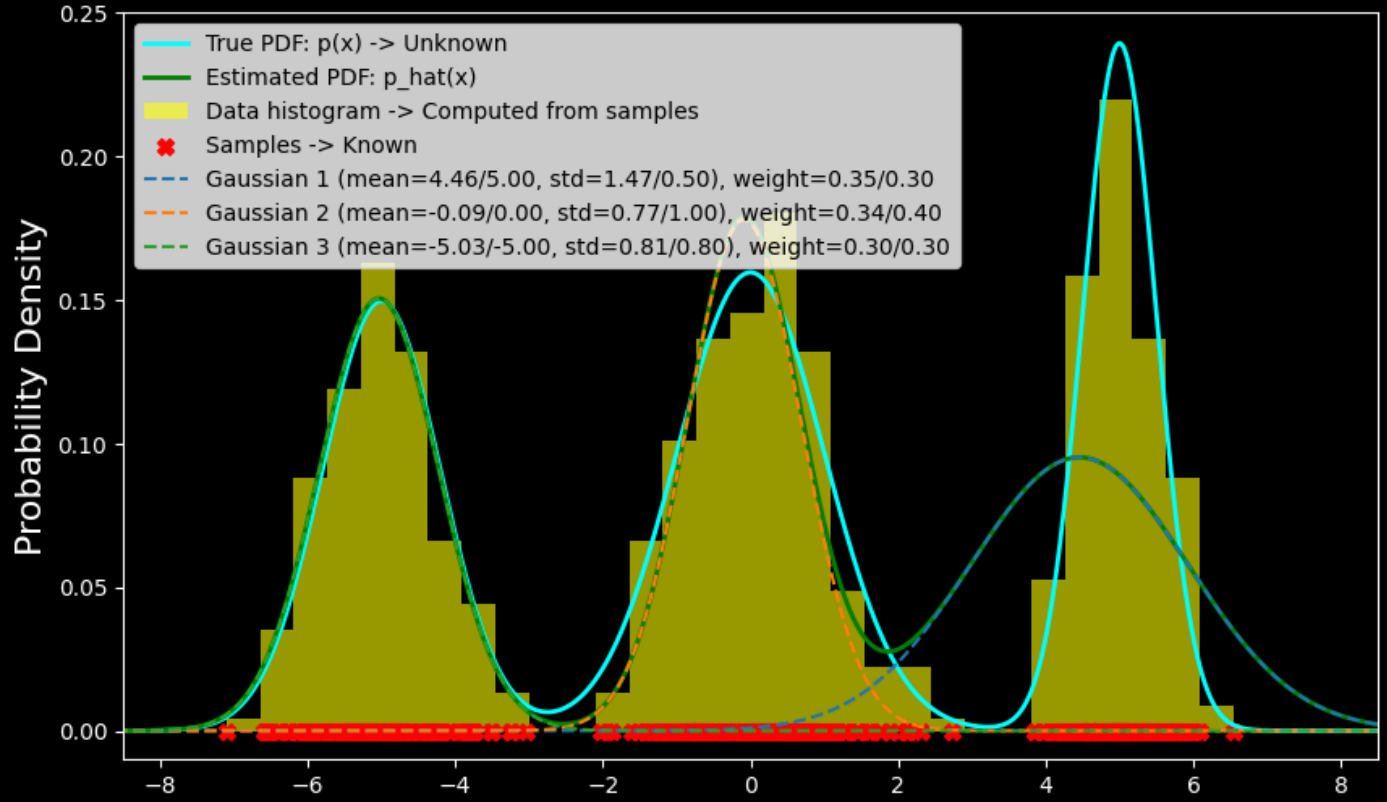
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



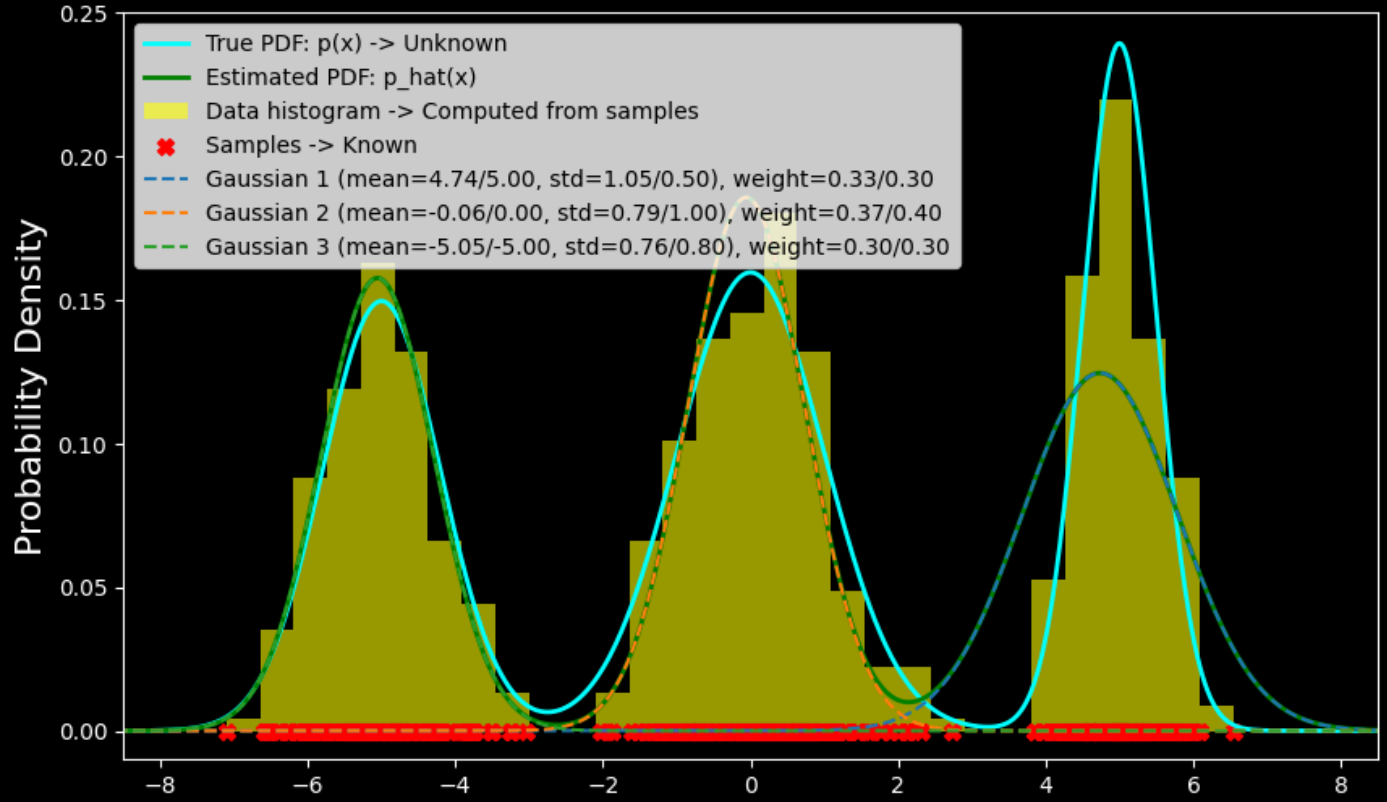
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



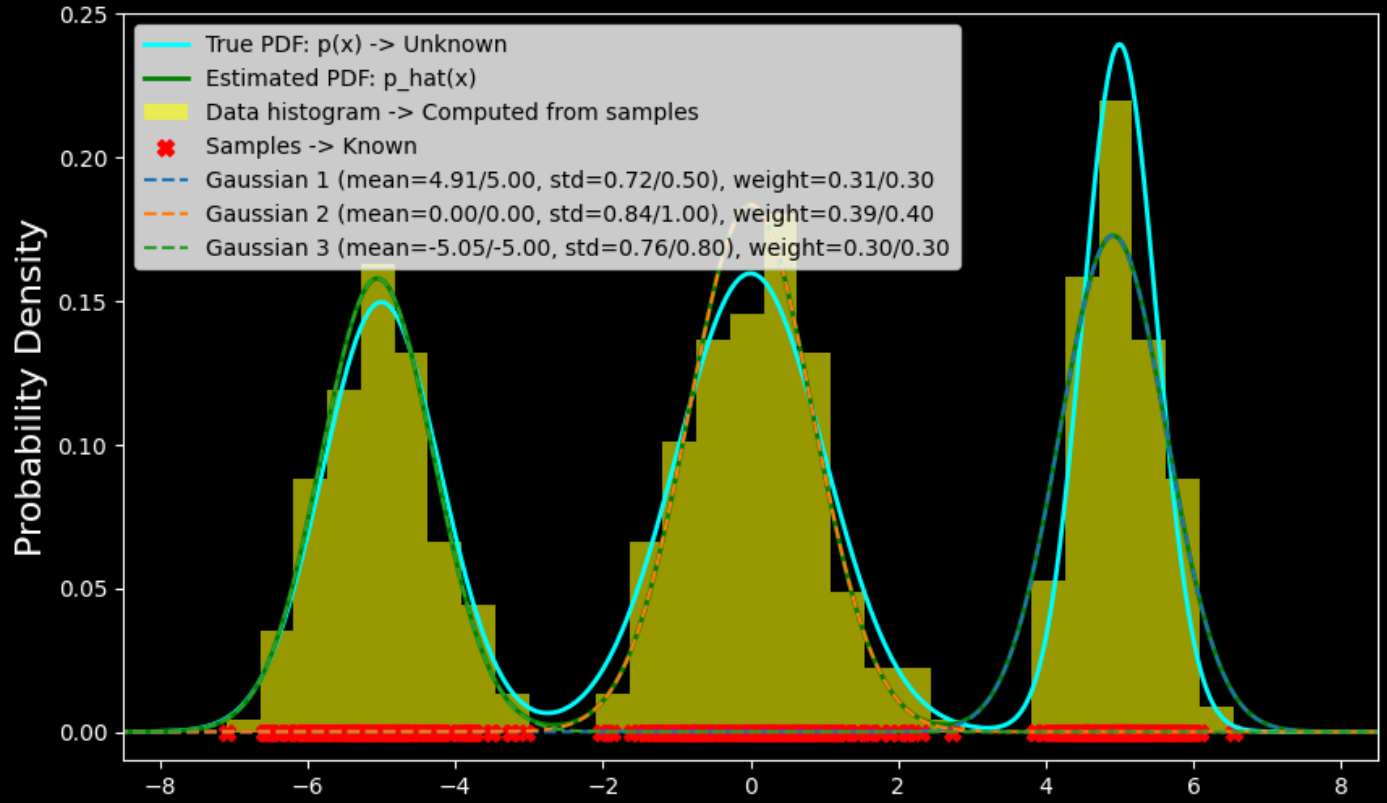
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



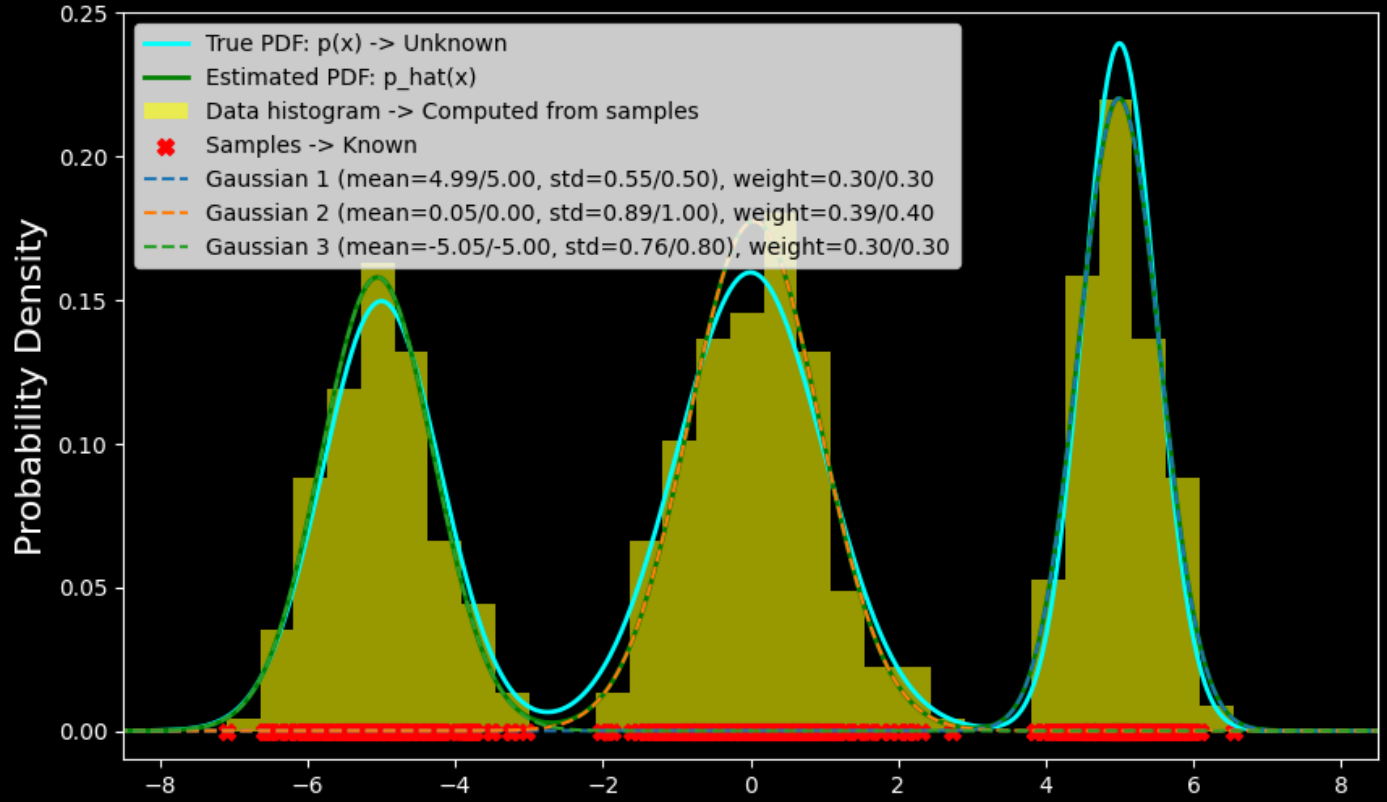
How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)



How can we learn the data distribution $p(x)$ from observations?

