Generative Adversarial Networks: A Third Way to Build a Generative Model — No Density, No ELBO, Just a Game Between Two Networks

Consolidated with Claude AI from course notes and HWs

1 Motivation

Two generative models seen so far both optimize an explicit density:

- HW4 Problem 3's Gaussian mixture maximizes an exact likelihood $p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$ via EM.
- The companion VAE note's model maximizes a *lower bound* on a likelihood (the ELBO), because the true likelihood $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})\, d\mathbf{z}$ is intractable once the decoder is a neural network.

A **Generative Adversarial Network (GAN)** takes a genuinely different approach: it never writes down, evaluates, or bounds any density $p(\mathbf{x})$ at all. Instead, it trains a network to produce samples that another network cannot distinguish from real data. This is sometimes called an **implicit density model** — you get a sampler, not a likelihood.

2 The Two Players

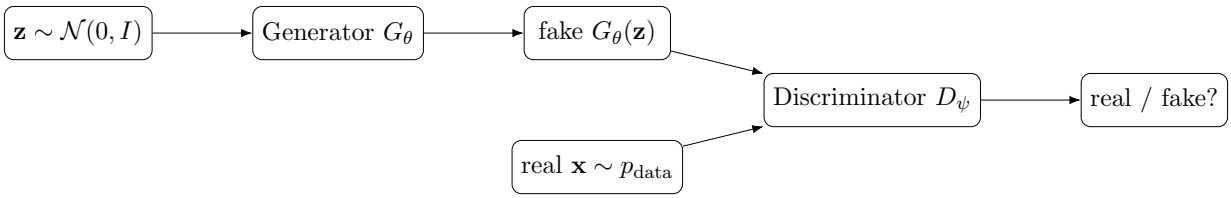
A GAN consists of two networks trained against each other:

- A **generator** $G_\theta : \mathbb{R}^k \to \mathbb{R}^d$, mapping noise $\mathbf{z} \sim p(\mathbf{z})$ (typically $\mathcal{N}(0, I)$) to a fake data point $G_\theta(\mathbf{z})$.
- A **discriminator** $D_\psi : \mathbb{R}^d \to [0, 1]$, outputting the probability that its input is real data rather than something G_θ produced.

D_ψ is trained to get better at telling real from fake; G_θ is trained to get better at fooling D_ψ. Neither network ever sees a likelihood, a KL term, or a latent-variable posterior — the only signal either one receives is the other network's output.

How new is this, structurally? Not at all, for the architectures themselves — only for how they're trained. The **generator** is the same "stack of `Linear` + activation" recipe as the decoder half of HW5 Problem 1's autoencoder (or the VAE's decoder): a vector goes in, a few layers happen, a data-shaped vector comes out — the only difference is that its input is pure noise instead of a code with any relationship to real data. The **discriminator** has the same shape as HW5 Problem 2's classifier network: a vector goes in, a score comes out. Two small implementation details are worth flagging up front, though, since they differ from what Problems 1–2 do: the generator's final layer usually has *no* squashing activation at all (it outputs raw data values, not a probability, unlike Problem 1/2's sigmoid outputs), and the discriminator's final layer is typically left as a raw linear score with no sigmoid, letting the loss function (`BCEWithLogitsLoss`) apply the sigmoid internally for numerical stability, rather than building it into the network the way Problem 2 does.

Remark. Same code-reuse opportunity as the VAE note: whatever small helper builds a plain MLP for Problem 1's decoder or Problem 2's classifier can build both G_θ and D_ψ here unchanged in *shape* — just drop the final activation for G_θ, and drop it (or fold it into the loss) for D_ψ.



3 The Minimax Game

The idea, before the notation: we're not writing down one loss and minimizing it the usual way. D is trying to win one game (catch every fake), and G is trying to win the opposite game (never get caught), and we let them fight it out with alternating gradient steps. The formula below is just a compact way of writing "D wants this number big, G wants it small" as a single expression, so we can talk about both networks' goals at once.

Training is framed as a two-player minimax game with value function:

$$\min_{\theta} \max_{\psi} \; V(D_\psi, G_\theta) \;=\; \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\big[\log D_\psi(\mathbf{x})\big] \;+\; \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\log\big(1 - D_\psi(G_\theta(\mathbf{z}))\big)\big]. \tag{1}$$

Reading Eq. (1) term by term: the discriminator ψ wants $D_\psi(\mathbf{x})$ close to 1 on real data and $D_\psi(G_\theta(\mathbf{z}))$ close to 0 on fakes (maximizing V); the generator θ wants $D_\psi(G_\theta(\mathbf{z}))$ close to 1 — fooling the discriminator — which *minimizes* the second term. Unlike every other objective in this course, no single network is doing straightforward gradient descent on a fixed loss: each network's loss depends on the *current state* of the other, so training is a saddle-point search, not a minimization.

