CSL2050 PATTERN RECOGNITION AND MACHINE LEARNING

LAB REPORT-09

NAME:	SAMEER SHARMA
ROLL NUMBER:	B21CS066
LAB TITLE:	Neural Networks

Q1)

Part 1)

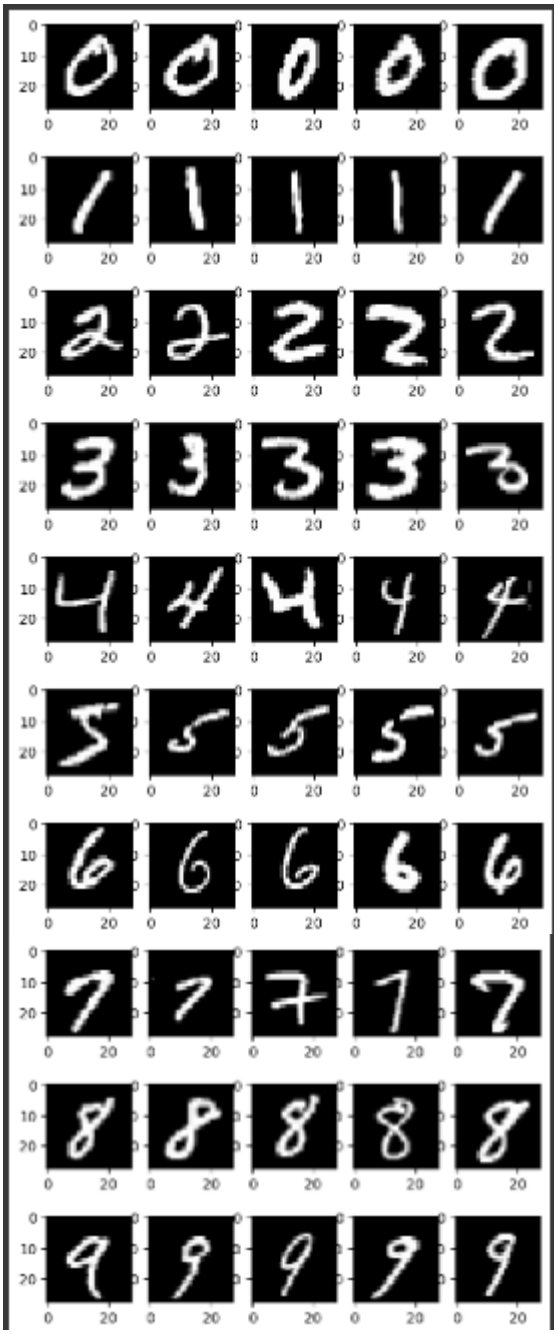
- Downloaded the MNIST dataset
- Calculated mean and standard deviation for normalizing the data
- Splitted the training dataset into training and testing dataset
- Applied the required transformation on training, validation and testing dataset

Part 2)

- Plotted five images from each class (0-9)
- Created data loader for training, validation and testing dataset with batch size of 64

Part 3)

- Created a multi layer perceptron (MLP) using the pytorch library.
- There is one input layer, one output layer and two hidden layers in MLP. All the layers are linear layers.



- Created a MLP object having 784 (28*28) nodes in input layer (size of the image), two hidden layers with 256 nodes each and an output layer with 10 (number of distinct classes) nodes.
- The total number of total trainable parameters for the above object are as follows: `No of trainable parameters are  269322`

Part 4)

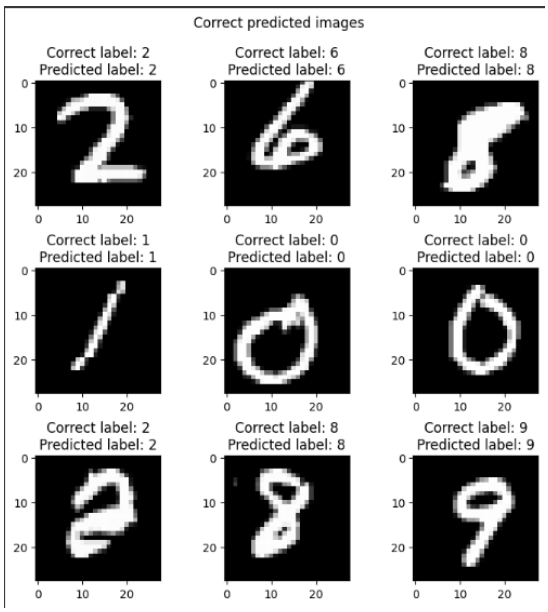
- Trained the model on the training dataset for 5 epochs. For training, optimizer is Adam optimizer and Loss function is cross entropy loss. Well commented code for the same is in the colab notebook.
- Evaluation of the model on the validation dataset for each epoch is as follows:

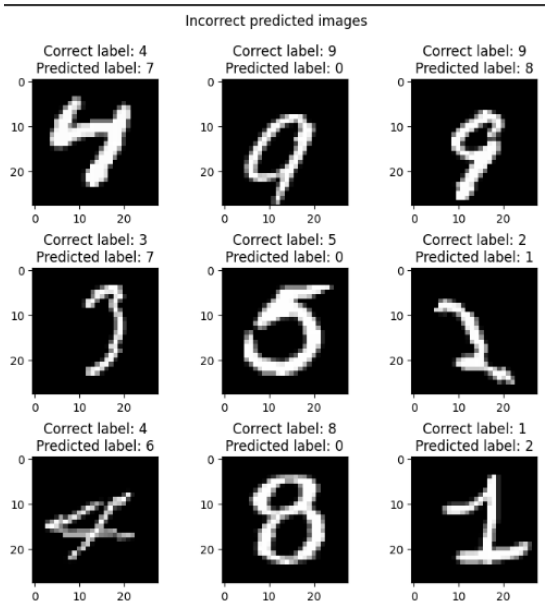
```
Epoch 1/5: train_loss: 0.427648909392811, train_accuracy: 0.8658571428571429, val_loss: 0.19254267569382985, val_accuracy: 0.9387222222222222
Epoch 2/5: train_loss: 0.17658629707601808, train_accuracy: 0.9444761904761905, val_loss: 0.18608793167273205, val_accuracy: 0.9438333333333333
Epoch 3/5: train_loss: 0.14374302809153283, train_accuracy: 0.9548333333333333, val_loss: 0.13693338375290234, val_accuracy: 0.9596111111111111
Epoch 4/5: train_loss: 0.12217633931977408, train_accuracy: 0.9611666666666666, val_loss: 0.1228022087348832, val_accuracy: 0.9623888888888888
Epoch 5/5: train_loss: 0.11241747491771266, train_accuracy: 0.9648095238095238, val_loss: 0.12568898743225468, val_accuracy: 0.9627222222222223
```

Clearly, both training and validation loss is decreasing after each epoch while accuracy is increasing.

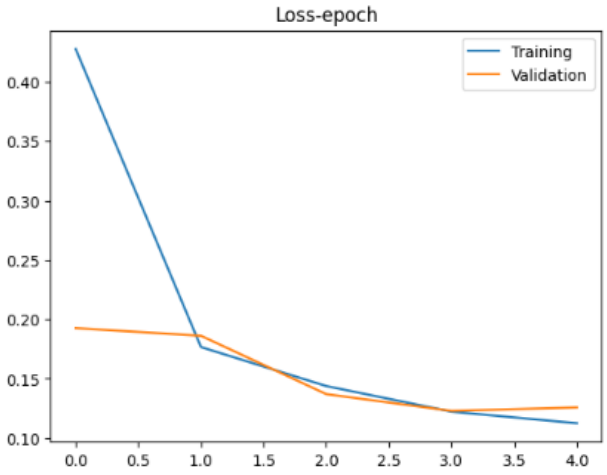
Part 5)

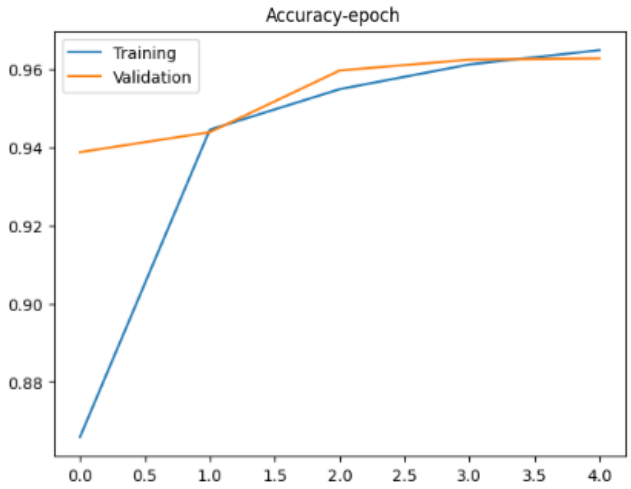
- Images which model predicted correctly vs the images that model predicted incorrectly





- Loss Epoch and Accuracy Epoch graphs for validation and training dataset are as follows:

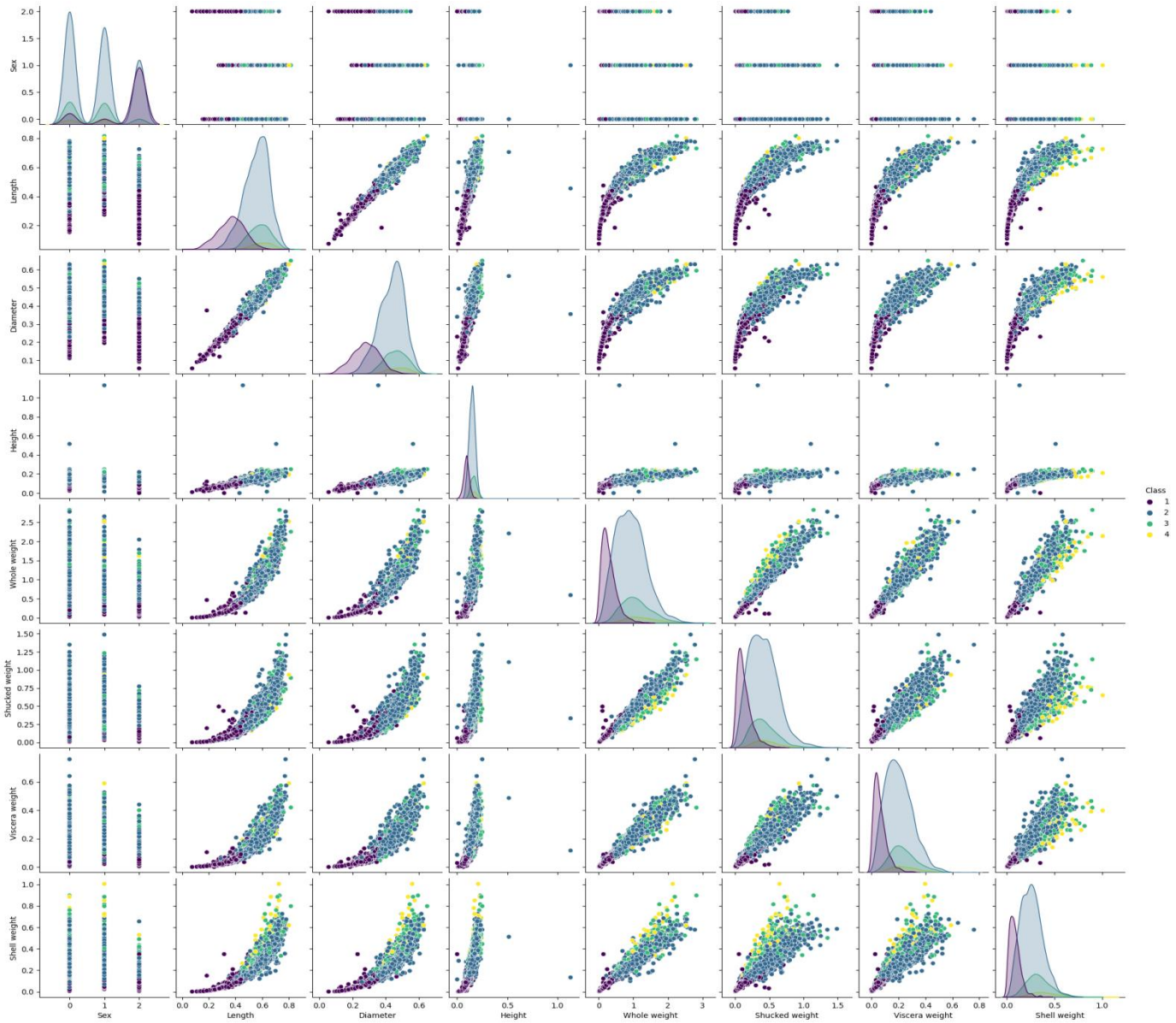




Q2)

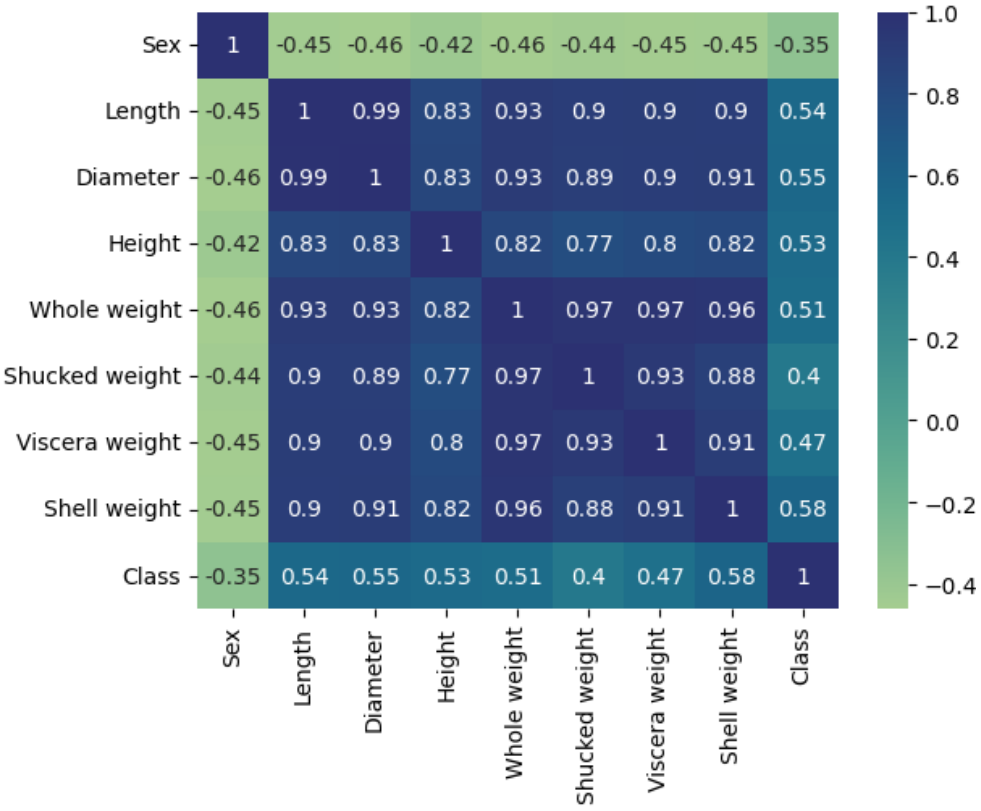
Part 1)

