Generative AI

Dr. Xiao Wang
J.O. Berger and M.E. Bock Professor of Statistics
Purdue University
URL: https://www.stat.purdue.edu/~wangxiao/

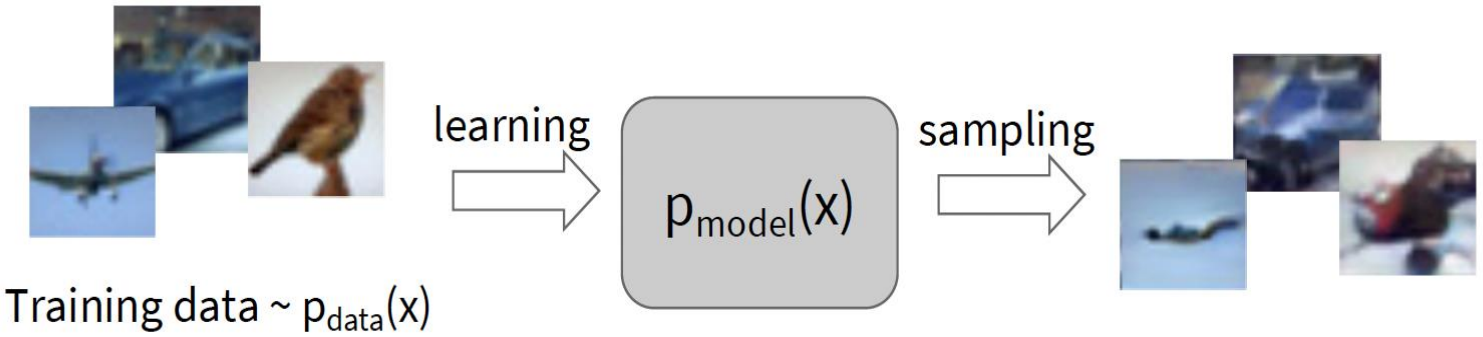
Content

Content

1 Introduction

What are Generative Models?

Definition: Generative models learn to generate new data samples resembling a given dataset.



Objectives:

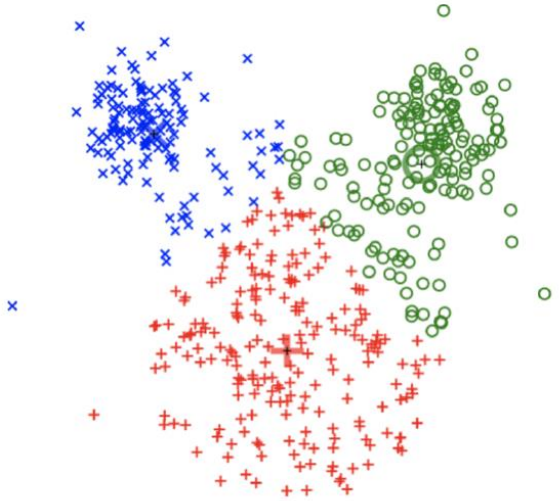
1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. Sampling new x from $p_{model}(x)$

Major Generative Models:

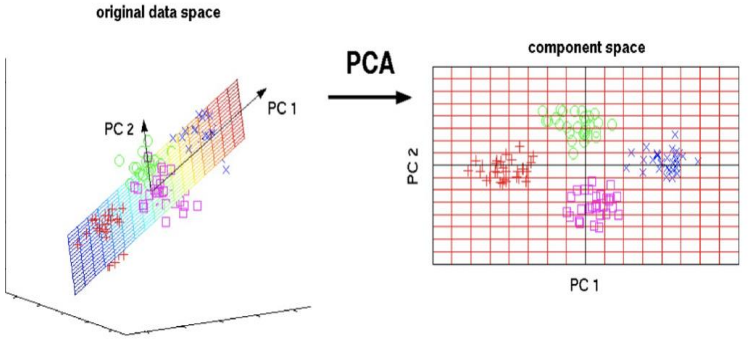
- **Explicit Density Models:** Estimate probability distributions (e.g., Gaussian Mixture Models, VAEs).
- **Implicit Density Models:** Generate samples without explicit density estimation (e.g., Generative Adversarial Network (GAN)s, Diffusion Models).

Unsupervised Learning

- **Data:** X — Just data, no labels
- **Goal:** Learn some underlying hidden structure or distribution of the data
- **Examples:** clustering, dimension reduction, feature learning, density estimation, etc.
- **Generative models are a subset of unsupervised learning, but not all unsupervised learning techniques are generative (e.g., k-means, PCA)**



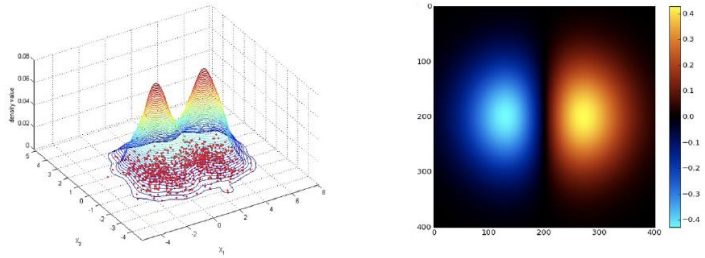
K-means clustering



PCA

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Modeling p(x)

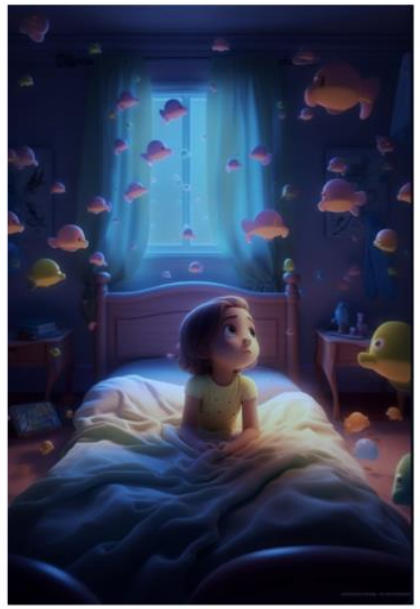
2-d density images left and right are CC0 public domain

GenAI on Face Generation

- Better Quality
- High Resolution

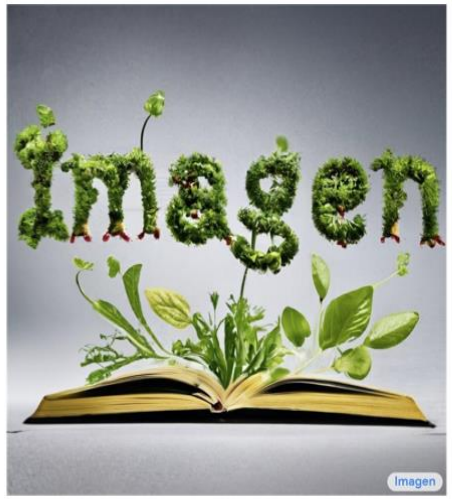
1024*1024 Images generated by a GAN created by NVIDIA. (source, 2018)

AI Art



Midjourney v5

Emerging Generative Models in 2022-



alphacode
it solutions

Parti
Pathways Autoregressive Text-to-Image Model

DESIGNS.AI

DALL·E 2

Stable Diffusion

Additional Applications

Text to Image

Fashion Design

Deep Generative Models

Deep generative models are neural network-based models designed to learn complex data distributions and generate realistic synthetic samples that resemble the original training data. These models leverage deep learning to approximate the true underlying data distribution.

Let $X \sim P_X$, where P_X is the distribution of X. Let its density function be p_X.
There are two ways to learn the distribution of X:

- The explicit modeling approach assumes $p_X \in \mathcal{P}_\Theta$, or estimates p_X directly nonparamtrically.
- Generative models learn a generator function $G : \mathbb{R}^m \to \mathbb{R}^p$ such that $G(\eta) \sim P_X$, where $\eta \sim P_\eta$, a known reference distribution.
  - If a generator function G is known, then we know everything about P_X, since we can first sample $\eta \sim P_\eta$, then $G(\eta) \sim P_X$.
  - We usually take the reference distribution to be $N(\mathbf{0}, \mathbf{I}_m)$ or uniform distribution on $[0, 1]^m$.

Taxonomy of Deep Generative Models

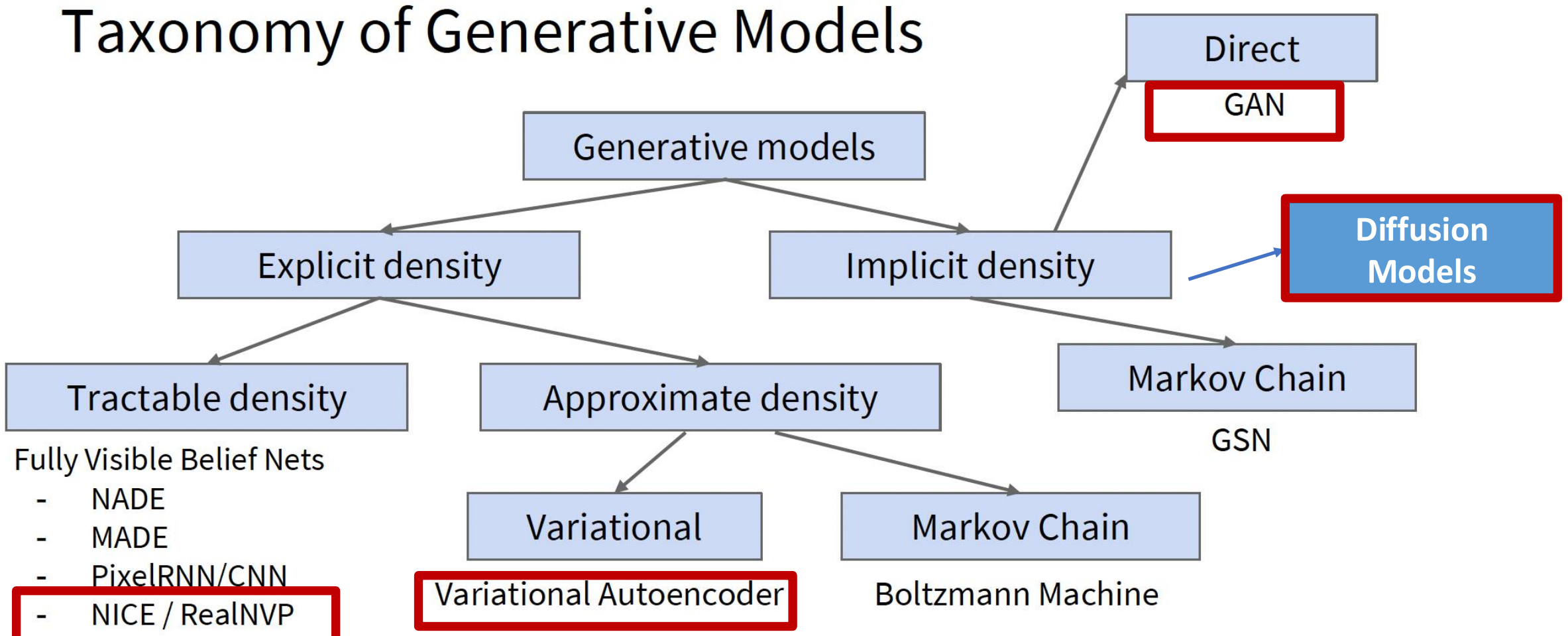
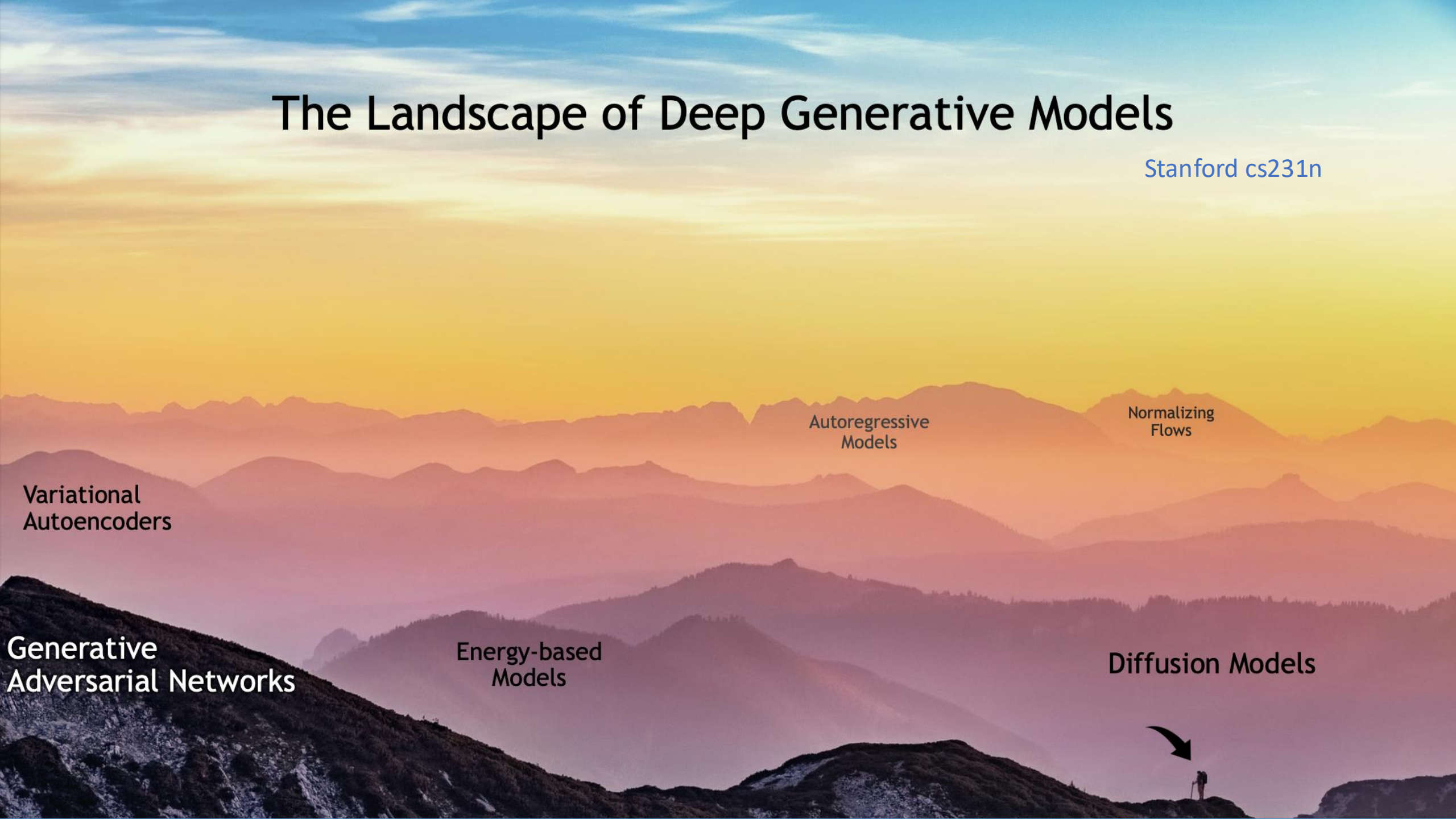


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.


