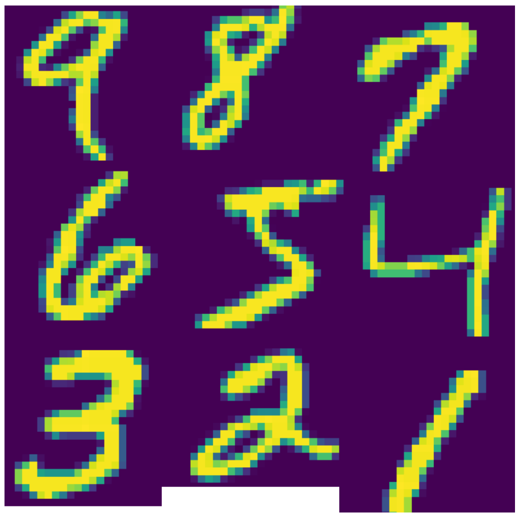
PRML LAB REPORT - 9

Devang Shrivastava
B21CS024

Question-1

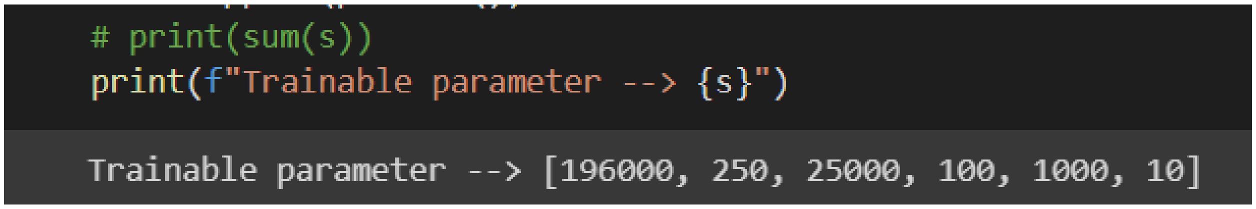
- Downloaded data using datasets.MNIST
- Random split using torch.utils.data.random_split
- Transformed Data according to required parameters, as given in the question.
- Loaded data using DataLoader



Plotted Pictures of each class

First, we created an MLP class:

- We first crearted the input layer then the hidden layer and then the final output layer
- Then we created a forward function for the forward propagation

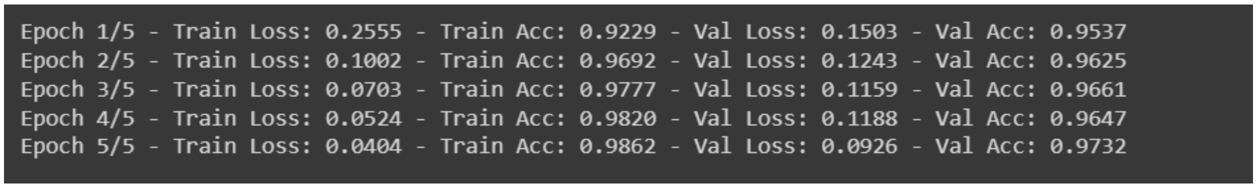


```
    # print(sum(s))
    print(f"Trainable parameter --> {s}")

    Trainable parameter --> [196000, 250, 25000, 100, 1000, 10]
```

