Diffusion Models: Theory and Applications

Lecture 6: Conditional Generation - From Random to Controllable

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

June 11, 2025

Last Time: We Mastered Sampling

We learned to turn noise into data...

But what if we want CONTROL over what we generate?

- ✔ **DDPM sampling:** Stochastic reverse process with 1000 steps
- ✔ **DDIM sampling:** Deterministic acceleration with 20-50 steps
- ✔ **Trade-offs:** Quality vs. speed vs. diversity
- **Today:** Add steering wheels to our diffusion models!

How do we tell diffusion models WHAT to generate?

The Challenge:

- Random generation is not enough
- Need to guide the sampling process
- Want specific classes, styles, or attributes
- Must maintain high quality

Three Main Approaches:

- **Class-conditional:** Simple, direct conditioning
- **Classifier guidance:** External steering during sampling
- **Classifier-free guidance:** The modern breakthrough

From chaos to creation - with intention!

The Fundamental Problem: Unconditional Limitations ?

Why unconditional generation isn't enough...

The randomness dilemma

- You get random samples from the data distribution
- No control over what specific content is generated
- Need to generate many samples to find what you want
- Wastes computational resources and time

The Fundamental Problem: Unconditional Limitations ❓

What we really want 🎯

- **Class control:** "Generate a dog" vs. "Generate a cat"
- **Style control:** "In the style of Van Gogh"
- **Text control:** "A sunset over mountains"
- **Spatial control:** "Put the object here"

Solution: Replace $p(\mathbf{x})$ with $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{y}$ is our condition! 🔑

Reverse Process Recap: Our Mathematical Foundation

Remember how we parameterized the reverse process?

The learned reverse transition

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \tilde{\sigma}_t^2\mathbf{I})$$

Mean parameterization (from noise prediction)

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

Fixed variance

$$\tilde{\sigma}_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

Key insight: We learn the mean, variance comes from the noise schedule!

The Neural Network Behind Diffusion: U-Net

Before diving into conditioning, let's understand our core architecture...

What exactly is $\epsilon_\theta(\mathbf{x}_t, t)$?

- **Input:** Noisy image $\mathbf{x}_t$ + timestep t
- **Output:** Predicted noise $\hat{\epsilon}$
- **Architecture:** U-Net - the workhorse of diffusion models

Why U-Net?

- Originally designed for image segmentation
- Perfect for image-to-image tasks (noise $\rightarrow$ same size noise)
- Preserves spatial information while processing
- Natural fit for denoising operations

Understanding U-Net is key to understanding conditioning!

U-Net: The Backbone of Diffusion Models

Understanding the architecture that makes conditional generation possible

Why U-Net for diffusion?

- **Originally designed:** Biomedical image segmentation (2015)
- **Perfect fit:** Image-to-image transformation with spatial preservation
- **Key insight:** Denoising is just another image-to-image task!
- **Conditioning ready:** Architecture naturally supports additional inputs

The segmentation heritage

- U-Net solved: *"Given a noisy microscopy image, output a clean segmentation"*
- Diffusion adapted: *"Given a noisy image + timestep, output predicted noise"*
- **Same spatial reasoning, different task!**

From 30 biomedical images to billion-parameter generative models!

U-Net Architecture: The "U" Shape Design

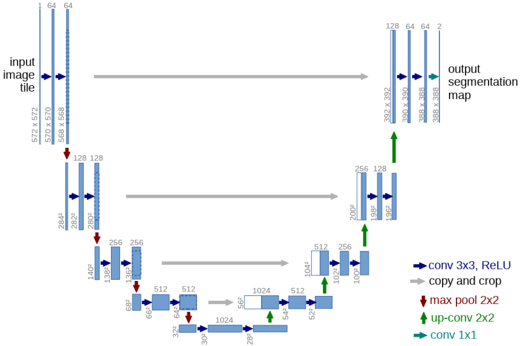

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Image taken from original paper: https://arxiv.org/abs/1505.04597v1

