Building a neural network FROM SCRATCH

Important

The content and code in this document are based on a YouTube tutorial by the original creator (credited in the online reference). These are my personal notes summarizing and implementing the concepts demonstrated in the video.

https://petite-salmon-b93.notion.site/Building-a-neural-network-FROM-SCRATCH-1fe136d1791480079f04ca81cd562ab9?pvs=73

OVERVIEW

In this, we will be building a NEURAL NETWORK from scratch.

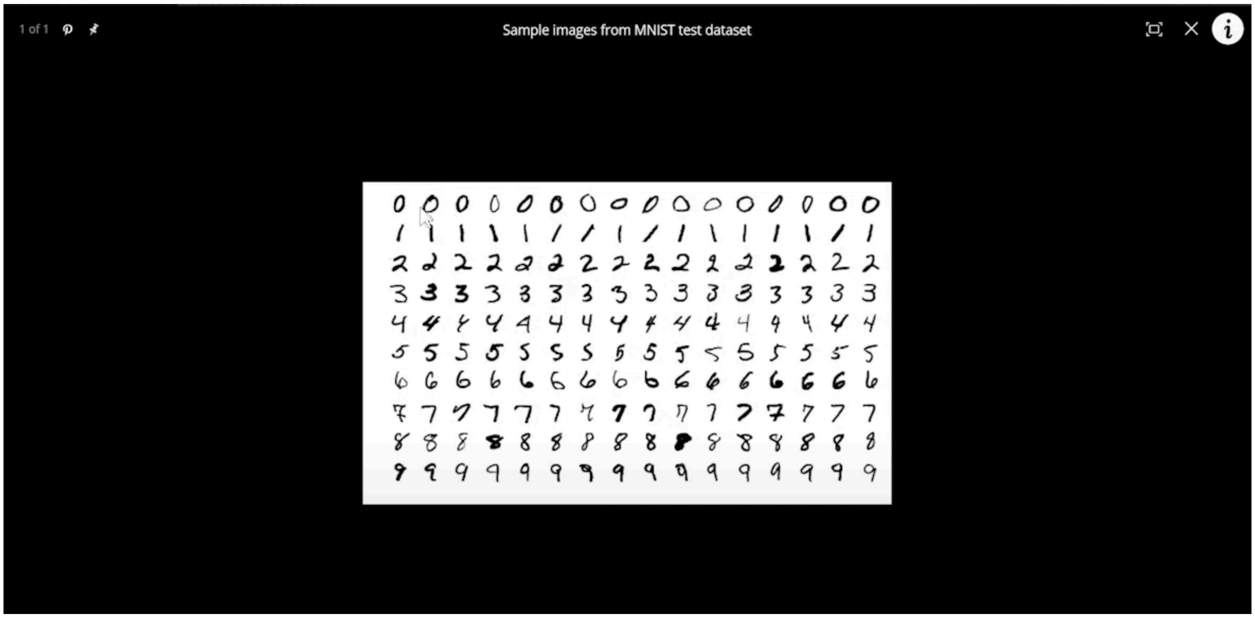
Using only:-

1. NUMPY
2. LINEAR ALGEBRA AND EQUATIONS

For this tutorial, we will have 3 layers (1 → Input Layer, 2 → Hidden Layer and 3 → Output Node/Layer)