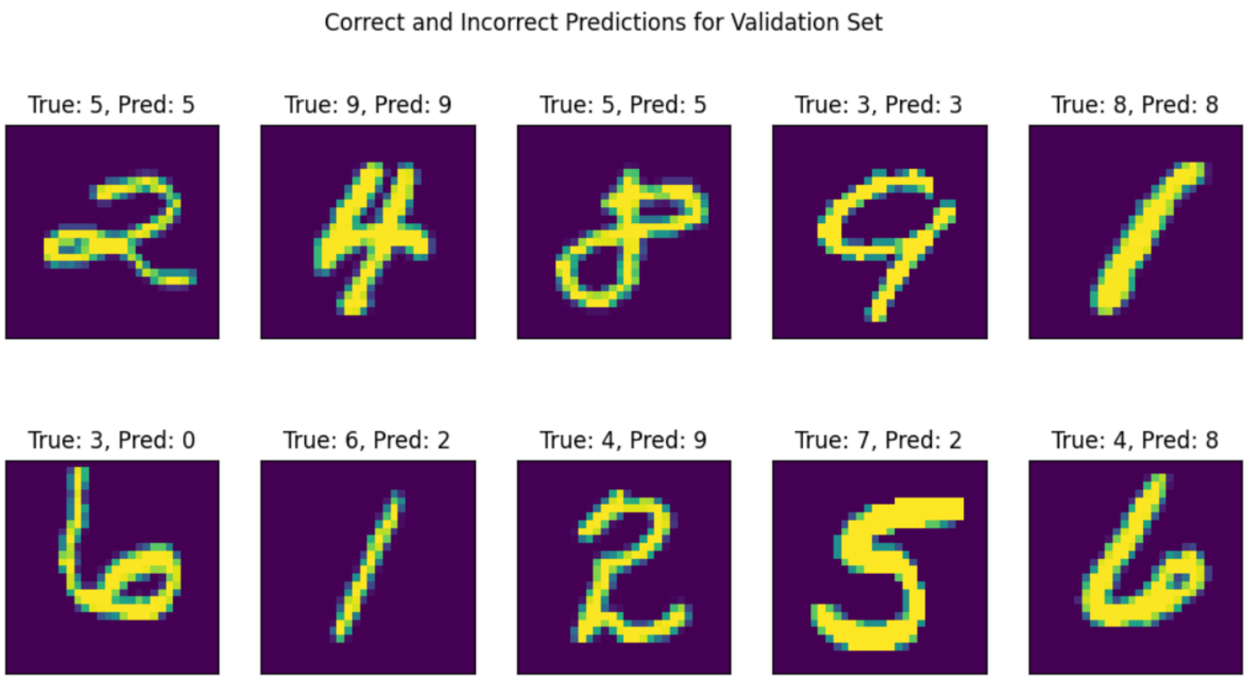
Trainable Parameters

- For 5 epochs, we then trained the model with Adam as the optimizer and CrossEntropyLoss as the Loss Function.

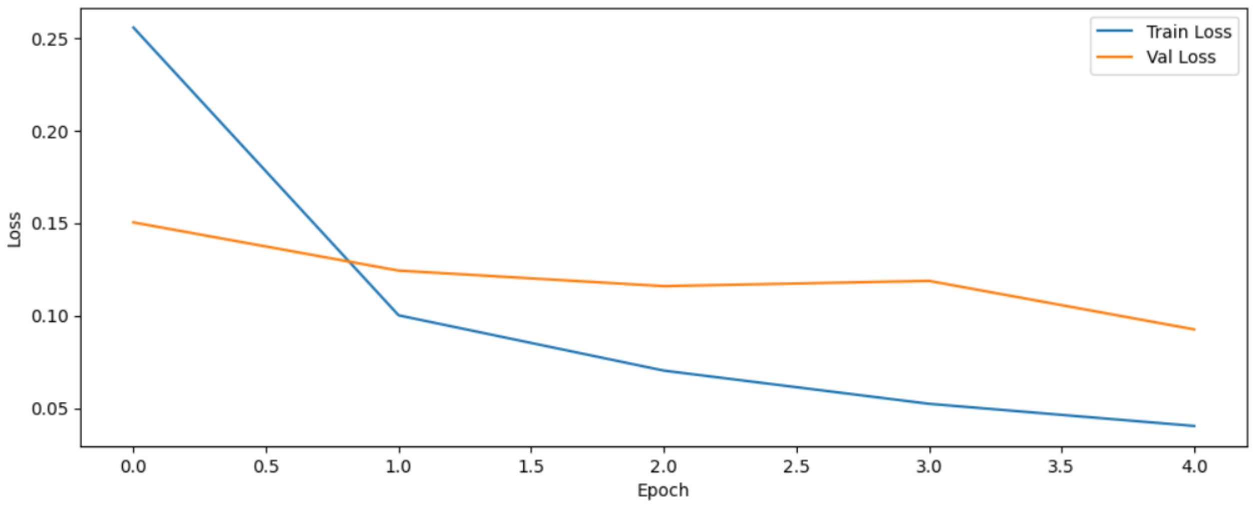


```
Epoch 1/5 - Train Loss: 0.2555 - Train Acc: 0.9229 - Val Loss: 0.1503 - Val Acc: 0.9537
Epoch 2/5 - Train Loss: 0.1002 - Train Acc: 0.9692 - Val Loss: 0.1243 - Val Acc: 0.9625
Epoch 3/5 - Train Loss: 0.0703 - Train Acc: 0.9777 - Val Loss: 0.1159 - Val Acc: 0.9661
Epoch 4/5 - Train Loss: 0.0524 - Train Acc: 0.9820 - Val Loss: 0.1188 - Val Acc: 0.9647
Epoch 5/5 - Train Loss: 0.0404 - Train Acc: 0.9862 - Val Loss: 0.0926 - Val Acc: 0.9732
```

