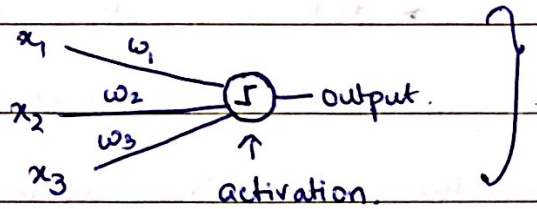
Date

Neural Networks

→ Neural Networks are based on the design of neurons in the human body

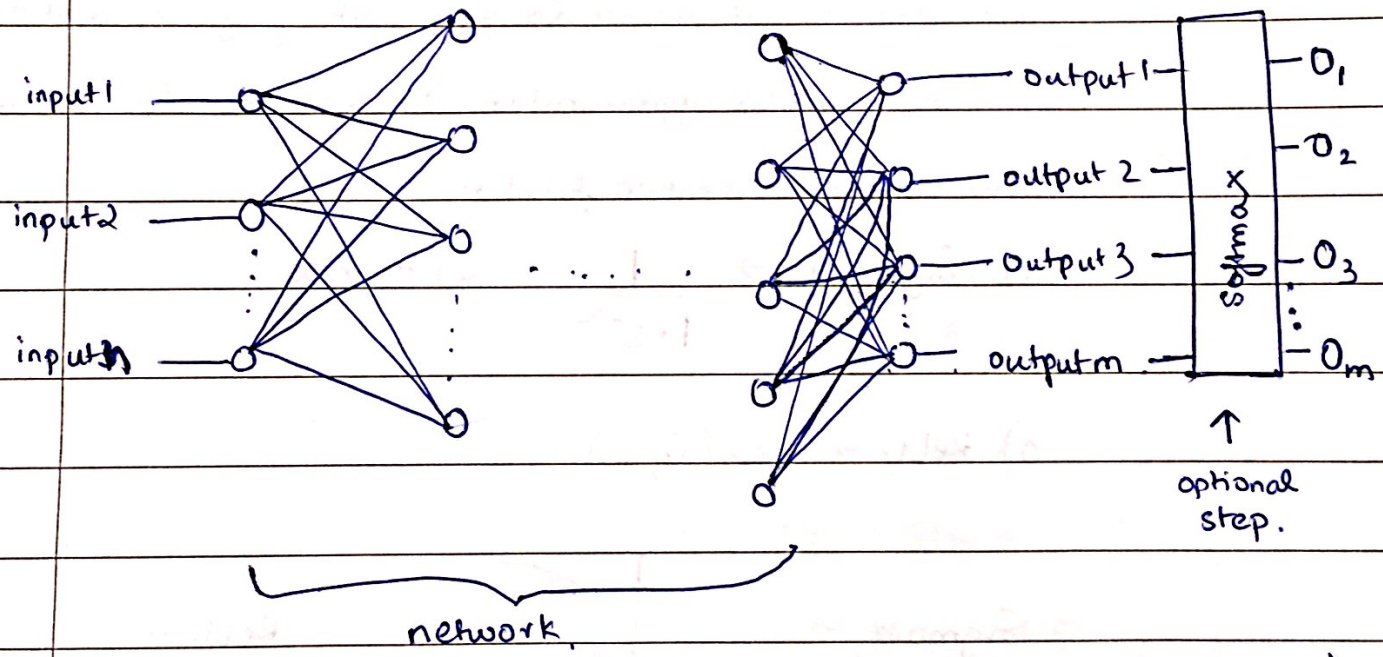
A single neuron/node/perceptron can be denoted as



Node.

~~output = (x·ω + b)~~

output = activation $(\vec{x} \cdot \vec{\omega} + b)$



So you can either stop before the softmax as your output (final result) or do a softmax on the network outputs & the result which comes out of it can be your final output.

Doing a softmax is encouraged since it squashes your outputs to [0,1] range

$$\text{Softmax output}_i = \frac{O_i}{\sum_{i=1}^{m} O_i}$$

→ Activation functions:-

Activation functions are added so that neural networks can approximate non-linear functions.

