Diffusion Models: Theory and Applications

Lecture 7: Score-Based Generative Models - Learning the Geometry of Data

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

June 12, 2025

Last Time: We Mastered Conditional Generation ✔

We learned to control what diffusion models generate...

But WHY do these models work so well? ❓

- ✔ **Class-conditional:** Simple embedding-based conditioning
- ✔ **Classifier guidance:** External steering with gradients
- ✔ **Classifier-free guidance:** The modern breakthrough
- 🚀 **Today:** The geometric foundation underlying it all!

The Lost Hiker Analogy: Understanding Score Functions

Image source: https://stock.adobe.com/in/images/mountain-travel-lost-man-with-the-map-in-the-forest/146343754

Imagine you're lost in a mountainous landscape at night...

- You're trying to find the highest peak where a bonfire awaits
- You can't see the entire terrain (the full probability distribution)
- But you have a magical compass that points toward the steepest uphill direction
- By following this compass step by step, you eventually reach the summit

The Lost Hiker Analogy: Understanding Score Functions

Image source: https://stock.adobe.com/in/images/mountain-travel-lost-man-with-the-map-in-the-forest/146343754

This is exactly the intuition behind score-based models!

- **The mountain:** Probability density $p(\mathbf{x})$
- **The peaks:** High-probability regions (data modes)
- **The magical compass:** Score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- **Following the compass:** Langevin dynamics sampling

Score functions provide a navigation system through probability space!

The Lost Hiker Analogy: Understanding Score Functions

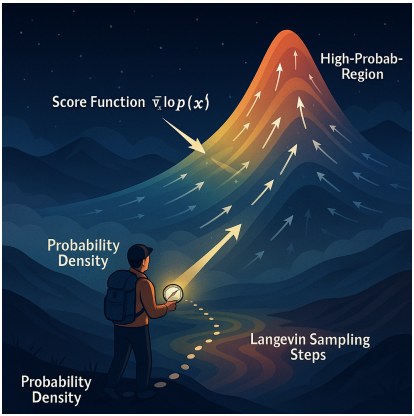

Figure: Score functions provide a navigation system through probability space!

Image generated using GPT-4o

Today's Mission: Understanding the "Why"

What makes diffusion models fundamentally work?

The Deep Questions:

- Why does noise prediction work?
- What is the model really learning?
- How does this connect to probability theory?
- What's the geometric intuition?

Score-Based Perspective:

- **Score functions:** Gradients of log-probability
- **Geometric view:** Learning the shape of data
- **Equivalence:** Deep connection to diffusion
- **Unified theory:** One framework, multiple views

From algorithms to understanding - the geometry of generation!

A Different Perspective on Generation

The fundamental shift: instead of learning densities, learn directions!

Traditional approaches face the normalization problem

Most probability models have the form: $p(\mathbf{x}) = \frac{1}{Z}\tilde{p}(\mathbf{x})$
Problem: Computing $Z = \int \tilde{p}(\mathbf{x})d\mathbf{x}$ is intractable!

Score-based insight: The partition function vanishes!

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \nabla_{\mathbf{x}} \log \left(\frac{1}{Z}\tilde{p}(\mathbf{x}) \right) \tag{1}$$

$$= \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}) - \nabla_{\mathbf{x}} \log Z \tag{2}$$

$$= \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}) \quad \text{(since } Z \text{ doesn't depend on } \mathbf{x}) \tag{3}$$

This seemingly minor shift has profound implications!

The Score Function: Your Probability Compass

Definition: Score Function

For a probability density function $p(\mathbf{x})$, the **score function** is:

$$\boxed{\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})} \tag{4}$$

Breaking it down

- $\log p(\mathbf{x})$: Taking logarithm transforms products to sums (more tractable)
- $\nabla_{\mathbf{x}}$: Computes direction and magnitude of steepest increase
- $\mathbf{s}(\mathbf{x})$: Vector field pointing toward higher likelihood regions

Simple example: 1D Gaussian

For $p(x) = \mathcal{N}(x; \mu, \sigma^2)$:

$$s(x) = \frac{d}{dx} \log p(x) = -\frac{x - \mu}{\sigma^2} \tag{5}$$

Intuition: Score points toward mean μ with strength inversely proportional to variance!

Geometric Intuition: The Probability Landscape

Think of probability density as a mountainous landscape...

The Landscape

- **Peaks:** High-probability regions (modes)
- **Valleys:** Low-probability regions
- **Slopes:** Rate of probability change
- **Height:** Probability density $p(\mathbf{x})$

Score Vector Properties

- **Direction:** Points "uphill" toward higher probability
- **Magnitude:** Large where probability changes rapidly (steep slopes)
- **Zero:** Exactly at local maxima (peaks)
- **Roadmap:** Complete navigation system to high-density regions

Key insight

Score functions provide arrows scattered throughout the landscape, each pointing toward the nearest "realistic" data!

