

# MO433 - Unsupervised Learning

## Image Generation by Diffusion Models

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

# What are Diffusion Models?

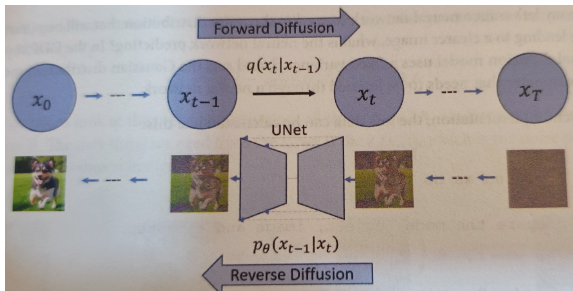
**Generative models** that learn to generate data by

- ▶ gradually adding noise to data (**forward process**) and
- ▶ gradually removing noise from noisy data (**reverse process**).

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

$$\text{Reverse: } p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta_t I)$$

$$\text{Objective: } \mathcal{L} = \mathbb{E}_{t, x_0, \epsilon} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$$



# Why Diffusion Models?

## Advantages:

- ▶ High quality generation with computationally intensive training.
- ▶ Stable training compared to GANs.
- ▶ Flexible conditioning (time steps, text).
- ▶ State-of-the-art results.

## Applications:

- ▶ Text-to-image.
- ▶ Image editing.
- ▶ Super-resolution.
- ▶ Video generation.

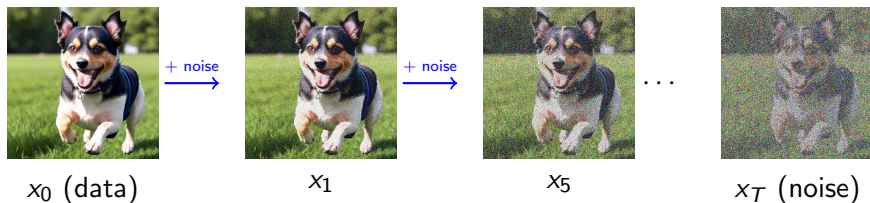
# Agenda

- ▶ The Forward Diffusion Process.
- ▶ Forward Process: Visual Results.
- ▶ The Reverse Diffusion Process.
- ▶ Reverse Process: Visual Results.
- ▶ Important implementation details.



# The Forward Diffusion Process

**Goal:** Gradually destroy data by adding Gaussian noise.



## Properties:

- ▶ Markov chain:  $x_t$  only depends on  $x_{t-1}$ .
- ▶ Fixed variance schedule:  $\beta_1, \beta_2, \dots, \beta_T$ .
- ▶ Eventually:  $x_T \sim \mathcal{N}(0, I)$  for sufficiently large  $T$  (e.g.,  $T = 1000$  for typical noise schedules).

# Forward Process: Incremental Step

**Add noise at each timestep  $t$ :**

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

**Equivalently (reparameterization trick):**

$$x_t = \sqrt{1 - \beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I).$$

**Understanding the parameters:**

- ▶  $\beta_t \in (0, 1)$ : **noise variance schedule** (how much noise to add).
- ▶  $\sqrt{1 - \beta_t}$ : signal scaling factor (shrinks the signal slightly).
- ▶  $\sqrt{\beta_t}$ : noise scaling factor.
- ▶ Typical:  $\beta_1 = 10^{-4}$  to  $\beta_T = 0.02$  (linear schedule).

## Notation: $\beta_t$ , $\alpha_t$ , and $\bar{\alpha}_t$

To simplify notation for direct sampling, we define:

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

**Then the forward process becomes:**

$$x_t = \sqrt{\alpha_t} \cdot x_{t-1} + \sqrt{1 - \alpha_t} \cdot \epsilon_t.$$

where

- ▶  $\beta_t$ : amount of noise added at step  $t$ .
- ▶  $\alpha_t = 1 - \beta_t$ : fraction of signal retained at step  $t$ .
- ▶  $\bar{\alpha}_t$ : **cumulative signal retention** after  $t$  steps.

