Diffusion Models: Theory and Applications

Lecture 2: The Forward Diffusion Process: Learning to Destroy Data Systematically

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

June 9, 2025

Last Time: We Learned Why Diffusion Models Matter

Instead of trying to learn complicated mappings in one shot...

What if we learned many simple steps?

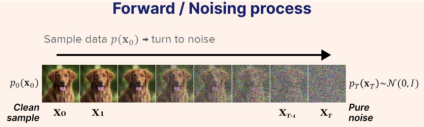
- ✔ GANs: Fast but unstable training
- ✔ VAEs: Stable but blurry samples
- **Diffusion**: Stable training + High quality samples

Today's Mission: The Art of Controlled Destruction

How do we destroy data systematically?

Why "destroy" first?

- Learning to reverse destruction
- Each step must be *learnable*
- Need mathematical guarantees



The Forward Process = Our Demolition Plan

Traditional Approach (GANs/VAEs):

Learn BOTH: Encode $x \to z$ AND Decode $z \to x$

Diffusion's Brilliant Twist:

FIX the forward process **LEARN** only the reverse!

- Forward: Simple, mathematical, predetermined noise injection
- Reverse: Complex, learned, neural network-based denoising
- Result: Transform intractable optimization $\to$ elegant learning problem!

The Controlled Demolition Metaphor

Think of a building demolition expert...

Random Destruction:

- Unpredictable collapse
- Impossible to reverse
- Chaotic process

Controlled Demolition:

- Precise explosion sequence
- Predictable structure collapse
- Reversible if we know the plan!

Forward Process:

Perfect Image

$\downarrow$ *Controlled noise injection*

Slightly noisy

$\downarrow$ *More controlled noise*

Pure static

The Mathematical Foundation: Markov Chains

Definitions

- **What's a Markov Chain?** A sequence where the future depends only on the present state
- **Markov Property:** $q(\text{future}|\text{present, past}) = q(\text{future}|\text{present})$

Non-Markov Example:

- Stock price depends on entire history
- Weather depends on long-term patterns
- Need to track everything!

Markov Example:

- Random walk: next step depends only on current position
- Game state: next move depends only on current board
- Much simpler to analyze!

The Mathematical Foundation: Markov Chains 🔗

Definitions

- **What's a Markov Chain?** A sequence where the future depends only on the present state
- **Markov Property:** $q(\text{future}|\text{present, past}) = q(\text{future}|\text{present})$

For diffusion

$$q(x_t|x_{t-1}, x_{t-2}, \ldots, x_0) = q(x_t|x_{t-1})$$

This means

To corrupt image at step t, we only need step $t-1$! *No need to remember entire corruption history*

The Forward Process: Our Specific Corruption Rule

The forward process q is our predetermined noise injection:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Breaking this down: 🔍

- $q(x_t|x_{t-1})$: Forward transition (data $\rightarrow$ noise)
- β_t: Noise schedule (how much noise to add)
- $\sqrt{1-\beta_t}$: Signal preservation factor
- $\beta_t I$: Fresh Gaussian noise variance

Remember

q is FIXED, we'll learn p (reverse process) later! *(Markov property makes both tractable)*

Why the Markov Property is Our Secret Weapon

Markov Property: Future depends only on present, not entire past

In our demolition metaphor:

- Next explosion depends only on current building state
- Don't need demolition history
- Each step is *memoryless*
- Makes reverse engineering tractable!

Mathematically:

- $p(x_t | x_{t-1}, x_{t-2}, \ldots, x_0) = p(x_t | x_{t-1})$
- Vastly simplifies analysis
- Enables closed-form solutions
- Each reverse step only needs local info

Without Markov property: Intractable dependencies

With Markov property: Elegant mathematical structure!

