Neural Network 4

Activation function

We saw that how sigmoid can be used as a activation function for hidden layer.

- Domain of sigmoid is $(-\infty,\infty)$
- Range is (0,1)
- Derivative of sigmoid also lies between (0,1)

tanh function

- Shifted version of sigmoid function
- Works better than sigmoid almost all the time - mean value is zero
- Inputs lies in the range: $(-\infty,\infty)$
- Output lies in the range: $(-1,1)$
- We don't use tanh function very often, unless we want output to lie in the range of (-1, 1).
- This lies between (0,1)

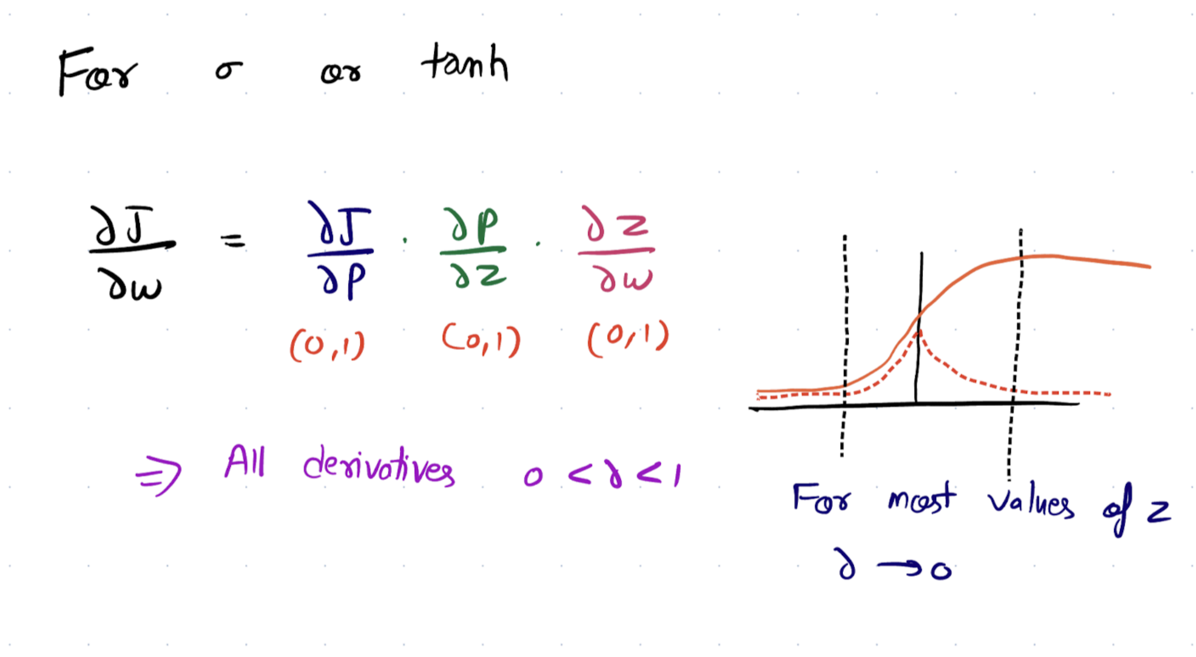
Vanishing Gradients

Downside of both sigmoid and tanh is that their gradient is <1, for most of the values of z

- This hampers the gradient descent process, and the calculated gradients will be very small.

