Generative AI:
Foundations & Architectures

Antoun Yaacoub Ph.D.

4 days

- **Day 1:** From Core Concepts to Foundational Models
- **Day 2:** LLMs & Fine-Tuning
- **Day 3:** Diffusion & Multimodality
- **Day 4:** Evaluation, Alignment & Applications

Agenda for Day 1

Lecture:

- Part 1: What is Generative AI?
    - Generative vs discriminative
    - Applications & recent history
- Part 2: Probabilistic Foundations
    - Modeling distributions, latent variables, inference
- Part 3: GANs & VAEs
    - Mechanics, math, training pitfalls, comparison

Lab:

- Hands-on Implementation: Your First GAN
- Hands-on Implementation: Your First VAE
- Comparison and Analysis

Learning Goals for Day 1

- By the end of today, you should:
  - Explain **generative vs discriminative** modeling
  - Describe key **applications** & recent breakthroughs
  - Understand **probability view** of generative models
  - Derive **GAN loss** and **VAE ELBO** at a high level
  - Understand trade-offs between **GANs and VAEs**

Assumed Background

You should already be comfortable with:

- **Basic deep learning**: MLPs, CNNs, backpropagation
- **Optimization**: SGD/Adam, loss functions
- **Basic probability**: random variables, distributions, expectation

If some of this is rusty: treat today as **gentle but dense refresher**.

How We'll Use Math

- We will use **equations**, but always tie them to:
  - Implementation intuition
  - Empirical behavior
- Focus on:
  - What to optimize
  - Where gradients come from
  - Why training can fail

Part 1 – What is Generative AI?

Defining Generative AI

- **Generative AI:**
  Models that **learn a data distribution** and can **sample new instances** from it.

- **Input**: training examples $x \sim p_{data}(x)$
- **Output**: new samples $\hat{x}$ that "look like" they are from p_{data}
- Not just classification – they *create* new artifacts (images, text, audio, code, molecules).

Generative vs Discriminative (Conceptual)

- **Discriminative models:**
  - Learn $p(y \mid x)$ or decision boundary between classes
  - *Answer*: “What is this?”

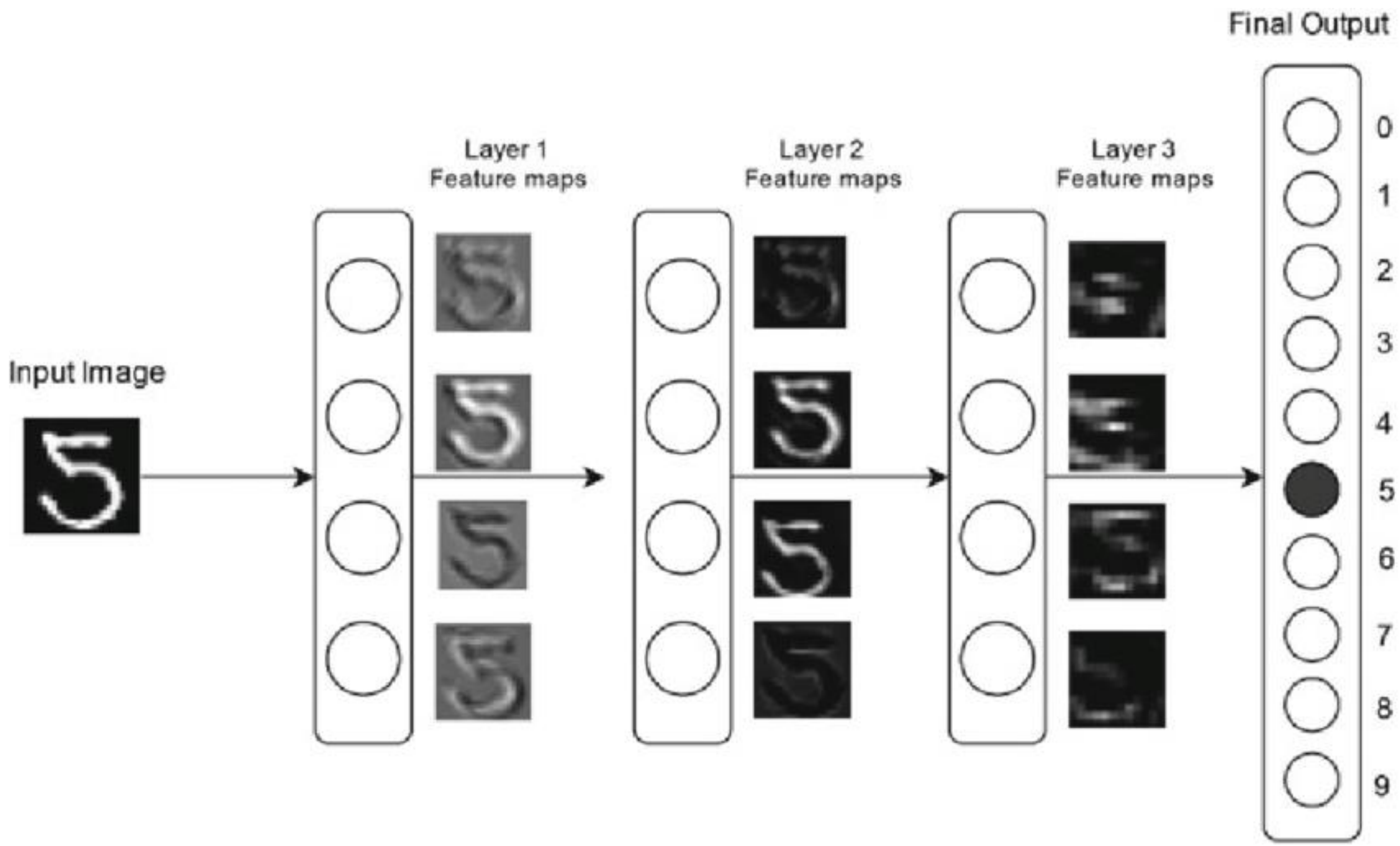
- **Generative models:**
  - Learn $p(x)$ or $p(x \mid y)$
  - *Answer*: “Generate a plausible x that looks like data”

Discriminative Example

Image classifier:

- *Input*: 28×28 grayscale digit
- *Output*: $y \in \{0,\ldots,9\}$
- *Model*: $f_\theta(x)$ → softmax over 10 classes
- Trained via cross-entropy to approximate $p(y|x)$

