Diffusion Models — Theory

Lecture 21 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Explain the **forward process** in one sentence and write it down.
2. Derive the closed-form $q(x_t \mid x_0)$ and use it in code.
3. Write the **DDPM training loss** and describe what each term is doing.
4. Describe the **reverse process** step-by-step.
5. Understand **noise schedules** (linear vs cosine) and pick one for a task.
6. Connect **DDPM** to **score matching** via Langevin dynamics.
7. State why diffusion won over GANs and VAEs for image/video/audio.

Where we are

- **VAE** (L19) · probabilistic encoder-decoder; good structure, blurry samples.
- **GAN** (L20) · sharp samples, unstable training, mode collapse.

Today · **diffusion**. Sharp samples + stable training + tractable likelihood. SOTA since 2021 for image, video, audio, 3D generation.

> REFERENCE
>
> Today maps to **Prince Ch 18 (early)** + Ho et al. 2020 (DDPM) + Song & Ermon 2020 (score-based).

Four questions:

1. What's the **forward process**?
2. What's the **closed-form** for $q(x_t \mid x_0)$?
3. How do we **train** a diffusion model?
4. What's the connection to **score matching**?

PART 1

Forward & reverse · the big picture

Corrupt then learn to uncorrupt

The intuition in one sentence

KEY IDEA

Gradually turn an image into pure noise, then train a network to reverse that process one tiny step at a time.

At the end of training, you can start from random noise and reverse-diffuse it into a brand new image. Each small step is easy to learn; chained together they generate.

A physical analogy · ink in water

Drop a drop of ink into a glass of water. It stays concentrated, then slowly spreads, then uniformly tints the water.

Forward (easy)

Ink diffuses into water · we can describe this with a simple diffusion equation. Watching a drop blur is what "noise corrupts the signal" looks like in pictures.

Reverse (hard)

"Un-diffuse" the ink back into a drop. Physics says impossible (entropy only grows). But with data · we have many *examples* of initial states. A neural network can learn the reverse direction from those examples.

Diffusion models learn the miracle "reverse" that physics doesn't give you — but they learn it from data, not first principles.

Why this is better than GANs

GAN problems

- minimax: two networks playing a game
- mode collapse
- unstable training
- hyper-sensitive to hyperparameters

Diffusion advantages

- regression loss: MSE on predicted noise
- **one** network
- stable training
- default settings usually work

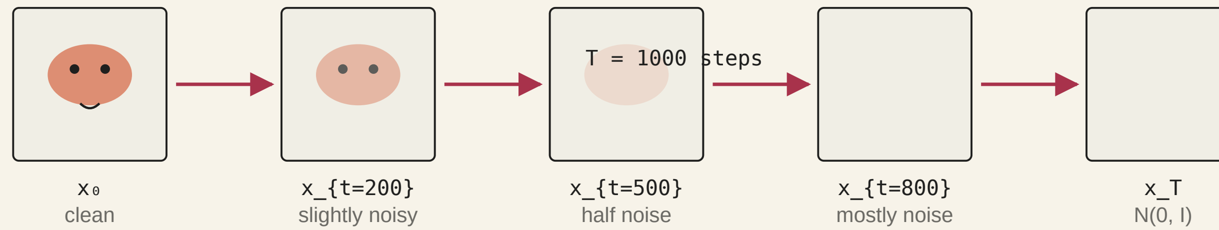
INTUITION

A GAN asks a network to hit a moving target (the discriminator's decision boundary). A diffusion model asks a network to match a *static* target (the noise that was added). Static targets are fundamentally easier to optimize.

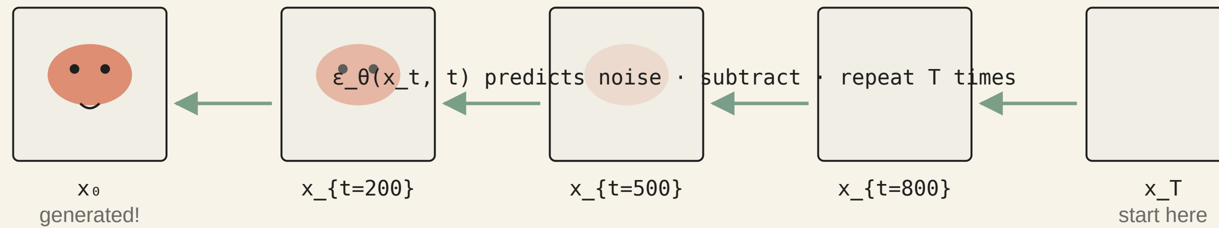
Forward corrupts · reverse reassembles

Diffusion — forward corrupts image with noise; reverse learns to denoise

FORWARD · ADD GAUSSIAN NOISE · Q(X_T | X_{T-1})



