GANS: Generative Adversarial Networks

A generative adversarial network (GAN) is a class of machine learning frameworks . Two neural networks (Generator and Discriminator) compete with each other like in a game. This technique learns to generate new data using the same statistics as that of the training set, given a training set. It is basically an implicit generative model. First let us look at the types of generative models.

TYPES OF GENERATIVE MODELS

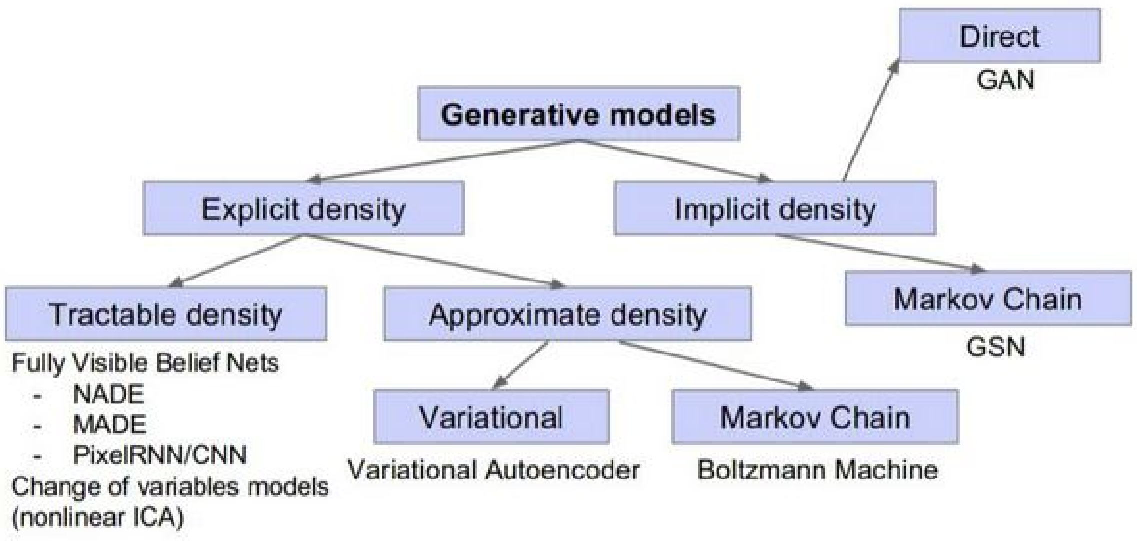
There are basically 2 types of generative models

- Explicit Density Models

Assumes some prior distribution about the data
Example: Maximum Likelihood

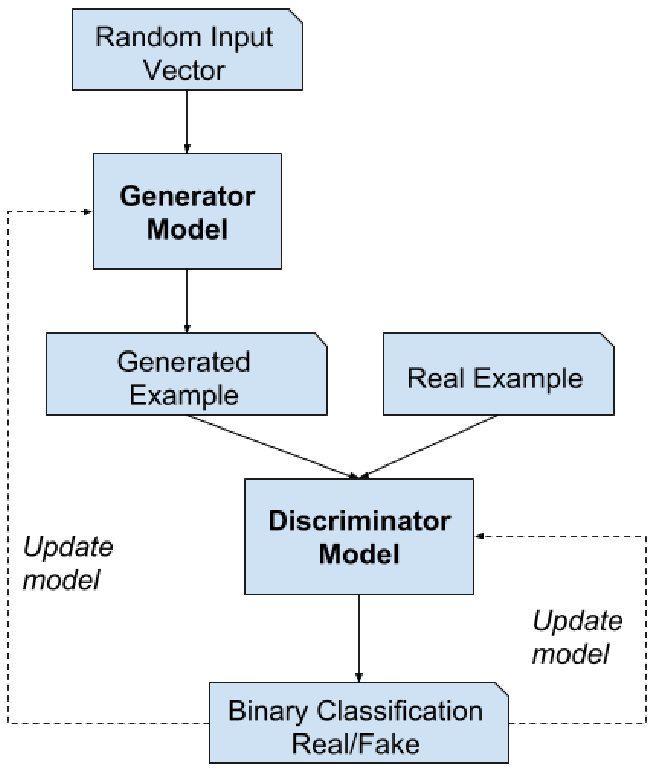
- Implicit Density Models

Defines a stochastic procedure that directly generates data
Example: Generative Adversarial Networks (GANs)



GANs don’t work with any explicit density estimation like Variational Autoencoders. Instead, it is based on a Game Theory approach with an objective to find Nash equilibrium between the two networks, Generator and Discriminator.