**Insight:**  $\bar{\alpha}_t$  decreases from 1 to  $\approx 0$  as  $t$  goes from 0 to  $T$ .

# Direct Sampling: The closed-form solution

**Question:** Can we jump directly to  $x_t$  from  $x_0$  without computing all intermediate steps?

**Answer:** Yes! Using the reparameterization repeatedly:

$$\begin{aligned}x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1 - \alpha_t}\epsilon_{t-1} \\&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1 - \alpha_{t-1}}\epsilon_{t-2}) + \sqrt{1 - \alpha_t}\epsilon_{t-1} \\&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1 - \alpha_{t-1}) + (1 - \alpha_t)}\epsilon_{t-1} \\&\vdots \\&= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon\end{aligned}$$

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

# Why is the closed-form important?

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I)$$

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

## Iterative:

▶  $x_1 = \sqrt{\alpha_1}x_0 + \sqrt{1 - \alpha_1}\epsilon_1$

▶  $x_2 = \sqrt{\alpha_2}x_1 + \sqrt{1 - \alpha_2}\epsilon_2$

▶  $\vdots$

▶  $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1 - \alpha_t}\epsilon_t$

## Direct (closed-form):

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

Single step!

Requires  $t$  steps

## Benefits:

1. **Training efficiency:** Sample any timestep  $t$  directly.
2. **Mathematical elegance:** Clean formulation for analysis.
3. **Flexibility:** Easy to experiment with different noise schedules.

# Algorithm: Forward Diffusion (Iterative)

---

**Algorithm 1** Iterative Forward Diffusion

---

**Require:** Image  $x_0$ , noise schedule  $\{\beta_1, \dots, \beta_T\}$

**Ensure:** Noisy image  $x_T$

- 1:  $x \leftarrow x_0$
  - 2: **for**  $t = 1$  to  $T$  **do**
  - 3:   Sample  $\epsilon_t \sim \mathcal{N}(0, I)$
  - 4:    $x \leftarrow \sqrt{1 - \beta_t} \cdot x + \sqrt{\beta_t} \cdot \epsilon_t$
  - 5: **end for**
  - 6: **return**  $x_T = x$
- 

**Time complexity:**  $O(T)$  steps.

# Algorithm: Forward Diffusion (Direct)

---

## Algorithm 2 Direct Forward Diffusion

---

**Require:** Image  $x_0$ , timestep  $t$ , noise schedule  $\{\beta_1, \dots, \beta_T\}$

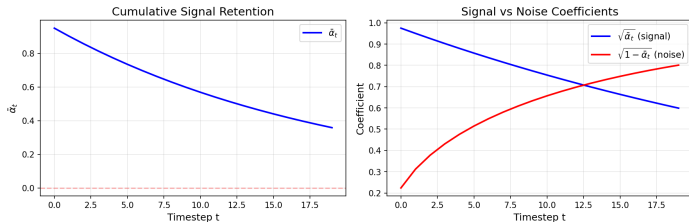
**Ensure:** Noisy image  $x_t$

- 1: Compute  $\bar{\alpha}_t \leftarrow \prod_{s=1}^t (1 - \beta_s)$
  - 2: Sample  $\epsilon \sim \mathcal{N}(0, I)$
  - 3:  $x_t \leftarrow \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$
  - 4: **return**  $x_t$
- 

**Time complexity:**  $O(1)$  for sampling (after precomputing  $\bar{\alpha}_t$ ).

**Key advantage:** Can jump to any timestep directly!

# Visualizing $\bar{\alpha}_t$ and Signal Decay



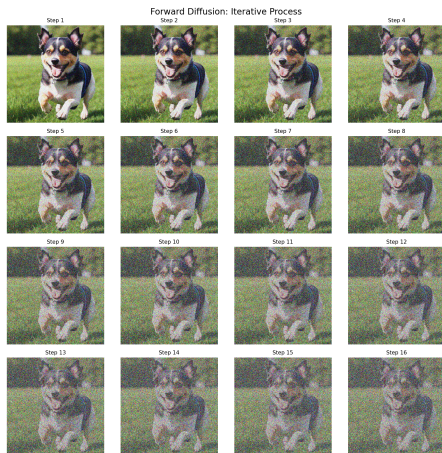
(Demo with  $T=20$  for illustration; standard practice uses  $T=1000$ ).

