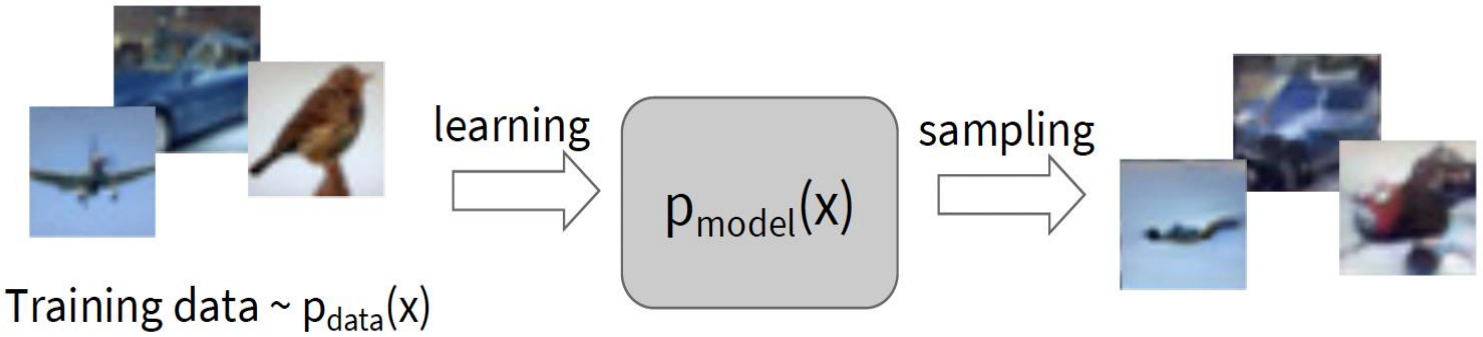
What are Generative Models?

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).

Explicitly Density Models

A simple form of generative learning is to learn a deterministic function

- Generate $\eta \sim N(\mathbf{0}, \mathbf{I}_m), m \geq 1$.
- Estimate a generator function G such that

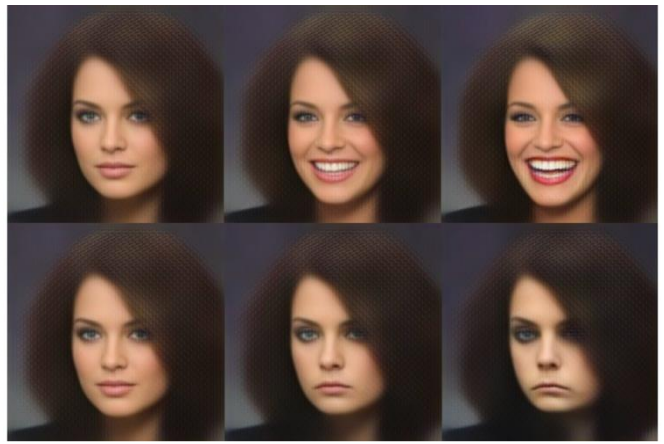
$$G(\eta) \sim P_X.$$

- Let $X = G_\theta(Z)$ with $Z \sim p_Z$. The distribution function of X is

$$P(X \leq \boldsymbol{x}) = P(G_\theta(Z) \leq \boldsymbol{x}) = \int_{G_\theta(\boldsymbol{z}) \leq \boldsymbol{x}} p_Z(\boldsymbol{z}) d\boldsymbol{z}$$

with density function

$$p_{G_\theta}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}} \int_{G_\theta(\boldsymbol{z}) \leq x} p_Z(\boldsymbol{z}) dz.$$



If $G_\theta(\boldsymbol{z})$ is invertible, then we have

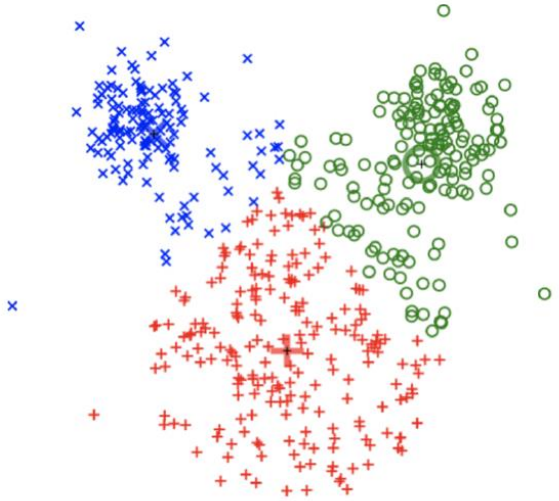
$$p_{G_\theta}(\boldsymbol{x}) = \pi(G_\theta^{-1}(\boldsymbol{x}))|\det(\nabla_{\boldsymbol{x}} G_\theta^{-1}(\boldsymbol{x}))|.$$

How?

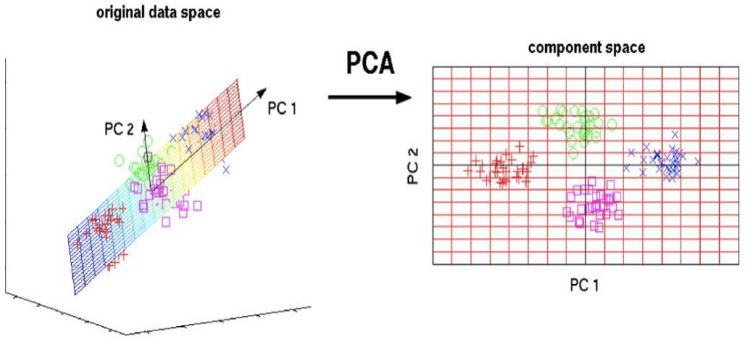
It is still difficult to calculate the inverse G_θ^{-1}.

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)**



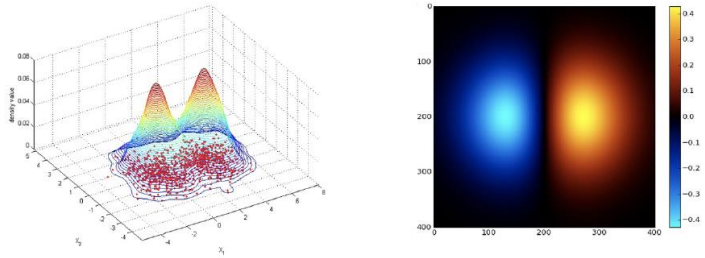
K-means clustering



PCA

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation

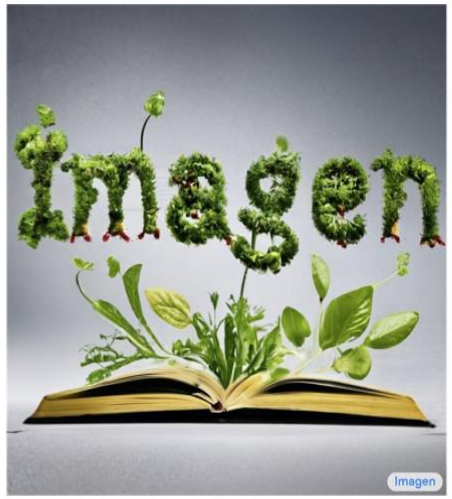


2-d density estimation

Modeling p(x)

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

Emerging Generative Models in 2022-



alphacode
it solutions

Parti
Pathways Autoregressive Text-to-Image Model

DESIGNS.AI

DALL·E 2

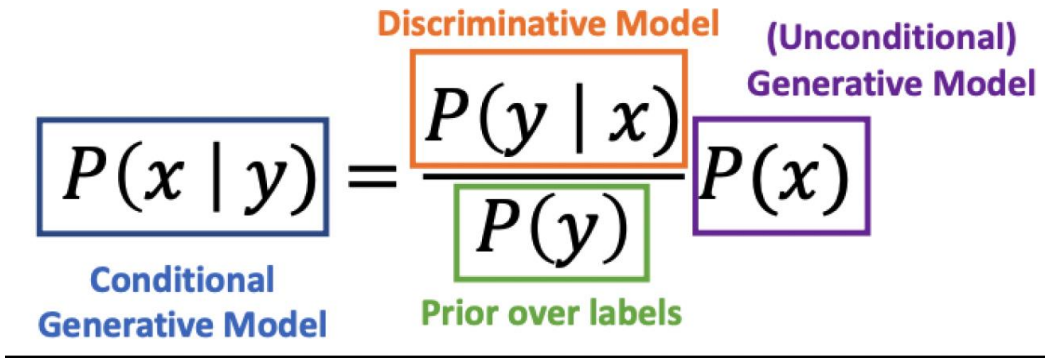
Stable Diffusion

What are Conditional Generative Models?

Definition: A **conditional generative model** is a type of generative model that generates new data samples based on additional context or conditioning information. Mathematically, it models the conditional probability distribution,

Applications: Image-to-Image translation, Text-to-Image generation, Super-Resolution, etc.

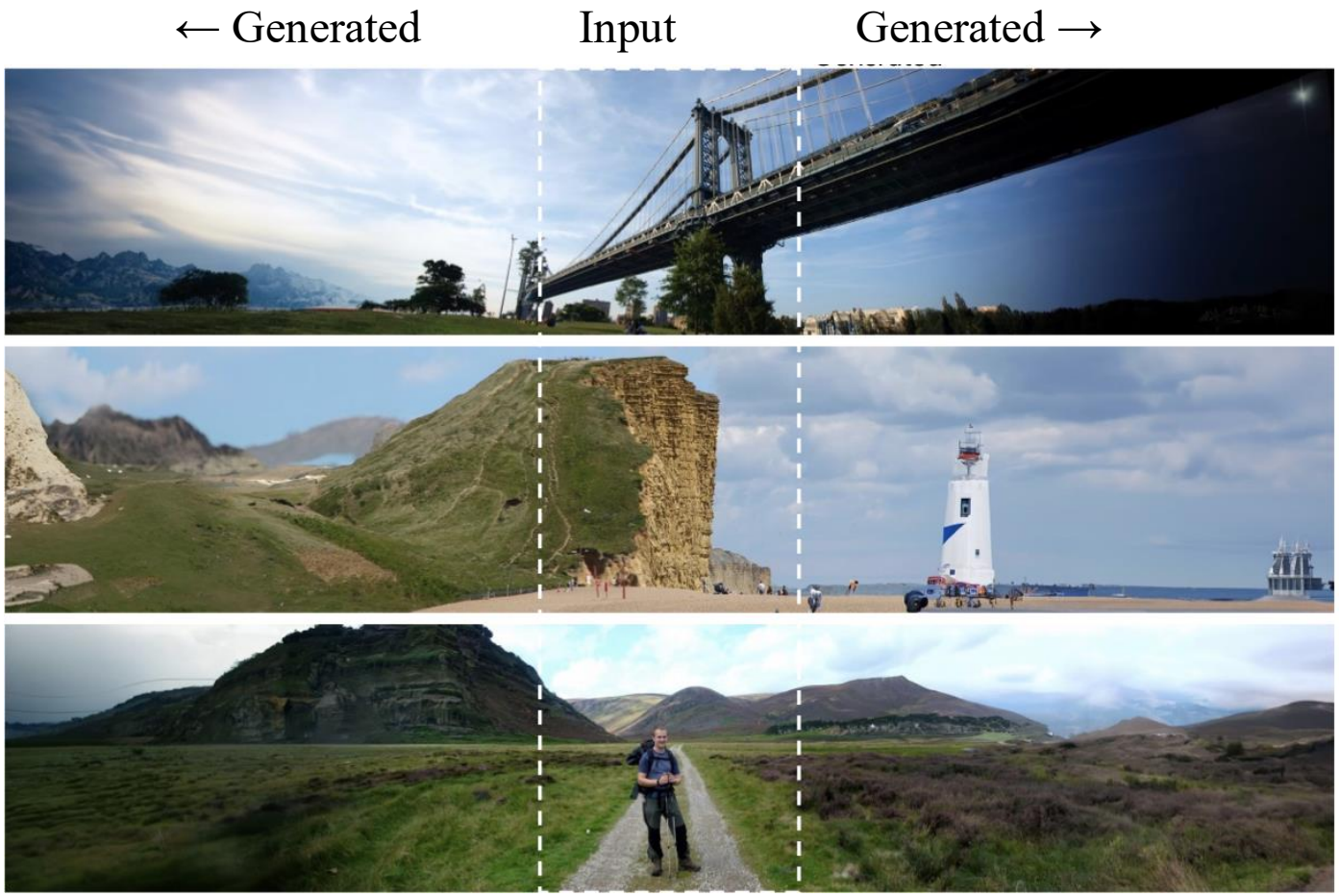
$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$



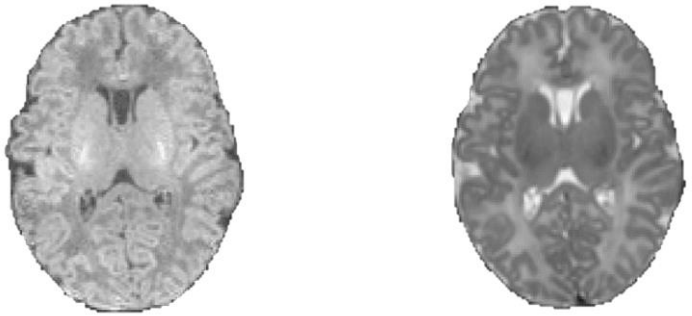
We can build a conditional generative model from other components!

Why Conditional Generative Models?

Outfilling/Missing Data Imputation



T1w<---------→T2w



- **Infant brain MR images (T1w/T2w)**
  - **Low** tissue contrast and **dynamic** change in appearance

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 $[0, 1]^m$.

Common DGMs:

- Deep Belief Networks (DBNs)
- Deep Boltzmann Machines (DBMs)
- Denoising Autoencoders (DAEs)
- Generative Stochastic Networks (GSNs)
- PixelRNN/PixelCNN
- Generative Adversarial Network (GANs)
- Variational Autoencoder

