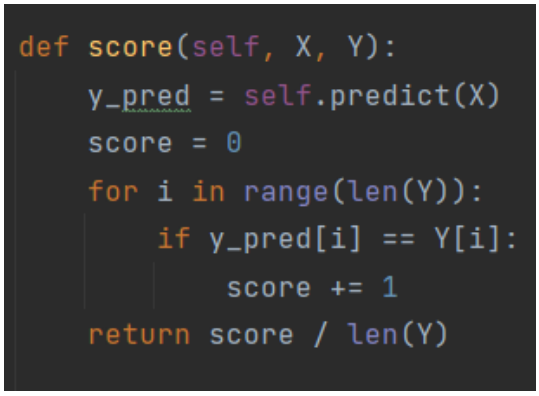
MACHINE LEARNING (CSE343/ECE343)
ASSIGNMENT-3

Sourav Goyal
2020341
SECTION-B

Ans-1
Neural Network class :-

```
class NeuralNetwork:

    def __init__(self, number_of_layers, hidden_layers_sizes, learning_rate, Activation_func, Weight_init_func, epochs,
                 batch_size):
        self.number_of_layers = number_of_layers
        self.layer_sizes = hidden_layers_sizes
        self.lr = learning_rate
        self.activation = Activation_func
        self.weight_init = Weight_init_func
        self.epochs = epochs
        self.batch_size = batch_size
        self.Layers = []
        self.initializeLayers()

    def initializeLayers(self):
        self.Layers.append(Layer(ActivationFunction.Linear(), self.weight_init, 784, 256))
        for i in range(self.number_of_layers - 1):
            self.Layers.append(Layer(self.activation, self.weight_init, self.layer_sizes[i], self.layer_sizes[i + 1]))

        self.Layers.append(Layer(self.activation, self.weight_init, self.layer_sizes[-1], 10))

    def predict_proba(self, X):
        output = X
        for i in range(len(self.Layers)):
            output = self.Layers[i].forward_propogation(output)

        return output
```

```
def fit(self, X_train, Y_train):
    for i in range(self.epochs):
        loss = 0
        for j in range(0, X_train.shape[0], self.batch_size):
            X_t = X_train[j:j + self.batch_size]
            Y_t = Y_train[j:j + self.batch_size]
            for k in range(len(Y_t)):
                x = X_t[k]
                for layer in self.Layers:
                    x = layer.forward_propagation(x)

                error = self.cross_entropy_loss(Y_t[k], x)
                loss += error
                for layer in range(len(self.Layers) - 1, 0, -1):
                    error = self.Layers[layer].backward_propagation(error, self.lr)

        loss /= len(Y_train)
        print(loss)

def cross_entropy_loss(self, Y_true, Y_predict):
    loss = 0
    for i in range(len(Y_true)):
        loss -= math.log(Y_predict[i][Y_true[i] - 1] + 1e-15)
    return loss

def predict(self, X):
    y_pred = self.predict_proba(X)
    return y_pred.argmax(axis=1)
```



```
def score(self, X, Y):
    y_pred = self.predict(X)
    score = 0
    for i in range(len(Y)):
        if y_pred[i] == Y[i]:
            score += 1
    return score / len(Y)
```

Ans-2

```python
import numpy as np


class sigmoid:

    @staticmethod
    def func(x):
        return 1 / (1 - np.exp(-x))

    @staticmethod
    def derivative(x):
        return sigmoid.func(x) * (1 - sigmoid.func(x))


class tanh:

    @staticmethod
    def func(x):
        return np.tanh(x)

    @staticmethod
    def derivative(x):
        return 1 - np.tanh(x) ** 2


class ReLU:

    @staticmethod
    def func(x):
        return np.maximum(0, x)
```

```python
class Leaky_ReLU:

    @staticmethod
    def func(x, a):
        r = np.maximum(0, x)
        if r <= 0:
            return a * x

        else:
            return x

    @staticmethod
    def derivative(x, a):
        if Leaky_ReLU.func(x, a) == x:
            return 1
        else:
            return a


class softmax:

    @staticmethod
    def func(x):
        return np.exp(x) / np.sum(np.exp(x))
```

```python
class Linear:

    @staticmethod
    def func(x):
        return x

    @staticmethod
    def derivative(self):
        return 1
```