## Observations:

- ▶  $\bar{\alpha}_t$  decreases from 1 toward 0.
- ▶ Signal coefficient  $\sqrt{\bar{\alpha}_t}$  drops rapidly.
- ▶ Noise coefficient  $\sqrt{1 - \bar{\alpha}_t}$  increases.
- ▶ At  $t = T$ : with  $T = 1000$  and proper schedule,  $\bar{\alpha}_T \approx 0$  (pure noise).



# Forward Diffusion: Visual Results



**Observation:** Structure gradually disappears, leaving only noise at  $t = 1000$  (see [code1-forward-diffusion.py](#)).

# Forward Diffusion at Different Timesteps



- ▶  $t = 0$ : Original image ( $\bar{\alpha}_0 = 1.000$ ).
- ▶  $t = 5$ : Slight noise ( $\bar{\alpha}_5 \approx 0.774$ ).
- ▶  $t = 10$ : Moderate noise ( $\bar{\alpha}_{10} \approx 0.599$ ).
- ▶  $t = 19$ : Significant noise ( $\bar{\alpha}_{19} \approx 0.377$ ).

*Note: With more steps ( $T=1000$ ) and proper schedule,  $\bar{\alpha}_T \rightarrow 0$ .*

# Forward Process Endpoint



After  $T = 1000$  steps,  $x_T \approx \mathcal{N}(0, I)$  ( $\bar{\alpha}_T \approx 0.0001 \approx 0$ )

## Why does this matter?

- ▶ We know the distribution of  $x_T$  exactly!
- ▶ We can start sampling from  $x_T \sim \mathcal{N}(0, I)$ .
- ▶ If we can **reverse** this process, we can generate data!

# Forward Process: Summary

## What we learned:

### 1. Incremental diffusion:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1 - \alpha_t}\epsilon_t$$

### 2. Direct sampling (closed-form):

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

### 3. Notation:

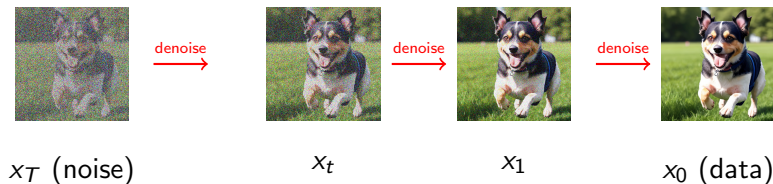
- ▶  $\beta_t$ : noise variance at step  $t$ .
- ▶  $\alpha_t = 1 - \beta_t$ : signal retention at step  $t$ .
- ▶  $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ : cumulative signal after  $t$  steps.

### 4. Endpoint: $x_T \sim \mathcal{N}(0, I)$ (known distribution!)

**Next:** How do we **reverse** this process to generate images?

# The Reverse Diffusion Process

**Goal:** Learn to reverse the forward process and generate data from noise.



## Challenge:

- ▶ Forward process  $q(x_t|x_{t-1})$  is known and fixed.
- ▶ Reverse process  $q(x_{t-1}|x_t)$  is **intractable** (requires knowing all data).
- ▶ Solution: Learn  $p_\theta(x_{t-1}|x_t)$  to approximate the reverse process!

# Reverse Process: Mathematical Formulation

**Learn a parameterized reverse process:**

$$p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \beta_t I)$$

where  $\mu_{\theta}(x_t, t)$  predicts **what  $x_{t-1}$  should look like**, and is learned by a neural network.

