Assignment #1

Sima Shafaei

Jan 28

1. Show that the *tanh* function is a re-scaled sigmoid function with both horizontal and vertical streching, as well as vertical translation:

$$\tanh(v) = 2sigmoid(2v) - 1$$

$$sigmoid(x) = \frac{e^x}{1+e^x} \Longrightarrow$$

$$2\,sigmoid(2v) - 1 = 2\,\frac{e^{2v}}{1+e^{2v}} - 1 = \frac{2e^{2v} - 1 - e^{2v}}{1+e^{2v}} = \frac{e^{2v}-1}{e^{2v}+1} = \frac{e^{2v}-1}{e^{2v}+1} \times \frac{e^{-v}}{e^{-v}}$$
$$= \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} = \tanh(v)$$

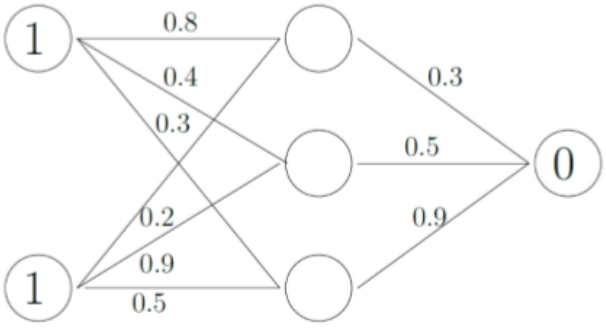
2. Show the following properties of the sigmoid and tanh activation functions:
   a. Sigmoid: $\Phi(-v) = 1 - \Phi(v)$
   b. Tanh activation: $\Phi(-v) = -\Phi(v)$
   c. Hard tanh activation: $\Phi(-v) = -\Phi(v)$

a) $sigmoid(x) = \frac{e^x}{1+e^x} \Longrightarrow sigmoid(-x) = \frac{e^{-x}}{1+e^{-x}} = \frac{e^{-x}}{1+e^{-x}} \times \frac{e^x}{e^x} = \frac{1}{e^x+1} =$
$\frac{1+e^x-e^x}{1+e^x} = \frac{1+e^x}{1+e^x} - \frac{e^x}{1+e^x} = 1 - sigmoid(x)$

b) $\tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}} \Longrightarrow tanh(-x) = \frac{e^{-x}-e^x}{e^{-x}+e^x} = -1 \times \frac{e^x-e^{-x}}{e^x+e^{-x}} = -tahn(x)$

c) $hard\_\tanh(x) = \begin{cases} -1 & : \quad x < -1 \\ x & : -1 \le x \le 1 \\ 1 & : \quad x > 1 \end{cases} \Longrightarrow hard\_\tanh(-x) =$
$\begin{cases} -1 & : \quad x > 1 \\ -x & : -1 \le x \le 1 \\ 1 & : \quad x < -1 \end{cases} = -1 \times \begin{cases} -1 & : \quad x < -1 \\ x & : -1 \le x \le 1 \\ 1 & : \quad x > 1 \end{cases} = -hard\_\tanh(x)$

3. Consider the neural network:



All the activation functions from the neurons in the hidden layer are sigmoids and the error is calculated by using squared error function:

a. Describe all the essential parts from the neural network.
b. Given the input {1,1} compute the predicted output of the network step by step and calculate the error if the target output is 0.
c. Compute step by step 2 training epochs using Back-Propagation algorithm.

a)

The Neural Network is constructed from 3 type of layers and an additional parameter(bias)

***Input Layer**: The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons. The input layer is the very beginning of the workflow for the artificial neural network.*

***Hidden layer**: A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. It is a typical part of nearly any neural network in which engineers simulate the types of activity that go on in the human brain.*

***Output Layer:** the output layer produces the result for given inputs*

***Bias**: It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron. Thus, Bias is a constant which helps the model in a way that it can fit best for the given data.*

b) $\theta_1^1 = [0.8,0.2], \theta_2^1 = [0.4,0.9], \theta_3^1 = [0.3,0.5] \implies \Theta^1 = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.9 \\ 0.3 & 0.5 \end{bmatrix}$

$\Theta^2 = [0.3 \quad 0.5 \quad 0.9]$

$z_1^2 = 0.8 \times 1 + 0.2 \times 1 = 1 \implies g_1^2(z_1^2) = \frac{e^1}{1+e^1} = 0.731 = \hat{y}_1^2$

$z_2^2 = 0.4 \times 1 + 0.9 \times 1 = 1.3 \implies g_2^2(z_2^2) = \frac{e^{1.3}}{1+e^{1.3}} = 0.785 = \hat{y}_2^2$

$z_3^2 = 0.3 \times 1 + 0.5 \times 1 = 0.8 \implies g_3^2(z_3^2) = \frac{e^{0.8}}{1+e^{0.8}} = 0.689 = \hat{y}_3^2$