Ans-3

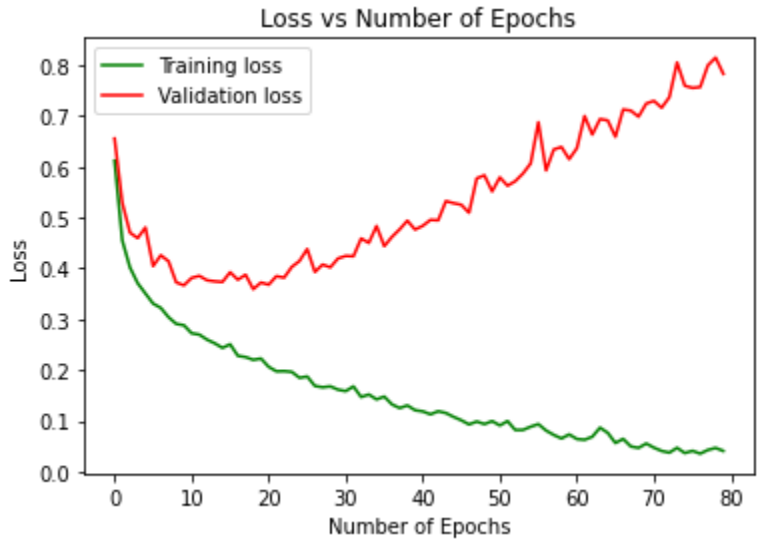
```python
import numpy as np


class zero_init:

    @staticmethod
    def func(m, n):
        return np.zeros((m, n))


class random_init:

    @staticmethod
    def func(m, n):
        return np.random.rand(m,n) * 0.001


class normal_init:

    @staticmethod
    def func(m, n):
        return np.random.normal(size=(m,n)) * 0.1
```