Taxonomy of Deep Generative Models

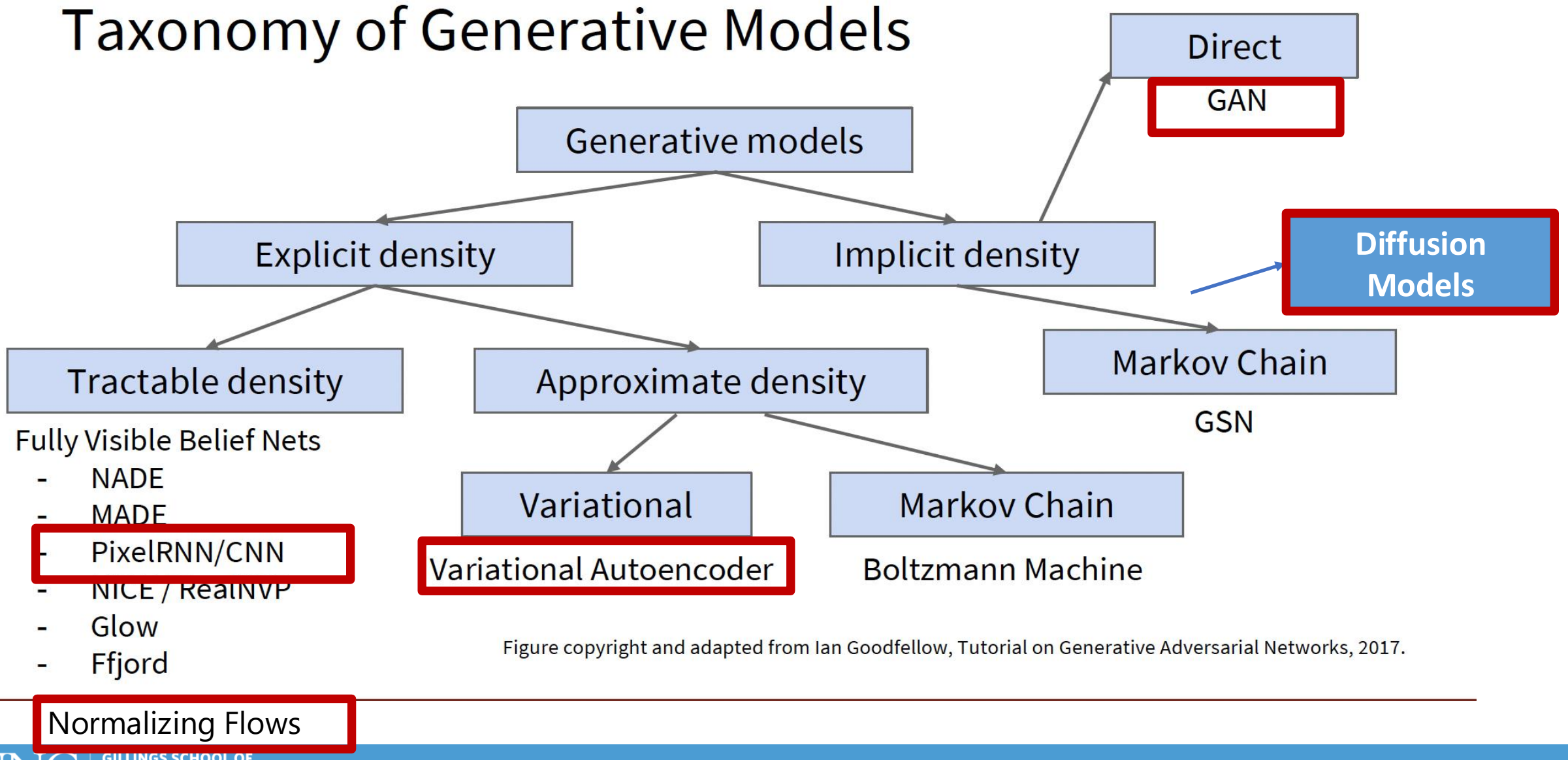
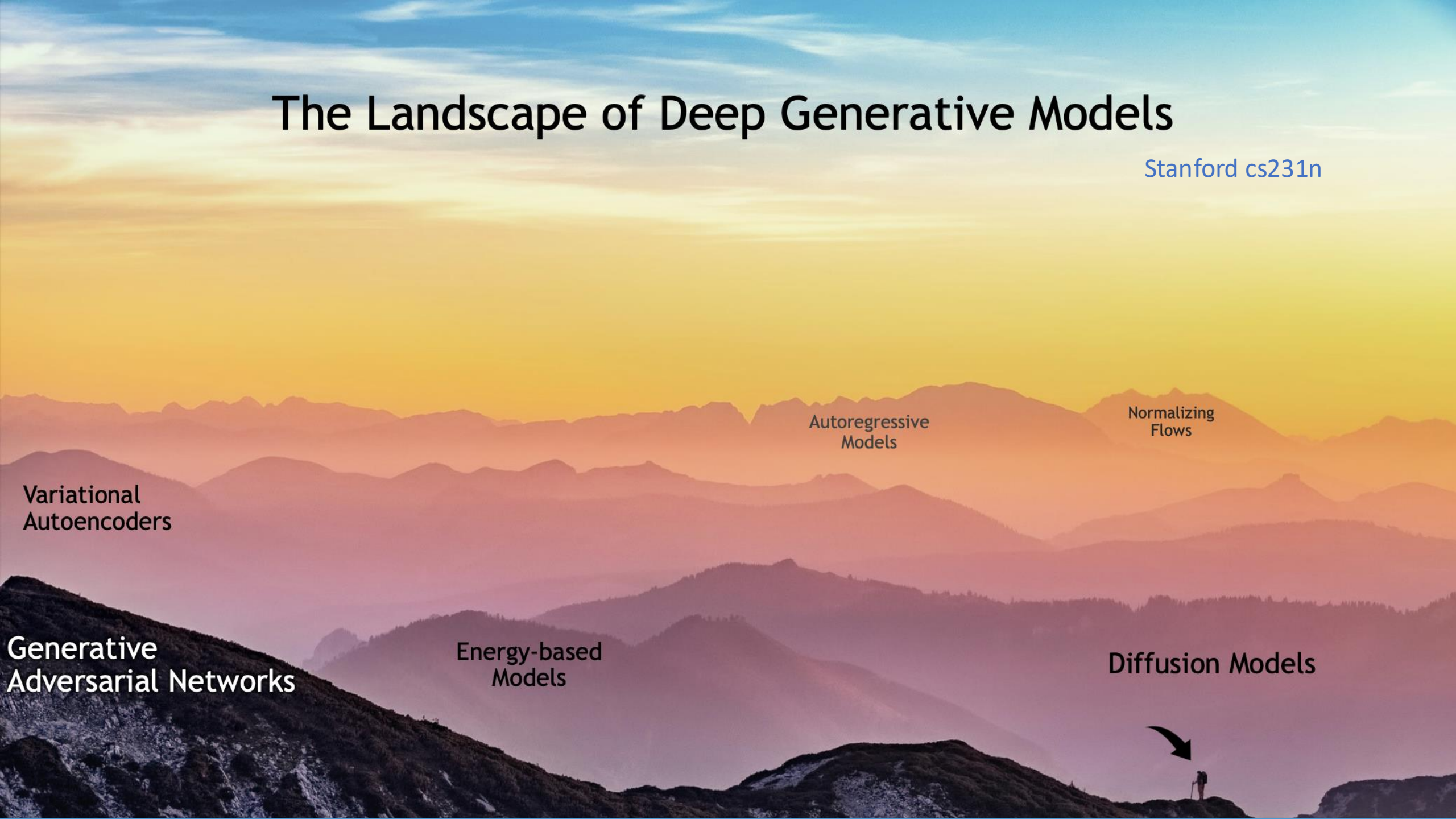


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.


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

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 $\hat{x} = d(z) = d(e(x))$.

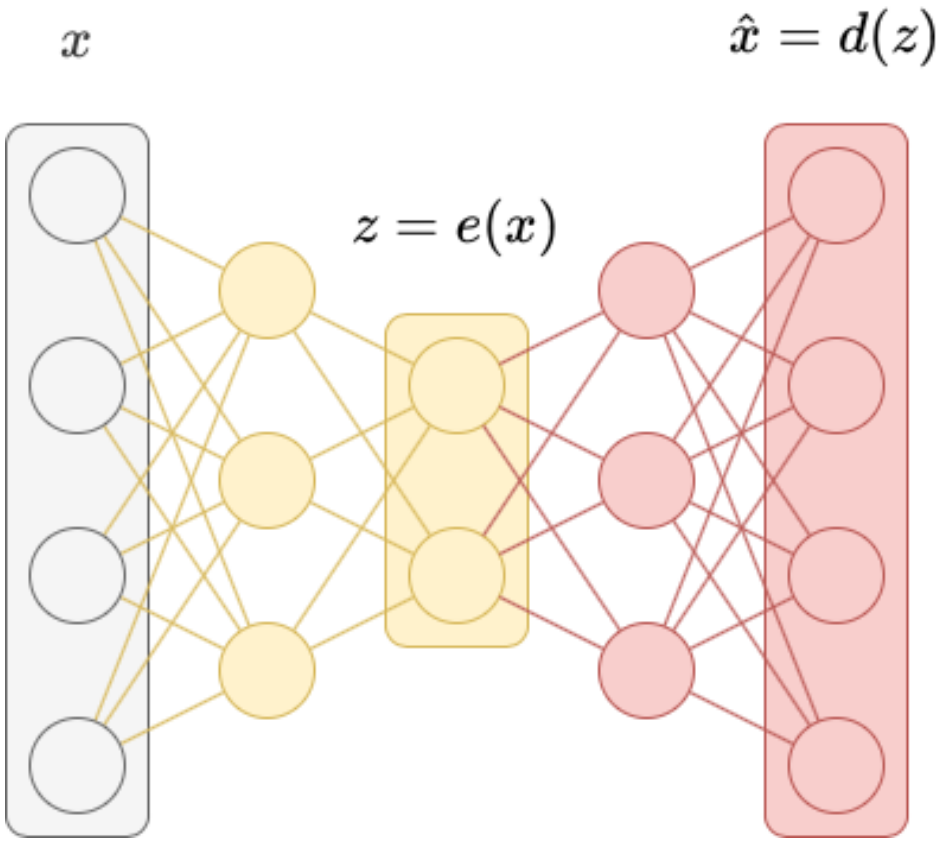
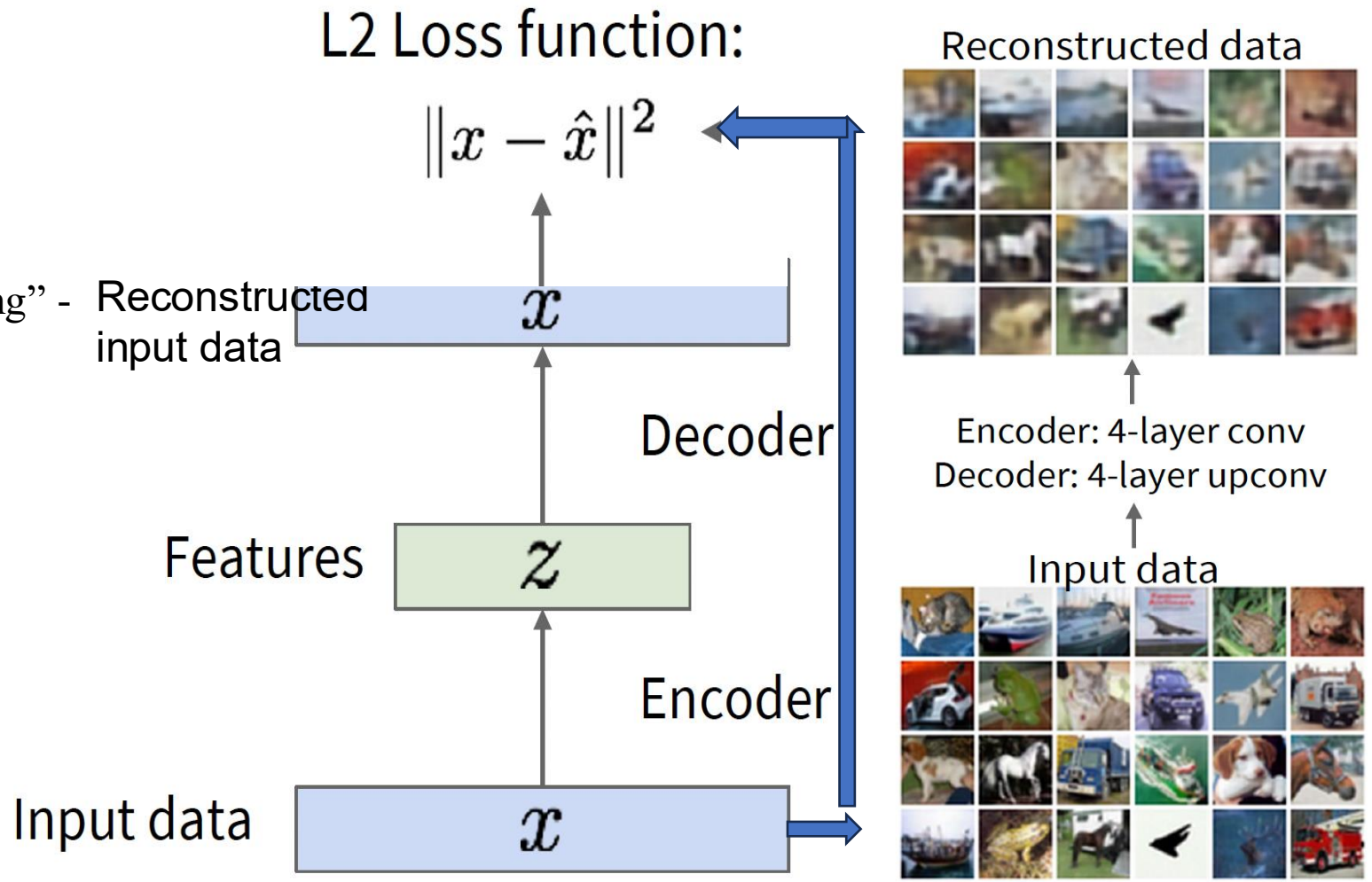


Illustration of autoencoder (source)

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.

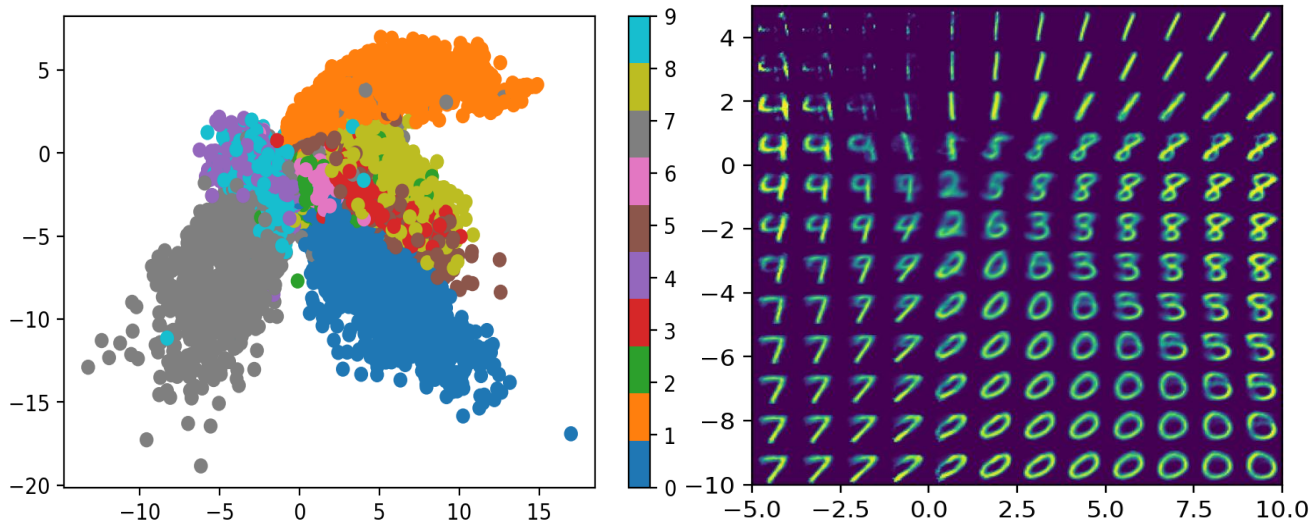


Illustration of example latent vectors using the MNIST dataset (source)

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! 🚀

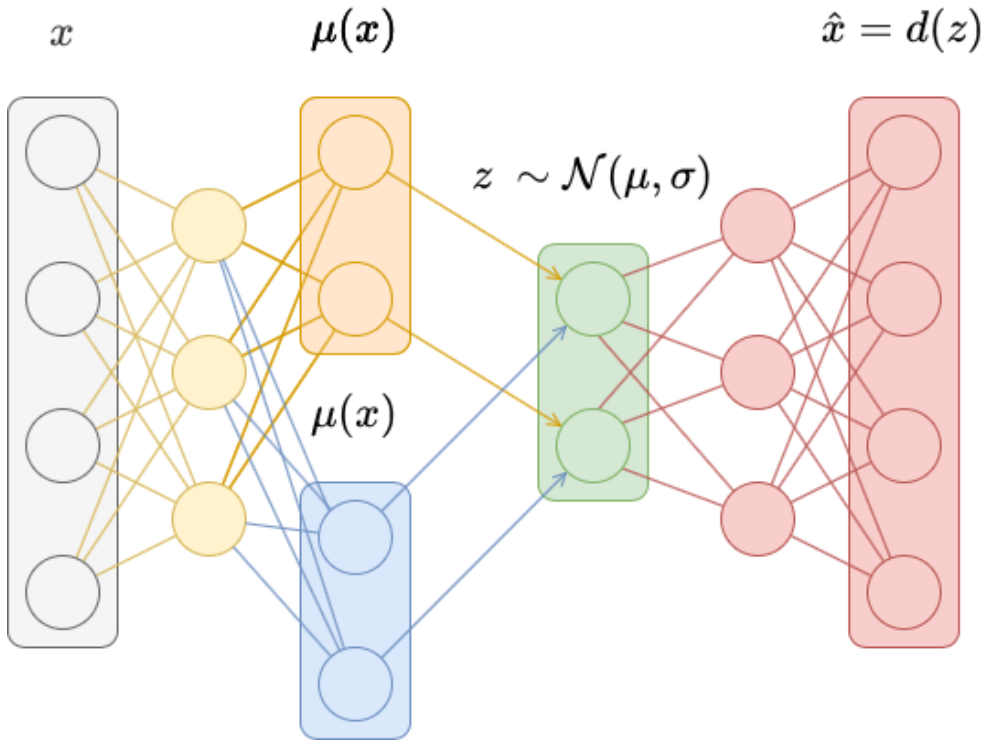


Illustration of VAE (source)

Variational Autoencoder as a DGM

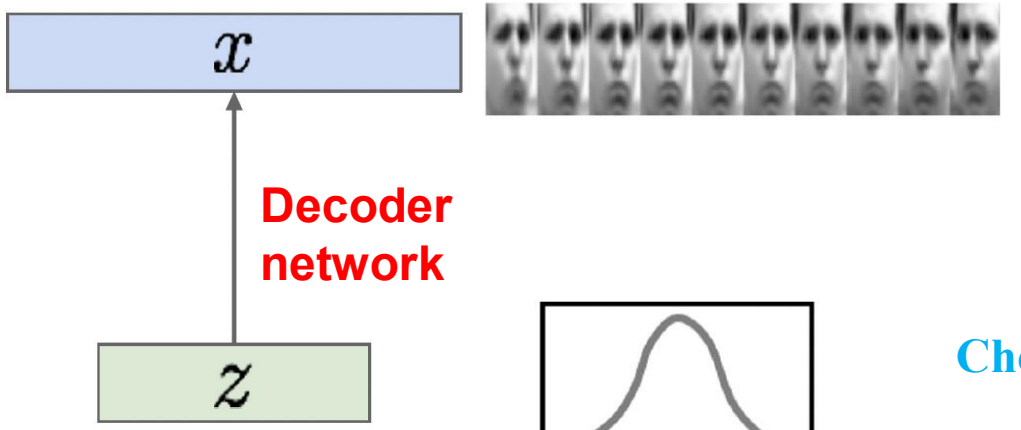
VAEs define an intractable density function with latent

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

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 VGE?