3.1 What the Optimal Discriminator Looks Like

Why bother working this out: it turns out that at the mathematical optimum of this whole game, something clean falls out — the generator ends up producing exactly the real data distribution. That's reassuring: this two-network competition isn't just a heuristic that happens to sort of work, it has a well-defined "correct answer" underneath it, even though nothing in training ever computes that answer directly. Here is how to see that.

Fix G_θ (equivalently, fix the distribution p_g that G_θ induces over $\mathbf{x}$), and ask: what D maximizes $V(D, G_\theta)$? Pointwise in $\mathbf{x}$, we are maximizing $p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))$ over $D(\mathbf{x}) \in [0, 1]$. Differentiating and setting to zero gives:

$$D^\star(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{2}$$

Claim 1. *At the global optimum of the minimax game, $p_g = p_{data}$, $D^\star(\mathbf{x}) \equiv \frac{1}{2}$ everywhere, and $V(D^\star, G^\star) = -\log 4$.*

Sketch. Substituting Eq. (2) into V and simplifying shows $V(D^\star_G, G) \;=\; -\log 4 + 2 \cdot \mathrm{JSD}(p_{\text{data}} \,\|\, p_g)$, where JSD is the (always ≥ 0) Jensen–Shannon divergence. This is minimized over G exactly when $\mathrm{JSD}(p_{\text{data}} \| p_g) = 0$, i.e. $p_g = p_{\text{data}}$, giving $V = -\log 4$. □

Key point: In plain terms: even though neither network ever computes a density, the competition between them is mathematically equivalent to shrinking a genuine measure of "how different are these two distributions" down to zero. So a GAN really is fitting a distribution to the data — it just gets there by two networks playing a game, instead of anyone ever writing down or evaluating $p(\mathbf{x})$.

4 Training Procedure

Because D_ψ needs to stay near-optimal for the theory above to apply, the standard recipe takes several discriminator steps per generator step (Goodfellow et al., 2014):

Algorithm: GAN Training (one iteration)

---

1. Repeat k times (e.g. $k = 1$ for a simple/small problem): sample a minibatch of real $\mathbf{x}$ and a minibatch of noise $\mathbf{z}$; take a gradient step on ψ to *increase* $V(D_\psi, G_\theta)$ (ascend on the discriminator's own objective).
2. Sample a fresh minibatch of noise $\mathbf{z}$; take a gradient step on θ to *decrease* $\mathbb{E}_{\mathbf{z}}[\log(1-D_\psi(G_\theta(\mathbf{z})))]$ (descend on the generator's objective, holding ψ fixed).
3. Repeat.

4.1 The Non-Saturating Generator Loss (a practical fix)

Early in training, G_θ is bad and D_ψ can reject its samples with high confidence, so $D_\psi(G_\theta(\mathbf{z})) \approx 0$. But $\log(1 - D_\psi(G_\theta(\mathbf{z})))$ has *very small gradient* near $D_\psi(G_\theta(\mathbf{z})) = 0$ (the curve is nearly flat there) — exactly when the generator needs the strongest learning signal. The standard fix replaces the generator's loss with a *non-saturating* version that pursues the same goal (make $D_\psi(G_\theta(\mathbf{z}))$ large) but has a much larger gradient in this regime:

$$\begin{aligned} & \text{minimize (original):} && \mathbb{E}_{\mathbf{z}}\big[\log(1 - D_\psi(G_\theta(\mathbf{z})))\big] \\ \longrightarrow \quad & \text{maximize (non-saturating):} && \mathbb{E}_{\mathbf{z}}\big[\log D_\psi(G_\theta(\mathbf{z}))\big]. \end{aligned} \tag{3}$$

Both push G_θ in the same direction (fool D_ψ more); Eq. (3)'s version is what essentially every practical GAN implementation actually trains, including the one HW5 Problem 5 asks you to build.

5 Common Failure Modes

- **Mode collapse:** G_θ discovers a single (or a few) outputs that reliably fool D_ψ and stops producing diverse samples — e.g. generating points from only one of HW5 Problem 5's Gaussian components instead of matching the full mixture. Nothing in Eq. (1) explicitly penalizes low diversity; it only rewards fooling the current D_ψ.
- **Vanishing generator gradients:** if D_ψ becomes too good too early (perfectly separating real from fake), $D_\psi(G_\theta(\mathbf{z})) \approx 0$ almost everywhere and, even with the non-saturating loss, the generator gets little signal to improve. Balancing how many steps each network takes (the k in the algorithm above) is a practical lever for this.
- **No monotone convergence curve:** because training is a saddle-point search, not a minimization, the losses of D_ψ and G_θ do *not* need to decrease monotonically the way a single network's training loss does elsewhere in this course — oscillation during training is normal, not automatically a bug.

