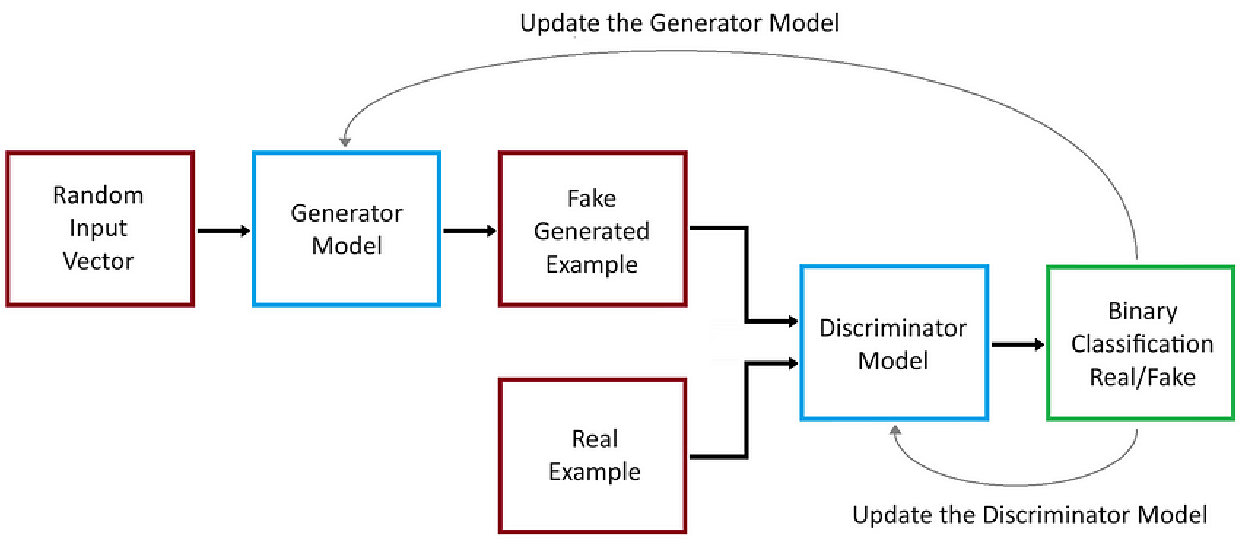
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

Generative Adversarial Networks

The GAN architecture was first described in the 2014 paper by Ian Goodfellow, et al. titled "Generative Adversarial Networks." Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.
A Generative Adversarial Network (GAN) consists of two neural networks, namely the Generator and the Discriminator, which are trained simultaneously through adversarial training.


Update the Generator Model
Random Input Vector
Generator Model
Fake Generated Example
Real Example
Discriminator Model
Binary Classification Real/Fake
Update the Discriminator Model

1. Generator: This network takes random noise as input and produces data (like images). Its goal is to generate data that's as close as possible to real data.
2. Discriminator: This network takes real data and the data generated by the Generator as input and attempts to distinguish between the two. It outputs the probability that the given data is real.

During training, the Generator tries to produce data that the Discriminator can't distinguish from real data, while the Discriminator tries to get better at differentiating real data from fake data. The two networks are in essence competing in a game: the Generator aims to produce convincing fake data, and the Discriminator aims to tell real from fake. This adversarial process leads to the Generator creating increasingly better data over time.

Internally GAN has two neural networks that compete with each other. The goal of the generator network is to deceive the discriminator network and the goal of the discriminator network is to correctly identify if the input is real or fake.

Generator:

- **Input Layer**: The generator starts with an input layer that takes in a random noise vector, usually sampled from a normal or uniform distribution.
- **Fully Connected Layers**: Early in the network, fully connected layers may be used to transform the input noise vector into a suitable shape for further processing.
- **Batch Normalization**: This technique is often used between layers to stabilize learning by normalizing the output of a previous layer. It helps in addressing issues like mode collapse and aids in faster convergence.
- **Activation Functions**: ReLU (Rectified Linear Unit) or Leaky ReLU are common choices for activation functions in the generator. These functions introduce non-linearity to the model, enabling it to generate complex data.
- **Transposed Convolutional Layers**: Also known as deconvolutional layers (though technically a misnomer), these layers are fundamental in the generator. They upsample the input from the previous layer to a higher spatial dimension, effectively doing the opposite of what convolutional layers do in a CNN.
- **Reshaping Layers**: These layers are used to reshape the data into the desired output format, especially when generating images.
- **Output Layer**: The final layer usually employs a tanh or sigmoid activation function, depending on the nature of the data being generated. For image generation, a tanh function is often used to output pixel values in a normalized range.
- **Training**: During training, the generator continuously learns to produce data that the discriminator can't distinguish from real data. The generator updates its weights based on feedback from the discriminator, refining its ability to create convincing fake data.

