GANs

Discriminative Models?

A **discriminative model** is a machine learning model that **directly learns the boundary between classes** (or directly predicts y given x).

Formally:

$P(y \mid x)$

- They **do not model how the data is generated**.
- Instead, they focus only on **separating classes** or predicting outputs.

🔑 Examples

- Logistic Regression
- Support Vector Machines (SVMs)
- Most Neural Networks used for classification (CNNs, RNNs, Transformers in classification tasks)
- Conditional Random Fields (CRFs)

🔷 How they work (intuition)

Imagine you want to classify cats vs dogs.

- A **discriminative model** looks at the features (e.g., fur length, ear shape, weight) and **draws a boundary** that best separates cats from dogs.
- It doesn't care about *how cats look overall* or *how dogs are distributed*.
- It only cares about **where the dividing line should be** in feature space.

🔷 Advantages

✅ Often achieves **high accuracy** when lots of labeled data is available.

✅ **Simpler and faster** to train than generative models.

✅ Flexible — you can plug in any features and learn directly.

🔷 Limitations

1. **Require labeled data**
    - Since they only learn $P(y \mid x)$, they need lots of labeled training examples.
    - They cannot easily leverage unlabeled data.
2. **No understanding of data distribution**
    - They don't model $P(x)$ (how the input itself is distributed).
    - So they can't **generate data**, **handle missing inputs gracefully**, or **reason about likelihood**.
3. **Poor generalization with limited data**
    - If data is scarce, they don't have prior knowledge of structure (unlike generative models, which can leverage assumptions about $P(x,y)$).
4. **Vulnerable to out-of-distribution (OOD) inputs**
    - If you give an input far from training data, they may still output a confident (but wrong) prediction.
    - Example: a cat-vs-dog classifier might confidently classify a car as "dog."

🔷 Quick comparison

- **Generative model**: Learns full joint distribution $P(x,y)$. Can generate new samples and handle missing data. Examples: Naive Bayes, Gaussian Mixture Models, GANs.
- **Discriminative model**: Learns only $P(y \mid x)$. Great at classification, but nothing more.

✅ **Summary in one line:**

Discriminative models directly learn decision boundaries ($P(y|x)$) and are great for classification, but they can't model the data distribution, struggle with limited labeled data, and fail on unseen/out-of-distribution inputs.

Generative Models

A **generative model** tries to learn the **joint probability distribution**:

$$P(x, y) = P(x \mid y)P(y)$$

- It models **how the data is generated**:
    1. Pick a label y.
    2. Generate input x conditioned on y.
- Once we know P(x,y), we can derive the classifier using Bayes' rule:
  $P(y \mid x) = \frac{P(x|y)P(y)}{P(x)}$

🔷 How do they cope with the Discriminative model limitations

1. ✅ Limited labeled data

- Since they learn **how data is distributed** (P(x ∣ y)), they can leverage **unlabeled data** to improve learning.
- Example: Semi-supervised generative models can use unlabeled images to learn P(x), then use a small amount of labels for P(y ∣ x).

2. ✅ Understanding the data distribution

- Generative models explicitly model P(x).
- This means they **know what valid data looks like**.
- Applications:
    - Detecting **out-of-distribution** samples (OOD detection).
    - **Denoising** or imputing missing data.
    - Generating synthetic samples (new data).

3. ✅ Better generalization

- By capturing the structure of data, they can generalize better when labeled data is scarce.
- Example: A Gaussian Naive Bayes classifier can perform well even with few training points, because it assumes a generative structure (features conditioned on class follow Gaussians).

---

4. ✅ Out-of-distribution robustness

- Since they know what the data distribution P(x) looks like, they can recognize when an input **doesn't belong** to the training distribution.
- Example: A discriminative cat-vs-dog classifier might label a car as "dog," while a generative model can say, "This sample doesn't look like cats or dogs I know."