The Landscape of Deep Generative Models
Stanford cs231n
Autoregressive Models
Normalizing Flows
Variational Autoencoders
Generative Adversarial Networks
Energy-based Models
Diffusion Models

Evaluation Metrics

Metric	Measures	Best for	Limitations
Inception Score (IS)	Quality & diversity	Image GANs	Doesn't compare to real data
Fréchet Inception Distance (FID)	Realism & diversity	Image GANs	Requires feature extraction
Precision & Recall	Fidelity & coverage	Any model	Computationally expensive
Log-Likelihood	Probability assignment	VAEs, Flows	Doesn't match human perception
Human Evaluation	Subjective quality	Any model	Expensive and subjective
Downstream Task Performance	Utility in real tasks	Task-driven models	Domain-dependent

Applications of Generative Models in AI

- **Understanding Probability Distributions**
  - Generative models help represent and manipulate high-dimensional probability distributions across various fields.
- **Role in Reinforcement Learning (RL)**
  - Used in model-based RL to simulate possible futures for planning & decision-making.
  - Enables learning in imaginary environments, reducing risks of real-world errors.
  - Guides exploration by tracking visited states & attempted actions.
  - Supports inverse RL for learning from expert demonstrations.
- **Handling Missing Data & Semi-Supervised Learning**
  - Can train with missing data and predict missing inputs.
  - Enables semi-supervised learning, reducing the need for labeled data.
- **Multi-Modal Learning & Sample Generation**
  - Allows multiple correct outputs for a single input (e.g., video frame prediction).
  - GANs excel in generating realistic samples for various AI applications.

Content

What are Autoencoders?

Autoencoders are neural networks designed for dimensionality reduction and feature extraction by compressing and reconstructing data.

They consist of two main components:

- **Encoder (e)**: Maps input x to a **low-dimensional latent space** z, where similar inputs have similar latent representations.

  e: X→Z, z=e(x) with dim(X)≫dim(Z)

- **Decoder (d)**: Reconstructs x from its latent representation z, mapping back to the original input space.

  d: Z→X and $\hat{x} = d(z) = d(e(x))$.

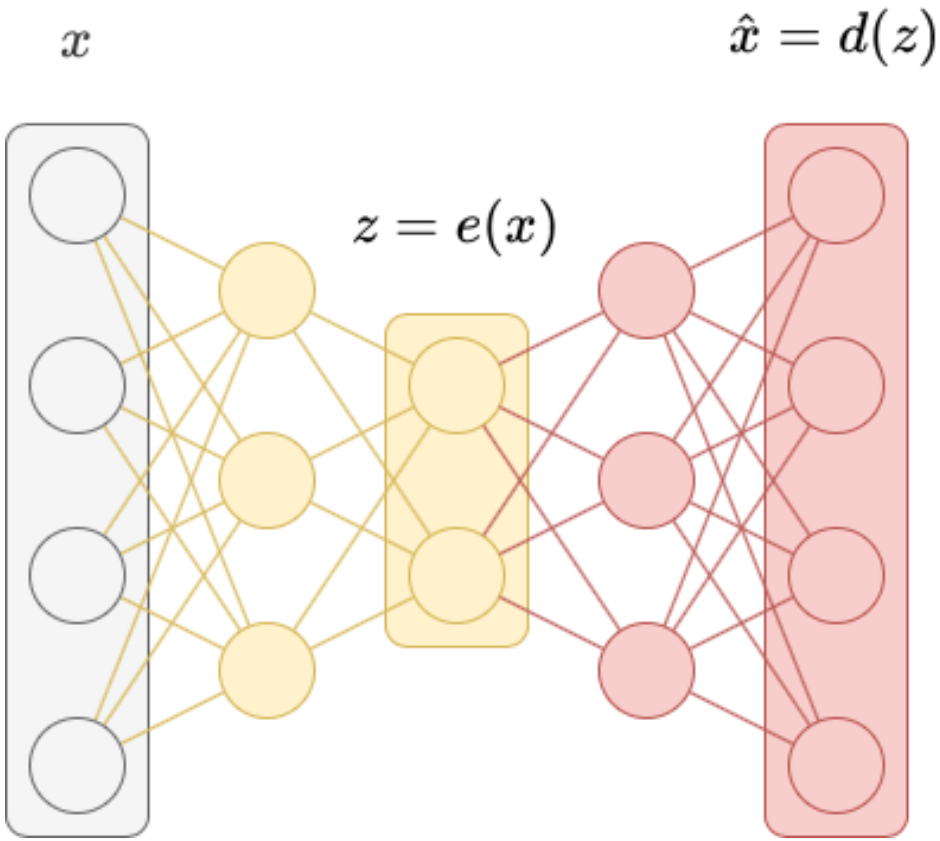
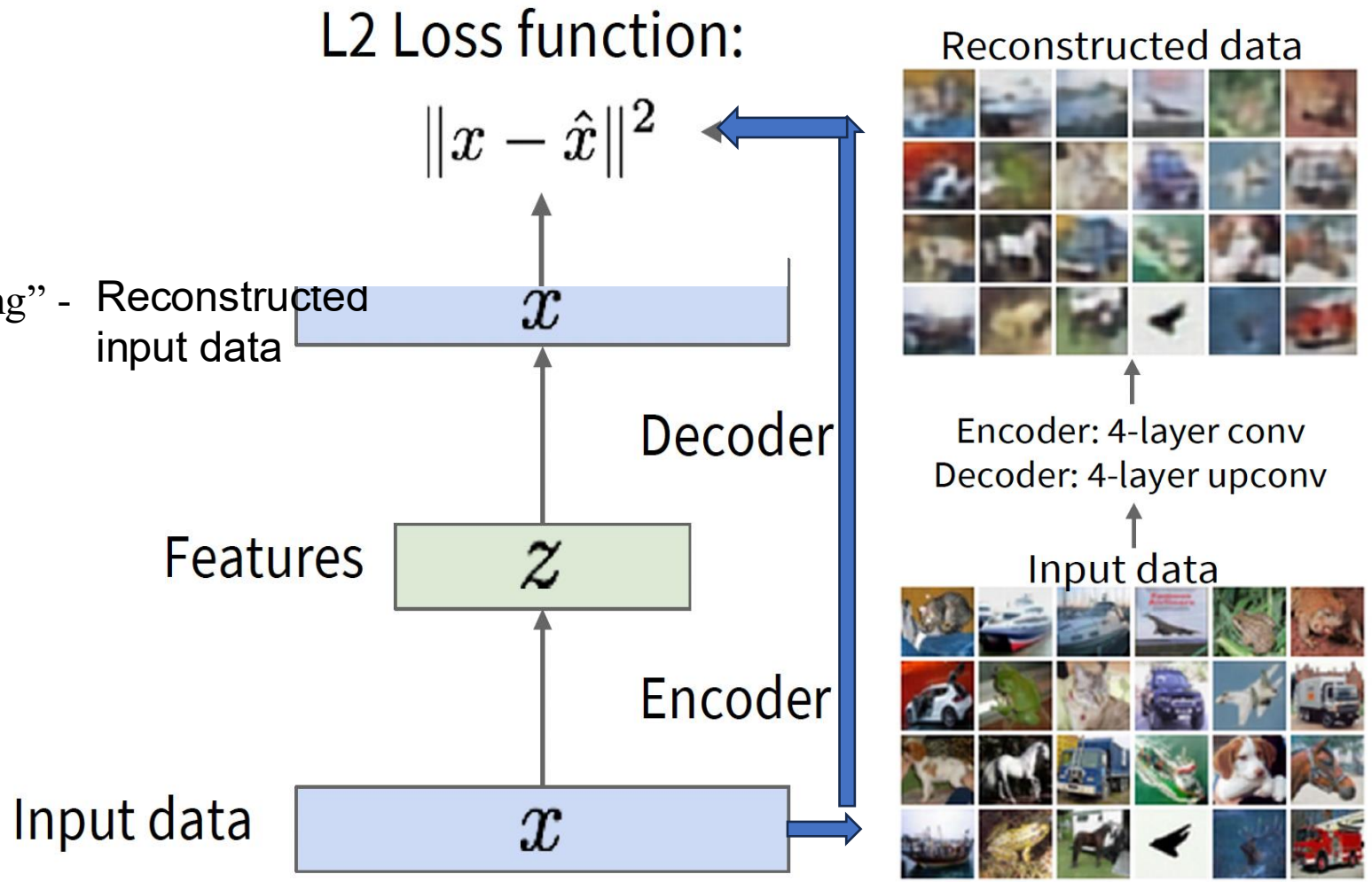


Illustration of autoencoder (source)

What are Autoencoders?

Train such that features can be used to reconstruct original data "Autoencoding" - encoding input itself

Want features to capture meaningful factors of variation in data



Autoencoder Latent Space and Its Limitations

- **Trained on MNIST**, the autoencoder clusters similar digits in the **latent space**.
- **Decoder can reconstruct images** from latent vectors, but gaps in the latent space cause issues.
- **Generative models** aim to produce new samples, but **disjoint latent spaces** in autoencoders make some sampled latent vectors meaningless.
- **Illustration:** In the **top-left corner of the latent space**, unseen regions result in unrealistic reconstructions.
- **Solution: Variational Autoencoders (VAEs)** introduce structured latent spaces to ensure continuity and improve generative performance.

📌 **Key Issue:** Autoencoders are great for representation learning but struggle as generative models due to fragmented latent spaces.

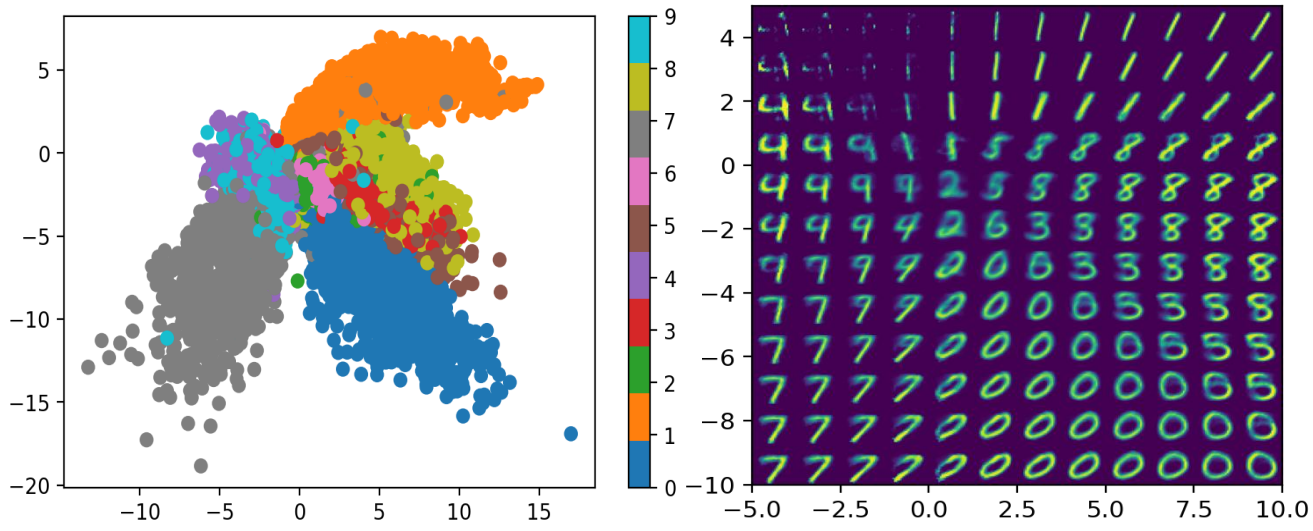


Illustration of example latent vectors using the MNIST dataset (source)

What is a Variational Autoencoder?

📌 **VAE = Autoencoder + Generative Modeling**

- **Same structure as a traditional autoencoder:**
  - ❖ **Encoder:** Compresses input into a latent space representation, but instead of a single point, outputs a probability distribution (Gaussian).
  - ❖ **Decoder:** Samples from this distribution and reconstructs the input.

