Diffusion Models

Oualid Missaoui

Agenda

- Overview
- Introduction
- VAE
- HVAE
- DDPM
- Score Matching Langevin Dynamics
- Stochastic Differential Equation (SDE) perspective to Diffusion Models
- Review
- References

Introduction: Mathematical Setup of Deep Generative Models (DGM)

Unknown true distribution

We cannot draw new samples from it directly

$$p_{\text{data}}(\mathbf{x})$$

$$\{\mathbf{x}^{(i)}\}_{i=1}^{N} \sim p_{\text{data}}$$

Model distribution

$p_{\phi}(\mathbf{x})$ ϕ The network's trainable parameters

Generative Model

Find ϕ^* that satisfies $p_{\phi^*}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x})$

Capability of DGM.

Once a proxy of the data distribution, $p_{\phi}(\mathbf{x})$, is available, we can:

- generate an arbitrary number of new data points using sampling methods such as Monte Carlo sampling from $p_{\phi}(\mathbf{x})$.
- Additionally, we can compute the probability (or likelihood) of any given data sample $\mathbf{x}'$ by evaluating $p_{\phi}(\mathbf{x}')$.

Introduction: Illustration of DGM

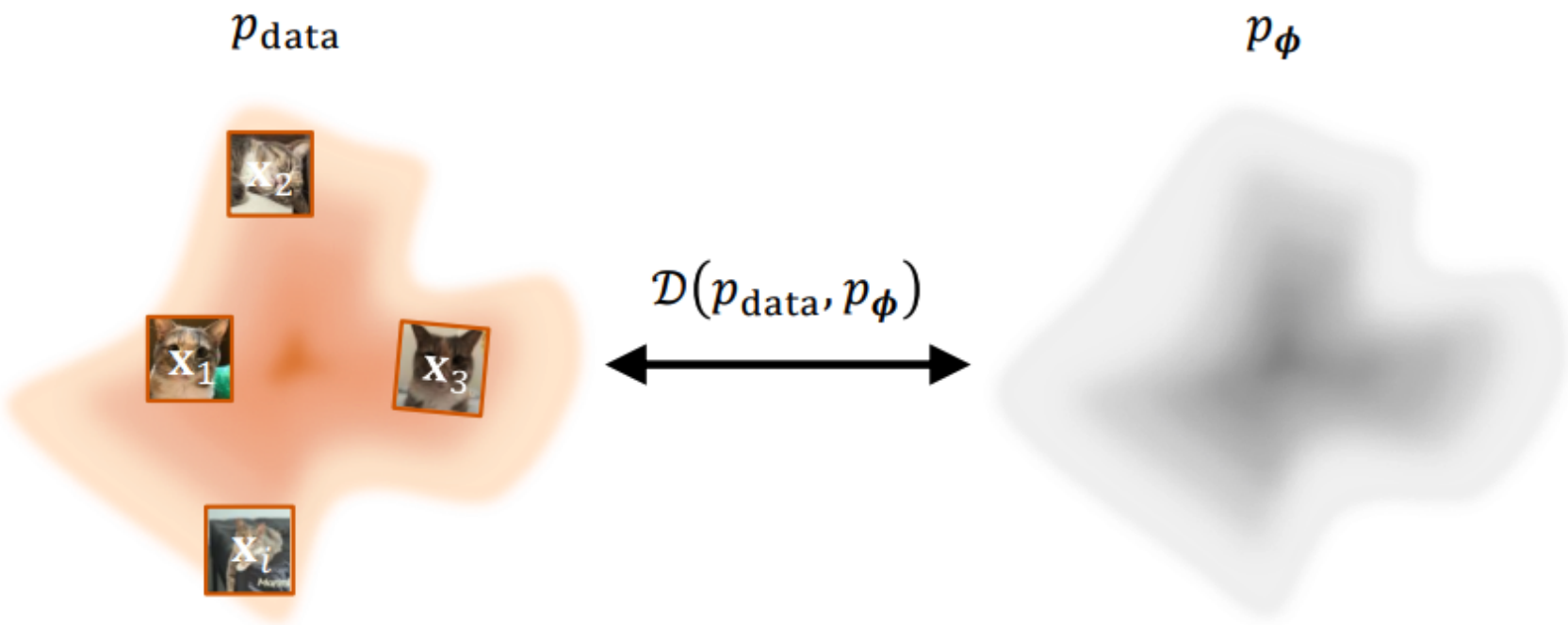
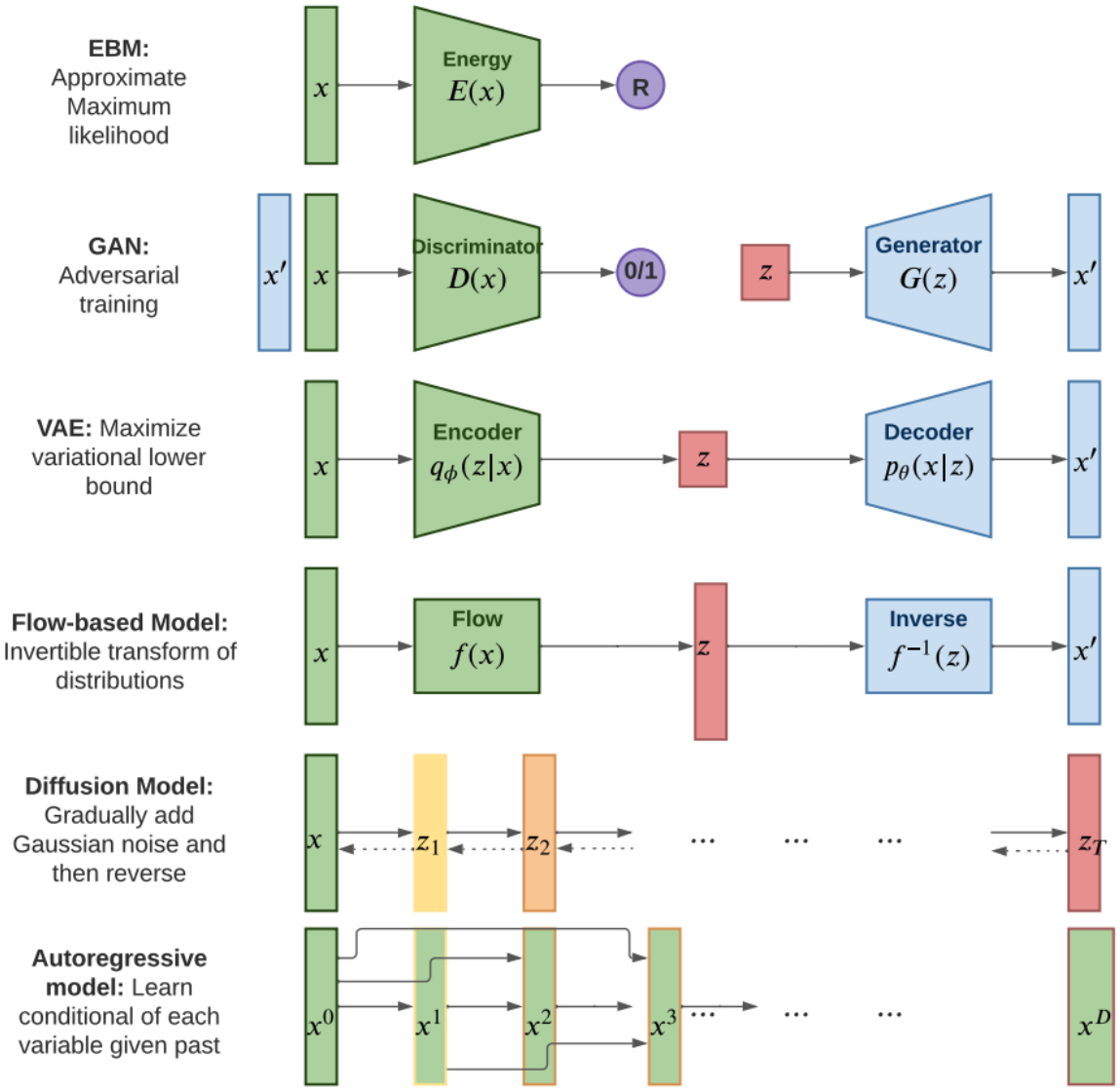


Figure 1.1: Illustration of the target in DGM. Training a DGM is essentially minimizing the discrepancy between the model distribution $p_{\boldsymbol{\phi}}$ and the unknown data distribution p_{data}. Since p_{data} is not directly accessible, this discrepancy must be estimated efficiently using a finite set of independent and identically distributed (i.i.d.) samples, $\mathbf{x}_i$, drawn from it.

Introduction

There are two approaches to fitting $P(x, y)$ to data $\mathcal{D}$:

- Generative: $p(x, y) = p(x|y)p(y)$
  - prior information about the structure of the data
  - finding a suitable generative data model is a difficult task
- Discriminative: $p(x, y) = p(y|x)p(x)$
  - directly addresses finding an accurate classifier based on modeling the decision boundary
  - usually trained as "block-box" classifier with little prior knowledge built.



Source: Kevin Murphy, 2023, Probabilistic Machine learning: Advanced topics (part IV: Generation)

Introduction

Table 1.1: Comparison of Explicit and Implicit Generative Models

	Explicit		Implicit
	Exact Likelihood	**Approx. Likelihood**	
Likelihood	Tractable	Bound/Approx.	Not Directly Modeled/ Intractable
Objective	MLE	ELBO	Adversarial
Examples	NFs, ARs	VAEs, DMs	GANs

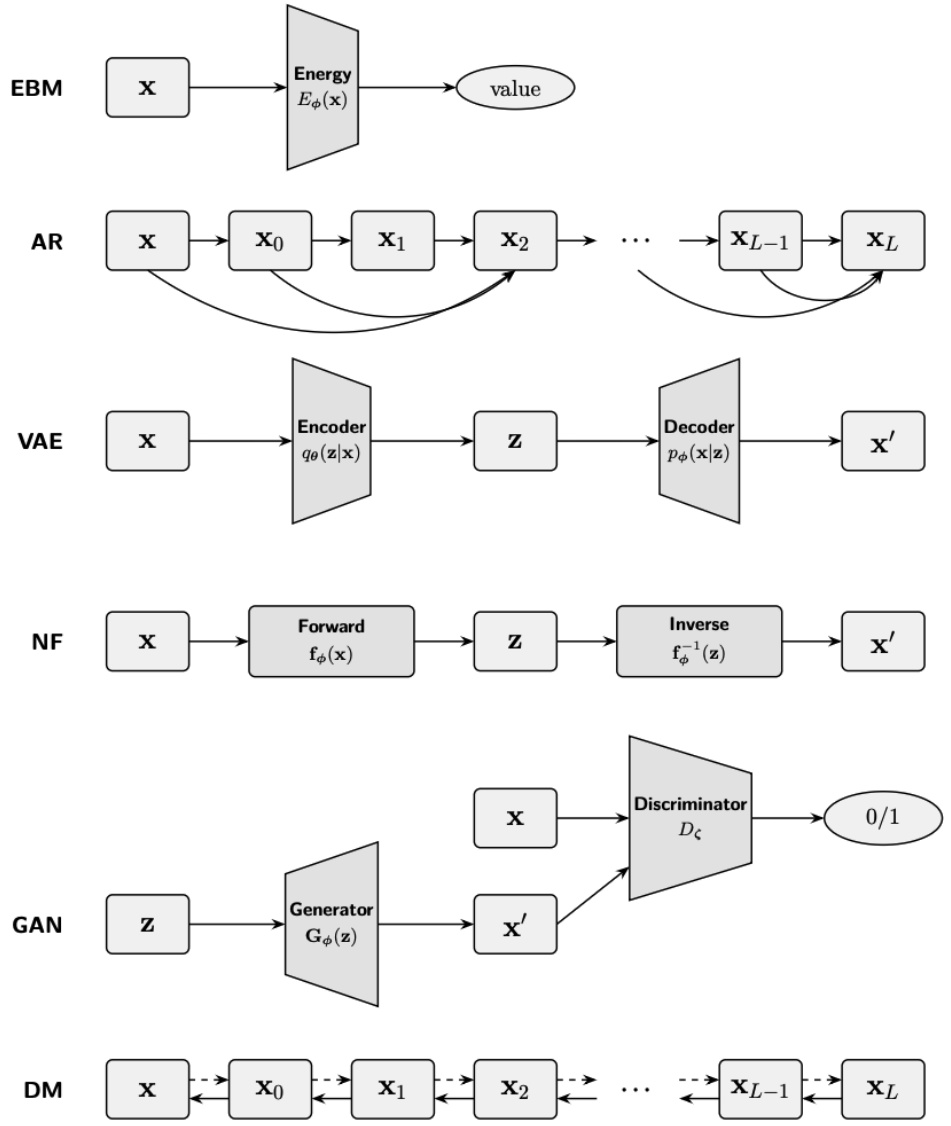
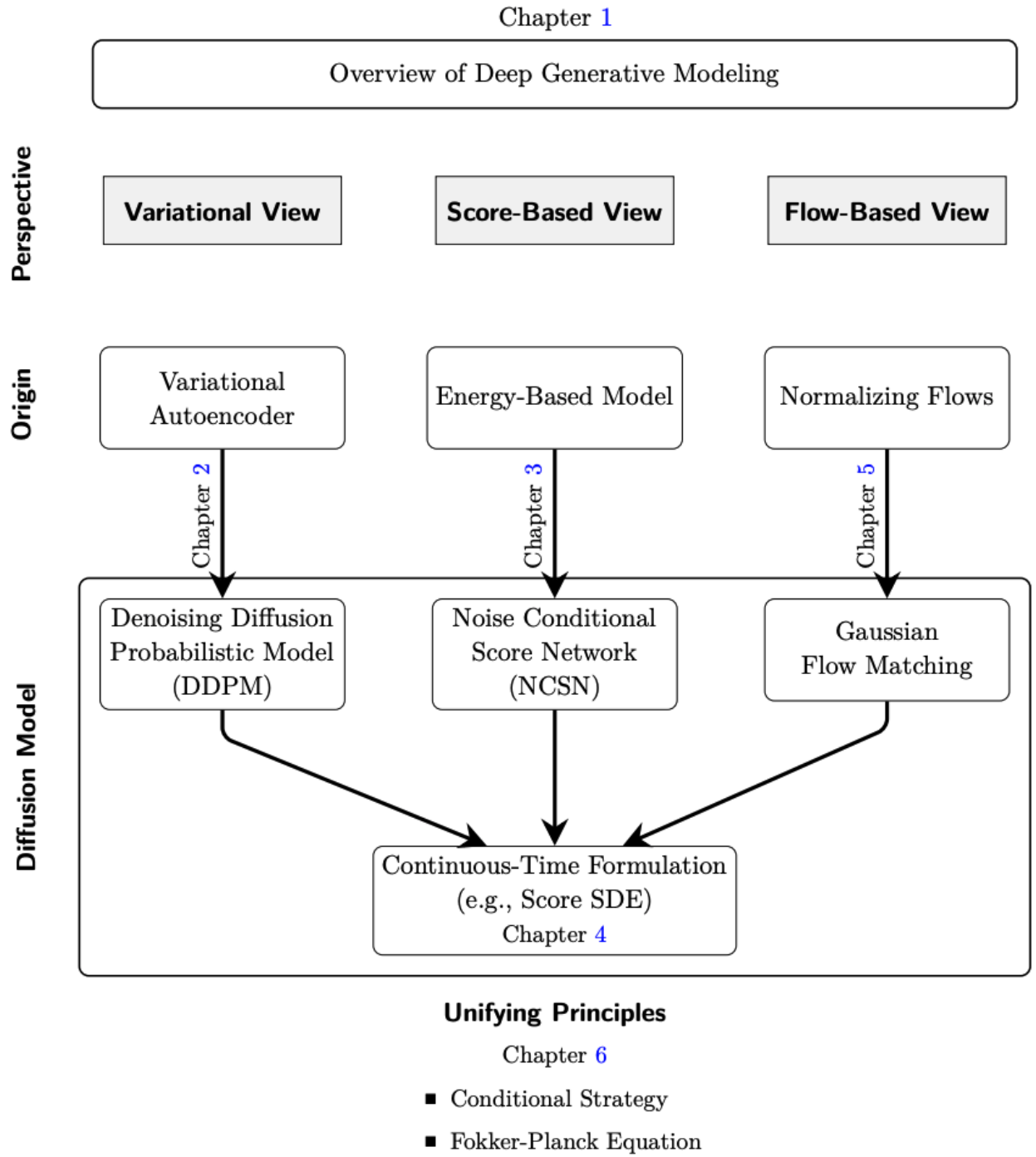


Figure 1.2: Computation graphs of prominent deep generative models. Top to bottom: **EBM** maps an input $\mathbf{x}$ to a scalar energy; **AR** generates a sequence $\{\mathbf{x}_\ell\}$ left to right with causal dependencies; **VAE** encodes $\mathbf{x}$ to a latent $\mathbf{z}$ and decodes to a reconstruction $\mathbf{x}'$; **NF** applies an invertible map $\mathbf{f}_\phi$ between $\mathbf{x}$ and $\mathbf{z}$ and uses $\mathbf{f}_\phi^{-1}$ to produce $\mathbf{x}'$; **GAN** transforms noise $\mathbf{z}$ to a sample $\mathbf{x}'$ that is judged against real $\mathbf{x}$ by a discriminator D_ς; **DM** iteratively refines a noisy sample through a multi-step denoising chain $\{\mathbf{x}_\ell\}$. Boxes denote variables, trapezoids are learnable networks, ovals are scalars; arrows indicate computation flow.

Chieh-Hsin Lai et al (2025), The Principles of Diffusion Models


Chapter 1
Overview of Deep Generative Modeling
Perspective
Variational View
Score-Based View
Flow-Based View
Origin
Variational Autoencoder
Energy-Based Model
Normalizing Flows
Chapter 2
Chapter 3
Chapter 5
Diffusion Model
Denoising Diffusion Probabilistic Model (DDPM)
Noise Conditional Score Network (NCSN)
Gaussian Flow Matching
Continuous-Time Formulation (e.g., Score SDE)
Chapter 4
Unifying Principles
Chapter 6
Conditional Strategy
Fokker-Planck Equation

Introduction

- **Common viewpoint:** Treat diffusion models as part of the *variational* family of generative models.
- **VAEs:**
  - Represent data via latent variables.
  - Learn an **encoder** (data → latents) and **decoder** (latents → data) by maximizing a tractable lower bound on log-likelihood (ELBO).
- **Hierarchical VAEs:**
  - Stack multiple latent layers to capture structure at different scales.
- **Diffusion models (DDPMs):**
  - Reuse the encoder–decoder template:
    - **Encoder** is fixed as a forward noising process (data → noise).
    - **Decoder** is learned as a reverse denoising process (noise → data) over many steps.
- **Unifying idea:** VAEs, hierarchical VAEs, and diffusion models all optimize a likelihood surrogate given by a variational bound, providing a shared foundation for the methods in this chapter.

Training of DGM

We learn parameters ϕ of a model family $\{p_\phi\}$ by minimizing a discrepancy $\mathcal{D}(p_{\text{data}}, p_\phi)$:

$$\phi^* \in \arg\min_\phi \mathcal{D}(p_{\text{data}}, p_\phi).$$

- Because p_{data} is unknown, a practical choice of $\mathcal{D}$ must admit efficient estimation from i.i.d. samples from p_{data}.
- With sufficient capacity, p_{ϕ^*} can closely approximate p_{data}.

The discrepancy can be:

- Forward KL and Maximum Likelihood Estimation (MLE)
- Fisher Divergence
- Beyond KL: f-divergences

Forward KL and Maximum Likelihood Estimation (MLE)

A standard choice for the discrepancy between the data distribution and a parametric model p_ϕ is the (forward) Kullback–Leibler divergence:

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_\phi) &:= \int p_{\mathrm{data}}(\mathbf{x}) \, \log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_\phi(\mathbf{x})} \, d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}\sim p_{\mathrm{data}}}\Big[\log p_{\mathrm{data}}(\mathbf{x}) - \log p_\phi(\mathbf{x}) \Big]. \qquad (1)\end{aligned}$$

It is **asymmetric**:

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_\phi) \neq \mathcal{D}_{\mathrm{KL}}(p_\phi \,\|\, p_{\mathrm{data}}). \qquad (2)$$

Mode covering (intuition)

If there exists a set A with positive p_{data}-mass where $p_\phi(\mathbf{x}) = 0$ for $\mathbf{x} \in A$, then the integrand in (1) contains $\log\big(p_{\mathrm{data}}(\mathbf{x})/0\big) = +\infty$ on A. Hence

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_\phi) = +\infty,$$

so minimizing forward KL **forces** the model to assign nonzero probability wherever the data has support—i.e., it encourages *mode covering*.

Decomposing the forward KL

Start from (1):

$$\begin{aligned}\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_\phi) &= \mathbb{E}_{p_{\mathrm{data}}}[\log p_{\mathrm{data}}(\mathbf{x})] - \mathbb{E}_{p_{\mathrm{data}}}[\log p_\phi(\mathbf{x})] \\ &= -\,\mathbb{E}_{p_{\mathrm{data}}}[\log p_\phi(\mathbf{x})] \;-\; \mathcal{H}(p_{\mathrm{data}}),\end{aligned} \tag{3}$$

where

$$\mathcal{H}(p_{\mathrm{data}}) := -\,\mathbb{E}_{p_{\mathrm{data}}}[\log p_{\mathrm{data}}(\mathbf{x})]$$

is the (Shannon) entropy of the data distribution and **does not depend on ϕ**.

