EE698R: Advanced Topics for Machine Learning
Department of Electrical Engineering
Indian Institute of Technology Kanpur

Lecture Notes - GANs

Muskan - 190520
Videeta Sharma - 190959

Neural Network

First, we were introduced to the structure of a simple neural network.The building block of Neural Networks are neurons. A neuron is an entity that takes inputs, performs mathematical operations on them and returns an output. A simple neural network for performing regression with one hidden layer can be depicted as:

$$h_{i_1} = \sum_{i_0} (w_{i_1 i_0} x_{i_0}) + b_{i_1} \tag{1}$$

$$v_{i_1} = \sigma(h_{i_1}) \tag{2}$$

$$y_{i_3} = \sum_{i_2} (w_{i_3 i_2} v_{i_2}) + b_{i_3} \tag{3}$$

where σ is the sigmoid function, given as:

$$\sigma(h) = \frac{1}{1 + exp(-h)} \tag{4}$$

x_{i_0} are the inputs to the neural network, where $0 <= i_0 < N_0$. These inputs are fed to the hidden layer where the neurons give the output $h_{i_1}, 1 <= i_1 <= N_1$.
b_{i_1} is the bias term associated with the hidden layer and $w_{i_1 i_0}$ are the weights associated with neurons in the hidden layer. To introduce non-linearity, sigmoid function is used. The final output is calculated by using h_{i_1} as inputs, $w_{i_3 i_2}$ as weights and b_{i_3} as the bias terms associated with the output layer.
The trainable parameters are represented by $\theta = [w, b]$. Taking Mean Square error as the Loss function for this model,

$$L(\theta) = \sum_{data} \sum_{i_3} (y^d_{i_3} - y_{i_3})^2 \tag{5}$$

$$L(\theta) = \mathop{\mathbb{E}}_{(x, y_d) \sim data} [\sum_{i_3} (y^d_{i_3} - y_{i_3})^2] \tag{6}$$

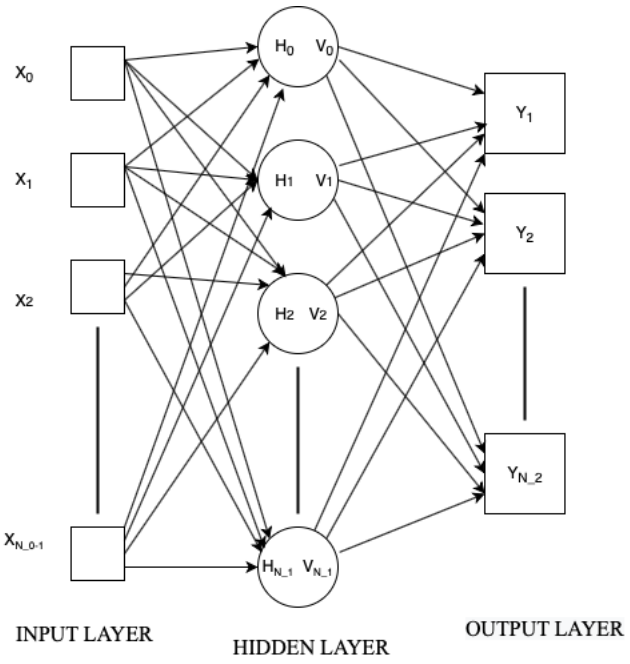


Figura 1: Depiction of Neural Network with one hidden layer.

If we want to use neural networks for Classification, the sigmoid function can be used in the last layer, instead of (3).

$$y_{i_3} = \sigma(v_{i_3}) \tag{7}$$

For performing multilabel classification, Binary Cross Entropy Loss function is used.

$$L(\theta) = -\sum_{data}\sum_{i_3}(y^d_{i_3}\log(y_{i_3}) + (1-y^d_{i_3})\log(1-y_{i_3})) \tag{8}$$

$$L(\theta) = -\mathop{\mathbb{E}}_{(x,y_{i_3})\sim data}[\sum_{i_3}(y^d_{i_3}\log(y_{i_3}) + (1-y^d_{i_3})\log(1-y_{i_3}))] \tag{9}$$

$$L(\theta) = -\sum_{i_3}(\mathop{\mathbb{E}}_{(x,y)\sim data, y^d_{i_3}=1}\log(y_{i_3}) + \mathop{\mathbb{E}}_{(x,y)\sim data, y^d_{i_3}=0}(1-\log(y_{i_3})) \tag{10}$$