In-Class Exercise 1: Design Your Noise Schedule

You're the demolition expert! Design β_t values:

Scenario: Convert cat photo $\rightarrow$ static in 4 steps

- $\beta_1 = ?$ (First explosion - should be gentle!)
- $\beta_2 = ?$ (Second explosion)
- $\beta_3 = ?$ (Third explosion)
- $\beta_4 = ?$ (Final explosion - can be stronger!)

Constraints: $0 < \beta_t < 1$ and $\beta_1 \leq \beta_2 \leq \beta_3 \leq \beta_4$

Discuss with your neighbor:

What happens if β_1 is too large? Too small?

Understanding the Demolition Schedule

The noise schedule β_t controls destruction rate:

Signal Preservation: $\sqrt{1-\beta_t}$

- When β_t small $\to \sqrt{1-\beta_t} \approx 1$
- Most signal preserved
- Gentle corruption

Noise Injection: $\beta_t I$

- Fresh Gaussian noise
- Independent in each dimension
- Prevents correlated artifacts

Common Schedules:

Linear: $\beta_1 = 10^{-4}, \beta_T = 0.02$

- Simple and widely used
- Steady destruction rate

Cosine: Slower early, faster later

- Better balances easy/hard steps
- Often improves quality

Key: Gradual increase ensures each reverse step is learnable!

Noise Schedule Showdown: Linear vs. Cosine

Linear Schedule:

- $\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1)$
- Steady destruction rate
- Simple and interpretable
- Standard choice: $\beta_1 = 10^{-4}, \beta_T = 0.02$

Pros: Simple, well-tested
Cons: May waste computation on easy steps

Cosine Schedule:

- Slower corruption early
- Faster corruption later
- Better balances difficulty
- Improves sample quality

Pros: Often higher quality
Cons: Less intuitive, newer

Key Insight

Schedule choice affects both training efficiency and sample quality!

What Happens When Schedules Go Wrong?

Common failure modes and their fixes

Too aggressive early (β_1 too large):

- Signal destroyed too quickly
- Reverse steps become impossible
- **Fix:** Start gentler, $\beta_1 \leq 10^{-4}$

Too conservative throughout:

- Signal never fully destroyed
- x_T still contains structure
- **Fix:** Increase T or final β_T

Perfect schedule: Gradual increase, fully destroys signal by end

The Reparameterization Challenge

We have a problem with our forward process...

Our current formulation

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

The issue

- This requires *sampling* from a Gaussian distribution
- Sampling operations are not differentiable!
- No gradients $=$ no neural network training
- We need to make randomness "learnable"

The Solution

Transform: stochastic sampling $\rightarrow$ deterministic computation

Tool: The reparameterization trick!

The Reparameterization Trick: From VAEs to Diffusion

The general principle (borrowed from VAEs)

Instead of: Sampling $z \sim \mathcal{N}(\mu, \sigma^2)$ ✖

Do this: $z = \mu + \sigma \cdot \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$ ✔

Why this works

- **Same distribution:** Both produce $\mathcal{N}(\mu, \sigma^2)$
- **Isolated randomness:** All randomness is in ϵ
- **Differentiable path:** $\mu + \sigma \cdot \epsilon$ is just arithmetic!
- **Gradient flow:** Can backpropagate through this operation

Key insight: Move randomness outside the computation! →

In-Class Exercise 2: Practice the Trick

Quick Check: Apply Reparameterization

Transform this sampling operation:

$$z \sim \mathcal{N}(3x + 1, 4)$$

Your turn: Write this as $z = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$

Think about it

- What is μ?
- What is σ?
- Why does this preserve the same distribution?

Discuss with your neighbor for 2 minutes!

