**Generative Adversarial Network**

***Architecture:***

* **Generator**
  1. **Function**: The generator creates synthetic data (e.g., images) from random noise or a latent space.
  2. **Training Goal**: Its goal is to generate data that is as similar as possible to real data to fool the discriminator.
* **Discriminator:**
  1. **Function**: The discriminator evaluates data to determine whether it is real (from the training dataset) or fake (generated by the generator).
  2. **Training Goal**: Its goal is to correctly classify data as real or fake.

***Interaction Between Generator and Discriminator:***

* **Training Process:**
  1. **Generator’s Role:** During training, the generator produces a batch of synthetic data.
  2. **Discriminator’s Role:** The discriminator receives both real data and synthetic data from the generator. It tries to distinguish between the real and fake data.
  3. **Loss Calculation:** Based on the discriminator’s feedback (Loss function is the discriminator’s error), the generator’s objective is to maximize the likelihood that the discriminator classifies its synthetic data as real. Conversely, the discriminator aims to minimize the error in distinguishing between real and fake data.
* **Adversarial Training:**
  1. This process is adversarial because the generator and discriminator have opposing goals. The generator improves its ability to create realistic data, while the discriminator becomes better at detecting synthetic data.
  2. They are trained simultaneously, with the generator improving its outputs based on the feedback from the discriminator and the discriminator improving its accuracy in distinguishing data.

***Architectural Setup:***

* **Separate Networks:** They are typically implemented as separate neural network architectures:
  + **Generator:** Often a deep neural network or a convolutional network that takes random noise and outputs data in the desired format.
  + **Discriminator:** Often a separate deep neural network or convolutional network that takes data (real or generated) and outputs a probability score indicating the likelihood of the data being real.

***Example Architecture Setup:***

* **Simple GAN Setup:**
  1. **Generator Network:** Takes a random noise vector z as input and outputs synthetic data G(z).
  2. **Discriminator Network:** Takes data x (either real or synthetic) as input and outputs a probability D(x) indicating the likelihood that x is real.
* **High-level conceptual flow**
  1. **Input Noise:** z (random noise vector) → **Generator** → Synthetic Data G(z)
  2. **Real Data:** x\_real→ **Discriminator** → Probability D(x\_real)
  3. **Synthetic Data:** G(z) → **Discriminator** → Probability D(G(z))

***Latent Space***

* **Definition:** 
  1. **Latent Space:** A high-dimensional space where each point represents a compressed representation of data. It’s often a lower-dimensional space compared to the original data space.
  2. **Latent Vector:** A point in the latent space. In GANs, this is usually a vector of random values that the generator network uses as input.
* **Purpose in GANs:** 
  1. **Generator Input:** The generator network takes a latent vector (random noise) as input and transforms it into synthetic data (e.g., images). This latent vector encodes the features that the generator will use to create new data.
  2. **Exploring Latent Space:** By varying the latent vector, you can generate different outputs. The latent space is structured so that similar vectors produce similar outputs, allowing for smooth interpolation between different types of generated data.
  3. **Example:** Imagine you’re generating images of faces. The latent space might have dimensions that correspond to various abstract features (e.g., age, gender, hairstyle). By navigating this space, you can generate faces with different characteristics based on the position in the latent space.
* **Notes: (Interpretable vector math in Latent Vector Space)**
  1. There will be smooth transition along the latent space (vectors close together, when used as inputs, will generate similar fake images)  
     This is because, there is an innate mapping learned by the generator from the latent space to the fake generated image output space
  2. This also means, the directions movable in the latent space, the axis, are meaningful as certain features of the images have been extracted and “understood” by the generator, thus creating some form of structure (one axis represents a feature of the image/output) (enabling the smooth transition within the latent space, similar vectors = similar looking images (based on any attributes)), eg one axis for color of hair, one axis for size of nose, ...etc.
  3. That means there can be latent vector manipulation to generate a specific type of output/image. For example, simple arithmetic operations (or more complicated ones), can be used to generate a specific type of images, now that there is a structure in the latent space and the axis are feature representative   
     (eg: in the latent space, one axis = men, one axis = women, one axis = smiling, I can take men + smiling latent vector – neutral man latent vector + neutral women latent vector and then pass this latent vector as an input to the generator, I can get women + smiling images)

***Notes:***

\*\*\* In an ideal and perfect scenario, the end-game is to:

Generator: Creates perfectly indistinguishable fake images from the real images, the probability distribution of noise vector inputs = probability distribution of the real data\

Discriminator: Outputs a probability of 0.5 when asked to distinguish real and fake images