GAN’s ARCHITECTURE



GANs are composed of two different networks

- GENERATIVE NETWORK

The aim of the generator network is to generate images given some random noise as input. This can be a simple neural network or CNN .We pass some random noise and this network generates an image using that noise.

- DISCRIMINATOR NETWORK

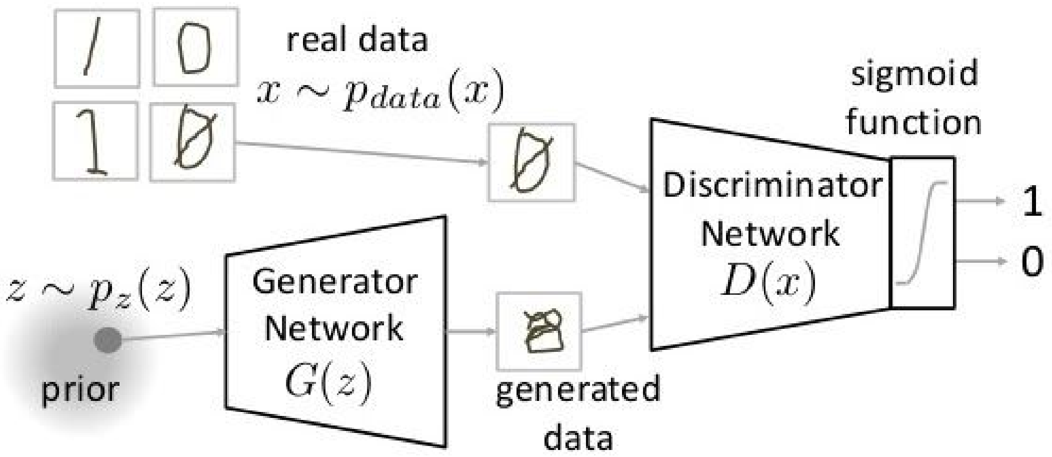
It identify whether the input is real or fake i.e it’s task is to perform binary Classification.

The advantage of this model is that it is completely differentiable. Since both generator and discriminator are neural networks (or convolutional neural networks), we get a completely differentiable network.

HOW DO GANs WORK

By now we have a basic overview of the GAN model . Let's look at the complete mathematics on how GANs works.
Before we dive into the derivation ,let's derive some parameters and variables.

$$
\begin{aligned}
&D = \text{Discriminator}\\
&G = \text{Generator}\\
&\theta_d = \text{Parameters of discriminators}\\
&\theta_g = \text{Parameters of generator}\\
&P_z(z) = \text{Input noise distribution}\\
&P_{data}(x) = \text{Original data distribution}\\
&P_g(x) = \text{Generated distribution}
\end{aligned}
$$



The loss function described in the paper can be derived from binary cross entropy loss
function below.

$$L(\hat{y}, y) = [y.log\hat{y} + (1 - \underset{\substack{\uparrow \\ \text{Original data}}}{\boxed{y}}).log(1 - \underset{\substack{\uparrow \\ \text{Reconstructed data}}}{\boxed{\hat{y}}})]$$

While training discriminator , the label of data coming from $p_{data}(x)$ is $y = 1$ (real data) and $\widehat{y} = D(x)$ so:

$$L(D(x), 1) = log(D(x))$$

And for data coming from generator , the label is y=0(fake data) and $\widehat{y}$ =D(G(z)) so:

$$L(D(G(z)), 0) = log(1 - D(G(z)))$$

Now we know that the objective of the discriminator is to correctly classify the fake and real dataset.For this, the above equations should be maximized because both D(x) and D(G(x)) are probabilities and between 0 and 1 and thus maximizing these 2 will give D(x)=1 which implies that the it correctly classifies the data that is coming from original data $p_{data}(x)$ and D(G(x)) =0 i.e all the data that is coming from generator is classified 0 by the discriminator i.e fake data. So final loss function for the discriminator can be given as,

$$L^{(D)} = \max[log(D(x)) + log(1 - D(G(z)))] \tag{3}$$

And the generator is competing against discriminator. So, it will try to minimize the equation (3) and loss function is given as,