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 VGE?

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

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

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

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

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

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 VGE?

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

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

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

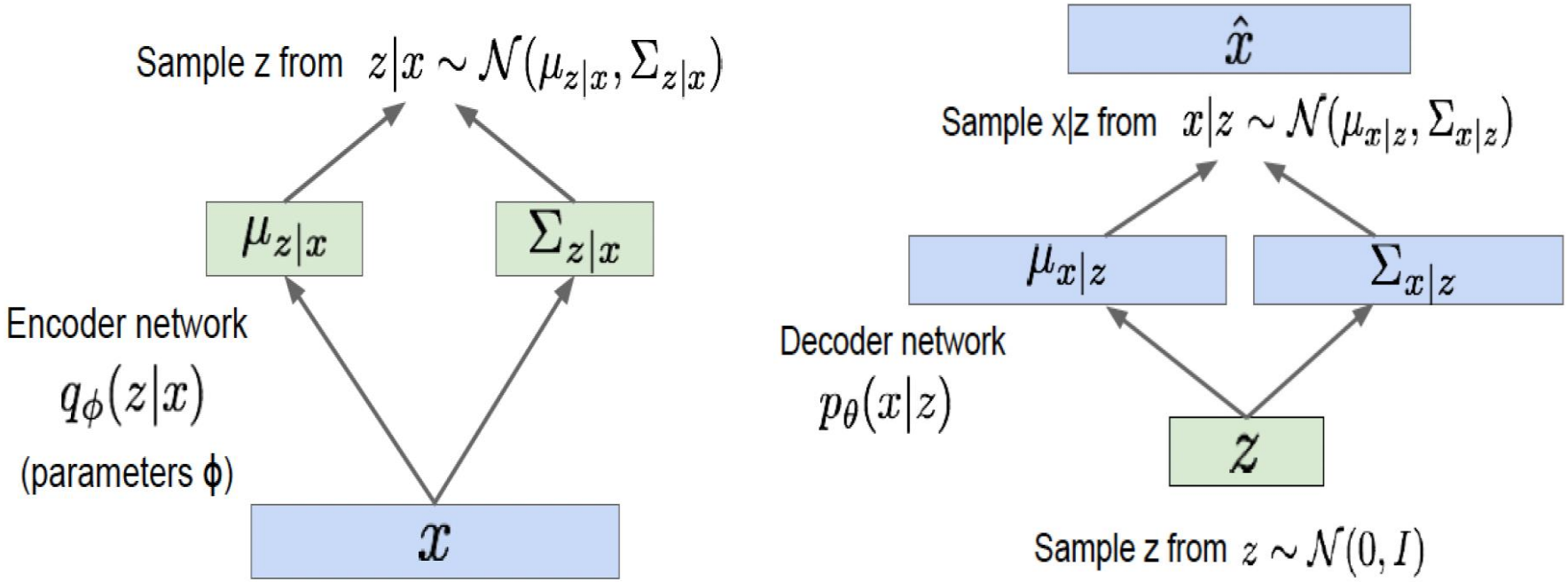
$$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right]$$

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



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

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

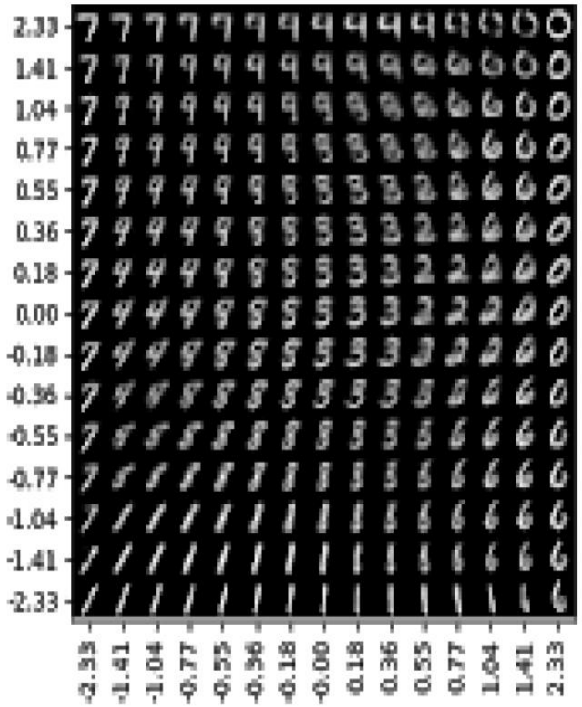
$$\log p(\mathbf{x}|\mathbf{z}) = \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I})$$

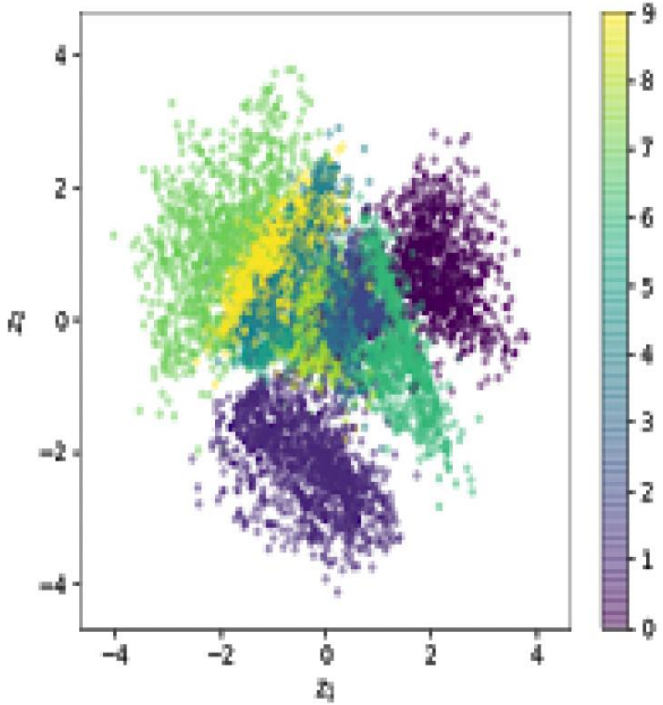
$$\text{where } \boldsymbol{\mu} = \mathbf{W}_4 \mathbf{h} + \mathbf{b}_4$$

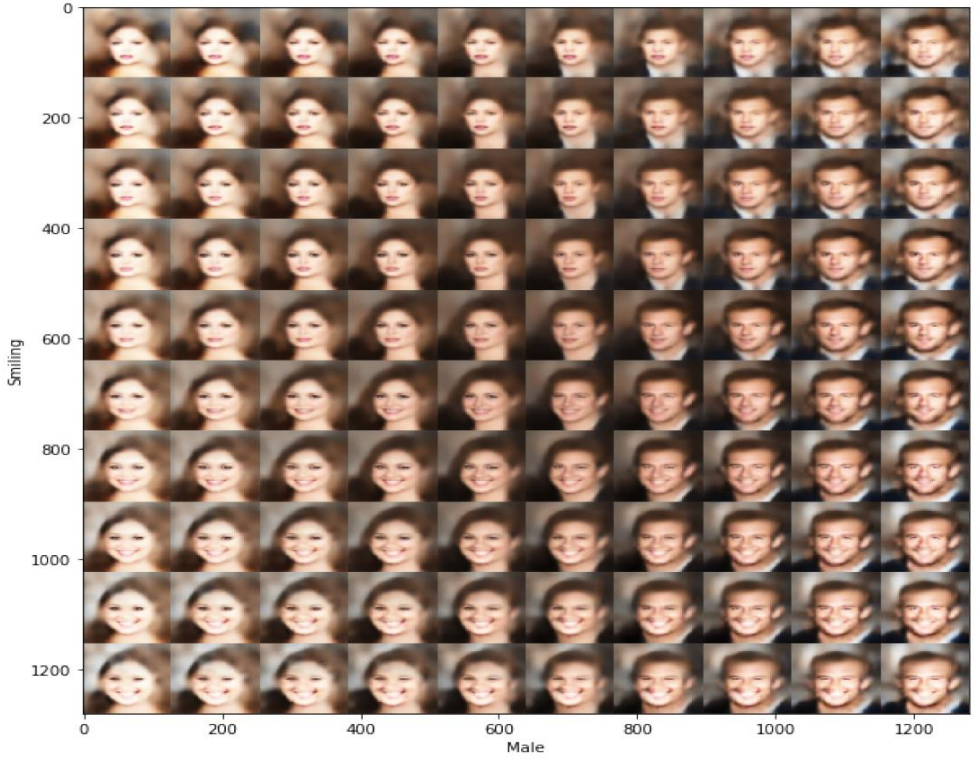
$$\log \boldsymbol{\sigma}^2 = \mathbf{W}_5 \mathbf{h} + \mathbf{b}_5$$