**Complete reverse process (joint probability  $p_{\theta}(x_{0:T})$ ):**

$$p_{\theta}(x_{0:T}) = p(x_T) \prod_{t=1}^T p_{\theta}(x_{t-1}|x_t)$$

$$p(x_T) = \mathcal{N}(x_T; 0, I) \quad (\text{start from pure noise})$$

## Insight

If we can learn  $p_{\theta}(x_{t-1}|x_t)$  to match  $q(x_{t-1}|x_t)$ , we can sample new data by starting from  $x_T \sim \mathcal{N}(0, I)$  and iteratively denoising!

# What should the network predict to obtain $\mu_\theta(x_t, t)$ ?

## 1. Predict the denoised image $\hat{x}_0$ :

$$\hat{x}_0 = f_\theta(x_t, t), \quad \text{then compute } \mu_\theta(x_t, t) \text{ from } \hat{x}_0$$

using the **posterior mean** formula.

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

## 2. Predict the mean directly:

$$\mu_\theta(x_t, t) = f_\theta(x_t, t).$$

## 3. Predict the noise $\epsilon$ (most stable, **preferable!**):

$$\epsilon_\theta(x_t, t) = f_\theta(x_t, t), \quad \text{then compute } \mu_\theta(x_t, t) \text{ from } \epsilon_\theta(x_t, t).$$

## Computing $\mu_\theta(x_t, t)$ from $\epsilon_\theta(x_t, t)$

**Step 1:** Recall forward process.

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

**Step 2:** Estimate  $x_0$  using predicted noise  $\epsilon_\theta(x_t, t)$ .

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

**Step 3:** Substitute  $\hat{x}_0$  into posterior mean formula and simplify.

$$\mu_\theta(x_t, t) = \tilde{\mu}_t(x_t, \hat{x}_0) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$$

### Insight

The network predicts the noise  $\epsilon_\theta$ , which lets us estimate  $x_0$ , which lets us use the posterior formula to compute  $\mu_\theta$ !



# Training Objective

## Notation:

- ▶  $\epsilon$ : actual noise added to the image (ground truth).
- ▶  $\epsilon_{\theta}(x_t, t)$ : network's prediction of that noise.

**Goal:** Train network to predict the actual noise  $\epsilon$ .

## Simplified loss function (DDPM):

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t \sim \text{Uniform}(1, T), x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0, I)} \left[ \left\| \underbrace{\epsilon}_{\text{true noise}} - \underbrace{\epsilon_{\theta}(x_t, t)}_{\text{predicted noise}} \right\|^2 \right]$$

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

**Interpretation:** Minimize the difference between true and predicted noise.

# Training Algorithm

---

## Algorithm 3 Training Diffusion Models (DDPM)

---

**Require:** Dataset of images, noise schedule  $\{\beta_1, \dots, \beta_T\}$

- 1: **repeat**
  - 2:   Sample batch  $\{x_0^{(i)}\}_{i=1}^B \sim q(x_0)$  {Batch of images}
  - 3:   Sample  $\{t^{(i)}\}_{i=1}^B \sim \text{Uniform}(\{1, \dots, T\})$  {Random timestep per image}
  - 4:   Sample  $\{\epsilon^{(i)}\}_{i=1}^B \sim \mathcal{N}(0, I)$  {Noise per image}
  - 5:   Compute  $x_t^{(i)} = \sqrt{\bar{\alpha}_{t^{(i)}}} x_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t^{(i)}}} \epsilon^{(i)}$  for all  $i$  {Vectorized}
  - 6:   Predict  $\epsilon_\theta(x_t^{(i)}, t^{(i)})$  for all  $i$  using UNet {Batch forward pass}
  - 7:   Compute loss:  $\mathcal{L} = \frac{1}{B} \sum_{i=1}^B \|\epsilon^{(i)} - \epsilon_\theta(x_t^{(i)}, t^{(i)})\|^2$
  - 8:   Update the UNet's parameters  $\theta$ :  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
  - 9: **until** converged
-

