Generative Models

Nikita Kiselev

Deep Learning, Intelligent Systems

November 17, 2025

Nikita Kiselev

Research Interests
1) Generative AI
2) Computer Vision
3) Optimization

Topics in this course
1) Generative Models
2) Acceleration

Education

Sep 2024 – Present
MSc in Computer Science
Intelligent Systems, MIPT

Sep 2020 – Aug 2024
BSc in Applied Mathematics and Physics
Intelligent Systems, MIPT

Experience

Jun 2024 – Present
Research Engineer
Kandinsky, Sber AI

Sep 2024 – Jul 2025
Mathematician-programmer
Research Center for Artificial Intelligence, Innopolis University

Oct 2023 – Apr 2024
Technician
Laboratory of Mathematical Methods of Optimization, MIPT

Which face is real?

1

2

3

Which face is real?

1

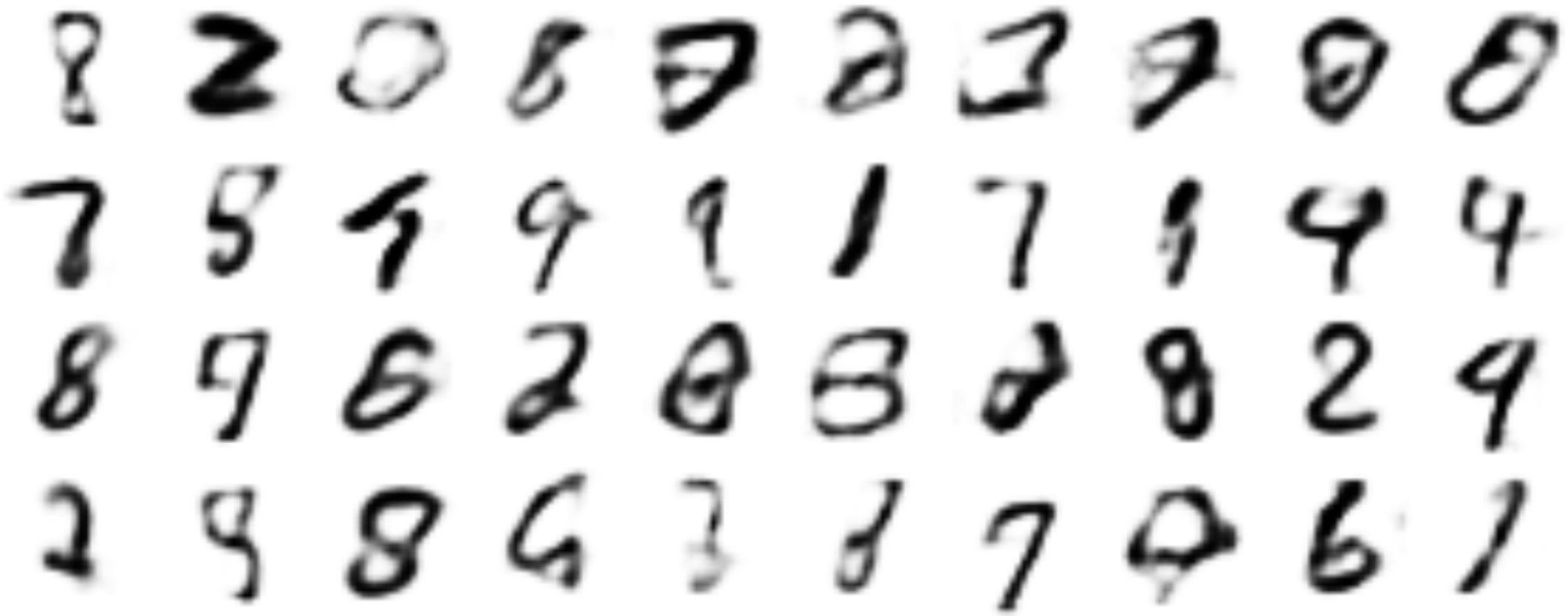
2

3

NONE OF THEM! ALL ARE AI-GENERATED!

Evolution of Generative Models

VAE – first scalable approach



Kingma D. P., Welling M. Auto-encoding variational bayes, 2013

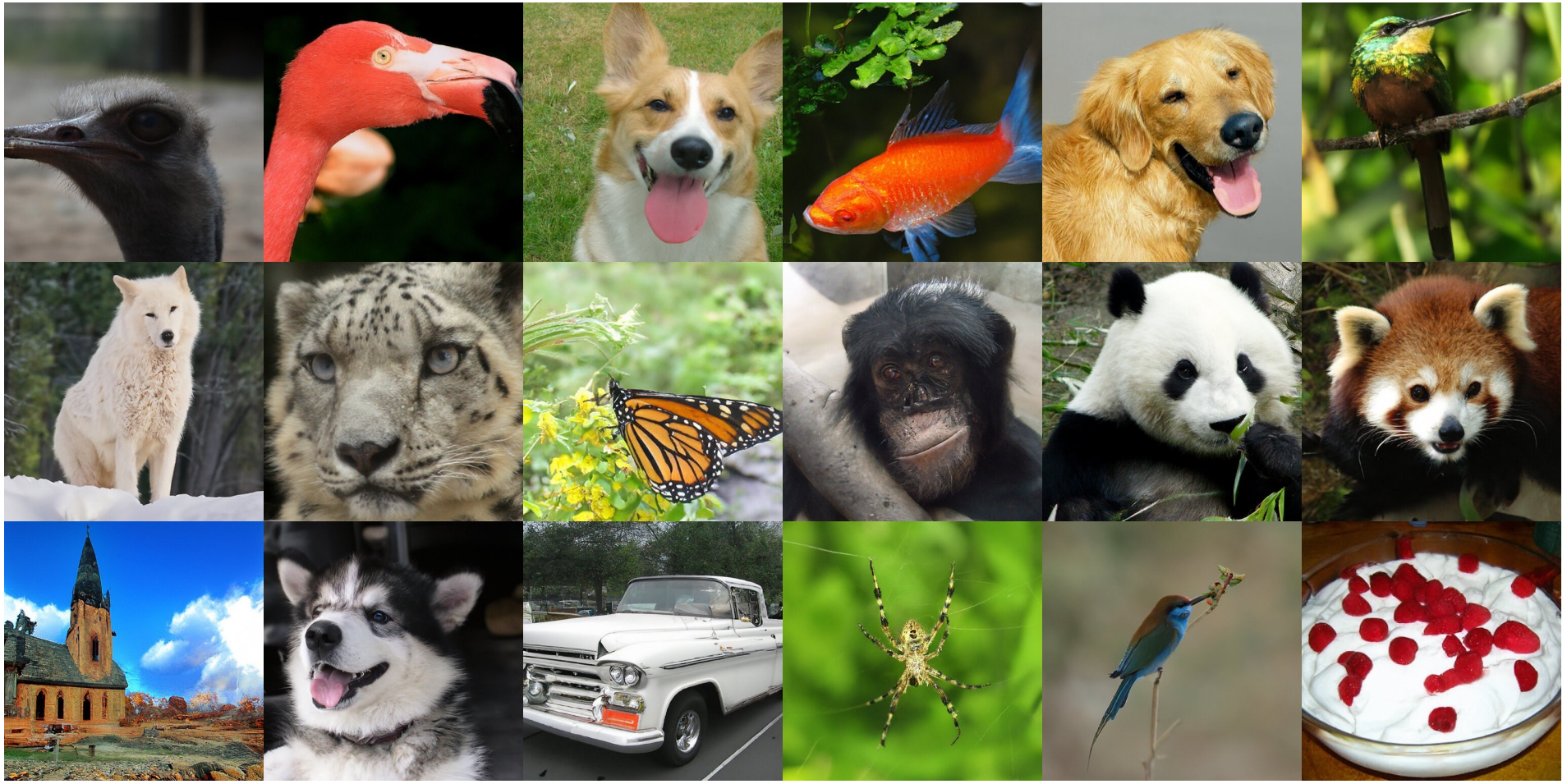
DCGAN – first convolutional GAN

Radford A., Metz L., Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015

StyleGAN – high quality faces

Karras T., Laine S., Aila T. A style-based generator architecture for generative adversarial networks, 2018

Denoising Diffusion Probabilistic Models



Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021

FLUX – cutting-edge flow matching model

https://flux1ai.com/dev

Veo3 – SOTA video + audio generation

https://deepmind.google/models/veo/

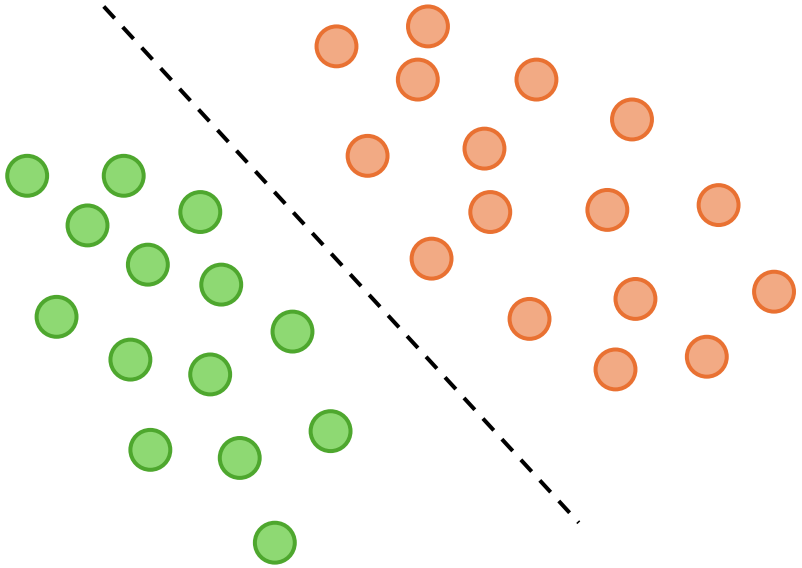
Contents

- What are Generative Models?
- Taxonomy
- Autoregressive Models
- Autoencoders
- VAEs
- GANs

* Extra: Modern Architectures

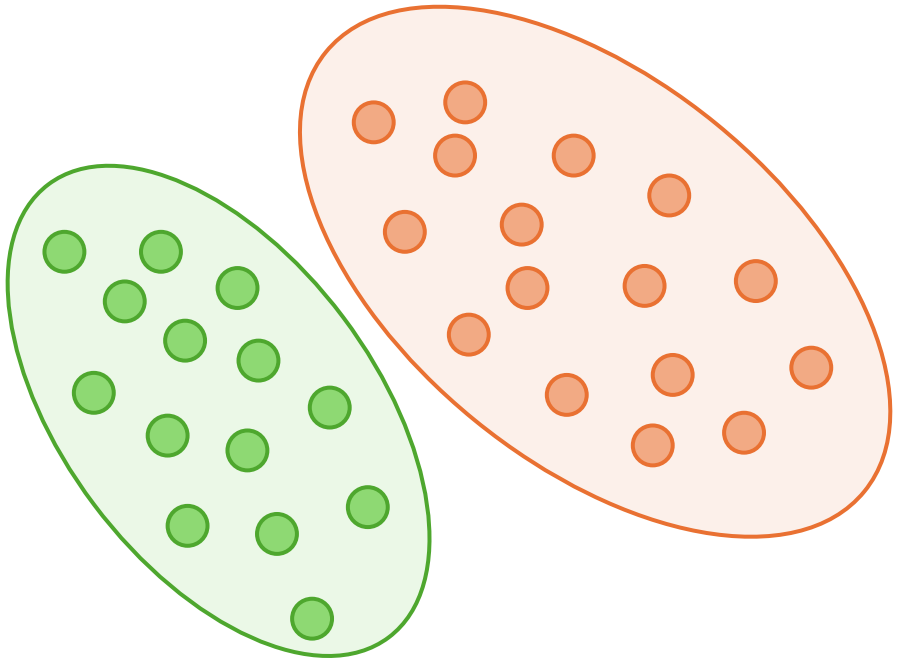
What are Generative Models?

Discriminative vs Generative Models



Discriminative

$p(\mathbf{y}|\mathbf{x})$



Generative

$p(\mathbf{x}, \mathbf{y})$

Discriminative vs Generative Models

Discriminative

- Learn to find the **boundary between classes**
- Model probability of a label given features
- Focus: classification, regression
- Examples: Linear Regression, Logistic Regression, SVM, ResNet, etc.

Generative

- Learn to model the **data distribution itself**
- Model joint probability of both features and label
- Focus: generating new samples, understanding data structure
- Examples: [we will se on this lecture]

Generative Modeling

Given

$D = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ from $p_{\text{data}}(\mathbf{x})$

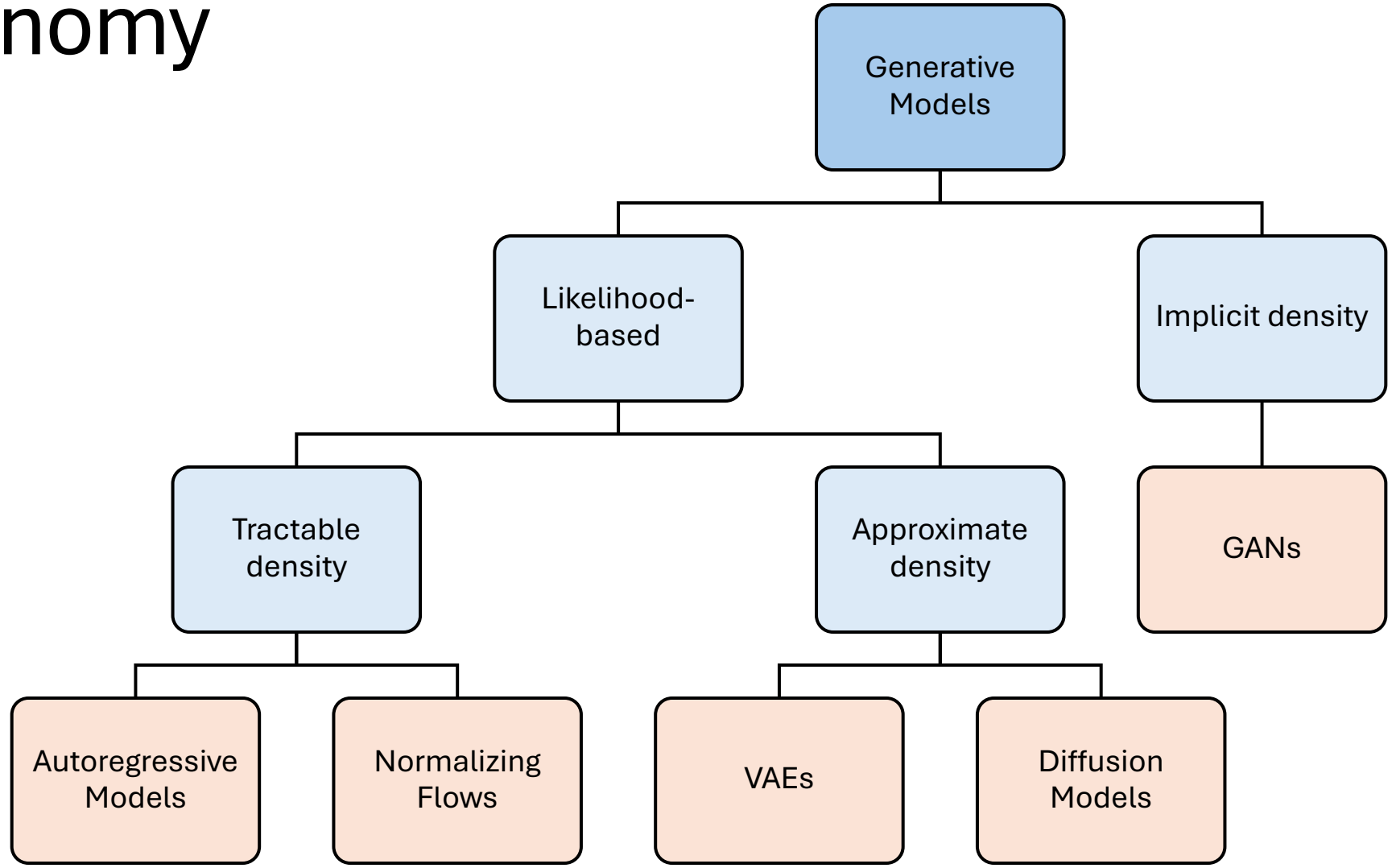
Goals

1. Produce new samples from $p_{\text{data}}(\mathbf{x})$
2. Obtain meaningful data representations
3. Estimate density for data points

