GANs

Lecture 20 · 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. Describe the **GAN minimax game** and the Nash equilibrium.
2. Explain the **non-saturating G loss** and why it matters.
3. Apply the **DCGAN cookbook** architectural guidelines.
4. Diagnose **mode collapse** and apply fixes.
5. Derive why **Wasserstein distance** stabilizes training (JS vs EMD).
6. Place GANs vs VAE vs Diffusion in the 2026 generative landscape.

Where we are

- **VAE** (L19) · probabilistic encoder, latent-space structure, blurry samples.

Today: **GANs**. Completely different philosophy — no likelihood, no prior, two networks duking it out.

> REFERENCE
>
> Today maps to **Prince Ch 15** (GANs) + Goodfellow 2014 (original GAN) + Radford 2015 (DCGAN) + Arjovsky 2017 (WGAN).

Four questions:

1. What is the **minimax game**?
2. Why is GAN training so **unstable**?
3. What is **mode collapse** and how do we fight it?
4. What is **WGAN** and what did it fix?

PART 1

Why a new paradigm?

First the intuition, then the math

What "generate" even means

Classification asks: given x, predict y. Training signal · labels.
Generation asks: draw new samples x that look like training data. Training signal · ???

KEY IDEA

There is **no label** saying "this image is correct." Every plausible image is correct, and every implausible one is wrong. How do you train a network on that?

The answer from VAE (L19) was: compress-then-decompress with a probabilistic latent. The answer from GAN (today) is: **train a critic to tell you if your output is plausible, use its gradient as the label.**

The generation task · picture

Imagine you're given 10,000 photos of cats. You must produce new cat photos that look real — not in the dataset, but plausibly drawable from the same distribution.

DERIVATION

Mathematically: estimate $p_{\text{data}}(x)$ and sample from it. Or equivalently, learn a map $G : z \to x$ from noise to a distribution indistinguishable from p_{data}.

This looks innocent. But p_{data} over 256×256 RGB images lives in $\mathbb{R}^{196,608}$ and has support on a tiny curved manifold. Density estimation there is nightmarish.

Why not just fit a Gaussian?

Tempting: fit $p_{\text{data}} \approx \mathcal{N}(\mu, \Sigma)$ from the data, then sample.

WATCH OUT

Doesn't work. A Gaussian's support is all of $\mathbb{R}^d$. Natural images occupy a curved manifold of vastly lower dimension. Gaussian samples are uniformly noise-like — never anything resembling an image.

Same problem for any simple parametric family. Images live where *simple* doesn't reach.

The deep-generative idea

Don't try to write down p_{data}. Instead, define

$$p_G(x) = \mathcal{N}(0, I) \xrightarrow{G} x$$

i.e. · sample a simple distribution (Gaussian noise z), push it through a deep network G. The **output distribution** of that network is our generative model.

KEY IDEA

The neural network is the *distribution*. We never write it down; we just sample from it by sampling noise and running forward. That's the big shift in the 2014 era.

Training question · how do we make G's output distribution match p_{data} when we can't even compute $p_G(x)$?

The 2014 insight · use a classifier

Goodfellow's idea · even if we can't compute p_G directly, we *can* train a **classifier D** to tell real samples from generated ones. If D is perfect, its gradient tells G exactly where to move fakes to fool it.

KEY IDEA

GAN = generator G + adversary D, trained together. The adversary's loss is well-defined (binary cross-entropy). Its gradients carry *implicit* information about p_{data} that G uses without ever evaluating a density.

One paper → a decade of progress in generative models.

The forger-and-detective analogy

KEY IDEA

G is a counterfeiter. It makes fake paintings and tries to pass them off as real.

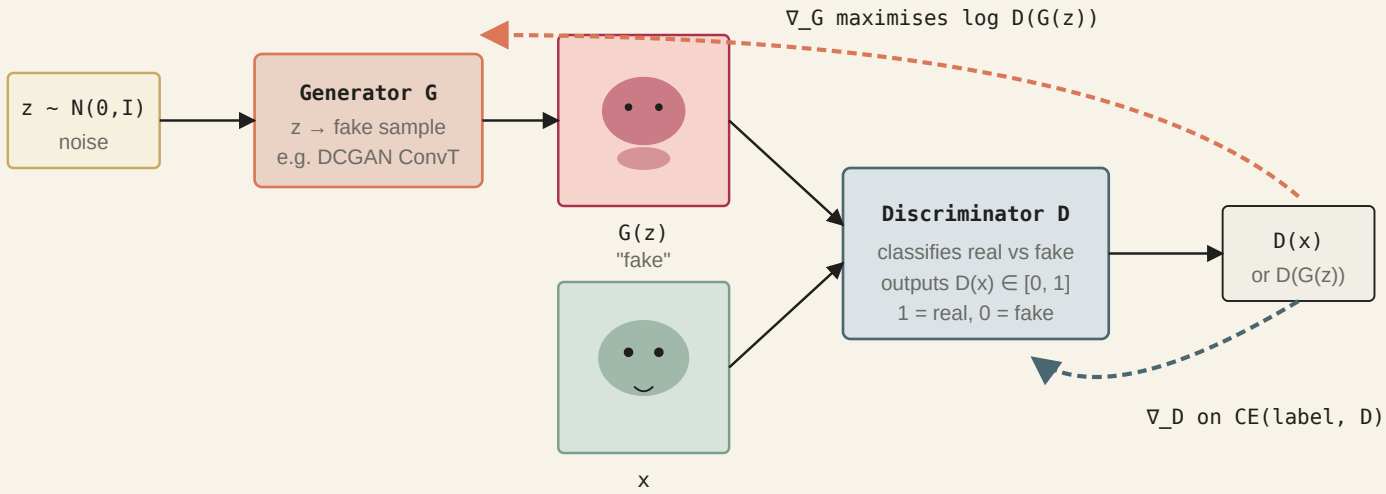
D is an art detective. It sees a mix of real and fake paintings and labels each.

- D gets feedback: *"that one was fake — here's how I should have known"*. D gets better.
- G gets D's gradient: *"here's what fooled you, and here's what didn't"*. G gets better.
- Eventually · G paints indistinguishably from real. D is reduced to random guessing.

