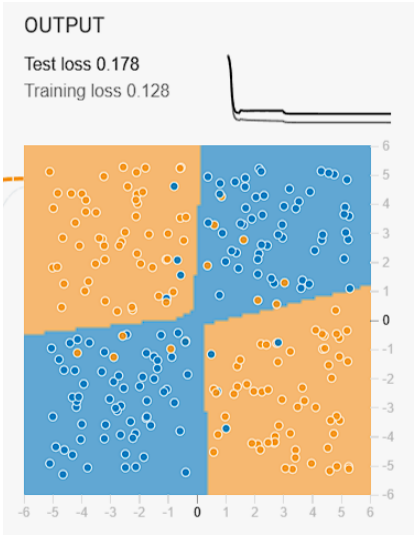
Lab Assignment 9
Prakhar Gupta
B21AI027

Question 1:

Part a-

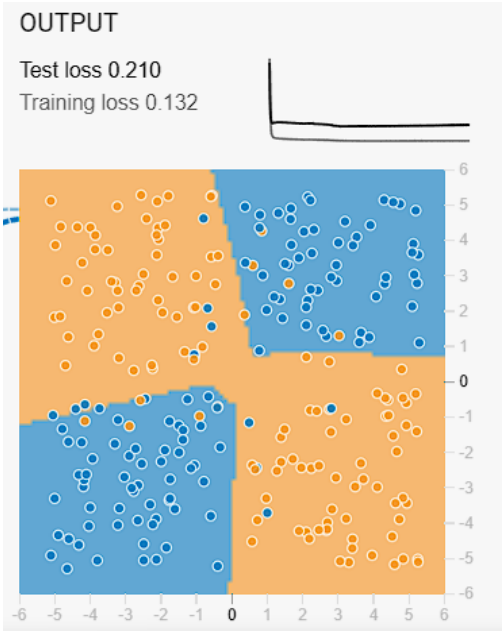
- Yes, the model starts learning non-linear functions and makes a better fit on the model (just that it is not working with linear activation function). It takes more time for the model to predict a good decision boundary(converging). Yes, complex models tend to converge to a good decision boundary

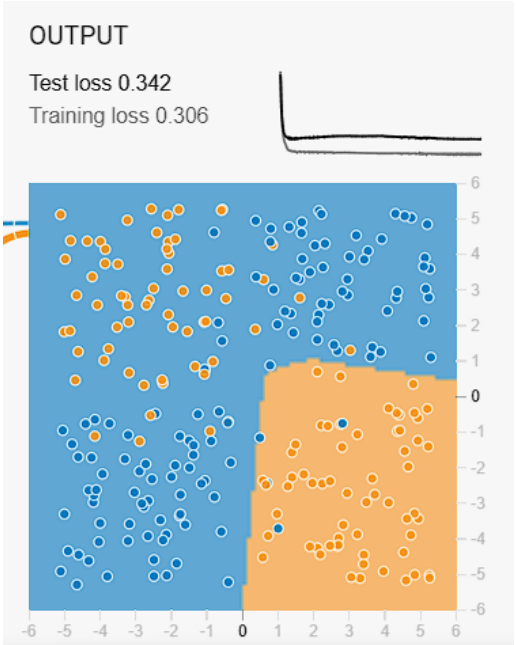


Part b-

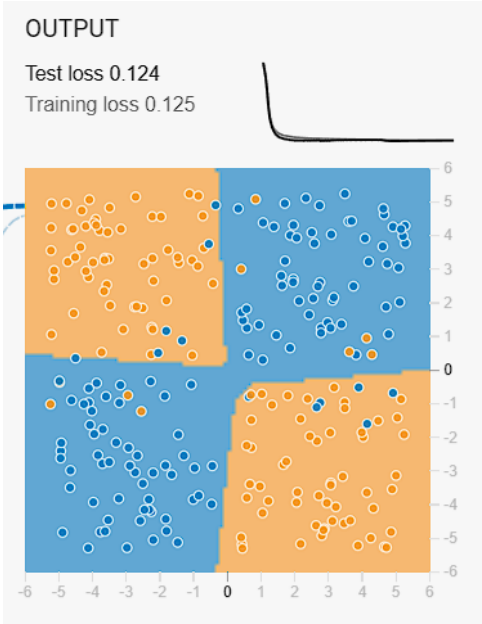
- The shape that each model converges to

- The initialisation parameter in non-convex optimisation does not affect the results, given it is trained for a long time
- By increasing model size , the decision boundary starts out to be completely different for different parameter initialisation