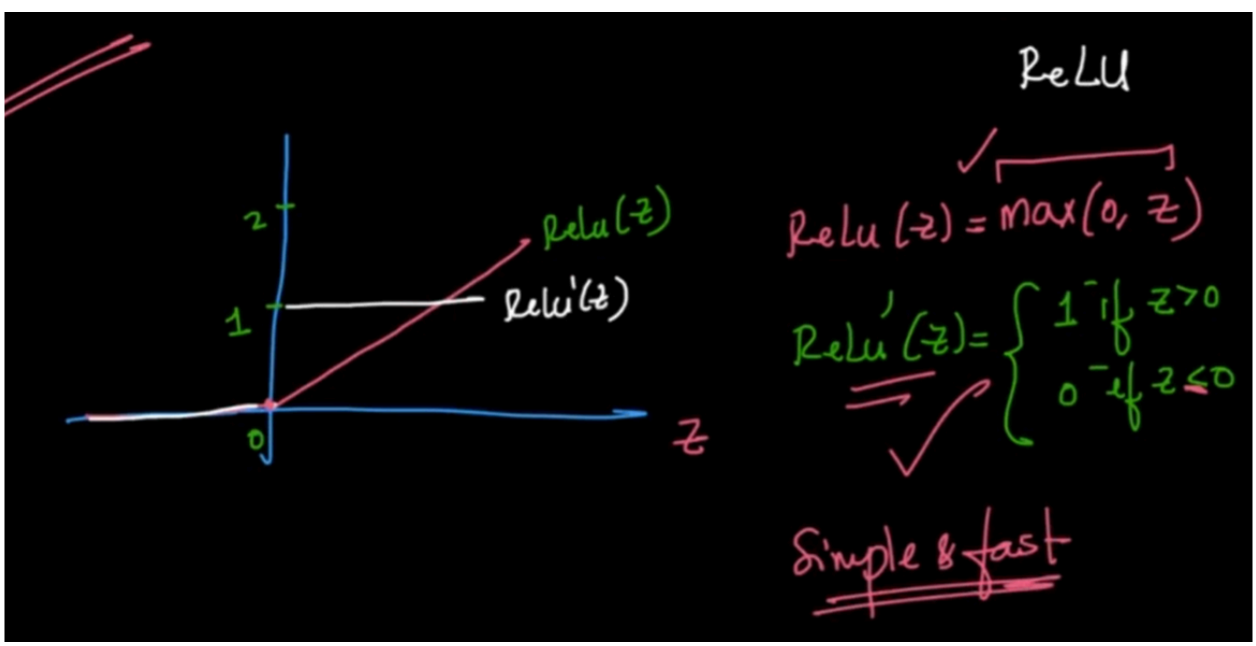
Why does small gradient hampers GD process ?
Since the activation functions are sigmoid or tanh

- We know that derivative of these functions lie between (0, 1).
- So, the product of these terms inside the bracket, will become very small.
- In fact, as the number of layers in the NN increase, this product will become smaller and smaller.



- We just saw that this partial derivative value becomes miniscule, as the number of layers increase.
- As a result, the NN gets trained very very slowly.