📌 **Key Difference from Traditional Autoencoders**

❖ Traditional autoencoders map inputs deterministically to a single latent vector z=e(x).

❖ VAEs introduce probabilistic encoding, ensuring smooth and structured latent spaces for better generative performance.

✨ Benefit: Enables meaningful interpolation and sampling for generating new data! 🚀

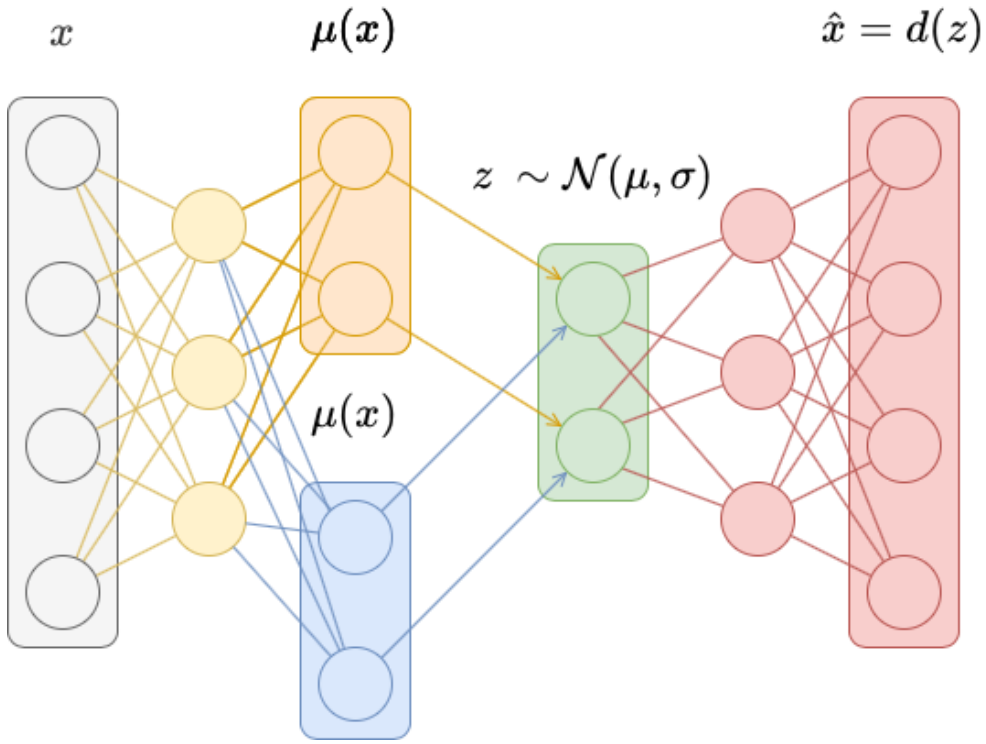


Illustration of VAE (source)

Variational Autoencoder as a DGM

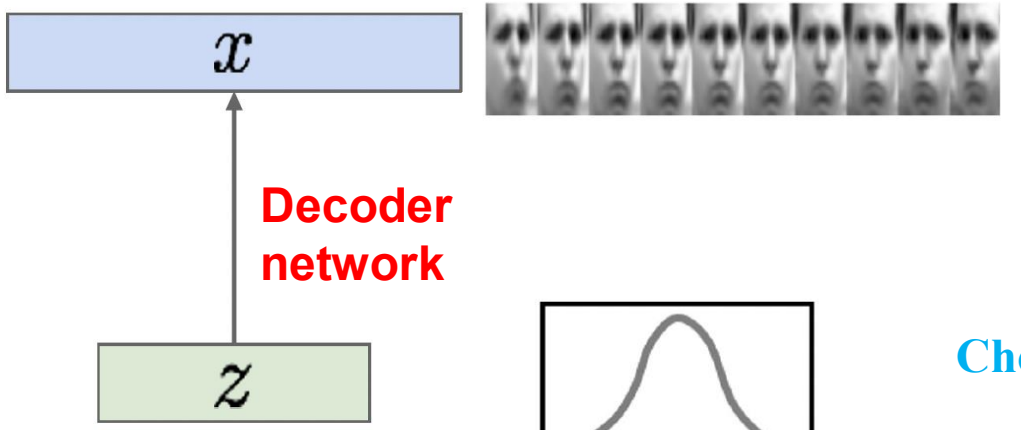
VAEs define an intractable density function with latent

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from the distribution of unobserved (latent) representation z

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$



Conditional p(x|z) is complex (generates image) => represent with neural network

Sample from true prior $z^{(i)} \sim p_{\theta^*}(z)$

Choose prior p(z) to be simple, e.g. Gaussian.

How to train VAE?

Learn model parameters to maximize likelihood of training data

We want to estimate the true parameters of this generative model given training data $\{x^{(i)}\}_{i=1}^{N}$

Q: What is the problem with this? Intractable!

Data Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^{k} p(x|z^{(i)}), \text{where } z^{(i)} \sim p(z)$$

Intractable to compute p(x|z) for every z!

Monte Carlo estimation is too high variance

Posterior distribution

$$p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$$

Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network

$$q_\theta(z|x) \approx p_\theta(z|x)$$

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

How to approximate VAE?

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives pθ(x|z), can compute estimate of this term through sampling (need some trick to differentiate through sampling).

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

pθ(z|x) intractable (saw earlier), can't compute this KL term. But we know KL divergence always >= 0.

How to approximate VAE?

Decoder: reconstruct the input data

Encoder: make approximate posterior distribution. close to prior

$$\log p_\theta(x^{(i)}) = \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

We want to maximize the data likelihood.

Tractable lower bound which we can take gradient of and optimize!
($p\theta(x|z)$ differentiable, KL term differentiable)

Variational evidence lower bound (ELBO):

$$\mathcal{L}(x, \theta, \phi) \leq \log p_\theta(x)$$

Training: Maximize lower bound

$$\hat{\theta}, \hat{\phi} = \arg\max_{\theta,\phi} \sum_{i=1}^{n} \mathcal{L}(x_i, \theta, \phi)$$

Stochastic Optimization of ELBO

Algorithm 1: Stochastic optimization of the ELBO. Since noise originates from both the minibatch sampling and sampling of $p(\boldsymbol{\epsilon})$, this is a doubly stochastic optimization procedure. We also refer to this procedure as the *Auto-Encoding Variational Bayes* (AEVB) algorithm.

Data:

$\mathcal{D}$: Dataset

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model

$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model

Result:

$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters

while *SGD not converged* **do**

  $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)

  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)

  Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$

  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

end

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

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

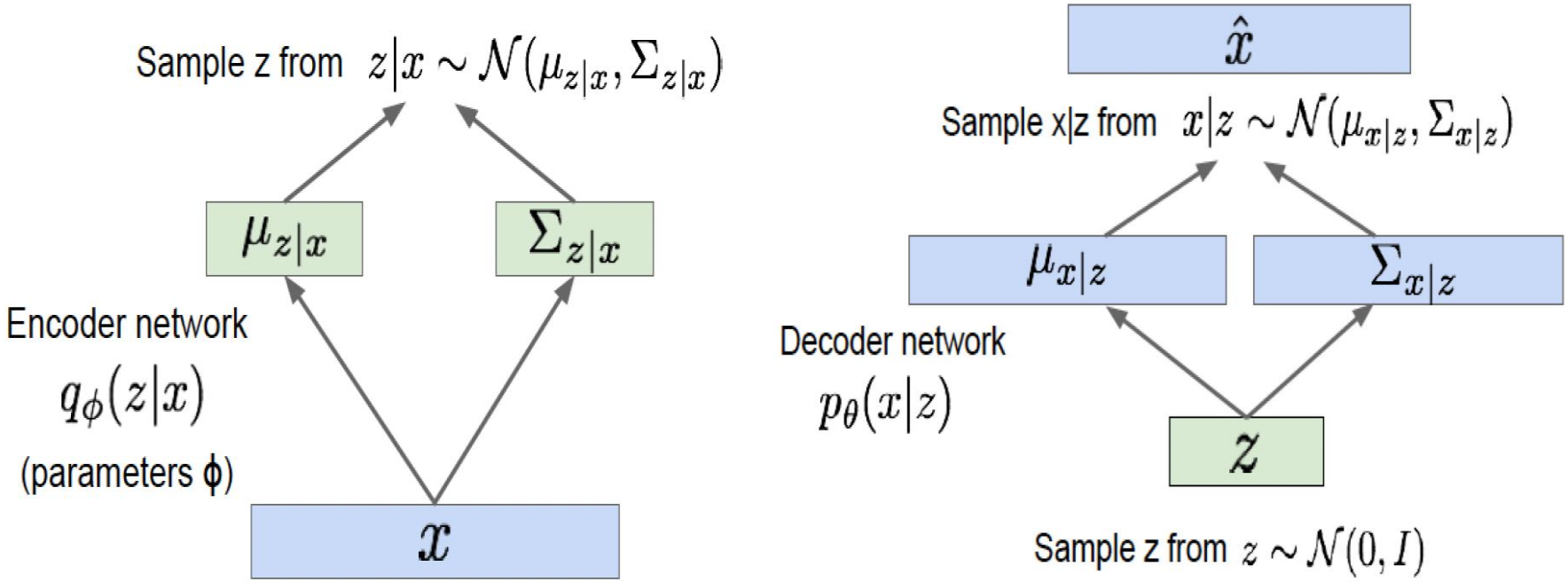
$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]$$

Reparametrization Trick: $\mathbf{z} = \mathbf{g}(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \mathbf{x})$

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right]$$

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f(\mathbf{z})\right] \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} f(\mathbf{z})\right] \\ &\simeq \nabla_{\boldsymbol{\phi}} f(\mathbf{z})\end{aligned}$$

A Theoretical Example



$$\int q_{\boldsymbol{\theta}}(\mathbf{z}) \log p(\mathbf{z})\, d\mathbf{z} = \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})\, d\mathbf{z}$$

$$= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^{J} (\mu_j^2 + \sigma_j^2)$$

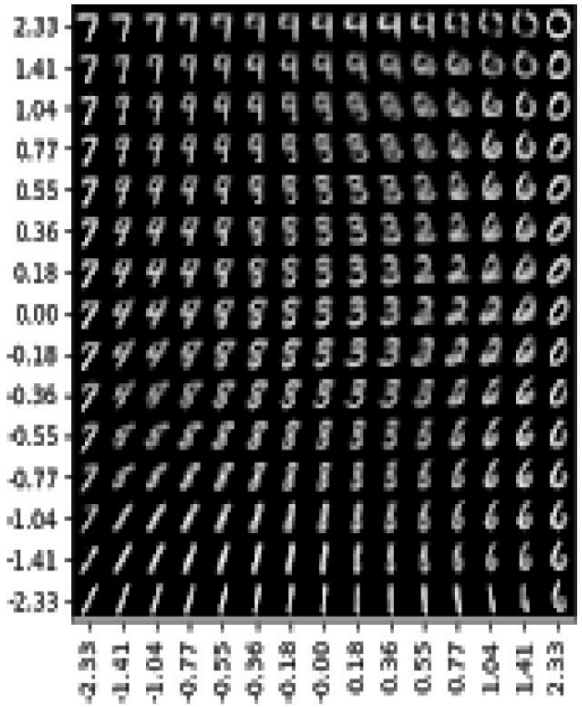
$$\log p(\mathbf{x}|\mathbf{z}) = \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I})$$

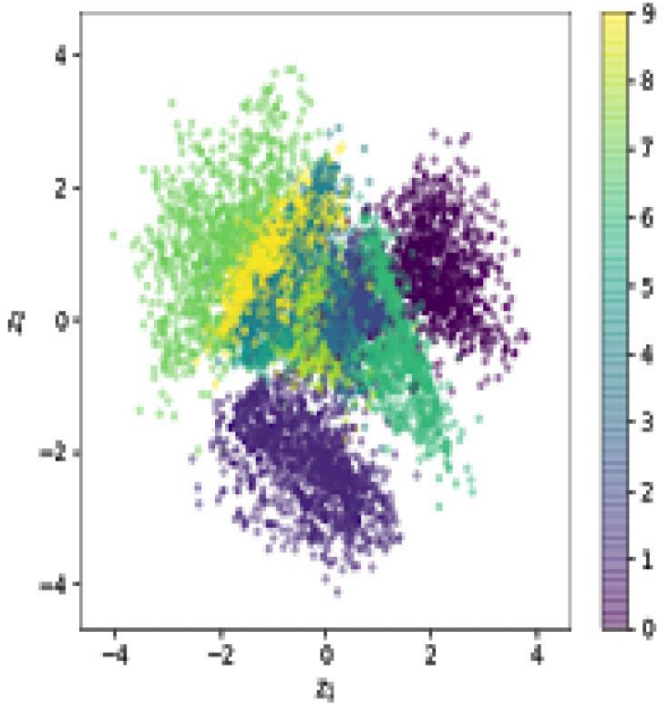
$$\text{where } \boldsymbol{\mu} = \mathbf{W}_4 \mathbf{h} + \mathbf{b}_4$$