PROBLEM STATEMENT

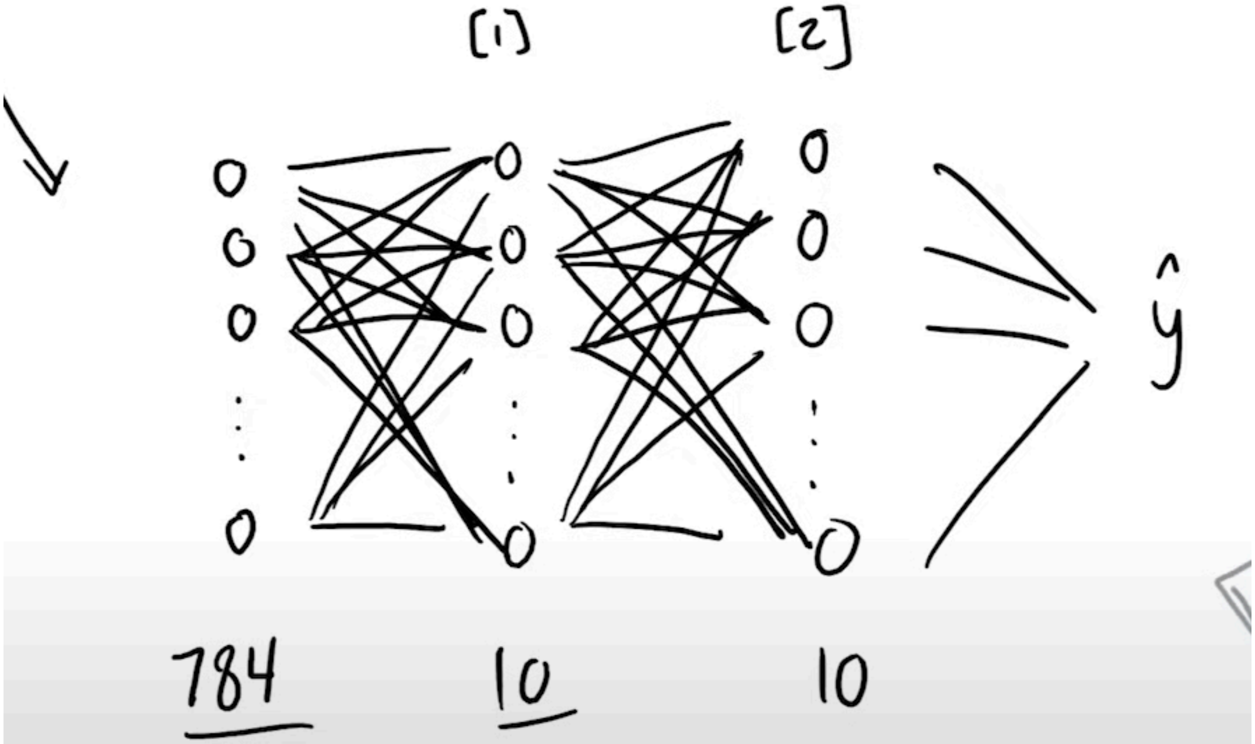
For the example, we are using images of hand written digits (0 to 9).



Neural Network which will classify any image of hand written Digit to the actual digit.

WORKING

Graph



💡

> Here the reason 784, 10,10 is there its because

Also the dimensions of the layers are as follows:
- Input layer: 784 features (pixels)
- Hidden layer: 10 neurons (can vary, chosen arbitrarily for simplicity)
- Output layer: 10 neurons (one for each digit)

Math

The training images are 28 × 28 i= 784 pixels (IN TOTAL)

Each pixel value has a range 0 to 255 (255 = Black and 0 = white)…

In laymen term, we can convert the each image into 25 rows and 25 columns. Each data having a value from 0 to 255.

Total dataset = m

Total Pixels = 784m (NOT IMP IG... Just for info)

HENCE

$$\chi = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix}^{\top} = \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix}$$

Each x is an image, and hence we have m rows...

Each row will have 784 values... Correct??? Or we can say each row will have 784 columns

And after we transpose it, IT BECOMES VICE VERSA...

Remember the first layer will be the INPUT LAYER or 0th layer and will have 784 nodes (for each pixel).

STEP 1 → FOR (Forward Propagation) PROPOGATION

BASIC PRINCIPLE:-

💡

> **Neural Network Node Principle**
>
> Each neuron's output is computed by taking a **weighted sum** of its inputs, adding a **bias**, and passing the result through an **activation function**:

$$a = g(w\top x + b)a = g(w^{\top}x + b)a = g(w\top x + b)$$

💡 **Why do we use Activation Function?**

We add them because if there is no activation function, the value from the output layer will be a sum of linear expression of all the hidden nodes.

Eg:- input layer. the first Layer will be a linear expression of input layer. Second Hidden layer will be another linear expression of the prev one (which is already a linear expression of the input layer) and there is no point of having hidden layer. And the output is just a Linear expression rather than being a complex algo.