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

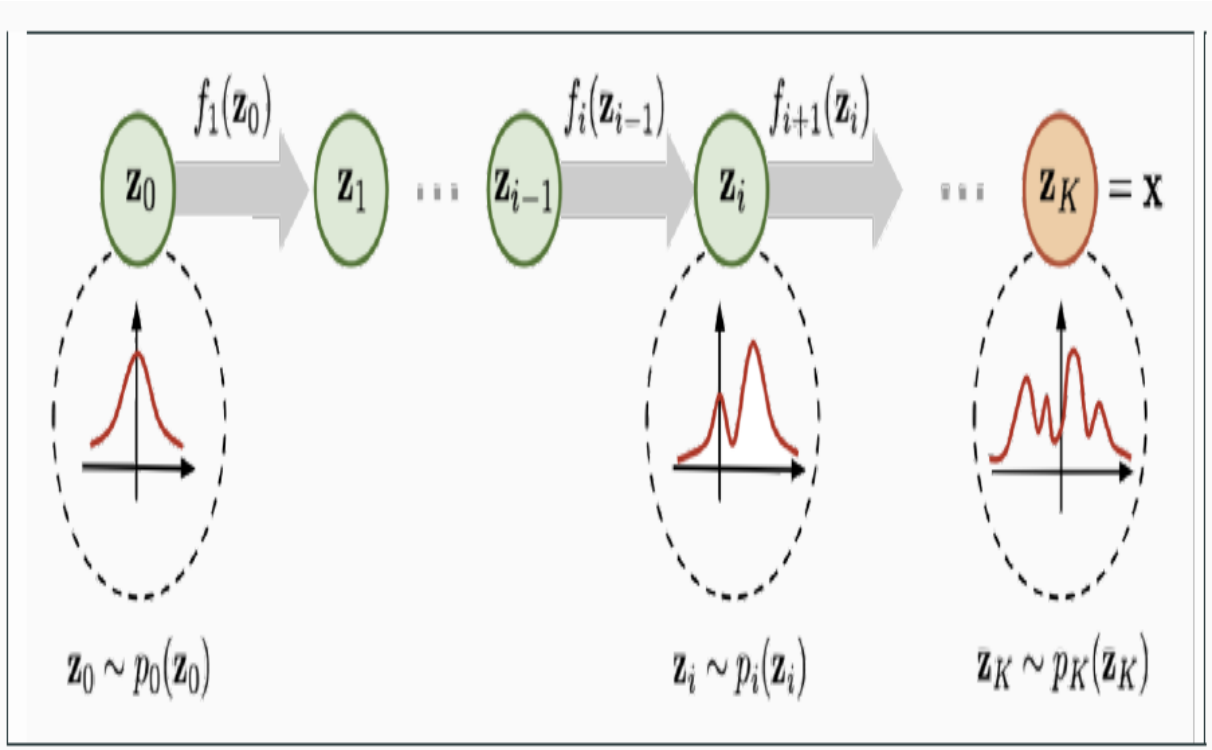
Real Examples

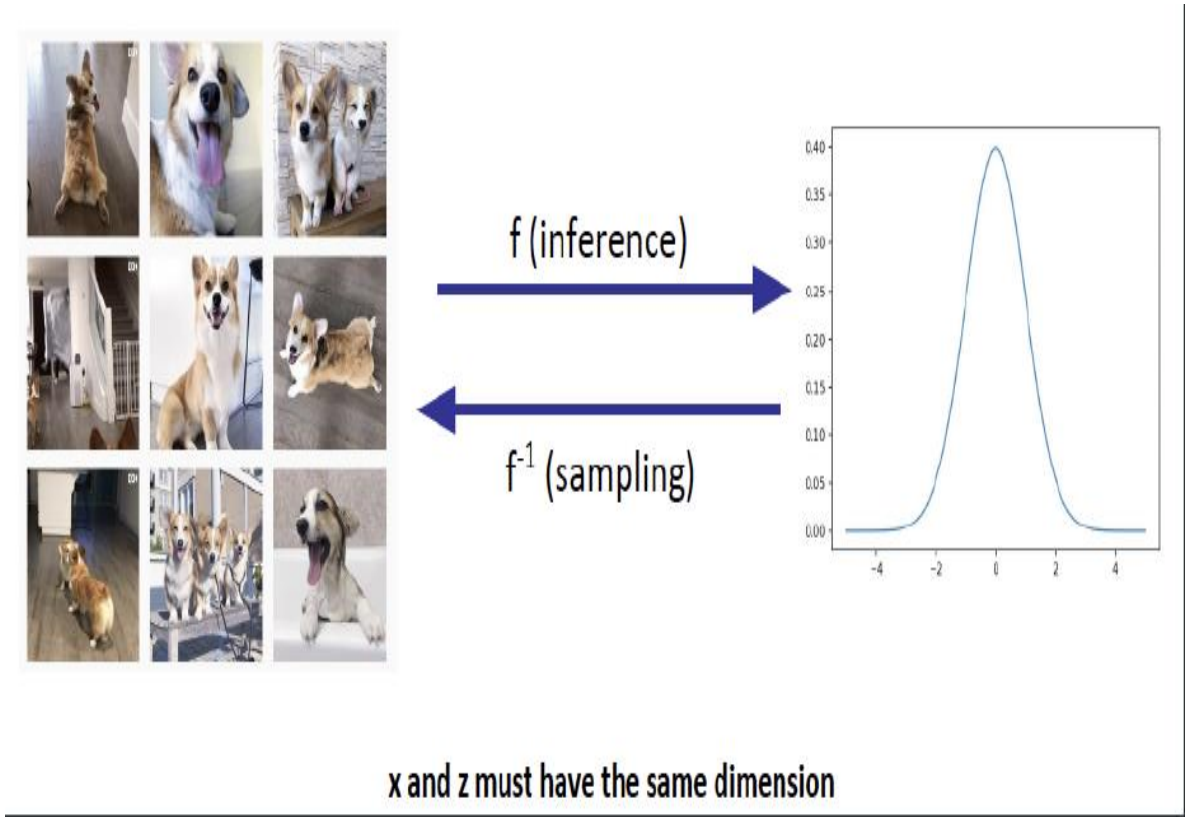






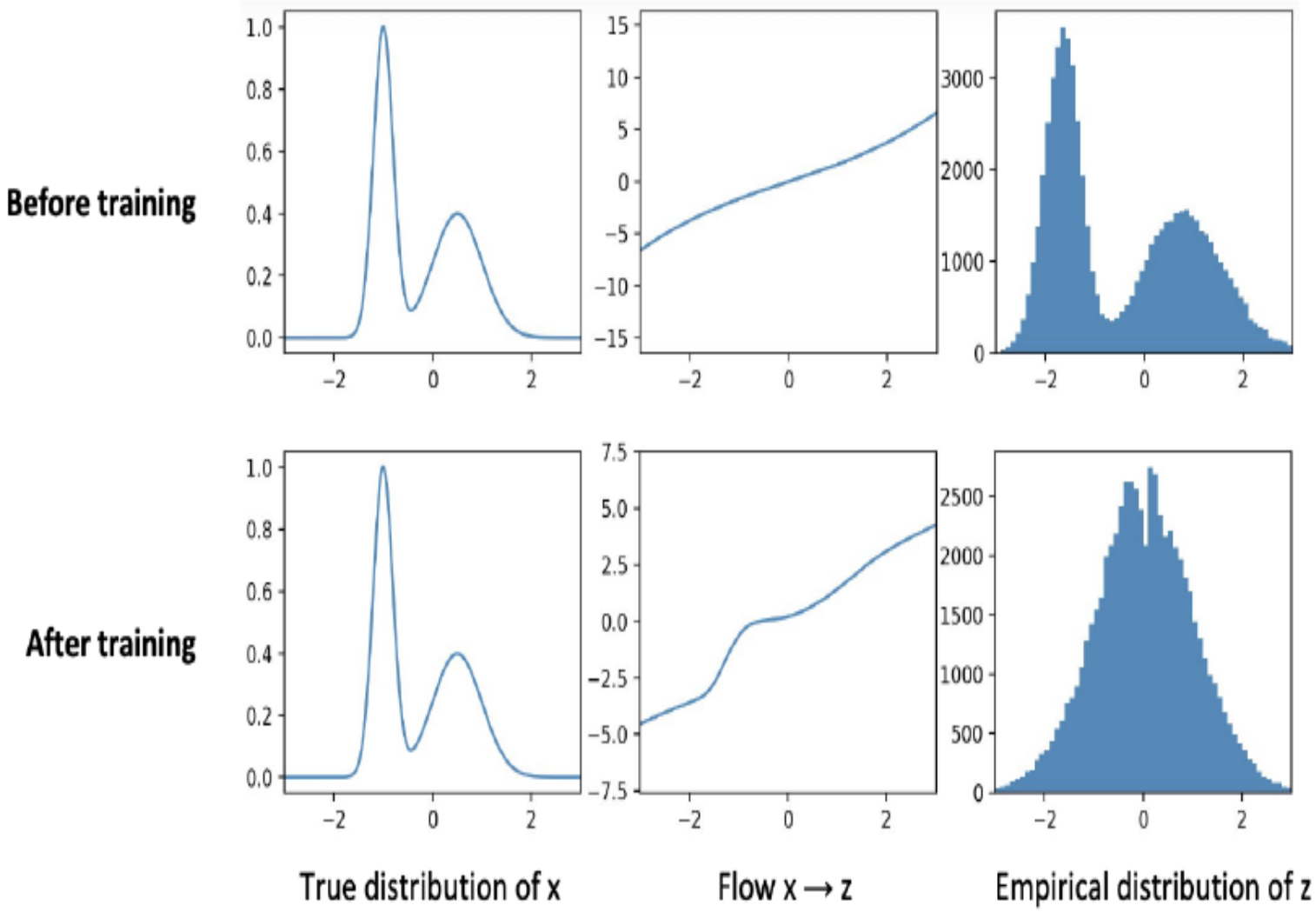
Normalizing Flow





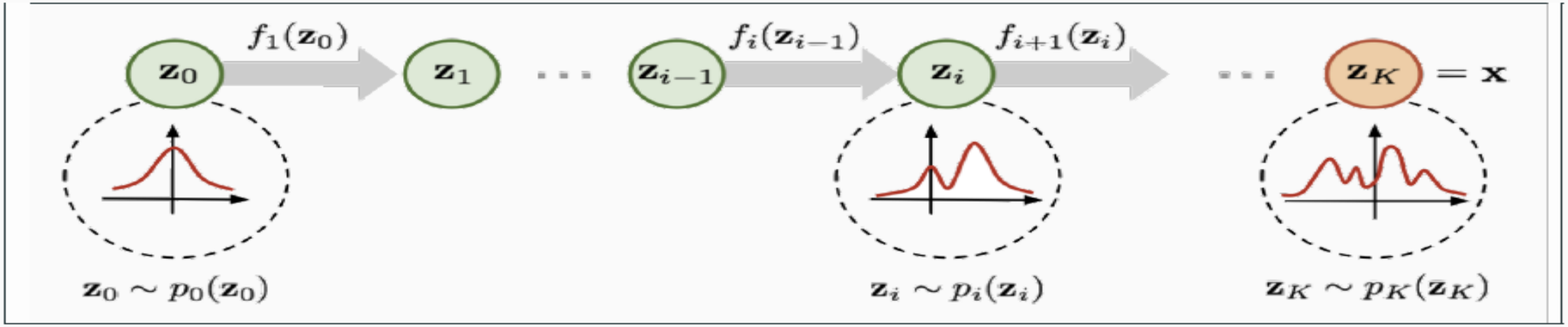
- **Normalizing Flow (NF)**: a flow of invertible transformations that takes $z_0 \sim \mathcal{N}(0, I)$ and outputs x following a complex distribution

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



It shows that training a normalizing flow learns a nonlinear, invertible map that turns a complex x-distribution into a near-standard normal z-distribution—making both **exact likelihood evaluation** and **generation** possible.

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

A normalizing flow learns a sequence of invertible maps so that data can be transformed into simple Gaussian noise (for training and likelihood), and Gaussian noise can be transformed back into realistic samples (for generation).

Unlocking Normalizing Flows: The Future of Generative Modeling

The Flow Mechanism

Simple Base Distribution
(e.g., Gaussian)

The Power of Diffeomorphisms
Transformations must be differentiable and invertible to ensure the density is well-defined.

Complex Target Distribution
(Real Data)

From Simple Base to Complex Target
A simple distribution is warped through a sequence of invertible bijections to model comples data.

Change of Varisbles Law
Uses Jacobian determinants to quantify volume changes, ensuring the total probability mass remains one.

Jacobian Determinant

Strategic Advantages & Applications

Exact Latent Variable Inference
Unlike VAEs, flows provide a unique, perfectly reversible encoding for every data point.

Stable Likelihood Training
Negative log likelihood can be directly computed and minimized as a loss function for stability.

Adversarial Loss (GAN)

On-Manifold Semantic Perturbations
Modifying latent codes allows for generating realiolis. label preserving data augmentations that improve classifier generalization.

Generative Model Architectures Comparison

	Exact	Likelihood	Latent Inference
Invertibility	Normalizing Flows	GANs	VAEs
	Exact & Reversible	Non-Invertible	Approximate
	Explicitly Modeled	Implicitly Modeled	Lower Bound (ELBO)
	Exact Inference	Hard to Compute	Approximate

NotebookLM

Expressive Power of Flow-Based Models

Normalizing flows can approximate a broad class of smooth densities arbitrarily well via invertible transformations of a simple base distribution. In practice, their expressive power is governed by architectural constraints required for tractable Jacobian determinants (e.g., triangular structure), which can increase the depth needed to capture strong cross-dimensional interactions, sharp multimodality, or near-disconnected supports.

$$p_X(\mathbf{x}) = \prod_{i=1}^{D} p_X(x_i \mid x_{<i})$$

Rosenblatt Transformation

$$z_i = F_i(x_i, x_{<i}) = \int_{-\infty}^{x_i} p_X(x_i' \mid x_{<i})\, dx_i' = \Pr(X_i' \le x_i \mid x_{<i})$$

$$x_i = \left(F_i(\cdot, x_{<i})\right)^{-1}(z_i), \qquad i = 1, \ldots, D$$

$$\det J_F(\mathbf{x}) = \prod_{i=1}^{D} \frac{\partial F_i}{\partial x_i} = \prod_{i=1}^{D} p_X(x_i \mid x_{<i}) = p_X(\mathbf{x})$$

$$p_Z(\mathbf{z}) = p_X(\mathbf{x}) \left|\det J_F(\mathbf{x})\right|^{-1} = p_X(\mathbf{x}) \left|p_X(\mathbf{x})\right|^{-1} = 1$$

Conclusion

The transformed variable
z = F(x)
is uniformly distributed
on the open unit cube (0,1)^D.

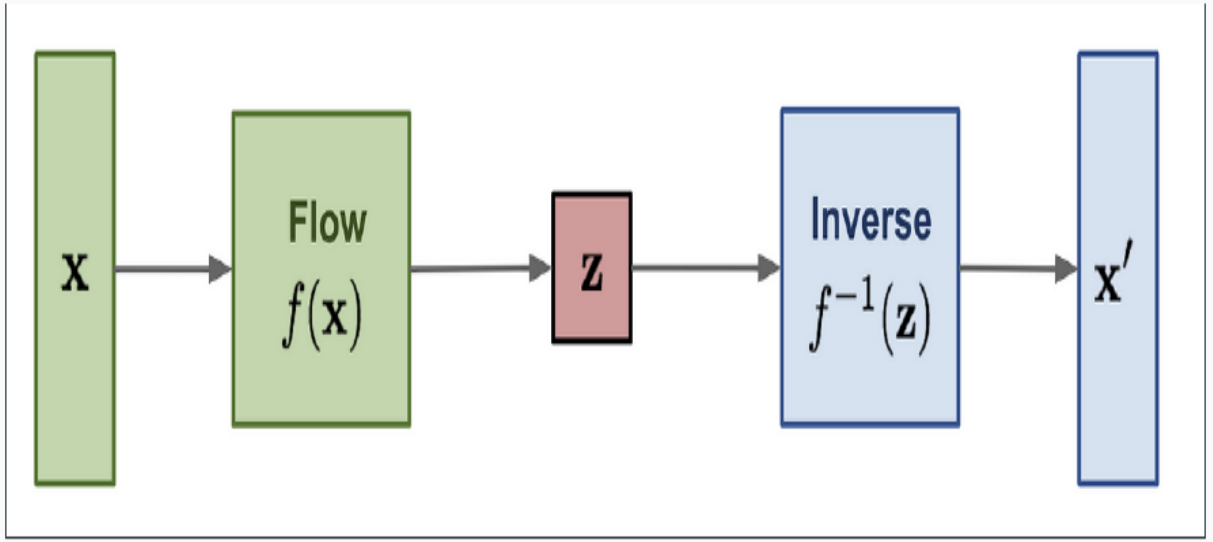
Normalizing Flow – Deriving p(x)

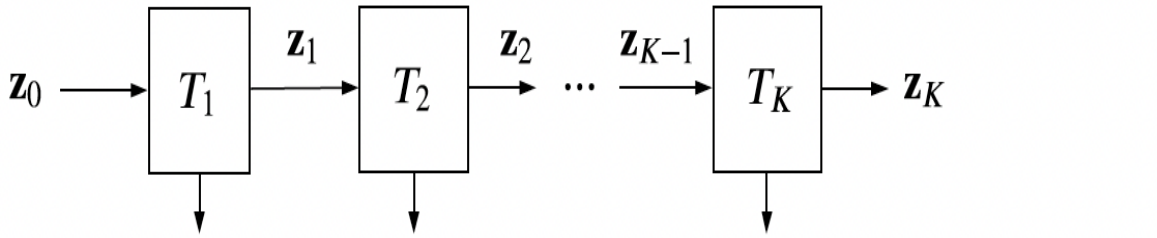
- **Change of Variables Theorem:**

  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}





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

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

Normalizing Flow

- 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

$$z_0 \sim p_0, \quad z_k = f_k(z_{k-1}), \quad x = z_K.$$

$$-\log p_\theta(x) = -\log p_0(z_0) + \sum_{k=1}^{K} \log \left|\det J_{f_k}(z_{k-1})\right|.$$

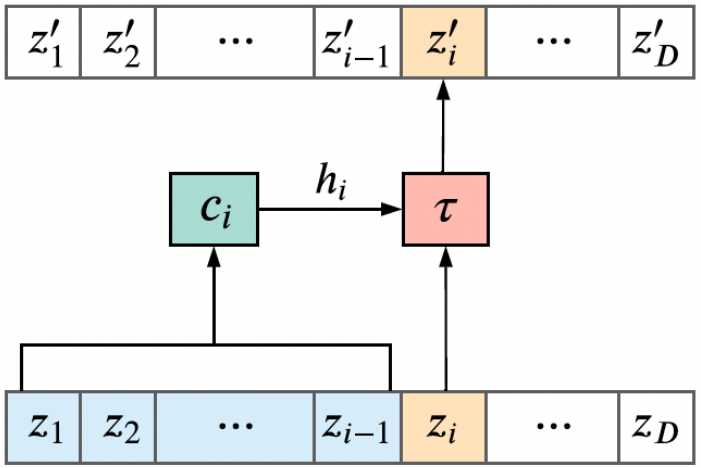
You want f_k to be:

- **Expressive** (can warp distributions a lot), but also
- **Computationally tractable**, meaning:

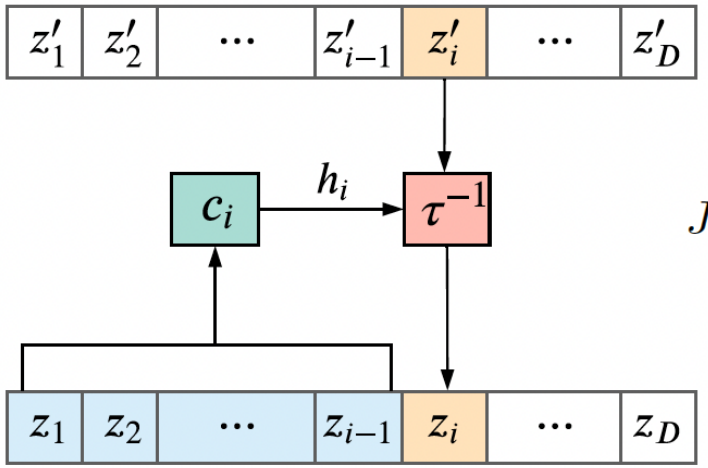
  inverse f_k^{-1} is fast, and $log\,|\det J_{f_k}|$ is fast.

This is why flows use special architectures that yield easy Jacobians.

Autoregressive Flow



(a) Forward



(b) Inverse

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial z_1}(z_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial z_D}(z_D; \boldsymbol{h}_D) \end{bmatrix}.$$

$$z'_i = \tau(z_i; \boldsymbol{h}_i) \quad \text{where} \quad \boldsymbol{h}_i = c_i(\mathbf{z}_{<i})$$

$$\log \left|\det J_{f_\phi}(\mathbf{z})\right| = \log \left| \prod_{i=1}^{D} \frac{\partial \tau}{\partial z_i}(z_i; \boldsymbol{h}_i) \right| = \sum_{i=1}^{D} \log \left| \frac{\partial \tau}{\partial z_i}(z_i; \boldsymbol{h}_i) \right|$$

$$z_i = \tau^{-1}(z'_i; \boldsymbol{h}_i) \quad \text{where} \quad \boldsymbol{h}_i = c_i(\mathbf{z}_{<i}).$$

τ is termed the *transformer* and c_i the i-th *conditioner*.

The transformer is a strictly monotonic function of z_i (and therefore invertible)

Autoregressive Flow: Transformers

Affine transformers

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = \alpha_i \mathrm{z}_i + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\alpha_i, \beta_i\}$$

$$\log \left| \det J_{f_{\boldsymbol{\phi}}}(\mathbf{z}) \right| = \sum_{i=1}^{D} \log |\alpha_i| = \sum_{i=1}^{D} \tilde{\alpha}_i.$$

Combination-based transformers

- Conic combination: $\tau(\mathrm{z}) = \sum_{k=1}^{K} w_k \tau_k(\mathrm{z})$, where $w_k > 0$ for all k.
- Composition: $\tau(\mathrm{z}) = \tau_K \circ \cdots \circ \tau_1(\mathrm{z})$.