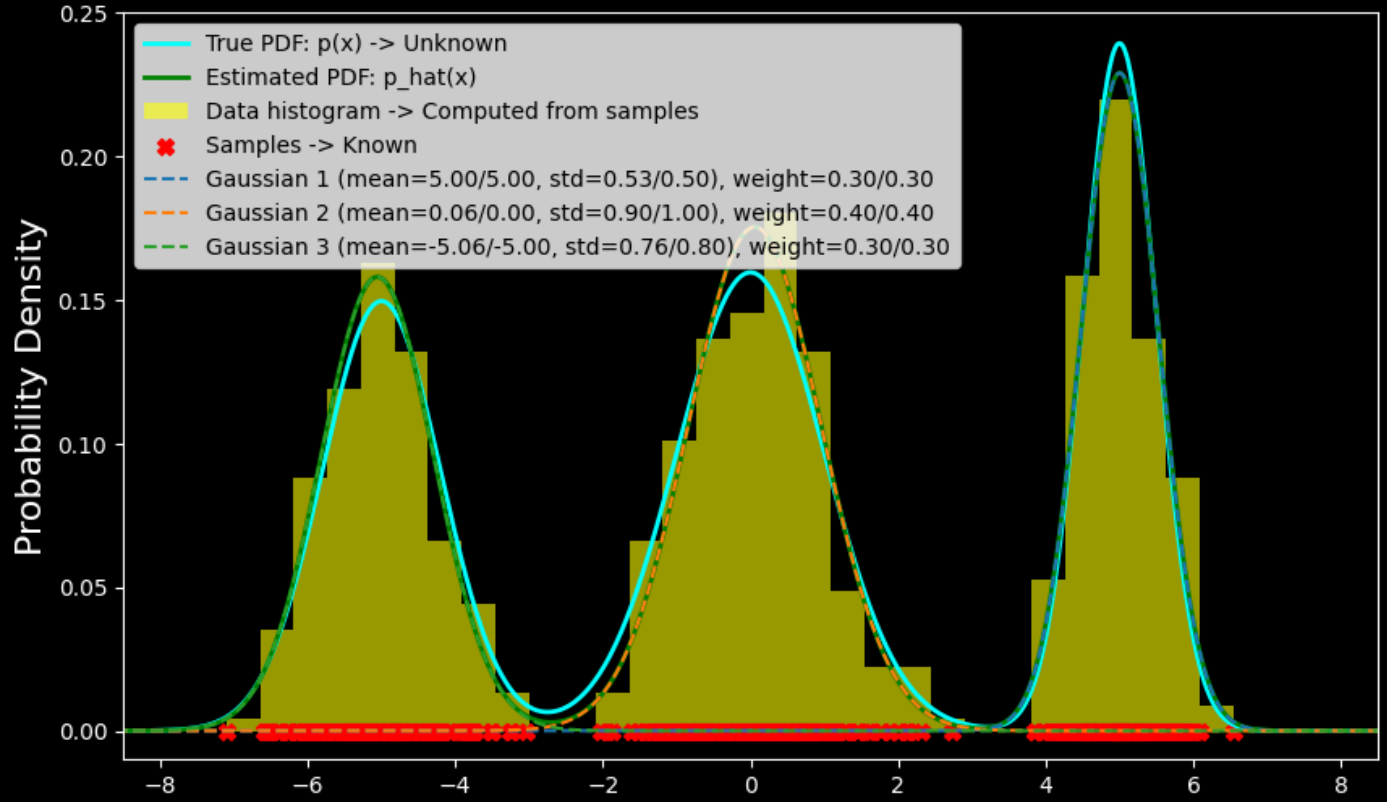
1. Gaussian Mixture Models (GMMs)



How can we learn the data distribution $p(x)$ from observations?

1. Gaussian Mixture Models (GMMs)

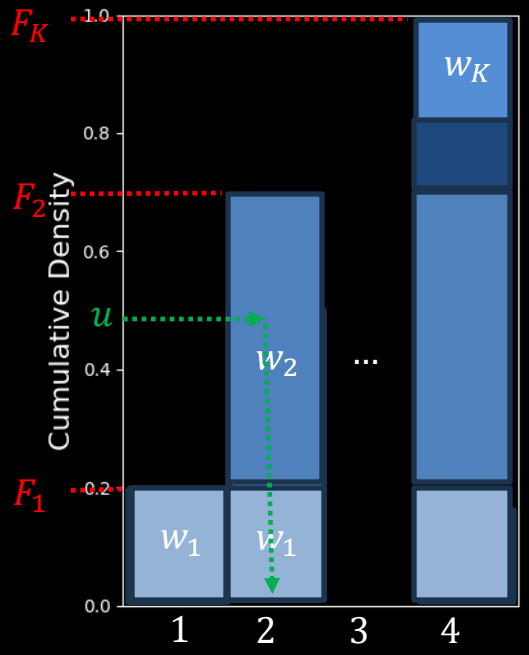
only for low dimensions -> the more dimensions, the more difficult to find grouping (space more sparse)



How can we generate samples from $p(x)$ learned by Gaussian Mixture Models (GMMs)?

1. Randomly select one of the K Gaussian components according to their weights w_k

   Remember Inverse Transform Sampling for Discrete Variables

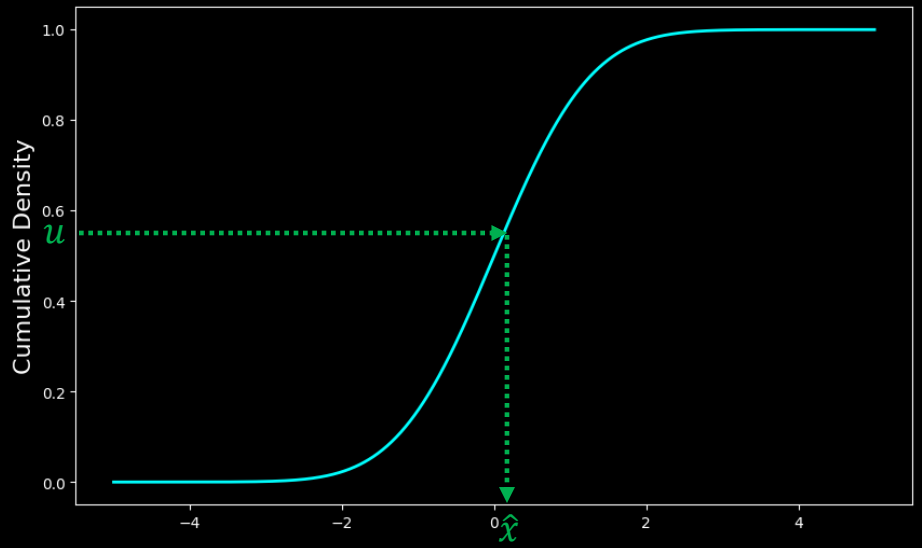


How can we generate samples from $p(x)$ learned by Gaussian Mixture Models (GMMs)?

1. Randomly select one of the K Gaussian components according to their weights w_k

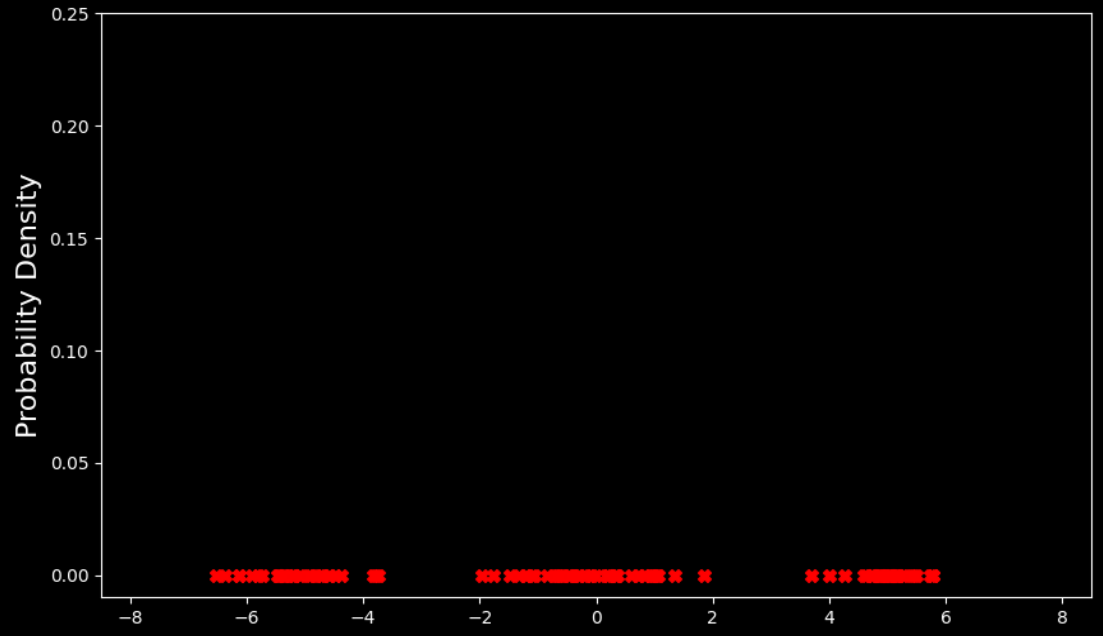
2. Randomly sample from the selected Gaussian with parameters $N(\mu^{\mathrm{T}}, \sigma^{T})$

Remember Inverse Transform Sampling for Continuous Variables



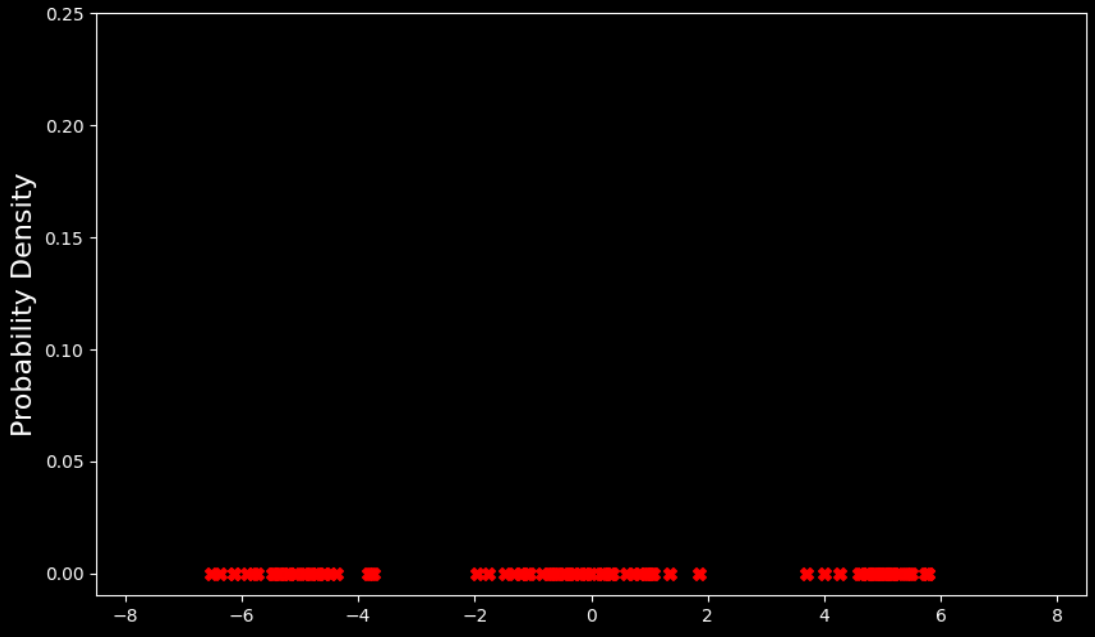
How can we learn the data distribution $p(x)$ from observations?