That's the **Nash equilibrium** — and it's what the math below formalizes.

The GAN pipeline

GAN — generator tries to fool discriminator; discriminator tries not to be fooled

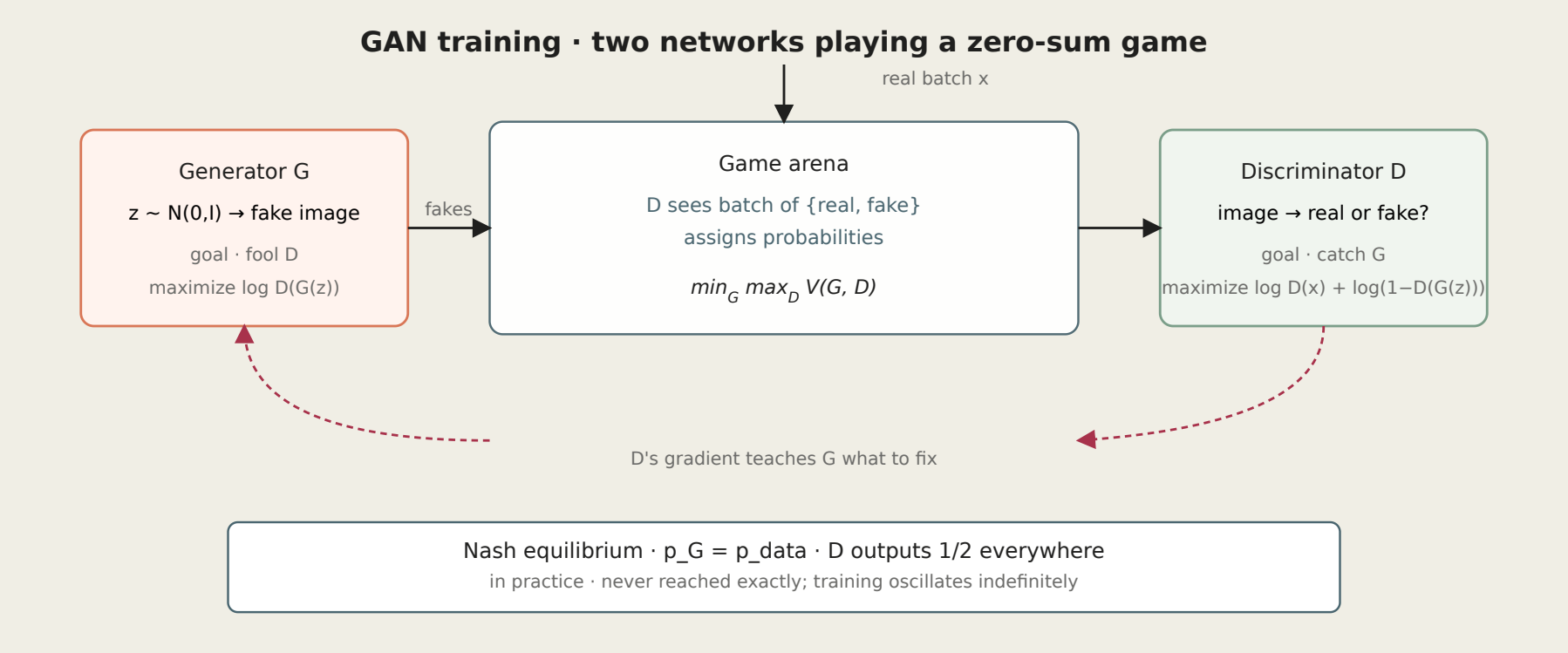


MINIMAX OBJECTIVE

$$\min_G \max_D \; \mathbb{E}[\log D(x)] + \mathbb{E}[\log(1 - D(G(z)))]$$

D maximises → distinguish real from fake · G minimises → fool D

Two networks, one game



IN PRACTICE

▶

Interactive: scrub through training steps; watch G's distribution slide onto the real one and D(x) flatten to 0.5 — gan-minimax-dance.

A 1D toy · watch G learn a bimodal target

Target data · two Gaussians at $x = -2$ and $x = 2$.

DERIVATION

Step 0 · $G(z) \sim \mathcal{N}(0, 1.5^2)$ — a wide blob centered at zero. D detects it easily (real = ±2, fake near 0).

Step 100 · G has shifted its mass outward; two bumps emerge, near but not on the modes.

Step 500 · G matches the two modes closely. D's output is ~0.5 everywhere.

Step 1000 · p_G = p_data exactly. D is random guessing. Nash equilibrium.

This trajectory is **animated in the interactive**. The math that follows formalizes each bullet above.

PART 2

The minimax objective

The math behind the dance

Minimax · derived from binary cross-entropy

D is just a binary classifier. We know how to train those.

D's goal. For real x, output near 1. For fake $G(z)$, output near 0. Combined (BCE):
$\text{maximize } \log D(x) + \log(1 - D(G(z)))$

Average over the data distribution and noise:

$$\max_D \; \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$$

G's goal. Opposite — make $D(G(z))$ near 1. Equivalent to **minimizing** the same expression. So:

$$\min_G \max_D \; V(D, G)$$

A two-player minimax game. Nash equilibrium · $p_G = p_{\text{data}}$ and $D \equiv 0.5$.

Worked numeric · a single step

One real x and one fake $G(z)$. Initial state · D learning, G bad.

- $D(x) = 0.7$ (D thinks real is somewhat real)
- $D(G(z)) = 0.2$ (D thinks fake is mostly fake)

D's objective value.
$\log 0.7 + \log(1 - 0.2) = -0.36 + \log 0.8 = -0.36 - 0.22 = \mathbf{-0.58}$
Maximize → push $D(x) \to 1$ and $D(G(z)) \to 0$.

G's objective value. G only sees the second term: $\log(1 - 0.2) = -0.22$.
Minimize → push $D(G(z)) \to 1$.

Each step both networks adjust → an equilibrium dance.

⚠️

optional · deriving the optimal D

Fix G. Maximize over D pointwise · for each x, maximize $a \log D + b \log(1 - D)$ with $a = p_{\text{data}}(x)$, $b = p_G(x)$.