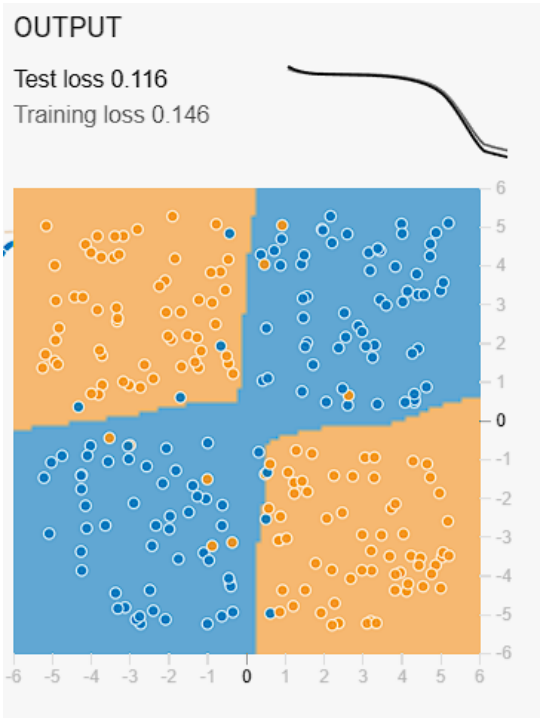




Part c-



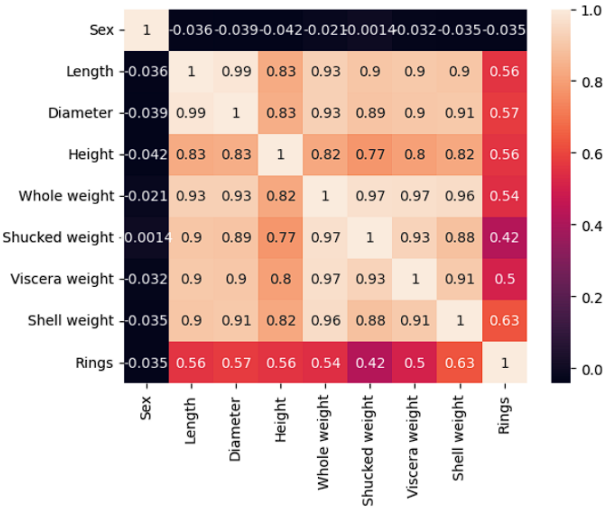
- **Best test loss = 0.170**
- The model surface is not so smooth
- We get slightly better model when we add Sin(X1) and Sin(X2), the model output also gets smoother



Question 2:

Part A-

- Downloaded dataset using **wget** command called using **os.system**
- Loaded the **abalone.data** file into df using pd.read_csv
- Added column names **column_names = ["Sex", "Length", "Diameter", "Height", "Whole weight", "Shucked weight", "Viscera weight", "Shell weight", "Rings"]**
- Checked for not filled rows using **df.isnull().sum()**
- Used **df.describe()** to get insights about the dataset
- Converted **Gender ['I','F','M']** to `F -> 0 ,I -> 1 ,M -> 2`
- Plotted bar plots using **seaborn.barplot**
- Plotted **heatmap of covariance matrix**
- Applied **StandardScaler()** to normalise data
- Putting class labels which has less than 3 counts in **train data**
- Printed the distribution of class labels in **train and test**
- Applied **one-hot-encoding** as we are going to use **categorical_crossentropy**