- We will talk about two methods:
  - ✓ Gaussian Mixture Models (GMMs)
  - 2. Kernel Density Estimation (KDE)



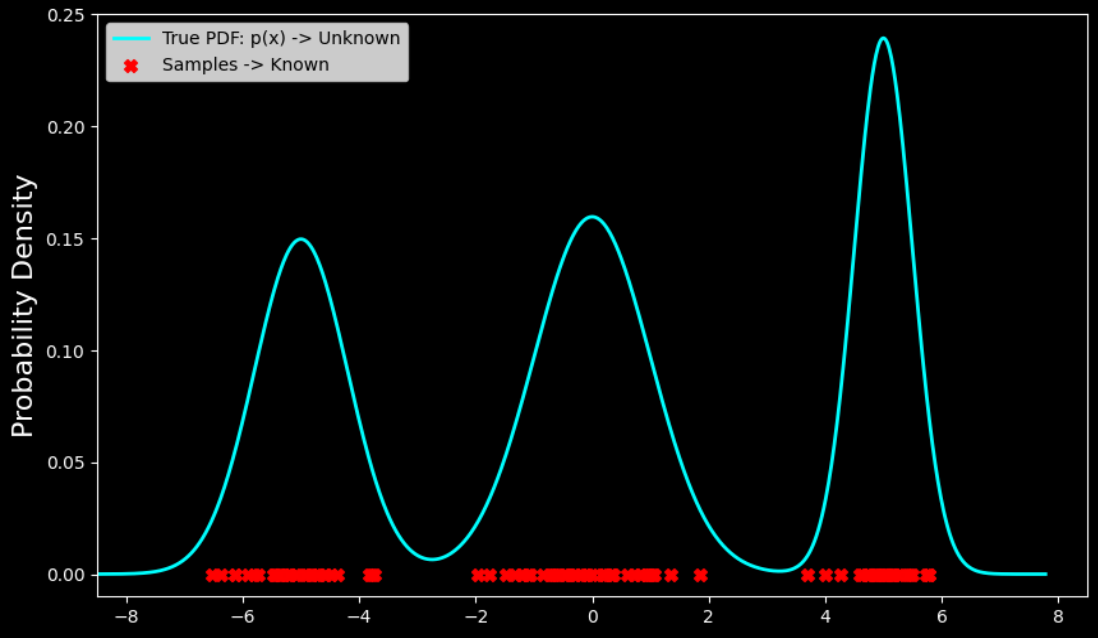
How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)



How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)



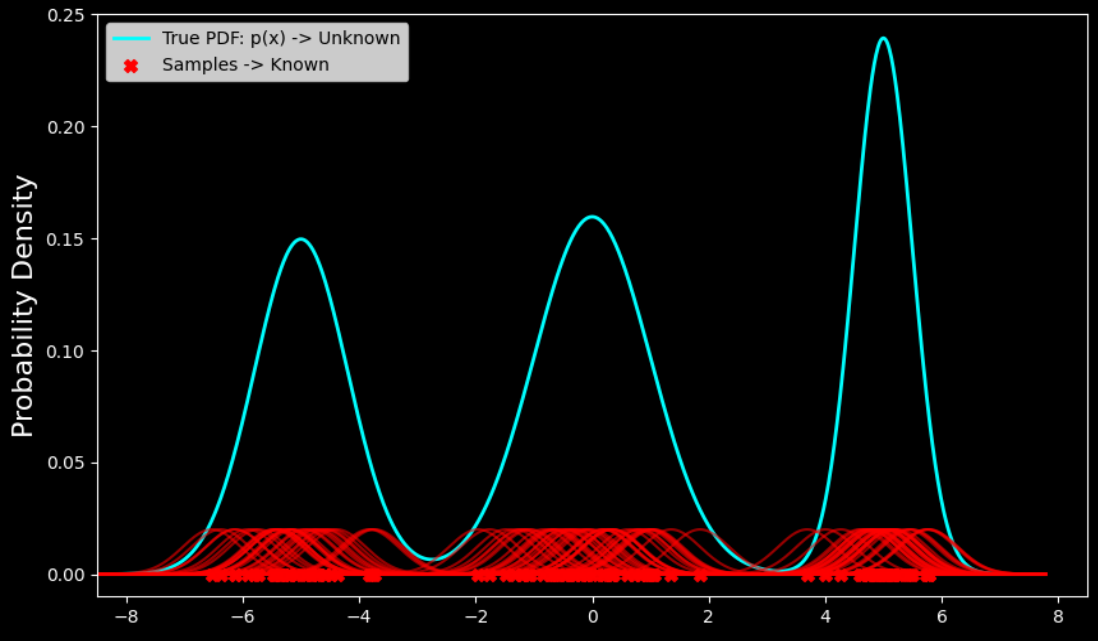
- Given N samples KDE is defined in a naïve way as follows:

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} K_\theta(x, x_n), \text{ where}$$

$$\int K_\theta(x, x_n)dx = 1$$

How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)



- Given N samples KDE is defined in a naïve way as follows:

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} K_\theta(x, x_n), \text{ where}$$

$$\int K_\theta(x, x_n)dx = 1$$

1. We place a kernel K at each sample

- Gaussian kernel is a common choice where σ^2 (also called as bandwidth) determines the smoothness of $\hat{p}(x)$

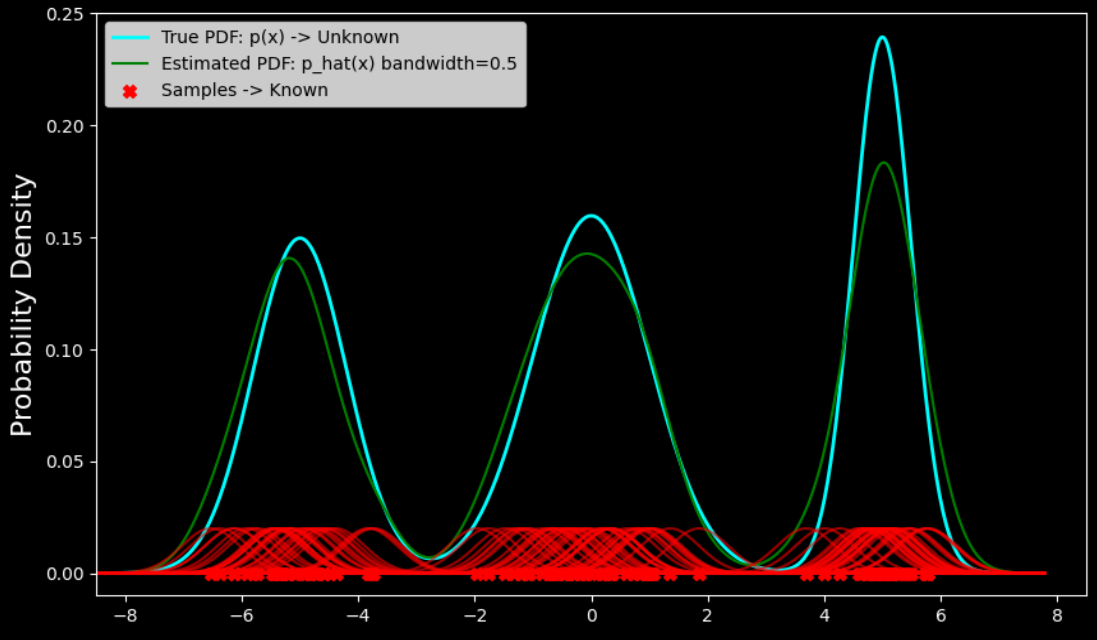
width of function

$$K(x; \mu = x_n, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

point itself

How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)



- Given N samples KDE is defined in a naïve way as follows:

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} K_\theta(x, x_n), \text{ where}$$

$$\int K_\theta(x, x_n)dx = 1$$

1. We place a kernel K at each sample
2. Determine the likelihood of a new sample x based on these kernels

- Gaussian kernel is a common choice where σ^2 (also called as bandwidth) determines the smoothness of $\hat{p}(x)$

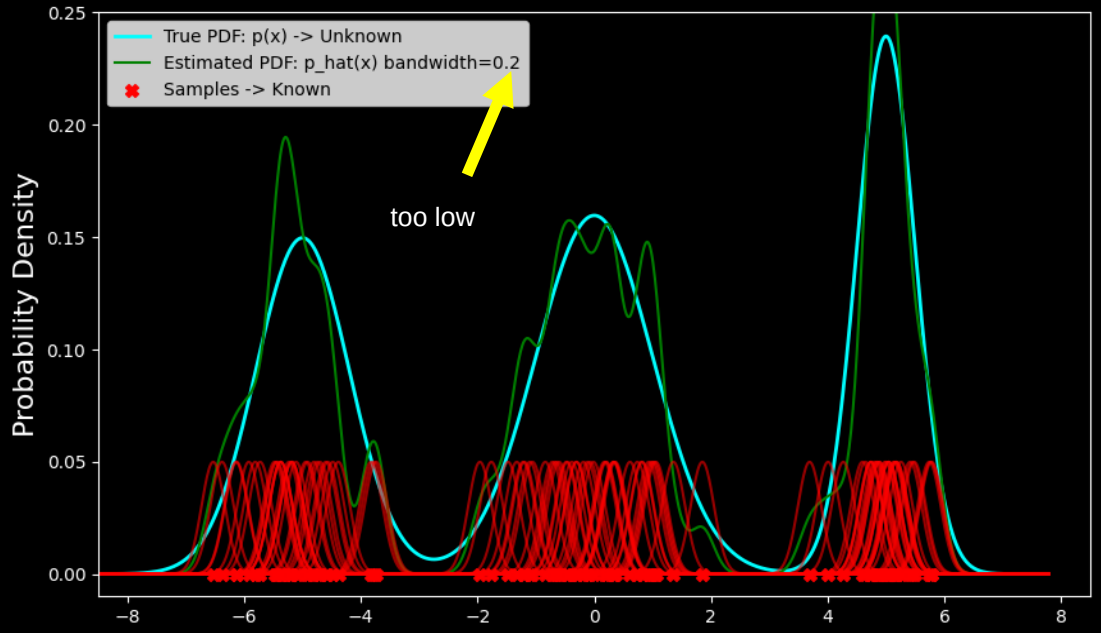
$$K(x; \mu = x_n, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)

- Gaussian kernel is a common choice where σ^2 (also called as bandwidth) determines the smoothness of $\hat{p}(x)$

$$K(x; \mu = x_n, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

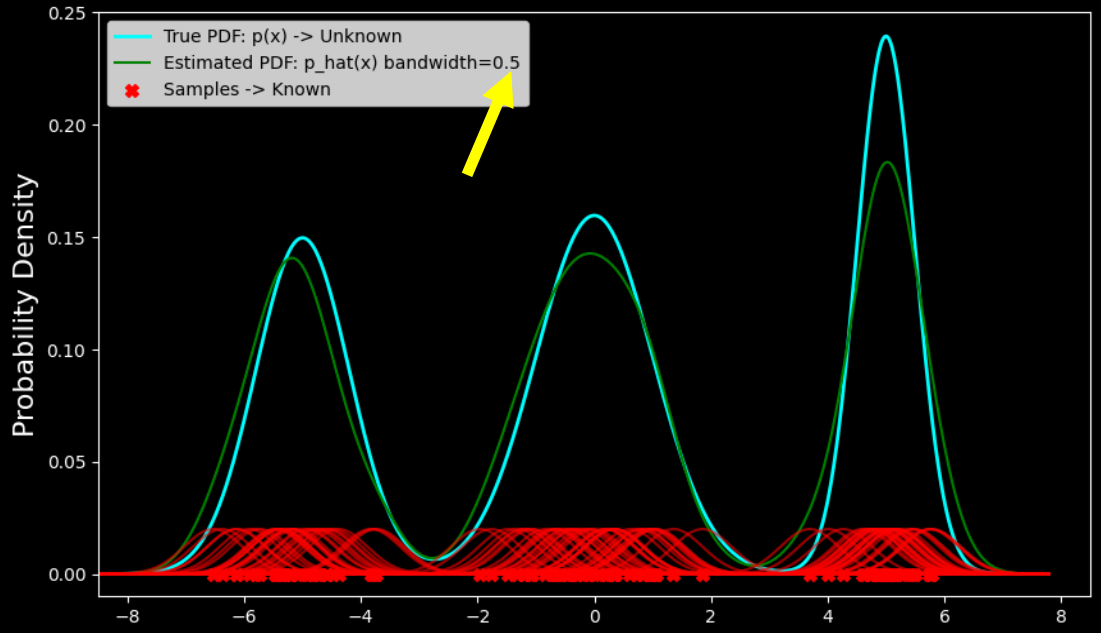


How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)

