Machine Learning and Optimization

Lecture 23: Diffusion Models I — Learning to Follow the Score

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/.../mlandopt.html`

The blurriness of VAEs

Recall from Lecture 21: VAEs can produce blurry samples.

Why?

- The decoder mean satisfies $\mu_\theta(\mathbf{z}) = \mathbb{E}[\mathbf{x} \mid \mathbf{z}]$
- With MSE / Gaussian decoder likelihood, predictions are pulled toward this conditional mean
- If the latent posterior does not perfectly separate visual modes, averaging appears as blur

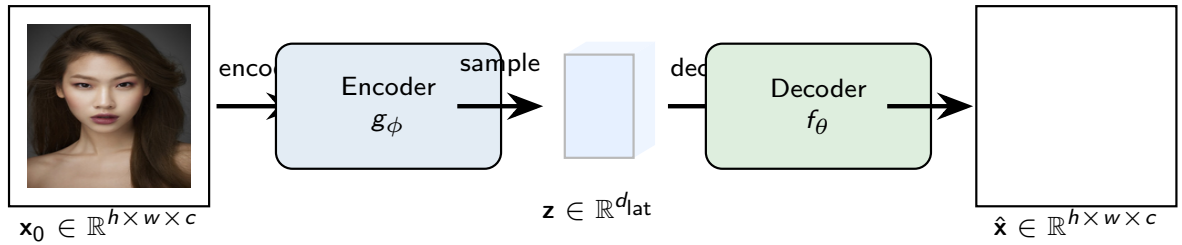
The issue is not a single decoder call by itself; blur comes from averaging across plausible structures represented under nearby latent codes.

Motivation for diffusion

What if generation made many small stochastic denoising decisions instead of one large decode decision?

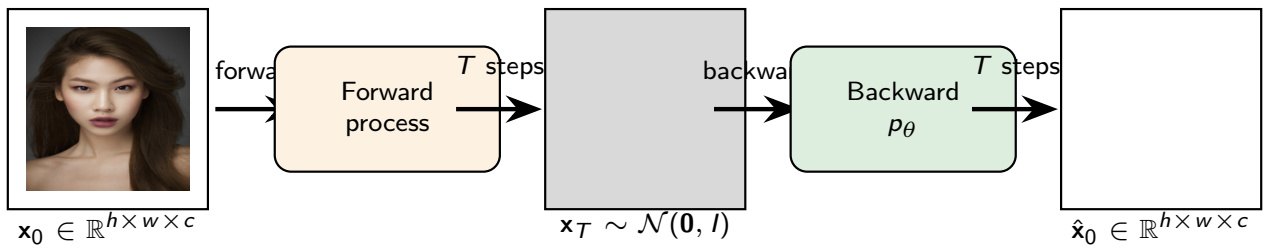
Two Paths to Generation

VAE (Lecture 21): Compress to generate



$d_{\text{lat}} \ll d$ — compresses to a low-dimensional code; decoder averages to sample

Diffusion (this lecture): Denoise to generate



Forward process adds noise to reach prior distribution, reverse process removes it to approximately sample from p_{data}.

Forward and Reverse Diffusion Processes

Forward process: Given $\mathbf{x}_0 \sim p_{\text{data}}$, add a little random Gaussian noise to get $\mathbf{x}_1$. Given $\mathbf{x}_{t-1}$ sample $\mathbf{x}_t$ by adding Gaussian noise to get $\mathbf{x}_t$. Do this T times.

Forward Markov sampling

$\mathbf{x}_t \sim q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, where $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is the Gaussian density $\mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t I)$

Backward (learned): Given a sample from the prior $\mathbf{x}_T \sim \mathcal{N}(0, I)$, denoise it using a learned neural network to obtain a sample $\mathbf{x}_{T-1}$. Given sample $\mathbf{x}_t$, denoise it with *the same network* to obtain a sample $\mathbf{x}_{t-1}$. Then $\mathbf{x}_0$ is an approximate sample from p_{data}.

Backward Markov sampling (learned)

$\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \sigma_\theta^2(\mathbf{x}_t, t))$

Learning Objectives

By the end of this lecture, you should be able to:

1. **Explain** why learning the score of raw data requires knowing $p_{\text{data}}(\mathbf{x})$ (chicken-and-egg problem) and how adding noise makes the score tractable
2. **State** Vincent's theorem: matching conditional scores learns the marginal score because the intractable posterior cancels
3. **Derive** the relationship between noise prediction $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ and the score function
4. **Explain** how the score field connects training (score matching) to sampling (Langevin dynamics)
5. **Recognize** that score matching and likelihood maximization (ELBO) yield the same training objective (full ELBO derivation in appendix)
6. **Implement** Langevin dynamics and DDPM sampling[1]

[1] DDPM = Denoising Diffusion Probabilistic Model (Ho et al., 2020)

Lecture Roadmap

The Sampling Problem: Following the Score

The Forward Process: Multiple Noise Levels

Score Matching: From Objective to Epsilon Prediction

The Training Objective: Two Views

Sampling: Following the Score Field

Implementation and Architecture

Summary

The Sampling Problem: Following the Score

The Score Function

Definition

The **score** of a distribution $p(\mathbf{x})$ is:

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

Properties:

- Points in the direction of steepest increase in log-density
- Orthogonal to level sets of $p(\mathbf{x})$
- At a mode: $s(\mathbf{x}) = \mathbf{0}$ (critical point)

Intuition: The score is a *vector field* that points toward higher probability regions.

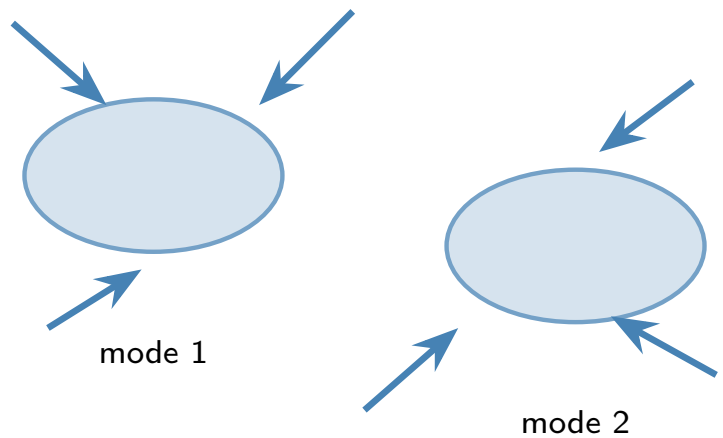
Question

If we knew $s(\mathbf{x})$, could we generate samples by following it?

A 2D Mixture Example

Example setup:

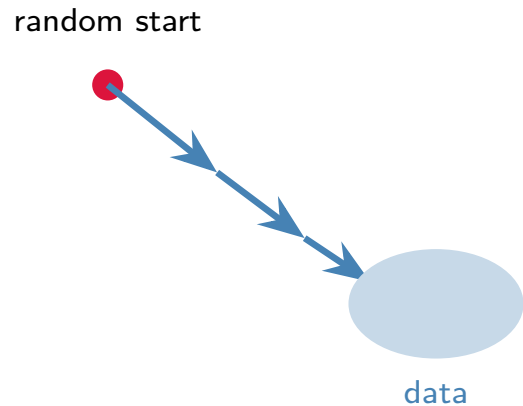
- Data distribution $p_{\text{data}}(\mathbf{x})$ is a mixture of two Gaussians in $\mathbb{R}^2$
- We can visualize density contours, score vectors, and sampling trajectories directly



The Ideal Approach: Langevin Dynamics

If we knew the score $s(\mathbf{x}) = \nabla \log p_{\text{data}}(\mathbf{x})$, we could sample:

1. Start from random noise (anywhere in space)
2. Look at the score arrow at your current location
3. Take a small step in the direction the arrow points
4. Add a little random noise to explore
5. Repeat until you reach high-density regions



The problem: We do not know $s(\mathbf{x}) = \nabla \log p_{\text{data}}(\mathbf{x})$ in closed form. We must learn it from data.

Pause and Check: Why the Score?

Question:
Consider a simple 2D Gaussian: $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \sigma^2 \mathbf{I})$.

1. What is $\nabla_{\mathbf{x}} p(\mathbf{x})$ (the raw gradient)?
2. What is $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ (the score)?
3. How do they differ in behavior as you move away from the origin?

Think about:

- Which one gives stronger signal far from the mode?
- Which one is more numerically stable?

Why the Score?

For $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$**:**

- $\nabla_{\mathbf{x}} p(\mathbf{x}) = -\frac{\mathbf{x}}{\sigma^2} p(\mathbf{x})$
- $\nabla_{\mathbf{x}} \log p(\mathbf{x}) = -\frac{\mathbf{x}}{\sigma^2}$

Why score, not raw density gradient?

- ∇p vanishes in low-density regions, so guidance disappears far from data
- $\nabla \log p$ preserves directional signal and gives stable long-range drift
- Langevin dynamics uses $\nabla \log p$ and has p as stationary distribution

Score Matching: Learning the Score

Goal: Learn a neural network $s_\theta(\mathbf{x}) \approx \nabla \log p_{\text{data}}(\mathbf{x})$ from data samples.

Score Matching Objective (Hyvärinen 2005)

Minimize the expected squared error between the learned score and true score:

$$J(\theta) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\| s_\theta(\mathbf{x}) - \nabla \log p_{\text{data}}(\mathbf{x}) \|^2 \right]$$

The problem: We can't compute this objective!

- We have samples $\{\mathbf{x}_i\}$ from $p_{\text{data}}(\mathbf{x})$, but we don't know $p_{\text{data}}(\mathbf{x})$ itself
- Therefore we can't compute $\nabla \log p_{\text{data}}(\mathbf{x})$
- Chicken-and-egg: need the score to learn the score

We need a way to learn $\nabla \log p_{\text{data}}(\mathbf{x})$ without knowing $p_{\text{data}}(\mathbf{x})$.

Smoothing the Data Distribution

There is a second, more fundamental obstacle beyond the chicken-and-egg problem.

Real data (images, text embeddings, etc.) often lives on or near a lower-dimensional surface inside $\mathbb{R}^d$. On that surface p_{data} may be concentrated, but *off* the surface p_{data} is essentially zero.

Where $p_{\text{data}}(\mathbf{x}) = 0$, the quantity $\log p_{\text{data}}(\mathbf{x})$ is $-\infty$, so $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ is not even defined.

We need a version of p_{data} that is positive *everywhere* in $\mathbb{R}^d$ so the score exists at every point.

The Smoothed Distribution p_σ

A natural way to spread the density everywhere is to add Gaussian noise.

Definition

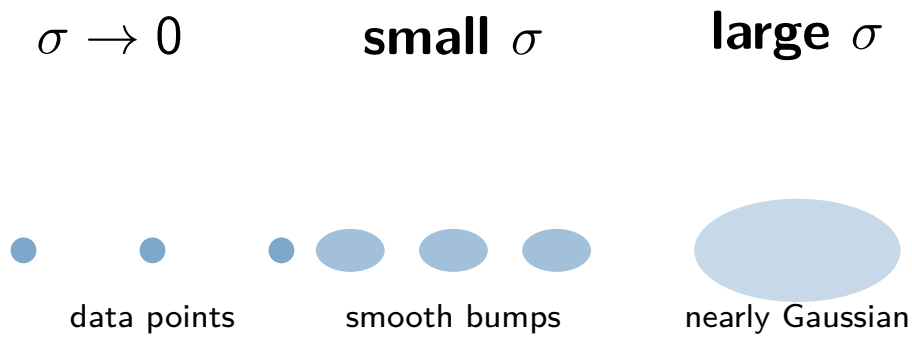
Given a noise level $\sigma > 0$, the **smoothed distribution** is

$$p_\sigma(\mathbf{x}) = \int p_{\text{data}}(\mathbf{x}_0) \cdot \underbrace{\frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_0\|^2}{2\sigma^2}\right)}_{\text{Gaussian centered at } \mathbf{x}_0} d\mathbf{x}_0$$

This is a convolution: every data point $\mathbf{x}_0$ gets replaced by a Gaussian blob of width σ, and we average the result.

Because the Gaussian is positive everywhere, the resulting $p_\sigma(\mathbf{x}) > 0$ for *all* $\mathbf{x} \in \mathbb{R}^d$, and so $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$ is well-defined everywhere.

Visualizing Smoothing



As $\sigma \to 0$, the smoothed distribution approaches the original: $p_\sigma \to p_{\text{data}}$.

So if we can learn $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$ for a small σ, we get an approximation to the score of the data distribution itself.

A Change of Plan

Our original goal was to learn $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$.

We will instead learn $\nabla_{\mathbf{x}} \log p_{\sigma}(\mathbf{x})$, the score of the smoothed distribution.

But we still face the same type of difficulty: we have samples from p_{data}, not a formula for p_{σ}. So we cannot directly compute $\nabla_{\mathbf{x}} \log p_{\sigma}(\mathbf{x})$ either.

Over the next several slides we will show that this problem can be solved by a single, elegant mathematical identity: the score of p_{σ} can be expressed in terms of quantities we *can* compute from data.

