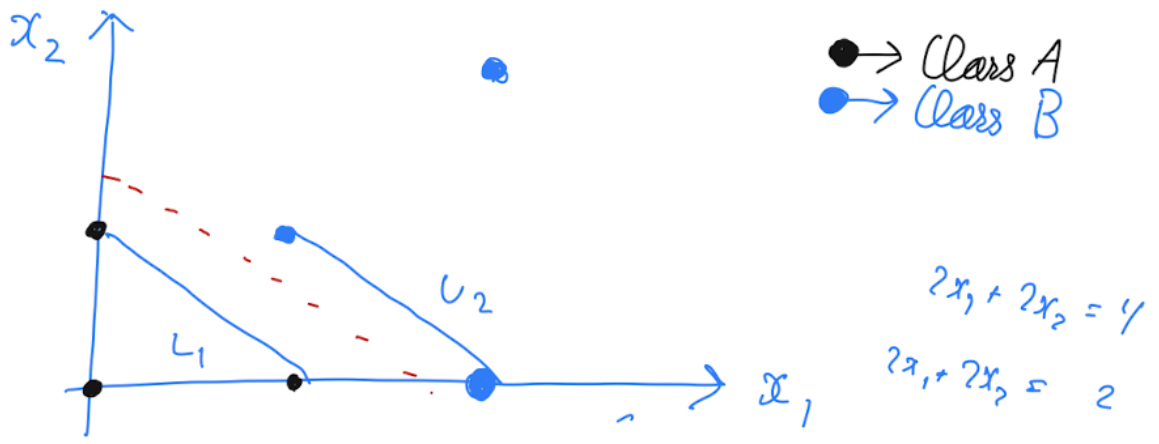
Section A:

Ans See A

Class	x_1	x_2
A	0	0
A	1	0
A	0	1
B	1	1
B	2	2
B	2	0

The data is follows



Yes, the two classes are linearly separable

2) $L_1 = x_1 + x_2 = 1 \Rightarrow 2x_1 + 2x_2 - 3 = -1$

$L_2 = x_1 + x_2 = 2 \quad 2x_1 + 2x_2 - 3 = 1$

The lines containing support vectors are as follows

$w^T x + b = -1 \Rightarrow 2x_1 + 2x_2 - 3 = -1$

$w^T x + b = 1 \Rightarrow 2x_1 + 2x_2 - 3 = 1$

We know that hyperplane has equation

$w^T x + b = 0$

$2x_1 + 2x_2 - 3 = 0$

The weight vector $\Rightarrow \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$b = -3$

Support vectors $\Rightarrow$ (0,1) $\Rightarrow$ Negative side
(1,0) $\Rightarrow$ Negative side
(1,1) $\Rightarrow$ Positive side
(2,0) $\Rightarrow$ Positive side

Ans 3) The effect will be dependent on which point is being removed

If (0,1) or (2,0) is removed then the hyperplane will remain the same and so will the margin

for other points will change the hyperplane and so will the size of the margin will also change.

Ans 4) The original boundary will depend on the position of the dataset spreaded. On removing any of the support vectors may or may not change the current decision boundary. Since removing the support vector will result in relaxing the originally existing constraints. The new constraints will have a tendency of having a margin greater than or equal to the current decision boundary.

Section B:

1)

Sample class is created by passing specific parameters to that class N: denoting the number of hidden layers, A: list giving sizes of each hidden layer, lr: learning rate, activ: activation function at each layer, init: weight initializer, epochs: mentioning the number of epochs, batch_size: specifying the mini-batch on which the model gets trained on. After getting all the appropriate values for the following parameters, it is followed by initialize function and returns a dictionary for each hidden layer initializing the weights, biases, activation functions, and backpropagation function. Since this is a multiclass classification, we the activation for last layer is softmax activation.

Various functions for activation are sigmoid- $\frac{1}{1+e^{-(x)}}$

Tanh: $\frac{2}{1+e^{-2x}} - 1$

Relu: $max\,(0, x)$

Leaky_Relu: $max\,(0.1x, x)$

Linear: X

Softmax : $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

The weights are also initialized with different initial values such as with zeros, with random values or with some normalization.