- Gaussian kernel is a common choice where σ^2 (also called as bandwidth) determines the smoothness of $\hat{p}(x)$

$$K(x; \mu = x_n, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

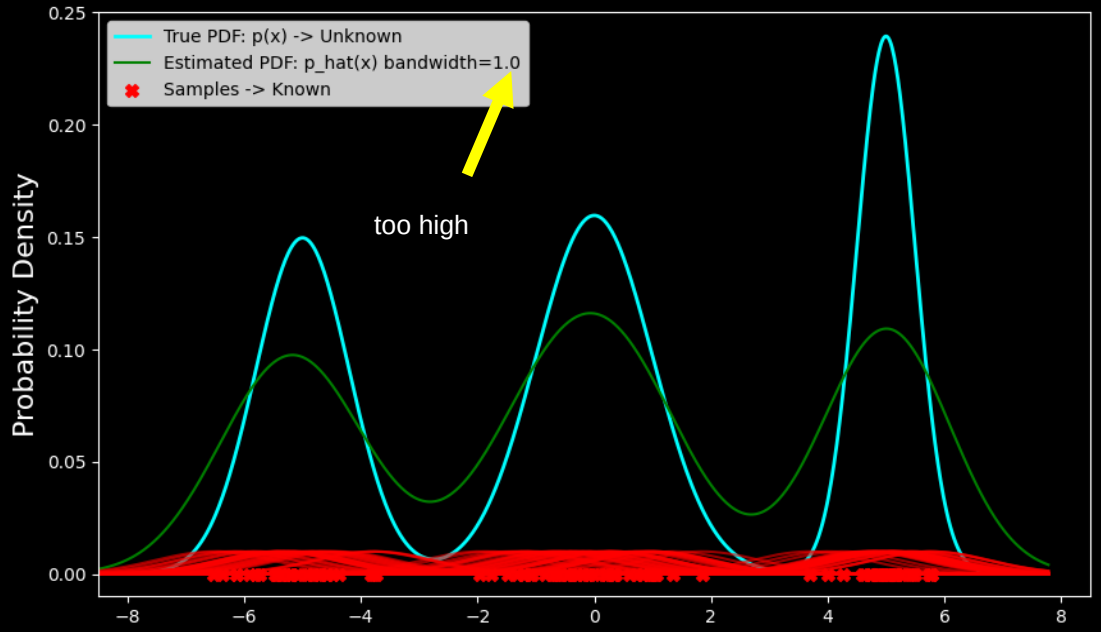


How can we learn the data distribution $p(x)$ from observations?

2. Kernel Density Estimation (KDE)

- Gaussian kernel is a common choice where σ^2 (also called as bandwidth) determines the smoothness of $\hat{p}(x)$

$$K(x; \mu = x_n, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



How can we generate samples from $p(x)$ learned by Kernel Density Estimation (KDE) ?

1. Randomly select one of the N data points uniformly, i.e., each data point has equal probability of being selected

   Remember Inverse Transform Sampling for Discrete Variables

2. Sample from a Gaussian centered on the selected data point, i.e., $N(\mu = x_i, \sigma)$ where x_i is the selected sample

   Remember Inverse Transform Sampling for Continuous Variables

How can we learn the data distribution $p(x)$ from observations?

- We will talk about two methods:
  - ✓ Gaussian Mixture Models (GMMs)
  - ✓ Kernel Density Estimation (KDE)

Summary of what we learned so far…

1. Low-dimensional data

- ✓ How can we evaluate the likelihood $p(x)$?
- ✓ How can we generate samples from $p(x)$?
- ✓ How can we learn the data distribution $p(x)$ from observations?

Summary of what we learned so far…

1. Low-dimensional data

- ✓ How can we evaluate the likelihood $p(x)$?
- ✓ How can we generate samples from $p(x)$?
  - ✓ Inverse Transform Sampling for Discrete and Continuous PDFs
- ✓ How can we learn the data distribution $p(x)$ from observations?
  - ✓ Gaussian Mixture Models (GMMs)
  - ✓ Kernel Density Estimation (KDE)

In this lecture, we will mainly talk about …

1. Low-dimensional data

- ✓ How can we evaluate the likelihood $p(x)$?
- ✓ How can we generate samples from $p(x)$?
- ✓ How can we learn the data distribution $p(x)$ from observations?

2. High-dimensional data -> Deep Generative Models

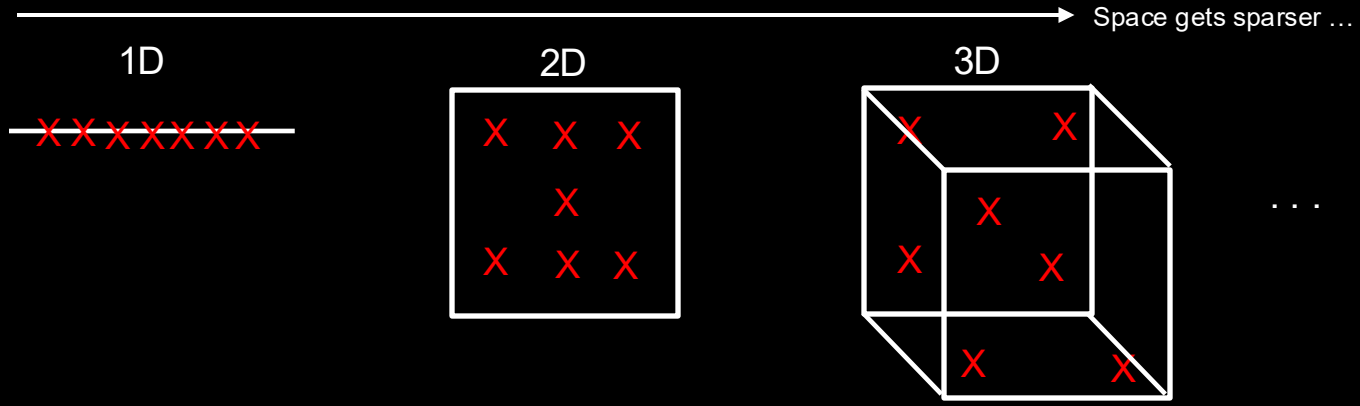
In this lecture, we will mainly talk about …

2. High-dimensional data -> Deep Generative Models

Challenges with High-dimensional data

Challenges with High-dimensional data

- Let's consider RGB images with a resolution of 256×256
- An image can be represented by $256 \times 256 \times 3$ vector
- This means that an image is a point in $256 \times 256 \times 3$ dimensional space



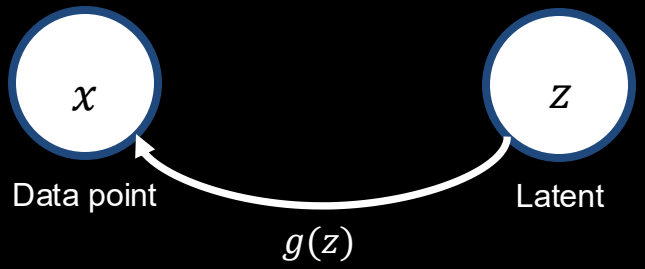
- For GMM and KDE to work, roughly speaking, you need to somehow "fill" the space
- To fill a space of $256 \times 256 \times 3$, you need a lot of samples, we need to find a better solution!

The Basic Idea

- Map a simple distribution $p(z)$ (e.g. a standard Gaussian distribution $N(x; \mu = 0, \sigma = I)$) to the data distribution $p(x)$

- z: Latent variable
- $p(z)$: Latent distribution

- If we have a such mapping $g: z \rightarrow x$, we can sample from $p(z)$ and map it to a data point x



- We use neural network to learn g in Deep Generative Models

A Taxonomy of Deep Generative Models

1. **Latent Variable Models:**
    - ***Implicit Models:*** Generative Adversarial Networks (GANs)
    - ***Prescribed Models:*** Variational Autoencoders (VAEs) and Diffusion Models
2. **Autoregressive Models: Convs, RNNs, Transformers**
3. **Score-based Models: Score matching, Flow matching, Stochastic DEs**
4. **Flow-based Models: Normalizing Flows (NFs), Integer NFs, Continuous NFs**
5. **Energy-based Models: Boltzmann machines, Joint models**

In the rest of the lecture, we will cover

1. **Latent Variable Models:**
    - ***Implicit Models:*** Generative Adversarial Networks (GANs)
    - ***Prescribed Models:*** Variational Autoencoders (VAEs) and Diffusion Models

Generative Adversarial Networks (GANs)

Generative Adversarial Nets

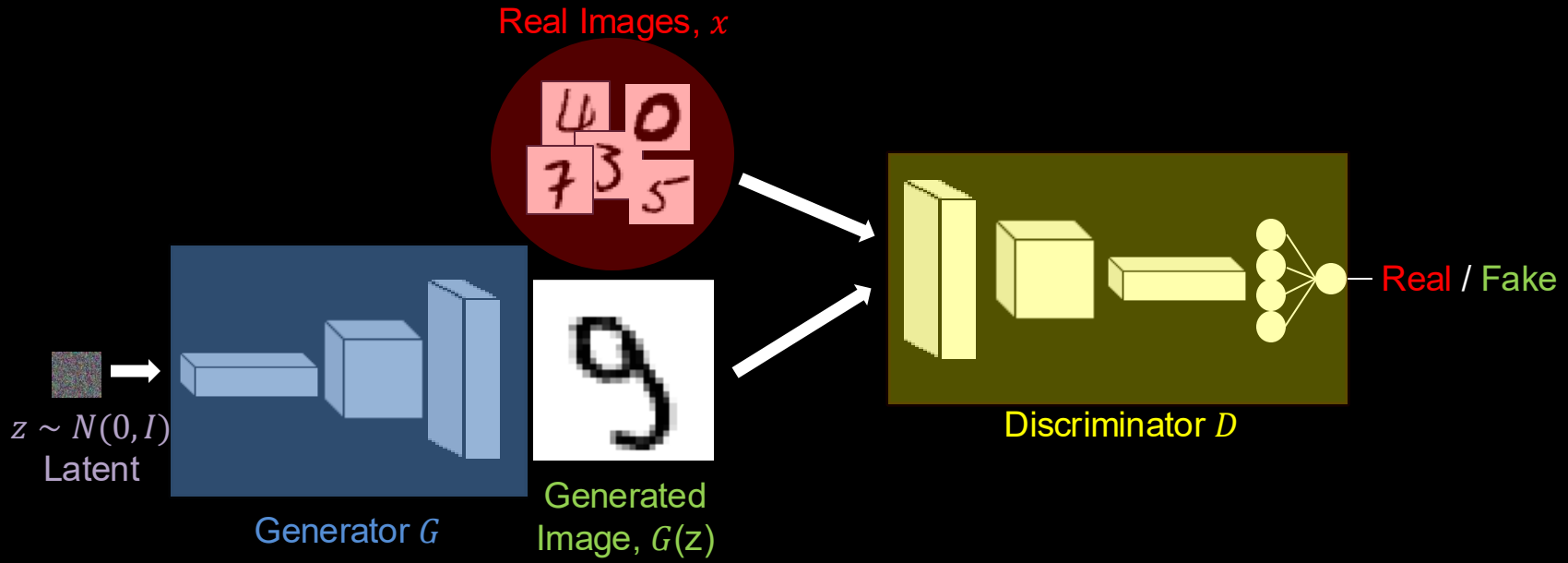
Ian J. Goodfellow, Jean Pouget-Abadie,* Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

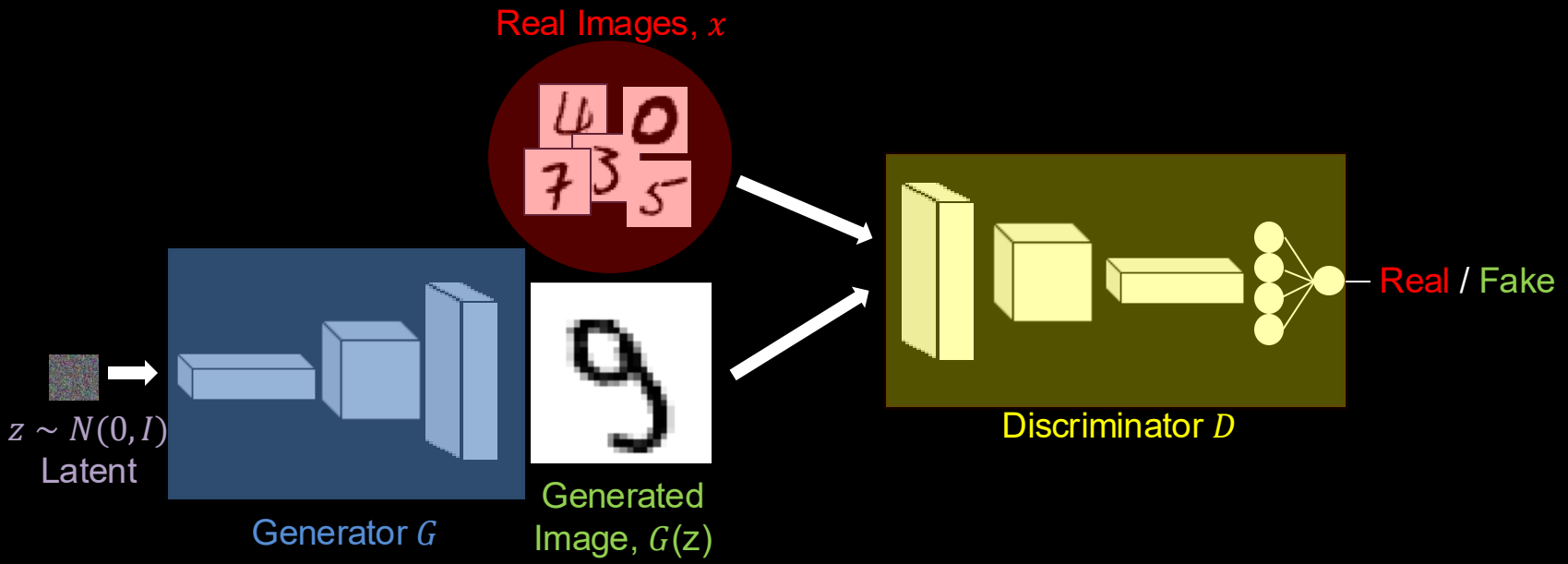
Generative Adversarial Networks (GANs)