Setting Up the Derivation

Write $q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$ for the Gaussian noise density centered at $\mathbf{x}_0$:

$$q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \;=\; \mathcal{N}(\mathbf{x};\, \mathbf{x}_0,\, \sigma^2 \mathbf{I}) \;=\; \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\!\Big(-\frac{\|\mathbf{x}-\mathbf{x}_0\|^2}{2\sigma^2}\Big)$$

In this notation, the smoothed distribution is

$$p_\sigma(\mathbf{x}) \;=\; \int p_{\text{data}}(\mathbf{x}_0)\; q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; d\mathbf{x}_0$$

We want $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$. To get there, we will first compute $\nabla_{\mathbf{x}} p_\sigma(\mathbf{x})$ and then use the relationship between the gradient of a function and the gradient of its logarithm.

Computing $\nabla_{\mathbf{x}} p_\sigma(\mathbf{x})$

Differentiating the integral with respect to $\mathbf{x}$:

$$\nabla_{\mathbf{x}}\, p_\sigma(\mathbf{x}) \;=\; \nabla_{\mathbf{x}} \int p_{\text{data}}(\mathbf{x}_0)\; q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; d\mathbf{x}_0$$

The factor $p_{\text{data}}(\mathbf{x}_0)$ does not depend on $\mathbf{x}$ at all—it is just the probability of the clean data point $\mathbf{x}_0$. So the gradient passes through it and acts only on $q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$:

$$\nabla_{\mathbf{x}}\, p_\sigma(\mathbf{x}) \;=\; \int p_{\text{data}}(\mathbf{x}_0)\; \nabla_{\mathbf{x}}\, q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; d\mathbf{x}_0$$

(Formally, we also exchange $\nabla_{\mathbf{x}}$ and $\int d\mathbf{x}_0$; this is valid because q_σ is a smooth Gaussian.)

From ∇p to $\nabla \log p$

For any positive function $f(\mathbf{x}) > 0$, the chain rule gives

$$\nabla \log f(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{f(\mathbf{x})} \qquad \Longleftrightarrow \qquad \nabla f(\mathbf{x}) = f(\mathbf{x})\, \nabla \log f(\mathbf{x})$$

Applying the left form to $f = p_\sigma$:

$$\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) \;=\; \frac{\nabla_{\mathbf{x}}\, p_\sigma(\mathbf{x})}{p_\sigma(\mathbf{x})} \;=\; \frac{1}{p_\sigma(\mathbf{x})} \int p_{\text{data}}(\mathbf{x}_0)\, \nabla_{\mathbf{x}}\, q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\, d\mathbf{x}_0$$

Now apply the right form to $f = q_\sigma(\cdot \mid \mathbf{x}_0)$ to rewrite $\nabla_{\mathbf{x}} q_\sigma = q_\sigma\, \nabla_{\mathbf{x}} \log q_\sigma$:

$$\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) \;=\; \frac{1}{p_\sigma(\mathbf{x})} \int p_{\text{data}}(\mathbf{x}_0)\, q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\, \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\, d\mathbf{x}_0$$

This Is a Conditional Expectation

Look at the formula we just derived:

$$\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) \;=\; \frac{1}{p_\sigma(\mathbf{x})} \int p_{\text{data}}(\mathbf{x}_0)\; q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; d\mathbf{x}_0$$

The product $p_{\text{data}}(\mathbf{x}_0)\, q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$ is the *joint density* of the pair $(\mathbf{x}_0, \mathbf{x})$: first draw a clean point $\mathbf{x}_0$, then corrupt it to get $\mathbf{x}$.

By Bayes' rule, the *posterior* over clean points given the noisy observation is

$$p_\sigma(\mathbf{x}_0 \mid \mathbf{x}) \;=\; \frac{p_{\text{data}}(\mathbf{x}_0)\; q_\sigma(\mathbf{x} \mid \mathbf{x}_0)}{p_\sigma(\mathbf{x})}$$

Substituting this into our formula:

$$\begin{aligned} \nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) &= \int p_\sigma(\mathbf{x}_0 \mid \mathbf{x})\; \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\; d\mathbf{x}_0 \\ &= \mathbb{E}_{p_\sigma(\mathbf{x}_0 \mid \mathbf{x})}\big[\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\big] \end{aligned}$$

What This Formula Says

$$\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) \;=\; \mathbb{E}_{p_\sigma(\mathbf{x}_0|\mathbf{x})}\big[\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\big]$$

In words: the score of the smoothed distribution at a noisy point $\mathbf{x}$ equals the average of the **conditional scores** $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$, averaged over all the clean points $\mathbf{x}_0$ that could have produced $\mathbf{x}$.

Each conditional score $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$ depends only on the Gaussian q_σ, which we chose ourselves—so we can compute it in closed form.

The remaining question is practical: how do we take this expectation over $\mathbf{x}_0 \mid \mathbf{x}$ during training, when we do not know the posterior $p_\sigma(\mathbf{x}_0 \mid \mathbf{x})$?

From Theory to a Training Procedure

We want our network $s_\theta(\mathbf{x})$ to learn $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$.

During training we can sample from the *joint* distribution over $(\mathbf{x}_0, \mathbf{x})$ easily:

$$\mathbf{x}_0 \sim p_{\text{data}}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \qquad \mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$$

Each such pair $(\mathbf{x}_0, \mathbf{x})$ is a sample from the joint $p_{\text{data}}(\mathbf{x}_0)\, q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$.

For each pair, we can compute $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$ in closed form (we will do this explicitly in a moment).

So we train s_θ by minimizing the mean squared error over these joint samples:

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \right\|^2 \right], \qquad \mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$$

Why This Objective Learns the Right Thing

The function $f(\mathbf{x})$ that minimizes $\mathbb{E}\big[\|Y - f(X)\|^2\big]$ over all measurable f is the conditional expectation $f(\mathbf{x}) = \mathbb{E}[Y \mid X = \mathbf{x}]$.

(This is sometimes called the *regression principle*; it follows from the law of total expectation. It is the reason least-squares regression predicts conditional means.)

In our objective, X is the noisy observation $\mathbf{x}$ and Y is the conditional score $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$.

Therefore, the network that minimizes this score-matching objective satisfies

$$s_\theta(\mathbf{x}) \;=\; \mathbb{E}_{p_\sigma(\mathbf{x}_0|\mathbf{x})}\big[\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\big]$$

But does this conditional expectation actually equal $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$, the quantity we want?

Connecting Back: the Conditional Expectation *Is* the Score

On an earlier slide we proved that

$$\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x}) \;=\; \int p_\sigma(\mathbf{x}_0 \mid \mathbf{x})\, \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\, d\mathbf{x}_0$$

The right-hand side is, by definition, the conditional expectation of $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)$ given $\mathbf{x}$:

$$\int p_\sigma(\mathbf{x}_0 \mid \mathbf{x})\, \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\, d\mathbf{x}_0 \;=\; \mathbb{E}_{p_\sigma(\mathbf{x}_0 \mid \mathbf{x})}\big[\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)\big]$$

So the regression principle tells us s_θ learns $\mathbb{E}_{p_\sigma(\mathbf{x}_0|\mathbf{x})}[\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0)]$, and our derivation tells us this equals $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$.

Training network using the score-matching objective learns the score of the smoothed distribution, without ever computing the posterior $p_\sigma(\mathbf{x}_0 \mid \mathbf{x})$ or the marginal $p_\sigma(\mathbf{x})$!

The Conditional Score in Closed Form

The corruption density is a Gaussian centered at $\mathbf{x}_0$:

$$q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \;=\; \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\!\Big(-\frac{\|\mathbf{x}-\mathbf{x}_0\|^2}{2\sigma^2}\Big)$$

Taking the logarithm eliminates the normalizing constant (it becomes an additive constant that vanishes under $\nabla_{\mathbf{x}}$):

$$\log q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \;=\; -\frac{\|\mathbf{x}-\mathbf{x}_0\|^2}{2\sigma^2} \;+\; \text{const}$$

Now differentiate with respect to $\mathbf{x}$. Recall that $\nabla_{\mathbf{x}}\|\mathbf{x}-\mathbf{x}_0\|^2 = 2(\mathbf{x}-\mathbf{x}_0)$, so:

$$\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \;=\; -\frac{\mathbf{x}-\mathbf{x}_0}{\sigma^2}$$

Conditional Score as Noise Prediction

We just showed that $\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0) = -(\mathbf{x} - \mathbf{x}_0)/\sigma^2$.

Since $\mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$, we have $\mathbf{x} - \mathbf{x}_0 = \sigma\boldsymbol{\epsilon}$, and so

$$\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x} \mid \mathbf{x}_0) \;=\; -\frac{\sigma\boldsymbol{\epsilon}}{\sigma^2} \;=\; -\frac{\boldsymbol{\epsilon}}{\sigma}$$

The conditional score is proportional to the noise $\boldsymbol{\epsilon}$ that was added, just rescaled by $1/\sigma$.

So predicting the score and predicting the noise are the same problem, up to a factor of $-1/\sigma$.

The Denoising Score Matching Objective

Combining everything: we train a noise-prediction network $\boldsymbol{\epsilon}_\theta(\mathbf{x}, \sigma)$ to minimize

$$\mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}},\, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_0 + \sigma \boldsymbol{\epsilon},\ \sigma) \right\|^2 \right]$$

Given a noisy input $\mathbf{x} = \mathbf{x}_0 + \sigma \boldsymbol{\epsilon}$, the network tries to recover the noise $\boldsymbol{\epsilon}$ that was added.

Once trained, we recover the score of the smoothed distribution via

$$s_\theta(\mathbf{x}) \;=\; -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}, \sigma)}{\sigma} \;\approx\; \nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$$

This is **denoising score matching** (Vincent, 2011): learning the score by learning to denoise.

The Forward Process: Multiple Noise Levels

The Limitation of a Single Noise Level

We now have a powerful tool: given a noise level σ, we can train a network to predict the score of the smoothed distribution $p_\sigma(\mathbf{x})$.

But think about what happens at sampling time. We want to start from pure noise and walk toward data, following the score field.

If we trained at one small σ, the score field is accurate near the data manifold, but a random starting point $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is very far away—the score there is unreliable.

If we trained at one large σ, the score field covers the whole space, but it has smoothed out the fine details of p_{data}. Following it leads to blurry samples.

Neither extreme works alone.

A Coarse-to-Fine Strategy

The natural idea: use *many* noise levels, from large to small.

At a large noise level, the score field is smooth and easy to follow—it gives us the coarse structure (where are the modes?).

At smaller noise levels, the score field captures finer detail—it sharpens our samples.

So we want a **sequence of noise levels** $\sigma_1 < \sigma_2 < \cdots < \sigma_T$, and a single network $\boldsymbol{\epsilon}_\theta(\mathbf{x}, t)$ trained to predict the noise at every level t.

At sampling time, we start at $t = T$ (heavy noise) and progressively denoise down to $t = 1$ (light noise). This is the core idea of diffusion models.

Defining the Forward Process

We need a systematic way to create training pairs (noisy input, known noise target) at each level t.

The simplest approach: define a **Markov chain** that adds a small amount of noise at each step. Choose a schedule of noise strengths $\beta_1, \beta_2, \ldots, \beta_T \in (0, 1)$.

At each step, the transition is:

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

starting from a clean data point $\mathbf{x}_0 \sim p_{\text{data}}$.

There are two things to explain here: why do we *scale down* $\mathbf{x}_{t-1}$ by $\sqrt{1-\beta_t}$, and why is the noise scaled by $\sqrt{\beta_t}$?

Why the Scaling?

If we just added noise without scaling—say $\mathbf{x}_t = \mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t$—then the variance of $\mathbf{x}_t$ would grow at every step:

$$\mathrm{Var}(\mathbf{x}_t) = \mathrm{Var}(\mathbf{x}_{t-1}) + \beta_t$$

After T steps the variance would be $\mathrm{Var}(\mathbf{x}_0) + \sum_t \beta_t$, which can be very large. The process would drift off to infinity.

By multiplying $\mathbf{x}_{t-1}$ by $\sqrt{1-\beta_t}$ before adding noise, we *shrink* the signal at the same rate we add noise. The variance of each step stays controlled:

$$\mathrm{Var}(\mathbf{x}_t) = (1-\beta_t)\,\mathrm{Var}(\mathbf{x}_{t-1}) + \beta_t$$

If $\mathbf{x}_{t-1}$ had unit variance, then $\mathrm{Var}(\mathbf{x}_t) = 1 - \beta_t + \beta_t = 1$. The variance is preserved at each step. Eventually, the signal is fully replaced by noise and $\mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The Transition as a Gaussian

We can write the forward step in distributional form.

Since $\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t$ with $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, this is a linear transformation of a Gaussian, so

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \;=\; \mathcal{N}\big(\mathbf{x}_t;\; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\; \beta_t\,\mathbf{I}\big)$$

At each step, the mean of $\mathbf{x}_t$ is a slight shrinkage of $\mathbf{x}_{t-1}$, and the variance is β_t (a small number).