Exercise 2 Solution: Reparameterization in Action

The Solution ✔

Given: $z \sim \mathcal{N}(3x + 1, 4)$

Reparameterized form:

$$z = (3x + 1) + 2\epsilon$$

where $\epsilon \sim \mathcal{N}(0, 1)$

Breaking it down:

- **Mean (μ):** $3x + 1$
- **Standard deviation (σ):** $\sqrt{4} = 2$
- **Variance:** $\sigma^2 = 4$

Exercise 2 Solution: Reparameterization in Action

The Solution

Given: $z \sim \mathcal{N}(3x + 1, 4)$
Reparameterized form:

$$z = (3x + 1) + 2\epsilon$$

where $\epsilon \sim \mathcal{N}(0, 1)$

Why this preserves the distribution

- **Mean:** $\mathbb{E}[z] = \mathbb{E}[(3x + 1) + 2\epsilon] = (3x + 1) + 2 \cdot 0 = 3x + 1$
- **Variance:** $\text{Var}[z] = \text{Var}[2\epsilon] = 4\text{Var}[\epsilon] = 4 \cdot 1 = 4$
- **Same Gaussian family!** The randomness is just isolated in ϵ

Key insight: We can now differentiate through $(3x + 1) + 2\epsilon$!

Applying the Trick to Our Diffusion Step

Our Gaussian transition

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Identify the components

- **Mean:** $\mu = \sqrt{1-\beta_t}x_{t-1}$
- **Covariance:** $\Sigma = \beta_t I$

For multivariate case: $z = \mu + \Sigma^{1/2}\epsilon$

What's $\Sigma^{1/2}$ for $\beta_t I$?

$$\Sigma^{1/2} = (\beta_t I)^{1/2} = \sqrt{\beta_t} I$$

Why? $(\sqrt{\beta_t} I)(\sqrt{\beta_t} I) = \beta_t I^2 = \beta_t I$ ✔

The Magic Formula Revealed

Putting it all together

$$x_t = \mu + \Sigma^{1/2}\epsilon_{t-1} \tag{1}$$
$$= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}I \cdot \epsilon_{t-1} \tag{2}$$
$$= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \tag{3}$$

where $\epsilon_{t-1} \sim \mathcal{N}(0, I)$

The Revolutionary Formula

$$\boxed{x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}}$$

This changes everything!

From stochastic sampling → deterministic computation

In-Class Exercise 3: Coefficient Analysis

Understanding the Balance

Given: $x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$
Scenario: $\beta_t = 0.01$ (very small noise)

Calculate these coefficients:

- Signal preservation: $\sqrt{1-\beta_t} = ?$
- Noise injection: $\sqrt{\beta_t} = ?$
- Which is larger? Why does this make sense?

Quick Poll

What happens if $\beta_t = 0.5$?
A) More signal preserved
B) More noise added
C) Equal signal and noise

In-Class Exercise 3 Solution: Coefficient Analysis

Solution for $\beta_t = 0.01$ ✔

- **Signal preservation:** $\sqrt{1-\beta_t} = \sqrt{1-0.01} = \sqrt{0.99} \approx 0.995$
- **Noise injection:** $\sqrt{\beta_t} = \sqrt{0.01} = 0.1$
- **Which is larger?** Signal coefficient (0.995) >> Noise coefficient (0.1)

Why this makes sense ◎

Small β_t = gentle corruption:

- Nearly all signal preserved (99.5%)
- Very little noise added (10% of unit noise)
- Perfect for early timesteps where we want gradual destruction!

In-Class Exercise 3 Solution: Coefficient Analysis

Solution for $\beta_t = 0.01$ ✔

- **Signal preservation:** $\sqrt{1-\beta_t} = \sqrt{1-0.01} = \sqrt{0.99} \approx 0.995$
- **Noise injection:** $\sqrt{\beta_t} = \sqrt{0.01} = 0.1$
- **Which is larger?** Signal coefficient (0.995) >> Noise coefficient (0.1)

Quick Poll Answer: $\beta_t = 0.5$

Signal: $\sqrt{1-0.5} = \sqrt{0.5} \approx 0.707$
Noise: $\sqrt{0.5} \approx 0.707$
Answer: C) Equal signal and noise!
This would be a 50-50 balance point

Pattern: As β_t increases, signal $\downarrow$ and noise $\uparrow$

Why This Transformation Changes Everything

1. Preserves Statistical Properties

- Same mean: $\mathbb{E}[x_t] = \sqrt{1-\beta_t}x_{t-1}$
- Same variance: $\mathrm{Var}[x_t] = \beta_t I$
- Identical distribution to original formulation

Why This Transformation Changes Everything

2. Enables Gradient Flow

- Randomness isolated in ϵ_{t-1}
- Deterministic transformation is differentiable
- Neural networks can train through this operation!