Generative Adversarial Nets

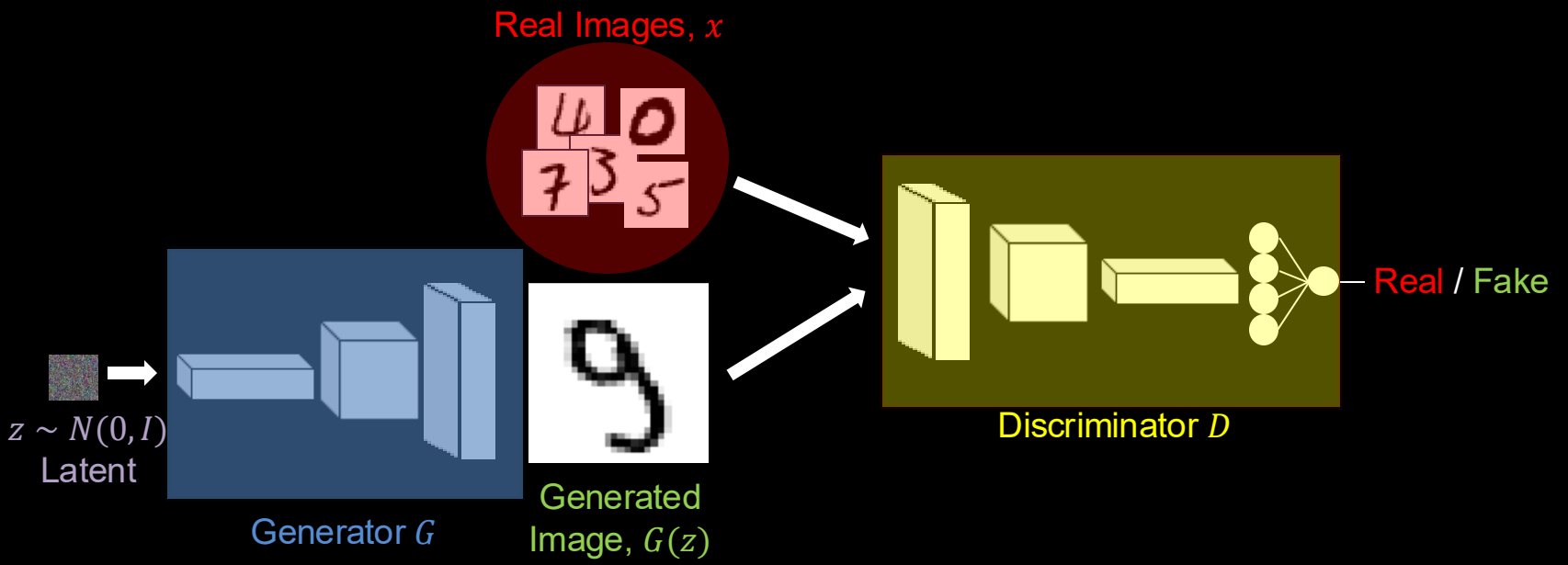
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

- Generator G that takes a latent vector sampled from a unit Gaussian ($N(\mu = 0, \sigma = I)$) as input and generates a synthetic image
- Discriminator D takes either a real or generated image as input and classifies it as either real or fake



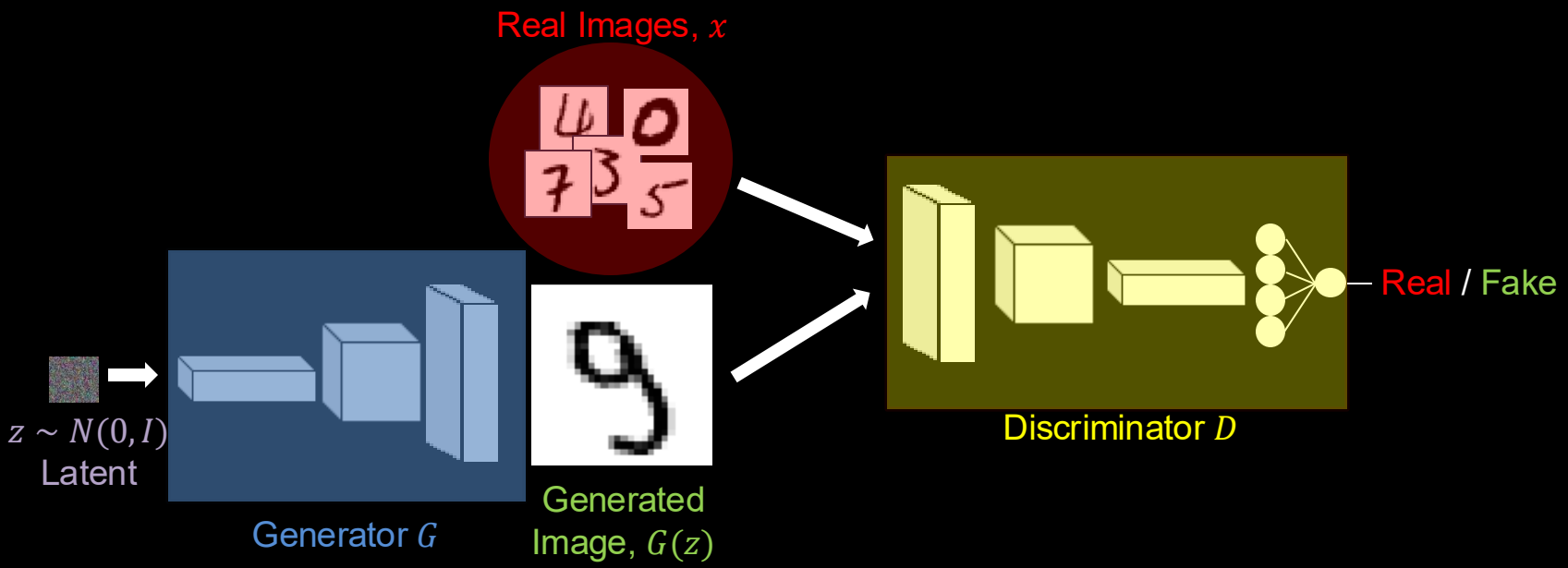


- Generator G and Discriminator D compete against each other!
- G tries to generate real-like images to fool D and D tries to distinguish real and fake images



Loss Function:

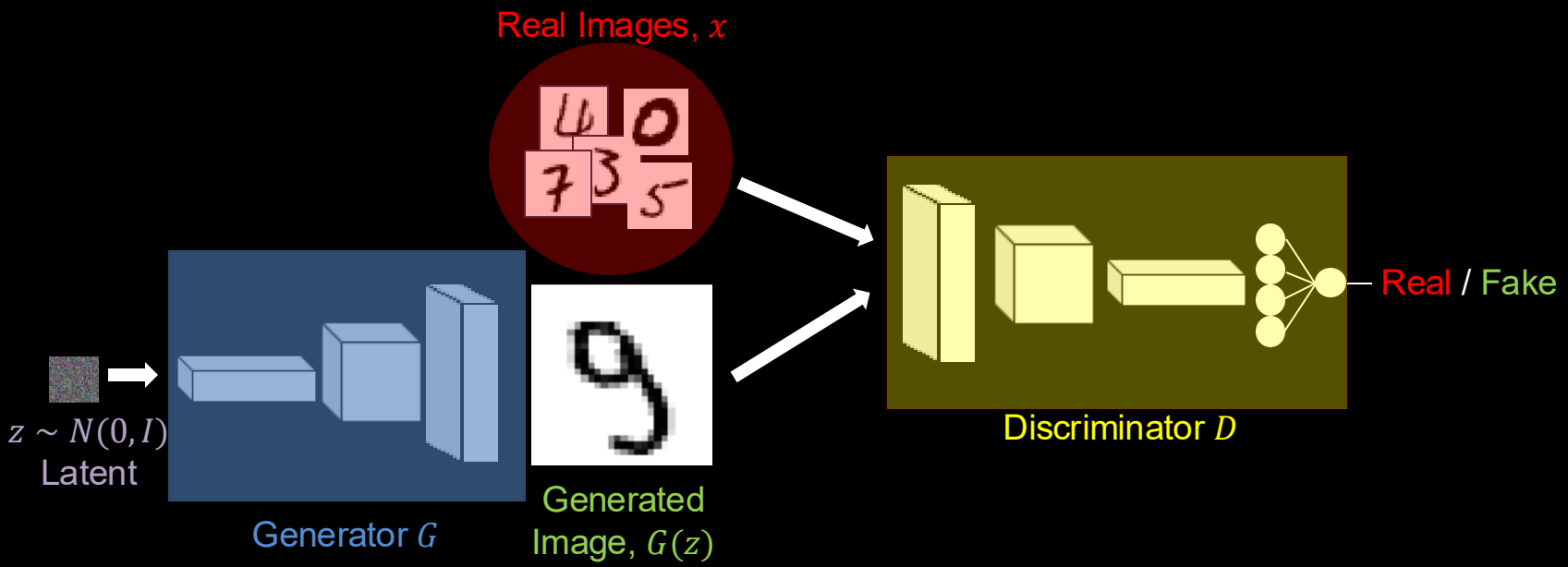
$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

Samples from $p(x)$. We do not need to know $p(x)$, we use the samples we have from it