SECTION-C

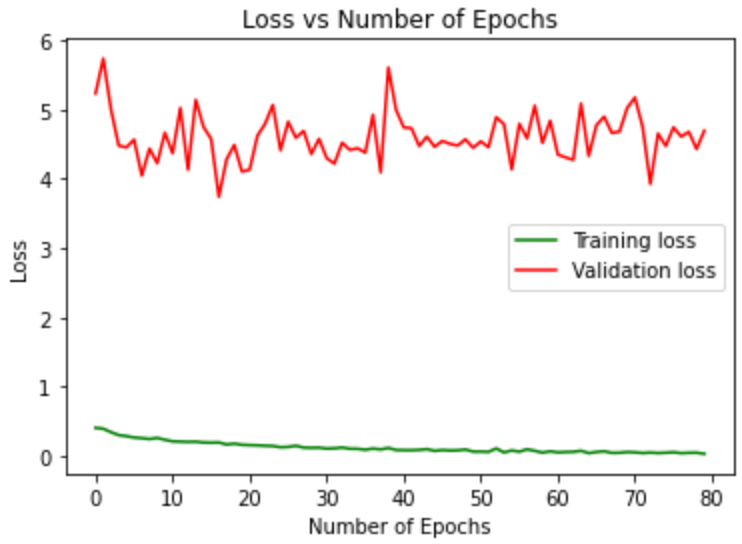
Ans - 1 :-

Training Loss and Validation Loss vs epochs plot :-

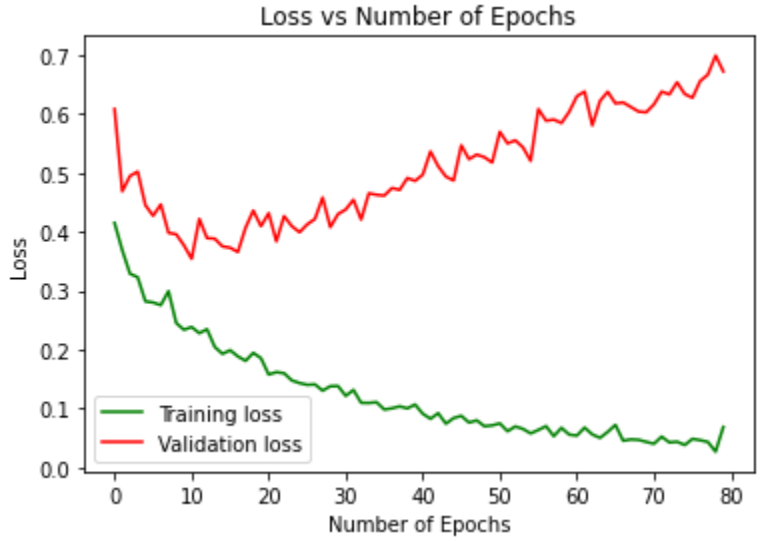
For sigmoid :-



For ReLU :-



For tanh :-



For linear :-

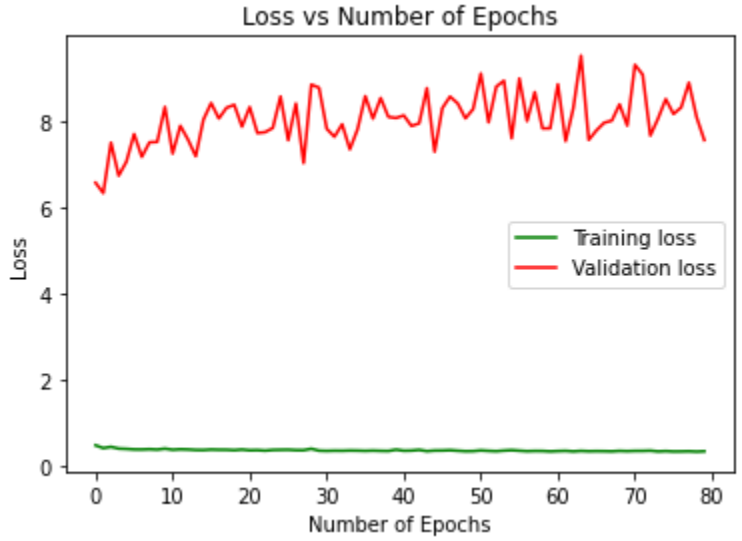


Table for Validation Accuracy and Testing Accuracy For all models :-

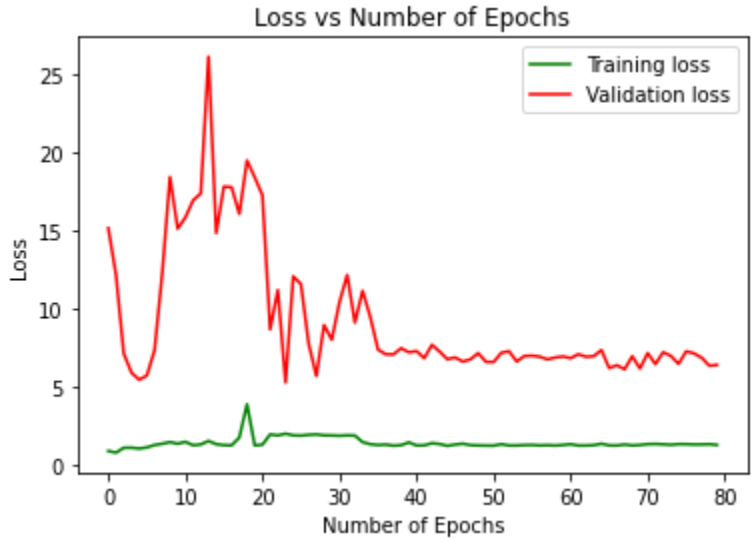
Model Name	Validation Accuracy	Testing Accuracy
linear	0.7742222222222223	0.8455
Sigmoid	0.8456666666666667	0.8987
tanh	0.8601111111111112	0.8896
ReLU	0.8626666666666667	0.8959

We got best validation accuracy on ReLU but testing accuracy is slightly lower than Sigmoid but we see overall accuracy then ReLU performs best.From the Loss curve we can see that sigmoid and tanh having low validation loss and linear and ReLU have high validation loss relatively,this is because in ReLU for some sample of dataset the loss value is very high and same for linear but in case of sigmoid and tanh the loss we getting for all the samples is low.So, now for model selection we will prefer ReLU because it gives correct prediction for more samples than sigmoid and tanh as it's validation accuracy is greater than all other models.

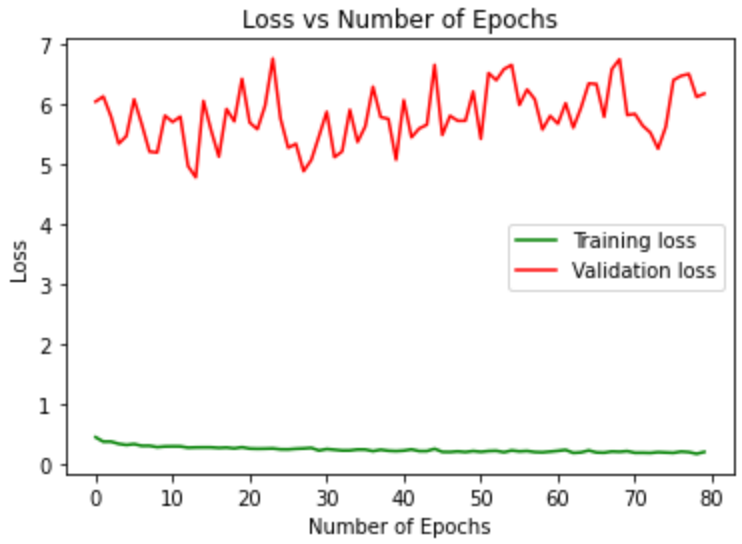
So,the best model is ReLU.