Energy-Based Perspective: Nature's Way

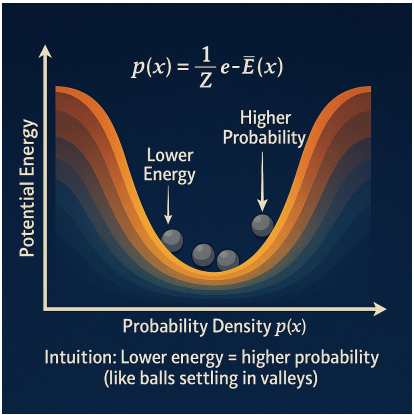

Figure: Another lens: think about data in terms of energy. Any probability distribution can be written as: $p(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})}$ *Image generated using GPT-4o*

Energy-Based Perspective: Nature's Way

Score = Negative energy gradient

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) = -\nabla_{\mathbf{x}} E(\mathbf{x}) \tag{6}$$

Physical interpretation: Score function is the "force" pushing toward lower energy!

Beautiful insight

- Natural, realistic data has **low energy**
- Unnatural, unrealistic data has **high energy**
- Score function points toward more natural configurations
- We learn the force field without computing the intractable Z!

Concrete Examples: Seeing Scores in Action

Let's build intuition with distributions we understand...

Example 1: Multivariate Gaussian

For $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\mathbf{s}(\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) \tag{7}$$

Beautiful properties:

- Always points toward mean $\boldsymbol{\mu}$
- Strength depends on $\boldsymbol{\Sigma}^{-1}$ (inverse covariance)
- Far points experience strong "pull", near points gentle guidance

Example 1: Multivariate Gaussian Score Function

The Setup

For a multivariate Gaussian distribution $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, we want to find:

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) \tag{8}$$

Why this matters:

- Gaussians are fundamental in diffusion models
- Closed-form solution provides intuition
- Foundation for understanding score matching
- Connects to denoising objectives

Our journey: PDF $\rightarrow$ Log-PDF $\rightarrow$ Gradient $\rightarrow$ Geometric Insight

Step 1: Write the Full Probability Density

The multivariate Gaussian PDF in all its glory:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{9}$$

Breaking down the components:

- d: dimensionality of $\mathbf{x}$
- $(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}$: normalization constant
- $|\mathbf{\Sigma}|$: determinant of covariance matrix
- $(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$: Mahalanobis distance squared

Key Insight

The exponential contains all the $\mathbf{x}$-dependence we care about for computing gradients!

Step 2: Take the Natural Logarithm

Why logarithms are magical:

- Convert exponentials to polynomials
- Turn products into sums
- Make derivatives much simpler!

Breaking down $\log p(\mathbf{x})$**:**

$$\log p(\mathbf{x}) = \log \left[\frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \right] + \log \left[\exp \left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \right) \right] \tag{10}$$

$$= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \tag{11}$$

The Magic Moment

The scary exponential becomes a simple quadratic form!

Step 3: Identify **x**-dependent Terms

For the gradient $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, constants vanish:

$$\log p(\mathbf{x}) = \underbrace{-\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}|}_{\text{constants w.r.t. } \mathbf{x}} - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \tag{12}$$

Simplification:

$$\boxed{\log p(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) + \text{const}} \tag{13}$$

Calculus Reminder

$\frac{d}{dx}[f(x) + C] = \frac{d}{dx}f(x)$ — constants disappear when differentiating!

What's left: A beautiful quadratic form in **x**

Step 4: Matrix Calculus Magic

Let $\mathbf{z} = \mathbf{x} - \boldsymbol{\mu}$. We need to compute:

$$\frac{\partial}{\partial \mathbf{x}} \left[-\frac{1}{2} \mathbf{z}^T \boldsymbol{\Sigma}^{-1} \mathbf{z} \right] \tag{14}$$

Key Matrix Calculus Identity

For symmetric matrix $\mathbf{A}$:

$$\frac{\partial}{\partial \mathbf{x}} (\mathbf{x}^T \mathbf{A} \mathbf{x}) = 2\mathbf{A}\mathbf{x} \tag{15}$$

Applying the identity: Since $\boldsymbol{\Sigma}^{-1}$ is symmetric (covariance matrices are always symmetric):

$$\nabla_{\mathbf{x}} \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] = -\frac{1}{2} \cdot 2\boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \tag{16}$$

$$= -\boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \tag{17}$$

Step 5: The Beautiful Final Result

Gaussian Score Function

$$\boxed{\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})} \tag{18}$$

Immediate observations:

- **Direction:** Always points toward the mean $\boldsymbol{\mu}$
- **Magnitude:** Proportional to distance from mean
- **Shaping:** $\boldsymbol{\Sigma}^{-1}$ (precision matrix) determines strength
- **Linearity:** Simple linear function of $(\mathbf{x} - \boldsymbol{\mu})$

Intuitive Check

- At the mean ($\mathbf{x} = \boldsymbol{\mu}$): $\mathbf{s}(\boldsymbol{\mu}) = \mathbf{0}$
- Far from mean: large magnitude pointing toward $\boldsymbol{\mu}$
- High variance directions: weaker pull (small eigenvalues of $\boldsymbol{\Sigma}^{-1}$)

Concrete Examples: Seeing Scores in Action

Let's build intuition with distributions we understand...

Example 2: Mixture of Gaussians

For mixture $p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$:

$$\mathbf{s}(\mathbf{x}) = \sum_{k=1}^{K} \gamma_k(\mathbf{x}) \cdot (-\boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)) \tag{19}$$

where $\gamma_k(\mathbf{x})$ is the posterior probability of component k.
Insight: Weighted combination automatically balances competing influences from multiple modes!