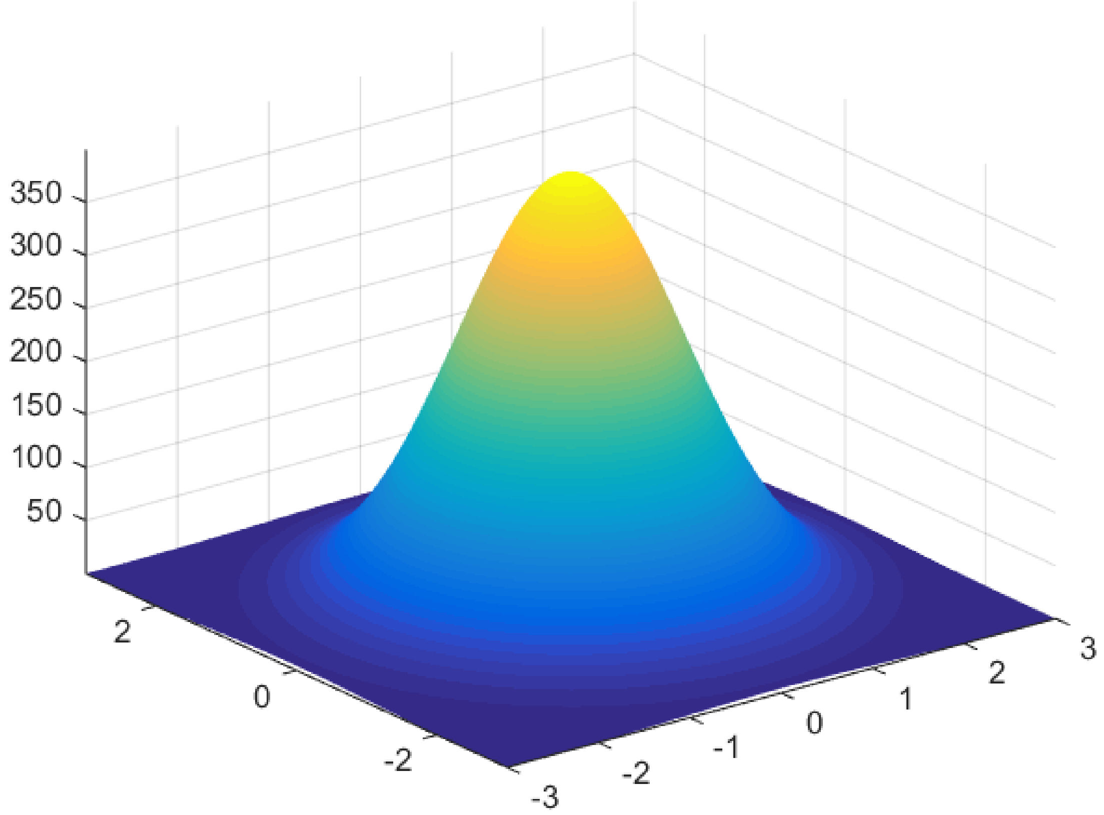
Good for recognition, not for generating digits.



Generative Example

Digit generator:

- *Input*: random noise $z \sim N(0, I)$ or latent vector
- *Output*: fake 28×28 images x̂
- Model approximates a **generator** $G_\theta(z)$ such that x̂ $\sim p_{model}(x) \approx p_{data}(x)$



We judge it by **visual realism** and **diversity**.

Why Generative Models Matter

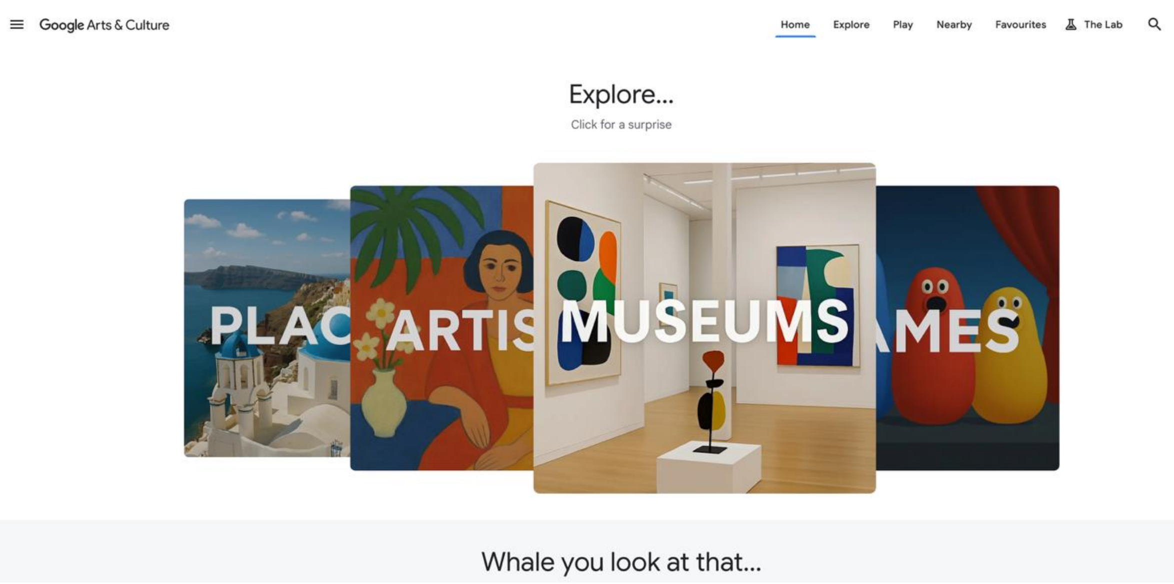
- Data augmentation (images, rare cases, simulation)
- Compression & representation learning (latent spaces)
- Creative tools (art, music, design)
- Scientific discovery (molecules, proteins, materials)
- Simulation & planning (world models, RL training)

They are not just **toys**; they're **core infrastructure** for modern AI.

Applications: Images & Vision

- Image synthesis (portraits, landscapes, product shots)
- Super-resolution (enhancing resolution)
- Inpainting (fill missing or masked regions)
- Style transfer (e.g., “Van Gogh” style)
- Domain translation (day ↔ night, summer ↔ winter)

GANs & diffusion models are central here.



https://artsandculture.google.com/

Applications: Text & Language

- Text generation (stories, chats, code docs)
- Summarization (compress articles, reports)
- Translation, paraphrasing, rewriting
- Code generation & assistance (Copilot-like)
- Search + synthesis (RAG, Q&A systems)

Today we focus on **principles**, not specific LLMs.

Applications: Audio & Speech

- Text-to-speech (TTS) and voice cloning
- Music generation (melodies, accompaniments)
- Sound effects and ambience generation
- Speech enhancement (denoising, source separation)

Models are often **auto-regressive** or based on **diffusion in waveform/spectrogram space**.

Applications: Science & Molecules

- De novo molecule generation
- Protein structure and sequence design
- Materials design (alloys, polymers)
- Synthetic scientific data for simulation

Generative AI moves from *predicting* to *proposing* hypotheses.

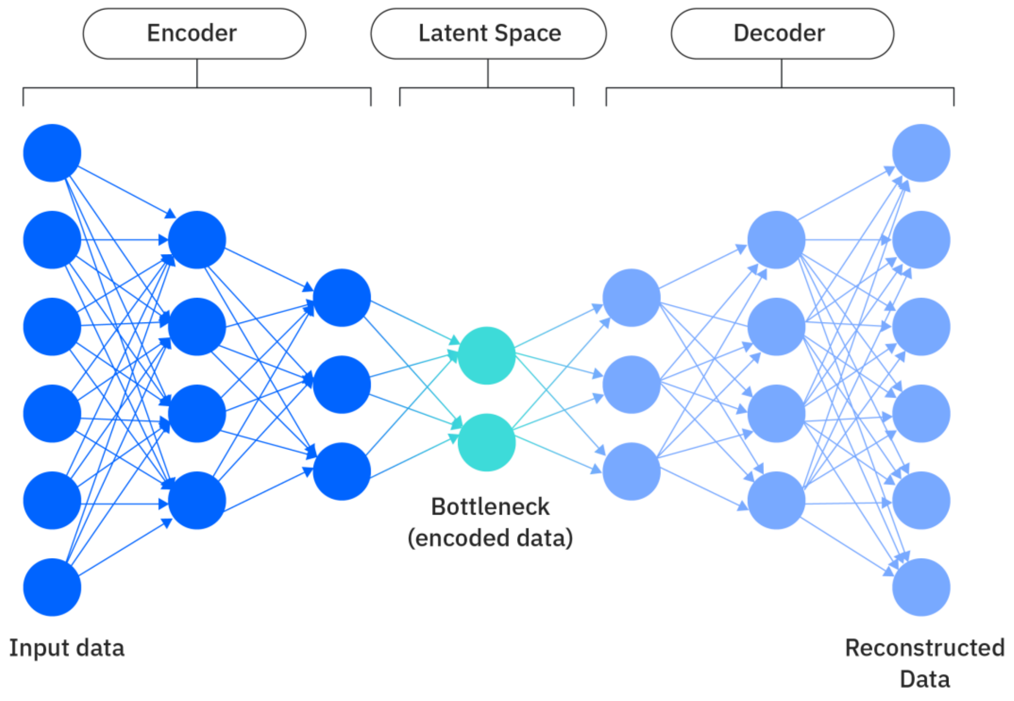
Generative vs Discriminative

Aspect	Discriminative ($p(y|x)$)	Generative ($p(x)$, $p(x|y)$)
Task	Predict label	Generate data
Output	Class/score	Sample $\hat{x}$ resembling data
Training	Cross-entropy, classification	Likelihood, adversarial, variational
Applications	Recognition, detection	Synthesis, imputation, simulation