U-Net Architecture: The "U" Shape Design

↓ Contracting Path (Encoder):

- **Purpose:** Build semantic understanding
- **Operations:** Conv + Pool (downsample)
- **Effect:** $\downarrow$ spatial size, $\uparrow$ channels
- **Result:** Rich contextual features

↑ Expanding Path (Decoder):

- **Purpose:** Recover spatial precision
- **Operations:** Upsample + Conv
- **Effect:** $\uparrow$ spatial size, $\downarrow$ channels
- **Result:** Precise spatial output

Spatial resolution:

- $512\times512 \to 256\times256 \to 128\times128 \to 64\times64 \to 32\times32$

Channel progression:

- $64 \to 128 \to 256 \to 512 \to 1024$

The magic: Skip connections

High-resolution features from encoder are concatenated with decoder features $\to$ **Best of both worlds!**

Skip Connections: The Key Innovation

The fundamental problem

- **Downsampling:** Necessary for semantic understanding but loses spatial details
- **Upsampling:** Can't perfectly recover lost information
- **Trade-off:** Context vs. localization - can't have both?

Skip connections solve this!

- **Preserve details:** High-res features bypass the bottleneck
- **Combine information:** Concatenate rather than choose
- **Multi-scale features:** Each decoder level gets appropriate details
- **Gradient flow:** Better training with direct paths to early layers

Why this matters for diffusion

- **Noise prediction:** Needs both global context AND local spatial precision
- **Fine details:** Skip connections preserve edge and texture information
- **Conditioning:** Multiple injection points for different scales of control

U-Net in Diffusion: Key Adaptations

How diffusion models modified the original U-Net design...

Time embedding injection

- **Challenge:** Network needs to know the noise level
- **Solution:** Sinusoidal position encoding for timestep t
- **Injection:** Add time embeddings to features after each ResNet block
- **Effect:** Network learns noise-level-specific denoising

U-Net in Diffusion: Key Adaptations

How diffusion models modified the original U-Net design...

Attention mechanisms

- **Self-attention:** Added at multiple resolutions for global context
- **Cross-attention:** For conditioning on text, classes, etc.
- **Placement:** Typically at 16×16 and 8×8 spatial resolutions
- **Benefit:** Long-range dependencies and flexible conditioning

U-Net: Perfect for Conditioning

Multiple injection points

- **Embedding level:** Time + class embeddings combined
- **Feature level:** Cross-attention with text/image features
- **Skip connections:** Additional conditioning paths
- **Multi-scale:** Different conditions at different resolutions

Hierarchical conditioning

- **Global:** Text/class conditioning affects overall generation
- **Regional:** Spatial conditioning controls local areas
- **Fine-scale:** Detail conditioning affects textures and edges
- **Flexible:** Can combine multiple conditioning types seamlessly

U-Net's multi-scale nature makes it the perfect canvas for complex conditioning!

From Unconditional to Conditional U-Net

Unconditional U-Net

- **Input:** Noisy image $\mathbf{x}_t$ + timestep t
- **Output:** Predicted noise $\epsilon_\theta(\mathbf{x}_t, t)$
- **Training:** MSE on unconditional noise prediction

Conditional U-Net

- **Input:** Noisy image $\mathbf{x}_t$ + timestep t + condition $\mathbf{y}$
- **Output:** Predicted noise $\epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t)$
- **Training:** MSE on conditional noise prediction

Key question: How to inject $\mathbf{y}$?

- Where in the network should conditioning information go?
- How to combine image features with condition features?
- How to ensure the network learns to use the conditioning?

Approach 1: Class-Conditional Diffusion

The simplest form of conditioning: class labels

The basic idea

- Train on dataset with class labels: $\{(\mathbf{x}^{(i)}, y^{(i)})\}$
- Learn $p(\mathbf{x}|y)$ instead of $p(\mathbf{x})$
- Modify U-Net to take class information as input
- Generate specific classes during sampling

Mathematical formulation

Original objective: $\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2\right]$
Conditional objective: $\mathcal{L} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{y},\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t)\|^2\right]$

Same training procedure, just add the class label!

