PRML LAB REPORT 9

MANISH(B21CS044)

QUESTION 1

A. Download the MNIST dataset using torch-vision. Split into train, test and validation dataset. Apply Augmentions to images:
   a. Training dataset: RandomRotation(5 degrees), RandomCrop(size=28, padding = 2), ToTensor and Normalize.
   b. Testing dataset and validation dataset: ToTensor and Normalize

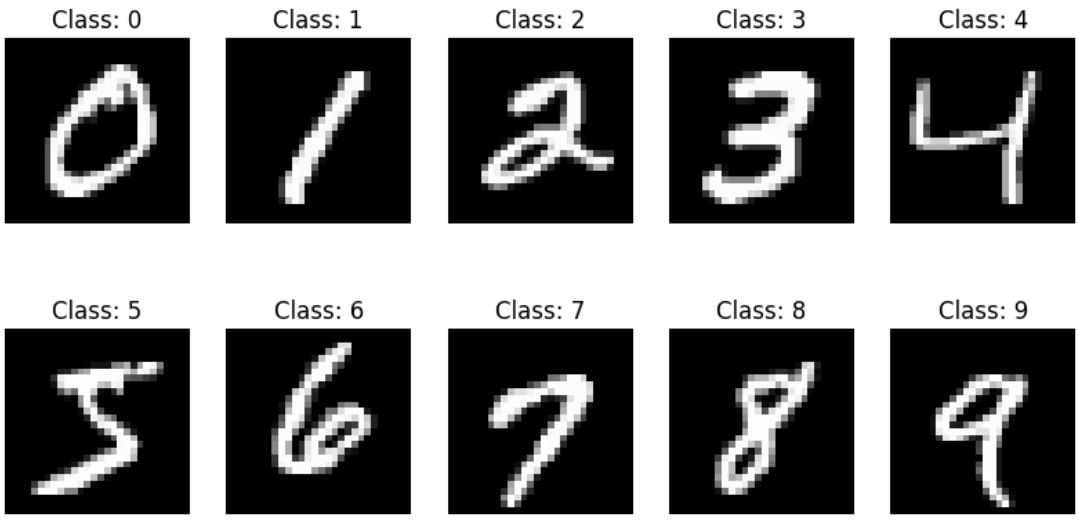
Downloaded the data using pytorch and train-test split of data is performed.

For a part used random rotation as 5 degrees and random crop as size = 28.

For part b performed Tensor and Normalization of the data.

B. Plot a Few Images from each class. Create a data loader for the training dataset as well as the testing dataset.

Created the data loaders for training and testing data using torch.utils.
Here is the images from each class:



C. Write a 3-Layer MLP using PyTorch all using Linear layers. Print the number of trainable parameters of the model.

Implemented the MLP 3-layer using Pytorch and all using linear layers.
Here are the printed parameters of the model:

```
<bound method Module.parameters of MLP(
  (fc1): Linear(in_features=784, out_features=196, bias=True)
  (fc2): Linear(in_features=196, out_features=98, bias=True)
  (fc3): Linear(in_features=98, out_features=10, bias=True)
  (relu): ReLU()
)>
```

D. Train the model for 5 epochs using Adam as the optimizer and CrossEntropyLoss as the Loss Function. Make sure to evaluate the model on the validation set after each epoch and save the best model as well as log the accuracy and loss of the model on training

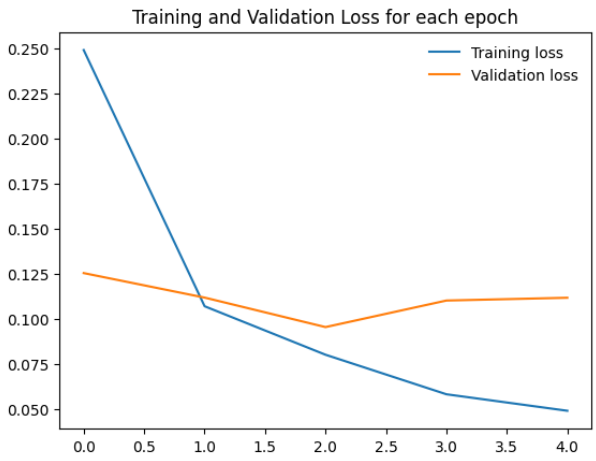
Trained the model for 5 epochs using Adam as optimizer and Cross Entropy Loss.
Here are the results after each epoch:

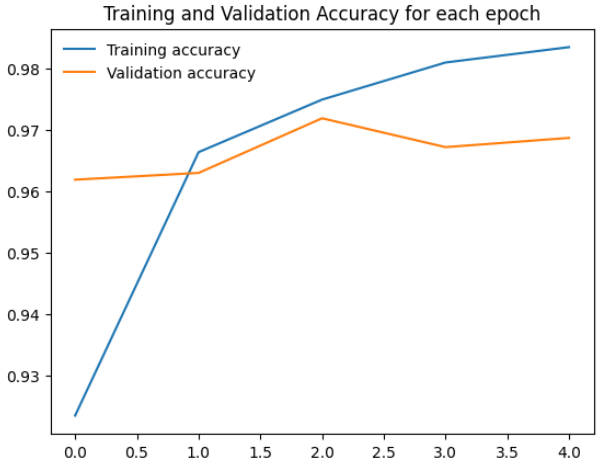
```
Epoch: 1        Training Loss: 0.249167         Training Accuracy: 0.923480     Validation Loss: 0.125431       Validation Accuracy: 0.961900
Epoch: 2        Training Loss: 0.107037         Training Accuracy: 0.966380     Validation Loss: 0.111835       Validation Accuracy: 0.963000
Epoch: 3        Training Loss: 0.080138         Training Accuracy: 0.974940     Validation Loss: 0.095475       Validation Accuracy: 0.971900
Epoch: 4        Training Loss: 0.058205         Training Accuracy: 0.980980     Validation Loss: 0.110204       Validation Accuracy: 0.967200
Epoch: 5        Training Loss: 0.049057         Training Accuracy: 0.983480     Validation Loss: 0.111749       Validation Accuracy: 0.968700
```

and validation data at the end of each epoch.

E. Visualize correct and Incorrect predictions along with Loss-Epoch and Accuracy-Epoch graphs for both training and validation.

Here are the plots for the same as asked in Question:

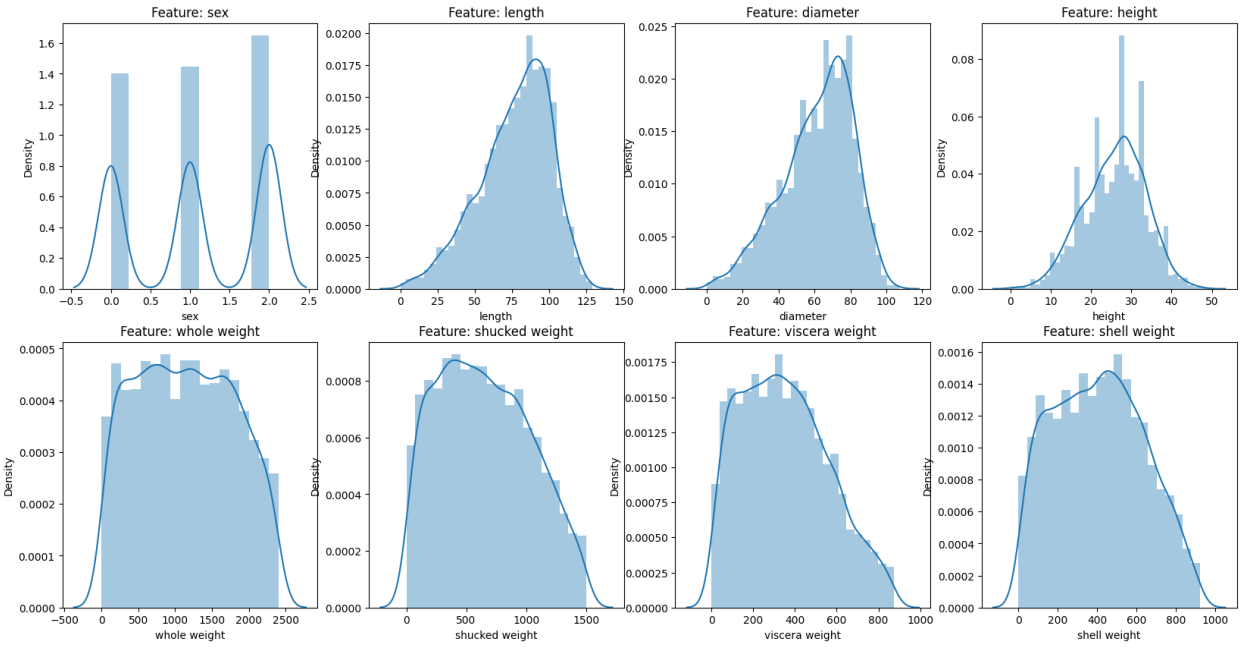


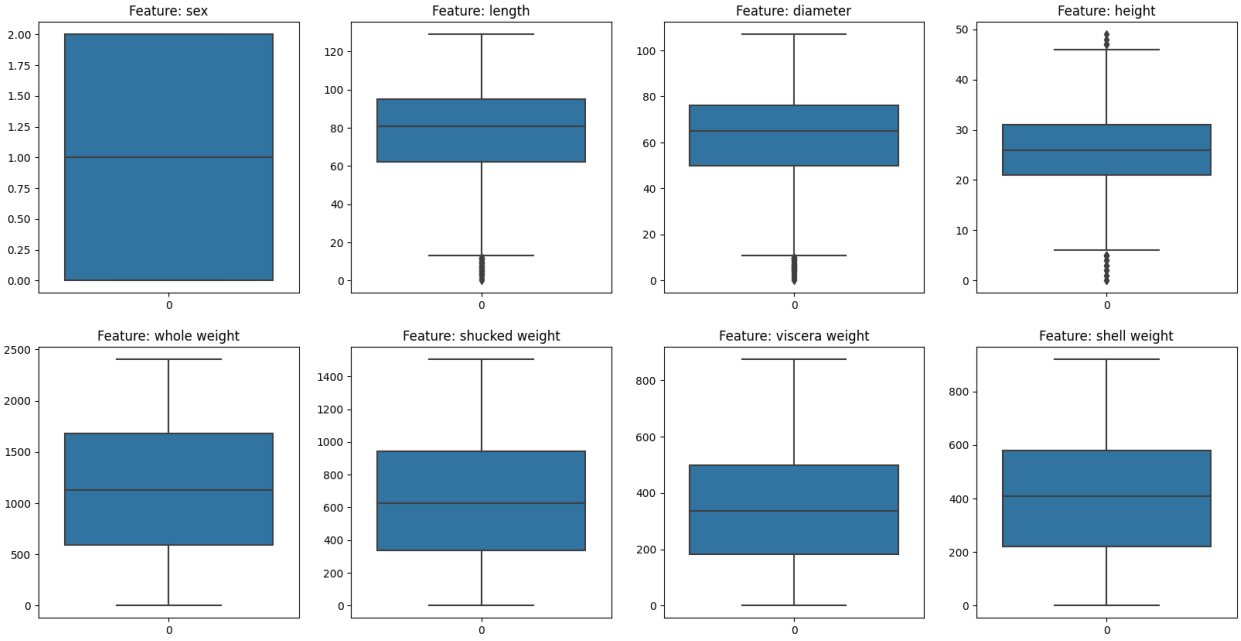


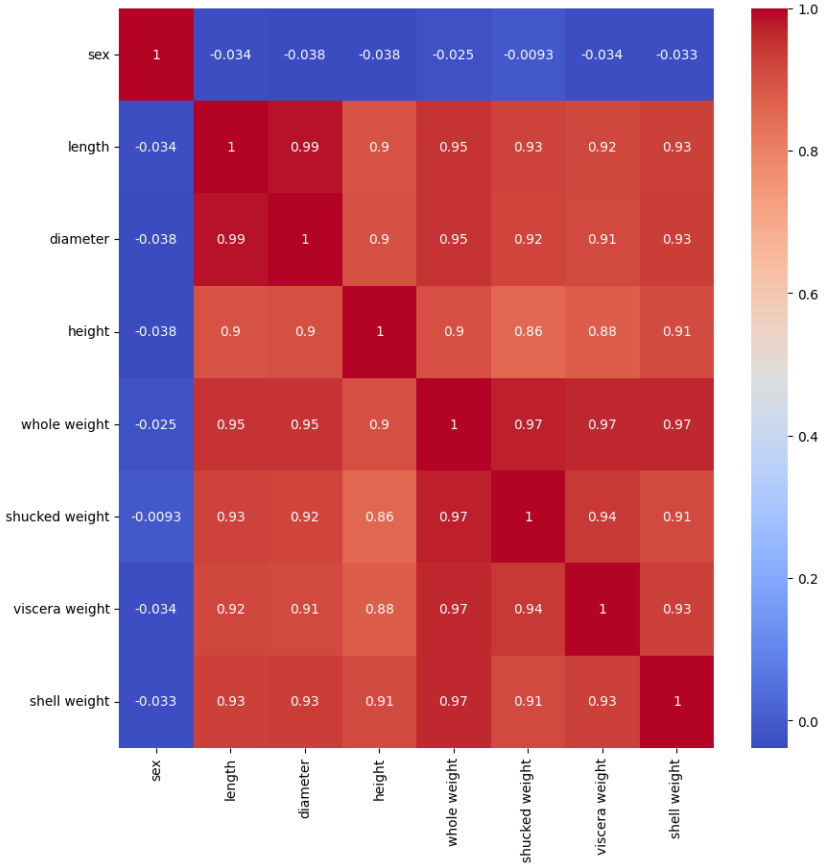
QUESTION 2

A. For this classification, you need to use a multi-layer perceptron.
   a. Preprocess & visualize the data. Create train, val, and test splits but take into consideration the class distribution (Hint: Look up stratified splits). ~ [5]

Here we preprocessed the data and Here are plots for the same:




Feature: sex
Feature: length
Feature: diameter
Feature: height
Feature: whole weight
Feature: shucked weight
Feature: viscera weight
Feature: shell weight


sex 1 -0.034 -0.038 -0.038 -0.025 -0.0093 -0.034 -0.033
length -0.034 1 0.99 0.9 0.95 0.93 0.92 0.93
diameter -0.038 0.99 1 0.9 0.95 0.92 0.91 0.93
height -0.038 0.9 0.9 1 0.9 0.86 0.88 0.91
whole weight -0.025 0.95 0.95 0.9 1 0.97 0.97 0.97
shucked weight -0.0093 0.93 0.92 0.86 0.97 1 0.94 0.91
viscera weight -0.034 0.92 0.91 0.88 0.97 0.94 1 0.93
shell weight -0.033 0.93 0.93 0.91 0.97 0.91 0.93 1
sex length diameter height whole weight shucked weight viscera weight shell weight
1.0 0.8 0.6 0.4 0.2 0.0