REVERSE · LEARN TO DENOISE · P_Θ(X_{T-1} | X_T)



Forward process is ***fixed*** *· reverse is* ***learned*** *· neural network ε_θ predicts the noise at each step.*

Forward + reverse · sequence view

Forward · destroy · Reverse · learn to rebuild

t = 0	**t = 250**	**t = 500**	**t = 750**	**t = 1000**
x_0 · clean data	$\bar{\alpha}$ = 0.82 (cosine)	$\bar{\alpha}$ = 0.50	$\bar{\alpha}$ = 0.18	x_T · N(0, I)

Forward · add noise · FIXED (closed form)

Reverse · predict noise · LEARNED (U-Net ε_θ)

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

jump to any t in one step · no need to iterate 500 forward passes during training.

IN PRACTICE

Interactive: slide t, see a 2D spiral dissolve into noise; press "Reverse animate" to watch it reassemble — diffusion-denoise.

PART 2

The forward process

Fixed · Markov · Gaussian

One forward step · slightly blurring a photo

INTUITION

Analogy. Take a sharp photo and make it one step blurrier:

1. Fade the original a tiny bit (e.g. to 99.5% opacity).
2. Add a faint layer of random static (Gaussian noise).

That's it. β_t controls both the fade amount and the static intensity.

Formally:

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

i.e. $x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$.

Worked numeric. $x_{t-1} = 100$, $\beta_t = 0.01$.

- Fade · $\sqrt{0.99} \approx 0.995$, so mean = 99.5.
- Noise · std $= \sqrt{0.01} = 0.1$.
- Update · $x_t = 0.995 \cdot 100 + 0.1 \cdot \epsilon$.

Over $T = 1000$ steps, the signal washes out into pure $\mathcal{N}(0, I)$. Forward is **not learned** — fixed dynamical

Why shrink *and* add noise

You might ask · why not just add noise? Why shrink the signal too?

KEY IDEA

Shrinking keeps the **total variance bounded**. If you only add noise, the variance grows without limit; x_T would be impossibly noisy and nothing-like-$\mathcal{N}(0, I)$.

DERIVATION

Variance check · if $x_{t-1} \sim \mathcal{N}(0, I)$, then $x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon$.

Variance of x_t = $(1-\beta_t) + \beta_t$ = **1**. Always.

This is why the forward process preserves unit variance — it's a **variance-preserving SDE**.

A step-by-step · small β = 0.01

Start with $x_0 = 2.0$. Apply 5 forward steps with $\beta_t = 0.01$:

DERIVATION

t	x_t	NOISE ADDED
0	2.00	—
1	1.99 · √0.99 + 0.1·ε = 1.97 + 0.08 = 2.05	ε = 0.8
2	2.02	ε = -0.4
3	2.00	ε = -0.2
4	2.03	ε = 0.3
5	2.06	ε = 0.2

After 5 steps the signal is barely disturbed. After 1000 steps with growing β, it becomes standard normal. The *accumulated* effect, not each step, turns signal into noise.

The closed form · compounding fades

After 1 step, signal is faded by $\sqrt{\alpha_1}$. After 2 steps · $\sqrt{\alpha_2\alpha_1}$. After t steps · $\sqrt{\bar{\alpha}_t} = \sqrt{\prod_{s=1}^{t} \alpha_s}$.

All the per-step noise additions also "pool together" into one big Gaussian:

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

Variance check · we designed the process so total variance stays 1. If $\bar{\alpha}_t$ is the fraction left from the signal, $1 - \bar{\alpha}_t$ is the fraction from noise. **Sums to 1.**

Worked numeric · jump from x_0 to x_{500}.
$x_0 = 2.0$, $\bar{\alpha}_{500} = 0.17$.

- Signal scale · $\sqrt{0.17} \approx 0.412$.
- Noise scale · $\sqrt{0.83} \approx 0.911$.
- Sample $\epsilon = -0.8$.
- $x_{500} = 0.412 \cdot 2.0 + 0.911 \cdot (-0.8) = 0.824 - 0.729 = \mathbf{0.095}$.

After 500 steps the original signal of 2.0 has nearly washed away. **No iteration needed during training.**

Why closed-form matters · training speed

Naive (iterative) forward

To get x_{500}, you'd apply 500 Gaussian steps sequentially. 500× forward passes per training example.