$z_1^3 = 0.3 \times 0.731 + 0.5 \times 0.785 + 0.9 \times 0.689 = 1.23 \implies g_1^3(z_1^3) = \frac{e^{1.23}}{1+e^{1.23}} = 0.774 = \hat{Y}$

Error : $J = \sum_{i=1}^{1}\sum_{k=1}^{1}\frac{1}{2}\left(Y_k(i) - \widehat{Y_k}(i)\right)^2 = \frac{1}{2}(0-0.77)^2 = 0.296$

c) *First training epoch (updating weights)*

$\theta_j^p(new) = \theta_j^p(old) + \Delta\theta_j^p$

$\Delta\theta_j^p = -\alpha \frac{\partial J}{\partial \theta_j^p} \quad , \alpha = 0.5$

$J = \frac{1}{2}(y^3 - \hat{y}^3)^2 \qquad \hat{y}^3 = \frac{1}{1+e^{-z_1^3}} \qquad z_1^3 = (\theta_1^2)^T . \hat{y}^2$

$$\Delta\theta_1^2 = -\alpha \frac{\partial J}{\partial \theta_1^2} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \theta_1^2}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1+e^{-z_1^3}\right)^2} \times \hat{y}^2 =$$

$$-\alpha \times -(0-0.77) \times 0.17 \times \begin{bmatrix}0.73\\0.78\\0.69\end{bmatrix} = -\alpha \times \begin{bmatrix}0.095\\0.102\\0.09\end{bmatrix} = -\begin{bmatrix}0.0494\\0.0531\\0.0466\end{bmatrix}$$

$$\theta_1^2(new) = \begin{bmatrix}0.3\\0.5\\0.9\end{bmatrix} - \begin{bmatrix}0.0494\\0.0531\\0.0466\end{bmatrix} = \begin{bmatrix}0.2505\\0.4468\\0.8533\end{bmatrix}$$

$J = \frac{1}{2}(y^3 - \hat{y}^3)^2 \qquad \hat{y}_1^2 = \frac{1}{1+e^{-z_1^2}} \qquad z_1^2 = (\theta_1^1)^T . \hat{y}^2$

$$\Delta\theta_1^1 = -\alpha \frac{\partial J}{\partial \theta_1^1} = -\alpha \frac{\partial J}{\partial \hat{y}_1^2} \times \frac{\partial \hat{y}_1^2}{\partial z_1^2} \times \frac{\partial z_1^2}{\partial \theta_1^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_1^2} \times \frac{\partial \hat{y}_1^2}{\partial z_1^2} \times \frac{\partial z_1^2}{\partial \theta_1^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1+e^{-z_1^3}\right)^2} \times 0.3 \times \frac{e^{-z_1^2}}{\left(1+e^{-z_1^2}\right)^2} \times X =$$

$$-\alpha \times -(0-0.77) \times 0.17 \times 0.3 \times 0.19 \times \begin{bmatrix}1\\1\end{bmatrix} = -\alpha \times \begin{bmatrix}0.007\\0.007\end{bmatrix} = -\begin{bmatrix}0.0035\\0.0035\end{bmatrix}$$

$$\theta_1^1(new) = \begin{bmatrix}0.8\\0.2\end{bmatrix} - \begin{bmatrix}0.0035\\0.0035\end{bmatrix} = \begin{bmatrix}0.7965\\0.1965\end{bmatrix}$$

$$\Delta\theta_2^1 = -\alpha \frac{\partial J}{\partial \theta_2^1} = -\alpha \frac{\partial J}{\partial \hat{y}_2^2} \times \frac{\partial \hat{y}_2^2}{\partial z_2^2} \times \frac{\partial z_2^2}{\partial \theta_2^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_2^2} \times \frac{\partial \hat{y}_2^2}{\partial z_2^2} \times \frac{\partial z_2^2}{\partial \theta_2^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1 + e^{-z_1^3}\right)^2} \times 0.5 \times \frac{e^{-z_2^2}}{\left(1 + e^{-z_2^2}\right)^2} \times X$$

$$= -\alpha \times -(0 - 0.77) \times 0.17 \times 0.5 \times 0.21 \times \begin{bmatrix}1\\1\end{bmatrix} = -\alpha \times \begin{bmatrix}0.013\\0.013\end{bmatrix}$$

$$= -\begin{bmatrix}0.0065\\0.0065\end{bmatrix}$$

$\theta_2^1(new) = \begin{bmatrix}0.4\\0.9\end{bmatrix} - \begin{bmatrix}0.0065\\0.0065\end{bmatrix} = \begin{bmatrix}0.3935\\0.8935\end{bmatrix}$

