Task:

Design a Neural Network of MNIST dataset on plain python.

Using Tensorflow Model

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from tensorflow import keras
```

Load MNIST dataset from Keras

```
[2] mnist = keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Flatten images and normalize pixel values

```
[3] train_images = train_images.reshape((60000, 28 * 28)).astype('float32') / 255.0
```

```
[4] test_images = test_images.reshape((10000, 28 * 28)).astype('float32') / 255.0
```

Convert labels to one-hot encoding

```
[5] train_labels_one_hot = keras.utils.to_categorical(train_labels)
```

```
[6] test_labels_one_hot = keras.utils.to_categorical(test_labels)
```

Build and compile the Keras model

```
[7] model = keras.Sequential([
        keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
    ])
```

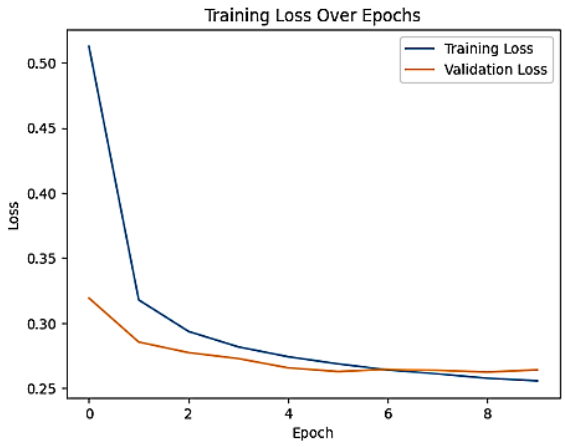
```
[8] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Train the model

```
[9] history = model.fit(train_images, train_labels_one_hot, epochs=10, validation_split=0.2, verbose=0)
```

Plot training loss over epochs

```
[10] plt.plot(history.history['loss'], label='Training Loss')
     plt.plot(history.history['val_loss'], label='Validation Loss')
     plt.xlabel('Epoch')
     plt.ylabel('Loss')
     plt.title('Training Loss Over Epochs')
     plt.legend()
     plt.show()
```



Evaluate on test set

```
[11] test_loss, test_accuracy = model.evaluate(test_images, test_labels_one_hot)
     print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2698 - accuracy: 0.9264
Test Accuracy: 92.64%
```

Now get the value of weights and bias from the model

```
[12] weights, bias = model.get_weights()
```

```
[13] weights
```

```
array([[-0.07202272, -0.07711218,  0.05182669, ...,  0.06341704,
         0.02931244,  0.0829015 ],
       [-0.05868655,  0.04416948,  0.04840187, ...,  0.03160163,
        -0.00801334,  0.07250386],
       [-0.05211153,  0.01679573, -0.07133962, ...,  0.05363155,
        -0.02165066, -0.06756246],
       ...,
       [ 0.07924647,  0.00606389, -0.06496885, ...,  0.05554896,
        -0.07870241,  0.08497936],
       [ 0.02361482, -0.01315942,  0.05453864, ...,  0.07539699,
         0.04969002,  0.08483953],
       [ 0.06968097,  0.06087834,  0.04394469, ...,  0.00038122,
         0.01825912,  0.05936564]], dtype=float32)
```

```
[14] bias
```

```
array([-0.40560353,  0.61046225,  0.09786658, -0.37033457,  0.14473958,
        1.2148662 , -0.18581295,  0.6701259 , -1.3553874 , -0.2920264 ],
      dtype=float32)
```

Design a Neural Network of MNIST dataset on plain python.

Import Libraires