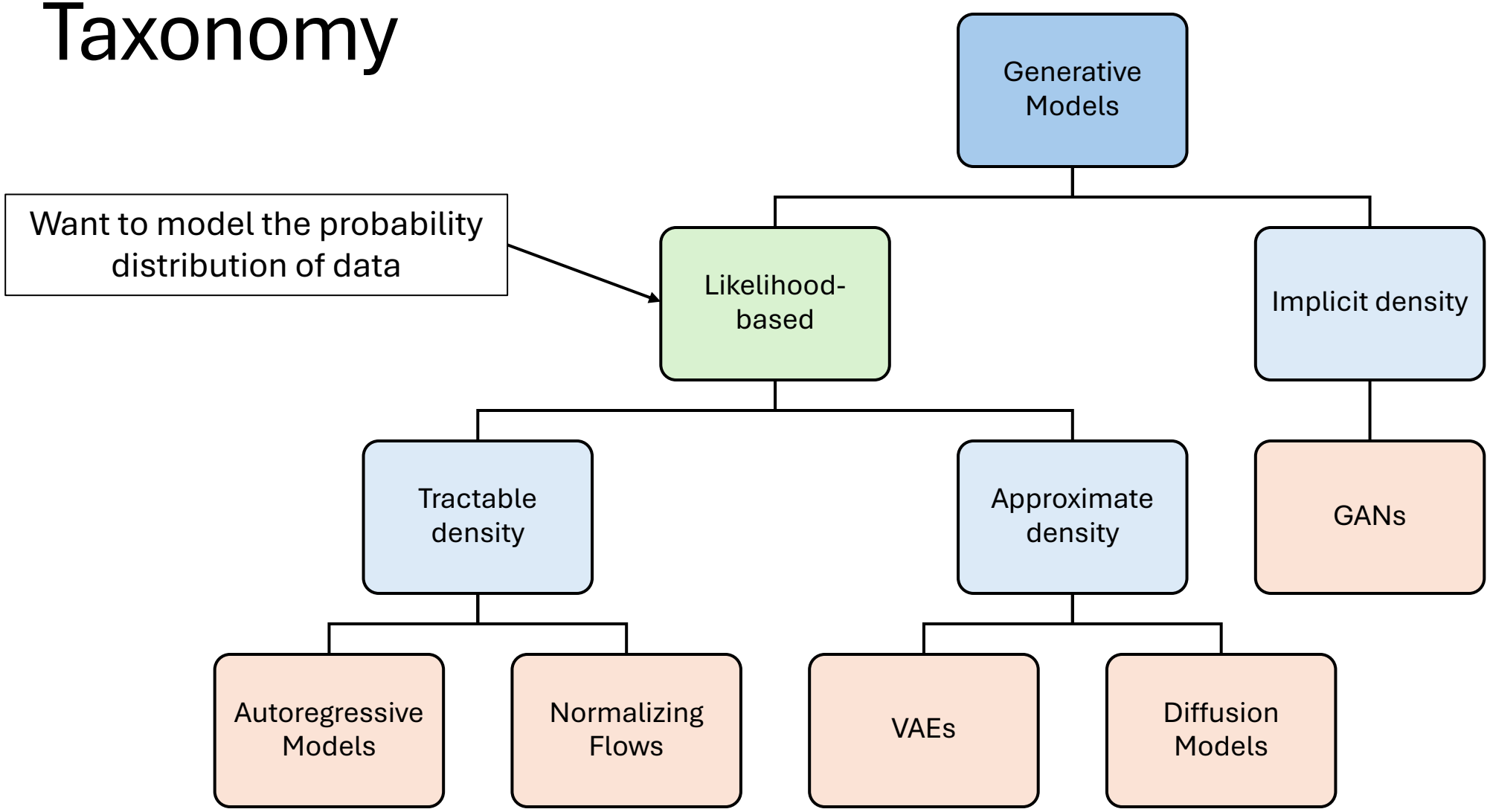
Data Distribution

Taxonomy

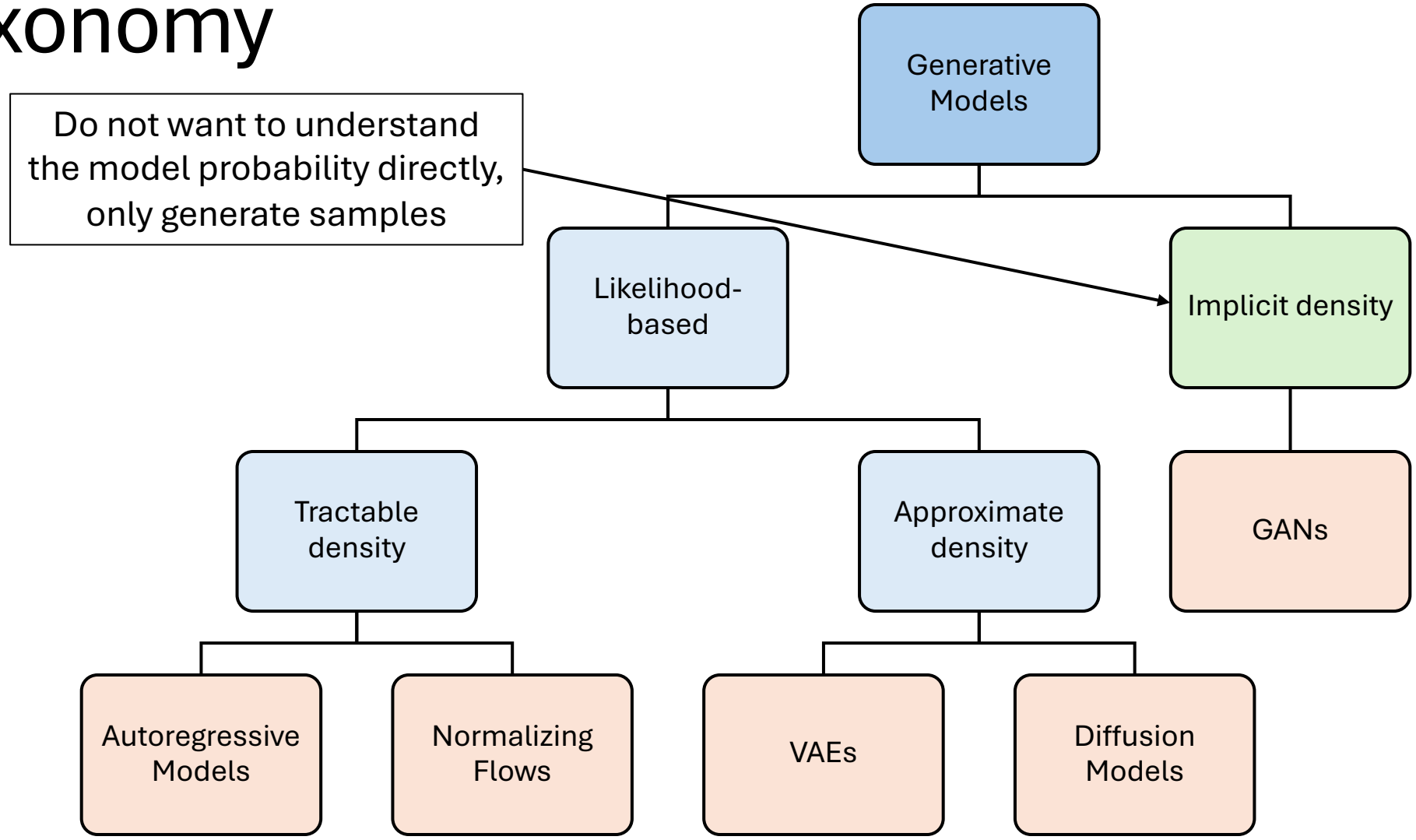
Taxonomy


Generative Models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

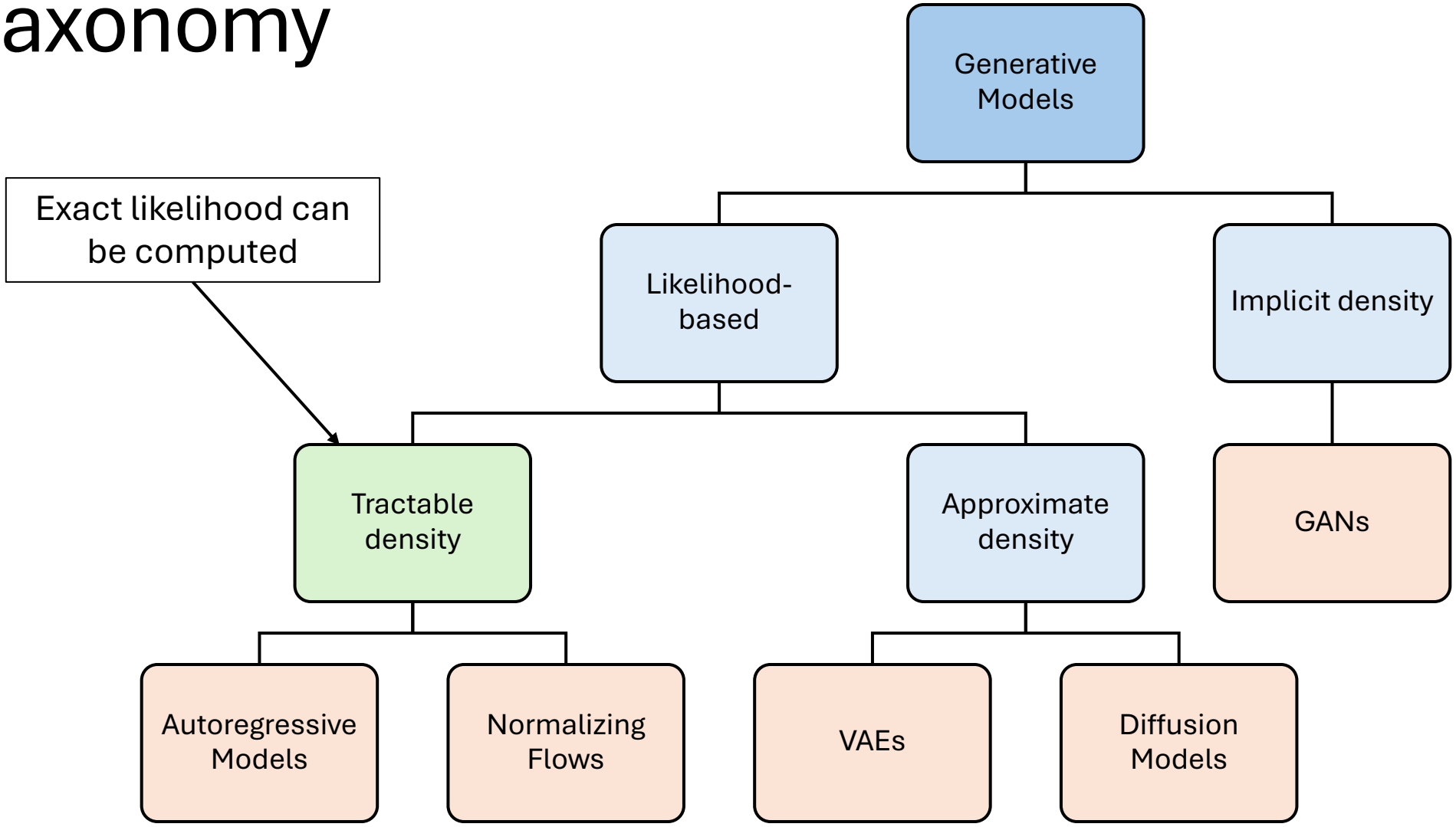
Taxonomy


Generative Models
Want to model the probability distribution of data
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

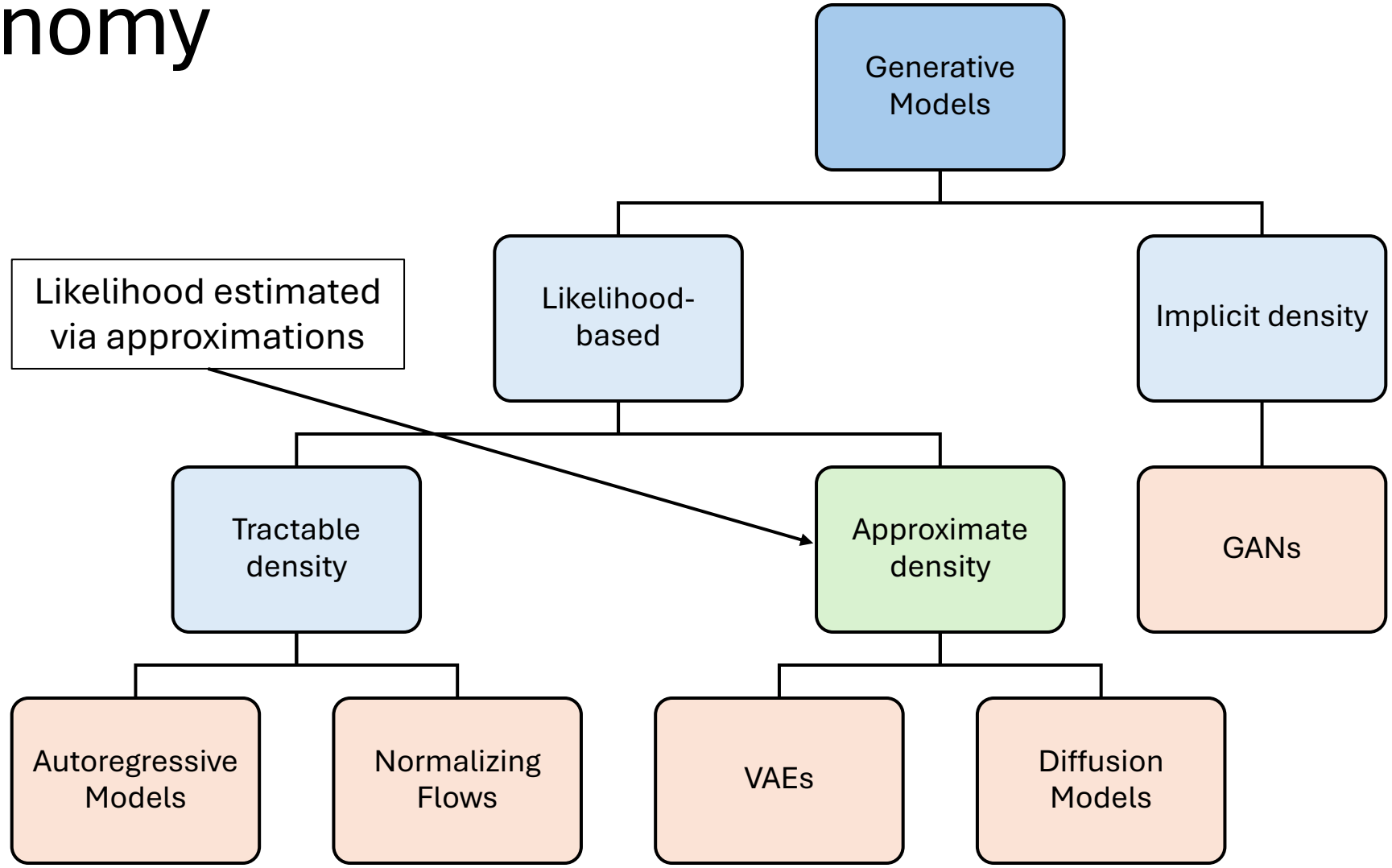
Taxonomy


Do not want to understand the model probability directly, only generate samples
Generative Models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

Taxonomy

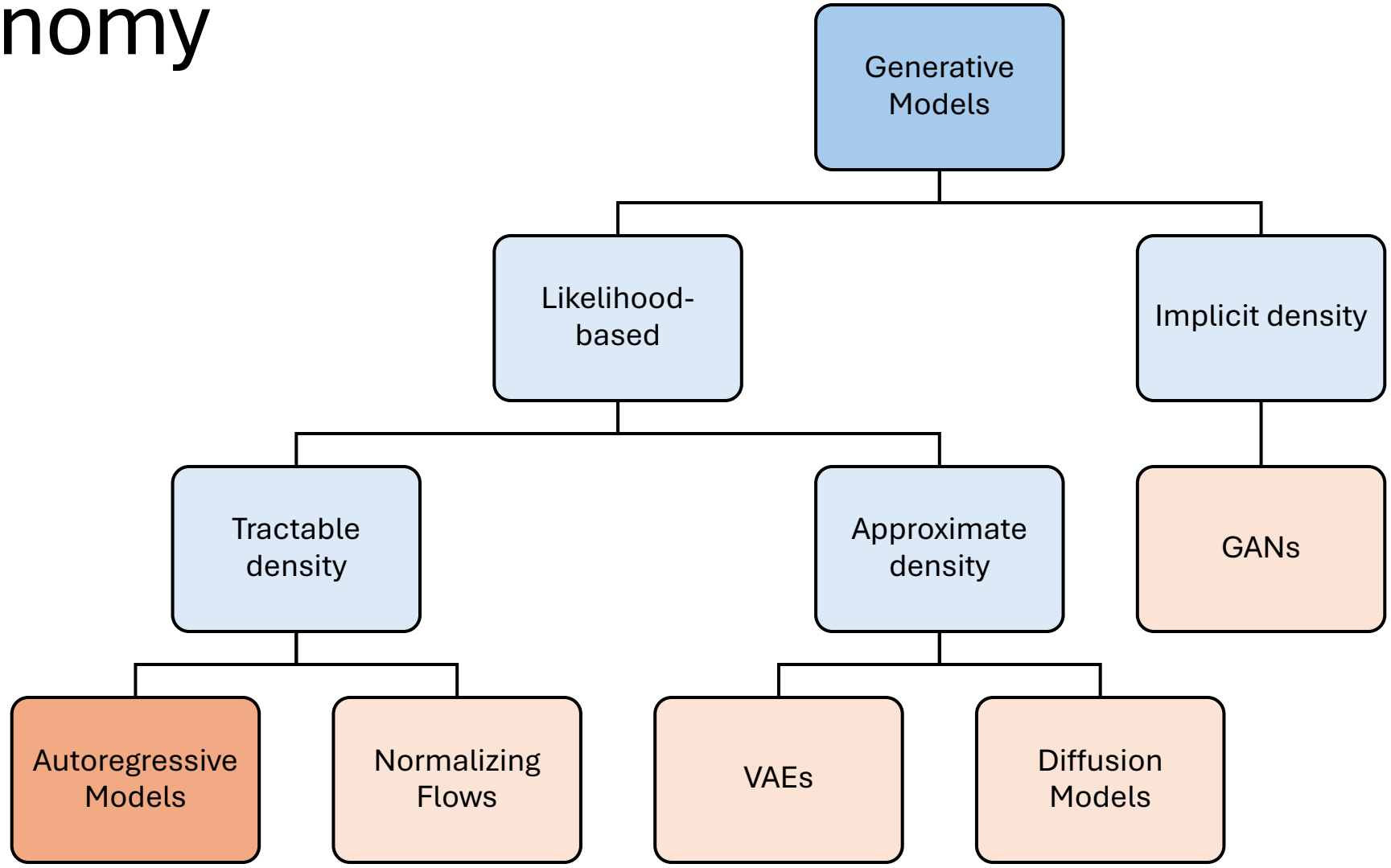

Generative Models
Likelihood-based
Implicit density
Exact likelihood can be computed
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

Taxonomy


Generative Models
Likelihood-based
Implicit density
Likelihood estimated via approximations
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

Autoregressive Models

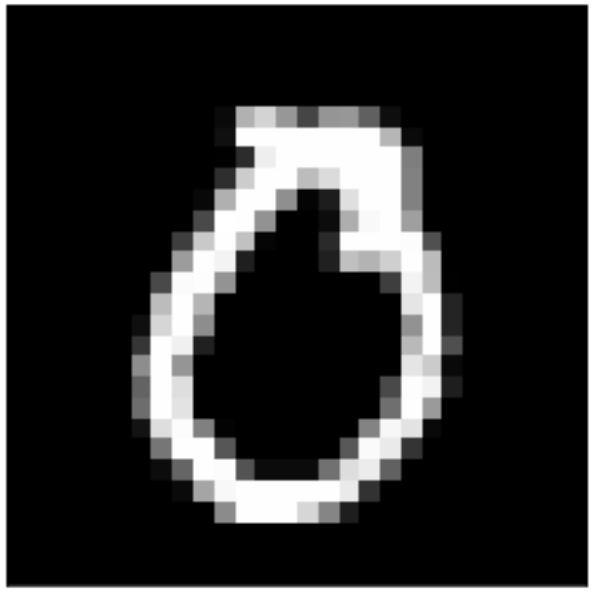
Taxonomy

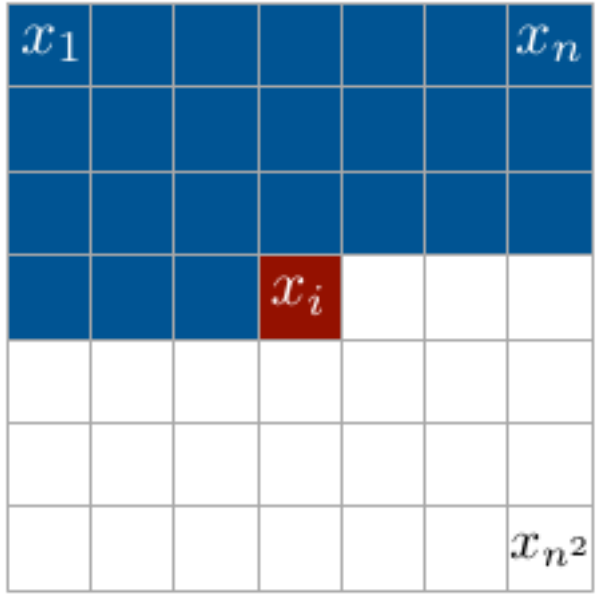

Generative Models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

PixelRNN & PixelCNN

Goal: generate binary images (e.g., MNIST)

Idea: represent joint distribution $p(\mathbf{x})$ via chain rule (cond. prob)





Chain Rule

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

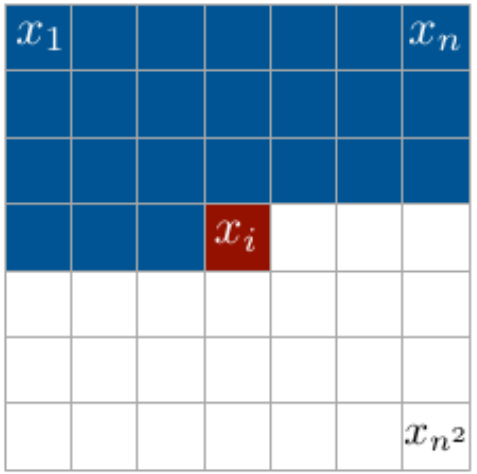
Oord A., Kalchbrenner N., Kavukcuoglu K. Pixel recurrent neural networks, 2016

Parameterization

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i|\mathbf{x}_{<i}) = \prod_{i=1}^{n^2} p(x_i|x_1, \ldots, x_{i-1})$$

Let's parameterize history with some function (neural network $f_{\boldsymbol{\theta}}$).