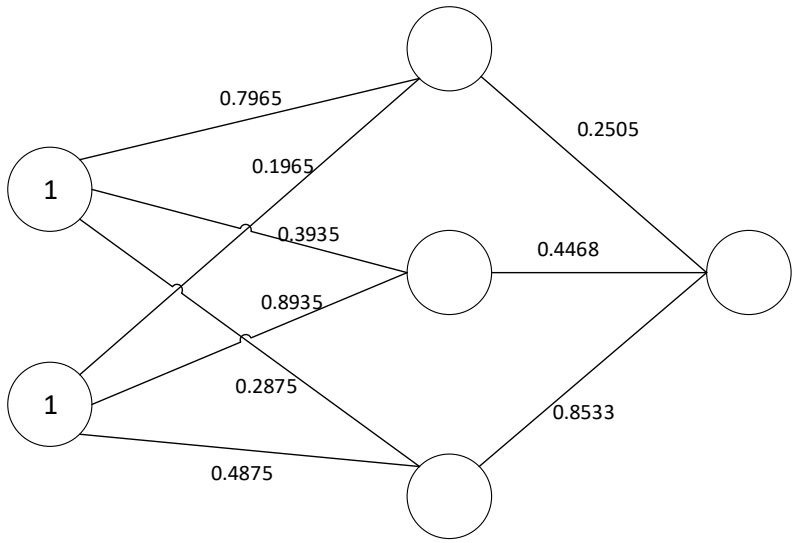
$$\Delta\theta_3^1 = -\alpha \frac{\partial J}{\partial \theta_3^1} = -\alpha \frac{\partial J}{\partial \hat{y}_3^2} \times \frac{\partial \hat{y}_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial \theta_3^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_3^2} \times \frac{\partial \hat{y}_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial \theta_3^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1 + e^{-z_1^3}\right)^2} \times 0.9 \times \frac{e^{-z_3^2}}{\left(1 + e^{-z_3^2}\right)^2} \times X$$

$$= -\alpha \times -(0 - 0.77) \times 0.17 \times 0.9 \times 0.213 \times \begin{bmatrix}1\\1\end{bmatrix} = -\alpha \times \begin{bmatrix}0.025\\0.025\end{bmatrix}$$

$$= -\begin{bmatrix}0.0125\\0.0125\end{bmatrix}$$

$\theta_3^1(new) = \begin{bmatrix}0.3\\0.5\end{bmatrix} - \begin{bmatrix}0.0125\\0.0125\end{bmatrix} = \begin{bmatrix}0.2875\\0.4875\end{bmatrix}$



Second Iteration:

$$z_1^2 = 0.7965 \times 1 + 0.1965 \times 1 = 0.993 \implies g_1^2(z_1^2) = \frac{e^{0.993}}{1 + e^{0.993}} = 0.729 = \hat{y}_1^2$$

$$z_2^2 = 0.3935 \times 1 + 0.8935 \times 1 = 1.288 \implies g_2^2(z_2^2) = \frac{e^{1.287}}{1 + e^{1.287}} = 0.783 = \hat{y}_2^2$$

$$z_3^2 = 0.2875 \times 1 + 0.4875 \times 1 = 0.773 \implies g_3^2(z_3^2) = \frac{e^{0.775}}{1+e^{0.775}} = 0.684 = \hat{y}_3^2$$

$$z_1^3 = 0.2505 \times 0.729 + 0.4468 \times 0.783 + 0.8533 \times 0.684 = 1.117 \implies g_1^3(z_1^3) = \frac{e^{1.12}}{1+e^{1.12}} = 0.753 = \hat{Y}$$

$$Error: J = \sum_{i=1}^{1}\sum_{k=1}^{1}\frac{1}{2}\left(Y_k(i) - \widehat{Y_k}(i)\right)^2 = \frac{1}{2}(0-0.753)^2 = 0.28$$

Backpropagation:

$$J = \frac{1}{2}(y^3 - \hat{y}^3)^2 \qquad \hat{y}^3 = \frac{1}{1+e^{-z_1^3}} \qquad z_1^3 = (\theta_1^2)^T . \hat{y}^2$$

$$\Delta\theta_1^2 = -\alpha \frac{\partial J}{\partial \theta_1^2} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \theta_1^2}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1+e^{-z_1^3}\right)^2} \times \hat{y}^2 = -\begin{bmatrix}0.051\\0.054\\0.047\end{bmatrix}$$

$$\theta_1^2(new) = \begin{bmatrix}0.2505\\0.4468\\0.8533\end{bmatrix} - \begin{bmatrix}0.051\\0.054\\0.047\end{bmatrix} = \begin{bmatrix}0.199\\0.391\\0.805\end{bmatrix}$$

$$J = \frac{1}{2}(y^3 - \hat{y}^3)^2 \qquad \hat{y}_1^2 = \frac{1}{1+e^{-z_1^2}} \qquad z_1^2 = (\theta_1^1)^T . \hat{y}^2$$

$$\Delta\theta_1^1 = -\alpha \frac{\partial J}{\partial \theta_1^1} = -\alpha \frac{\partial J}{\partial \hat{y}_1^2} \times \frac{\partial \hat{y}_1^2}{\partial z_1^2} \times \frac{\partial z_1^2}{\partial \theta_1^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_1^2} \times \frac{\partial \hat{y}_1^2}{\partial z_1^2} \times \frac{\partial z_1^2}{\partial \theta_1^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1+e^{-z_1^3}\right)^2} \times 0.3 \times \frac{e^{-z_1^2}}{\left(1+e^{-z_1^2}\right)^2} \times X$$