Take derivative · $a/D - b/(1 - D) = 0 \rightarrow a(1 - D) = bD \rightarrow D^* = a/(a + b)$:

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

Sanity check on D^*:

- Clearly real ($p_{\text{data}} = 0.9, p_G = 0.1$): $D^* = 0.9/1.0 = 0.9$ ✓
- Clearly fake ($p_{\text{data}} = 0.05, p_G = 0.95$): $D^* = 0.05/1.0 = 0.05$ ✓
- **Nash equilibrium ($p_{\text{data}} = p_G$):** $D^* = 0.5$ everywhere ✓

Plug D^* back into the GAN objective and simplify · the outer min becomes

$$\min_G \; 2\,\text{JSD}(p_{\text{data}} \,\|\, p_G) - \log 4$$

The GAN objective is **equivalent to minimizing Jensen–Shannon divergence**. JSD = 0 iff $p_G = p_{\text{data}}$.

Alternating updates · the training loop

```
for batch in loader:
    # 1. Update discriminator
    opt_D.zero_grad()
    real = batch
    fake = G(torch.randn(BATCH, NOISE_DIM)).detach()   # stop grad through G here
    loss_D = -(D(real).log() + (1 - D(fake)).log()).mean()
    loss_D.backward(); opt_D.step()

    # 2. Update generator
    opt_G.zero_grad()
    fake = G(torch.randn(BATCH, NOISE_DIM))
    loss_G = -(D(fake).log()).mean()     # non-saturating (see next slide)
    loss_G.backward(); opt_G.step()
```

Key pattern — alternate between updating D and G. Balance is critical: if D gets too strong, G can't learn.

Why `.detach()` matters

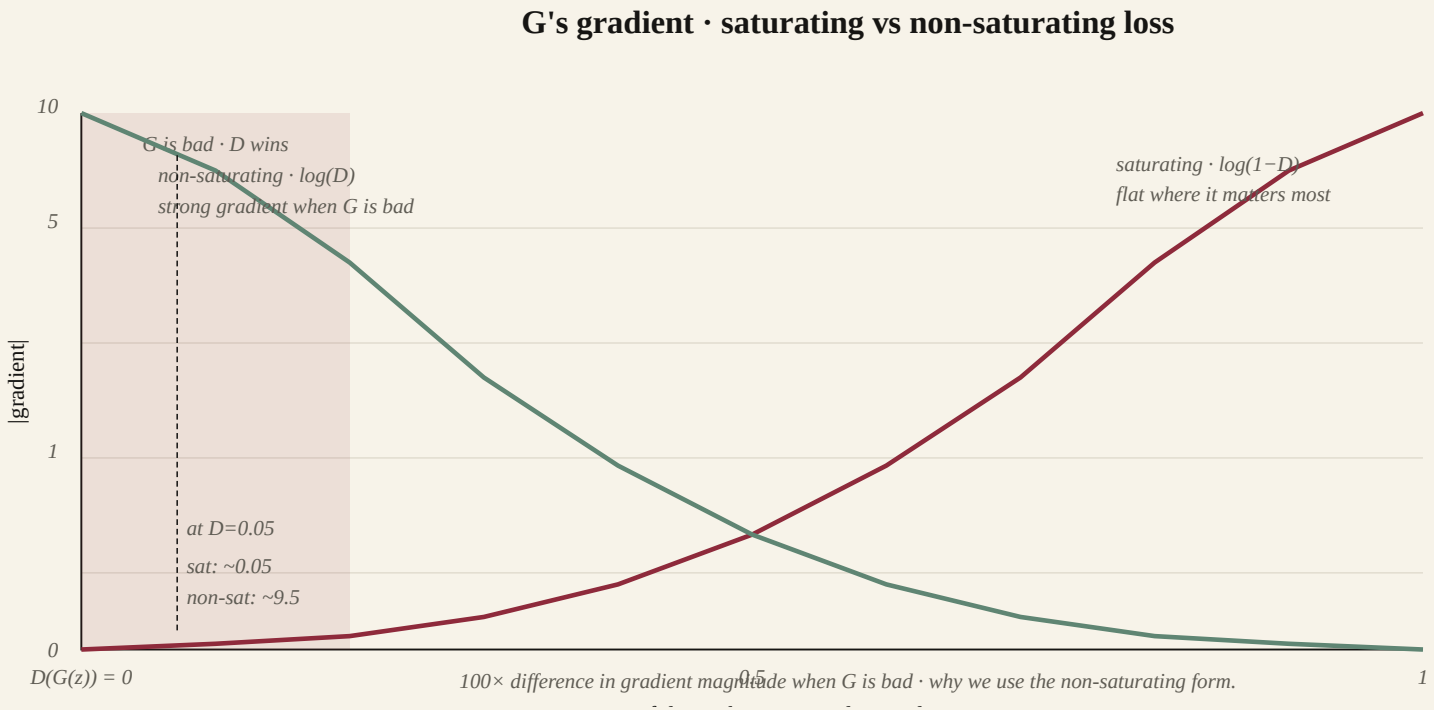
In the D-update, we compute `fake = G(z).detach()`. Why?

KEY IDEA

Without `detach()`, PyTorch builds a graph through G. Calling `backward()` on D's loss would compute G's gradients too — but we don't want to update G yet. `detach()` snips the graph at G's output. G's params get no gradient from the D update.

It's a subtle but critical detail. Forgetting it is one of the top-3 bugs in from-scratch GAN code.

G's gradient · saturating vs non-saturating



The non-saturating trick

Intuition · early in training, D always wins — fakes look nothing like real. So $D(G(z)) \approx 0$, and $\log(1 - D(G(z))) \approx \log 1 = 0$. Flat. **No gradient for G to improve.**

The original G objective — minimize $\log(1 - D(G(z)))$ — **saturates** when D is confident about fake samples. Gradient goes to zero.

DERIVATION

Non-saturating G objective (Goodfellow 2014 footnote, became the standard):

$$\max_G \ \mathbb{E}_z[\log D(G(z))]$$

Maximize $\log D(G(z))$ instead of minimizing $\log(1 - D(G(z)))$. Same optimum, much better gradients early in training.

Everyone uses the non-saturating version.

Worked numeric · the gradient gap

Let D's pre-sigmoid activation be a, so $D(G(z)) = \sigma(a)$. Suppose D is confident · $\sigma(a) = 0.01$ (so $a \approx -4.6$). G's update depends on $\partial L / \partial a$.

