Sayantan Dutta

I have achieved accuracy of 98.09% in test data. Below is the screenshot:

```
    if (epoch+1) % 5 == 0:
      print(f'Epoch {epoch+1}: Training loss = {train_loss}, Test loss = {test_loss}')
      print(f'Accuracy on train set is {predict(weight_dict_train["z2"], y_train)} %')
      print(f'Accuracy on test set is {predict(weight_dict_test["z2"], y_test)} %')
      print()
```

```
Epoch 5: Training loss = 0.03994524364363759, Test loss = 0.07564415975565933
Accuracy on train set is 98.85 %
Accuracy on test set is 97.63 %

Epoch 10: Training loss = 0.01666063806122683, Test loss = 0.06257853567833026
Accuracy on train set is 99.7367 %
Accuracy on test set is 98.05 %

Epoch 15: Training loss = 0.013332412879018141, Test loss = 0.0628763775228552
Accuracy on train set is 99.85 %
Accuracy on test set is 98.09 %

Epoch 20: Training loss = 0.011781469283703147, Test loss = 0.0628244344769465
Accuracy on train set is 99.8883 %
Accuracy on test set is 98.09 %
```

The MNIST dataset was prepared. One-hot encoding was used for this case of multi-class output. Train and test data were separated. The train data was shuffled.

The hyper-parameters used:

initial learning rate (lr) = 0.5,
no. of epochs (for training) = 20,
number of hidden layers = 1
hidden units (in the hidden layer) = 100
batch size = 16

Other parameters:

input units = 784 (28 * 28)
output units = 10

Weights and biases are initialized randomly for each neuron. The variance in the initialization of the neurons is scaled by dividing with the square root of the number of inputs to a neuron.

Dictionary ‘parameter_dict’ stored the weights and biases values of the initial parameters.

Sigmoid function is used as activation function.
Function name: '**sigmoid_function**'

Negative log likelihood or Cross-Entropy function is used as loss function.
Function name: '**calculate_loss**'

During the training of the neural network, in the feed-forward process dictionary 'weight_dict' stores the weights and biases as the network predicts the outputs.
Function name: '**feedforward**'

In the back-propagation process, Mini-batch Stochastic Gradient Descent process is used.
Function name: '**backpropagation'**

Dictionary 'gradients' stored the gradient value. Mini batches of size 16 is used over many iterations to train the network within each epoch. Parameters are updated after each mini batch is used for feedforward, backward process. The training/testing loss and accuracy are calculated after completion of each epoch.

The initial learning rate is set to 0.5 and is kept unchanged till 30% of the total epochs are completed for training. When the network training is between 30% and 80% of the total epochs the learning rate is reduced to 0.1. Once the network is trained above 80% of the total epochs the learning rate is reduced to 0.01.