Neural Nerworks

Shamik Mukherji
School of Computer Science and Engineering,
B.Tech, 2nd year, VIT Vellore

May 2025

Acknowledgement

This project is done under the supervision of Prof. Rajat De , ISI Kolkata. I am grateful to him for spending his valuable time with me in spite of his busy schedule and for sharing his insights.

1 Aim of the Project

We need to build a Neural Network model that examines the behavior of multilayer perceptrons and their accuracy in predicting the output of the MNIST dataset.

2 Implementation of Multilayer Perceptron using Python

2.1 Example Taken: MNIST Dataset

```
MNIST dataset is a large database of handwritten digits (up to 10)
that is commonly used for training various image processing systems.
It is a dataset of 60,000 28x28 greyscale images of the 10 digits,
along with a test set of 10,000 images
```

2.2 Diagram of the network

There are a total of 784 inputs. Considering 20 perceptrons in Hidden Layer and 10 perceptrons in the Output Layer.

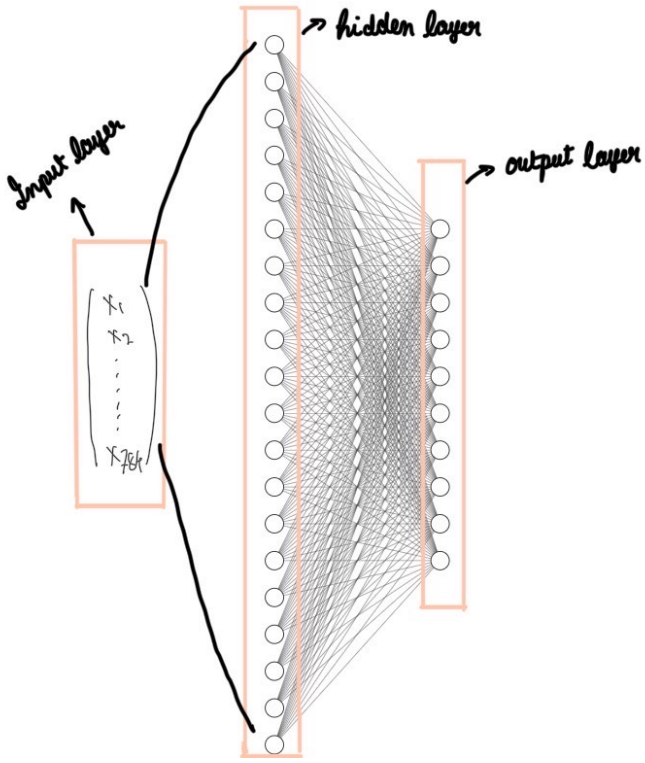


Figure 1: MNIST Neural Network Diagram

2.3 Important Theory

Gradient Desent- Powerful optimization algorithm for machine learning models.

There are 3 types of gradient decent learning algorithms:

1> BATCH: Sums the entry of each point on the dataset, updating the model after all the training examples have been evaluated. It is highly computationally effective but has a slow processing time.

2> STOCHASTIC: Evaluates each example one at a time instead of the whole batch. It its not as computationally effective as compared to Batch mode but has a fast processing time,

3> MINIBATCH: Splits the training data into small batches and performs updates on each of those.

The only disadvantage of gradient descent is that it faces problem in finding the global minimum of non comvex problems.

There are three main types of learnings in every machine learning model:

1>Supervised Learning: Machine learning model is trained on a labeled dataset.It learns from the dataset by iteratively making predictions and adjustinf weights
Eg-classification and regression

2>Unsupervised Learning: No labels are given to the machine learning model. These algorithms discover hidden pattern in the data without the need of human intervension.
Eg: Clustering (where algorithm gorups similar experiences togather), Assosiation, Dimensional reduction.

3> Semi Supervised Learning: A training dataset is used whose some part is labeled and the rest is not. Used when we have a high volume of data.

Validation Process(Done for Hyperparameter tuning):

Cross-Validation:

It is a statistical method used in machine learning to evaluate the performance of a model.

The primary goal is to assess how well a machine learning model generalizes to unseen data.

Involves splitting the dataset into multiple subsets where the model is trained on one portion of the data and tested on other.

Advantages:

1>Reduce overfitting
(train data accuracy ↑↑, test data accuracy ↓↓)

2>Efficient use of data

3>Reliable performance estimation

K-fold cross-validation:

The dataset is divided into k-equally sized subsets or folds.The model is trained on k-1 folds and tested on the remaining fold. This process is repeated k times with each fold serving as the test once.