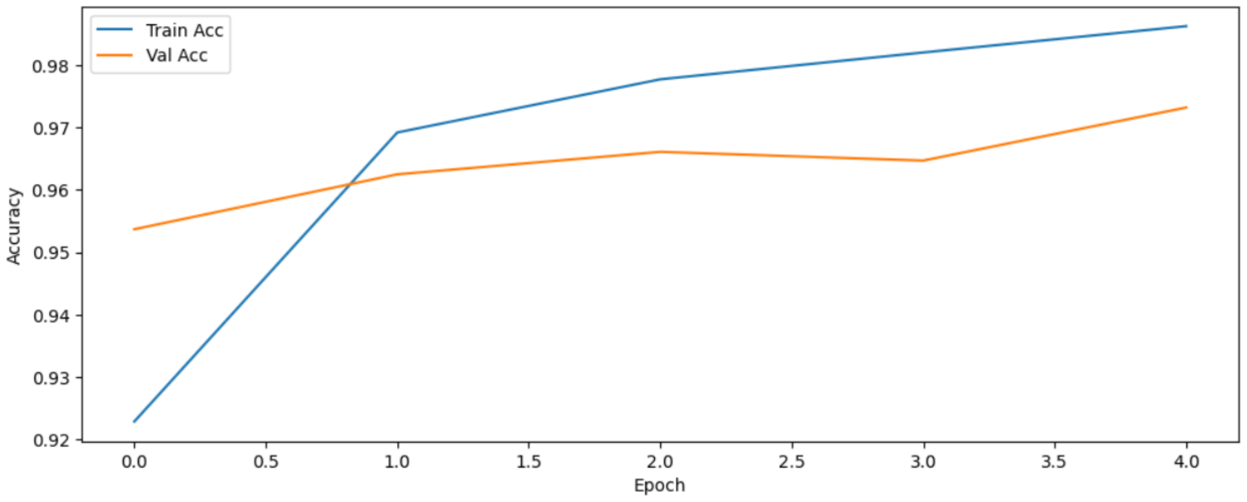
accuracy and loss of the model at each epoch.



