Pattern Recognition and Machine Learning

Lab Assignment 9

Artificial Neural Networks

Jatin Lohar

B21CS091

Question 1 (Basis Neural Network)

Imported numpy, torch vision and datasets from torchvison. Downloaded the test data and train data using ***'datasets.MNIST'.*** Found the standard deviation and mean of the dataset.

```
The mean of the data is : 0.13066047
The STD of the data is : 0.3081078
```

A.

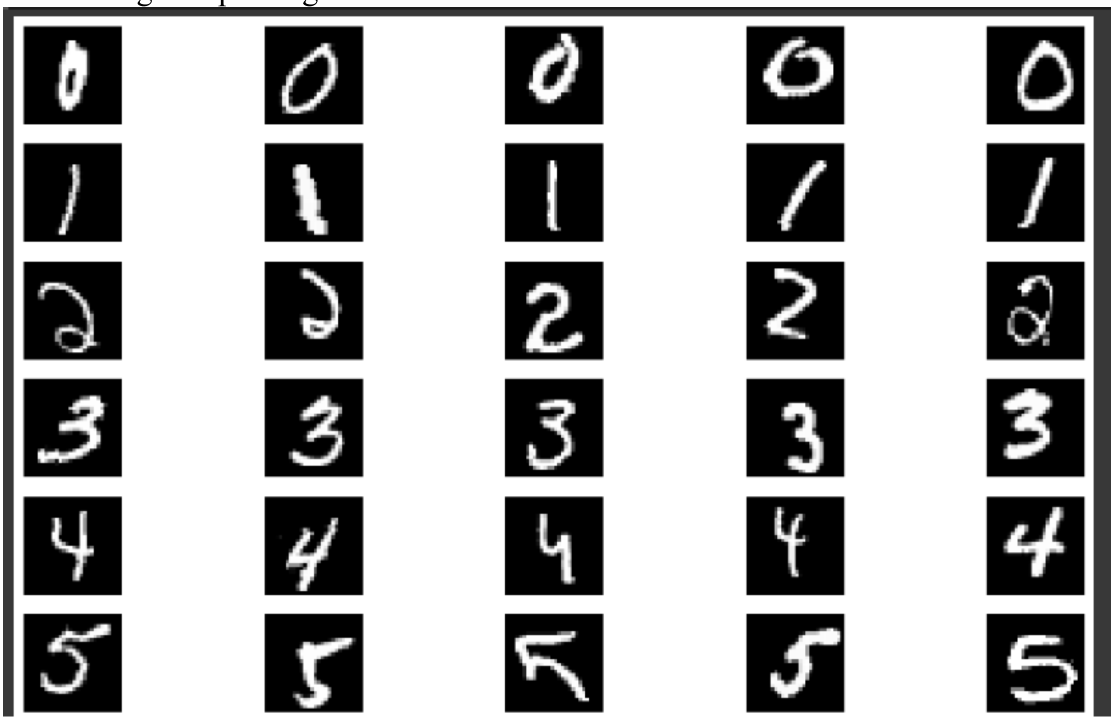
Imported transforms from torchvision. Transformed the data according to given specifications.

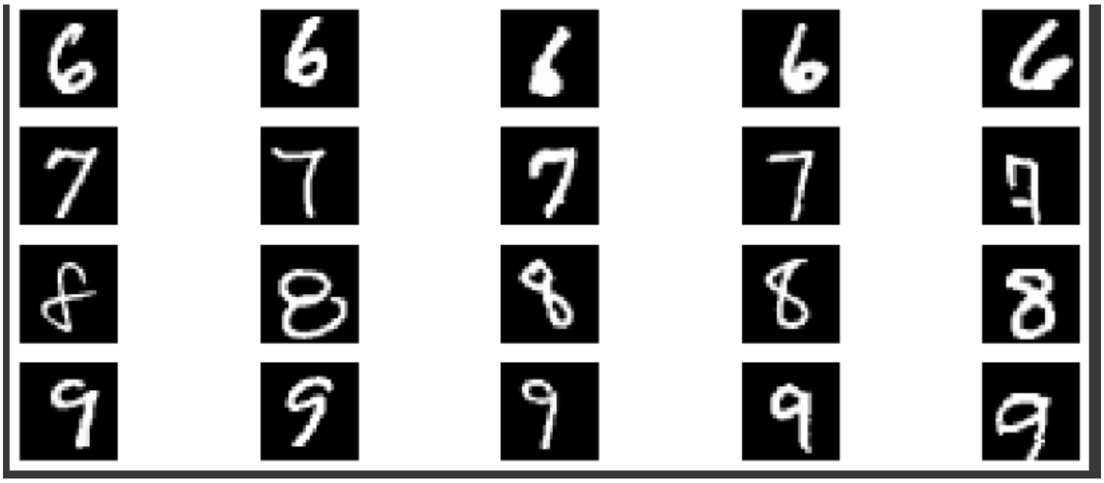
```
train_trans = transforms.Compose([transforms.RandomRotation(5), transforms.RandomCrop(size = 28, padding = 2),
                                  transforms.ToTensor(), transforms.Normalize(mean = mean, std = std)])
test_trans = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean = [mean], std = [std])])
```

Downloaded the dataset again by applying the transformation. Then, split the train_data into validation and training_data.