$$= -\begin{bmatrix}0.0034\\0.0034\end{bmatrix}$$

$$\theta_1^1(new) = \begin{bmatrix}0.7965\\0.1965\end{bmatrix} - \begin{bmatrix}0.0034\\0.0034\end{bmatrix} = \begin{bmatrix}0.7925\\0.1925\end{bmatrix}$$

$$\Delta\theta_2^1 = -\alpha \frac{\partial J}{\partial \theta_2^1} = -\alpha \frac{\partial J}{\partial \hat{y}_2^2} \times \frac{\partial \hat{y}_2^2}{\partial z_2^2} \times \frac{\partial z_2^2}{\partial \theta_2^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_2^2} \times \frac{\partial \hat{y}_2^2}{\partial z_2^2} \times \frac{\partial z_2^2}{\partial \theta_2^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1+e^{-z_1^3}\right)^2} \times 0.5 \times \frac{e^{-z_2^2}}{\left(1+e^{-z_2^2}\right)^2} \times X$$

$$= -\begin{bmatrix}0.0053\\0.0053\end{bmatrix}$$

$$\theta_2^1(new) = \begin{bmatrix}0.3935\\0.8935\end{bmatrix} - \begin{bmatrix}0.0053\\0.0053\end{bmatrix} = \begin{bmatrix}0.3890\\0.8890\end{bmatrix}$$

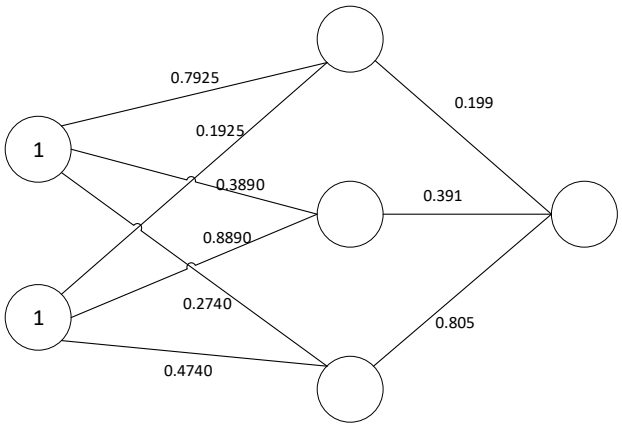
$$\Delta\theta_3^1 = -\alpha \frac{\partial J}{\partial \theta_3^1} = -\alpha \frac{\partial J}{\partial \hat{y}_3^2} \times \frac{\partial \hat{y}_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial \theta_3^1} = -\alpha \frac{\partial J}{\partial \hat{y}^3} \times \frac{\partial \hat{y}^3}{\partial z_1^3} \times \frac{\partial z_1^3}{\partial \hat{y}_3^2} \times \frac{\partial \hat{y}_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial \theta_3^1}$$

$$= -\alpha \times -(y^3 - \hat{y}^3) \times \frac{e^{-z_1^3}}{\left(1 + e^{-z_1^3}\right)^2} \times 0.9 \times \frac{e^{-z_3^2}}{\left(1 + e^{-z_3^2}\right)^2} \times X$$

$$= -\begin{bmatrix} 0.0128 \\ 0.0128 \end{bmatrix}$$

$$\theta_3^1(new) = \begin{bmatrix} 0.2875 \\ 0.4875 \end{bmatrix} - \begin{bmatrix} 0.0128 \\ 0.0128 \end{bmatrix} = \begin{bmatrix} 0.2740 \\ 0.4740 \end{bmatrix}$$

Network after two iterations:



$$z_1^2 = 0.7925 \times 1 + 0.1925 \times 1 = 0.985 \implies g_1^2(z_1^2) = \frac{e^{0.985}}{1 + e^{0.985}} = 0.728 = \hat{y}_1^2$$

$$z_2^2 = 0.3935 \times 1 + 0.8935 \times 1 = 1.278 \implies g_2^2(z_2^2) = \frac{e^{1.278}}{1+e^{1.278}} = 0.782 = \hat{y}_2^2$$