FOR LAYER 0 TO 1:-

To calculate the value at each layer we use these functions:-

$$A^{(0)} = X \quad (784 \times m)$$

$$Z^{(1)} = W^{(1)}A^{(0)} + b^{(1)} \qquad \begin{array}{l} W^{(1)} : 10 \times 784 \\ A^{(0)} : 784 \times m \\ b^{(1)} : 10 \times 1 \Rightarrow 10 \times m \end{array}$$

$$A^{(1)} = g(Z^{(1)}) = \text{ReLU}(Z^{(1)})$$

There are many Types of Activation Function which will be covered in another PAGE

In our case, we will be using ReLu Activation Function

```
y ↑
 |
 |    /
 |   /
 |  /
 | /
--+----------------→ x
```

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

FOR LAYER 1 TO 2:-

The process for this layer is same as the above one but the KEY Difference is ACTIVATION FUNCTION.

Rather than Using the ReLu Activation Function, we are using Softmax Activation Function

$$
Z^{(2)} = W^{(2)}A^{(1)} + b^{(2)} \qquad \begin{aligned} W^{(2)} &: 10 \times 10 \\ A^{(1)} &: 10 \times m \\ b^{(2)} &: 10 \times 1 \Rightarrow 10 \times m \end{aligned}
$$

$$
A^{(2)} = \text{softmax}(Z^{(2)})
$$

SOFTMAX EQUATION

$$
\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}
$$

So it will look like this

$$\textbf{Input vector:} \quad \mathbf{z} = \begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix}$$

$$\textbf{Softmax formula:} \quad \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\textbf{Exponentials:} \quad \begin{aligned} e^{1.3} &\approx 3.669 \\ e^{5.1} &\approx 164.022 \\ e^{2.2} &\approx 9.025 \\ e^{0.7} &\approx 2.014 \\ e^{1.1} &\approx 3.004 \end{aligned}$$

$$\textbf{Denominator (sum):} \quad \sum_{j=1}^{5} e^{z_j} = 3.669 + 164.022 + 9.025 +$$

$$2.014 + 3.004$$

$$= 181.734$$

$$\textbf{Softmax outputs:} \quad \begin{aligned} \text{softmax}(z_1) &= \frac{3.669}{181.734} \approx 0.02 \\ \text{softmax}(z_2) &= \frac{164.022}{181.734} \approx 0.90 \\ \text{softmax}(z_3) &= \frac{9.025}{181.734} \approx 0.05 \\ \text{softmax}(z_4) &= \frac{2.014}{181.734} \approx 0.01 \\ \text{softmax}(z_5) &= \frac{3.004}{181.734} \approx 0.02 \end{aligned}$$

$$\textbf{Final probabilities:} \quad \text{softmax}(\mathbf{z}) = \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

💡 WHY USE SOFTMAX???

The Reason is simple that each node's value will lie in the range of 0 to 1.

Here 1 is Absolute certainty and 0 is the opposite.

STEP 2 → BACK (Backward Propagation) PROPOGATION

BASIC PRINCIPLE:-

💡

> ### Principle
>
> As the name suggests, backward propagation works from the last layer to the first, computing gradients and adjusting the weights and biases accordingly. This allows the network to learn which parameters contribute most to the error. To optimize the weights and biases, this step is repeated iteratively rather than performed just once. By running multiple iterations, the network progressively adjusts its parameters to minimize the loss and improve its predictions.

In Laymen Term's we can say:-

We Start with the prediction and we find out how much the prediction deviated with the actual input. This will give us the **ERROR**.

Then we find how much each of the **Bias and Weights** contributed to that error. Then, we adjust them accordingly.