Saturating loss $L = \log(1 - \sigma(a))$.
Chain rule · $\partial L / \partial a = -\sigma(a)$. At $\sigma = 0.01$: gradient $= \mathbf{-0.01}$.

Non-saturating loss $L = -\log \sigma(a)$.
Chain rule · $\partial L / \partial a = \sigma(a) - 1$. At $\sigma = 0.01$: gradient $= 0.01 - 1 = \mathbf{-0.99}$.

The non-saturating gradient is ~100× stronger in this common early-training regime. That gap kept vanilla GANs untrainable for 2 years until Goodfellow's footnote fix.

Why GAN training is fundamentally hard

Every optimizer you've seen (SGD, Adam) is for **single-objective minimization**. You're walking a fixed landscape.

GANs are different — **neither loss is fixed**. When G moves, D's optimal landscape shifts. When D moves, G's optimal landscape shifts.

> WATCH OUT
>
> This is a *saddle-point search* in a 10^9-dimensional game. Standard optimization guarantees (convergence, stability, unique optimum) do not apply. Everything in GAN lore — DCGAN tricks, spectral normalization, WGAN — is fighting this.

PART 3

DCGAN · the architecture that worked

Radford et al. 2015

Before DCGAN · GANs barely trained

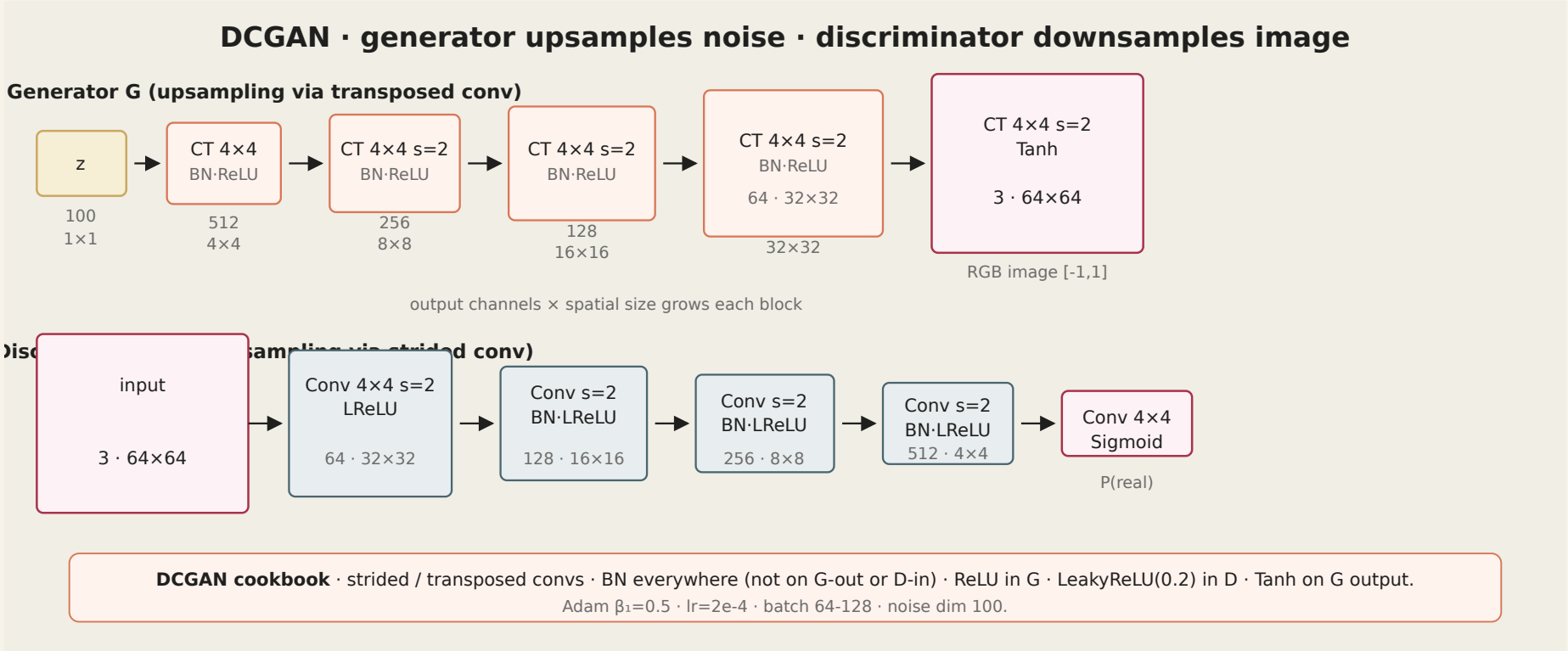
2014-2015 papers reported:

- MNIST: barely legible digits.
- CIFAR-10: looks like smudged noise.
- Faces: training diverged often before converging.

Dozens of competing GAN variants; none consistently trained. Needed were **architectural** norms, not loss tweaks.

> REFERENCE
>
> Radford, Metz, Chintala 2015 · *"Unsupervised Representation Learning with Deep Convolutional GANs"* — a cookbook that made the whole field tractable.

DCGAN · architecture at a glance



DCGAN · why these specific tricks?

The DCGAN rules aren't arbitrary · each is a **stabilizer** for the tricky GAN game.

DERIVATION

- **LeakyReLU in D** · stops "dead neurons" so gradients keep flowing even on clearly-fake samples.
- **BN everywhere** · keeps activations from drifting · preserves the delicate G/D balance.
- **Tanh on G output** · matches the real-image normalization $[-1, 1]$. Without this, G can output anywhere and D learns "real images have bounded pixels" as a useless signal.
- **Strided convs (no pooling)** · learnable downsampling lets G/D adapt to the data distribution.

Each rule is a small wedge that prevents a known failure mode. Combined, they made GANs train.

DCGAN · five architectural guidelines

1. Replace pooling with **strided convolutions** (both D and G).
2. Use **batch normalization** in both.
3. Remove fully-connected hidden layers.
4. **ReLU** in G (except output, which uses Tanh).
5. **LeakyReLU** in D.

These aren't deep insights — they are a cookbook that made GANs actually train.