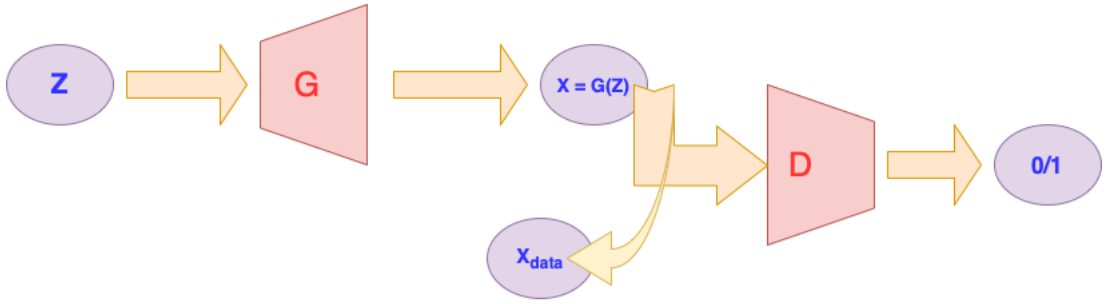
Generative Adversarial Networks

If we are given samples from a Complicated Probability Distribution $p(x)$, then to sample more values from unknown $p(x)$, we can use Generative Adversarial Networks (GANs).
It consists of a Generator and Discriminator. A generator takes samples z from a simple, known probability distribution $p(z)$ and generates x.

$$\vec{x} = G(\vec{z}) \tag{11}$$

Since G is not a bijective function, we cannot calculate the probability distribution of $\vec{x}$ by calculating Jacobian of the inverse transform. Consider θ_g as the trainable parameters of the Generator.

Now, we connect a Discriminator which takes $\vec{x}$ and $\vec{x}_{data}$ as the input and gives output 0 and 1 for them respectively. The discriminator works as the classifier that differentiates between real and false data.



STRUCTURE OF A GAN

Consider θ_d as the trainable parameters of the Discriminator represented by D. The loss function is given by:

$$L(\theta_g, \theta_d) = -[\underset{(x,y^d)\sim data, y^d=1, \vec{x}=\vec{x}_{data}}{\mathbb{E}} [\log(D(\vec{x})] + \underset{\vec{x}=G(\vec{z}), \vec{z}\sim p(\vec{z})}{\mathbb{E}} [\log(1 - D(\vec{x})]] \tag{12}$$

On simplifying this, we can write

$$L(\theta_g, \theta_d) = -[\underset{\vec{x}\sim p_{data}(\vec{x})}{\mathbb{E}} [\log(D(\vec{x})] + \underset{\vec{z}\sim p_{\vec{z}}(\vec{z})}{\mathbb{E}} [\log(1 - D(G(\vec{z}))]] \tag{13}$$

For training, the Loss Function has to be minimised with respect to θ_d, which enables the Discriminator to classify accurately, and maximised with respect to θ_g which enables the Generator to give more accurate, real-like samples. Between these 2 pursuits, maximising θ_g is treated as the priority.

Training Algorithm for GANs

- Run a loop for N iterations and perform the following steps:
  - Perform K iterations of the following:
    - Sample a minibatch of m samples, $\vec{z} \sim p_{\vec{z}}(\vec{z})$, $\vec{x} = G(\vec{z})$.
    - Sample another minibatch of m samples of $\vec{x}_{data} \sim p_{data}(\vec{x})$
    - Update parameters of Discriminator $\theta_d = \theta_d - \eta \nabla_{\theta_d} L(\theta_g, \theta_d)$.
  - Sample a minibatch of m samples of $\vec{z} \sim p_{\vec{z}}(\vec{z})$.
  - Update parameters of Generator $\theta_g = \theta_g + \eta \nabla_{\theta_g} L(\theta_g, \theta_d)$. For the generator's parameters, the second term of loss function will be considered i.e $\nabla_{\theta_g} \sum_{s=1}^{m} \log(1 - D(G(\vec{z}^s))$. The paper suggests that instead of this, we can use $-\nabla_{\theta_g} \sum_{s=1}^{m} \log(D(G(\vec{z}^s))$ for simplifying the calculation. Therefore, we can perform the following update operation, $\theta_g = \theta_g - \eta \nabla_{\theta_g} \sum_{s=1}^{m} \log(D(G(\vec{z}^s))$.