FOR LAYER 2 TO 1:-

To calculate the value at each layer we use these functions:-

$$dz^{[2]} = A^{[2]} - Y \tag{1}$$
$$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T} \tag{2}$$
$$db^{[2]} = \frac{1}{m} \sum dz^{[2]} \tag{3}$$

Here dz² is the error of the 2² Layer

FOR LAYER 1 TO 0:-

To calculate the value at each layer we use these functions:-

$$dz^{[1]} = W^{[2]T} \cdot dz^{[2]} \circ g'(z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dz^{[1]} \cdot X^T$$
$$db^{[2]} = \frac{1}{m} \sum dz^{[1]}$$

Here dz² is the error of the 2² Layer

g′(z) is the Derivative of the Activation Function

STEP 3 → Updating the Values

In this step, we update 2 values:-

1. Weights
2. Biases

$$
\begin{aligned}
W^{[1]} &:= W^{[1]} - \alpha\, dW^{[1]} \\
b^{[1]} &:= b^{[1]} - \alpha\, db^{[1]} \\
W^{[2]} &:= W^{[2]} - \alpha\, dW^{[2]} \\
b^{[2]} &:= b^{[2]} - \alpha\, db^{[2]}
\end{aligned}
$$

💡 **α** is the the hyper-paramater. It is not trained by the Model. When you runt eht eabove cycle aka Gradient Descent, is set by us.

CODE

Below is the link to the code and in the comments are the notes:-

https://github.com/AviralTanwar/Nerual_Network_with_Numpy

OUTPUT

Since I have written this in python file (i.e. .py file) and not in jupiter or .ipnyb file, the code ran on terminal and many of the output was lost.