Loss and Accuracy for Training and Validation Sets



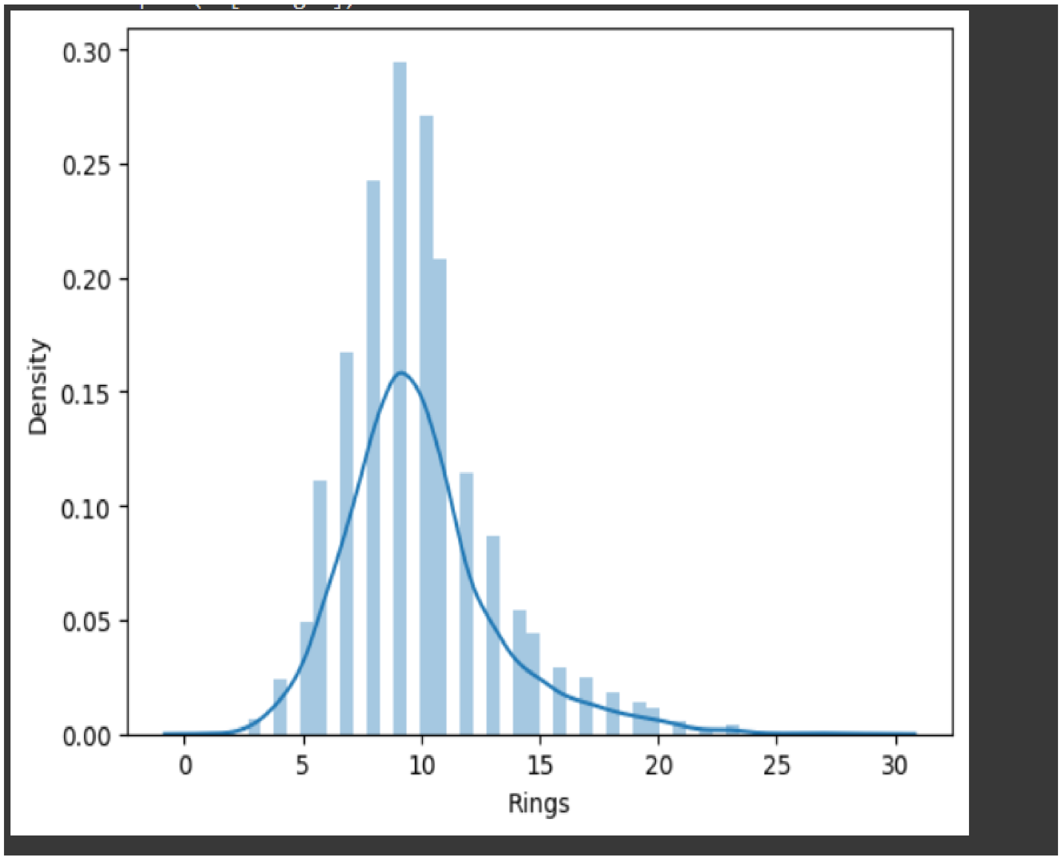
Loss vs Epoch

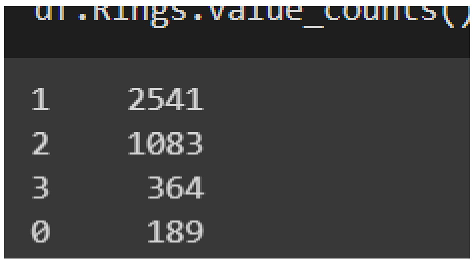


Accuracy vs Epoch

Question-2

- Data was read and the necessary features were extracted.
- The column named “Sex” was encoded using sklearn.LabelEncoder
- Some classes were unevenly distributed



- So I combined classes to form 3 major classes:
    - 0 → ring no. 1-9
    - 1 → ring no. 10-18
    - 2 → ring no. 19-27
    -
- 

```
df.Rings.value_counts()

1    2541
2    1083
3     364
0     189
```

• Forward Propagation:

❖ The data cruises in forward direction of the neural network using this function. The nput layer using randomly generated weights goes to the next layer. For this, we used our knowledge of matrix multiplication.

❖ First of all, the input layer generates the next layer, which is the first hidden layer in my case. The first hidden layer is made up of some neuron which can be pre decided. The matrix of input (which is our input data point or row), which in our case is a
matrix of size 1x8 is multiplied with initialised weights, which is a matrix of size 8xn (n stands for number of neurons in next layer. From this we will get next layer of neurons of shape 1xn. Now this layer produces the output layer of 7 classes using the same concept of matrices.