Equation (3) also shows

$$\mathcal{D}_{\mathrm{KL}}(p_{\mathrm{data}} \,\|\, p_\phi) = \underbrace{-\,\mathbb{E}_{p_{\mathrm{data}}}[\log p_\phi(\mathbf{x})]}_{\text{cross-entropy } \mathcal{H}(p_{\mathrm{data}}, p_\phi)} - \underbrace{\left(-\,\mathbb{E}_{p_{\mathrm{data}}}[\log p_{\mathrm{data}}(\mathbf{x})]\right)}_{\text{entropy } \mathcal{H}(p_{\mathrm{data}})}.$$

Minimizing KL

Because $\mathcal{H}(p_{\text{data}})$ is constant in ϕ, minimizing the forward KL is equivalent to maximizing the expected log-likelihood under the data:

$$\boxed{\min_{\phi} \mathcal{D}_{\text{KL}}(p_{\text{data}} \,\|\, p_\phi) \iff \max_{\phi} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_\phi(\mathbf{x})].} \tag{4}$$

This is precisely **maximum likelihood estimation (MLE)** at the population level.

$$\phi^* \in \arg\min_{\phi} \mathcal{D}_{\text{KL}}(p_{\text{data}}, p_\phi) \iff \phi^* \in \arg\max_{\phi} \mathbb{E}_{p_{\text{data}}}\big[\log p_\phi(\mathbf{x})\big].$$

Variational Autoencoder

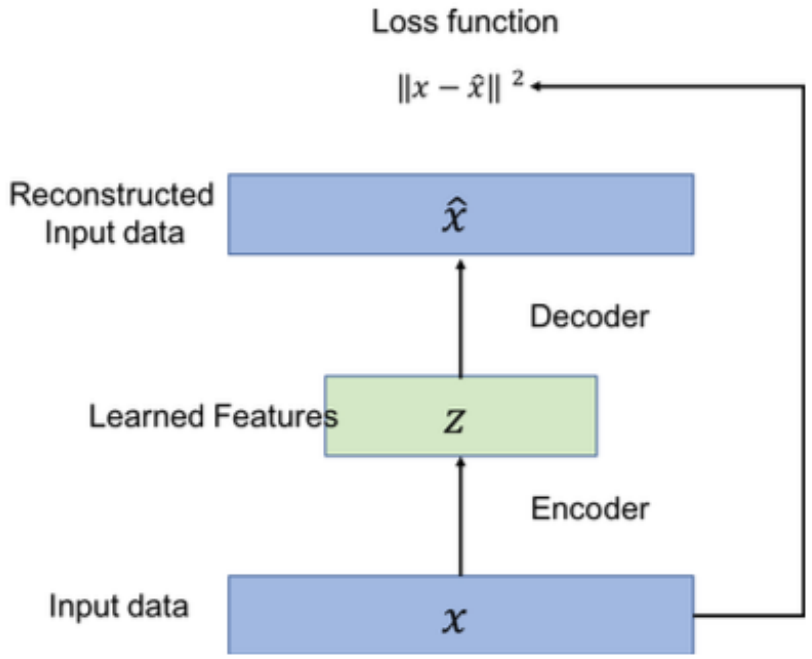


Figure: Autoencoder for feature learning

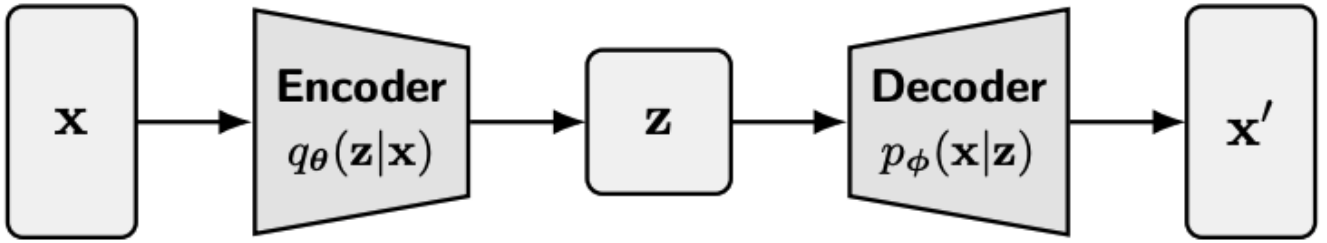


Figure 2.1: Illustration of a VAE. It consists of a stochastic encoder $q_\theta(\mathbf{z}|\mathbf{x})$ that maps data $\mathbf{x}$ to a latent variable $\mathbf{z}$, and a decoder $p_\phi(\mathbf{x}|\mathbf{z})$ that reconstructs data from the latent.

Formulation

Decoder / Generator $p_\phi(x \mid z)$	Encoder / Inference Network $q_\theta(z \mid x)$
Variables: observed data x, latent variables z capturing hidden factors (shape, color, style, ...).	**Goal**: given an observation x, infer which latent codes (z) could have produced it.
Prior on latents: sample from a simple prior, typically $$p_{\text{prior}}(z) = \mathcal{N}(0, I).$$	**True posterior (intractable)**: $$p_\phi(z \mid x) = \frac{p_\phi(x \mid z)p(z)}{p_\phi(x)}, \quad p_\phi(x) = \int p_\phi(x \mid z)p(z)\, dz.$$
Generative model: map latent z to data via a decoder distribution $$p_\phi(x \mid z).$$ Often chosen simple (e.g., factorized Gaussian) so learning focuses on useful features rather than memorizing data.	**Problem**: computing $p_\phi(z \mid x)$ requires the marginal likelihood $p_\phi(x)$, an integral over all z that is intractable for expressive nonlinear decoders.
Sampling / generation: 1) draw $z \sim p_{\text{prior}}$; 2) decode to data $$x \sim p_\phi(x \mid z).$$ Latent z is a compact code that makes generation easier than predicting pixels directly.	**Variational solution**: introduce a tractable approximation $$q_\theta(z \mid x) \approx p_\phi(z \mid x),$$ implemented by a neural network. This encoder maps each x to a distribution over latent codes, providing a trainable path from data space back to latent space.

Training

Theorem 2.1.1: Evidence Lower Bound (ELBO)

For any data point $\mathbf{x}$, the log-likelihood satisfies:

$$\log p_{\boldsymbol{\phi}}(\mathbf{x}) \geq \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}),$$

where the ELBO is given by:

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})\right]}_{\text{Reconstruction Term}} - \underbrace{\mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right)}_{\text{Latent Regularization}}. \tag{2.1.1}$$

$$D_{\text{KL}}\left(q_\theta(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right) = \mathbb{E}_{\mathbf{z}\sim q_\theta(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\theta(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})}\right]$$

$$= \int q_\theta(\mathbf{z} \mid \mathbf{x}) \log \frac{q_\theta(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})}\, d\mathbf{z}.$$

Proof for Theorem.

The ELBO arises from Jensen's inequality:

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

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

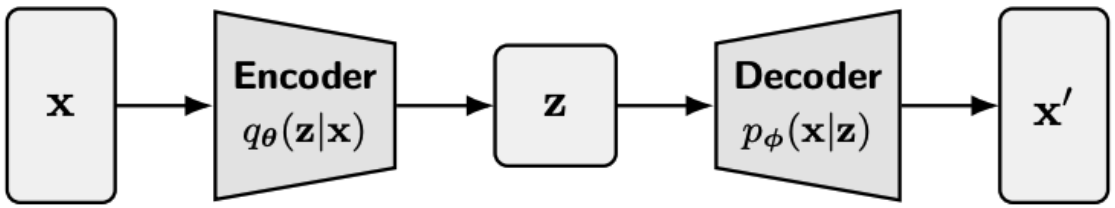


Figure 2.1: Illustration of a VAE. It consists of a stochastic encoder $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ that maps data $\mathbf{x}$ to a latent variable $\mathbf{z}$, and a decoder $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$ that reconstructs data from the latent.

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})\right]}_{\text{Reconstruction Term}} - \underbrace{\mathcal{D}_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right)}_{\text{Latent Regularization}}$$

- **Reconstruction:** Encourages accurate recovery of $\mathbf{x}$ from its latent code $\mathbf{z}$. With Gaussian encoder and decoder assumptions, this term reduces exactly to the familiar reconstruction loss of an autoencoder (cf. Section 2.1.3). However, as in autoencoders, optimizing this term alone risks memorizing the training data, motivating an additional regularization.

- **Latent KL:** Encourages the encoder distribution $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ to stay close to a simple Gaussian prior $p_{\text{prior}}(\mathbf{z})$. This regularization shapes the latent space into a smooth and continuous structure, enabling meaningful generation by ensuring that samples drawn from the prior can be reliably decoded.

General chain rule for KL Divergence

Let $q(x, z)$ and $p(x, z)$ be any two joint distributions, with marginals $q(x), p(x)$ and conditionals $q(z \mid x), p(z \mid x)$.

Start from the definition

$$D_{\mathrm{KL}}(q(x, z) \,\|\, p(x, z)) = \mathbb{E}_{q(x,z)} \left[\log \frac{q(x, z)}{p(x, z)} \right].$$

Factor joints into marginal $\times$ conditional:

$$q(x, z) = q(x)q(z \mid x), \qquad p(x, z) = p(x)p(z \mid x).$$

Then

$$\begin{aligned} D_{\mathrm{KL}}(q(x, z) \,\|\, p(x, z)) &= \mathbb{E}_{q(x,z)} \left[\log \frac{q(x)q(z \mid x)}{p(x)p(z \mid x)} \right] \\ &= \mathbb{E}_{q(x,z)} \left[\log \frac{q(x)}{p(x)} \right] + \mathbb{E}_{q(x,z)} \left[\log \frac{q(z \mid x)}{p(z \mid x)} \right]. \end{aligned}$$

Treat each term separately.

First term

$$\mathbb{E}_{q(x,z)} \left[\log \frac{q(x)}{p(x)} \right] = \mathbb{E}_{q(x)} \left[\log \frac{q(x)}{p(x)} \right] = D_{\mathrm{KL}}(q(x) \,\|\, p(x)).$$

Second term

$$\begin{aligned} \mathbb{E}_{q(x,z)} \left[\log \frac{q(z \mid x)}{p(z \mid x)} \right] &= \mathbb{E}_{q(x)} \left[\mathbb{E}_{q(z|x)} \left[\log \frac{q(z \mid x)}{p(z \mid x)} \right] \right] \\ &= \mathbb{E}_{q(x)} \Big[D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z \mid x)) \Big]. \end{aligned}$$

So we get the exact decomposition

$$\boxed{D_{\mathrm{KL}}(q(x, z) \,\|\, p(x, z)) = D_{\mathrm{KL}}(q(x) \,\|\, p(x)) + \mathbb{E}_{q(x)} \big[D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z \mid x)) \big].}$$

Since KL divergences are always non-negative,

$$\mathbb{E}_{q(x)} \big[D_{\mathrm{KL}}(q(z \mid x) \,\|\, p(z \mid x)) \big] \;\geq\; 0,$$

hence

$$\boxed{D_{\mathrm{KL}}(q(x) \,\|\, p(x)) \;\leq\; D_{\mathrm{KL}}(q(x, z) \,\|\, p(x, z)).}$$

ELBO as an Information-Theoretic Divergence Bound

- In maximum likelihood, we want the model $p_\phi(x)$ to match the data:
  - This is equivalent to **minimizing**

$$D_{\mathrm{KL}}\big(p_{\mathrm{data}}(x) \,\|\, p_\phi(x)\big).$$

- Variational inference introduces **joint distributions** over (x, z):
  - **Generative joint** (model):

$$p_\phi(x, z) = p(z)\, p_\phi(x \mid z).$$

  - **Inference joint** (data + encoder):

$$q_\theta(x, z) = p_{\mathrm{data}}(x)\, q_\theta(z \mid x).$$

- Chain rule for KL divergence:

$$D_{\mathrm{KL}}\big(p_{\mathrm{data}}(x) \,\|\, p_\phi(x)\big) \;\le\; D_{\mathrm{KL}}\big(q_\theta(x, z) \,\|\, p_\phi(x, z)\big).$$

- **Intuition:**
  Comparing only the marginals in x can hide mismatches.
  Once we also compare the latent variables z, the mismatch can only increase, giving a **tighter "total error" bound** on model fit.

Decomposing the Joint KL: Modeling vs. Inference Error

We can expand the joint KL as

$$D_{\mathrm{KL}}\big(q_\theta(x,z)\,\|\,p_\phi(x,z)\big) = \mathbb{E}_{q_\theta(x,z)}\left[\log \frac{p_{\mathrm{data}}(x)\,q_\theta(z\mid x)}{p_\phi(x)\,p_\phi(z\mid x)}\right].$$

Rewriting the fraction inside the log:

$$\begin{aligned} D_{\mathrm{KL}}\big(q_\theta(x,z)\,\|\,p_\phi(x,z)\big) &= \mathbb{E}_{p_{\mathrm{data}}(x)}\left[\log \frac{p_{\mathrm{data}}(x)}{p_\phi(x)} + D_{\mathrm{KL}}\big(q_\theta(z\mid x)\,\|\,p_\phi(z\mid x)\big)\right] \\ &= D_{\mathrm{KL}}\big(p_{\mathrm{data}}\,\|\,p_\phi\big) + \mathbb{E}_{p_{\mathrm{data}}(x)}[D_{\mathrm{KL}}\big(q_\theta(z\mid x)\,\|\,p_\phi(z\mid x)\big)]. \end{aligned}$$

- **True modeling error:**
  $D_{\mathrm{KL}}\big(p_{\mathrm{data}}\,\|\,p_\phi\big)$
  measures how well the generative model fits the data *in principle*.

- **Inference error:**
  $\mathbb{E}_{p_{\mathrm{data}}(x)}[D_{\mathrm{KL}}(q_\theta(z\mid x)\,\|\,p_\phi(z\mid x))]$
  is the gap between the **approximate posterior** and the **true posterior**.

Because the inference term is always ≥ 0, we recover the inequality

$$D_{\mathrm{KL}}\big(p_{\mathrm{data}}(x)\,\|\,p_\phi(x)\big) \;\leq\; D_{\mathrm{KL}}\big(q_\theta(x,z)\,\|\,p_\phi(x,z)\big).$$

Moreover, for each x,

$$\log p_\phi(x) - \mathcal{L}_{\mathrm{ELBO}}(\theta,\phi;x) = D_{\mathrm{KL}}\big(q_\theta(z\mid x)\,\|\,p_\phi(z\mid x)\big),$$

so **maximizing the ELBO** directly **reduces inference error**, tightening the gap between the log-likelihood and its variational lower bound.

Gaussian VAE

Encoder: $q_\theta(z \mid x)$

- Approximate posterior over latent code z given input x
- Modeled as a Gaussian with mean and (diagonal) variance from an encoder network

$$q_\theta(z \mid x) = \mathcal{N}\big(z;\, \mu_\theta(x),\, \mathrm{diag}(\sigma_\theta^2(x))\big),$$

- $\mu_\theta : \mathbb{R}^D \to \mathbb{R}^d$
- $\sigma_\theta : \mathbb{R}^D \to \mathbb{R}^d_+$

Decoder: $p_\phi(x \mid z)$

- Likelihood (reconstruction model) of data x given latent code z
- Gaussian with **fixed** variance $\sigma^2 I$

$$p_\phi(x \mid z) = \mathcal{N}\big(x;\, \mu_\phi(z),\, \sigma^2 I\big),$$

- $\mu_\phi : \mathbb{R}^d \to \mathbb{R}^D$ is a neural network
- $\sigma > 0$ is a small constant controlling observation noise

ELBO under Gaussian decoder (fixed variance)

Using the Gaussian decoder, the reconstruction term simplifies to

$$\mathbb{E}_{q_\theta(z|x)}\big[\log p_\phi(x \mid z)\big] = -\frac{1}{2\sigma^2}\mathbb{E}_{q_\theta(z|x)}\big[\|x - \mu_\phi(z)\|^2\big] + C,$$

where C is constant w.r.t. θ, ϕ.

So the **negative ELBO** (training loss) becomes

$$\min_{\theta,\phi}\ \mathbb{E}_{q_\theta(z|x)}\left[\frac{1}{2\sigma^2}\|x - \mu_\phi(z)\|^2\right] + D_{\mathrm{KL}}\big(q_\theta(z \mid x)\,\|\,p_{\mathrm{prior}}(z)\big),$$

i.e. **regularized reconstruction loss**: squared error term + KL term (closed-form for Gaussians).

Drawbacks of Standard VAE (1)

> *Even if you train the VAE perfectly, with a Gaussian decoder + MSE it is **structurally** forced to output averages of training examples that land in the same latent region → blur is not an optimization bug, it's what the objective wants.*

1. What are we optimizing?

Decoder:

$$p_{\text{dec}}(x \mid z) = \mathcal{N}(x; \mu(z), \sigma^2 I),$$

with learnable network $\mu(z)$.

Fix some encoder $q_{\text{enc}}(z \mid x)$ (any Gaussian network).

The ELBO for a VAE has two terms: a reconstruction term and a KL term.
For a fixed encoder and a Gaussian decoder with **fixed** σ^2, the KL term does not depend on μ, so optimizing w.r.t. μ is equivalent to minimizing the expected squared error:

$$\arg\min_{\mu} \mathbb{E}_{p_{\text{data}}(x)\, q_{\text{enc}}(z|x)} \left[\|x - \mu(z)\|^2 \right].$$

So: **for each** z, $\mu(z)$ is chosen to be the best least–squares prediction of x given z.

Drawbacks of Standard VAE (2)

2. Least squares ⇒ conditional mean

Given any joint over (x, z), the minimizer of

$$\mathbb{E}\left[\|x - f(z)\|^2\right]$$

over functions f is

$$f^*(z) = \mathbb{E}[x \mid z].$$

Apply this to our joint

$$p_{\text{joint}}(x, z) = p_{\text{data}}(x)\, q_{\text{enc}}(z \mid x).$$

Then the optimal decoder is

$$\mu^*(z) = \mathbb{E}_{q_{\text{enc}}(x|z)}[x],$$

i.e. *the conditional mean of data x given latent z*, under the encoder-induced posterior $q_{\text{enc}}(x \mid z)$.

So far: **VAE + Gaussian decoder + MSE ⇒ decoder learns $\mathbb{E}[x \mid z]$.**

Drawbacks of Standard VAE (3)

3. "Bayesian" rewrite: μ*(z) is a weighted average of all data points

They then write Bayes' rule for the *encoder-induced* posterior over inputs:

$$q_{\text{enc}}(x \mid z) = \frac{q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x)}{p_{\text{prior}}(z)}$$

(where $p_{\text{prior}}(z)$ is the marginal over z under the same joint).

If you expand $\mathbb{E}[x \mid z]$ using this,

$$\mu^*(z) = \sum_x x\, q_{\text{enc}}(x \mid z) = \frac{\mathbb{E}_{p_{\text{data}}(x)}[q_{\text{enc}}(z \mid x)\, x]}{\mathbb{E}_{p_{\text{data}}(x)}[q_{\text{enc}}(z \mid x)]}.$$

Interpretation:

- Think over the whole dataset.
- Each training point x "votes" for latent z with weight $q_{\text{enc}}(z \mid x)$ (how likely the encoder maps x to z).
- The optimal $\mu^*(z)$ is **the weighted average of all data points, with those weights.**

So for any fixed z, you're literally averaging training examples that "live near" that z according to the encoder.

By Bayes:

$$q_{\text{enc}}(x \mid z) = \frac{q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x)}{p_{\text{prior}}(z)},$$

where

$$p_{\text{prior}}(z) = \sum_x q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x)$$

is the marginal over z induced by the encoder and the data distribution.

So

$$\begin{aligned}\mu^*(z) &= \sum_x x\, q_{\text{enc}}(x \mid z) \\ &= \sum_x x \frac{q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x)}{p_{\text{prior}}(z)} \\ &= \frac{1}{p_{\text{prior}}(z)} \sum_x x\, q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x).\end{aligned}$$

Recall that for any function $f(x)$,

$$\mathbb{E}_{p_{\text{data}}(x)}[f(x)] = \sum_x p_{\text{data}}(x) f(x).$$

Apply this to the numerator and denominator:

- Numerator:

$$\sum_x x\, q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x) = \mathbb{E}_{p_{\text{data}}(x)}[\, q_{\text{enc}}(z \mid x)\, x\,].$$

- Denominator:

$$p_{\text{prior}}(z) = \sum_x q_{\text{enc}}(z \mid x)\, p_{\text{data}}(x) = \mathbb{E}_{p_{\text{data}}(x)}[\, q_{\text{enc}}(z \mid x)\,].$$

Putting it together:

$$\mu^*(z) = \frac{\mathbb{E}_{p_{\text{data}}(x)}[\, q_{\text{enc}}(z \mid x)\, x\,]}{\mathbb{E}_{p_{\text{data}}(x)}[\, q_{\text{enc}}(z \mid x)\,]}.$$

Hierarchical VAEs

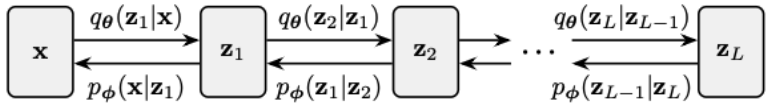


Figure 2.2: Computation graph of the HVAE. It has a hierarchical structure with stacked, trainable encoders and decoders across multiple latent layers.

- **Flat VAE limitation 1 – variational family**
  - Standard posterior:
    $q_\theta(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_\theta(\mathbf{x}), \mathrm{diag}(\sigma^2_\theta(\mathbf{x})))$ → for each $\mathbf{x}$, a *single unimodal Gaussian ellipsoid*.
  - If true $p_\phi(\mathbf{z} \mid \mathbf{x})$ is multi-peaked, this family cannot match it → **loose ELBO, weak inference**.
- **Flat VAE limitation 2 – posterior collapse**
  - ELBO objective can be rewritten as
    $\mathbb{E}[\log p_\phi(\mathbf{x} \mid \mathbf{z})] - I_q(\mathbf{x}; \mathbf{z}) - D_{\mathrm{KL}}(q_\theta(\mathbf{z}) \| p(\mathbf{z}))$.
  - Powerful decoders can model data *without* using $\mathbf{z}$: optimum with $q_\theta(\mathbf{z} \mid \mathbf{x}) = p(\mathbf{z})$, $I_q(\mathbf{x}; \mathbf{z}) = 0$ → **code carries no information, no controllable generation**.
- **HVAE idea**
  - Introduce **multiple latent levels** $\mathbf{z}_L \to \cdots \to \mathbf{z}_1 \to \mathbf{x}$ with

$$p_\phi(\mathbf{x}, \mathbf{z}_{1:L}) = p_\phi(\mathbf{x} \mid \mathbf{z}_1) \prod_{i=2}^{L} p_\phi(\mathbf{z}_{i-1} \mid \mathbf{z}_i) p(\mathbf{z}_L).$$

  - Matching bottom-up encoder
    $q_\theta(\mathbf{z}_{1:L} \mid \mathbf{x}) = q_\theta(\mathbf{z}_1 \mid \mathbf{x}) \prod_{i=2}^{L} q_\theta(\mathbf{z}_i \mid \mathbf{z}_{i-1})$.
  - ELBO decomposes into **reconstruction + a sum of local KLs** between adjacent levels → distributes information cost, encourages:
    - top latents = coarse/global structure
    - lower latents = finer/local details
      → more structured, progressive generation than flat VAE.