Generative Models as Data Compressors

Informal view:

- If you can learn $p(x)$ well, you can **compress x**:
  - Encode x into a shorter representation (**latent**)
  - Decode from latent back to x
- VAEs explicitly do this
- GANs implicitly learn a compressive mapping from $z \rightarrow x$



Good generative models capture **structure + variation**

Generative Models as World Models (Conceptual)

Not going into agents, but:

- $p(x_t \mid x_{<t})$: generative models of **trajectories**
- Can simulate plausible futures or alternative scenarios
- Useful for planning/decision-making in RL

We'll revisit this on Day 4 when discussing **applications**.

Key Families of Generative Models

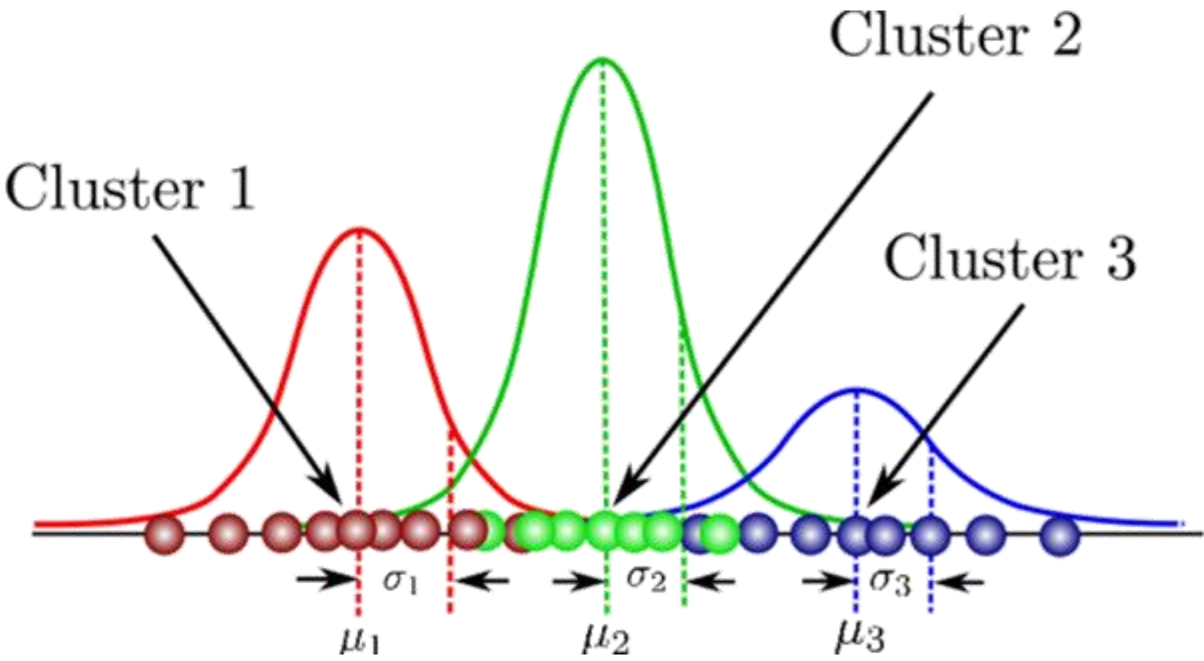
- **Auto-regressive** models (PixelCNN, GPT-like)
- **Latent variable models** (VAEs)
- **Adversarial models** (GANs)
- **Energy-based** models
- **Diffusion** models

Today’s focus: **latent variable** & **adversarial** (VAEs, GANs).

A Timeline of Breakthroughs (Pre-Deep Learning)

- Mixture models (e.g., Gaussian Mixture Models)
- Probabilistic graphical models (HMMs, Bayesian networks)
- RBMs & Deep Belief Networks

These were early attempts at flexible generative modeling.



Timeline: Deep Generative Models (2014–2017)

- **2014:** GANs (Goodfellow et al.)
- **2013–2014:** VAEs (Kingma & Welling; Rezende et al.)
- **2014–2017:** Autoregressive pixel models (PixelCNN/PixelRNN)

These are the **foundations** for modern architectures.

Timeline: From GANs & VAEs to Diffusion & LLMs

- **2017:** Transformer (“Attention Is All You Need”)
- **2018–2020:** BigGAN, StyleGAN, VQ-VAE + Transformer (DALL·E style)
- **2020–2022:** DDPM & diffusion models, Latent Diffusion (Stable Diffusion)
- **2018–present:** Large-scale Transformers → GPT, PaLM, etc.

We’ll connect back to these when comparing architectures.

Questions?

Part 2 – Probabilistic Foundations of Generative Modeling

Probability View: What Are We Learning?

Goal:

- We observe samples x $\sim p_{data}(x)$
- We want a model $p_\theta(x)$ that approximates $p_{data}(x)$

A good $p_\theta(x)$ allows us to:

- Evaluate likelihoods $p_\theta(x)$
- Sample new x̂ ~ $p_\theta(x)$
- Sometimes compute gradients $\nabla_\theta \log p_\theta \,(x)$

Explicit vs Implicit Models

- **Explicit density models:**
  - Directly define $p_\theta(x)$ (e.g., normalizing flows, autoregressive models)
  - Often tractable log-likelihood
- **Implicit models:**
  - Only define a **procedure to sample** from $p_\theta(x)$ (e.g., GANs)
  - No closed-form density

GANs: **implicit**;

VAEs: **explicit (via variational lower bound)**.

Likelihood-Based Training

If $p_\theta(x)$ is tractable:

- Use **maximum likelihood estimation (MLE)**:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta \left(x^{(i)}\right)$$

Interpretation:

- Find parameters such that data points have high probability under the model.

KL Divergence & MLE

KL divergence:

$$D_{KL}(p_{data} \parallel p_\theta) = \mathbb{E}_{x \sim p_{data}}[\log p_{data}(x) - \log p_\theta(x)]$$

- Minimizing KL w.r.t θ is equivalent to **maximizing expected log-likelihood**.
- MLE ≈ minimize $D_{KL}(p_{data} \parallel p_\theta)$.
- VAEs approximate this objective.

Latent Variable Models

Introduce latent z:

- Prior: $p(z)$ (often $N(0, I)$)
- Generative process: x ~ $p_\theta(x \mid z)$
- Marginal:

$$p_\theta(x) = \int p_\theta\,(x \mid z) p(z)\, dz$$

Latent models learn **compressed representations** z that explain data x.

Inference Problem

We want $p_\theta(z \mid x)$:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p(z)}{p_\theta(x)}$$

But $p_\theta(x)$ involves that intractable integral.

Solution: **approximate posterior** via:

- Variational inference (VAEs)
- Sampling (MCMC)
- Learned inference networks $q_\phi(z \mid x)$

VAEs use **amortized variational inference**.

Variational Inference: Basic Idea

We define a family of distributions $q_\phi(z \mid x)$:

- Try to make $q_\phi(z \mid x) \approx p_\theta(z \mid x)$
- Optimize ϕ to minimize $D_{KL}\left(q_\phi(z \mid x) \parallel\ p_\theta(z \mid x)\right)$

We can't access the true posterior directly, but we can maximize a **lower bound** on $\log p_\theta\,(x)$.