Batch of 128, 100k examples · ~10^9 operations just to make noise targets. Days on a single GPU.

Closed-form

One sample of ϵ, one scaled add. 500× faster per example.

Batch of 128 in **one step** · microseconds. Hours instead of days.

KEY IDEA

This closed-form is the single biggest practical advantage over continuous-time score-SDE approaches. Without it, DDPM training would cost 500× more.

optional · Closed-form · the derivation in 3 lines

DERIVATION

Start from one step: $x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \epsilon_t$.

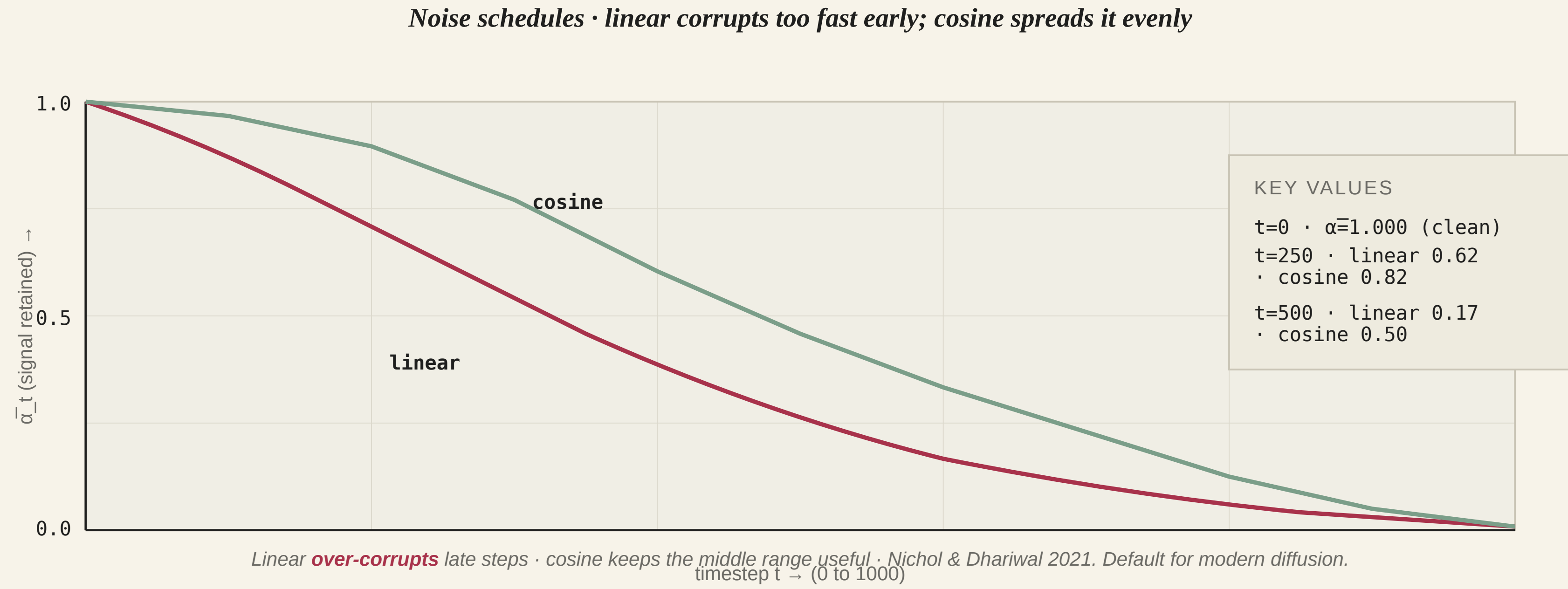
Unroll: $x_t = \sqrt{\alpha_t}[\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}} \epsilon_{t-1}] + \sqrt{1-\alpha_t} \epsilon_t$

Merge Gaussians (sum of independent Gaussians = Gaussian with summed variances):

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

where $\bar{\epsilon} \sim \mathcal{N}(0, I)$ replaces the t-step chain of independent ϵ's. Gaussian closed under convolution — this is the magic.

Noise schedules · in one chart



Linear **over-corrupts** late steps · cosine keeps the middle range useful · Nichol & Dhariwal 2021. Default for modern diffusion.

Noise schedules · the numbers

t	LINEAR Ā_T	COSINE Ā_T	WHAT'S LEFT
0	1.00	1.00	clean signal
250	0.62	0.82	linear: 38% destroyed · cosine: 18%
500	0.17	0.50	linear: already mostly gone
750	0.02	0.18	linear: essentially pure noise
1000	0.00	0.00	both: N(0, I)

INTUITION

Linear schedule wastes computation on steps near T where everything is noise anyway. Cosine keeps the middle range useful — the middle is where the model actually learns.