Why HVAEs Still Fall Short: Opening the Door to DDPM

- **Still ELBO + latent-code–centric**
  - Training is tied to a variational lower bound; mismatch between simple Gaussian priors/conditionals and complex data remains.
  - Optimization can be brittle: top levels under-used, lower levels absorb most information; some collapse modes persist.
- **Likelihood vs sample-quality trade-off**
  - Strong likelihood often corresponds to **blurry samples**; sharpening the decoder can re-introduce posterior collapse.
  - Ancestral sampling through many latent layers amplifies small modeling errors.
- **Representation bottleneck in high dimensions**
  - Even with hierarchy, we must infer a global latent path $\mathbf{z}_{1:L}$ for each $\mathbf{x}$.
  - Approximating all multi-modal posteriors $p(\mathbf{z}_{1:L} \mid \mathbf{x})$ with tractable Gaussians remains hard for natural images and complex data.
- **DDPM / score-based models as next step**
  - Keep the **progressive, coarse-to-fine generation** idea, but:
    - **Drop global latent codes and ELBO.**
    - Work directly in data space with a simple noise prior (Gaussian) and a **Markov noising–denoising process**.
    - Learn **score functions** $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ at many noise levels instead of posteriors $q(\mathbf{z} \mid \mathbf{x})$.
  - This shift avoids variational bottlenecks and posterior collapse, giving models (DDPMs) that better cover complex high-dimensional densities with sharp, diverse samples.

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

Abstract

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at https://github.com/hojonathanho/diffusion.

Jonathan Ho, Ajay Jain, Pieter Abbeel (2020), Denoising Diffusion Probabilistic Models

From VAE to DDPM: A Guided Intuition

VAE

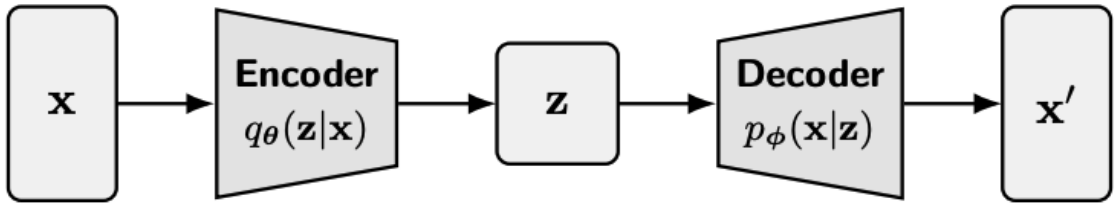


Figure 2.1: Illustration of a VAE. It consists of a stochastic encoder $q_\theta(\mathbf{z}|\mathbf{x})$ that maps data $\mathbf{x}$ to a latent variable $\mathbf{z}$, and a decoder $p_\phi(\mathbf{x}|\mathbf{z})$ that reconstructs data from the latent.

HVAE

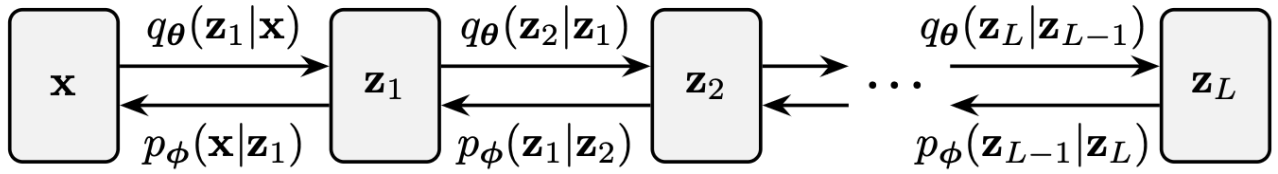


Figure 2.2: Computation graph of the HVAE. It has a hierarchical structure with stacked, trainable encoders and decoders across multiple latent layers.

DDPM

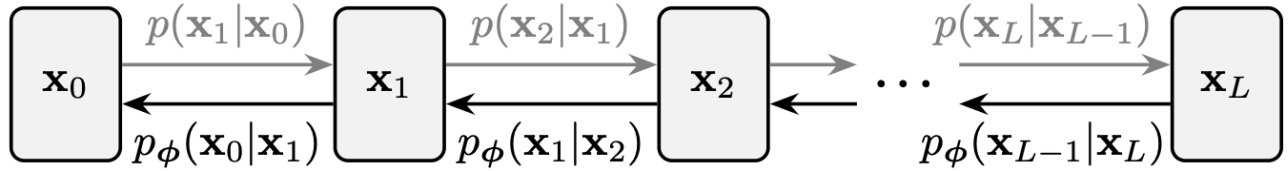


Figure 2.3: Illustration of DDPM. It consists of a fixed forward process (in gray) that gradually adds Gaussian noise to the data, and a learned reverse process that denoises step-by-step to generate new samples.

From VAE to DDPM: A Guided Intuition

Part 1 — The VAE Foundation

Step 1 — The VAE setup

- A VAE has a *learned encoder* $q_\phi(z|x)$ and a *learned decoder* $p_\theta(x|z)$
- The encoder maps data to a latent space; the decoder reconstructs from it
- Key constraint: the latent z is encouraged to be normally distributed, i.e., $z \sim \mathcal{N}(0, I)$

Step 2 — A core intuition

- Since we're doing self-supervised learning (input = reconstruction target), the decoder essentially *reverses* what the encoder did
- The encoder "destroys" structure and maps toward noise; the decoder "undoes" that destruction

Part 2 — The Hierarchical VAE Extension

Step 3 — A natural question about expressiveness

- A single latent bottleneck z may be too restrictive
- What if we allowed *multiple* layers of latent variables, each capturing structure at different scales?

Step 4 — Introduce the Hierarchical VAE

- Instead of $x \to z \to \hat{x}$, we now have a chain: $x \to z_1 \to z_2 \to \cdots \to z_T$
- Both encoder and decoder are still *learned*, but they operate through multiple stochastic layers
- The top-level latent z_T is encouraged to be $\mathcal{N}(0, I)$; intermediate latents carry progressively more structure

Step 5 — The generative process becomes iterative

- To generate, we sample $z_T \sim \mathcal{N}(0, I)$, then progressively refine: $z_T \to z_{T-1} \to \cdots \to z_1 \to \hat{x}$
- Each step adds a bit of structure back — the decoder "unpeels" the latent hierarchy

From VAE to DDPM: A Guided Intuition

Part 3 — The DDPM Simplification

Step 6 — A simplifying insight

- In an HVAE, the encoder's job is still to progressively destroy structure until we reach Gaussian noise
- But we *know* a simple fixed process that does exactly this: *iteratively add Gaussian noise*
- Do we even need to *learn* the encoder?

Step 7 — Commit to a fixed encoder

- Let the encoder be a *fixed* forward process: $x_0 \to x_1 \to x_2 \to \cdots \to x_T \approx \mathcal{N}(0, I)$
- Each step just adds a little Gaussian noise with a predetermined schedule — no parameters to learn
- All the learning budget now goes to the decoder

Concept	VAE	HVAE	DDPM
Encoder	Learned	Learned	**Fixed** (add noise)
Decoder	Learned	Learned	Learned (denoise)
Latent structure	Single z	Chain $z_1, \ldots, z_T$	Chain $x_1, \ldots, x_T$
Prior	$\mathcal{N}(0, I)$	Top-level $\mathcal{N}(0, I)$	$x_T \sim \mathcal{N}(0, I)$

Step 8 — Design the decoder to mirror the encoder

- The fixed encoder adds noise step-by-step, so the decoder should *remove* noise step-by-step
- It makes sense for the decoder to have the *same functional form* as the encoder (Gaussian transitions), but with *learned* parameters
- The decoder becomes: $x_T \to x_{T-1} \to \cdots \to x_0 = \hat{x}$

Step 8b — A further simplification: just learn the noise

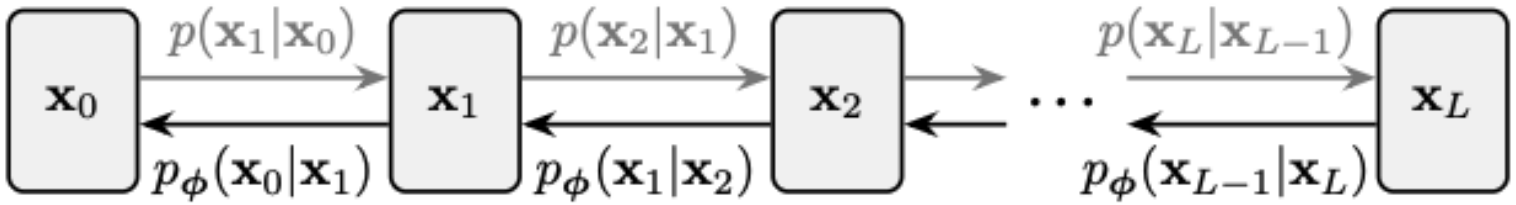
- Do we need a giant network to parameterize the entire reverse distribution?
- Simpler idea: at each forward step, we *know* we added some noise $\epsilon \sim \mathcal{N}(0, I)$
- What if the network just learns to *predict that noise* $\epsilon_\theta(x_t, t)$?
- Then to reverse a step, we simply subtract the predicted noise and walk backward toward the data
- This is the **ε-prediction** formulation: instead of learning $p_\theta(x_{t-1}|x_t)$ directly, we learn $\epsilon_\theta(x_t, t)$ and derive the reverse step from it
- The training loss becomes beautifully simple: $\mathcal{L} = \mathbb{E}_{t, x_0, \epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$

Step 9 — Arrive at DDPM

- Fixed encoder = forward diffusion process $q(x_t|x_{t-1})$
- Learned decoder = reverse diffusion process $p_\theta(x_{t-1}|x_t)$
- Training objective = maximize the variational lower bound (ELBO), just like VAE and HVAE
- In practice, we use the simplified ε-prediction loss from Step 8b
- **This is exactly a Denoising Diffusion Probabilistic Model**

Transition	What changes	Key insight
VAE → HVAE	Single latent → Chain of latents	Hierarchy enables progressive refinement
HVAE → DDPM	Learned encoder → Fixed encoder	If the encoder just adds noise, why learn it?

Variational Perspective: DDPM



Forward Pass (Fixed Encoder)

- Gradually **corrupts the data** by injecting Gaussian noise over multiple steps.
- Uses a fixed transition kernel

$$p(x_i \mid x_{i-1}).$$

- As i increases, the data distribution evolves toward an **isotropic Gaussian**,
  effectively becoming **pure noise**.
- Because this corruption process is *predefined*, the **encoder is fixed and not learned**.

Reverse Denoising Process (Learnable Decoder)

- A **neural network** learns to reverse the noise corruption via a parameterized distribution

$$p_\phi(x_{i-1} \mid x_i).$$

- Starts from **pure noise** and iteratively **denoises** to produce realistic samples.
- Each denoising step only needs to undo a *small amount* of noise, making it **easier than generating a full sample in one shot**.
- This contrasts with VAEs, which often try to generate x **directly from a single latent** z.

Forward Process (Fixed Encoder)

- Forward process defined by a fixed Gaussian transition kernel:

$$q(\mathbf{x}_i \mid \mathbf{x}_{i-1}) := \mathcal{N}\big(\mathbf{x}_i; \sqrt{1-\beta_i^2}\,\mathbf{x}_{i-1}, \beta_i^2 \mathbf{I}\big),$$

  where $\beta_i \in (0,1)$ is a pre-specified noise schedule.

- Define

$$\alpha_i := \sqrt{1-\beta_i^2}.$$

- Equivalent iterative update:

$$\mathbf{x}_i = \alpha_i \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}) \text{ i.i.d.}$$

- Intuition: at each step we
  - shrink the previous state by α_i, and
  - inject fresh Gaussian noise scaled by β_i.

Theorem (Why $\sqrt{1-\beta^2}$ and β?)

Consider a simplified setting with **constant** coefficients $\alpha_i \equiv \alpha$ and $\beta_i \equiv \beta$.

Transition kernel

$$q(\mathbf{x}_i \mid \mathbf{x}_{i-1}) = \mathcal{N}\big(\mathbf{x}_i; \alpha\,\mathbf{x}_{i-1}, \beta^2 \mathbf{I}\big), \qquad \mathbf{x}_i = \alpha \mathbf{x}_{i-1} + \beta \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}).$$

Goal. We want to choose α and β so that, as $i \to \infty$, the marginal distribution of $\mathbf{x}_i$ converges to $\mathcal{N}(0, \mathbf{I})$ (a standard-normal stationary distribution).

Claim. This happens **if and only if**

$$\alpha^2 + \beta^2 = 1 \quad \Longleftrightarrow \quad \beta^2 = 1 - \alpha^2.$$

Thus, once we pick a contraction factor $\alpha \in (0,1)$, the noise scale is forced to be $\beta = \sqrt{1-\alpha^2}$ — this is the origin of the $\sqrt{1-\beta_i^2}$ vs. β_i parameterization in fixed Gaussian transitions.

Proof

Unrolling the Recursion

Start from the update

$$\mathbf{x}_i = \alpha \mathbf{x}_{i-1} + \beta \boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}) \text{ i.i.d.}$$

Unroll the recursion up to time t:

$$\begin{aligned}
\mathbf{x}_t &= \alpha \mathbf{x}_{t-1} + \beta \boldsymbol{\epsilon}_t \\
&= \alpha(\alpha \mathbf{x}_{t-2} + \beta \boldsymbol{\epsilon}_{t-1}) + \beta \boldsymbol{\epsilon}_t \\
&= \alpha^2 \mathbf{x}_{t-2} + \alpha\beta \boldsymbol{\epsilon}_{t-1} + \beta \boldsymbol{\epsilon}_t \\
&\vdots \\
&= \alpha^t \mathbf{x}_0 + \beta \sum_{k=0}^{t-1} \alpha^k \boldsymbol{\epsilon}_{t-k}.
\end{aligned}$$

Define the noise accumulation term

$$\mathbf{u}_t := \beta \sum_{k=0}^{t-1} \alpha^k \boldsymbol{\epsilon}_{t-k}.$$

Then

$$\mathbf{x}_t = \alpha^t \mathbf{x}_0 + \mathbf{u}_t.$$

Assume $\mathbb{E}[\mathbf{x}_0] = \mathbf{0}$ and $\mathbb{E}[\boldsymbol{\epsilon}_i] = \mathbf{0}$.
Then $\mathbb{E}[\mathbf{x}_t] = \mathbf{0}$ for all t.

Covariance and Stationarity Condition

Because the $\boldsymbol{\epsilon}_i$ are independent with $\mathrm{Cov}(\boldsymbol{\epsilon}_i) = \mathbf{I}$, the covariance of $\mathbf{u}_t$ is

$$\begin{aligned}
\mathrm{Cov}[\mathbf{u}_t] &= \beta^2 \sum_{k=0}^{t-1} \alpha^{2k} \mathbf{I} \\
&= \beta^2 \frac{1 - \alpha^{2t}}{1 - \alpha^2} \mathbf{I}.
\end{aligned}$$

Thus

$$\begin{aligned}
\mathrm{Cov}[\mathbf{x}_t] &= \alpha^{2t} \mathrm{Cov}[\mathbf{x}_0] + \mathrm{Cov}[\mathbf{u}_t] \\
&= \alpha^{2t} \mathrm{Cov}[\mathbf{x}_0] + \beta^2 \frac{1 - \alpha^{2t}}{1 - \alpha^2} \mathbf{I}.
\end{aligned}$$

If $0 < \alpha < 1$, then $\alpha^{2t} \to 0$ as $t \to \infty$, so

$$\lim_{t \to \infty} \mathrm{Cov}[\mathbf{x}_t] = \beta^2 \frac{1}{1 - \alpha^2} \mathbf{I}.$$

To make the limiting distribution $\mathcal{N}(0, \mathbf{I})$, we require

$$\beta^2 \frac{1}{1 - \alpha^2} = 1 \quad \Longrightarrow \quad \beta^2 = 1 - \alpha^2.$$

Therefore, the forward kernel has a standard-normal stationary distribution **only** when

$$q(\mathbf{x}_i \mid \mathbf{x}_{i-1}) = \mathcal{N}\big(\mathbf{x}_i; \alpha\, \mathbf{x}_{i-1}, (1 - \alpha^2)\mathbf{I}\big),$$

which, in the DDPM notation, corresponds to choosing

$$\alpha_i = \sqrt{1 - \beta_i^2}, \qquad \beta_i^2 = 1 - \alpha_i^2.$$

Perturbation Kernel and Conditional Distribution

Forward (noising) process

At each step we apply the *fixed Gaussian transition*
(from the "Fixed Gaussian Transitions / Perturbation Kernel" view):

$$q(\mathbf{x}_i \mid \mathbf{x}_{i-1}) = \mathcal{N}\big(\mathbf{x}_i; \alpha_i \mathbf{x}_{i-1}, \beta_i^2 \mathbf{I}\big),$$

with

$$\alpha_i^2 + \beta_i^2 = 1, \qquad \mathbf{x}_i = \alpha_i \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}) \text{ i.i.d.}$$

Define the **cumulative signal coefficient**

$$\bar{\alpha}_i := \prod_{k=1}^{i} \alpha_k.$$

Theorem (Conditional Distribution $q_i(\mathbf{x}_i \mid \mathbf{x}_0)$) The conditional distribution of the noisy sample $\mathbf{x}_i$ given the original clean data $\mathbf{x}_0$ is

$$q_i(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_i; \bar{\alpha}_i \mathbf{x}_0, (1 - \bar{\alpha}_i^2)\, \mathbf{I}\big),$$

where

$$\bar{\alpha}_i := \prod_{k=1}^{i} \alpha_k.$$

Equivalently, we can **sample $\mathbf{x}_i$ directly from $\mathbf{x}_0$** as

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2}\, \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

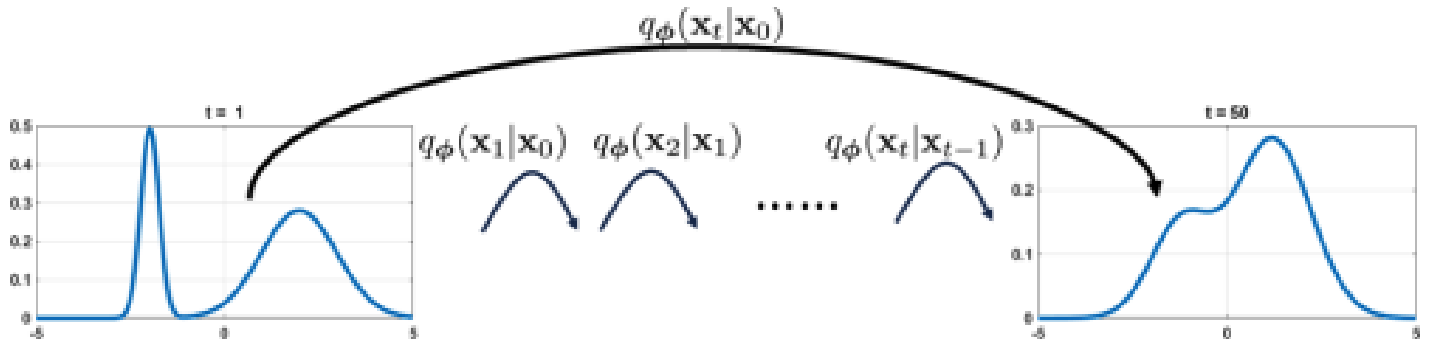


Figure 2.6: The difference between $q_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})$ and $q_\phi(\mathbf{x}_t|\mathbf{x}_0)$.

Stanley Chan (2025), Tutorial on Diffusion Models for Imaging and Vision

Proof

We use the recursion

$$\mathbf{x}_i = \alpha_i \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i, \qquad \beta_i^2 = 1 - \alpha_i^2, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}) \text{ i.i.d.}$$

Step 1 — Unrolling the recursion

Unroll the recursion up to step i:

$$\begin{aligned}
\mathbf{x}_1 &= \alpha_1 \mathbf{x}_0 + \beta_1 \boldsymbol{\epsilon}_1, \\
\mathbf{x}_2 &= \alpha_2 \mathbf{x}_1 + \beta_2 \boldsymbol{\epsilon}_2 = \alpha_2(\alpha_1 \mathbf{x}_0 + \beta_1 \boldsymbol{\epsilon}_1) + \beta_2 \boldsymbol{\epsilon}_2 \\
&= (\alpha_2 \alpha_1)\mathbf{x}_0 + \alpha_2 \beta_1 \boldsymbol{\epsilon}_1 + \beta_2 \boldsymbol{\epsilon}_2, \\
&\vdots \\
\mathbf{x}_i &= \bar{\alpha}_i \mathbf{x}_0 + \mathbf{w}_i,
\end{aligned}$$

where the **accumulated noise term** $\mathbf{w}_i$ is a sum of independent Gaussians:

$$\mathbf{w}_i := \sum_{k=1}^{i} \Big(\beta_k \prod_{j=k+1}^{i} \alpha_j \Big) \boldsymbol{\epsilon}_k.$$

Since a sum of independent Gaussian random variables is Gaussian, $\mathbf{w}_i$ is Gaussian with mean zero. Thus

$$\mathbb{E}[\mathbf{x}_i \mid \mathbf{x}_0] = \bar{\alpha}_i \mathbf{x}_0.$$

Step 2 — Covariance of the accumulated noise

We compute the covariance:

$$\mathrm{Cov}[\mathbf{w}_i] = \sum_{k=1}^{i} \Big(\beta_k^2 \prod_{j=k+1}^{i} \alpha_j^2 \Big) \mathbf{I} \quad (\text{independence and } \mathrm{Cov}[\boldsymbol{\epsilon}_k] = \mathbf{I}).$$

Using $\beta_k^2 = 1 - \alpha_k^2$ and the identity

$$\sum_{k=1}^{i} \Big((1 - \alpha_k^2) \prod_{j=k+1}^{i} \alpha_j^2 \Big) = 1 - \prod_{k=1}^{i} \alpha_k^2 = 1 - \bar{\alpha}_i^{\,2},$$

we obtain

$$\mathrm{Cov}[\mathbf{w}_i] = (1 - \bar{\alpha}_i^{\,2})\, \mathbf{I}.$$

Therefore,

$$\mathrm{Cov}[\mathbf{x}_i \mid \mathbf{x}_0] = \mathrm{Cov}[\mathbf{w}_i] = (1 - \bar{\alpha}_i^{\,2})\, \mathbf{I}.$$

Step 3 — Concluding the form of $q_i(\mathbf{x}_i \mid \mathbf{x}_0)$

We have shown that, conditioned on $\mathbf{x}_0$,

- $\mathbf{x}_i$ is Gaussian,
- with mean $\bar{\alpha}_i \mathbf{x}_0$,
- and covariance $(1 - \bar{\alpha}_i^{\,2})\mathbf{I}$.