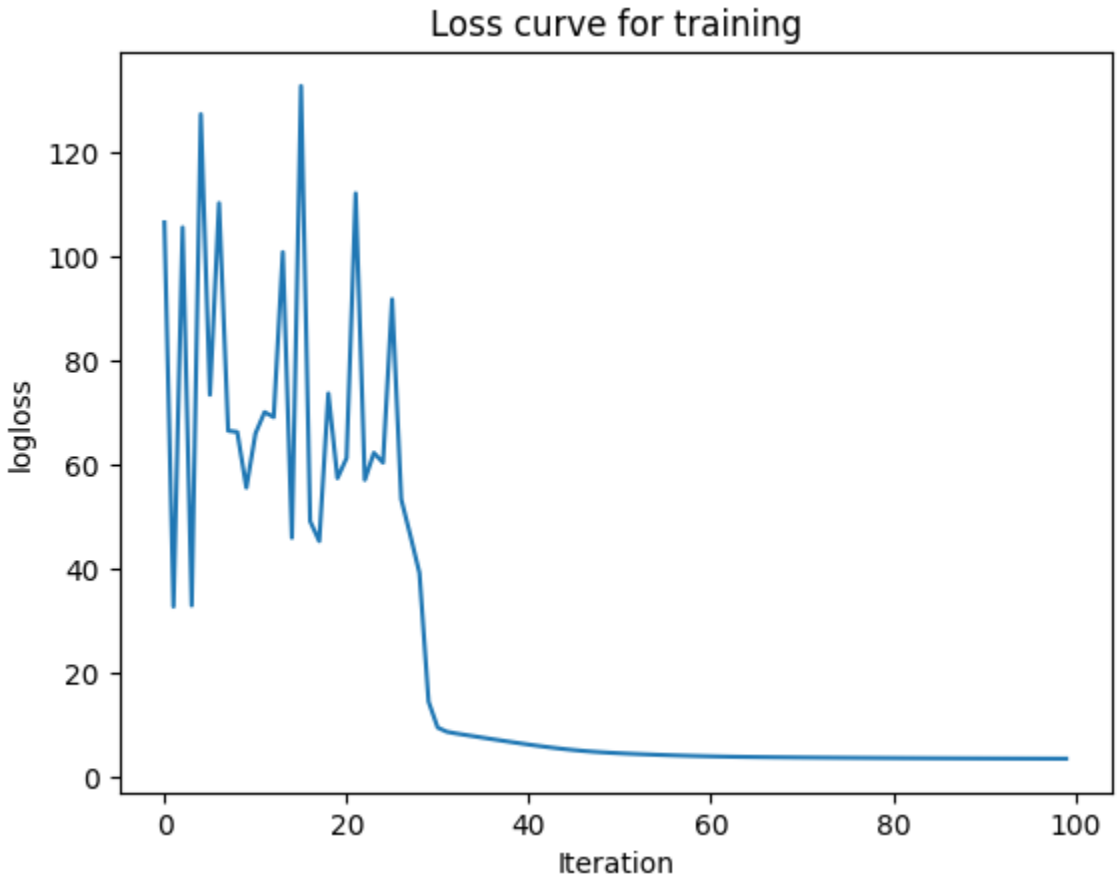
B. b. Implement a multi-layer perceptron from scratch. This would include the following ~[40]
    a. Write activation functions.
    b. Forward propagate the input.
    c. Back propagate the error.
    d. Train the network using stochastic gradient descent.
    e. Predict the output for a given test sample and compute the accuracy.

In this part I have implemented the MLP from scratch for different activation functions such as Tanh, Sigmoid, Relu. Used Stochastic Gradient descent and also implemented accuracy functions as well.

Also implemented Forward propagation and backward propagation.