**High-level Overview of GAN’s structure**

A diagram of a computer game

Description automatically generated

**A diagram of a graph

Description automatically generated with medium confidence**

* This is the structural flow of GAN
* **z** = noise vector; **x** = original data vector; **y** = label (0 for fake, 1 for real);  
  **Theta(g)** = weight parameters for the Generator network, G;  
  **Theta(d)** = weight parameters for the Discriminator network, D;  
  **Pz** = Probability distribution of the noise vectors;  
  **Pdata** = Probability distribution of the real data;  
  **Pg(X)** = Probability distribution of the generated data fake data, G(z) which follows a similar probability distribution to the probability distribution of the real data Pdata(x) as it is trying to fool the Discriminator;  
  **G(z)** = the output of the generator (fake generated data);  
  **P(input is real data)** = the output of the discriminator, D(G(z)) or D(x), which is the probability that the input data is real.

*\*\*\* Key Notes\*\*\**

**1*. Real data’s probability distribution, Pdata***

* **Pdata is not explicitly known.** 
  + In practice, you do not know the exact probability distribution of the real data. (e.g. the probability distribution of height for the adult population) Instead, you have access to a dataset consisting of samples from this distribution (training data are samples from the entire population)
* **You don’t determine Pdata explicitly at the start for the objective function**
  + Instead, you rely on your dataset, which provides a finite set of samples from Pdata
* **Usage in GAN**
  + The GAN’s training process involves the generator trying to produce data that the discriminator cannot distinguish from these real samples, effectively learning the structure of Pdata indirectly.
* **Key (approximation in Objective Function)**
  + That is why the Expectation term in the objective function, Ex~Pdata, cannot be calculated explicitly and will be approximated using simple summation (Monte Carlo Approximation)  
    [See Training GAN algorithm part]

***2. Noise’s probability distribution, P(z) -> Noise = random input vectors to generator***

* **P(z) is typically chosen and explicitly known beforehand.** 
  + You define the noise distribution before training begins. (or define the latent space)
  + distribution should be easy to sample from and have enough flexibility to allow the generator to learn complex mappings from noise to data*.*
  + simple, well-known distribution from which you can easily sample. The most common choices are:
    - Standard Normal Distribution: ~N(0, 1)
    - Uniform Distribution: ~U(-1, 1)
* **Usage in GAN**
  + The generator takes random samples from this noise distribution and transforms them into data points that resemble those from the real data distribution. The choice of ​P(z) is crucial because it defines the latent space from which the generator creates new samples.
* **Key (approximation in Objective Function)**
  + The Expectation term in the objective function, Ez~P(z), will be approximated using simple summation (Monte Carlo Approximation)  
    [See Training GAN algorithm part]
  + Not the integration formula for expectation of random variables with continuous probability distribution

**Understanding the Minimax Objective Function**

A screenshot of a computer game

Description automatically generated

***Terms Defined and Explained***

1. **Pdata**: True distribution of the real data (we want the GAN to learn and generate from)
2. **P(z)**: Distribution of noise( GAN’s generator samples random noise or latent vector). These noise vectors serve as the initial input to the generator, which it then transforms into synthetic data (e.g., images)
3. **Theta d**: Discriminator network’s weight parameters
4. **Theta g**: Generator network’s weight parameters
5. **D Theta d ()**: Discriminator’s output function (Probability of input being real)
6. **G Theta g ()**: Generator’s output function (generated fake data vector)
7. **Ex~Pdata**: Expectation with respect to real data, x, with x being picked from probability distribution of real data, Pdata
8. **Ez~P(z)**: Expectation with respect to noise generator-input data, z, with z being picked from probability distribution of noise data, P(z)
9. **Ex~Pdata (log D Theta d (x))**: Real data expectation
   1. What it means: This represents the expected value (average) of   
      log D Theta d(x) over all possible real data points, x, sampled from the real data distribution, Pdata
   2. Purpose: It measures how well the discriminator, D, is at identifying real data as real. The goal is to maximize this term, so the discriminator gets better at recognizing real data.
10. **Ez~P(z) (log (1 - D Theta d (G Theta g (z))**: Generated data expectation
    1. What it means: This represents the expected value (average) of   
       log(1 - D(G(z))) over all possible generated data points G(z), where z is sampled from the noise probability distribution P(z)A group of math equations