Sampling Without Likelihood: The Power of Score Functions

The remarkable property that revolutionized generative modeling...

The profound insight

If we can estimate $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ accurately, we can sample from $p(\mathbf{x})$ without ever computing $p(\mathbf{x})$ itself!

This bypasses the intractable partition function that has plagued probabilistic modeling for decades!

But how do we turn this gradient into actual samples?

Langevin Dynamics: Turning Scores into Samples

The solution: Langevin Dynamics

An elegant physics-inspired process that lets us **walk through probability space**.

- **Systematic guidance:** Follow the score field $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ to move uphill toward high-density regions
- **Random exploration:** Inject noise at each step to avoid getting trapped in local modes

This combination enables efficient sampling from complex, high-dimensional distributions!

Back to our hiker analogy

Imagine the hiker now has two tools:

- **Magical compass:** Points toward nearest peak (score function)
- **Random steps:** Occasionally moves in unexpected directions

This combination ensures the hiker:

- ✔ Doesn't get stuck in local valleys
- ✔ Eventually explores all peaks in the mountain range
- ✔ Spends time proportional to peak height (probability)

Key Result:

Systematic + Stochastic

$\Downarrow$

Perfect Sampling

No normalization needed!

The Magic: Generation via Langevin Dynamics

Langevin Dynamics Algorithm

Input: Score function $\mathbf{s}(\mathbf{x})$, step size η, steps T
Initialize $\mathbf{x}_0$ randomly (e.g., from $\mathcal{N}(\mathbf{0}, \mathbf{I})$)
for $t = 0, 1, \ldots, T-1$ **do**
    Sample $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \mathbf{s}(\mathbf{x}_t) + \sqrt{2\eta}\boldsymbol{\epsilon}_t$
end for
Return: $\mathbf{x}_T$

Understanding the update rule

- $\eta \mathbf{s}(\mathbf{x}_t)$: **Drift term** - systematic move toward higher probability
- $\sqrt{2\eta}\boldsymbol{\epsilon}_t$: **Diffusion term** - controlled noise for exploration
- Balance prevents getting stuck in local modes while making progress

The Learning Challenge: How to Learn the Compass?

The fundamental chicken-and-egg problem

- We want to learn $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ to sample from $p(\mathbf{x})$
- But to compute $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, we need to know $p(\mathbf{x})$
- We don't know $p(\mathbf{x})$—learning it is the whole point!

Naive approach (doesn't work!)

Train neural network $\mathbf{s}_\theta(\mathbf{x})$ to minimize:

$$\mathcal{L}_{\text{naive}} = \mathbb{E}_{p(\mathbf{x})} \left[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2 \right] \tag{20}$$

Fatal flaw: We don't know $\nabla_{\mathbf{x}} \log p(\mathbf{x})$—that's what we're trying to learn!

Solution: Score matching - a mathematical miracle!

The score matching insight

Question: Can we transform this intractable loss into something we *can* compute?

Answer: Yes! Through integration by parts, we can eliminate the unknown true score entirely.

Let's see this mathematical magic in action...

Score Matching: The Mathematical Transformation

Step-by-step derivation of the miracle...

- Starting with the intractable naive loss:

$$\mathcal{L}_{\text{naive}} = \mathbb{E}_{p(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2\right] \tag{21}$$

- Expanding the squared norm:

$$\mathcal{L}_{\text{naive}} = \mathbb{E}_{p(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x})\|^2 - 2\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \|\nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2\right] \tag{22}$$

Key observation

- **First term:** $\mathbb{E}[\|\mathbf{s}_\theta(\mathbf{x})\|^2]$ - depends only on our learned function
- **Second term:** $\mathbb{E}[2\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x})]$ - involves unknown true score
- **Third term:** $\mathbb{E}[\|\nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2]$ - constant w.r.t. θ (can ignore for optimization)

The magic happens in transforming the problematic second term...

The Challenge: One Problematic Term

After expanding, we identified our roadblock...

What we learned from expansion

$$\mathcal{L}_{\text{naive}} = \mathbb{E}_{p(\mathbf{x})}\Big[\|\mathbf{s}_\theta(\mathbf{x})\|^2 \tag{23}$$

$$-2\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x}) \tag{24}$$

$$+\|\nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2\Big] \tag{25}$$

The bottleneck

Second term: $\mathbb{E}[\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x})]$ contains the unknown true score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
Question: Can we somehow transform this term to eliminate the unknown?

We need a mathematical trick to "move" the unknown... ?

The Strategy: Move Derivatives Around

The core idea

Current situation: We have derivatives of the unknown $p(\mathbf{x})$
Goal: Transform to get derivatives of the known $\mathbf{s}_\theta(\mathbf{x})$
Mathematical tool: Integration by parts — swaps which function gets differentiated!

Integration by parts intuition

General principle: $\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$
What this does:

- **Original integral:** u multiplied by derivative of v
- **Transformed integral:** v multiplied by derivative of u
- **Key insight:** Swaps which function gets differentiated!

Plan: Make the unknown $p(\mathbf{x})$ "plain" and differentiate the known $\mathbf{s}_\theta(\mathbf{x})$!