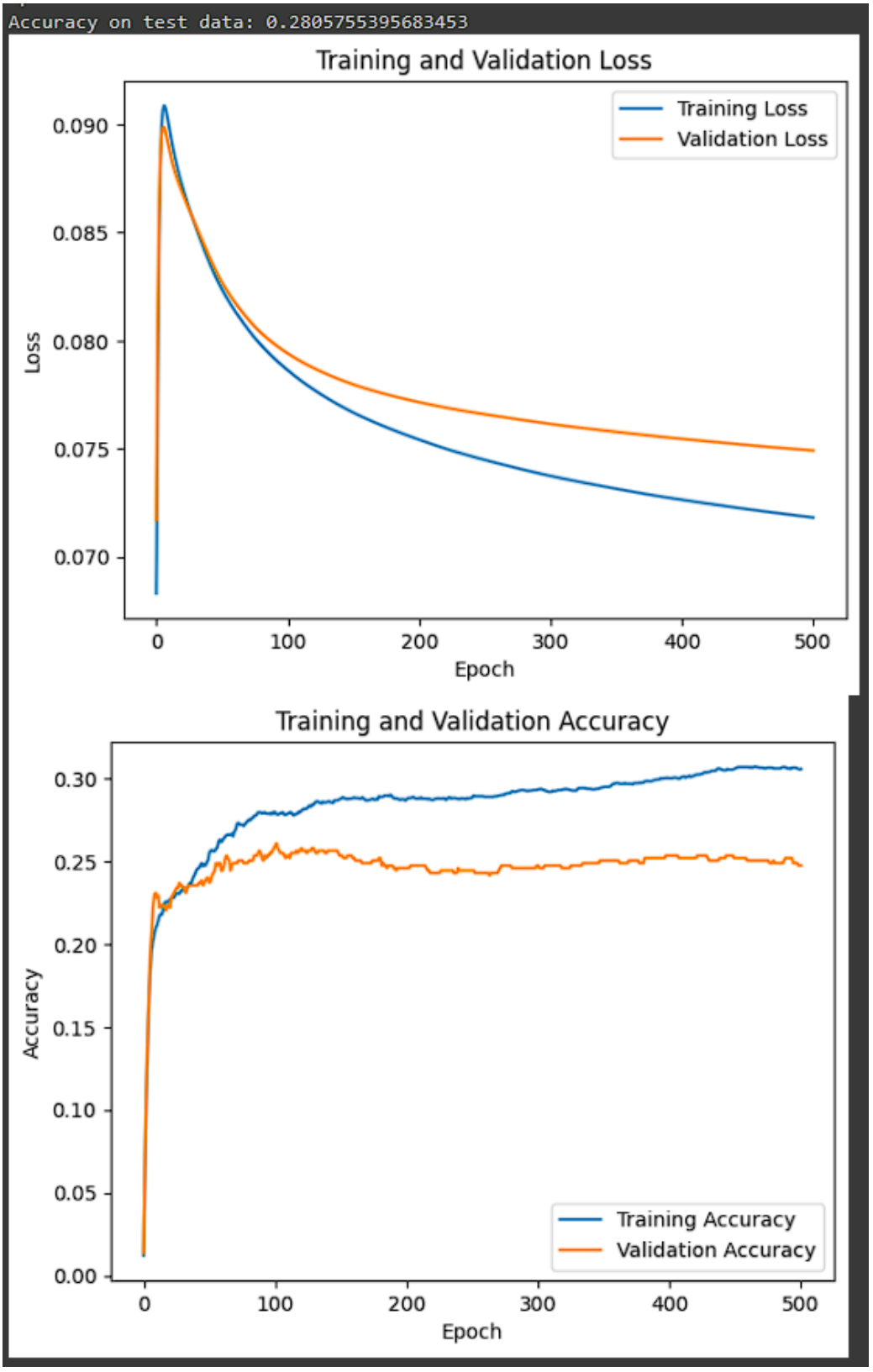


Part B-

- Implemented **scratch built ANN**
- *layers_init:* initializes weights, bias, and activation function parameters for each layer.
- *activation:* applies activation function on input and returns the result.
- *loss:* calculates the loss for the given loss function and predictions.
- *loss_fxn_da:* calculates the derivative of the loss function w.r.t the predictions.
- *activation_derivative:* calculates the derivative of the activation function for a given input.
- *forward_propagate:* propagates the input through the layers and returns the output.
- *backward_propagate:* backpropagates the errors from the output layer to the input layer and updates the weights and biases of each layer.
- *update_parameters:* Updates the weights and biases of a layer based on the calculated gradients and learning rate.
- *single_pass_loss:* Calculates the loss for a single pass through the neural network for a given set of training data.
- *train:* Trains the neural network for a given number of epochs using a specified loss function and parameters such as learning rate and decay rate. It also calculates and stores the training and validation loss and accuracy for each epoch.
- Some small **helper functions** are also implemented such as **accuracy, save_weights, load_weights**