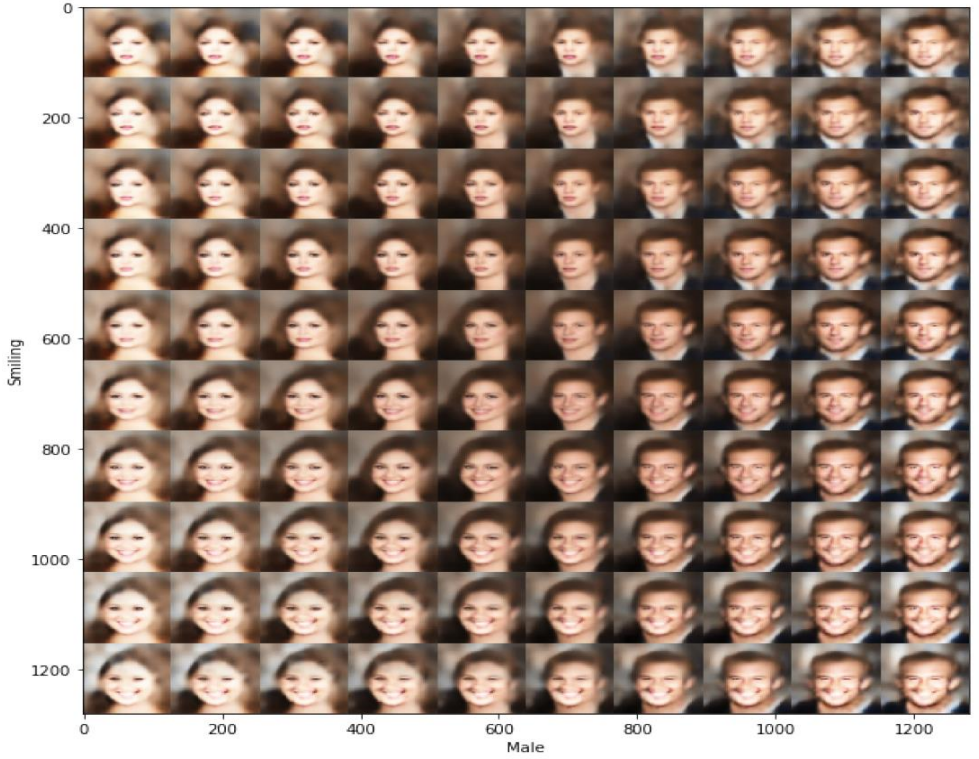
$$\log \boldsymbol{\sigma}^2 = \mathbf{W}_5 \mathbf{h} + \mathbf{b}_5$$

$$\mathbf{h} = \tanh(\mathbf{W}_3 \mathbf{z} + \mathbf{b}_3)$$

Real Examples







Strengths & Limitations

📌 **Key Idea:**

- Adds a probabilistic spin to traditional autoencoders, enabling data generation.
- Defines an intractable density, requiring variational inference to derive and optimize a lower bound (ELBO).

📌 **Pros:**

✓ Principled generative approach based on probabilistic modeling.
✓ Interpretable latent space enables meaningful structure in representations.
✓ Inference of q(z|x) allows feature extraction for other tasks.

📌 **Cons:**

❌ Optimizes a lower bound on likelihood, which may not be an ideal evaluation metric.
❌ Lower sample quality compared to PixelRNN/PixelCNN.
❌ Blurry reconstructions compared to GANs, which generate sharper images.

📌 **Active Research Areas:**

🔷 Flexible Approximate Posteriors: Moving beyond diagonal Gaussian assumptions to richer models like Gaussian Mixture Models (GMMs) or Categorical Distributions.
🔷 Disentangled Representations: Learning independent latent factors for better interpretability.
🔷 Improving Training Objectives: Hybrid models incorporating adversarial learning (VAE-GANs).
🚀 **Future Directions**: Enhancing sample quality while retaining VAE's structured latent space!

Content

“This (GANS), and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion”

–Yann LeCun

What is GAN?

Problem:

- ❖ We want to sample from a complex, high-dimensional training distribution.
- ❖ There is no direct way to explicitly learn or model the data distribution.

Solution:

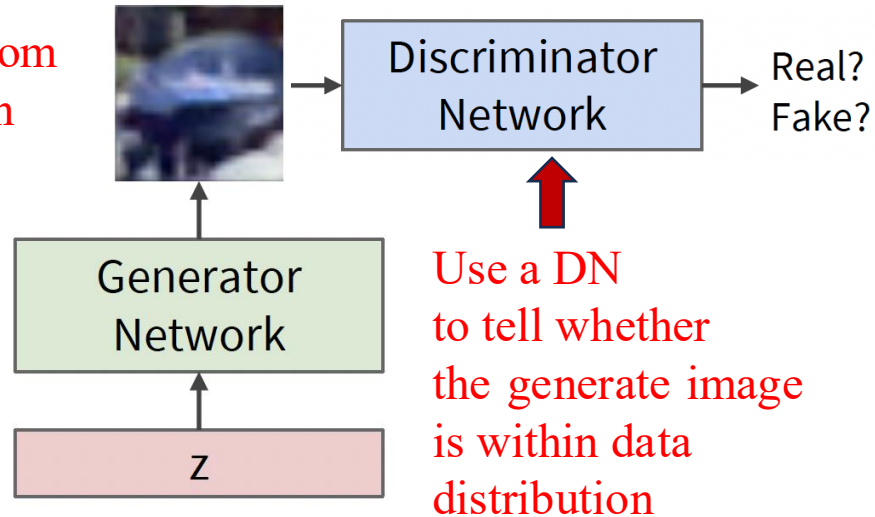
- ➢ Instead of learning the distribution explicitly, GANs learn a transformation.
- ➢ Start by sampling from a simple distribution (e.g., Gaussian noise).
- ➢ Train a neural network to transform the simple distribution into the training data distribution.

Key Idea:

- ✓ GANs learn to generate new samples indirectly through adversarial training.
- ✓ The model never explicitly estimates the probability density function of the data.

Objective: generated images should look "real"

Output: Sample from training distribution



Use a DN to tell whether the generate image is within data distribution ("real") or not

Input: Random noise

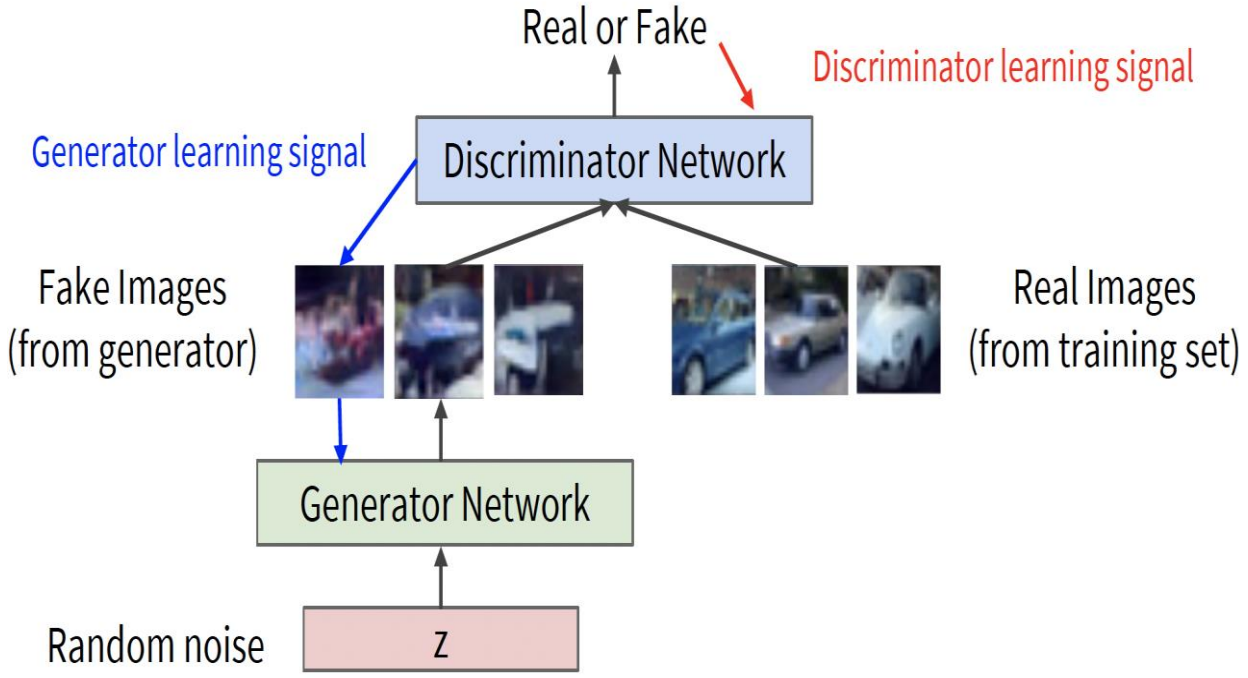
Generative Vs Discriminative Models

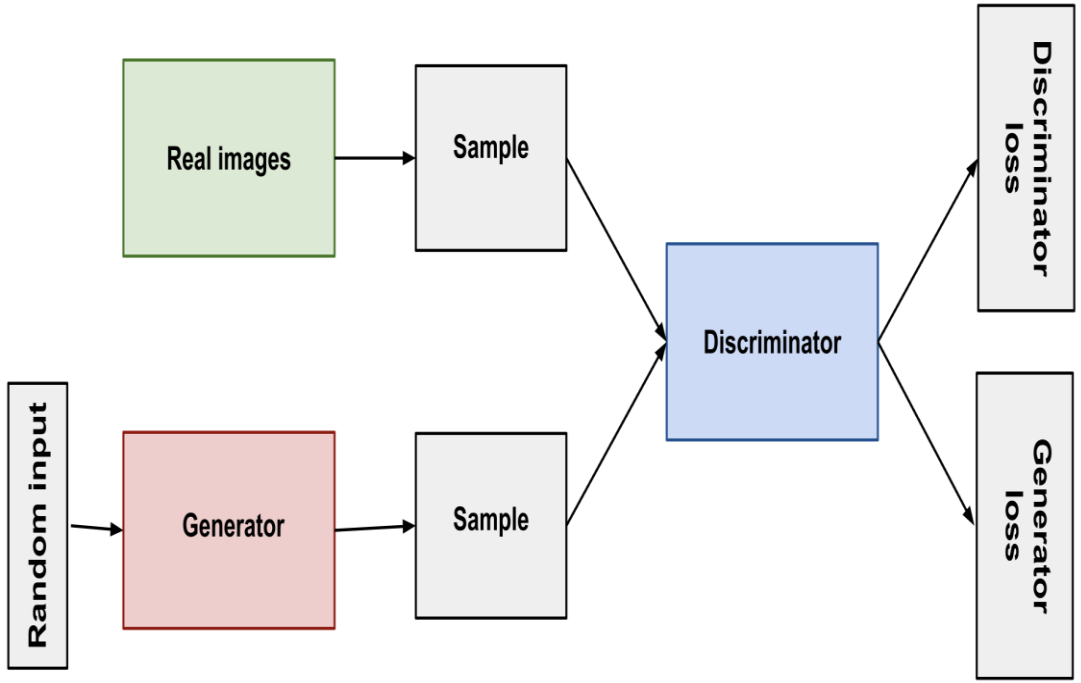
Generative Models:

- Can generate new data samples resembling real data.
- Example: GANs generate realistic images that resemble real ones.

Discriminative Models:

- Focus on classification by distinguishing between different categories.
- Example: A decision tree can classify dogs and cats but cannot generate them.





Overview of GAN Training

Discriminator Network: Tries to distinguish between real and fake images.
Generator Network: Tries to fool the discriminator by generating real-looking images.
Training Process: Both networks are trained jointly in a minimax game.

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Generator objective

Discriminator objective

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- **Discriminator** (θ_d) wants to maximize the objective such that $D(x) \approx 1$ (real) and $D(G(z)) \approx 0$ (fake).
- **Generator** (θ_g) wants to minimize the objective such that $D(G(z)) \approx 1$ (fooling the discriminator).