Thus we move to the conditional distribution $p_{\boldsymbol{\theta}}(x_i|\mathbf{x}_{<i})$.



$$p_{\boldsymbol{\theta}}(x_i|\mathbf{x}_{<i}) = \mathrm{Softmax}(f_{\boldsymbol{\theta}}(\mathbf{x}_{<i}))[x_i]$$

Predict the next pixel based on the previous ones

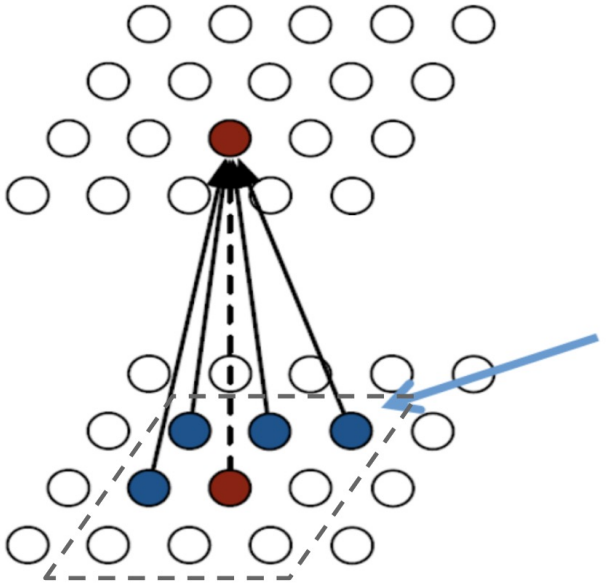
Oord A., Kalchbrenner N., Kavukcuoglu K. Pixel recurrent neural networks, 2016

How to choose the history function?

Problem: long context

Solutions:

- RNN
- LSTM
- Cut context → CNN



PixelCNN

Masked convolution

1	1	1
1	0	0
0	0	0

Oord A., Kalchbrenner N., Kavukcuoglu K. Pixel recurrent neural networks, 2016

How to train autoregressive image models?

MLE: maximize likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ w.r.t. model parameters $\boldsymbol{\theta}$.

We train by minimizing negative log-likelihood:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n^2} \log p_{\boldsymbol{\theta}}(x_i|\mathbf{x}_{<i})$$

In case of binary images from MNIST it transforms into BCE loss:

$$-\log p_{\boldsymbol{\theta}}(x_i|\mathbf{x}_{<i}) = -\Big[x_i \log \sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x}_{<i})\right) + (1-x_i)\log\left(1-\sigma\left(f_{\boldsymbol{\theta}}(\mathbf{x}_{<i})\right)\right)\Big]$$

Pros and Cons

Advantages

- Exact likelihood training
- Good sample diversity

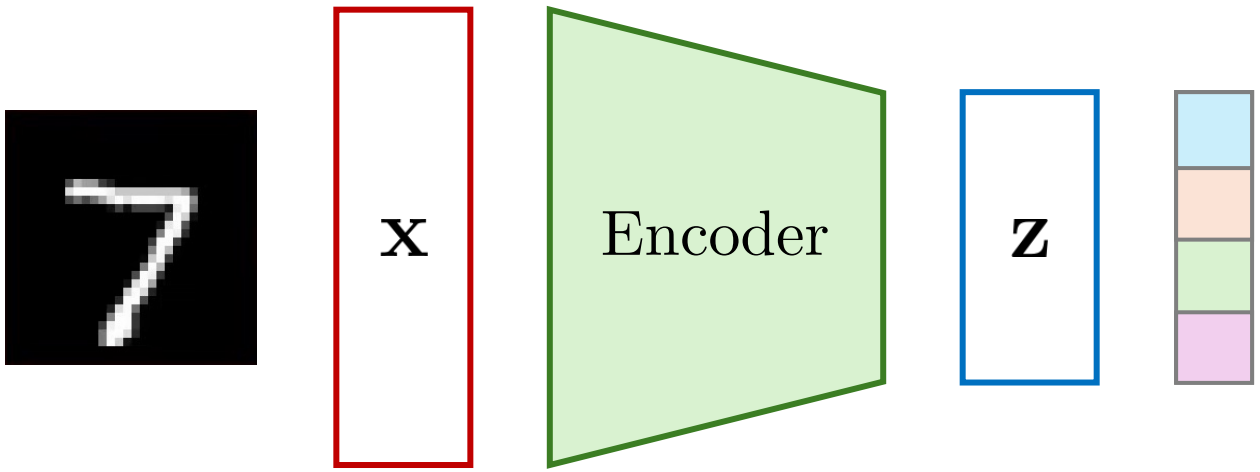
Disadvantages

- Sequential generation – pixels generated one-by-one
- Limited global coherence – struggles to capture long-range spatial dependencies
- Scalability – computationally expensive for high-resolution data

Autoencoders

Autoencoders: background

Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data.

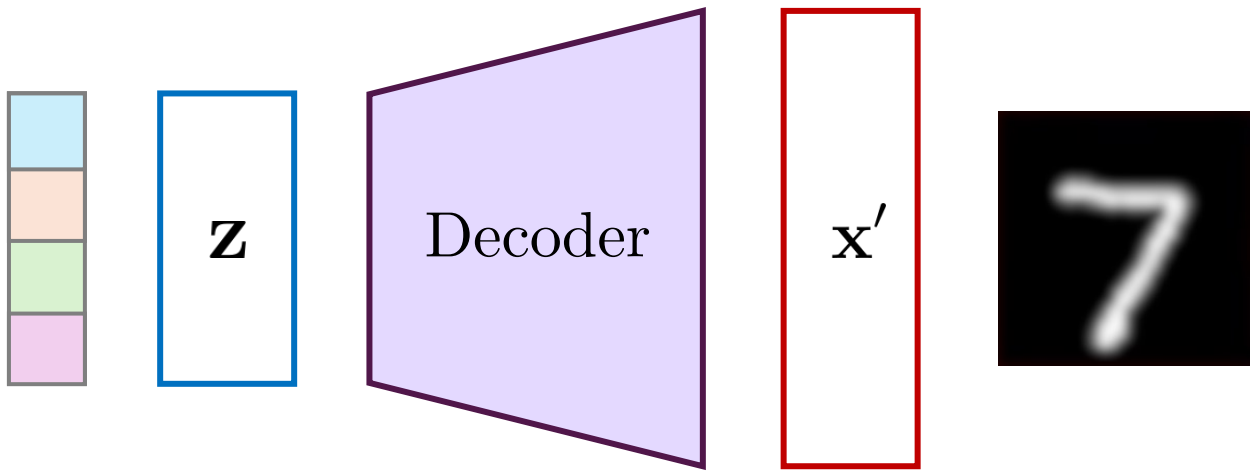


Encoder learns mapping from the data, $\boldsymbol{x}$, to a **low-dimensional** latent space, $\boldsymbol{z}$.

Analogues:
PCA, Matrix Factorization

How can we learn this latent space?

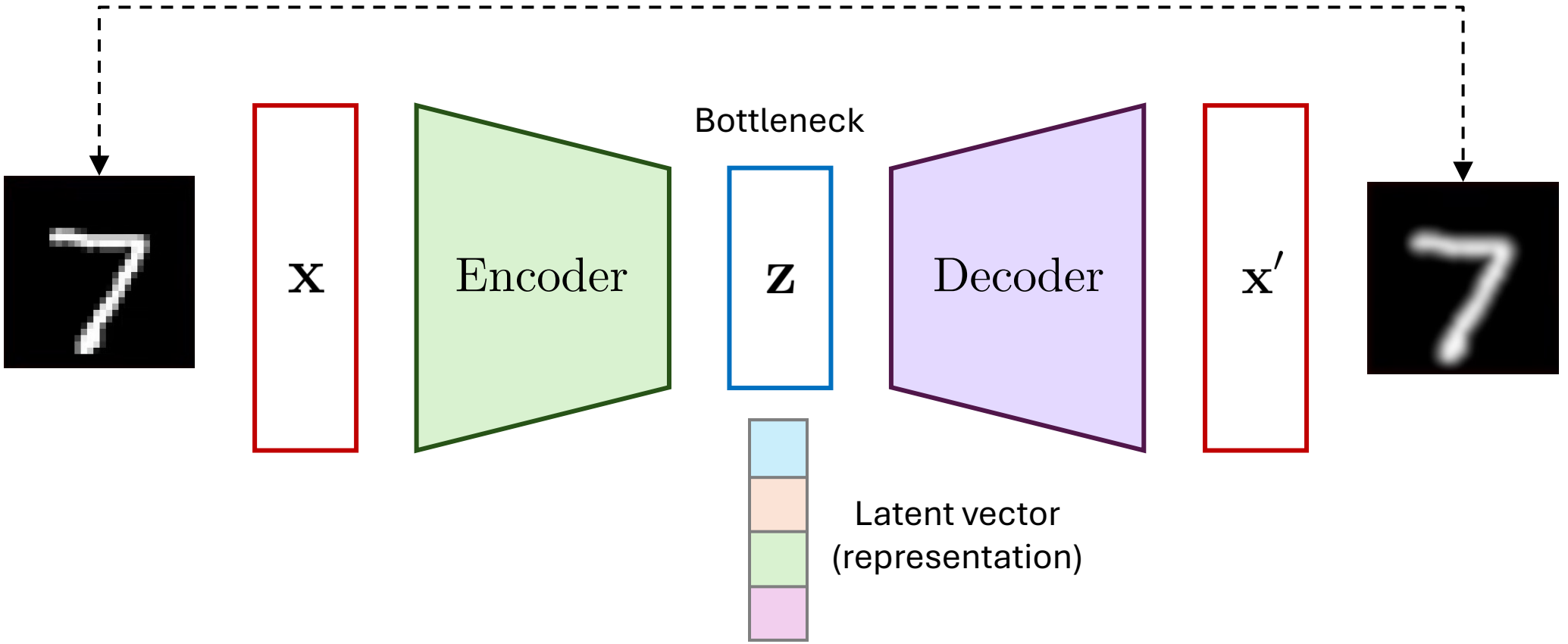
Train another model to use these features to **reconstruct the input**.



Decoder learns the mapping back from latent space, $\mathbf{z}$, to a reconstructed observation, $\mathbf{x}$'.

Reconstruction loss

Basic MSE loss function: $\mathcal{L} = \|\mathbf{x} - \mathbf{x}'\|_2^2$



Representation Learning

Bottleneck hidden layer forces network to learn a compressed latent representation.

Reconstruction loss $\mathcal{L} = ||\mathbf{x} - \mathbf{x}'||_2^2$ forces the latent representation $\mathbf{z}$ to capture (or encode) as much "information" about the data $\mathbf{x}$ as possible.

Autoencoding = **Auto**matically **encoding** data

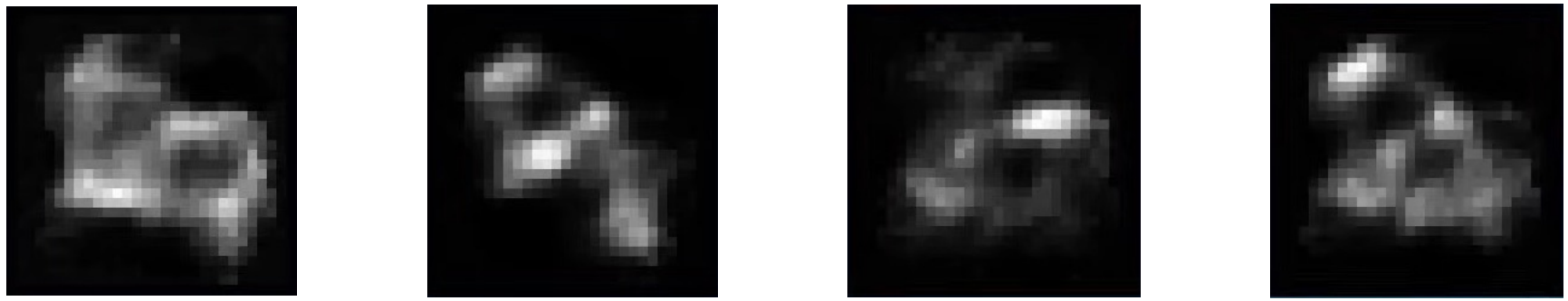
"Auto" = **self**-encoding

Why not simple Autoencoders?

Goal: generate new data using Autoencoder's latent space

Idea: randomly sample points from the latent space and feed them trough the decoder

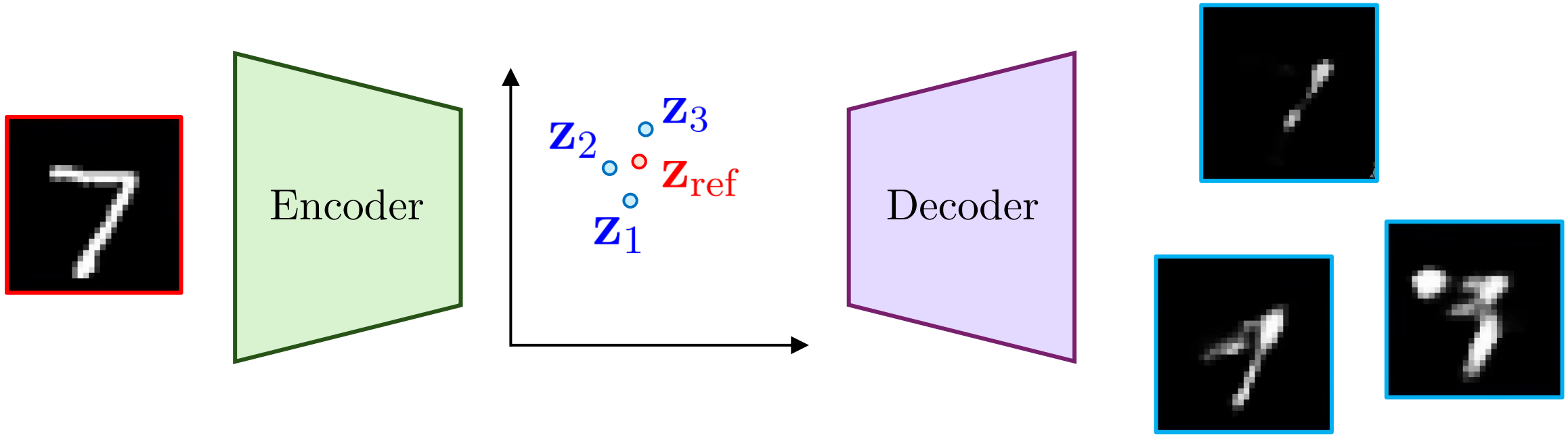
Problem: in most cases, this approach will not product useful results, mainly because the **latent space is disorganized** and irregular – so large areas of it will not producte meaningful data



Sampling latent vectors near a reference

Goal: obtain images that resemble the reference one

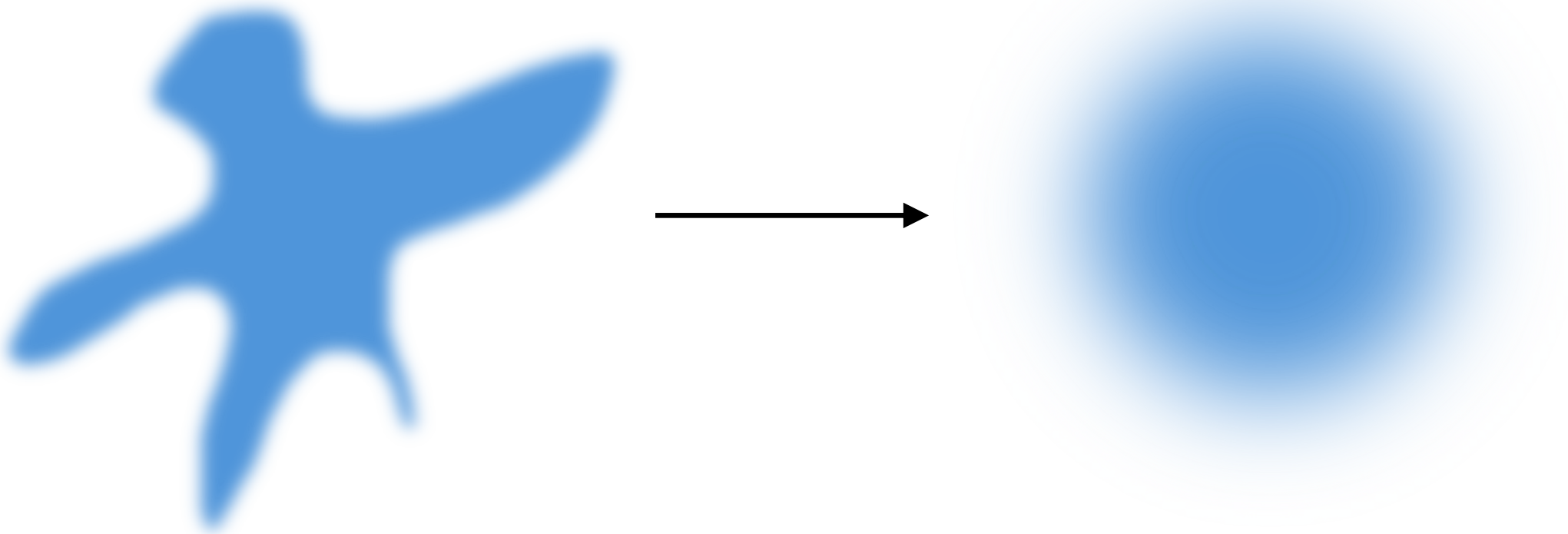
Problem: the latent space is so poorly structured that even nearby points often don't correspond to meaningful variations



What is the solution?