       Description automatically generated



* The structure of the objective function is similar to the Binary Cross entropy Function
  + Here, we use the structure of the Binary Cross Entropy Loss Function for the discriminator as it is essentially a binary classifier, trying to classify real images as 1 and fake images as 0.
    - Thus, for real images x, predicted output = D(x), label = 1, when we sub into BCE, we get SUM[ -ln(D(x)) ], or - EX [ ln(D(x)) ]
    - Thus, for the fake images G(z), predicted output = D(G(z)), label = 0, when we sub into BCE, we get SUM[ -ln(1-D(G(z))) ], or  
      -EX[ -ln(1-D(G(z))) ]
    - Sum of these 2 cases, Loss Function for Discriminator to minimize  
      = - { EX [ ln(D(x)) ] + EX[ -ln(1-D(G(z))) ] }
  + Here, the negative sign in Binary Cross Entropy Loss Function (to minimize in binary classification problems) is removed (\* -1) in the relation to GAN’s objective function as we are to first maximize the loss function, which is minimizing the -1 \* loss function.
    - Thus Loss Function for Discriminator to maximize   
      = EX [ ln(D(x)) ] + EX[ -ln(1-D(G(z))) ], which is the correct predictions of both real and fake data
    - And since Generator wants to minimize the prediction of Discriminator, GAN’s total mission is too:  
      Min (theta G) Max (theta D) { EX [ ln(D(x)) ] + EX[ -ln(1-D(G(z))) ] }

A blackboard with math equations

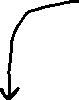
Description automatically generated

* The expectation accounts for the summation term in the original Binary Cross Entropy Function where we are finding the average value of the terms across all possible values of the primary input terms from their respective probability distributions.

**Gradient Ascent/Descend for updating weights of Discriminator and Generator**

A screenshot of a computer game

Description automatically generated



- Since the generator, weights theta G is only in G theta g in EX[ -ln(1-D(G(z))) ], Min (theta G) { EX [ ln(D(x)) ] + EX[ -ln(1-D(G(z))) ] } is equivalent to:  
Min (theta G) { EX[ -ln(1-D(G(z))) ] } for the generator

A screenshot of a math game

Description automatically generated



**Inner Loop, Training the Discriminator first for k steps**

A white paper with black text

Description automatically generated

A white paper with black text and black text

Description automatically generated

A white paper with black text and black text

Description automatically generated



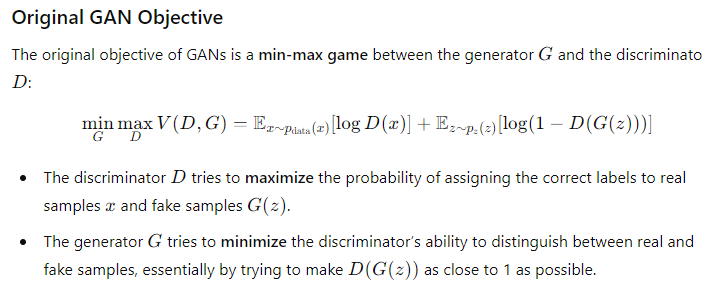
**Training the Generator, once per k training Discriminator steps in the inner for loop**

A white page with black text and symbols

Description automatically generated

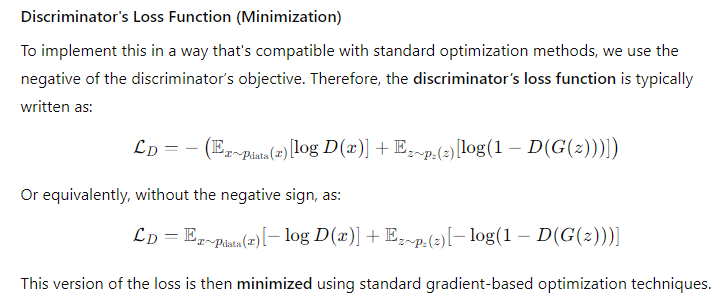


**KEY (WHAT ACTUAL HAPPENS IN PRACTICE), Summary**



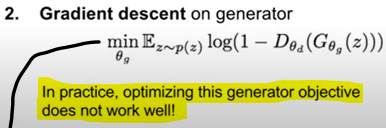
**Discriminator:**

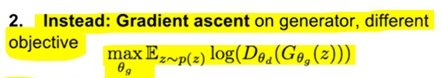
- Actually perform minimizing of Loss function (modify to \* -1 to be exact same as in BCE), maximizing was only to help understanding “Max correct predictions”