$$L^{(G)} = \min[log(D(x)) + log(1 - D(G(z)))] \tag{4}$$

We can combine equations (3) and (4) and write as,

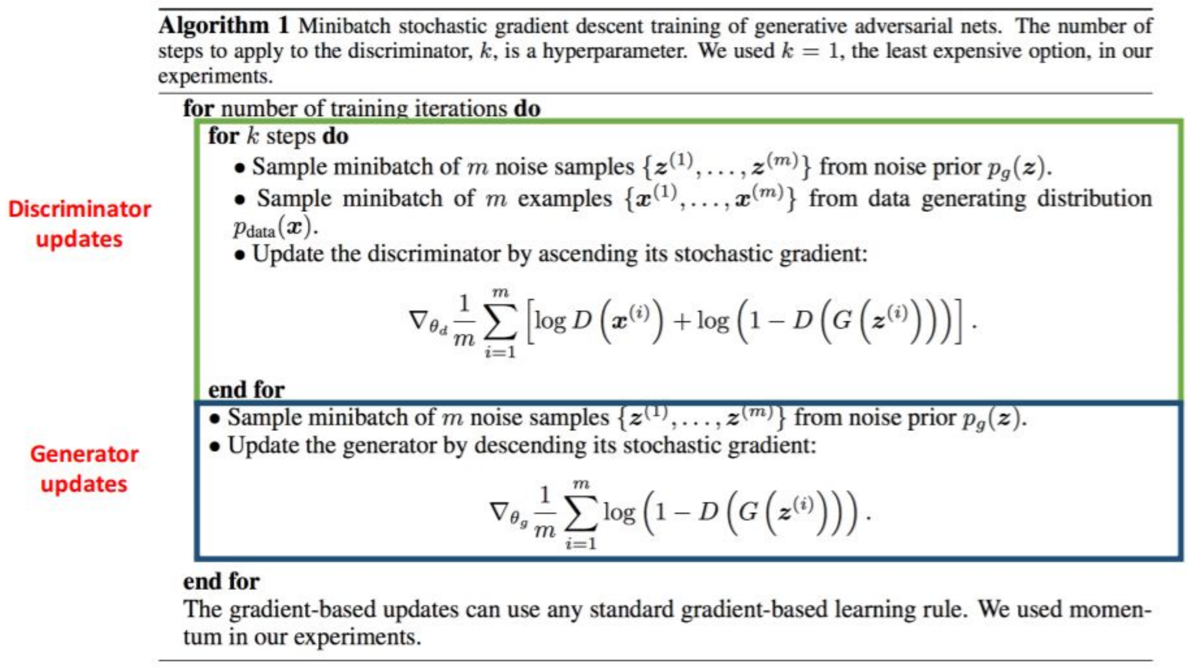
$$L = \min_{G} \max_{D}[log(D(x)) + log(1 - D(G(z)))] \tag{5}$$

We have considered the above loss function that is valid only for a single data point, to consider entire dataset we need to take the expectation of the above equation as

$$\min_G \max_D V(D,G) = \min_G \max_D \left(E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[log(1-D(G(z)))]\right) \tag{6}$$

Now let's look at the complete algorithm :

ALGORITHM


Discriminator updates
Generator updates

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

It can be noticed from the above algorithm that the generator and discriminator are trained separately. In the first section, real data and fake data are inserted into the discriminator with correct labels and training takes place. Gradients are propagated keeping generator fixed. Also, we update the discriminator by ascending its stochastic gradient because for discriminator we want to maximize the loss function given in equation (6).
On the other hand, we update the generator by keeping discriminator fixed and passing fake data with fake labels in order to fool the discriminator. Here, we update the generator by descending its stochastic gradient because for the generator we want to minimize the loss function given in equation (6).

The optimal discriminator D for any given generator G can be found by taking derivative of the loss function (equation (6)),

$$-\frac{P_{data}(x)}{D(x)} + \frac{P_g(x)}{1 - D(x)} = 0$$

$$\Rightarrow D_G^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)} \tag{7}$$

The above equation is very important mathematically but in reality, you cannot calculate optimal D as Pdata(x) is not known. Now, the loss for G when we have optimal D can be obtained by substituting equation (7) into loss function as,

$$L^{(G)} = E_{x\sim P_{data}} log\frac{P_{data}(x)}{\frac{1}{2}(P_{data}(x) + P_g(x))} + E_{x\sim P_g} log\frac{P_g(x)}{\frac{1}{2}(P_{data}(x) + P_g(x))} - 2.log2 \tag{8}$$

Now KL and Jensen-Shannon(JS) divergences are given by,

$$\Rightarrow KL(P_1||P_2) = E_{x\sim P_1} log\frac{P_1}{P_2}$$

$$\Rightarrow JS(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{P_1+P_2}{2}) + \frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$$

so,

$$L^{(G)} = 2.JS(P_{data}||P_g) - 2.log2 \tag{9}$$

The above equation reduces to -log(4) as the p_g approaches $p_{data}(x)$ because divergence becomes zero.