Class Conditioning: Architecture Modifications

How to inject class information into U-Net...

Step 1: Class embedding

- Convert class label y to dense embedding: $\mathbf{e}_y = \text{Embedding}(y)$
- Learnable embedding table (e.g., 1000 classes $\rightarrow$ 512-dim vectors)
- Same dimension as time embeddings for easy combination

Class Conditioning: Architecture Modifications

How to inject class information into U-Net...

Step 2: Embedding fusion

- Combine with time embedding: $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$
- Alternative: concatenation $\mathbf{e} = [\mathbf{e}_t; \mathbf{e}_y]$
- Pass through MLP for projection if needed

Class Conditioning: Architecture Modifications

How to inject class information into U-Net...

Step 3: Feature injection

- Inject combined embedding into each U-Net block
- Add to feature maps after normalization layers
- Scale and shift operations (similar to batch norm)

Class-Conditional Training Algorithm

Input: Dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}$, model ϵ_θ
while not converged **do**
  Sample batch $\{(\mathbf{x}, \mathbf{y})\}$ from dataset
  Sample timesteps $t \sim \text{Uniform}(1, T)$ for each sample
  Sample noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for each sample
  Compute noisy samples: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
  Embed class: $\mathbf{e}_y = \text{ClassEmbedding}(\mathbf{y})$
  Embed timestep: $\mathbf{e}_t = \text{TimeEmbedding}(t)$
  Combine embeddings: $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$
  Predict noise: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{e}, t)$
  Compute loss: $\mathcal{L} = \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2$
  Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
end while

Simple and effective for basic conditional generation! ✔

Class-Conditional Sampling ▶

Generating specific classes at inference time...

DDIM sampling with class conditioning

Input: Trained model ϵ_θ, desired class y^*, timesteps $\{\tau_1, \ldots, \tau_S\}$
Sample $\mathbf{x}_{\tau_S} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Embed class: $\mathbf{e}_y = \text{ClassEmbedding}(y^*)$
for $i = S, S-1, \ldots, 1$ **do**
  $t = \tau_i$, $s = \tau_{i-1}$
  Embed timestep: $\mathbf{e}_t = \text{TimeEmbedding}(t)$
  Combine: $\mathbf{e} = \mathbf{e}_t + \mathbf{e}_y$
  Predict noise: $\hat{\epsilon} = \epsilon_\theta(\mathbf{x}_t, \mathbf{e}, t)$
  Predict clean: $\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}$
  Update: $\mathbf{x}_s = \sqrt{\bar{\alpha}_s}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_s}\hat{\epsilon}$
end for
Return: $\mathbf{x}_0$

Same DDIM sampling, just with class embedding!

Class-Conditional: Pros and Cons

Advantages:

- **Simple:** Easy to implement and understand
- **Fast:** No computational overhead during sampling
- **Reliable:** Works consistently across datasets
- **Memory efficient:** Small additional parameters

Limitations:

- **Fixed:** Only works with predefined classes
- **Limited:** Can't combine or modify conditions
- **Retraining:** Need new model for new classes
- **Basic:** No fine-grained control within classes

Great for simple applications, but what about more complex control?

Enter classifier guidance... →

Approach 2: Classifier Guidance - The Steering Wheel

What if we could steer ANY pre-trained diffusion model?

The brilliant insight

- Take an existing unconditional model (already trained!)
- Use a separate classifier to guide the sampling process
- No need to retrain the diffusion model
- Can add multiple types of guidance

The mathematical foundation

From Bayes' rule: $p(\mathbf{x}_t|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x}_t)p(\mathbf{x}_t)$
Taking gradients: $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t)$

Conditional score = Unconditional score + Classifier gradient!

From Score Functions to Noise Predictions

Understanding the connection between $\nabla \log p(\mathbf{x})$ and ϵ...