The fit function is used to train the data on the training values. The data is multiplied with weights and biases for preparing the data and then passed on to the activation function. After the loss function calculates the loss, then afterward, loss is estimated and based on that info, backpropagation takes place using backprop functions. This whole process takes place for mini-batches of data for epochs several times.

The predict function, once all the weights and biases are calculated they, is used to predict the outcome from the passed data. Since the last layer is softmax it will provide the probabilities against each class among the classes. The final label will depend on the class will depend on which class has the maximum likelihood.

The predict_proba function gives the outcome based on the input provided to it. Since the final layer has activation of softmax, it will have probabilities only, which will be returned by the function.

The score function uses predict function only to find the class corresponding to each training sample. By comparing it against the actual value, it finds the number of correctly classified samples. By dividing it by the total number of instances, we will get the accuracy of the model.

4) Fo the MNIST data, we were asked to train the data on hidden layers as follows = (256,128,64,32)

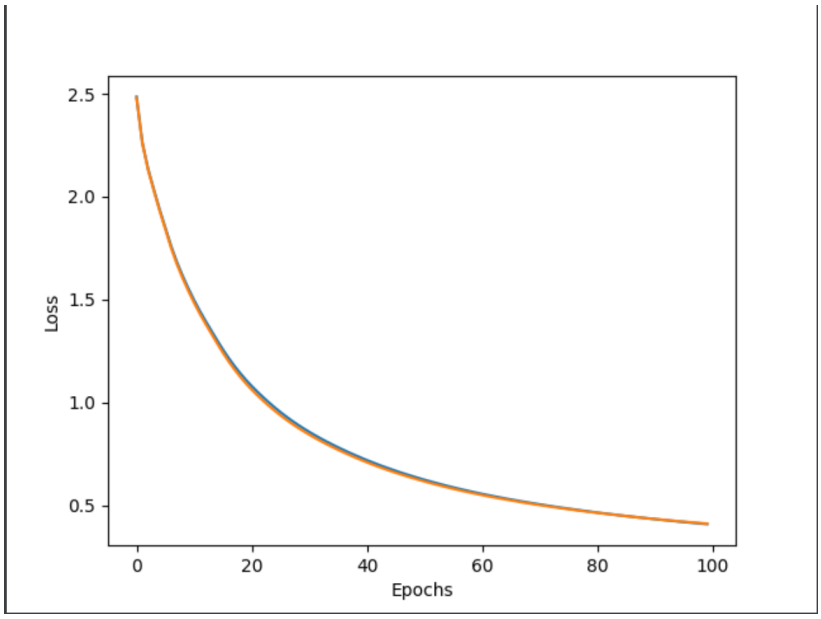
The number of epochs will be 100, and the batch size will be 128. We train this on a learning rate of 0.01 along with random initialization of weights.

The training and validation loss against the number of epochs are as follows:

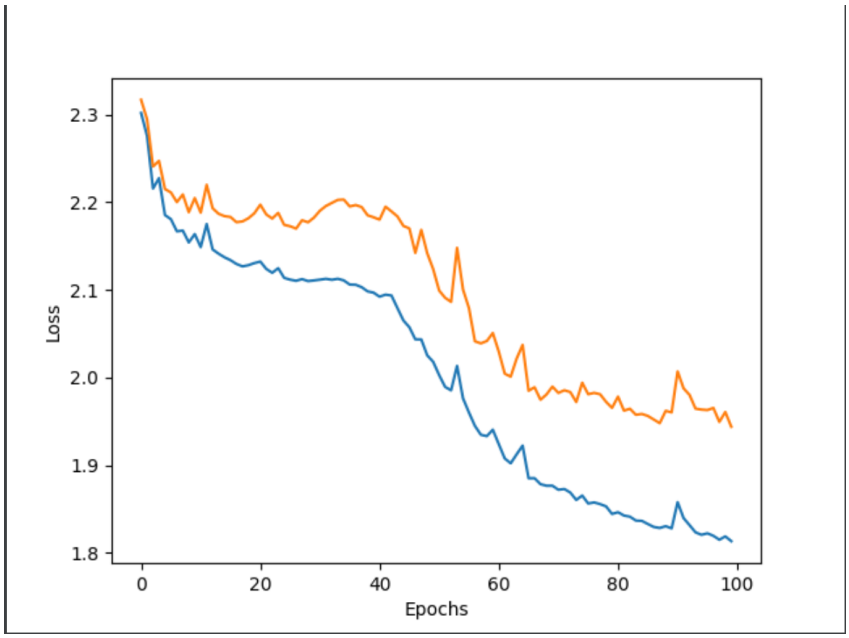
Training Loss-Blue,

Testing Loss-Orange

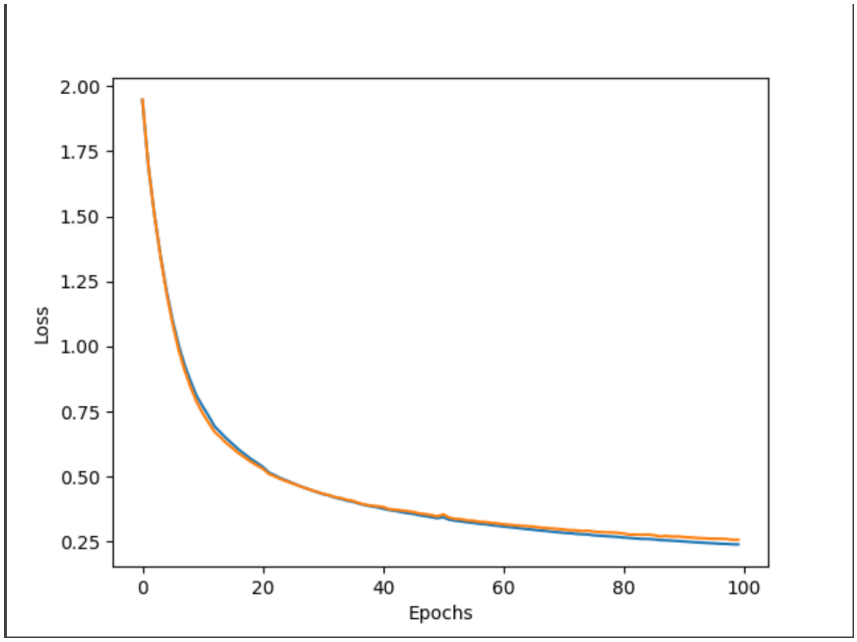
Sigmoid Loss



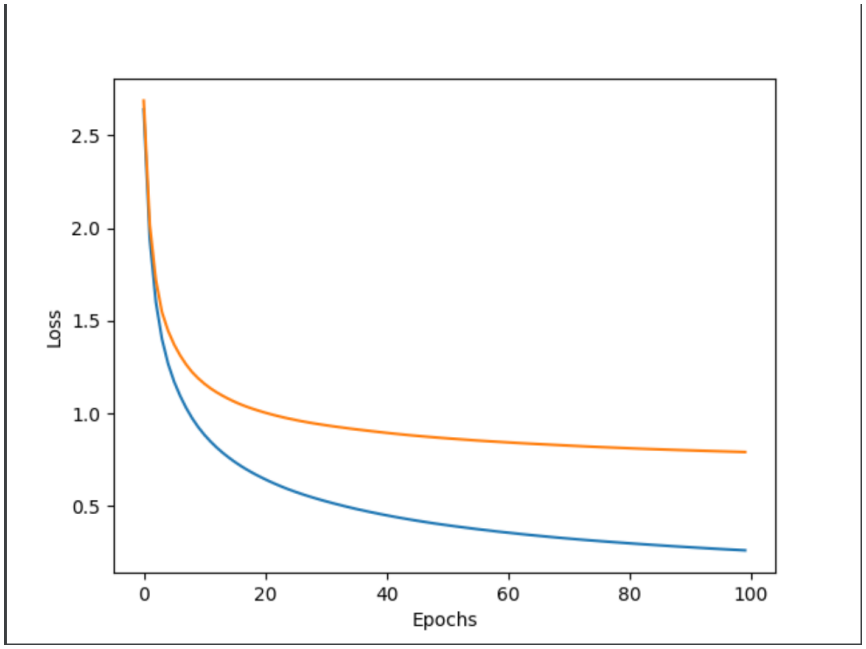
Relu Loss:



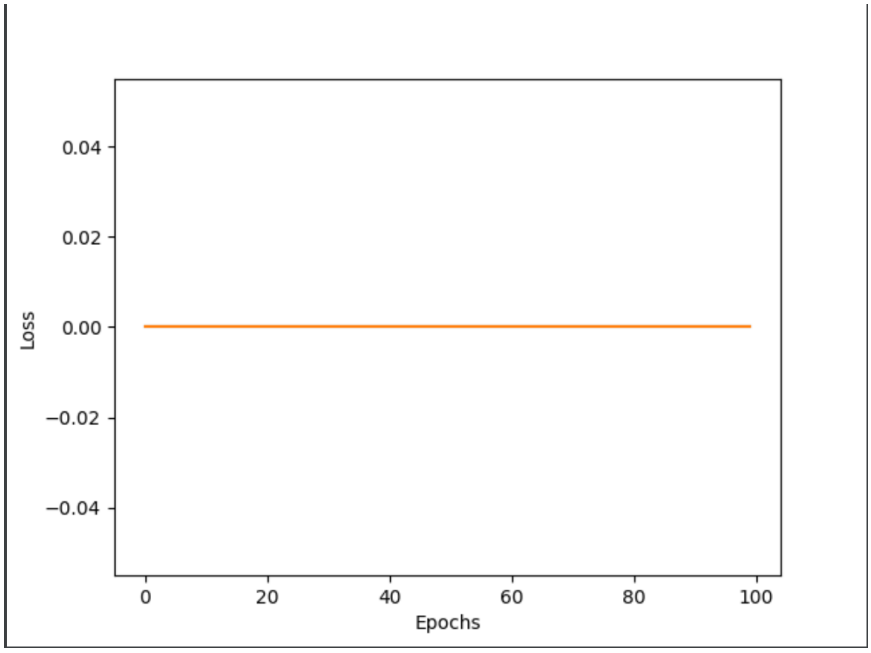
Leaky Relu loss:



Tanh Loss:



Linear Loss

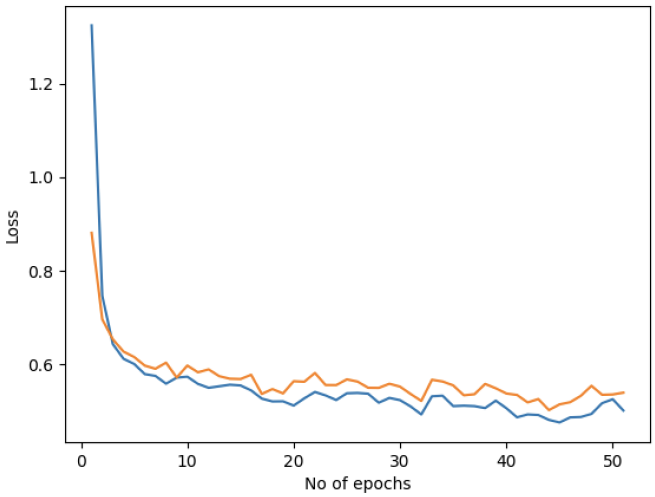


The Accuracy Scores are follows:

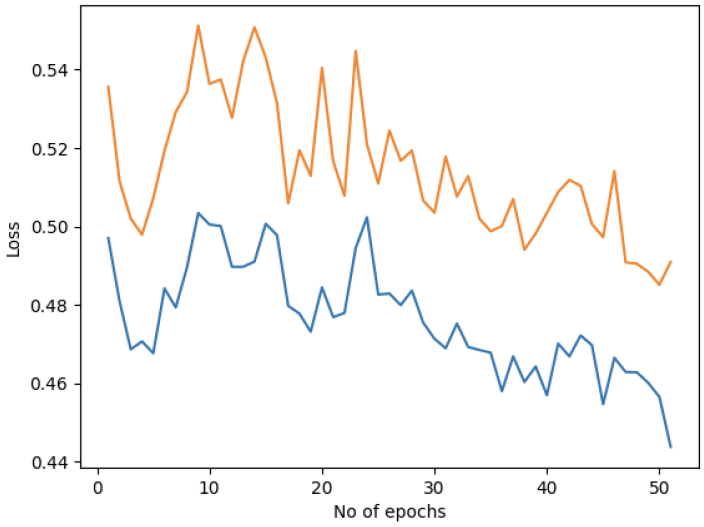
Model	Accuracy Score
Sigmoid	0.9017
Relu	0.2917
Leaky Relu	0.9246
Tanh	0.77
Linear	0.18