Evidence Lower Bound (ELBO)

Starting from:

$$\log p_\theta(x) = \mathcal{L}(\theta, \phi, x) + D_{KL}\left(q_\phi(z \mid x) \parallel p_\theta(z \mid x)\right)$$

where

$$\mathcal{L}(\theta, \phi, x) = \underbrace{\mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\textit{reconstruction term}} - \underbrace{D_{KL}\left(q_\phi(z \mid x) \parallel p(z)\right)}_{\textit{regularization term}}.$$

ELBO: $\mathcal{L}$ is a **lower bound** on $\log p_\theta(x)$.

Maximizing ELBO both:

- Increases likelihood
- Makes q_ϕ close to posterior.

VAE Objective (Intuition)

- ELBO has two terms:

1. **Reconstruction term**:
    - $\mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta (x \mid z)]$
    - Encourage decoder to reconstruct x from z
2. **Regularization term**:
    - $-D_{KL}\left(q_\phi(z \mid x) \ \| \ p(z)\right)$
    - Encourage approximate posterior to stay close to prior

Trade-off between **reconstruction fidelity** and **latent regularity**.

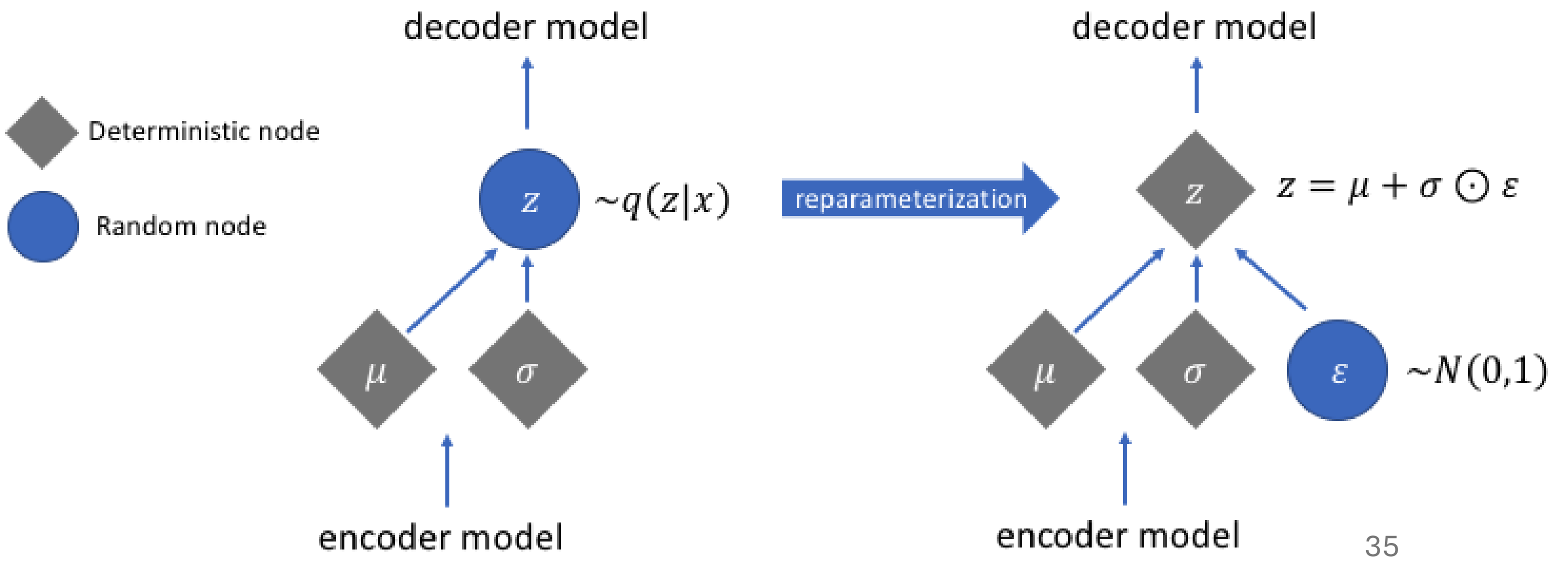
Reparameterization Trick (High-Level)

To backprop through sampling:

- Instead of sampling $z \sim q_\phi(z \mid x)$ directly:
  - Sample $\varepsilon \sim \mathcal{N}(0, I)$
  - Set $z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon$

Allows gradients to flow through μ_ϕ and σ_ϕ.

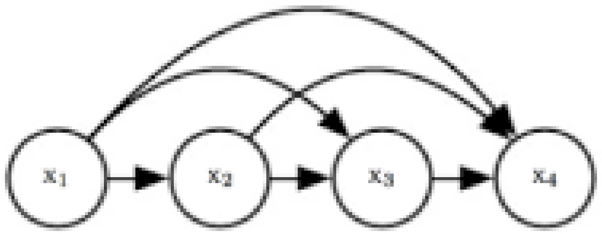
This is the key trick that makes VAEs trainable with standard backprop.



Contrast: Autoregressive Models

Another family:

- Factorize $p(x)$ as $p(x) = \prod_i p\,(x_i \mid x_{<i})$
- Train via teacher forcing (MLE)
- *Examples*: PixelCNN, GPT-like text models



$$
\begin{aligned}
p(x) &= \prod_i p(x_i | x_1, \ldots, x_{i-1}) \\
&= p(x_1)\, p(x_2 | x_1) \ldots p(x_i | x_1, \ldots, x_{i-1})
\end{aligned}
$$

Pros:

- Exact likelihood

Cons:

- Sampling is **sequential** → slower for large inputs.

Contrast: Energy-Based Models (EBMs)

Energy-based:

- Define energy $E_\theta(x)$ (low for likely data)
- $p_\theta(x) \propto \frac{e^{-E_\theta(x)}}{Z_\theta}$
- Hard part: **partition function Zθ**

Learning often uses contrastive methods (e.g., MCMC, contrastive divergence).

Not our main focus today, but conceptually related to GANs.