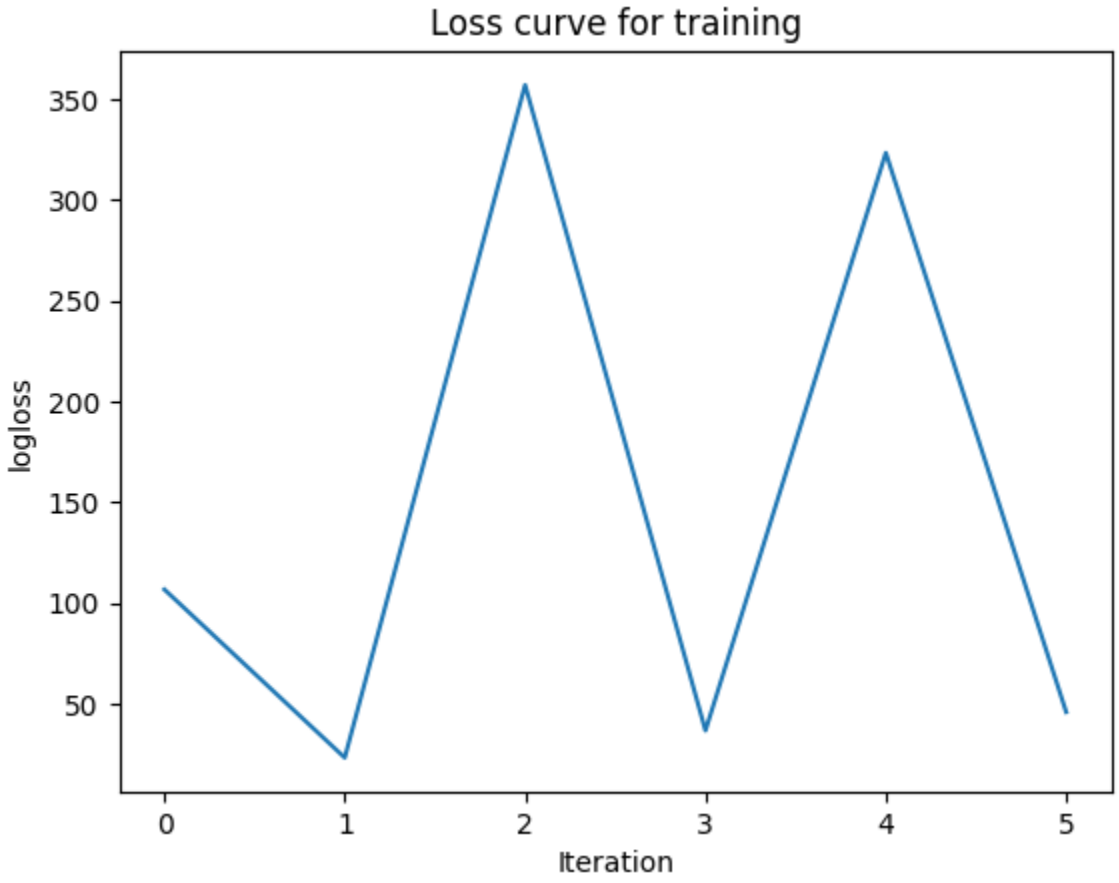
Here are the plots for the different activations functions used

Here are the results for RELU function:



```
Training accuracy:  0.9411764705882353
Test accuracy:  0.834375
```

Here are the results for the Sigmoid functions:



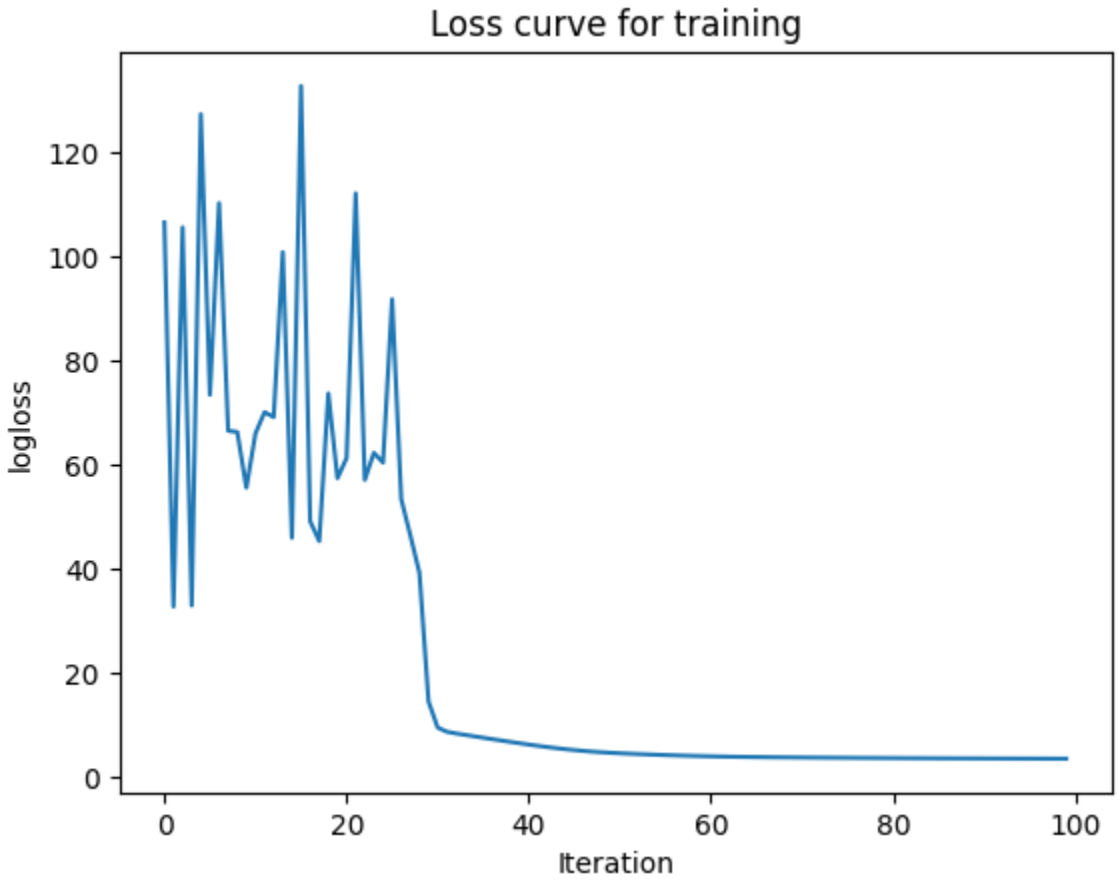

```
Training accuracy:  0.85
Test accuracy:  0.9859375
```

From above we can see that the sigmoid function learns better than the other functions as expected.