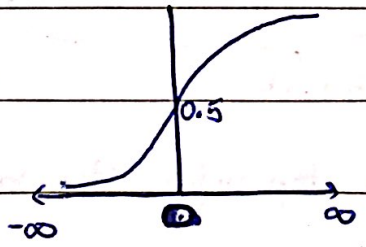
Let us say we never did the activation, then you can see that as the output progresses through the nodes, it will just be a linear sum of the inputs. Now since linear functions are restricted to just linear decision making you cannot expect it to be universal approximators. You have to add non-linearity to it.

Some popular activation functions:-

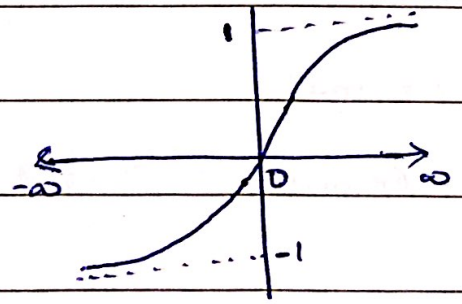
1) Sigmoid → $\frac{1}{1+e^{-x}}$ 2) $\tanh x$

3) ReLU → $\max(0, x)$

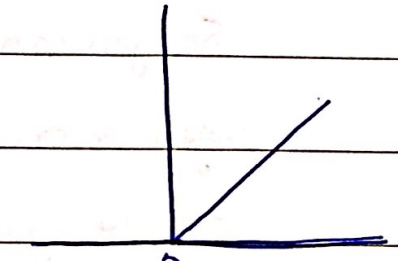
Sigmoid →



tanh →



ReLU →



for
(zero ~~before~~ x≤0 & x after x>0)

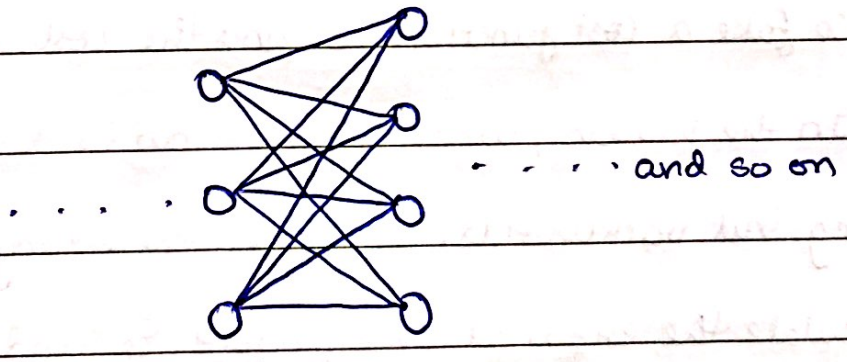
So there are a few problems with sigmoid & tanh. At the extremes of the graph you can see that the gradients at those points are almost zero. In the future we are going to ~~calculate~~ learn to calculate the weights as the gradient of the cost function. Now if our gradients start becoming zero after a point (Vanishing Gradients), we might have a problem. Therefore in practical cases we should go for ReLU.

BACK PROPAGATION:-

Let us first develop the intuition for neural networks. You have.

input layer → multiple hidden layers → output layer.

Each node in layer 'l-1' is connected to each node in layer 'l' as shown:-



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

k ∈ no of nodes in layer 'l-1'

j ∈ no of nodes in layer 'l'

w ∈ weights from layer 'l-1' to 'l'

b ∈ biases for nodes in layer 'l'

a_k^{l-1} → output of k^{th} neuron in layer l-1. This is the input to our current layer 'l'

a_j^l → output of j^{th} neuron in layer 'l'.

Now as expected our weights & biases are unknown & we have to calculate them.

Here are a few more equations which are important.

$$z_j^l = \sum w_{jk}^l a_k^{l-1} + b_j^l$$
$$a_j^l = \sigma(z_j^l)$$

} we just broke the previous two equations.

Lets talk about the intuition for the back propagation now.

Feed forward

" Lets say we started with random weights and biases. We the feed our input into the ~~network~~ network and get an output". Obviously the output is wrong. So what we do is backpropagate.

