Generative Adversarial Networks (GANs) - A Detailed Guide

By : A.Mohammed Ismail
23090359

Github: https://github.com/ashiq0503/Generative-Adversarial-Networks-GANs---A-Comprehensive-Tutorial.git

Introduction to GANs

Generative Adversarial Networks (GANs) are a type of deep learning model that can generate highly realistic synthetic data. They were first introduced by **Ian Goodfellow** and his colleagues in **2014**. GANs work by learning patterns in real data and using that knowledge to create similar but entirely new examples.

Overview of GAN Structure

A **Generative Adversarial Network (GAN)** consists of two key components:

1. **The Generator** – This neural network is responsible for creating synthetic data that resembles real-world data. The goal of the generator is to improve its outputs until they become indistinguishable from real data.
2. **The Discriminator** – This neural network acts as a classifier, distinguishing between real data and the generator's synthetic outputs. The discriminator is trained to penalize the generator whenever it produces unrealistic data.

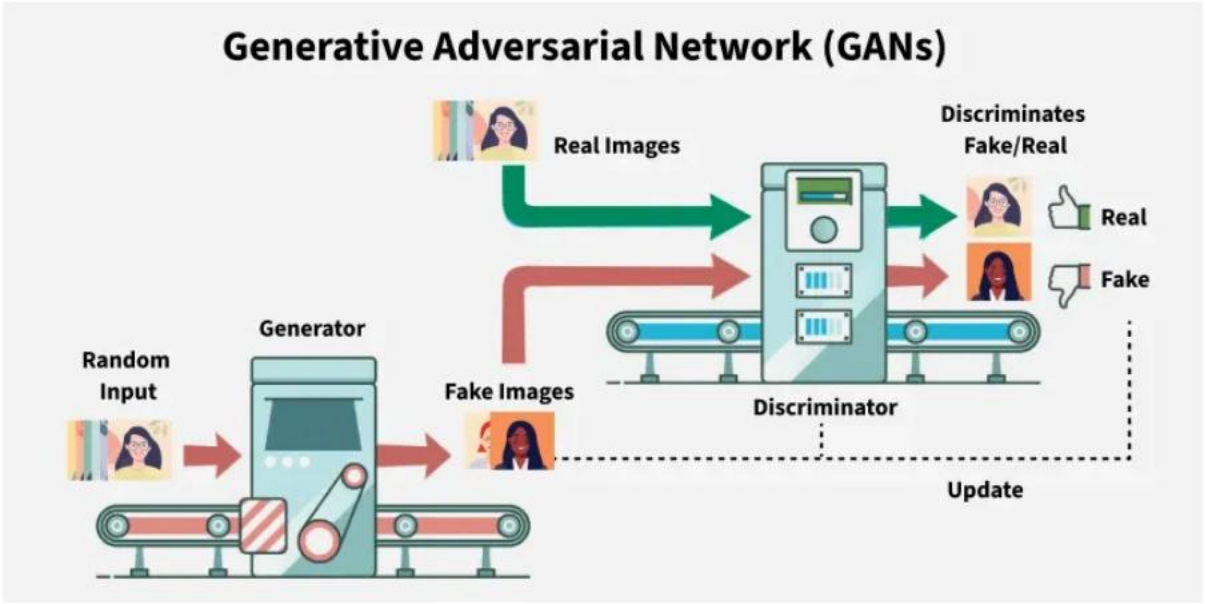

Figure 1: Generative Adversarial Network (GAN)

Training Process

Initially, the generator produces poor-quality outputs that are easily identified as fake by the discriminator.

- **Early Stage:** The generator's initial attempts result in outputs that are clearly different from real data. The discriminator can easily classify them as "fake" and real-world samples as "real."
- **Progressive Improvement:** As training continues, the generator refines its outputs to resemble real data more closely. The discriminator still identifies most synthetic data as fake, but some outputs start becoming more convincing.
- **Final Stage:** If training is successful, the generator produces high-quality outputs that closely mimic real data. At this point, the discriminator struggles to differentiate between real and generated data, leading to decreased classification accuracy.

System Architecture

A GAN operates through a structured adversarial framework:

1. **Real Data Input:** Real-world images or data are fed into the discriminator.
2. **Random Input:** A random noise vector is provided as input to the generator.
3. **Generator:** The generator processes the random input and attempts to create realistic synthetic data.
4. **Discriminator:** Both real and generated data samples are passed to the discriminator, which classifies them as either "real" or "fake."
5. **Loss Calculation:** The system computes separate loss values for both the discriminator and the generator, which are then used to adjust their respective weights through backpropagation.

This adversarial process continues iteratively, driving the generator toward producing highly realistic data while challenging the discriminator to improve its classification capabilities. The ultimate goal is for the generator to produce data so convincing that the discriminator can no longer distinguish between real and synthetic samples.