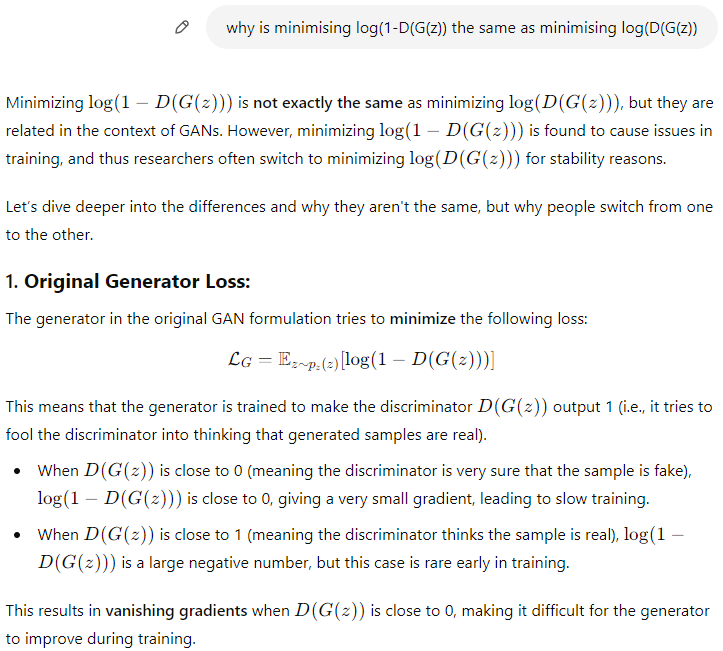


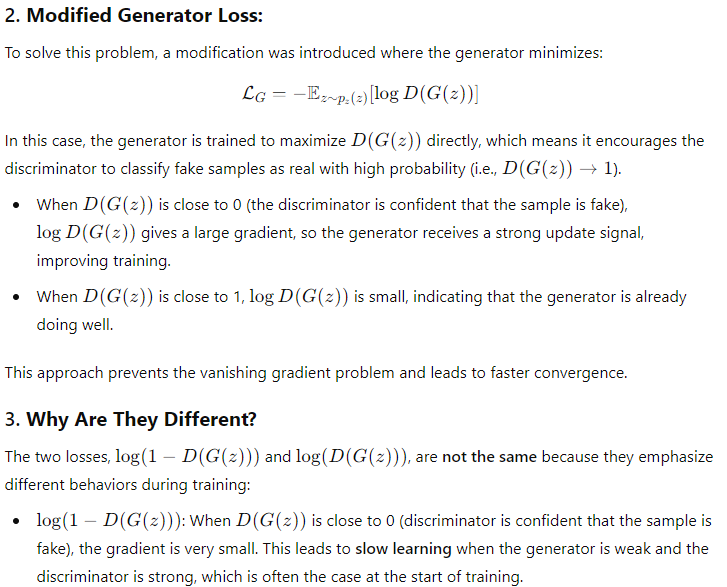
**Generator**:

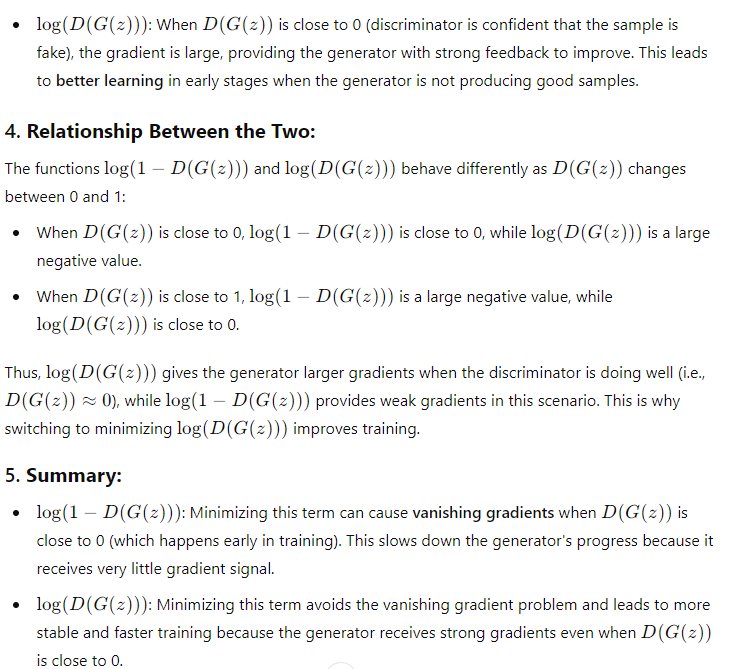
- Maximize, perform gradient ascent to improve training (modified Loss function)



****

****

****

****

**GAN Training Algorithm**

A screenshot of a computer game

Description automatically generated

**Improved GAN (with Convolutional Architecture)**

A screenshot of a computer

Description automatically generated

**More on GAN**

**A diagram of a diagram of a network

Description automatically generated with medium confidence**

**A screenshot of a computer screen

Description automatically generated**

**A close-up of a web page

Description automatically generated**

**A white background with black text

Description automatically generated**