Score function definition

The **score function** is the gradient of the log probability:

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

This tells us the direction of steepest increase in probability density.

Key insight

In diffusion models, we know that:

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

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. The score function is related to this noise!

Step 1: Start with the Marginal Distribution ▶

What we know from diffusion theory

From the forward process, we have:

$$p(\mathbf{x}_t|\mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

This means... →

- Mean: $\boldsymbol{\mu} = \sqrt{\bar{\alpha}_t}\mathbf{x}_0$
- Variance: $\sigma^2 = 1 - \bar{\alpha}_t$ (for each dimension)
- Covariance matrix: $\boldsymbol{\Sigma} = (1-\bar{\alpha}_t)\mathbf{I}$

Next step ⏩

We need to write out the explicit probability density function for this Gaussian distribution.

Step 1 (continued): Write the Gaussian PDF

Multivariate Gaussian formula

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

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

For our case: $\boldsymbol{\Sigma} = (1 - \bar{\alpha}_t)\mathbf{I}$

- $|\boldsymbol{\Sigma}| = (1 - \bar{\alpha}_t)^d$ (determinant of diagonal matrix)
- $\boldsymbol{\Sigma}^{-1} = \frac{1}{1-\bar{\alpha}_t}\mathbf{I}$ (inverse of diagonal matrix)
- $(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = \frac{\|\mathbf{x}-\boldsymbol{\mu}\|^2}{1-\bar{\alpha}_t}$

Step 1 (final): Complete PDF Expression ✓

Substituting our values

$$p(\mathbf{x}_t|\mathbf{x}_0) = \frac{1}{(2\pi)^{d/2}(1-\bar{\alpha}_t)^{d/2}} \exp\left(-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}\right)$$

Simplifying the normalization constant

$(2\pi)^{d/2}(1-\bar{\alpha}_t)^{d/2} = (2\pi(1-\bar{\alpha}_t))^{d/2}$

So:

$$\boxed{p(\mathbf{x}_t|\mathbf{x}_0) = \frac{1}{(2\pi(1-\bar{\alpha}_t))^{d/2}} \exp\left(-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}\right)}$$

Step 2: Take the Logarithm

Starting with our PDF

$$p(\mathbf{x}_t|\mathbf{x}_0) = \frac{1}{(2\pi(1-\bar{\alpha}_t))^{d/2}} \exp\left(-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}\right)$$

Apply logarithm properties

$$\log p(\mathbf{x}_t|\mathbf{x}_0) = \log\left[\frac{1}{(2\pi(1-\bar{\alpha}_t))^{d/2}}\right] + \log\left[\exp\left(-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}\right)\right]$$

Using $\log(1/x) = -\log(x)$ and $\log(\exp(x)) = x$

$$\boxed{\log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{d}{2}\log(2\pi(1-\bar{\alpha}_t)) - \frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}}$$

Step 3: Compute the Gradient - Setup

What we're computing

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0) = \nabla_{\mathbf{x}_t} \left[-\frac{d}{2} \log(2\pi(1-\bar{\alpha}_t)) - \frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)} \right]$$

Breaking it into parts

Term 1: $\nabla_{\mathbf{x}_t} \left[-\frac{d}{2} \log(2\pi(1-\bar{\alpha}_t)) \right]$
This term doesn't depend on $\mathbf{x}_t$, so its gradient is $\mathbf{0}$.
Term 2: $\nabla_{\mathbf{x}_t} \left[-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)} \right]$
This is where all the action happens!