For the first perceptron in the hidden layer, there are 784 different weights and only 1 bias. This is the same for all the perceptrons in the hidden layer.

So the weight Matrix for this is of the dimension 20x784 and the bias matrix for this is of the dimension 20x1. But for simplicity, the bias elements are submerged with the weight elements and a new weight matrix is formed. This new weight matrix is of the dimension 20x785.

Similarly for the perceptrons present in the output layer, there are 20 different weights and 1 bias for each perceptron and there are a total of 10 perceptrons in the output layer.

So the weight matrix for this is of the dimension 10x20 and the bias matrix for this is of the dimension 10x1. But again the bias elements are submerged with the weight elements and a new weight matrix is formed. This new weight matrix is of the dimension 10x21.

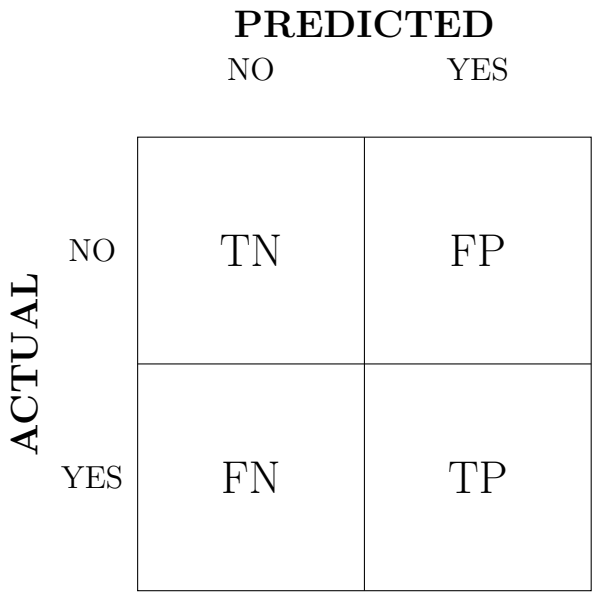
We need to build a model such that we get the highest accuracy for prediction. For that, we need to train the weights and biases so that they obtain the best value for highest accuracy prediction.

There are two main steps for training the model:

1. Forward Propagation : It is the process of passing input data through the neural network to get an output prediction.It calculatesthe output of the network based on current weights and biases. It is also used to compute the loss (error) which we want to minimize.

2. Backward Propagation: It is the process of making the model learn form its mistake. It tells us how to adjust the weights and biases so that the error can be minimized. It basically updates the values of the weights and biases so that the error is reduced.

We will also use the sigmoid function which maps any number between 0 and 1




```
Here TN : True Negative
     TP : True Positive
     FN : False Negative
     FP : False Positive
```

$$\textbf{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Error Rate} = 1 - \text{Accuracy}$$

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

$$\textbf{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

3 Mathematical Interpretation

3.1 Forward propagation Equations:

```
z1 =  first_weight_matrix . input_matrix
```

```
a1 = sigmoid(z1)
```

$$z^{[1]} = \begin{bmatrix} w_0^{[1]} & w_1^{[1]} & w_2^{[1]} & w_3^{[1]} & w_4^{[1]} & \cdots & w_{784}^{[1]} \\ w_{785}^{[1]} & w_{786}^{[1]} & w_{787}^{[1]} & w_{788}^{[1]} & w_{789}^{[1]} & \cdots & w_{1569}^{[1]} \\ w_{1570}^{[1]} & w_{1571}^{[1]} & w_{1572}^{[1]} & w_{1573}^{[1]} & w_{1574}^{[1]} & \cdots & w_{2354}^{[1]} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{14915}^{[1]} & w_{14916}^{[1]} & w_{14917}^{[1]} & w_{14918}^{[1]} & w_{14919}^{[1]} & \cdots & w_{15699}^{[1]} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{784} \end{bmatrix}$$

$$z^{[1]} = \begin{bmatrix} \sum_{j=0}^{784} w_j^{[1]} x_j \\ \sum_{j=0}^{784} w_{785+j}^{[1]} x_j \\ \sum_{j=0}^{784} w_{1570+j}^{[1]} x_j \\ \vdots \\ \sum_{j=0}^{784} w_{14915j}^{[1]} x_j \end{bmatrix}$$