❖ All the values in forward propagation are passed through an activation function and an activation value is obtained

• After this, a loss function is used. I have used the cross entropy loss function because it minimises the distance between the predicted and actual probability distributions.

• Backwards Propagation:

❖ Back propagation helps us in optimizing the weights using the loss obtained.

❖ The loss calculated in the loss function is passed to the backwards propagation. The losses are minimised using the concept of optimization used in differentia calculus (For this, functions for the derivatives of different activation functions are formed as well). If
the losses obtained are less, weights are updated.

• Activation Function:

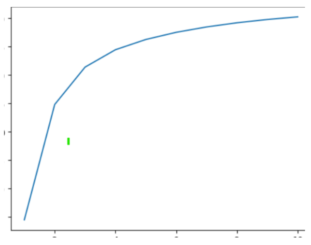
❖ A function that helps the neural network to learn complex patterns in the data. It decides what goes at next layer. A neural network without any activation carries out just a linear neural network process.

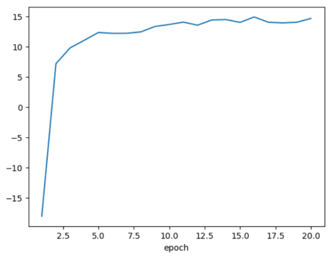
❖ We implemented sigmoid, softmax, ReLu and tanH.

• The network was trained using stochastic gradient approach in which all the rows are iterated one by one.
• For weight initialisation, random matrices of the given scenarios (random, zeros and constants) were made using the respective functions.
• A function train was made for training the model in which all these above mentioned functions are more or less compiled and necessary information was stored.

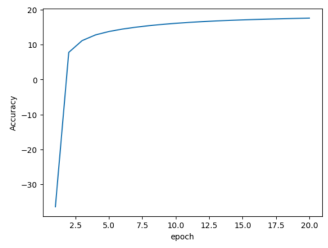
• A function predict was made for getting the prediction array and accuracy score of testing data. It carries out the forward propagation using the updated weights.

Activation Function	Weight Initialisation	Accuracy Score
Sigmoid	Random	50.19%
ReLu	Random	75.47
Tanh	Random	70.21

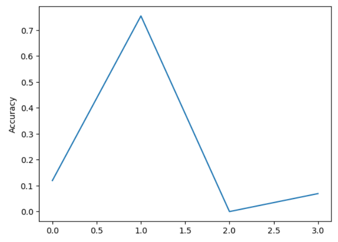

Sigmoid



ReLu

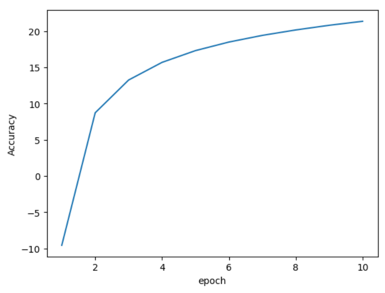


tanh

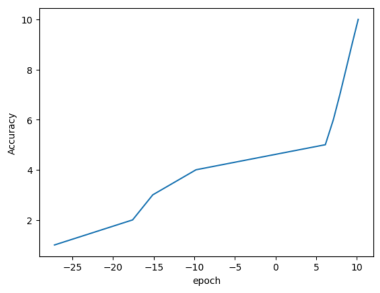


In this figure for test accuracies, 0 represents the accuracy for sigmoid, 1 on X axis represents ReLu and 2 represents tanH activation functions.