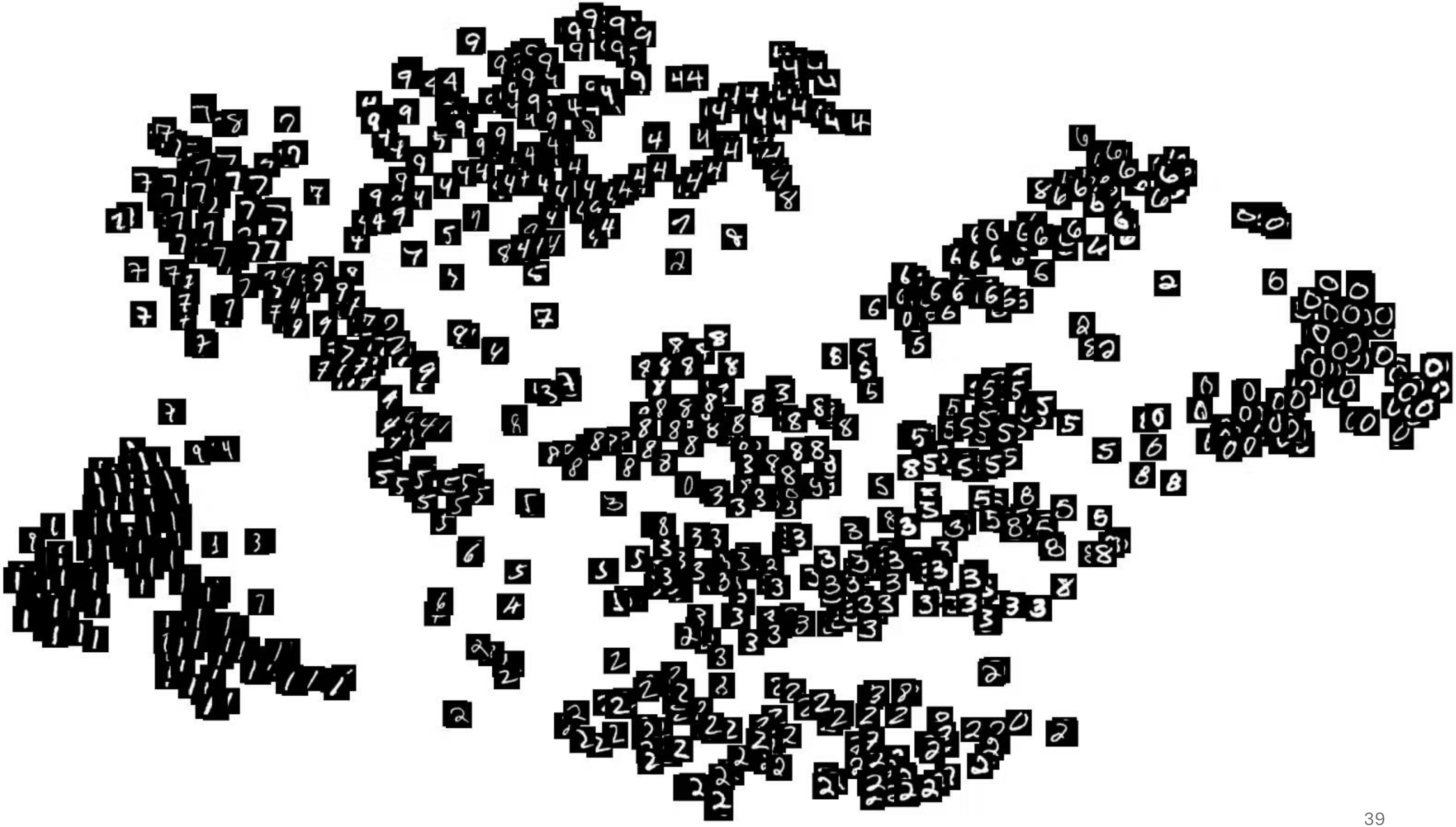
Why So Many Generative Families?

No single approach dominates:

- Likelihood-based vs adversarial
- Explicit vs implicit densities
- Latent vs non-latent

Trade-offs: sample quality, mode coverage, training stability, computational cost.

GANs and VAEs are **two ends of a spectrum**.



Visualizing Latent Space

With latent models:

- Each x maps to some z
- Latent space often has **smooth semantics**:
  - Interpolate between z's → smooth morphing between images
  - Directions correspond to attributes (e.g., smile, rotation)

VAEs explicitly provide this; GANs provide it implicitly via input noise.

Mixture Models as Latent Variable Models

Gaussian Mixture Model (GMM):

- Latent variable $c \in \{1, \dots, K\}$ (cluster index)
- $z = c; \quad p(c = k) = \pi_k$
- $x \mid c = k \sim \mathcal{N}(\mu_k, \Sigma_k)$

GMMs are a **classic latent variable model** trained with EM.

VAEs generalize this idea using deep networks and continuous z.

EM Algorithm (Very High-Level)

For latent variable models:

1. **E-step:** compute posterior over latent variables given current θ
2. **M-step:** maximize expected complete-data log likelihood

VAEs can be seen as a differentiable, amortized version of this idea.

Summary of Probabilistic Foundations (1)

Key ideas:

- We want $p_\theta(x) \approx p_{data}(x)$
- Latent variable models introduce z to explain x
- Inference is approximated via $q_\phi(z \mid x)$

These ideas underpin VAEs and, indirectly, some aspects of GANs.

Summary of Probabilistic Foundations (2)

- MLE $\Leftrightarrow$ minimize KL(p_{data} || p_θ)
- Variational inference approximates intractable posteriors
- ELBO is our optimization target in VAEs
- Reparameterization trick makes training z-parameterized networks feasible

Now: move to **GANs** – an adversarial alternative.

Transition: From Likelihood to Adversarial Training

Motivating GANs:

- Sometimes $p_\theta(x)$ is complicated; direct likelihood is hard
- Instead of **explicit likelihood**, optimize against a discriminator that tries to distinguish real vs fake

Next: the **GAN minimax game**.

Deep Dive 1: Generative Adversarial Networks (GANs)

- Minimax formulation
- Training dynamics
- Mode collapse & stability tricks

Neural Network Building Blocks (Quick Reminder)

- Fully connected layers
- Convolutions (for images)
- Activations: ReLU, LeakyReLU, etc.
- BatchNorm / LayerNorm

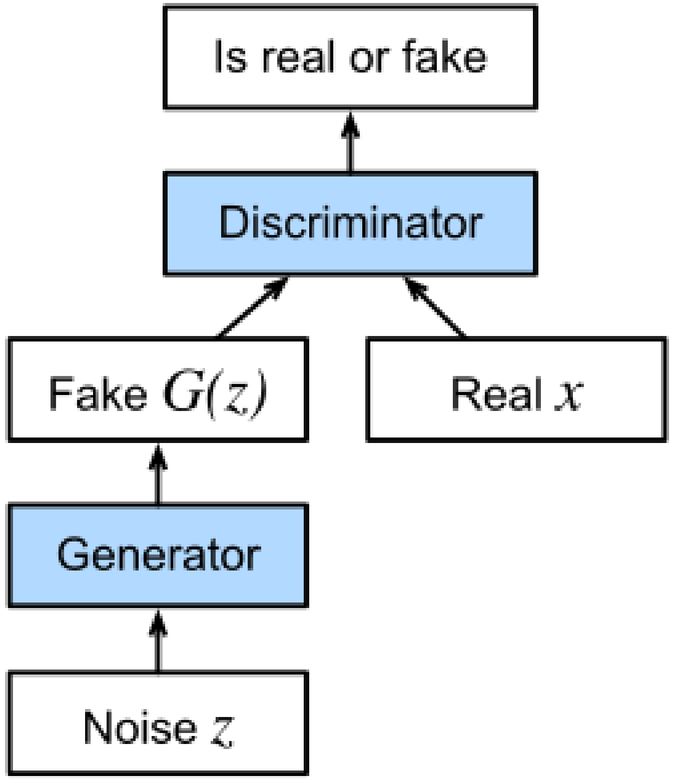
- We'll use these in G (generator) and D (discriminator).

Notation & Setup for GANs

We define:

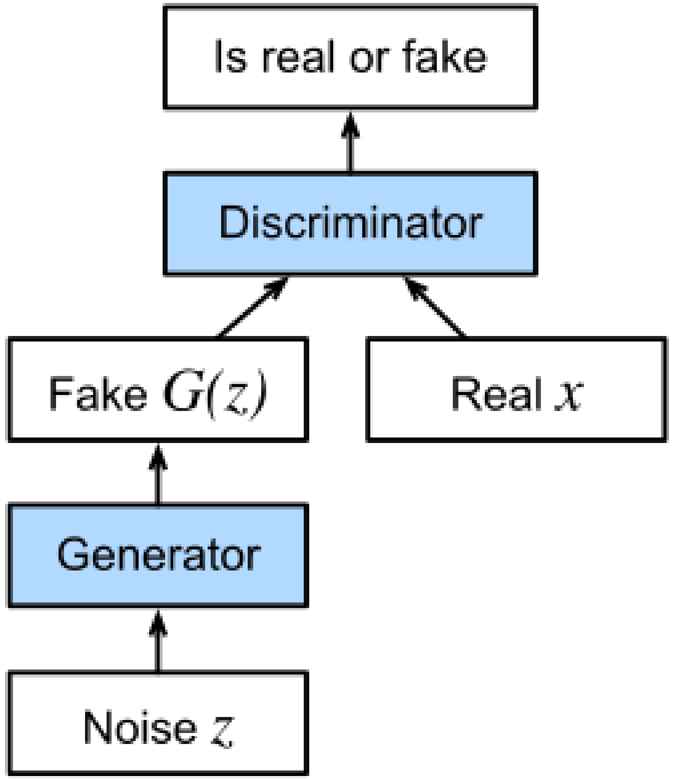
- $z \sim p_z(z)$ (simple prior, e.g. $\mathcal{N}(0, I)$)
- $G_\theta(z) \rightarrow \hat{x}$ (fake samples)
- $D_\phi(x) \rightarrow [0,1]$ (probability x is real)

Objective: train G and D in an **adversarial game**.