As we can see that Leaky Relu gives the best accuracy among all the models

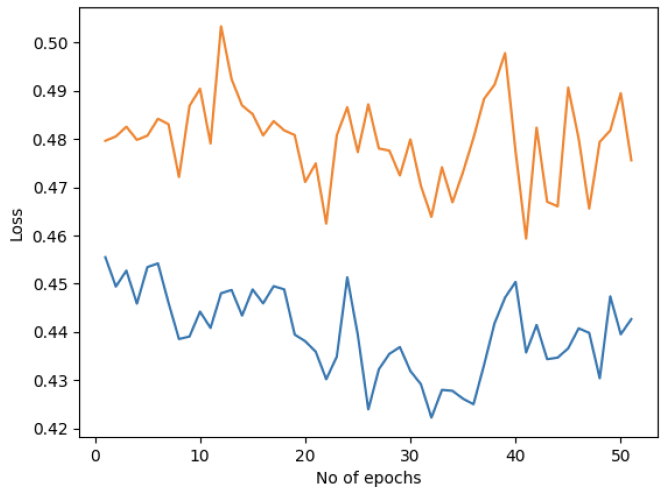
Section C:
The loss values vs. epochs graphs for each activation are as follows:
The best results achieved depend on the number of epochs. For 50 epochs, the best results are achieved from ‘tanh’ activation, but on increasing the epochs to 100, the best impact is given by ‘relu’ activation.
The graphs are as follows
Orange-Testing Loss
Blue-Training Loss
Case - Sigmoid Loss



