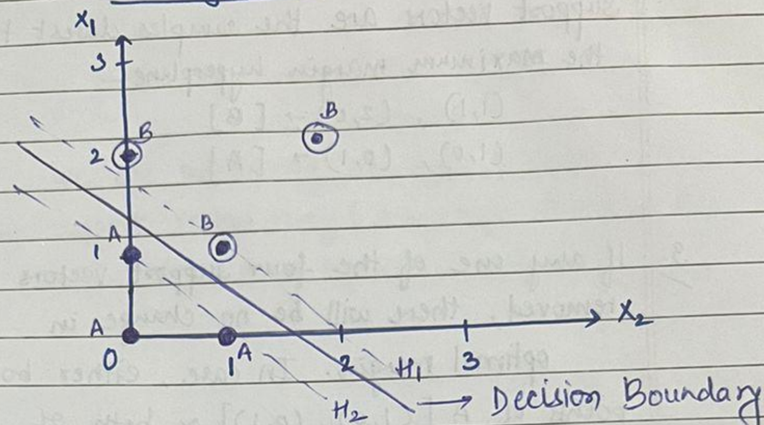
ML ASSIGNMENT 3

Section A:

ML Assignment - 3



Yes, the 2 classes are linearly separable.
The decision boundary cuts the x_1 and x_2 lines at $(1.5, 0)$ and $(0, 1.5)$ points

$\therefore$ slope $= \frac{\text{rise}}{\text{run}} = \frac{+1.5}{-1.5} = -1$

Using this, we get
the equation of the line as $x_1 + x_2 = 1.5 = 3/2$

2. Maximum margin Hyperplane is the distance b/w hyperplane to closest example of either class is maximum. Hence, it should pass the point $(3/2, 0)$ or $(0, 3/2)$ and it must have a slope same as H1 & H2, i.e. -1.

$\therefore$ Maximum margin Hyperplane $\rightarrow x_1 + x_2 = 3/2$

Weight vector : $W = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $b = 3/2$

Support vectors are the samples closest to the maximum margin hyperplane –

(1,1) , (2,0) → [B]

(1,0), (0,1) → [A]

3. If any one of the four support vectors is removed, there will be no change in optimal margin. In case, either both points of A [(1,0), (0,1)] or both of B [(2,0), (1,1)] are removed, the optimal margin increases. In case all 4 support vectors are removed, the optimal margin increases.

4. The presence of support vectors poses a constraint as on which hyperplane is chosen as the optimal one. If support vectors are reduced, the constraints decrease which leads to more no. of possible solutions which can even be better than previous (they will be atleast as good as earlier ones). Let us say there are 2 constraints C_1 and C_2 and $g(z)$ is to be maximised (Max dual problem)

then we can say that :

$$\max_{x \in C_1, x \in C_2} g(x) \leq \max_{x \in C_1} g(x)$$

(if C_2 is removed)

in SVM, we are maximising the dual problem.
Removing support vectors may / may not change the optimal margin depending upon the dataset.