Why This Strategy Will Work

The beautiful consequence of our plan...

What integration by parts will give us

Before transformation:

$$\int \mathbf{s}_\theta(\mathbf{x}) \cdot \frac{\partial}{\partial x_i}[\text{unknown}]\, dx \tag{26}$$

After transformation:

$$\int \frac{\partial}{\partial x_i}[\mathbf{s}_\theta(\mathbf{x})] \cdot \text{unknown}\, dx \tag{27}$$

Why this is revolutionary

- **New derivative term:** $\frac{\partial \mathbf{s}_\theta(\mathbf{x})}{\partial x_i}$ — we can compute this!
- **Unknown part:** Becomes just $p(\mathbf{x})$ (no derivatives)
- **Final step:** The $p(\mathbf{x})$ becomes our expectation — computable from data!

Step 1: From Expectation to Integral ➔

Converting our problematic term to integral form...

Our target: the problematic term

$$\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x})\right] \tag{28}$$

Using the definition of expectation

By definition, $\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] = \int f(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x}$
Applying this to our case:

$$\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x})\right] = \int \mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x}) \cdot p(\mathbf{x})\, d\mathbf{x} \tag{29}$$

Now we have an integral we can work with!

Step 2: Understanding the Chain Rule for Logarithms

The key mathematical identity we need...

Recall: Chain rule for logarithmic derivatives

For any function $f(\mathbf{x})$, the chain rule gives us:

$$\frac{d}{d\mathbf{x}} \log f(\mathbf{x}) = \frac{1}{f(\mathbf{x})} \cdot \frac{df(\mathbf{x})}{d\mathbf{x}} \tag{30}$$

Applying to our probability density

For $f(\mathbf{x}) = p(\mathbf{x})$:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \cdot \nabla_{\mathbf{x}} p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})} \tag{31}$$

This is just basic calculus - but it's the key to our transformation!

Step 3: The Crucial Algebraic Manipulation

Starting from the chain rule result

We just showed that:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})} \tag{32}$$

Multiply both sides by $p(\mathbf{x})$

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) \cdot p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})} \cdot p(\mathbf{x}) = \nabla_{\mathbf{x}} p(\mathbf{x}) \tag{33}$$

The beautiful result

$$\boxed{\nabla_{\mathbf{x}} \log p(\mathbf{x}) \cdot p(\mathbf{x}) = \nabla_{\mathbf{x}} p(\mathbf{x})} \tag{34}$$

We've converted log-derivatives to plain derivatives!

Step 4: Applying the Transformation

Substituting our identity into the integral...

Our integral before transformation ←

$$\int \mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} \log p(\mathbf{x}) \cdot p(\mathbf{x})\, d\mathbf{x} \tag{35}$$

Using our identity: $\nabla_\mathbf{x} \log p(\mathbf{x}) \cdot p(\mathbf{x}) = \nabla_\mathbf{x} p(\mathbf{x})$ ↓

$$\int \mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} \log p(\mathbf{x}) \cdot p(\mathbf{x})\, d\mathbf{x} = \int \mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} p(\mathbf{x})\, d\mathbf{x} \tag{36}$$

Perfect! Now we have derivatives of $p(\mathbf{x})$, not $\log p(\mathbf{x})$! ✔

Ready for integration by parts! 🚀

Setting Up Integration by Parts

Breaking it down component-wise

Since $\mathbf{s}_\theta(\mathbf{x})$ is a vector, we can write:

$$\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} p(\mathbf{x}) = \sum_{i=1}^{d} s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} \tag{37}$$

So our integral becomes:

$$\int \sum_{i=1}^{d} s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} \, d\mathbf{x} = \sum_{i=1}^{d} \int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} \, d\mathbf{x} \tag{38}$$

Now we can apply integration by parts to each component separately! ↓

Integration by Parts Formula Reminder

The basic integration by parts formula

For functions $u(x)$ and $v(x)$ on interval $[a, b]$:

$$\int_a^b u(x)\frac{dv}{dx}dx = u(x)v(x)\Big|_a^b - \int_a^b v(x)\frac{du}{dx}dx \tag{39}$$

In multivariable form (Divergence Theorem)

For integration over all space $\mathbb{R}^d$:

$$\int_{\mathbb{R}^d} u(\mathbf{x})\frac{\partial v(\mathbf{x})}{\partial x_i}d\mathbf{x} = \underbrace{\lim_{R\to\infty}\int_{\partial B_R} u(\mathbf{x})v(\mathbf{x})n_i\, dS}_{\text{surface integral}} - \int_{\mathbb{R}^d} v(\mathbf{x})\frac{\partial u(\mathbf{x})}{\partial x_i}d\mathbf{x} \tag{40}$$

Where B_R is a ball of radius R and n_i is the i-th component of the outward normal.

The key insight

Effect: Transfers the derivative from v to u

- **Before:** $u \times$ (derivative of v)
- **After:** $v \times$ (derivative of u) (plus surface integral)
- **Surface integral:** Vanishes under regularity conditions

Choosing u and v for Our Problem

Our integral for component i

$$\int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} \tag{41}$$

Strategic choice

Let:

- $u = s_{\theta,i}(\mathbf{x})$ (the function we know and can differentiate)
- $v = p(\mathbf{x})$ (the unknown density)

Why this choice?