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ \vdots \\ z_{20}^{[1]} \end{bmatrix}$$

$$a^{[1]} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \vdots \\ \sigma(z_{20}^{[1]}) \end{bmatrix}$$

```
z2= second_weight_matrix . a1

a2=sigmoid(z2)
```

$$z^{[2]} = \begin{bmatrix} w_0^{[2]} & w_1^{[2]} \cdots & w_{20}^{[2]} \\ \vdots & \vdots & \vdots \\ w_{189}^{[2]} & w_{190}^{[2]} \cdots & w_{209}^{[2]} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \\ \vdots \\ \sigma(z_{20}^{[1]}) \end{bmatrix}$$

$$z^{[2]} = \begin{bmatrix} w_0^{[2]} + \sum_{j=1}^{20} w_j^{[2]} \sigma(z_j^{[2]}) \\ w_{21}^{[2]} + \sum_{j=1}^{20} w_{21+j}^{[2]} \sigma(z_j^{[2]}) \\ \vdots \\ w_{189}^{[2]} + \sum_{j=1}^{20} w_{189+j}^{[2]} \sigma(z_j^{[2]}) \end{bmatrix}$$

$$a^{[2]} = \begin{bmatrix} \sigma(z_1^{[2]}) \\ \sigma(z_2^{[2]}) \\ \vdots \\ \sigma(z_{10}^{[2]}) \end{bmatrix}$$

3.2 Loss Function:

`The binary cross-entropy loss function is given by:`

$$\mathcal{L} = -\left[y \log \hat{y} + (1-y) \log(1-\hat{y})\right]$$

where

- y is the ground truth (actual label)
- $\hat{y}$ is the model generated output (predicted probability)

3.3 Backward propagation Equations:

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = -\left[y \frac{\partial}{\partial w^{[2]}} \log \sigma(w^{[2]} a^{[1]}) + (1-y) \frac{\partial}{\partial w^{[2]}} \log(1 - \sigma(w^{[2]} a^{[1]}))\right]$$

We know:

$$\sigma(w^{[2]}a^{[1]}) = a^{[2]}$$

and also:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Therefore,

$$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = -\left[\frac{y}{a^{[2]}} a^{[2]}(1 - a^{[2]})(a^{[1]})^T\right]$$

The transpose is taken to abide by the matrix multiplication rule i.e.the column number of the first matrix should match with the row number of the second matrix

$$= -y(1 - a^{[2]})(a^{[1]})^T - (1 - y)a^{[2]}(a^{[1]})^T$$

$$= -\left[ya^{[1]T} - ya^{[2]}a^{[1]T} - a^{[2]}a^{[1]T} + ya^{[2]}a^{[1]T}\right]$$

$$= -(y - a^{[2]})a^{[1]T}$$

$$= -\left[ya^{[1]T} - a^{[2]}a^{[1]T}\right]$$

$$= -(a^{[1]})^T(y - a^{[2]})$$

$$= -a^{[1]T}(y - a^{[2]})$$

Updated Weights:

$$w^{[2]} = w^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial w^{[2]}}$$

Where $\alpha \rightarrow$ learning rate.

$$\frac{\partial \mathcal{L}}{\partial w^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

We know:

$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = -\left[y \cdot \frac{d}{da^{[2]}} \log a^{[2]} + (1 - y) \cdot \frac{d}{da^{[2]}} \log(1 - a^{[2]})\right]$$

$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = -\left[y \cdot \frac{1}{a^{[2]}} - (1-y) \cdot \frac{1}{1-a^{[2]}} \right]$$
$$= -\left[\frac{y(1-a^{[2]}) - (1-y)a^{[2]}}{a^{[2]}(1-a^{[2]})} \right]$$
$$= -\left[\frac{y - ya^{[2]} - a^{[2]} + ya^{[2]}}{a^{[2]}(1-a^{[2]})} \right]$$
$$= -\left[\frac{y - a^{[2]}}{a^{[2]}(1-a^{[2]})} \right]$$

Now,

$$\frac{da^{[2]}}{dz^{[2]}} = \sigma'(z^{[2]}) = a^{[2]}(1-a^{[2]})$$

So,

$$\frac{dz^{[2]}}{da^{[1]}} = w^{[2]} \quad \text{and} \quad \frac{da^{[1]}}{dz^{[1]}} = \sigma'(z^{[1]}) = a^{[1]}(1-a^{[1]})$$