Ans - 2 :-
Training Loss and Validation Loss vs epochs plot :-

For learning rate = 0.1 :-



For learning rate = 0.01 :-



For learning rate = 0.001 :-

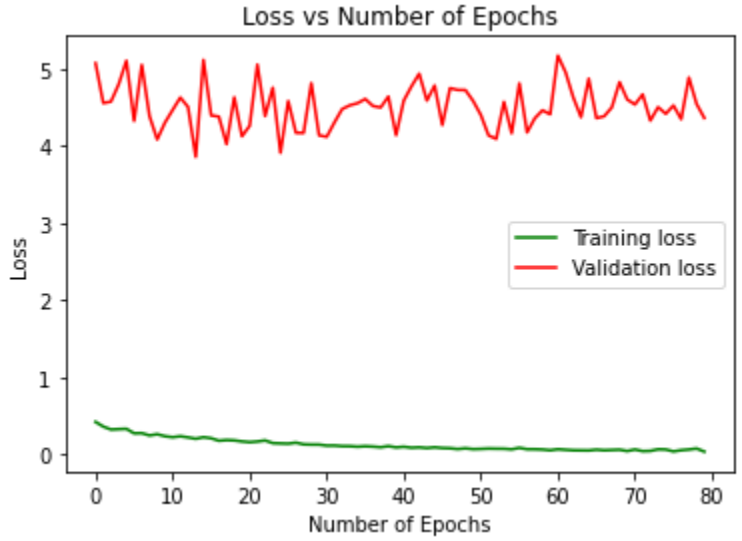


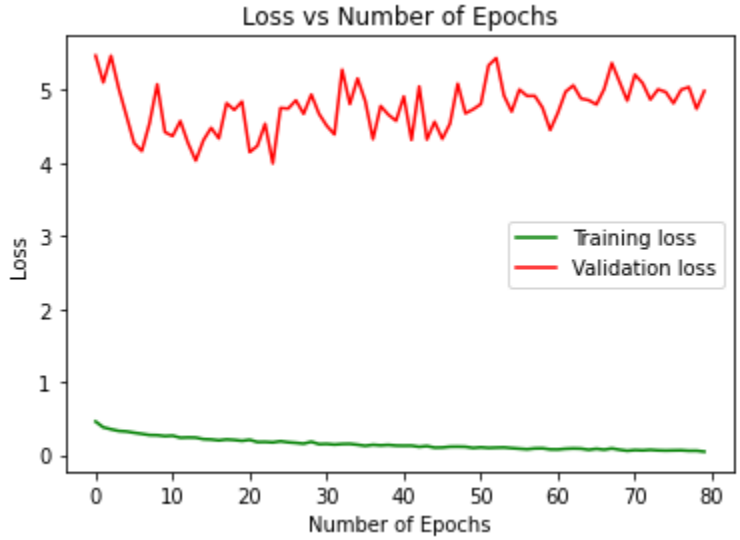
Table for Validation Accuracy and Testing Accuracy For all learning rate :-

Learning Rate	Validation Accuracy	Testing Accuracy
0.1	0.2897777777777778	0.4301
0.01	0.8177777777777778	0.8794
0.001	0.8723333333333333	0.8982

