Neural Nets from Scratch

 ⇒ 'm' no. of such training images

image – 28 x 28 pixels = 784 pixels


28 x 28

⇒ 0, 1, 2 9

10 classes

each class is for each digit 0-9

↳ each of these 784 pixels is a value b/w 0 & 255 → Totally white

↓ totally black

so, if we make an array out of the pixels of each digit and map the value b/w 0 and 255.

we get an array X = $\begin{bmatrix} \text{---} & x^{(1)} & \text{---} \\ \text{---} & x^{(2)} & \text{---} \\ & \vdots & \\ \text{---} & x^{(m)} & \text{---} \end{bmatrix}$

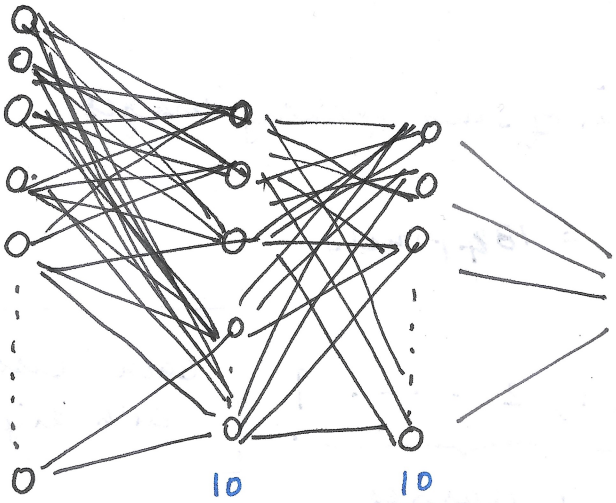
where, Each row = example image of digit

& 784 columns in one row = corresponding pixels of that one sample image.

If I were to transpose the matrix, each column is an example digit and rows correspond to pixels.

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

Structure of the Net



Setup of the Neural net -

Step 1 # - FORWARD PROPAGATION

$A^{[0]} = X \ (784 \times m)$ i.e. unactivated first layer which is input from dataset itself.

& $g^{[0]}$ = activation function - it serves to have multiple connections to a node in all layers.

(!) In the absence of an activation function, each layer's nodes are just a combination of previous nodes, and we get just Linear combinations.

In other words if, no multiple node connection to each layer $\Longrightarrow$ Fancy Linear regression ✓

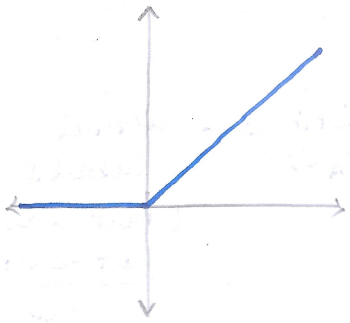
we add weights & a bias -

$$Z^{[1]} = W^{[1]} \cdot A^{[0]} + b^{[1]} \longrightarrow Z^{[1]} = \text{term } A^{[0]} \text{ with, weight} = W \text{ \& bias} = b$$

$Z^{[1]}$: $10 \times m$; $W^{[1]}$: 784×10 ; $A^{[0]}$: $784 \times m$; $b^{[1]}$: $10 \times m$

Activation function - any non-linear function such as tanh or sigmoid or ReLu

using ReLu(x), (rectified linear unit)



$$ReLu(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

so, my $A^{[1]}$ becomes the output get after I perform $ReLU(z^{[1]})$

$$\Rightarrow \boxed{A^{[1]} = ReLu(z^{[1]})}$$

This is for the "Hidden" middle layer

For the output layer however, we need each of the 10 nodes to correspond to each of the 10 digits that can be recognised

⇒ Activation function we use is → SOFTMAX(x)

$$\text{i.e. } SOFTMAX(x) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

SOFTMAX(x) makes the input vector 'z' of K real no.s and normalizes it into a probability distribution consisting of

['K' probabilities ∝ input of number's exponent]

⇓

we need each value of each of the output nodes to be b/w 0 & 1 where,

0 = no chance at all

1 = absolute certainity.

so,

$$\boxed{A^{[2]} = softmax(z^{[2]})}$$

Step 2 # – BACKWARD PROPAGATION

⇒ Essentialy we start with our prediction, find out how much our weights & biases offset us from the correct ones

⇒ Find and adjust weights & biases accordingly to minimize the loss function.