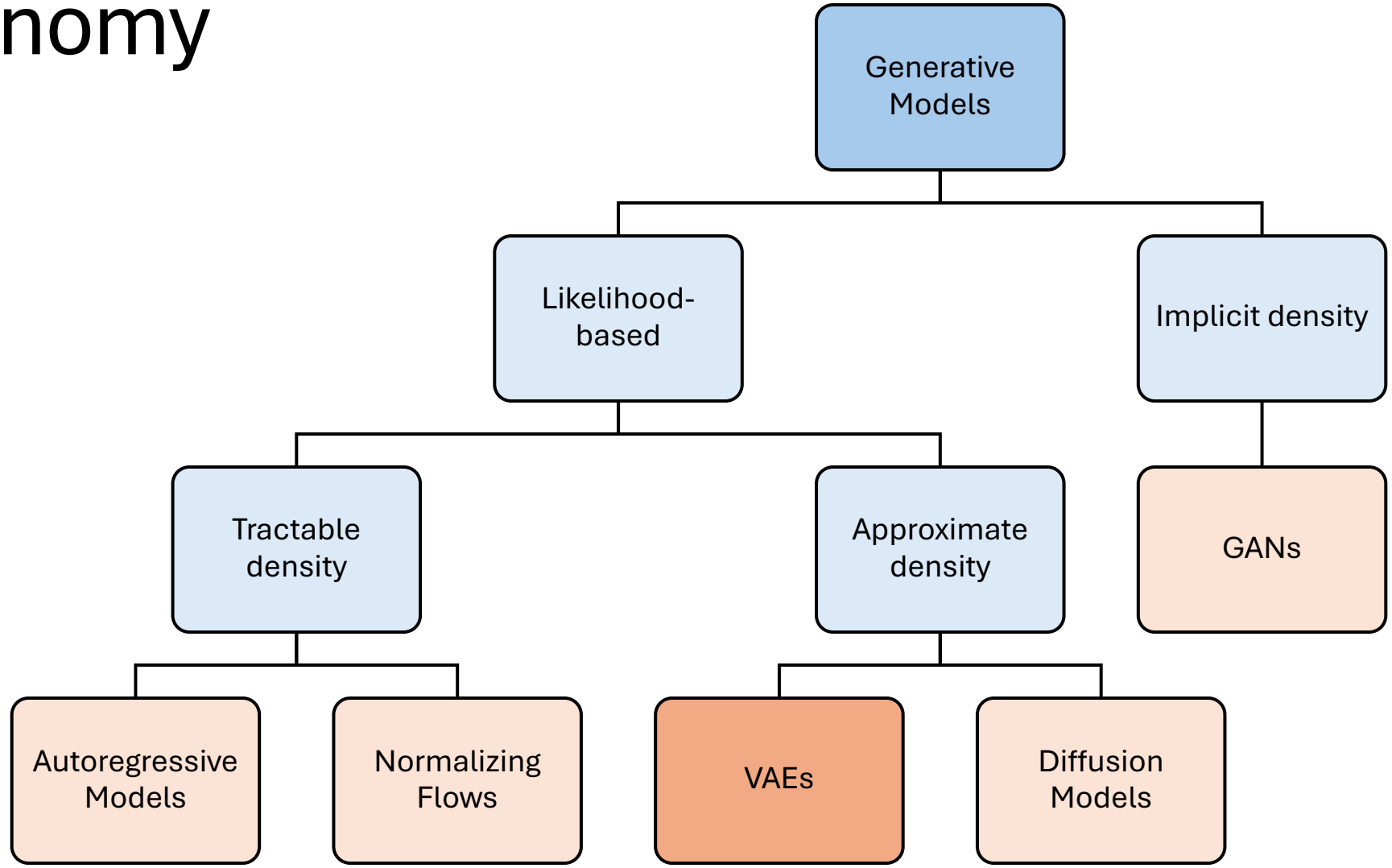
Organize the latent space!

But how…?



VAEs

Taxonomy


Generative Models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

Data and Latent Distributions

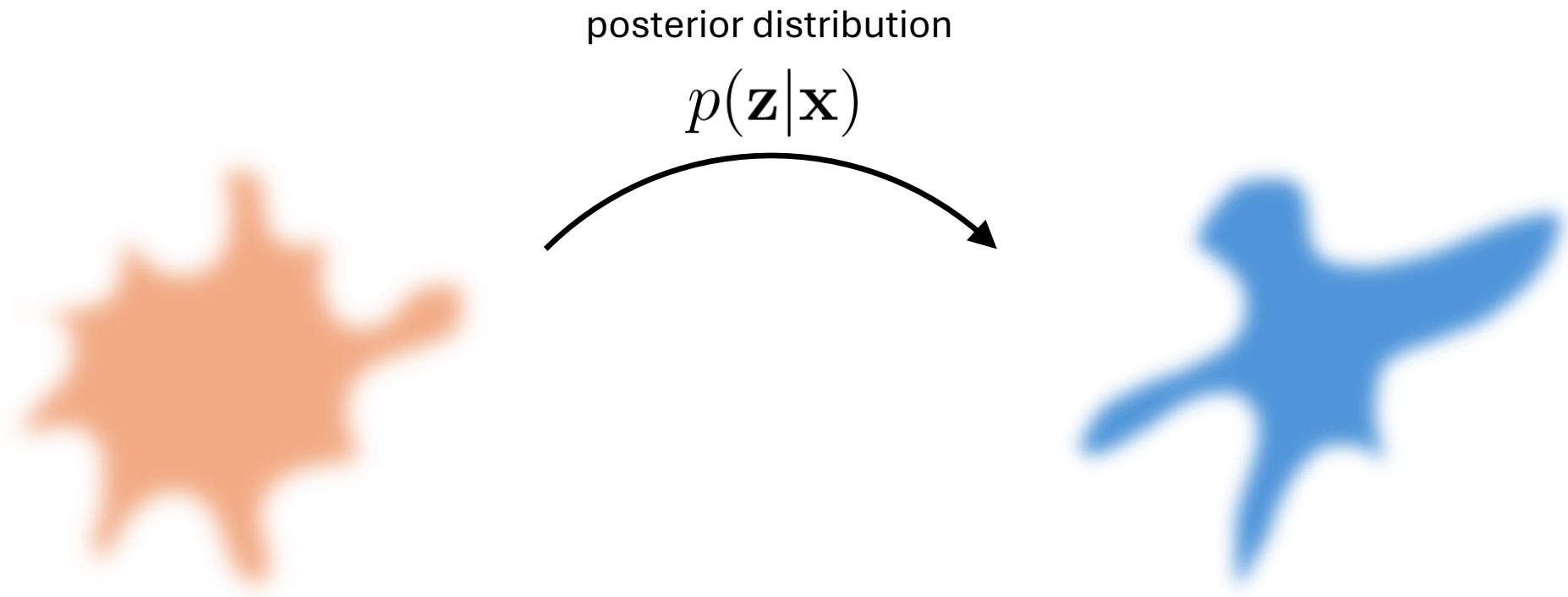
Data Distribution

$p(\mathbf{x})$

Latent Distribution

$p(\mathbf{z})$

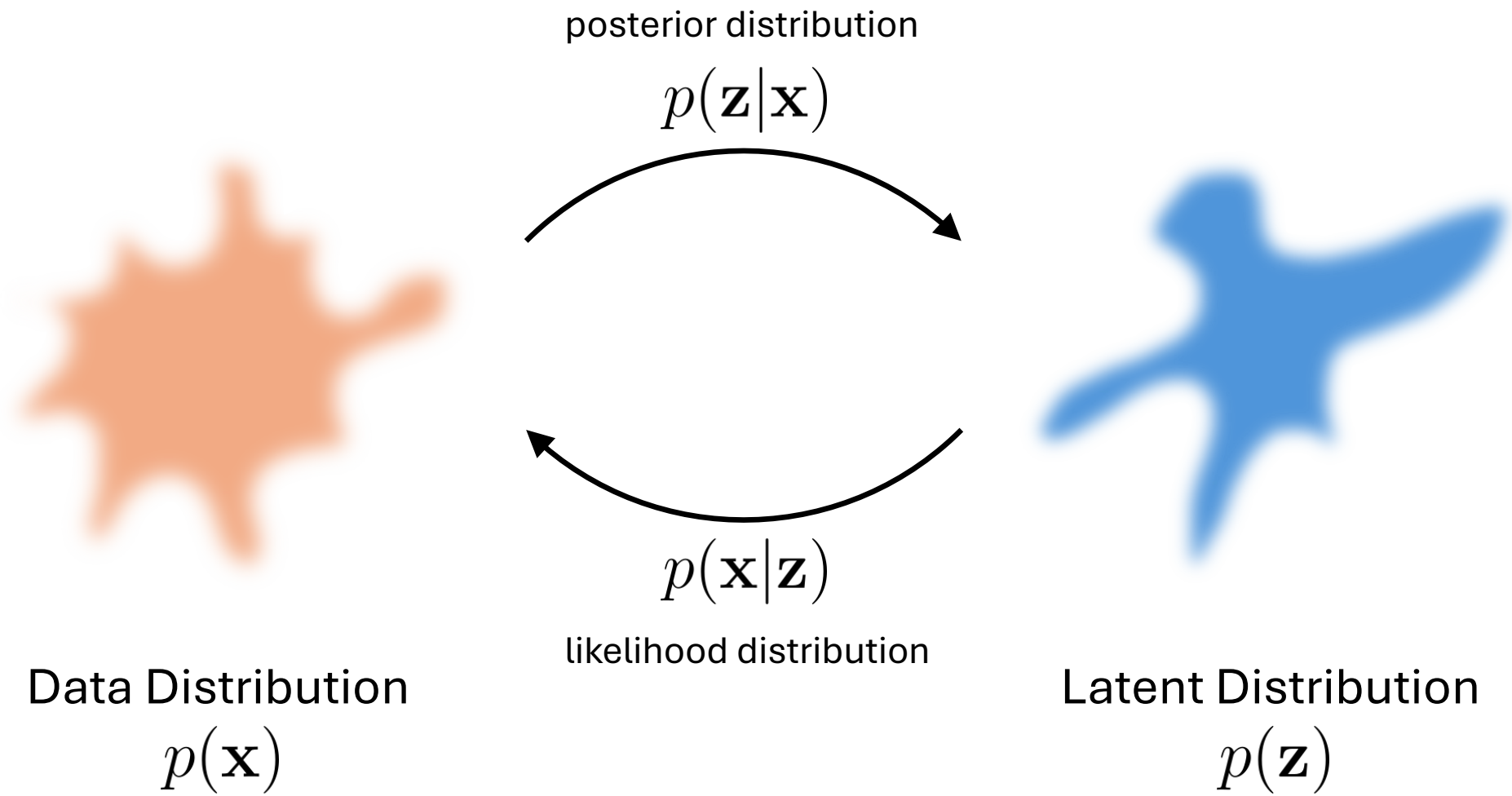
Data and Latent Distributions



Data Distribution
$p(\mathbf{x})$

Latent Distribution
$p(\mathbf{z})$

Data and Latent Distributions



Data and Latent Distributions

Issue: we don't know the latent distribution, therefore we cannot derive a mapping

posterior distribution

$$p(\mathbf{z}|\mathbf{x})$$

$$p(\mathbf{x}|\mathbf{z})$$

likelihood distribution

Data Distribution

$$p(\mathbf{x})$$

Latent Distribution

$$p(\mathbf{z})$$

Data and Latent Distributions

Idea: choose a simple latent distribution

posterior distribution

$$p(\mathbf{z}|\mathbf{x})$$

$$p(\mathbf{x}|\mathbf{z})$$

likelihood distribution

becomes tractable

Data Distribution

$$p(\mathbf{x})$$

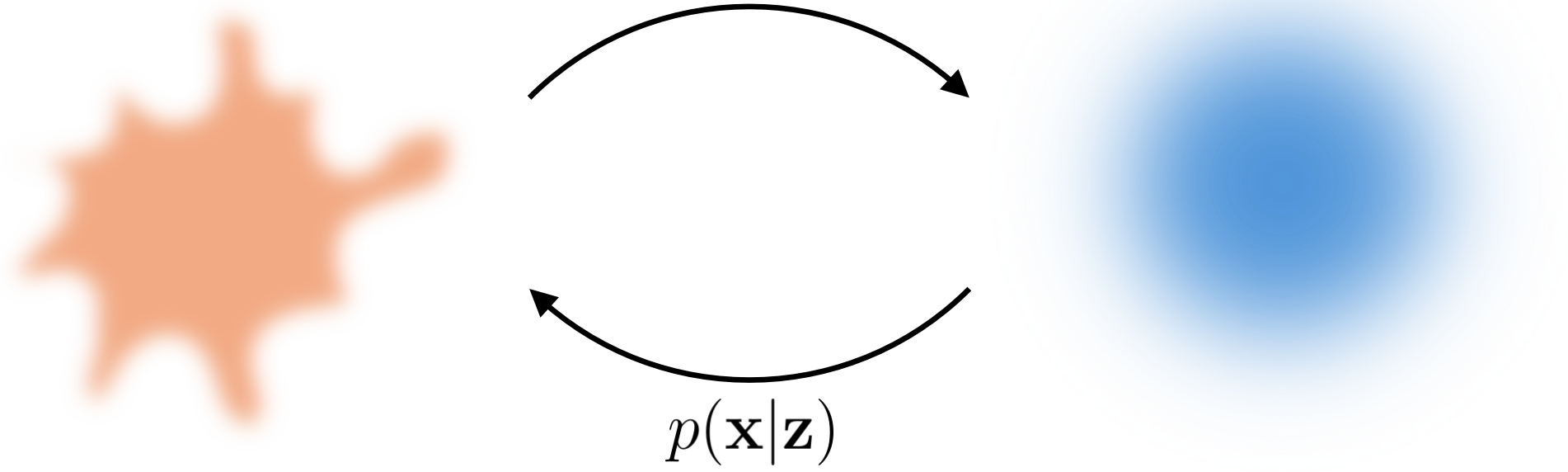
Latent Distribution

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Variational Bayes

approximate posterior distribution with parameterized variational distribution

$$q_\phi(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z}|\mathbf{x})$$



$$p(\mathbf{x}|\mathbf{z})$$

likelihood distribution

↓

becomes tractable

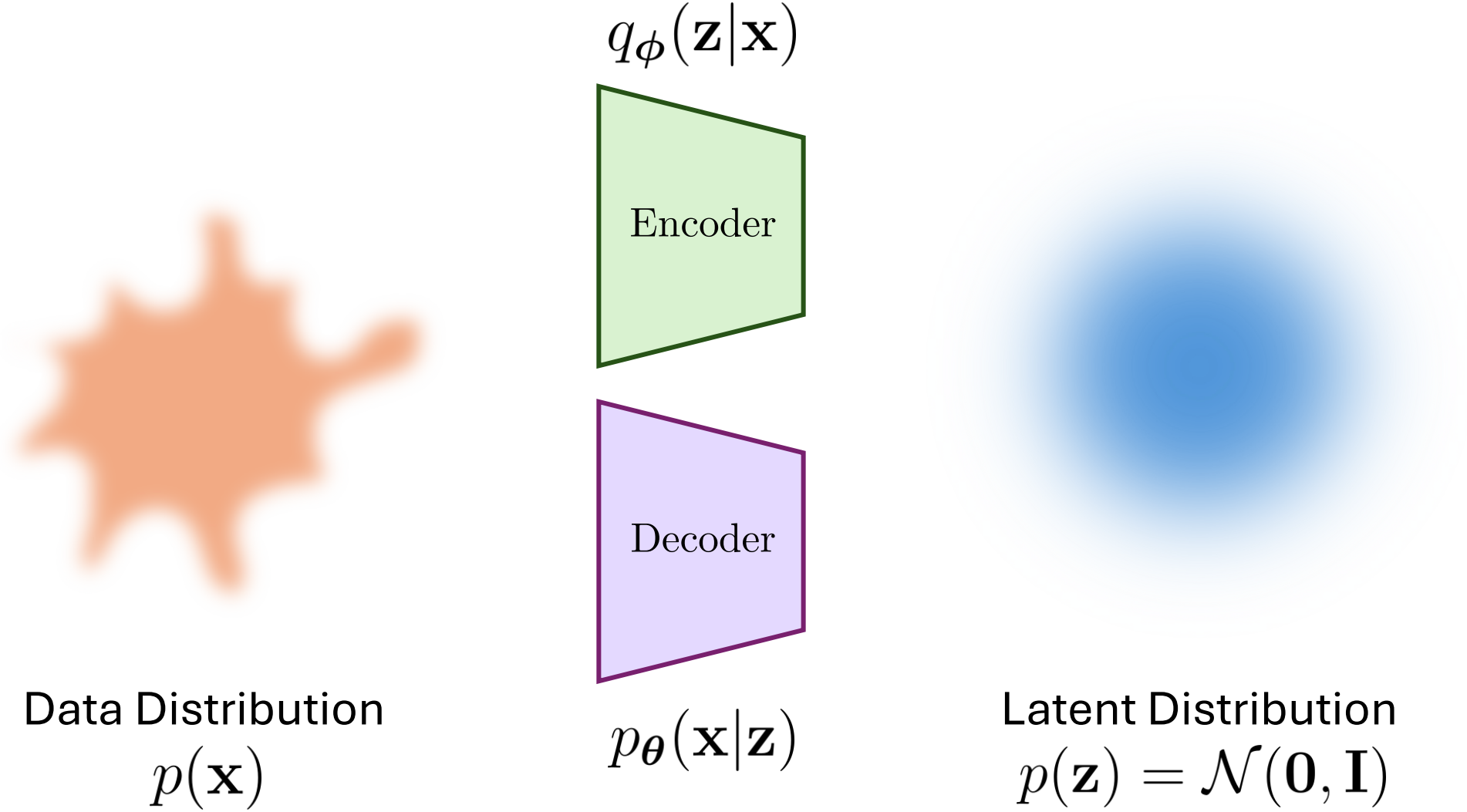
Data Distribution

$$p(\mathbf{x})$$

Latent Distribution

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Autoencoder Variational Bayes



How to train VAE?

Formally, we want to maximize the likelihood of data: $p(\mathbf{x})$

Let's do it in a parameterized distributions family: $p_{\boldsymbol{\theta}}(\mathbf{x})$

How to train VAE?