Also,

$$\frac{dz^{[1]}}{dw^{[1]}} = x^T \quad \text{(input matrix)}$$

The transpose is taken to abide by the matrix multiplication rule i.e.the column number of the first matrix should match with the row number of the second matrix

$$\Rightarrow \frac{d\mathcal{L}}{dw^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{da^{[2]}}{dz^{[2]}} \cdot \frac{dz^{[2]}}{da^{[1]}} \cdot \frac{da^{[1]}}{dz^{[1]}} \cdot \frac{dz^{[1]}}{dw^{[1]}}$$

$$\Rightarrow \frac{d\mathcal{L}}{dw^{[1]}} = -\left(\frac{y - a^{[2]}}{a^{[2]}(1-a^{[2]})} \right) \cdot \left(a^{[2]}(1-a^{[2]}) \right) \cdot \left(w^{[2]} \right) \cdot \left(a^{[1]}(1-a^{[1]}) \right) \cdot x^T$$
$$= -\left[(y - a^{[2]}) \cdot w^{[2]} \cdot a^{[1]}(1-a^{[1]}) \cdot x^T \right]$$

Updated Weights :

$$w^{[1]} = w^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial w^{[1]}}$$

`Thus the weights keep on getting updated in such a way that the loss value decreases as a result the accuracy increases.`