Why This Transformation Changes Everything

3. Computational Efficiency

- Simple arithmetic instead of sampling algorithms
- Vectorizes perfectly on GPUs
- Predictable memory access patterns

The Elegant Variance Evolution

How does variance change as we add noise?

If x_{t-1} has variance σ_{t-1}^2, then:

Variance calculation

$$\text{Var}[x_t] = \text{Var}[\sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}] \quad (4)$$
$$= (1-\beta_t)\text{Var}[x_{t-1}] + \beta_t \text{Var}[\epsilon_{t-1}] \quad (5)$$
$$= (1-\beta_t)\sigma_{t-1}^2 + \beta_t \quad (6)$$

The beautiful insight

- Keep fraction $(1-\beta_t)$ of previous variance
- Add β_t units of fresh noise variance
- Controlled, predictable variance evolution!

Perfect balance: Preserve structure while adding chaos

In-Class Exercise 4: Variance Tracking

Trace the Variance

Starting point: x_0 has variance $\sigma_0^2 = 1.0$
Given: $\beta_1 = 0.1$ and $\beta_2 = 0.2$

Calculate step by step:

Using: $\text{Var}[x_t] = (1 - \beta_t)\sigma_{t-1}^2 + \beta_t$

Step 1: $\text{Var}[x_1] = ?$
Step 2: $\text{Var}[x_2] = ?$

Discussion Question

Pattern check: Is variance increasing or decreasing?
What would happen if we kept going to x_{100}?

Exercise 4 Solution: Variance Tracking

Step-by-Step Calculations ✔

Formula: $\text{Var}[x_t] = (1 - \beta_t)\sigma_{t-1}^2 + \beta_t$

Step 1: $\text{Var}[x_1] = (1 - 0.1) \times 1.0 + 0.1 = 0.9 + 0.1 = 1.0$
Step 2: $\text{Var}[x_2] = (1 - 0.2) \times 1.0 + 0.2 = 0.8 + 0.2 = 1.0$

The Surprising Result! !

Variance stays constant at 1.0!

- We lose $(1 - \beta_t)$ of previous variance
- But we add β_t units of fresh noise variance
- Perfect balance: $(1 - \beta_t) \times 1 + \beta_t = 1$

Discussion Answers ◎

Pattern: Variance stays constant (neither increasing nor decreasing!)
At x_{100}: Still variance = 1.0!
Why this is brilliant: Prevents variance explosion or collapse
Stable training: Neural network always sees same variance scale

Key insight: Diffusion preserves variance by design! ⚖

Watching the Image Dissolve: Step-by-Step

Let's trace a cat photo through corruption

Original Image (x_0): Perfect, structured cat photo

First Corruption (x_1)

$$x_1 = \sqrt{1 - \beta_1} x_0 + \sqrt{\beta_1} \epsilon_0$$

Barely perceptible grain, every detail still visible

Second Corruption (x_2)

$$x_2 = \sqrt{1 - \beta_2} x_1 + \sqrt{\beta_2} \epsilon_1$$

Cat still recognizable, but quality degraded

The pattern: Each step mixes previous state with fresh noise!

In- Exercise 4: Expand the Recursion

Let's build x_2 together!

Start with: $x_2 = \sqrt{1-\beta_2}x_1 + \sqrt{\beta_2}\epsilon_1$
We know: $x_1 = \sqrt{1-\beta_1}x_0 + \sqrt{\beta_1}\epsilon_0$

Your mission: Substitute and expand!