```
[15] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

Dataset Loading

```
[19] train = pd.read_csv('/content/drive/MyDrive/AI Lab Semester 5/AI Lab 10/Home Task/mnist_train.csv')
```

```
[20] test = pd.read_csv('/content/drive/MyDrive/AI Lab Semester 5/AI Lab 10/Home Task/mnist_test.csv')
```

```
[21] train_labels = train['label']
```

```
[22] train_images = train.drop('label', axis=1)
```

```
[23] test_labels = test['label']
```

```
[24] test_images = test.drop('label', axis=1)
```

```
[25] def log_loss(y_true, y_pred, epsilon=1e-15):
         y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
         loss = - np.sum(y_true * np.log(y_pred)) / len(y_true)
         return loss
```

```
[26] def sigmoid_numpy(x):
       return 1.0 / (1.0 + np.exp(-np.clip(x, -700, 700)))
```

```
[27] class myNN:
         def __init__(self):
             self.weights = np.random.randn(train_images.shape[1], 10) * np.sqrt(2 / train_images.shape[1])
             self.bias = np.zeros(10)
         def calculate_accuracy(self, y_true, y_predicted):
             if len(y_true.shape) > 1:
                 y_true = np.argmax(y_true, axis=1)
             y_pred_classes = np.argmax(y_predicted, axis=1)
             accuracy = np.mean(y_true == y_pred_classes)
             return accuracy
         def predict(self, X):
             weighted_sum = np.dot(X, self.weights) + self.bias
             y_predicted = sigmoid_numpy(weighted_sum)
             return y_predicted
         def fit(self, X, y, epochs, loss_threshold, learning_rate=0.01, clip_threshold=1.0):
             y_one_hot = pd.get_dummies(y).values
             self.losses = []
             n = len(X)
             for i in range(epochs):
                 weighted_sum = np.dot(X, self.weights) + self.bias
                 y_predicted = sigmoid_numpy(weighted_sum)
                 loss = log_loss(y_one_hot, y_predicted)
                 wd = (1/n) * np.dot(np.transpose(X), (y_predicted - y_one_hot))
                 bias_d = np.mean(y_predicted - y_one_hot)
                 wd = np.clip(wd, -clip_threshold, clip_threshold)
                 bias_d = np.clip(bias_d, -clip_threshold, clip_threshold)
                 self.weights = self.weights - learning_rate * wd
                 self.bias = self.bias - learning_rate * bias_d
                 self.losses.append(loss)
                 if i % 50 == 0:
                     accuracy = self.calculate_accuracy(np.argmax(y_one_hot, axis=1), y_predicted)
                     print(f'Epoch:{i}, Accuracy: {accuracy}, Loss:{loss}')
                 if loss <= loss_threshold:
                     accuracy = self.calculate_accuracy(np.argmax(y_one_hot, axis=1), y_predicted)
                     print(f'Epoch:{i}, Accuracy: {accuracy}, Loss:{loss}')
                     break
```

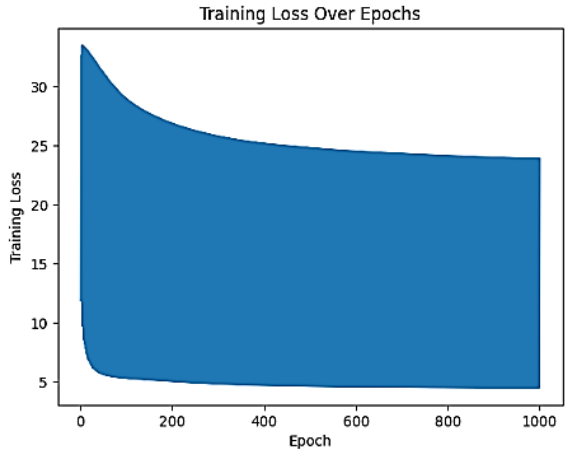
```
[28] customModel = myNN()
```

```
customModel.fit(train_images, train_labels, epochs=1000, loss_threshold=0.4631)
```

```
Epoch:0, Accuracy: 0.08318333333333333, Loss:11.859549767667446
Epoch:50, Accuracy: 0.5864666666666667, Loss:5.600875928839614
Epoch:100, Accuracy: 0.6246666666666667, Loss:5.297582221512847
Epoch:150, Accuracy: 0.6455666666666666, Loss:5.159567188493348
Epoch:200, Accuracy: 0.6595333333333333, Loss:5.029491398385932
Epoch:250, Accuracy: 0.6714166666666667, Loss:4.91539683636012
Epoch:300, Accuracy: 0.6792666666666667, Loss:4.823990940605462
Epoch:350, Accuracy: 0.6850666666666667, Loss:4.746745387261592
Epoch:400, Accuracy: 0.6897666666666666, Loss:4.6931557616510675
Epoch:450, Accuracy: 0.6937333333333333, Loss:4.654140225570119
Epoch:500, Accuracy: 0.6964, Loss:4.619395824024862
Epoch:550, Accuracy: 0.6989833333333333, Loss:4.591996747087752
Epoch:600, Accuracy: 0.70095, Loss:4.571007472803092
Epoch:650, Accuracy: 0.7031166666666666, Loss:4.555105441475565
Epoch:700, Accuracy: 0.7041666666666667, Loss:4.542289528256257
Epoch:750, Accuracy: 0.7054166666666667, Loss:4.528320509039047
Epoch:800, Accuracy: 0.7063833333333334, Loss:4.516215387561198
Epoch:850, Accuracy: 0.7073833333333334, Loss:4.505877323558311
Epoch:900, Accuracy: 0.7082166666666667, Loss:4.496045645361549
Epoch:950, Accuracy: 0.7090166666666666, Loss:4.487611581676622
```

Plotting Training Loss Over Epochs

```
epochs = len(customModel.losses)
plt.plot(range(1, epochs + 1), customModel.losses)
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.title('Training Loss Over Epochs')
plt.show()
```



Evaluate on Test Set

```
[32] def calculate_accuracy(model, test_images, test_labels):
         predicted_labels = np.argmax(model.predict(test_images), axis=1)
         accuracy = np.mean(predicted_labels == test_labels)
         return accuracy
```

```
[33] test_accuracy = calculate_accuracy(customModel, test_images, test_labels)
     print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
Test Accuracy: 70.73%
```