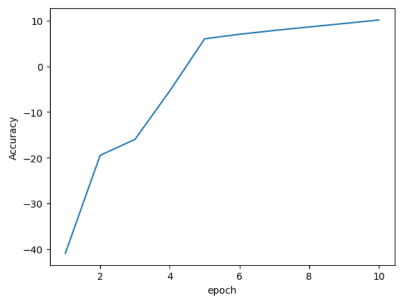
Activation Function	Weight Initialisation	Accuracy Score
Sigmoid	Random	72.27
Sigmoid	Zero	70.69
Sigmoid	Constant	70.63



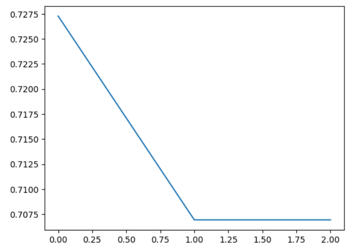
Random



Zero



Constant

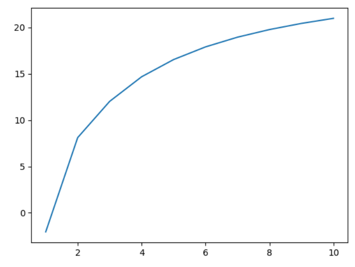


In this figure for test accuracies, 0 represents the accuracy for random, 1 on X axis represents zero and 2 represents constant weight initialisations.

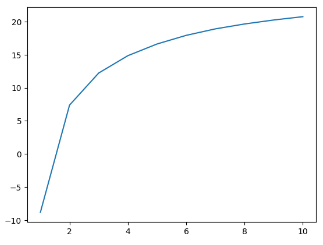
Comments

• Weight initialisation aims at preventing the activation layer outputs from becoming too small and too large. In either case loss gradients will be either too large or too small and neural network will take much more time.
• The derivative of sigmoid function is steep, which means it can get more change in value for a small change in input. It means it is a fast learner. Also, its range is (0,1) which helps in better functioning of the model.
• TanH is symmetric around 0 whereas ReLu is not. Also when we apply derivative on ReLu for a negative value, we we will always get 0. With TanH, it gives a value different than 0. ReLu will tend to quickly set negative gradients as 0. So, ReLu model quickly stops learning. Due to these reasons, ReLu might be able to perform better than TanH as it can handle the problem of vanishing gradients.
• If all the weights are initialised as zeros, the derivative is also same for every w and hence all weights will have the same value in the next layers as well. Also, zero initialisation causes the neuron to memorise the functions.
• The similar thing happens for constant weights as well.
• Random weights, on the contrary gives the model the independence from such overfitting. No symmetry provides better results.

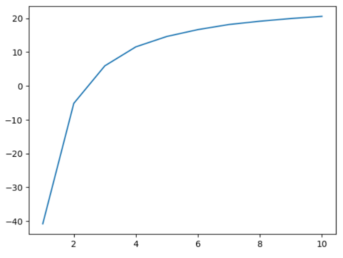
• In the subsequent parts, we have varied the number of neurons in the hidden layer. I have varied them from 25 to 125 with a step of 25.
• Plots for the same are:
X axis represents epochs and Y axis represents accuracies.



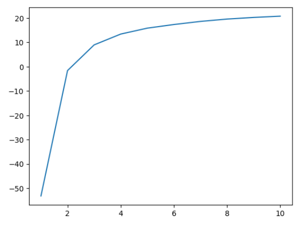
25



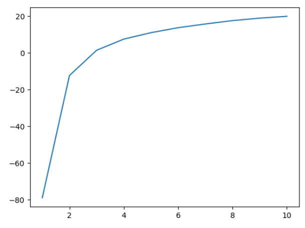
50



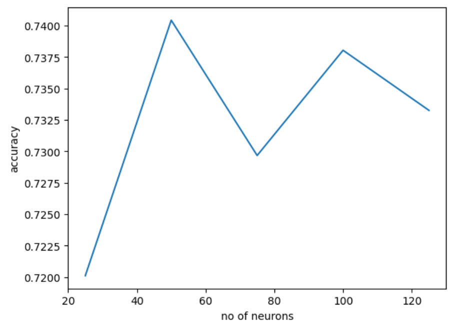
75



100



125



For the final part of the question, we are already returning and storing the values of w1 and w2 after the back propagation.