- Loaded the abalone dataset and encoded the categorical features.
- Since there are no missing values, no need to drop any rows or columns
- The objective is to train a classifier that predicts age of abalones on the basis of the number of rings. So, I defined four intervals and thus classes for classification purpose. Sample with lower class label is younger. The four intervals are [1,7], [8,12], [13,17] and [18,29]. The class label will give us a rough idea about the age of abalone whether it is immature, young or mature.
- Visualization of the dataset is as follows



From the plots, we can infer that older abalones have other features also greater than the younger ones that is obvious (skip sex feature here since it is categorical).

- The above fact can be confirmed from the correlation heatmap of the dataset. From the heatmap, we can infer that (exclude sex feature since it is categorical), features are highly correlated to each other. Also, higher the features greater the age of alabone.



- In the last, splitted the dataset into training, validation and testing dataset in the ratio 50:20:30. In place of normal split, I did stratified split in which samples belonging to each class gets splitted in the same ratio and thus each class have representation in both the datasets. Its well commented code is in the colab notebook.

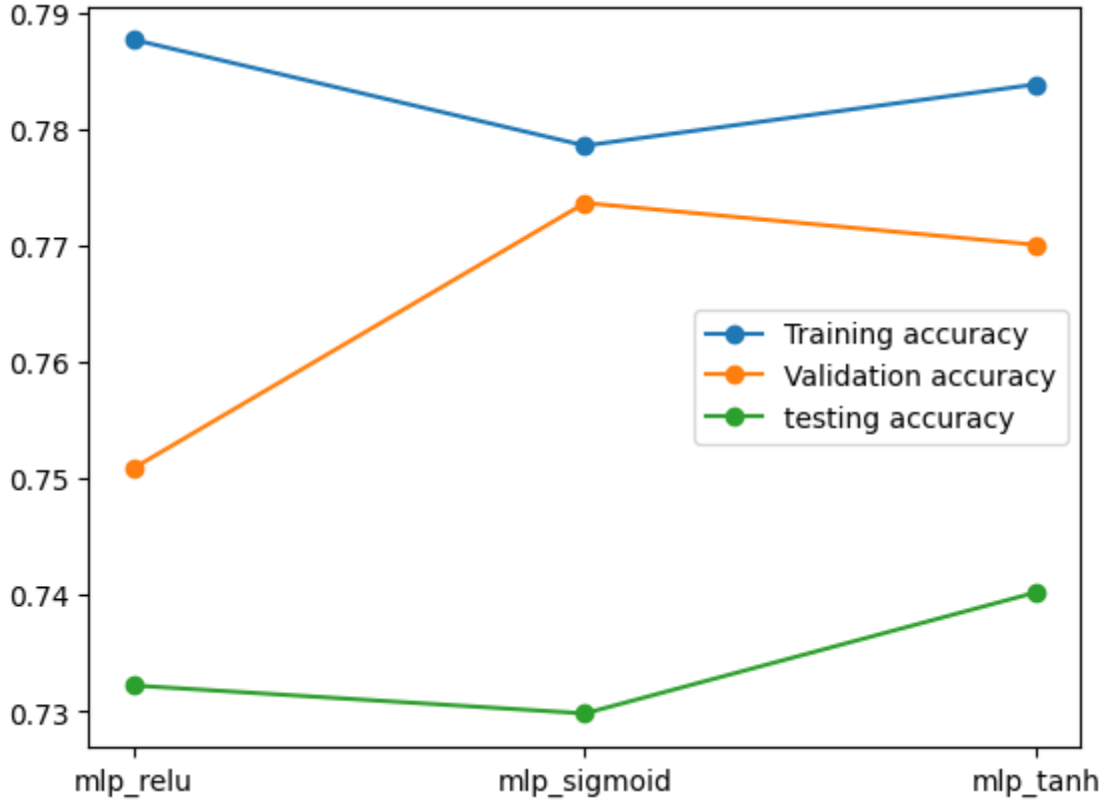
Part 2)

- Implemented a neural network from scratch using only numpy library. The well commented code for the same is in the colab notebook.
- Build a neural network trained on the abalone dataset and evaluated its accuracy on Training dataset

```
Final Training accuracy:  0.787661406025825
```