Transposed convolution · upsampling primitive

G needs to go from `(batch, noise_dim)` to `(batch, 3, 64, 64)` — *upsampling*. Use `ConvTranspose2d`:

DERIVATION

A normal conv shrinks (or preserves) spatial size. A transposed conv *inflates* · each input pixel is multiplied by the kernel and spread into a larger output.

Dimension formula (inverse of conv) · $O = (W - 1) \cdot S - 2P + K$

For $W = 1, S = 1, P = 0, K = 4$ · output is 4×4. Four such blocks, each stride-2, take $1 \times 1 \to 64 \times 64$.

Alternative · nearest-neighbor upsample + regular conv. Often cleaner; fewer "checkerboard" artifacts.

Why BN in both networks

Batch norm stabilizes GANs because:

1. **Centers activations.** Prevents internal covariate shift from one side pulling the other off balance.
2. **Acts as implicit regularizer.** Reduces the capacity for any single layer to dominate.
3. **Makes training less sensitive to initialization.**

> WATCH OUT
>
> **Exception**: the generator's output layer and the discriminator's input layer should *not* have BN — they'd destroy the fine-grained info there. The DCGAN paper is explicit about this.

Generator in PyTorch · DCGAN

```python
class Generator(nn.Module):
    def __init__(self, nz=100, ngf=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf*8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf*8), nn.ReLU(True),

            nn.ConvTranspose2d(ngf*8, ngf*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf*4), nn.ReLU(True),

            nn.ConvTranspose2d(ngf*4, ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf*2), nn.ReLU(True),

            nn.ConvTranspose2d(ngf*2, 3, 4, 2, 1, bias=False),
            nn.Tanh()                                    # output in [-1, 1]
        )

    def forward(self, z):
        return self.net(z.view(z.size(0), -1, 1, 1))
```

Noise shape: `[batch, 100]` → reshaped to `[batch, 100, 1, 1]` → upsampled to `[batch, 3, 32, 32]`.

Discriminator in PyTorch · DCGAN

```python
class Discriminator(nn.Module):
    def __init__(self, ndf=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(ndf, ndf*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf*2), nn.LeakyReLU(0.2, True),

            nn.Conv2d(ndf*2, ndf*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf*4), nn.LeakyReLU(0.2, True),

            nn.Conv2d(ndf*4, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x): return self.net(x).view(-1)
```

Mirror of G, basically. No BN on the first layer; LeakyReLU helps gradients flow for negative activations (dead-neuron fix).

Hyperparameter recipe that works

DERIVATION

- **Optimizer** · Adam with $\beta_1 = 0.5$, $\beta_2 = 0.999$ (not 0.9 — stability matters).
- **Learning rate** · 2×10^{-4}, same for D and G.
- **Batch size** · 64-128.
- **Noise dim** · 100.
- **Weight init** · `N(0, 0.02)` for conv layers; `N(1, 0.02)` for BN gamma.

This exact recipe trains on most small-to-medium image datasets without hand-holding. Useful starting point for any GAN project.

PART 4

Training instability & mode collapse

The pathologies

Why GANs are hard to train

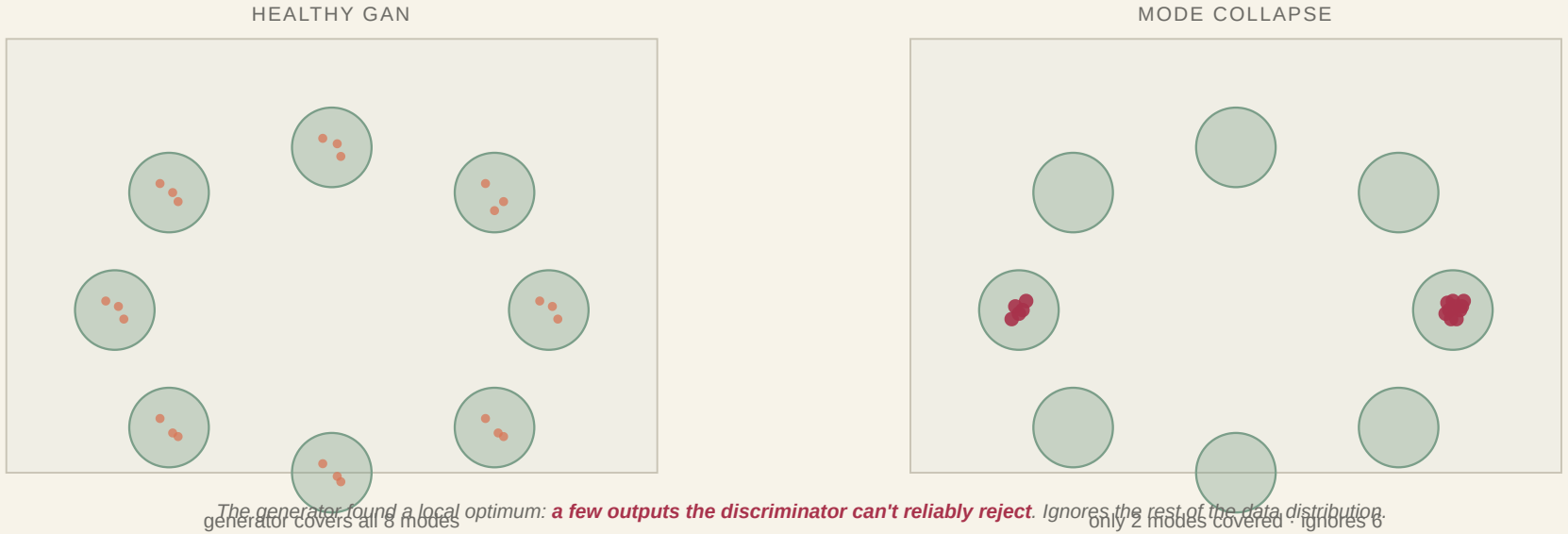
GAN training is a **non-cooperative game**. Three common failure modes:

1. **D too strong** — confidently rejects all fakes early. G gets zero gradient, never improves.
2. **G too strong** — fools D completely. D loses discriminative power, G has no signal.
3. **Mode collapse** — G finds a few outputs that consistently fool D. Stops exploring.