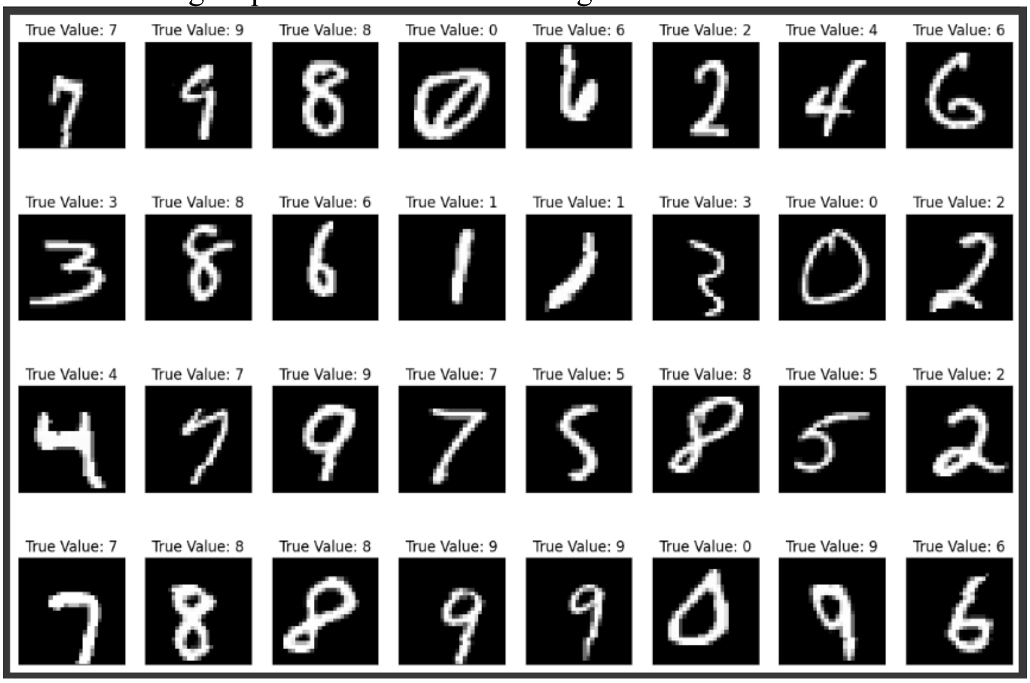
B.

Plotted 5 images of each number by looping to around the training data simply. Incase 5 images have been plotted, then continuing else plotting the number.





Then, created a dataloader using ***'torch.utils.data.DataLoader'.*** Made data loader for each of the training, testing and validation data. Again plotted some random images of the dataloader.



C.

Made MLP function using ***'torch.nn',*** using **nn.linear**. Then used the input as 28*28, that is the size of the images. And number of output as 10 (0-9). Made the model with given specifications.
Then printed the number if Trainable Parameters using numel.

```
Number of Trainanble Parameters : 222360
```

D.

Made the optimizer as Adam using ***'torch.optim.Adam'.*** and Criteria as ***'nn.CrossEntropyLoss'.***
Then made the device as ***'cuda'*** in case if cuda is available else used ***'cpu'.***

The device found was :

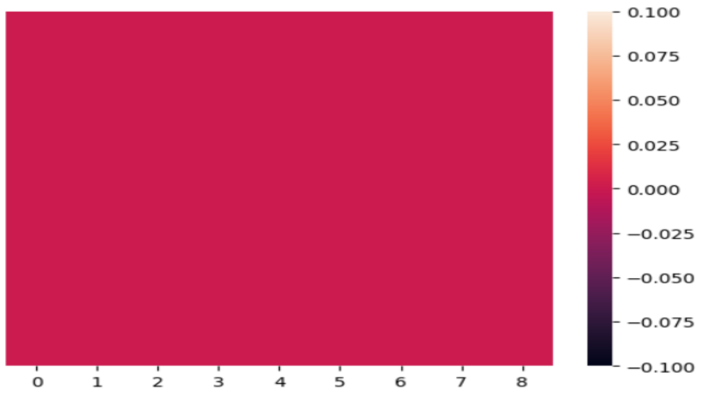
```
The device used is: cpu
```

Made the training model by using backward() for optimising the loss.

Question 2 (ANN from Scratch)

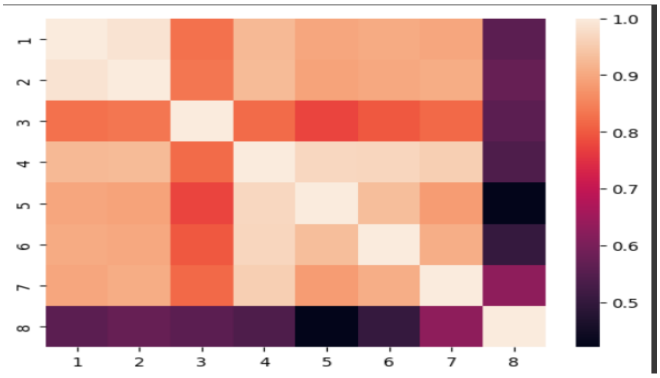
A.

Imported necessary libraries and then imported the data in variable ***'data'*** .
Check for NAN values using ***'sns.heatmap'.***



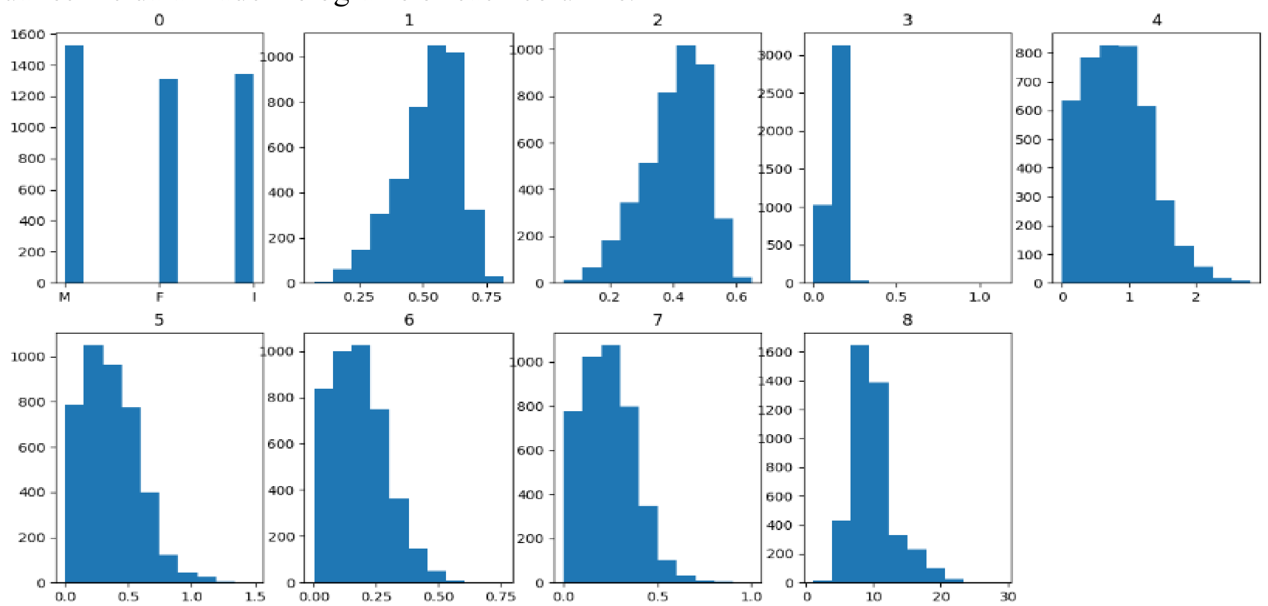
This shows that it does not have any NaN values.