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = w_{i0} + \sum_{k=1}^{K} w_{ik} \sigma(\alpha_{ik} \mathrm{z}_i + \beta_{ik}) \quad \text{where} \quad \boldsymbol{h}_i = \{w_{i0}, \ldots, w_{iK}, \alpha_{ik}, \beta_{ik}\}$$

$$\alpha_{ik} > 0 \text{ and } w_{ik} > 0 \text{ for all } k \geq 1$$

Integration-based transformers

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = \int_0^{\mathrm{z}_i} g(\mathrm{z}; \boldsymbol{\alpha}_i)\, d\mathrm{z} + \beta_i \quad \text{where} \quad \boldsymbol{h}_i = \{\boldsymbol{\alpha}_i, \beta_i\}$$

$g(\cdot; \boldsymbol{\alpha}_i)$ can be any positive-valued neural network parameterized by $\boldsymbol{\alpha}_i$

$$\tau(\mathrm{z}_i; \boldsymbol{h}_i) = \int_0^{\mathrm{z}_i} \sum_{k=1}^{K} \left(\sum_{\ell=0}^{L} \alpha_{ik\ell}\, \mathrm{z}^{\ell} \right)^2 d\mathrm{z} + \beta_i$$

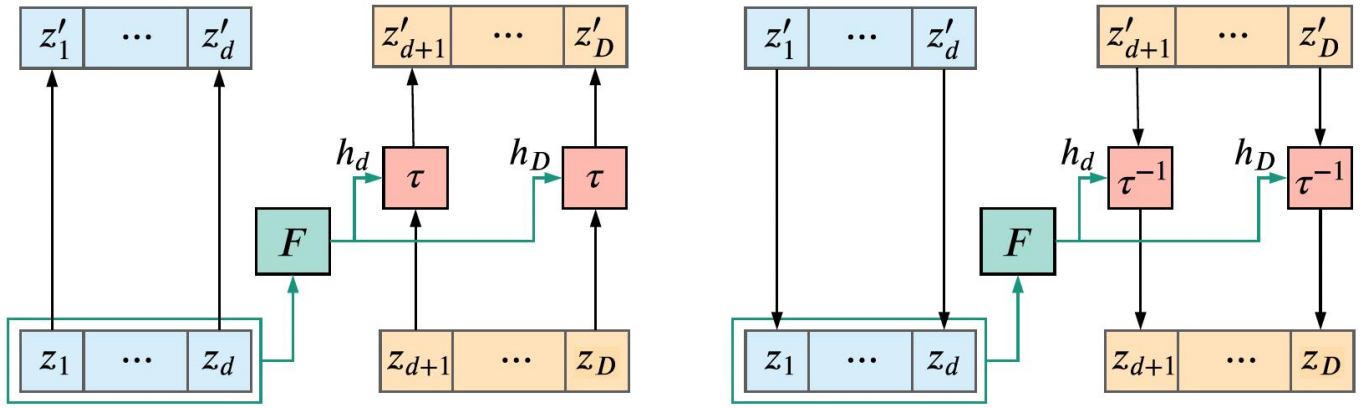
Autoregressive Flow: Conditioner

Recurrent conditioners

$$\boldsymbol{h}_i = c(\boldsymbol{s}_i) \quad \text{where} \quad \begin{aligned} \boldsymbol{s}_1 &= \text{initial state} \\ \boldsymbol{s}_i &= \text{RNN}(\mathrm{z}_{i-1}, \boldsymbol{s}_{i-1}) \text{ for } i > 1. \end{aligned}$$

Coupling layers

- Parameters $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_d)$ are constants, i.e. not a function of $\mathbf{z}$.
- Parameters $(\boldsymbol{h}_{d+1}, \ldots, \boldsymbol{h}_D)$ are functions of $\mathbf{z}_{\leq d}$ only, i.e. they don't depend on $\mathbf{z}_{>d}$.



(a) Forward (b) Inverse

RealNVP

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

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

- 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

- 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

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

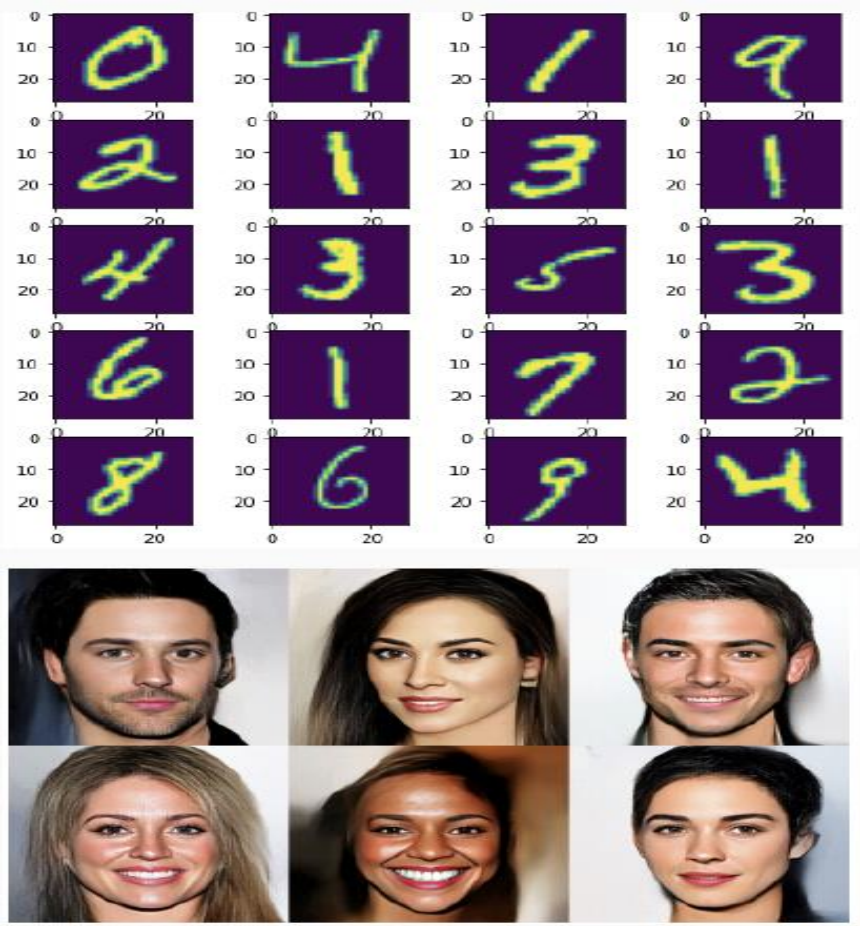
RealNVP – important of good masking

This image illustrates a comparative performance analysis of different **coupling layer masking strategies** in a Normalizing Flow generative model (likely a variant of **RealNVP** or **Glow**), trained on a dataset of human faces (similar to CelebA). The comparison demonstrates how different spatial and channel-wise partitioning affects the model's ability to synthesize coherent global structures and fine details.

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

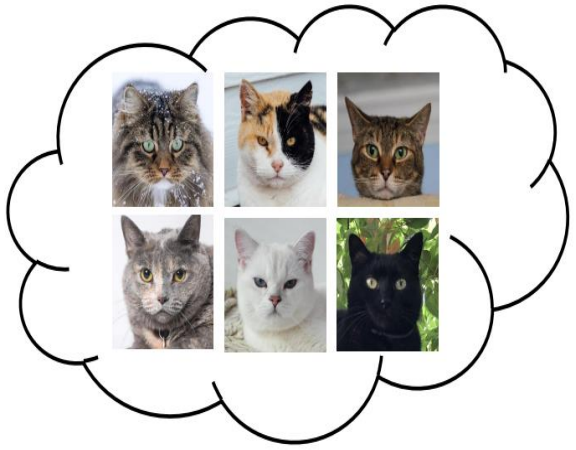
(Mask top half; mask bottom half; mask left half; mask right half) x2

Generative Images



What is diffusion model? –generative model

Learning to generate data



Samples from a Data Distribution

Train

Neural Network

Sample

Top 50 keywords

Keyword	Count
Large Language Models	318
Reinforcement Learning	201
Graph Neural Networks	123
Diffusion Models	112

Sidenote: Top3 conferences in AI are NeurIPS, ICML and ICLR.

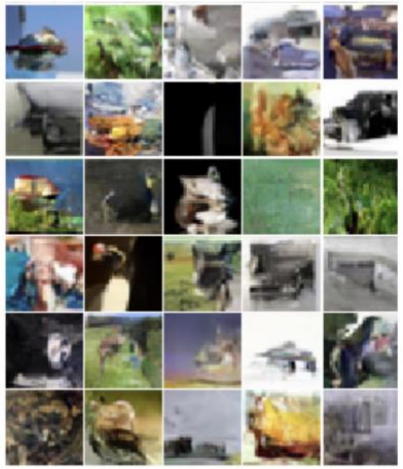
The history of generative models

VAE->GAN->Diffusion

VAEs, 2013

GANs, 2014

PixelCNN, 2016



BigGAN, 2019

Imagen, 2022

VAE → GAN → Diffusion

This summarizes the **evolutionary path** of generative models—from early variational methods, through adversarial training, to diffusion-based architectures, each offering better **fidelity**, **control**, and **semantic understanding**.

Text to Image

DALL·E 2

"a teddy bear on a skateboard in times square"

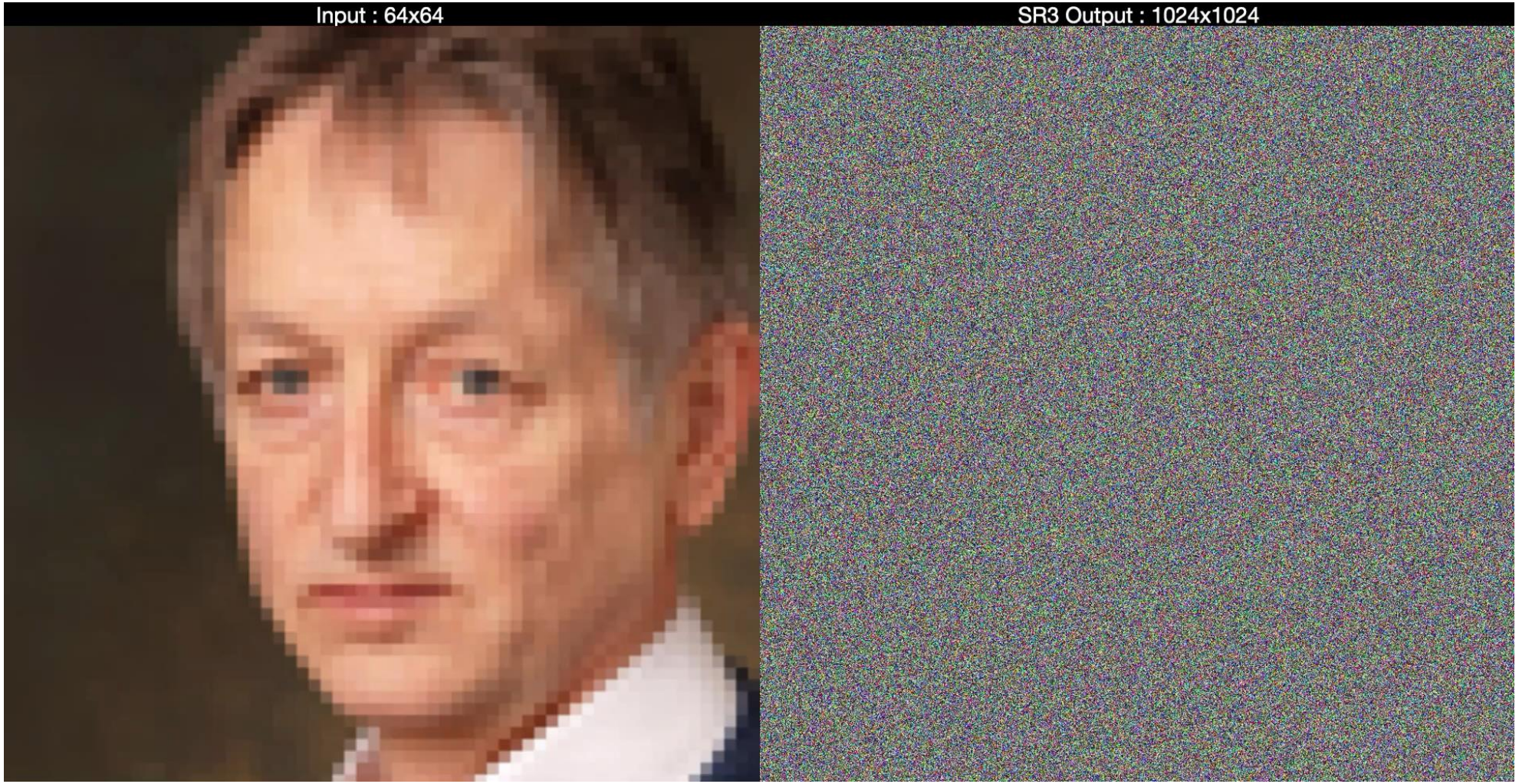
"Hierarchical Text-Conditional Image Generation with CLIP Latents" Ramesh et al., 2022

Imagen

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

"Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", Saharia et al., 2022

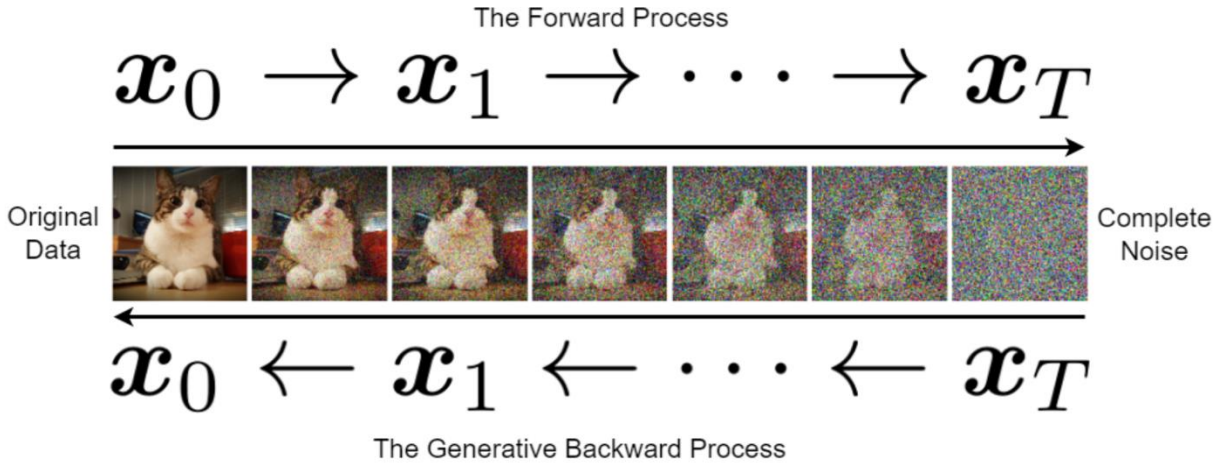
Image Resolution



Discrete time diffusion models

Diffusion models are incremental updates where the assembly of the whole gives us the encoder-decoder structure.