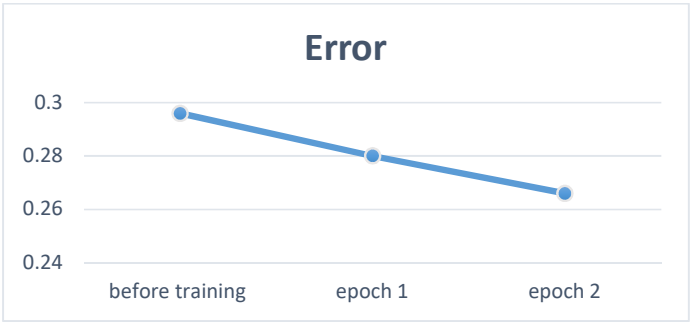
$$z_3^2 = 0.2875 \times 1 + 0.4875 \times 1 = 0.748 \implies g_3^2(z_3^2) = \frac{e^{0.748}}{1+e^{0.748}} = 0.678 = \hat{y}_3^2$$

$$z_1^3 = 0.199 \times 0.728 + 0.391 \times 0.782 + 0.805 \times 0.678 = 0.998 \implies g_1^3(z_1^3) = \frac{e^{0.998}}{1 + e^{0.998}}$$

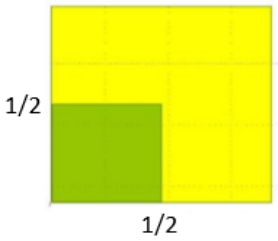
$$= 0.730 = \hat{Y}$$

Error : $J = \sum_{i=1}^{1} \sum_{k=1}^{1} \frac{1}{2} \left(Y_k(i) - \widehat{Y_k}(i)\right)^2 = \frac{1}{2}(0 - 0.730)^2 = 0.266$

As it is obvious from this figure, the error is reducing in each iteration



4. Program your own MLP in Python for a basic neural network with one or to hidden layers and binary output (select the proper activation function)
    a. Generate a random data set for binary classification where each class correspond to the 2D region depicted in the figure. The random data should have normal distribution with variance $\sigma^2 = 0.08$. Use 200 points (100 in each region) to train your neural network and report the results in terms of the loss function and the training epochs.



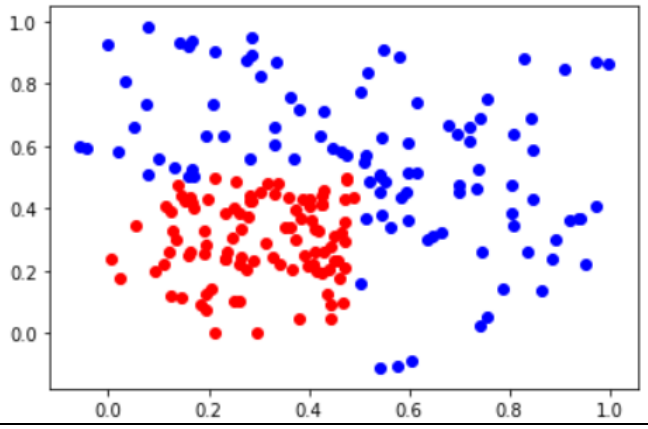
Random Data Generation:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def dataGeneration1(mu, sigma, num ):
    n=0
    x=np.zeros((num,2))
    while n<num:
        r=np.random.normal(mu, sigma , 1)
        if 0<=r<=0.5:
            x[n,0]=r
            n+=1
    n=0
    while n<num:
        r=np.random.normal(mu, sigma , 1)
        if 0<=r<=0.5:
            x[n,1]=r
            n+=1
    return x
```

```
def dataGeneration2(mu, sigma, num ):
    n=0
    x=np.zeros((num,2))
    while n<num:
        r1=np.random.normal(mu, sigma , 1)
        r2=np.random.normal(mu, sigma , 1)
        if (r1>0.5 or r2>0.5) and r1<=1 and r2<=1:
            x[n,0]=r1
            x[n,1]=r2
            n+=1
    return x
```

```
#data generation
x1=dataGeneration1(0.5, 0.08**0.5, 100)
shp=(200,2)
d=np.zeros(shp)
shp=(200,1)
target=np.zeros(shp)
plt.scatter(x1[:,0],x1[:,1],c='red')
for i in range (100):
    d[i,0]=x1[i,0]
    d[i,1]=x1[i,1]
    target[i,0]=0


x2=dataGeneration2(0.5, 0.08**0.5, 100)
plt.scatter(x2[:,0],x2[:,1],c='blue')
for i in range (100):
    d[i+100,0]=x2[i,0]
    d[i+100,1]=x2[i,1]
    target[i,0]=1
```



***Activation functions**: Two different activation functions (tanh and sigmoid) are tested for this problem. The results obtained from tanh are more accurate. The following results are based on tanh function*

```
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    return t

def d_tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return dt

def sigmoid(x):
    return 1/(1+np.exp(-x))

def d_sigmoid(x):
    return np.exp(-x)/((1+np.exp(-x))**2)
```

Neural Network class:

I used a MLP NN with 1 hidden layer (the number of nodes in hidden layer, number of epoch and learning rate can be determined by user)