Formally, we want to maximize the likelihood of data: $p(\mathbf{x})$

Let's do it in a parameterized distributions family: $p_{\boldsymbol{\theta}}(\mathbf{x})$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$ (1) marginalize

$$= \log \int \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$ (2) multiply and divide

$$= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right]$$ (3) expectation

$$\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] = \mathcal{L}_{\boldsymbol{\theta}, \boldsymbol{\phi}}(\mathbf{x})$$

(4) Jensen's inequality

$$\mathbb{E}\left[f(x)\right] \geq f\left(\mathbb{E}\left[x\right]\right)$$

ELBO optimization

We obtained lower bound for the data likelihood:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \qquad \text{(1) definition}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \qquad \text{(2) Bayes' rule}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \log \frac{p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \qquad \text{(3) obvious}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z})\right) \qquad \text{(4) KL}$$

VAE loss function

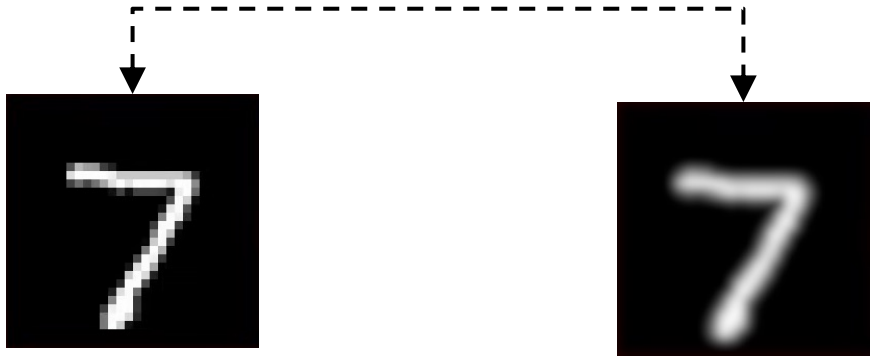
$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)$$

VAE loss function

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})}_{\text{Reconstruction Loss}} - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right)$$

We can model data likelihood with a Bernolli distribution, then Reconstruction Loss becomes a simple MSE loss:

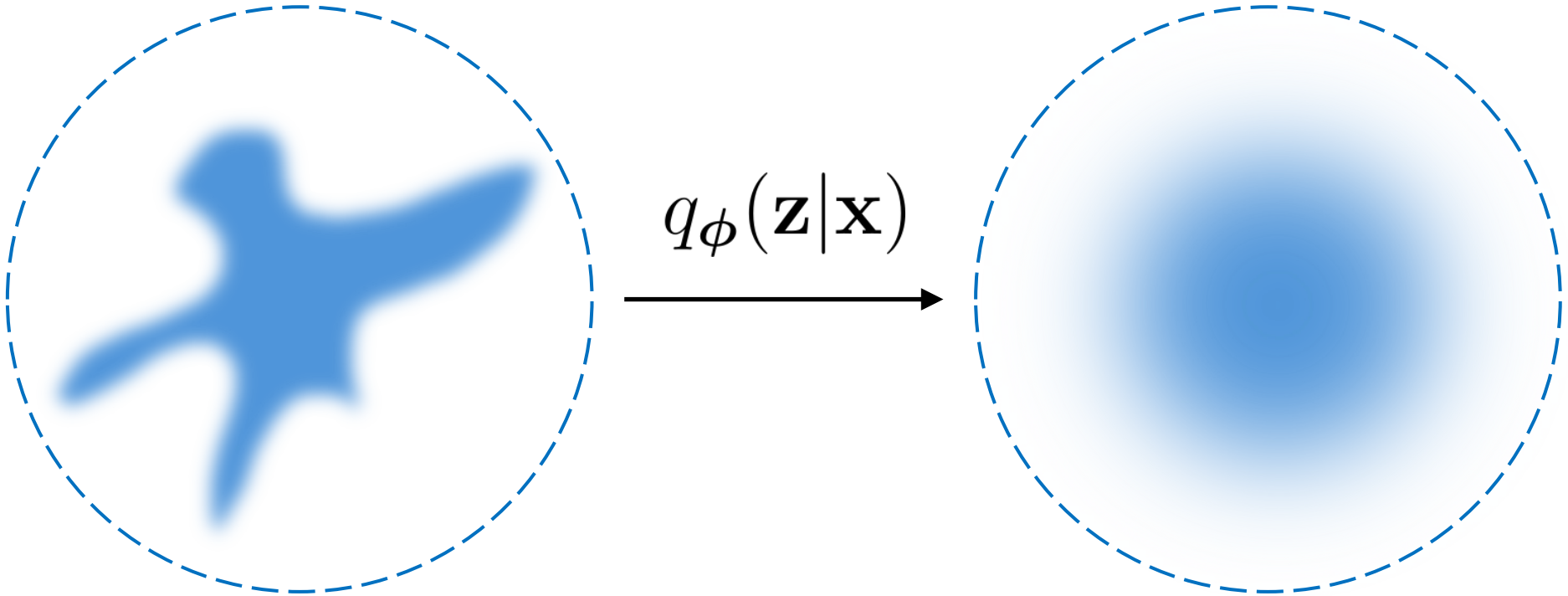
$$\mathcal{L} = \|\mathbf{x} - \mathbf{x}'\|_2^2$$



VAE loss function

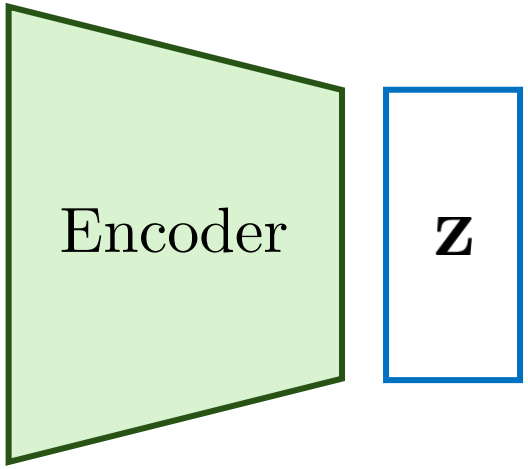
$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) || p(\mathbf{z})\right)$$

Regularization Loss

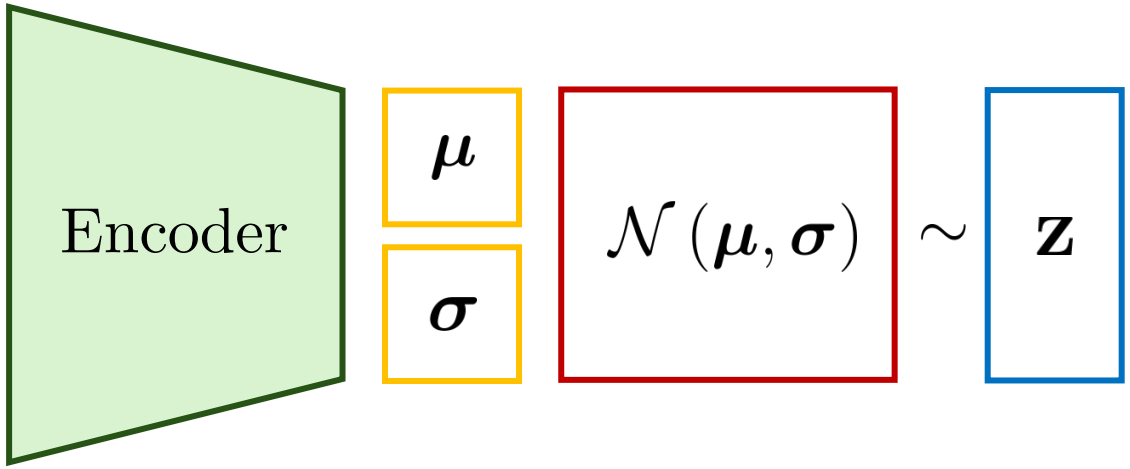


Autoencoder vs VAE

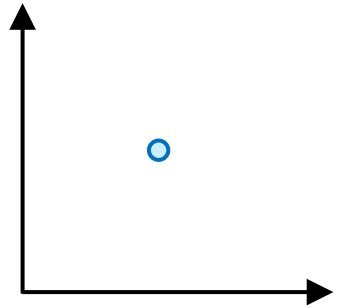
AE

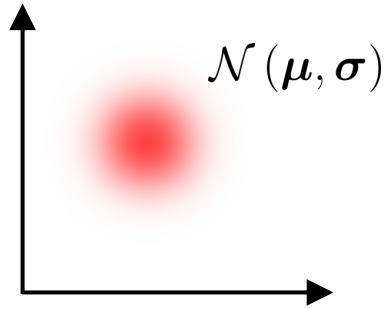


VAE



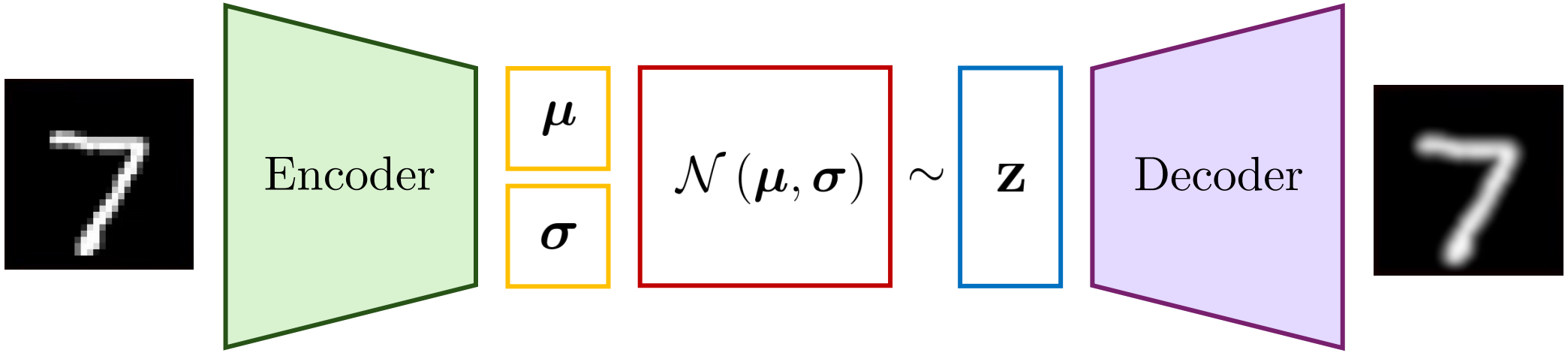
Latent Space





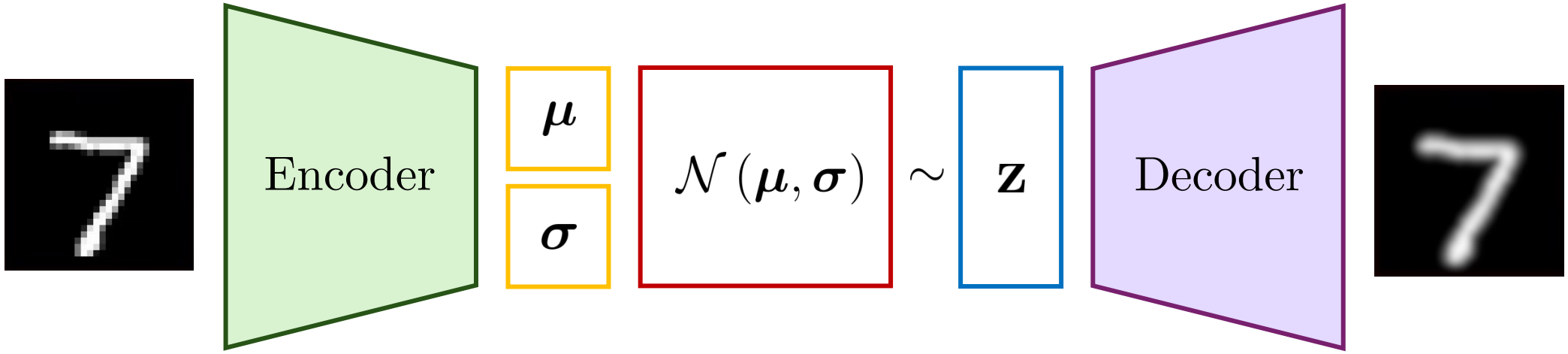
How do we do in practice?

1. Put the image into Encoder
2. Encoder outputs mean and standard deviation
   + Regularuzation Loss
3. Sample latent variable from the normal distribution
4. Put the latent vector into Decoder
5. Decoder outputs parameters of data distribution
   + Reconstruction Loss



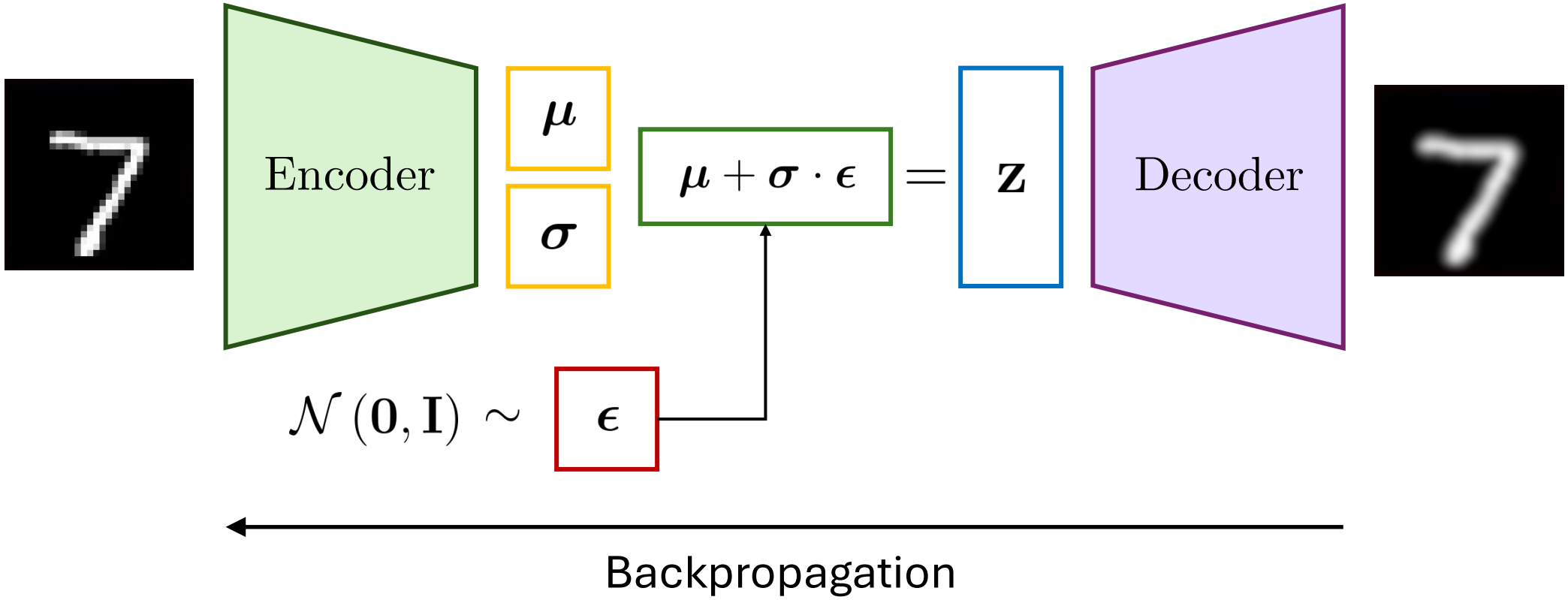
Reparameterization Trick

How to backpropagate through the sampling process?



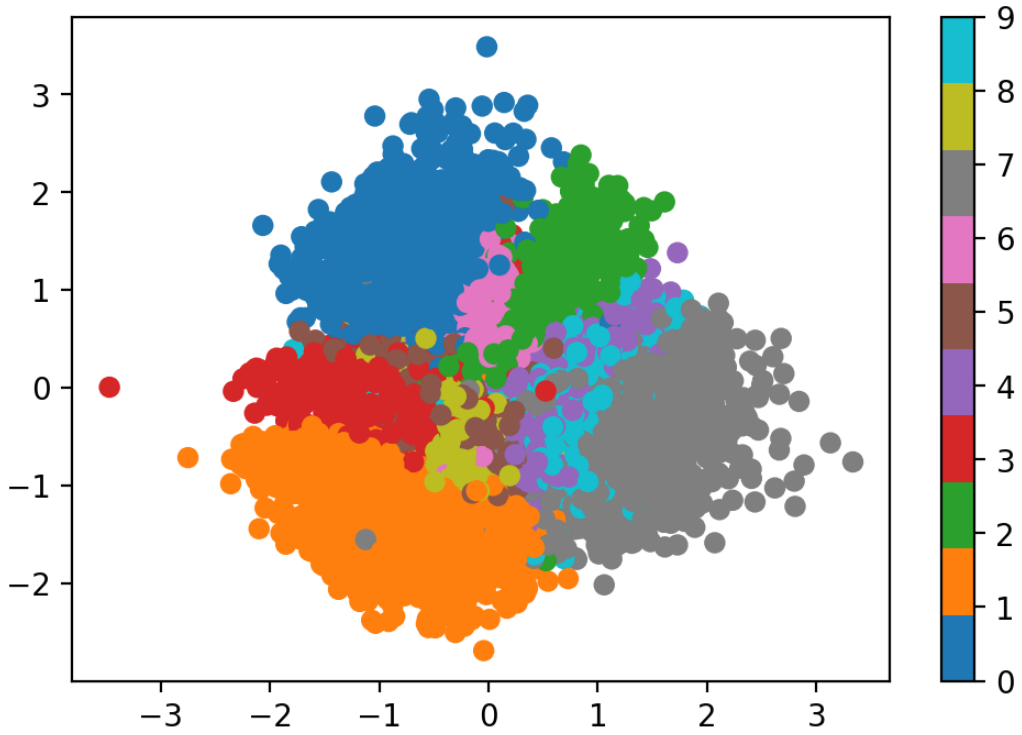
Reparameterization Trick

Use property of normal distribution

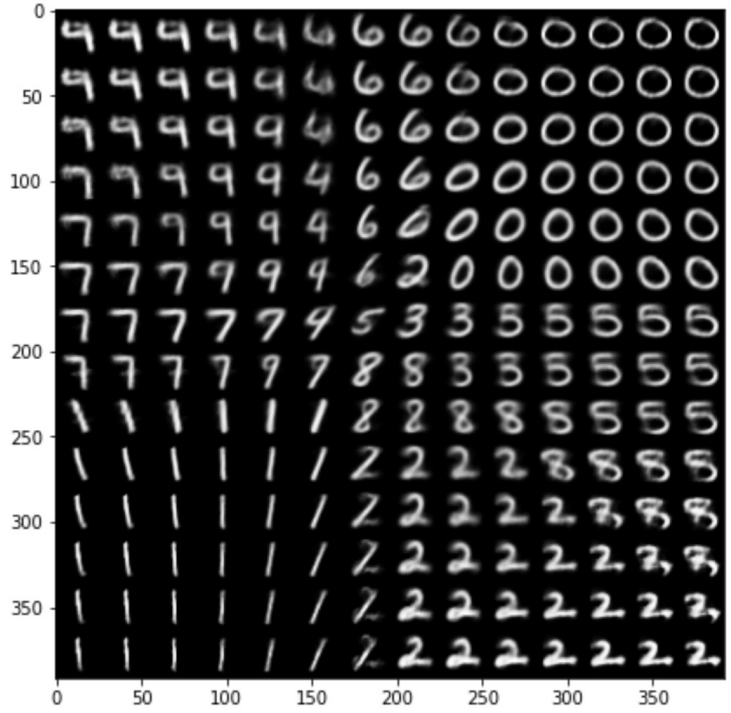


Properties of the VAE's Latent Space

Organization



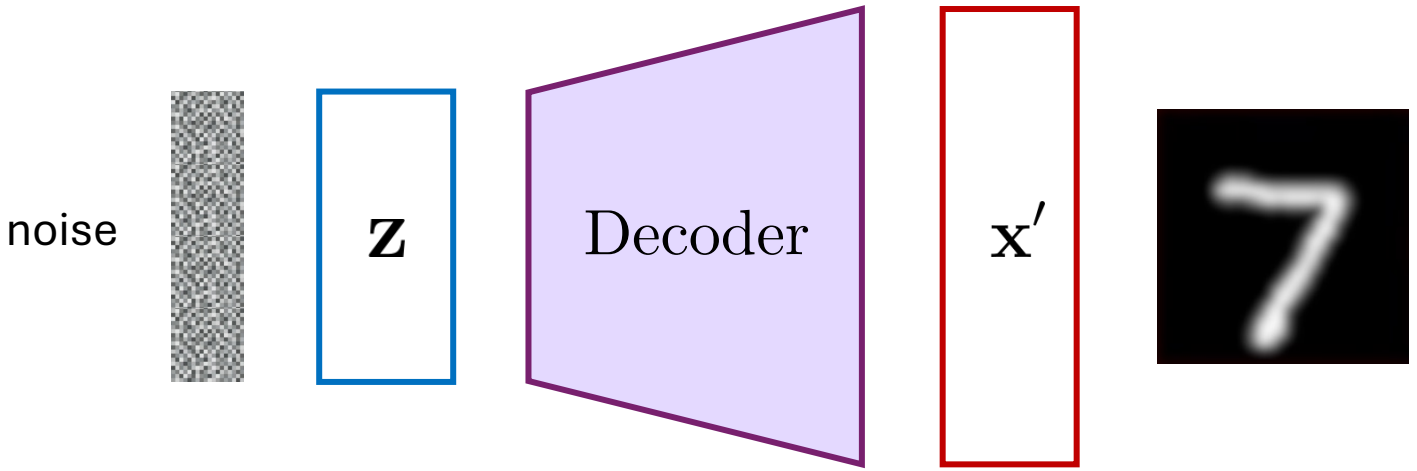
Interpolation



https://avandekleut.github.io/vae/

VAE as a generative model

1. Sample noise latent $\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
2. Put $\mathbf{z}$ into Decoder and get $\mathbf{x}$'