> WATCH OUT
>
> Most of 2015–2019 GAN research is fighting these failure modes.

Mode collapse visually

Mode collapse — generator produces few distinct outputs, ignores most of the data



*The generator found a local optimum: **a few outputs the discriminator can't reliably reject**. Ignores the rest of the data distribution.*

Why mode collapse happens · mechanism

G doesn't need to produce the *full* distribution to get low loss. It just needs to fool D on whatever samples it produces.

KEY IDEA

Suppose G finds one output (say, one "mode" of faces) that reliably fools D. D can only fix this by *re-learning* that particular mode; while it does, G moves to another mode. The two chase through a few modes and **settle on whichever is easiest** — never seeing the full distribution.

Picture · instead of p_G covering p_data, p_G is a point mass (or thin ridge) sitting inside p_data.

Fixing mode collapse · the toolbox

- **Minibatch discrimination** · let D see samples as a batch, not one-by-one. If G gives 64 near-identical fakes, D instantly spots it.
- **Feature matching** · G optimizes to match mean feature activations of real batch, not just fool D.
- **Unrolled GANs** · let G see D's *next-few-step* updates when computing its loss; makes G consider D's response.
- **WGAN (next section)** · Wasserstein distance naturally doesn't mode-collapse.
- **Two Time-Scale Update Rule (TTUR)** · different learning rates for D and G.

> IN PRACTICE
>
> In 2026, if you need a GAN you almost always use WGAN-GP or StyleGAN architecture; both mostly eliminate mode collapse in practice.

Diagnosing GAN health

Standard diagnostics:

- **D loss** · should hover ~0.5 (balanced). If D loss → 0, D is winning too much.
- **G loss** · should plateau, not grow.
- **Samples** · visually check for diversity. Do you see the same face repeatedly? Mode collapse.
- **Inception Score (IS)** · diversity + quality metric for image GANs.
- **FID (Fréchet Inception Distance)** · distance between fake and real feature distributions. Lower is better.
  FID is the main quantitative metric for image generation in 2026.

FID in one paragraph

Fréchet Inception Distance takes two sets of images (real vs fake), runs both through a pretrained Inception-V3, gets 2048-dim feature vectors, and computes:

DERIVATION

$$\text{FID} = \|\mu_r - \mu_f\|^2 + \text{tr}\left(\Sigma_r + \Sigma_f - 2(\Sigma_r\Sigma_f)^{1/2}\right)$$

i.e. · Wasserstein-2 distance between two Gaussians fitted to the feature distributions.

Lower is better. FID of 10-20 · "looks good". FID of 3-5 · "basically indistinguishable". SOTA diffusion on ImageNet is ~2.

PART 5

WGAN · Wasserstein distance

Arjovsky et al. 2017

The problem with JS

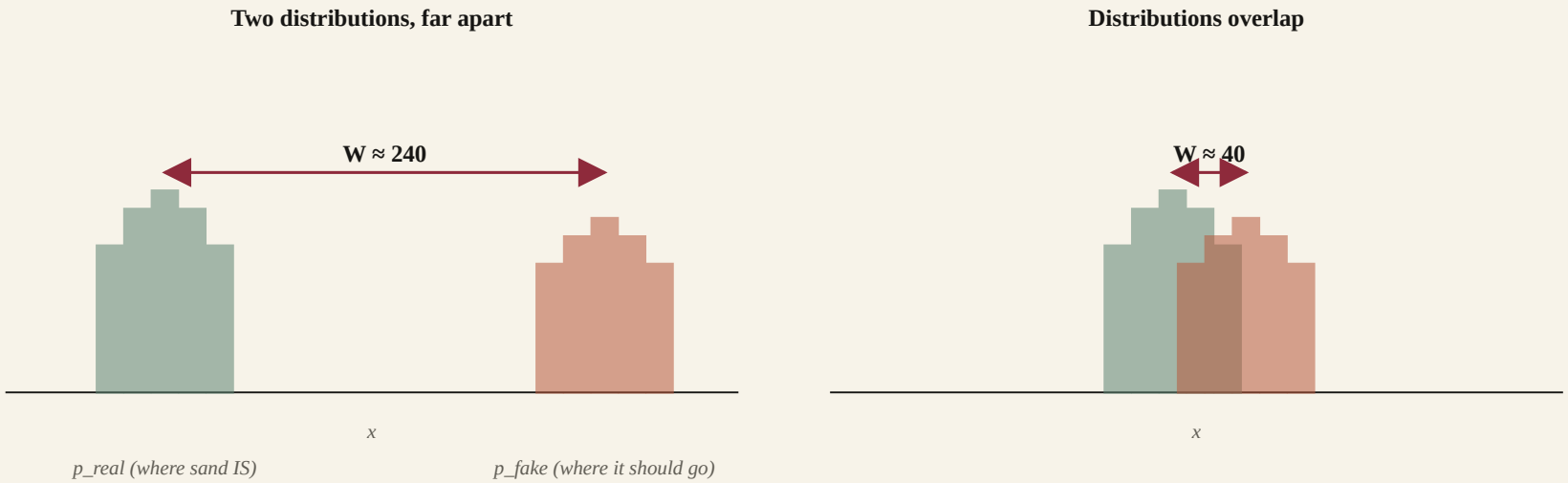
Recall · the original GAN minimizes JS divergence. When p_{data} and p_G don't overlap, JS is constant (≈log 2). **The gradient of a constant is zero.** G has no signal to improve.

KEY IDEA

Early in training, fake samples are far from real ones — they barely overlap. JS is saturated. G's gradient is zero. This is the fundamental reason vanilla GANs struggle at the start.

Wasserstein distance · the picture

Wasserstein-1 distance · the minimum work to move sand pile A onto pile B



JS divergence · saturates at log 2 when distributions don't overlap · gradient = 0 → G learns nothing.
Wasserstein · linear in distance · gradient always points toward p_real · G keeps improving.

Earth-mover distance · intuition

Imagine two piles of sand (two distributions). **How much work** to reshape one pile into the other — moving each grain the minimum distance?

DERIVATION

$$W(p, q) = \inf_{\gamma} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

where γ is a "transport plan" — how much mass moves from x to y. Minimum cost over all plans = Wasserstein distance.