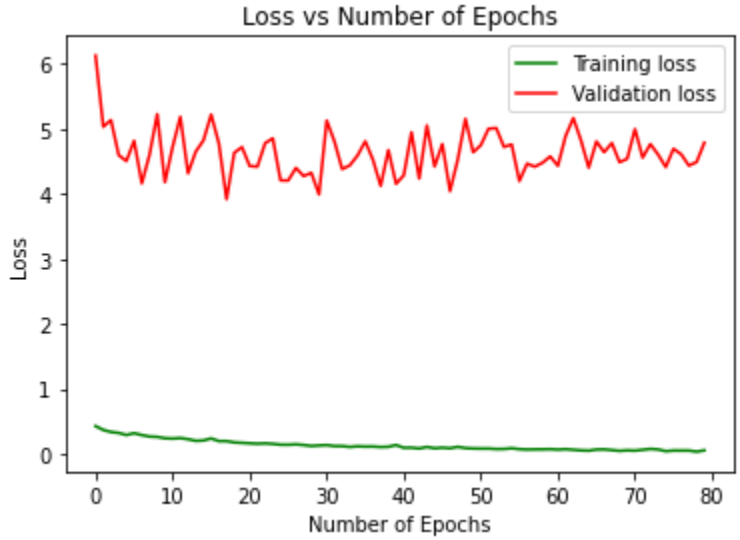
When learning rate = 0.1 we have worst accuracy and this is because 0.1 learning rate is very high for our model as it will overshoot and never come to the minima point where we have best accuracy and for learning rate = 0.01 we have very close accuracy to learning rate 0.001 but not equal because it also slightly lagging to achieve that minima because of its higher rate which

makes our model to slightly overshoot and we have highest accuracy or best performance for learning rate = 0.001 which we can see from the above table.

Ans-3 :-
Training Loss and Validation Loss vs epochs plot :-
For hidden layers size = (128,16) :-



For hidden layers size = (200,24) :-



For hidden layers size = (150,20) :-

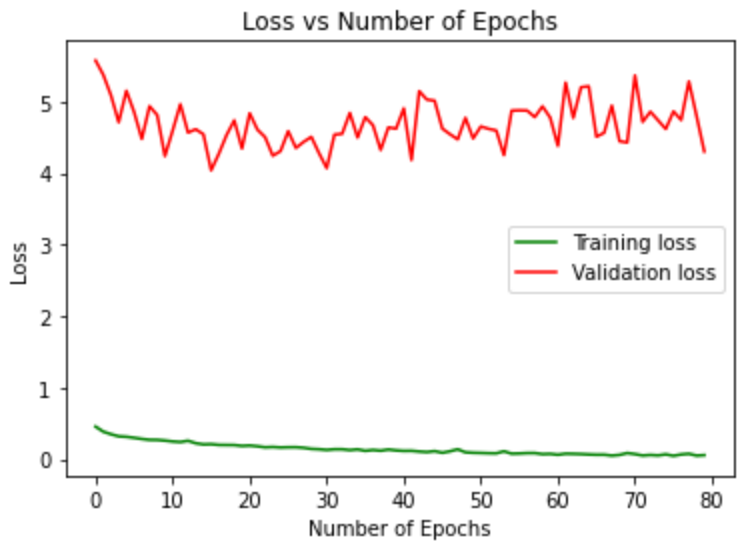


Table for Validation Accuracy and Testing Accuracy For all hidden layer sizes :-

Hidden Layer Sizes	Validation Accuracy	Testing Accuracy
(128,16)	0.8531111111111112	0.8897
(200,24)	0.8601111111111112	0.8896
(150,20)	0.8735555555555555	0.892
(256,32)	0.8723333333333333	0.8959

From the above table we can see that for validation Accuracy the hidden layer sizes are in order :- (128,16) < (200,24) < (150,20) < (256,32) and for testing accuracy we have :- (256,32) > (150,24) > (128,16) > (200,24) and for this behavior one possible reason is when we increasing the layers neuron the model complexity increases and their will be higher chances of overfitting and vice versa can leads to condition of underfitting but here the trend is not strictly increasing or decreasing so this because of our dataset learning which we have, because (150,20) hidden layer size is work better than (200,24) and (128,16) testing accuracy is lower than (200,24) so in case of higher sizes our model is not always overfit and in less sizes our model is not always underfit it totally depends on our loss if it is less than our model is good.

Ans-4 :-

```
parameters = {
    "learning_rate_init": [0.001,0.005],
    "hidden_layer_sizes": [(256,32),(280,40)],
    "batch_size": [256,128],
    "activation": ["relu","logistic"]
}
```

We pass above parameters in grid search and the corresponding result we have below.

```
{'activation': 'relu', 'batch_size': 128, 'hidden_layer_sizes': (280, 40), 'learning_rate_init': 0.001}
```

From here also best activation function we have is Relu and this is seen from 1 st part that ReLU gives highest accuracy and batch size 128 which we take for our above parts and we gave two hidden layer sizes which is (256,32) and (280,40) because we only observe for lower values in 3 rd part and now we see that grid search confirms that on increase hidden layer sizes performance can be increase and for learning rate we have again less value preferable by grid search also because at less learning rate we achieve the minima more precisely.