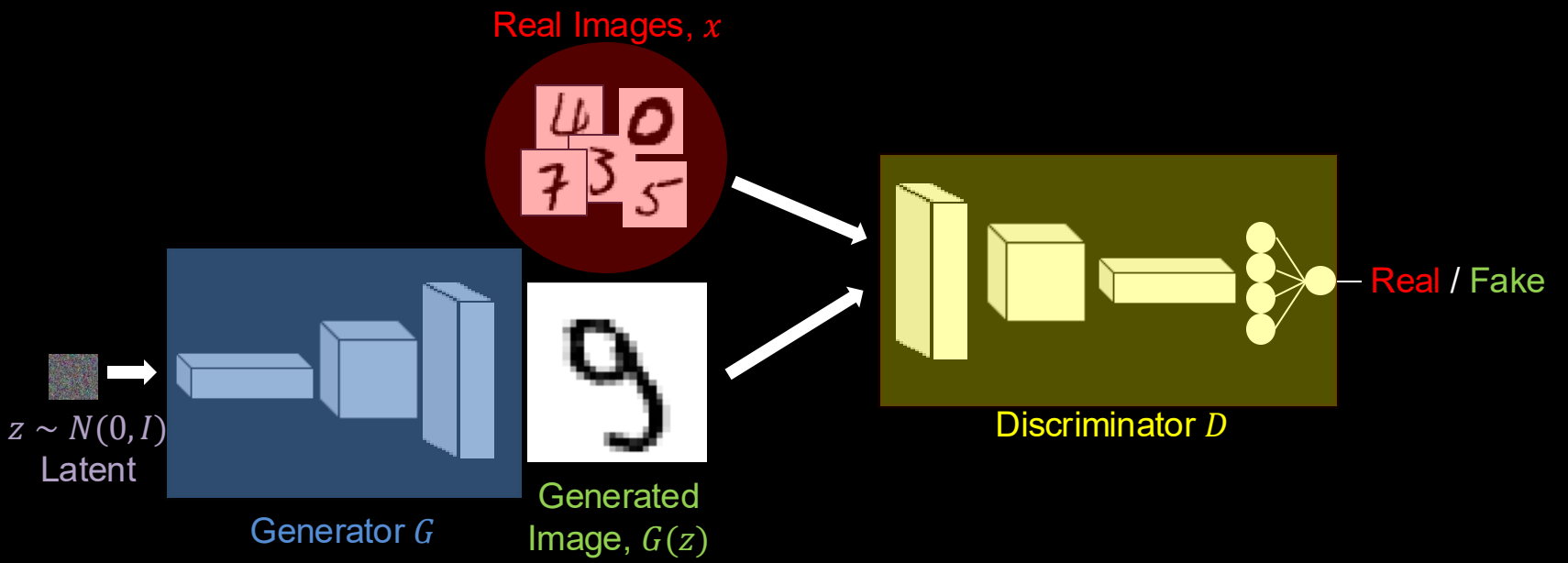


Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

min parameters of G;
max parameters of D

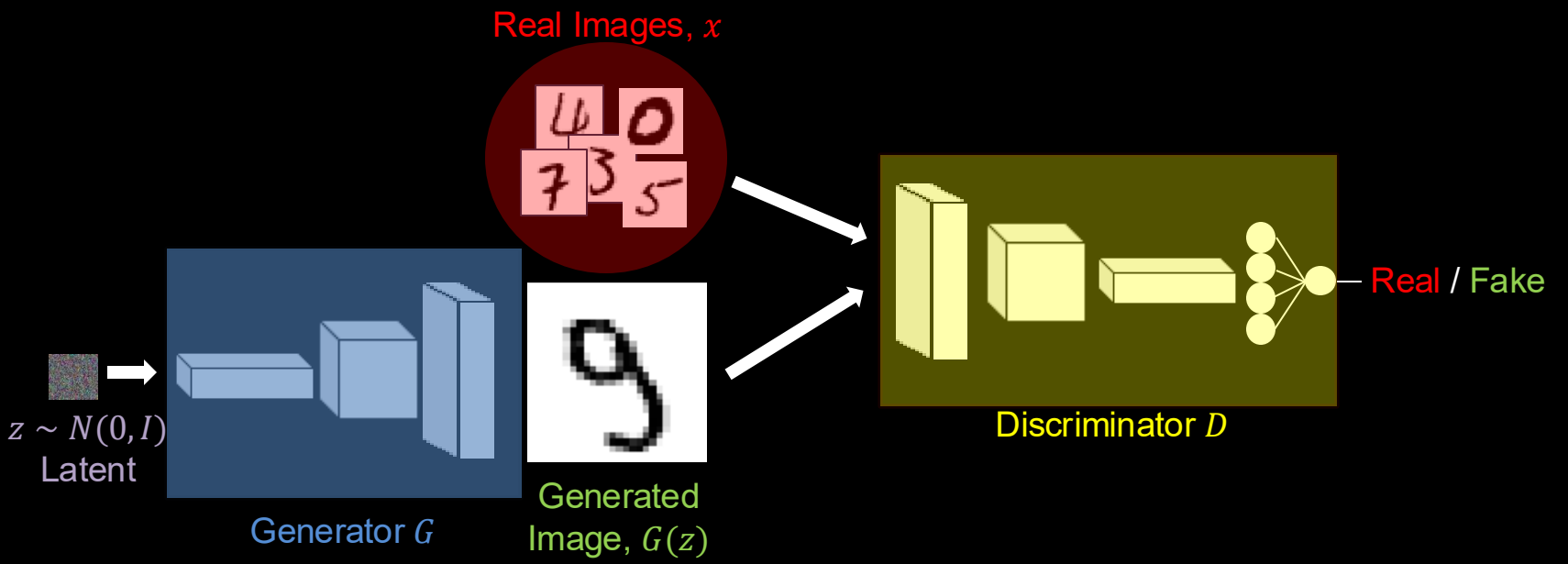
The predicted probability of real image x being a real image



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

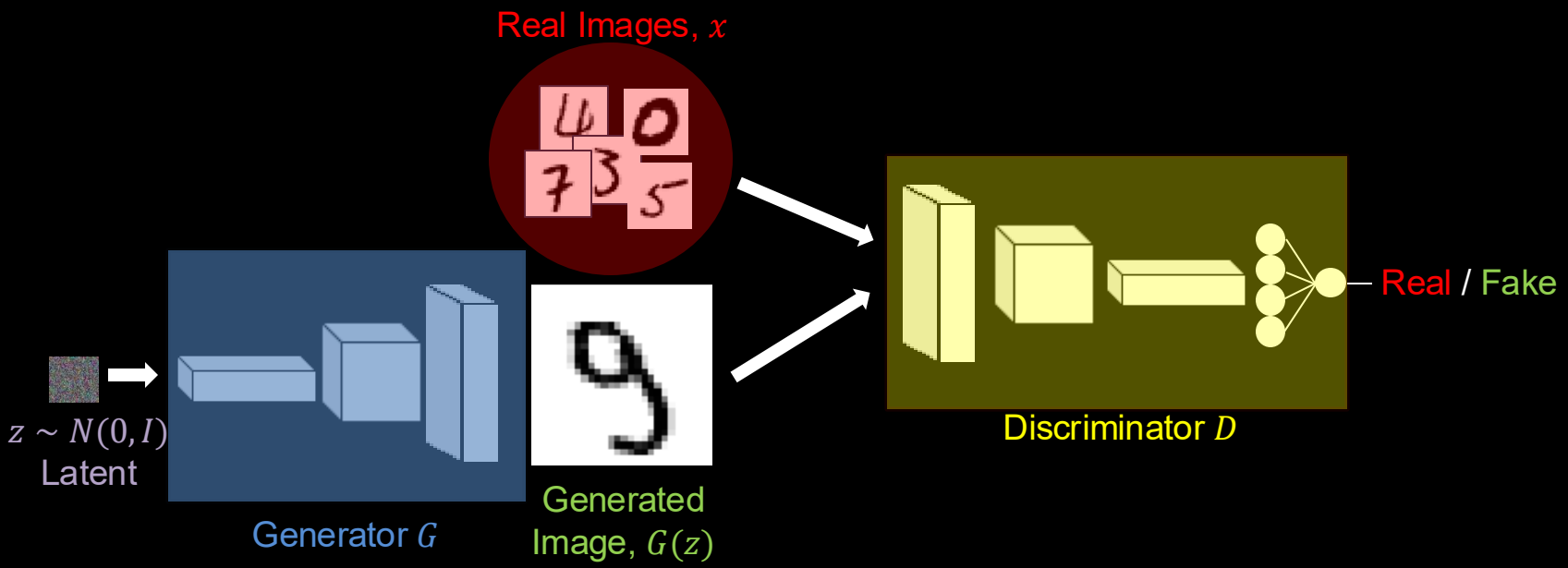
The average predicted probability of all real images x being a real image



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

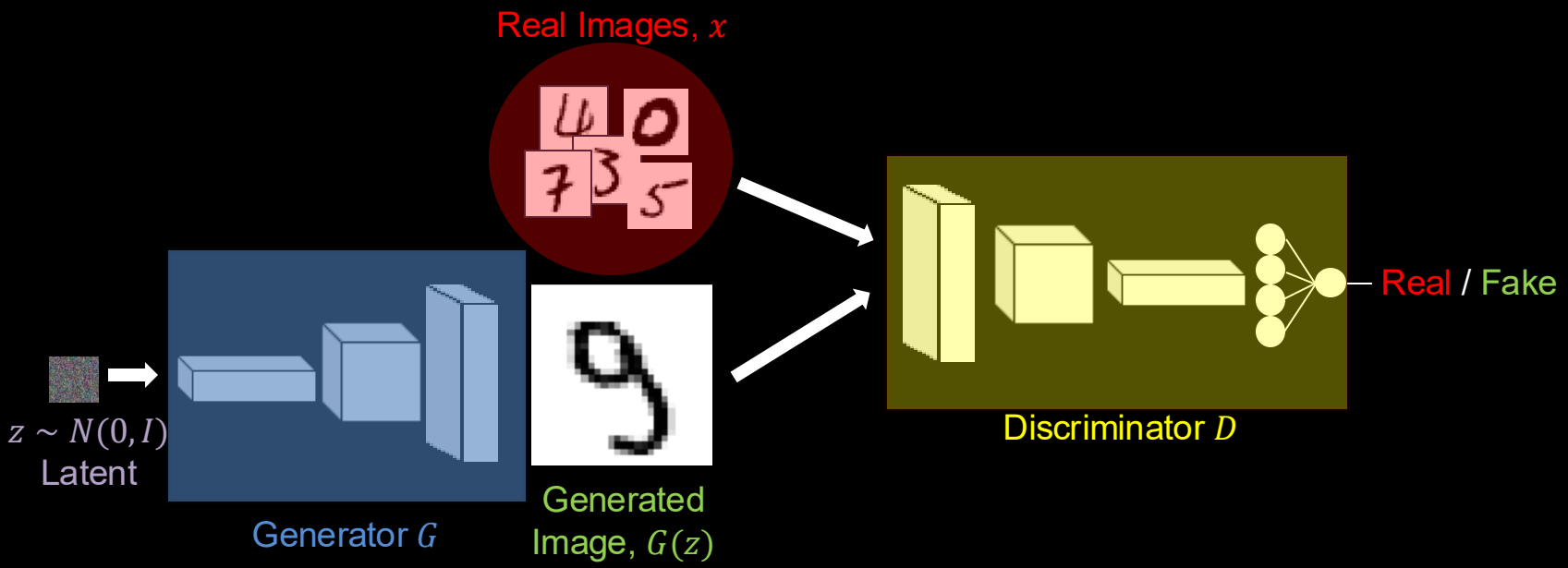
Latent vector sampled from standard Gaussian



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

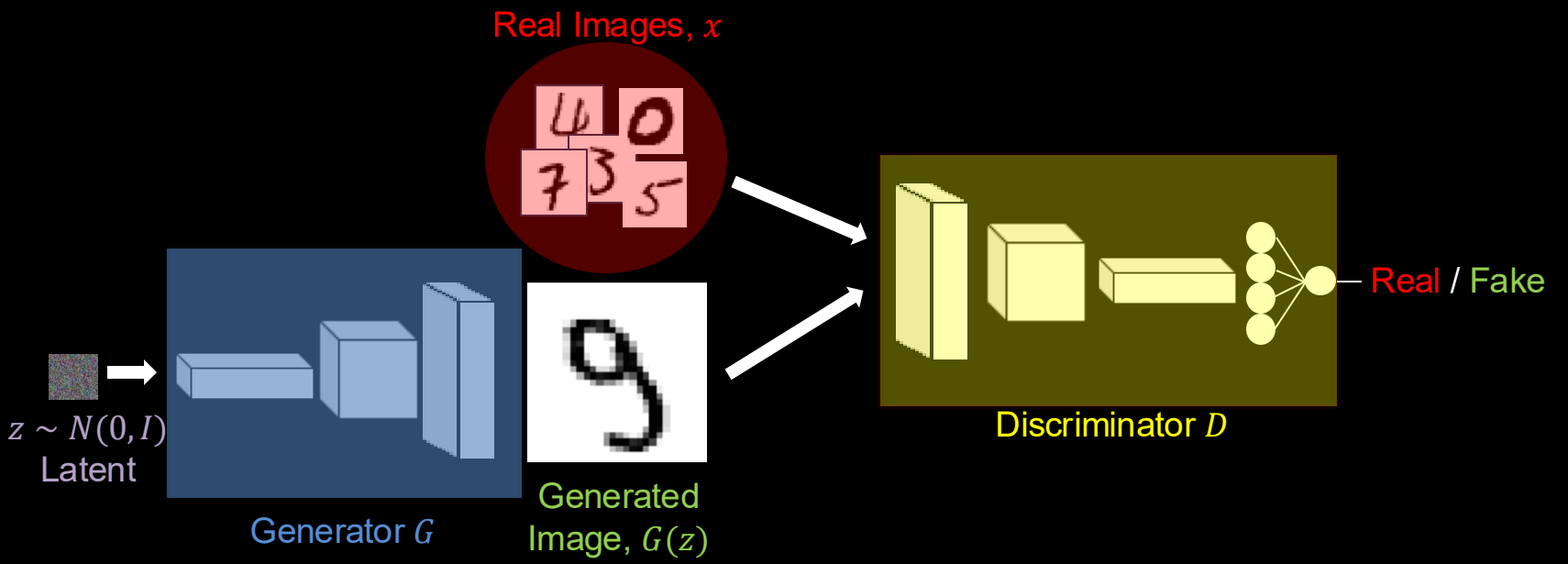
The predicted probability of the fake image $G(z)$ being a real image