Noise schedules · linear vs cosine

Two common schedules:

Linear (original DDPM) · β_t grows linearly from 10^{-4} to 0.02 over $T = 1000$ steps.

Cosine (Nichol & Dhariwal 2021) · $\bar{\alpha}_t = \cos^2\left(\frac{t/T+s}{1+s} \cdot \frac{\pi}{2}\right)$ — smoother, better for smaller T.

INTUITION

Cosine schedule adds noise more gradually at the start and faster at the end. Better quality at fewer diffusion steps. Used in most modern diffusion models.

Picking T · the hyperparameter most people ignore

T	BEHAVIOR
50	too coarse; each step must learn a big jump; sample quality hurts
200	works but poor quality at the extremes
1000	**default**; great quality with cosine schedule
4000	slight quality gain; 4× inference cost; rarely worth it

IN PRACTICE

DDPM (Ho 2020) used T=1000 with linear schedule. Nichol & Dhariwal 2021 showed cosine + T=4000 gave marginal gains; T=1000+cosine is today's sweet spot.

PART 3

Training objective

Predict the noise

The reverse process · parameterized

We want $p_\theta(x_{t-1} \mid x_t)$ — learn to denoise.

Ho et al. 2020 showed that if $q(x_{t-1} \mid x_t, x_0)$ is Gaussian (it is), the optimal reverse process is also Gaussian. So parameterize:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Further: parameterize to predict the **noise** ϵ rather than the mean directly. Simpler, better signal.

Why predict ϵ instead of the mean?

Given $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$, there are three equivalent prediction targets:

DERIVATION

- Predict x_0 · known as "x0-prediction" or "v-prediction variant"
- Predict $\mu_\theta(x_t, t)$ · the mean of the reverse distribution
- Predict ϵ · the noise that was added

Ho et al. 2020 showed **ϵ-prediction gives the best sample quality**. Intuition · noise is unit-variance and dimension-independent; the network doesn't need to learn the scale of the signal.

Modern models (SDXL, Imagen) often use "v-prediction" — a weighted combination that's more numerically stable at small t.

Training · the noise-guessing game

KEY IDEA

How do we teach the network to "un-blur"?

Take a clean image. Add a **known amount of random noise** ϵ. Show the noisy result to the network. Ask it: *"What noise did I just add?"*

The better it gets at guessing the noise, the better it is at **denoising** · because subtracting the predicted noise gets us back to a cleaner image.

That's the entire training objective · MSE between predicted noise and the true noise added.

DDPM loss · surprisingly simple

DERIVATION

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t,x_0,\epsilon}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}\,x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon,\ t\right)\right\|^2\right]$$

In plain words:

1. Sample a clean image x_0 from the dataset.
2. Sample a timestep $t \in \{1, \ldots, T\}$.
3. Sample Gaussian noise ϵ.
4. Compute x_t (closed form).
5. Ask the network to predict ϵ from (x_t, t).
6. MSE loss on the prediction.

That's it. Much simpler than GAN minimax or VAE ELBO.

Worked example · one training step

DERIVATION

Suppose $x_0 = [2.0, 1.0]$ (a 2D data point), $t = 500$, $\bar{\alpha}_{500} = 0.5$. Sample $\epsilon = [0.3, -0.2]$.

1. $x_{500} = \sqrt{0.5} \cdot [2.0, 1.0] + \sqrt{0.5} \cdot [0.3, -0.2] = [1.414, 0.707] + [0.212, -0.141]$ = $[1.626, 0.566]$
2. Feed $(x_{500}, t = 500)$ to the network. Prediction · $\hat{\epsilon} = [0.25, -0.15]$
3. Loss · $||\hat{\epsilon} - \epsilon||^2 = (0.25 - 0.3)^2 + (-0.15 + 0.2)^2 = 0.005$
4. Backprop through $\hat{\epsilon}$ to update the network.

Single example, single t. Sum the loss over a batch and all is ready. No adversarial game, no multiple networks, no cross-entropy.