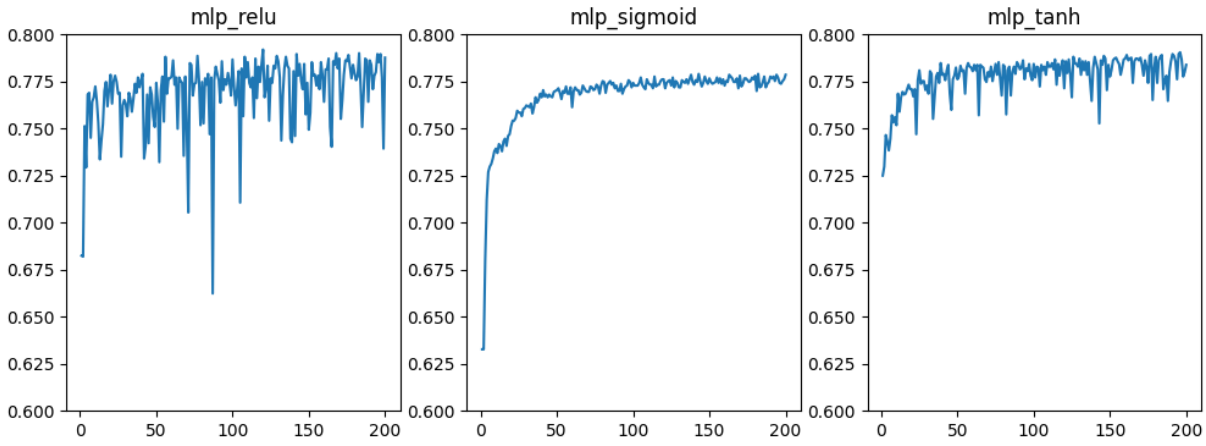
Part 3)

- Implemented three activation functions namely ReLu, Sigmoid and Tanh from scratch.
- The accuracy plots for the three activation functions after the completion of final epoch/iteration are as follows



Clearly, the accuracies for the three activation functions are nearly same.

- But when we observe the plot of accuracy vs epoch/iteration graph we find something different

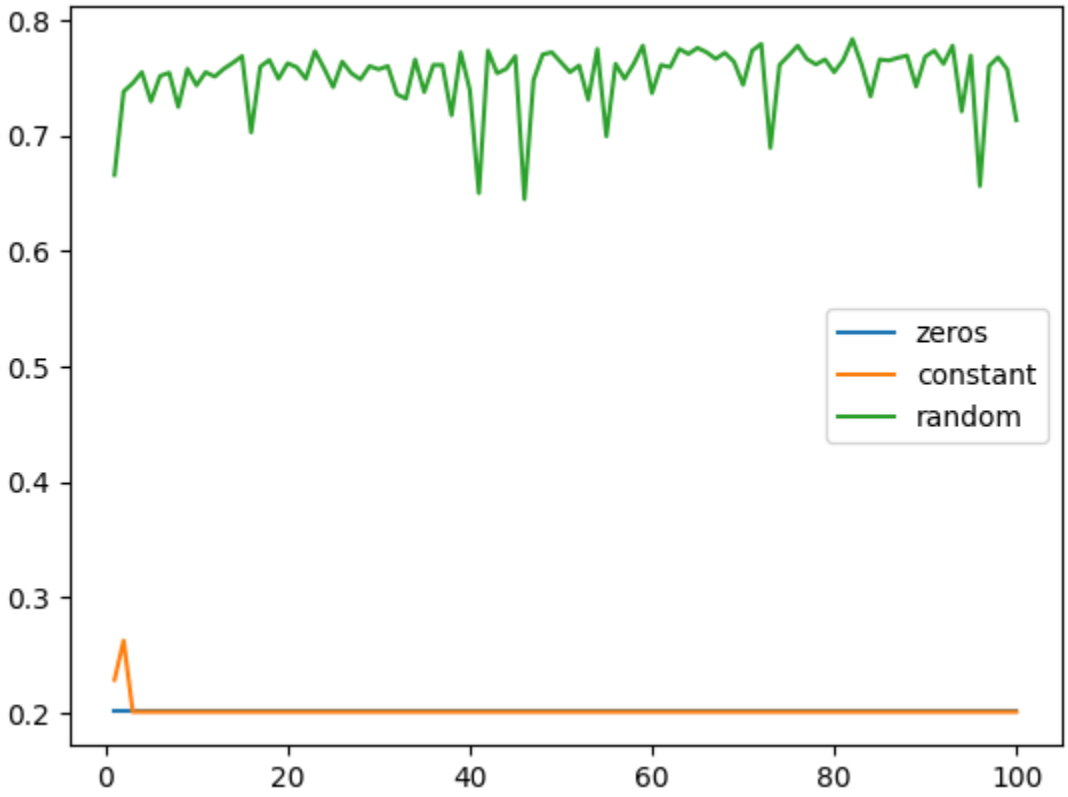


The accuracy for mlp using ReLu activation function is fluctuating very much and not stable while that for sigmoid activation function is pretty much stable. Tanh activation function performs nearly in middle of the remaining two.

Also, the accuracy for ReLu increased very high and reached its convergence accuracy in lesser time when compared to the other two even when the learning rate is same.