GAN Training

Alternate between:

1. Gradient ascent on discriminator

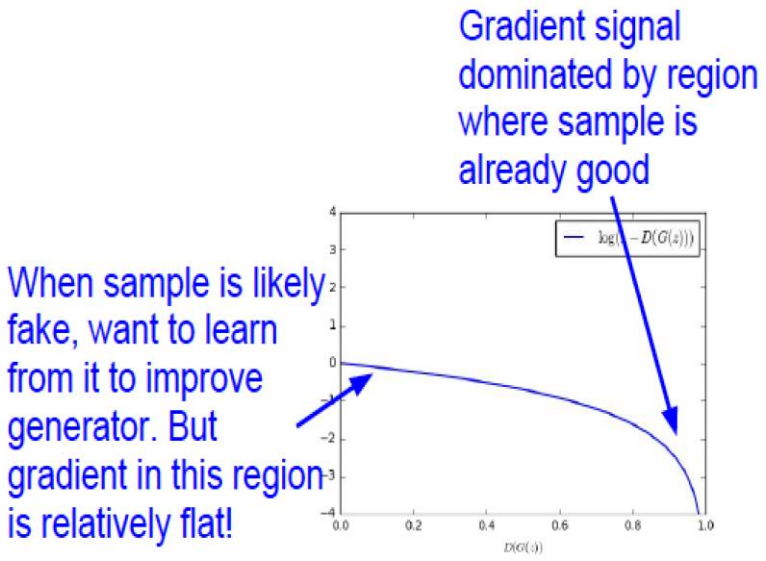
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

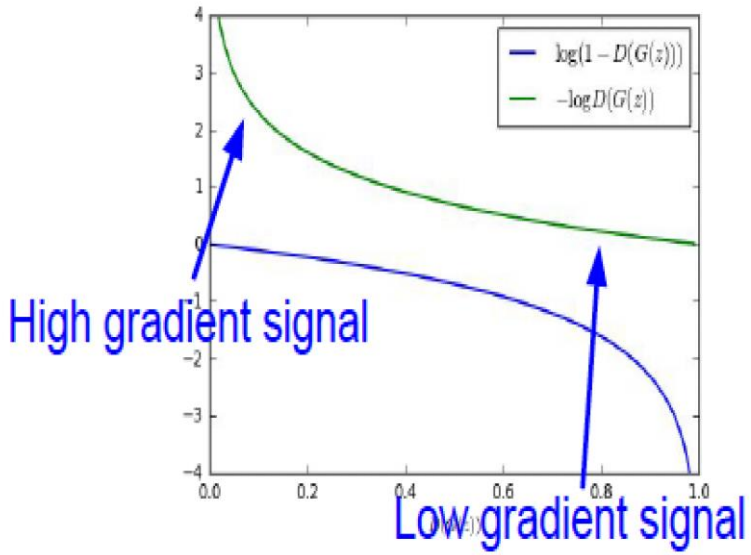
2. Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Instead: Consider a different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$





GAN Algorithms

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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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 ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Alternating Training for GANs

GAN Training Process (Alternating Phases):

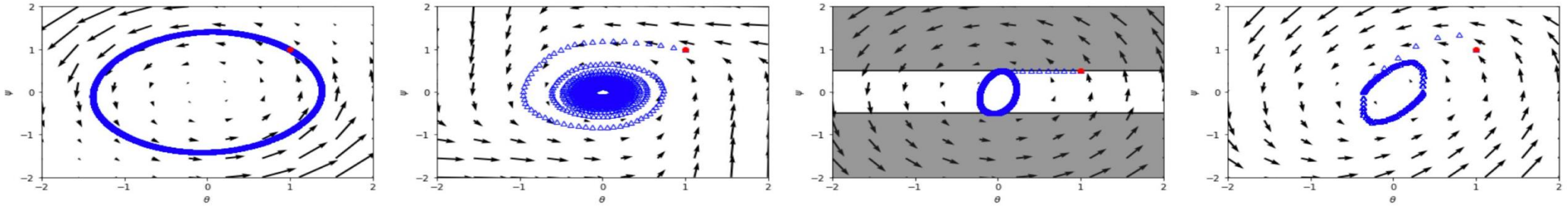
1. Discriminator Training: Trains for one or more epochs while the generator remains unchanged. It learns to differentiate real from generated data, adapting to the generator's flaws.
2. Generator Training: Trains for one or more epochs while the discriminator remains unchanged. This prevents the generator from chasing a moving target.

Training Dynamics:

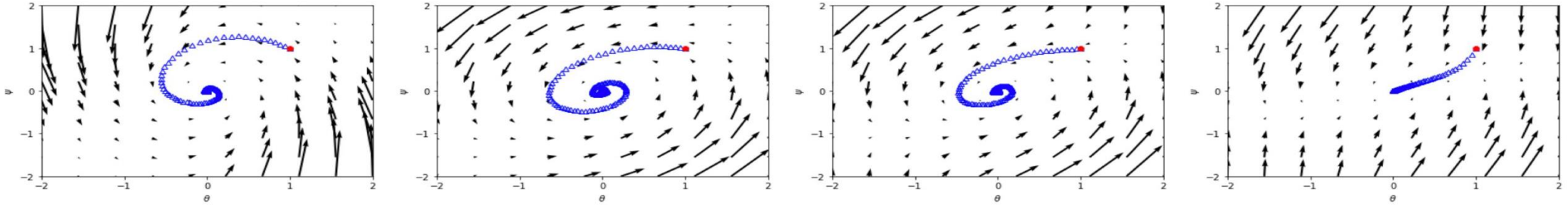
- As the generator improves, the discriminator struggles to distinguish real from fake data.
- A perfect generator results in a discriminator with 50% accuracy (random guessing).
- Overtraining can degrade performance, leading to unstable convergence where the generator receives meaningless feedback.

Challenges in GAN Training

- **Hyperparameter Sensitivity:** GANs are sensitive to learning rates, batch sizes, and architectural choices.
- **Mode Collapse:** The generator produces limited diversity.
- **Training Instability:** The minimax optimization is difficult to balance.
- **Vanishing/Exploding Gradients:** The discriminator can become too strong or weak, leading to poor gradients.
- **Non-Convergence:** The model oscillates instead of converging.



(a) Standard GAN (b) Non-saturating GAN (c) WGAN ($n_d = 5$) (d) WGAN-GP ($n_d = 5$)



(e) Consensus optimization (f) Instance noise (g) Gradient penalty (h) Gradient penalty (CR)

Mode Collapse

Mode Collapse in GANs refers to a common failure mode where the **generator** fails to capture the full diversity of the data distribution and produces **limited variations** of samples. Instead of generating a wide range of outputs, it collapses to generating a few or even a single type of sample repeatedly.

Why Does Mode Collapse Occur?

- **Imbalanced Generator-Discriminator Learning**
- **Training Instability**
- **Lack of Diversity-Promoting Mechanisms**

1.

Effects of Mode Collapse

- **Reduced Sample Diversity** → Poor representation of the real dataset.
- **Low-Quality Generation** → Outputs look repetitive and lack variety.
- **Unreliable Model** → The generator fails to generalize.

Illustration of example monotonous output. (source)

Techniques to Mitigate Mode Collapse

1.Minibatch Discrimination

Encourages diversity by comparing samples in each batch.

2.Feature Matching

Instead of just fooling the discriminator, the generator learns to match feature statistics of real data.

3.Wasserstein GAN (WGAN)

Uses the Earth Mover (Wasserstein) distance to **stabilize training** and avoid collapsing to few modes.

4.Unrolled GANs

Allows the generator to anticipate discriminator updates, preventing it from getting stuck in mode collapse.

5.Mutual Information Regularization

Forcing the generator to learn **meaningful latent representations** that generate diverse outputs.

.

More details of example GAN suffering mode collapse: https://neptune.ai/blog/gan-failure-modes

Different GANs

Model	Stability	Mode Collapse	Convergence	Sample Quality	Special Features
GAN	Low	High	Unstable	Medium	Baseline
LSGAN	Medium	Medium	More stable	Medium	Least squares loss
WGAN	High	Low	More stable	High	Wasserstein distance
WGAN-GP	Very High	Very Low	Very stable	Very High	Gradient Penalty
cGAN	Medium	Medium	Stable	High	Class conditioning
StyleGAN	High	Low	Stable	Very High	Style control

Loss-variant GANs

- Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, and is an active area of research.
- $X \sim P_X$ vs. $G(Z) \sim P_G$ with $Z \sim N(0, I)$.
- Training GAN is equivalent to minimizing Jensen-Shannon divergence between generator and data distributions.
- $D(P_X, P_G) = \sup_{f \in \mathcal{F}} \left\{ \mathbb{E}_{X \sim P_X} \phi_1(f(X)) - \mathbb{E}_{Y \sim P_G} \phi_2(f(Y)) \right\}$

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x \sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x \sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x \sim p_d}[D(x)] + \mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda \mathbb{E}_{\hat{x} \sim p_g}[(\|\nabla D(\alpha x + (1 - \alpha \hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x \sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x} \sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x} \sim p_g}[(D(\hat{x} - 1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda \mathbb{E}_{\hat{x} \sim p_d + \mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x \sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t \mathbb{E}_{\hat{x} \sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x} \sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$

GAN-related Loss Functions

GANs aim to approximate the real data distribution $p_{data}(x)$ using a generator network $G(\eta)$, where $\eta \sim p_z(\eta)$ is drawn from a simple prior distribution.

- The training process is driven by a discriminator D(x), which distinguishes real from generated samples.
- The loss function should measure **the divergence** between $p_{data}(x)$ and $p_g(x)$ where $p_g(x)$ is the distribution of generated samples

Minimax Loss: In the original GANs, the generator tries to minimize the following function while the discriminator tries to maximize it:

$$\min_G \max_D V(D,G) = E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$$

The formula derives from the cross-entropy between the real and generated distributions.

$$\max_D V(D,G) = \max_D \{\mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1 - D(x))]\}$$

For a given x, the optimal discriminator is given by

$$D^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_g(x)}$$

Thus, minimizing the GAN objective is equivalent to minimizing the Jensen-Shannon divergence as follows:

$$\min_G V(G,D^*) = \min_G JS(P_{\text{data}} \| P_g) + \log 4 = \min_G KL(P_{\text{data}} \| M) + KL(P_g \| M)$$

$$M(x) = \tfrac{1}{2}(P_{\text{data}}(x) + P_g(x))$$

Wasserstein GAN (WGAN)

Let Ω be a subset of $\mathbb{R}^d$. Let $\mathcal{B}_p(\Omega)$ be the set of Borel probability measures on Ω with finite pth moment. The p-Wasserstein metric is defined as

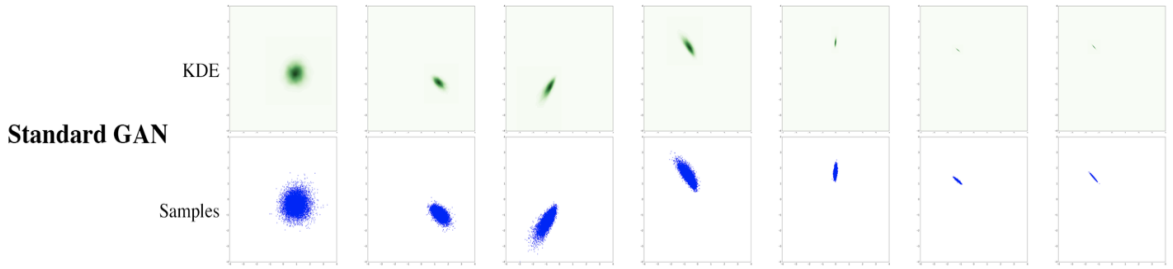
$$W_p(\mu,\nu) = \left(\inf_{\gamma\in\Gamma(\mu,\nu)}\int |x-y|^p d\gamma(x,y)\right)^{1/p}, \quad \mu \text{ and } \nu \in \mathcal{B}_p(\Omega).$$

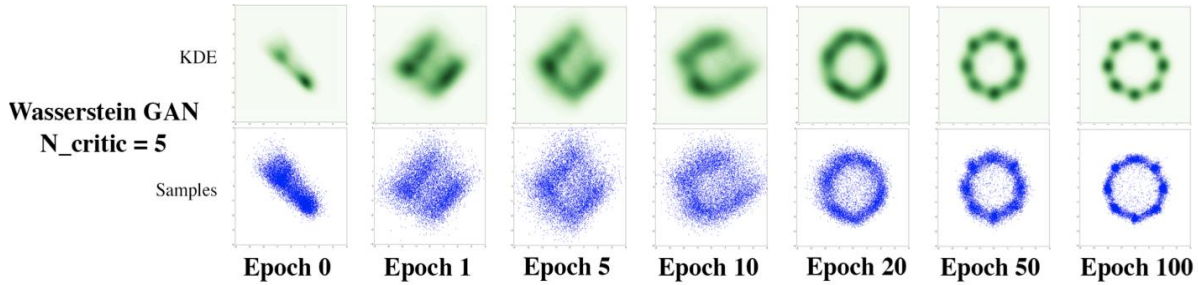
For the special case of p = 1, the p-Wasserstein metric is also known as the Monge-Rubinstein metric, or the earth mover distance.

The 1-Wasserstein metric can be expressed as (Villani, 2008),

$$W_1(\mu,\nu) = \sup_{f\in\mathcal{F}_1}\left\{\int f(x)d\mu(x) - \int f(x)d\nu(x)\right\},$$

This expression of 1-Wasserstein metric is computationally convenient, which is used in the construction of Wasserstein generative adversarial networks (WGAN) (Arjovsky et al., 2017).





Training WGAN

Critic (Discriminator) Loss:

$$L_D = \mathbb{E}_{x \sim P_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim P_z}[D(G(z))]$$

$$\nabla_{\theta_D} L_D = \nabla_{\theta_D} \left(\mathbb{E}_{x \sim P_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim P_z}[D(G(z))] \right)$$

$$\theta_D \leftarrow \theta_D + \eta_D \nabla_{\theta_D} L_D$$

Methods to Enforce Lipschitz Constraint:

Weight Clipping (Original WGAN):

$$-c \leq w \leq c$$

Gradient Penalty (WGAN-GP):

$$L_{\text{GP}} = \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$

Generator Loss:

$$L_G = -\mathbb{E}_{z \sim P_z}[D(G(z))]$$

$$\nabla_{\theta_G} L_G = -\nabla_{\theta_G} \mathbb{E}_{z \sim P_z}[D(G(z))]$$

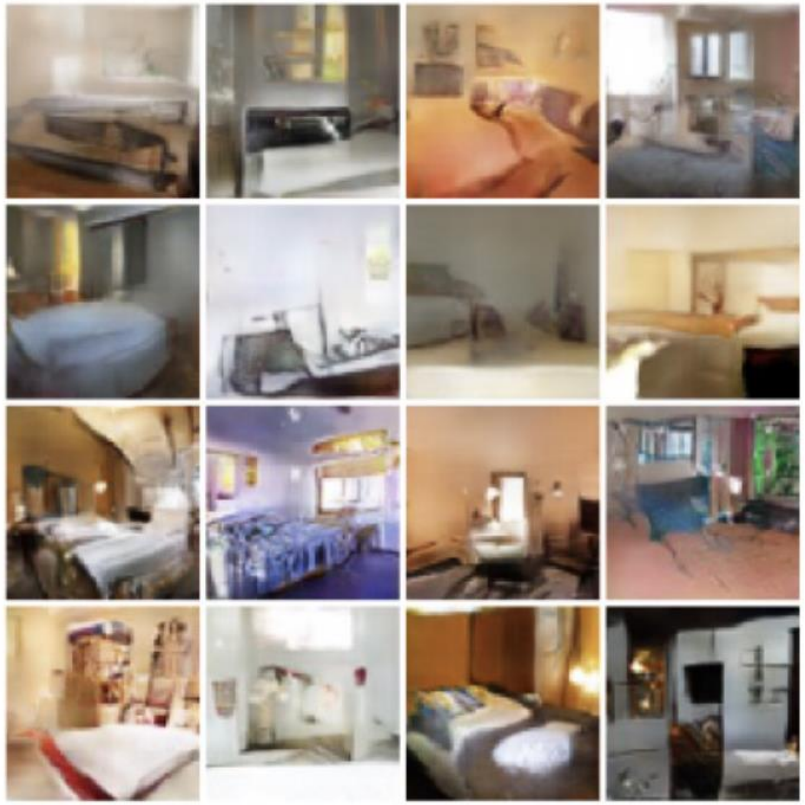
$$\theta_G \leftarrow \theta_G + \eta_G \nabla_{\theta_G} L_G$$

Training Process:

1. Update the critic D multiple times per generator update.
2. Compute Wasserstein distance using the critic's output.
3. Update generator G to minimize the critic's output.