Here η is the learning rate, which is a hyperparameter and retains a constant value during the training of the model. It is optimised by testing for different values on the validation data set.
The update equations for trainable parameters are used to steer the parameters towards their optimised value by using the partial derivatives.

Theoretical Analysis

- The minimax game (13) has a global optimum for $p_g = p_{data}$.
- The above algorithm obtains the desired result.

1. For G fixed, the optimal discriminator D is

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{14}$$

Proof:

$$L(\theta_g, \theta_d) = -[\mathop{\mathbb{E}}_{\vec{x}\sim p_{data}(\vec{x})}[\log(D(\vec{x})] + \mathop{\mathbb{E}}_{\vec{x}\sim p_g(\vec{x})}[\log(1 - D(\vec{x})]] \tag{15}$$

$$L(\theta_g, \theta_d) = -\int (p_{data}(x)\log(D(x) + p_g(x)\log(1 - D(x))dx \tag{16}$$

We want to minimize the function with respect to D. The function is of the form $f(y) = -(a \cdot log(y) + b \cdot log(1 - y))$ To find minimum value, take the derivative.The function will be minimum at

$$y = \frac{a}{a + b} \tag{17}$$

Using (16) and (17),

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{18}$$

2. For fixed $D = D_G^*(x)$, optimal G is

$$min- > L(\theta_g, \theta_d) = \mathop{\mathbb{E}}_{\vec{x}\sim p_{data}(\vec{x})}[\log(D_G^*(x))] + \mathop{\mathbb{E}}_{\vec{z}\sim p_{\vec{z}}(\vec{z})}[\log(1 - D_G^*(x))] \tag{19}$$

$$L(\theta_g, \theta_d) = \mathop{\mathbb{E}}_{\vec{x}\sim p_{data}(\vec{x})}[\log(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)})] + \mathop{\mathbb{E}}_{\vec{z}\sim p_{\vec{z}}(\vec{z})}[\log(\frac{p_g(x)}{p_{data}(x) + p_g(x)})] \tag{20}$$

As p_g is the function of generator, we want to optimize the above equation with respect to it.To optimize, a trick is used. The first term looks like KL divergence. The only problem is that the denominator is not a pdf. To convert it into a pdf, we can take it's average. So, inside the log term, divide both numerator and denominator by 2.

$$L(\theta_g, \theta_d) = \mathop{\mathbb{E}}_{\vec{x}\sim p_{data}(\vec{x})}[\log(\frac{\frac{p_{data}(x)}{2}}{\frac{p_{data}(x) + p_g(x)}{2}})] + \mathop{\mathbb{E}}_{\vec{z}\sim p_{\vec{z}}(\vec{z})}[\log(\frac{\frac{p_g(x)}{2}}{\frac{p_{data}(x) + p_g(x)}{2}})] \tag{21}$$

$$L(\theta_g, \theta_d) = -log2 + KL[p_{data}(x)||\frac{p_{data}(x) + p_g(x)}{2}] - log2 + KL[p_g(x)||\frac{p_{data}(x) + p_g(x)}{2}] \tag{22}$$

Minimum value of KL divergence is zero when both the pdfs are equal. The sum of these two KL divergence's is called **Jensen–Shannon divergence**. It can be said that JSD is a symmetric variant of KL Divergence. So now (22) is transformed to

$$L(\theta_g, \theta_d) = -2log2 + 2JS\,D(p_{data}||p_g) \tag{23}$$

The value of JSD is also zero when both the pdfs are equal.So value of JSD is zero when $p_{data} = p_g$. This is the condition for **global optimum**.

Limitations of GANs

Consider the case when G(z) is mapped to a valid x ∈ data for all z. This is called **Mode Collapse**.

$$L(\theta_g, \theta_d) = -[\underset{\vec{x}\sim p_{data}(\vec{x})}{\mathbb{E}}[\log(D_G(x))] + \underset{\vec{z}\sim p_{\vec{z}}(\vec{z})}{\mathbb{E}}[\log(1 - D_G(x))]] \tag{24}$$

As $G(z) = x_1$ value of D(x) will be 1. So first term in the loss function will be zero and second term is $-\infty$. So because of negative sign, loss becomes very large.

Reference

- Generative Adversarial Nets: Ian J. Goodfellow