Mathematics behind GANs

Discriminative Algorithms: These models classify based on the inputs to the corresponding target values.

It learns conditional probability distribution

P(y|x) :: The Probability of *y* given *x.*

It is not based upon the distribution of input. A discriminative model ignores the question of whether a given instance is likely, and just tells you how likely a label is to apply to the instance.

E.g., Logistic Regression, Decision Trees, SVMs

On the other hand,

Generative Models: Given inputs, we want to build a model that can understand the inputs to generate similar inputs and its labels from the targets.

It learns joint probability distribution. A generative model includes the distribution of the data itself, and tells you how likely a given example is. For example, models that predict the next word in a sequence are typically generative models (usually much simpler than GANs) because they can assign a probability to a sequence of words.

*P(x, y) = P(x|y)*P(y)* :: The Probability of *x* given *y.*

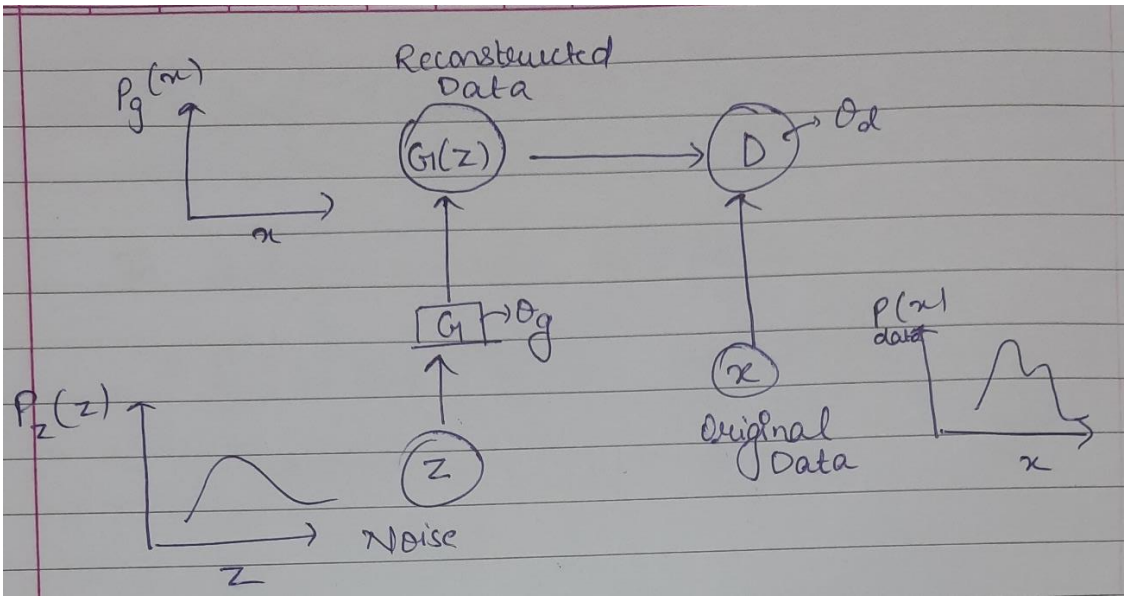
E.g., Naive Bayes

Generative Adversarial Networks (GAN) are based on the idea of two neural networks competing against each other in a way that they improve.

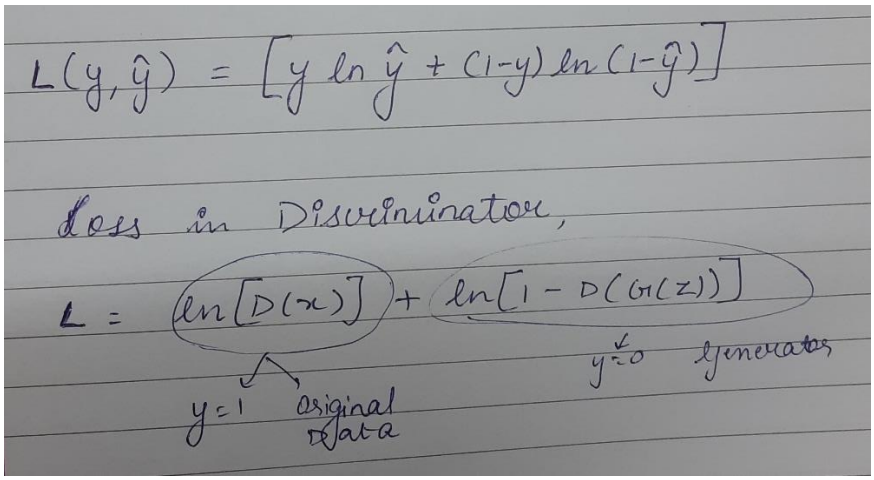
It is composed of two neural networks:

- Discriminator: Takes either fake or real images from the Generator and training set as input and correctly classifies them.
- Generator: It outputs fake data and is tasked with improving the output fake data based on the gradients received from the Discriminator.

  Interestingly, the Generator never sees any real images; instead, it gradually learns to create convincing fake images. It only receives the gradients that flow back through the discriminator. As a result, as the discriminator improves, more information about the real images is stored in these second-hand gradients, allowing the generator to make significant progress.



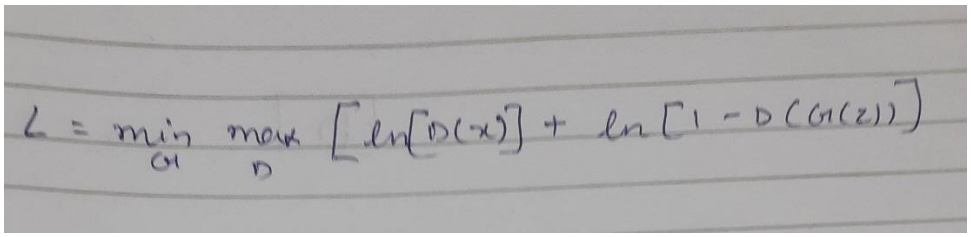
x : Original data or training sample
z : Noise vector
$P_z(z)$: Noise Distribution
$P_g(x)$: Generated Distribution
$P_{data}(x)$: Original Data distribution
G(z) : Generator's output
D(x) : Discriminator's output for the real inputs (x_{real}) : $P(y|x_{real})$
D(G(z)) : Discriminator's output for inputs by generator (x_{fake}) : $P(y|x_{fake})$