GAN Intuition via Two-Player Game

- **Discriminator ($\boldsymbol{D}$):**
  - Given x, decide real (from data) or fake (from G)
- **Generator ($\boldsymbol{G}$):**
  - Given z, produce $\hat{x}$ to fool D



Training process:

- D tries to **maximize** its classification accuracy
- G tries to **minimize** D's ability to detect fakes

Original GAN Objective

Value function:

$$\min_{G}\max_{D} V(D,G) = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Interpretation:

- D maximizes correct classification (real vs fake)
- G minimizes $\log(1 - D(G(z)))$ → makes $D(G(z)) \approx 1$

Part 3 – GANs: Math, Training, Pathologies

Discriminator's Objective

Given fixed G :

- Maximize:

$$E_{x \sim p_{data}}[logD(x)] + E_{x \sim p_{model}}\left[\log\Big(1 - D\big(G(z)\big)\Big)\right]$$

Optimal D^* (for fixed G):

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

This tells how D separates real vs generated densities.

Generator's Objective

Given D:

- Minimize:

$$\mathbb{E}_{z\sim p_z}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Alternative (non-saturating) loss:

$$\max_{G} \quad \mathbb{E}_{z\sim p_z}\left[\log D\left(G(z)\right)\right]$$

This variant provides stronger gradients early in training.

GAN and JS Divergence

With optimal D:

- The minimax game reduces to minimizing **Jensen–Shannon divergence** between p_{data} and p_{model} :

$$\min_{G} \quad 2 \cdot JS(p_{data} \parallel p_{model}) - 2\log 2$$

- Thus, GAN training aims to align distributions in an adversarial way.

GAN Training Algorithm (Pseudocode)

1. For k steps: update D
   - Sample minibatch of real $\{x\}$
   - Sample noise $\{z\}$
   - Compute D loss: $-[\log D\,(x) + \log(1 - \mathrm{D}(\mathrm{G}(\mathrm{z})))]$
   - Gradient step on φ
2. Then update G once:
   - Sample noise $\{z\}$
   - Compute G loss: $-\log \mathrm{D}(\mathrm{G}(\mathrm{z}))$ (non-saturating)
   - Gradient step on θ

Repeat until convergence (or until images look good).

Architecture Choices for G and D

For images:

- **Generator (G):**
  - *Input*: z (dense) → reshape → series of transposed convolutions (upsampling)
  - *Output*: image (e.g., 1×28×28 or 3×64×64)
- **Discriminator (D):**
  - *Input*: image
  - Conv layers + downsampling → dense layers → sigmoid output

DCGAN is a classic architecture.

Generator (G): noise → image

```
z (random vector, e.g. 100-d)
                    |
              Dense layer
                    |
          Reshape to small block
                  (e.g. 256×4×4)
                    |
          +------------------------+
          |  Transposed Conv 1     |
          |   Upsample: 4×4→8×8    |
          +------------------------+
                    |
          +------------------------+
          |  Transposed Conv 2     |
          |  Upsample: 8×8→16×16   |
          +------------------------+
                    |
          +------------------------+
          |  Transposed Conv 3     |
```

```
        |
+------------------------+
|  Transposed Conv 3     |
| Upsample: 16×16→32×32  |
+------------------------+
         |
+------------------------+
|  Transposed Conv 4     |
| Upsample: 32×32→64×64  |
+------------------------+
         |
  Final output layer
         |
Generated image (e.g. 3×64×64)
```

Discriminator (D): image → real/fake

```
Input image (e.g. 3×64×64)
                   |
          +-----------------------+
          |    Conv Layer 1       |
          | Downsample: 64×64→32×32|
          +-----------------------+
                   |
          +-----------------------+
          |    Conv Layer 2       |
          | Downsample: 32×32→16×16|
          +-----------------------+
                   |
          +-----------------------+
          |    Conv Layer 3       |
          | Downsample: 16×16→8×8 |
          +-----------------------+
                   |
          +-----------------------+
          |    Conv Layer 4       |
```

```
          |
 +-----------------------+
 |    Conv Layer 4       |
 | Downsample: 8×8→4×4   |
 +-----------------------+
          |
  Flatten features
          |
   Dense layers
          |
     Sigmoid
          |
Output: probability real (0–1)
```

Key Constraints in Discriminator Design

- Strong enough to detect differences, but not too strong early on
- Overpowered D can saturate gradients for G
- Underpowered D can't provide useful signal

Regularization (weight decay, spectral norm) helps stabilize training.

Visualization: GAN Training Dynamics (Conceptual)

- **Early**: G produces pure noise; D easily distinguishes
- **Mid**: G finds some modes; D gets challenged on those regions
- **Late**: ideally, G approximates full data distribution; D’s accuracy ≈ 0.5 everywhere

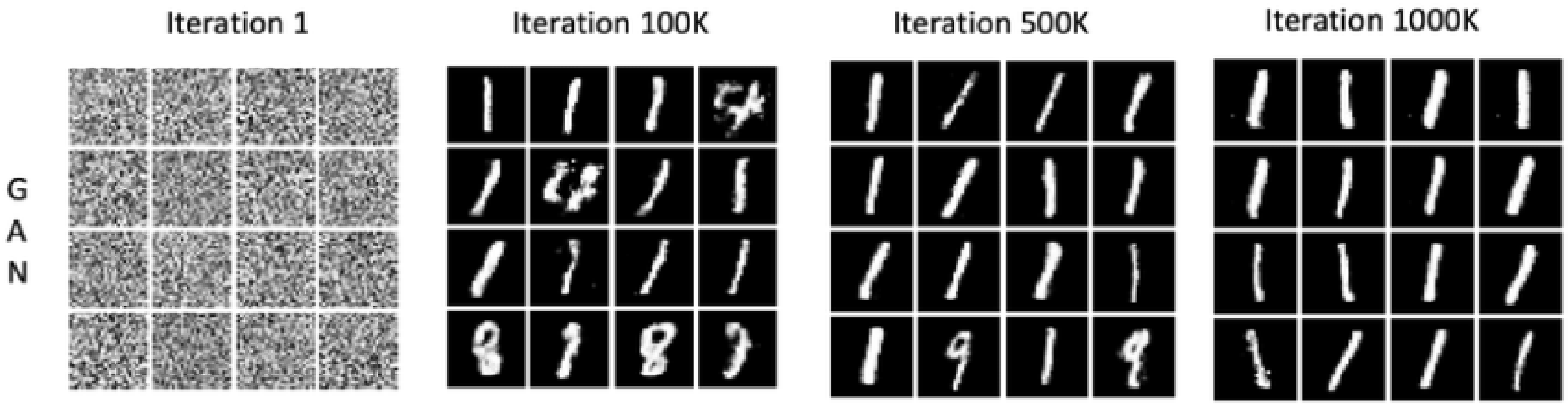
In practice, training rarely converges cleanly – we stop when samples look good.