DDPM in PyTorch · 30 lines

```python
def ddpm_loss(model, x0, T=1000):
    B = x0.size(0)
    t = torch.randint(0, T, (B,), device=x0.device)
    noise = torch.randn_like(x0)

    alpha_bar = alpha_bar_schedule[t].view(B, 1, 1, 1)
    x_t = alpha_bar.sqrt() * x0 + (1 - alpha_bar).sqrt() * noise

    pred_noise = model(x_t, t)                         # network input: noisy img + t
    return F.mse_loss(pred_noise, noise)

def sample(model, shape, T=1000):
    x = torch.randn(shape)                              # start from N(0, I)
    for t in reversed(range(T)):
        alpha_t     = alpha_schedule[t]
        alpha_bar_t = alpha_bar_schedule[t]
        predicted   = model(x, torch.tensor([t]))
        mean = (x - (1 - alpha_t) / (1 - alpha_bar_t).sqrt() * predicted) / alpha_t.sqrt()
        if t > 0:
            x = mean + alpha_t.sqrt() * torch.randn_like(x)
        else:
            x = mean
    return x
```

The network architecture is a **U-Net** (L9) with time-step conditioning injected into each block.

Reverse step · denoise then re-noise a little

To go from x_t to x_{t-1}:

1. **Denoise.** Predict ϵ, subtract a scaled version from x_t → estimate of clean signal.
2. **Re-noise a little.** Add a small fresh noise so the chain stays stochastic.

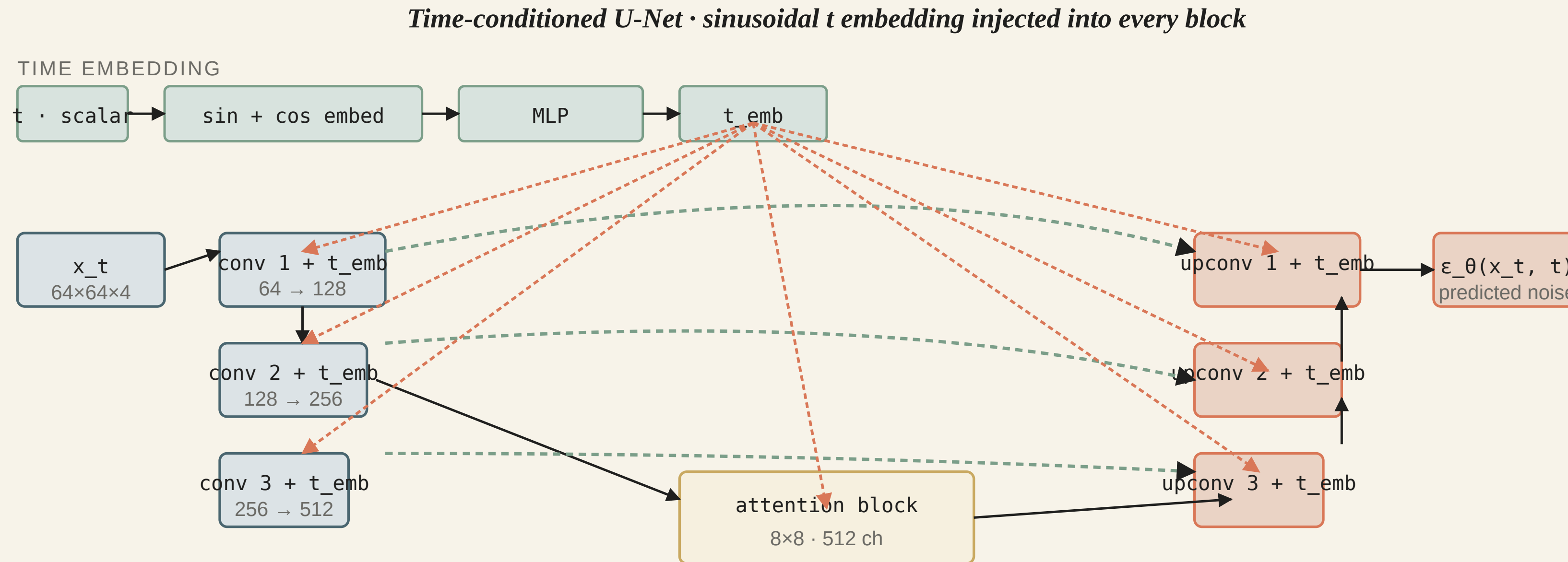
$$x_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(x_t, t)\right)}_{\text{denoised mean}} + \underbrace{\sigma_t\, z}_{\text{small fresh noise}}$$

Worked numeric (1D). $x_{100} = 1.5$. Schedule · $\alpha_{100} = 0.99$, $\bar{\alpha}_{100} = 0.8$. Network predicts $\epsilon_\theta = 0.6$.