Step 3: Compute the Gradient - The Chain Rule

Focus on the non-constant term

$$\nabla_{\mathbf{x}_t}\left[-\frac{\|\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0\|^2}{2(1-\bar{\alpha}_t)}\right]$$

Let $\mathbf{u} = \mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0$, so $\|\mathbf{u}\|^2 = \mathbf{u}^T\mathbf{u}$

$$\nabla_{\mathbf{x}_t}\left[-\frac{\|\mathbf{u}\|^2}{2(1-\bar{\alpha}_t)}\right] = -\frac{1}{2(1-\bar{\alpha}_t)}\nabla_{\mathbf{x}_t}[\mathbf{u}^T\mathbf{u}]$$

Using $\nabla_{\mathbf{x}}[\mathbf{x}^T\mathbf{A}\mathbf{x}] = 2\mathbf{A}\mathbf{x}$ when $\mathbf{A} = \mathbf{I}$

$$\nabla_{\mathbf{x}_t}[\mathbf{u}^T\mathbf{u}] = 2\mathbf{u} = 2(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)$$

Step 3: Final Gradient Result 🏁

Putting it all together

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{1}{2(1-\bar{\alpha}_t)} \cdot 2(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)$$

Simplifying

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{2(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)}{2(1-\bar{\alpha}_t)}$$

$$= -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t}$$

Final result! ⚠

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

Step 4: Recall the Forward Process

What we know from diffusion theory

The forward process gives us:

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

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is the noise added.

Our goal

We want to connect our gradient result:

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

to the noise term ϵ.

Key insight

The term $\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0$ in our gradient is related to the noise!

Step 4: Isolate the Noise Term

Starting with the forward process equation

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

Rearrange to isolate the noise part

Subtract $\sqrt{\bar{\alpha}_t}\mathbf{x}_0$ from both sides:

$$\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0 = \sqrt{1-\bar{\alpha}_t}\epsilon$$

What this tells us

The difference $\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0$ is exactly the **scaled noise** that was added during the forward process!

Step 5: Solve for the Noise

Starting with our isolated noise term

$$\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0 = \sqrt{1-\bar{\alpha}_t}\epsilon$$

Divide both sides by $\sqrt{1-\bar{\alpha}_t}$

$$\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}} = \frac{\sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{1-\bar{\alpha}_t}}$$

$$\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}} = \epsilon$$

Key result

$$\boxed{\epsilon = \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}}$$

Step 6: Substitute Back into the Gradient

Our gradient formula from Step 3

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

$$= -\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t} \times \frac{1}{\sqrt{1-\bar{\alpha}_t}} \tag{2}$$

$$= -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}} \tag{3}$$

Step 6: The Beautiful Final Result

The key relationship!

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

What this means intuitively

- The **score function** (gradient of log probability) is proportional to the **noise**
- The negative sign means: to increase probability, move **opposite** to the noise direction
- The factor $\frac{1}{\sqrt{1-\bar{\alpha}_t}}$ scales based on the noise level at time t

Score functions and noise predictions are fundamentally connected! →

The Problem: Conditional vs Unconditional ?

What we have vs what we need

What we derived: $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}}$

What we need: $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$ (unconditional score)

Why is this a problem? ⚠

- The conditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)$ assumes we know $\mathbf{x}_0$
- During generation, we don't know $\mathbf{x}_0$ - that's what we're trying to create!
- We need the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$ that works for any $\mathbf{x}_t$

The key question ?

How do we go from conditional to unconditional scores?

Step 1: The Marginalization Relationship

Connecting conditional and unconditional distributions

The unconditional distribution is the marginalization:

$$p(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0$$

Taking the gradient

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log \left[\int p(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0 \right]$$

The challenge

This integral is generally **intractable** to compute analytically! We need a different approach.

Step 2: The Denoising Score Matching Insight

Key theoretical result (Vincent 2010, Song and Ermon 2019)

Minimizing the denoising objective:

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

is equivalent to learning the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$.

What this means

If we train a neural network $s_\theta(\mathbf{x}_t, t)$ to match the conditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)$ across all possible $(\mathbf{x}_t, \mathbf{x}_0)$ pairs, it will automatically learn the unconditional score!

The magic

Training on conditional scores gives us unconditional scores for free!