The main and important outputs are being shown below:-

```
Iteration: 1998
-------------------------------------------------------------------------------------------------------------------------------------------
-------------
Iteration: 1999
FINAL VA:UES OF THE WIEGHTS AND BIASES
W1: [[ 0.98411848 -1.29751216  0.28987067 ...  0.24964724  0.79800207
   0.80334867]
 [-0.4841924   1.10207873  0.65232149 ... -0.13034877  0.7621568
  -1.46672642]
 [-1.63936524 -1.11220145 -1.0146169  ... -0.43665716  0.22525685
  -0.39218341]
 ...
 [-1.14261946  2.81762964  0.70675579 ...  0.91739395  0.98839215
  -0.89004227]
 [ 1.9930897  -0.54271756  0.08513634 ...  0.58301527 -0.81838441
  -0.73108455]
 [ 0.55219396  1.74089319 -0.60519861 ...  0.03808267 -0.9331412
   1.24228231]]
b1: [[-0.04755573]
 [ 1.70135233]
 [-0.43539611]
 [ 0.94729015]
 [ 0.5198886 ]
 [ 1.60238648]
 [-0.88289931]
 [-1.18815634]
 [ 0.8980137 ]
 [-0.88235011]]
```

```
W2: [[-0.23175169  1.14114939 -1.00732695  0.04133117  0.73883826 -0.48001537
   0.39659933 -0.03163318 -0.00712701 -1.18102818]
 [-0.09602214 -0.35500419  0.54141036 -0.05828282 -0.06987261 -0.52879247
   0.27734165  0.4865551   0.41932041  0.1889699 ]
 [ 0.14770422 -0.31327452  0.28161426  0.23302932  0.43187098 -0.34471239
  -0.71052194  0.78744086  0.22631045  0.08882853]
 [-0.13381231 -0.52021448  0.28067533  0.14746433  0.37300311 -0.27116849
  -0.01092294  0.74995439  0.10791184  1.33202398]
 [-0.0437759  -1.68103466  0.1862439   0.66870048  0.53905865 -0.17484516
  -1.32827969  1.12289824  0.37693665 -2.62747926]
 [-0.25295205  0.26312881  0.29465887  0.2619702   0.59717791 -0.16665529
   0.69849649 -0.19232367  0.21615599  0.28108815]
 [ 0.04620833  0.69686339  0.16492401  0.32750921  0.51993676 -0.39739649
   0.2451636   0.49931428  0.70404646 -0.95237423]
 [-0.1852929   0.05381727  0.46139306  0.81658779  0.3762854  -0.88812929
  -0.22803042  1.10201045  0.21774813  1.6513143 ]
 [-0.04265643  0.47295783  0.40265768  0.20762916  0.39963678 -0.12010362
  -0.06032292  0.27629114  0.04924559  0.48360761]
 [-0.10267804  0.03179111  0.35559466  0.78057489  0.34639001 -0.34101352
  -0.45390375  1.13716717  0.36303738  0.4881365 ]]
b2: [[ 0.94043284]
 [ 1.42841243]
 [ 0.25842898]