Thus the conditional distribution is

$$q_i(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_i; \bar{\alpha}_i \mathbf{x}_0, (1 - \bar{\alpha}_i^{\,2})\, \mathbf{I}\big),$$

which matches the **Perturbation Kernel and Prior Distribution** notation and gives the closed-form expression used in DDPMs.

Reverse Denoising Process (Learnable Decoder)

Figure 2.5: Illustration of DDPM reverse (denoising) process. Starting from noise $\mathbf{x}_L \sim p_{\text{prior}}$, the model sequentially samples $\mathbf{x}_{i-1} \sim p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ for $i = L, \ldots, 1$ to obtain a newly generated data $\mathbf{x}$. The oracle transition $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is unknown; thus, we aim to approximate it.

- **Forward diffusion**:
  - Starting from data $\mathbf{x}_0 \sim p_{\text{data}}$, we apply fixed Gaussian transitions $q(\mathbf{x}_i \mid \mathbf{x}_{i-1})$ to gradually corrupt $\mathbf{x}_0$ into pure noise.
  - After many steps, the final state $\mathbf{x}_L$ is (approximately) drawn from a simple prior $p_{\text{prior}} = \mathcal{N}(0, \mathbf{I})$.
- **Reverse diffusion (generation)**:
  - The *core idea* of DDPMs is to **reverse** this controlled degradation.
  - Starting from $\mathbf{x}_L \sim p_{\text{prior}}$, we iteratively **denoise** $\mathbf{x}_L \to \mathbf{x}_{L-1} \to \cdots \to \mathbf{x}_0$, so that a coherent data sample emerges.

> **Question (Central challenge of DDPMs)**
> Can we precisely compute, or at least effectively approximate, the reverse transition kernels
>
> $$p_\theta(\mathbf{x}_{i-1} \mid \mathbf{x}_i),$$
>
> especially when the marginal distribution of $\mathbf{x}_i$,
>
> $$\mathbf{x}_i \sim p_i(\mathbf{x}_i),$$
>
> is highly complex?

- **Training objective viewpoint**:
  - Instead of immediately tackling the full ELBO derivation (as in the original DDPM paper), we first seek an **intuitive, tractable formulation**.
  - We will leverage **conditional probabilities** to derive an objective that directly encourages a good approximation of these reverse kernels $p_\theta(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$.

Overview: Modeling and Training Objective

- Goal: approximate the **unknown true reverse transition kernel**

$$p(\mathbf{x}_{i-1} \mid \mathbf{x}_i).$$

- Introduce a **learnable parametric model**

$$p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i),$$

  and train it by minimizing the **expected KL divergence** under the forward marginal $p_i(\mathbf{x}_i)$:

$$\mathbb{E}_{p_i(\mathbf{x}_i)}\Big[D_{\mathrm{KL}}\big(p(\mathbf{x}_{i-1} \mid \mathbf{x}_i) \,\|\, p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\big)\Big].$$

- Using the joint $p(\mathbf{x}_{i-1}, \mathbf{x}_i) = p(\mathbf{x}_{i-1} \mid \mathbf{x}_i) p_i(\mathbf{x}_i)$, this expected KL can be rewritten as

$$\mathbb{E}_{p(\mathbf{x}_{i-1}, \mathbf{x}_i)}\big[\log p(\mathbf{x}_{i-1} \mid \mathbf{x}_i) - \log p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\big].$$

  - The first term, $\mathbb{E}[\log p(\mathbf{x}_{i-1} \mid \mathbf{x}_i)]$, **does not depend on** ϕ (it is fixed by the forward process).
  - Therefore, minimizing the expected KL w.r.t. ϕ is **equivalent (up to an additive constant)** to **maximizing the expected log-likelihood**

$$\mathbb{E}_{p(\mathbf{x}_{i-1}, \mathbf{x}_i)}\big[\log p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\big].$$

  - Intuitively: we are training p_ϕ to be a **maximum-likelihood estimator** of the true reverse kernel.

Why the Target $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$ is Intractable

- Directly computing the true target distribution $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$ is difficult.
- By Bayes' theorem:

$$p(\mathbf{x}_{i-1} \mid \mathbf{x}_i) = p(\mathbf{x}_i \mid \mathbf{x}_{i-1}) \frac{p_{i-1}(\mathbf{x}_{i-1})}{p_i(\mathbf{x}_i)}.$$

  - We know the **forward kernel** $p(\mathbf{x}_i \mid \mathbf{x}_{i-1}) = q(\mathbf{x}_i \mid \mathbf{x}_{i-1})$ (Gaussian, by design).
  - But the ratio $\frac{p_{i-1}(\mathbf{x}_{i-1})}{p_i(\mathbf{x}_i)}$ is **intractable**.
- The marginals $p_i(\mathbf{x}_i)$ and $p_{i-1}(\mathbf{x}_{i-1})$ are expectations over the unknown data distribution p_{data}:

$$p_i(\mathbf{x}_i) = \int p_i(\mathbf{x}_i \mid \mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)\, d\mathbf{x}_0,$$

  and analogously for $p_{i-1}(\mathbf{x}_{i-1})$.
- Since p_{data} is unknown, these integrals have **no closed-form expression**; at best, they can be approximated from samples, so the exact densities are not available in practice.
- This intractability is why DDPMs seek **alternative parameterizations and training objectives** (e.g., noise-prediction losses) that avoid directly evaluating $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$.

Overcoming Intractability with Conditioning

- Recall that directly modeling the reverse kernel $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$ is intractable, because it depends on complicated marginals $p_i(\mathbf{x}_i)$.
- **Key DDPM trick:** *condition on a clean data sample* $\mathbf{x}$ (drawn from p_{data}) and consider instead the conditional reverse kernel $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x})$.
- By Bayes' rule, this conditional reverse kernel is

$$p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}) = p(\mathbf{x}_i \mid \mathbf{x}_{i-1}) \, \frac{p(\mathbf{x}_{i-1} \mid \mathbf{x})}{p(\mathbf{x}_i \mid \mathbf{x})}.$$

- Tractability comes from two properties of the forward process:
  - **Markov property:** $p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{x}) = p(\mathbf{x}_i \mid \mathbf{x}_{i-1})$.
  - **Gaussian transitions:** all involved distributions are Gaussian.
- As a result, $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x})$ is Gaussian and admits a closed-form expression , which lets us construct a **tractable training objective** equivalent to the original marginal KL.

Equivalence Between Marginal and Conditional KL Minimization

Theorem

The following equality holds:

$$\mathbb{E}_{p_i(\mathbf{x}_i)}\left[D_{\mathrm{KL}}\left(p(\mathbf{x}_{i-1} \mid \mathbf{x}_i) \,\|\, p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\right)\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{p(\mathbf{x}_i|\mathbf{x})}\left[D_{\mathrm{KL}}\left(p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}) \,\|\, p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\right)\right] + C,$$

where C is a constant independent of ϕ.

Moreover, the minimizer of satisfies

$$p^*(\mathbf{x}_{i-1} \mid \mathbf{x}_i) = \mathbb{E}_{p(\mathbf{x}|\mathbf{x}_i)}\left[p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x})\right] = p(\mathbf{x}_{i-1} \mid \mathbf{x}_i), \qquad \mathbf{x}_i \sim p_i.$$

Interpretation

- Minimizing the **marginal** KL is equivalent (up to a constant) to minimizing the **conditional** KL over $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x})$.
- This brings the problem into a setting where all distributions are Gaussian and can be written in closed form.

Reverse Conditional Transition Kernel

Lemma

The conditional reverse transition

$$p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x})$$

is Gaussian with closed-form expression

$$p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}) = \mathcal{N}\big(\mathbf{x}_{i-1};\, \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}, i),\, \sigma^2(i)\,\mathbf{I}\big),$$

where, using the DDPM notation $\bar{\alpha}_i := \prod_{k=1}^{i} \alpha_k$,

$$\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}, i) := \frac{\bar{\alpha}_{i-1}\,\beta_i^2}{1-\bar{\alpha}_i^2}\,\mathbf{x} + \frac{(1-\bar{\alpha}_{i-1}^2)\,\alpha_i}{1-\bar{\alpha}_i^2}\,\mathbf{x}_i,$$

$$\sigma^2(i) := \frac{1-\bar{\alpha}_{i-1}^2}{1-\bar{\alpha}_i^2}\,\beta_i^2.$$

- This is the **exact reverse conditional kernel** under the fixed Gaussian forward process.

Modeling the Reverse Transition Kernel $p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$

KL Objective Over All Time Steps

- For a given clean sample $\mathbf{x}_0 \sim p_{\text{data}}$, define the **diffusion loss** as the KL divergence between the true conditional reverse kernel and the model:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) := \sum_{i=1}^{L} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \Big[D_{\text{KL}} \big(p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}_0) \,\|\, p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i) \big) \Big].$$

- Intuition:
  - For each time step i, we want the **learned reverse kernel** $p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)$ to match the **true Gaussian kernel** $p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}_0)$, averaged over $\mathbf{x}_i$ drawn from the forward process.
- Because both kernels are Gaussian with:
  - same covariance $\sigma^2(i)\mathbf{I}$, and
  - means $\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)$ (true) and $\boldsymbol{\mu}_\phi(\mathbf{x}_i)$ (model), the KL admits a closed-form and simplifies to:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) = \sum_{i=1}^{L} \frac{1}{2\sigma^2(i)} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \Big[\big\| \boldsymbol{\mu}_\phi(\mathbf{x}_i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i) \big\|_2^2 \Big] + C,$$

where C is a constant independent of ϕ.

From KL to a Squared-Error Objective

- Averaging over the data distribution and dropping C, the **final DDPM training objective** is:

$$\mathcal{L}_{\text{DDPM}}(\phi) := \sum_{i=1}^{L} \frac{1}{2\sigma^2(i)} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \Big[\big\| \boldsymbol{\mu}_\phi(\mathbf{x}_i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i) \big\|_2^2 \Big],$$

with $\mathbf{x}_0 \sim p_{\text{data}}$.

- **Takeaway:** under the Gaussian parameterization, minimizing the KL over reverse kernels is equivalent to a **weighted squared-error regression** problem for the mean function $\boldsymbol{\mu}_\phi$.

Loss parametrization choices

$$\mathcal{L}_{\text{DDPM}}(\phi) := \sum_{i=1}^{L} \frac{1}{2\sigma^2(i)} \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \left[\left\| \boldsymbol{\mu}_\phi(\mathbf{x}_i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i) \right\|_2^2 \right]$$

ϵ-Prediction

$$\mathcal{L}_{\text{simple}}(\phi) := \mathbb{E}_{i, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i) - \boldsymbol{\epsilon} \right\|_2^2 \right]$$

x-Prediction

$$\mathbb{E}_{i, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\omega_i \left\| \mathbf{x}_\phi(\mathbf{x}_i, i) - \mathbf{x}_0 \right\|_2^2 \right]$$

ϵ—Prediction: Noise-Prediction View of DDPMs

- **Forward process (recap).**
  At diffusion step i, a noisy sample $\mathbf{x}_i$ is generated from clean data $\mathbf{x}_0 \sim p_{\text{data}}$ via

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2}\, \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}).$$

- **Reverse mean in terms of ϵ.**
  Using this expression, the reverse conditional mean $\mu(\mathbf{x}_i, \mathbf{x}_0, i)$ can be rewritten as

$$\mu(\mathbf{x}_i, \mathbf{x}_0, i) = \frac{1}{\alpha_i} \left(\mathbf{x}_i - \frac{1 - \alpha_i^2}{\sqrt{1 - \bar{\alpha}_i^2}}\, \boldsymbol{\epsilon} \right).$$

- **ϵ-prediction parameterization.**
  This suggests parameterizing the model mean μ_ϕ using a neural network $\boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i)$ that predicts the noise:

$$\mu_\phi(\mathbf{x}_i, i) = \frac{1}{\alpha_i} \left(\mathbf{x}_i - \frac{1 - \alpha_i^2}{\sqrt{1 - \bar{\alpha}_i^2}}\, \boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i) \right), \quad (\epsilon\text{-prediction})$$

  i.e., the network is trained to estimate the random noise added at step i.

- **From mean loss to noise loss.**
  Plugging this into the DDPM mean-squared loss yields

$$\left\| \mu_\phi(\mathbf{x}_i, i) - \mu(\mathbf{x}_i, \mathbf{x}_0, i) \right\|_2^2 \propto \left\| \boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i) - \boldsymbol{\epsilon} \right\|_2^2,$$

  up to a known weight depending on i.
  Intuitively, the model acts as a **noise detective**: it tries to recover the exact noise that was injected by the forward process.

ϵ—Prediction: Noise-Prediction View of DDPMs

- **Simplified ϵ-prediction loss.**
  Dropping the weighting term gives the widely used DDPM training loss:

$$\mathcal{L}_{\text{simple}}(\phi) := \mathbb{E}_{i,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\right\|_2^2\right],$$

  where $\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i^2}\,\boldsymbol{\epsilon}$, with $\mathbf{x}_0 \sim p_{\text{data}}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.

- **Why this works.**
  - The target noise has the **same variance at every timestep**, so the regression problem is well-scaled (no exploding/vanishing targets).
  - Both $\mathcal{L}_{\text{DDPM}}$ and $\mathcal{L}_{\text{simple}}$ share the same optimal solution

$$\boldsymbol{\epsilon}^*(\mathbf{x}_i, i) = \mathbb{E}[\boldsymbol{\epsilon} \mid \mathbf{x}_i], \qquad \mathbf{x}_i \sim p_i,$$

    so ϵ-prediction is just a convenient least-squares formulation of the original DDPM training objective.

x-Prediction: Clean-Prediction View of DDPMs

- Beyond ϵ-prediction, DDPMs admit an **x-prediction** (clean prediction) view:
  - A neural network $\mathbf{x}_\phi(\mathbf{x}_i, i)$ is trained to predict the **clean sample** $\mathbf{x}_0$ from a noisy input $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$ at noise level i.
- Plugging $\mathbf{x}_\phi(\mathbf{x}_i, i)$ into the reverse mean gives the model mean

$$\mu_\phi(\mathbf{x}_i, i) = \frac{\bar{\alpha}_{i-1}\beta_i^2}{1-\bar{\alpha}_i^2}\,\mathbf{x}_\phi(\mathbf{x}_i, i) + \frac{(1-\bar{\alpha}_{i-1}^2)\alpha_i}{1-\bar{\alpha}_i^2}\,\mathbf{x}_i.$$

- **Training objective (x-prediction form).** Analogously to ϵ-prediction, the mean-matching loss

$$\|\mu_\phi(\mathbf{x}_i, i) - \mu(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2$$

  is proportional to the reconstruction loss

$$\|\mu_\phi(\mathbf{x}_i, i) - \mu(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 \propto \|\mathbf{x}_\phi(\mathbf{x}_i, i) - \mathbf{x}_0\|_2^2, \qquad \mathbf{x}_0 \sim p_{\text{data}}.$$

  Thus the model is trained to recover the original clean sample from its noisy version.

- Averaging over data, timesteps, and noise yields

$$\mathbb{E}_{i,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\omega_i \,\|\mathbf{x}_\phi(\mathbf{x}_i, i) - \mathbf{x}_0\|_2^2\right],$$

  for some weighting function ω_i.
  Being a least-squares problem, the optimal predictor is

$$\mathbf{x}^*(\mathbf{x}_i, i) = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_i], \qquad \mathbf{x}_i \sim p_i.$$

- **Equivalence with ϵ-prediction.** The x-prediction and ϵ-prediction parameterizations are mathematically equivalent and related by

$$\mathbf{x}_i = \alpha_i\,\mathbf{x}_\phi(\mathbf{x}_i, i) + \sqrt{1-\alpha_i^2}\;\boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i).$$

  That is, one may either predict the clean sample $\mathbf{x}_\phi(\mathbf{x}_i, i)$ or the noise $\boldsymbol{\epsilon}_\phi(\mathbf{x}_i, i)$; combined , they reconstruct the same noisy $\mathbf{x}_i$ generated by the forward process.

From Reverse Transitions to a Generative Model

- With the reverse kernels

$$p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i) = \mathcal{N}\left(\mathbf{x}_{i-1}; \mu_\phi(\mathbf{x}_i, i), \sigma^2(i)\mathbf{I}\right)$$

  the **joint generative distribution** in DDPM is

$$p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L}) := p_\phi(\mathbf{x}_0 \mid \mathbf{x}_1) p_\phi(\mathbf{x}_1 \mid \mathbf{x}_2) \cdots p_\phi(\mathbf{x}_{L-1} \mid \mathbf{x}_L)\, p_{\text{prior}}(\mathbf{x}_L).$$

- The **marginal generative model for data** is

$$p_\phi(\mathbf{x}_0) := \int p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L})\, d\mathbf{x}_{1:L}.$$

- Training via the KL/likelihood objectives we derived can be placed in a **maximum-likelihood** framework using an ELBO, just like in VAEs.

DDPM's ELBO

> **Theorem (DDPM's ELBO).**
> For any $\mathbf{x}_0$,
>
> $$\begin{aligned} -\log p_\phi(\mathbf{x}_0) \leq & -\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \phi) \\ & := \mathcal{L}_{\text{prior}}(\mathbf{x}_0; \phi) \\ & + \mathcal{L}_{\text{recon}}(\mathbf{x}_0; \phi) \\ & + \mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi). \end{aligned}$$

- The three terms are
  - **Prior term**

$$\mathcal{L}_{\text{prior}}(\mathbf{x}_0; \phi) := D_{\text{KL}}\big(p(\mathbf{x}_L \mid \mathbf{x}_0) \,\|\, p_{\text{prior}}(\mathbf{x}_L)\big).$$

  - **Reconstruction term**

$$\mathcal{L}_{\text{recon}}(\mathbf{x}_0; \phi) := \mathbb{E}_{p(\mathbf{x}_1|\mathbf{x}_0)}\big[-\log p_\phi(\mathbf{x}_0 \mid \mathbf{x}_1)\big].$$

  - **Diffusion (transition-matching) term**

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) := \sum_{i=1}^{L} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)}\Big[D_{\text{KL}}\big(p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}_0) \,\|\, p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i)\big)\Big].$$

Interpreting the ELBO and Variational View

- **Practical notes on the three terms**
  - $\mathcal{L}_{\text{prior}}$ can be made negligible by choosing the noise schedule $\{\beta_i\}$ such that $p(\mathbf{x}_L \mid \mathbf{x}_0)$ is close to the chosen prior p_{prior}.
  - $\mathcal{L}_{\text{recon}}$ can be approximated and optimized via Monte Carlo estimates (as in VAE-style objectives).
  - $\mathcal{L}_{\text{diffusion}}$ is exactly the transition-matching loss we derived earlier: it enforces $p_\phi(\mathbf{x}_{i-1} \mid \mathbf{x}_i) \approx p(\mathbf{x}_{i-1} \mid \mathbf{x}_i, \mathbf{x}_0)$ for all i.
- **Data-processing / HVAE perspective**
  - The ELBO $\mathcal{L}_{\text{ELBO}}$ can also be seen through the **Data Processing Inequality**: with latent variables $\mathbf{z} = \mathbf{x}_{1:L}$,

$$D_{\text{KL}}(p_{\text{data}}(\mathbf{x}_0) \,\|\, p_\phi(\mathbf{x}_0)) \;\le\; D_{\text{KL}}\big(p(\mathbf{x}_0, \mathbf{x}_{1:L}) \,\|\, p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L})\big),$$

where the joint $p(\mathbf{x}_0, \mathbf{x}_{1:L})$ follows the forward noising chain.

- Diffusion's variational view fits an **HVAE template**:
  - The "encoder" is the *fixed* forward noising chain.
  - Latents $\mathbf{x}_{1:L}$ have the same dimensionality as the data.
  - There is no learned encoder and no per-level KL penalties: the problem decomposes into a sequence of **well-conditioned denoising subproblems**, from large to small noise (coarse-to-fine).
- This yields **stable optimization** and high sample quality while preserving a coarse-to-fine hierarchy over time / noise levels.

Training

Training DDPM using $\widehat{\epsilon}_{\theta}(\mathbf{x}_t)$. 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

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

Stanley Chan (2025), Tutorial on Diffusion Models for Imaging and Vision

Inference

Inference of DDPM using $\widehat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{x}_t)$.