# Sampling Algorithm (DDPM)

---

## Algorithm 4 Sampling from Diffusion Models

---

**Require:** Trained network  $\epsilon_\theta$ , noise schedule  $\{\beta_1, \dots, \beta_T\}$

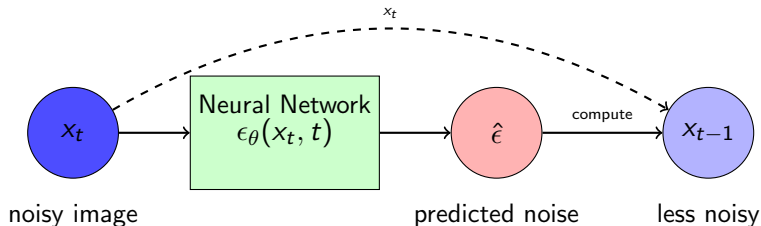
- 1: Sample  $x_T \sim \mathcal{N}(0, I)$  {Start from pure noise}
  - 2: **for**  $t = T, T-1, \dots, 1$  **do**
  - 3:   Predict noise:  $\hat{\epsilon} = \epsilon_\theta(x_t, t)$  {UNet forward pass}
  - 4:   Compute mean:  $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \hat{\epsilon} \right)$
  - 5:   Sample  $z \sim \mathcal{N}(0, I)$  if  $t > 1$ , else  $z = 0$  {Stochastic sampling}
  - 6:    $x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z$  {Sample from  $p_\theta(x_{t-1}|x_t)$ }
  - 7: **end for**
  - 8: **return**  $x_0$  {Generated image}
- 

where  $\sigma_t = \sqrt{\beta_t}$  or  $\sigma_t = \sqrt{\tilde{\beta}_t}$  with  $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t$

## Insight

Each iteration samples  $x_{t-1} \sim p_\theta(x_{t-1}|x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_t^2 I)$

## Reverse Process: Summary



### Denoising step:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t z$$

- ▶ Use current noisy image  $x_t$  and timestep  $t$  as input.
- ▶ Network predicts noise  $\epsilon_{\theta}(x_t, t)$ .
- ▶ Use predicted noise to compute  $x_{t-1}$ .

see [code2-reverse-diffusion.py](#).

# Reverse Diffusion: Visual Results (T=1000)

## Reverse Diffusion: Progressive Denoising (256×256, T=1000)

Step 0/1000  
t=999



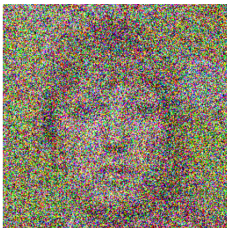
Step 200/1000  
t=799



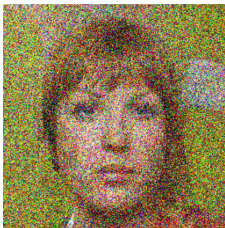
Step 400/1000  
t=599



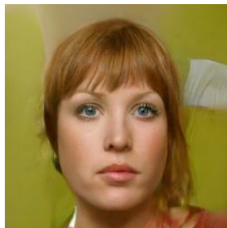
Step 600/1000  
t=399



Step 800/1000  
t=199

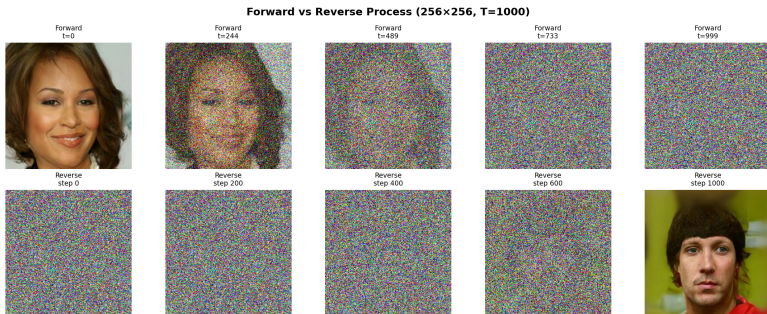


Step 999/1000  
t=0



**Observation:** Starting from pure noise, the model progressively reveals structure over 1000 denoising steps.

# Forward vs Reverse Comparison (T=1000)



**Key insight:** Forward process destroys structure, reverse process reconstructs it using the learned denoising network.