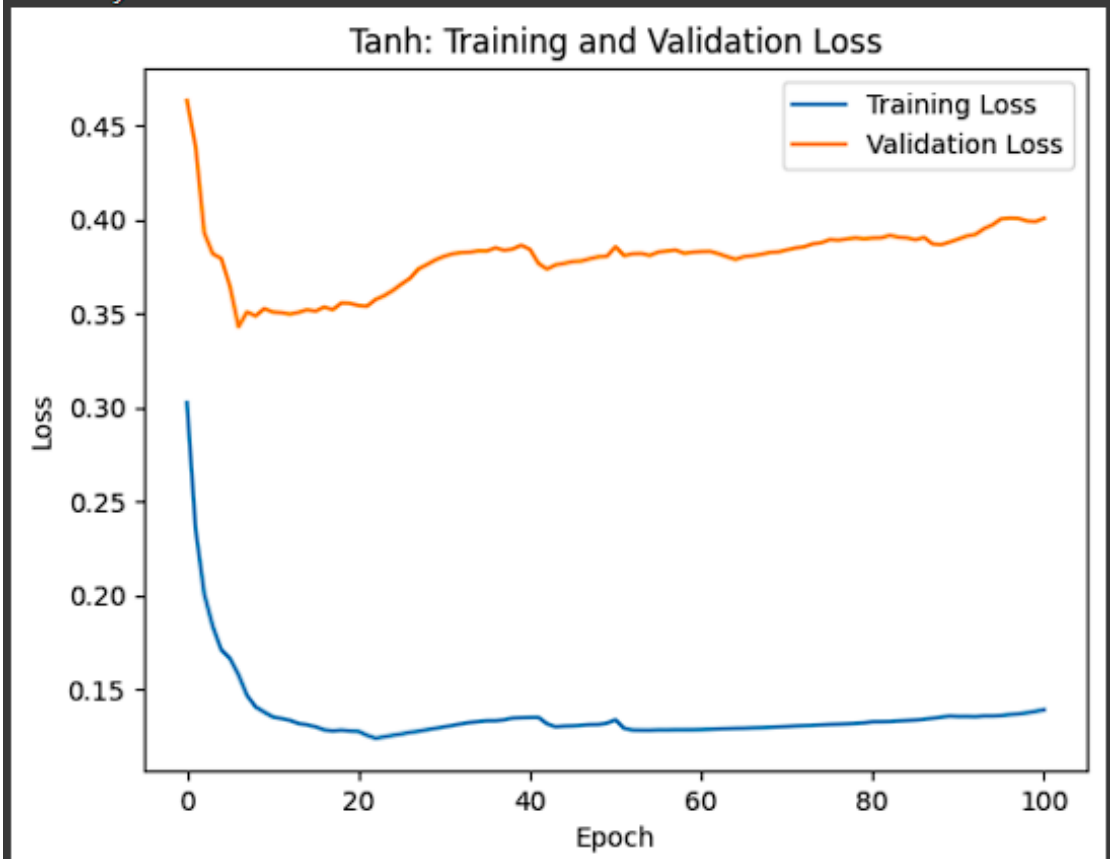
Part 3-

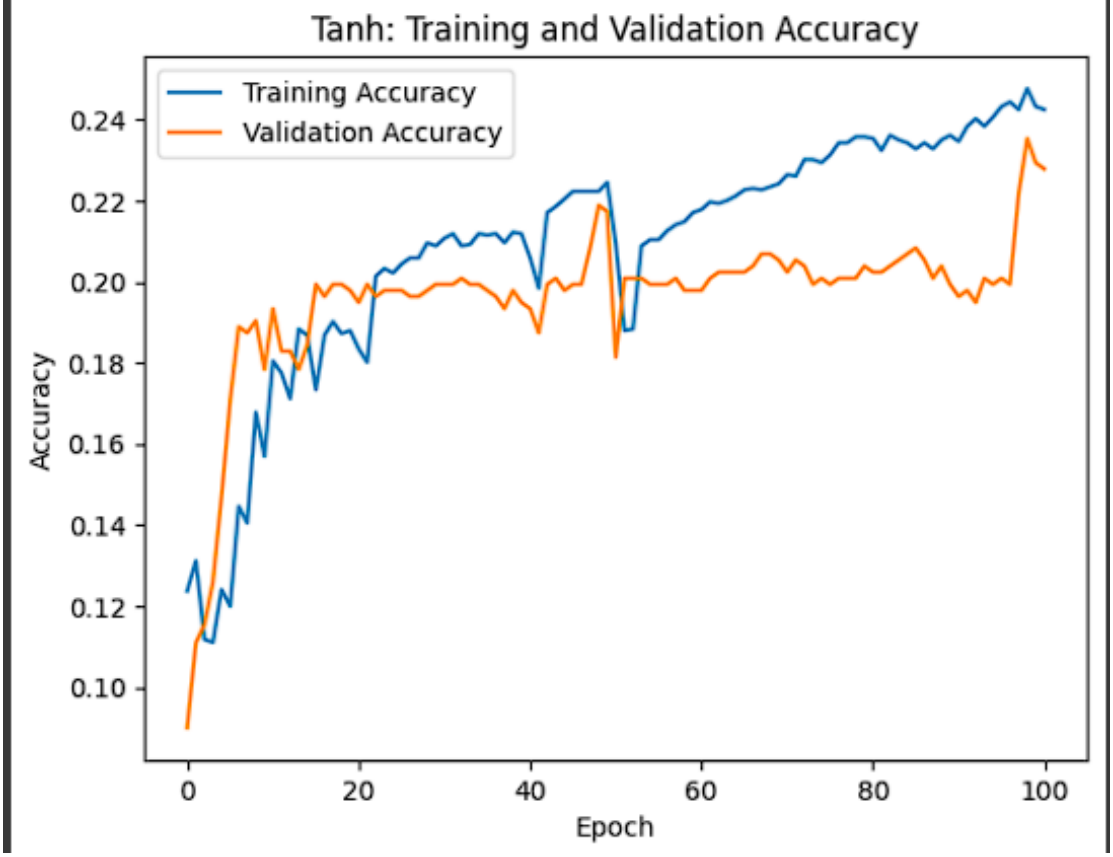
- **Sigmoid**