- $1/\sqrt{0.99} \approx 1.005$
- $(1-0.99)/\sqrt{1-0.8} = 0.01/\sqrt{0.2} \approx 0.0224$
- Mean · $1.005 \cdot (1.5 - 0.0224 \cdot 0.6) = 1.005 \cdot 1.4866 \approx \mathbf{1.494}$
  Add noise · $\sigma_{100} = 0.1$, $z = -0.3$ → noise term = -0.03.
  $x_{99} = 1.494 - 0.03 = \mathbf{1.464}$.

We took one small step from noisier (1.5) to slightly cleaner (1.464). At $t = 1$, drop the noise term — final step is deterministic.

Network architecture · in one picture



Same U-Net **learns to denoise at every noise level**. Time embedding tells each block "how noisy is this?".

Network architecture · U-Net with time

A diffusion model's $\epsilon_\theta(x_t, t)$ is typically a U-Net:

- Encoder downsamples, decoder upsamples.
- Skip connections between matching resolutions (from L9).
- **Time embedding** · t becomes a sinusoidal vector, projected, and added into every block.
- **Attention** at low spatial resolutions (globally mix features).

IN PRACTICE

For 512×512 images · ~1B param U-Net; ~50 steps of sampling; ~5 seconds on a single GPU. Stable Diffusion's architecture is a direct descendant.

Sinusoidal time embedding

The time t is an integer $\in \{1, \ldots, T\}$. Represent it as a dense vector using the same positional encoding from L13:

```
def timestep_embedding(t, d):
    half = d // 2
    freqs = torch.exp(
        -math.log(10000) * torch.arange(half).float() / half
    )
    args = t.float()[:, None] * freqs[None, :]
    return torch.cat([args.cos(), args.sin()], dim=-1)
```

INTUITION

The same reason as in Transformers (L13) · sinusoidal basis gives multi-scale time representation that the network can read at any scale. Learned embeddings work too; sinusoidal is more robust across training-time changes in T.

Time conditioning · inject at every block

```
class TimestepBlock(nn.Module):
    def forward(self, x, t_emb):
        # x: image features. t_emb: timestep embedding
        h = self.norm1(x)
        h = self.conv1(F.silu(h))
        # project time and add as bias (broadcast over spatial dims)
        h = h + self.time_mlp(t_emb)[:, :, None, None]
        h = self.conv2(F.silu(self.norm2(h)))
        return x + h
```

Each U-Net residual block receives the time embedding and adds it to the channel dimension. The **same network weights** handle all timesteps — time is just another input, not a different model per step.