WGAN vs WGAN-NP

Wasserstein GAN (WGAN)



Arjovsky, Chintala, and Bouttou, "Wasserstein GAN", 2017

WGAN with Gradient Penalty (WGAN-GP)

Gulrajani et al, "Improved Training of Wasserstein GANs", NeurIPS 2017

Tips to improve GAN performance

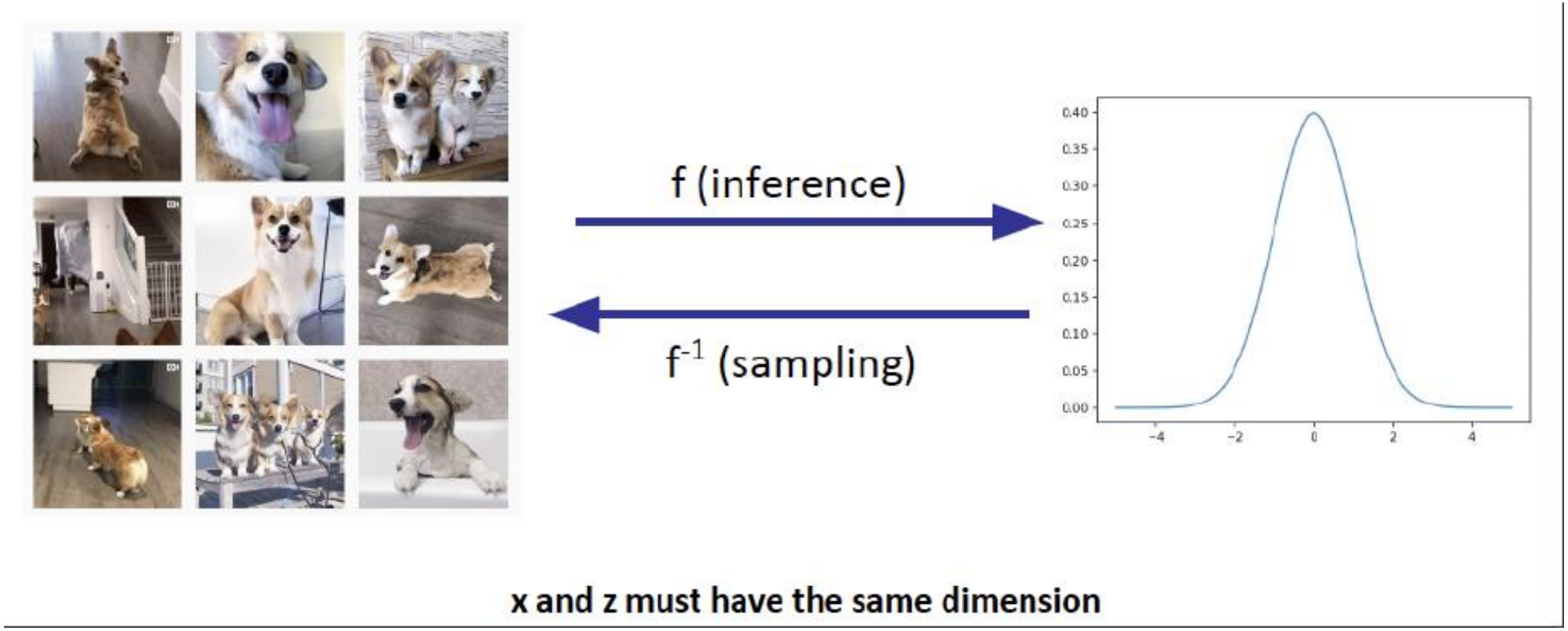
- Change the cost function for a better optimization goal.
- Add additional penalties to the cost function to enforce constraints.
- Avoid overconfidence and overfitting.
- Better ways of optimizing the model.
- Add labels (Conditional GAN).
- More details and other implementation tips: https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

Content

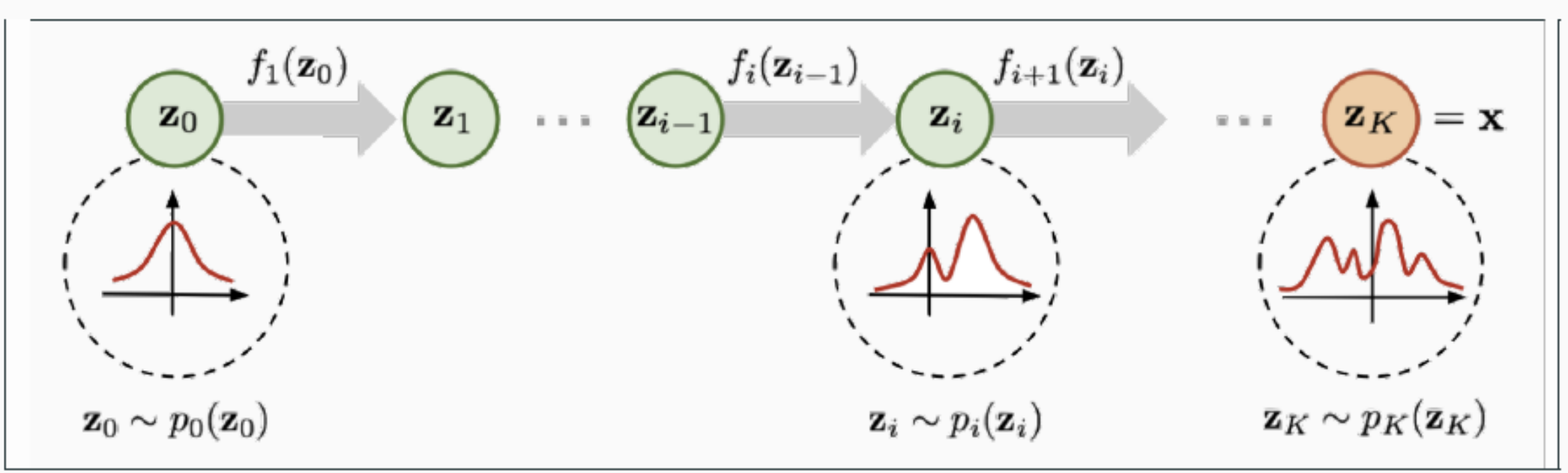
General Goal

- Fit a density model $p_\theta(x)$ for a continuous random variable $X \in \mathbb{R}^D$
  - Good fit to the training data (really, the underlying distribution!)
  - For new x, ability to evaluate $p_\theta(x)$
  - Ability to sample from $p_\theta(x)$
  - And, ideally, a latent representation that's meaningful

High Dimensional Data

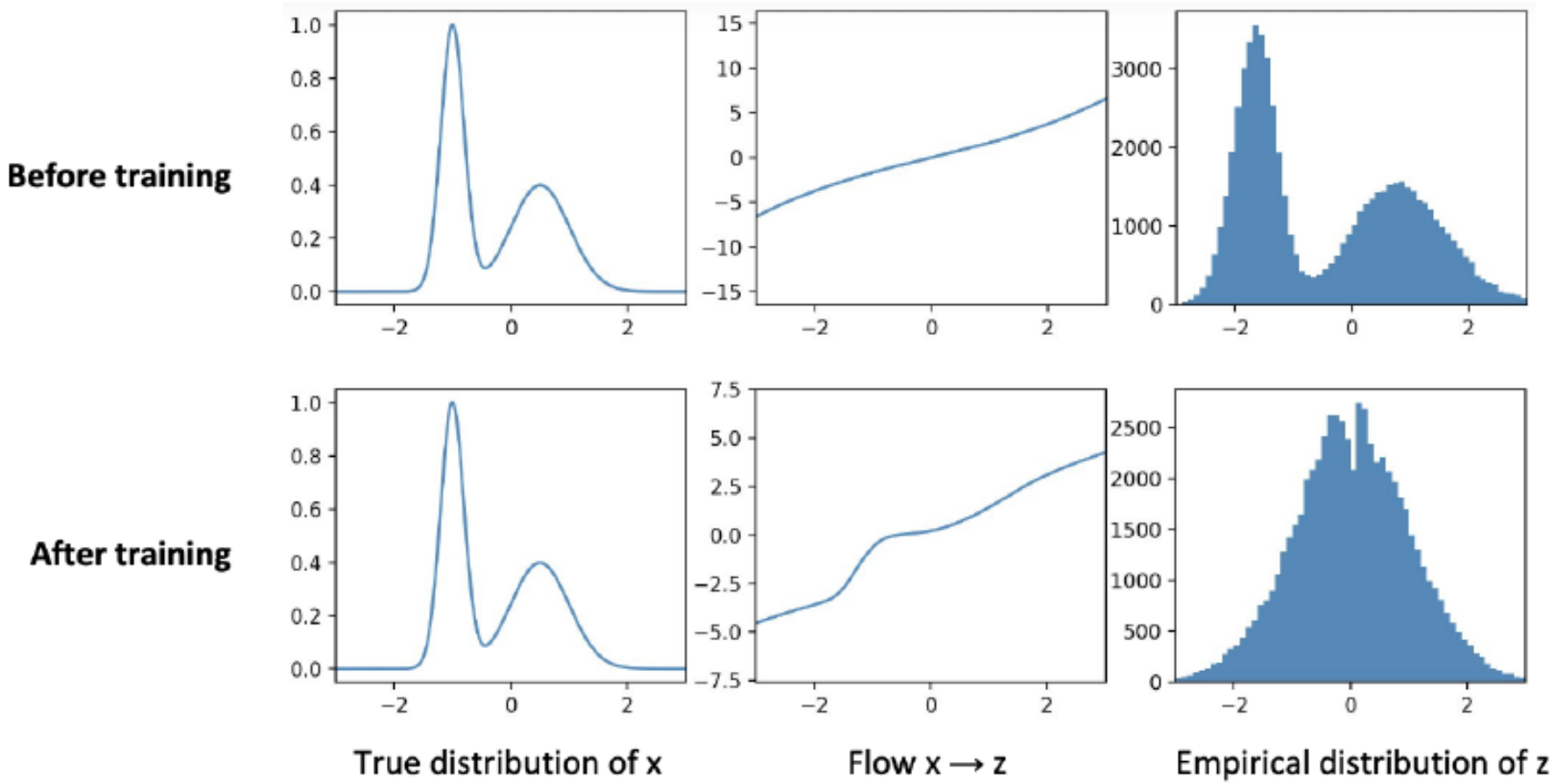


Normalizing Flow

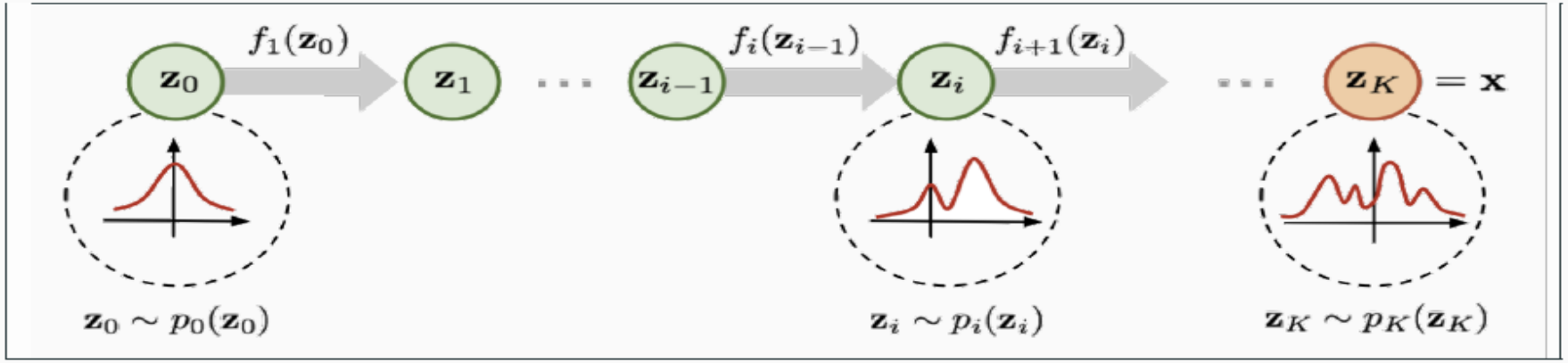


- **Normalizing Flow (NF)**: a flow of invertible transformations that takes $z_0 \sim \mathcal{N}(0, I)$ and outputs x following a complex distribution