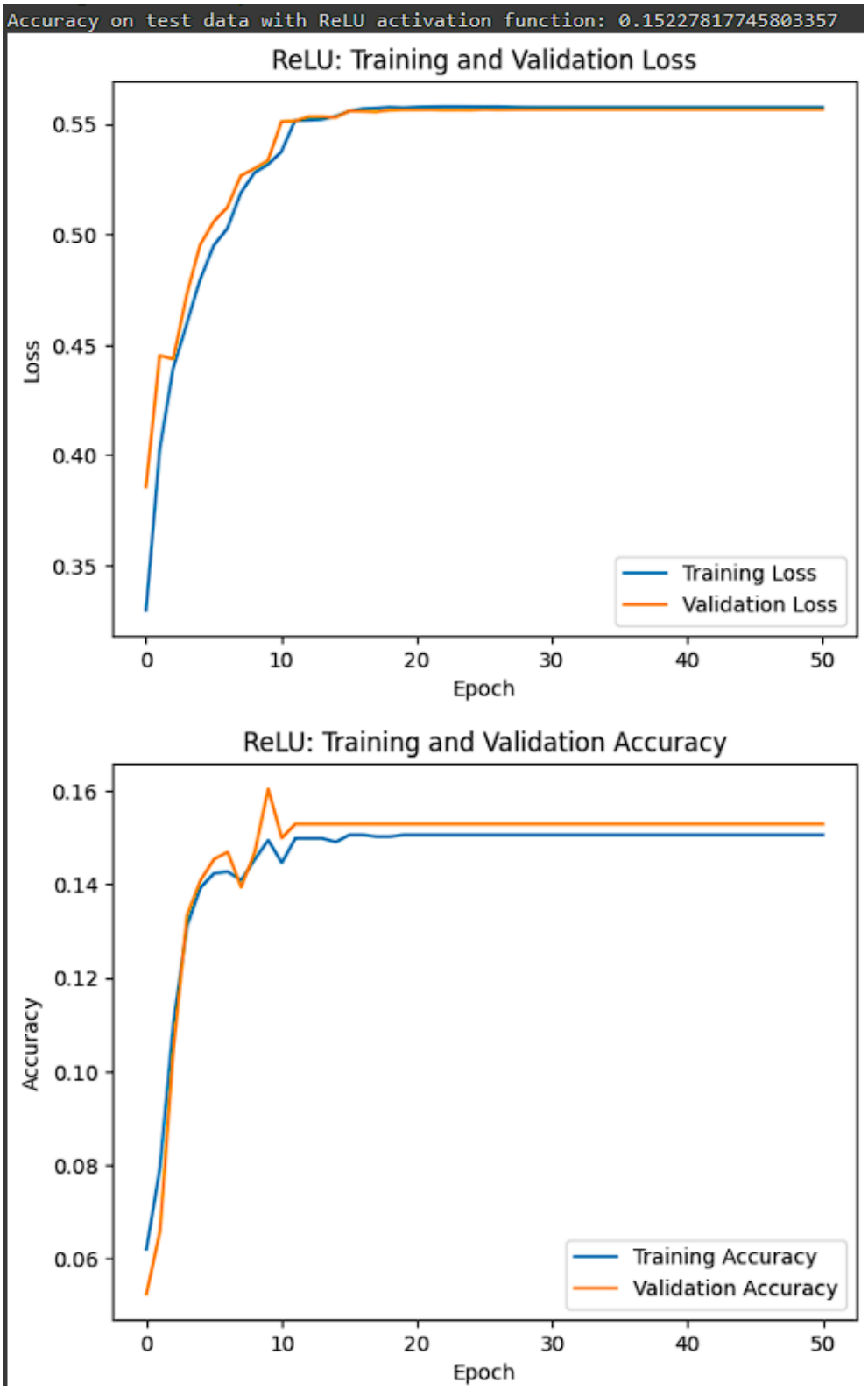
- **Tanh**

```
Accuracy on test data with Tanh activation function: 0.23501199040767387
```