- 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

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

Stanley Chan (2025), Tutorial on Diffusion Models for Imaging and Vision

Score-Based Perspective

- Energy based models (EBMs)

Let $\mathbf{x} \in \mathbb{R}^D$ denote a data point. EBMs define a probability density via an energy function $E_\phi(\mathbf{x})$, parameterized by ϕ, which assigns lower energy to more likely configurations. The resulting distribution is given by

$$p_\phi(\mathbf{x}) := \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi}, \quad Z_\phi := \int_{\mathbb{R}^D} \exp(-E_\phi(\mathbf{x}))\, \mathrm{d}\mathbf{x},$$

where Z_ϕ is called the *partition function* ensuring normalization:

$$\int_{\mathbb{R}^D} p_\phi(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1.$$

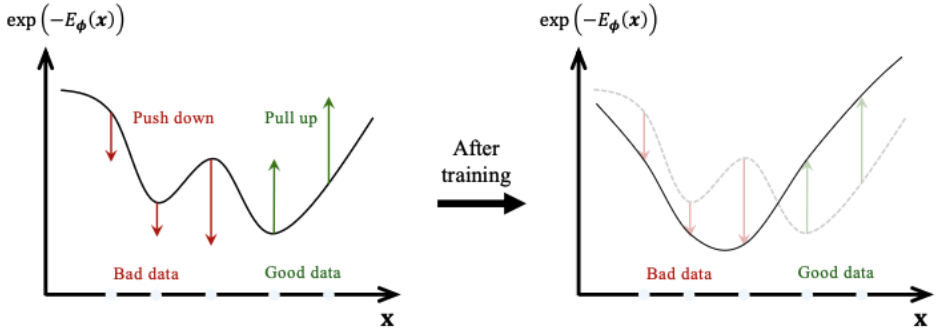


Figure 3.1: Illustration of EBM training. The model lowers density (raises energy) at "bad" data points (red arrows), and raises density (lowers energy) at "good" data points (green arrows).

EBMs borrow this from statistical physics:

- In a physical system at temperature T, the equilibrium distribution is

$$p(\text{state } x) \propto \exp\left(-\frac{E(x)}{kT}\right).$$

- Lower energy states are exponentially more likely than higher energy states.

Our EBM just sets $kT = 1$ and reuses the same idea: **energy encodes how "costly" or "unlikely" a configuration is**, and the exponential map turns that cost into a normalized probability.

Challenges of Maximum Likelihood Training in EBMs

- In principle, Energy-Based Models can be trained by **maximum likelihood**:

$$\mathcal{L}_{\text{MLE}}(\phi) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\log \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi}\right],$$

  where

$$Z_\phi = \int \exp(-E_\phi(\mathbf{x}))\, d\mathbf{x}$$

  is the **partition function**.

- Expanding the objective:

$$\mathcal{L}_{\text{MLE}}(\phi) = -\,\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[E_\phi(\mathbf{x})] \;-\; \log \int \exp(-E_\phi(\mathbf{x}))\, d\mathbf{x}.$$

  - First term (**lowers energy of data**): pushes real data points into low-energy (high-density) regions.
  - Second term (**global regularization**): enforces normalization via the partition function, balancing energy across the whole space.
- **Main difficulty in high dimensions**
  - Computing $\log Z_\phi$ and its gradient is **intractable**: it requires expectations under the model distribution.
  - This motivates **alternative training objectives** that:
    - approximate the partition term (e.g., contrastive divergence), or
    - avoid it altogether (e.g., **score matching**).

Score-Based Generative Models: Sampling High-Density Regions

- **Goal of generation:** draw samples from a target distribution $p(\mathbf{x})$.
  - Intuitively, we want samples that lie in **high-probability / high-density regions** of $p(\mathbf{x})$.
- **Score function**
  - The **score** is the gradient of log-density:

$$
\begin{aligned}
s_\phi(\mathbf{x}) &:= \nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}) \\
&= \nabla_{\mathbf{x}} \log \frac{\exp -E_\phi(x)}{Z_\phi} \\
&= \nabla_{\mathbf{x}} (-E_\phi(x) - \underbrace{\log Z_\phi}_{\text{Independent of } x}) \\
&= -\nabla_{\mathbf{x}} E_\phi(\mathbf{x}),
\end{aligned}
$$

  which **does not depend on** Z_ϕ.
  - The score fully characterizes the distribution (up to a constant in log p):

$$
\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^1 s\big(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)\big)^\top (\mathbf{x} - \mathbf{x}_0)\, dt.
$$

  - So **modeling the score is as expressive as modeling** $p(\mathbf{x})$.

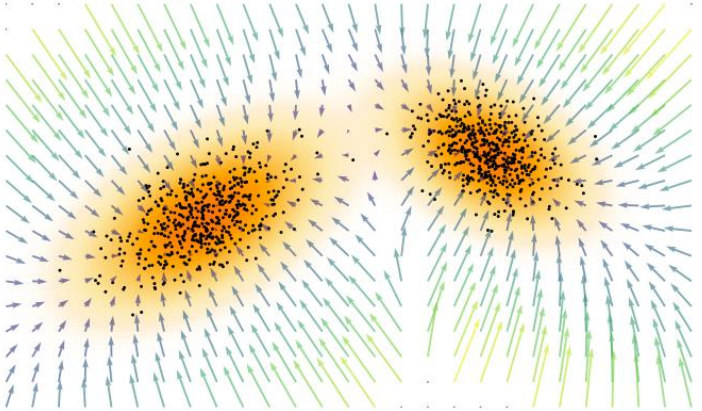


Figure 3.2: Illustration of score vector fields. Score vector fields $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ indicate directions of increasing density.

Proof: Score Function as a Complete Representation

Let the **score function** be

$$s(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}).$$

Define the straight-line path from a reference point $\mathbf{x}_0$ to $\mathbf{x}$:

$$\gamma(t) := \mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0), \quad t \in [0, 1].$$

Consider the scalar function

$$f(t) := \log p(\gamma(t)) = \log p\big(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)\big).$$

By the chain rule,

$$\frac{d}{dt} f(t) = \nabla_{\mathbf{x}} \log p(\gamma(t))^\top \gamma'(t) = s\big(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)\big)^\top (\mathbf{x} - \mathbf{x}_0).$$

Integrating from 0 to 1 and applying the fundamental theorem of calculus:

$$\log p(\mathbf{x}) - \log p(\mathbf{x}_0) = \int_0^1 \frac{d}{dt} f(t)\, dt = \int_0^1 s\big(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)\big)^\top (\mathbf{x} - \mathbf{x}_0)\, dt.$$

Rearranging gives the desired formula:

$$\boxed{\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^1 s\big(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0)\big)^\top (\mathbf{x} - \mathbf{x}_0)\, dt}$$

Thus, knowing the score $s(\mathbf{x})$ everywhere determines $\log p(\mathbf{x})$ up to the additive constant $\log p(\mathbf{x}_0)$, so the score fully characterizes the distribution.

Intuition

Classical likelihood modeling

- We want to model the data density $p_\theta(x)$ and maximize log-likelihood:

$$\log p_\theta(x) = \log \tilde{p}_\theta(x) - \log Z(\theta),$$

  where $Z(\theta) = \int \tilde{p}_\theta(x)\, dx$ is the **normalizing constant**.
- In many interesting models, $Z(\theta)$ is **intractable** $\Rightarrow$ computing $\log p_\theta(x)$ is hard.

Score matching intuition

- Instead of fitting $p_\theta(x)$ directly, we learn a parametric score:

$$s_\phi(x) \approx \nabla_x \log p_{\text{data}}(x).$$

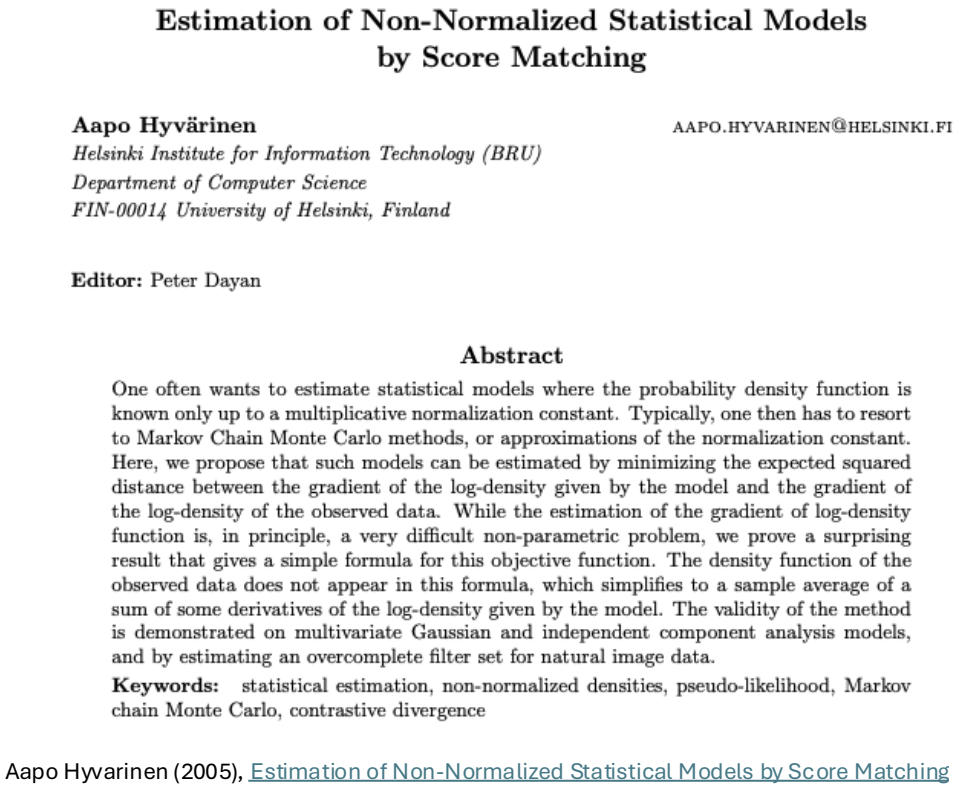
- We choose ϕ so that $s_\phi(x)$ is as close as possible to the **true score** of the data distribution (Hyvärinen score matching, denoising score matching, etc.).
- We have now learned a **vector field** that points towards higher-density regions of the (implicit) data density.

Using the learned score: Langevin dynamics

- Given $s_\phi(x)$, we can sample via (unadjusted) Langevin dynamics:

$$x_{k+1} = x_k + \varepsilon\, s_\phi(x_k) + \sqrt{2\varepsilon}\, \xi_k, \quad \xi_k \sim \mathcal{N}(0, I).$$

- This Markov chain uses **only the score**, not the normalized density.
- In the limit (small ε, long run), samples approximate a distribution whose log-density has gradient $s_\phi(x)$ — an **implicit model** of $p_{\text{data}}(x)$.



Estimation of Non-Normalized Statistical Models by Score Matching

Aapo Hyvärinen AAPO.HYVARINEN@HELSINKI.FI
Helsinki Institute for Information Technology (BRU)
Department of Computer Science
FIN-00014 University of Helsinki, Finland

Editor: Peter Dayan

Abstract

One often wants to estimate statistical models where the probability density function is known only up to a multiplicative normalization constant. Typically, one then has to resort to Markov Chain Monte Carlo methods, or approximations of the normalization constant. Here, we propose that such models can be estimated by minimizing the expected squared distance between the gradient of the log-density given by the model and the gradient of the log-density of the observed data. While the estimation of the gradient of log-density function is, in principle, a very difficult non-parametric problem, we prove a surprising result that gives a simple formula for this objective function. The density function of the observed data does not appear in this formula, which simplifies to a sample average of a sum of some derivatives of the log-density given by the model. The validity of the method is demonstrated on multivariate Gaussian and independent component analysis models, and by estimating an overcomplete filter set for natural image data.

Keywords: statistical estimation, non-normalized densities, pseudo-likelihood, Markov chain Monte Carlo, contrastive divergence

Aapo Hyvarinen (2005), Estimation of Non-Normalized Statistical Models by Score Matching

Takeaway

- When **likelihood is hard** because of an intractable normalizing constant, it can be easier to:
  - **learn the score field** $s_\phi(x) \approx \nabla_x \log p_{\text{data}}(x)$,
  - then use **Langevin dynamics** (or reverse SDEs) to sample from the induced implicit distribution.
- The parameters ϕ are explicitly parameters of the **score**, and *implicitly* define a density we can **sample from and explore**, even if we cannot write its normalized form in closed form.

Estimation the score function

The ideal score-matching loss

- In an ideal world, we would like to fit the model score

$$s_\phi(x) := \nabla_x \log p_\phi(x)$$

  to the **true** data score:

$$\nabla_x \log p_{\text{data}}(x).$$

- The ideal objective is:

$$\mathcal{L}_{\text{SM}}(\phi) = \frac{1}{2}\,\mathbb{E}_{p_{\text{data}}(x)}\Big[\big\|\nabla_x \log p_\phi(x) - \nabla_x \log p_{\text{data}}(x)\big\|_2^2\Big].$$

Why this is not computable

- To evaluate this loss, for every training sample x we would need the **label**

$$\nabla_x \log p_{\text{data}}(x).$$

- But:
  - We do **not** know $p_{\text{data}}(x)$ in closed form.
  - Therefore we also do **not** know its gradient or log-gradient.
  - The empirical distribution

$$\hat{p}(x) = \frac{1}{n}\sum_i \delta(x - x_i)$$

    is a sum of spikes — its "gradient" is not a useful object.

> Intuition:
> Asking for $\nabla_x \log p_{\text{data}}(x)$ is like asking for the slope of a function we have never written down — we only know random points sampled from it.

What is $p_{\text{data}}(x)$?

- $p_{\text{data}}(x)$ is the **true, unknown** data-generating density.
- In practice, we never see p_{data} analytically.
- We only see **samples**:

$$x_1, \ldots, x_n \sim p_{\text{data}}(x).$$

Training EBMs via Score Matching

- EBM density:

$$p_\phi(\mathbf{x}) = \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi}.$$

- **Model score** (gradient of log-density):

$$\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\phi(\mathbf{x}),$$

  which is **independent of the partition function** Z_ϕ.

Score Matching Objective

- Goal: match the **model score** to the (unknown) **data score** $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$.
- Ideal score matching loss:

$$\mathcal{L}_{\text{SM}}(\phi) = \frac{1}{2}\,\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2\right].$$

Explicit

- Although the data score is not directly accessible, **integration by parts** yields an equivalent form that depends only on the **energy** and its derivatives:

$$\mathcal{L}_{\text{SM}}(\phi) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\,\text{Tr}\left(\nabla_{\mathbf{x}}^2 E_\phi(\mathbf{x})\right) + \tfrac{1}{2}\left\|\nabla_{\mathbf{x}} E_\phi(\mathbf{x})\right\|_2^2\right] + C,$$

  where $\nabla_{\mathbf{x}}^2 E_\phi(\mathbf{x})$ is the Hessian and C is a constant independent of ϕ.

Implicit

- **Pros:** avoids computing / differentiating Z_ϕ and does not require sampling from the model during training.
- **Cons:** involves **second-order derivatives** of the energy, which can be computationally expensive in high dimensions.

Langevin Dynamics: Moving Toward High-Density Areas

- **Key idea:** use the score to *move samples toward higher density* regions without ever computing the normalization constant Z_ϕ.
- **Discrete-time Langevin dynamics**
  - Starting from $\mathbf{x}_0$ (often Gaussian noise), update:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta \nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}_n) + \sqrt{2\eta}\,\boldsymbol{\epsilon}_n, \quad \boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \mathbf{I}),$$

    where $\eta > 0$ is a step size.
  - The **drift term** $\eta \nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}_n)$ pushes samples **up the log-density landscape** (toward high-density areas).
  - The **noise term** $\sqrt{2\eta}\,\boldsymbol{\epsilon}_n$ adds exploration, preventing collapse to a single mode.
- **Continuous-time limit (SDE)**

$$d\mathbf{x}(t) = \nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}(t))\, dt + \sqrt{2}\, d\mathbf{w}(t),$$

  whose stationary distribution is $p_\phi(\mathbf{x})$.
- **Big picture**
  - Score-based / Langevin generative models **generate data by repeatedly nudging noise samples toward regions where $p_\phi(\mathbf{x})$ is large.**
  - In other words, **generation boils down to using the score field to sample from high-density regions of the learned distribution.**

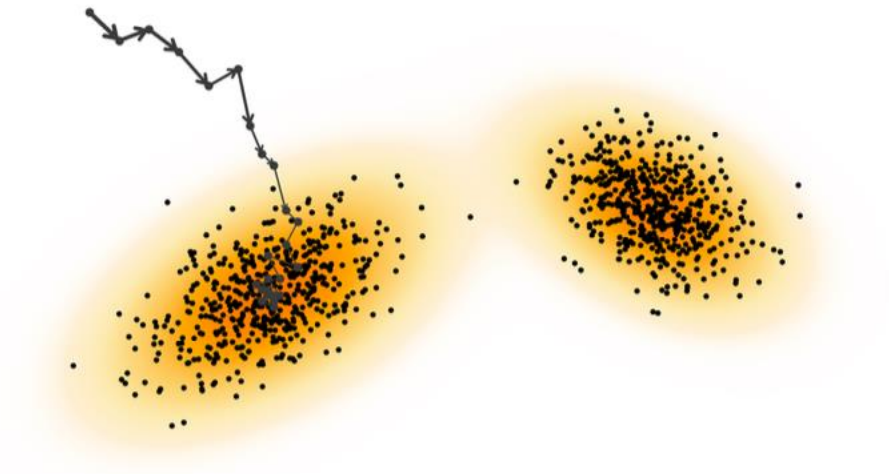


Figure 3.3: Illustration of Langevin sampling. Langevin sampling using the score function $\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x})$ to guide trajectories toward high-density regions via the update in Equation (3.1.5) (indicating by arrows).

Score Matching: From Infeasible to tractable

Goal. Learn the score (log-density gradient) of the data:

$$\mathbf{s}(x) := \nabla_x \log p_{\text{data}}(x).$$

We parameterize a vector field (e.g. a neural net)

$$\mathbf{s}_\phi(x) \approx \mathbf{s}(x).$$

Ideal (but infeasible) score matching loss

$$\mathcal{L}_{\text{SM}}(\phi) := \frac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\left[\left\|\mathbf{s}_\phi(x) - \mathbf{s}(x)\right\|_2^2\right].$$

- This is just **MSE between estimated and true scores**.
- Problem: the **target** $\mathbf{s}(x) = \nabla_x \log p_{\text{data}}(x)$ is unknown $\Rightarrow$ we cannot evaluate this loss directly.

Hyvarinen's tractable form

Hyvarinen & Dayan (2005) showed that, under mild conditions, we can rewrite:

$$\mathcal{L}_{\text{SM}}(\phi) = \tilde{\mathcal{L}}_{\text{SM}}(\phi) + C,$$

where C does **not** depend on ϕ and

$$\tilde{\mathcal{L}}_{\text{SM}}(\phi) := \mathbb{E}_{x\sim p_{\text{data}}}\left[\operatorname{Tr}\left(\nabla_x \mathbf{s}_\phi(x)\right) + \frac{1}{2}\left\|\mathbf{s}_\phi(x)\right\|_2^2\right].$$

- This form depends **only** on:
  - samples $x \sim p_{\text{data}}$, and
  - model derivatives $\mathbf{s}_\phi(x)$ and its Jacobian.
- Minimizer is the **true score**:

$$\mathbf{s}^*(x) = \nabla_x \log p(x).$$

Intuition for the Tractable Score Matching Loss

$$\tilde{\mathcal{L}}_{\mathrm{SM}}(\phi) = \mathbb{E}_{x\sim p_{\mathrm{data}}}\left[\underbrace{\tfrac{1}{2}\|\mathbf{s}_\phi(x)\|_2^2}_{\text{magnitude term}} + \underbrace{\mathrm{Tr}(\nabla_x \mathbf{s}_\phi(x))}_{\text{divergence term}}\right]$$

Stationarity from the magnitude term

- Expectation is taken under $p_{\mathrm{data}}(x)$:
  - **High-density regions** contribute most to the loss.
- The magnitude term

  $$\tfrac{1}{2}\|\mathbf{s}_\phi(x)\|_2^2$$

  penalizes large scores where $p_{\mathrm{data}}(x)$ is large.
- This pushes $\mathbf{s}_\phi(x) \to 0$ at high-probability locations
  $\Rightarrow$ these points become **stationary points** of the learned field.

Sinks and concavity from the divergence term

- The divergence term

  $$\mathrm{Tr}(\nabla_x \mathbf{s}_\phi(x)) = \nabla_x \cdot \mathbf{s}_\phi(x)$$

  is encouraged to be **negative** in high-density regions.
- Negative divergence $\Rightarrow$ nearby vectors **converge** rather than spread out
  $\Rightarrow$ stationary points act as **sinks** (trajectories flow inward).

Assume $\mathbf{s}_\phi = \nabla_x u$ for some scalar potential $u : \mathbb{R}^D \to \mathbb{R}$:

- Then $\nabla_x \mathbf{s}_\phi(x) = \nabla_x^2 u(x)$ (Hessian), and
  $\nabla_x \cdot \mathbf{s}_\phi(x) = \mathrm{Tr}(\nabla_x^2 u(x))$.
- At a stationary point $x_\star$ with $\mathbf{s}_\phi(x_\star) = 0$, a 2nd-order Taylor expansion gives:

  $$u(x) \approx u(x_\star) + \tfrac{1}{2}(x - x_\star)^\top \nabla_x^2 u(x_\star)(x - x_\star).$$

- If the Hessian is **negative definite**, u (and hence $\log p$) is locally concave:
  - $x_\star$ is a **local maximum** of the log-density,
  - the field has negative divergence there,
  - and $x_\star$ is a **stable sink**: trajectories are contracted back toward it.

Takeaway

- The tractable score matching loss shapes the learned vector field so that:
  - high-density regions become **stationary points** ($\mathbf{s}_\phi \approx 0$),
  - these points are **attractive sinks** with negative divergence.
- Thus $\mathbf{s}_\phi$ learns to behave like the gradient of a **peaked log-density** concentrated where real data live.

From Tractable SM to Sliced & Denoising Score Matching

Problem with Hyvarinen's SM in High Dimension

Recall the tractable score matching loss:

$$\tilde{\mathcal{L}}_{\mathrm{SM}}(\phi) = \mathbb{E}_{x\sim p_{\mathrm{data}}}\left[\mathrm{Tr}(\nabla_x \mathbf{s}_\phi(x)) + \tfrac{1}{2}\|\mathbf{s}_\phi(x)\|_2^2\right].$$

- Still requires the **trace of the Jacobian** $\mathrm{Tr}(\nabla_x \mathbf{s}_\phi(x))$.
- Worst-case complexity: $\mathcal{O}(D^2)$ in dimension D ⇒ problematic for high-dimensional data (e.g. images).