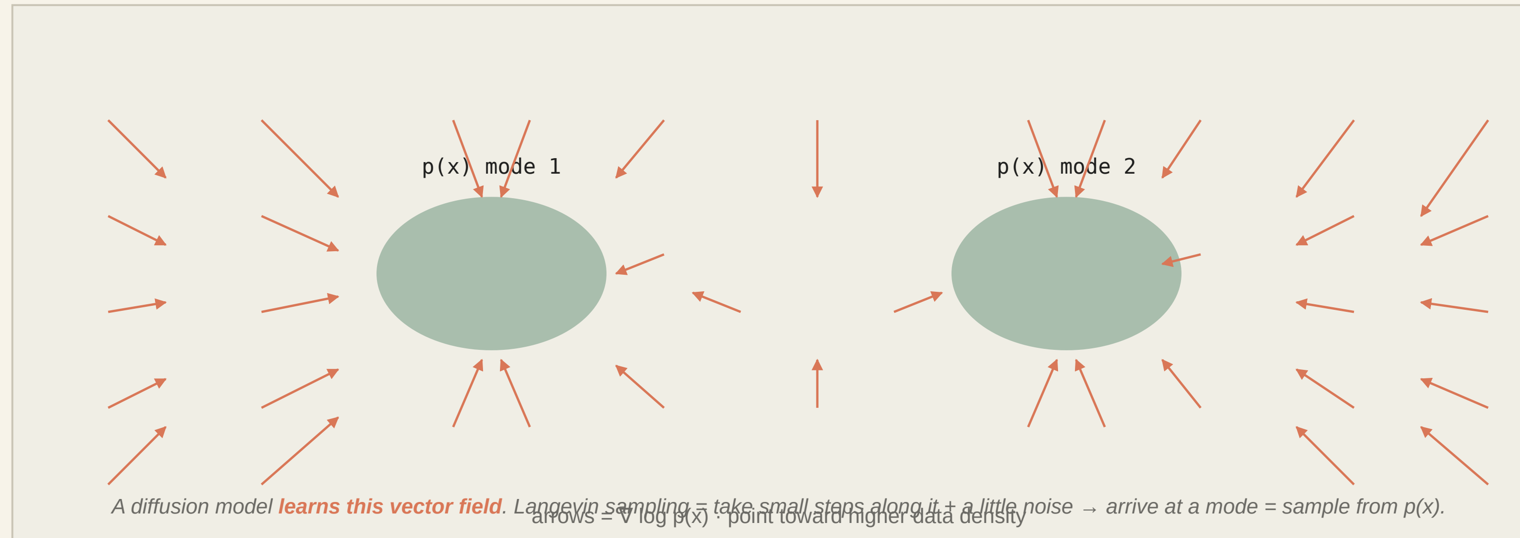
PART 4

Connection to score matching

Same thing, different derivation

The score field in one picture

Score field · ∇_x log p(x) points toward high-density regions (the data)



*A diffusion model **learns this vector field**. Langevin sampling = take small steps along it + a little noise → arrive at a mode = sample from p(x).*

Score · the "uphill arrows" view

KEY IDEA

Pause and look at this from a different angle. Imagine our data points sit at the **bottom of valleys** in a landscape.

The **score** is an arrow at every point in space that points in the steepest **uphill** direction.

If we can learn this field of "uphill" arrows, we can just **follow them backwards** to always go downhill toward valleys (i.e., toward real data).

That's score-based generation. Mathematically equivalent to DDPM · just a different lens. Picking either lens is fine; many find score-based more intuitive (gradients pointing toward data).

Score · the mountain-range analogy

INTUITION

Imagine probability is a landscape. Real data points sit in **deep valleys**; noise sits on **high flat plains**.

The **score** is an arrow at every point pointing **uphill** — toward higher density.

If we learn this field of "uphill" arrows, we can generate data · start on a high plain (noise) and walk **toward the arrows** until we land in a valley (real data).

The score function · math

Define $s(x) = \nabla_x \log p(x)$ — gradient of log density.

- $p(x)$ · density (high for real data, low elsewhere).
- $\log$ · just makes math nice (peaks of p = peaks of $\log p$).
- ∇_x · vector pointing in the direction of steepest ascent.

If we have s, we can sample with **Langevin dynamics**:

$$x_{k+1} = x_k + \eta\, s(x_k) + \sqrt{2\eta}\,\xi, \quad \xi \sim \mathcal{N}(0, I)$$

A small step *toward* high-density regions, plus a bit of noise to keep exploring. Looks just like the reverse diffusion step · *follow a learned signal + add a little noise*.

Score vs density · why use the score?

Density $p(x)$

- Must be non-negative.
- Must integrate to 1.
- Intractable normalizing constant for complex distributions.

Hard to model with a neural network.

Score $\nabla_x \log p(x)$

- Any vector field.
- Normalizer disappears: $\nabla_x \log(p \cdot Z) = \nabla_x \log p$.
- Easy to model with a neural network.

Parametrize the *derivative*, not the function itself.

Samples are what we want anyway.

KEY IDEA

Modeling the score sidesteps the normalizer problem — and the score is exactly what you need to run Langevin sampling.

Diffusion ≈ score matching · derivation

The noisy distribution is Gaussian:
$x_t \sim \mathcal{N}(\mu = \sqrt{\bar{\alpha}_t}\, x_0,\ \sigma^2 = 1 - \bar{\alpha}_t)$

Log density (up to const):
$\log q(x_t \mid x_0) = -\frac{1}{2\sigma^2}(x_t - \mu)^2 + C$

Differentiate w.r.t. x_t:
$\nabla_{x_t} \log q = -\frac{1}{1 - \bar{\alpha}_t}(x_t - \sqrt{\bar{\alpha}_t}\, x_0)$

But the forward equation rearranges to $\epsilon = (x_t - \sqrt{\bar{\alpha}_t}\, x_0)/\sqrt{1 - \bar{\alpha}_t}$. Substitute:

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{\epsilon}{\sqrt{1 - \bar{\alpha}_t}}$$

Punchline. The true score is just (negative, scaled) noise. Predicting ϵ with MSE = predicting the score:

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

DDPM (Ho 2020) and score-SDE (Song 2020) are two lenses on the **same** model.

Two views side-by-side

DDPM view (Ho 2020)

- Forward · fixed Markov chain.
- Reverse · learned Gaussian chain.
- Loss · MSE between predicted and true noise.
- Intuition · *denoising* at multiple scales.