Step 3: From Score Matching to Noise Prediction

Substituting our conditional score formula

We know: $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0) = -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}}$

So the denoising score matching objective becomes:

$$\mathcal{L} = \mathbb{E}_{p(\mathbf{x}_0)} \mathbb{E}_{p(\epsilon)} \left[\left\| -\frac{\epsilon}{\sqrt{1-\bar{\alpha}_t}} - s_\theta(\mathbf{x}_t, t) \right\|^2 \right]$$

Reparameterizing the score function

Let's write: $s_\theta(\mathbf{x}_t, t) = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$

Then:

$$\mathcal{L} = \mathbb{E}_{p(\mathbf{x}_0)} \mathbb{E}_{p(\epsilon)} \left[\left\| \frac{\epsilon - \epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} \right\|^2 \right]$$

Step 4: Simplifying the Loss Function

Factoring out constants

$$\mathcal{L} = \mathbb{E}_{p(\mathbf{x}_0)}\mathbb{E}_{p(\epsilon)}\left[\frac{1}{1-\bar{\alpha}_t}\left\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\right\|^2\right]$$

Since $\frac{1}{1-\bar{\alpha}_t}$ is just a constant (doesn't depend on ϵ or $\mathbf{x}_0$):

$$\mathcal{L} = \frac{1}{1-\bar{\alpha}_t}\mathbb{E}_{p(\mathbf{x}_0)}\mathbb{E}_{p(\epsilon)}\left[\left\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\right\|^2\right]$$

The key insight

Minimizing this is equivalent to minimizing:

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

This is exactly the DDPM training objective!

Step 5: The Beautiful Connection

What we've proven

Training $\epsilon_\theta(\mathbf{x}_t, t)$ to predict noise with the simple MSE loss:

$$\mathcal{L} = \mathbb{E}[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2]$$

is equivalent to learning the unconditional score function!

Therefore

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1 - \bar{\alpha}_t}}$$

Noise prediction = Score function learning!

Why this is profound

We never directly computed the intractable integral $\int p(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0$. Instead, the neural network learned it implicitly through the simple noise prediction task!

Classifier Guidance: Putting It All Together

We have:

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) \tag{4}$$

We know:

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} \tag{5}$$

and

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) = -\frac{\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t)}{\sqrt{1-\bar{\alpha}_t}} \tag{6}$$

Substituting Equations 5 and 6 in Equation 4 we get:

$$-\frac{\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t)}{\sqrt{1-\bar{\alpha}_t}} = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}} + \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)$$

Solving for $\tilde{\epsilon}$:

$$\boxed{\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t) = \epsilon_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t}\nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)}$$

Classifier Guidance: The Mathematical Magic

How to modify noise predictions using classifier gradients...

The score connection

Since $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) = -\frac{\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{1-\bar{\alpha}_t}}$, we get:

$$\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t) = \epsilon_\theta(\mathbf{x}_t, t) - \omega\sqrt{1-\bar{\alpha}_t}\nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)$$

What each term means

- $\epsilon_\theta(\mathbf{x}_t, t)$: Original unconditional noise prediction
- $\nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)$: Classifier gradient (direction toward class $\mathbf{y}$)
- ω: Guidance scale (how strong the steering is)
- $\sqrt{1-\bar{\alpha}_t}$: Noise level adjustment

Subtract classifier gradient to steer toward desired class! →

Classifier Guidance Sampling Algorithm

Input: Unconditional model ϵ_θ, classifier p_ϕ, target class $\mathbf{y}$, guidance scale ω
Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, T-1, \ldots, 1$ **do**
  // Step 1: Get unconditional noise prediction
  $\epsilon_u = \epsilon_\theta(\mathbf{x}_t, t)$
  // Step 2: Compute classifier guidance
  $\mathbf{g} = \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{y}|\mathbf{x}_t)$ ▷ Requires backprop!
  // Step 3: Apply guidance to noise prediction
  $\tilde{\epsilon} = \epsilon_u - \omega\sqrt{1-\bar{\alpha}_t}\mathbf{g}$
  // Step 4: Standard DDIM update with guided noise
  $\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\tilde{\epsilon}}{\sqrt{\bar{\alpha}_t}}$
  $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\tilde{\epsilon}$
end for
Return: $\mathbf{x}_0$