Sliced Score Matching & Hutchinson's Trick

Idea: Estimate the trace term via random projections instead of explicit Jacobians.

Let $u \in \mathbb{R}^D$ be an **isotropic random vector** (e.g. Rademacher or standard Gaussian) with $\mathbb{E}[u] = 0$, $\mathbb{E}[uu^\top] = I$.

Hutchinson identities

$$\mathrm{Tr}(A) = \mathbb{E}_u[u^\top A u], \quad \mathbb{E}_u[(u^\top \mathbf{s}_\phi(x))^2] = \|\mathbf{s}_\phi(x)\|_2^2.$$

Plugging into $\tilde{\mathcal{L}}_{\mathrm{SM}}$ gives the **sliced SM loss**:

$$\tilde{\mathcal{L}}_{\mathrm{SM}}(\phi) = \mathbb{E}_{x,u}\left[u^\top(\nabla_x \mathbf{s}_\phi(x))u + \tfrac{1}{2}(u^\top \mathbf{s}_\phi(x))^2\right].$$

- Can be evaluated efficiently with **JVP/VJP** ops instead of forming full Jacobians/Hessians.
- Averaging over K random probes gives variance $\mathcal{O}(1/K)$.

Limitations of Sliced Score Matching

Even though sliced SM avoids explicit Jacobians, it still:

- Relies directly on the **raw data distribution** $p_{\text{data}}(x)$.
  - For data on low-dimensional manifolds (e.g. images), $\nabla_x \log p_{\text{data}}(x)$ may be **undefined or unstable**.
- Constrains the score field **only at the observed points**, giving weak control in neighborhoods.
- Suffers from:
  - variance introduced by random probes, and
  - repeated JVP/VJP costs.

$\Rightarrow$ Motivates a more robust alternative: **Denoising Score Matching (DSM).**

Denoising Score Matching: Vincent's C

We would like to learn the **true score** of the data:

$$s(x) := \nabla_x \log p_{\text{data}}(x).$$

The ideal score matching loss is

$$\mathcal{L}_{\text{SM}}^{\text{ideal}}(\phi) = \tfrac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\left[\|s_\phi(x) - \nabla_x \log p_{\text{data}}(x)\|_2^2\right].$$

Problems:

- In practice we *only* have samples $x_1, \ldots, x_n$ from p_{data}.
- The **empirical distribution**

$$\hat{p}_{\text{data}}(x) = \tfrac{1}{n}\sum_{i=1}^{n} \delta(x - x_i)$$

  is a sum of spikes: its log and its gradient are not useful objects.
- So we cannot compute $\nabla_x \log p_{\text{data}}(x)$ (or of $\hat{p}_{\text{data}}$).

A Connection Between Score Matching and Denoising Autoencoders

Pascal Vincent
vincentp@iro.umontreal.ca
Département d'Informatique, Université de Montréal,
Montréal (QC) H3C 3J7, Canada

Denoising autoencoders have been previously shown to be competitive alternatives to restricted Boltzmann machines for unsupervised pretraining of each layer of a deep architecture. We show that a simple denoising autoencoder training criterion is equivalent to matching the score (with respect to the data) of a specific energy-based model to that of a nonparametric Parzen density estimator of the data. This yields several useful insights. It defines a proper probabilistic model for the denoising autoencoder technique, which makes it in principle possible to sample from them or rank examples by their energy. It suggests a different way to apply score matching that is related to learning to denoise and does not require computing second derivatives. It justifies the use of tied weights between the encoder and decoder and suggests ways to extend the success of denoising autoencoders to a larger family of energy-based models.

Source

Smooth the empirical distribution (Parzen / KDE view)

Idea: replace the spiky empirical distribution by a **smoothed version**.

Add Gaussian noise to each data point:

$$\tilde{x} = x + \sigma\epsilon, \quad x \sim p_{\text{data}},\ \epsilon \sim \mathcal{N}(0, I).$$

The resulting **noisy marginal** is

$$p_\sigma(\tilde{x}) = \int p_\sigma(\tilde{x} \mid x)\, p_{\text{data}}(x)\, dx, \quad p_\sigma(\tilde{x} \mid x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I).$$

If we only know the empirical distribution, this is exactly a **Parzen / kernel density estimator**:

$$p_\sigma(\tilde{x}) \approx \frac{1}{n} \sum_{i=1}^{n} \mathcal{N}(\tilde{x}; x_i, \sigma^2 I),$$

i.e. a Gaussian bump around each data point.

- For **small** σ: p_σ is very close to p_{data}.
- But unlike $\hat{p}_{\text{data}}$, p_σ is **smooth**, so $\log p_\sigma(\tilde{x})$ and its gradient are well-defined.

So a *practical* surrogate for the ideal loss is:

$$\mathcal{L}_{\text{SM}}(\phi; \sigma) = \tfrac{1}{2}\, \mathbb{E}_{\tilde{x} \sim p_\sigma} \left[\| s_\phi(\tilde{x}; \sigma) - \nabla_{\tilde{x}} \log p_\sigma(\tilde{x}) \|_2^2 \right].$$

As $\sigma \to 0$,

$$p_\sigma(\tilde{x}) \to p_{\text{data}}(x) \quad \text{and} \quad \nabla_{\tilde{x}} \log p_\sigma(\tilde{x}) \to \nabla_x \log p_{\text{data}}(x)$$

(under mild regularity), so $\mathcal{L}_{\text{SM}}(\phi; \sigma)$ becomes a good **approximation** to the ideal $\mathcal{L}_{\text{SM}}^{\text{ideal}}(\phi)$.

From SM to DSM: Conditioning Makes It Tractable

Although $\nabla_{\tilde{x}} \log p_\sigma(\tilde{x})$ is still intractable in general, Vincent showed that **conditioning on the clean data** yields an equivalent, tractable objective:

$$\mathcal{L}_{\mathrm{DSM}}(\phi;\sigma) := \tfrac{1}{2}\,\mathbb{E}_{x\sim p_{\mathrm{data}},\tilde{x}\sim p_\sigma(\cdot|x)}\left[\|\mathbf{s}_\phi(\tilde{x};\sigma) - \nabla_{\tilde{x}} \log p_\sigma(\tilde{x}\mid x)\|_2^2\right].$$

- The conditional score $\nabla_{\tilde{x}} \log p_\sigma(\tilde{x}\mid x)$ is **known in closed form** (by construction of the noise).
- The optimal minimizer satisfies

$$\mathbf{s}^*(\tilde{x};\sigma) = \nabla_{\tilde{x}} \log p_\sigma(\tilde{x}),$$

  so DSM still learns the **marginal noisy score**.

Theorem (Equivalence)
For any fixed $\sigma > 0$,

$$\mathcal{L}_{\mathrm{SM}}(\phi;\sigma) = \mathcal{L}_{\mathrm{DSM}}(\phi;\sigma) + C,$$

with constant C independent of ϕ.
Both losses share the same minimizer $\mathbf{s}^*(\cdot;\sigma)$.

Special Case: Additive Gaussian Noise

Let

$$x \sim p_{\text{data}}, \quad \tilde{x} = x + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I),$$

so

$$p_\sigma(\tilde{x} \mid x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I).$$

Then the conditional score is analytic:

$$\nabla_{\tilde{x}} \log p_\sigma(\tilde{x} \mid x) = \frac{x - \tilde{x}}{\sigma^2}.$$

Plugging into DSM gives:

$$\begin{aligned}\mathcal{L}_{\text{DSM}}(\phi; \sigma) &= \tfrac{1}{2}\, \mathbb{E}_{x,\tilde{x}|x}\left[\left|\left|\mathbf{s}_\phi(\tilde{x}; \sigma) - \tfrac{x-\tilde{x}}{\sigma^2}\right|\right|_2^2\right] \\ &= \tfrac{1}{2}\, \mathbb{E}_{x,\epsilon}\left[\left|\left|\mathbf{s}_\phi(x + \sigma\epsilon; \sigma) + \tfrac{\epsilon}{\sigma}\right|\right|_2^2\right].\end{aligned}$$

This is precisely the **denoising score matching objective** used in score-based diffusion models.

Training

Algorithm: Training Denoising Score Matching Model

Require: Dataset $\mathcal{D}$, noise level σ, number of epochs N_{epochs}

Ensure: Trained score network s_θ

Initialize neural network s_θ to estimate $\nabla_x \log p(x)$

Initialize optimizer with parameters θ

for epoch $= 1$ to N_{epochs} **do**

  $\mathcal{L}_{epoch} \leftarrow 0$

  for batch $\mathbf{x} \in \mathcal{D}$ **do**

    Sample noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$

    Create noisy samples $\tilde{\mathbf{x}} \leftarrow \mathbf{x} + \sigma\boldsymbol{\epsilon}$

    Compute score estimate $\hat{\mathbf{s}} \leftarrow s_\theta(\tilde{\mathbf{x}})$

    Compute target score $\mathbf{s}^* \leftarrow -\boldsymbol{\epsilon}/\sigma^2$

    Compute DSM loss $\mathcal{L} \leftarrow \mathbb{E}_{\mathbf{x},\boldsymbol{\epsilon}}\left[\|s_\theta(\mathbf{x} + \sigma\boldsymbol{\epsilon}) + \boldsymbol{\epsilon}/\sigma\|^2\right]$

    Gradient update $\theta \leftarrow \theta - \eta\nabla_\theta\mathcal{L}$

    $\mathcal{L}_{epoch} \leftarrow \mathcal{L}_{epoch} + \mathcal{L}$

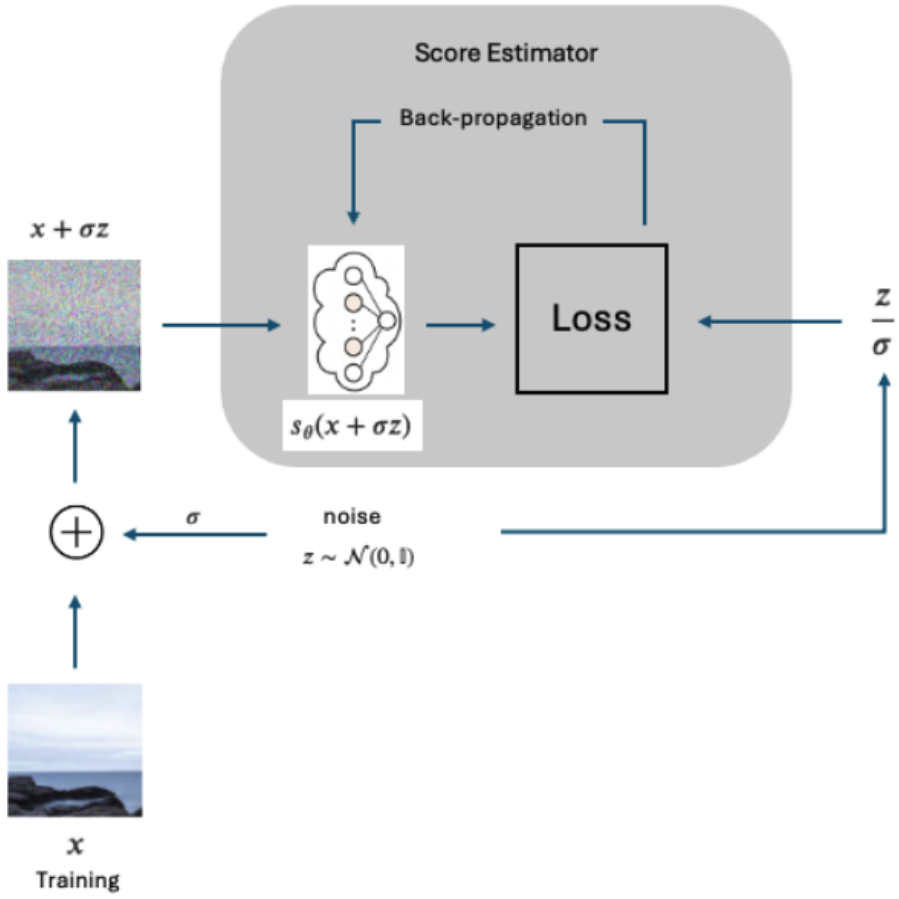
  end for

  $\mathcal{L}_{avg} \leftarrow \mathcal{L}_{epoch}/N_{batches}$

  $\log(\text{epoch}, \mathcal{L}_{avg})$

end for

return s_θ



Sampling/Inference

Algorithm: Sampling from Denoising Score Matching Model

Require: Score network s_θ, data dimension d, number of samples N, steps T, step size η

Ensure: Generated samples

Initialize $\mathbf{x}_0 \sim \mathcal{N}(0, \mathbf{I})$ of shape $[N, d]$

for $t = 0$ to $T - 1$ **do**

  Compute score $\mathbf{s}_t \leftarrow s_\theta(\mathbf{x}_t)$

  Sample noise $\mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$

  Update samples $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \frac{\eta}{2}\mathbf{s}_t + \sqrt{\eta}\mathbf{z}_t$

  if t mod log_interval $= 0$ **then**

    $\log(t, T)$

  end if

end for

return $\mathbf{x}_T$

Multi-Noise Levels of Denoising Score Matching

- **Problem with a single noise level**
  - Add Gaussian noise with a fixed variance to data → smoothed distribution.
  - **Low noise:** Langevin dynamics struggles to move between modes
    → gradients vanish in low-density regions.
  - **High noise:** sampling easier, but model only captures **coarse structure**
    → blurry samples with little fine detail.
  - In high dimensions, Langevin dynamics:
    - can converge very slowly,
    - can get stuck on plateaus / saddle points,
    - is sensitive to initialization.
- **Idea:** inject **multiple noise levels** and train a **noise-conditional score network (NCSN)**, then sample with **annealed Langevin dynamics** to combine global exploration (high noise) with fine details (low noise).

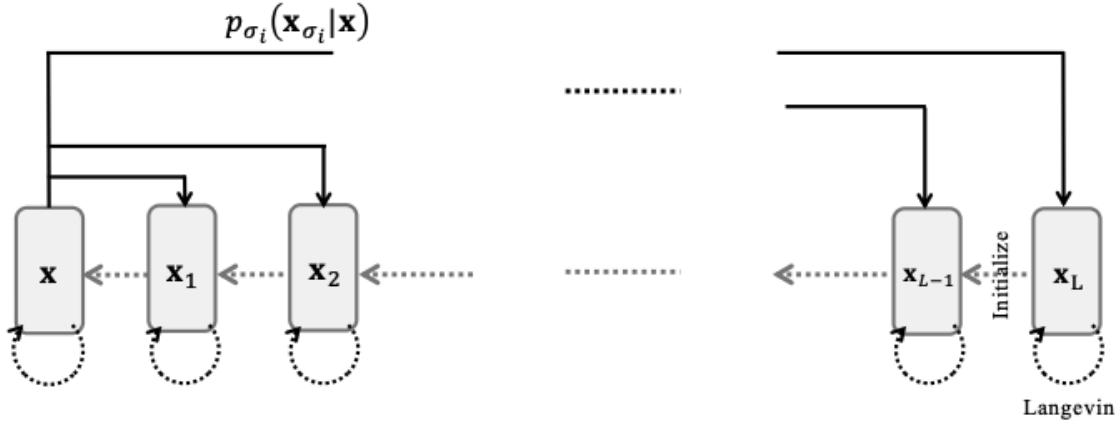


Figure 3.7: Illustration of NCSN. The forward process perturbs the data with multiple levels of additive Gaussian noise $p_\sigma(\mathbf{x}_\sigma|\mathbf{x})$. Generation proceeds via Langevin sampling at each noise level, using the result from the current level to initialize sampling at the next lower variance.

NCSN Objective: Multi-Noise DSM

- Train a **noise-conditional score network**

$$s_\phi(x, \sigma) \approx \nabla_x \log p_\sigma(x)$$

  for all noise levels $\sigma \in \{\sigma_i\}_{i=1}^{L}$.

- **Overall NCSN loss:**

$$\mathcal{L}_{\mathrm{NCSN}}(\phi) := \sum_{i=1}^{L} \lambda(\sigma_i)\, \mathcal{L}_{\mathrm{DSM}}(\phi; \sigma_i),$$

  where $\lambda(\sigma_i) > 0$ weights each scale.

- **DSM loss at a fixed noise level σ:**
  - Sample $x \sim p_{\mathrm{data}},\ \tilde{x} \sim p_\sigma(\cdot \mid x)$.
  - Penalize the error between the network and the true score:

$$\mathcal{L}_{\mathrm{DSM}}(\phi; \sigma) = \frac{1}{2}\, \mathbb{E}\Big[\big\| s_\phi(\tilde{x}, \sigma) - \frac{x - \tilde{x}}{\sigma^2} \big\|_2^2 \Big].$$

- Minimizing this is just **denoising score matching at each noise level**; the optimum satisfies

$$s^*(x, \sigma) = \nabla_x \log p_\sigma(x), \quad \forall \sigma \in \{\sigma_i\}_{i=1}^{L}.$$

Relationship to DDPM Denoising Loss

- Let $x_\sigma = x + \sigma\varepsilon$ with $\varepsilon \sim \mathcal{N}(0, I)$ and marginal $p_\sigma(x_\sigma)$.
- By **Tweedie's formula**:

$$\nabla_{x_\sigma} \log p_\sigma(x_\sigma) = -\frac{1}{\sigma}\,\mathbb{E}[\varepsilon \mid x_\sigma].$$

- Define:
  - **Score optimum (NCSN):**

$$s^*(x_\sigma, \sigma) = \nabla_{x_\sigma} \log p_\sigma(x_\sigma).$$

  - **Optimal DDPM denoiser:**

$$\varepsilon^*(x_\sigma, \sigma) = \mathbb{E}[\varepsilon \mid x_\sigma].$$

- They are exactly equivalent via:

$$s^*(x_\sigma, \sigma) = -\frac{1}{\sigma}\,\varepsilon^*(x_\sigma, \sigma), \qquad \varepsilon^*(x_\sigma, \sigma) = -\sigma\, s^*(x_\sigma, \sigma).$$

- In DDPM with discrete noise levels σ_i, the relation becomes

$$s^*(x_i, i) = -\frac{1}{\sigma_i}\,\mathbb{E}[\varepsilon_i \mid x_i].$$

- **Conclusion:** DDPM's conditional denoiser for ε is just a **re-parameterization** of the score at that noise level.

Sampling with NCSN: Annealed Langevin Dynamics

- After training, we have score networks at multiple noise levels:

$$s_\phi(\cdot, \sigma_1),\ \ldots,\ s_\phi(\cdot, \sigma_L).$$

- **Annealed Langevin sampler:**

  1. Initialize at highest noise level:
     $x_0 \sim \mathcal{N}(0, I)$ at σ_L.
  2. For each noise level σ_ℓ from σ_L down to σ_1:
     - Run K steps of Langevin dynamics:

$$x_{k+1} = x_k + \eta_\ell\, s_\phi(x_k, \sigma_\ell) + \sqrt{2\eta_\ell}\,\varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, I).$$

     - Use the final x_K at σ_ℓ as initialization for $\sigma_{\ell-1}$.
  3. At the lowest noise level σ_1, the final sample $x^{(1)}$ is returned.
- The step size typically scales with noise:

$$\eta_\ell = \delta \cdot \frac{\sigma_\ell^2}{\sigma_1^2}, \quad \delta > 0.$$

- Intuition:
  - High noise $\rightarrow$ easy global exploration.
  - Gradual decrease of noise $\rightarrow$ refine into sharp, detailed samples.

Limitation: Slow Sampling in NCSN

- NCSN uses **annealed MCMC** (Langevin dynamics) across all noise levels $\{\sigma_i\}_{i=1}^{L}$.
- For each σ_i, we perform K updates of the form:

$$\text{“update } x_n \text{ using } s_\phi(x_n, \sigma_i) + \text{small random perturbation.”}$$

- Why we need many steps (large $L \times K$):
  1. **Local accuracy & stability**
     - Learned scores are reliable only for **small perturbations**.
     - Requires **small step sizes** and **many iterations** per noise level to avoid bias or instability.
  2. **Slow mixing in high dimensions**
     - Local MCMC moves explore **multi-modal, high-dimensional** targets poorly.
     - Many iterations needed to reach typical regions of the distribution.
- Overall cost:
  - Each update needs a **forward pass** through the network.
  - Completely sequential dependence between steps.
  - Total complexity:

$$\mathcal{O}(LK)$$

network evaluations $\rightarrow$ **computationally slow sampling**.

Summary: A Comparative View of NCSN and DDPM

Comparison.

We can line up NCSN (Noise Conditional Score Networks) and DDPM (Denoising Diffusion Probabilistic Models) side by side as follows.

Quantity	NCSN	DDPM
$x_{i+1} \mid x_i$	$x_{i+1} = x_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\,\epsilon$	$x_{i+1} = \sqrt{1-\beta_i}\,x_i + \sqrt{\beta_i}\,\epsilon$
$x_i \mid x$	$x_i = x + \sigma_i \epsilon$	$x_i = \bar{\alpha}_i x + \sqrt{1-\bar{\alpha}_i}\,\epsilon$
p_{prior}	$\mathcal{N}(0,\, \sigma_1^2 I)$	$\mathcal{N}(0, I)$
Loss	$\mathbb{E}_i \mathbb{E}_{x\sim p_{\text{data}}} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\left\| s_\phi(x_i, \sigma_i) + \frac{\epsilon}{\sigma_i} \right\|_2^2 \right]$	$\mathbb{E}_i \mathbb{E}_{x\sim p_{\text{data}}} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \left[\left\| \epsilon_\phi(x_i, i) - \epsilon \right\|_2^2 \right]$
Sampling	Run Langevin dynamics at each noise level; use the result as initialization for the next level	Follow the Markov chain backward with transition $p_\phi(x_{i-1} \mid x_i)$

A Shared Bottleneck

Even though NCSN and DDPM are formulated differently, both require a **finely discretized time grid**.
This means:

- Many diffusion steps (often hundreds or thousands) are needed.
- Sampling is therefore slow and computationally heavy.

Question

> *How can we make sampling in diffusion models faster?*

Diffusion Models Today: Score SDE Framework

There is only one precise way of presenting the laws, and that is by means of differential equations. They have the advantage of being fundamental and, so far as we know, precise.