A Simple Illustration: X->N(0, 1)



Normalizing Flow



- **A flow-based generative model** takes samples of x and learns $f_1^{-1}, f_2^{-1}, \cdots f_k^{-1}$ such that

$$z_0 = f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_K^{-1}(x)$$

- Once the functions are learned, it uses

$$x = f_k \circ f_{k-1} \circ \cdots \circ f_1(z_0)$$

to approximate the distribution of x

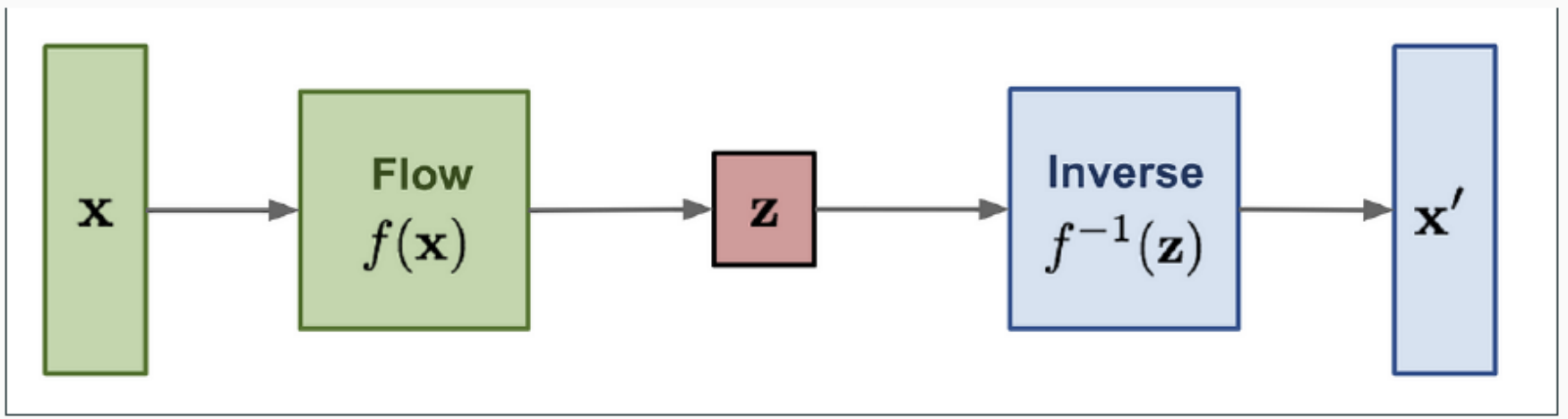
NF – Deriving p(x) Explicitly

- **Change of Variables Theorem**:
  Let $z \sim \pi(z)$ and $x = f(z)$, where f is invertible. Then

$$p(x) = \pi(z) \left| \det \frac{dz}{dx} \right| = \pi(f^{-1}(x)) \left| \det \frac{df^{-1}(x)}{dz} \right|,$$

  where we call the matrix $\frac{df^{-1}(x)}{dx}$ the Jacobian of f^{-1}

NF – Deriving p(x) Explicitly



- Applying the change of variables theorem recursively and taking the log, we obtain the log-likelihood:

$$\log p(x) = \log \pi_0(z_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{dz_{i-1}} \right|$$

NF – Remarks

- With NF architecture, the objective function arises naturally. For samples $x^{(1)}, x^{(2)}, \cdots, x^{(N)}$,

$$\mathcal{L}(x^{(1)}, x^{(2)}, \cdots, x^{(N)}) = -\frac{1}{N} \sum_{i=1}^{N} \log p(x^{(i)})$$

- The key is in finding a suitable class of the transformations f_i.
- Challenges: Ideally, the transformations are
  — capable of growing arbitrarily complex
  —- yet their inverse and Jacobian are easy to compute

RealNVP

- Short for "Real-valued non-volume preserving" model
- Each transformation $f_i : \mathbb{R}^D \rightarrow \mathbb{R}^D$ for some $i \in \{1, 2, \cdots K\}$ used in this model is termed an **affine coupling layer**

RealNVP – affine coupling layers

Say $f(x) = y$ is an affine coupling layer. Then for some $d < D$ and some functions $s : \mathbb{R}^d \to \mathbb{R}^{D-d}$ and $t : \mathbb{R}^d \to \mathbb{R}^{D-d}$,

$$
\begin{aligned}
y_{1:d} &= x_{1:d} \\
y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}).
\end{aligned}
$$

The inverse is

$$
\begin{aligned}
x_{1:d} &= y_{1:d} \\
x_{d+1:D} &= (y_{d+1:D} - t(y_{1:d})) \odot \exp(-s(y_{1:d})).
\end{aligned}
$$

RealNVP – affine coupling layers

- The Jacobian of f is

$$J = \begin{bmatrix} \mathbb{I}_n & 0_{d\times(D-d)} \\ \dfrac{\partial y_{d+1:D}}{\partial x_{1:d}} & diag\left[\exp\left(s(x_{1:d})\right)\right] \end{bmatrix}$$

- Thus

$$det(J) = \prod_{i=1}^{D-d} \exp\left(s(x_{1:d})_i\right) = \exp\left(\sum_{i=1}^{D-d} s(x_{1:d})_i\right)$$

- Remark: s and t does not contribute to the complexity of the inverse or the Jacobian matrix
  - s and t can be arbitrarily complex – deep neural nets

RealNVP – important of good masking

- Each affine flow involves partitioning the elements of the input vector into two groups
  - Termed *masking* by the original paper

- Let b be a *binary mask*, a vector with d ones and $D - d$ zeros. Then we can rewrite y as

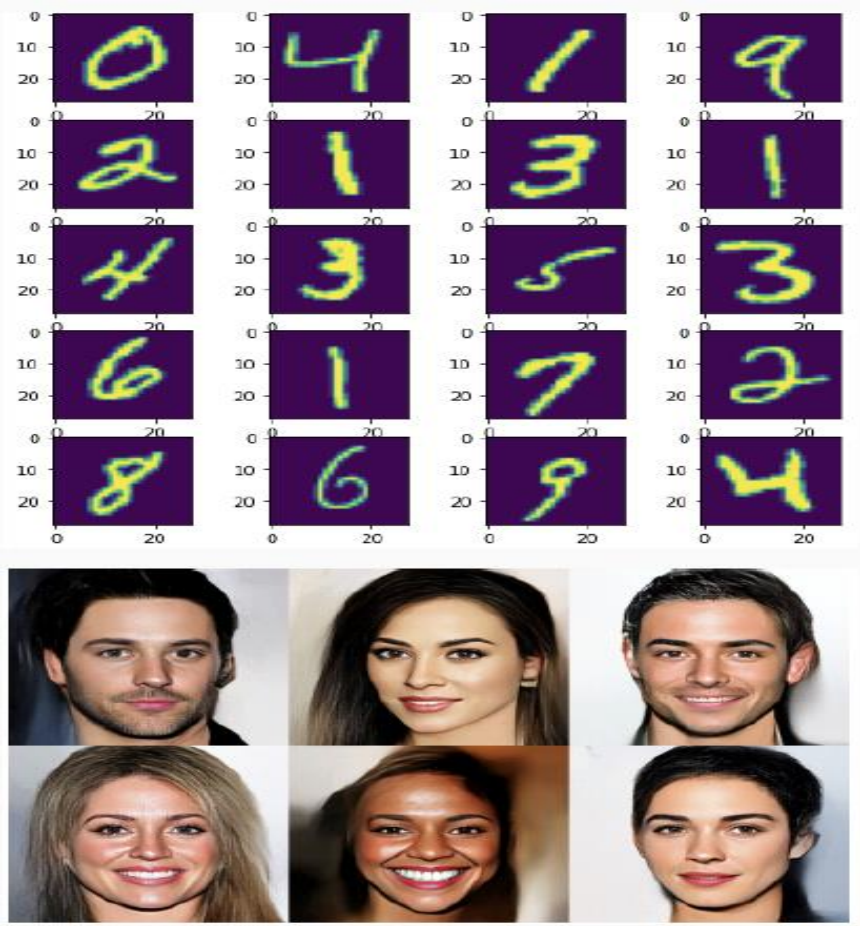
$$y = b \odot x + (1 - b) \odot (x \odot \exp(s(b \odot x)) + t(b \odot x))$$

RealNVP – important of good masking

Checkerboard x4; channel squeeze; channel x3; channel unsqueeze; checkerboard x3

(Mask top half; mask bottom half; mask left half; mask right half) x2

Generative Images

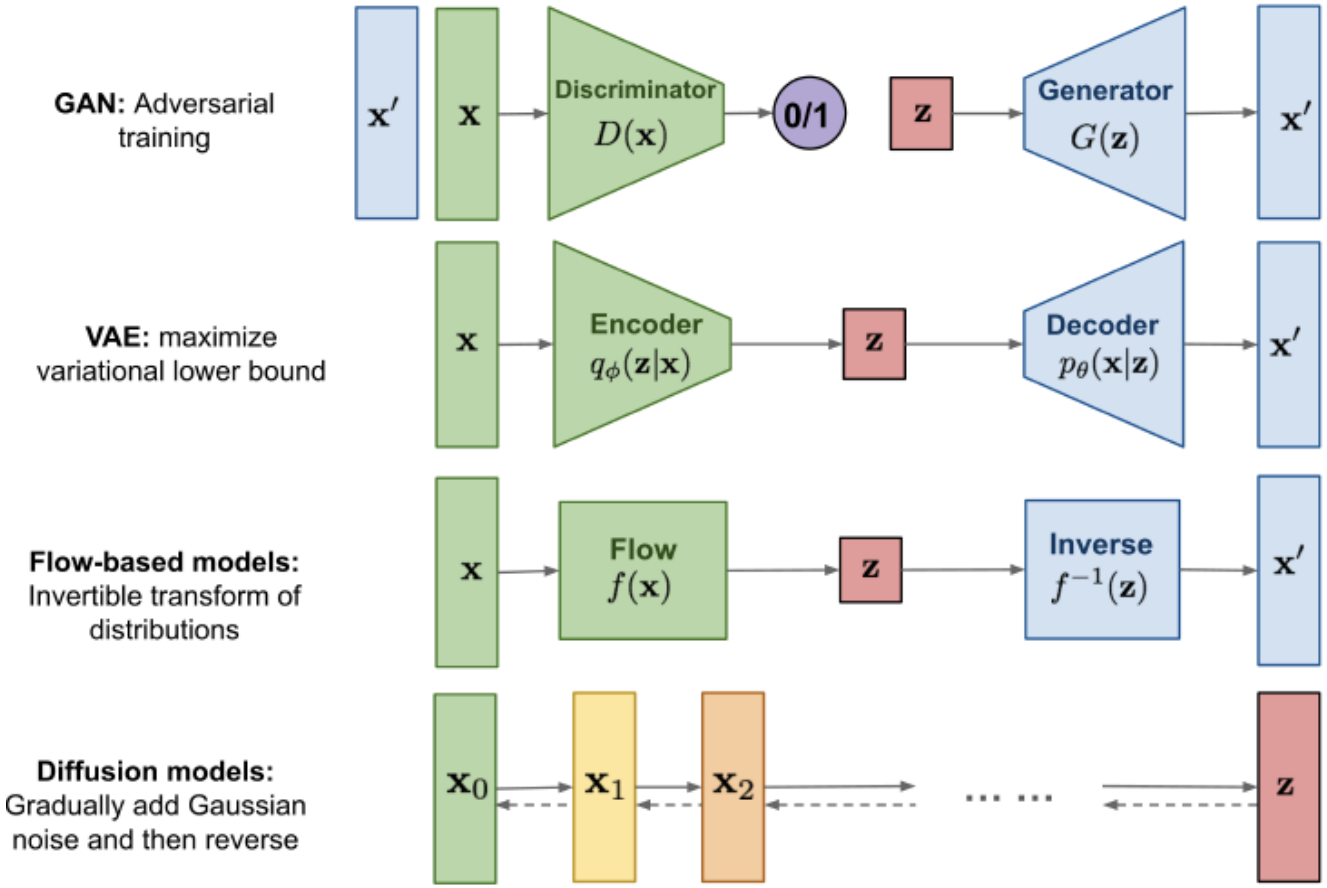


Flow-Based Models Summary

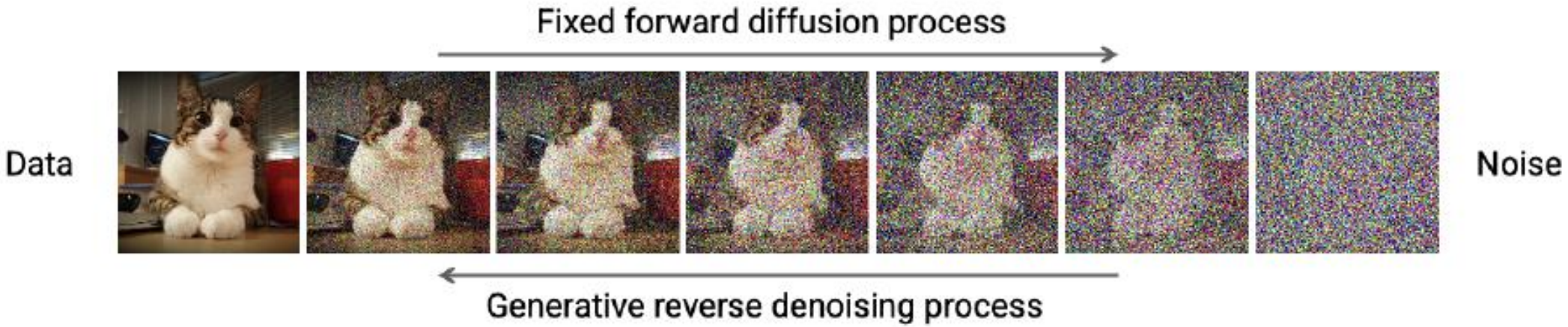
- The ultimate goal: a likelihood-based model with
  - fast sampling
  - fast inference
  - fast training
  - good samples
  - good compression
- Flows seem to let us achieve some of these criteria.
- But how exactly do we design and compose flows for great performance? That is an open question.