4 Python Code

```
import numpy as np
import matplotlib.pyplot as plt
import pathlib
from tensorflow.keras.datasets import mnist #imports the MNIST dataset from
Keras, which is a high-level API for building and training deep learning
models in TensorFlow

# Save MNIST data once (only the first time)
(x_train, y_train), (_, _) = mnist.load_data()
np.savez("mnist.npz", x_train=x_train, y_train=y_train) # saving mnist data
to a file named mnist.npz

# Function to load and preprocess the dataset
def get_mnist():
    data_path = pathlib.Path(__file__).parent / "mnist.npz"
    with np.load(data_path) as f:
        images, labels = f["x_train"], f["y_train"]
    images = images.astype("float32") / 255 #normalizes the pixel values of
    image data so that each pixel is between 0.0 and 1.0 instead of 0 to 255.
    images = images.reshape((images.shape[0], 784))  # 60000x784
    labels = np.eye(10)[labels]  # one-hot encode
    return images, labels

# Load data
images, labels = get_mnist()

# Initialize weights and biases
w_i_h = np.random.uniform(-0.5, 0.5, (20, 785))  # input → hidden #20x785
w_h_o = np.random.uniform(-0.5, 0.5, (10, 21))   # hidden → output #10x21


# Training hyperparameters
learn_rate = float(input("Enter learning rate:"))
epochs = int(input("Enter number of iterations:"))

# Training loop
for epoch in range(epochs):
    nr_correct = 0
    for img, l in zip(images, labels):
        img = img.reshape(784, 1) # Original 784×1 vector
        img_augmented = np.vstack(([1], img)) #785x1
        l = l.reshape(10, 1) #10x1
```

```
        # --- Forward propagation ---
        h_pre = w_i_h @ img_augmented #20x1
        h = 1 / (1 + np.exp(-h_pre))  # Sigmoid activation #20x1
        h_augmented = np.vstack(([1], h)) #21x1
        o_pre =w_h_o @ h_augmented  #10x1
        o = 1 / (1 + np.exp(-o_pre))  # Sigmoid activation #10x1

        # --- Error & accuracy ---
        e = (1 / len(o)) * np.sum((o - l) ** 2) # error is calculated through
        Mean Square Error not Cross-Entropy
        nr_correct += int(np.argmax(o) == np.argmax(l))

        # --- Backpropagation ---
        delta_o = o - l #10x1
        w_h_o -= learn_rate * delta_o @ h_augmented.T #10x21

        delta_h = (w_h_o.T @ delta_o) * (h_augmented * (1 - h_augmented)) #21x1
        w_i_h -= learn_rate * delta_h[1:] @ img_augmented.T #20x785


    # Epoch accuracy
    acc = round((nr_correct / images.shape[0]) * 100, 2)
    print(f"Epoch {epoch + 1}: Accuracy = {acc}%")
all_preds = []
all_labels = []

for img, l in zip(images, labels):
    img = img.reshape(784, 1)
    img_augmented = np.vstack(([1], img))

    h_pre = w_i_h @ img_augmented
    h = 1 / (1 + np.exp(-h_pre))
    h_augmented = np.vstack(([1], h))
    o_pre = w_h_o @ h_augmented
    o = 1 / (1 + np.exp(-o_pre))

    all_preds.append(o.flatten()) #10x1
    all_labels.append(l.flatten()) #10x1
all_preds = np.array(all_preds)
all_labels = np.array(all_labels)
y_true = np.argmax(all_labels, axis=1)
y_pred = np.argmax(all_preds, axis=1)

accuracy = np.sum(y_true == y_pred) / len(y_true)

num_classes = 10
```

```
precision_per_class = []
recall_per_class = []
f1_per_class = []

for cls in range(num_classes):
    tp = np.sum((y_pred == cls) & (y_true == cls))  #how many elemsnts have
    actual and predicted values same
    fp = np.sum((y_pred == cls) & (y_true != cls))   #how many elements have
    the actual value wrong but prediction is correct
    fn = np.sum((y_pred != cls) & (y_true == cls))   # how many element have
    actual value correct but prediction is wrong
    tn= np.sum((y_pred != cls) & (y_true != cls))
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0
    f1 = (2 * precision * recall) / (precision + recall) if (precision +
    recall) != 0 else 0


    precision_per_class.append(precision)
    recall_per_class.append(recall)
    f1_per_class.append(f1)

macro_precision = np.mean(precision_per_class)
macro_recall = np.mean(recall_per_class)
macro_f1 = np.mean(f1_per_class)

print("\n--- Manual Classification Metrics ---")
print(f"Accuracy : {accuracy * 100:.2f}%")
print(f"Precision: {macro_precision * 100:.2f}%")
print(f"Recall   : {macro_recall * 100:.2f}%")
print(f"F1 Score : {macro_f1 * 100:.2f}%")
# Calculate ROC and AUC for each class
thresholds = np.linspace(0, 1, 100)  # 100 threshold points from 0.0 to 1.0
tpr_dict = {}
fpr_dict = {}

for cls in range(num_classes):
    y_true = all_labels[:, cls]     # True labels: 1 if the actual digit is `cls`, else 0
    y_score = all_preds[:, cls]     # Model's predicted probabilities for digit `cls`

    tpr_list = []
    fpr_list = []

    for threshold in thresholds:
        y_pred = (y_score >= threshold).astype(int)  # Apply threshold to get
        binary prediction
```

```
        tp = np.sum((y_pred == 1) & (y_true == 1))
        fp = np.sum((y_pred == 1) & (y_true == 0))
        fn = np.sum((y_pred == 0) & (y_true == 1))
        tn = np.sum((y_pred == 0) & (y_true == 0))

        tpr_val = tp / (tp + fn) if (tp + fn) != 0 else 0
        fpr_val = fp / (fp + tn) if (fp + tn) != 0 else 0

        tpr_list.append(tpr_val)
        fpr_list.append(fpr_val)

    tpr_dict[cls] = tpr_list
    fpr_dict[cls] = fpr_list
plt.figure(figsize=(10, 8))
for cls in range(num_classes):
    plt.plot(fpr_dict[cls], tpr_dict[cls], label=f"Digit {cls}")
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Manual ROC Curves for Each Digit (One-vs-All)")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()




# --- Testing loop ---
while True:
    index = int(input("Enter a number (0 - 59999): "))
    img = images[index].reshape(784, 1)
    label = np.argmax(labels[index])
    img_augmented = np.vstack(([1], img))


    # Forward pass
    h_pre = w_i_h @ img_augmented
    h = 1 / (1 + np.exp(-h_pre))
    h_augmented = np.vstack(([1], h))
    o_pre =  w_h_o @ h_augmented
    o = 1 / (1 + np.exp(-o_pre))

    prediction = np.argmax(o)
    print(f"Label: {label}")
```

```
plt.imshow(img.reshape(28, 28), cmap="Greys")
plt.show()
```