Part 4)

- Plot of training accuracy vs epoch/iteration for all the three weight initialization techniques i.e. random, zero and constant

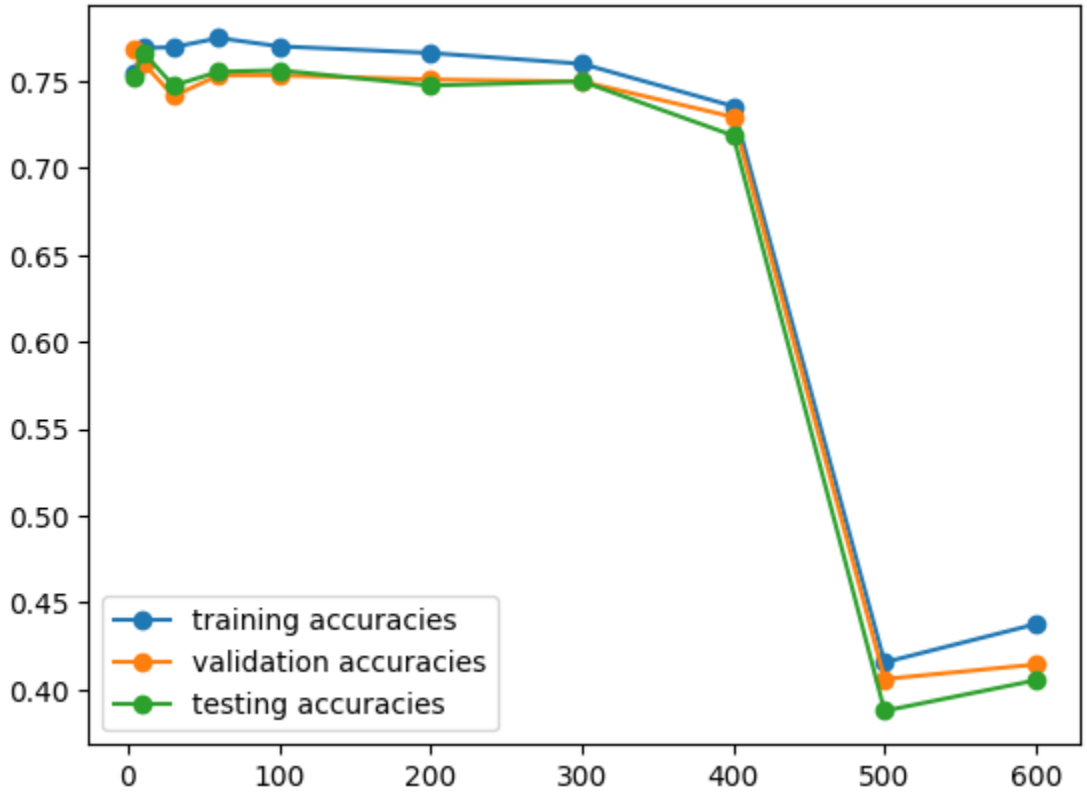


Clearly, accuracy for constant and zero weight initialization is very low while that of random initialization is pretty well.
This is because if all the weights are initialized to a constant value, the derivatives will remain same for each and every weight in the neural

network. Thus, it will learn same feature in each iteration and thus will give us low accuracy.

Part 5)

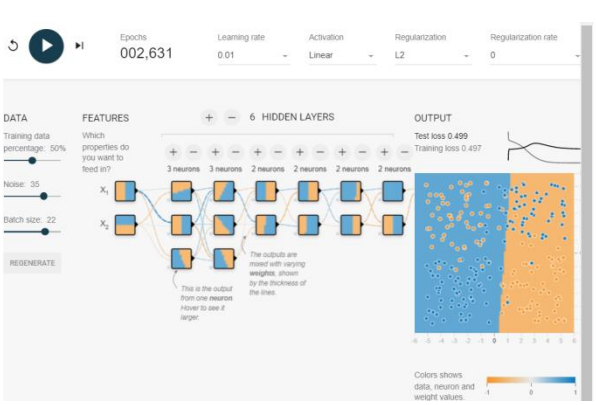
- Plot of accuracy vs number of nodes is as follows:

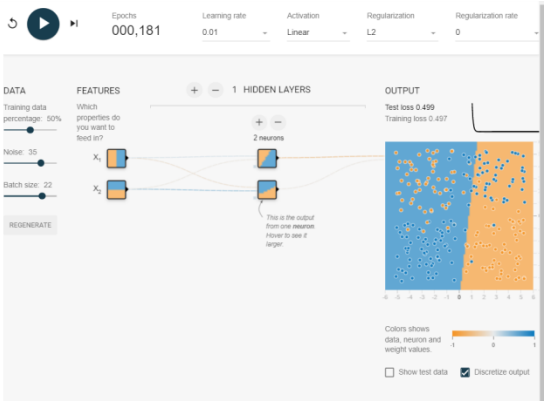


Clearly, the accuracies are nearly same for low number of nodes, the accuracy is nearly constant with small deviations but for higher nodes it is decreases. This is so because, as we increase the number of hidden nodes, the number of epochs/iterations to needed to reach convergence also increases. Since number of iterations is set to hundred for all the models in plot, the models with higher number of nodes didn't converges and we get low accuracy.