- Forward diffusion process $q(x_t|x_{t-1})$, gradually adds Gaussian noise to an image, until you end up with pure noise.



forward from $\mathbf{x}_0$ to $\mathbf{x}_T$:

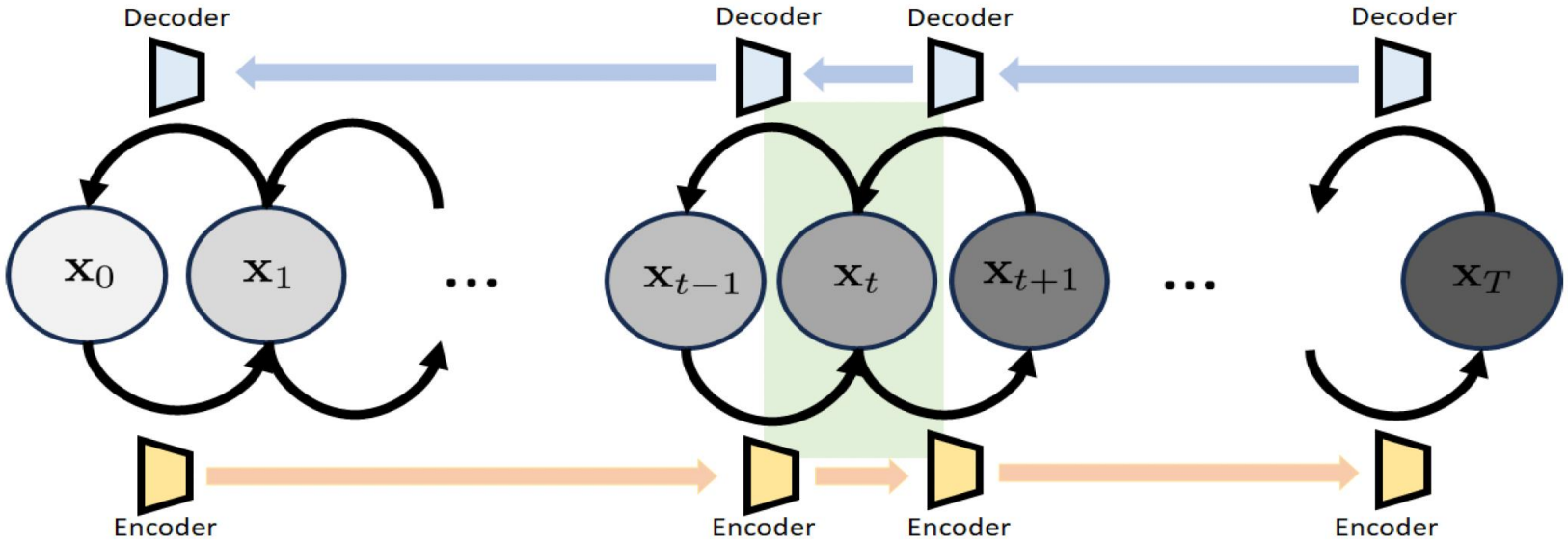
$$q_{\boldsymbol{\phi}}(\mathbf{x}_{0:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_{\boldsymbol{\phi}}(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

reverse from $\mathbf{x}_T$ to $\mathbf{x}_0$:

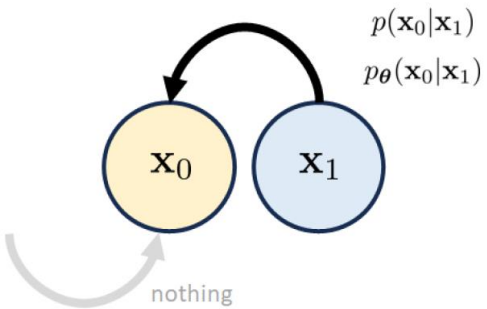
$$p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

- Reverse denoising diffusion process $p_\theta(x_{t-1}|x_t)$, where a neural network is trained to gradually denoise an image starting from pure noise, until you end up with an actual image.

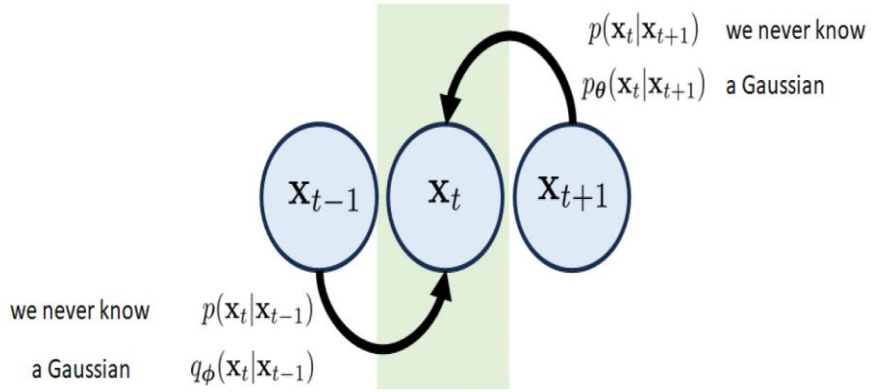
Variational diffusion models



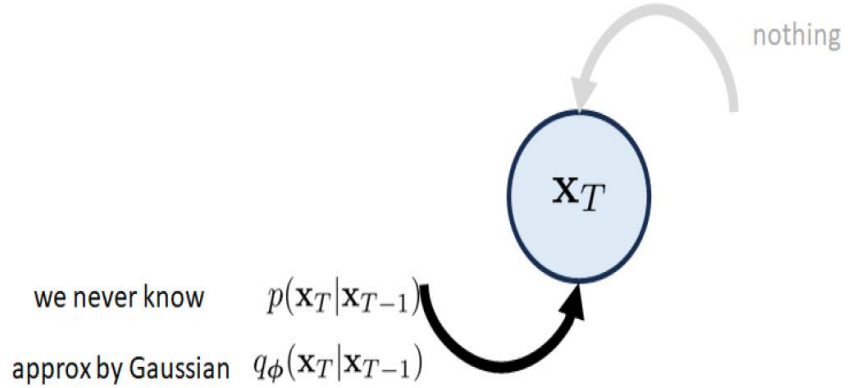
Initial Block



Transitional Block



Final Block



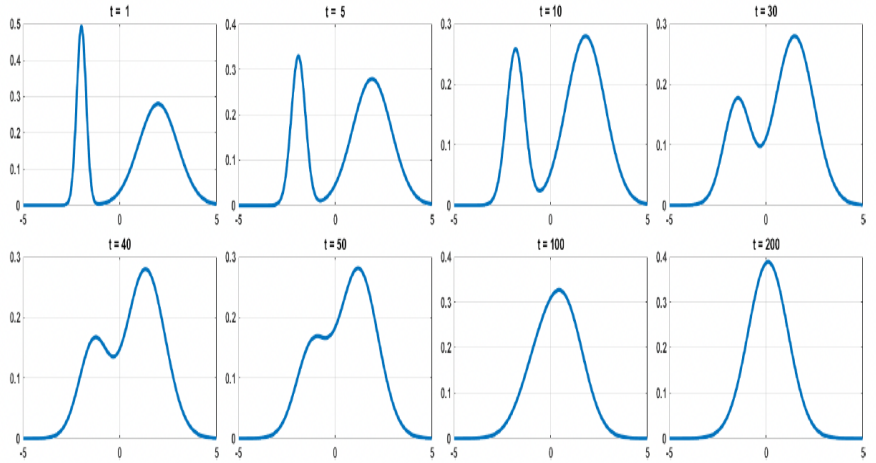
Forward Process

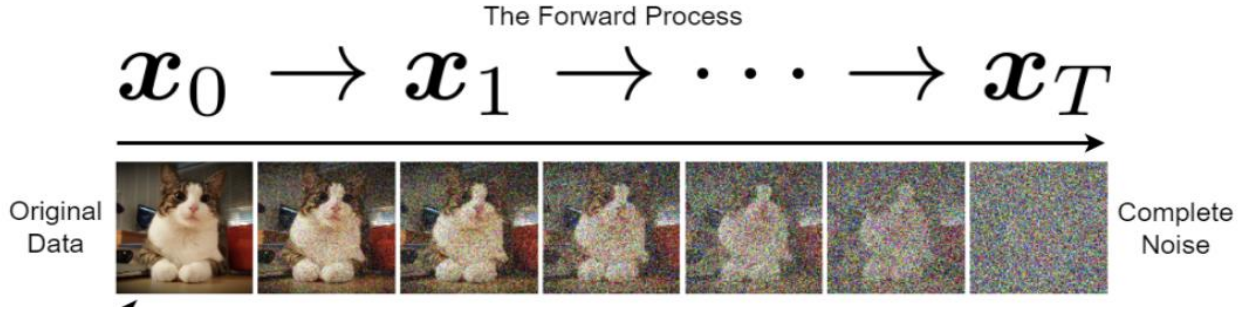
Theorem 2.1. (Why $\sqrt{\alpha}$ and $1-\alpha$?) Suppose that $q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t \,|\, a\mathbf{x}_{t-1}, b^2\mathbf{I})$ for some constants a and b. If we want to choose a and b such that the distribution of $\mathbf{x}_t$ will become $\mathcal{N}(0, \mathbf{I})$, then it is necessary that

$$a = \sqrt{\alpha} \qquad \text{and} \qquad b = \sqrt{1-\alpha}.$$

Therefore, the transition distribution is

$$q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_{t-1}) \overset{\text{def}}{=} \mathcal{N}(\mathbf{x}_t \,|\, \sqrt{\alpha}\mathbf{x}_{t-1}, (1-\alpha)\mathbf{I}). \tag{2.4}$$





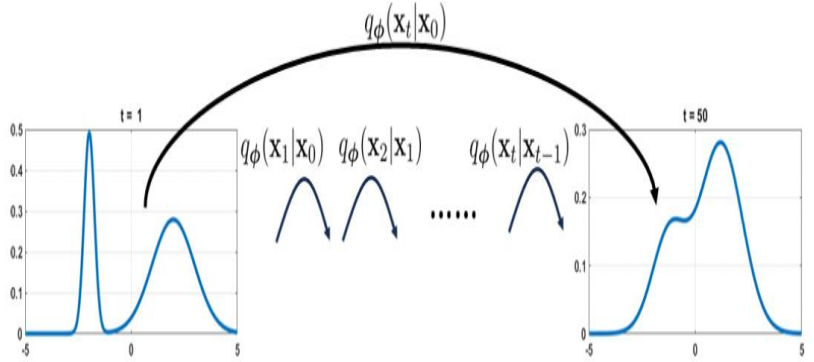
forward from $\mathbf{x}_0$ to $\mathbf{x}_T$:

$$q_{\boldsymbol{\phi}}(\mathbf{x}_{0:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q_{\boldsymbol{\phi}}(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

Theorem 2.2. (Conditional Distribution $q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_0)$). The conditional distribution $q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_0)$ is given by

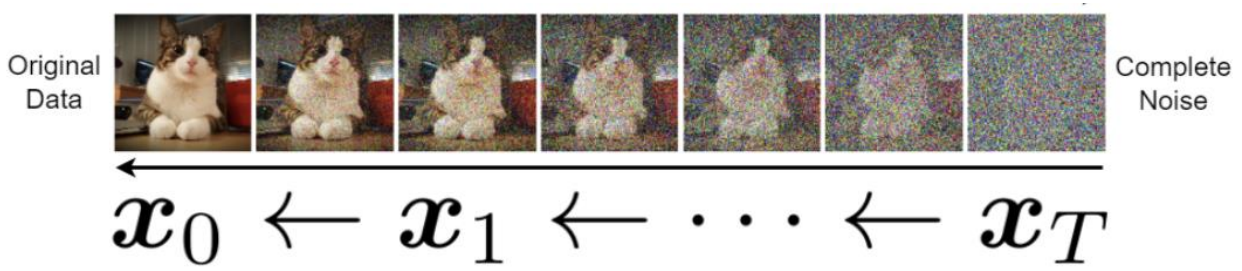
$$q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \,|\, \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \; (1-\bar{\alpha}_t)\mathbf{I}), \tag{2.8}$$

where $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.



Backward Process

The more interesting part is the backward process, which is realized through a chain of **denoising operations.** Each denoising step should be coupled with the corresponding step in the forward process. We try to obtain $p_\theta(x_{t-1}|x_t)$, so we can generate image x_0 from Gaussian noise x_T. However, the explicit form of $p_\theta(x_{t-1}|x_t)$ is not available, but we can model it with a Gaussian and try to learn the parameters with neural network:



The Generative Backward Process

reverse from $\mathbf{x}_T$ to $\mathbf{x}_0$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

In the original Denoising Diffusion Probabilistic Model (DDPM) paper, the author assume the variance Σ_θ to be fixed and only μ_θ is unknown. In the improved DDPM paper, variance is also modeled.

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}| \quad \underbrace{\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{a network}} \quad , \sigma_q^2(t)\mathbf{I}\right),$$

$$\underbrace{\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{a network}} \overset{\text{def}}{=} \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}\mathbf{x}_t + \frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t} \underbrace{\widehat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{another network}} .$$

How to learn the parameters?

Evidence Lower Bound

The training tries to maximize the likelihood of x_0.

For training, we can form variational upper bound (negative ELBO) that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

We can obtain the following decomposition:

Theorem 2.3. (ELBO for Variational Diffusion Model). The ELBO for the variational diffusion model is

$$\begin{aligned}\text{ELBO}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x}_1|\mathbf{x}_0)}\Big[\log \underbrace{p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)}_{\text{how good the initial block is}}\Big] \\ - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x}_{T-1}|\mathbf{x}_0)}\Big[\underbrace{\mathbb{D}_{\text{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{x}_T|\mathbf{x}_{T-1})\|p(\mathbf{x}_T)\Big)}_{\text{how good the final block is}}\Big] \\ - \sum_{t=1}^{T-1} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1},\mathbf{x}_{t+1}|\mathbf{x}_0)}\Big[\underbrace{\mathbb{D}_{\text{KL}}\Big(q_{\boldsymbol{\phi}}(\mathbf{x}_t|\mathbf{x}_{t-1})\|p_{\boldsymbol{\theta}}(\mathbf{x}_t|\mathbf{x}_{t+1})\Big)}_{\text{how good the transition blocks are}}\Big], \qquad (2.14)\end{aligned}$$

where $\mathbf{x}_0 = \mathbf{x}$, and $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.

$$q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1}, \mathbf{x}_{t+1}|\mathbf{x}_0)$$

- **Challenge**:
  - It's not explicitly known.
  - Even though it's assumed to be **Gaussian**, we still require access to **future samples** ($\mathbf{x}_{t+1}$ (to draw the current sample ($\mathbf{x}_t$,(which is **counterintuitive**.

Evidence Lower Bound

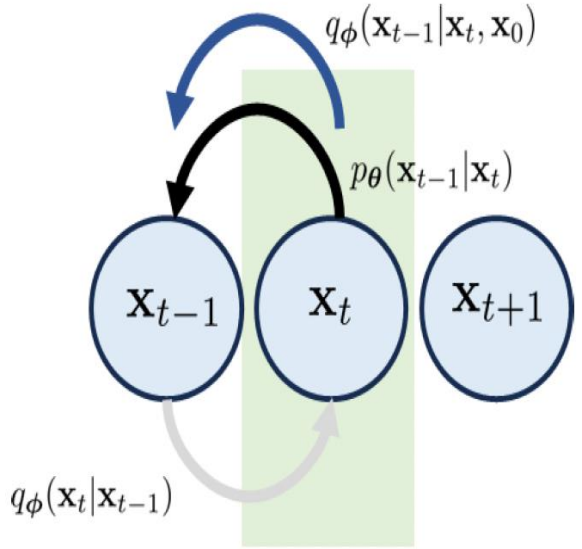
Bayesian Trick.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t)q(\mathbf{x}_t)}{q(\mathbf{x}_{t-1})} \quad \xrightarrow{\text{condition on } \mathbf{x}_0} \quad q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}.$$

Theorem 2.4. (ELBO for Variational Diffusion Model). Let $\mathbf{x} = \mathbf{x}_0$, and $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. The ELBO for a variational diffusion model in Theorem 2.3 can be equivalently written as

$$\text{ELBO}_{\phi,\theta}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{x}_1|\mathbf{x}_0)}[\log \underbrace{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{\text{same as before}}] - \underbrace{\mathbb{D}_{\text{KL}}\left(q_\phi(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T)\right)}_{\text{new prior matching}}$$

$$- \sum_{t=2}^{T} \mathbb{E}_{q_\phi(\mathbf{x}_t|\mathbf{x}_0)} \underbrace{\left[\mathbb{D}_{\text{KL}}\left(q_\phi(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\right)\right]}_{\text{new consistency}}. \qquad (2.21)$$



Distribution of Backward Process

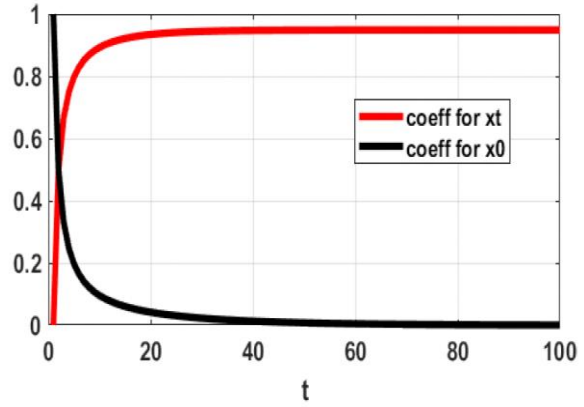
Theorem 2.5. The distribution $q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ takes the form of

$$q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1} \,|\, \boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \boldsymbol{\Sigma}_q(t)),$$

where

$$\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}\mathbf{x}_t + \frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}\mathbf{x}_0$$
$$\boldsymbol{\Sigma}_q(t) = \frac{(1-\alpha_t)(1-\sqrt{\overline{\alpha}_{t-1}})}{1-\overline{\alpha}_t}\mathbf{I} \stackrel{\text{def}}{=} \sigma_q^2(t)\mathbf{I},$$

where $\overline{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.