Because each step only depends on $\mathbf{x}_{t-1}$ (not on $\mathbf{x}_{t-2}, \ldots, \mathbf{x}_0$), the full forward process $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ is a Markov chain:

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

Skipping Ahead: Can We Jump Directly to Step t?

For training, we need to generate $\mathbf{x}_t$ at a randomly chosen level t.

If we had to simulate the chain step by step from $\mathbf{x}_0$ to $\mathbf{x}_t$, training would require t sequential operations per sample. For $T = 1000$, that is far too slow.

Fortunately, because each step is a linear-Gaussian operation, we can derive a *closed-form expression* for $\mathbf{x}_t$ directly in terms of $\mathbf{x}_0$ and a single noise draw.

The derivation uses one basic property of Gaussians that we will work through carefully.

Background: Adding Independent Gaussians

We will need the following fact repeatedly.

If $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \sigma_1^2\,\mathbf{I})$ and $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \sigma_2^2\,\mathbf{I})$ are *independent*, then their sum has distribution

$$\mathbf{u} + \mathbf{v} \;\sim\; \mathcal{N}(\mathbf{0},\; (\sigma_1^2 + \sigma_2^2)\,\mathbf{I})$$

This follows from the fact that the variance of a sum of independent random variables is the sum of their variances.

An equivalent restatement: if $\boldsymbol{\epsilon}, \boldsymbol{\epsilon}'$ are independent draws from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, then

$$\sigma_1\,\boldsymbol{\epsilon} + \sigma_2\,\boldsymbol{\epsilon}' \;\sim\; \mathcal{N}\big(\mathbf{0},\; (\sigma_1^2 + \sigma_2^2)\,\mathbf{I}\big) \;=\; \sqrt{\sigma_1^2 + \sigma_2^2}\;\boldsymbol{\epsilon}''$$

where $\boldsymbol{\epsilon}'' \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a single fresh draw.

Two independent Gaussian noise terms can always be replaced by a single Gaussian whose variance is the sum of the two variances.

Deriving the Accumulated Effect: Two Steps

Let us write out $\mathbf{x}_2$ in terms of $\mathbf{x}_0$.

Define $\alpha_t = 1 - \beta_t$ for convenience. The forward step is $\mathbf{x}_t = \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_t$.

Expand $\mathbf{x}_2$ by substituting the expression for $\mathbf{x}_1$:

$$\begin{aligned}
\mathbf{x}_2 &= \sqrt{\alpha_2}\,\mathbf{x}_1 + \sqrt{1-\alpha_2}\,\boldsymbol{\epsilon}_2 \\
&= \sqrt{\alpha_2}\big(\sqrt{\alpha_1}\,\mathbf{x}_0 + \sqrt{1-\alpha_1}\,\boldsymbol{\epsilon}_1\big) + \sqrt{1-\alpha_2}\,\boldsymbol{\epsilon}_2 \\
&= \sqrt{\alpha_1\alpha_2}\,\mathbf{x}_0 \;+\; \underbrace{\sqrt{\alpha_2(1-\alpha_1)}\,\boldsymbol{\epsilon}_1 + \sqrt{1-\alpha_2}\,\boldsymbol{\epsilon}_2}_{\text{sum of two independent Gaussians}}
\end{aligned}$$

Since $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ are independent standard Gaussians, the underbraced term has variance $\alpha_2(1-\alpha_1) + (1-\alpha_2) = 1 - \alpha_1\alpha_2$.

So we can replace it by $\sqrt{1-\alpha_1\alpha_2}\,\boldsymbol{\epsilon}$ for a single $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$\mathbf{x}_2 = \sqrt{\alpha_1\alpha_2}\,\mathbf{x}_0 + \sqrt{1-\alpha_1\alpha_2}\,\boldsymbol{\epsilon}$$

The General Accumulation Formula

The same argument extends by induction. At each step, when we substitute and collect the noise terms, we always get a sum of independent Gaussians that can be merged into a single draw.

Define $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s = \prod_{s=1}^{t} (1 - \beta_s)$.

Then for any t, we can sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$:

$$\boxed{\mathbf{x}_t \;=\; \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 \;+\; \sqrt{1 - \bar{\alpha}_t}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}$$

In distributional form, this says
$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\, \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\, (1 - \bar{\alpha}_t)\,\mathbf{I})$.

This is enormously useful: to generate a training example at any level t, we only need one multiplication, one addition, and one random draw—no chain simulation required.

A Subtlety: Independence in the Accumulation

There is an important distinction to keep in mind.

In the Markov chain, the noise $\boldsymbol{\epsilon}_t$ at step t is independent of the noise $\boldsymbol{\epsilon}_s$ at step $s \neq t$, but $\mathbf{x}_t$ and $\mathbf{x}_s$ are *not* independent—they share the same underlying $\mathbf{x}_0$.

The accumulation formula $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$ uses a *single fresh* $\boldsymbol{\epsilon}$—not the same $\boldsymbol{\epsilon}$ as in any other step.

So if we use the formula to sample $\mathbf{x}_t$ and $\mathbf{x}_s$ independently (with separate draws $\boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon}'$), we get the correct *marginal* distribution at each level, but we do *not* get the correct joint distribution $q(\mathbf{x}_s, \mathbf{x}_t \mid \mathbf{x}_0)$.

For training, this is perfectly fine: we only need the marginal at one randomly chosen t. The joint distribution matters for sampling (the reverse process), which we will address later.

The Noise Schedule

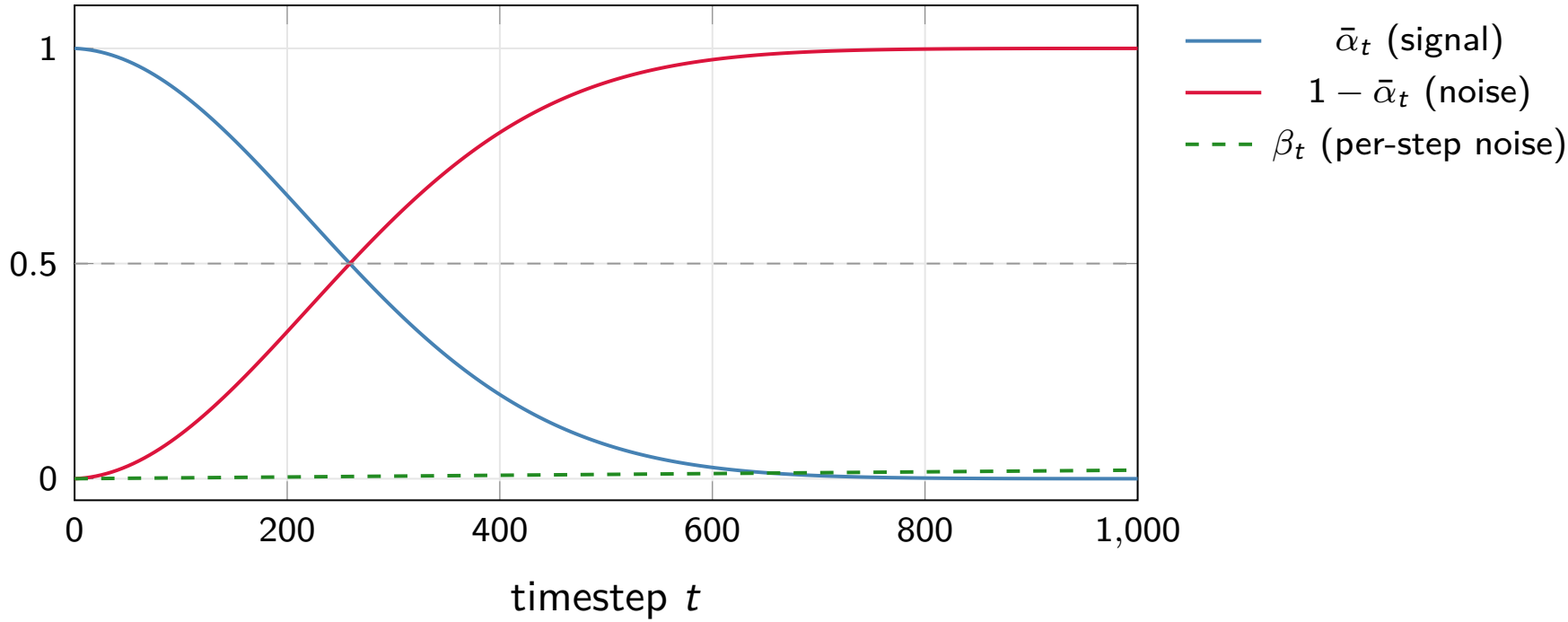
The schedule $\{\beta_t\}_{t=1}^{T}$ controls how quickly the signal is destroyed.

A common choice is a **linear schedule**: β_t increases linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$ over $T = 1000$ steps. Each step adds a tiny bit more noise than the last.

An alternative is the **cosine schedule** (Nichol & Dhariwal, 2021), which spends more steps at low noise levels where the fine details of the data are being resolved.

What matters is not β_t directly, but the *cumulative* quantities $\bar{\alpha}_t$ and $1 - \bar{\alpha}_t$, since those determine how much signal and noise remain at level t.

Visualizing the Schedule

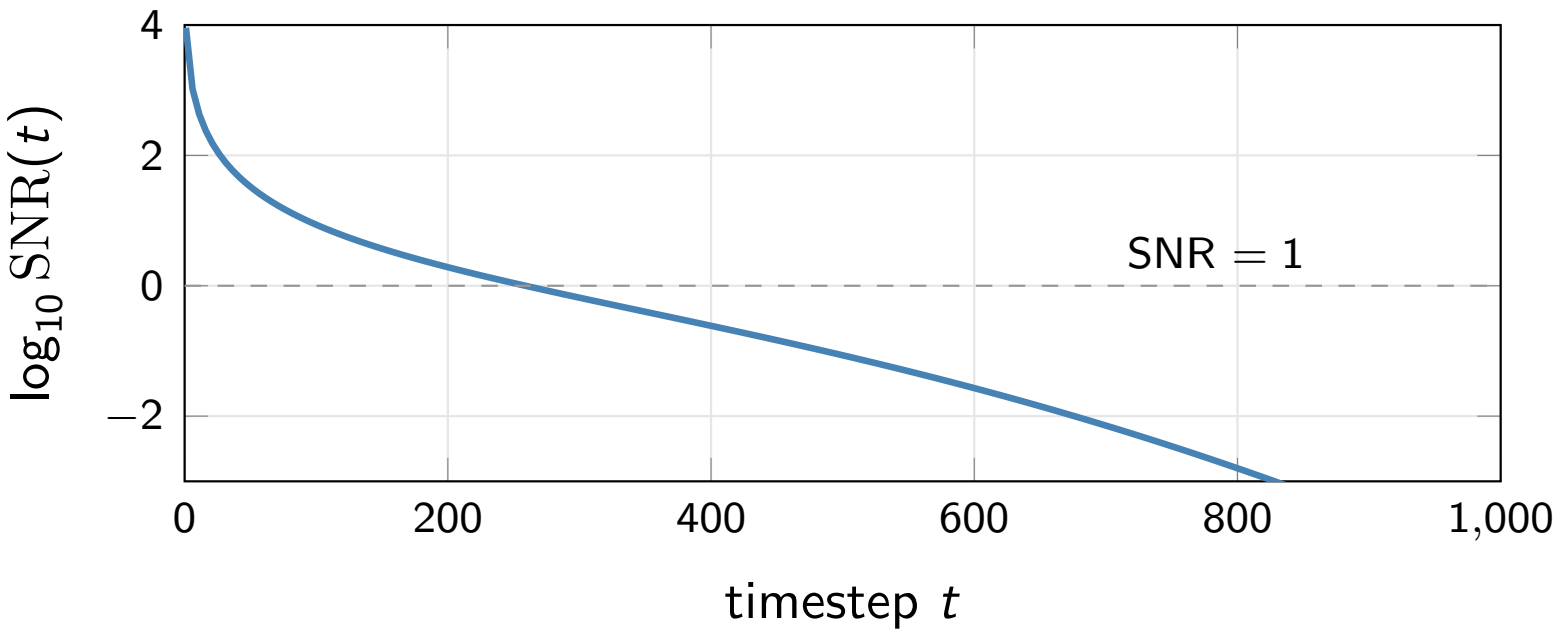


When $\bar{\alpha}_t \approx 1$, the signal dominates: $\mathbf{x}_t \approx \mathbf{x}_0$, so the noise $\boldsymbol{\epsilon}$ is a small perturbation that is relatively easy to predict. When $\bar{\alpha}_t \approx 0$, the noise dominates: $\mathbf{x}_t \approx \boldsymbol{\epsilon}$, and the score is approximately $-\mathbf{x}_t$, which is trivial. Near the crossover, signal and noise are comparable, so the network must disentangle the two—this is the hardest regime.