Step 1: Replace x_1 in the x_2 equation
Step 2: Distribute $\sqrt{1-\beta_2}$
Step 3: Identify the pattern

What do you notice?

- How many ϵ terms appear?
- What's the coefficient of x_0?
- **Predict:** What would x_3 look like?

The Beautiful Recursive Structure

Let's expand x_2 to see the full pattern

$$x_2 = \sqrt{1-\beta_2}x_1 + \sqrt{\beta_2}\epsilon_1 \tag{7}$$
$$= \sqrt{1-\beta_2}(\sqrt{1-\beta_1}x_0 + \sqrt{\beta_1}\epsilon_0) + \sqrt{\beta_2}\epsilon_1 \tag{8}$$
$$= \sqrt{(1-\beta_2)(1-\beta_1)}x_0 + \sqrt{(1-\beta_2)\beta_1}\epsilon_0 + \sqrt{\beta_2}\epsilon_1 \tag{9}$$

The emerging pattern

- x_t = weighted combination of original image + accumulated noise
- Original image coefficient shrinks over time
- Multiple independent noise terms accumulate
- Each ϵ_i contributes differently based on when it was added

This recursive structure sets up our next breakthrough...

Cleaning Up Our Notation

Let's introduce cleaner notation for what comes next:

$$\alpha_t = 1 - \beta_t \quad \text{(signal preservation factor)} \tag{10}$$

$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s \quad \text{(cumulative signal preservation)} \tag{11}$$

Why this notation helps

- α_t is more intuitive than writing $1 - \beta_t$ everywhere
- $\bar{\alpha}_t$ directly tells us: "How much original signal remains?"
- Makes the forward jump formula much cleaner
- Standard notation used in all diffusion papers

Key insight: $\bar{\alpha}_t$ decreases from 1 to 0 as t increases!

Instead of 1000 sequential steps...

Can we jump directly to any timestep? ⚡

Instead of 1000 sequential steps...

Can we jump directly to any timestep?

The answer: YES! Thanks to **GAUSSIAN ARITHMETIC MAGIC!**

The Mathematical Breakthrough: Forward Jumps!

Instead of 1000 sequential steps...

Can we jump directly to any timestep?

The answer: YES! Thanks to **GAUSSIAN ARITHMETIC MAGIC!**

The cornerstone insight: Multiple independent Gaussian noise terms combine beautifully:

$$\sqrt{a}\epsilon_1 + \sqrt{b}\epsilon_2 \sim \mathcal{N}(0, (a+b)I)$$

This simple property is THE foundation of DDPM practicality!

Without it: Diffusion models would be computationally impossible

The Forward Jump Formula: Pure Mathematical Beauty

The magical forward jump formula

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

where $\epsilon \sim \mathcal{N}(0, I)$

This means

From ANY clean image x_0, we can instantly compute the noisy version at ANY timestep t!

No need to simulate the entire chain!

In-Class Exercise 5: Forward Jump Analysis

Let's analyze the forward jump formula:

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

Understanding the coefficients:

- At $t = 0$: $\bar{\alpha}_0 = 1$, so $x_0 = 1 \cdot x_0 + 0 \cdot \epsilon$ ✔
- As $t \to \infty$: $\bar{\alpha}_t \to 0$, so $x_t \to 0 \cdot x_0 + 1 \cdot \epsilon = \epsilon$ ✔

Signal-to-Noise Ratio:

$$\text{SNR}_t = \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t}$$

Quick question: What happens to SNR as training progresses? ↓

In-Class Exercise 5 Solutions & Deep Insights

Let's work through the SNR analysis

Signal-to-Noise Ratio: $\text{SNR}_t = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t}$

What this tells us:

- High SNR (early timesteps): Easy to denoise, lots of signal
- Low SNR (late timesteps): Hard to denoise, mostly noise
- SNR decreases monotonically: $\text{SNR}_0 = \infty \to \text{SNR}_T \approx 0$

Training implications:

- Early steps: Network learns fine details
- Late steps: Network learns global structure
- Balanced training across difficulty levels

Key insight: SNR naturally curriculum learns easy $\to$ hard!