Made the heatmap to check the correlations.



Here column 3 has maximum correlations with other columns.

To visualise the data made histograms of each columns.



Then split the data into X and Y. Also, since columns = 0 had character entries so converting the columns into integer entries. Later checked for the values counts of each class in Y. Since we need one type of class in each of the test, train, validation data. So, created at least 3 copies of each class if it does not exist.

```
9       689
10      634
8       568
11      487
7       391
12      267
6       259
13      203
14      126
5       115
15      103
16       67
17       58
4        57
18       42
19       32
20       26
3        15
21       14
23        9
22        6
27        2
24        2
1         1
26        1
29        1
2         1
25        1
```

Before increasing

```
[[  1    3]
 [  2    3]
 [  3   15]
 [  4   57]
 [  5  115]
 [  6  259]
 [  7  391]
 [  8  568]
 [  9  689]
 [ 10  634]
 [ 11  487]
 [ 12  267]
 [ 13  203]
 [ 14  126]
 [ 15  103]
 [ 16   67]
 [ 17   58]
 [ 18   42]
 [ 19   32]
 [ 20   26]
 [ 21   14]
 [ 22    6]
 [ 23    9]
 [ 24    3]
 [ 25    3]
 [ 26    3]
 [ 27    3]
 [ 29    3]]
```

After increasing

Then split the data into test, train and validation.

B.

To make Multilayer Perceptron from scratch. Initially took parameters as input while declaring the class. The inputs are number of nodes in input layer, number of nodes in hidden layer, number of nodes in Output Layer, Learing Rate, max epochs, Activation Functions, initial weights, etc.

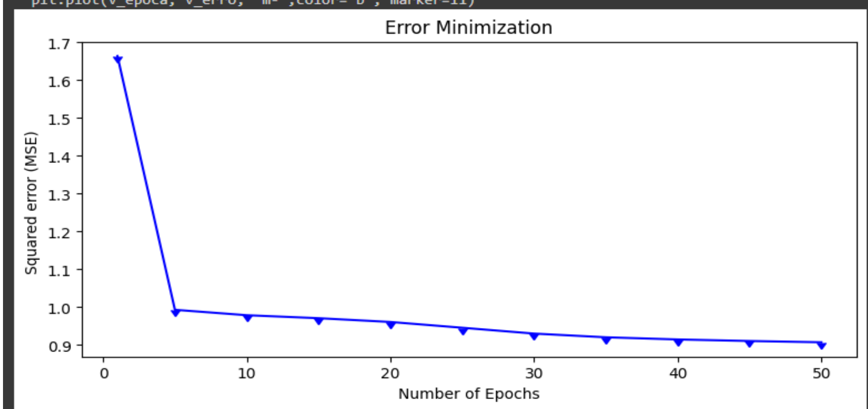
- Made the functions for back prorogation. By using the derivative of activation functions. And changing the weights accordingly.
- Made a function to plot the Error obtained with the number of epochs.
- Predict function to predict the output based on the observed weights.
- Fit function to train the model using the given dataset. Run a loop until max_epochs is reached. Found the output using the activations functions. And hence found the error. And the did Back Proporgation for the same. Found the total error and appended it into the error_array list. Also, in case if the error begins to increase, I used a condition by which if the error increases then break the loop. Then plotted the error/epoch plot using make_erro_plot function.

C.

Using Tanh functions

Using the activation function = tanh, input layer = 8, hiddenlayer = 5, OutputLayer = 29, LearningRate = 0.005, and initializing random weights as parameters, Then fitted the model on X_train and Y_train.

```
Epoch  1 -> Total Error:  1.6641416793337143
Epoch  5 -> Total Error:  0.993198810473574
Epoch  10 -> Total Error:  0.9789158259810572
Epoch  15 -> Total Error:  0.9710411630937257
Epoch  20 -> Total Error:  0.9610540539741661
Epoch  25 -> Total Error:  0.9457651317797449
Epoch  30 -> Total Error:  0.9303539802327063
Epoch  35 -> Total Error:  0.9205615333432318
Epoch  40 -> Total Error:  0.9148276659501691
Epoch  45 -> Total Error:  0.9107515180937733
Epoch  50 -> Total Error:  0.9073839194799903
<ipython-input-228-f9ae77606c4f>:49: UserWarning: color is redundantly defined by the 'color' keyword ar
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```



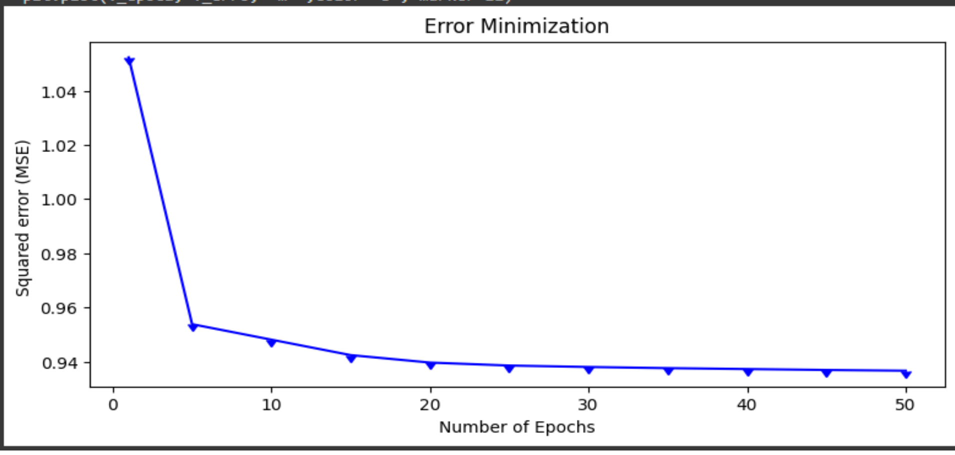
Then tested the model on X_test and found its accuracy.