Unlike JS, W is **smooth** even when supports don't overlap. Move the pile 1 meter → $W = 1$. Move it 10 meters → $W = 10$. Always differentiable.

WGAN · critic with a speed limit

Vanilla GAN's classifier draws an infinitely steep cliff between real/fake. When supports don't overlap, the cliff has zero gradient at its base. **G gets no signal.**

INTUITION

WGAN fix · 1-Lipschitz critic. A "critic" D scores realness (any real number, not a probability). Its slope is capped at 1 — never steeper than 45°. Now there's *always* a gentle non-zero slope from fakes to reals → G always gets gradient.

Using Kantorovich–Rubinstein duality:

$$W(p_{\text{data}}, p_G) = \sup_{\|D\|_L \leq 1} \mathbb{E}_{p_{\text{data}}}[D(x)] - \mathbb{E}_{p_G}[D(G(z))]$$

D's job · **maximize** the score difference (high on real, low on fake). G's job · **minimize** $\mathbb{E}_{p_G}[D]$. No sigmoid, no log, just raw scores.

1-Lipschitz check. $D(x) = 5x$? With $a = 2, b = 3$: $|D(a) - D(b)| = 5 > 1 = |a - b|$. **Not** 1-Lipschitz. $D(x) = 0.5x$? $|D(a) - D(b)| = 0.5 \leq 1$. **Yes.**

WGAN-GP · highway-patrol analogy

INTUITION

We want D's slope to be 1 everywhere — but checking *everywhere* is impossible. Highway patrol can't put a camera on every metre, so it places **random** ones. WGAN-GP picks **random points** $\hat{x}$ on the line between real and fake, and penalizes any slope $\neq 1$ there. Surprisingly enough.

WGAN-GP · the gradient penalty, term by term

$$\mathcal{L}_{\text{GP}} = \lambda\, \mathbb{E}_{\hat{x}}\big[\big(\underbrace{\|\nabla_{\hat{x}} D(\hat{x})\|_2}_{\text{slope at } \hat{x}} - 1\big)^2\big]$$

1. **Where do we sample?** $\hat{x} = \epsilon\, x + (1-\epsilon)\, G(z)$ for random $\epsilon \in [0, 1]$ — a random point between a real and a fake.
2. **How do we measure slope?** Compute $\nabla_{\hat{x}} D(\hat{x})$ (gradient of critic w.r.t. its input), take L2 norm.
3. **Penalty.** $(\text{slope} - 1)^2$:

- slope = 1 → 0 (no penalty)
- slope = 1.5 → 0.25
- slope = 0.2 → 0.64

Critic's full loss · $\mathbb{E}[D(G(z))] - \mathbb{E}[D(x)] + \lambda \cdot \text{GP}$.

Stabilizes training and effectively eliminates mode collapse.

Why WGAN fixed everything

- **Smooth gradients even for disjoint distributions** · G always gets a meaningful signal.
- **Loss correlates with sample quality** · for the first time, you can watch training curves and know if it's working.
- **Less sensitive to hyperparameters** · default lr=1e-4, batch=64, $\beta_1 = 0$, $\beta_2 = 0.9$ works for most datasets.
- **Mode collapse rare** · earth-mover distance doesn't allow p_G to collapse inside p_data.

IN PRACTICE

Between 2017 and 2020 WGAN-GP became the default "safe" GAN variant. Beyond it, spectral normalization (SN-GAN) added more robustness and scales better to 1024×1024.

PART 6

StyleGAN · the GAN peak

Disentangled latents, hyper-realistic faces

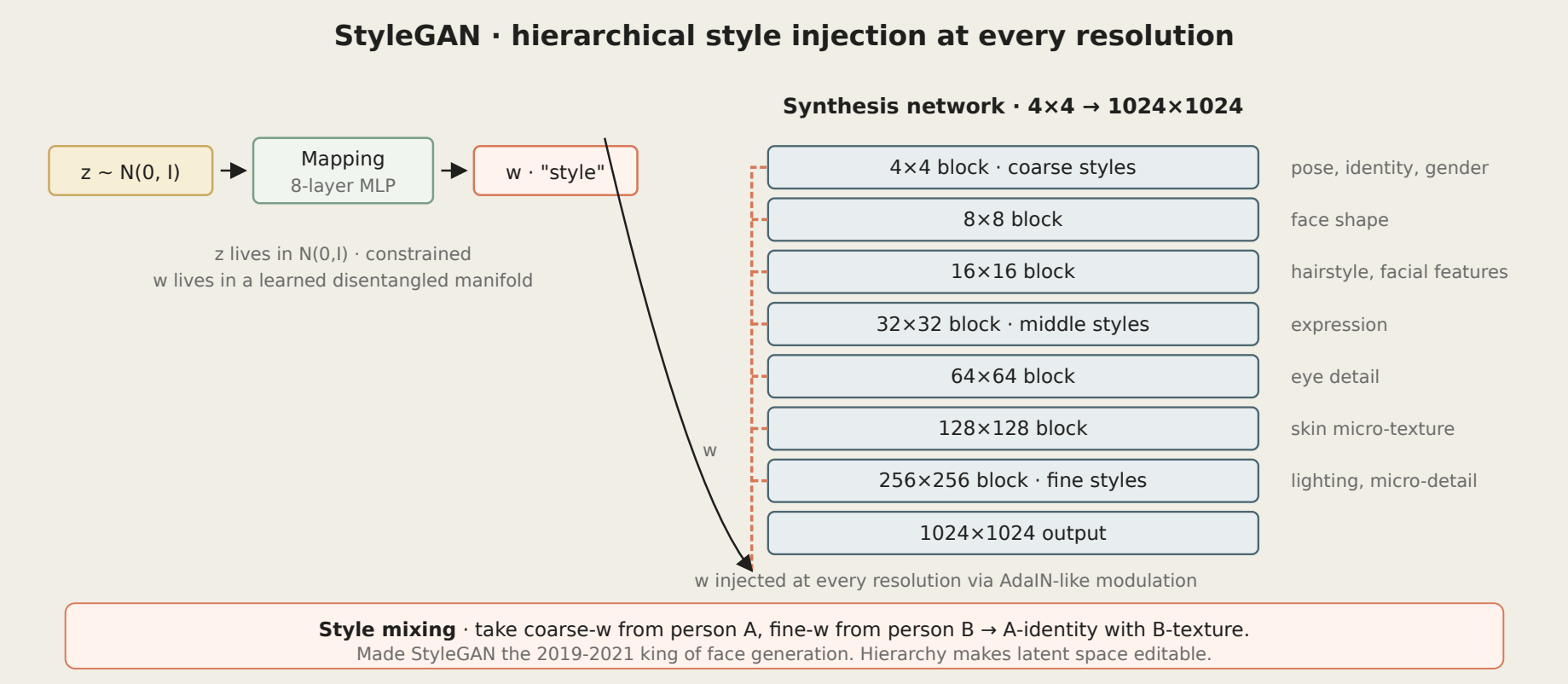
What StyleGAN changed

Karras et al. (NVIDIA) 2018-2021 · three generations of StyleGAN.