Homework Problem 1: Deriving the Forward Jump

Your Mission: Prove the Forward Jump Formula

We've seen that diffusion can "jump" directly to any timestep. Your job is to derive this mathematically for the first few steps.

Given Information:

- Single step: $x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1}$ where $\epsilon_{t-1} \sim \mathcal{N}(0, I)$
- Recall: $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
- **Gaussian Magic:** If $\epsilon_0, \epsilon_1 \sim \mathcal{N}(0, I)$ are independent, then $\sqrt{a}\epsilon_0 + \sqrt{b}\epsilon_1 = \sqrt{a + b}\epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$

Homework Problem 1: Deriving the Forward Jump (continued)

Part A: Two-Step Derivation (3 points)

Step 1: Write out x_1 and x_2 using the single-step formula
Step 2: Substitute x_1 into the expression for x_2
Step 3: Use Gaussian magic to combine the noise terms
Step 4: Show that $x_2 = \sqrt{\bar{\alpha}_2} x_0 + \sqrt{1 - \bar{\alpha}_2} \epsilon$

Part B: General Case (2 points)

Prove by induction: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ for any t

Homework Problem 1 (continued)

Part C: Understanding the Magic (3 points)

Explain in your own words:

1. Why can we combine $\sqrt{a}\epsilon_0 + \sqrt{b}\epsilon_1$ into $\sqrt{a+b}\epsilon$?
2. What does this property tell us about Gaussian distributions?
3. Why is this crucial for making diffusion models computationally practical?

Part D: Computational Advantage (2 points)

Compare the computational complexity:

- **Without forward jump:** To get x_{1000} from x_0
- **With forward jump:** To get x_{1000} from x_0

Explain why this difference matters for training diffusion models.

The Nightmare vs. The Dream vs.

WITHOUT Gaussian Closure:

- Track ALL noise terms $\epsilon_0, \epsilon_1, \ldots, \epsilon_{t-1}$
- Memory explodes: $O(t)$ per forward step
- For $T = 1000$: Need $1000\times$ more memory!
- Computing x_t needs $O(t)$ operations
- Training becomes $O(T^2)$ complexity
- Sequential dependencies kill parallelization
- **RESULT: Completely impractical!**

WITH Gaussian Closure:

- ALL noise collapses to $\sqrt{1-\bar{\alpha}_t}\epsilon$
- Constant $O(1)$ memory per sample
- Constant $O(1)$ computation time
- Perfect parallelization across timesteps!
- Random timestep sampling
- Scales to ANY number of steps
- **RESULT: Practical magic!**

The Complete Picture: From Order to Chaos

Forward jump reveals the beautiful structure:

- **At $t = 0$:** $\bar{\alpha}_0 = 1$, so x_0 is pure signal
- **As t increases:** $\bar{\alpha}_t$ decreases, signal fades, noise grows
- **At $t = T$:** $\bar{\alpha}_T \approx 0$, so $x_T \approx \epsilon$ is pure noise

This transformation is:

- ✔ **Deterministic:** Every data point follows same path
- ✔ **Universal:** All complex data $\rightarrow$ same noise distribution
- ✔ **Mathematically tractable:** Closed-form expressions
- ✔ **Gradual:** Each step involves small perturbation

The reverse process must learn to walk backwards along this path!

Training Data Generation: The Complete Algorithm </>

Here's exactly how we create unlimited training data:

Input: Clean image x_0, total timesteps T
Output: Training triple (x_t, t, ϵ)

1. Sample timestep: $t \sim \text{Uniform}\{1, 2, \ldots, T\}$
2. Sample noise: $\epsilon \sim \mathcal{N}(0, I)$
3. Compute: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$
4. Return: (x_t, t, ϵ) as training sample

Why this works brilliantly: ★

- Every image gives us T different training samples!
- Random sampling ensures balanced difficulty
- No sequential dependencies = perfect parallelization
- Ground truth ϵ is always available

Homework Problem 2: Implementing Diffusion Training

Your Mission: Build the Training Pipeline

You're tasked with implementing the core training data generation for a diffusion model. This problem tests your understanding of the forward process and training methodology.