# Multiple Diverse Samples

**Multiple Samples from Reverse Diffusion (256×256, T=1000)**

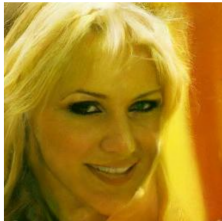
Sample 1



Sample 2



Sample 3



Sample 4



Sample 5



Sample 6



Starting from different random noise produces diverse outputs while maintaining quality (all generated with T=1000 steps).

# Effect of Number of Inference Steps



## Trade-off:

- ▶ Fewer steps (T=10, 50): Faster but lower quality.
- ▶ More steps (T=250, 1000): Slower but higher quality.
- ▶ Standard practice: T=1000 for training, T=50-250 for fast inference.



# Reverse Process: What we learned so far

1. **Goal:** Learn to reverse the forward diffusion process.

$$p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \sigma_t^2 I)$$

2. **Approach:** Train network to predict noise  $\epsilon_{\theta}(x_t, t)$ .

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

3. **Sampling:** Iteratively denoise from  $x_T \sim \mathcal{N}(0, I)$ .

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t z$$

4. **Architecture:** U-Net with timestep embedding.

# Reverse Process: What we learned so far

1. **Goal:** Learn to reverse the forward diffusion process.

$$p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \sigma_t^2 I)$$

2. **Approach:** Train network to predict noise  $\epsilon_{\theta}(x_t, t)$ .

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

3. **Sampling:** Iteratively denoise from  $x_T \sim \mathcal{N}(0, I)$ .

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t z$$

4. **Architecture:** U-Net with timestep embedding.

What are the important implementation details?

# UNet Architecture for Diffusion Models

**UNet Structure:**  $(x_t, t) \mapsto \epsilon_\theta(x_t, t)$

```
1 Input:  $x_t \in \mathbb{R}^{3 \times H \times W}$ 
2   ↓
3 Encoder (downsampling):
4    $h_1 \in \mathbb{R}^{C_1 \times H \times W} \rightarrow \text{skip}_1$ 
5    $h_2 \in \mathbb{R}^{C_2 \times H/2 \times W/2} \rightarrow \text{skip}_2$ 
6    $h_3 \in \mathbb{R}^{C_3 \times H/4 \times W/4} \rightarrow \text{skip}_3$ 
7    $h_4 \in \mathbb{R}^{C_4 \times H/8 \times W/8} \rightarrow \text{skip}_4$ 
8   ↓
9 Bottleneck:
10   $h_b \in \mathbb{R}^{C_{\max} \times H/16 \times W/16}$ 
11  ↓
12 Decoder (upsampling):
13   $d_4 \in \mathbb{R}^{C_4 \times H/8 \times W/8}$  [concatenate with skip4]
14   $d_3 \in \mathbb{R}^{C_3 \times H/4 \times W/4}$  [concatenate with skip3]
15   $d_2 \in \mathbb{R}^{C_2 \times H/2 \times W/2}$  [concatenate with skip2]
16   $d_1 \in \mathbb{R}^{C_1 \times H \times W}$  [concatenate with skip1]
17  ↓
18 Output:  $\epsilon_\theta \in \mathbb{R}^{3 \times H \times W}$ 
```

**Typical channels:**

$C_1 = 64, C_2 = 128, C_3 = 256, C_4 = 512, C_{\max} = 512.$

# UNet Architecture: Key Features

- ▶ Both encoder and decoder contain **residual blocks** (ResBlocks).
- ▶ Timestep  $t$  is encoded via **sinusoidal embeddings** and injected into each ResBlock.
- ▶ **Self-attention** is applied at resolutions  $\leq 32 \times 32$  (bottleneck and lower encoder/decoder levels), due to computational cost.
- ▶ **Skip connections** preserve spatial information from encoder to decoder.