 [ 2.26520397]
 [-2.48785916]
 [ 0.60413411]
 [-0.41416557]
 [ 0.16786511]
 [-0.30859612]
 [-0.1697713 ]]
Predicted: [5]
Actual: 5
Predicted: [2]
Actual: 2
Predicted: [8]
Actual: 9
Predicted: [6]
```

```
Actual: 6
Predictions: [2 0 6 7 7 9 5 8 9 9 7 9 7 9 4 2 2 1 7 3 7 5 0 7 9 4 9 9 5 7 7 9 3 9 9 6 7
 8 8 1 0 8 8 9 1 5 0 0 4 7 2 5 7 2 5 6 3 1 4 5 6 8 9 1 2 9 3 9 3 7 0 8 1 5
 5 0 3 9 6 5 1 9 0 6 7 0 1 5 7 6 1 2 3 2 1 7 3 3 6 7 3 9 4 3 2 0 0 1 8 3 8
 9 1 0 3 9 2 1 0 8 1 6 9 6 2 9 3 8 4 4 1 7 7 1 9 0 5 3 2 5 8 9 8 6 8 8 9 7
 3 2 3 1 2 3 8 1 7 8 1 2 7 2 9 2 1 9 1 7 0 6 9 1 5 0 0 0 6 0 7 1 4 2 1 9 9
 0 7 0 0 9 2 5 7 9 1 3 9 2 2 8 7 2 6 7 7 3 1 2 9 3 9 2 1 9 8 0 0 1 5 1 0 9
 7 6 0 9 9 6 9 1 1 8 0 3 0 5 7 2 9 2 0 9 3 0 7 0 1 1 9 3 1 4 7 0 0 2 6 2 3
 4 1 2 2 2 7 9 9 4 5 1 9 7 9 6 9 0 7 3 9 8 5 1 1 9 6 0 6 2 5 1 0 7 9 8 9 6
 9 3 9 4 5 3 8 7 5 6 0 2 1 3 9 0 9 2 9 8 2 2 9 9 4 2 2 0 2 7 0 9 3 4 8 9 4
 9 7 0 5 9 7 7 2 6 0 3 5 9 2 9 5 1 0 2 9 4 1 0 9 6 8 2 9 4 7 0 7 7 2 6 3 2
 1 2 7 9 9 9 5 3 9 6 8 3 2 5 1 6 2 8 6 9 6 0 2 5 0 8 7 1 1 7 0 3 7 1 9 3 7
 5 7 3 9 5 5 2 8 8 6 2 1 8 9 9 3 8 1 4 5 1 1 0 5 9 9 7 9 2 8 7 5 9 7 1 9 9
 6 6 1 8 7 3 6 0 2 3 7 6 2 6 5 9 8 5 0 5 1 1 1 0 5 5 8 9 1 6 8 0 2 0 1 6 3
 1 5 3 1 8 9 9 3 4 9 9 1 5 4 4 9 5 0 0 2 1 0 1 7 2 5 0 5 0 6 1 9 2 3 0 2 4
 0 0 8 8 0 7 9 6 6 7 2 6 9 9 7 8 3 2 2 2 0 6 2 5 5 1 2 5 2 0 3 0 3 9 3 9 7
 7 0 6 2 0 0 9 0 8 9 7 8 0 6 6 1 7 8 1 5 5 9 3 8 6 7 6 0 9 9 5 3 9 1 6 9 4
 8 7 2 0 8 6 3 0 6 9 7 2 2 1 7 9 7 7 0 7 7 0 3 6 7 1 0 3 9 9 8 9 3 1 6 5 4
 0 9 1 0 9 8 3 8 2 0 7 8 8 8 2 9 1 1 9 7 1 1 0 5 9 8 1 7 2 3 2 0 8 9 1 9 1
 5 9 9 1 5 6 7 6 5 7 7 8 6 0 7 8 2 7 4 5 7 3 9 3 5 0 5 2 9 7 9 7 2 0 9 5 7
 5 9 7 5 1 2 6 5 2 1 7 9 1 6 1 6 8 5 9 5 9 9 0 1 2 9 9 0 1 1 9 6 4 5 9 4 9
 7 6 9 9 5 4 9 9 6 8 0 9 7 1 3 1 6 8 7 3 7 9 2 7 7 8 5 1 6 2 4 8 6 9 1 7 9
 8 6 3 7 7 5 7 2 4 9 0 2 3 1 7 8 9 2 7 0 0 2 3 9 0 5 1 9 6 2 7 2 0 2 0 7 9
 3 7 9 3 0 1 3 8 5 4 3 6 1 8 5 0 1 3 7 0 5 1 8 9 9 9 7 5 6 2 0 0 6 0 6 3 2
 2 4 5 0 5 1 9 1 3 0 9 2 0 8 8 6 3 7 6 4 4 1 5 2 9 0 5 6 5 0 6 4 8 8 9 4 1
 5 0 3 8 7 2 0 0 6 0 5 7 3 2 2 1 6 6 1 7 5 8 3 7 4 9 0 5 7 3 2 2 5 4 9 5 9
 0 2 5 9 1 3 0 1 4 5 3 0 8 9 2 1 3 0 6 7 9 2 2 6 2 9 5 4 9 5 4 0 9 9 0 3 5
 1 0 9 2 9 1 9 6 8 1 3 5 6 9 2 3 7 9 2 1 9 2 6 5 5 9 1 3 7 7 5 1 7 9 0 7 7