: The trajectory of the coefficients for $\mathbf{x}_t$ and for $\mathbf{x}_0$, as t grows.

Theorem 2.6. The **ELBO for a variational diffusion model** in Eqn (2.21) can be simplified to

$$\text{ELBO}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)] - \underbrace{\mathbb{D}_{\text{KL}}\Big(q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T)\Big)}_{\text{nothing to train}}$$
$$-\sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\Big[\frac{1}{2\sigma_q^2(t)}\|\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)\|^2\Big], \qquad (2.38)$$

where $\mathbf{x} = \mathbf{x}_0$, and $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.

$$q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\Big(\mathbf{x}_{t-1} \,|\, \underbrace{\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)}_{\text{known}}, \underbrace{\sigma_q^2(t)\mathbf{I}}_{\text{known}}\Big),$$

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\Big(\mathbf{x}_{t-1}| \underbrace{\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{neural network}}, \underbrace{\sigma_q^2(t)\mathbf{I}}_{\text{known}}\Big).$$

Model training

L_{t-1}: Tractable term. From Bayes rule:

$$q_{\boldsymbol{\phi}}(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)=\mathcal{N}\Big(\mathbf{x}_{t-1}\,|\,\underbrace{\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0)}_{\text{known}},\underbrace{\sigma_q^2(t)\mathbf{I}}_{\text{known}}\Big),$$

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)=\mathcal{N}\Big(\mathbf{x}_{t-1}|\quad\underbrace{\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{neural network}}\quad,\underbrace{\sigma_q^2(t)\mathbf{I}}_{\text{known}}\Big).$$

$$\boldsymbol{\mu}_q(\mathbf{x}_t,\mathbf{x}_0)=\frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}\mathbf{x}_t+\frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}\mathbf{x}_0.$$

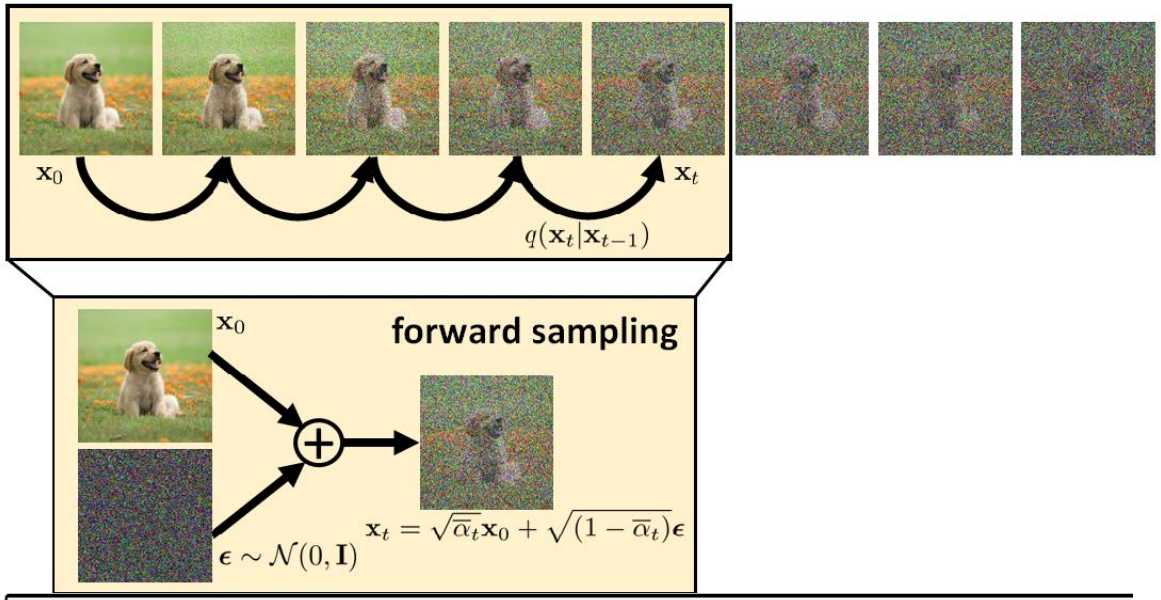
$$\underbrace{\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{a network}}\overset{\text{def}}{=}\frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}\mathbf{x}_t+\frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}\underbrace{\widehat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t)}_{\text{another network}}.$$

$$\mathbf{x}_t=\sqrt{\overline{\alpha}_t}\mathbf{x}_0+\sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}_0$$

$$\begin{aligned}
&\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\sum_{\mathbf{x}_0\in\mathcal{X}}\text{ELBO}(\mathbf{x}_0)\\
&=\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\sum_{\mathbf{x}_0\in\mathcal{X}}\sum_{t=1}^{T}\mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{1}{2\sigma_q^2(t)}\frac{(1-\alpha_t)^2\overline{\alpha}_{t-1}}{(1-\overline{\alpha}_t)^2}\left\|\widehat{\mathbf{x}}_{\boldsymbol{\theta}}\left(\sqrt{\overline{\alpha}_t}\mathbf{x}_0+\sqrt{(1-\overline{\alpha}_t)}\boldsymbol{\epsilon}_t\right)-\mathbf{x}_0\right\|^2\right]\\
&=\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\sum_{\mathbf{x}_0\in\mathcal{X}}\sum_{t=1}^{T}\frac{1}{M}\sum_{m=1}^{M}\frac{1}{2\sigma_q^2(t)}\frac{(1-\alpha_t)^2\overline{\alpha}_{t-1}}{(1-\overline{\alpha}_t)^2}\left\|\widehat{\mathbf{x}}_{\boldsymbol{\theta}}\left(\sqrt{\overline{\alpha}_t}\mathbf{x}_0+\sqrt{(1-\overline{\alpha}_t)}\boldsymbol{\epsilon}_t^{(m)}\right)-\mathbf{x}_0\right\|^2,
\end{aligned}$$

Training Diffusion in DDPM

Forward Diffusion in DDPM

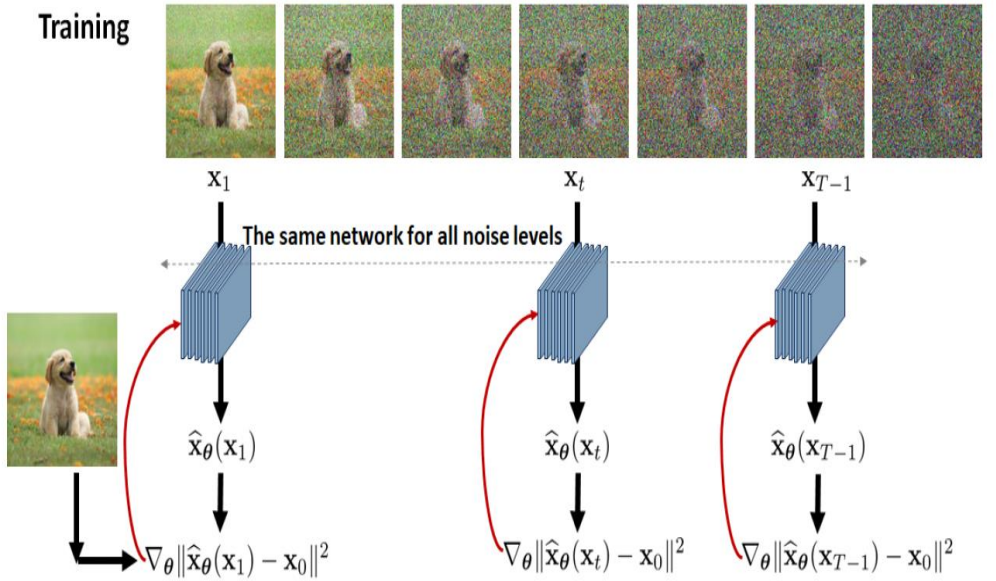


Training Algorithm for DDPM. For every image $\mathbf{x}_0$ in your training dataset:

- Repeat the following steps until convergence.
- Pick a random time stamp $t \sim \text{Uniform}[1, T]$.
- Draw a sample $\mathbf{x}_t^{(m)} \sim \mathcal{N}(\mathbf{x}_t \,|\, \sqrt{\bar{\alpha}_t}\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I})$, i.e.,

$$\mathbf{x}_t^{(m)} = \bar{\alpha}_t \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon_t^{(m)}, \qquad \epsilon_t^{(m)} \sim \mathcal{N}(0, \mathbf{I}).$$

Training Diffusion in DDPM



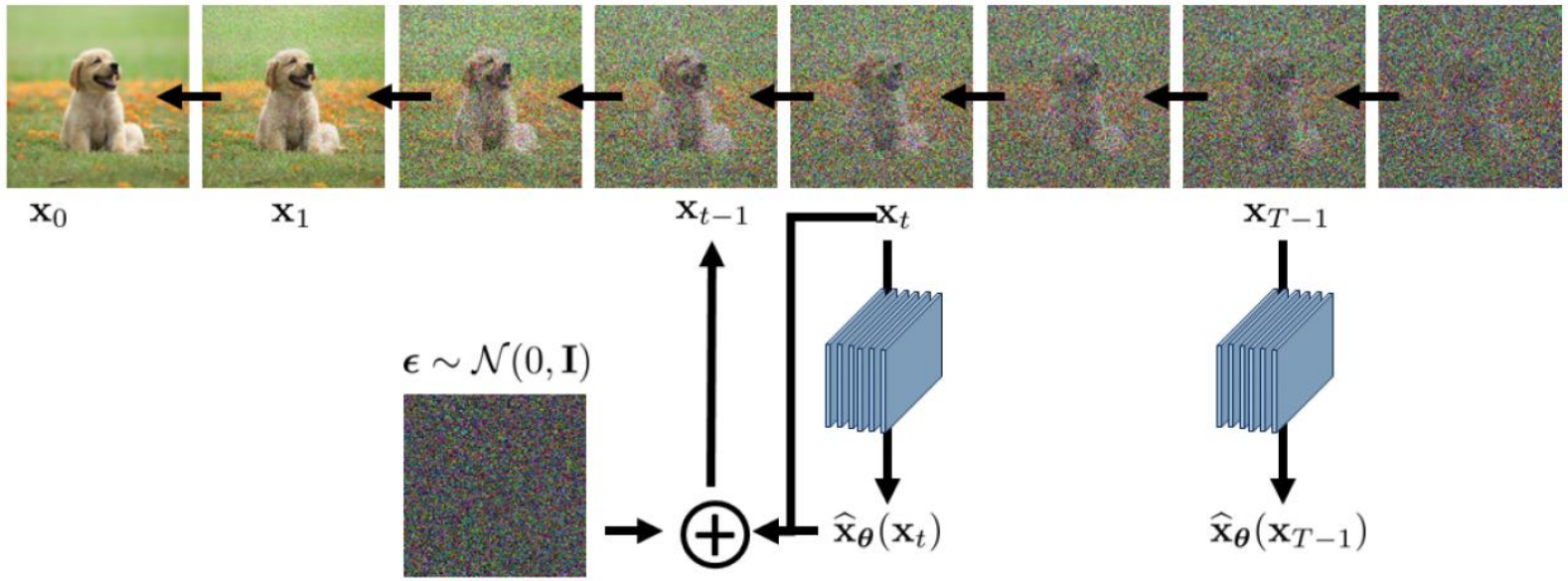
- Take gradient descent step on

$$\nabla_\theta \left\{ \frac{1}{M} \sum_{m=1}^{M} \left\| \hat{\mathbf{x}}_\theta(\mathbf{x}_t^{(m)}) - \mathbf{x}_0 \right\|^2 \right\}.$$

You can do this in batches, just like how you train any other neural networks. Note that, here, you are training **one** denoising network $\hat{\mathbf{x}}_\theta$ for **all** noisy conditions.

In DDPM, α_t is usually not chosen directly. Instead, one specifies a variance schedule β_t, sets $\alpha_t = 1 - \beta_t$, and designs the cumulative quantity $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ to decay smoothly from 1 to near 0. Common choices include linear and cosine schedules.

Inference in DDPM



Inference of DDPM.

- You give us a white noise vector $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
- Repeat the following for $t = T, T-1, \ldots, 1$.
- We calculate $\widehat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t)$ using our trained denoiser.
- Update according to

$$\mathbf{x}_{t-1} = \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t}\mathbf{x}_t + \frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t}\widehat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t) + \sigma_q(t)\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

Prediction Noise

Training DDPM using $\widehat{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t)$. For every image $\mathbf{x}_0$ in your training dataset:

- Repeat the following steps until convergence.
- Pick a random time stamp $t \sim \text{Uniform}[1, T]$.
- Draw a sample $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(0, \mathbf{I})$.
- Draw a sample $\mathbf{x}_t \sim \mathcal{N}(\mathbf{x}_t \,|\, \sqrt{\overline{\alpha}_t}\mathbf{x}_0, \;\; (1-\overline{\alpha}_t)\mathbf{I})$, i.e.,

$$\mathbf{x}_t = \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\overline{\alpha}_t)}\boldsymbol{\epsilon}_0.$$

- Take gradient descent step on

$$\nabla_{\boldsymbol{\theta}} \left\| \widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t) - \boldsymbol{\epsilon}_0 \right\|^2.$$

Inference of DDPM using $\hat{\epsilon}_{\theta}(\mathrm{x}_t)$.

- You give us a white noise vector $\mathrm{x}_T \sim \mathcal{N}(0, \mathrm{I})$.
- Repeat the following for $t = T, T-1, \ldots, 1$.
- We calculate $\hat{\mathrm{x}}_{\theta}(\mathrm{x}_t)$ using our trained denoiser.
- Update according to

$$\mathrm{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathrm{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathrm{x}_t)\right) + \sigma_q(t)\mathrm{z}, \qquad \mathrm{z} \sim \mathcal{N}(0, \mathrm{I}).$$

$$\begin{aligned}
\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\overline{\alpha}_{t-1}}(1-\alpha_t)\mathbf{x}_0}{1-\overline{\alpha}_t} \\
&= \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})\mathbf{x}_t + \sqrt{\overline{\alpha}_{t-1}}(1-\alpha_t)\cdot\frac{\mathbf{x}_t - \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}_0}{\sqrt{\overline{\alpha}_t}}}{1-\overline{\alpha}_t} \\
&= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\boldsymbol{\epsilon}_0.
\end{aligned}$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t).$$

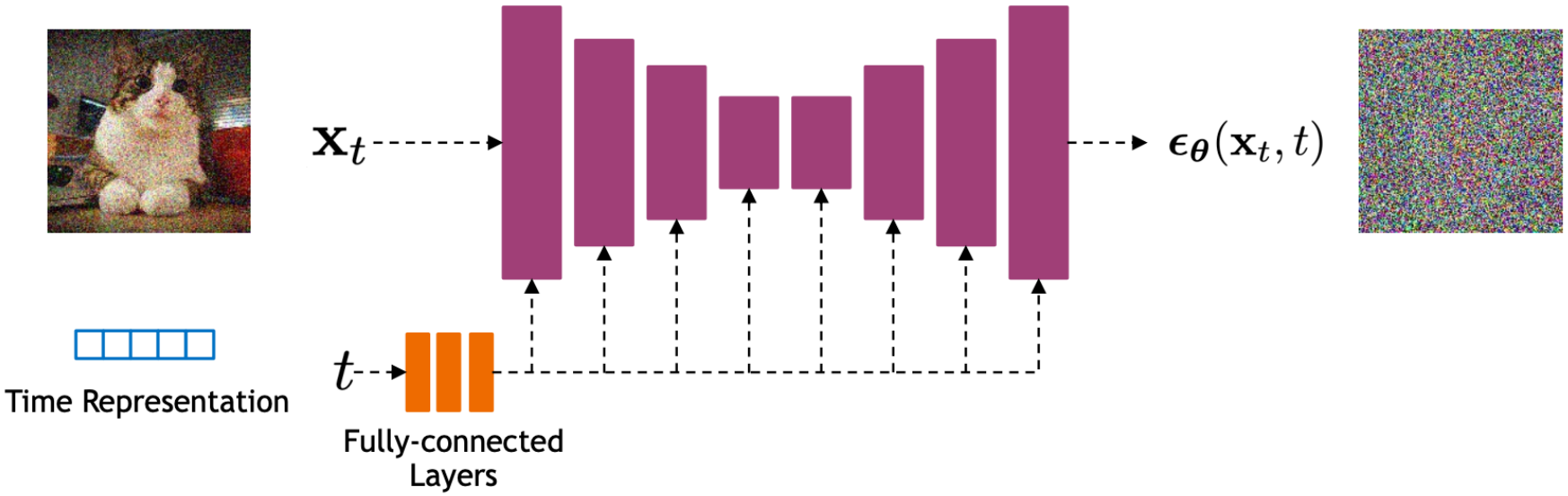
$$\begin{aligned}
\text{ELBO}_{\boldsymbol{\theta}}(\mathbf{x}_0, \boldsymbol{\epsilon}_0) &= -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{1}{2\sigma_q^2(t)}\frac{(1-\alpha_t)^2}{(1-\overline{\alpha}_t)\alpha_t}\left\|\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t) - \boldsymbol{\epsilon}_0\right\|^2\right] \\
&= -\sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}\left[\frac{1}{2\sigma_q^2(t)}\frac{(1-\alpha_t)^2}{(1-\overline{\alpha}_t)\alpha_t}\left\|\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\left(\sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\boldsymbol{\epsilon}_0\right) - \boldsymbol{\epsilon}_0\right\|^2\right]
\end{aligned}$$

$$\begin{aligned}
\mathbf{x}_{t-1} &= \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t) + \sigma_q(t)\mathbf{z}, \qquad \text{where} \qquad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}) \\
&= \left(\frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t)\right) + \sigma_q(t)\mathbf{z} \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t)\right) + \sigma_q(t)\mathbf{z}.
\end{aligned}$$

Model training

So the loss minimization degenerate to noise prediction task, which is minimizing the noise prediction MSE:

In practice, we can use Unet with time encoding to construct ϵ.