- We want to get rid of $\frac{\partial p(\mathbf{x})}{\partial x_i}$ (derivative of unknown)
- We're okay with getting $\frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i}$ (derivative of known)

Applying Integration by Parts

Using the formula with our choices →

Formula: $\int_{\mathbb{R}^d} u \frac{\partial v}{\partial x_i} d\mathbf{x} = \lim_{R\to\infty} \int_{\partial B_R} uvn_i \, dS - \int_{\mathbb{R}^d} v \frac{\partial u}{\partial x_i} d\mathbf{x}$
Our substitution: $u = s_{\theta,i}(\mathbf{x})$, $v = p(\mathbf{x})$

The transformation

$$\int_{\mathbb{R}^d} s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} = \lim_{R\to\infty} \int_{\partial B_R} s_{\theta,i}(\mathbf{x}) p(\mathbf{x}) n_i \, dS \tag{42}$$

$$- \int_{\mathbb{R}^d} p(\mathbf{x}) \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} d\mathbf{x} \tag{43}$$

What we achieved ★

- **Eliminated:** $\frac{\partial p(\mathbf{x})}{\partial x_i}$ (derivative of unknown)
- **Introduced:** $\frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i}$ (derivative of known)
- **Plus:** A surface integral we need to handle

Why the Surface Integral Vanishes

The surface integral in question ?

$$\lim_{R \to \infty} \int_{\partial B_R} s_{\theta,i}(\mathbf{x}) p(\mathbf{x}) n_i \, dS \tag{44}$$

This integrates $s_{\theta,i}(\mathbf{x})p(\mathbf{x})$ over the surface of a sphere of radius R as $R \to \infty$.

Why it vanishes: Decay rates

- **Probability densities:** $p(\mathbf{x}) \to 0$ exponentially fast as $\|\mathbf{x}\| \to \infty$
- **Score networks:** $s_{\theta,i}(\mathbf{x})$ grows at most polynomially with $\|\mathbf{x}\|$
- **Surface area:** Grows like R^{d-1} (polynomial in R)
- **Mathematical result:** Exponential decay dominates polynomial growth

Intuitive explanation

- Far from data (large $\|\mathbf{x}\|$): probability $p(\mathbf{x}) \approx 0$
- Even if score is large, $s_{\theta,i}(\mathbf{x}) \times$ tiny $p(\mathbf{x}) \approx 0$
- The surface integral goes to zero faster than the surface area grows
- **Result:** Surface integral vanishes in the limit $R \to \infty$

The Final Result: Component i ✅

For component i, we started with ←

$$\int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} \tag{45}$$

After integration by parts →

$$\int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} = -\int p(\mathbf{x}) \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} d\mathbf{x} \tag{46}$$

(boundary term vanished)

Converting back to expectation ↓

$$-\int p(\mathbf{x}) \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} d\mathbf{x} = -\mathbb{E}_{p(\mathbf{x})} \left[\frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \right] \tag{47}$$

Beautiful! No unknown functions remain - only derivatives of our learned score! 🪄

Summing Over All Components ➕

Recall: We had a sum over all components 🔢

$$\int \mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} p(\mathbf{x})\, d\mathbf{x} = \sum_{i=1}^{d} \int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i}\, d\mathbf{x} \tag{48}$$

Each component gave us 🟰

$$\int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} = -\mathbb{E}_{p(\mathbf{x})} \left[\frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \right] \tag{49}$$

Summing Over All Components +

Summing over all $i = 1, 2, \ldots, d$

$$\sum_{i=1}^{d} \int s_{\theta,i}(\mathbf{x}) \frac{\partial p(\mathbf{x})}{\partial x_i} d\mathbf{x} = -\sum_{i=1}^{d} \mathbb{E}_{p(\mathbf{x})} \left[\frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \right] \tag{50}$$

$$= -\mathbb{E}_{p(\mathbf{x})} \left[\sum_{i=1}^{d} \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \right] \tag{51}$$

$$= -\mathbb{E}_{p(\mathbf{x})} \left[\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) \right] \tag{52}$$

But what is this sum of partial derivatives? ?

What is $\sum_{i=1}^{d} \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i}$? ?

Understanding the mathematical object we've created...

The Jacobian matrix

For vector function $\mathbf{s}_\theta(\mathbf{x}) = [s_{\theta,1}(\mathbf{x}), s_{\theta,2}(\mathbf{x}), \ldots, s_{\theta,d}(\mathbf{x})]^T$:

$$\text{Jacobian } J = \nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}) = \begin{bmatrix} \frac{\partial s_{\theta,1}}{\partial x_1} & \frac{\partial s_{\theta,1}}{\partial x_2} & \cdots & \frac{\partial s_{\theta,1}}{\partial x_d} \\ \frac{\partial s_{\theta,2}}{\partial x_1} & \frac{\partial s_{\theta,2}}{\partial x_2} & \cdots & \frac{\partial s_{\theta,2}}{\partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial s_{\theta,d}}{\partial x_1} & \frac{\partial s_{\theta,d}}{\partial x_2} & \cdots & \frac{\partial s_{\theta,d}}{\partial x_d} \end{bmatrix} \tag{53}$$

What is $\sum_{i=1}^{d} \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i}$? ?