```
class NeuralNetwork:
    def __init__(self, x, y,hiddenNode,epoch,alpha):

        self.input      = x
        self.weights1   =np.random.rand(self.input.shape[1],hiddenNode)

        self.weights2   =np.random.rand(hiddenNode,1)
        self.y          = y
        self.output     = np.zeros(self.y.shape)
        self.alpha      = alpha
        self.epoch      = epoch
    def feedforward(self):
        self.z2=np.dot(self.input, self.weights1)
        self.y_hat2 = tanh(self.z2) #sigmoid(self.z2)
        self.z3=np.dot(self.y_hat2, self.weights2)
        self.y_hat3 = tanh(self.z3)#sigmoid(self.z3)
        self.error=1/self.input.shape[0] * sum(0.5 *(self.y-self.y_hat3)**2)
    def backproppagation(self):
        d_weights2 = np.dot(self.y_hat2.T, (self.alpha*(self.y - self.y_hat3) *
                                            d_tanh(self.z3)))
        d_weights1 = np.dot(self.input.T,  np.dot(self.alpha*(self.y - self.y_hat3) *
                                                  d_tanh(self.z3), self.weights2.T) *
                                                  d_tanh(self.z2))

        self.weights1 += d_weights1
        self.weights2 += d_weights2
```

```
    def fit(self):
        err= np.zeros(self.epoch)
        for k in range(self.epoch):
            self.feedforward()
            self.backproppagation()
            err[k]=self.error
        #plt.figure()
        #plt.plot(err)
        return err

    def test(self,test_data,test_output):
        z2=np.dot(test_data, self.weights1)
        test_y_hat2 = tanh(z2)
        z3=np.dot(test_y_hat2, self.weights2)
        test_y_hat3 = tanh(z3)
        error=1/test_data.shape[0] * sum(0.5 *(test_output-test_y_hat3)**2)
        #plt.figure()
        TP=FP=FN=TN=0
        for i in range(len(y_test)) :
            if test_output[i]==0 and abs(test_y_hat3[i])<=0.5:
                #plt.scatter(test_data[i,0],test_data[i,1],c='green')
                TN+=1
            elif test_output[i]==0 and abs(test_y_hat3[i])>0.5:
                #plt.scatter(test_data[i,0],test_data[i,1],c='red')
                FN+=1
            elif test_output[i]==1 and abs(test_y_hat3[i])>=0.5:
                #plt.scatter(test_data[i,0],test_data[i,1],c='blue')
                TP+=1
            elif test_output[i]==1 and abs(test_y_hat3[i])<0.5:
                #plt.scatter(test_data[i,0],test_data[i,1],c='yellow')
                FP+=1
        print((TN+TP)/(TN+TP+FP+FN))
        return error,(TN+TP)/(TN+TP+FP+FN)
```

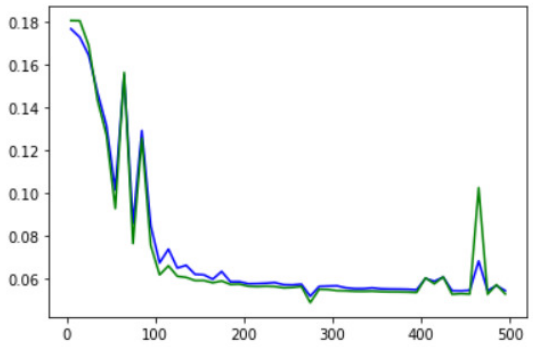
Split data into test and train dataset: *20% of data (40 points out of 200 points) is selected as test data and 80% (160 points out of 200 points) is used as training data*

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(d, target, test_size=0.2)
```

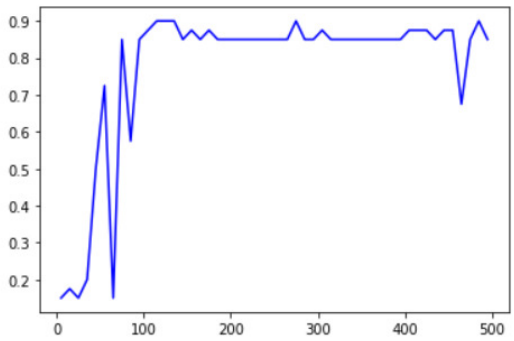
Run Neural Network:

we run network with learning rate=0.01, number of hidden node=5 and different value for epoch (5,15,25, … 445)

```
# learn the model:
trn_err=[]
test_err=[]
test_acuracy=[]
for i,e in enumerate(range(5,500,10)):
    nn=NeuralNetwork(X_train,y_train,3,e,0.01)
    train_err=nn.fit()
    trn_err.append(train_err[e-1])
    err,acuracy=nn.test(X_test,y_test)
    test_err.append(err)
    test_acuracy.append(acuracy)

plt.figure()
e=range(5,500,10)
plt.plot(e,trn_err,'b')
plt.plot(e,test_err,'g')
plt.figure()
plt.plot(e,test_acuracy,'b')
```



Error on test (green line) and train(blue line) data



Accuracy on test data for different value of epoch

Figure 1

The green diagram shows the change of error obtained over test data for each value of epoch and the blue line shows the obtained error over train data