# Timestep Encoding: From scalar to vector

The timestep  $t \in \{1, 2, \dots, T\}$  is just a scalar. How do we represent it in a way the network can use?

Convert  $t$  into a continuous vector  $p_t \in \mathbb{R}^d$  using

$$\text{PE}(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right)$$

$$\text{PE}(t, 2i + 1) = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

for  $i = 0, 1, \dots, d/2 - 1$  (typically  $d = 128$  or  $256$ ).

**Process through MLP:**

$$e_t = W_2 \cdot \text{SiLU}(W_1 p_t + b_1) + b_2$$

**Result:**  $e_t \in \mathbb{R}^{d'}$  (typically  $d' = 512$  or  $1024$ ).

# How timestep is combined with image features

**Goal:** Inject timestep information  $e_t$  into image features  $x \in \mathbb{R}^{C \times H \times W}$  (the input to a ResBlock).

**Two main approaches:**

1. **Adaptive Group Normalization (AdaGN)** - Most common in DDPM.
2. **Direct Addition** - Simpler alternative.

## Why condition on timestep?

The network needs to know **how much noise** is present to denoise appropriately.

- ▶ High  $t$  (lots of noise): Focus on global structure.
- ▶ Low  $t$  (little noise): Focus on fine details.

# Adaptive Group Normalization: Inside each ResBlock

**Step 1:** Apply GroupNorm, activation, and convolution.

$$h^{(0)} = \text{SiLU}(\text{GN}(\text{Conv}(x))) \in \mathbb{R}^{C \times H \times W}$$

**Step 2:** Generate scale  $\gamma$  and shift  $\beta$  from timestep.

$$[\gamma, \beta] = \text{Linear}(e_t)$$

where  $\gamma, \beta \in \mathbb{R}^C$

**Step 3:** Apply affine transformation (channel-wise).

$$h_{c,i,j}^{(1)} = \gamma_c \cdot h_{c,i,j}^{(0)} + \beta_c$$

## Intuition

After standard processing, the timestep **modulates** the features: scaling and shifting each channel based on the noise level.

# Direct Addition: Inside each ResBlock

**Step 1:** Apply GroupNorm, activation, and convolution.

$$h^{(0)} = \text{SiLU}(\text{GN}(\text{Conv}(x))) \in \mathbb{R}^{C \times H \times W}$$

**Step 2:** Project timestep to same channel dimension.

$$t_{\text{proj}} = \text{Linear}(e_t) \in \mathbb{R}^C$$

**Step 3:** Add timestep vector to all spatial locations.

$$h_{c,i,j}^{(1)} = h_{c,i,j}^{(0)} + t_{\text{proj},c}$$

## Intuition

After standard processing, simply **add** the timestep information to features at every spatial location. Simpler than AdaGN but still effective.



# Complete ResBlock with timestep conditioning

**ResBlock structure with input  $x$ :**

$$h^{(0)} = \text{SiLU}(\text{GN}(\text{Conv}(x)))$$

$$h^{(1)} = h^{(0)} + \text{Linear}(e_t) \quad [\text{timestep injection}]$$

$$h^{(2)} = \text{SiLU}(\text{GN}(\text{Conv}(h^{(1)})))$$

$$\text{output} = x + h^{(2)} \quad [\text{residual connection}]$$

## Key points:

- ▶ Timestep  $e_t$  is injected **once** in the middle of the block
- ▶ This happens at **every ResBlock** throughout the UNet
- ▶ The same timestep embedding  $e_t$  is shared across all blocks

## Result

The entire network is conditioned on the noise level, allowing it to adapt its processing at every layer!

# Self-Attention in UNet: Intuition

## What does it do?

Each spatial location  $(i, j)$  can **attend to** (look at) **all other locations**  $(i', j')$  in the feature map.