```
Accuracy with Tanh Activation Function is :20.883%
```

Using Relu functions

Using the activation function = Relu, input layer = 8, hiddenlayer = 5, OutputLayer = 29, LearningRate = 0.005, and initializing random weights as parameters, Then fitted the model on X_train and Y_train.

```
Epoch  1 -> Total Error:  1.0523390228386038
Epoch  5 -> Total Error:  0.9538100103573603
Epoch  10 -> Total Error:  0.9481007639597826
Epoch  15 -> Total Error:  0.9423948247586752
Epoch  20 -> Total Error:  0.9396411350997158
Epoch  25 -> Total Error:  0.9385416815649802
Epoch  30 -> Total Error:  0.9380134946045602
Epoch  35 -> Total Error:  0.9375868159709658
Epoch  40 -> Total Error:  0.9372490029665448
Epoch  45 -> Total Error:  0.9369105181709505
Epoch  50 -> Total Error:  0.9366348705294066
<ipython-input-228-f9ae77606c4f>:49: UserWarning: color is redundantly defined by the 'color' keyword a
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```



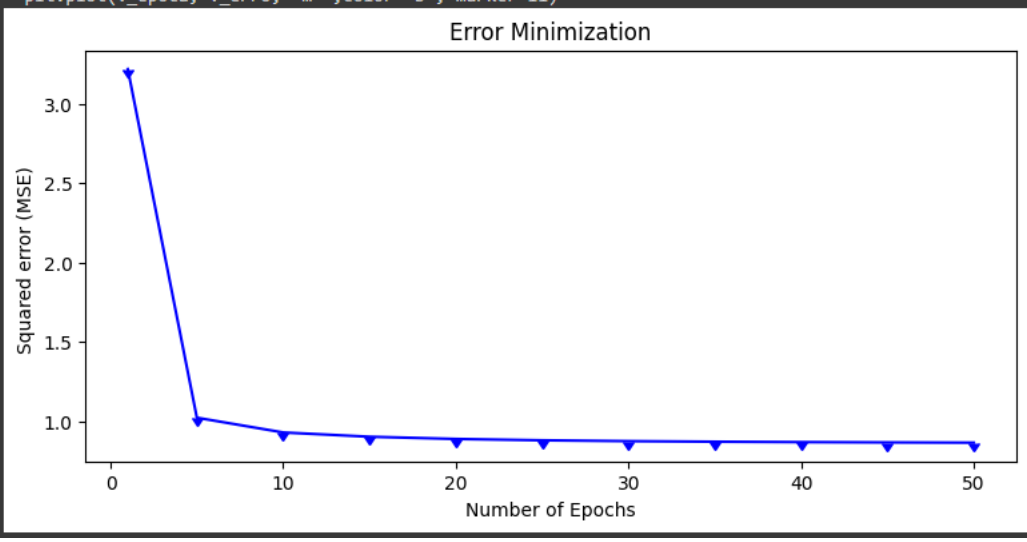
Then tested the model on X_test and found its accuracy.

```
Accuracy with Relu Activation Function is :11.695%
```

Using Sigmoid functions

Using the activation function = sigmoid, input layer = 8, hiddenlayer = 5, OutputLayer = 29, LearningRate = 0.005, and initializing random weights as parameters, Then fitted the model on X_train and Y_train.

```
Epoch  1 -> Total Error:  3.2199955592929648
Epoch  5 -> Total Error:  1.022249573265286
Epoch  10 -> Total Error:  0.9289808530910031
Epoch  15 -> Total Error:  0.9021312396708309
Epoch  20 -> Total Error:  0.8880560045767913
Epoch  25 -> Total Error:  0.8796274018585676
Epoch  30 -> Total Error:  0.8743336157460826
Epoch  35 -> Total Error:  0.8707868866193365
Epoch  40 -> Total Error:  0.8682412759503492
Epoch  45 -> Total Error:  0.8663043565307081
Epoch  50 -> Total Error:  0.8647608934597925
<ipython-input-294-1e7df31fdca0>:49: UserWarning: color is redundantly defined by the 'color' keyword a
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```



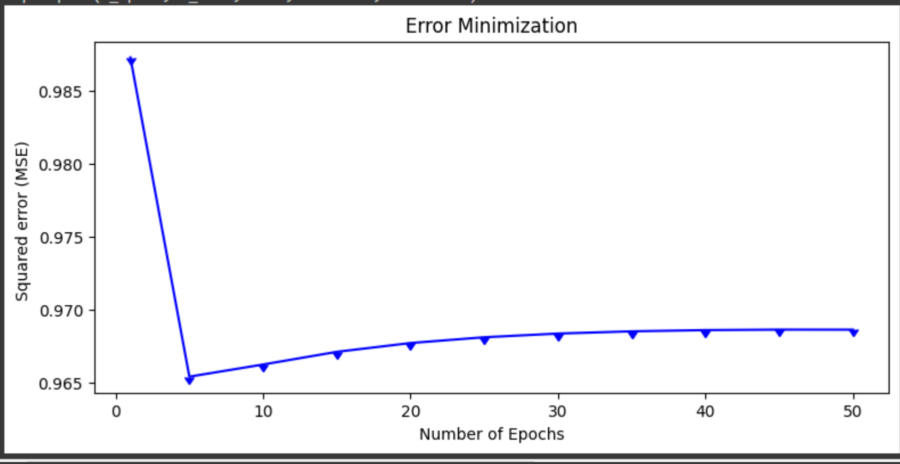
Then tested the model on X_test and found its accuracy.