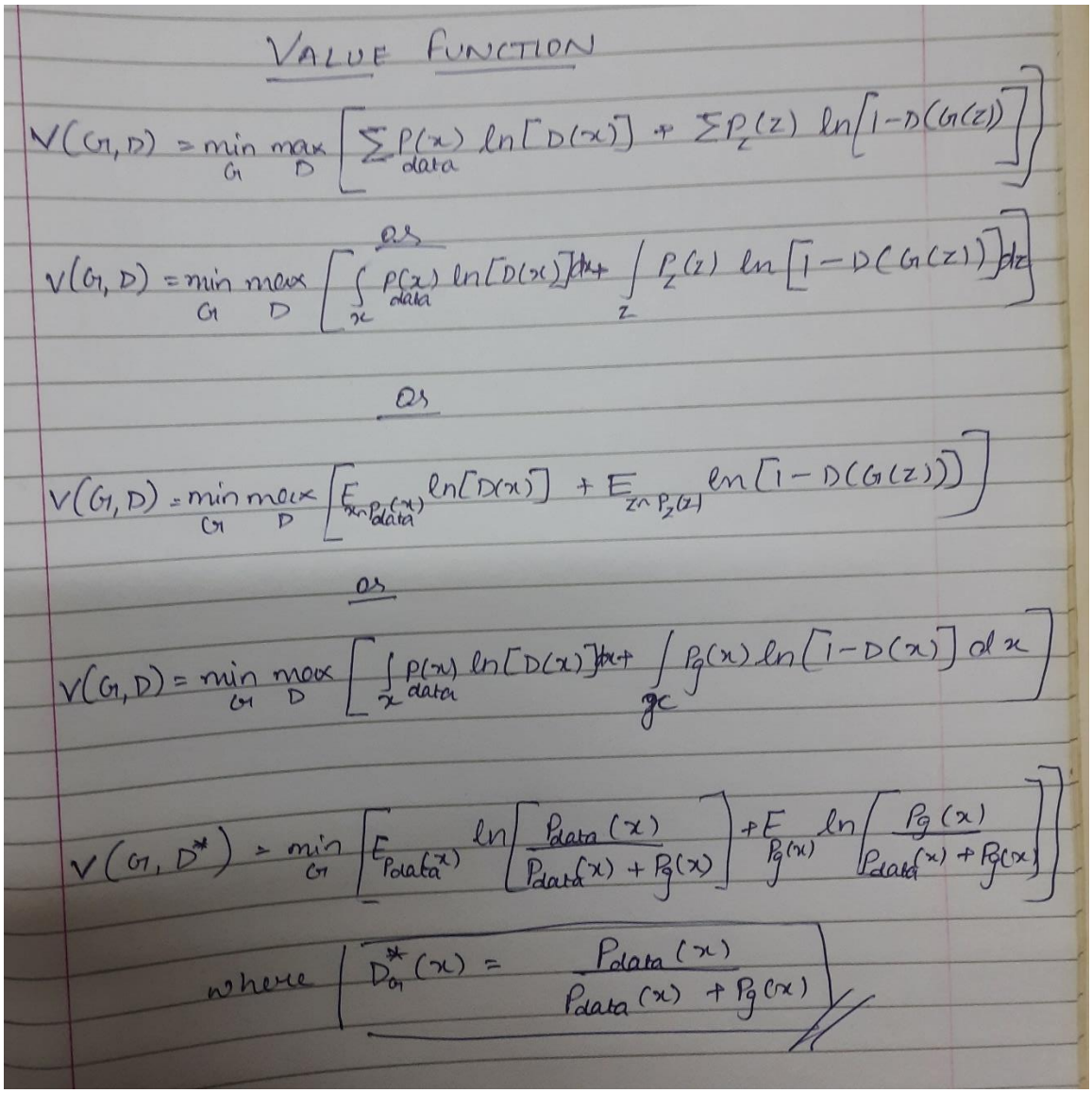
Since the Discriminator needs to classify between the fake and real, the loss function should be maximized.

And similarly, the Generator is competing against the Discriminator, it would try to minimize the function.

Combining the loss function:



The final Value Function would be for the whole all the datapoints:



The Discriminator and Generator are trained separately.

In GAN, each iteration is divided into two parts:

- Training the Discriminator: Discriminator is trained by first giving the inputs from the training set which consists of real images and also from images from the Generator which are fake. It learns to classify the images by optimization of the weights through backpropagation during this part.
  The weights of the Generator are frozen during this part.
- Training the Generator: In this part the Generator is trained in a manner such that it produces batch of fake images and these fake images classified as real are fed to the Discriminator.
  The weights of the Discriminator are frozen during this part.

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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

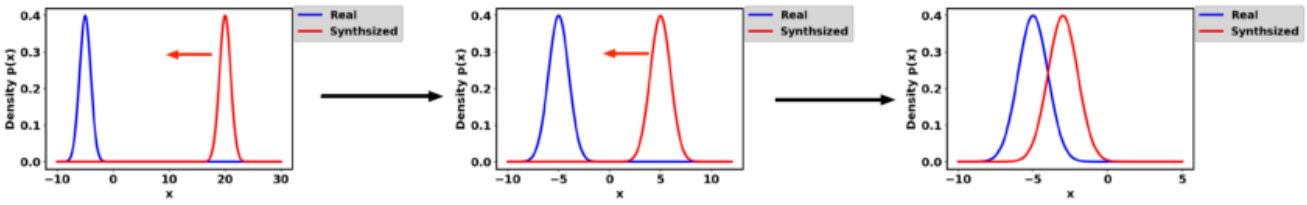
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

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

The Jensen-Shannon divergence approaches zero when $P_g(x)$ approaches $P_{data}(x)$.



Limitations of GAN

- Vanishing Gradient
- Mode Collapse

References:

https://arxiv.org/pdf/1406.2661.pdf

https://developers.google.com/machine-learning/gan

https://towardsdatascience.com/the-math-behind-gans-generative-adversarial-networks-3828f3469d9c

https://medium.com/deep-math-machine-learning-ai/ch-14-general-adversarial-networks-gans-with-math-1318faf46b43