Richard P. Feynman

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

Jascha Sohl-Dickstein et al (2015), Deep Unsupervised Learning using Nonequilibrium Thermodynamics

SCORE-BASED GENERATIVE MODELING THROUGH STOCHASTIC DIFFERENTIAL EQUATIONS

Yang Song*
Stanford University
yangsong@cs.stanford.edu

Jascha Sohl-Dickstein
Google Brain
jaschasd@google.com

Diederik P. Kingma
Google Brain
durk@google.com

Abhishek Kumar
Google Brain
abhishk@google.com

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

Ben Poole
Google Brain
pooleb@google.com

ABSTRACT

Creating noise from data is easy; creating data from noise is generative modeling. We present a stochastic differential equation (SDE) that smoothly transforms a complex data distribution to a known prior distribution by slowly injecting noise, and a corresponding reverse-time SDE that transforms the prior distribution back into the data distribution by slowly removing the noise. Crucially, the reverse-time SDE depends only on the time-dependent gradient field (a.k.a., score) of the perturbed data distribution. By leveraging advances in score-based generative modeling, we can accurately estimate these scores with neural networks, and use numerical SDE solvers to generate samples. We show that this framework encapsulates previous approaches in score-based generative modeling and diffusion probabilistic modeling, allowing for new sampling procedures and new modeling capabilities. In particular, we introduce a predictor-corrector framework to correct errors in the evolution of the discretized reverse-time SDE. We also derive an equivalent neural ODE that samples from the same distribution as the SDE, but additionally enables exact likelihood computation, and improved sampling efficiency. In addition, we provide a new way to solve inverse problems with score-based models, as demonstrated with experiments on class-conditional generation, image inpainting, and colorization. Combined with multiple architectural improvements, we achieve record-breaking performance for unconditional image generation on CIFAR-10 with an Inception score of 9.89 and FID of 2.20, a competitive likelihood of 2.99 bits/dim, and demonstrate high fidelity generation of 1024×1024 images for the first time from a score-based generative model.

Yang Song et al (2021), Score-Based Generative Modeling Through Stochastic Differential Equations

Motivation: From Discrete to Continuous-Time Processes

Discrete forward noise injection

NCSN

- Noise levels: $\{\sigma_i\}_{i=1}^{L}$
- Clean sample $\mathbf{x} \sim p_{\text{data}}$ is perturbed as

$$\mathbf{x}_{\sigma_i} = \mathbf{x} + \sigma_i \boldsymbol{\epsilon}_i, \qquad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

On a discrete time grid with step (\Delta t):

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \sqrt{\sigma_{t+\Delta t}^2 - \sigma_t^2}\, \boldsymbol{\epsilon}_t \approx \mathbf{x}_t + \sqrt{\frac{d\sigma_t^2}{dt}\, \Delta t}\, \boldsymbol{\epsilon}_t.$$

Unified Euler-Maruyama form

Both schemes fit

$$\mathbf{x}_{t+\Delta t} \approx \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\, \Delta t + g(t)\sqrt{\Delta t}\, \boldsymbol{\epsilon}_t, \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

NCSN

$$\mathbf{f}(\mathbf{x}, t) = \mathbf{0}, \qquad g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}.$$

DDPM

$$\mathbf{f}(\mathbf{x}, t) = -\tfrac{1}{2}\beta(t)\mathbf{x}, \qquad g(t) = \sqrt{\beta(t)}.$$

DDPM

- Variance schedule: $\{\beta_i\}_{i=1}^{L}$
- Forward update:

$$\mathbf{x}_{t+\Delta t} = \sqrt{1-\beta_t}\, \mathbf{x}_t + \sqrt{\beta_t}\, \boldsymbol{\epsilon}_t, \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

Assume $\beta_t = \beta(t)\Delta t$ is small; then

$$\sqrt{1-\beta_t} \approx 1 - \tfrac{1}{2}\beta_t = 1 - \tfrac{1}{2}\beta(t)\Delta t,$$

so

$$\mathbf{x}_{t+\Delta t} \approx \mathbf{x}_t - \tfrac{1}{2}\beta(t)\mathbf{x}_t\, \Delta t + \sqrt{\beta(t)\Delta t}\, \boldsymbol{\epsilon}_t.$$

Continuous-time limit: SDE

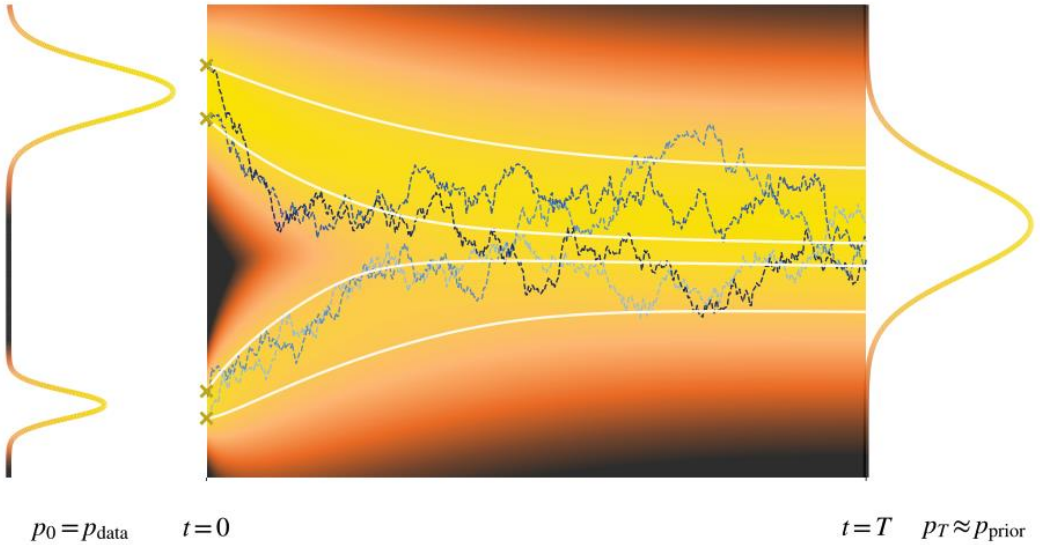
As $\Delta t \to 0$, the discrete-time process converges to the Itô SDE

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\, dt + g(t)\, d\mathbf{w}(t),$$

where $\mathbf{w}(t)$ is a standard Wiener process (Brownian motion).

This gives a **continuous-time description** of the forward noising dynamics for both NCSN and DDPM.

Forward Time SDEs: from data to noise



$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}(t), \quad \mathbf{x}(0) \sim p_{\text{data}}$$

Here, $\mathbf{f}(\cdot, t)\colon \mathbb{R}^D \to \mathbb{R}^D$ is the drift, $g(t) \in \mathbb{R}$ is the scalar diffusion coefficient, and $\mathbf{w}(t)$ denotes a standard Wiener process. We refer to this as the *forward SDE*, which describes how clean data is gradually perturbed into noise over time.

Forward SDE Diffusion Process

Forward SDE

We model the *forward diffusion* of data with an SDE

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\, dt + g(t)\, d\mathbf{w}(t),$$

where $\mathbf{w}(t)$ is a standard Wiener process.

Perturbation kernels $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$

- The **conditional law**

  $$p_t(\mathbf{x}_t \mid \mathbf{x}_0)$$

  tells us how a clean data point $\mathbf{x}_0 \sim p_{\text{data}}$ evolves into its noisy version $\mathbf{x}_t$ at time t.

- In general, the drift $\mathbf{f}(\mathbf{x}, t)$ can be arbitrary, but a very convenient choice is an **affine drift**

  $$\mathbf{f}(\mathbf{x}, t) = f(t)\, \mathbf{x},$$

  where $f(t)$ is a scalar (typically non–positive).

Closed-form conditional distribution (affine drift)

Under this affine drift, the process stays Gaussian for all t, and

$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_t;\, \mathbf{m}(t),\, P(t)\mathbf{I}_D\big),$$

with mean and variance

$$\mathbf{m}(t) = \exp\left(\int_0^t f(u)\, du\right)\mathbf{x}_0, \qquad P(t) = \int_0^t \exp\left(2\int_s^t f(u)\, du\right) g^2(s)\, ds,$$

and initial conditions

$$\mathbf{m}(0) = \mathbf{x}_0, \qquad P(0) = 0.$$

This **simulation-free** form lets us sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$ without numerically integrating the SDE.

Marginal densities $p_t(\mathbf{x}_t)$

The time-marginal density is obtained by integrating over the perturbation kernel:

$$p_t(\mathbf{x}_t) := \int p_t(\mathbf{x}_t \mid \mathbf{x}_0)\, p_{\text{data}}(\mathbf{x}_0)\, d\mathbf{x}_0, \qquad p_0 = p_{\text{data}}.$$

Choosing $f(t)$ and $g(t)$ appropriately:

- The forward process gradually adds noise.
- The influence of the initial state $\mathbf{x}_0$ is eventually forgotten.

As $T \to \infty$,

$$\mathbf{m}(T) = \exp\left(\int_0^T f(u)\, du\right)\mathbf{x}_0 \;\longrightarrow\; 0,$$

provided $f(u)$ is non–positive so that the exponential decays.

At the same time, the variance $P(t)$ grows and stabilizes to match a chosen **prior distribution** p_{prior} (typically Gaussian). Then

$$p_T(\mathbf{x}_T) = \int p_T(\mathbf{x}_T \mid \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0)\, d\mathbf{x}_0 \approx p_{\text{prior}}(\mathbf{x}_T),$$

and

$$p_T(\mathbf{x}_T \mid \mathbf{x}_0) \approx p_{\text{prior}}(\mathbf{x}_T).$$

Intuition

- The forward diffusion SDE **maps any data distribution** p_{data} into a simple, tractable prior p_{prior}.
- This gives a convenient starting point for the **reverse-time process**, which we use for *generation*.

Variance – Exploding (VE) vs Variance – Preserving (VP)

We start from the forward SDE

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t)\, dt + g(t)\, d\mathbf{w}(t),$$

and choose $\mathbf{f}, g$ so that the **variance over time** behaves as we want.

Variance–Exploding (VE) SDE

- Drift: $\mathbf{f}(\mathbf{x}, t) = \mathbf{0}$
- Diffusion: $g(t) = \sqrt{\dfrac{d\sigma^2(t)}{dt}}$

With affine drift $\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x}$ and $f(t) = 0$, the conditional kernel is

$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_0,\ \sigma^2(t)\mathbf{I}\big), \quad \sigma^2(t) = \int_0^t g^2(s)\, ds.$$

- Mean stays at $\mathbf{x}_0$
- Variance grows:
  $\mathrm{Cov}[\mathbf{x}_t] = \mathrm{Cov}[\mathbf{x}_0] + \sigma^2(t)\mathbf{I}$

Noise level **explodes** with t.
This is the continuous-time view of **NCSN / VE SDE**.

Variance–Preserving (VP) SDE

- Drift: $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$
- Diffusion: $g(t) = \sqrt{\beta(t)}$

For 1D, the variance $v(t) = \mathrm{Var}[x_t]$ solves

$$\frac{dv}{dt} = -\beta(t)v(t) + \beta(t),$$

whose solution tends to

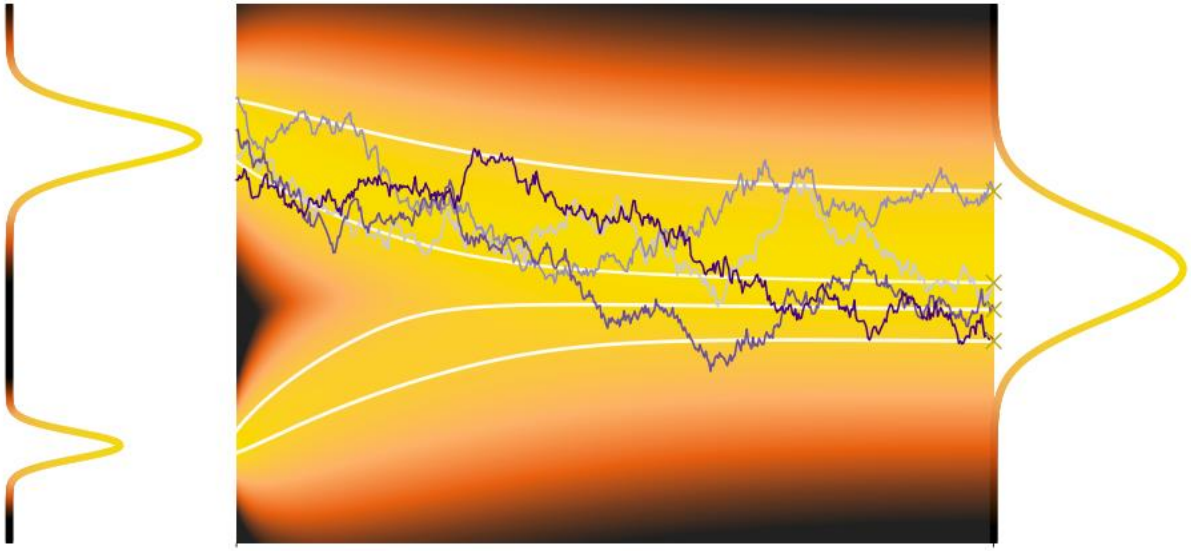
$$v(t) \to 1 \quad \text{as } t \text{ increases}$$

(assuming we choose the schedule appropriately).

- Drift gently pulls toward 0
- Noise pushes outward with the same rate

Overall variance is kept $O(1)$: it is **variance–preserving**.
This is the continuous-time limit of **DDPM / VP SDE**.

Reverse Time Stochastic Process for Generation



$p_0 = p_{\text{data}}$ $t=0$ $t=T$ $p_T \approx p_{\text{prior}}$

$$\mathrm{d}\bar{\mathbf{x}}(t) = \left[\mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))\right] \mathrm{d}t + g(t)\, \mathrm{d}\bar{\mathbf{w}}(t),$$
$$\bar{\mathbf{x}}(T) \sim p_{\text{prior}} \approx p_T.$$

Here, $\bar{\mathbf{w}}(t)$ denotes a standard Wiener process in reverse time, defined as $\bar{\mathbf{w}}(t) := \mathbf{w}(T-t) - \mathbf{w}(T)$.

Stochastic Processes and their Applications 12 (1982) 313–326
North-Holland Publishing Company

REVERSE-TIME DIFFUSION EQUATION MODELS*

Brian D.O. ANDERSON**
Department of Electrical Engineering, The University of Newcastle, N.S.W. 2308, Australia

Received 13 November 1979
Revised 12 May 1980

Reverse-time stochastic diffusion equation models are defined and it is shown how most processes defined via a forward-time or conventional diffusion equation model have an associated reverse-time model.

Stochastic equations
reverse-time equations
Fokker–Planck equations
diffusion equations
Kolmogorov equations

source

Score SDE: its training

We approximate the oracle score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ using a **time conditioned neural network**

$$s_\phi = s_\phi(\mathbf{x}, t)$$

across all $t \in [0, T]$ by minimizing a score matching objective:

$$\mathcal{L}_{\text{SM}}(\phi; \omega(\cdot)) := \frac{1}{2}\, \mathbb{E}_{t \sim p_{\text{time}}}\, \mathbb{E}_{\mathbf{x}_t \sim p_t} \Big[\omega(t) \left\| s_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \right\|_2^2 \Big],$$

where p_{time} is some time distribution (e.g. uniform on $[0, T]$), and $\omega(\cdot)$ is a time–weighting function.

To avoid relying on the intractable oracle score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we instead use the **Denoising Score Matching (DSM)** loss.

Conditioned on a data point $\mathbf{x}_0$, this approach uses the analytically tractable score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$.

Concretely, we optimize the loss

$$\boxed{\mathcal{L}_{\text{DSM}}(\phi; \omega(\cdot)) := \frac{1}{2}\, \mathbb{E}_t\, \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}}\, \mathbb{E}_{\mathbf{x}_t \sim p_t(\mathbf{x}_t \mid \mathbf{x}_0)} \Big[\omega(t) \left\| s_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0) \right\|_2^2 \Big]}$$

where $\mathbf{x}_0 \sim p_{\text{data}}$. Equation can be interpreted as the **continuous time counterpart** of , with the summation in the discrete case replaced by integration.

Score SDE: its training

In practice:

- We replace $\nabla_x \log p_t(x)$ by the **learned score** $s_\phi(x, t)$.
- We simulate this SDE backwards with **Euler–Maruyama**:

$$x_{t-\Delta t} \leftarrow x_t + \left[f(x_t, t) - g^2(t) s_\phi(x_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, I).$$

This gives a valid sampling procedure; but it is **stochastic** and **slow**. In fact, reverse-time SDE sampling has some drawbacks:

- Each step uses fresh **random noise** ϵ.
- To follow the trajectory accurately we need **many small steps** (thousands).
- Paths are random:
  same x_T and same model $\rightarrow$ different trajectories each run.
- Numerically, SDE solvers are less standard than ODE solvers.

Idea:

> If we can find a **deterministic** process that has the **same marginals** p_t as the forward SDE, we can sample with an ODE instead.

This is where the **Fokker–Planck equation** and the **Probability Flow ODE (PF-ODE)** come in.

Fokker–Planck and Probability Flow ODE

The **Fokker–Planck equation** tells us how the marginal densities $p_t(x)$ of the forward SDE evolve:

$$\partial_t p_t(x) = -\nabla_x \cdot \big[f(x,t)p_t(x)\big] + \tfrac{1}{2}g^2(t)\,\Delta_x p_t(x).$$

We can **design an ODE** with velocity field

$$v(x,t) = f(x,t) - \tfrac{1}{2}g^2(t)\,\nabla_x \log p_t(x)$$

such that its solution has **exactly the same marginals (p_t)**.

This ODE is the **Probability Flow ODE (PF-ODE)**:

$$\frac{d\tilde{x}(t)}{dt} = v(\tilde{x}(t),t) = f(\tilde{x}(t),t) - \tfrac{1}{2}g^2(t)\,\nabla_x \log p_t(\tilde{x}(t)).$$

Again we replace the true score by the learned one:

$$\frac{d\tilde{x}_\phi^{\mathrm{ODE}}(t)}{dt} = f(\tilde{x}_\phi^{\mathrm{ODE}}(t),t) - \tfrac{1}{2}g^2(t)\,s_\phi(\tilde{x}_\phi^{\mathrm{ODE}}(t),t).$$

Sampling Algorithms

Reverse-Time SDE (stochastic)

1. Sample $x_T \sim p_{\text{prior}}$.
2. For $t = T, T - \Delta t, \ldots, \Delta t$:

$$x_{t-\Delta t} \leftarrow x_t + \left[f(x_t, t) - g^2(t) s_\phi(x_t, t)\right] \Delta t + g(t)\sqrt{\Delta t}\, \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I).$$

3. Output x_0 as a **random** sample.

Probability Flow ODE (deterministic)

1. Sample $x_T \sim p_{\text{prior}}$.
2. Solve the ODE backwards from $t = T$ to $t = 0$:

$$\frac{dx(t)}{dt} = f(x(t), t) - \tfrac{1}{2} g^2(t)\, s_\phi(x(t), t), \quad x(T) = x_T.$$

   Using any ODE solver (Euler, RK4, etc.):

$$x_{t-\Delta t} \leftarrow x_t + \left[f(x_t, t) - \tfrac{1}{2} g^2(t) s_\phi(x_t, t)\right] \Delta t.$$

3. Output x_0 as a **deterministic** function of x_T.

Both methods produce samples whose distribution **approximates** p_{data}, because they share the same family of marginals p_t.

Key Takeaways

- The **forward SDE** gradually turns data into a simple prior.
- We learn the **score of the intermediate marginals** p_t.
- For generation, we run dynamics **backwards**:
  - either with a **reverse SDE** (stochastic paths),
  - or with a **Probability Flow ODE** (deterministic paths).
- The **Fokker–Planck equation** guarantees that both processes have the **same marginals**.
- Conceptual slogan:

> *Sampling from diffusion models = solving a corresponding SDE or probability flow ODE.*

This viewpoint explains why diffusion sampling is slow (many solver steps) and why improving **numerical solvers** can directly speed up generation.