```
Accuracy with Sigmoid Activation Function is :11.098%
```

D.

Initialized the weights as zeroes using np.zeros. And fitted into the model.

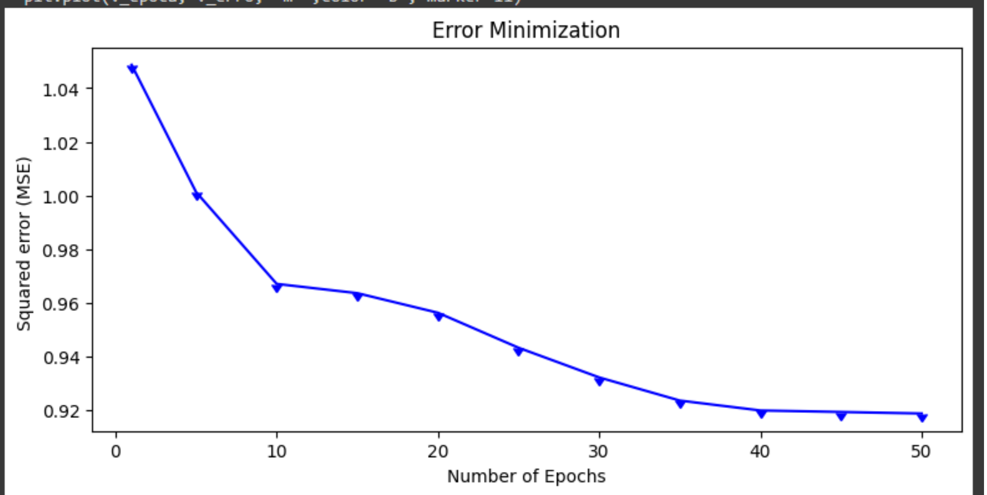
```
Epoch  1 -> Total Error:  0.9872963620647252
Epoch  5 -> Total Error:  0.965413784122676
Epoch  10 -> Total Error:  0.9662480286048958
Epoch  15 -> Total Error:  0.9671164245251775
Epoch  20 -> Total Error:  0.9677201164161422
Epoch  25 -> Total Error:  0.968115588501865
Epoch  30 -> Total Error:  0.9683680840395518
Epoch  35 -> Total Error:  0.968521851128603
Epoch  40 -> Total Error:  0.9686057989589433
Epoch  45 -> Total Error:  0.9686393899863528
Epoch  50 -> Total Error:  0.9686363030826309
<ipython-input-294-1e7df31fdca0>:49: UserWarning: color is redundantly defined by the 'color' keyword argu
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```




```
Accuracy with Zero wieght Initialization and Tanh Function is :16.587%
```

Using Constant weights as initial weight.

```
Epoch  1 -> Total Error:  1.0486669735281728
Epoch  5 -> Total Error:  1.001290672594173
Epoch  10 -> Total Error:  0.9671129157665785
Epoch  15 -> Total Error:  0.9636628736878649
Epoch  20 -> Total Error:  0.9563711341163346
Epoch  25 -> Total Error:  0.9433458215732743
Epoch  30 -> Total Error:  0.9323029694593844
Epoch  35 -> Total Error:  0.9236894923361512
Epoch  40 -> Total Error:  0.919999018396736
Epoch  45 -> Total Error:  0.9194377409045109
Epoch  50 -> Total Error:  0.9188744965397393
<ipython-input-294-1e7df31fdca0>:49: UserWarning: color is redundantly defined by the 'color' keyword
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```

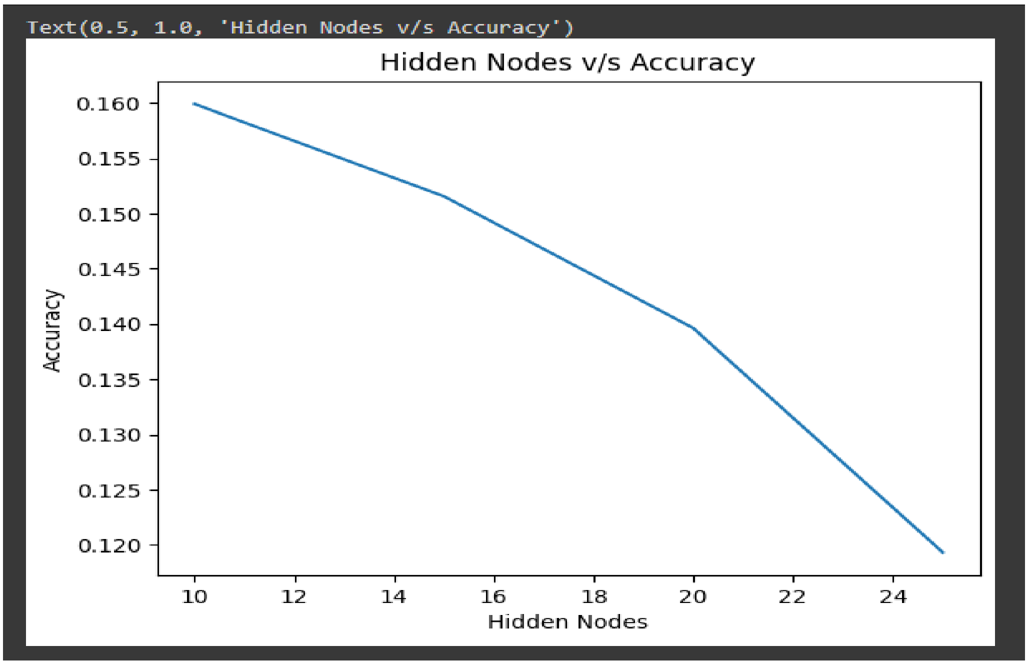



```
Accuracy with Constant wieght Initialization and Tanh Function is :19.690%
```

E.

Used 4 different values of hidden layer and found the accuracy of each model and got the following results.