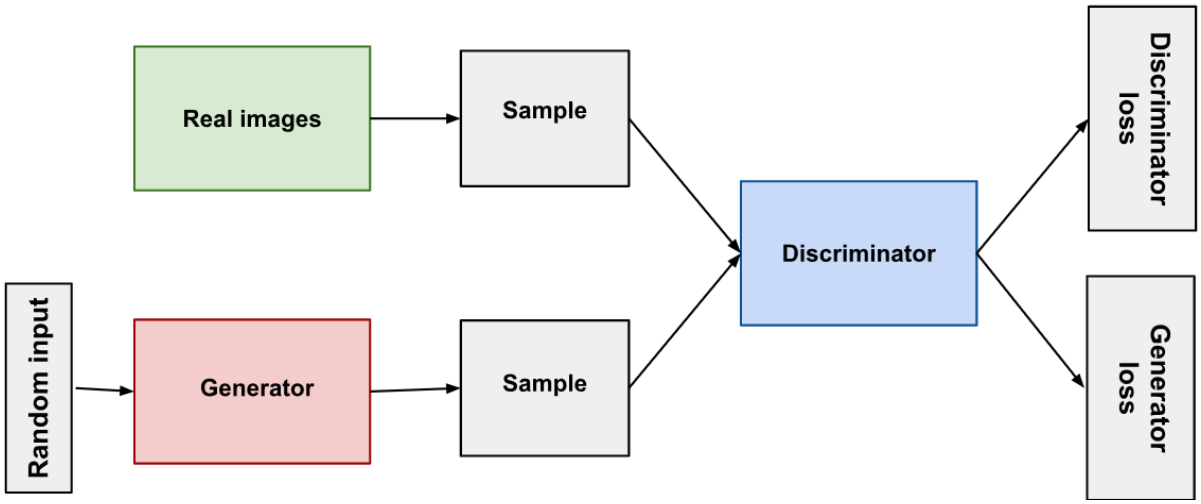

Figure 2: Whole System working

How GANs Work

GANs consist of two key neural networks that compete against each other:

1. **Generator (G):**
    - The generator's role is to **create synthetic data** from **random noise**.
    - Its objective is to **fool the discriminator** into believing that the generated data is real.
    - Over time, it improves its ability to generate data that is **indistinguishable** from real data.
2. **Discriminator (D):**
    - The discriminator acts as a **binary classifier** that evaluates whether the data it receives is real (from the dataset) or fake (from the generator).
    - Its job is to **detect fake data** as accurately as possible.
    - As the generator improves, the discriminator also gets better at identifying fake data.

The **generator and discriminator are in constant competition**, where the generator tries to fool the discriminator while the discriminator learns to detect fake samples. This process helps GANs generate **highly realistic outputs** over time.

Detailed GAN Architecture

1. Generator Model

The generator is a **deep neural network** that takes in **random noise** and generates **realistic-looking data** (such as images or text). It learns patterns from real data through **backpropagation** and **gradient descent**.

The **loss function** for the generator is defined as:

$$J_G = -\frac{1}{m}\sum_{i=1}^{m} \log D(G(z_i))$$

Where:

- J_G measures how well the generator is fooling the discriminator.
- $D(G(z_i))$ represents the probability that the discriminator classifies the generated sample as real.
- The generator **aims to minimize** this loss so that the discriminator **classifies its outputs as real** (value close to 1).