Content

GAN, VAE, Flow, and DM



Denoising Diffusion Models

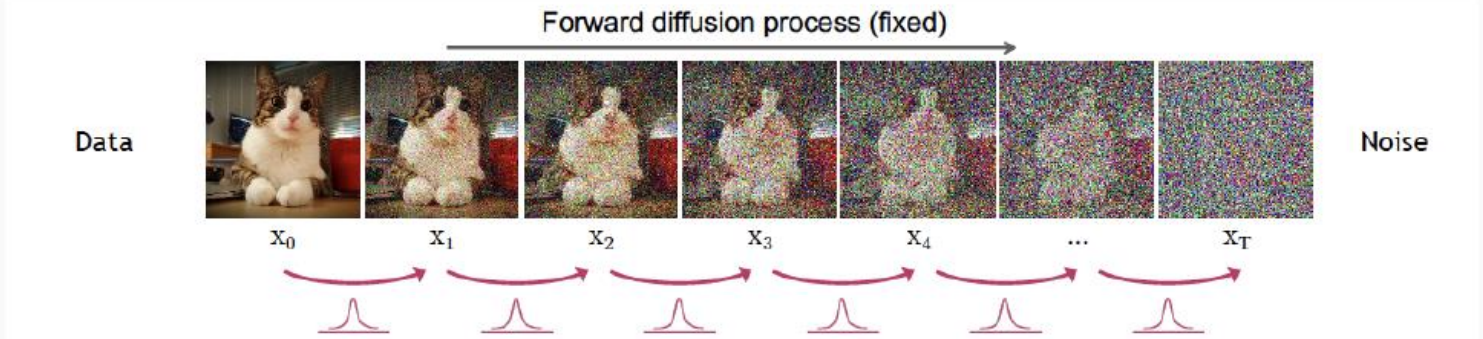


Forward Diffusion Process

- The latent dimension is exactly equal to the data dimension
- The structure of the latent encoder at each timestep is not learned; it is pre-defined as a linear Gaussian model. In other words, it is a Gaussian distribution centered around the output of the previous timestep
- The Gaussian parameters of the latent encoders vary over time in such a way that the distribution of the latent at final timestep T is a standard Gaussian

Forward Diffusion Process

The formal definition of the forward process in T steps:

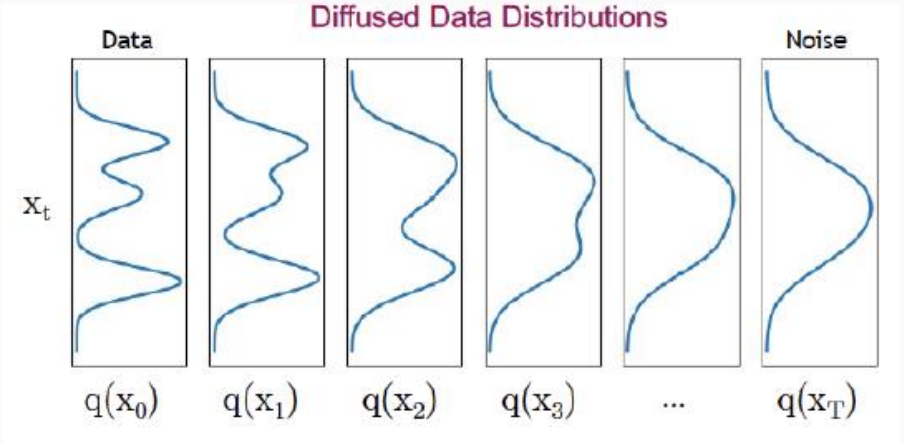


$$x_t \Big| x_{t-1} \sim N(\sqrt{1-\beta_t}\, x_{t-1}, \beta_t I)$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

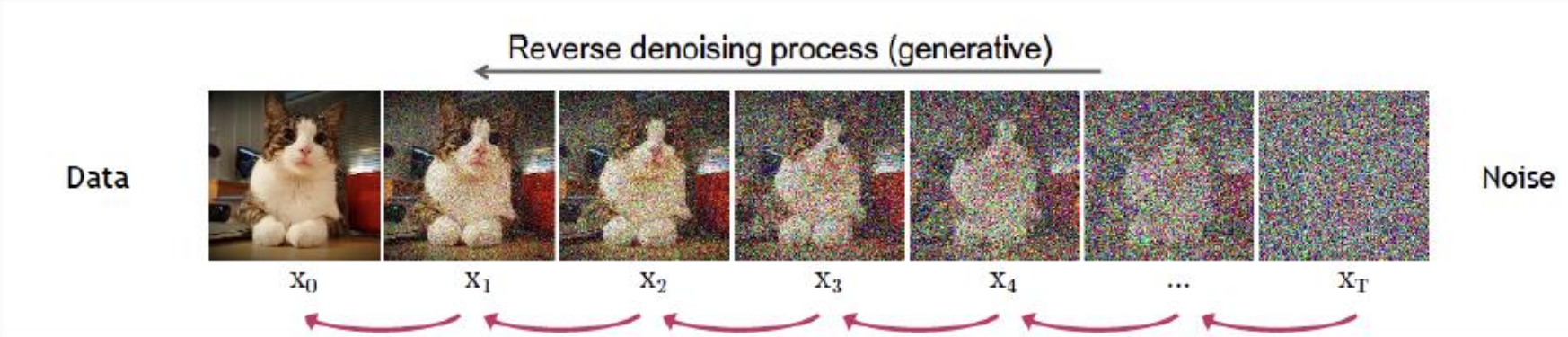
Diffusion Kernel

- Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$
- $x_t|x_0 \sim N(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)I)$
- For sampling: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon$, where $\epsilon \sim N(0, I)$
- β_t is designed such that $\bar{\alpha}_T \to 0$ and $x_T|x_0 \dot{\sim} N(0, I)$
- Marginally, $q(x_t) = \int q(x_0)q(x_t|x_0)dx_0$



Reverse Denoising Process

Formal definition of reverse processes in T steps:



$$x_T \sim N(0, I), \quad x_{t-1}|x_t \sim N(\mu_\theta(x_t, t), \sigma_t^2 I)$$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Learning Denoising Model

- For training, we can form the ELBO

$$\mathbb{E}_{q(x_0)}[-\log p_\theta(x_0)] \leq \mathbb{E}_{q(x_0)q(x_{1:T}|x_0)}[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}] := L$$

- It is shown (Sohl-Dickstein et al. 2015 and Ho et al. 2020)that

$$\begin{aligned} L &= \mathbb{E}_q\Big[-\log p_\theta(x_0|x_1) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) \\ &\qquad + D_{KL}(q(x_T|x_0)\|p(x_T))\Big] \\ &:= L_0 + \sum_{t>1} L_{t-1} + L_T \end{aligned}$$

where

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

and

$$x_{t-1}|x_t, x_0 \sim N(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t, \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

Learning Denoising Model

- $\mathbb{E}_{q_{x_1|x_0}}[-\log p_\theta(x_0|x_1)]$ can be interpreted as a reconstruction term; like its analogue in the ELBO of a vanilla VAE, this term can be approximated and optimized using a Monte Carlo estimate.
- $D_{KL}(q(x_T|x_0)\|p(x_T))$ represents how close the distribution of the final noisified input is to the standard Gaussian prior. It has no trainable parameters, and is also equal to zero under our assumptions.
- $\mathbb{E}_{q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t))]$ is a denoising matching term. We learn desired denoising transition step $p_\theta(x_{t-1}|x_t)$ as an approximation to tractable, ground-truth denoising transition step $q(x_{t-1}|x_t, x_0)$.

Reparameterizing the Denoising Model

- Due to the normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right] + C$$

- Ho et al. (2020) observe that:

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1-\beta_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$$

- Ho et al. (2020) propose to represent the mean of the denoising model using a noise-prediction network:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$$

Reparameterizing the Denoising Model

- With this reparametrization

$$L_{t-1} = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim N(0,I)} \left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(x_t, t)\|^2 \right] + C$$

  where $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$

- Ho et al. (2020) observe that simply setting the time dependent coefficient being 1 improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim N(0,I), t \sim Unif[1,T]} \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t\right) \right\|^2$$

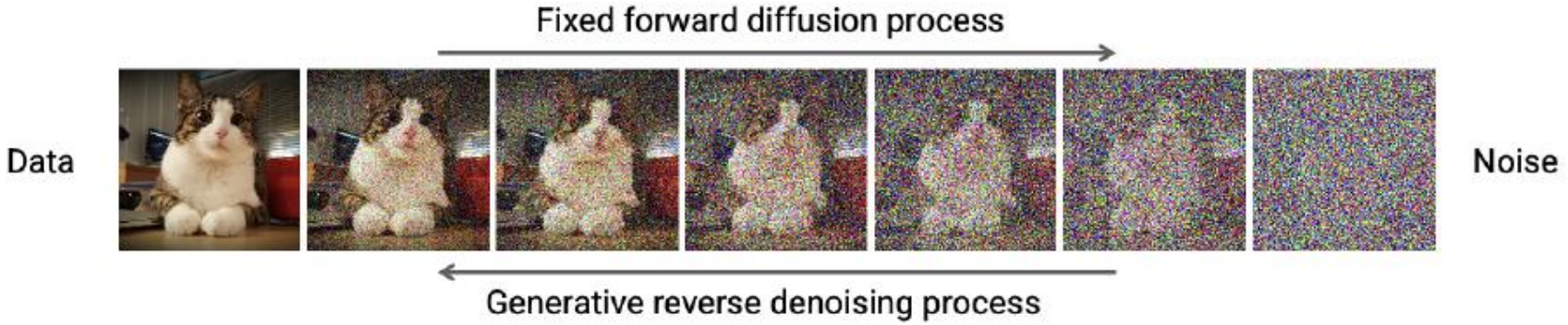
Algorithm

Algorithm 1 Training

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

Algorithm 2 Sampling

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



Open Problems

- Sampling from diffusion model is still slow — How can one sample with even fewer steps?
- How good is the latent space of diffusion model for downstream tasks?
  — ResNet on ImageNet gives us great image features
  — LLM gives great text features
  — Can diffusion model beat imagenet feature?
  — Can diffusion model help us in discriminative tasks?

References

Aggarwal, A., Mittal, M., & Battineni, G. (2021). Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, *1*(1), 100004.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In International Conference on Machine Learning.

Brophy, E., Wang, Z., She, Q., & Ward, T. (2023). Generative adversarial networks in time series: A systematic literature review. *ACM Computing Surveys*, *55*(10), 1-31.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE signal processing magazine*, *35*(1), 53-65.

Loaiza-Ganem, Gabriel, Brendan Leigh Ross, Rasa Hosseinzadeh, Anthony L. Caterini, and Jesse C. Cresswell. "Deep generative models through the lens of the manifold hypothesis: A survey and new connections." *arXiv preprint arXiv:2404.02954* (2024).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems, pages 2672{2680.

Gui, J., Sun, Z., Wen, Y., Tao, D., & Ye, J. (2021). A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE transactions on knowledge and data engineering*, *35*(4), 3313-3332.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. arXiv:1704.00028.

Kingma, D.P. and Welling, M. (2019). An Introduction to variational autoencoders. Foundations and Trends® in Machine Learning 12 (4), 307-392

Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784* (2014).

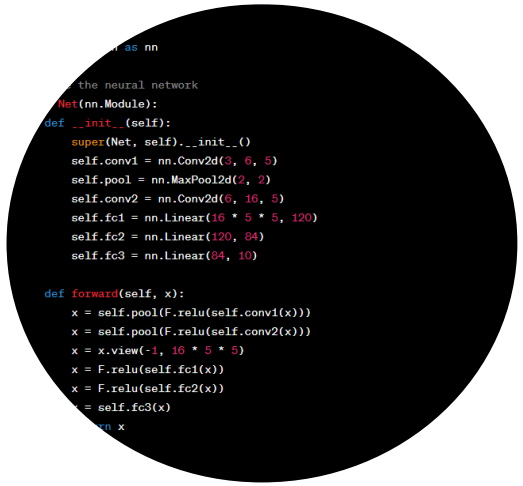
Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., & Zheng, Y. (2019). Recent progress on generative adversarial networks (GANs): A survey. *IEEE access*, *7*, 36322-36333.

Yi, Xin, Ekta Walia, and Paul Babyn. "Generative adversarial network in medical imaging: A review." *Medical image analysis* 58 (2019): 101552.

Xu, Lei, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. "Modeling tabular data using conditional gan." *Advances in neural information processing systems* 32 (2019).

Zhou, X., Jiao, Y., Liu, J., and Huang, J. (2023). A deep generative learning approach to conditional sampling. Journal of the. American Statistical Association, 118(543):1837-1848.

How to succeed in this course?



Practice

Explore

Visualize

Discuss

Ask