Score-SDE view (Song 2020)

- Forward · SDE driving data to noise.
- Reverse · another SDE driving noise to data.
- Loss · score matching.
- Intuition · gradient field pointing to data.

INTUITION

Use whichever is easier for your problem. DDPM's discrete-time recipe is simpler to code; Score-SDE gives more flexibility for continuous-time / arbitrary-schedule models (e.g., flow matching in 2023+).

PART 5

Why diffusion won

Diffusion vs VAE vs GAN

	VAE	GAN	DIFFUSION
Sample quality	blurry	sharp	**SOTA**
Training	stable, fast	brittle	stable, slow
Likelihood	ELBO	✗	ELBO (loose)
Sampling	1 forward	1 forward	**T forwards**
Mode coverage	strong	mode collapse risk	**strong**
Interpretability	structured latent	messy latent	uniform latent

KEY IDEA

Diffusion's big cost · slow sampling. This is what L22 will focus on — classifier-free guidance, latent diffusion, DDIM.

Why diffusion beat GANs on image quality

1. **Training signal is always strong** · MSE on noise has a meaningful gradient at every step and every x_t. GAN's adversarial loss often gives near-zero gradient early.
2. **No mode collapse** · every training example teaches the model to denoise independently. The model can't "cheat" by producing one output.
3. **Iterative refinement** · generation is 50-1000 tiny corrections. Errors at each step are small; the chain self-corrects. GANs must produce the final output in one forward pass.
4. **Infinite data augmentation** · every (x_0, t, ε) triple is a new training example. A dataset of 10k images gives you a virtually infinite training stream.

A picture of why iteration helps

Think about drawing a face. A GAN must commit · "these pixels are skin, these are eyes, this is hair" — all in one forward pass. Wrong commitments cascade.

> KEY IDEA
>
> Diffusion starts with pure noise; the *first* reverse step sketches the rough layout; the *second* adds features; the *hundredth* adds skin texture. The network revises its answer 1000 times, getting it right in the limit.

This is why diffusion samples look sharper and more coherent than any single-forward-pass generator.

Applications · 2026 state

- **Text-to-image** · Stable Diffusion, Midjourney, DALL-E 3, Imagen.
- **Video** · Sora, Runway Gen-3, VEO.
- **Audio** · AudioGen, Riffusion.
- **Molecule design** · RFdiffusion for proteins.
- **Robotics policies** · diffusion policy (Chi et al. 2023).

Diffusion has become the default generative model across modalities.

Frontier · where diffusion is heading

Faster sampling

- DDIM (L22) · deterministic, 20 steps.
- Flow matching · 5-10 steps.
- Consistency models · 1-4 steps.

Richer conditioning

- CFG (L22) · text steering.
- ControlNet · per-pixel conditioning (pose, depth).
- Inpainting · mask what to regenerate.

IN PRACTICE

Consistency models and flow matching are closing the "slow sampling" gap. In 2026 · expect 1-step diffusion samplers to become competitive with GANs on speed.

Common questions · FAQ

Q. Is diffusion a likelihood-based model?

A. Yes, approximately. The DDPM loss corresponds to a variational lower bound on $\log p(x)$, but with a specific weighting. Tight bounds need "improved DDPM" tricks.

Q. Why is the schedule Gaussian, not uniform?

A. Because Gaussians are closed under convolution — lets us write $q(x_t \mid x_0)$ in closed form. Other noise distributions (uniform, Laplacian) don't give this gift.

Q. What if the data isn't image-like?

A. Use a different architecture (Transformer for sequences, GNN for graphs). The diffusion recipe is independent of architecture — only the noise-prediction network changes.

Summary · Lecture 21 — summary

- **Forward process** · add small Gaussian noise over T steps; closed form $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$.
- **Reverse process** · neural net predicts the noise; subtract step by step.
- **DDPM loss** · MSE between true noise and predicted noise. Stable, simple.
- **Schedule** · linear or cosine; cosine is modern default.
- **Architecture** · U-Net with sinusoidal time embedding + attention at low res.
- **Score matching** · same model through a different lens; reverse diffusion ≈ Langevin dynamics along the score.

Read before Lecture 22

Prince Ch 18 (later sections) + HF `diffusers` docs + Rombach 2022 (Stable Diffusion).

Next lecture

Diffusion Models — Practice — classifier-free guidance, latent diffusion, DDIM, DiT.

NOTEBOOK

Notebook 21 · `21-ddpm-2d.ipynb` — implement DDPM on a 2D toy dataset (Swiss roll); visualize forward noising + reverse denoising animations.