🔷 Examples of Generative Models

- **Classic (probabilistic):**
  - Naive Bayes
  - Gaussian Mixture Models (GMM)
  - Hidden Markov Models (HMM)
- **Modern (deep learning):**
  - Variational Autoencoders (VAEs)
  - Generative Adversarial Networks (GANs)
  - Diffusion Models

🔷 Tradeoff

- **Discriminative models** → better at classification when you have *lots of labeled data*.
- **Generative models** → more flexible, can work with less labeled data, handle OOD detection, and can generate data... but often harder to train and less accurate in pure classification tasks.

✅ **Summary in one line:**

Generative models overcome the main weaknesses of discriminative models by learning the **data distribution itself** (P(x,y)), which lets them use unlabeled data, detect outliers, generate samples, and generalize better with scarce labels.

Implicit Density Estimation Problem

1. Background: Density Estimation

- In probability and machine learning, **density estimation** means trying to learn the probability distribution p(x) that generates the data.
- Example: if we have handwritten digit images, we'd like to know the probability distribution over all possible digit images.

There are two broad ways to do this:

- **Explicit Density Estimation**: We define a probability model with a tractable form and learn its parameters.
    - Example: Gaussian Mixture Models (GMMs), Autoregressive models, Normalizing Flows.
    - They allow exact evaluation of p(x).
- **Implicit Density Estimation**: We don't explicitly define or compute p(x). Instead, we **learn a model that can sample from the distribution**, even if we can't write down its probability function.

---

2. What is the Implicit Density Estimation Problem?

- In **implicit models** (like GANs), we can generate samples that *look like* the data.
- However, we **don't have direct access to the probability density function p(x)**.
- That means:
    - We **can't evaluate** how likely a given point is under the learned model.
    - We **can't compute likelihood-based metrics** directly.

    - Training becomes harder because we can't just maximize log-likelihood.

---

3. Example: GANs

- GANs define a generator G(z) that transforms noise z $\sim$ p(z) into samples x = G(z).
- This gives us a way to sample from $p_\theta(x)$, but we **can't compute the probability density** of any particular x.
- The discriminator helps indirectly guide the generator without computing likelihoods.

---

4. Why is it a "Problem"?

- Because without explicit density:
    - It's harder to measure how good the model is (no exact log-likelihood).
    - It's harder to combine with probabilistic reasoning tasks (e.g., anomaly detection).
    - Training relies on surrogate losses (e.g., adversarial loss, divergence minimization).

✅ **Key takeaway**:

- **Explicit models** = "I can *describe* the probability distribution."
- **Implicit models** = "I can't describe it, but I can *mimic* it and generate samples from it."

✅ **Summary in Simple Words**:

The **implicit density estimation problem** is that some generative models (like GANs) can generate data that looks realistic, but **cannot tell you the actual probability of that data**. They only learn to **mimic** the distribution, not to describe it mathematically.

Generative Adversarial Networks (GANs)

1. Core Idea

GANs introduce a clever *game* between two neural networks:

- **Generator (G):** Learns to produce fake samples that resemble real data.
- **Discriminator (D):** Learns to distinguish between real data and fake (generated) data.

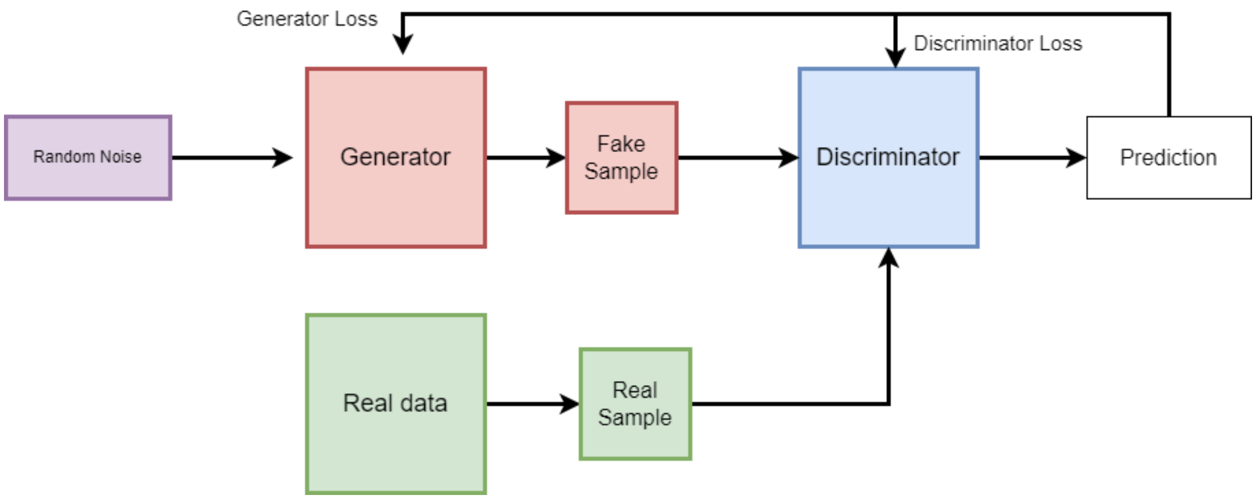
They are trained together in a **minimax game**:

- G tries to fool D.
- D tries to catch G.

This competition pushes the generator to produce increasingly realistic samples.

---

2. Architecture



1. Generator (G)

The **generator's job** is to take in random noise and transform it into realistic-looking data (images, audio, text, etc.).

🔶 Input

- A random vector z $\sim$ p(z).
- Typical choice: Gaussian $\mathcal{N}(0, I)$ or Uniform distribution.

- Dimension: usually much smaller than the output dimension (e.g., $z \in \mathbb{R}^{100}$ for an image of size 64 × 64 × 3).

🔶 Layers

- **Fully connected layers (MLPs)** → map noise to a higher-dimensional representation.
- **Convolutional layers (in image GANs)**:
  - Use **transposed convolution (a.k.a. deconvolution)** to progressively upsample from a small feature map to a large image.
  - Example: start from a 4 × 4 feature map → upsample to 8 × 8 → 16 × 16→ 64 × 64.
- **Batch Normalization** → stabilizes training.
- **Activation functions**:
  - ReLU or LeakyReLU in hidden layers.
  - **Tanh** in the final layer (so pixel values fall in [−1,1]).

🔶 Output

- A fake data sample x_{fake} = G(z).
- Example: 64 × 64 × 3 RGB image.

---

2. Discriminator (D)

The **discriminator's job** is to classify whether input data is real (from dataset) or fake (from generator).

🔶 Input

- A data sample x (either real or fake).
- Example: 64 × 64 × 3 image.

🔶 Layers

- **Convolutional layers (for images)**:
  - Reduce spatial size (downsampling) while increasing depth.

    - Example: 64 × 64 × 3 → 32 × 32 × 64 → 16 × 16 × 128.
- **Batch Normalization** → sometimes avoided in the first layer to avoid gradient issues.
- **Activation functions**:
    - LeakyReLU (to prevent dying ReLU problem).

🔶 Output

- A single probability value D(x) ∈ [0,1].
- Represents the likelihood that x is real.
- Uses **Sigmoid activation** at the last layer.

---

3. Training Interaction

GAN training is a **two-player minimax game**:

- The **Discriminator (D)** tries to classify samples correctly (real = 1, fake = 0).
- The **Generator (G)** tries to fool D (make fake samples look real).

1️⃣. **Discriminator Loss**

The discriminator's job:

- Maximize the probability of classifying real data as real.
- Maximize the probability of classifying fake data as fake.

Mathematically:

$$L_D = -\left(\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]\right)$$

- First term: reward D for outputting high probability on **real data**.
- Second term: reward D for outputting low probability on **fake data**.

👉 Training update: **minimize** L_D w.r.t. discriminator's weights.

(Equivalently, maximize the log-likelihood of correct classification.)

2️⃣. **Generator Loss**

The generator's job:

- Fool the discriminator into believing generated samples are real.

Two versions exist:

(a) **Original GAN Loss (Minimax)**

$$L_G = \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

- Generator minimizes this.
- But problem: if D is too strong early, D(G(z)) $\approx$ 0, log saturates → **vanishing gradients**.