Output: The generator produces data as its output. In the case of a GAN designed for image generation, the output would be an image. The data is then passed to the discriminator to be classified as real or fake.

Training: During training, the generator tries to deceive the discriminator by producing data that the discriminator can't distinguish from real data.

The generator updates its weights based on feedback from the discriminator. If the discriminator correctly identifies the generated data as fake, the generator adjusts its weights to produce more convincing data during the next iteration.

The process is a kind of "game" where the generator constantly tries to improve, aiming to eventually produce data that's nearly indistinguishable from real data.

Discriminator:

The architecture of the discriminator often mirrors that of traditional convolutional neural networks (CNNs), but with some adjustments. It usually consists in the sequence of the following components.

- **Convolutional Layers**: These layers are fundamental in processing image data. They help in extracting features from the input images. The number of convolutional layers can vary depending on the complexity of the data.
- **Batch Normalization**: This is sometimes used between layers to stabilize learning by normalizing the input to a layer.
- **Activation Functions**: Leaky ReLU is a common choice for activation functions in the discriminator. It allows a small gradient when the unit is not active, which can help maintain the gradient flow during training.
- **Pooling Layers**: Some architectures use pooling layers (like max pooling) to reduce the spatial dimensions of the input data progressively.
- **Fully Connected Layers**: At the end of the network, fully connected layers are used to process the features extracted by the convolutional layers, culminating in a final output layer.
- **Output Layer**: The final layer is typically a single neuron with a sigmoid activation function to output a probability value.
- **Training**: During training, the discriminator updates its weights in the following way: It is shown real data and should be trained to output values close to 1. It is shown fake data (generated by the generator) and should be trained to output values close to 0.

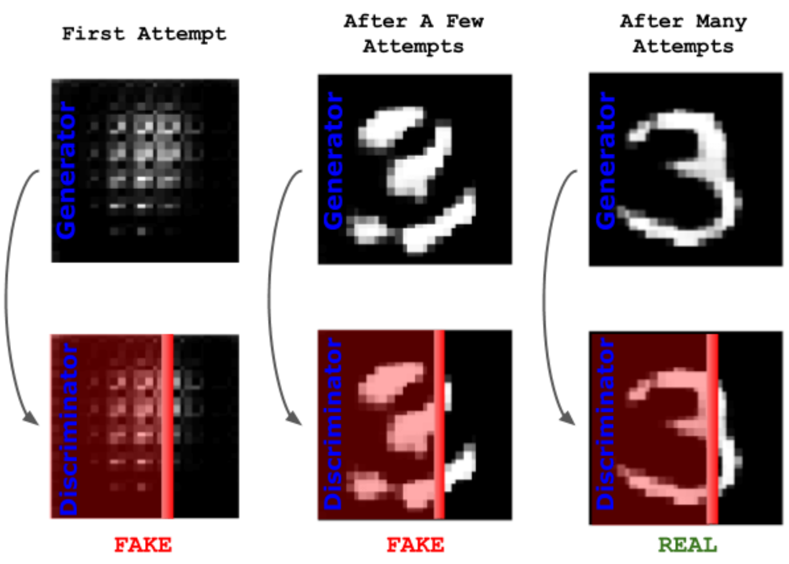
GAN Loss

Generative Adversarial Networks (GANs) utilize loss functions to train both the generator and the discriminator. The loss function helps adjust the weights of these models during training to optimize their performance. Both the generator and the discriminator use the binary cross-entropy loss to train the models, that can be written as

$$L(y,p) = -(y\log(p) + (1-y)\log(1-p))$$

where:

- $L(y,p)$ is the loss value;
- y is the true label (either 0 or 1);
- p is the predicted probability of the sample belonging to class 1.


First Attempt
After A Few Attempts
After Many Attempts
Generator
Generator
Generator
Discriminator
Discriminator
Discriminator
FAKE
FAKE
REAL