* Forward Propagation and backward propagation in Neural network

→ A neural network training involves both forward propagation and backward propagation.

→ Forward propagation is the process in which the network's weights are updated according to the input, output and gradient of the neural network

→ Consider the following neural network

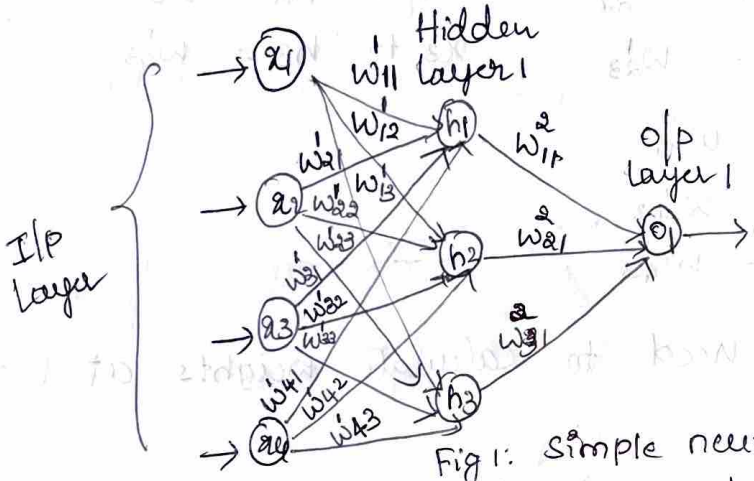


Fig 1: simple neural network

Forward propagation for the above neural network involves following steps.

As in fig 1, I/p layer consists of 4 neurons There exists one hidden layer with 3 hidden neurons and output layer has one output neuron.

Step 1: Apply weights for each neurons of the neural network.

→ weights can be initially set randomly, usually between 0 and 1

→ since there are 4 input neurons namely x_1, x_2, x_3, x_4 and in hidden layer H1, it has three hidden neurons h_1, h_2, h_3. The respetively weights can be as follows.

x_1 to h_1 as w_{11}^1 [where it is of w_{ij}^k format with i = Neuron number, j = connection i.e connection to hidden layer, k = layer number]

similarly, x_1 to h_2 can be $= w_{12}^1$

x_1 to h_3 can be w_{13}^1

x_2 to $h_1 = w_{21}^1$
x_2 to $h_2 = w_{22}^1$
x_2 to $h_3 = w_{23}^1$

x_3 to $h_1 = w_{31}^1$
x_3 to $h_2 = w_{32}^1$
x_3 to $h_3 = w_{33}^1$

x_4 to $h_1 = w_{41}^1$
x_4 to $h_2 = w_{42}^1$
x_4 to $h_3 = w_{43}^1$

similarly, we need to calculate weights at hidden layer 1 as

h_1 to $O_1 = w_{11}^2$
h_2 to $O_1 = w_{21}^2$
h_3 to $O_1 = w_{31}^2$

Once the weights are assigned, at each layer we must calculate cumulative sum of weights and inputs at each hidden neuron at layer 1 It can be calculated as follows.

At hidden layer H1 we have 3 neurons, hence we need to calculate the weighted sum at all 3 hidden neurons.

weighted sum at h1 is given by,

$$y_1 = (x_1 w_{11}^1 + x_2 w_{21}^1 + x_3 w_{31}^1 + x_4 \cdot w_{41}^1) + \text{bias}$$

step2 at h1

Then we must apply activation function at h1 for the value y_1

i.e $Z_1 =$ Activation(y_1)

similarly, at hidden neuron h2,

step1:

$$y_2 = (x_1 \cdot w_{12}^1 + x_2 \cdot w_{22}^1 + x_3 w_{32}^1 + x_4 w_{42}^1) + \text{bias}$$

step2: $Z_2 =$ Activation(y_2)

At neuron h3

step1: $y_3 = (x_1 w_{13}^1 + x_2 w_{23}^1 + x_3 \cdot w_{33}^1 + x_4 \cdot w_{43}^1) + \text{bias}$

step2: $Z_3 =$ Activation(y_3).

→ Once we complete all the iteration of hidden layer, the value of activation must be propagated to the next layer.

→ In our example, next layer is the output layer with one neuron O1.

→ Now we need to perform the computation at output layer i.e O1

step1:

$$y = (Z_1 \times w_{11}^2 + Z_2 w_{21}^2 + Z_3 \times w_{31}^2) + \text{bias}$$

step2: Apply activation function to y

$Z =$ Activation(y)

→ The computed value z is now propagated as output.

→ Hence " forward propagation is propagating the input value to the output."

→ The output value we get from forward propagation is called as predicted value. ($\hat{y}$)

→ The predicted value needs to be compared with the actual output during training process.

→ The difference or the loss we get after forward propagation needs to be minimized by re-updating the weights. Hence we require a back-propagation.