Signal-to-Noise Ratio

A useful way to think about the schedule is through the **signal-to-noise ratio** (SNR):

$$\mathrm{SNR}(t) = \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t}$$



At early steps, $\mathrm{SNR} \gg 1$: signal dominates. At late steps, $\mathrm{SNR} \ll 1$: noise dominates. The log-SNR drops roughly linearly for the linear schedule.

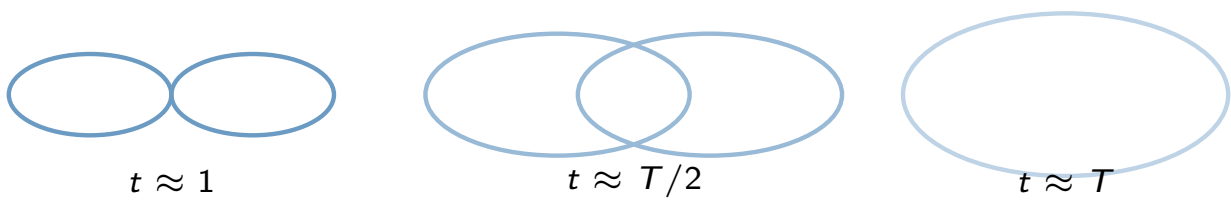
What the Marginals Look Like

Since $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$, the marginal $q_t(\mathbf{x}_t)$ is the data distribution convolved with Gaussian noise of variance $1-\bar{\alpha}_t$, and rescaled by $\sqrt{\bar{\alpha}_t}$.

At $t \approx 1$: $\bar{\alpha}_t \approx 1$, so $\mathbf{x}_t \approx \mathbf{x}_0$. The marginal is nearly the data distribution.

At $t \approx T/2$: $\bar{\alpha}_t \approx 0.5$, so $\mathbf{x}_t$ is a blurred, shrunken version of the data. Modes merge; the score field becomes smoother.

At $t \approx T$: $\bar{\alpha}_t \approx 0$, so $\mathbf{x}_t \approx \boldsymbol{\epsilon}$. The marginal is nearly $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and the score is approximately $s_t(\mathbf{x}_t) \approx -\mathbf{x}_t$.



From Single-σ to the Forward Process

Everything we proved in the single-noise-level setting carries over. The forward process is just a structured way of sampling from many noise levels at once.

The correspondence is:

Single-σ setting	**Forward-process setting**
$\mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$	$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$
noise variance $= \sigma^2$	noise variance $= 1 - \bar{\alpha}_t$
smoothed density $p_\sigma(\mathbf{x})$	marginal density $q_t(\mathbf{x}_t)$
$\nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x}\|\mathbf{x}_0) = -\boldsymbol{\epsilon}/\sigma$	$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t\|\mathbf{x}_0) = -\boldsymbol{\epsilon}/\sqrt{1-\bar{\alpha}_t}$

Every identity we derived—the conditional score formula, the noise-prediction equivalence, the denoising score matching objective—transfers to each noise level t by the substitution $\sigma^2 \leftrightarrow 1 - \bar{\alpha}_t$.

Score Matching: From Objective to Epsilon Prediction

Score Matching at Multiple Noise Levels

Recall: At a single noise level σ, Vincent's theorem says: match conditional scores $\Rightarrow$ learn the marginal score.

For each timestep t, we want $s_t(\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$, so we match the conditional score $\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0)$.

The Conditional Score at Level t

Since $q_t(\mathbf{x}_t \mid \mathbf{x}_0)$ is the Gaussian density $\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$:

$$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0}{1-\bar{\alpha}_t} = -\frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}}$$

Same calculation as the single-σ case, with $\sigma^2 \mapsto 1 - \bar{\alpha}_t$.

The conditional score depends only on $\boldsymbol{\epsilon}$.

Score-Noise Relationship

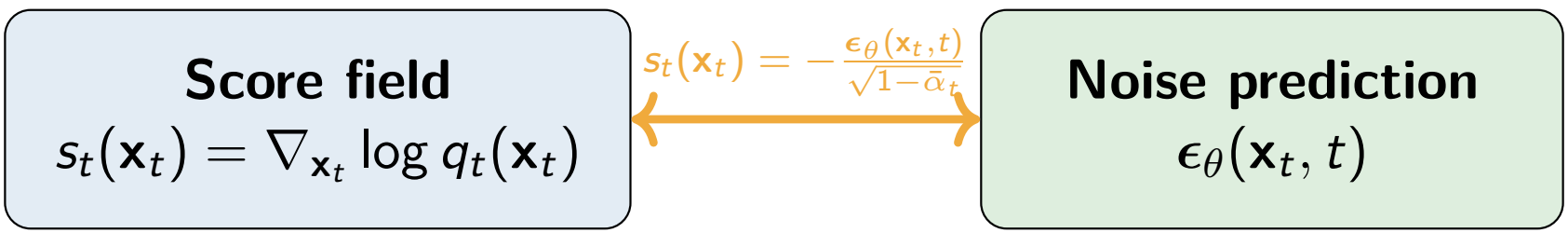
From the previous slide, the conditional score at level t is

$$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}}.$$

Notice: the right-hand side depends on the noise $\boldsymbol{\epsilon}$ that was added, not directly on $\mathbf{x}_0$.

So if a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ can predict $\boldsymbol{\epsilon}$ from the noisy observation $\mathbf{x}_t$, we immediately get the score:

$$s_t(\mathbf{x}_t) \approx -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}.$$



The Training Objective

Denoising score matching loss:

$$\mathcal{L}_{\text{DSM}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0) - s_\theta(\mathbf{x}_t, t)\|^2\right]$$

Learn a score prediction network that works well at *all* scales of noise.

Substituting the relationship: Let $s_\theta(\mathbf{x}_t, t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)}{\sqrt{1-\bar{\alpha}_t}}$

Simplified objective:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

In other words, $\mathcal{L}_{\text{simple}}$ is score matching in disguise: predicting the noise *is* learning the score field, just in a more convenient parameterization. So score-matching can be achieved by denoising.

Training Algorithm

Putting everything together, a single training iteration looks like this:

Algorithm: DDPM Training (noise-prediction form)

Input: p_{data}, schedule $\{\beta_t\}_{t=1}^{T}$, network $\boldsymbol{\epsilon}_\theta$

Repeat:

1. Sample $\mathbf{x}_0 \sim p_{\text{data}}$ (clean training example)
2. Sample $t \sim \text{Uniform}\{1, \dots, T\}$ (choose noise level)
3. Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (known target noise)
4. Form $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$ (forward corruption in one shot)
5. Update θ on $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$ (DSM objective at level t)

The loop itself is short. All the conceptual work—choosing the noise schedule, parameterizing the score via ϵ-prediction, using the one-shot formula for $\mathbf{x}_t$—is already baked into these five lines.

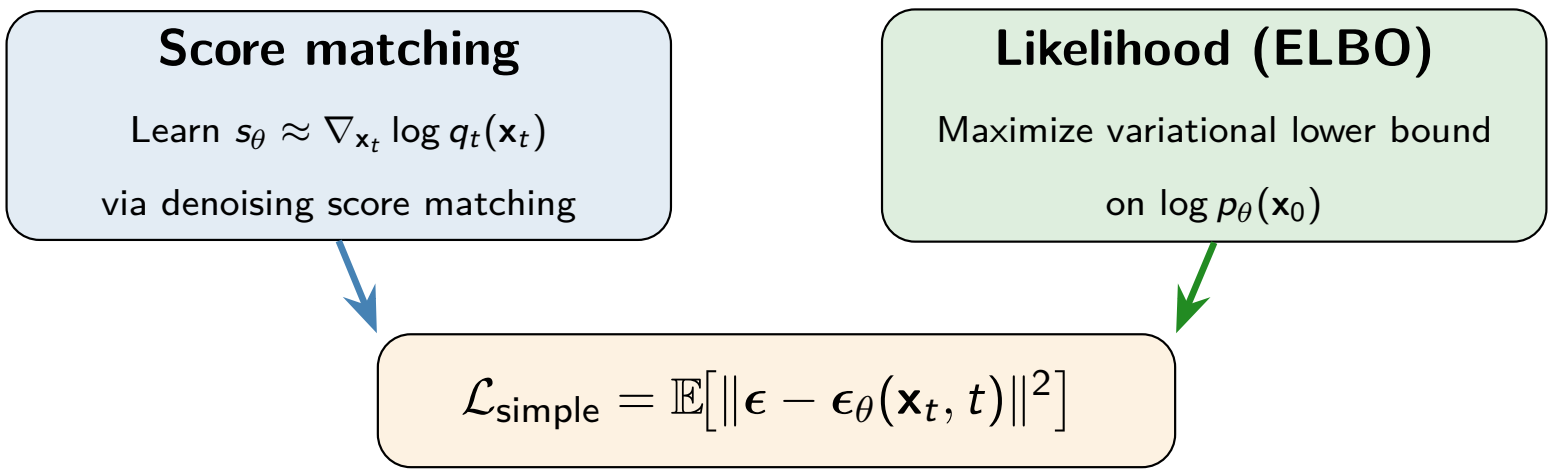
The Training Objective: Two Views

Two Perspectives on the Training Objective

From score matching, we derived the objective

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\,\mathbf{x}_0,\,\boldsymbol{\epsilon}}\Big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\Big].$$

Is there an independent justification? **Yes!** We can justify incremental denoising by using the ELBO. This also tells us how tightly diffusion models approximate the evidence.



The full ELBO derivation in given in the appendix. Next: the key ingredients from the likelihood view needed for sampling.

The Forward Posterior

To build a reverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we need to know what distribution it should match at each step.

The **forward posterior** $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ tells us: given a noisy observation $\mathbf{x}_t$ and the original data $\mathbf{x}_0$, what was the likely value of $\mathbf{x}_{t-1}$?

Forward posterior (derived in appendix):

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_{t-1};\ \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0),\ \tilde{\beta}_t\, \mathbf{I}\big)$$

where

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\,\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\,\mathbf{x}_t, \qquad \tilde{\beta}_t = \frac{(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)}\,\beta_t.$$

The mean $\tilde{\mu}_t$ is a weighted average of $\mathbf{x}_0$ and $\mathbf{x}_t$: when noise is low (small t), it trusts $\mathbf{x}_0$ when noise is high (large t), it relies more on $\mathbf{x}_t$.

From Matching to Noise Prediction

We model the reverse process as $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mu_\theta(\mathbf{x}_t, t),\ \tilde{\beta}_t\,\mathbf{I})$, fixing the variance to match the forward posterior and learning only the mean.

With matched variances, minimizing $D_{\mathrm{KL}}(q\|p_\theta)$ reduces to matching means:

$$\mathcal{L}_t = \frac{1}{2\tilde{\beta}_t}\,\|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2\,.$$

Using $\mathbf{x}_0 = (\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon})/\sqrt{\bar{\alpha}_t}$ to eliminate $\mathbf{x}_0$:

$$\tilde{\mu}_t(\mathbf{x}_t, \boldsymbol{\epsilon}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\right).$$

Parameterizing via $\boldsymbol{\epsilon}_\theta$:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right).$$

Minimizing $\mathcal{L}_t$ is then (up to a constant factor) the same as minimizing $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\|^2$.

Two Roads, One Objective

Perspective	**Guarantee**	**Design choices**
Score matching	Learns $\nabla \log q_t$	ϵ-parameterization
ELBO (appendix)	Maximizes likelihood bound	Gaussian p_θ, matched var.

Both perspectives yield

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\,\mathbf{x}_0,\,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2\right].$$

What this means for practice:

- Predicting noise *is* learning the score field
- Learning the score field *is* maximizing a likelihood bound
- The same $\boldsymbol{\epsilon}_\theta$ serves both training and sampling

Sampling: Following the Score Field

Intuition: Score-Guided Sampling

Before presenting the sampling algorithm, let's build geometric intuition.

We trained ϵ_θ to predict the score at every noise level t. How do we use this to generate samples?

Key idea from optimization: If you have the score $s(\mathbf{x}) = \nabla \log p(\mathbf{x})$, you can sample from $p(\mathbf{x})$ via **Langevin dynamics**:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\eta}{2} s(\mathbf{x}_k) + \sqrt{\eta}\, \mathbf{z}_k, \qquad \mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Two components:

- **Drift:** $+\frac{\eta}{2} s(\mathbf{x}_k)$ pushes toward higher density
- **Noise:** $+\sqrt{\eta}\, \mathbf{z}_k$ adds randomness to explore

This is the template DDPM follows: drift toward data + controlled noise.

DDPM Has the Same Structure

DDPM sampling follows the Langevin template: drift + noise.