- **Style-based generator** · noise z is projected to a disentangled latent w, then injected at each resolution via AdaIN-like modulation.
- **Progressive growing** (v1) · train at 4×4 first, add layers, train at 8×8, ...
- **Anti-aliasing & equivariance** (v3) · images stay smooth under translation/rotation.

  Result · 1024×1024 face generation indistinguishable from photographs.

StyleGAN · hierarchical style injection



StyleGAN · the latent hierarchy

Each resolution block receives a separate injection from w. This creates a **semantic hierarchy**:

DERIVATION

- **Coarse blocks** (4×4 → 16×16) · control pose, identity, gender.
- **Middle blocks** (32×32 → 64×64) · control hair style, eye shape, facial features.
- **Fine blocks** (128×128+) · control lighting, micro-texture, skin details.

You can mix · coarse-w from person A, fine-w from person B → hybrid face. This is the "style mixing" demos that made StyleGAN famous.

GAN applications · the 2017-2021 peak

- **Face generation** · This Person Does Not Exist, deepfakes, avatar creation.
- **Art & style transfer** · CycleGAN (unpaired), pix2pix (paired), Style transfer.
- **Super-resolution** · SRGAN, ESRGAN — take low-res, output high-res.
- **Image editing** · GAN inversion → edit in w → re-decode.
- **Data augmentation** · synthetic data for rare classes.
- **Domain adaptation** · summer → winter photos, horses → zebras.

This era gave us "AI-generated photo" as a concept.

PART 7

GANs in 2026

Still alive, but niche

The GAN era · 2014-2020

YEAR	MODEL	WHAT IT DID
2014	GAN	original paper — 28×28 MNIST
2015	DCGAN	convolutional, stable training
2017	WGAN-GP	Wasserstein + gradient penalty
2017	ProGAN	progressive growing → 1024×1024 faces
2019	StyleGAN	disentangled latent, hyper-realistic faces
2021	StyleGAN3	temporal consistency, aliasing fixes

After GANs · diffusion took over

	GANS	DIFFUSION
Training stability	✗ brittle	✓ stable
Sample quality	✓ (SOTA 2018-2020)	✓✓ (SOTA 2021+)
Likelihood	✗	≈
Inference speed	✓✓ one pass	✗ many passes
Diversity	mode collapse risk	✓ natural
Latent space	rich, explorable	uniform-ish

IN PRACTICE

In 2026 · **diffusion dominates** text-to-image / video. GANs survive where inference speed matters (real-time face generation, StyleGAN-based editing).

Why diffusion won · the real reasons

1. **Stable training** · diffusion's regression loss (MSE on noise) has a unique global optimum; no adversarial game.
2. **Mode coverage** · diffusion naturally covers the whole distribution — no equilibrium to collapse.
3. **Scales to massive data** · GANs plateau around CelebA scale; diffusion keeps improving with more data/compute.
4. **Text conditioning cleaner** · cross-attention at every denoising step beats GAN's class-conditioning hacks.
5. **Likelihood-like metrics** · ELBO bounds let you measure progress properly.

GANs didn't disappear — they were **outperformed at their own strength** (sample quality) while being simultaneously worse at training, coverage, and conditioning.

When to still reach for a GAN

- **Real-time generation** · one forward pass is orders of magnitude faster than 20+ diffusion steps.
- **Editing existing content** · GAN inversion + latent editing is the cleanest available pipeline.
- **Small, closed-domain datasets** · faces, fashion, specific object categories — StyleGAN still wins on some metrics.
- **Research pedagogy** · the minimax game teaches adversarial thinking; foundational for many ideas beyond GANs.

> INTUITION
>
> Analogy · RNNs vs Transformers. RNNs didn't vanish, just found niches (tiny devices, streaming). GANs are the same story.

Adversarial thinking beyond GANs

The idea of "use a critic to train a generator" shows up everywhere:

- **Reinforcement learning from human feedback** (L16) · reward model trained on pairs is a critic.
- **Robust optimization** · minimax between defender and adversary (adversarial examples, robust training).
- **Domain adversarial training** · classifier + domain discriminator for transfer.
- **Actor-critic methods in RL** · value function as a learned reward signal for the policy.

> KEY IDEA
>
> The single most durable idea from GANs is **"a neural network can act as a learned loss function."** That framework outlived the original GAN itself.

Summary · Lecture 20 — summary

- **GAN** · two networks, minimax objective · G generates, D classifies real vs fake.
- **Non-saturating G loss** · maximise $\log D(G(z))$ — better gradients early.
- **DCGAN** · architectural cookbook (stride convs, BN, ReLU/LeakyReLU, Tanh output).
- **Mode collapse** · G produces few distinct outputs; fight with diversity regularizers or WGAN.
- **WGAN-GP** · Wasserstein distance + gradient penalty · stable training.
- **StyleGAN** · disentangled w-space, hyper-realistic faces, the 2019-2021 peak.
- **2026** · diffusion has largely replaced GANs; StyleGAN still used where real-time generation matters.

Read before Lecture 21

Prince Ch 18 · Diffusion models (early sections).

Next lecture

Diffusion Models — Theory — forward noising, DDPM training, score matching.

NOTEBOOK

Notebook 20 · `20-dcgan-mnist.ipynb` — DCGAN on MNIST or CelebA subset; monitor D/G loss balance; generate face grid.