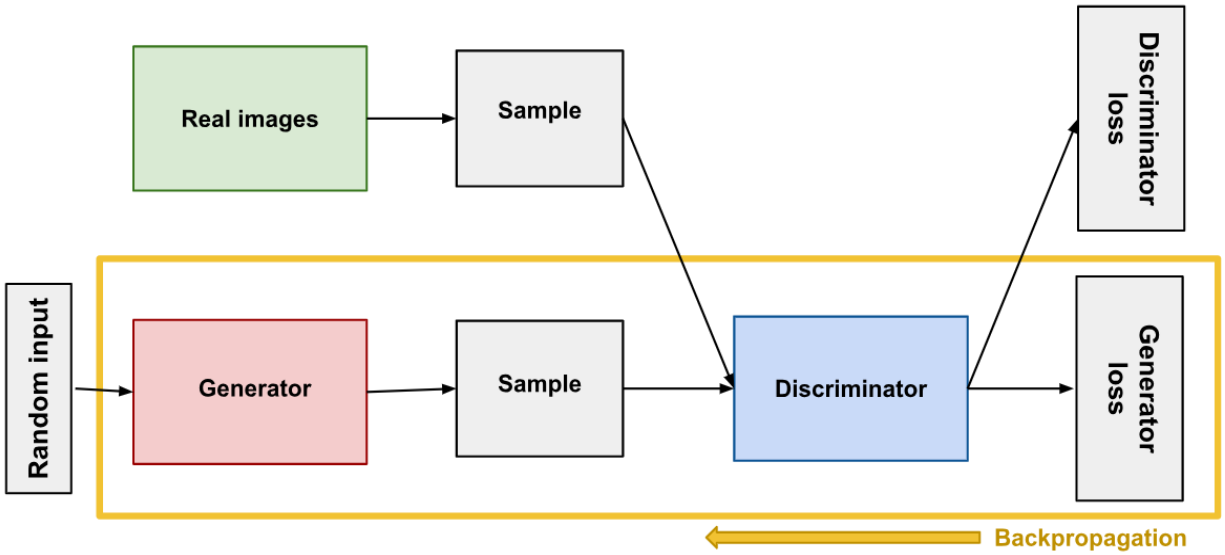

Figure 3:Generator Model

2. Discriminator Model

The discriminator is also a **deep neural network** that classifies input samples as either **real or fake**. It trains by improving its ability to **differentiate between real and generated samples**

The **loss function** for the discriminator is given by:

$$J_D = -\frac{1}{m}\sum_{i=1}^{m}\log D(x_i) - \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z_i)))$$

Where:

- J_D measures the discriminator's accuracy in classifying real vs. fake samples.
- $D(x_i)$ represents the probability that the discriminator correctly identifies **real data**.
- $D(G(z_i))$ represents the probability that the discriminator **incorrectly classifies** generated data as real.
- The discriminator **aims to maximize** this loss, improving its classification ability.

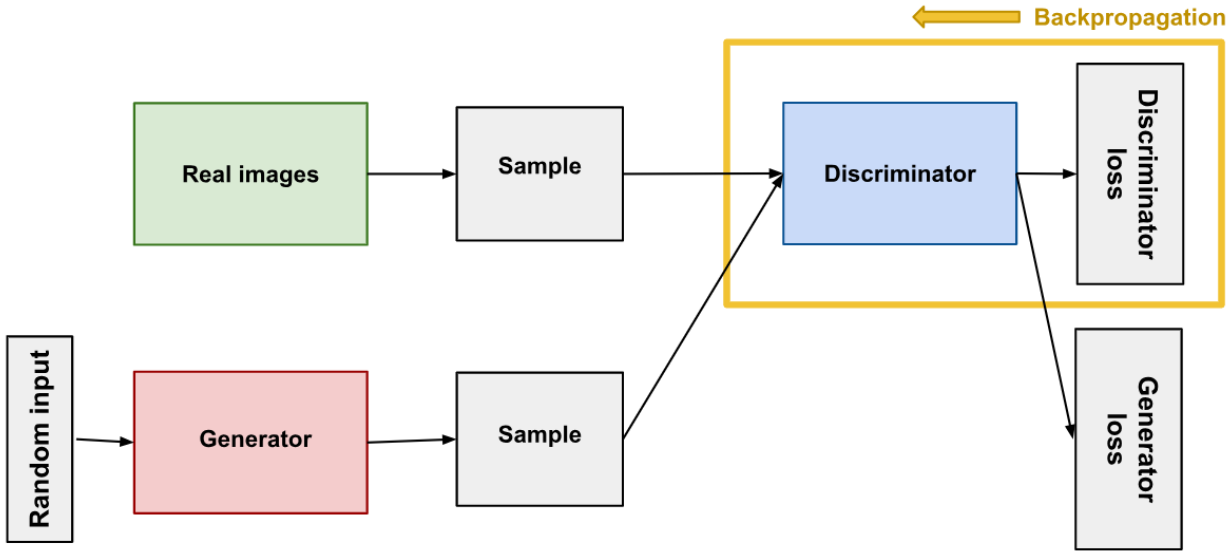

Figure 4: Backpropagation in discriminator training.

Training a GAN

Since a Generative Adversarial Network (GAN) consists of two separate neural networks—the generator and the discriminator—its training process presents two main challenges:

1. **Dual Training Process** – The generator and discriminator require different training approaches.
2. **Convergence Difficulty** – Determining when the GAN has properly converged is not straightforward.

Alternating Training Strategy

To effectively train both networks, GANs follow an alternating training cycle:

1. The **discriminator** is trained for one or more epochs while keeping the generator unchanged.
2. The **generator** is then trained for one or more epochs while keeping the discriminator unchanged.
3. These steps are repeated iteratively to refine both networks over time.

During the discriminator's training phase, it learns to distinguish real data from generated (fake) data. Since the generator is fixed during this step, the discriminator can focus on identifying the generator's weaknesses.