Remark. Contrast this with EM (always improves a well-defined objective every iteration, HW4 Problem 4) and with the VAE (a single well-defined loss, gradient descent on both θ, ϕ jointly, HW5 Problem 4) — a GAN is the only model in this course where "the loss isn't going down smoothly" is not, by itself, evidence of a bug.

6 Continuity: Three Ways to Model the Same Data

HW4 Problem 3 and HW5 Problems 4 and 5 fit *the same toy 2-D Gaussian-mixture data* (the files `2gaussian.txt` and `3gaussian.txt`) three different ways. Lining them up:

	EM/GMM (HW4)	**VAE (HW5 Pb. 4)**	**GAN (HW5 Pb. 5)**
Density $p(\mathbf{x})$	explicit, exact	explicit, lower-bounded (ELBO)	never computed
Latent variable	discrete, K components	continuous, learned encoder	noise only, no encoder
Objective	maximize log-likelihood	maximize ELBO	minimax game (Eq. 1)
Training dynamics	coordinate ascent, monotone	joint gradient ascent, roughly monotone	saddle-point search, non-monotone

Key point: This is HW5 Problem 5(C)'s question in one sentence: EM and the VAE both have some number they can point to and say "this is (a bound on) how likely the data is under our model." A GAN has no such number, ever — only two networks' competing scores. The earlier claim about Jensen–Shannon divergence is what justifies calling the GAN's sample-comparing game a genuine (if indirect) way of matching distributions, despite that missing number.

7 HW5 Problem 5: Implementation Notes

- **Scale.** This is deliberately the smallest possible GAN: 2-D data, 2-D (or even 1-D) noise, small MLPs for both G_θ and D_ψ (e.g. two hidden layers of width 16–32 are already generous for this toy problem). No convolutions, no image data — the goal is to see the adversarial training dynamic clearly, not to generate realistic images.
- **Loss to implement.** Use Eq. (3)'s non-saturating generator loss, not the raw minimax form — with a small/simple discriminator on 2-D data, the raw form can stall visibly, while the non-saturating version trains reliably.
- **Part (B)'s checkpoints.** If training is healthy, the early scatter plot should show the generator's cloud in roughly the wrong place entirely (near the noise prior's shape); the middle checkpoint should show it beginning to find the real data's modes; by the final checkpoint it should cover *all* the modes in `2gaussian.txt`/`3gaussian.txt`, not just one. If your generator ends up producing only one tight cluster, that is mode collapse (Section 6 above) — try reducing the discriminator's relative strength (fewer D steps per G step) or the learning rate before concluding your implementation is wrong.
- **Part (C).** This is a written question, not code — answer it in terms of what Section 6's comparison table lays out: EM and the VAE both have some quantity they can point to as "the (bound on the) log-likelihood going up"; a GAN has no such quantity to point to at all, only two networks' competing objectives.

8 Take-Home Points

- A GAN replaces "optimize a density" with "win a game against an adversary that is trying to catch you" — an *implicit* density model: you get samples, never a likelihood.
- The minimax value function (Eq. (1)), at its saddle point, corresponds to the generator's distribution matching the data distribution exactly (Jensen–Shannon divergence = 0) — but this is a property of the idealized optimum, not a guarantee of any particular training run.
- The non-saturating generator loss (Eq. (3)) is a practical fix for weak early-training gradients, not a change to what the generator is ultimately trying to achieve.
- Training is a saddle-point search: oscillating losses are expected, not automatically a bug, unlike every other model in this course.
- Mode collapse — the generator finding a few outputs that fool the current discriminator and stopping there — is the GAN-specific failure mode to watch for, especially visible at HW5 Problem 5's toy multi-modal scale.
- See the companion VAE lecture note for the other two ways this course builds a generative model on the same idea (explicit exact likelihood via EM; explicit lower-bounded likelihood via the ELBO) — Section 6 above compares all three side by side.

References

- Goodfellow, I. et al. (2014). *Generative Adversarial Networks.* NeurIPS.
- Goodfellow, I. (2016). *NIPS 2016 Tutorial: Generative Adversarial Networks.* arXiv:1701.00160 — covers the non-saturating loss and common failure modes in more depth.