 6]
```

```
Y: [2 0 6 7 2 9 5 5 4 4 7 9 7 4 4 2 2 1 7 3 7 4 5 7 4 4 9 9 8 7 7 8 5 7 5 6 7
 8 8 1 0 3 5 9 7 5 0 0 4 7 2 5 7 2 3 6 3 1 4 9 6 3 4 1 2 9 5 9 3 7 0 1 1 4
 5 0 5 7 6 3 3 9 0 6 7 0 2 4 7 6 7 3 3 2 1 7 3 5 4 7 3 9 4 8 2 0 0 3 8 3 8
 9 1 0 3 4 3 1 0 8 9 6 4 6 2 7 3 8 4 4 1 7 7 1 9 0 9 5 2 5 3 6 3 6 8 8 4 5
 8 2 8 1 2 3 8 1 7 3 1 6 7 2 9 4 1 9 8 7 0 2 9 1 4 5 0 0 6 0 7 1 9 1 1 4 4
 0 7 0 3 9 2 5 8 9 2 5 4 2 2 8 7 6 6 7 7 3 1 2 9 3 7 2 1 9 5 0 0 1 8 1 0 9
 7 6 0 7 2 2 4 1 1 8 0 3 5 5 7 2 4 4 0 9 8 0 9 0 1 1 9 5 1 4 7 6 0 2 2 2 3
 9 5 8 6 2 7 4 9 4 4 1 9 7 9 6 4 0 7 3 4 8 8 1 1 4 6 6 6 2 9 3 0 9 7 3 9 4
 4 3 4 4 6 5 8 7 5 6 0 2 1 3 8 6 9 9 9 3 2 2 9 7 4 2 2 0 2 7 0 9 3 9 8 9 4
 9 7 0 3 8 7 7 2 3 0 5 5 4 2 9 3 3 0 8 9 4 1 6 4 6 8 2 4 4 7 0 7 3 2 7 1 2
 1 2 7 9 1 9 5 8 4 6 8 8 6 3 1 6 8 3 6 9 6 0 2 5 0 8 7 1 1 7 0 2 7 9 4 2 7
 5 3 3 9 5 5 2 2 3 6 4 1 8 4 4 3 8 1 4 9 3 1 0 5 8 9 7 9 2 5 7 5 9 7 1 9 9
 6 4 1 8 7 5 6 0 3 3 7 6 2 6 5 9 8 8 0 0 1 1 1 0 5 5 8 9 1 5 8 0 1 2 5 6 3
 8 3 3 3 8 4 8 8 4 9 9 1 5 4 4 7 0 0 0 2 1 0 1 7 2 8 0 5 0 6 1 9 2 3 0 4 4
 0 0 8 5 0 7 4 6 6 7 2 6 1 7 7 5 8 2 2 6 0 6 2 6 5 1 6 5 2 0 1 0 5 4 5 4 7
 7 0 6 2 0 0 7 0 3 9 7 8 0 6 1 1 5 8 1 5 5 9 6 7 7 7 9 0 9 9 6 3 9 1 6 4 4
 9 9 7 0 8 6 8 0 0 4 7 9 2 1 7 2 7 7 0 7 1 0 6 0 7 1 0 3 9 9 3 9 3 1 6 5 4
 0 9 1 5 4 8 3 2 2 0 9 5 8 8 3 8 1 2 4 7 1 1 0 5 4 8 1 7 2 3 3 0 8 9 1 9 1
 5 9 2 1 8 6 3 6 8 7 9 3 6 6 7 5 2 7 4 5 7 3 7 5 5 0 9 2 9 7 9 7 3 0 9 5 7
 5 4 7 5 1 2 6 5 2 6 7 4 1 6 1 6 5 8 9 8 8 4 5 3 2 4 4 0 1 1 9 6 5 6 9 4 4
 7 6 2 4 5 5 4 4 6 8 0 3 7 1 3 1 6 8 3 8 7 4 2 7 7 8 5 1 6 2 4 8 6 9 1 7 4
 3 4 5 7 7 8 7 2 4 9 0 3 3 1 9 8 9 2 7 0 0 2 5 9 0 3 1 7 6 2 7 2 0 2 0 7 9
 3 7 4 3 0 1 3 8 5 4 3 2 1 5 5 0 1 8 7 0 5 1 8 9 9 9 7 5 6 2 0 0 6 0 6 2 0
 2 4 5 0 6 1 4 1 2 0 4 2 0 8 4 6 3 7 6 2 4 1 8 2 9 0 5 6 5 0 6 4 0 8 9 4 3
 5 0 3 8 0 2 0 0 2 0 5 7 3 2 2 7 6 2 1 7 5 8 4 7 4 9 0 5 7 8 2 2 8 4 4 5 9
 0 5 3 9 1 3 0 1 4 5 8 0 9 9 2 3 3 0 3 7 9 8 2 4 2 4 5 4 4 9 4 0 1 7 5 3 8
 1 0 9 2 4 1 9 6 3 1 3 8 6 4 2 8 7 9 2 5 9 8 6 3 3 9 1 0 7 7 5 1 7 9 0 7 7
 6]
```

```
Y: [2 0 6 7 2 9 5 5 4 4 7 9 7 4 4 2 2 1 7 3 7 4 5 7 4 4 9 9 8 7 7 8 5 7 5 6 7
 8 8 1 0 3 5 9 7 5 0 0 4 7 2 5 7 2 3 6 3 1 4 9 6 3 4 1 2 9 5 9 3 7 0 1 1 4