```
Accuracies with 10 Hidden Nodes : 0.15990453460620524
Accuracies with 15 Hidden Nodes : 0.151551312649164
Accuracies with 20 Hidden Nodes : 0.13961813842482101
Accuracies with 25 Hidden Nodes : 0.11933174224343675
```

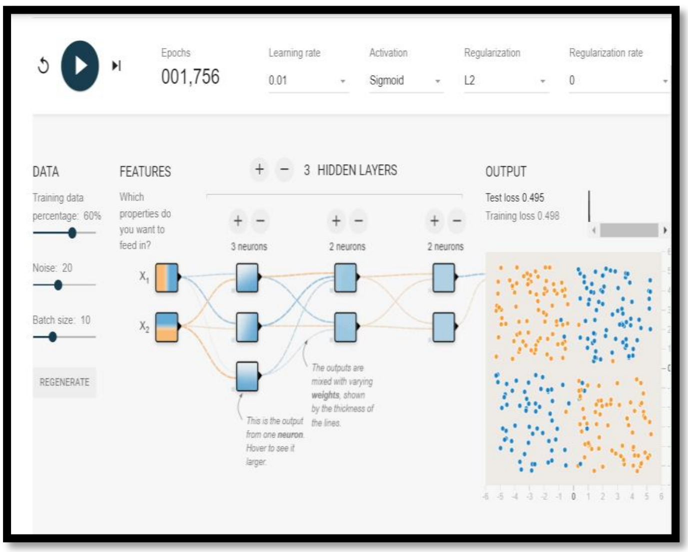


So, in general here, Accuracy decreases with increase in number if Hidden Nodes

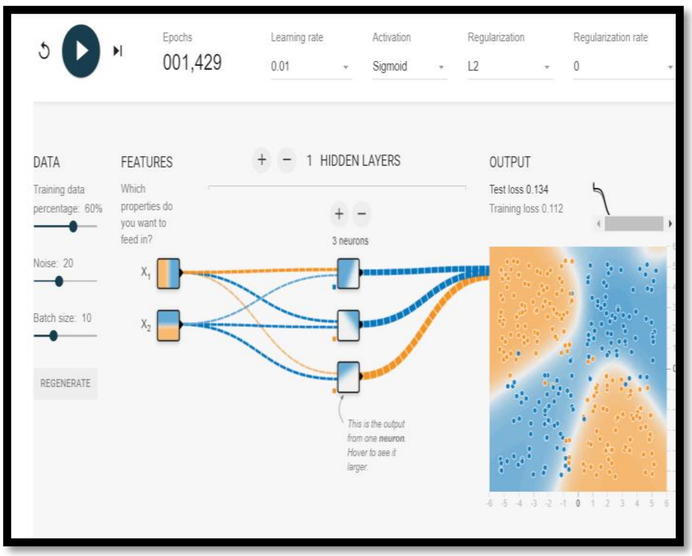
---

Question 3 (Experiments with Architecture)

a. Using small model showed better results. Small Model converged quickly and just after about 1500 epochs it gave good results, whereas Big size model did not converge even after 1700 epochs and shoed high error.

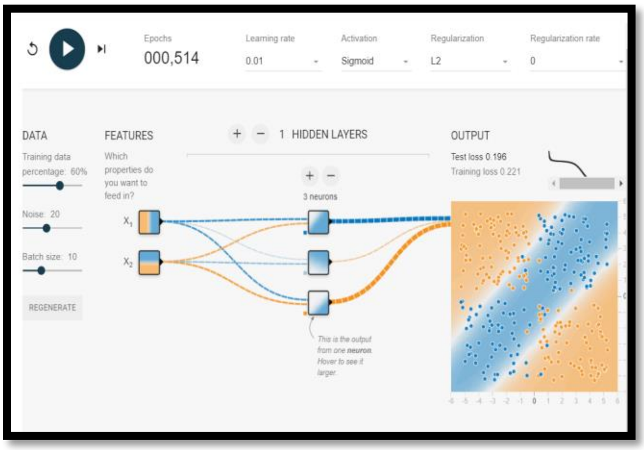