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

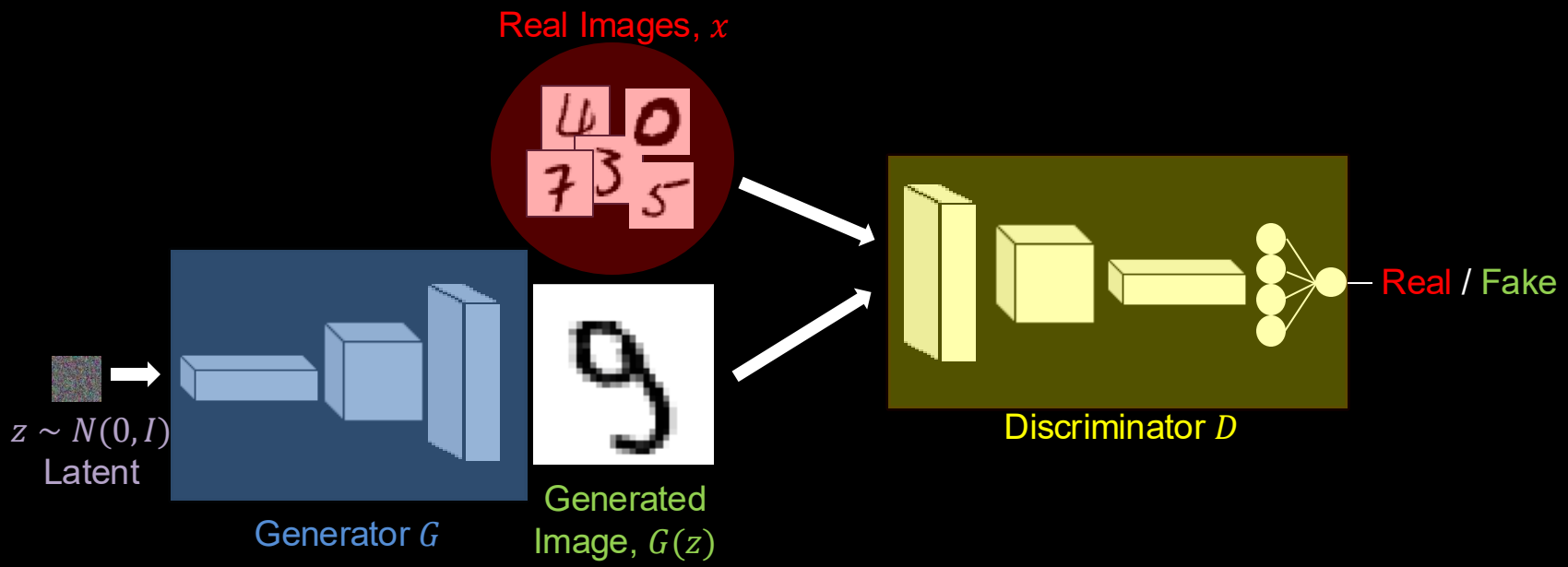
The predicted probability of the fake image $G(z)$ being a fake image



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

The average predicted probability of all fake images $G(z)$ being a fake image



Loss Function:

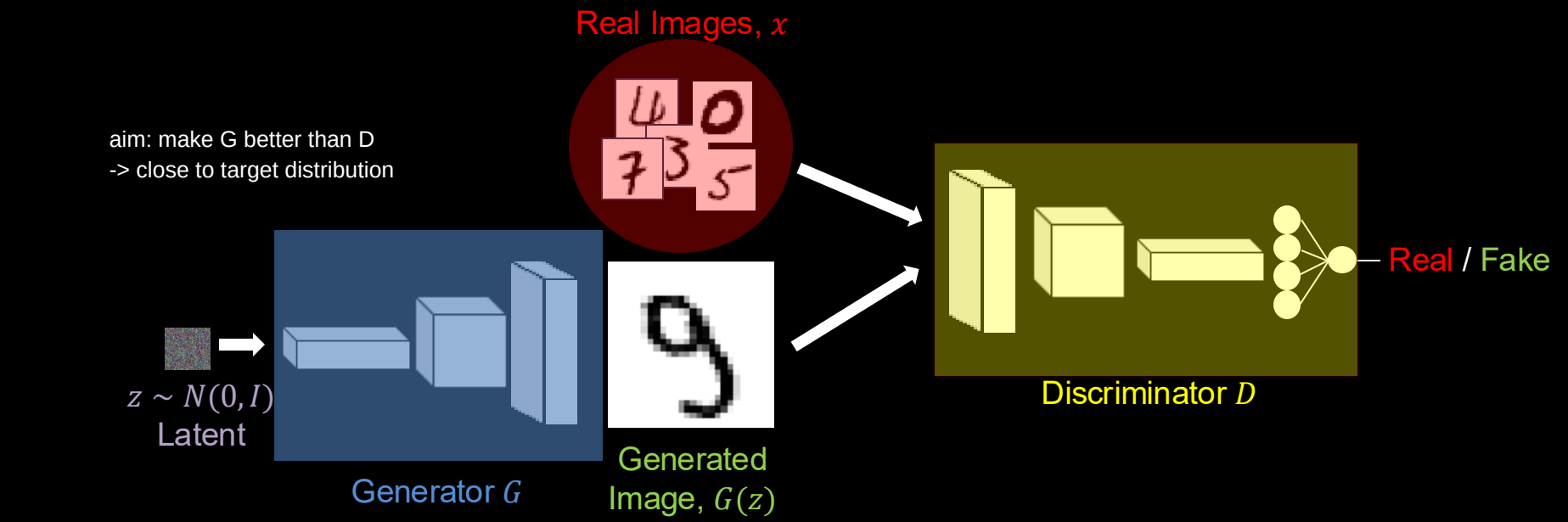
$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

When we maximize with respect to D

D increases the probability real image x being a real image

D decreases the probability of synthetic image $G(z)$ being a real image

This optimization makes D better at distinguishing x and $G(z)$



Loss Function:

$$\min_G \max_D L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

When we minimize with respect to G

This term doesn't have any effect because it doesn't contain G

G increases the probability of the synthetic image $G(z)$ being a real image

This optimization makes G generating more realistic samples

Generative Adversarial Networks (GANs)

Loss Function:

$$\min_{G} \max_{D} L(x, z) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

A min-max optimization problem is known as very difficult to solve

1. **Non-convergence and Instability:** The discriminator becomes too strong and can easily distinguish between real and fake samples, leading to almost zero gradients and halting its learning.

2. **Mode collapse:** The generator tends to collapse to a small part of the data distribution instead of capturing the full diversity of the true data distribution.

Generative Adversarial Networks (GANs)

- How can we evaluate the likelihood $p(x)$?
  - We cannot because we don't estimate $p(x)$
- How can we generate samples from $p(x)$?
  - $z \sim N(0,1)$ and compute $\mathrm{G}(z)$

feed gaussian noise to generator

- How can we learn the data distribution $p(x)$ from observations?
  - GANs do not learn $p(x)$ explicitly, they learn how to sample from it.

Generative Adversarial Networks (GANs)

Index	Software name	Language	Backend	Link	Ref.
1	CGAN	Python	PyTorch	https://github.com/Lornatang/CGAN-PyTorch	[2]
2	DCGAN	Python	PyTorch	https://github.com/Natsu6767/DCGAN-PyTorch	[1], [23], [47]
3	AAEs	Python	TensorFlow	https://github.com/conan7882/adversarial-autoencoders	[117]
4	InfoGAN	Python	TensorFlow	https://github.com/openai/InfoGAN	[14]
5	SAD-GAN	–	–	–	[120]
6	LSGAN	Python	PyTorch	https://github.com/xudonmao/LSGAN	[121]
7	SRGAN	Python	TensorFlow	https://github.com/tensorlayer/SRGAN	[124], [186], [187]
8	WGAN	Python	PyTorch	https://github.com/Zeleni9/pytorch-wgan	[109], [122]
9	CycleGAN	Python	TensorFlow	https://github.com/junyanz/CycleGAN	[3], [188]
10	ProGAN	Python	PyTorch	https://github.com/tkarras/progressive_growing_of_gans	[5]
11	MidiNet	Python	TensorFlow	https://github.com/RichardYang40148/MidiNet	[8]
12	SN-GAN	Python	PyTorch	https://github.com/hanyoseob/pytorch-SNGAN	[133]
13	RGAN	Python	TensorFlow	https://github.com/ratschlab/RGAN	[134], [189]
14	StarGAN	Python	PyTorch	https://github.com/yunjey/stargan	[138]
15	BigGAN	Python	PyTorch	https://github.com/ajbrock/BigGAN-PyTorch	[110]
16	MI-GAN	Python	TensorFlow	https://github.com/hazratali/MI-GAN	[146]
17	AttGAN	Python	TensorFlow	https://github.com/LynnHo/AttGAN-Tensorflow	[148], [190]
18	PATE-GAN	Python	TensorFlow	https://github.com/vanderschaarlab/mlforhealthlabpub/tree/main/alg/pategan	[161]
19	DM-GAN	Python	PyTorch	https://github.com/MinfengZhu/DM-GAN	[152]
20	SinGAN	Python	PyTorch	https://github.com/tamarott/SinGAN	[158]
21	POLY-GAN	Python	PyTorch	https://github.com/nile649/POLY-GAN	[95]
22	MIEGAN	–	–	–	[168]
23	VQGAN	Python	PyTorch	https://github.com/dome272/VQGAN-pytorch	[169], [191]
24	DALL-E	Python	PyTorch	https://github.com/lucidrains/DALLE-pytorch	[172], [173]
25	CEGAN	–	–	–	[99]
26	Seismogen	Python	PyTorch	https://github.com/Miffka/seismogen	[87]
27	MetroGAN	Python	PyTorch	https://github.com/zwy-Giser/MetroGAN	[84]
28	M3GAN	Python	PyTorch	https://github.com/SLZWVICTOR/M3GAN	[102]
29	CNTS	Python	PyTorch	https://github.com/BomBooooo/CNTS/tree/main	[103]
30	RidgeGAN	Python	PyTorch	https://github.com/rahisha-thottolil/ridgegan	[10]

In the rest of the lecture, we will cover

1. **Latent Variable Models:**
   - ✓ ***Implicit Models:*** Generative Adversarial Networks (GANs)
   - • ***Prescribed Models:*** Variational Autoencoders (VAEs) and Diffusion Models

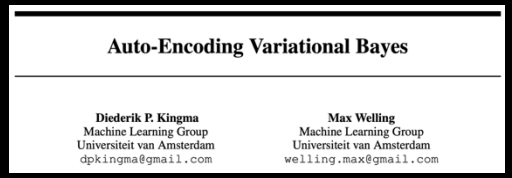
Variational Autoencoders (VAEs)

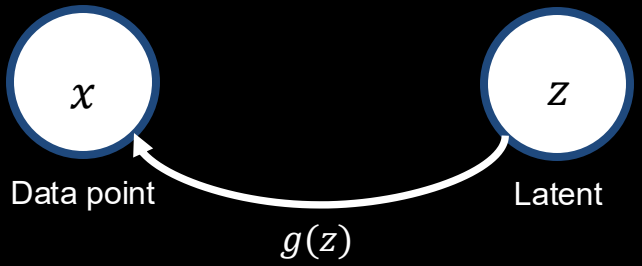
Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Variational Autoencoders (VAEs)





$$p(z) = N(z; 0, I)$$

$$p(x|z) = N(x; g(z), \sigma I)$$

For all given images, we want to maximize the marginal probability $p(x)$

$$p(x) = \int p(x|z)p(z)dz$$???

How to compute the integral?

Intractable, i.e., cannot be solved standard analytical methods or closed form solutions

Variational Autoencoders (VAEs)

- Instead of directly maximizing $p(x)$

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

- We maximize the lower bound of $\log p(x)$:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]}_{KL[q_\phi(z|x)||p(z)]}$$

kr-divergence -> how similar distributions are

-> want to minimize

Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

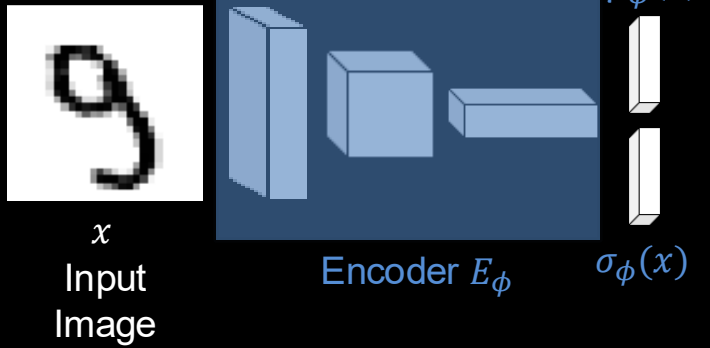
$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]$$

Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]$$

$$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma_\phi(x))$$



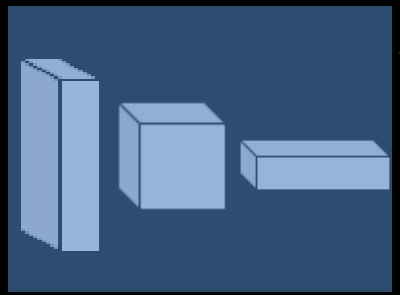
Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]$$

$$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma_\phi(x))$$

x
Input Image



Encoder E_ϕ

$\mu_\phi(x)$

$\sigma_\phi(x)$

$$\eta \sim p(z) = N(0,1)$$

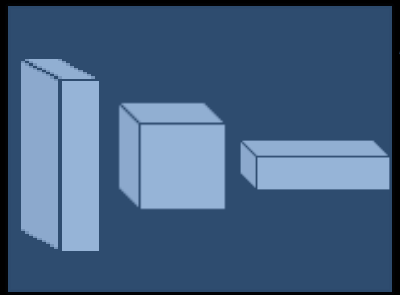
Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]$$

$$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma_\phi(x))$$

x
Input Image



Encoder E_ϕ

$\mu_\phi(x)$

$\sigma_\phi(x)$

$z = \mu_\phi(x) + \eta \times \sigma_\phi(x)$

$\eta \sim p(z) = N(0,1)$

Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

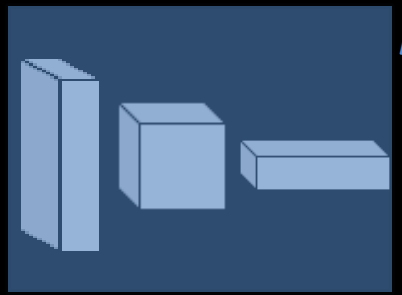
$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]$$