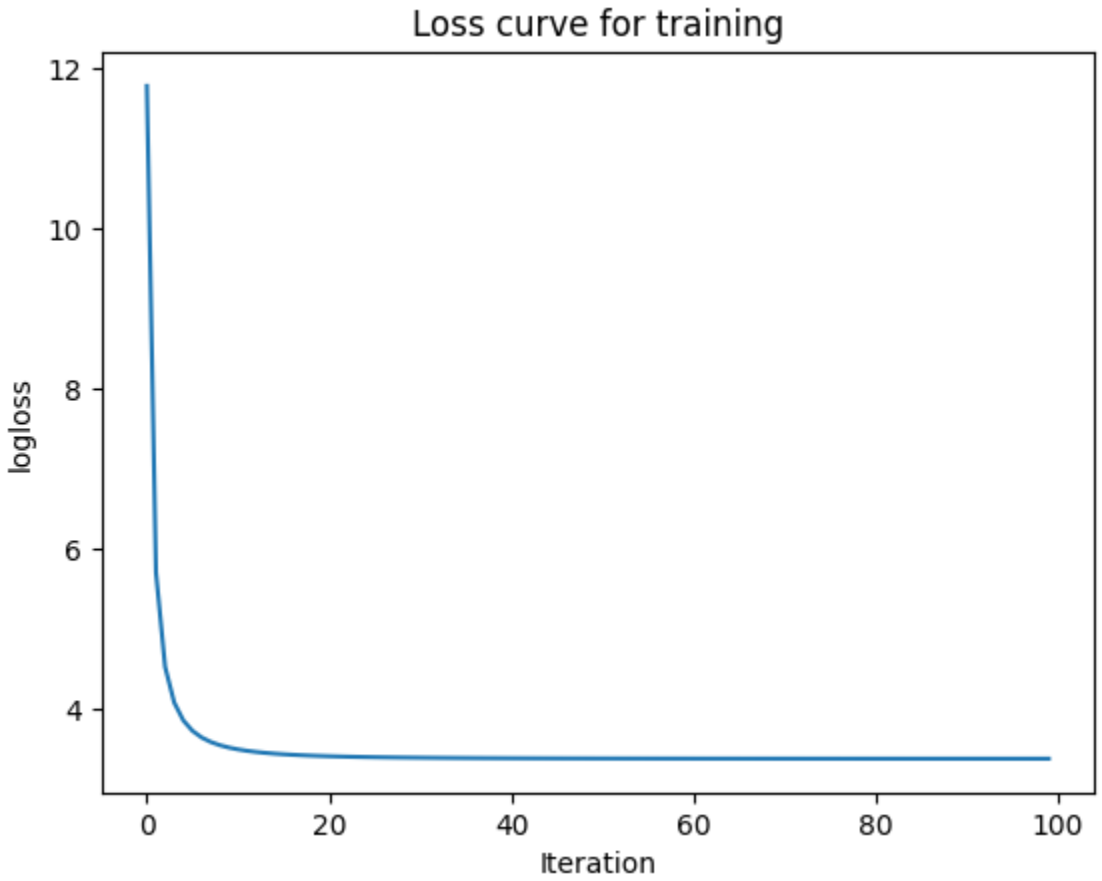
D. Experiment with different weight initialization: Random, Zero & Constant. Create plots

Here are the different plots for the different weights initialization.