(b) **Non-saturating Loss (Practical Version)**

$$L_G = -\mathbb{E}_{z\sim p(z)}[\log D(G(z))]$$

- Equivalent to maximizing $\log D(G(z))$.
- Gives stronger gradients when D is good.
- This is the **standard choice in practice**.

👉 Training update: **minimize** L_G w.r.t. generator's weights.

3️⃣. Training Algorithm (High-level)

1. Sample a minibatch of **real data** $x \sim p_{data}(x)$.
2. Sample random noise z $\sim$ p(z).
3. Generate fake samples G(z).
4. Update **Discriminator**:
   - Minimize L_D (real → 1, fake → 0).
5. Update **Generator**:
   - Minimize L_G (make D think fake is real).
6. Repeat steps until convergence (or until samples look realistic).

> **~** : This symbol is called a "tilde". In this context, it means **"is drawn from"** or **"is distributed as"**. It indicates that the variable on the left is a random sample from the probability distribution on the right.

4️⃣. **Intuition Recap**

- **Discriminator Loss** = “How well am I at telling real vs fake?”
- **Generator Loss** = “How well am I at fooling D into thinking fake = real?”
- Training is a **tug-of-war**:
  - If **D is too strong**, G can’t learn (no gradient).
  - If **G is too strong**, D becomes useless.
  - Balance is key → that’s why GAN training is tricky.

5️⃣. Summary:

- **Discriminator Loss**:

$$L_D = -\left[\log D(x) + \log(1 - D(G(z)))\right]$$

- **Generator Loss** (practical version):

$$L_G = -\log D(G(z))$$

---

4. Data Flow Summary

1. Noise z → Generator G(z) → Fake sample x_{fake}.
2. Real sample x or fake sample x_{fake} → Discriminator D(x).
3. D outputs probability of real vs fake.
4. Losses update both networks in opposite directions.

---

5. Architecture Variants

- **Vanilla GAN (MLPs only)** → early versions.

- **DCGAN (Deep Convolutional GAN)** → uses conv & deconv layers for images, much more powerful.
- **WGAN (Wasserstein GAN)** → stabilizes training using Wasserstein distance.
- **Conditional GAN (cGAN)** → conditions G and D on labels (e.g., generate a cat vs a dog).
- **StyleGAN** → advanced architecture for photorealistic faces.

✅ **Intuition Recap**:

- Generator = **artist** (paints fake images from imagination).
- Discriminator = **art critic** (judges whether the painting is real or fake).
- Training = **competition** that makes the artist improve until the critic can't tell real from fake.

---

3. Objective Function

The GAN loss is a **minimax optimization**:

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

- **Discriminator maximizes:** classify real as 1, fake as 0.
- **Generator minimizes:** fool D (make D(G(z)) $\approx$ 1).

---

4. Training Dynamics

- Step 1: Train **D** to better separate real vs fake.
- Step 2: Train **G** so that its fake samples get better at fooling D.
- Alternate between D and G updates.
- Ideally, the system converges when $p_{model}(x) = p_{data}(x)$ (generator distribution matches real distribution).

---

5. Advantages of GANs

✅ Can generate very realistic samples (images, music, text).

✅ No need for explicit likelihood computation (solves the implicit density problem).

✅ Highly expressive since G is a neural net.

---

6. Limitations of GANs

⚠ Training instability (non-convex minimax optimization).

⚠ Mode collapse: G produces limited diversity (e.g., always generates similar faces).

⚠ Hard to evaluate progress quantitatively (no likelihood).

---

7. Real-world Applications

- Image synthesis (e.g., **DeepFake**, StyleGAN).
- Text-to-image models (when combined with transformers).
- Data augmentation.
- Super-resolution (making images sharper).
- Art and creative content generation.

✅ **Intuition Summary**:

GANs are like a **counterfeit money maker (Generator)** competing with a **police detective (Discriminator)**.

- The generator wants to make fake currency that looks real.
- The discriminator wants to spot fakes.
- Over time, both get better → the generator eventually produces “fakes” that are indistinguishable from real data.