[Error of OUTPUT (2nd) Layer] $dZ^{[2]}_{10\times m} = A^{[2]}_{10\times m} - Y_{10\times m}$ → predictions ($A^{[2]}$) – actual labels (that are hot-encoded)

⇓

if $y = 4$ then 4th index of Y is 1 & all other elements are 0 (pos 4)

i.e. [0, 0, 0, 0, 1, 0, 0, 0 . . 0] (indices 0, 1, 2, 3, 4, m−1, m)

[derivative of the LOSS FUNCN w.r.t Weights] $dW^{[2]}_{10\times 10} = \frac{1}{m} dZ^{[2]}_{10\times m} \cdot A^{[1]T}_{m\times 10}$

Nothing but the Avg. Absolute Error in output layer

[derivative of LOSS FUNCN w.r.t biases] $db^{[2]}_{10\times 1} = \frac{1}{m} \sum dZ^{[2]}_{10\times 1}$

---

similarly, for the middle "hidden" layer we compute the errors and also account for the error of the output layer & apply the corresponding ~~errors~~ weights in reverse.

Intuition → error from OUTPUT (2nd) Layer ⇒ apply Weights to it in reverse ⇒ to get to the errors of the first (hidden) layer → multiply with derivative of the activation funcn

TO UNDO THE ACTIVATION

so,

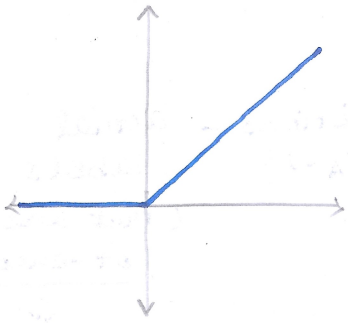
[Error of Output of HIDDEN 1st Layer] $dZ^{[1]}_{10\times m} = W^{[2]T}_{10\times 10}\, dZ^{[2]}_{10\times m} * g'(dZ^{[1]})_{10\times m}$

⇓ Transpose, to apply Weights to Errors in 2nd Layer to get to Errors in 1st Layer (HIDDEN)

⇓ derivative of the Activation funcn to undo the activation funcn to get proper error

Activation function - any non-linear function such as tanh or sigmoid or ReLu

using ReLu(x), (rectified linear unit)



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

so, my $A^{[1]}$ becomes the output get after I perform $\text{ReLU}(Z^{[1]})$

$$\Rightarrow \boxed{A^{[1]} = \text{ReLu}(Z^{[1]})}$$

This is for the "Hidden" middle layer

For the output layer however, we need each of the 10 nodes to correspond to each of the 10 digits that can be recognised

$\Rightarrow$ Activation function we use is $\longrightarrow$ SOFTMAX(x)

$$\text{i.e. } \text{SOFTMAX}(x) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

SOFTMAX(x) makes the input vector 'z' of K real no.s and normalizes it into a probability distribution consisting of

['k' probabilities $\propto$ input of number's exponent]

$\Downarrow$

we need each value of each of the output nodes to be b/w 0 & 1 where,

0 = no chance at all

1 = absolute certainity.

so,

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

How much did the weights & biases contribute to errors in the "Hidden" layer

$$dW^{[1]}_{10\times 784} = \frac{1}{m}\, dz^{[1]}_{10\times m} \cdot X^{[T]}_{m\times 784}$$

$$db^{[1]}_{10\times 1} = \frac{1}{m} \Sigma\, dz^{[1]}_{10\times 1}$$

Step 3 # UPDATE PARAMETERS

with 'α' (learning rate $\approx$ 0.01) — set by user during gradient decent run on the loss function

low value generally better

- once weights' and baises' losses are found, we can update parameters accordingly -

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

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

&

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

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