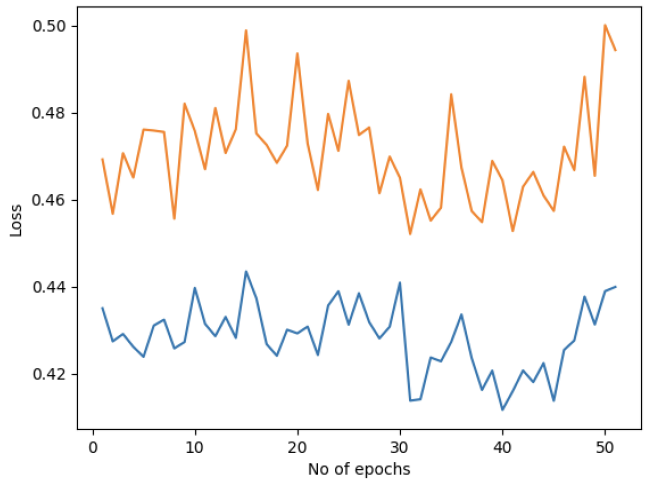
Case-Relu Loss



Case-Identity Loss



Case- Tanh Loss

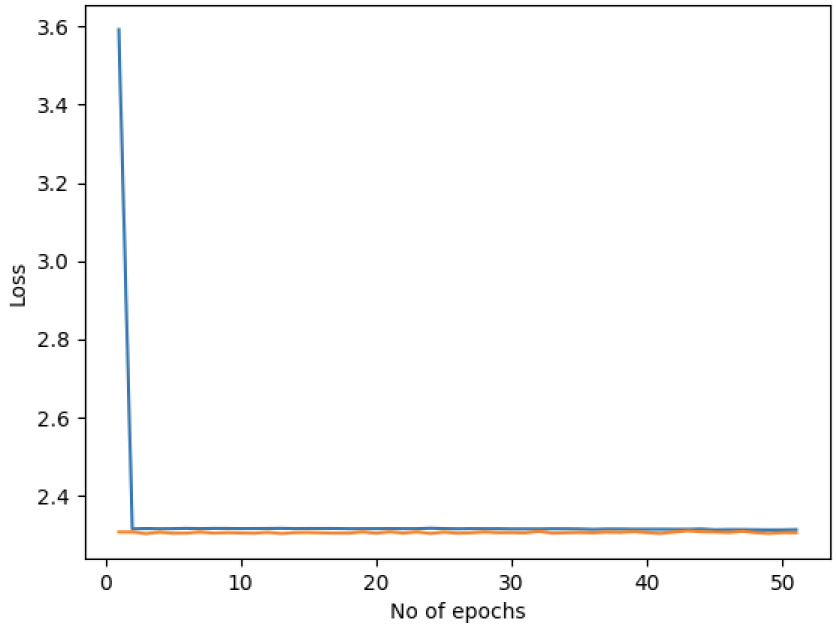


For the sigmoid and relu functions there is a decrease in the loss, but incase of the sigmoid it starts from a very high value and steadily reduces. In the case of relu function, it decreases but the graph is not steadily decreasing. In the case of the other two activation function the loss os fluctuates over the epochs. The accuracy in the case of 50 epochs is as follows:

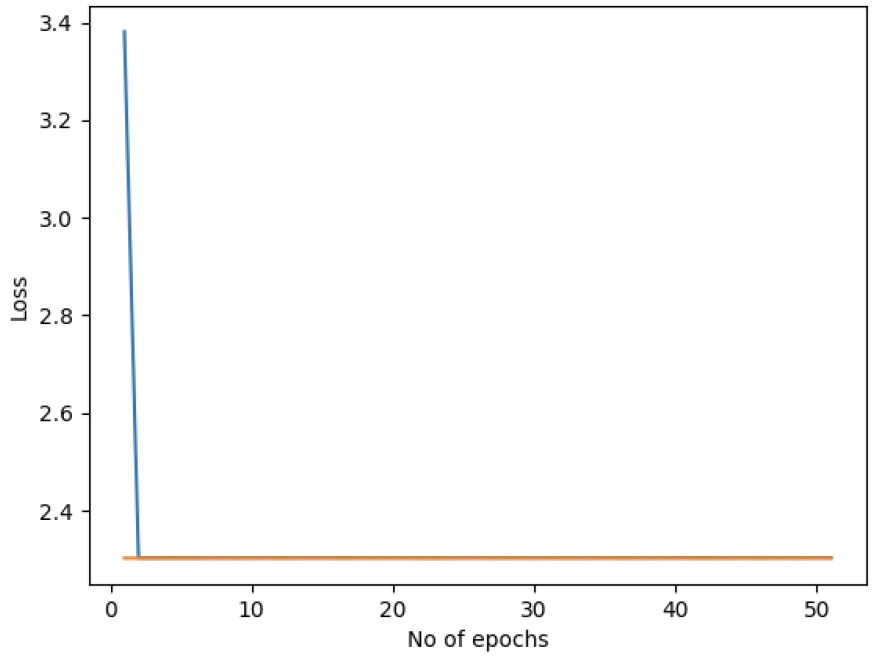
Activation Function	Accuracy
Sigmoid	0.813
Identity	0.814
Relu	0.840
Tanh	0.8329

Relu is the best to function among all the classifiers as it doesn't suffer from the problem of vanishing gradient over a huge range of values.

2)
Since we got the best results from relu we ran the classifier on 0.1, 0.01, and 0.001 with 50 epochs as the learning rate. The loss curves will be as follows.
Orange- Testing Loss
Blue- Training Loss
Rate-0.1



Rate-0.01



Rate-0.001

The results that can be seen from the graphs are as follows In the case of 0.1 and 0.01 loss, there is a sharp decrease in the loss in the case of the training, which is a result of a high learning rate. Whereas in the case of a low learning rate, the loss has a gradual decrease.