By running the network with 5 hidden node, epoch =150 and learning rate=0.01 we obtained test error =0.06

```
nn=NeuralNetwork(X_train,y_train,5,150,0.01)
train_err=nn.fit()
test_err=nn.test(X_test,y_test)
print(test_err)
```

```
[0.06244146]
```

Figure 2 shows the error on training data in each iteration in the above run

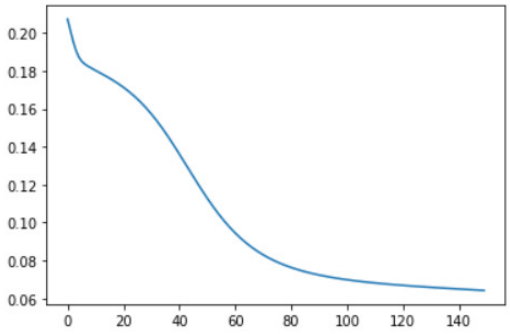


Figure 2

Figure 3 shows the miss classified points in test data. The green and blue dots belong to the 0 and 1 classes, respectively and are correctly labeled by our MLP. Red points came from class 0 but incorrectly labeled as class 1.

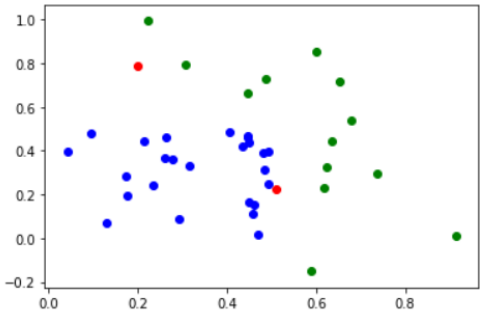


Figure 3

Figure 4 and 5 also show similar results of running the program with new random dataset

The green and blue dots are truly classified and the red and yellow dots are incorrectly classified

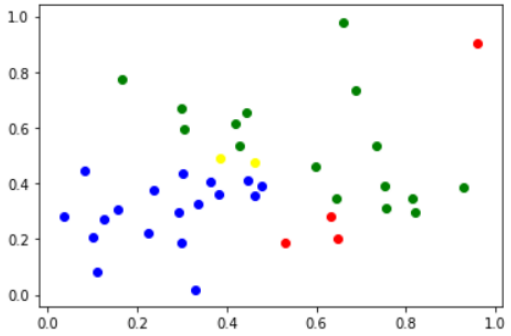


Figure 4

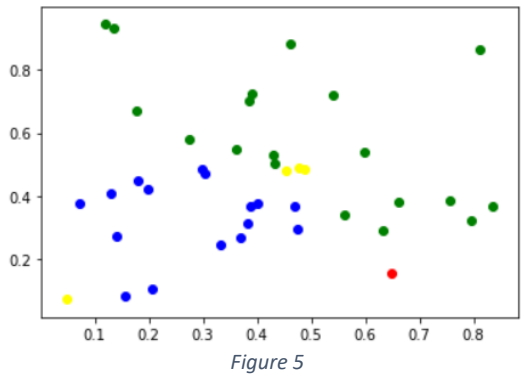


Figure 5

b. Repeat 4.a. for the following ıs:

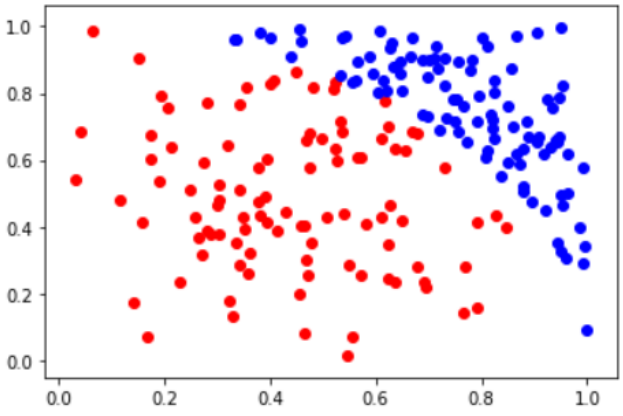
Random Data Generation:

```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
def dataGeneration1(mu, sigma, num ):
    n=0
    x=np.zeros((num,2))
    while n<num:
        r1=np.random.normal(mu, sigma , 1)
        r2=np.random.normal(mu, sigma , 1)
        if (r1**2 + r2**2)<1 and r1>=0 and r2>=0 :
            x[n,0]=r1
            x[n,1]=r2
            n+=1
    return x
```

```python
def dataGeneration2(mu, sigma, num ):
    n=0
    x=np.zeros((num,2))
    while n<num:
        r1=np.random.normal(mu, sigma , 1)
        r2=np.random.normal(mu, sigma , 1)
        if (r1**2 + r2**2)>=1 and 0<=r1<=1 and 0<=r2<=1:
            x[n,0]=r1
            x[n,1]=r2
            n+=1
    return x
```