At each timestep t, the DDPM update is

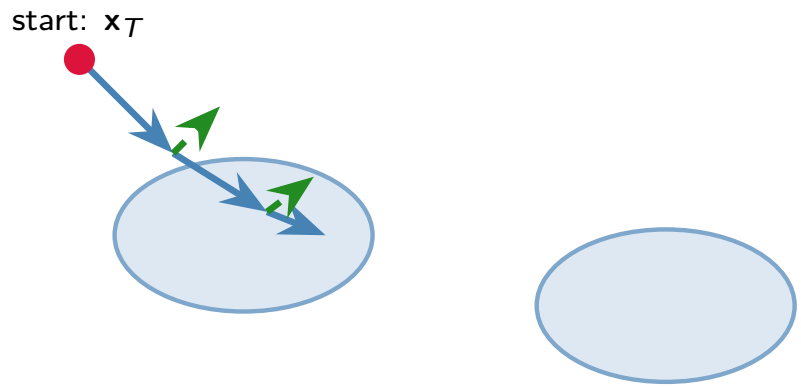
$$\mathbf{x}_{t-1} = \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\text{drift toward data}} + \underbrace{\sigma_t \mathbf{z}}_{\text{noise}}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- **Drift term $\mu_\theta(\mathbf{x}_t, t)$:** Uses ϵ_θ to estimate where $\mathbf{x}_{t-1}$ should be, pulling $\mathbf{x}_t$ toward the data manifold
- **Noise term $\sigma_t \mathbf{z}$:** Adds randomness so we explore the distribution instead of collapsing to a single mode

The difference from pure Langevin: μ_θ is derived by matching the forward posterior mean and the noise level σ_t decreases as we denoise.

What DDPM Does Geometrically

Visualizing the drift + noise structure in 2D:



- **Blue arrows (drift):** μ_θ guides the trajectory toward high-density regions
- **Green dashed (noise):** $\sigma_t \mathbf{z}$ adds stochastic exploration, preventing collapse to a single mode

As t decreases, the noise level σ_t shrinks and the trajectory converges.

The DDPM Sampling Algorithm

Putting it all together: sample from the learned reverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.

Algorithm (DDPM):

Input: Trained ϵ_θ, schedule $\{\beta_t\}_{t=1}^T$

1. Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ (start from pure noise)

2. For $t = T, T-1, \ldots, 1$:

$$\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right)}_{\mu_\theta(\mathbf{x}_t,t) = \text{drift toward data}} + \underbrace{\sigma_t \mathbf{z}}_{\text{add noise}}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where $\sigma_t = \sqrt{\tilde{\beta}_t} = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t}$ (variance from forward posterior)

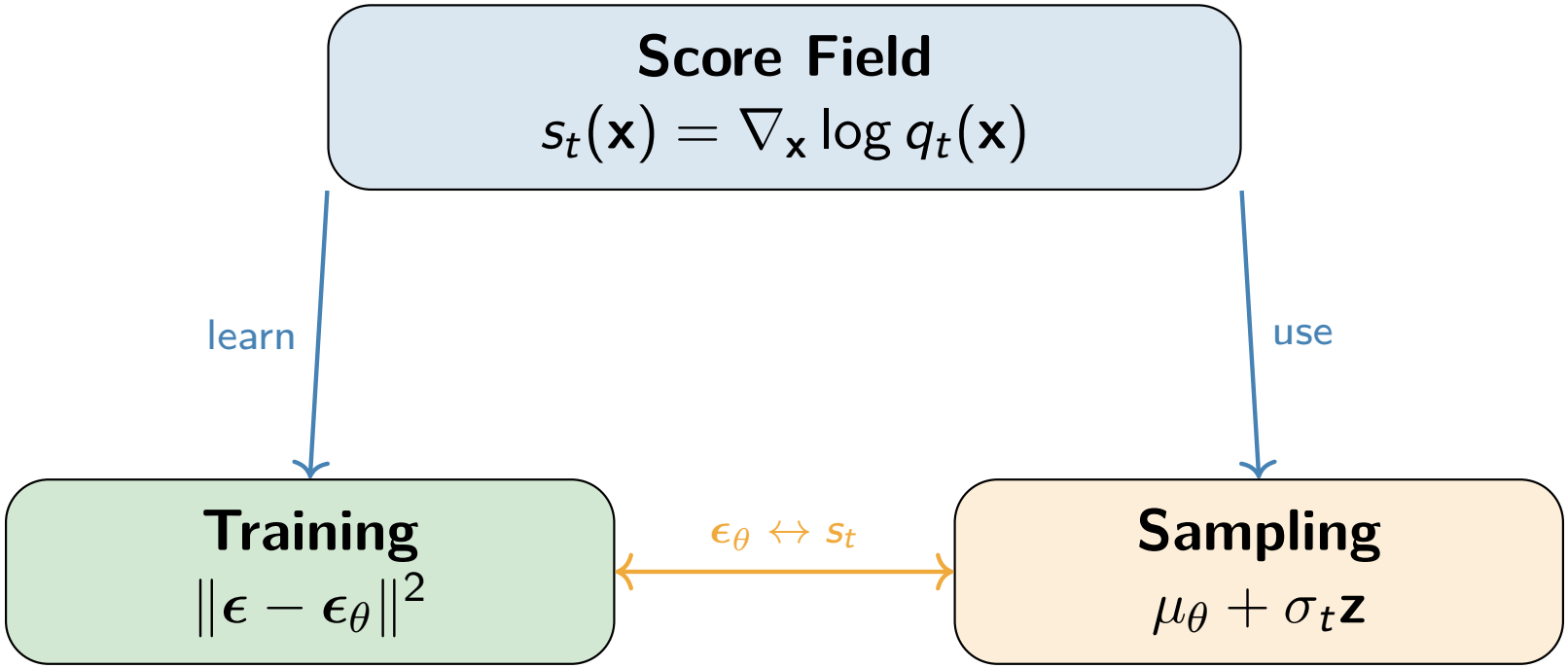
3. Return $\mathbf{x}_0$ (final denoised sample)

What the terms do:

- **Mean μ_θ:** Uses predicted noise ϵ_θ to estimate the cleaner $\mathbf{x}_{t-1}$
- **Variance $\sigma_t^2 = \tilde{\beta}_t$:** Matches the forward posterior variance we computed earlier

The Whole Picture

Everything connects through the score field:



The connections:

- We **train** ϵ_θ by matching the score at every noise level
- We **sample** by using ϵ_θ in μ_θ to follow the score field
- **Score** $\leftrightarrow$ **noise:** $s_t(\mathbf{x}_t) = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$
- **Mean** $\leftrightarrow$ **noise:** $\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$

Implementation and Architecture

Designing the network ϵ_θ

We need to learn $\epsilon_\theta : \mathbb{R}^{h \times w \times c} \times \{1, \ldots, T\} \to \mathbb{R}^{h \times w \times c}$ — taking noisy image $\mathbf{x}_t$ and timestep t, predicting the noise ϵ added to $\mathbf{x}_0$. Two requirements shape the architecture.

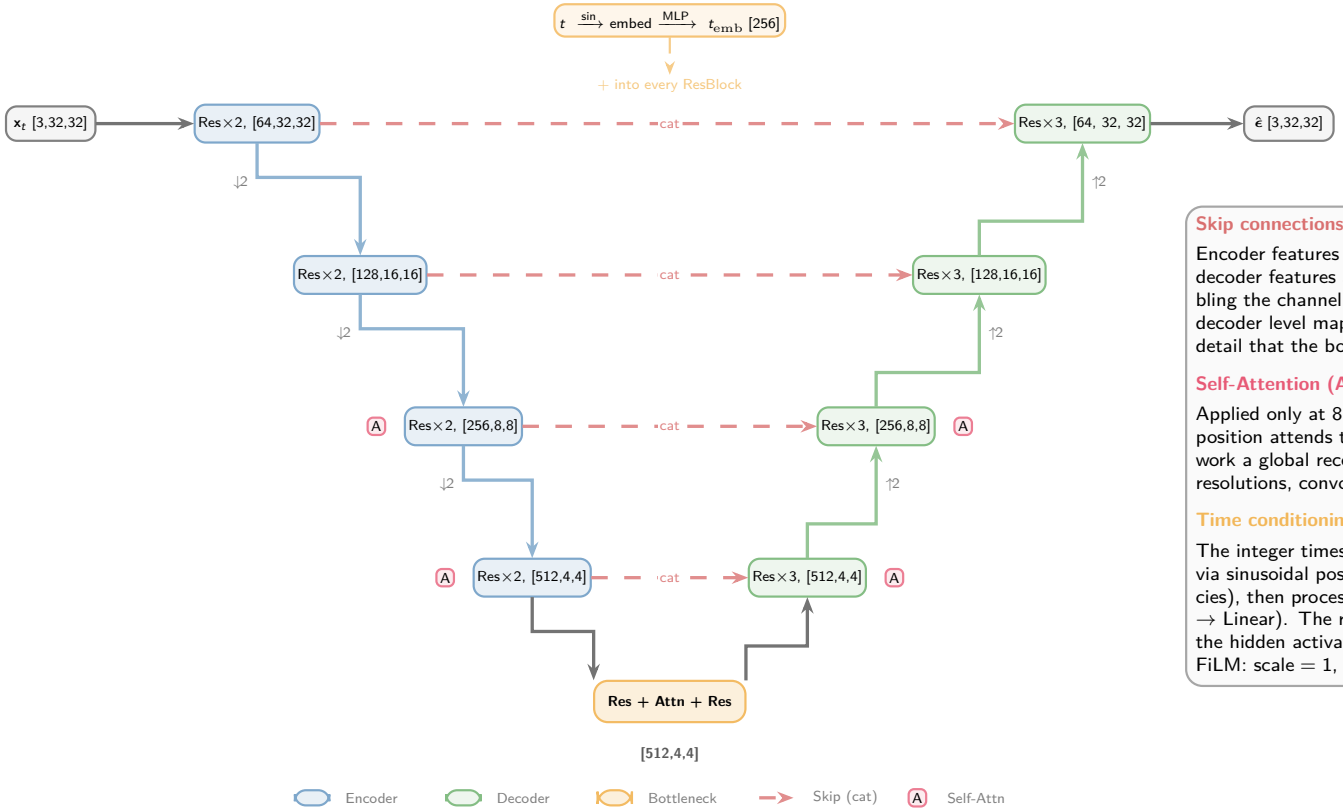
The output must have the *same size* as the input.

- A **U-Net** compresses the image into a hierarchy of feature maps (fine to coarse) then reconstructs an image of the same size.
- **Skip connections** feed encoder features directly to the decoder, restoring fine spatial detail lost at the bottleneck.

Behavior must adapt to the noise level t. We need a mechanism to change the network's behavior depending on t.

Our solution is to encode t as a **time embedding** t_{emb} and inject it into every **ResBlock** in the U-Net, changing the features it extracts.

DiffusionUNet Architecture



Skip connections

Encoder features are concatenated (channel axis) with decoder features at the same spatial resolution, doubling the channel count. The first ResBlock of each decoder level maps $2C \to C$. This preserves fine spatial detail that the bottleneck alone would lose.

Self-Attention (A)

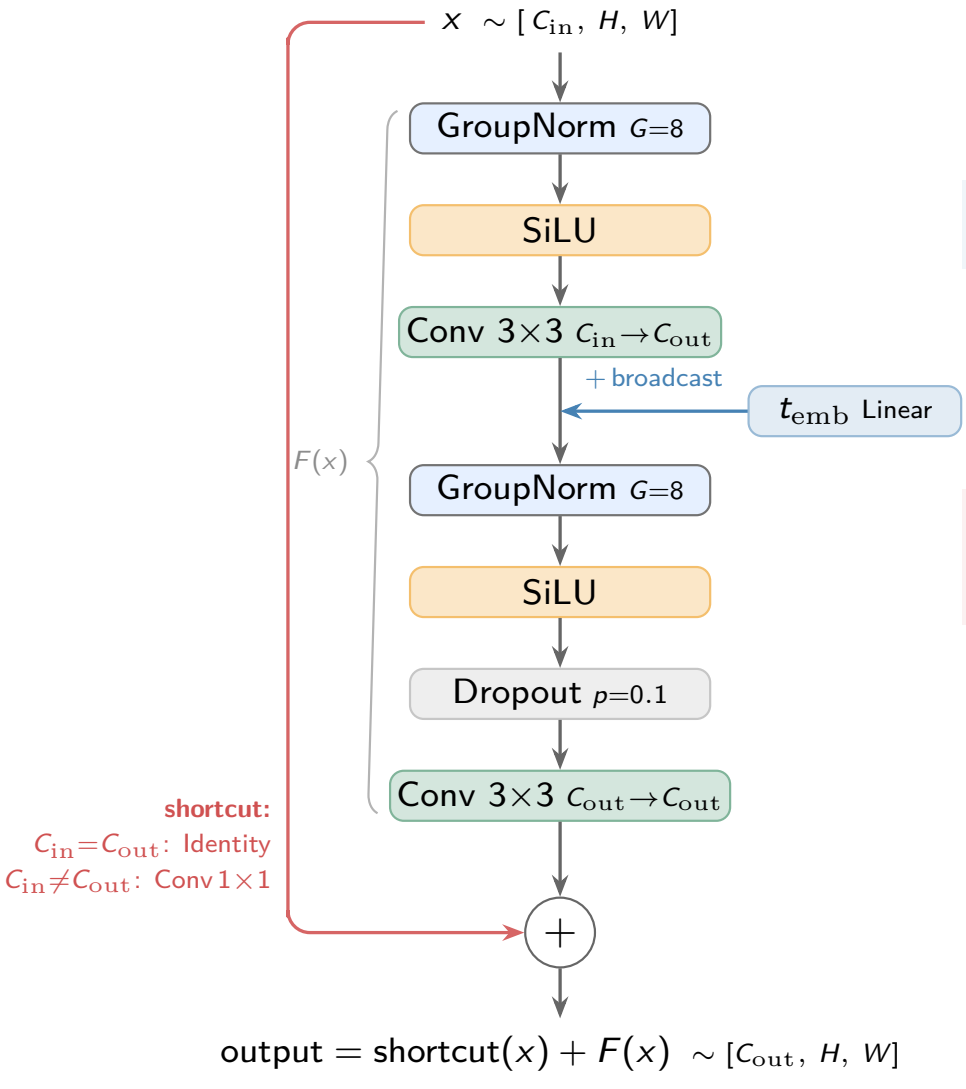
Applied only at 8×8 and 4×4 resolution. Every spatial position attends to every other position, giving the network a global receptive field at low resolution. At higher resolutions, convolutions alone handle local structure.