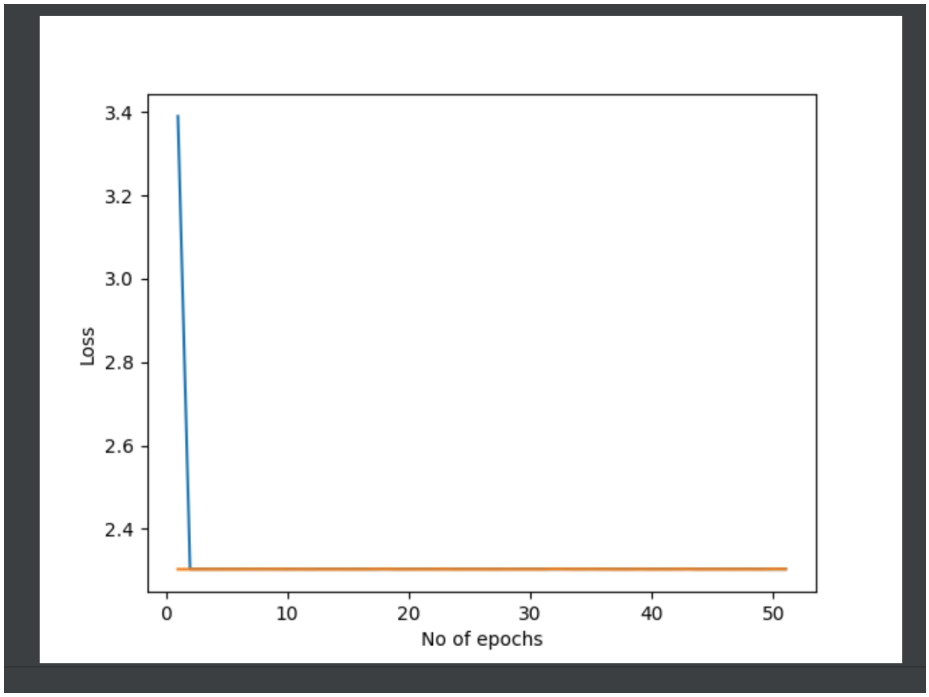
The accuracy score is as follows

Learning Rate	Accuracy Score
0.1	8.329
0.01	8.329
0.001	8.40

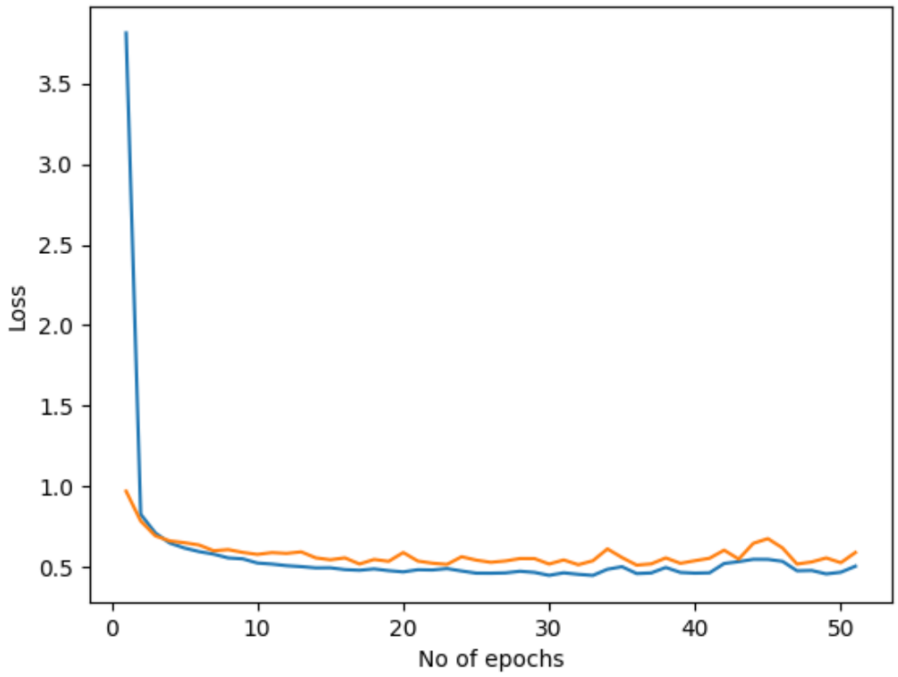
The best accuracy is when the learning rate is 0.001

3) For this part, we reduced the number of nodes on the hidden layer and observed the changes in the loss values against the epochs trying to find which gives the best accuracy:

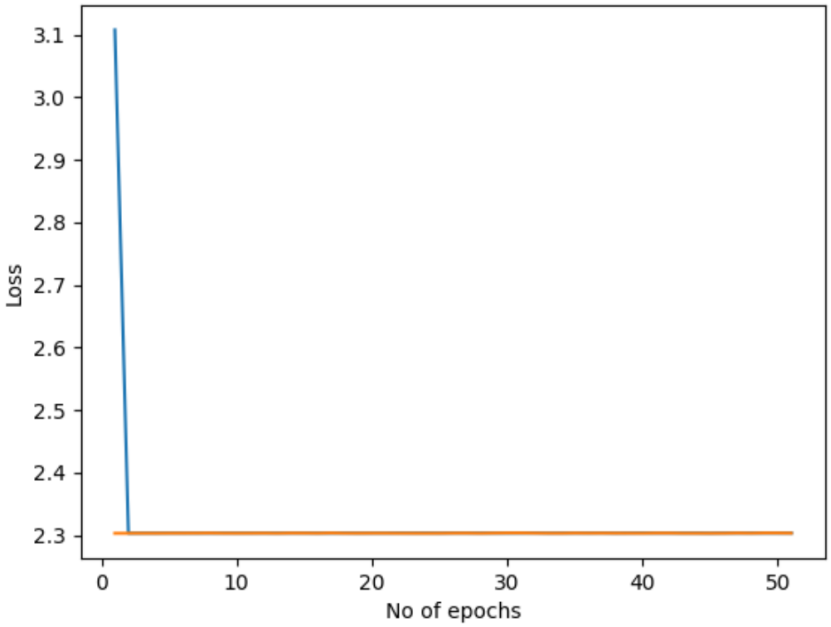
Hidden Layer size 1= 128



Hidden Layer size 2=64



Hidden Layer size 3=32

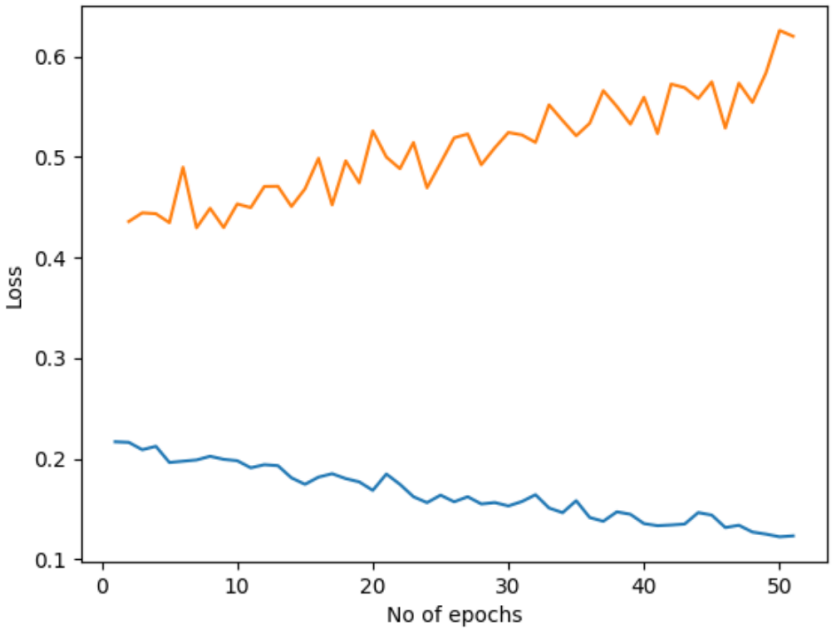


The accuracy scores are as follows:

Sizes of Layers	Accuracy score
(128,32)	0.818
(64,32)	0.8329
(32,16)	0.823

From this we got that the loss in the case when the number of nodes is (64,32) is the least as well as it is decreasing gradually whereas in the other two cases the loss drops rapidly and after that, it remains constant. The best layers size is given by 64 and 32 nodes.

4) For Grid Search CV, we try to find the best possible hyperparameters that can be passed on to the model that provides the best possible accuracy. For this particular case, the parameters passed on to the models were activations, including - sigmoid, relu, tanh, and identity. Another parameter that was tuned included the learning rate, which is varied with the values of 0.1, 0.01, and 0.001. The final classifier returns the model which gives the best accuracy among all and returns it.



This is the loss curve given by the best classifier which in turns out to be relu with a learning rate of 0.001. The loss curve also tells us the loss is pretty much constant and also remains less than 1.
Accuracy achieved for this is .840