 5 0 5 7 6 3 3 9 0 6 7 0 2 4 7 6 7 3 3 2 1 7 3 5 4 7 3 9 4 8 2 0 0 3 8 3 8
 9 1 0 3 4 3 1 0 8 9 6 4 6 2 7 3 8 4 4 1 7 7 1 9 0 9 5 2 5 3 6 3 6 8 8 4 5
 8 2 8 1 2 3 8 1 7 3 1 6 7 2 9 4 1 9 8 7 0 2 9 1 4 5 0 0 6 0 7 1 9 1 1 4 4
 0 7 0 3 9 2 5 8 9 2 5 4 2 2 8 7 6 6 7 7 3 1 2 9 3 7 2 1 9 5 0 0 1 8 1 0 9
 7 6 0 7 2 2 4 1 1 8 0 3 5 5 7 2 4 4 0 9 8 0 9 0 1 1 9 5 1 4 7 6 0 2 2 2 3
 9 5 8 6 2 7 4 9 4 4 1 9 7 9 6 4 0 7 3 4 8 8 1 1 4 6 6 6 2 9 3 0 9 7 3 9 4
 4 3 4 4 6 5 8 7 5 6 0 2 1 3 8 6 9 9 9 3 2 2 9 7 4 2 2 0 2 7 0 9 3 9 8 9 4
 9 7 0 3 8 7 7 2 3 0 5 5 4 2 9 3 3 0 8 9 4 1 6 4 6 8 2 4 4 7 0 7 3 2 7 1 2
 1 2 7 9 1 9 5 8 4 6 8 8 6 3 1 6 8 3 6 9 6 0 2 5 0 8 7 1 1 7 0 2 7 9 4 2 7
 5 3 3 9 5 5 2 2 3 6 4 1 8 4 4 3 8 1 4 9 3 1 0 5 8 9 7 9 2 5 7 5 9 7 1 9 9
 6 4 1 8 7 5 6 0 3 3 7 6 2 6 5 9 8 8 0 0 1 1 1 0 5 5 8 9 1 5 8 0 1 2 5 6 3
 8 3 3 3 8 4 8 8 4 9 9 1 5 4 4 7 0 0 0 2 1 0 1 7 2 8 0 5 0 6 1 9 2 3 0 4 4
 0 0 8 5 0 7 4 6 6 7 2 6 1 7 7 5 8 2 2 6 0 6 2 6 5 1 6 5 2 0 1 0 5 4 5 4 7
 7 0 6 2 0 0 7 0 3 9 7 8 0 6 1 1 5 8 1 5 5 9 6 7 7 7 9 0 9 9 6 3 9 1 6 4 4
 9 9 7 0 8 6 8 0 0 4 7 9 2 1 7 2 7 7 0 7 1 0 6 0 7 1 0 3 9 9 3 9 3 1 6 5 4
 0 9 1 5 4 8 3 2 2 0 9 5 8 8 3 8 1 2 4 7 1 1 0 5 4 8 1 7 2 3 3 0 8 9 1 9 1
 5 9 2 1 8 6 3 6 8 7 9 3 6 6 7 5 2 7 4 5 7 3 7 5 5 0 9 2 9 7 9 7 3 0 9 5 7
 5 4 7 5 1 2 6 5 2 6 7 4 1 6 1 6 5 8 9 8 8 4 5 3 2 4 4 0 1 1 9 6 5 6 9 4 4
 7 6 2 4 5 5 4 4 6 8 0 3 7 1 3 1 6 8 3 8 7 4 2 7 7 8 5 1 6 2 4 8 6 9 1 7 4
 3 4 5 7 7 8 7 2 4 9 0 3 3 1 9 8 9 2 7 0 0 2 5 9 0 3 1 7 6 2 7 2 0 2 0 7 9
 3 7 4 3 0 1 3 8 5 4 3 2 1 5 5 0 1 8 7 0 5 1 8 9 9 9 7 5 6 2 0 0 6 0 6 2 0
 2 4 5 0 6 1 4 1 2 0 4 2 0 8 4 6 3 7 6 2 4 1 8 2 9 0 5 6 5 0 6 4 0 8 9 4 3
 5 0 3 8 0 2 0 0 2 0 5 7 3 2 2 7 6 2 1 7 5 8 4 7 4 9 0 5 7 8 2 2 8 4 4 5 9
 0 5 3 9 1 3 0 1 4 5 8 0 9 9 2 3 3 0 3 7 9 8 2 4 2 4 5 4 4 9 4 0 1 7 5 3 8
 1 0 9 2 4 1 9 6 3 1 3 8 6 4 2 8 7 9 2 5 9 8 6 3 3 9 1 0 7 7 5 1 7 9 0 7 7
 6]
Development Set Accuracy: 0.696
```

ONLINE REFERENCES

You tube

https://www.youtube.com/watch?v=w8yWXqWQYmU&ab_channel=SamsonZhang

CODE on KAGGLE

https://www.kaggle.com/code/wwsalmon/simple-mnist-nn-from-scratch-numpy-no-tf-keras

DATASET

https://www.kaggle.com/datasets/ompanpatil12/simple-mnist-nn-from-scratch-numpy-no-tfkeras

Equations and Images

Used ChatGPT... LOL