Time conditioning

The integer timestep t is mapped to a 256-dim vector via sinusoidal positional encoding (log-spaced frequencies), then processed by a 2-layer MLP (Linear → SiLU → Linear). The resulting t_{emb} is broadcast-added to the hidden activations inside every ResBlock (bias-only FiLM: scale = 1, shift = t_{emb}).

Stage	Spatial	Ch	Res	Attn	Notes
Input proj	32^2	$3 \to 64$	–	–	Conv 3×3, padding 1
Enc 0	32^2	64	2		$\downarrow$ Conv stride 2 $\to$ 16^2
Enc 1	16^2	128	2		$\downarrow$ Conv stride 2 $\to$ 8^2
Enc 2	8^2	256	2	✓	$\downarrow$ Conv stride 2 $\to$ 4^2
Enc 3	4^2	512	2	✓	skip saved before $\downarrow$
Bottleneck	4^2	512	2	✓	Res $\to$ SelfAttn $\to$ Res
Dec 0	4^2	512	3	✓	cat skip: $1024 \to 512$
Dec 1	8^2	256	3	✓	$\uparrow$ ConvTranspose stride 2; cat skip
Dec 2	16^2	128	3		$\uparrow$ ConvTranspose stride 2; cat skip
Dec 3	32^2	64	3		$\uparrow$ ConvTranspose stride 2; cat skip
Output	32^2	$64 \to 3$	–	–	GroupNorm $\to$ SiLU $\to$ Conv 3×3

ResBlock: residual block with time conditioning



Layer	Details
GroupNorm	G=8 groups
SiLU	$x \cdot \sigma(x)$; aka swish
Conv2d 3×3	$C_{\text{in}} \to C_{\text{out}}$, padding 1
t_{emb} inject	Linear(d, C_{out}) add to each spatial position $H \times W$
GroupNorm	G=8 groups
SiLU	
Dropout	p=0.1
Conv2d 3×3	$C_{\text{out}} \to C_{\text{out}}$, padding 1
shortcut	$C_{\text{in}} = C_{\text{out}}$: **nn.Identity()** $C_{\text{in}} \neq C_{\text{out}}$: **Conv2d** 1×1 — channel projection only
+	elementwise add: $F(x) + \text{shortcut}(x)$

Design rationale:

- **Pre-activation order** (Norm→Act→Conv): gradients bypass nonlinearities via the shortcut.
- **Time after first conv**: before GroupNorm it would be normalized away.
- **GroupNorm** not BatchNorm: works at batch 1 (sampling).
- 1×1 **shortcut**: channel projection only, no spatial mixing.

Training Stability

Diffusion training is remarkably stable compared to other generative models.

The key reason: the forward process is fixed, not learned. We only optimize the reverse denoiser, so there is a single MSE objective $\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$ at each noise level.

Contrast this with VAEs, where the encoder and decoder must be balanced—the reconstruction term and the KL term can fight each other, leading to posterior collapse.

Diffusion avoids the need to balance terms, and the loss decreases monotonically with more training.

Summary

Summary

Component	**Formula / Description**	**Purpose**
Forward process	$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$	Generate training pairs
Training objective	$\mathbb{E}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$	Learn to denoise
Score relation	$s_t(\mathbf{x}_t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$	Noise $\leftrightarrow$ score
Reverse mean	$\mu_\theta = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta\right)$	Drift toward data
Sampling	$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sqrt{\tilde{\beta}_t}\,\mathbf{z}$	$\mathbf{x}_T \to \mathbf{x}_0$

The key insights:

- Smoothing makes score learning tractable (Vincent's theorem)
- Noise prediction parameterizes the score field
- Score matching and likelihood maximization (ELBO, appendix) give the same objective
- Sampling follows the learned score field via the reverse process

Questions?

Next: Stable Diffusion

Appendix: Historical Development

From score matching (2005) to diffusion models (2020):

Year	Development
2005	Hyvärinen: Score matching for parameter estimation
2011	Vincent: DAEs implicitly learn scores
2019	Song & Ermon: Multi-scale scores + annealed sampling
2020	Ho et al.: DDPM, ϵ-prediction, high-quality samples

Why did it take 15 years for score matching to become a generative modeling breakthrough?

The Foundation (2005–2011)

Hyvärinen (2005): Score matching for energy-based models

- Learn $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ without computing partition function Z
- Single noise level, used for parameter estimation (not generation)
- Manifold problem: score undefined where data lives

Vincent (2011): Denoising autoencoders learn scores

- Proved: training a DAE to denoise $\mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$ implicitly matches the score of p_σ
- Still single σ, used for representation learning
- The mathematical connection existed but no generative application

What was missing: Multi-scale noise, sampling procedure, deep architectures, generative framing

The Missing Ingredients

Between 2011 and 2019, score matching needed three breakthroughs:

1. **Multi-scale noise progression**
   - Single σ can't bridge from pure noise to data
   - Need a sequence: high noise $\rightarrow$ medium $\rightarrow$ low noise

2. **Annealed Langevin sampling**
   - Langevin dynamics was known, but not with gradually decreasing noise
   - Need an explicit sampling algorithm that uses learned scores

3. **Deep learning maturity**
   - U-Nets (2015), ResNets (2016), attention mechanisms (2017)
   - Architectures that could represent complex score fields at multiple scales

Song & Ermon (2019): Bridging the Gap

Key paper: "Generative Modeling by Estimating Gradients of the Data Distribution" (NeurIPS 2019)

The breakthrough: Combine multi-scale noise with annealed Langevin sampling

- Train score networks $s_{\sigma_i}(\mathbf{x})$ at multiple noise levels $\sigma_1 < \sigma_2 < \cdots < \sigma_L$
- Sample by running Langevin dynamics while gradually decreasing σ
- Frame it as **generative modeling**: "start from noise, follow the score, reach data"

Impact: First compelling image samples from score-based models. Proved the concept works.

Limitation: trained separate networks for each noise level (inefficient).

Ho et al. (2020): The DDPM Breakthrough

Key paper: "Denoising Diffusion Probabilistic Models" (NeurIPS 2020)

What made it click:

1. **ϵ-prediction parameterization**
   - Simpler and more stable than directly predicting the score
2. **ELBO connection**
   - Grounded diffusion in variational inference (familiar framework)
   - Connected to VAEs from Lecture 21
3. **Single time-conditioned network**
   - One $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ handles all noise levels (via FiLM)
   - Much more efficient than Song & Ermon's separate networks
4. **High-quality image generation (CIFAR-10, CelebA-HQ, LSUN)**
   - Samples competitive with GANs on standard benchmarks
   - Stable training, no mode collapse

Result: diffusion became the dominant paradigm for image generation.

Appendix: The ELBO Derivation (Full)

This appendix contains the full derivation of the ELBO for diffusion models, referenced in the main lecture.

We show that maximizing the ELBO (evidence lower bound) leads to the same training objective $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\|^2$ derived from score matching.

Roadmap:

1. Write the ELBO in terms of the forward and reverse chains
2. Rearrange it into a sum of KL divergences, one per timestep
3. Compute the forward posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$
4. Show that matching means requires predicting noise
5. Conclude: the ELBO objective is exactly $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\|^2$

See Lecture 21 for background on the ELBO and variational inference.

Roadmap: Validating the Objective via the ELBO

We already derived $\|\epsilon - \epsilon_\theta\|^2$ from score matching. What can we say about the likelihood of samples generated from score matching? By connecting score matching to a latent variable model and using the ELBO, we can validate the performance of score matching.

This appendix follows this path:

1. Write the ELBO in terms of the forward and reverse chains
2. Rearrange it into a sum of KL divergences, one per timestep
3. Figure out what distribution the reverse process should match at each step
4. Show that matching means requires predicting noise
5. Conclude: the ELBO objective is exactly $\|\epsilon - \epsilon_\theta\|^2$

The derivation is long but each piece is straightforward. We will pause twice to recap progress.

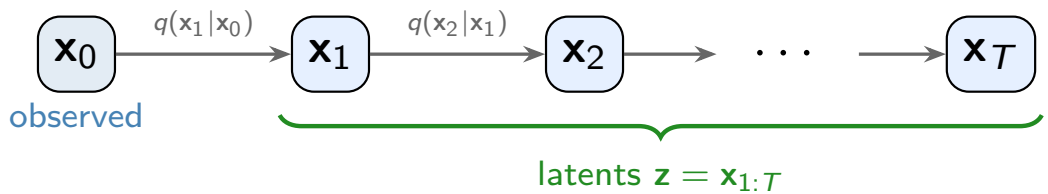
A Second Perspective: The ELBO

We have already derived the training objective $\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$ from score matching. Now we ask: does this same objective have a *likelihood-based* justification?

In Lecture 21 we saw the ELBO—a lower bound on $\log p_\theta(\mathbf{x})$ for any latent variable model:

$$\log p_\theta(\mathbf{x}) \;\geq\; \mathcal{L}_{\text{ELBO}} \;=\; \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right].$$

Diffusion models *are* latent variable models: the latent variables are the entire noisy chain $\mathbf{z} = \mathbf{x}_{1:T}$.



The forward process $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ is the approximate posterior (fixed, not learned). We still need to specify the reverse $p_\theta(\mathbf{x}_{0:T})$—a *design decision*.

Writing Out the ELBO for Diffusion

Substituting $\mathbf{z} = \mathbf{x}_{1:T}$ into the ELBO:

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

The forward process factors by the Markov property: $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$.

For the reverse, we *choose* to model p_θ as a Markov chain running backward from pure noise: $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. This is a **design choice**: our generative model reverses the chain one step at a time. We have not yet specified the *form* of each $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$.

With both factorizations, the log-ratio inside the expectation becomes

$$\log p(\mathbf{x}_T) + \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}.$$

The Rearrangement Problem

We need to rearrange the sum

$$\log p(\mathbf{x}_T) + \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}$$

into a form we can interpret and optimize.

The difficulty: the ratio $\frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}$ compares two distributions that condition on *different* variables. The numerator conditions on $\mathbf{x}_t$ and produces $\mathbf{x}_{t-1}$; the denominator conditions on $\mathbf{x}_{t-1}$ and produces $\mathbf{x}_t$.

To get a KL divergence (which requires comparing two distributions *over the same variable, conditioned on the same thing*), we need to "flip" the denominator so it also conditions on $\mathbf{x}_t$.

The tool for flipping conditionals is Bayes' rule.

Applying Bayes' Rule to the Forward Process

Recall Bayes' rule for conditional distributions: given random variables A, B, C,

$$q(A \mid B, C)\, q(B \mid C) = q(B \mid A, C)\, q(A \mid C).$$

This is just the chain rule for probability applied twice.

Set $A = \mathbf{x}_t$, $B = \mathbf{x}_{t-1}$, $C = \mathbf{x}_0$. Then:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)\, q(\mathbf{x}_t \mid \mathbf{x}_0).$$

By the Markov property of the forward process, $\mathbf{x}_t$ depends on $\mathbf{x}_{t-1}$ alone, so $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$.

Solving for $q(\mathbf{x}_t|\mathbf{x}_{t-1})$:

$$\boxed{q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\; q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}.}$$

Substituting into the Log-Ratio Sum

Replace each $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ in the denominator:

$$\sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}$$

$$= \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \cdot q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \cdot q(\mathbf{x}_t|\mathbf{x}_0)}$$

Split the log of the product into a sum of logs:

$$= \underbrace{\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{\text{reconstruction } (t=1)} + \sum_{t=2}^{T} \Bigg[\underbrace{\log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}}_{\text{compares reverse distributions}} + \underbrace{\log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}}_{\text{ratio of forward marginals}}$$

The first piece (for $t \geq 2$) compares $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$—both are distributions over $\mathbf{x}_{t-1}$ conditioned on $\mathbf{x}_t$—exactly what we wanted.