Pros and Cons

Advantages

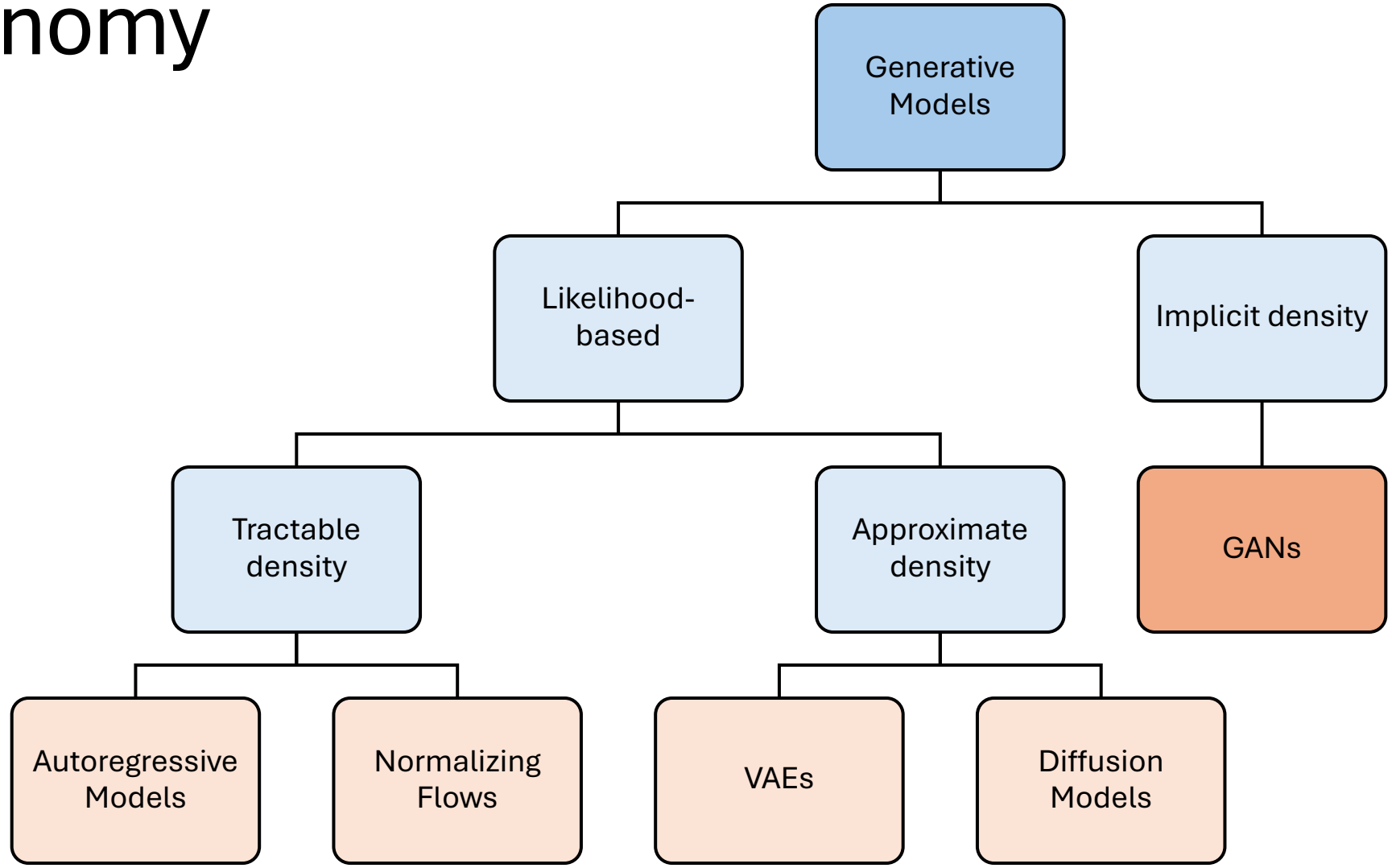
- Probabiltic, calibrated likelihoods (ELBO)
- Fast sampling (single pass), stable training
- Useful latent space for editing/interpolation

Disadvantages

- Blur/averaging from pixel likelihoods, perceptual gap
- Posterior collapse with strong decoders (small or zero KL)
- Sensitive hyperparameters tuning

GANs

Taxonomy

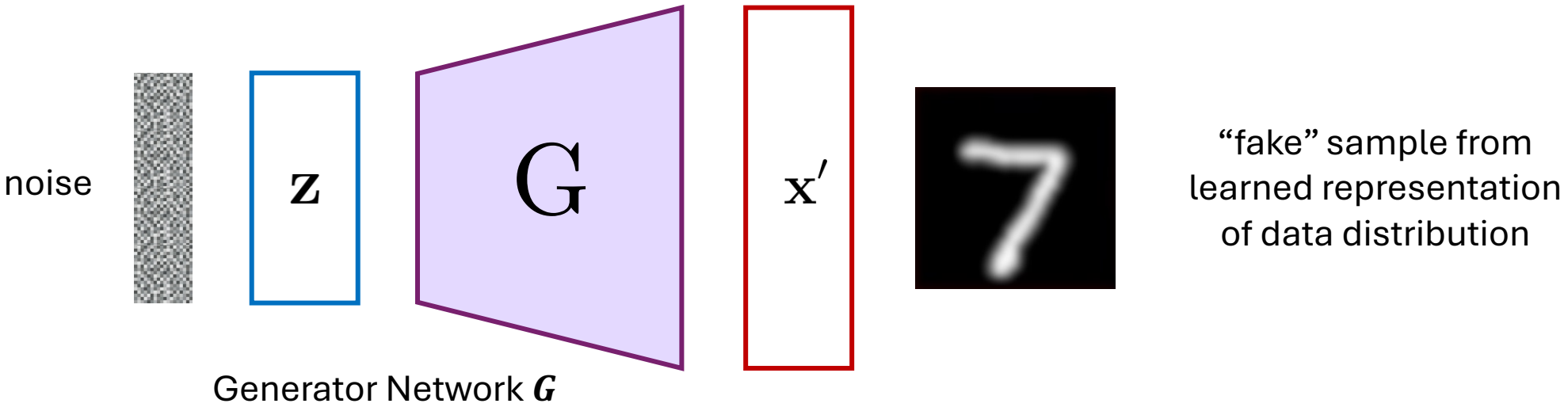

Generative Models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive Models
Normalizing Flows
VAEs
Diffusion Models

What if we just want to sample?

Idea: don't explicitly model density, and instead just sample to generate new instances.

Problem: want to sample from complex distribution – can't do this directly!

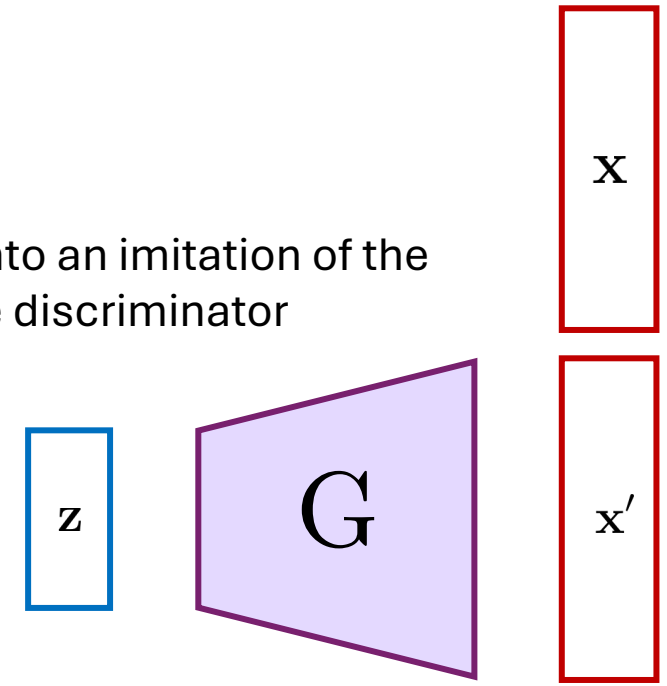
Solution: sample from something simple (e.g., noise), learn a transformation to the data distribution.



Generative Adversarial Networks (GANs)

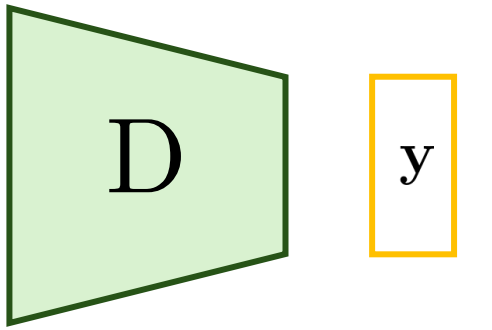
Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

The **generator** turns noise into an imitation of the data to try to trick the discriminator



Generator Network G

The **discriminator** tries to identify real data from fakes created by the generator

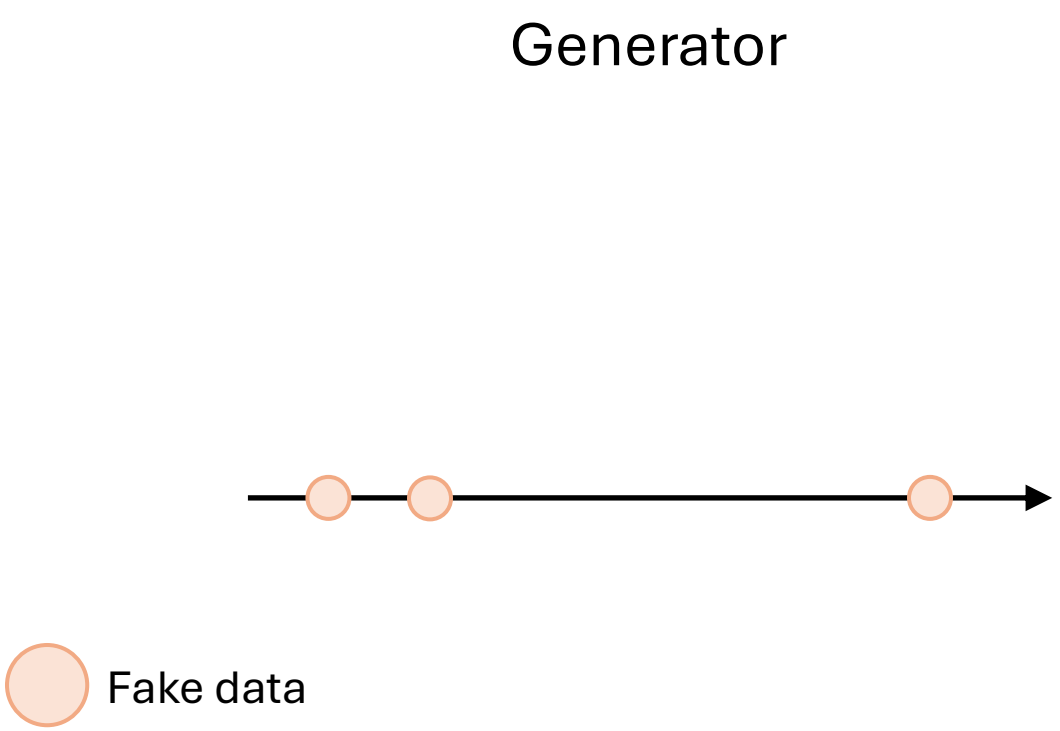


True / False

Discriminator Network D

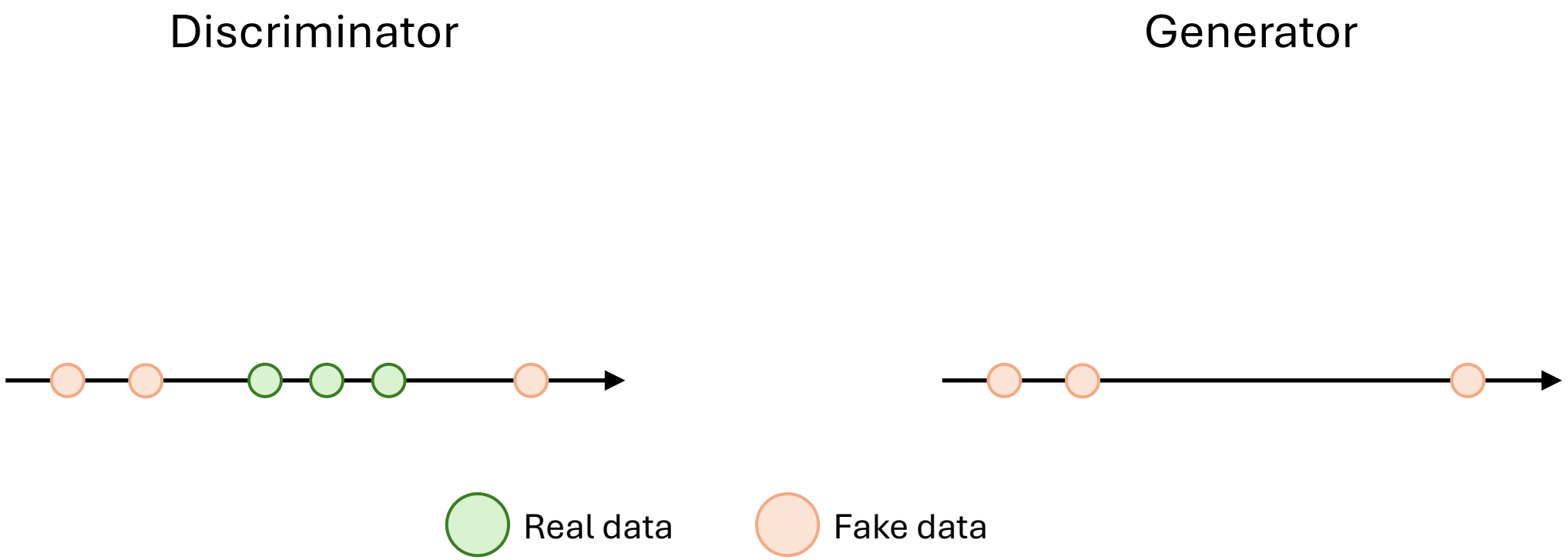
Intuition behind GANs

Generator starts from noise to try to create an imitation of the data.



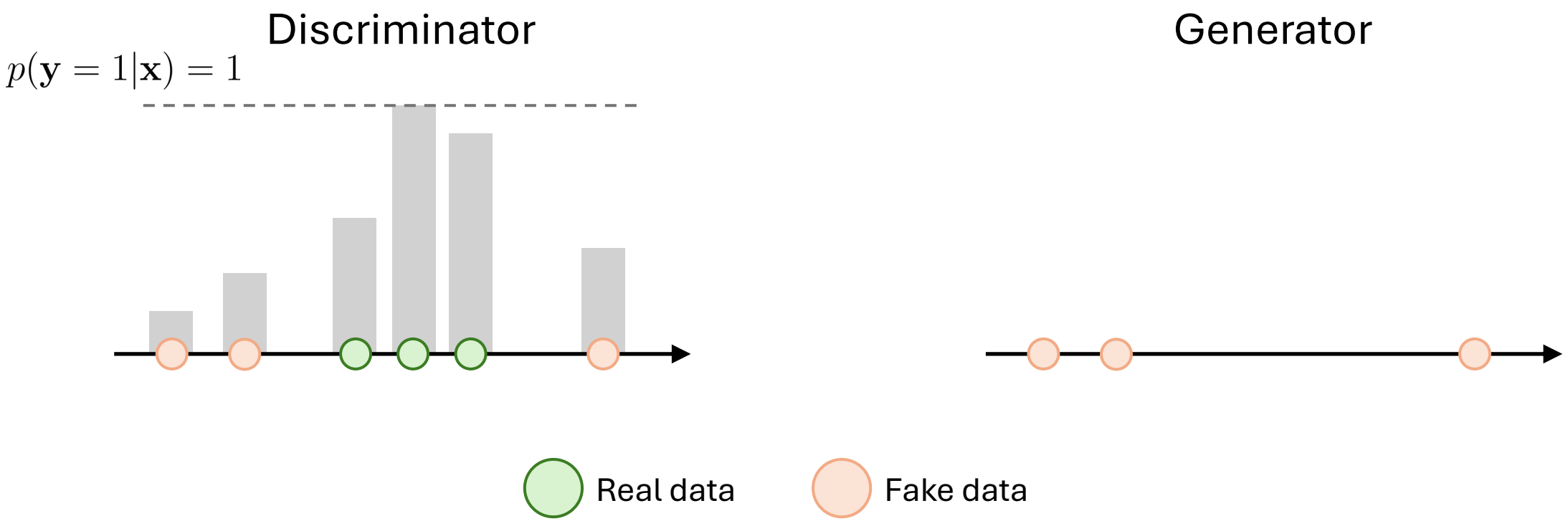
Intuition behind GANs

Discriminator looks at both real data and fake data created by the generator.