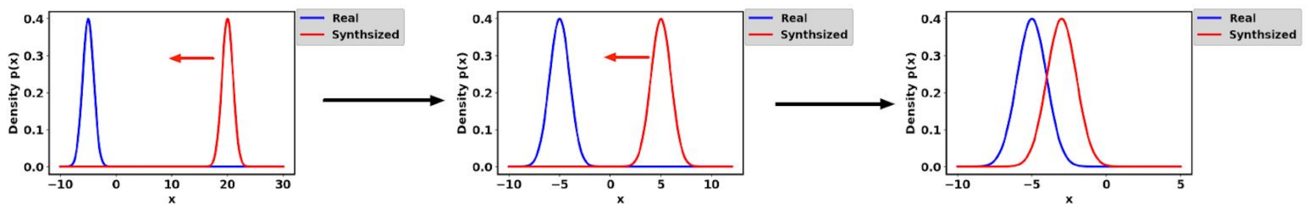
VANISHING GRADIENT PROBLEM STRIKES AGAIN !

The aim of optimization for equation (9) is to move p_g towards $p_{data}(x)$ for an optimal D. JS divergence remains constant if there is no overlap

between $p_{data}(x)$ and p_g .It can be observed that JS divergence is constant and its gradient is close to 0 when the distance is more than 5, which represents that the training process does not have any influence on G . The gradient is non-zero only when p_g and $p_{data}(x)$ have a significant overlap that means when D is close to optimal, G will face a vanishing gradient problem.
This problem can be countered by modifying the original loss function of G as,

$$\max_G E_{z\sim P_z(z)}[log(D(G(z)))] \tag{10}$$

This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning



Two normal distributions are used here for visualization. Given an optimal D, the objective of GANs is to update G in order to move the generated distribution Pg (red) towards the real distribution Pdata (blue) (G is updated from left to right in this figure. Left: initial state, middle: during training, right: training converging). However, JS divergence for the left two figures are both 0.693 and the figure on the right is 0.336, indicating that JS divergence does not provide sufficient gradient at the initial state.

EXPERIMENTS

- Datasets
    - MNIST, Toronto Face Database, CIFAR-10

- Generator model uses RELU and sigmoid activations.
- Discriminator model uses maxout and dropout.
- Evaluation Metric
    - Fit Gaussian Parzen window to samples obtained from G and compare log-likelihood.

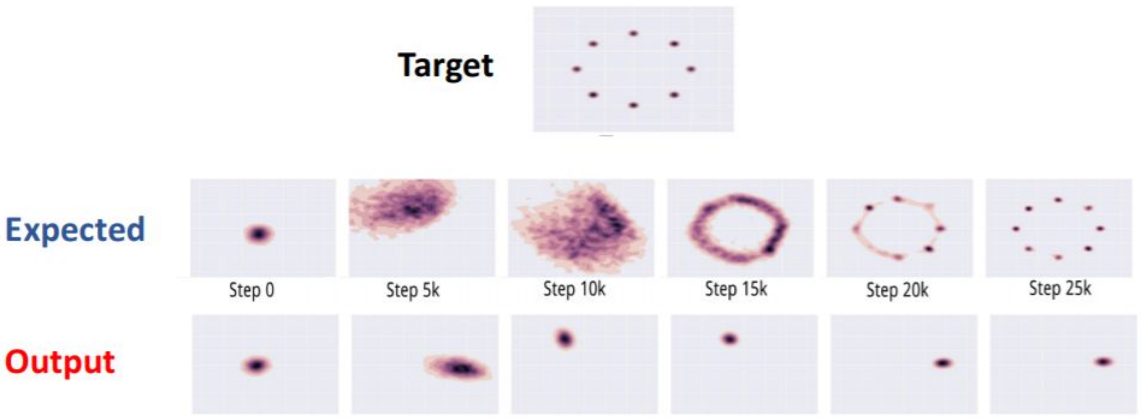
ADVANTAGES

- Computational advantages
    - Backprop is sufficient for training with no need for Markov chains or performing inference.
    - A variety of functions can be used in the model.
- Since G is trained only using the gradients from D, fewer chances of directly copying features from the true data.
- Can represent sharp (even degenerate) distributions.

DISADVANTAGES

- There is no explicit representation of $p_g(x)$
- MODE COLLAPSE

During training, the generator may get stuck into a setting where it always produces the same output. This is called mode collapse. This happens because the main aim of G was to fool D not to generate diverse output.


Target
Expected
Step 0
Step 5k
Step 10k
Step 15k
Step 20k
Step 25k
Output