The Telescoping Sum

The $t{=}1$ term in the first piece is the **reconstruction term** $\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)$, pulled out separately. For $t \geq 2$, write out the forward-marginal ratios:

$$t = 2: \quad \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{q(\mathbf{x}_2|\mathbf{x}_0)}$$

$$t = 3: \quad \log \frac{q(\mathbf{x}_2|\mathbf{x}_0)}{q(\mathbf{x}_3|\mathbf{x}_0)}$$

$$\vdots$$

$$t = T: \quad \log \frac{q(\mathbf{x}_{T-1}|\mathbf{x}_0)}{q(\mathbf{x}_T|\mathbf{x}_0)}$$

Each numerator cancels the denominator of the next term. After cancellation only the last denominator survives:

$$\sum_{t=2}^{T} \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \;=\; \log q(\mathbf{x}_1|\mathbf{x}_0) - \log q(\mathbf{x}_T|\mathbf{x}_0).$$

Assembling the ELBO

Collecting everything, pulling out the $t{=}1$ term (reconstruction), and recognizing KL divergences:

$$\mathcal{L}_{\text{ELBO}} = \underbrace{-D_{\mathrm{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big)}_{\text{prior matching}}$$

$$- \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)}_{\mathcal{L}_t\text{: denoising matching at step } t}$$

$$+ \underbrace{\mathbb{E}_q[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)]}_{\text{reconstruction}}$$

(Minus signs because we *maximize* the ELBO; training *minimizes* $-\mathcal{L}_{\text{ELBO}}$, i.e. minimizes the KL terms.)

Each $\mathcal{L}_t$ compares the learned reverse $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ against the forward posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ we just computed.

Interpreting the Three Pieces

The ELBO has decomposed into three types of terms, each with a clear role.

Prior matching term: $D_{\mathrm{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big)$

We choose $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ as our prior—pure isotropic noise. From the one-shot formula, $q(\mathbf{x}_T|\mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_T}\,\mathbf{x}_0,\ (1-\bar{\alpha}_T)\mathbf{I}\big)$.

We *design* the noise schedule so that $\bar{\alpha}_T \approx 0$, which means $\sqrt{\bar{\alpha}_T} \approx 0$ and $\sqrt{1-\bar{\alpha}_T} \approx 1$. Then $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{0}, \mathbf{I}) = p(\mathbf{x}_T)$, so this KL is ≈ 0.

Reconstruction term: $\mathcal{L}_0 = -\mathbb{E}_q[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)]$

This measures how well the model recovers the clean data $\mathbf{x}_0$ from the almost-clean $\mathbf{x}_1$. Since the first noise step β_1 is very small, $\mathbf{x}_1 \approx \mathbf{x}_0$, and this term is typically small.

The real work happens in the $\sum_{t=2}^{T} \mathcal{L}_t$ terms.

Checkpoint: ELBO Decomposition Complete

Where we are in the roadmap:

- ✓ Write the ELBO in terms of the forward and reverse chains
- ✓ Rearrange it into a sum of KL divergences, one per timestep
- ☐ Figure out what distribution the reverse process should match at each step
- ☐ Show that matching means requires predicting noise
- ☐ Conclude: the ELBO objective is exactly $\|\epsilon - \epsilon_\theta\|^2$

What we have so far: The ELBO decomposes into

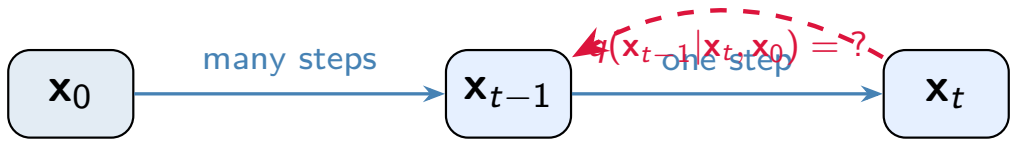
$$\mathcal{L}_{\text{ELBO}} = (\text{prior term}) - \sum_{t=2}^{T} D_{\text{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big) + (\text{reconstruction}).$$

The prior and reconstruction terms are small by design. The real question: what should $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ be to minimize these KL terms?

To answer that, we first need to know what $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is.

Why Condition on $\mathbf{x}_0$?

The KL terms involve $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$—the forward process conditioned on *both* $\mathbf{x}_t$ *and* the original clean data $\mathbf{x}_0$. Why?



Knowing only $\mathbf{x}_t$, we cannot determine where $\mathbf{x}_{t-1}$ was—many different clean images could have produced the same noisy $\mathbf{x}_t$. But if we also know $\mathbf{x}_0$, we know both where the chain began and where it is now, pinning down $\mathbf{x}_{t-1}$.

Both $q(\mathbf{x}_{t-1}|\mathbf{x}_0)$ and $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ are Gaussian, so $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is also Gaussian with a closed-form formula.

We call $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ the **forward posterior** at step t: *posterior* because it updates our belief about $\mathbf{x}_{t-1}$ after observing $\mathbf{x}_t$ and $\mathbf{x}_0$; *forward* because it uses only the forward process q (no learned parameters).

Gaussian Conditioning Theorem

To compute $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, we use a standard fact about multivariate Gaussians.

Theorem (Conditioning in a Joint Gaussian)

If $\mathbf{y}$ and $\mathbf{z}$ are jointly Gaussian (given some fixed conditioning variable),

$$\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_y\\ \boldsymbol{\mu}_z\end{pmatrix}, \begin{pmatrix}\Sigma_{yy} & \Sigma_{yz}\\ \Sigma_{zy} & \Sigma_{zz}\end{pmatrix}\right),$$

then the conditional distribution of $\mathbf{y}$ given $\mathbf{z}$ is

$$(\mathbf{y} \mid \mathbf{z}) \sim \mathcal{N}\big(\boldsymbol{\mu}_y + \Sigma_{yz}\Sigma_{zz}^{-1}(\mathbf{z} - \boldsymbol{\mu}_z),\ \Sigma_{yy} - \Sigma_{yz}\Sigma_{zz}^{-1}\Sigma_{zy}\big).$$

We will apply this with $\mathbf{y} = \mathbf{x}_{t-1}$ and $\mathbf{z} = \mathbf{x}_t$, both conditioned on $\mathbf{x}_0$. So we need the joint distribution $(\mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{x}_0)$: its mean vector and its 2×2 covariance matrix.

The Joint Distribution $(\mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{x}_0)$

From the forward process, both marginals conditioned on $\mathbf{x}_0$ are Gaussian. Writing d for the dimension of $\mathbf{x}$:

Means (from the one-shot formula):

$$\mathbb{E}[\mathbf{x}_{t-1} \mid \mathbf{x}_0] = \sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0, \qquad \mathbb{E}[\mathbf{x}_t \mid \mathbf{x}_0] = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0.$$

Marginal variances: $\mathrm{Var}(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = (1-\bar{\alpha}_{t-1})\,\mathbf{I}$, $\mathrm{Var}(\mathbf{x}_t \mid \mathbf{x}_0) = (1-\bar{\alpha}_t)\,\mathbf{I}$.

Cross-covariance: Since $\mathbf{x}_t = \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t$ with $\boldsymbol{\epsilon}_t$ independent of $\mathbf{x}_{t-1}$,

$$\mathrm{Cov}(\mathbf{x}_{t-1}, \mathbf{x}_t \mid \mathbf{x}_0) = \sqrt{\alpha_t}\,\mathrm{Var}(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = \sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\,\mathbf{I}.$$

All blocks of the joint covariance matrix are proportional to $\mathbf{I}$, so the conditioning theorem reduces to scalar formulas applied coordinate-wise.

Computing the Posterior Variance

Apply the conditioning theorem with $\mathbf{y} = \mathbf{x}_{t-1}$, $\mathbf{z} = \mathbf{x}_t$. Since all covariance blocks are proportional to $\mathbf{I}$, we work with scalars.

Posterior variance. Define $\tilde{\beta}_t$ as the scalar variance (the conditioning theorem gives us $\Sigma_{yy} - \Sigma_{yz}\Sigma_{zz}^{-1}\Sigma_{zy}$):

$$\begin{aligned}\tilde{\beta}_t &= (1-\bar{\alpha}_{t-1}) - \frac{\alpha_t(1-\bar{\alpha}_{t-1})^2}{1-\bar{\alpha}_t} \\ &= \frac{(1-\bar{\alpha}_{t-1})\big[(1-\bar{\alpha}_t) - \alpha_t(1-\bar{\alpha}_{t-1})\big]}{1-\bar{\alpha}_t}.\end{aligned}$$

To simplify the bracket, use $\bar{\alpha}_t = \alpha_t\bar{\alpha}_{t-1}$, so

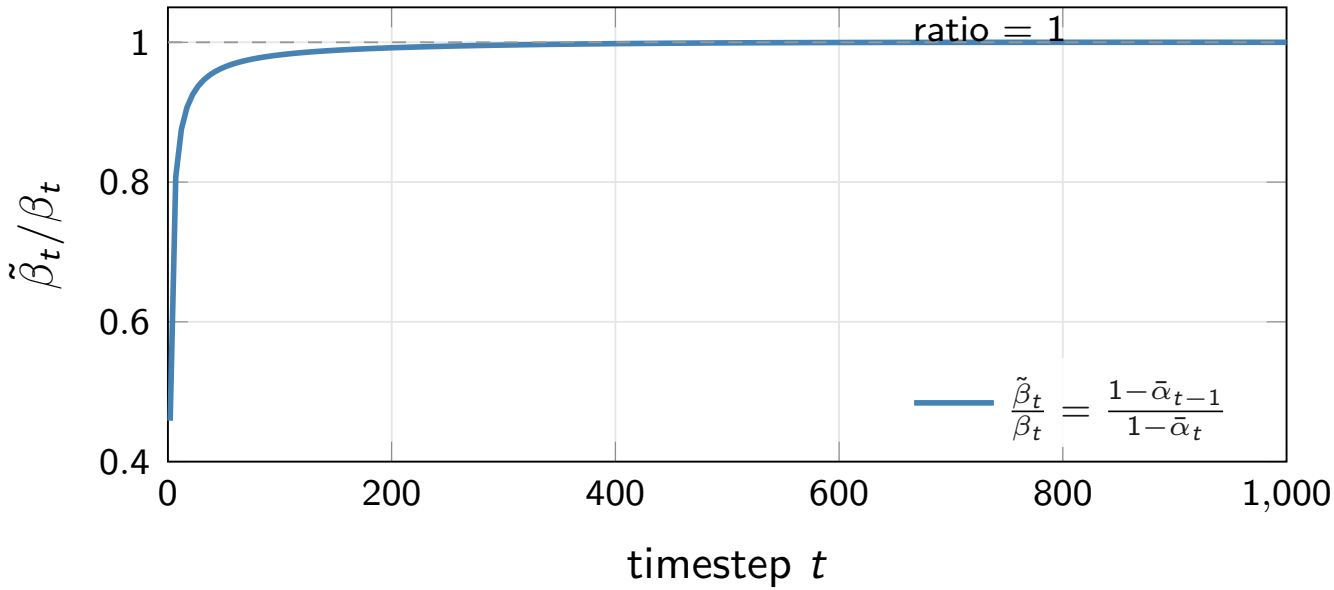
$$(1-\bar{\alpha}_t) - \alpha_t(1-\bar{\alpha}_{t-1}) = 1 - \alpha_t\bar{\alpha}_{t-1} - \alpha_t + \alpha_t\bar{\alpha}_{t-1} = 1-\alpha_t = \beta_t.$$

Therefore:

$$\boxed{\tilde{\beta}_t = \frac{(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)}\,\beta_t.}$$

Visualizing the Forward Posterior Variance

How much smaller is the posterior variance $\tilde{\beta}_t$ compared to the forward variance β_t?



Observations:

- At small t: ratio ≈ 0.5 — posterior variance is about *half* the forward variance
- Ratio increases monotonically toward 1 as noise accumulates
- At large t: ratio ≈ 1 — conditioning on $\mathbf{x}_t$ provides little extra information

Why Is the Posterior Variance Smaller?

Geometric intuition: When t is small and there's little noise, knowing *both* $\mathbf{x}_t$ and $\mathbf{x}_0$ pins down $\mathbf{x}_{t-1}$ more tightly than knowing just $\mathbf{x}_0$ alone—hence $\tilde{\beta}_t < \beta_t$.

When t is large and both are nearly pure noise, conditioning on $\mathbf{x}_t$ provides little extra information beyond $\mathbf{x}_0$, so $\tilde{\beta}_t \approx \beta_t$.

Mathematical reasoning: The conditioning theorem formula $\Sigma_{yy} - \Sigma_{yz}\Sigma_{zz}^{-1}\Sigma_{zy}$ subtracts off the variance already explained by the correlation between $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$.

At small t, the two are highly correlated (one forward step apart), so conditioning reduces variance significantly. At large t, both are dominated by independent noise, so little correlation to exploit.