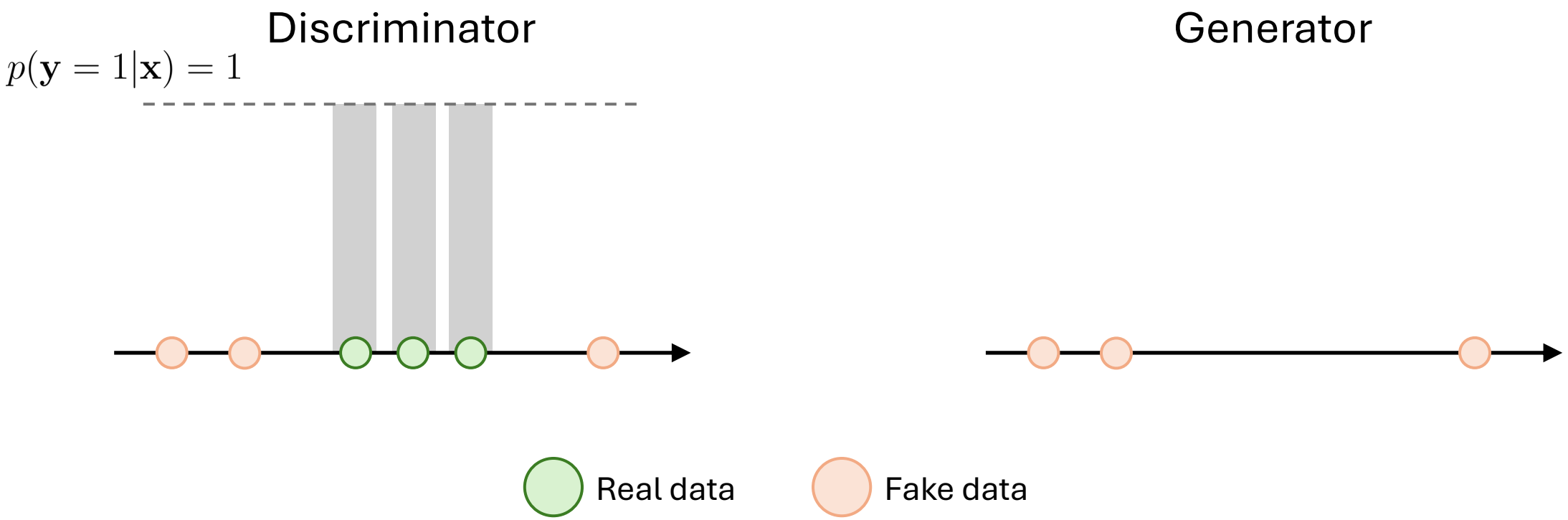
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



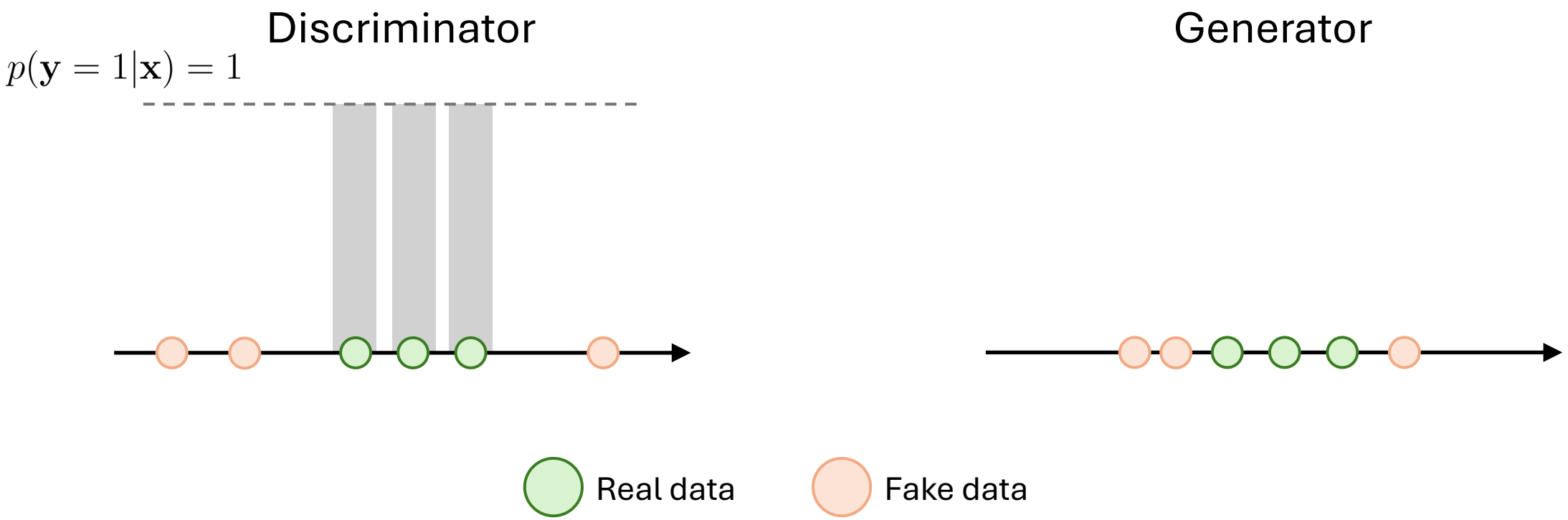
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



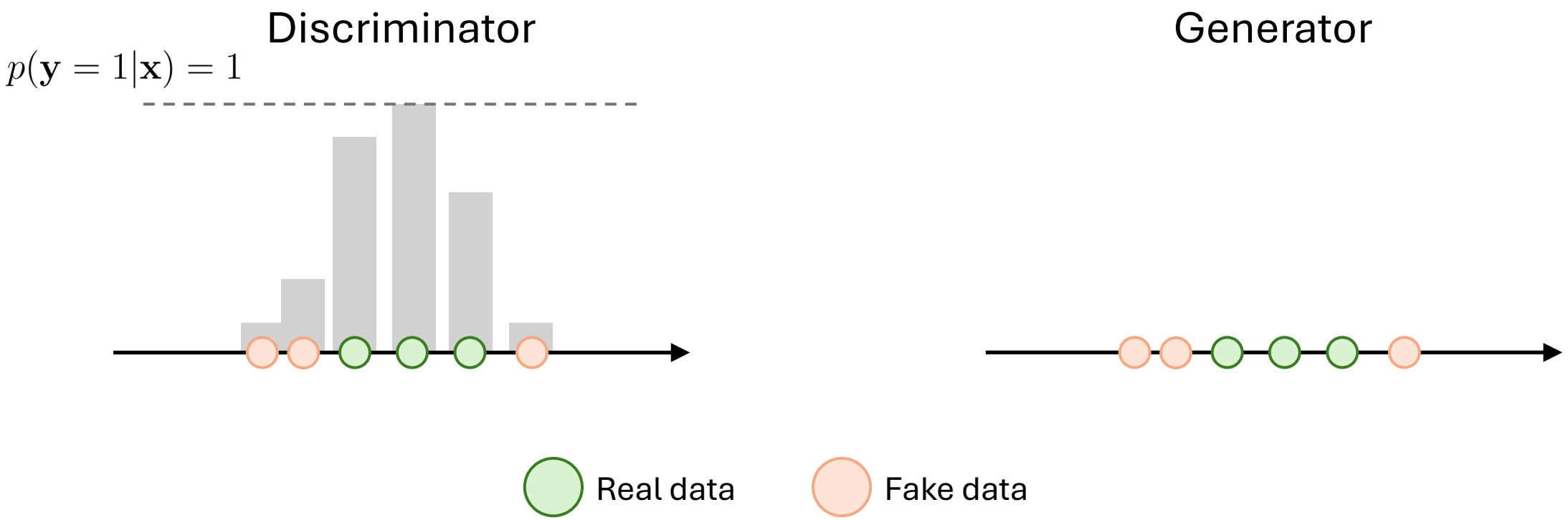
Intuition behind GANs

Generator tries to improve its imitation of data.



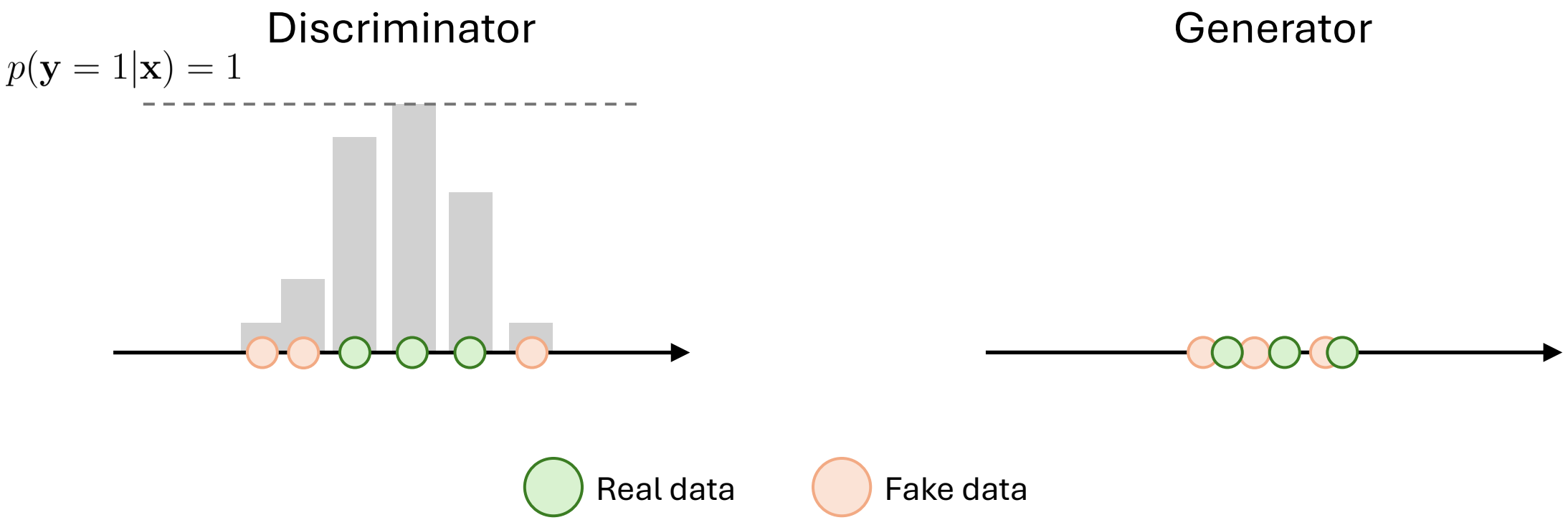
Intuition behind GANs

Discriminator tries to predict what's real and what's fake.



Intuition behind GANs

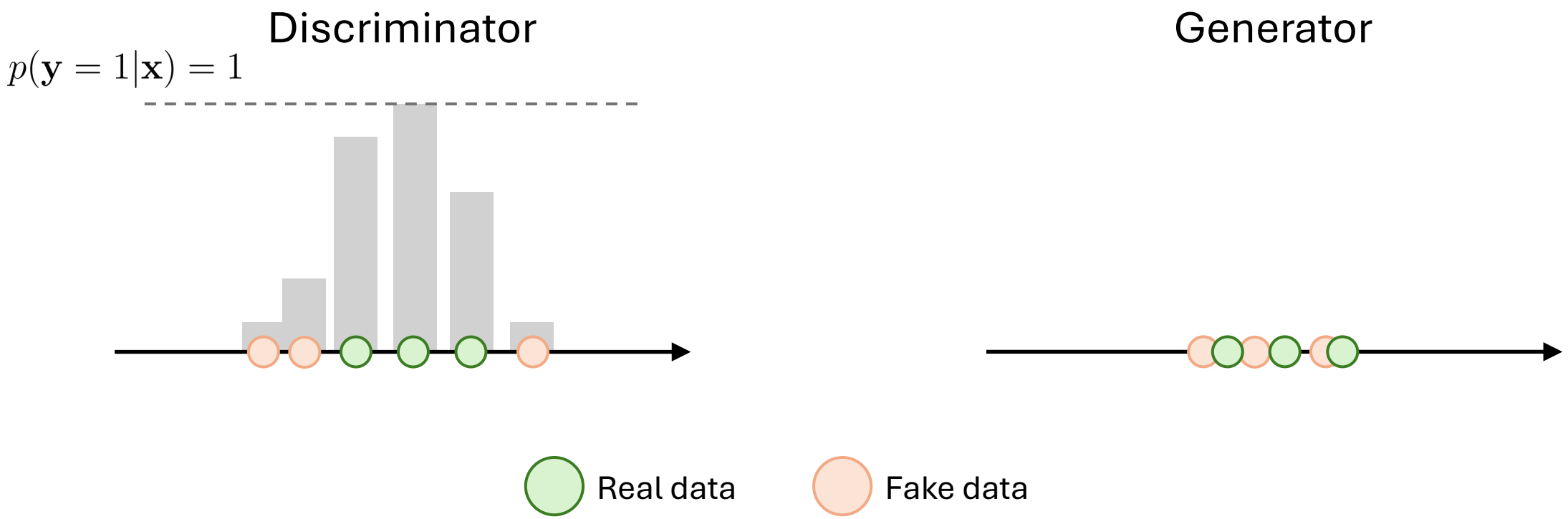
Generator tries to improve its imitation of data.



Intuition behind GANs

Discriminator tries to identify real data from fakes created by the generator.

Generator tries to create imitations of data to trick the discriminator.



Maths behind GANs

Assume generative model with the base latent distribution and deterministic map:

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$

noise

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \delta\left(\mathbf{x} - \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z})\right)$$

“fake” sample from learned representation of data distribution

Maths behind GANs

Discriminator: a classifier, which distinguish real samples from data distribution $\pi(\mathbf{x})$ and generated samples from $p_{\boldsymbol{\theta}}(\mathbf{x})$:

$$D_{\boldsymbol{\phi}}(\mathbf{x}) = p_{\boldsymbol{\phi}}(y = 1|\mathbf{x}) \in [0, 1]$$

Discriminator tries to **minimize** cross entropy.

$$\max_{D} \left[\mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z})))\right]$$

Maths behind GANs

Generator: generative model, which makes the generated sample more realistic:

$$\mathbf{x} = G_{\boldsymbol{\theta}}(\mathbf{z}), \quad \mathbf{z} \sim p(\mathbf{z})$$

Generator tries to **maximize** cross entropy.

$$\min_{G} \left[\mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z}))) \right]$$

GAN objective

$$\min_G \max_D \left[\mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z}))) \right]$$

What are optimal $\boldsymbol{G}$ and $\boldsymbol{D}$?

Assumption

Generative distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ equals to the true distribution $\pi(\mathbf{x})$ if we can not distinguish them using discriminative model D. It means that $D(\mathbf{x}) = 0.5$ for each sample $\mathbf{x}$.

Theorem

The minimax game $\min_G \max_D \left[\underbrace{\mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z})))}_{V(G,D)} \right]$

has the global optimum $\pi(\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x})$, in this case $D^*(\mathbf{x}) = 0.5$.

Proof

$$V(G,D) = \mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log\left(1 - D(\mathbf{x})\right)$$

$$= \int \underbrace{\left[\pi(\mathbf{x}) \log D(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x}) \log\left(1 - D(\mathbf{x})\right)\right]}_{y(D)} d\mathbf{x}$$

$$\frac{dy(D)}{dD} = \frac{\pi(\mathbf{x})}{D(\mathbf{x})} - \frac{p_{\boldsymbol{\theta}}(\mathbf{x})}{1 - D(\mathbf{x})} = 0$$

$$\boxed{D^*(\mathbf{x}) = \frac{\pi(\mathbf{x})}{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}}$$

Proof continued (fixed $\boldsymbol{D = D*}$)

$$
\begin{aligned}
V(G, D^*) &= \mathbb{E}_{\pi(\mathbf{x})} \log \left(\frac{\pi(\mathbf{x})}{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})} \right) + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})} \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x})}{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})} \right) \\
&= \mathrm{KL} \left(\pi(\mathbf{x}) \| \frac{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2} \right) + \mathrm{KL} \left(p_{\boldsymbol{\theta}}(\mathbf{x}) \| \frac{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2} \right) - 2 \log 2 \\
&= 2\mathrm{JSD} \left(\pi(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x}) \right) - 2 \log 2
\end{aligned}
$$

Jensen-Shannon divergence (symmetric KL divergence)

$$
\mathrm{JSD} \left(\pi(\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{x}) \right) = \frac{1}{2} \left[\mathrm{KL} \left(\pi(\mathbf{x}) \| \frac{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2} \right) + \mathrm{KL} \left(p_{\boldsymbol{\theta}}(\mathbf{x}) \| \frac{\pi(\mathbf{x}) + p_{\boldsymbol{\theta}}(\mathbf{x})}{2} \right) \right]
$$

Could be used as a distance measure!

$$
\boxed{V(G^*, D^*) = -2 \log 2, \quad \pi(\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{x}), \quad D^*(\mathbf{x}) = 0.5}
$$

GAN optimality

Expectations

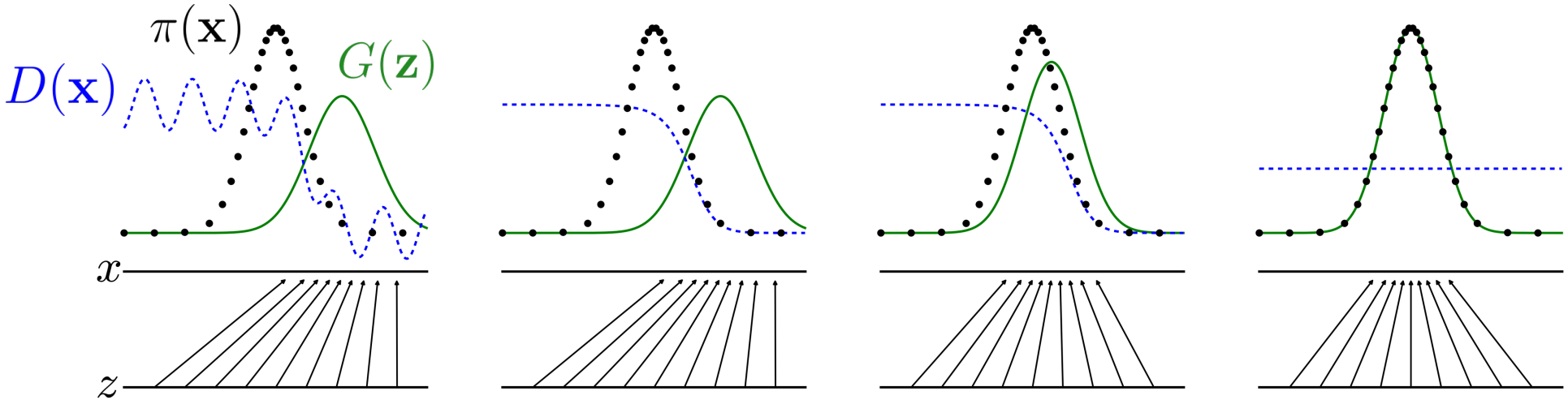
If the generator could be **any** function and the discriminator is **optimal** at every step, then the generator is **guaranteed to converge** to the data distribution.

Reality

- Generator updates are made in parameter space, discriminator is not optimal at every step
- Generator and discriminator loss keeps oscillating during GAN training.

GAN training

$$\min_G \max_D \left[\mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z})))\right]$$

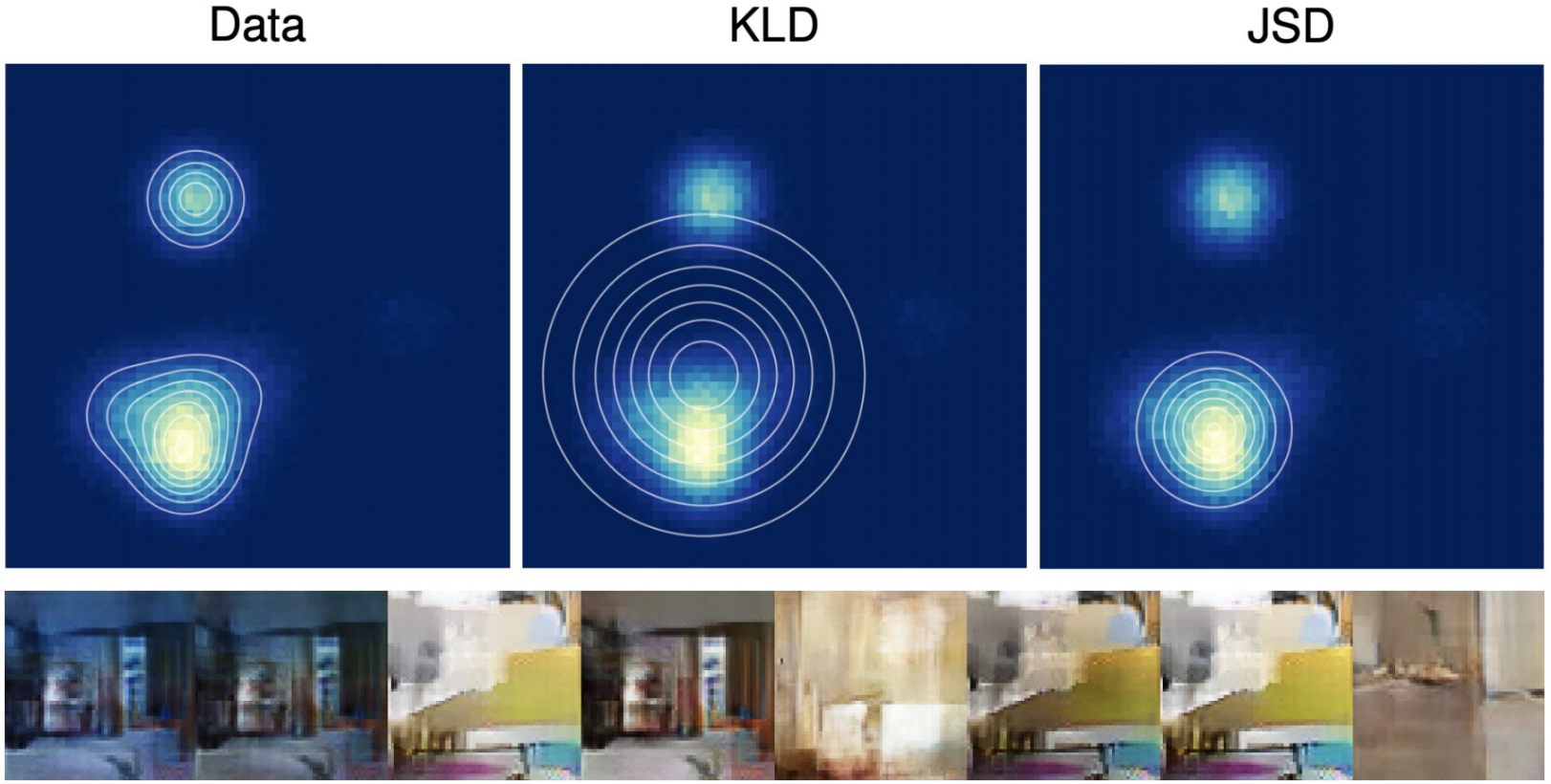


- $\mathbf{z} \sim p(\mathbf{z})$ is a latent variable.
- $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \delta\big(\mathbf{x} - G(\mathbf{z})\big)$ is deterministic decoder.
- We do not have encoder at all.