## Why is it important for diffusion?

- ▶ **Global coherence:** Ensures generated images are consistent across the entire spatial extent.
- ▶ **Compositional generation:** Coordinates different parts of the image (e.g., “cat on the left, dog on the right”).
- ▶ **Complex relationships:** Models dependencies beyond local neighborhoods.

# Self-Attention in UNet: Mechanism

**Step 1:** Reshape  $h \in \mathbb{R}^{C \times H \times W}$  to  $H \in \mathbb{R}^{(HW) \times C}$  (treat spatial locations as tokens).

**Step 2:** Compute queries, keys, values:

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V$$

**Step 3:** Apply scaled dot-product attention:

$$H' = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

**Step 4:** Reshape  $H'$  back to  $\mathbb{R}^{C \times H \times W}$  and add residual:

$$h' = \text{Reshape}(H') + h$$

# Common Noise Schedules

## 1. Linear Schedule (Original DDPM)

$$\beta_t = \beta_{\min} + \frac{t-1}{T-1}(\beta_{\max} - \beta_{\min})$$

Typical values:  $\beta_{\min} = 0.0001$ ,  $\beta_{\max} = 0.02$ ,  $T = 1000$

## 2. Cosine Schedule (Improved DDPM, Nichol & Dhariwal 2021)

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos^2 \left( \frac{t/T + s}{1+s} \cdot \frac{\pi}{2} \right)$$

where  $s = 0.008$  is a small offset.

## 3. Other schedules:

- ▶ Quadratic, sigmoid, learned schedules, etc.

# How Schedules Affect $\bar{\alpha}_t$

**Recall:** We visualized  $\bar{\alpha}_t$  decay in the forward process.

**Key insight:** The shape of this curve is determined by the noise schedule  $\{\beta_1, \dots, \beta_T\}$ !

## Linear Schedule:

$$\beta_t = \beta_{\min} + \frac{t-1}{T-1}(\beta_{\max} - \beta_{\min})$$

- ▶  $\beta_t$  increases linearly
- ▶  $\bar{\alpha}_t$  drops faster early
- ▶ Can make early timesteps "too easy"

## Cosine Schedule:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos^2(\dots)$$

- ▶ More gradual decline
- ▶ Better balanced difficulty
- ▶ Improved sample quality

# Why does the schedule matter?

## 1. Training stability:

- ▶ Too aggressive (large  $\beta_t$ ): Signal destroyed too quickly.
- ▶ Too conservative (small  $\beta_t$ ): Insufficient noise at  $t = T$ .

## 2. Perceptual quality:

- ▶ Early timesteps ( $t$  large): Generate global structure.
- ▶ Late timesteps ( $t$  small): Refine details.
- ▶ Good schedule ensures meaningful work at all stages.

## Which one to use

The cosine schedule is now standard because it provides more balanced difficulty across timesteps, leading to better sample quality.

# Schedule Design Principles

## Good noise schedule properties:

1. **Smooth progression:**  $\bar{\alpha}_t$  should decrease gradually.
2. **Endpoint constraints:**
  - ▶  $\bar{\alpha}_0 = 1$  (no noise at  $t = 0$ ).
  - ▶  $\bar{\alpha}_T \approx 0$  (pure noise at  $t = T$ ).
3. **Balanced difficulty:** Each timestep should contribute meaningfully to learning.
4. **Small steps:**  $\beta_t$  should be small enough that  $q(x_{t-1}|x_t, x_0)$  is well-approximated by a Gaussian.

Start with cosine schedule - it works well for most applications!

# What did we learn?

1. **Forward Process:** Gradually destroy data by adding Gaussian noise.

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

2. **Reverse Process:** Learn to denoise and generate data.

$$p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \beta_t I)$$

3. **Training:** Network learns to predict noise  $\epsilon_{\theta}(x_t, t)$ .

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

4. **Sampling:** Iteratively denoise from pure noise to data.

## Next Lecture

Stable Diffusion, text conditioning, and applications.