Neural Network from Scratch

- Project involves only Numpy & Math.
- The project will detect handwritten digits.
- Training Images :

784 pixels

28×28

pixel value 0 to 255, where 0 is black & 255 is white

- 10 classes :- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
(As there are 10 digits to numbers as our dataset)

- We can represent a matrix of the 'n' images in the dataset as;

$$X = \begin{bmatrix} — & x^{(1)} & — \\ — & x^{(2)} & — \\ & \vdots & \\ — & x^{(n)} & — \end{bmatrix} = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & (x)^{n} \\ | & | & & | \end{bmatrix}$$

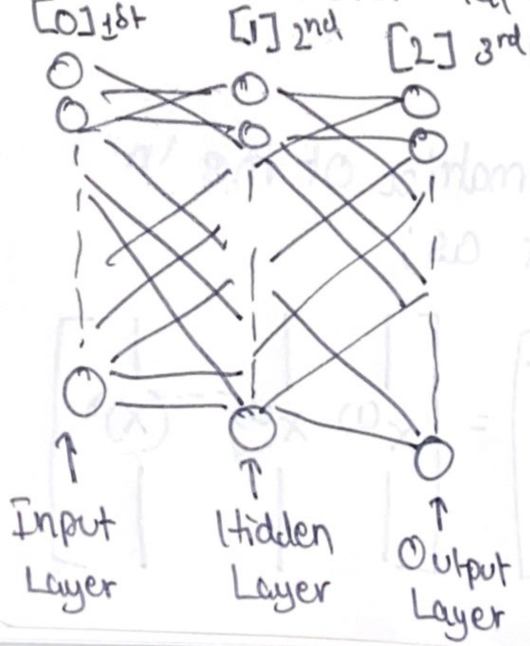
- Each row is going to be 784 pixels in total colums. Each element corresponding to a pixel in the image.

** So, rows represent i.e one row represent the values of one image distributed into 784 columns. (784 pixels in total for 1 image)

- Transpose the matrix, where ~~each row~~ instead of each row representing an image, each column will represent the image.

** So, one ~~row~~ column gives values of one image distributed into 784 rows & n columns.

→ Neural Network that we're gonna build:



→ 3- Part Training Method:

First Part: Forward Propagation

- $A^{[0]} = X \; (784 \times n)$

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

$10 \times n \quad 10 \times 784 \quad 784 \times n \quad 10 \times 1 \Rightarrow 10 \times n$

$A^{[0]} \Rightarrow$ First Layer (0^{th} input Layer)

$X \Rightarrow$ inputs

$n \Rightarrow$ total inputs/nodes

$Z^{[1]} \Rightarrow$ Unactivated First Layer (1^{st} Layer)

$W^{[1]} \Rightarrow$ weights

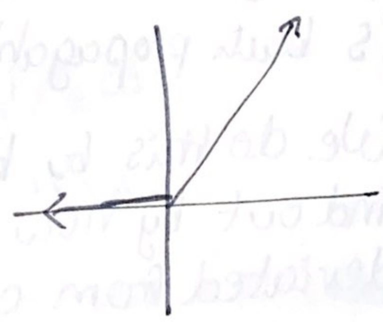
$b^{[1]} \Rightarrow$ bias term

- If you only have a Linear combination (weights & biases), you can never get some randomness or a different type of function. So, we apply an Activation function.

- $A^{[1]} = g(Z^{[1]}) = ReLU(Z^{[1]})$

Applying activation function gives us non-linear combinations.

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



Rectified Linear Unit.

- $A^{[1]} = g(Z^{[1]}) = ReLU(Z^{[1]})$

First-Layer eqn. which is activated.

- $\underset{10\times n}{Z^{[2]}} = \underset{10\times 10}{W^{[2]}} \underset{10\times n}{A^{[1]}} + \underset{10\times 1 \Rightarrow 10\times n}{b^{[2]}}$

Second-Layer Eqn. (Unactivated) $= Z^{[2]}$

$A^{[2]} = \text{Softmax}(Z^{[2]})$

- Activated-Second Layer eqn. $= A^{[2]}$

Output Layer — Softmax Activation function — Probabilities

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \Longrightarrow \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \Rightarrow \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Second Part: Backwards Propagation

- To optimize the weights & biases in forward propagation, we run an algorithm which is back propagation.
- We do this by, beginning our prediction & find out by how much the prediction deviated from actual label/value.

. This gives us an error, which shows us that how much the weights & biases in the model contributed to the error.

$$dZ^{[2]} = A^{[2]} - Y$$

$dZ^{[2]}$: $10 \times n$, $A^{[2]}$: $10 \times n$, Y: $10 \times n$

$dZ^{[2]}$ = Error of the Second Layer

$A^{[2]}$ = Predictions (Output)

Y = One-hot encode

Note:- One-hot encoding is a technique we use to represent categorical variables as numerical values.

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$ [loss function w.r.t weights]

$dW^{[2]}$: 10×10, $dZ^{[2]}$: $10 \times n$, $A^{[1]T}$: $n \times 10$

$$db^{[2]} = \frac{1}{m} \sum dZ^{[2]}$$ [Avg. of absolute error]

$db^{[2]}$: 10×1, $\sum dZ^{[2]}$: 10×1

Now, similarly for first layer

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} .* g'(Z^{[1]})$$

$dZ^{[1]}$: $10 \times n$, $W^{[2]T}$: 10×10, $dZ^{[2]}$: $10 \times n$, $g'(Z^{[1]})$: $10 \times n$

Here $W^{[2]T}$ means transpose of weights of 2nd Layer.

$$dW^{[1]} = \frac{1}{n} \underset{10\times n}{dZ^{[1]}} \underset{n\times 784}{X^T}$$

$$\underset{10\times 1}{db^{[1]}} = \frac{1}{m} \underset{10\times 1}{\Sigma dZ^{[1]}}$$

Third Part : Update all Parameters

$$W^{[1]} : W^{[1]} - \alpha dW^{[1]}$$

α = Learning rate

$$b^{[1]} : b^{[1]} - \alpha db^{[1]}$$

$$W^{[2]} : W^{[2]} - \alpha dW^{[2]}$$

$$b^{[2]} : b^{[2]} - \alpha db^{[2]}$$

→ Repeat all Processes for training the model.

- Dhairya Patel