Q3)

Part 1)



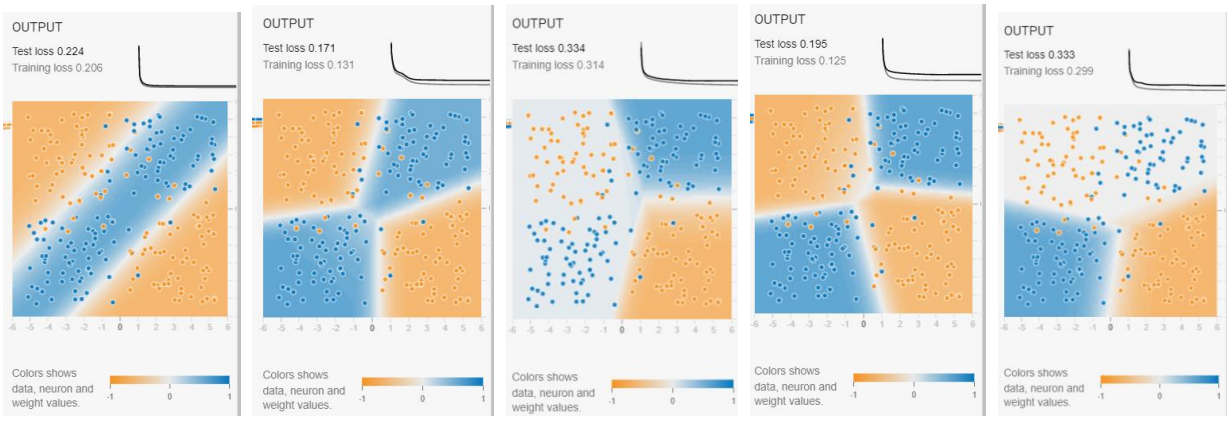


As one can observe from these two decision boundaries, both are nearly same with similar train and test loss. However, complex model took more epochs to fit on the data while less complex model took very few epochs. Since the boundary here is linear, with the given training dataset we cannot exceed accuracy of 0.5.

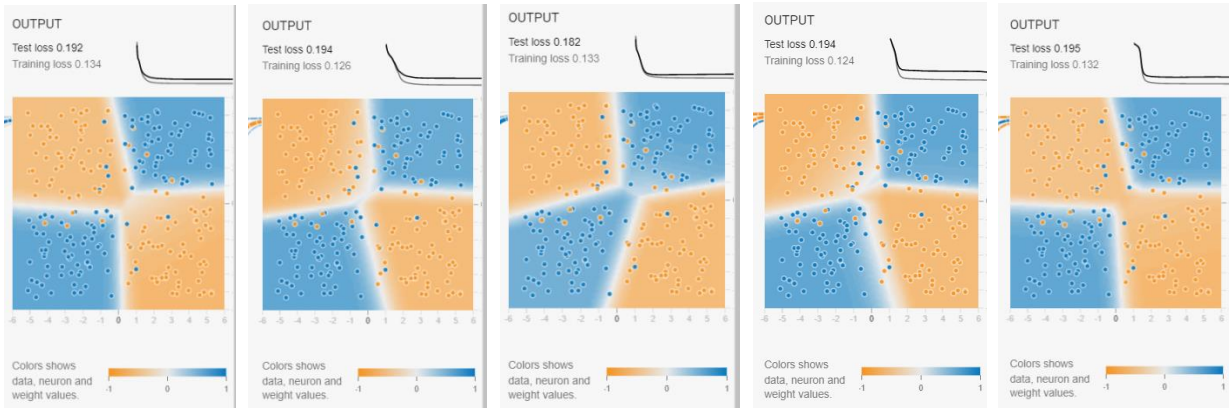
In general, if we have less layers and nodes, model may underfit the data and if we have more complex model it will overfit. In both the cases, we will get very less accuracy.

Part 2)

From the plots below, we can observe that shape of decision boundary is very different from each other with different train and test losses. In non convex optimization, initialization plays a very crucial role in deciding the shape to which the model will converge.



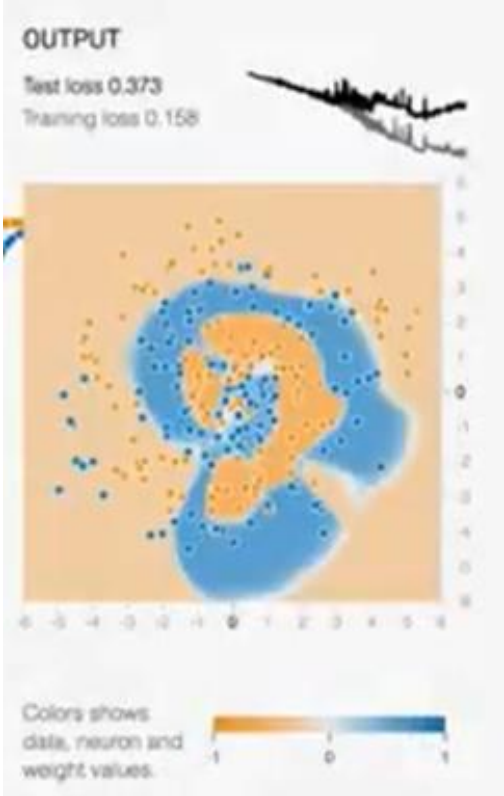
Now, after adding a new layer with three nodes in it, we can observe stabilization in shape to which model converge and now they are resembling each other more than the previous case.