Computing the Posterior Mean

Posterior mean. Define $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ as the mean vector (the conditioning theorem gives us $\boldsymbol{\mu}_y + \Sigma_{yz}\Sigma_{zz}^{-1}(\boldsymbol{z} - \boldsymbol{\mu}_z)$):

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\big(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0\big).$$

Expand and collect the $\mathbf{x}_0$ terms. The coefficient of $\mathbf{x}_0$ is:

$$\begin{aligned}&\sqrt{\bar{\alpha}_{t-1}} - \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})\sqrt{\bar{\alpha}_t}}{1-\bar{\alpha}_t} \\ &\quad = \sqrt{\bar{\alpha}_{t-1}}\left(1 - \frac{\alpha_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\right) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t},\end{aligned}$$

using the same bracket identity as the variance calculation.

Final result:

$$\boxed{\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\,\mathbf{x}_0 \;+\; \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\,\mathbf{x}_t.}$$

Interpreting the Forward Posterior

Putting the pieces together:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_{t-1};\ \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0),\ \tilde{\beta}_t\, \mathbf{I}\big).$$

The mean $\tilde{\mu}_t$ is a weighted average of the clean data $\mathbf{x}_0$ and the current noisy observation $\mathbf{x}_t$:

	Weight on $\mathbf{x}_0$	**Weight on $\mathbf{x}_t$**
Formula	$\frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}$	$\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$
Large t (high noise)	small	large
Small t (low noise)	large	small

This makes intuitive sense: when there is little noise (t small), $\mathbf{x}_t \approx \mathbf{x}_0$ already and the posterior trusts $\mathbf{x}_0$ heavily. When noise is high (t large), $\mathbf{x}_t$ carries less information about $\mathbf{x}_0$, so the posterior relies more on $\mathbf{x}_t$ itself.

Checkpoint: We Know What to Match

Where we are in the roadmap:

- ✓ Write the ELBO in terms of the forward and reverse chains
- ✓ Rearrange it into a sum of KL divergences, one per timestep
- ✓ Figure out what distribution the reverse process should match at each step
- ☐ Show that matching means requires predicting noise
- ☐ Conclude: the ELBO objective is exactly $\|\epsilon - \epsilon_\theta\|^2$

What we just computed: The distribution we need to match is

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\mathbf{x}_{t-1};\ \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0),\ \tilde{\beta}_t\,\mathbf{I}\big),$$

where $\tilde{\mu}_t$ and $\tilde{\beta}_t$ have closed-form expressions in terms of the noise schedule.

What's left: Choose a parameterization for $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and show that minimizing $D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)$ is equivalent to predicting the noise ϵ.

Choosing a Form for p_θ: Two Design Decisions

We now need to specify the form of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Nothing in the ELBO derivation forces a particular choice—this is where we make **modeling decisions**.

Design decision 1: The forward posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is Gaussian, so we *choose* $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to be Gaussian as well.

Design decision 2: We *fix* the variance of p_θ to match the forward posterior variance $\tilde{\beta}_t \mathbf{I}$, and *only learn the mean* (Ho et al., 2020).

With these choices, the ELBO's KL terms compare two Gaussians with the same variance:

Forward posterior (known given $\mathbf{x}_0$)

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0),\ \tilde{\beta}_t \mathbf{I}\big)$$

$\mathcal{L}_t = D_{\mathrm{KL}}$

Learned reverse (our design choice)

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\big(\mu_\theta(\mathbf{x}_t, t),\ \tilde{\beta}_t \mathbf{I}\big)$$

KL Between Gaussians with Matched Variance

When two Gaussians share the same covariance Σ, their KL divergence reduces to a squared Mahalanobis distance between their means:

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu_1, \Sigma) \,\|\, \mathcal{N}(\mu_2, \Sigma)\big) = \tfrac{1}{2}(\mu_1 - \mu_2)^\top \Sigma^{-1}(\mu_1 - \mu_2).$$

(This follows from the general KL formula for Gaussians. The log det and trace terms cancel when $\Sigma_1 = \Sigma_2$.)

Applying this to $\mathcal{L}_t$ with $\Sigma = \tilde{\beta}_t \, \mathbf{I}$:

$$\mathcal{L}_t = \frac{1}{2\tilde{\beta}_t} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|^2 .$$

Training the reverse process reduces to making its predicted mean $\mu_\theta(\mathbf{x}_t, t)$ match the true reverse mean $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$.

From Mean Matching to Noise Prediction

We want $\mu_\theta(\mathbf{x}_t, t) \approx \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$.

The true mean from the previous slide was:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\,\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\,\mathbf{x}_t.$$

This depends on $\mathbf{x}_0$, which we do not have at generation time.

Rearranging the one-shot formula: $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon})$.

Substituting and collecting the $\mathbf{x}_t$ terms:

$$\tilde{\mu}_t(\mathbf{x}_t, \boldsymbol{\epsilon}) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\right).$$

The true mean is $\mathbf{x}_t$ corrected by $\boldsymbol{\epsilon}$. We already have $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ (from score matching) to predict it, so we parameterize:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right).$$

The KL Reduces to Noise Prediction

Substituting $\tilde{\mu}_t$ and μ_θ into the matched-variance KL formula from two slides ago, the $\mathbf{x}_t$ terms cancel (both means have the same $\mathbf{x}_t/\sqrt{\alpha_t}$ component), leaving:

$$\mathcal{L}_t = \frac{\beta_t^2}{2\tilde{\beta}_t\,\alpha_t(1-\bar{\alpha}_t)}\,\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\,.$$

Each ELBO term $\mathcal{L}_t$ is a *weighted* squared error between the true noise $\boldsymbol{\epsilon}$ and our network's prediction $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$.

The weight $w_t = \frac{\beta_t^2}{2\tilde{\beta}_t\,\alpha_t(1-\bar{\alpha}_t)}$ depends only on the noise schedule—it varies with t but has no learnable parameters.

The Simplified Objective

Summing over all timesteps, the ELBO loss is a weighted sum of noise-prediction errors:

$$-\mathcal{L}_{\text{ELBO}} = \sum_{t=2}^{T} w_t \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2 + (\text{prior} + \text{reconstruction}),$$

where $w_t = \dfrac{\beta_t^2}{2\tilde{\beta}_t\, \alpha_t(1-\bar{\alpha}_t)}$ depends only on the noise schedule.

Ho et al. (2020) found empirically that *dropping the weights* w_t and using the unweighted objective

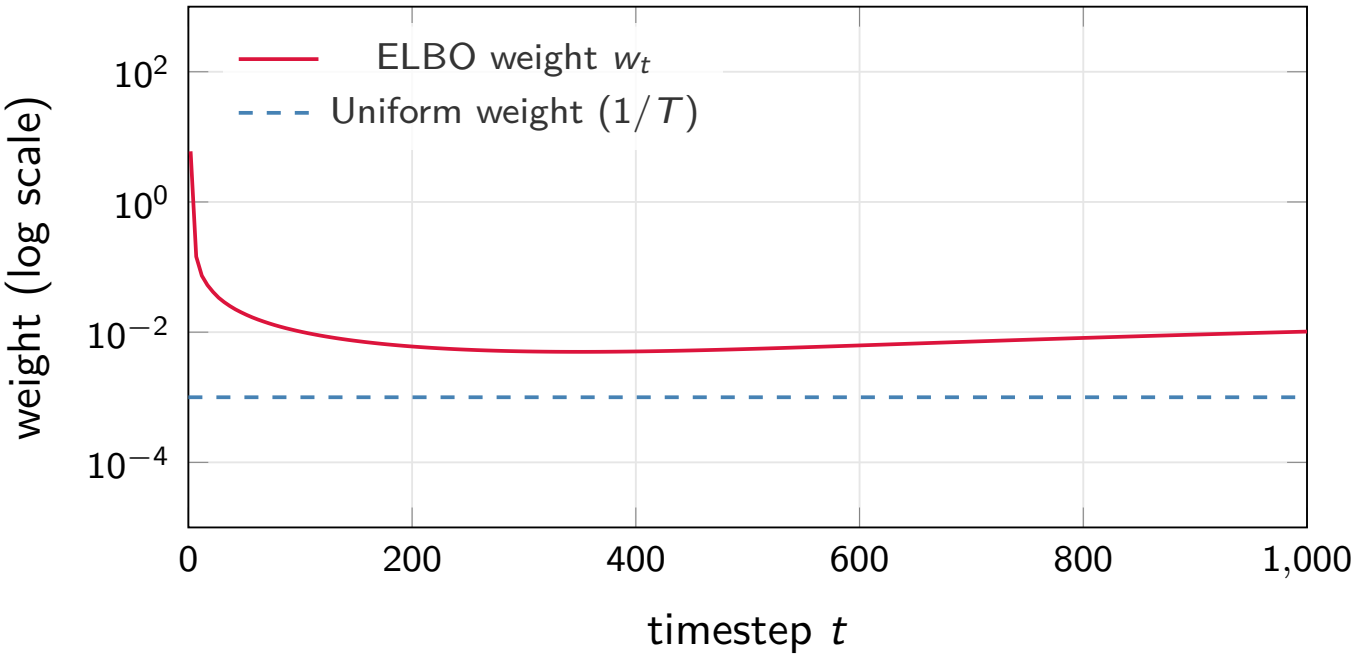
$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\, \mathbf{x}_0,\, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2 \right]$$

works better in practice.

Why would ignoring the theoretically correct weights help? The next slide visualizes the difference.

Visualizing ELBO Weights vs. Uniform Weights

Using the linear schedule $\beta_1 = 10^{-4}$, $\beta_T = 0.02$, $T = 1000$:



The ELBO weights w_t grow *exponentially*, reaching over 200 at t=1000. The theoretically correct objective heavily emphasizes high-noise timesteps.

Uniform weighting $(1/T)$ ignores this imbalance. Empirically, Ho et al. found uniform weighting works *better*—learning fine details at low t matters more for sample quality than the ELBO suggests.

Recap: From ELBO to ϵ-Prediction

All 5 steps complete

Where we are in the roadmap:

- ✓ Write the ELBO in terms of the forward and reverse chains
- ✓ Rearrange it into a sum of KL divergences, one per timestep
- ✓ Figure out what distribution the reverse process should match at each step
- ✓ Show that matching means requires predicting noise
- ✓ Conclude: the ELBO objective is exactly $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\|^2$

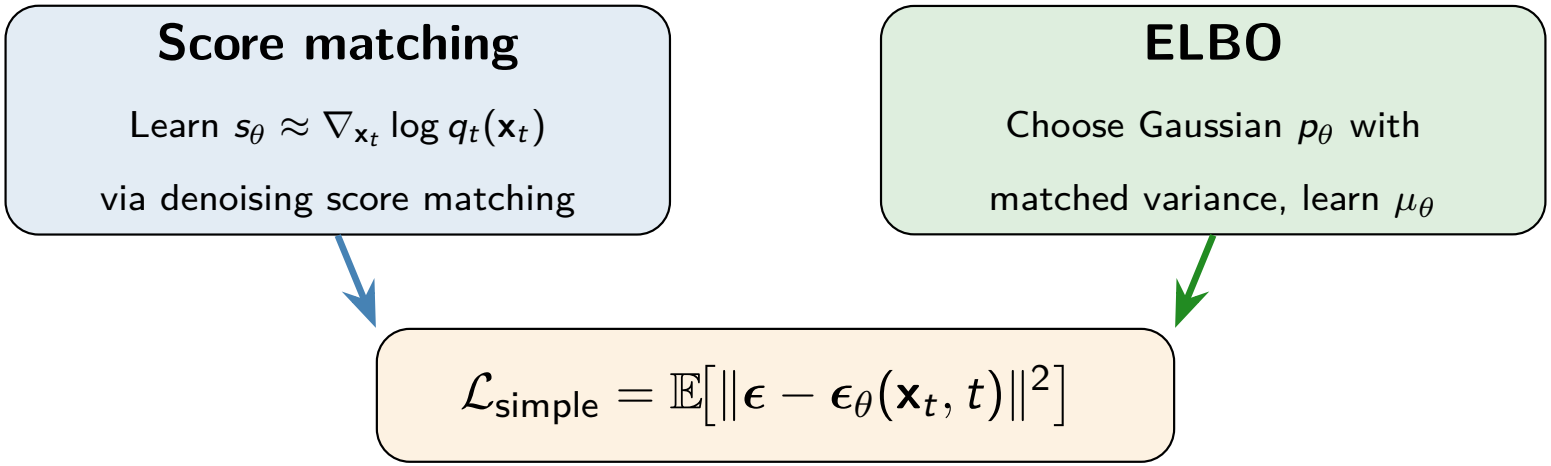
The punchline: Maximizing the ELBO is equivalent to minimizing

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\, \mathbf{x}_0,\, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2 \right].$$

This is exactly what we got from score matching. Two independent derivations, same objective.

Two Roads to the Same Objective

We arrived at $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$ by two independent routes:



Each route provides a different guarantee about what $\boldsymbol{\epsilon}_\theta$ is doing:

Perspective	**Guarantee**	**Design choices**
Score matching	Learns $\nabla \log q_t$	ϵ-parameterization
ELBO	Maximizes likelihood bound	Gaussian p_θ, matched var.

In practice, use whichever view helps you reason about the problem at hand.