Goodfellow I. J. et al. Generative adversarial nets //Advances in neural information processing systems. – 2014.

Mode collapse

The phenomena where the generator of a GAN collapses to one or few distribution modes.



Theis L., Oord A., Bethge M. A note on the evaluation of generative models, 2015

How do we do in practice?

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.

StyleGAN – high quality faces

Karras T., Laine S., Aila T. A style-based generator architecture for generative adversarial networks, 2018

Pros and Cons

Advantages

- Capable of producing highly realistic synthetic data
- Flexible framework adaptable to images, text, audio and more
- Encouraged rapid progress in generative modeling research

Disadvantages

- Training is unstable and prone to mode collapse
- Difficult to evaluate model performance quantitatively
- Require large datasets and computational resources

Recap

- What are Generative Models?
- Taxonomy
- Autoregressive Models
- Autoencoders
- VAEs
- GANs

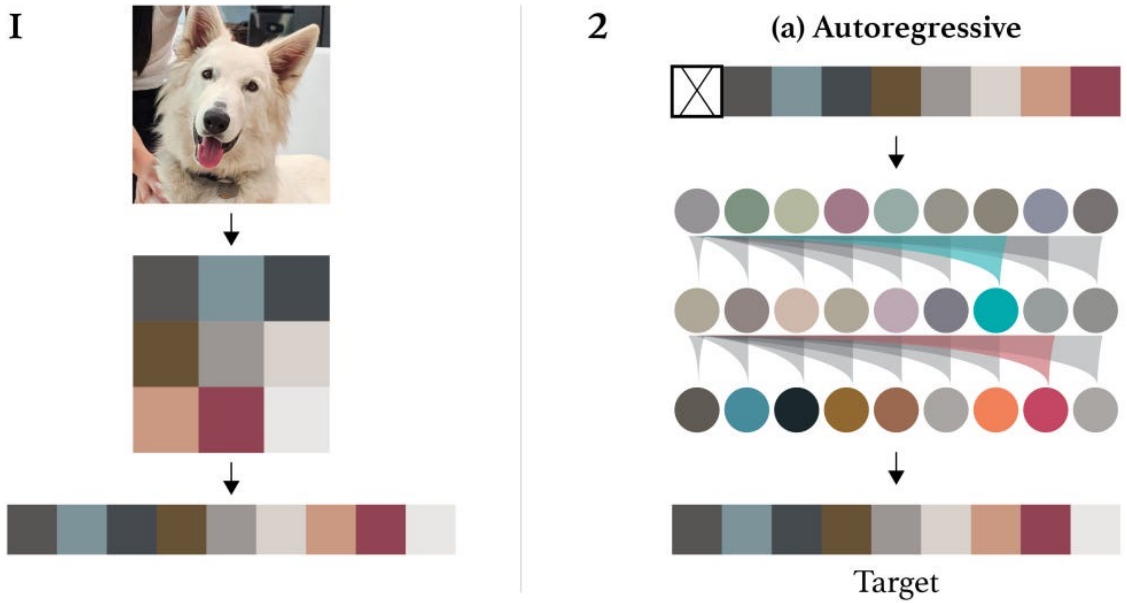
* Extra: Modern Architectures

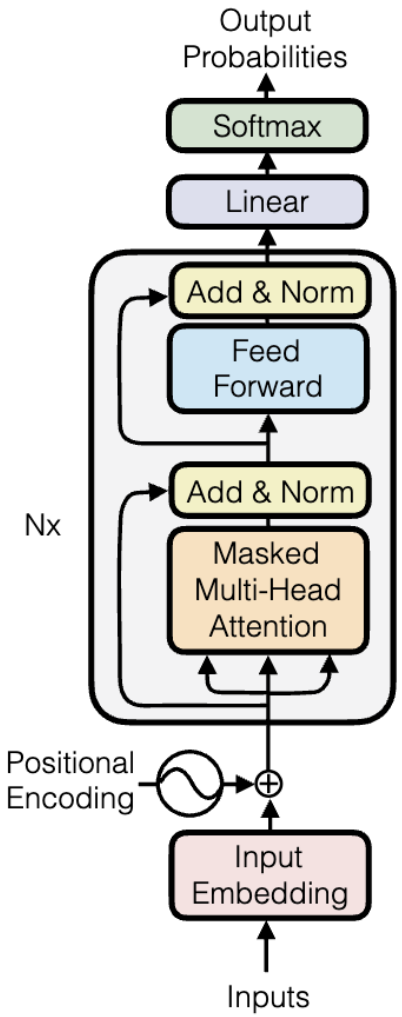
TIME FOR A BREAK

Extra: Modern Architectures

Autoregression

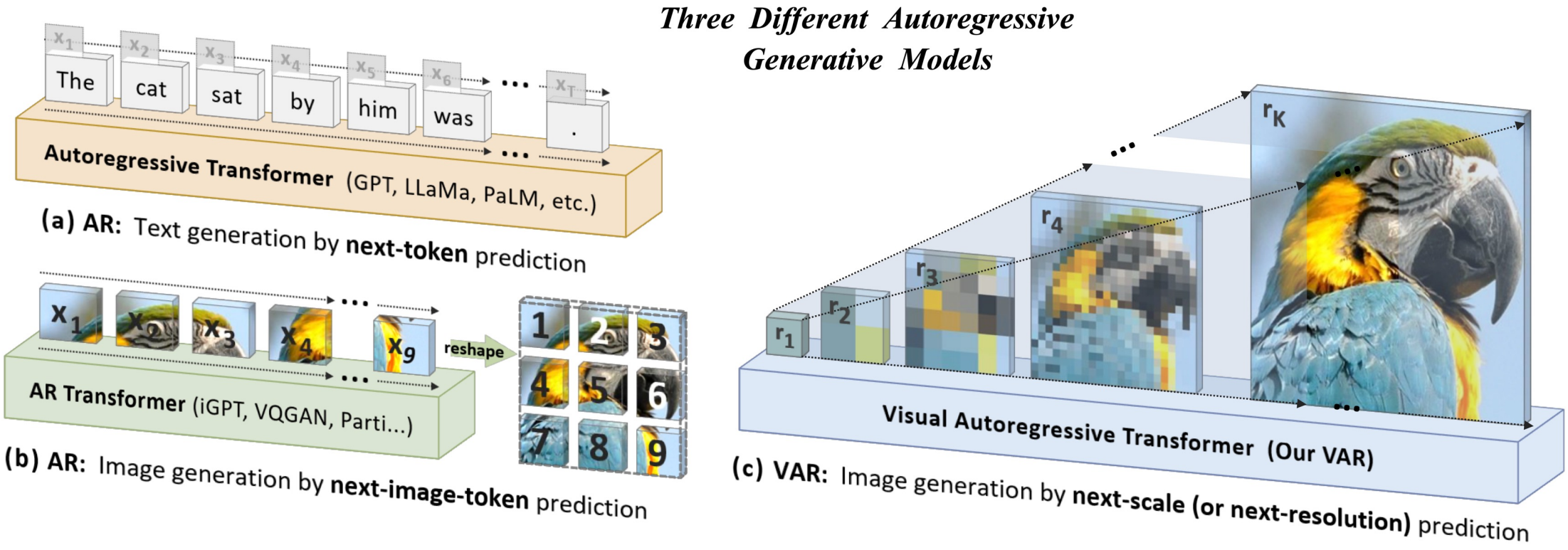
Image GPT





(OpenAI) Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D. Sutskever, I. Generative Pretraining From Pixels, 2020

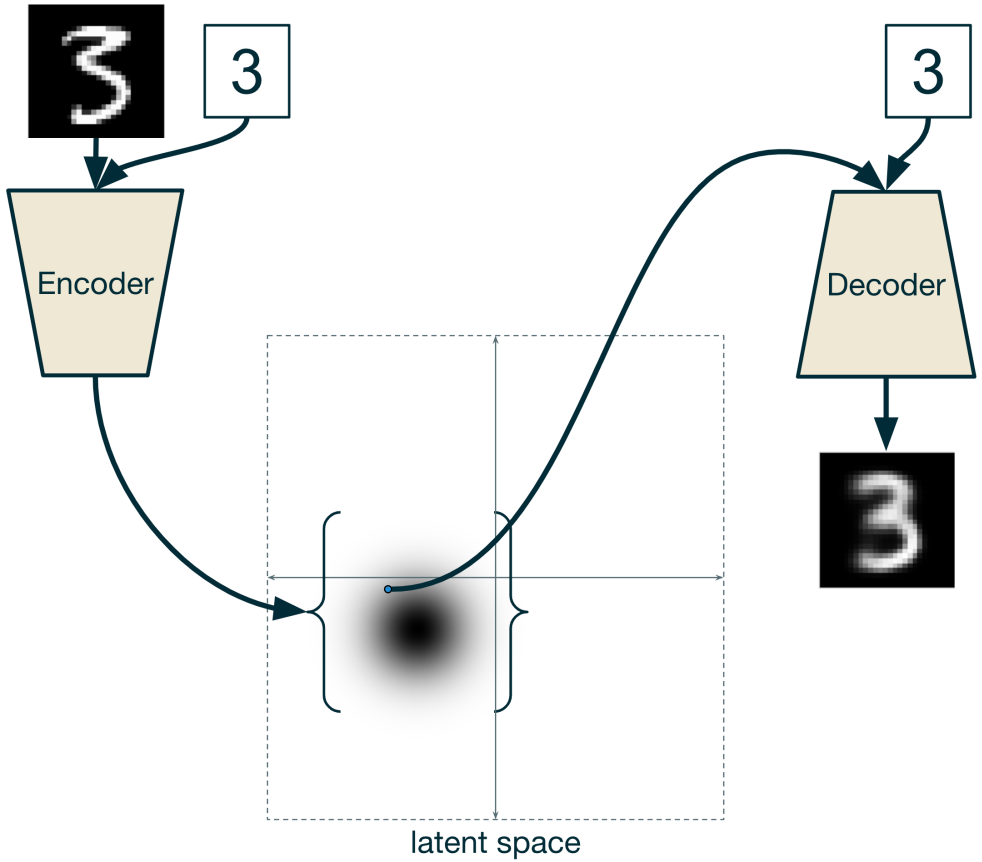
VAR: Visual Autoregressive Modeling



Tian K. et al. Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction, 2024

VAE

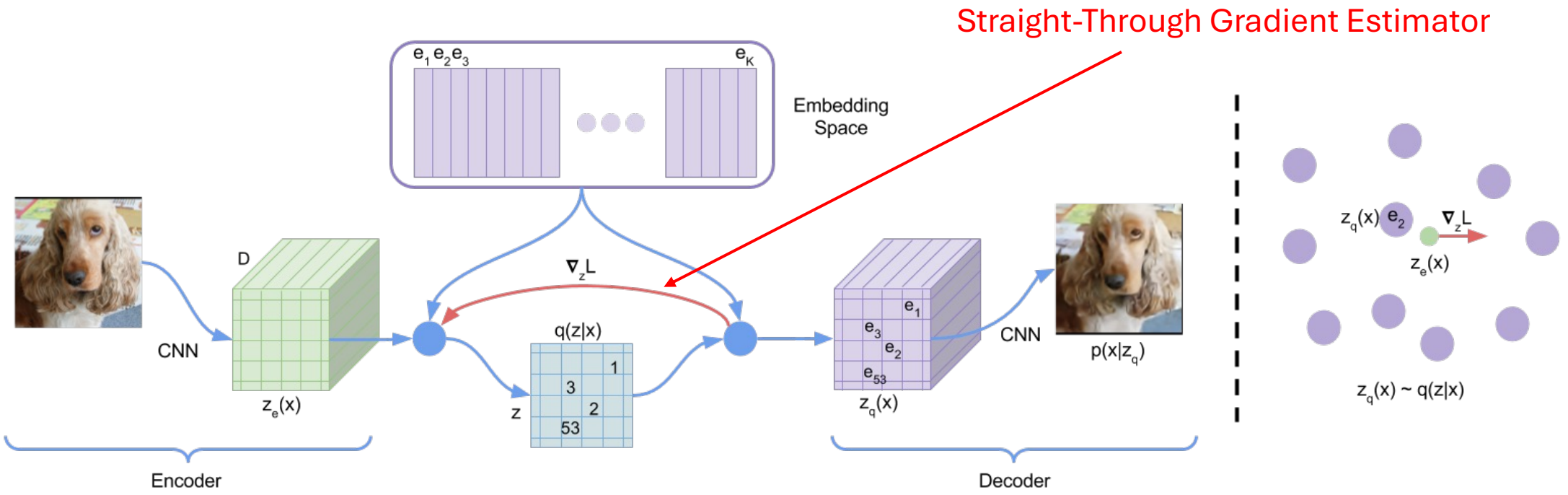
cVAE: Conditional VAE



$$\mathcal{L}_{\text{cVAE}}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x},\mathbf{y})} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, \mathbf{y}) - \text{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{y})||p(\mathbf{z}))$$

Kingma D. P. et al. Semi-supervised learning with deep generative models, 2014

VQ-VAE: Vector Quantized VAE



$$\mathcal{L}_{\text{VQ-VAE}}(\mathbf{x}) = \underbrace{\|\mathbf{x} - \mathcal{D}(\mathbf{e}_k)\|_2^2}_{\text{reconstruction loss}} + \underbrace{\|\text{sg}\left[\mathcal{E}(\mathbf{x})\right] - \mathbf{e}_k\|_2^2}_{\text{VQ loss}} + \underbrace{\beta \cdot \|\mathcal{E}(\mathbf{x}) - \text{sg}\left[\mathbf{e}_k\right]\|_2^2}_{\text{commitment loss}}$$

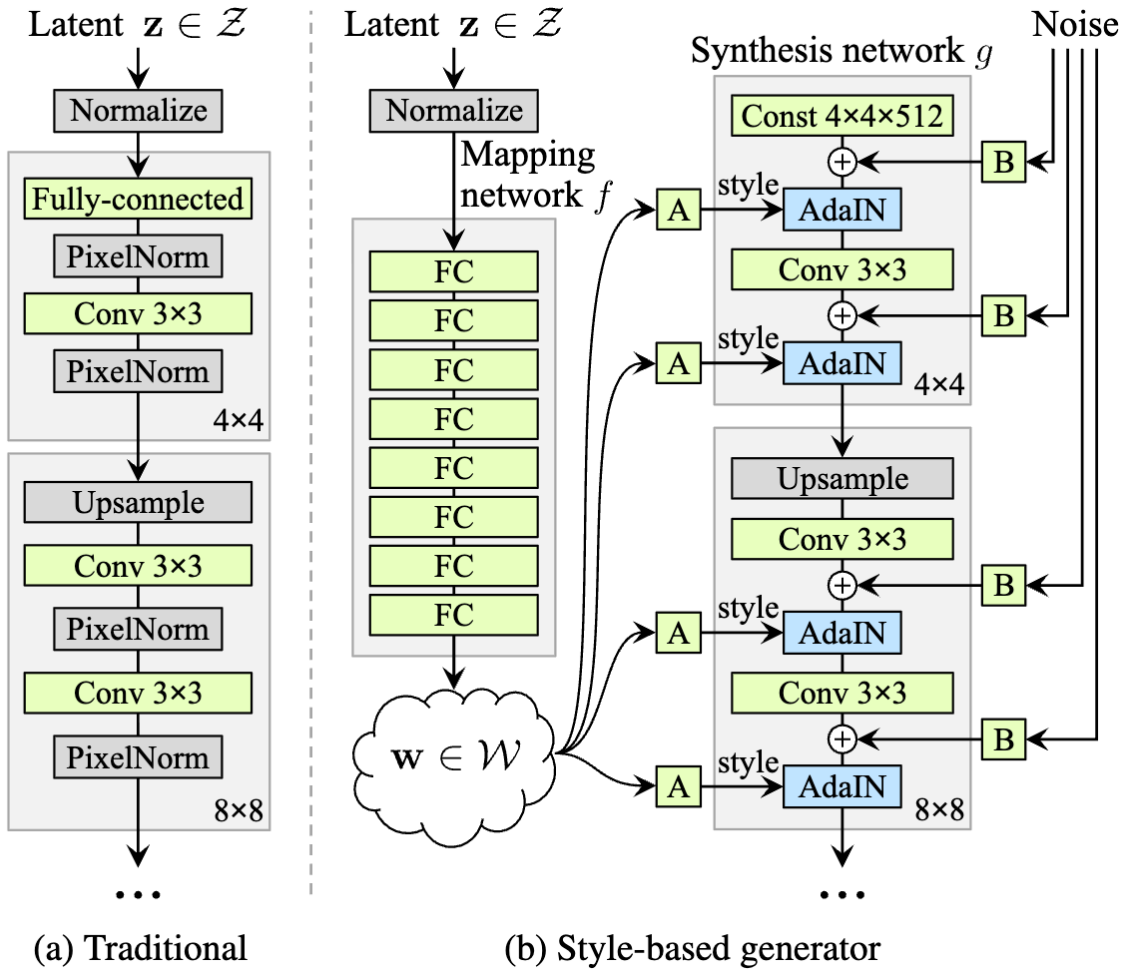
Van Den Oord A. et al. Neural discrete representation learning, 2017

GAN

StyleGAN

1. Style-based generator: maps latent **z** to intermediate latent **w** that controls image attributes
2. Stochastic variation: adds per-pixel noise to each layer creating realistic micro-details (pores, hair, textures)
3. Disentangled latent control: smooth and semantically meaningful manipulations in **w**-space (e.g., pose, lighting, identity)

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$



Karras T., Laine S., Aila T. A style-based generator architecture for generative adversarial networks, 2018