The Classifier Challenge

The noisy data problem

- Standard classifiers work on clean images
- During diffusion sampling, images are heavily corrupted with noise
- A clean-image classifier gives nonsensical gradients on noisy data
- Need a classifier that works across ALL noise levels

Solution: Noise-aware classifier training

- Train classifier on noisy images at all timesteps
- Input: $(\mathbf{x}_t, t)$ where $\mathbf{x}_t$ has noise level for timestep t
- Output: Class probabilities despite the noise
- Same noise schedule as the diffusion model

Training a robust classifier is non-trivial!

Classifier Guidance: The Guidance Scale Knob

The guidance scale ω controls the quality-diversity trade-off...

Low guidance ($\omega \to 0$):

- High sample diversity
- Natural-looking images
- Weak conditioning adherence
- Like unconditional generation

High guidance ($\omega \gg 1$):

- Strong conditioning adherence
- Clear class characteristics
- Lower sample diversity
- Risk of artifacts

Typical values

- $\omega = 1$: Theoretically correct Bayesian inference
- $\omega \in [2, 10]$: Common practical range
- Higher values: More conditioning strength but potential artifacts

Finding the sweet spot requires experimentation!

Classifier Guidance: Pros and Cons

Advantages:

- **Modular:** Works with any pre-trained model
- **Flexible:** Multiple classifiers for different attributes
- **No retraining:** Use existing diffusion models
- **Strong control:** High-quality conditional generation

Disadvantages:

- **Slow:** Extra classifier forward + backward pass
- **Complex:** Need robust noise-aware classifiers
- **Memory:** Store gradients for guidance
- **Brittle:** Sensitive to classifier quality

Powerful but complex - is there a simpler way?

Yes! Classifier-free guidance eliminates the classifier entirely...

Approach 3: Classifier-Free Guidance - The Breakthrough

What if we could get classifier guidance WITHOUT a classifier?

The revolutionary insight

From Bayes' rule: $\nabla_{\mathbf{x}_t} \log p(\mathbf{y}|\mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{y}) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$

The classifier gradient is just the difference between conditional and unconditional scores!

The elegant solution

- Train ONE model to do BOTH conditional and unconditional generation
- During training, randomly drop conditioning with probability p_{uncond}
- During sampling, compute both predictions and take their difference
- No separate classifier needed!

One model, unlimited control!

CFG Training: Teaching One Model Two Tasks

The conditioning dropout trick

Input: Dataset $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, dropout probability $p_{\text{uncond}} = 0.1$
while not converged **do**
  Sample batch $\{(\mathbf{x}, \mathbf{y})\}$ from training dataset
  Sample timesteps $t \sim \text{Uniform}(1, T)$ and noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Compute noisy samples: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
  for each sample $(\mathbf{x}_t, \mathbf{y}, t)$ in batch **do**
    if random() $< p_{\text{uncond}}$ **then** ▷ Randomly drop conditioning
      Set $\mathbf{y} = \emptyset$ ▷ Use null token
    end if
  end for
  Predict noise: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)$
  Compute loss: $\mathcal{L} = \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2$
  Update parameters
end while

Model learns to handle both real conditions and null conditions! ✔

The CFG Magic Formula

How to combine conditional and unconditional predictions...

The classifier-free guidance equation

$$\tilde{\epsilon}(\mathbf{x}_t, \mathbf{y}, t) = (1+\omega)\epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t) - \omega\epsilon_\theta(\mathbf{x}_t, \emptyset, t)$$

Intuitive understanding

Rewrite as: $\tilde{\epsilon} = \epsilon_{\text{uncond}} + (1+\omega)[\epsilon_{\text{cond}} - \epsilon_{\text{uncond}}]$

- Start with unconditional prediction
- Compute the "conditioning direction"
- Amplify this direction by $(1+\omega)$
- Move further in the conditioning direction!