The trace of the Jacobian

$$\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) = \sum_{i=1}^{d} \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \tag{54}$$

Trace = Sum of diagonal elements = each output component's derivative w.r.t. its corresponding input

Why Do We Want the Trace? The Divergence Connection

The geometric meaning behind our mathematical result...

The divergence of a vector field

For any vector field $\mathbf{v}(\mathbf{x}) = [v_1(\mathbf{x}), v_2(\mathbf{x}), \ldots, v_d(\mathbf{x})]^T$:

$$\text{div}(\mathbf{v}) = \nabla \cdot \mathbf{v} = \sum_{i=1}^{d} \frac{\partial v_i(\mathbf{x})}{\partial x_i} \tag{55}$$

Geometric meaning: How much the vector field "spreads out" or "contracts" at each point

Our Score Function Is A Vector Field!

- $\mathbf{s}_\theta(\mathbf{x})$ assigns a vector (direction + magnitude) to each point $\mathbf{x}$
- The trace $\text{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x}))$ is the divergence of this vector field
- It measures how much our learned score field "expands" or "contracts" locally

Why this matters for score matching

The divergence term ensures our learned score field has the right "flow properties" - it can't be too wild or unrealistic compared to true probability score fields.

The Final Integration by Parts Result

What we computed

$$\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{s}_\theta(\mathbf{x})^T \nabla_{\mathbf{x}} \log p(\mathbf{x})\right] = -\mathbb{E}_{p(\mathbf{x})}\left[\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}))\right] \tag{56}$$

In terms we now understand

Left side: Problematic term with unknown true score
Right side: Negative expectation of the divergence of our learned score field
Beautiful result: We transformed unknown derivatives into known derivatives!

Why this is exactly what we wanted

- **Computable:** Only involves our learned function $\mathbf{s}_\theta(\mathbf{x})$
- **Geometric:** Has clear interpretation as vector field divergence
- **Regularizing:** Prevents unrealistic score field behavior
- **Equivalent:** Mathematically identical to the original impossible loss

Putting It All Together: The Complete Transformation

Substituting our integration by parts result back into the original loss...

Recall

- We wanted to transform the problematic term: $\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} \log p(\mathbf{x})\right]$
- What integration by parts gave us:
  $\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} \log p(\mathbf{x})\right] = -\mathbb{E}_{p(\mathbf{x})}\left[\text{tr}(\nabla_\mathbf{x} \mathbf{s}_\theta(\mathbf{x}))\right]$

Substituting back into our expanded naive loss

$$\mathcal{L}_{\text{naive}} = \mathbb{E}_{p(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x})\|^2 - 2\mathbf{s}_\theta(\mathbf{x})^T \nabla_\mathbf{x} \log p(\mathbf{x}) + \|\nabla_\mathbf{x} \log p(\mathbf{x})\|^2\right] \tag{57}$$

$$= \mathbb{E}_{p(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x})\|^2 + 2\text{tr}(\nabla_\mathbf{x} \mathbf{s}_\theta(\mathbf{x})) + \|\nabla_\mathbf{x} \log p(\mathbf{x})\|^2\right] \tag{58}$$

We're almost there - one more step! ↓

The Score Matching Miracle: Final Result

Ignoring constants for optimization →

The term $\mathbb{E}_{p(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log p(\mathbf{x})\|^2\right]$ doesn't depend on θ, so we can ignore it.

The Score Matching objective ◎

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{p(\mathbf{x})}\left[\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|^2\right] \tag{59}$$

What we achieved ★

- **Eliminated:** All dependence on unknown true score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- **Computable:** Everything depends only on our learned function $\mathbf{s}_\theta(\mathbf{x})$
- **Tractable:** Can estimate this loss from data samples
- **Equivalent:** Minimizing this minimizing the original impossible loss

Mathematical miracle complete! ✔

Score Matching: Beautiful but Expensive

The elegant solution has a computational catch...

The computational bottleneck

Computing $\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}))$ requires:

$$\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) = \sum_{i=1}^{d} \frac{\partial s_{\theta,i}(\mathbf{x})}{\partial x_i} \tag{60}$$

That's d second-order derivatives per training sample!

The scaling nightmare

- **1D data:** 1 derivative - Easy
- **CIFAR ($32 \times 32 \times 3$):** 3,072 derivatives - Expensive
- **ImageNet ($224 \times 224 \times 3$):** 150,528 derivatives - Intractable

Denoising Score Matching: The Practical Path

Score matching is elegant but computationally expensive...

Computational bottleneck

Computing trace $\text{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}))$ requires:

- Full Jacobian of score network (expensive!)
- Scales quadratically with dimension
- Prohibitive for high-dimensional problems like images

Denoising insight: Learn scores of noisy data

Key idea: Instead of learning score of clean data, learn score of *slightly noisy* data.

$$\mathcal{L}_{\text{DSM}} = \mathbb{E}_{\mathbf{x}_0 \sim p_{data}} \mathbb{E}_{\mathbf{x} \sim \mathcal{N}(\mathbf{x}_0, \sigma^2 \mathbf{I})} \left[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log q(\mathbf{x}|\mathbf{x}_0)\|^2 \right] \tag{61}$$

For Gaussian noise: $\nabla_{\mathbf{x}} \log q(\mathbf{x}|\mathbf{x}_0) = -\frac{\mathbf{x}-\mathbf{x}_0}{\sigma^2}$

Denoising Score Matching: Simple Training

The remarkably simple algorithm for learning score functions...