Denoising Diffusion Implicit Model (DDIM)

It is very slow to generate a sample from DDPM by following the Markov chain of the reverse diffusion process, as T can be up to one or a few thousand steps.

DDPM Transition Distribution is defined as

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) \overset{\text{def}}{=} \mathcal{N}\Big(\mathbf{x}_t \,|\, \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I}\Big)$$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\Big(\mathbf{x}_t \,|\, \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \quad (1-\bar{\alpha}_t)\mathbf{I}\Big)$$

Markov Property:

- The forward diffusion process is memoryless.
- Each step only depends on the immediate previous step.
- This can lead to slow convergence.

DDIM Transition Distribution is defined as

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\Big(\sqrt{\alpha_t}\mathbf{x}_0, \quad (1-\alpha_t)\mathbf{I}\Big),$$

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\alpha_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_0}{\sqrt{1-\alpha_t}}\right) \quad , \quad \sigma_t^2\mathbf{I}\right)$$

DDIM Rational:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) \overset{\text{def}}{=} \mathcal{N}\Big(\mathbf{x}_t \,\Big|\, \sqrt{\frac{\alpha_t}{\alpha_{t-1}}}\mathbf{x}_{t-1}, \quad (1-\frac{\alpha_t}{\alpha_{t-1}})\mathbf{I}\Big)$$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\Big(\sqrt{\alpha_t}\mathbf{x}_0, \quad (1-\alpha_t)\mathbf{I}\Big)$$

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \qquad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}, \qquad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}$$

$$= \sqrt{\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\alpha_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_0}{\sqrt{1-\alpha_t}}\right).$$

Speed up Diffusion Model Sampling

One simple way is to run a strided sampling schedule by taking the sampling update every T/S steps to reduce the process from T to S steps. The new sampling schedule for generation is $\{\tau_1, \ldots, \tau_S\}$ where $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_S \in [1, T]$.

$$x_{\tau_i} = \sqrt{\bar{\alpha}_{\tau_i}}\, x_0 + \sqrt{1 - \bar{\alpha}_{\tau_i}}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I), \quad \Rightarrow \quad \epsilon = \frac{x_{\tau_i} - \sqrt{\bar{\alpha}_{\tau_i}}\, x_0}{\sqrt{1 - \bar{\alpha}_{\tau_i}}}.$$

$$x_{\tau_{i-1}} = \sqrt{\bar{\alpha}_{\tau_{i-1}}}\, x_0 + \sqrt{1 - \bar{\alpha}_{\tau_{i-1}} - \sigma_{\tau_i}^2}\, \epsilon + \sigma_{\tau_i} z, \qquad z \sim \mathcal{N}(0, I).$$

$$q_{\sigma,\tau}(\mathbf{x}_{\tau_{t-1}} | \mathbf{x}_{\tau_t}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{\tau_{t-1}}; \sqrt{\bar{\alpha}_{\tau_{t-1}}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{t-1}} - \sigma_{\tau_t}^2} \frac{\mathbf{x}_{\tau_i} - \sqrt{\bar{\alpha}_{\tau_t}}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_{\tau_t}}}, \sigma_{\tau_t}^2 \mathbf{I})$$

The other approach is adjusting the sample stochasticity, which is adopted in the denoising diffusion implicit model(DDIM) paper. Specifically,

$$\sigma_t^* = \eta \sigma_t.$$

In the DDIM paper, they set $\eta = 0$ making the sampling process deterministic. They observed that DDIM ($\eta = 0$) can produce the best quality samples when S is small, while DDPM ($\eta = 1$) performs much worse on small S. DDPM does perform better when we can afford to run the full reverse Markov diffusion steps ($S = T$).

Can be founded in improved DDPM paper

Conditional diffusion models

- The diffusion models introduced so far are all **unconditional**: they generate realistic samples, but they do not allow us to control the content of the output. For example, a model trained on ImageNet can generate natural images, but it cannot reliably generate a specific category, such as a cat, on demand.

- Unconditional models are useful for demonstrating the generative capability of diffusion models. However, to build systems that are practical for users, we need models that can follow instructions or conditions. This motivates the development of **conditional diffusion models**.

- Conditional diffusion models are typically divided into two main classes: **classifier-guided models** and **classifier-free models**. Classifier guidance is relatively easy to apply because it only requires an additional pretrained classifier and can be combined with an existing pretrained unconditional diffusion model. In contrast, classifier-free guidance requires retraining the diffusion model with conditioning information, but it usually provides better performance and has become the dominant approach in modern large-scale generative systems.

Conditional diffusion models

(a) Image synthesis with language guidance

A smiling woman with straight, blonde hair wearing sunglasses.

(b) Image synthesis with image guidance

Image Guidance

Generated images

(c) Image synthesis with both language and image guidance

Language Guidance + Image Guidance

A photo of a woman with curly hair. +

Generated images

Classifier guidance

With bayes rule

$$p_{\theta,\phi}(x_{t-1}|x_t, y) = \frac{p_\theta(x_{t-1}|x_t)p_\phi(y|x_{t-1}, x_t)}{p_\phi(y|x_t)}$$

Adding noise won't affect classifying, $p_\phi(y|x_{t-1}) \approx p_\phi(y|x_{t-1}, x_t)$

$$p_{\theta,\phi}(x_{t-1}|x_t, y) = p_\theta(x_{t-1}|x_t)\frac{p_\phi(y|x_{t-1})}{p_\phi(y|x_t)} = p_\theta(x_{t-1}|x_t)e^{\log p_\phi(y|x_{t-1}) - \log p_\phi(y|x_t)}$$

Apply Taylor expansion

$$\begin{aligned}\log p_\phi(y|x_{t-1}) - \log p_\phi(y|x_t) &\approx (x_{t-1} - x_t)\nabla_{x_t} \log p_\phi(y|x_t) \\ &\approx (x_{t-1} - \mu_\theta(x_t, t))\nabla_{x_t} \log p_\phi(y|x_t)\end{aligned}$$

We know that

$$p_\theta(x_{t-1}|x_t) \sim \mathcal{N}(x_t; \mu_\theta(x_t, t), \sigma_\theta^2(x_t, t)\mathbf{I}) \propto \exp\left(-\frac{(x_t - \mu_\theta(x_t,t))^2}{2\sigma_\theta^2(x_t,t)}\right)$$

We have

$$\begin{aligned}p_{\theta,\phi}(x_{t-1}|x_t, y) &\propto \exp\left(-\frac{(x_t - \mu_\theta(x_t, t))^2}{2\sigma_\theta^2(x_t, t)} + (x_{t-1} - \mu_\theta(x_t, t))\nabla_{x_t} \log p_\phi(y|x_t)\right) \\ &\propto \exp\left(-\frac{(x_t - \mu_\theta(x_t, t) - \sigma_\theta^2(x_t, t)\nabla_{x_t} \log p_\phi(y|x_t))^2}{2\sigma_\theta^2(x_t, t)}\right)\end{aligned}$$

Classifier Guidance Algorithm

Phase 1: Training Stage

Unconditional Diffusion Model Training

The objective is to learn the score function of the data distribution $p(x)$ without label dependency.

- **Forward Process:** For a clean image x_0, sample $t \sim \text{Uniform}(1, T)$ and noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.
- **Noisy Image Construction:**

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

- **Objective:** Train a U-Net $\epsilon_\theta(x_t, t)$ to minimize the L_2 loss:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{x_0, \epsilon, t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right] \tag{2}$$

Time-Dependent Classifier Training

The classifier must be robust to Gaussian noise to provide accurate gradients during the reverse process.

- **Input:** Pair (x_t, y) where x_t is the noisy variant of an image from class y at timestep t.
- **Architecture:** A backbone (ResNet/ViT) modified to ingest time embeddings t.
- **Objective:** Minimize cross-entropy loss $\mathcal{L}_{\text{cls}}$ to predict $p_\phi(y|x_t, t)$.

Phase 2: Sampling Stage

Algorithm 1: Classifier Guided Sampling

Input: Target class y, guidance scale s, pre-trained ϵ_θ and p_ϕ
Output: Generated sample $x_0 \sim p(x|y)$
Sample $x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T$ **to** 1 **do**

1. **Forward Passes:**
   Compute unconditional noise: $\epsilon_\theta(x_t, t)$
   Compute log-probability: $\log p_\phi(y|x_t, t)$
2. **Gradient Computation:**
   Calculate spatial gradient: $\mathbf{g} \leftarrow \nabla_{x_t} \log p_\phi(y|x_t, t)$
   Note: $\mathbf{g}$ *points toward the image manifold maximizing class* y *probability.*
3. **Mean Perturbation (Guidance):**
   Compute standard reverse mean $\mu_\theta(x_t, t)$ from $\epsilon_\theta(x_t, t)$
   $\mu_\theta^{\text{guided}} \leftarrow \mu_\theta(x_t, t) + s\Sigma_t\mathbf{g}$
4. **Sampling:**
   Draw $x_{t-1} \sim \mathcal{N}(\mu_\theta^{\text{guided}}, \Sigma_t)$

return x_0

- s **(Guidance Scale):** Typically $s > 1$. Controls the trade-off between sample fidelity (alignment with y) and diversity.
- Σ_t**:** The variance of the reverse diffusion step.

Classifier Free

Phase 1: Training Stage

Given a clean sample-condition pair (x_0, y), sample

$$t \sim \text{Uniform}\{1, \ldots, T\}, \qquad \epsilon \sim \mathcal{N}(0, I),$$

and construct the noisy sample

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

During training, the condition is randomly replaced by a null condition:

$$\tilde{y} = \begin{cases} y, & \text{with probability } 1 - p_{\text{drop}}, \\ \varnothing, & \text{with probability } p_{\text{drop}}. \end{cases}$$

This allows a single network to learn both conditional and unconditional denoising. The loss is

$$\mathcal{L}_{\text{CFG}}(\theta) = \mathbb{E}_{x_0, \epsilon, t, \tilde{y}} \left[\left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon,\, t,\, \tilde{y}\right) \right\|^2 \right].$$

- x_t is the noisy input,
- t is the timestep embedding,
- $\tilde{y}$ is either the true condition or the null token.

Phase 2: Sampling Stage

Prediction at inference time. At each reverse step t, the same network is evaluated twice:

$$\epsilon_{\text{cond}} = \epsilon_\theta(x_t, t, y), \qquad \epsilon_{\text{uncond}} = \epsilon_\theta(x_t, t, \varnothing).$$

These two predictions are combined as

$$\epsilon_{\text{guided}} = \epsilon_{\text{uncond}} + \omega\left(\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}\right),$$

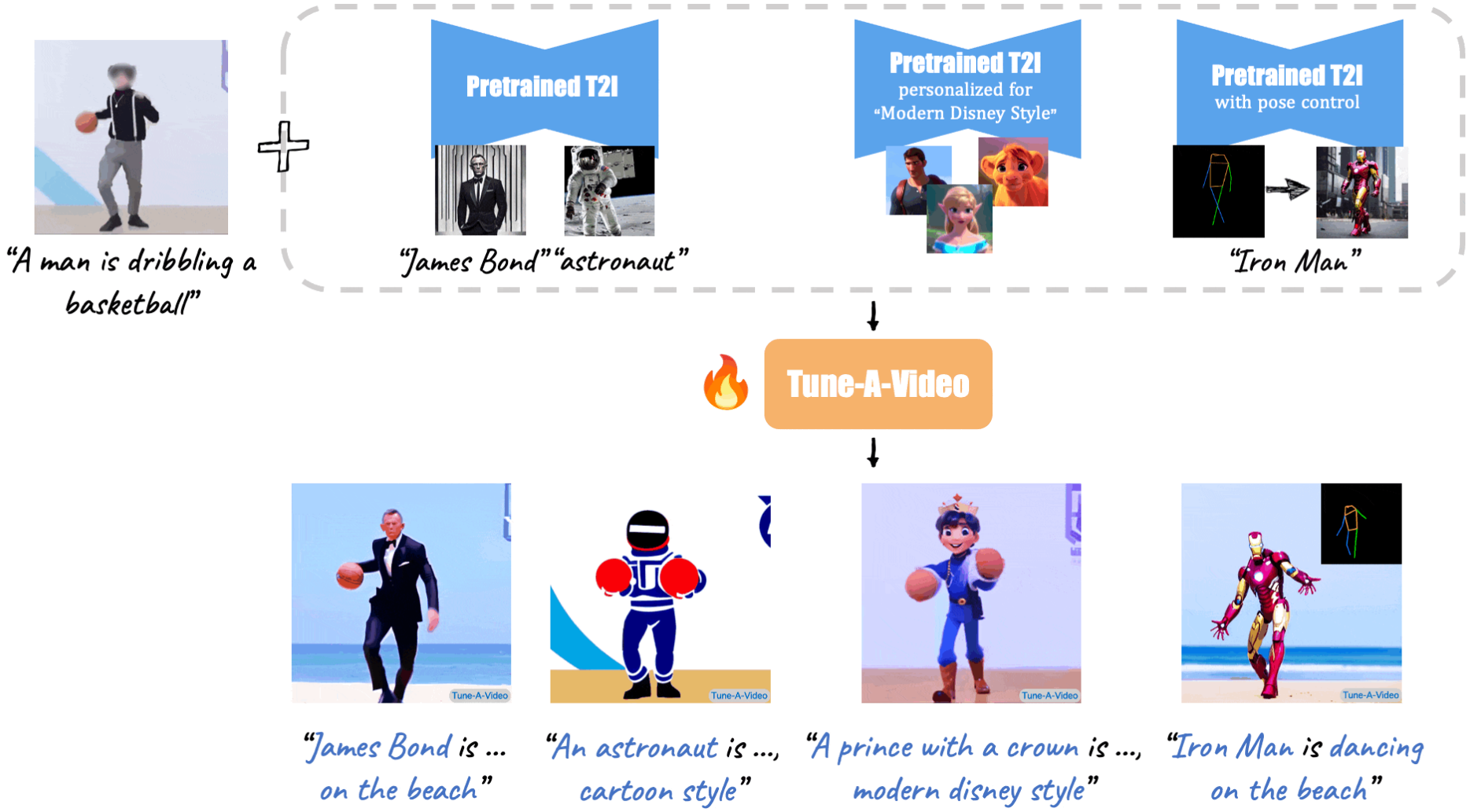
where ω is the guidance scale.

- Small ω: better realism and diversity, but weaker adherence to the condition
- Large ω: stronger adherence to the condition, but reduced diversity and more artifacts

$$x_{t-1} \sim \mathcal{N}(\mu_{\text{guided}}(x_t, t), \Sigma_t)\,.$$

$$\mu_{\text{guided}}(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\text{guided}} \right)$$

Video Generation



Try Yourself

DALL·E 3

Many diffusion models spaces in huggingface, try them yourself!