$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma_\phi(x))$

$p_\theta(x|z) = N(z; D_\theta(z), I)$

max-pooling (?)

x
Input Image



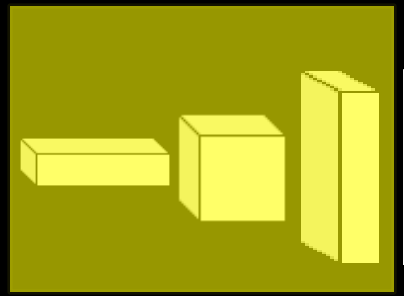
Encoder E_ϕ

$\mu_\phi(x)$

$\sigma_\phi(x)$

$z = \mu_\phi(x) + \eta \times \sigma_\phi(x)$

$\eta \sim p(z) = N(0,1)$



Decoder D_θ

$\hat{x} \sim p_\theta(x|z)$
Reconstructed Image

-> when generating, we get rid of the encoder and directly

Variational Autoencoders (VAEs)

- We maximize the lower bound of $\log p(x)$:

MSE between input and reconstruction

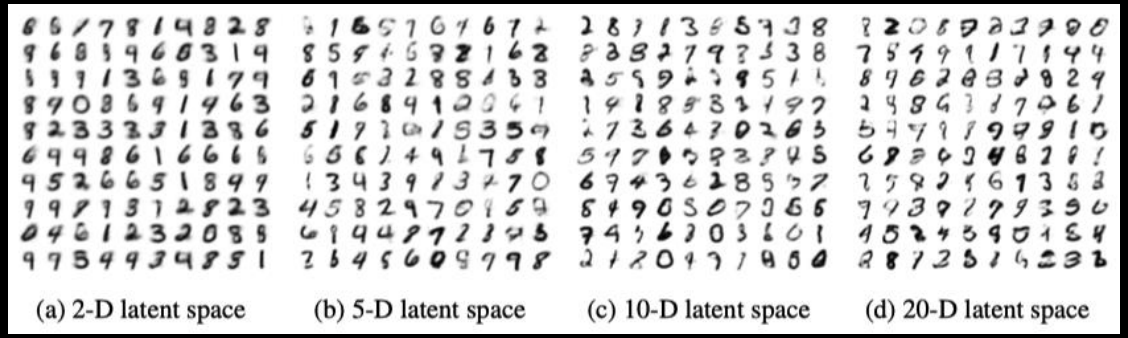
$$\max_{\phi,\theta} \log p(x) \geq \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Reconstruction term to be maximized (negative MSE)}} - \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log q_\phi(z|x) - \log p(z)]}_{\text{Prior matching term to be minimized}}$$

Mean Squared Error

Variational Autoencoders (VAEs)

- How can we evaluate the likelihood $p(x)$?
  - We evaluate the lower bound of $p(x)$
- How can we generate samples from $p(x)$?
  - $z \sim N(0,1)$ and $\hat{x} \sim p_\theta(x|z)$ generate sample from unit gaussian, pass throuigh decoder
- How can we learn the data distribution $p(x)$ from observations?
  - We can approximately learn $p(x)$ by maximizing its lower bound

Variational Autoencoders (VAEs) Results on MNIST and CelebA Datasets



(a) 2-D latent space (b) 5-D latent space (c) 10-D latent space (d) 20-D latent space

Hierarchical Architectures for VAEs

NVAE: A Deep Hierarchical Variational Autoencoder

Arash Vahdat, Jan Kautz
NVIDIA
{avahdat, jkautz}@nvidia.com

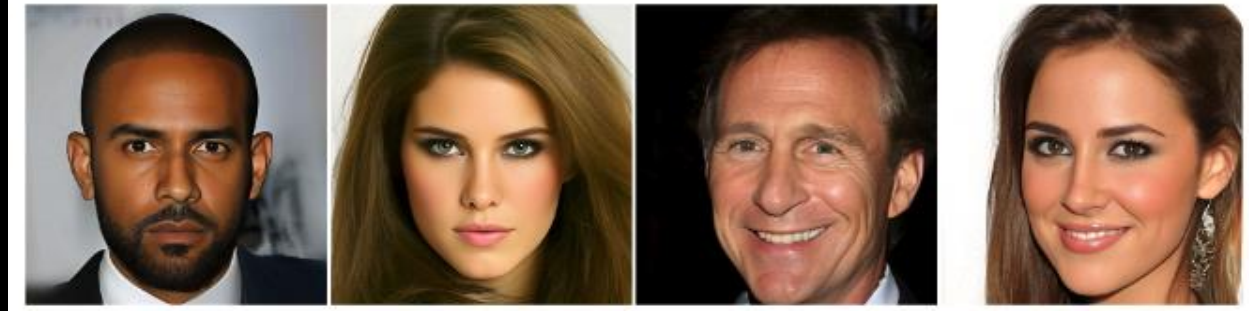
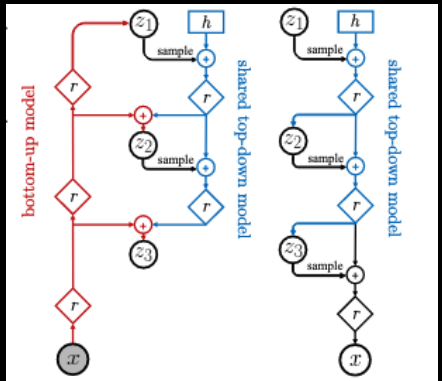


Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].



(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. ◇ denotes residual neural networks, ⊕ denotes feature combination (e.g., concatenation), and h is a trainable parameter.

Through a hierarchical architecture for the latent space, and an increased latent dimension, VAEs as well can generate very realistic images.

In the rest of the lecture, we will cover

1. **Latent Variable Models:**
   - ✓ ***Implicit Models:*** Generative Adversarial Networks (GANs)
   - • ***Prescribed Models:*** Variational Autoencoders (VAEs) and Diffusion Models

Diffusion Models

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

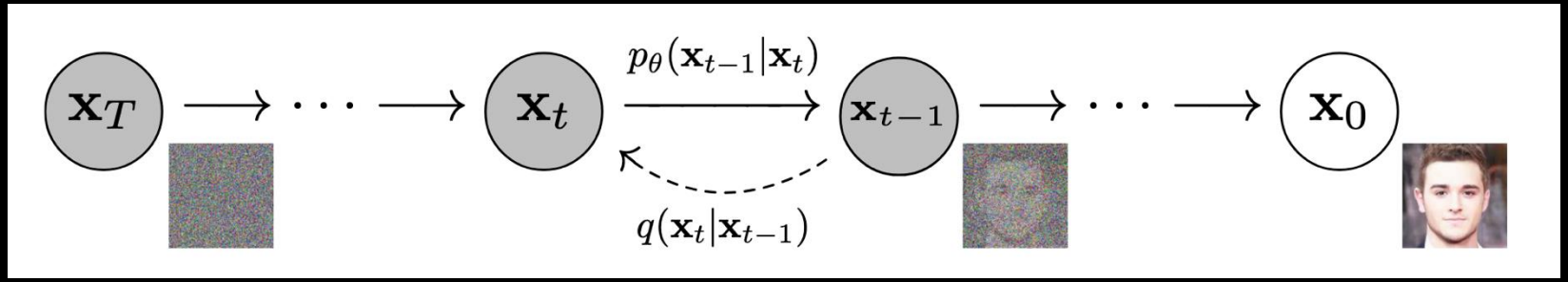
Diffusion Models

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

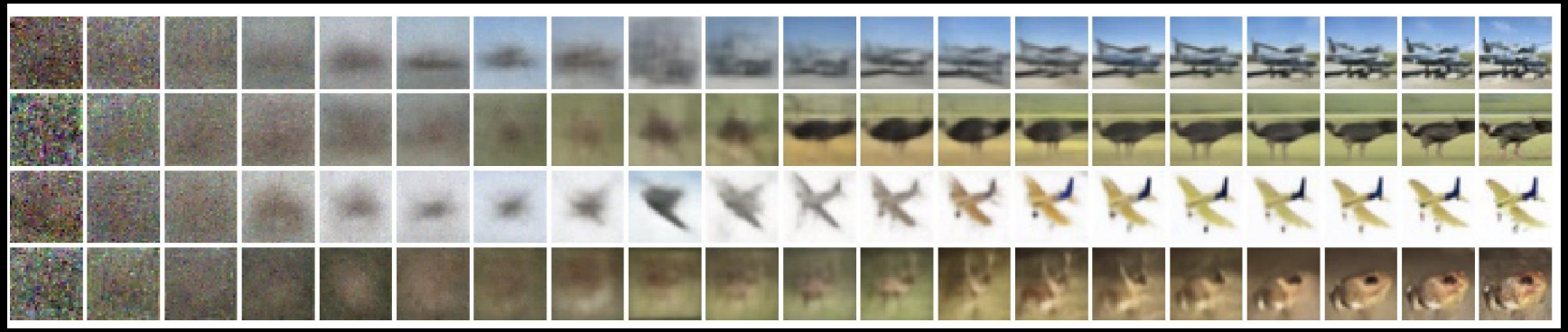


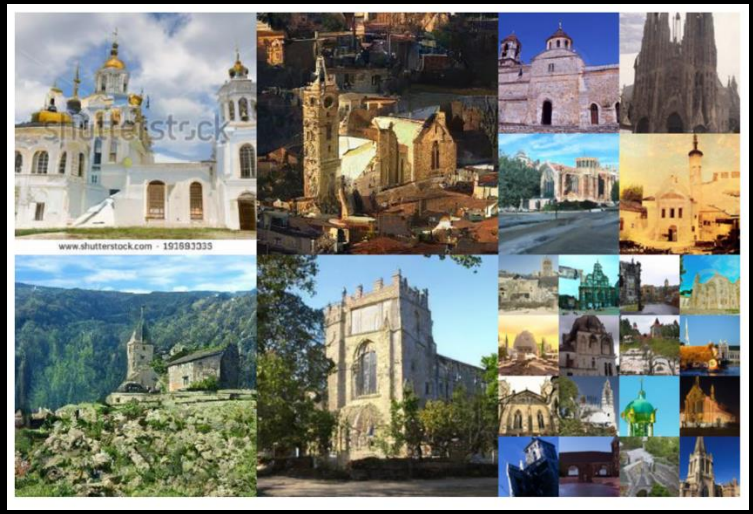
- Diffusion models consists of two steps: forward and backward processes:
- **Forward process**: we progressively add noise to an image to make if a pure noise image
- **Backward process**: we progressively predict the added noise and remove it from the image. This is modelled by a neural network.

Variational Autoencoders (VAEs)

- How can we evaluate the likelihood $p(x)$?
  - We evaluate the lower bound of $p(x)$
- How can we generate samples from $p(x)$?
  - $z \sim N(0,1)$, $\mathrm{f}(z) = \hat{z}$
- How can we learn the data distribution $p(x)$ from observations?
  - We can approximately learn $p(x)$ by maximizing its lower bound

Diffusion Models generate very realistic images from noise





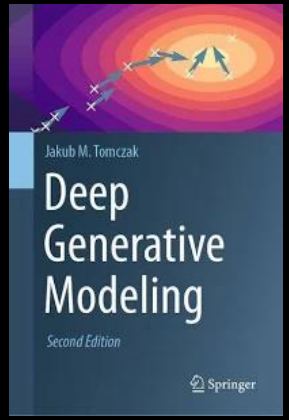
A full course on Deep Generative Models in Spring 2026

Thursday 14:15 – 16:00 (3 Credits)

1. **Latent Variable Models:**
    - ***Implicit Models:*** Generative Adversarial Networks (GANs)
    - ***Prescribed Models:*** Variational Autoencoders (VAEs) and Diffusion Models
2. **Autoregressive Models: Convs, RNNs, Transformers**
3. **Score-based Models: Score matching, Flow matching, Stochastic DEs**
4. **Flow-based Models: Normalizing Flows (NFs), Integer NFs, Continuous NFs**
5. **Energy-based Models: Boltzmann machines, Joint models**

Further Materials

1. Jakub M. Tomczak, Deep Generative Modeling



2. Online lecture: Diffusion Models and Their Applications by Minhyuk Sung, KAIST

   https://mhsung.github.io/kaist-cs492d-fall-2024/

3. Online lecture: Deep Generative Models: Probabilistic Foundations and learning algorithms by Volodymyr Kuleshov, Cornell Tech

   https://kuleshov-group.github.io/dgm-website/