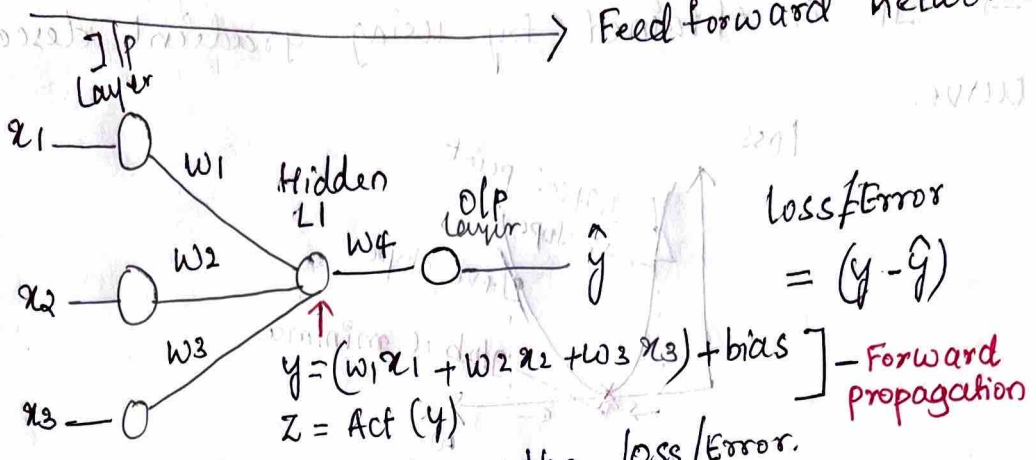
Back propagation algorithm

→ It is the method of fine tuning the weights of a neural network based on the error or loss obtained in the feed forward network.

→ Back propagation in neural network is a short form for " backward propagation of errors". This method helps to calculate the gradient of a loss function with respect to all the weights in the network.

→ The gradient can be calculated with the help of chain rule which helps to update the weights

Consider a neural network as shown below



main aim is to reduce the loss/Error.

y can also be written as

$$y = w^T x + b$$ ← Equation of a line.

Because of activation function we will be able to get non-linear value.

→ How can we reduce the Loss function? It can be reduced by updating weights initialized before using backpropagation.

→ Backpropagation involves 2 important things

a) weight updation formula

b) chain Rule of differentiation.

a) weight updation

Consider the neural network as shown above, we need to update the weights w_4, w_1, w_2, w_3 during backpropagation to reduce the error or loss.

weight updation formula =

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

(η → Learning Rate; $\frac{\partial L}{\partial w_{old}}$ ← i.e derivative of loss w.r.t to derivative of w_{old})

$\frac{\partial L}{\partial w_{old}}$ is what we call as a slope

How we will get the slope?

→ It is obtained by using gradient descent curve.

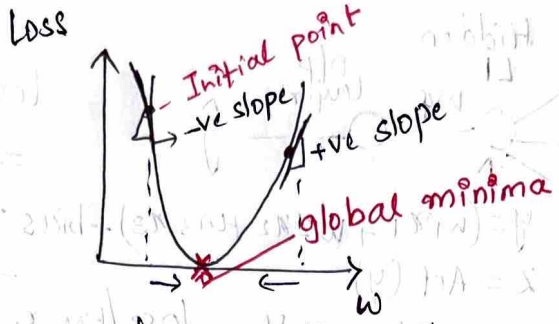


fig: Gradient descent curve.

→ gradient descent is a graph with respect to weight and loss function.

→ Our aim is to reach the global minima during training and updation of weights.

→ Now if we consider the slope in the weight updation formula, we need to calculate whether it is a +ve slope or -ve slope. by drawing the tangent to the point in the gradient descent curve.

→ If it is a -ve slope, to reach the global minima, we need to increase the weights, and if it is a +ve slope then we need to decrease the weights.

→ Usually learning rate (η) should be a small number (Eg: $\eta = 0.001$)

For +ve slope

$w_{new} = w_{old} - \eta(+ve\ value)$

$\boxed{w_{new} < w_{old}}$

For -ve slope

$w_{new} = w_{old} - \eta(-ve)$

$= w_{old} + \eta(value)$

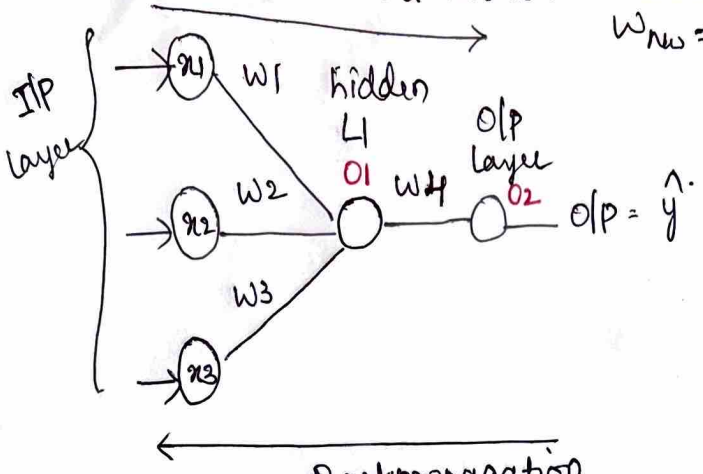
$\boxed{w_{new} > w_{old}}$

5) Chain Rule of differentiation.

feed forward network

weight updation formula.

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$



Backpropagation

During backpropagation, we need to update all the weights w_4, w_1, w_2, w_3.

$$w_{4new} = w_{4old} - \eta \frac{\partial L}{\partial w_{4old}}$$

$$\boxed{\frac{\partial L}{\partial w_{4old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial w_4}}$$ ← Basic chain rule.

as loss is dependent on output O_2, O_2 is dependent on w_4

To update w_1, $w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_{1old}}$.

$$\frac{\partial L}{\partial w_{1old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial w_{1old}}$$

[loss is dependent on O_2, O_2 is dependent on O_1, O_1 is dependent on w_1.]

similarly,

$$w_{2new} = w_{2old} - \eta \frac{\partial L}{\partial w_{2old}}$$

$$\frac{\partial L}{\partial w_{2old}} = \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial w_{2old}}$$

$$= \frac{\partial L}{\partial O_2} * \frac{\partial O_2}{\partial w_4} * \frac{\partial w_4}{\partial O_1} * \frac{\partial O_1}{\partial w_{2old}}$$

In the similar way, weights will be updated.