Training procedure

Input: Training data $\{\mathbf{x}_i\}$, noise level σ
while not converged **do**
    Sample clean data point $\mathbf{x}_0$ from training set
    Sample noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    Create noisy version: $\mathbf{x} = \mathbf{x}_0 + \sigma\boldsymbol{\epsilon}$
    Compute loss: $\mathcal{L} = \left\| \mathbf{s}_\theta(\mathbf{x}) + \frac{\boldsymbol{\epsilon}}{\sigma} \right\|^2$
    Update parameters θ using gradient descent
end while

Denoising Score Matching: Simple Training

The beautiful insight ★

From the loss $\mathcal{L} = \left\| \mathbf{s}_\theta(\mathbf{x}) + \frac{\boldsymbol{\epsilon}}{\sigma} \right\|^2$, we get:

$$\mathbf{s}_\theta(\mathbf{x}) = -\frac{\boldsymbol{\epsilon}}{\sigma} \tag{62}$$

Learning score functions = Learning to predict the noise that was added!
This connects directly to denoising autoencoders and diffusion models!

The Goldilocks Problem: Single Noise Level Issues ⚠

Choosing the right noise level is surprisingly difficult...

Small σ (low noise): ↓

+ Accurate near data points
+ Preserves fine details
- Poor coverage in empty regions
- Langevin sampling gets stuck
- Never learns to navigate globally

Large σ (high noise): ↑

+ Good coverage everywhere
+ Learns global navigation
- Over-smooths distribution
- Loses fine structure
- Generated samples lack details

The Goldilocks Problem: Single Noise Level Issues ⚠

The solution: Learn ALL noise levels! ◎

Train *noise-conditional* score network: $\mathbf{s}_\theta(\mathbf{x}, \sigma)$

$$\mathcal{L} = \mathbb{E}_{\sigma \sim p(\sigma)} \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\lambda(\sigma) \left\| \mathbf{s}_\theta(\mathbf{x}_0 + \sigma\boldsymbol{\epsilon}, \sigma) + \frac{\boldsymbol{\epsilon}}{\sigma} \right\|^2 \right] \tag{63}$$

Like training a universal translator that adapts its behavior based on the noise level!

Breaking Down the Multi-Scale Training Objective

The multi-scale training objective

$$\mathcal{L}_{\text{multi}} = \mathbb{E}_{\sigma\sim p(\sigma)}\mathbb{E}_{\mathbf{x}_0\sim p_{data}}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\lambda(\sigma)\left\|\mathbf{s}_\theta(\mathbf{x}_0+\sigma\boldsymbol{\epsilon},\sigma)+\frac{\boldsymbol{\epsilon}}{\sigma}\right\|^2\right] \tag{64}$$

Component breakdown

- $p(\sigma)$: Noise level distribution - often uniform over $[\sigma_{\text{min}}, \sigma_{\text{max}}]$
- $\lambda(\sigma)$: Weighting function - balances learning across noise levels
- $\mathbf{s}_\theta(\mathbf{x},\sigma)$: Noise-conditional score network
- $\mathbf{x}_0+\sigma\boldsymbol{\epsilon}$: Clean data + scaled noise
- $+\frac{\boldsymbol{\epsilon}}{\sigma}$: True score target (negative noise direction)

Why this works

Each training step teaches the network to handle a different "viewing distance" of the data manifold!

Annealed Langevin: Coarse-to-Fine Generation

With our noise-conditional score network in hand, we can now design a much more sophisticated sampling procedure called annealed Langevin dynamics

The annealing strategy

Input: Decreasing noise schedule $\{\sigma_1 > \sigma_2 > \cdots > \sigma_L\}$
Initialize $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \sigma_1^2 \mathbf{I})$ (start from pure noise)
for $\ell = 1, 2, \ldots, L$ **do** ▷ Each noise level
  for $t = 1, 2, \ldots, T_\ell$ **do** ▷ Multiple Langevin steps
    Sample $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_t = \mathbf{x}_{t-1} + \eta_\ell \mathbf{s}_\theta(\mathbf{x}_{t-1}, \sigma_\ell) + \sqrt{2\eta_\ell}\boldsymbol{\epsilon}_t$
  end for
  $\mathbf{x}_0 = \mathbf{x}_{T_\ell}$ ▷ Prepare for next level
end for
Return: $\mathbf{x}_0$

Annealed Langevin: Coarse-to-Fine Generation

Why annealing works beautifully

- **Large σ (coarse):** Global exploration, find general vicinity of data modes
- **Decreasing σ (refinement):** Gradually reveal finer details
- **Small σ (fine):** Final polishing for realistic details
- **Curriculum learning:** Easy-to-hard progression prevents local optima

The Great Revelation: Score = Diffusion Models!