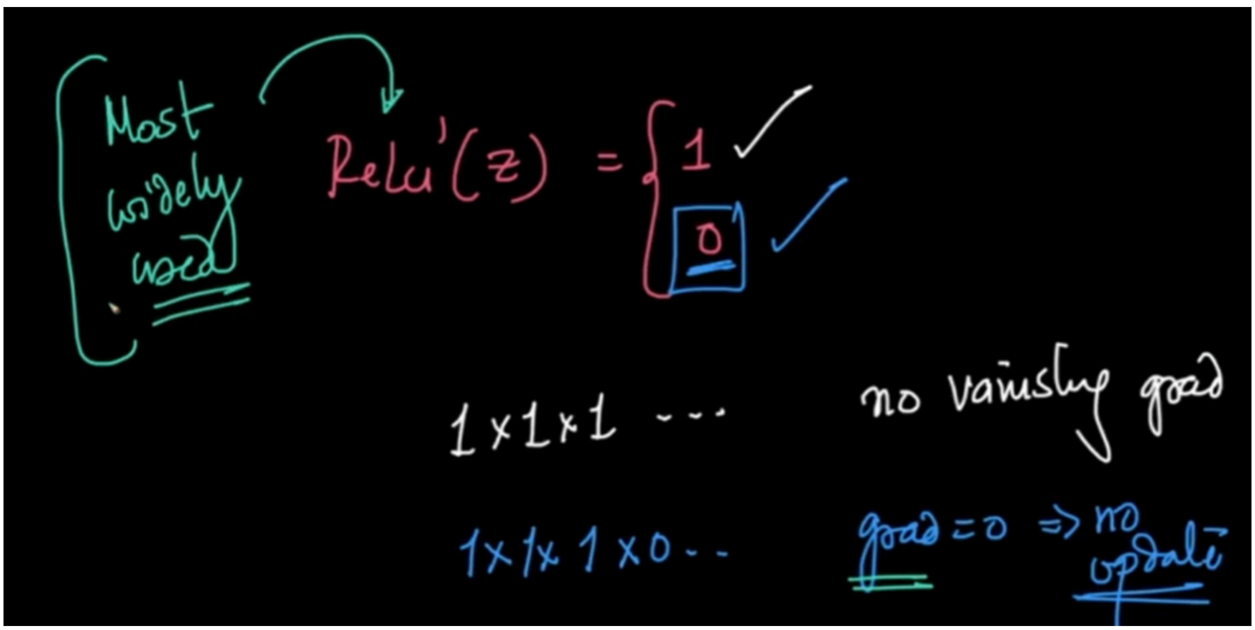
ReLu



Even though it is the most widely used activation function in the world of Deep Learning, there is a slight problem.

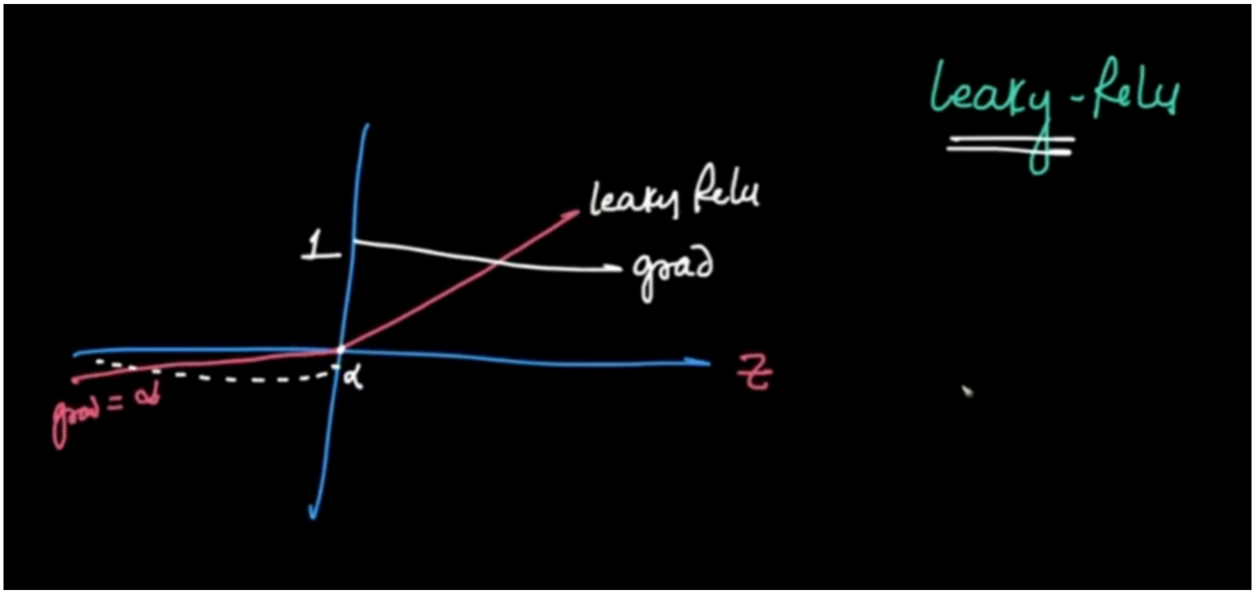
- If even one derivative term in calculation gets the value as 0, the entire term will become zero.
- Hence, there is no update in the value of weight.
- This is also known as **"dying ReLU"**
- So there's a potential **vanishing gradient** problem.
    - While calculating backprop, if one of the derivative is 0, the whole update will become zero and network won't update

However, to deal with this, a slight modification is made to ReLu, and we get another activation function known as **Leaky ReLu**



Leaky ReLu

- This is very similar to ReLu but there's a twist
- In case of negative values, we add a small gradient (alpha) associated with it, instead of having 0.



Forward Propagation

Forward Propagation is all easy.

- We just need to propagation our inputs from left to right
- Calculate the value of Zi
- Apply activation function on top of it
- And pass it to neuron in front of it.

Ultimately, we'll get the probabilities

- Use those probabilities to calcualte the loss.