Mode Collapse: Definition

Mode collapse:

- Generator finds a **few modes** (or one) that fool D
- Produces similar samples repeatedly
- Ignores large parts of data distribution

Example: on MNIST, G outputs only digit “1” variants.



Mode Collapse: Why It Happens

Reasons:

- Generator receives gradients only where D is currently 'surprised'
- Can converge to local optimum where one/few modes give enough reward
- Adversarial training is **non-convex, non-stationary**

Fixes often involve smoothing or regularizing objectives.

Techniques to Mitigate Mode Collapse

- **Mini-batch discrimination**: D sees multiple samples at once, penalizes lack of diversity
- **Feature matching**: G matches intermediate activations stats, not just D's output
- **Unrolled GAN**: approximate future discriminator updates when optimizing G
- **Ensemble/distributed training**: multiple generators or discriminators

No silver bullet; each helps in specific regimes.

Vanishing Gradients in GANs

Problem:

- If D is too good, $D(x) \ \approx \ 1, D(G(z)) \ \approx \ 0$
- Generator loss gradient becomes tiny (for original formulation)

Non-saturating generator loss helps, but issues remain.

Wasserstein GAN (WGAN) Idea

Replace JS divergence with **Wasserstein distance**:

- Provides smoother gradients even when distributions are far apart
- Requires D to be **1-Lipschitz** (enforced via weight clipping or gradient penalty)

Objective becomes:

$$\min_{G} \max_{D \in 1-Lip} \mathbb{E}_{x \sim p_{data}}[D(x)] - \mathbb{E}_{z \sim p_z}\left[D\left(G(z)\right)\right]$$

D outputs a **critic** (unbounded scalar), not probability.

WGAN-GP (Gradient Penalty)

Gradient penalty:

- Add penalty term $\lambda\left(\| \nabla_{\hat{x}} D(\hat{x}) \|_2 - 1\right)^2$ for interpolates $\hat{x}$
- Encourages gradient norm ≈ 1

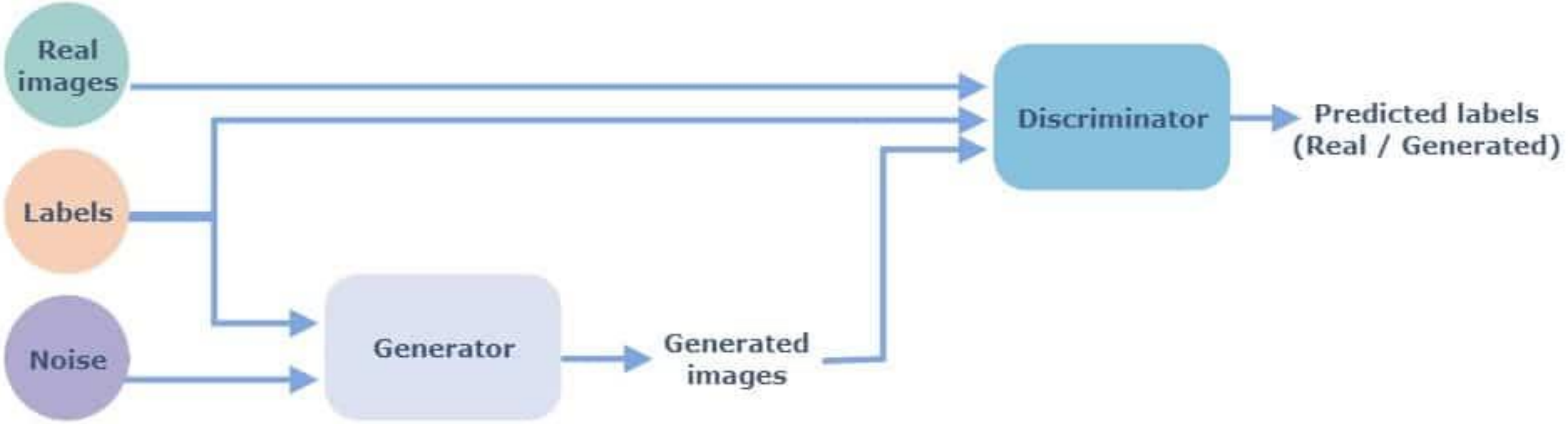
Improves stability and reduces mode collapse in practice.

Conditional GANs (cGANs)

Extend GANs with conditioning:

- Condition on label y or text embedding
- $G(z, y) \rightarrow \hat{x}$
- $D(x, y) \rightarrow$ “real or fake given y?”

Enables **class-conditional generation**, text-to-image (early forms), etc.



Conditional GAN

Evaluation of GANs

Metrics:

- Visual inspection (always necessary)
- **FID**, **IS**, precision/recall for generative models
- Downstream performance (e.g., style transfer, domain translation)

Evaluation remains partially subjective.

Strengths of GANs

- Very sharp, high-fidelity images (especially at high resolution)
- Fast sampling once trained (single forward pass)
- Strong results in conditional generation tasks (e.g., StyleGAN, BigGAN)

Weaknesses of GANs

- Training instability, hyperparameter sensitivity
- Mode collapse
- Hard to compute likelihood $p(x)$
- Hard to integrate with explicit probabilistic inference

Motivation for VAEs and diffusion models.

Recap: GAN Key Equations (Compact)

- Minimax:

$$\min_G \max_D E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

- Non-saturating G loss:

$$\max_G \quad \mathbb{E}_{z\sim p_z}\left[\log D\left(G(z)\right)\right]$$

- WGAN:

$$\min_G \quad \max_{D\in 1-Lip} \mathbb{E}_x[D(x)] - \mathbb{E}_z\left[D\big(G(z)\big)\right]$$

Transition: From GANs to VAEs

GANs:

- Implicit models, adversarial training
- No explicit likelihood; strong sampling, tricky training

VAEs:

- Explicit latent variable models, likelihood via ELBO
- Better latent structure, easier training, blurrier samples

Next: deep dive into **VAEs**.

Deep Dive 2: Variational Autoencoders (VAEs)

- Architecture (encoder/decoder)
- ELBO derivation
- Reparameterization trick
- Pros & cons vs GANs

Autoencoder Refresher

Standard autoencoder:

- Encoder $f_{\phi}(x)$ → latent z
- Decoder $g_{\theta}(z)$ → reconstruction $\hat{x}$
- Train to minimize reconstruction loss (e.g., MSE)

Drawback: no probabilistic interpretation, no generative prior on z.

VAE Architecture

VAE introduces probabilistic semantics:

- Encoder outputs parameters of $q_\varphi(z|x)$: mean $\mu(x)$, log variance $\log \sigma^2(x)$
- Sample $z \sim q_\varphi(z|x)$ via reparameterization
- Decoder outputs $p_\theta(x|z)$ (e.g., Bernoulli for pixels, Gaussian for continuous data)

Training: maximize ELBO.

VAE Generative Story

1. Sample $z \sim p(z) = N(0, I)$
2. Sample $x \sim p_\theta(x|z)$ (decoder)

This is a proper generative model: we can sample from $p(z)$ and decode.

VAE Loss: ELBO Expanded

Per data point x:

$$\mathcal{L}(x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta\,(x \mid z)] - D_{KL}\left(q_\phi(z \mid x)\ \parallel\ p(z)\right)$$

- Term 1: reconstruction (log-likelihood)
- Term 2: KL regularization

We **maximize** ELBO (or minimize –ELBO).

Gaussian Assumptions in VAE

Common choice:

- $q_{\varphi}\,(z \mid x) = \mathcal{N}(\mu(x), diag(\sigma^2(x)))$
- $p(z) \;=\; N(0, I)$
- $p_{\theta}(x|z)$ is factorized Gaussian or Bernoulli (for binary-ish pixels)

KL term has closed form for Gaussian–Gaussian.