Beyond Monte Carlo

- **Key methodological twist in this paper**
  - Because the forward DSDE is **linear** with **Gaussian transitions** and the score network is chosen in a simple parametric form, the expectations in $L_{X,h}^{\mathrm{DSM}}(\theta)$ can be reduced to **low-dimensional Gaussian integrals**.
  - These integrals are then evaluated **deterministically**:
    - Simpson's rule over time t,
    - Gauss–Hermite quadrature over the Gaussian variables.
  - Result: **no Monte Carlo noise** in the DSM objective, and training reduces to minimizing a *deterministically approximated* Lesniewski-style DSM loss instead of a stochastic one.

$$
\begin{aligned}
\mathcal{L}^{DSM}(\theta) =& \frac{1}{2n}\sum_{i=1}^{n}\int_0^1 \lambda_0(t)\int_{\mathbb{R}^d}\|K(x;\theta)+x_i-\mu(t,x_i)\|^2\,\mathfrak{f}_t^i(x)dx\,dt\\
=& \frac{1}{2n}\sum_{i=1}^{n}\sum_{k=1}^{d}\int_0^1 \lambda_0(t)(x_i^k-\mu(t,x_i)^k+d^k)^2dt\\
&+\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{h}\sum_{k=1}^{d}c^{jk}\int_0^1\lambda_0(t)(x_i^k-\mu(t,x_i)^k+d^k)\int_{\mathbb{R}^d}\mathfrak{a}(\langle w_j|x\rangle+b_j)\,\mathfrak{f}_t^i(x)dx\,dt\\
&+\frac{1}{2n}\sum_{i=1}^{n}\sum_{j,k=1}^{h}c^{ij}c^{ik}\int_0^1\lambda_0(t)\int_{\mathbb{R}^d}\mathfrak{a}(\langle w_j|x\rangle+b_j)\mathfrak{a}(\langle w_k|x\rangle+b_k)\,\mathfrak{f}_t^i(x)dx\,dt,
\end{aligned}
$$

where $\mathfrak{f}_t^i(x)$ is given by (75), and where $w_j, w_k \in \mathbb{R}^d$ denote the j-th and k-th rows of the matrix w, respectively. Notice that, with probability one, w_j and w_k, are linearly independent for $j \neq k$. In order to organize the computation, we rewrite this expression as follows:

$$
\begin{aligned}
\mathcal{L}^{DSM}(\theta) =& \frac{1}{2n}\sum_{i=1}^{n}\sum_{k=1}^{d}\int_0^1 \lambda_0(t)(x_i^k-\mu(t,x_i)^k+d^k)^2dt\\
&+\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{h}\sum_{k=1}^{d}c^{jk}\int_0^1\lambda_0(t)(x_i^k-\mu(t,x_i)^k+d^k)\mathfrak{I}_1^i(w_j,b_j,t)\,dt \qquad (80)\\
&+\frac{1}{2n}\sum_{i=1}^{n}\sum_{j,k=1}^{h}c^{ij}c^{ik}\int_0^1\lambda_0(t)\mathfrak{I}_2^i(w_j,w_k,b_j,b_k,t)\,dt,
\end{aligned}
$$

where

$$
\mathfrak{I}_1^i(w,b,t) = \int_{\mathbb{R}^d}\mathfrak{a}\big(\langle w|x\rangle+b\big)\mathfrak{f}_t^i(x)dx, \qquad (81)
$$

and

$$
\mathfrak{I}_2^i(w_1,w_2,b_1,b_2,t) = \int_{\mathbb{R}^d}\mathfrak{a}\big(\langle w_1|x\rangle+b_1\big)\mathfrak{a}\big(\langle w_2|x\rangle+b_2\big)\mathfrak{f}_t^i(x)dx, \qquad (82)
$$

Beyond Monte Carlo: Harnessing Diffusion Models to Simulate Financial Market Dynamics

Andrew Lesniewski and Giulio Trigila
Department of Mathematics
Baruch College
One Bernard Baruch Way
New York, NY 10010
USA

February 4, 2025

Abstract

We propose a highly efficient and accurate methodology for generating synthetic financial market data using a diffusion model approach. The synthetic data produced by our methodology align closely with observed market data in several key aspects: (i) they pass the two-sample Cramer - von Mises test for portfolios of assets, and (ii) Q - Q plots demonstrate consistency across quantiles, including in the tails, between observed and generated market data. Moreover, the covariance matrices derived from a large set of synthetic market data exhibit significantly lower condition numbers compared to the estimated covariance matrices of the observed data. This property makes them suitable for use as regularized versions of the latter. For model training, we develop an efficient and fast algorithm based on numerical integration rather than Monte Carlo simulations. The methodology is tested on a large set of equity data.

source

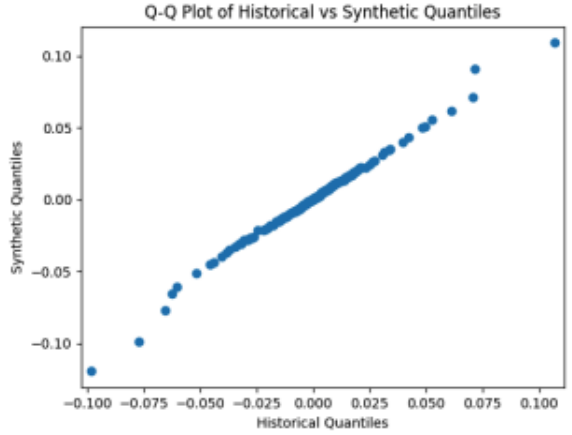


Figure 12: Q-Q plot of the historical versus synthetic returns in Experiment 6.

Diffusion to denoise covariance

Pipeline overview. Let $X \in \mathbb{R}^{T \times N}$ be the matrix of (demeaned) returns for N assets over T periods.

- **Training.** Fit a DGM (DDPM or FM) on X to learn a generative prior p_θ over returns.
- **Synthesis.** Sample $\widetilde{X} \in \mathbb{R}^{\widetilde{T} \times N}$ from p_θ with $\widetilde{T} \geq T$ to reduce Monte Carlo error.
- **Re-estimation.** Compute a PSD covariance estimate from the synthetic panel, e.g. $\widehat{C}_{\mathrm{DGM}} = \mathrm{corr}(\widetilde{X})$, $\widehat{D} = \mathrm{diag}(\widehat{\Sigma}_{\mathrm{emp}})$ or $\mathrm{diag}(\mathrm{var}(\widetilde{X}))$ and set $\widehat{\Sigma}_{\mathrm{DGM}} = \widehat{D}^{1/2}\widehat{C}_{\mathrm{DGM}}\widehat{D}^{1/2}$ to ensure positive semidefiniteness and control the marginal scales.
- **Hybrid (optional).** Apply Marčenko–Pastur (MP) eigenvalue cleaning to $\widehat{\Sigma}_{\mathrm{DGM}}$ to obtain $\widehat{\Sigma}_{\mathrm{DGM+MP}}$.

Table 1: RMSE statistics (means μ_{RMSE} and standard deviations σ_{RMSE}) for Maximum Sharpe and Minimum Variance portfolios, together with condition-number statistics (means μ_κ and standard deviations σ_κ) of the estimated covariance matrix, across denoising methods.

	Maximum Sharpe RMSE		Minimum Variance RMSE		Condition Number κ	
Method	μ_{RMSE}	σ_{RMSE}	μ_{RMSE}	σ_{RMSE}	μ_κ	σ_κ
Empirical	0.22	0.02	0.0089	0.0013	102.76	5.97
MP	0.199	0.01	0.0056	0.0008	94.23	4.42
DDPM	0.16	0.0019	0.018	0.0008	15.54	0.86
DDPM + MP	0.166	0.007	0.036	0.0009	11.15	0.58
FM	0.146	0.000 37	0.055	0.0002	8.10	0.41
FM + MP	0.148	0.001	0.053	0.0001	5.80	0.26

Deep Generative Modeling for Covariance Denoising in Finance

Oualid Missaoui*
EimiLabs.ai
New York, NY, USA
Department of Mathematics, Baruch College (CUNY)
New York, NY , USA
oualid@eimilabs.ai

Abstract

Estimating asset return covariances is fragile in high dimensions; classical fixes (shrinkage, graphical models, Marčenko-Pastur (MP)) often suppress noise at the cost of distorting structure. While ill-posedness is obvious when the number of assets (N) exceeds the number of observations (T), it can still manifest even when $T > N$, especially in the presence of significant correlation, underlying driving factors, heavy-tailedness, or regime changes. We propose a data-level denoising approach using deep generative models (DGM) such as Denoising Diffusion Probabilistic Models (DDPM) and Flow Matching (FM), that learn a data driven prior from finite panels, synthesize returns, and yield Positive Semi Definite (PSD) covariances reestimated from the synthetic data. Our covariances reduce ill-conditioning and improve both global minimum variance and maximum Sharpe portfolios versus raw samples and MP; applying MP after DGM adds little benefit. Deep generative modeling thus offers a practical, scalable alternative to classical denoisers for portfolio construction. Code: `github.com/omroot/DGM4Covariance`.

source

Diffusion Factor Model

2.2 Asset Returns and Unknown Factor Structure

To improve sample efficiency, especially in data-scarce settings, the central idea is to incorporate domain knowledge into the diffusion model. Specifically, we leverage a key insight from the finance literature: a relatively small set of latent factors—reflecting both macroeconomic and firm-specific variables—can effectively explain a broad class of asset returns (Ross 2013, Fan, Liao, and Wang 2016, Aït-Sahalia and Xiu 2019, Giglio and Xiu 2021, Bryzgalova et al. 2023, Kelly, Malamud, and Pedersen 2023). Following these studies, we consider the asset return $\mathbf{R} \sim P_{\text{data}}$ satisfying the following factor model structure:

$$\mathbf{R} = \boldsymbol{\beta}\mathbf{F} + \boldsymbol{\varepsilon}, \tag{8}$$

where $\mathbf{F} \in \mathbb{R}^k$ are *unknown* factors with $k \ll d$, $\boldsymbol{\beta} \in \mathbb{R}^{d\times k}$ is a factor loading matrix, and $\boldsymbol{\varepsilon} \in \mathbb{R}^d$ is the vector of idiosyncratic residuals.

We want to emphasize that, while we assume the data distribution P_{data} follows a factor model structure (8), the implementation and analysis of the diffusion models *do not require observing* the factors. Instead, our approach is capable of uncovering the latent low-dimensional factor space through the data generation process; see Section 5 for more details.

Assumption 1 (Factor model). *We assume the following conditions on the factor model (8):*

(i) The factor loading $\boldsymbol{\beta} \in \mathbb{R}^{d\times k}$ has orthonormal columns.

(ii) The factor $\mathbf{F} \in \mathbb{R}^k$ follows a distribution that has a density function denoted as p_{fac} and has a finite second moment, i.e., $\int \|\mathbf{f}\|_2^2\, p_{\text{fac}}(\mathbf{f})\mathrm{d}\mathbf{f} < \infty$.

(iii) The residual $\boldsymbol{\varepsilon}$ is Gaussian with density $\phi(\cdot; \mathbf{0}, \mathrm{diag}\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2\})$ and there exists a positive constant $\sigma_{\max} > 0$ such that

$$\sigma_{\max} \geq \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d > 0.$$

(iv) $\mathbf{F}$ and $\boldsymbol{\varepsilon}$ are independent.

As a result, $\mathbf{R}$ has a positive definite covariance matrix, defined as

$$\boldsymbol{\Sigma}_0 := \mathbb{E}[\mathbf{R}\mathbf{R}^\top] - \mathbb{E}[\mathbf{R}]\mathbb{E}[\mathbf{R}]^\top. \tag{9}$$

Next, for an arbitrary time $t \in [0, T]$, we consider a linear subspace $\mathcal{V}_t$ spanned by the column vectors of $\boldsymbol{\Lambda}_t^{-\frac{1}{2}}\boldsymbol{\beta}$, with $\boldsymbol{\Lambda}_t$ defined as

$$\boldsymbol{\Lambda}_t := \mathrm{diag}\left\{h_t + \sigma_1^2\alpha_t^2, h_t + \sigma_2^2\alpha_t^2, \ldots, h_t + \sigma_d^2\alpha_t^2\right\}. \tag{10}$$

We further define $\mathbf{T}_t$ as the matrix of orthogonal projection onto $\mathcal{V}_t$:

$$\mathbf{T}_t := \boldsymbol{\Lambda}_t^{-\frac{1}{2}}\boldsymbol{\beta}\boldsymbol{\Gamma}_t\boldsymbol{\beta}^\top\boldsymbol{\Lambda}_t^{-\frac{1}{2}} \quad \text{with} \quad \boldsymbol{\Gamma}_t := (\boldsymbol{\beta}^\top\boldsymbol{\Lambda}_t^{-1}\boldsymbol{\beta})^{-1}. \tag{11}$$

Matrix $\boldsymbol{\Gamma}_t$ is well-defined as $\boldsymbol{\beta}^\top\boldsymbol{\Lambda}_t{}^{1}\boldsymbol{\beta}$ is positive definite due to Assumption 1. The following lemma presents the score decomposition.

Lemma 1. *Suppose Assumption 1 holds. The score function $\nabla \log p_t(\mathbf{r})$ can be decomposed into a subspace score and a complement score as*

$$\nabla \log p_t(\mathbf{r}) = \underbrace{\mathbf{T}_t\boldsymbol{\Lambda}_t^{\frac{1}{2}}\boldsymbol{\beta} \cdot \nabla \log p_t^{\text{fac}}(\boldsymbol{\beta}^\top\boldsymbol{\Lambda}_t^{\frac{1}{2}}\mathbf{T}_t \cdot \boldsymbol{\Lambda}_t^{-\frac{1}{2}}\mathbf{r})}_{\textit{Subspace score}} \underbrace{-\boldsymbol{\Lambda}_t^{-\frac{1}{2}}(\mathbf{I} - \mathbf{T}_t) \cdot \boldsymbol{\Lambda}_t^{-\frac{1}{2}}\mathbf{r}}_{\textit{Complement score}}, \tag{12}$$

where $p_t^{\text{fac}}(\cdot) := \int \phi(\cdot; \alpha_t\mathbf{f}, \boldsymbol{\Gamma}_t)p_{\text{fac}}(\mathbf{f})\mathrm{d}\mathbf{f}$ and $\boldsymbol{\Lambda}_t$, $\boldsymbol{\Gamma}_t$, $\mathbf{T}_t$ are defined in (10) and (11).

Diffusion Factor Models: Generating High-Dimensional Returns with Factor Structure

Minshuo Chen*, Renyuan Xu†, Yumin Xu†, and Ruixun Zhang§

First version: April 2025; This version: July 2025

Abstract

Financial scenario simulation is essential for risk management and portfolio optimization, yet it remains challenging especially in high-dimensional and small data settings common in finance. We propose a *diffusion factor model* that integrates latent factor structure into generative diffusion processes, bridging econometrics with modern generative AI to address the challenges of the curse of dimensionality and data scarcity in financial simulation. By exploiting the low-dimensional factor structure inherent in asset returns, we decompose the score function—a key component in diffusion models—using time-varying orthogonal projections, and this decomposition is incorporated into the design of neural network architectures. We derive rigorous statistical guarantees, establishing non-asymptotic error bounds for both score estimation at $\widetilde{\mathcal{O}}\left(d^{5/2}n^{-\frac{2}{k+5}}\right)$ and generated distribution at $\widetilde{\mathcal{O}}\left(d^{5/4}n^{-\frac{1}{2(k+5)}}\right)$, primarily driven by the intrinsic factor dimension k rather than the number of assets d, surpassing the dimension-dependent limits in the classical nonparametric statistics literature and making the framework viable for markets with thousands of assets. Numerical studies confirm superior performance in latent subspace recovery under small data regimes. Empirical analysis demonstrates the economic significance of our framework in constructing mean-variance optimal portfolios and factor portfolios. This work presents the first theoretical integration of factor structure with diffusion models, offering a principled approach for high-dimensional financial simulation with limited data. Our code is available at https://github.com/xymmmm00/diffusion_factor_model.

Keywords: Generative Modeling; Diffusion Model; Asset Return Generation; Factor Model; Error Bound; Portfolio Construction;

source

NIPS 2025 Best Paper

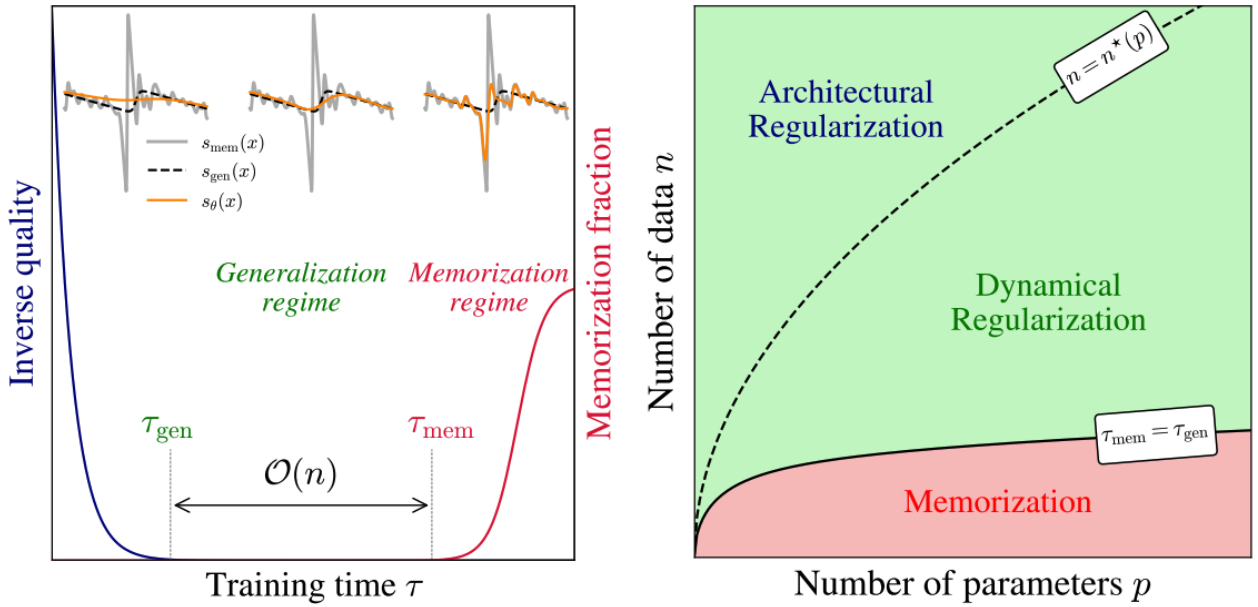


Figure 1: **Qualitative summary of our contributions.** *(Left)* Illustration of the training dynamics of a diffusion model. Depending on the training time τ, we identify three regimes measured by the inverse quality of the generated samples (blue curve) and their memorization fraction (red curve). The generalization regime extends over a large window of training times which increases with the training set size n. On top, we show a one dimensional example of the learned score function during training (orange). The gray line gives the exact empirical score, at a given noise level, while the black dashed line corresponds to the true (population) score. *(Right)* Phase diagram in the (n, p) plane illustrating three regimes of diffusion models: Memorization when n is sufficiently small at fixed p, Architectural Regularization for $n > n^\star(p)$ (which is model and dataset dependent, as discussed in [15, 25]), and Dynamical Regularization, corresponding to a large intermediate generalization regime obtained when the training dynamics is stopped early, i.e. $\tau \in [\tau_{\text{gen}}, \tau_{\text{mem}}]$.

Why Diffusion Models Don't Memorize: The Role of Implicit Dynamical Regularization in Training

Tony Bonnaire[†]
LPENS
Université PSL, Paris
tony.bonnaire@phys.ens.fr

Raphaël Urfin[†]
LPENS
Université PSL, Paris
raphael.urfin@phys.ens.fr

Giulio Biroli
LPENS
Université PSL, Paris
giulio.biroli@phys.ens.fr

Marc Mézard
Department of Computing Sciences
Bocconi University, Milano
marc.mezard@unibocconi.it

Abstract

Diffusion models have achieved remarkable success across a wide range of generative tasks. A key challenge is understanding the mechanisms that prevent their memorization of training data and allow generalization. In this work, we investigate the role of the training dynamics in the transition from generalization to memorization. Through extensive experiments and theoretical analysis, we identify two distinct timescales: an early time τ_{gen} at which models begin to generate high-quality samples, and a later time τ_{mem} beyond which memorization emerges. Crucially, we find that τ_{mem} increases linearly with the training set size n, while τ_{gen} remains constant. This creates a growing window of training times with n where models generalize effectively, despite showing strong memorization if training continues beyond it. It is only when n becomes larger than a model-dependent threshold that overfitting disappears at infinite training times. These findings reveal a form of implicit dynamical regularization in the training dynamics, which allow to avoid memorization even in highly overparameterized settings. Our results are supported by numerical experiments with standard U-Net architectures on realistic and synthetic datasets, and by a theoretical analysis using a tractable random features model studied in the high-dimensional limit.

Source

Implicit Dynamical Regularization

- **Question:** Why don't overparameterized diffusion models (DMs) simply *memorize* their training data?
- **Core answer:** Training dynamics induce an **implicit dynamical regularization** that delays memorization and creates a wide **generalization window**.
- **Two key timescales:**
  - **Generalization time** τ_{gen}
    – early time when samples become high quality and non-memorizing.
  - **Memorization time** τ_{mem}
    – later time when the model starts to reproduce training samples.
- **Scaling with data size** n**:**
  - τ_{gen} is essentially **independent of** n.
  - $\tau_{\text{mem}} \propto n$ grows **linearly** with dataset size.
- **Consequence:** As n increases, the interval

$$\tau \in [\tau_{\text{gen}},\ \tau_{\text{mem}}]$$

widens, creating a large region of training times where the model **generalizes well without memorizing**, even though very long training would eventually lead to memorization.

Key Empirical Findings — U-Net on CelebA

- **Setup**
  - U-Net score network trained on **32×32 grayscale CelebA**.
  - Optimized with **SGD + momentum**.
  - Metrics:
    - **FID** → sample quality / generalization → detects τ_{gen}.
    - **Memorization fraction** $f_{\text{mem}}(\tau)$ → detects τ_{mem}.
- **Generalization phase**
  - FID quickly drops to a minimum at $\tau_{\text{gen}} \approx 10^5$ steps.
  - $\tau_{\text{gen}} \approx$ **constant across different** n.
  - In this phase, $f_{\text{mem}}(\tau) \approx 0$ → **no memorization**.
- **Memorization phase**
  - $f_{\text{mem}}(\tau)$ grows only after a later time τ_{mem}.
  - When plotting $f_{\text{mem}}(\tau)$ vs. **rescaled time** τ/n, curves for different n **collapse**, rising around $\tau/n \approx 300$ → clear evidence that

$$\tau_{\text{mem}} \propto n.$$

  - The **overfitting time** (when L_{train} and L_{test} start to diverge) also scales linearly in n and appears **before** τ_{mem}.
- **Effect of model capacity (base width W)**
  - Increasing width W (and parameters p) makes both phases faster:

$$\tau_{\text{gen}} \propto W^{-1}, \qquad \tau_{\text{mem}} \propto nW^{-1}.$$

  - Larger models **reach good samples earlier** but also **memorize earlier** in absolute steps.
- **Phase diagram in (n, p)**
  - **Memorization regime**: small n at fixed p.
  - **Architectural regularization**: for $n > n^\star(p)$, the network cannot fit the empirical score → no memorization even as $\tau \to \infty$.
  - **Dynamical regularization**: large intermediate region where **early stopping** at $\tau \in [\tau_{\text{gen}}, \tau_{\text{mem}}]$ yields strong generalization without memorization.

References

- Papers
  - Jascha Sohl-Dickstein et al (2015), Deep Unsupervised Learning using Nonequilibrium Thermodynamics
  - Yang Song et al (2021), Score-Based Generative Modeling Through Stochastic Differential Equations

- Tutorials
  - Stanley Chan (2025), Tutorial on Diffusion Models for Imaging and Vision
  - Chieh-Hsin Lai et al (2025), The Principles of Diffusion Models (From Origins to Advances)