Back propagation

We take a cost function & find the cost. and try to minimize it by In the process, what we now do is do the same feed forward thing but backwards. We have the wrong output. We calculate the error at every node & update the weights so that the errors at each node is zero. We do this till we reach layer 1. Now we again do the feed forward with the new weights, calculate the cost & do the backpropagation again. We keep doing this for either some number of iterations or till we reach convergence or till we reach certain accuracy.

Mathematics of Back Propagation :-

Since we had mentioned a cost function, lets take MSE as our cost function (you can take any function as long as it is derivable)

$$C = \frac{1}{2n} \sum_{x} \| y(x) - a^L(x) \|^2$$

- $x \in$ input.
- $y(x) \in$ actual output for x
- $a^L(x) \in$ predicted output for x
- $L \in$ last layer
- $n \in$ number of inputs.

And since we are going to minimize the cost to find our weights & biases we are going to use the Gradient Descent Optimizer with the equations.

$$w_i : w_i - \eta \frac{\partial C}{\partial w_i}$$

$$b_i : b_i - \eta \frac{dC}{\partial b_i}$$

$\eta \in$ learning rate.

we can use two different learning rates of w & b optimization

We see that we have w_i, b_i, η. The two terms that are missing are $\frac{\partial C}{\partial w_i}$ & $\frac{\partial C}{\partial b_i}$

P.T.O

for each neuron

Let us define an error term that contributes to our overall cost function. Basically, remember that we mentioned that in the intuition, when we were doing backward pass & we were updating the weights so that we could get the correct output. Another way to do this would be to see it in this way:-

That the weights were correct but there was a change in the input that caused the error in the output. We are doing this because we dont have $\frac{\partial C}{\partial w}$.

We know our input to neuron 'j' of layer 'l' as z_j^l.

$$\therefore \delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad z_j^l \Rightarrow \text{output of previous node}$$

$$= \frac{\partial C}{\partial a_j^l} \times \frac{\partial a_j^l}{\partial z_j^l}. \qquad \Rightarrow a_j^l = \sigma(z_j^l)$$

$$\boxed{\delta_j^l = \frac{\partial C}{\partial a_j^l} \sigma'(z_j^l)} \quad \text{(we did this since we have } C \text{ in terms of } a_j^l\text{)}$$

→ ①.

Remember, all these intuitions are so that we can go through this process in a computationally less intensive way.

We could always have calculated $\frac{\partial C}{\partial w_i}$ but that would

require you to expand C from in terms of a_j^L to w_j^L and. for that you will need to known the entire numerical expansion from the input layer to the output which would have become a huge hassle.

Anyway, ~~combi~~ coming back to our explanation: -

$$\frac{\partial C}{\partial a_j^L} = \frac{\partial}{\partial a_j^L} \frac{1}{2} \sum_j (y_j - a_j^L)^2 = (a_j^L - y_j)$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}.$$

$$= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}.$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}.$$

$$\therefore \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l).$$

$$\therefore \boxed{\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l).} \quad -(2)$$

$$\frac{\partial C}{\partial b_j^l} = \sum \frac{\partial C}{\partial z_j^l} \times \frac{\partial z_j^l}{\partial b_j^l}.$$

$$= \delta_j^l \times \frac{\partial}{\partial b_j^l} \left(w_{jk}^l a_k^{l-1} + b_j^l \right)$$

$$\boxed{\frac{\partial C}{\partial b_j^l} = \delta_j^l} \quad -(3)$$

$$\frac{\partial C}{\partial w_j^l} = \sum \frac{\partial C}{\partial z_j^l} \times \frac{\partial z_j^l}{\partial w_{jk}^l}.$$

$$= \delta_j^l \times \frac{\partial}{\partial w_{jk}^l} \left(w_{jk}^l a_k^{l-1} + b_j^l \right).$$

$$\boxed{\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}} \quad -(4).$$

So we have the four equations for back propagation.

$$\delta^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^{L-1}) \circ \qquad \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l). \qquad \frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

This is enough for us to start on code.