5 User Input and Output

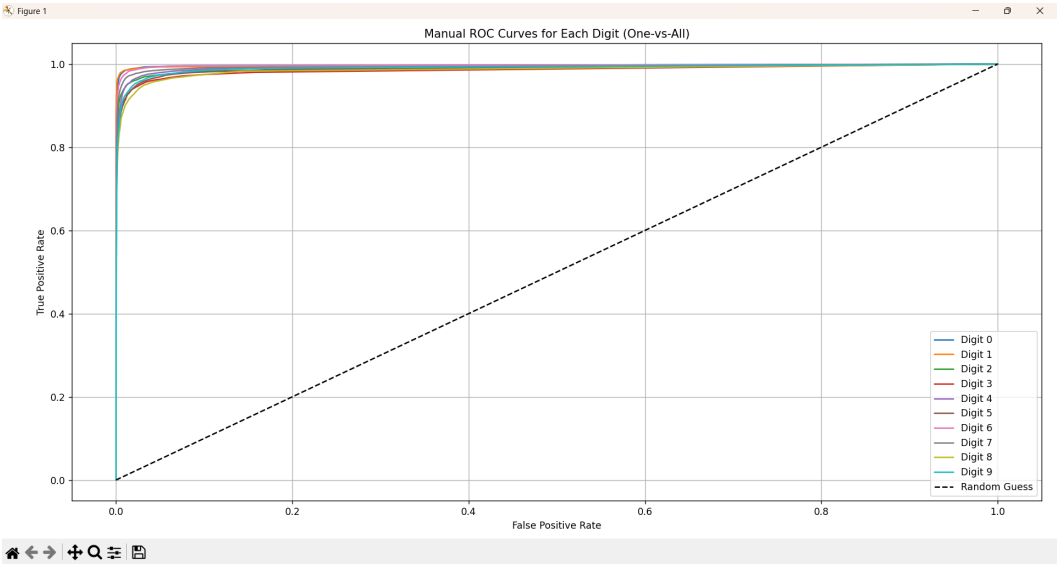


Figure 2: ROC Curve

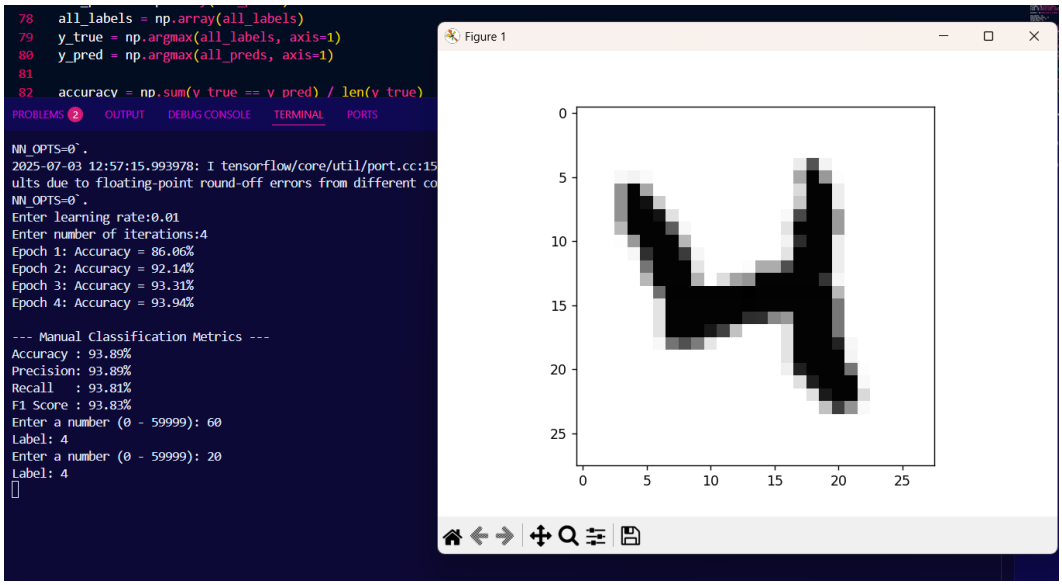


Figure 3: Output Figure

Here we see that the accuracy is being measured for each epoch (iteration). After that we are requested to enter a number between 0-59999. The code then returns us the approprite value (label) of the corresponding row present in the MNIST dataset.

6 Summary

This project implements a neural network model to classify handwritten digits from the MNIST dataset. We built a model.

Key objectives included:

1> Understanding and implementing forward and backward propagation.

2> Deriving the gradients analytically and confirming them through implementation.

3> Training the model to achieve high accuracy on unseen test data.

The trained model successfully recognized handwritten digits with high accuracy, demonstrating the effectiveness of neural networks in pattern recognition tasks. This project also provided hands-on experience in deep learning fundamentals such as weight initialization, activation functions, loss functions, and backpropagation