Likewise, during the generator's training phase, the discriminator remains unchanged. This stability prevents the generator from constantly adapting to a moving target, which could hinder its ability to improve.

This alternating process is what enables GANs to tackle complex generative tasks. The training starts with a simple classification problem (distinguishing real vs. fake), which gradually evolves into a more challenging generative task. However, if the discriminator fails to differentiate between real and random data generated from the start, the GAN training cannot progress effectively.

Understanding Convergence

As the generator improves, the discriminator's task becomes more difficult, as the fake data it evaluates starts resembling real data more closely. Ideally, if the generator reaches a point where it perfectly mimics real data, the discriminator's accuracy drops to 50%, essentially making random guesses.

This presents a challenge for GAN convergence. If training continues beyond this point, the discriminator's feedback becomes less useful, leading to poor guidance for the generator. As a result, the generator may begin learning from unreliable signals, which can degrade its performance rather than improve it.

In GANs, convergence is often temporary rather than a stable endpoint, requiring careful monitoring to prevent training from deteriorating.

Loss Functions in GAN Training

GANs aim to replicate probability distributions, so their loss functions must measure the difference between the **real** and **generated data distributions**. There are multiple ways to do this, but two commonly used loss functions include:

1. Minimax Loss

- Introduced in the original GAN paper, this loss function treats training as a minimax game.
- The **generator tries to minimize** the discriminator's ability to detect fake data, while the **discriminator tries to maximize** its ability to distinguish real from fake.

GANs use a **minimax optimization** approach where the generator tries to **minimize** the discriminator's success rate, while the discriminator tries to **maximize** it. The objective function is:

$$\min_G \max_D V(G, D) = E_{x \sim P_{data}(x)}[\log D(x)] + E_{z \sim P_z(z)}[\log(1 - D(G(z)))]$$

Where:

- G represents the generator network.
- D represents the discriminator network.
- $P_{data}(x)$ represents real data distribution.
- $P_z(z)$ represents noise distribution (usually normal or uniform distribution).

2. Wasserstein Loss

- A more stable loss function, introduced in a 2017 paper, that **reduces mode collapse** and improves convergence.
- Used in **Wasserstein GAN (WGAN)**, it optimizes the difference in probability distributions more smoothly than minimax loss.

Separate Loss Functions for Generator and Discriminator

GANs usually have **two loss functions**: one for the generator and one for the discriminator. Both originate from a single formula measuring the probability distribution difference, but:

- The **discriminator loss** considers both **real and fake data distributions**.
- The **generator loss** only focuses on the fake data distribution, ignoring real data terms.

This approach allows both networks to work towards refining the generated outputs while maintaining a structured adversarial training process.

Types of GANs

1. **Vanilla GAN:**
    - The simplest form of GAN with **basic multi-layer perceptrons (MLPs)**.
    - Often struggles with **unstable training** and **mode collapse**.
2. **Conditional GAN (CGAN):**

- Uses **conditional labels** to guide data generation.
- Helps in **targeted** synthetic data generation (e.g., generating images of specific objects).

3. **Deep Convolutional GAN (DCGAN):**
   - Uses **Convolutional Neural Networks (CNNs)** instead of MLPs.
   - Helps generate **high-quality images** with spatial understanding.
4. **Laplacian Pyramid GAN (LAPGAN):**
   - Uses a **multi-resolution approach** to generate **high-resolution images**.
5. **Super-Resolution GAN (SRGAN):**
   - Designed to **increase image resolution** and improve clarity.
   - Used in **image upscaling** to remove blurriness and enhance details.

Real World Applications of GANs

- **Image Generation:** Creating realistic human faces (e.g., ThisPersonDoesNotExist.com)
- **Data Augmentation:** Generating new training data for machine learning models
- **Style Transfer:** Converting sketches into realistic photos
- **Super-Resolution:** Enhancing image quality in low-resolution images
- **AI Art & Deepfakes:** Generating realistic AI art and face-swapping videos

References :

1. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. **Improved techniques for training GANs**. *Advances in Neural Information Processing Systems, 29 (NIPS 2016)*. Available at: https://arxiv.org/abs/1606.03498 [Accessed 26 March 2025].
2. Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F. and Zheng, Y., 2019. **Recent progress on generative adversarial networks (GANs): A survey**. *IEEE Access*, 7, pp.36322-36333. DOI: 10.1109/ACCESS.2019.2905015.
3. Alqahtani, H., Kavakli-Thorne, M. and Kumar, G., 2021. **Applications of generative adversarial networks (GANs): An updated review**. *Archives of Computational Methods in Engineering*, 28, pp.525–552. DOI: 10.1007/s11831-019-09388-y.

Note: Implementation is on Notebook attached