Extrapolation beyond normal conditioning for stronger control!

CFG Sampling Algorithm ▶

Input: Model $\boldsymbol{\epsilon}_\theta$, condition $\mathbf{y}$, guidance scale ω, timesteps $\{\tau_1, \ldots, \tau_S\}$
Sample $\mathbf{x}_{\tau_S} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $i = S, S-1, \ldots, 1$ **do**
  $t = \tau_i$, $s = \tau_{i-1}$
  // Step 1: Get both predictions
  $\boldsymbol{\epsilon}_c = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)$ ▷ Conditional
  $\boldsymbol{\epsilon}_u = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset, t)$ ▷ Unconditional
  // Step 2: Apply classifier-free guidance
  $\hat{\boldsymbol{\epsilon}} = (1+\omega)\boldsymbol{\epsilon}_c - \omega\boldsymbol{\epsilon}_u$
  // Step 3: Standard DDIM update
  $\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\hat{\boldsymbol{\epsilon}}}{\sqrt{\bar{\alpha}_t}}$
  $\mathbf{x}_s = \sqrt{\bar{\alpha}_s}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_s}\hat{\boldsymbol{\epsilon}}$
end for
Return: $\mathbf{x}_0$

Two forward passes per step, but no classifier needed! ✔

CFG vs. Classifier Guidance: The Comparison

How do these two approaches stack up?

Classifier Guidance

- \+ Works with any pre-trained model
- \+ Modular approach
- – Requires noise-aware classifier
- – Slower (classifier + gradients)
- – Complex implementation

Classifier-Free Guidance

- \+ No separate classifier needed
- \+ Simpler implementation
- \+ Better text conditioning
- – Requires retraining
- – 2x forward passes

The verdict

CFG has become the dominant approach - powers Stable Diffusion, DALL-E, and most modern text-to-image models!

CFG enables the text-to-image revolution!

The Three Approaches: When to Use What?

Choosing the right conditioning approach for your application...

Class-conditional models

Use when: Simple categorical control, speed critical, limited compute **Examples:** Medical imaging, game assets, scientific visualization

Classifier guidance

Use when: Retrofitting existing models, research applications, multiple objectives
Examples: Adding control to pre-trained models, experimental setups

Classifier-free guidance

Use when: Complex conditioning, production systems, text-to-image **Examples:** Stable Diffusion, DALL-E, commercial creative tools

Modern default: CFG for most new applications! ★

Example: Stable Diffusion

Prompt

cartoon character of a person with a hoodie , in style of cytus and deemo, ork, gold chains, realistic anime cat, dripping black goo, lineage revolution style, thug life, cute anthropomorphic bunny, balrog, arknights, aliased, very buff, black and red and yellow paint, painting illustration collage style, character composition in vector with white background.

Summary: Conditional Generation Mastered

What we've learned today

- **Class-conditional:** Simple embedding-based conditioning
- **Classifier guidance:** External steering with gradients
- **Classifier-free guidance:** The modern standard
- **U-Net architecture:** The backbone enabling all conditioning
- **Text conditioning:** Cross-attention for language control

Key practical insights

- CFG with cross-attention dominates modern applications
- Proper conditioning dropout is crucial for CFG training
- Guidance scale controls the quality-diversity trade-off
- U-Net's hierarchical structure enables multi-scale conditioning

We can now build controllable, practical diffusion systems!

Next Session Preview:
Latent Diffusion & Efficiency

We'll explore how to make diffusion models practical and scalable:

- How does Stable Diffusion work in latent space?
- What are the key efficiency techniques for production deployment?
- How do we handle ultra-high resolution generation?
- What are the latest advances in fast sampling?

From beautiful theory to real-world systems
that serve millions of users!

Ready to scale diffusion to the real world?