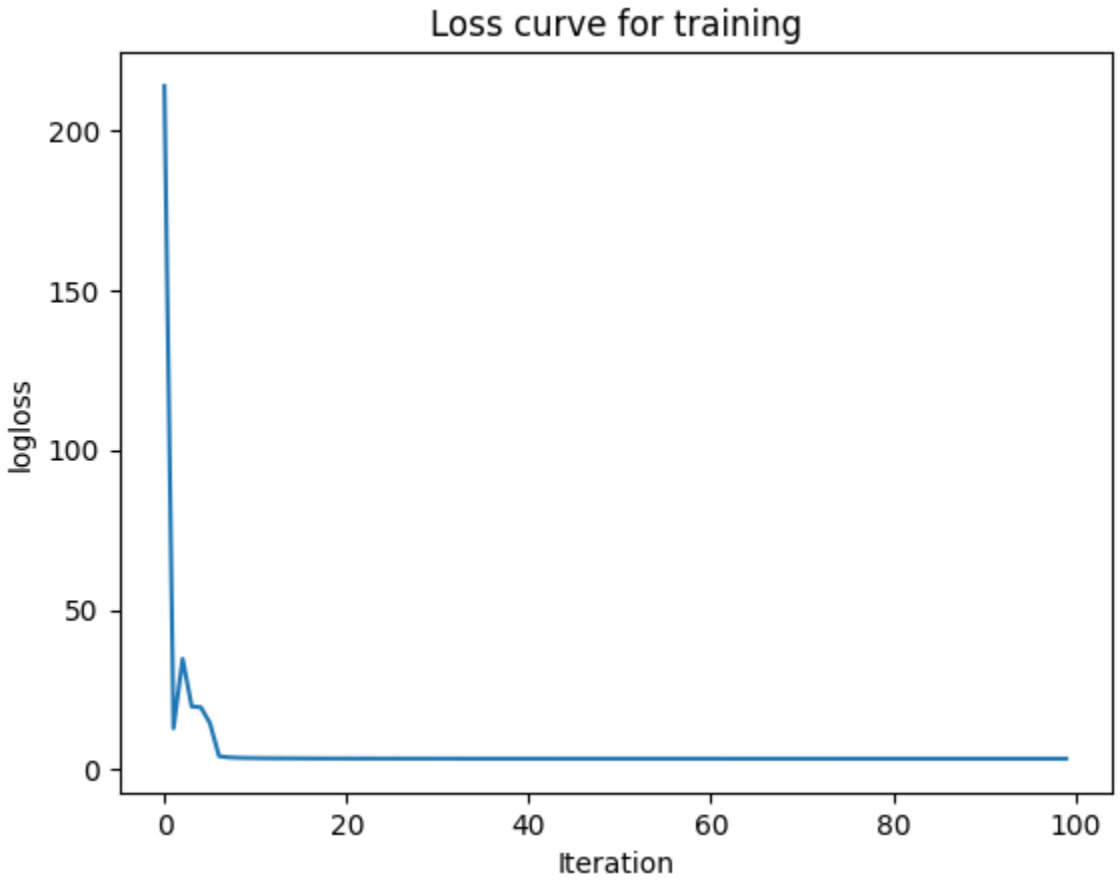
Plot for Random initialization.



Plot for the Zero weights:

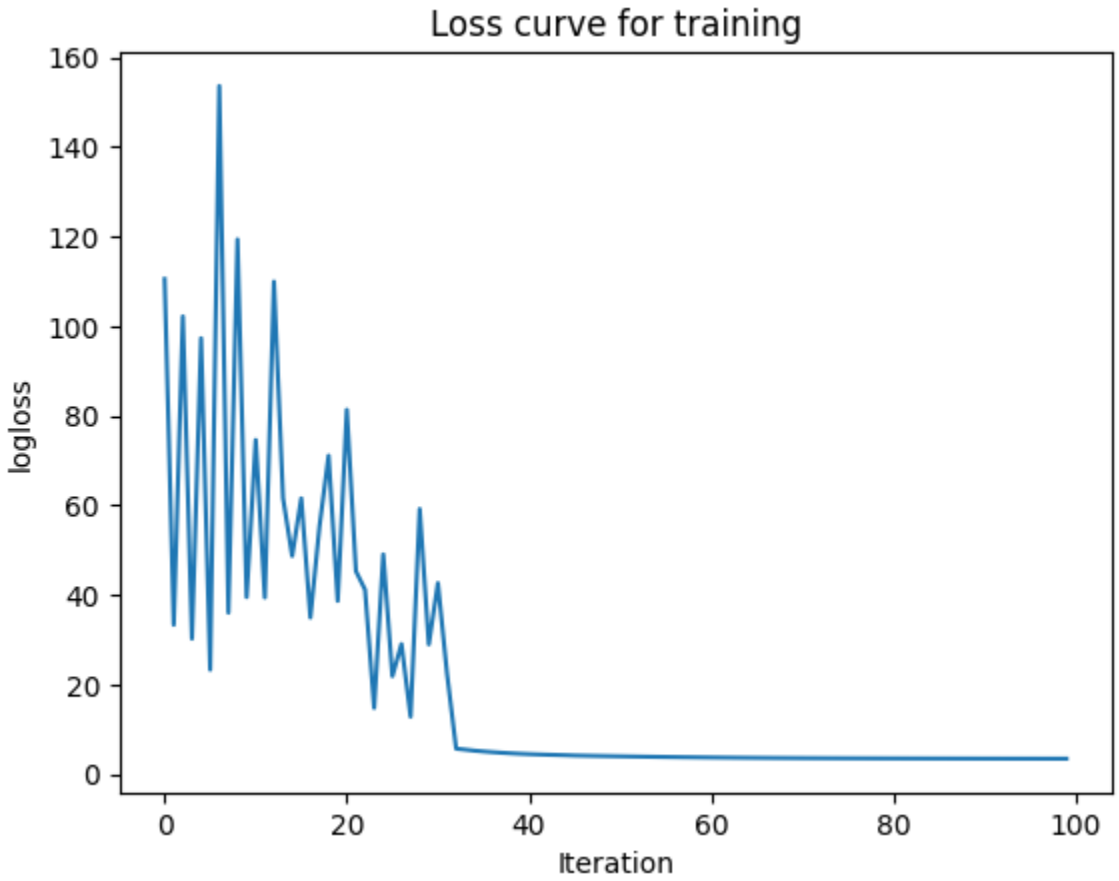


Plot for the one weights:



E. Change the number of hidden nodes and comment upon the training and accuracy. Create plots to support your arguments.Add a provision to save and load weights in the MLP.~[5]

Here is the plot for changing the hidden nodes from 100 to 50:



From the plot we can see that on decreasing the hidden nodes the loss curve is converging slower than the 100 nodes which is expected also.