Given Setup:

- Dataset of clean images $\{x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(N)}\}$
- Total timesteps $T = 1000$
- Forward jump formula: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
- Linear noise schedule: $\beta_1 = 0.0001, \beta_T = 0.02$

Homework Problem 2 (continued)

Part A: Algorithm Implementation (4 points)

Write pseudocode for the training data generation function:

- **Input:** Clean image x_0, precomputed $\bar{\alpha}_t$ values
- **Output:** Training triple (x_t, t, ϵ)
- **Include:** All random sampling steps and computations

Part B: Design Decisions (3 points each)

Answer these key questions with justifications:

1. Why sample timestep t randomly instead of sequentially during training?
2. What is the neural network learning to predict, and why this choice?
3. How does this training approach differ from VAE training?

Homework Problem 2 (continued)

Part C: Computational Analysis (4 points)

Memory and Speed Analysis:

- How many training samples can you generate from one image x_0?
- Compare memory usage: sequential sampling vs. forward jump approach
- Why is this approach "embarrassingly parallel"?

Part D: Implementation Details (3 points)

Practical Considerations:

1. How would you precompute and store the $\bar{\alpha}_t$ values?
2. What happens if $\bar{\alpha}_t$ becomes very small due to numerical precision?
3. Suggest one improvement to make the training more robust

Part E: Mini-Experiment (3 points)

Thought Experiment: Suppose you train two models:

- Model A: Always samples t from early timesteps (1-100)
- Model B: Uses uniform random sampling over all timesteps (1-1000)

Predict which model will perform better and explain why.

The Profound Implications: Why This Works So Well

Computational Revolution:

Training Efficiency:

- Random timestep sampling
- Single forward pass per step
- Perfect parallelization
- No sequential dependencies

Memory Efficiency:

- No intermediate storage
- Generate x_t on-demand
- Scales to any T

Mathematical Tractability:

- Analytical forward distributions
- Exact KL divergences
- Provable convergence
- Clean theoretical analysis

Implementation Simplicity:

- Complex sequential $\rightarrow$ simple parallel
- Hardware-friendly operations
- Predictable memory patterns

Without this property, diffusion models would be impossible!

Returning to our artist metaphor...

We now understand **exactly** how the masterpiece was destroyed:

- At timestep t, exactly $\sqrt{\bar{\alpha}_t}$ of original painting remains
- Mixed with exactly $\sqrt{1 - \bar{\alpha}_t}$ amount of random static
- The destruction follows a precise mathematical schedule
- Every image, no matter how complex, ends up as pure noise

The artist's job: Learn to remove static step by step!

Forward jump ensures unlimited training data:
Sample any corruption level instantly from any image!

Summary: The Forward Process Mastery

What we've learned today

- **Key Innovation:** Fix forward, learn reverse
- **Mathematical Foundation:** Markov chain with Gaussian transitions
- **Reparameterization:** Makes randomness differentiable
- **Forward Jumps:** $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
- **Computational Revolution:** $O(T^2) \rightarrow O(T)$ training

The stage is set

- Forward process: Systematic, predictable destruction ✔
- Training data: Unlimited noisy samples at any corruption level ✔
- Next challenge: Learn the reverse process!

From chaos, we will learn to create order...

Next Session Preview:

Mathematical Foundations

Now that you've seen HOW diffusion destroys data...
How do we LEARN to reverse this process?

We'll master the mathematical foundations:

- Why direct likelihood optimization fails
- Variational inference and tractable lower bounds
- The universal tools for ALL generative models
- Setting up the framework for learning!