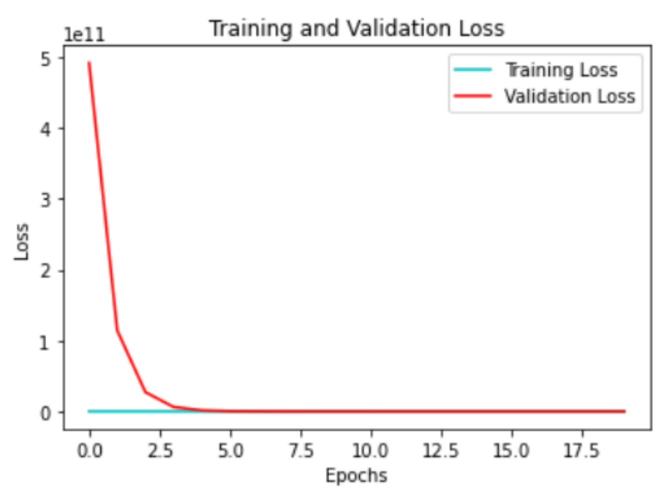
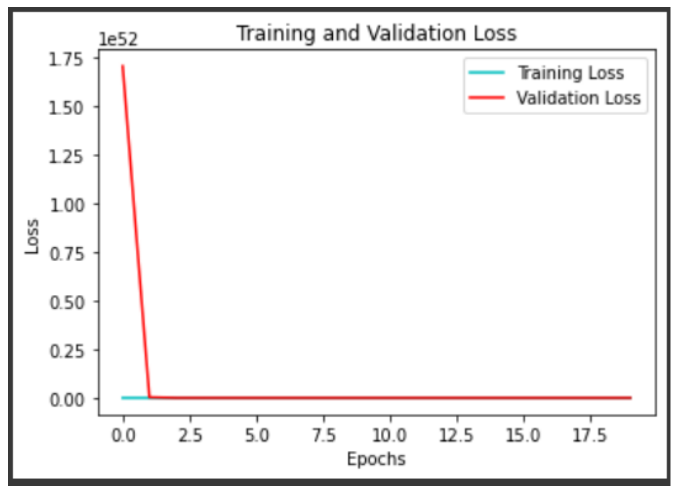
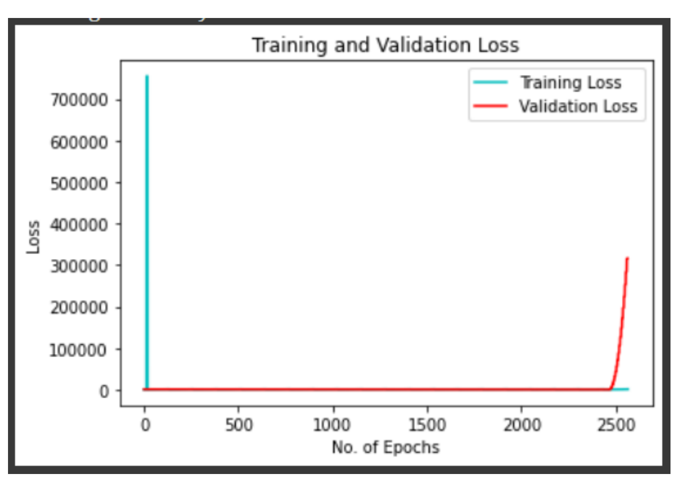
Section B:

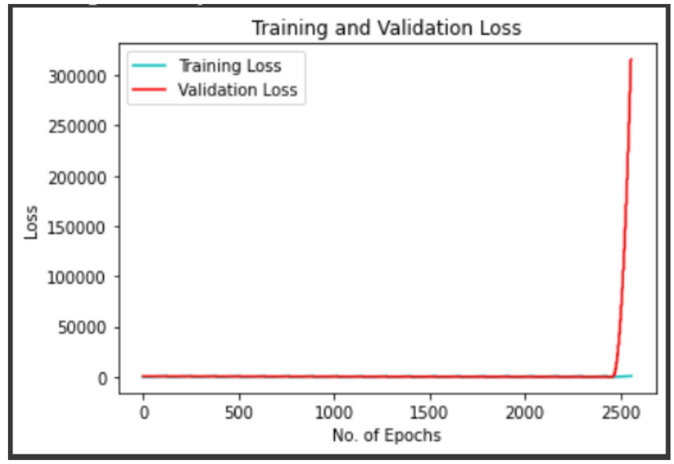
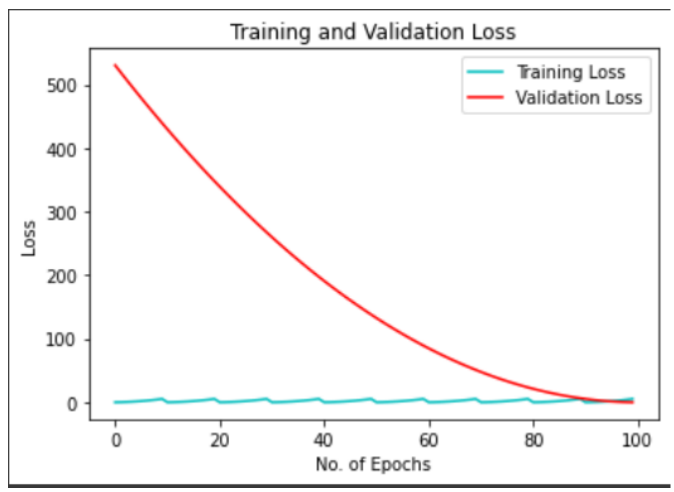
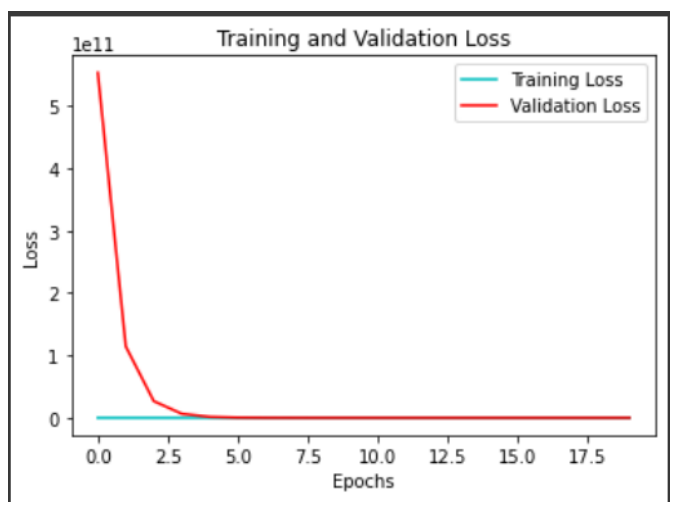
Neural network implemented on MNIST data. The pixels were normalized by dividing by 255 for better and faster training.

Functions implemented in class NeuralNetwork:

- sigmoid()
- relu()
- softmax()
- linear()
- leakyrelu()
- tanh()
- zero_weight()
- normal_weight()
- random_weight()
- init_AF()
- init_weight()
- forward()
- backward()
- fit()
- predict()
- predict_proba()
- score()
- plots()

Activation Function	Training Accuracy	Validation Accuracy	

Sigmoid	43.457	48.87867	
Tanh	11.587	24.017	
Relu	0.11543	0.158	

LeakyRelu	2.9384893	2.511886	
Linear	3.467	5.753	
Softmax	12.468	13.7654	

Section C:

We were given the mnist fashion dataset.

I started by doing the necessary preprocessing.

Normalizing the pixels by dividing by 255 for better training

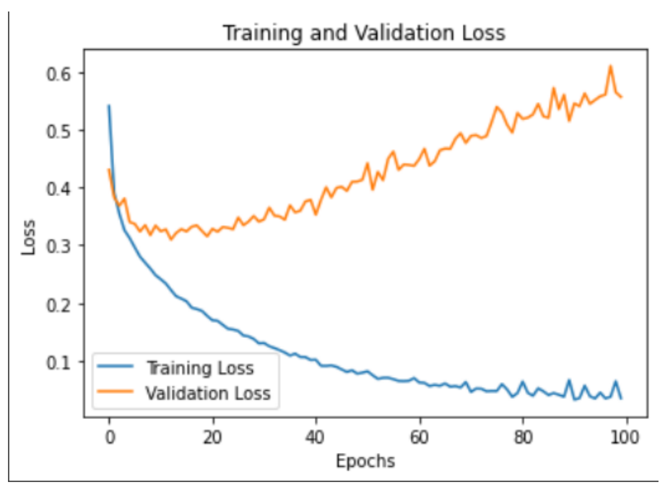
Splitting training set into training and validation:

```
print(x_train.shape)
print(y_train.shape)
print(x_val.shape)
print(y_val.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(51000, 784)
(51000,)
(9000, 784)
(9000,)
(10000, 784)
(10000,)
```

a)