QUESTION 3

A. [Google PlayGround]

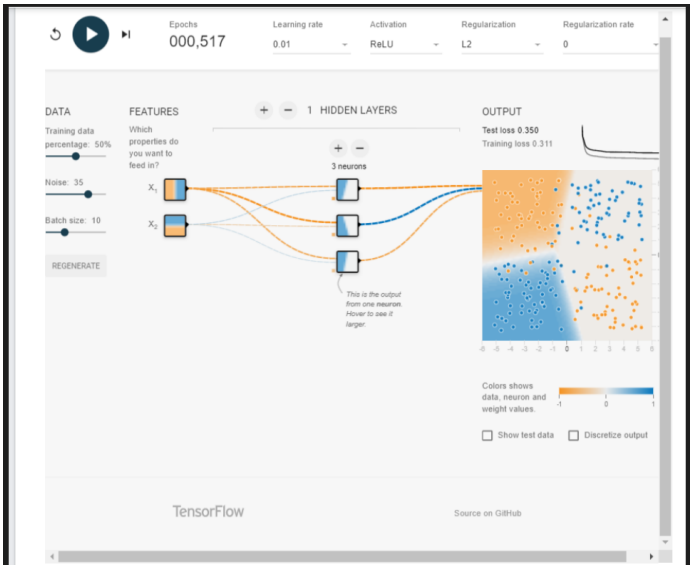
a. Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model?

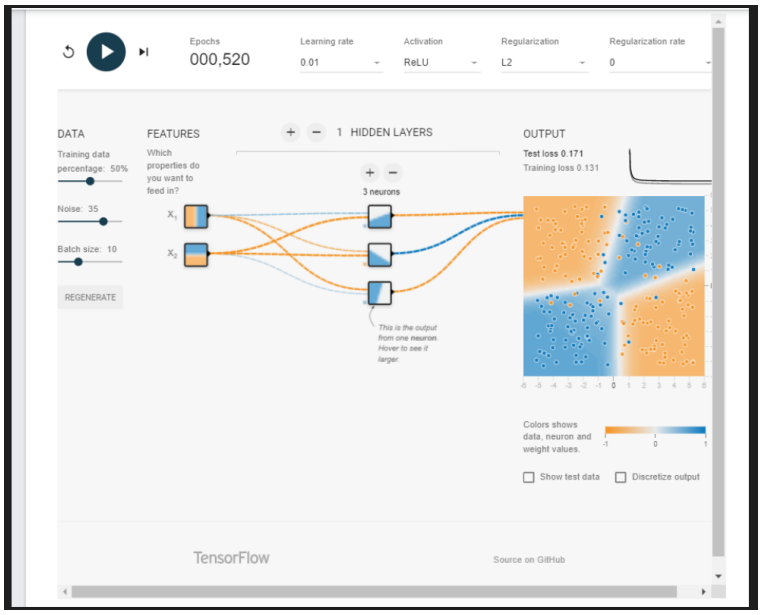
b. Run the model as given four or five times. Before each trial, hit the **Reset the network** button to get a new random initialization. (The **Reset the network** button is the circular reset arrow just to the left of the Play button.) Let each trial run for at least 500 steps to ensure convergence. What shape does each model output converge to? What does this say about the role of initialization in non-convex optimization? Try making the model slightly more complex by adding a layer and a couple of extra nodes. Repeat the trials from Task 1. Does this add any additional stability to the results?

c. Train the best model you can, using just X_1 and X_2. Feel free to add or remove layers and neurons, change learning settings like learning rate, regularization rate, and batch size. What is the best test loss you can get? How smooth is the model output surface?Even with Neural Nets, some amount of feature engineering is often needed to achieve best performance. Try adding in additional cross product features or other transformations like $\sin(X_1)$ and $\sin(X_2)$. Do you get a better model? Is the model output surface any smoother?

The plots for the same are as given below:

```
Part A) Yes, On increasing the model's complexity the fit of model becomes
good. And this also converges faster. But the model may overfit the data.
```

Part B) 




```
Thre role of Initialization is that on reseting the neural network the
weights are initialized randomly. So that the model can learn from the
data. So, it gives different decision boundaries for different
initializations and also the validation and training accuracy is different
for different initializations. And may converge to different local minima
and also can converge faster or slower than other initializations.
```

Part C)

```
Part C) On adding the extra features don't effect the neural network so
much. And also making model extra complex overfits the data. So, adding
extra features doesn't effect the model much. But on adding one or two
complex features may help it to converge faster and also to fit the data
better.
```

B. Neural Nets V2 Desmos Visualized
   a. Try tweaking the learning rate and report your observations regarding the stability of training by observing the decision boundary.

```
As we decrease the learning rate the model converges slower. And also the
```

```
model may not converge at all. And also the model may overfit the data.
So, we need to choose the learning rate carefully. In this case the on
decreasing the learning rate the model converges to a certain point and
then it doesn't converge.
```