Reparameterization Trick: Equation

Given $\mu(x), \sigma(x)$:

- Sample $\varepsilon \sim \mathcal{N}(0, I)$
- Compute $z = \mu(x) + \sigma(x) \odot \varepsilon$

Gradients wrt μ, σ propagate through z; sampling randomness is in ε.

Part 4 – VAEs: Details, Variants, and Comparison

Computing KL for Gaussian $q(z|x)$

For $q = N(\mu, \sigma^2 I), p = N(0, I)$:

$$D_{KL}(q \parallel p) = \frac{1}{2}\sum_j (\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1)$$

This term is simple to compute and differentiate.

Reconstruction Term in Practice

- For continuous x with Gaussian decoder:
  - Reconstruction loss ≈ MSE
- For binary or [0,1] x with Bernoulli decoder:
  - Reconstruction loss ≈ cross-entropy

In code, we implement – $E[\log p_\theta(x|z)]$ as `recon_loss`.

Intuition: VAE Latent Space

Because of the KL term:

- Latents for different x's are encouraged to cluster around prior $N(0, I)$
- Latent space becomes **smooth and filled**
- Interpolation between z's is meaningful

This is why VAEs are great for latent space manipulation.

VAE vs Plain Autoencoder

Plain AE:

- Can overfit: memorize x → arbitrary latent representation
- Latent space may be irregular, holes, non-smooth

VAE:

- KL regularization enforces structure in latent space
- Sacrifices some reconstruction fidelity for better generative properties.

Common VAE Issues: Blurry Outputs

Why VAEs can produce blurry images:

- Gaussian likelihood with MSE → encourages averaging many plausible reconstructions
- If data is multimodal for given z, reconstructions blur across modes

This is a core trade-off in standard VAEs.

β-VAE: Balance Reconstruction vs Regularization

β-VAE modifies objective:

$$\mathcal{L}_\beta(x) = \mathbb{E}_{q_\phi}[\log p_\theta(x \mid z)] - \boldsymbol{\beta} D_{KL}\left(q_\phi(z \mid x) \parallel p(z)\right)$$

- β > 1: stronger regularization, more disentanglement, worse recon
- β < 1: weaker regularization, better recon, less structured latent

Can encourage **disentangled latent factors**.

Hierarchical & Structured VAEs (Very Brief)

Extensions:

- Hierarchical latents: z_1, z_2,… at different levels
- Discrete latents (e.g., VQ-VAE)
- Structured priors informed by domain knowledge

These lead to powerful models (e.g., VQ-VAE + Transformers).

VQ-VAE (Conceptual)

VQ-VAE:

- Encoder maps x → continuous z_e(x)
- Quantize to nearest codebook vector e_k
- Decoder reconstructs from discrete index k

Benefits:

- Discrete latent tokens → can use **Transformers** as sequence models over indices (DALL·E-style).

VAEs vs GANs: High-Level Comparison

Aspect	VAE	GAN
Training	Likelihood via ELBO	Adversarial minimax
Latent space	Explicit, regularized, smooth	Implicit (input noise)
Sample sharpness	Often blurrier	Often sharper
Mode coverage	Usually good (less collapse)	Mode collapse possible
Likelihood	Approximate available	Not tractable

When to Use VAE vs GAN

Use VAEs when:

- You care about **latents** and representation learning
- You want **stable training** & good coverage
- You're OK with slightly blurrier outputs

Use GANs when:

- You need very **sharp images**
- You can tolerate training complexity

(In practice, later architectures mix ideas from both.)

VAE + GAN Hybrids

Some models combine VAE & GAN:

- Use VAE-style encoder/decoder
- Add GAN loss on top of reconstructions to sharpen outputs
- *Example*: VAE-GAN, adversarial autoencoders

Combines **probabilistic latents** with **adversarial perceptual loss**.

VAE Sampling vs Reconstruction

Two modes:

- **Reconstruction:** encode $x \rightarrow z$, decode $z \rightarrow \hat{x}$
- **Generation:** sample $z \sim p(z)$ and decode $\rightarrow \hat{x}$

Good generative model: both reconstructions and samples look realistic & diverse.

Diagnosing VAE Training Failures

Common issues:

- KL term collapses to 0 (posterior collapses to prior)
- Latent variables ignored (decoder too powerful)
- Poor reconstructions (mis-balanced loss terms)

Fixes:

- KL annealing (gradually increase weight)
- Limit decoder capacity or use skip connections carefully
- Tune β in β-VAE.

Summary: VAE Key Equations

- Generative model: $p_\theta(x, z)$= $p(z)\ p_\theta(x \mid z)$
- ELBO:

$$\mathcal{L}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta\,(x \mid z)] - D_{KL}\left(q_\phi(z \mid x)\ \parallel\ p(z)\right)$$

- Reparameterization: $z\ =\ \mu(x)\ +\ \sigma(x)\ \odot\ \varepsilon$

These are the pillars of VAE training.

Part 5 – GANs vs VAEs and Link to Modern Architectures

GANs vs VAEs: View from Optimization

GANs:

- Optimize via **adversarial training**
- Objective surface is moving as G and D update

VAEs:

- Optimize a **single, stable objective** (ELBO)

In practice:

- VAEs are easier to train reliably
- GANs need more tuning but can reach better perceptual quality.

GANs vs VAEs: View from Latent Spaces

- **VAE**s: z is explicitly inferred from x; approximate posterior $q_\phi(z \mid x)$
- **GAN**s: no encoder by default (but can add one)

For tasks like:

- Few-shot editing
- Conditional generation from attributes

VAEs (or encoders added to GANs) are useful.

From GANs/VAEs to Diffusion & LLMs

Conceptual links:

- Diffusion models use **score-based** training but still model $p(x)$
- VQ-VAE + Transformers (discrete latents) → DALL·E
- GANs inform how we think about adversarial training & perceptual losses

Days 2 & 3 will build on these foundations.

What You Should Be Able to Derive by Hand

You should be comfortable with:

- Writing down GAN loss and explaining each term
- Sketching derivation of D* and its relation to JS divergence
- Writing VAE ELBO and interpreting its terms
- Explaining the reparameterization trick intuitively

What You Should Be Able to Implement

By the lab / end of Day 1:

- A **simple GAN** on MNIST
- A **simple VAE** on MNIST
- Basic visual evaluation (grids, interpolations)
- Compare training curves (losses, FID-ish proxy if desired)

Lab Overview (GAN)

Lab tasks (GAN):

- Implement generator & discriminator
- Train on MNIST/Fashion-MNIST
- Track:
  - D loss, G loss
  - Visual outputs every N steps
- Try to provoke and identify **mode collapse** vs good coverage.

Lab Overview (VAE)

Lab tasks (VAE):

- Implement encoder & decoder
- Compute ELBO = recon_loss + KL_loss
- Train on MNIST/Fashion-MNIST
- Visualize:
  - Reconstructions
  - Latent space
  - Interpolations

Lab: Comparison & Short Analysis

Analysis questions:

- Compare GAN vs VAE outputs qualitatively
- Which has sharper digits? Which has broader variety?
- How does training stability differ?
- How do latent spaces differ (if you add an encoder to GAN)?

Write a short reflection.

Day 1 Technical Summary

We have covered:

- Generative vs discriminative models & key applications
- Probabilistic foundations: p(x), latent variables, ELBO
- GANs: minimax game, training, pathologies, WGAN
- VAEs: ELBO, reparameterization, latent spaces, β-VAE
- High-level comparison GAN vs VAE and links to modern models

This is the **foundational layer** for the rest of the course.

End of Day 1 Lecture – Start of Lab

Up next:

- Set up environment
- Implement **GAN + VAE**
- Visualize results, debug training, and reflect on model behavior

From now on, the best way to understand these models is to **train and break them yourself**.