Activation Function	Training Loss	Validation Loss	

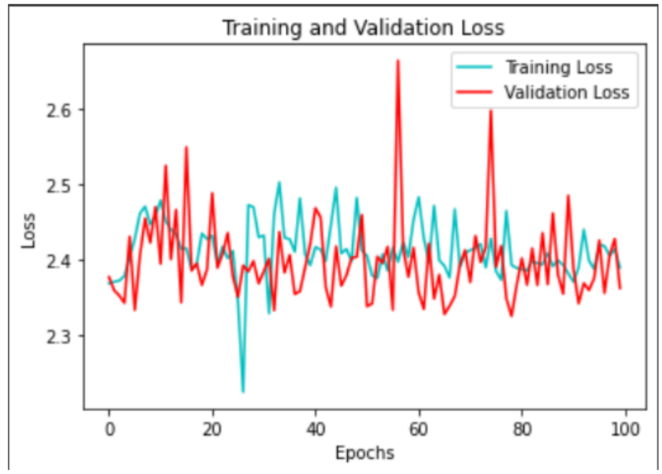
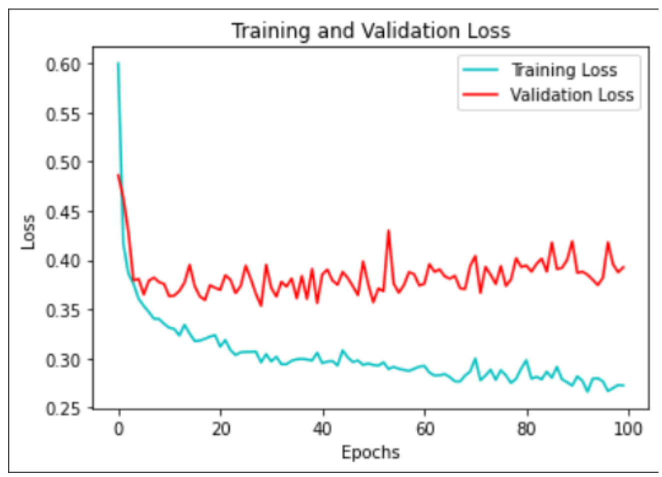
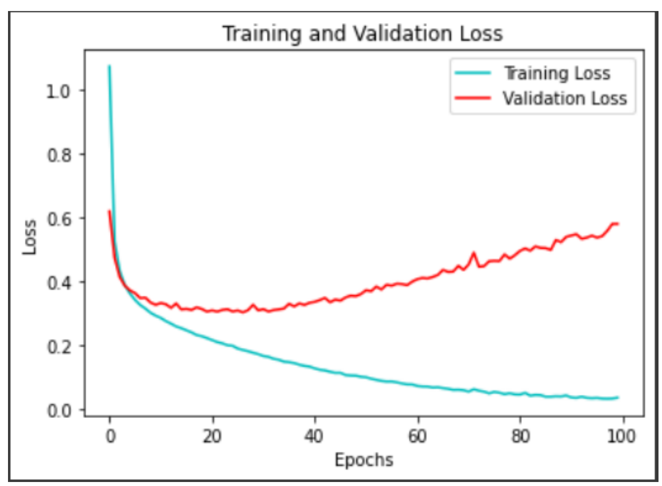
Sigmoid	0.03855906177947863	0.5677353933934056	
Relu	0.03593224917444538	0.7449868720617722	
Tanh	0.0415456268558159	0.5564393887536792	

Linear	0.35062303061717853	0.443836581390874	

Insights:
With Both Relu and Tanh, the model showed higher validation losses. The tanh activation function can be considered good for training. The linear activation function gives the highest training loss, and the validation loss is lower than the sigmoid and tanh graphs.
The sigmoid activation function has the lowest training loss (low bias) and a little higher validation loss, but the variance between them is low. This activation function can be considered fit for training.
I find the Sigmoid activation function to be the most accurate among all others by analyzing the graphs and losses. Hence, I have out further computations using this activation function.