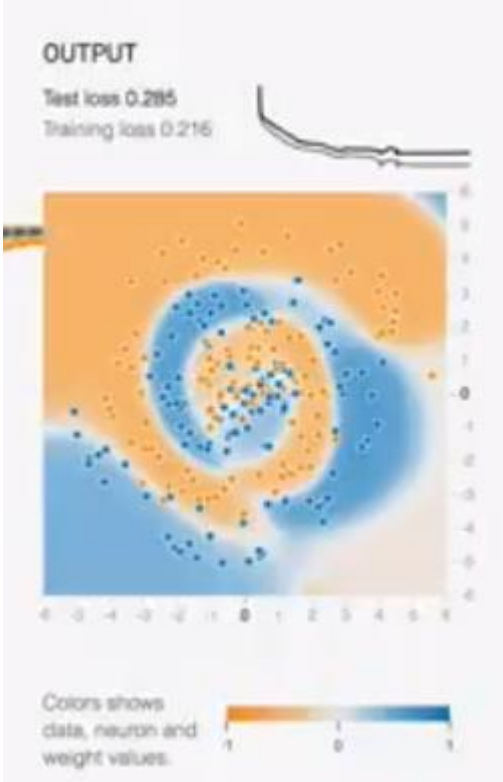


Part 3)

I created a neural network by adding many layers and nodes to it. It took around 3000 iterations to become stable but still it is not fitting the spiral data well and the test loss curve is also not smooth. The boundary is smooth but it does not resemble a spiral. Also, the model got overfitted (test loss is twice as much as train loss). The test loss here is 0.373.

In the second case, I also gave $\sin(X_1)$, $\sin(X_2)$, $X1^2$, $X2^2$ and X_1X_2 in the first layer. After fitting it I got better accuracy and also the shape looks like a spiral and thus fitting the data very well.



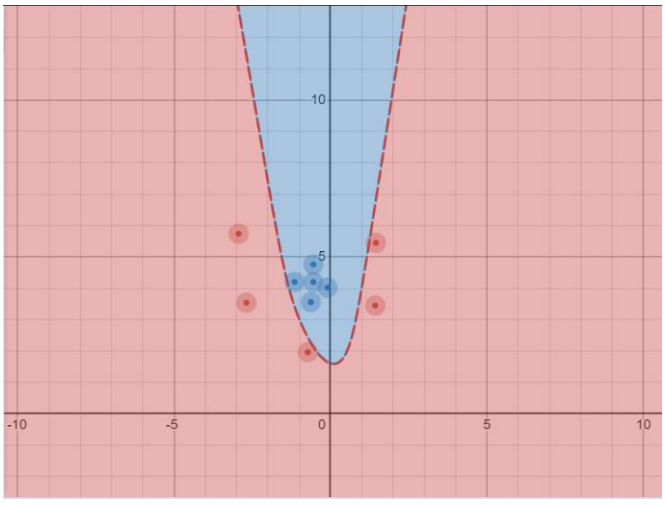


Without sin function

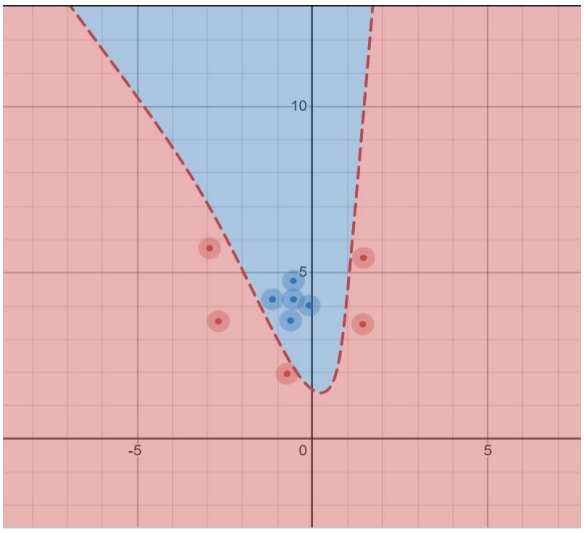
With sin function

Part 4)

With higher learning rate, the graph is very unstable and changes many times within a second. However, with lower learning rate, the plot is very stable and not changing with time after converging. Although it took more time to converge than that with higher learning rate.



Learning rate: 0.5



Learning rate: 0.25