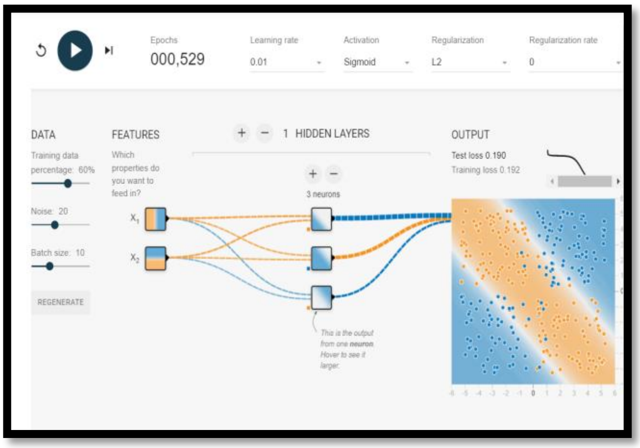
Big Model Size

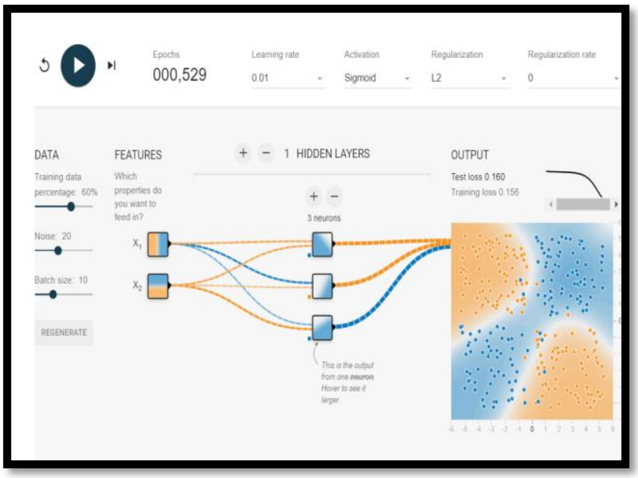


Less Model Size

b. Using random initializations, the model gave different fittings. This shows that random initializations can affect the accuracy and fitting of the model a huge extend.

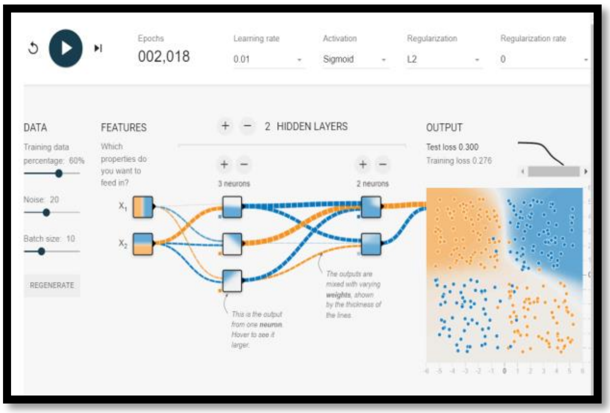


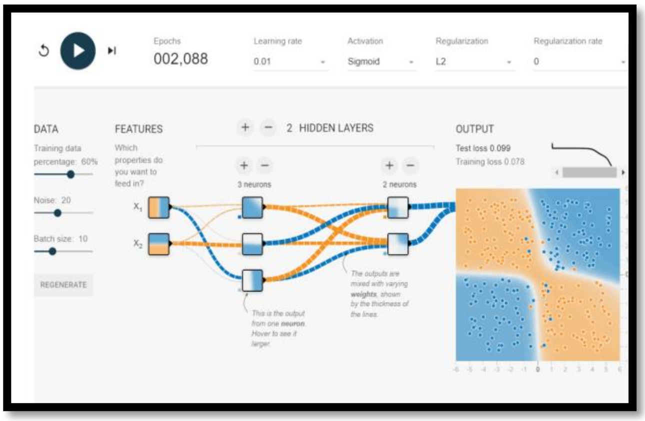


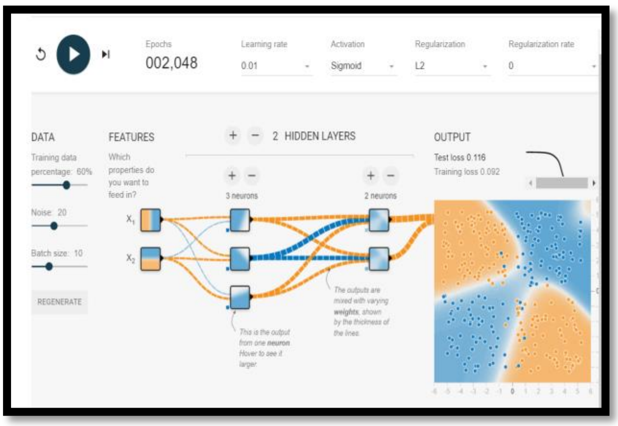


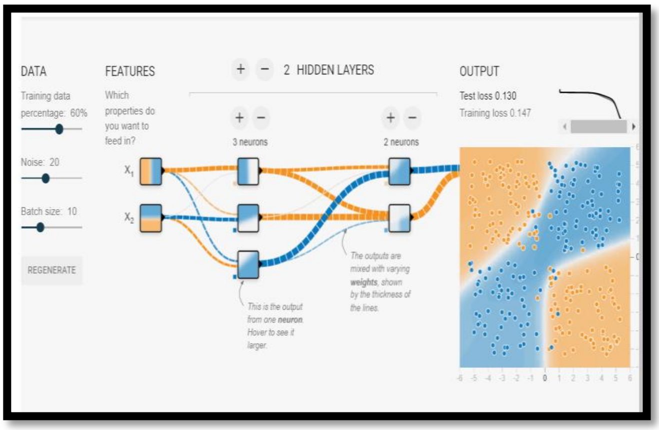
For big size model

The model also shoed different shape with different random initializations. In general the model took about 2000 epochs to converge that is very large with respect to small size model.

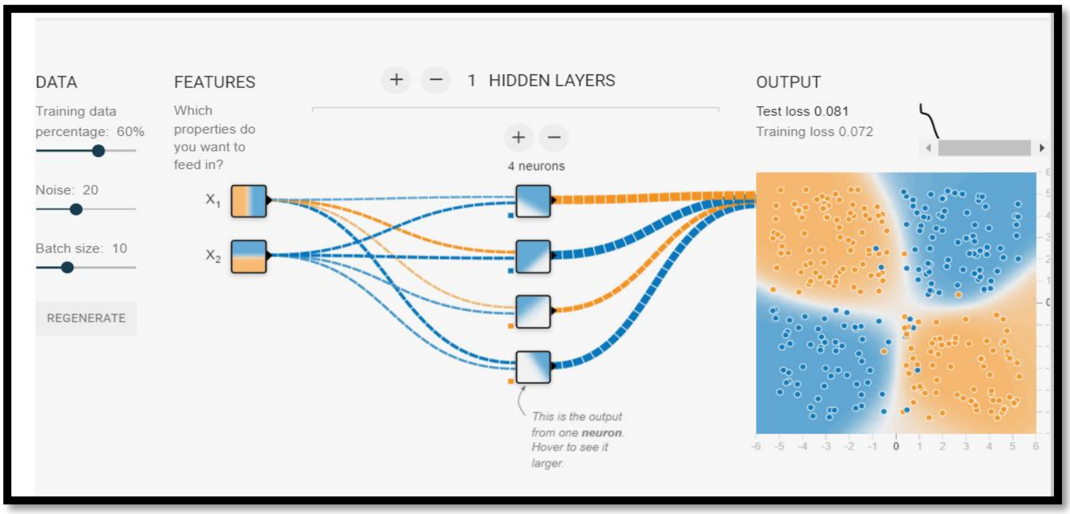








c.



