Building a neutral network from scratch.

only using numpy and math.

problem: Digit classification using MNIST Dataset.

↓

Contains (28 x 28) gnascale images.

(math)

image (784) → having intensities (0 - 255)

black ✓ white.

(28 x 28) → m training samples.

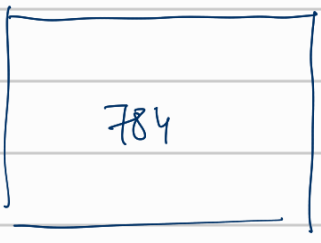
$$X = \begin{bmatrix} \text{---} & x^{(1)} & \text{---} \\ \text{---} & x^{(2)} & \text{---} \\ & \vdots & \\ \text{---} & x^{(m)} & \text{---} \end{bmatrix}^T = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$(m \times 784)$ → (Transpose) → $(784 \times m)$

↓ per image 28 x 28 intensities value

↓ 784 rows. Each row contains 784 intensities of x^i image. $(i - m)$

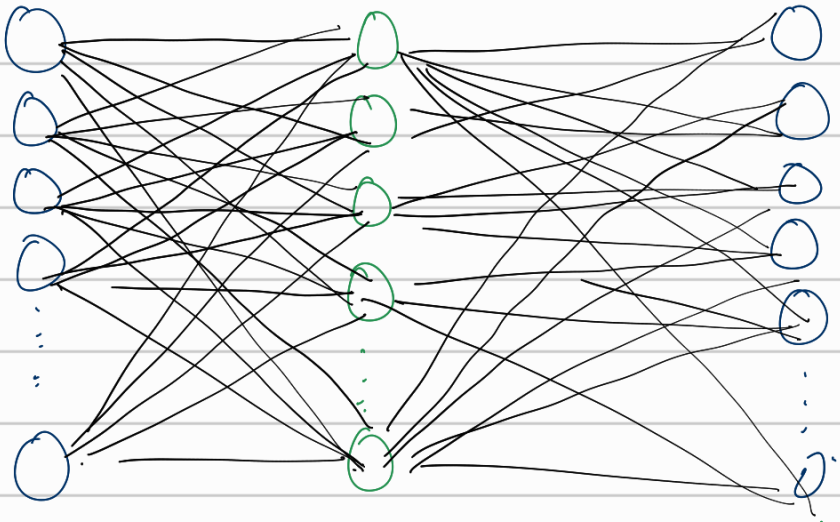
We have to classify.



$\Rightarrow (0, 1, 2, 3 \ldots 9)$

10 classes.

a neural net would be built like this (simple 2 layer)



784 (each intiensitie) will map to (0-9) classes across. nets.

10 (0-9) Hidden layer

10 (0-9) Output layer.

input layer

→ First part of training this network.

↓

forward propagation

- input neurons ka data hidden layer tak jata hai
- Hidden Layer us input ko weights aur activation function ke through process karti hai
- fin result o/p Layer tak jata hai

input → hidden layer → Activation → o/p layer → Final output.

Start

$A^{[0]} = X \quad [784 \times m]$

[input layer]

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

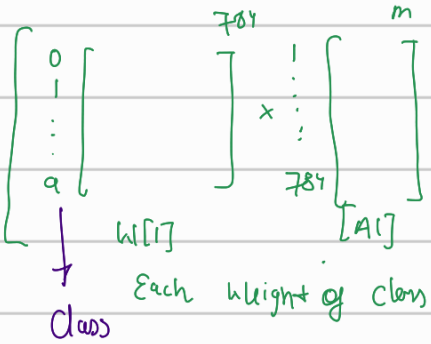
$[10 \times m] \quad [10 \times 784] \quad [784 \times m] \quad [10 \times 1]$

unactivated first layer

to avoid linearity and increase natural selection we apply activation function [tanh, Sigmoid, RELU, Softmax, etc)

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

activated hidden layer

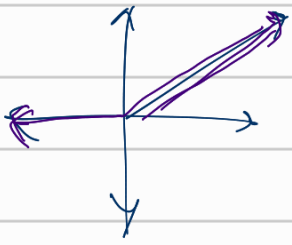


Each weight of class

Class

W[1] → (0-9) K weight image K 784 intensities ka value pe jaake multiply hoga fir soft max use karke hume max pool value uthake predict karenge [ek can have m images]

What is RELU? Now 😐?



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

$$Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$$

$(10 \times m) \quad (10 \times 10) \quad (10 \times m) \quad (10 \times 1) \Rightarrow 10 \times m$

$10 \times m \rightarrow m_1 \; m_2 \; m_3$

↓

unactivated Layer & i.e output Layer.

$A^{[2]} = \underline{\text{Softmax}}\,[Z^{(2)}] \longrightarrow$ Softmax $\dfrac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

We will get probabilities.

Ex:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \frac{e^1}{e^1 + e^2 + e^3}, \cdots$$

↓

(1)

Similary for 2, 3 = $\begin{bmatrix} - \\ - \\ - \end{bmatrix}$

This was forward propagation we took an image pass to the network and made a prediction Simple.

→ Now we need good weight and bias to improve our prediction So we will propagate backwards.

Backwards Propagation

↓

predictions from forward Lenge and backwards chalenge to find ennon and we will minimize this weights by adjusting weights. and bias optimal.

$dz^{[2]} \longrightarrow A^{[2]} - Y$

$dz^{[2]}$: $10 \times m$ — Error of Second Error. How much o/p Layer off by.

$A^{[2]}$: $10 \times m$ — prediction label

Y: $10 \times m$ — actual label. Suppose $Y = 2$ then one hot encoding is

	$A^{[2]}$	Y
0	0.02	0
1	0.03	0
2	0.4	1
⋮		0
9		0

Now from $z^{[2]}$ we will find how much the weight W and bias b contributed to the error

$$\rightarrow dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$

$dW^{[2]}$: 10×10 — derivative of loss function w.r.t weight

$dz^{[2]}$: $10 \times m$

$A^{[1]T}$: $m \times 10$

W: weight

$$\rightarrow db^{[2]} = \frac{1}{m} \sum dz^{[2]}$$

$db^{[2]}$: 10×1

$dz^{[2]}$: 10×1

b: bias

$$\rightarrow dz^{[1]} = W^{[2]T} dz^{[2]} .* g'(z^{(1)})$$

$dz^{[1]}$: $10 \times m$

$W^{[2]T}$: 10×10

$dz^{[2]}$: $10 \times m$

$g'(z^{(1)})$: $10 \times m$ — derivative of activating function

$$dW^{[1]} = \frac{1}{m} dz^{1} X^{T}$$

$dW^{[1]}$: 10×784

$\frac{1}{m}$: $10 \times m$

dz^{1}: $m \times 784$

$$db^{[2]} = \frac{1}{m} dz^{[1]}$$

$db^{[2]}$: (10×1)

$dz^{[1]}$: (10×1)

Now after finding the error contributed to each layer we update the weight accordingly.

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

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

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

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

$\alpha \longrightarrow$ learning rate

↓

hyperparameter.

Now this is repeated [fp + BP] until a certain accuracy is Reached.

maths Ends here :-)