b)

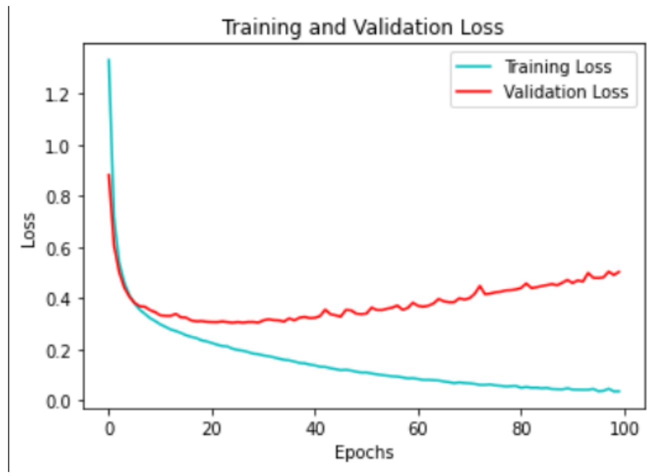
Learning Rate	Training Loss	Validation Loss	

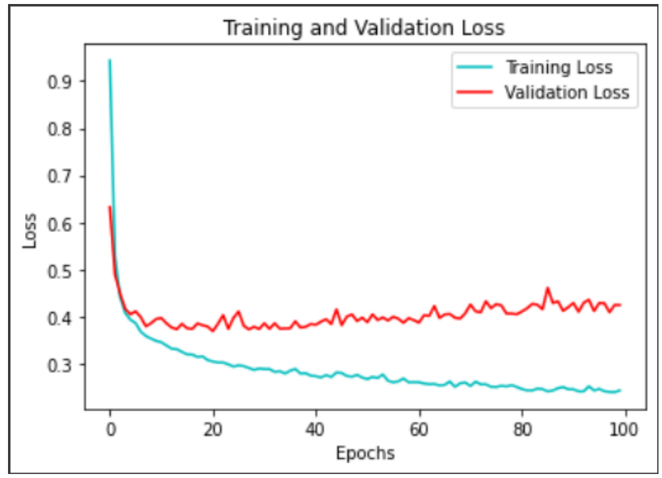
0.1	0.03855906177947863	0.5677353933934056	
0.01	0.03593224917444538	0.7449868720617722	
0.001	0.0415456268558159	0.5564393887536792	

Insights:

The model with learning rate 0.001 is taken to be best due to low bias and better testing accuracy than the rest. For a 0.1 learning rate, the training loss shoots up suddenly at some points. Although the variance is low, the bias is very high.

c)

Number of neurons	Training Loss	Validation Loss	
(256,16)	0.03631520260259406	0.5030008351567431	
(128,32)	0.05468631239536042	0.5083576463953313	

(64,8)	0.24356850096796182	0.42469993519122445	

Insights:
By fiddling with the number of hidden layers, we observe that when we decrease the number of neurons in the first layer, it does not affect much the loss values. When the number of neurons in the second layer is reduced the loss values also reduce, and we get better accuracy.

d)
The number of epochs were reduced in grid search as it took a lot of time to perform this .
Different parameters feeded in GridSearchCV() :

```
parameters={'hidden_layer_sizes': [(256,128),(32,16)],
            "activation":["tanh","logistic"],
            "learning_rate_init":[0.1,0.01,0.001],
            "max_iter" : [40],'batch_size': [64,128,256]}
```

Best features according to grid search:

{'activation': 'tanh', 'batch_size': 256, 'hidden_layer_sizes': (256, 128), 'learning_rate_init': 0.001, 'max_iter': 40}

```
{'activation': 'tanh', 'batch_size': 256, 'hidden_layer_sizes': (256, 128), 'learning_rate_init': 0.001, 'max_iter': 40}
```