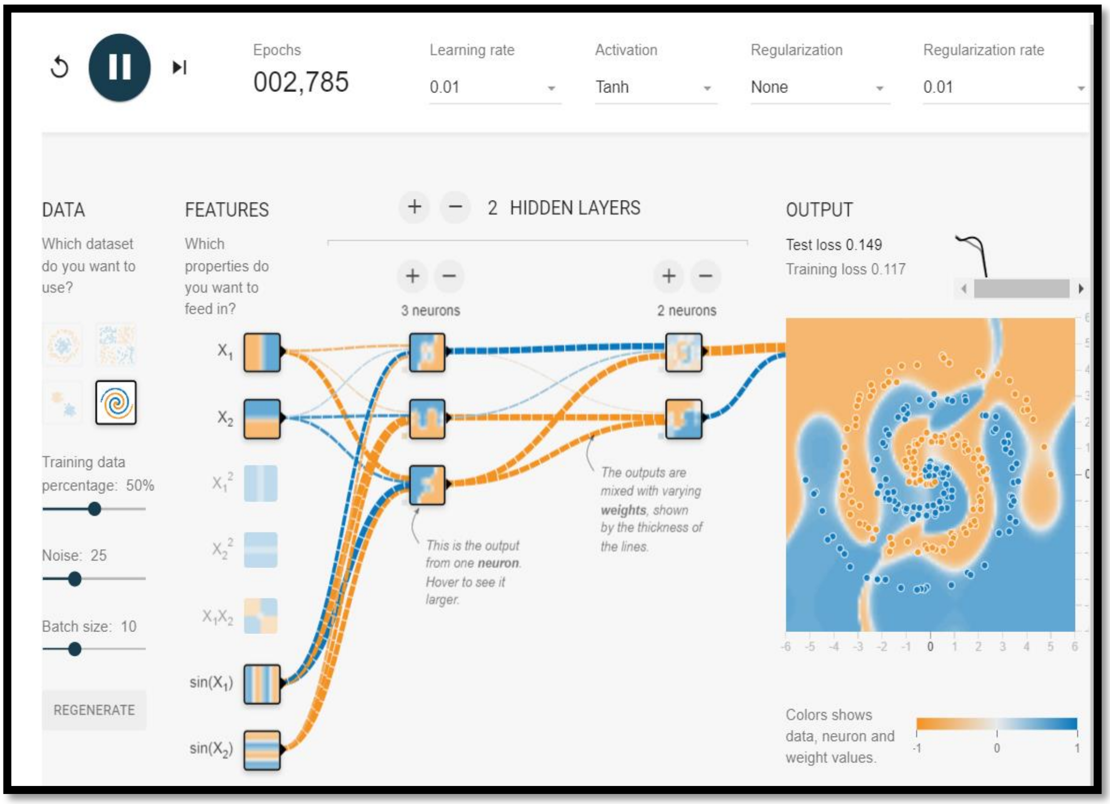
The best model I can get is following for noise 20 and 60% training data.

Hidden Layers = 1

Number of Neurons = 4

Test Loss = 0.081

Training Loss = 0.072



The best model I can get is Spiral Data for noise 20 and 60% training data.

Activation Function = Tanh

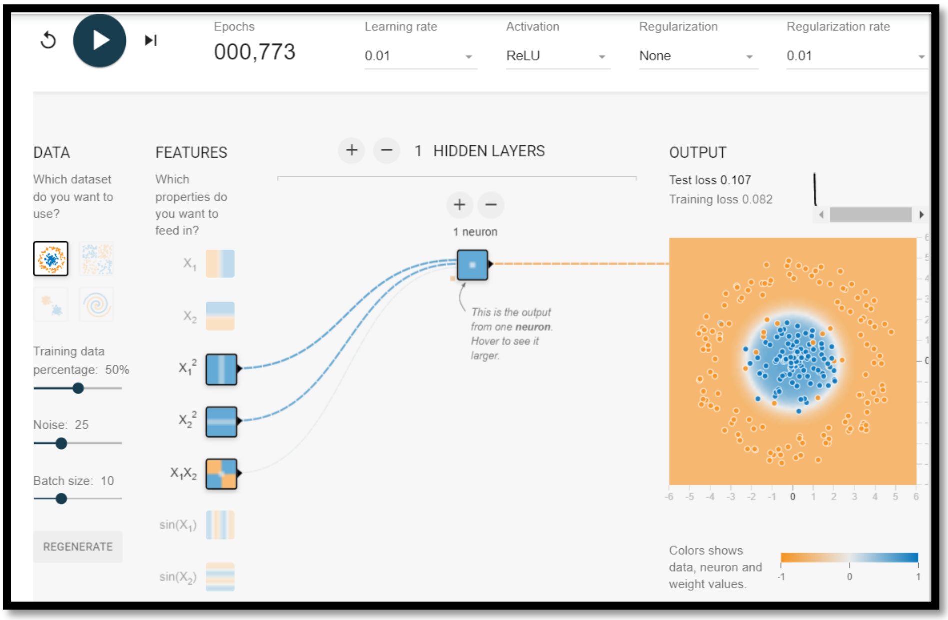
Hidden Layers = 2

Number of Neurons = 3, 2

Test Loss = 0.149

Training Loss = 0.117

Features Used = $\sin(x_1)$, $\sin(x_2)$



The best model I can get is Spiral Data for noise 20 and 60% training data.

Activation Function = Relu

Hidden Layers = 1

Number of Neurons = 1

Test Loss = 0.107

Training Loss = 0.082

Features Used = x_1^2 , x_2^2 , $x_1 x_2$

B.