Setting up the connection

Recall diffusion forward process:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{65}$$

Using the reparameterization: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$

Computing the score of this distribution

$$\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t} \tag{66}$$

$$= -\frac{\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}{1-\bar{\alpha}_t} \tag{67}$$

$$= -\frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}} \tag{68}$$

This is exactly what diffusion models learn to predict!

The Mathematical Rosetta Stone

The precise relationship that unifies everything...

The fundamental equivalence

$$\mathbf{s}_\theta(\mathbf{x}_t, t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} \tag{69}$$

What this miraculous equation means

- **Diffusion models** implicitly learn score functions without calling them that
- **Score-based models** are diffusion models with different parameterization
- **Noise prediction** $\leftrightarrow$ **Score prediction** (same information!)
- **Both approaches** learn the same underlying probability geometry

Training Objectives: Different Equations, Same Optimization

Making the connection concrete through mathematical equivalence...

Score-based training objective

$$\mathcal{L}_{\text{score}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0)\|^2\right] \tag{70}$$

Goal: Learn to predict score functions directly

Diffusion training objective

$$\mathcal{L}_{\text{diffusion}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}\|^2\right] \tag{71}$$

Goal: Learn to predict noise that was added

These look completely different... but are they really?

Step 1: Substituting the Score Expression ➔

Recall our fundamental connection 🔑

$$\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}} \tag{72}$$

Substituting into the score objective 🧮

$$\mathcal{L}_{\text{score}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t|\mathbf{x}_0)\|^2 \right] \tag{73}$$

$$= \mathbb{E} \left[\left\| \mathbf{s}_\theta(\mathbf{x}_t, t) - \left(-\frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}} \right) \right\|^2 \right] \tag{74}$$

$$= \mathbb{E} \left[\left\| \mathbf{s}_\theta(\mathbf{x}_t, t) + \frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}} \right\|^2 \right] \tag{75}$$

Now we have score objective in terms of noise! ✔

Step 2: Using the Score-Noise Relationship

Our fundamental equivalence

$$\mathbf{s}_\theta(\mathbf{x}_t, t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} \tag{76}$$

Substituting this relationship

$$\mathcal{L}_{\text{score}} = \mathbb{E}\left[\left\|\mathbf{s}_\theta(\mathbf{x}_t, t) + \frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}}\right\|^2\right] \tag{77}$$

$$= \mathbb{E}\left[\left\|-\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} + \frac{\boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}}\right\|^2\right] \tag{78}$$

$$= \mathbb{E}\left[\left\|\frac{-\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}}\right\|^2\right] \tag{79}$$

Step 3: The Final Algebraic Magic

Factoring out the constant

$$\mathcal{L}_{\text{score}} = \mathbb{E}\left[\left\| \frac{-\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \boldsymbol{\epsilon}}{\sqrt{1-\bar{\alpha}_t}} \right\|^2\right] \tag{80}$$

$$= \mathbb{E}\left[\frac{1}{1-\bar{\alpha}_t} \left\| -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \boldsymbol{\epsilon} \right\|^2\right] \tag{81}$$

$$= \frac{1}{1-\bar{\alpha}_t} \mathbb{E}\left[\left\| \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon} \right\|^2\right] \tag{82}$$

The spectacular result

$$\boxed{\mathcal{L}_{\text{score}} = \frac{1}{1-\bar{\alpha}_t} \mathcal{L}_{\text{diffusion}}} \tag{83}$$

The two objectives are identical up to a time-dependent weighting factor!

Training diffusion models = Training score models! =

The Profound Implications

What this mathematical equivalence reveals...

Unified understanding

$$\mathcal{L}_{\text{score}} = \frac{1}{1 - \bar{\alpha}_t} \mathcal{L}_{\text{diffusion}} \tag{84}$$

This means:

- **Same learning objective:** Both minimize the same mathematical quantity
- **Different parameterizations:** Score vs. noise prediction are equivalent views
- **Same convergence:** Both approaches learn identical functions
- **Interchangeable models:** Can convert between representations

Three Perspectives, One Mathematical Truth

Different views of the same underlying object...

Score-based perspective

- **Focus:** Learn score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- **Sampling:** Use Langevin dynamics to follow probability landscape
- **Intuition:** Emphasizes **geometric structure** of probability
- **Advantage:** Direct connection to underlying probability geometry

Three Perspectives, One Mathematical Truth

Different views of the same underlying object...

Diffusion perspective

- **Focus:** Learn to reverse gradual noising process
- **Sampling:** Use ancestral sampling to step-by-step remove noise
- **Intuition:** Emphasizes **temporal dynamics** of generation
- **Advantage:** Natural way to think about multi-step generation

Three Perspectives, One Mathematical Truth

Different views of the same underlying object...

Denoising perspective

- **Focus:** Learn to predict and remove added noise
- **Sampling:** Use iterative denoising for generation
- **Intuition:** Emphasizes **signal processing** intuition
- **Advantage:** Builds on decades of denoising research

Next Session Preview: Generative Models for IR, NLP & RAG

Now that we understand the geometric foundations, let's explore cutting-edge research opportunities:

- How can we revolutionize information retrieval with generative models?
- What breakthrough opportunities exist in generative RAG architectures?
- How do we apply diffusion models to personalized knowledge systems?
- Which research directions offer the highest impact potential for your career?