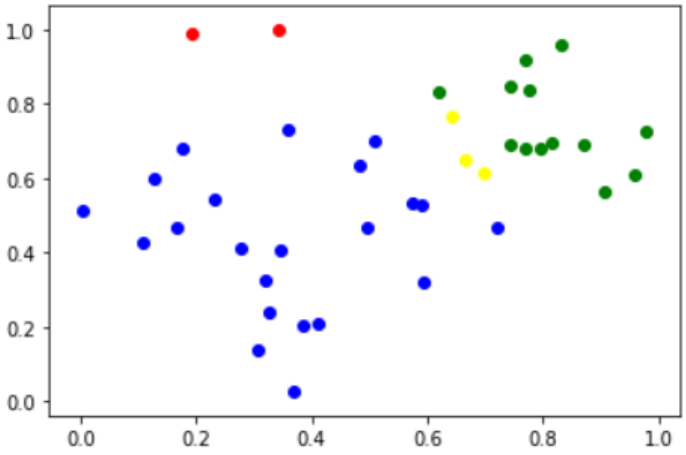
```python
#data generation
x1=dataGeneration1(0.5, 0.08**0.5, 100)
shp=(200,2)
d=np.zeros(shp)
shp=(200,1)
target=np.zeros(shp)
plt.scatter(x1[:,0],x1[:,1],c='red')
for i in range (100):
    d[i,0]=x1[i,0]
    d[i,1]=x1[i,1]
    target[i,0]=0


x2=dataGeneration2(0.5, 0.08**0.5, 100)
plt.scatter(x2[:,0],x2[:,1],c='blue')
for i in range (100):
    d[i+100,0]=x2[i,0]
    d[i+100,1]=x2[i,1]
    target[i,0]=1
```



Running NN for hidden node=3, epoch=150 and learning rate=0.01 shows the following result. The red and yellow dots are miss classified points

```
nn=NeuralNetwork(X_train,y_train,3,150,0.01)
train_err=nn.fit()
test_err=nn.test(X_test,y_test)
print(test_err)
```

```
[0.05439423]
```




```
# learn the model:
trn_err=[]
test_err=[]
test_acuracy=[]
for i,e in enumerate(range(5,500,10)):
    nn=NeuralNetwork(X_train,y_train,3,e,0.01)
    train_err=nn.fit()
    trn_err.append(train_err[e-1])
    err,acuracy=nn.test(X_test,y_test)
    test_err.append(err)
    test_acuracy.append(acuracy)

plt.figure()
e=range(5,500,10)
plt.plot(e,trn_err,'b')
plt.plot(e,test_err,'g')
plt.figure()
plt.plot(e,test_acuracy,'b')
```

In the following figure the green line shows the change of error obtained over test data for each value of epoch and the blue line shows the obtained error over train data

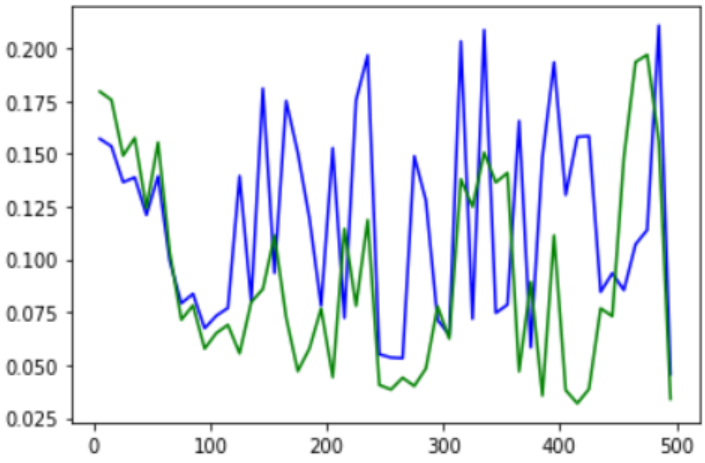


Figure 6: error on train data (blue line) and test data(green line)

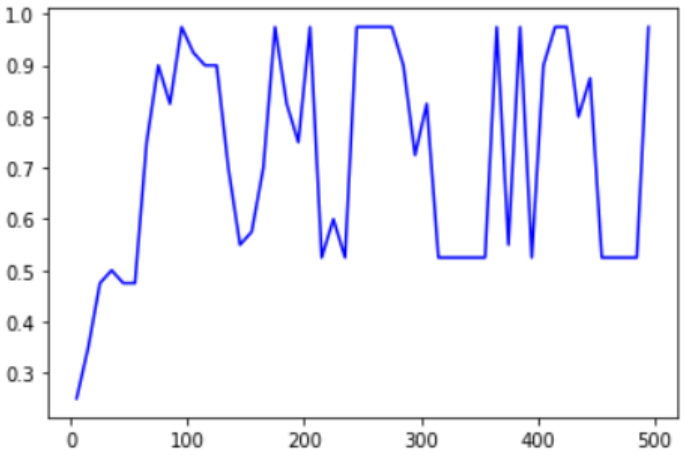


Figure 7: accuracy obtained on test data