- **Leaky_relu**



- The accuracy of the three increases in long run, but it is smoothest **leaky_relu > Sigmoid > Tanh**
- As the dataset was too complex or overlapping to fit, it doesn't have good graphs due to that also it affects the accuracy also.

Part 4-

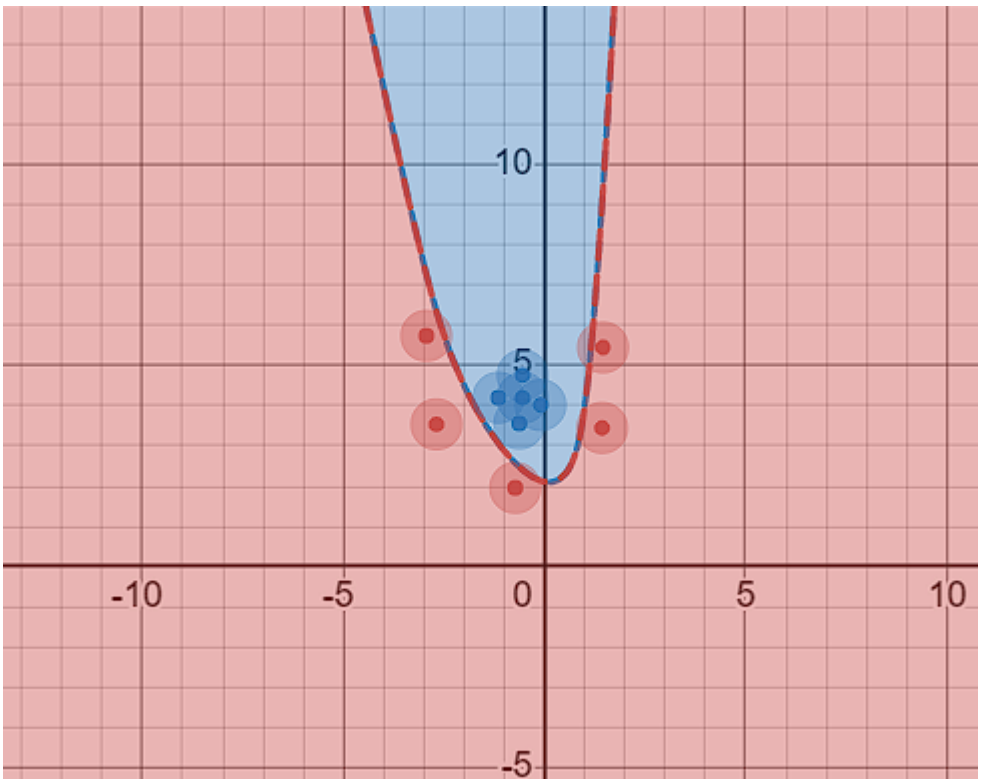
- `Random - Accuracy on test data: 0.26258992805755393`
- `Zero - Accuracy on test data: 0.16546762589928057`
- `Constant(0.25) - Accuracy on test data: 0.16546762589928057`

Part 5-

- `[512,128,64,num_labels] - Accuracy on test data: 0.2541966426858513`
- `layer_dim=[128,64,32,num_labels] - Accuracy on test data: 0.18944844124700239`
- `layer_dim=[32,16,8,num_labels]) - Accuracy on test data: 0.15587529976019185`

Question 3:

- In the start if the learning rate is less, it takes too much time to go towards good decision boundary, so we need a high learning rate at the start, but at the time of converging, if learning rate is high, the model kinda diverges, so we need to decrease the learning rate when the model starts forming good decision boundary(i.e. Starts converging), it starts going unstable at the verge of converging if learning rate is high


10
5
-10
-5
0
5
10
-5