BACKPROPAGATION

By Dr. Harish Kumar Taluja

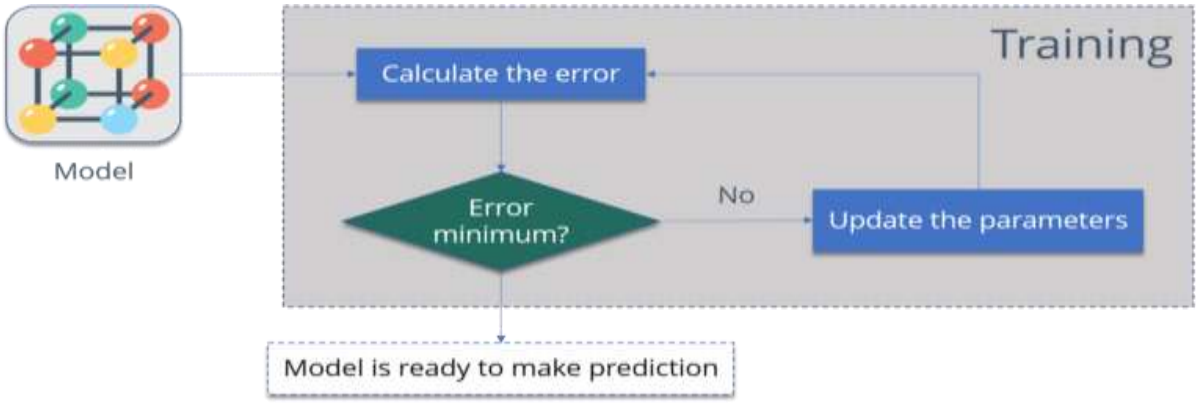
Backpropagation is a commonly used supervised learning algorithm used for training neural networks or Multi-layer Perceptron's (Artificial Neural Networks). The term backpropagation and its general use in neural networks was announced in Rumelhart, Hinton & Williams (1986a). Backpropagation is also known as "backward propagation of errors". Backpropagation is a combination of two words: Back and Propagation. The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Partial computations of the gradient from one layer are reused in the computation of the gradient for the previous layer. This backwards flow of the error information allows for efficient computation of the gradient at each layer versus the naive approach of calculating the gradient of each layer separately. An Artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. It is a generalization of the delta rule for perceptron's to multilayer feedforward neural networks.In fitting a neural network, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single input–output example, and does so efficiently, unlike a naive direct computation of the gradient with respect to each weight individually.

This efficiency makes it feasible to use gradient methods for training multilayer networks, updating weights to minimize loss; gradient descent, or variants such as stochastic gradient descent, are commonly used. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this is an example of dynamic programming[3].The term backpropagation strictly refers only to the algorithm for computing the gradient, not how the gradient is used; but the term is often used loosely to refer to the entire learning algorithm, including how the gradient is used, such as by stochastic gradient descent.[4] Backpropagation generalizes the gradient computation in the delta rule, which is the single-layer version of backpropagation, and is in turn generalized by automatic differentiation, where backpropagation is a special case of reverse accumulation (or "reverse mode")[5] .The term backpropagation and its general use in neural networks was announced in Rumelhart, Hinton & Williams (1986a), then elaborated and popularized in Rumelhart, Hinton & Williams (1986b), but the technique was independently rediscovered many times, and had many predecessors dating to the 1960s; see § History.[6] A modern overview is given in the deep learning textbook by Goodfellow, Bengio& Courville (2016).[7]

Why We Need Backpropagation?

While designing a Neural Network, in the beginning, we initialize weights with some random values or any variable for that fact.Now obviously, we are not *superhuman.* So, it's not necessary that whatever weight values we have selected will be correct, or it fits our model the best. Okay, fine, we have selected some weight values in the beginning, but our model output is way different than our actual output i.e. the error value is huge.Now, how will you reduce the error?

Basically, what we need to do, we need to somehow explain the model to change the parameters (weights), such that error becomes minimum. Let's put it in an another way, we need to train our model.One way to train our model is called as Backpropagation. Consider the diagram below:



Let me summarize the steps for you:

- **Calculate the error** – How far is your model output from the actual output.
- **Minimum Error** – Check whether the error is minimized or not.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

Consider the below table:

Input	Desired Output
0	0
1	2
2	4

Now the output of your model when 'W" value is 3:

Input	Desired Output	Model output (W=3)
0	0	0
1	2	3
2	4	6

Notice the difference between the actual output and the desired output:

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error

0	0	0	0	0
1	2	3	1	1
2	4	6	2	4

Let's change the value of 'W'. Notice the error when 'W' = '4'

Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=4)	Square Error
0	0	0	0	0	0	0
1	2	3	1	1	4	4
2	4	6	2	4	8	16

Now if you notice, when we increase the value of 'W' the error has increased. So, obviously there is no point in increasing the value of 'W' further. But, what happens if I decrease the value of 'W'? Consider the table below:

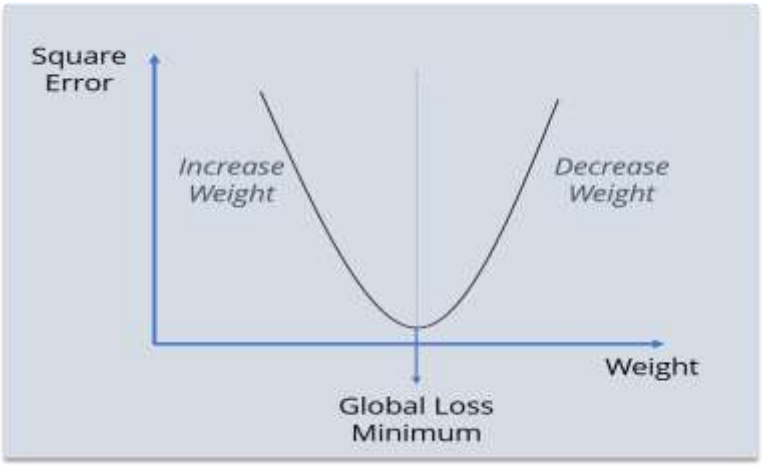
Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Square Error
0	0	0	0	0	0	0
1	2	3	2	4	3	0
2	4	6	2	4	4	0

Now, what we did here:

- We first initialized some random value to 'W' and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of 'W'.
- After that, also we noticed that the error has increased. We came to know that, we can't increase the 'W' value.
- So, we again propagated backwards and we decreased 'W' value.
- Now, we noticed that the error has reduced.

So, we are trying to get the value of weight such that the error becomes minimum. Basically, we need to figure out whether we need to increase or decrease the weight value. Once we know that, we keep on updating the weight value in that direction until error becomes minimum. You might reach a point, where if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value.

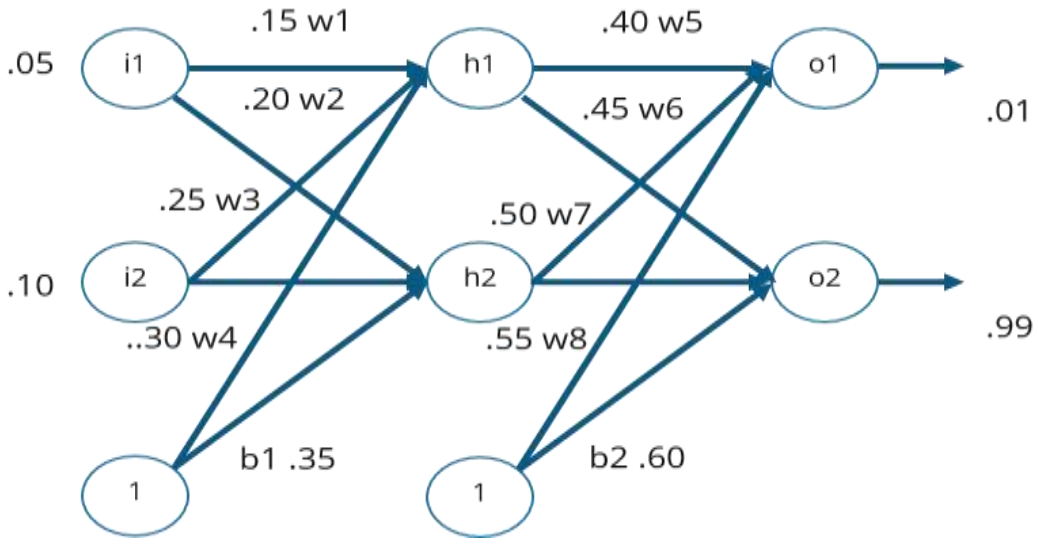
Consider the graph below:



We need to reach the 'Global Loss Minimum'. This is nothing but Backpropagation.Let's now understand the math behind Backpropagation.

How Backpropagation Works?

Consider the below Neural Network:



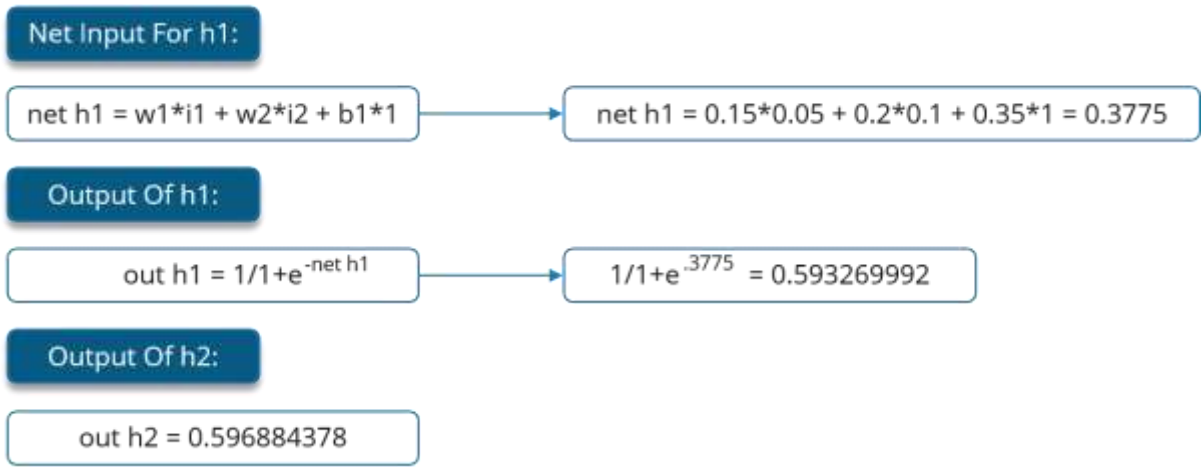
The above network contains the following:

- two inputs
- two hidden neurons
- two output neurons
- two biases

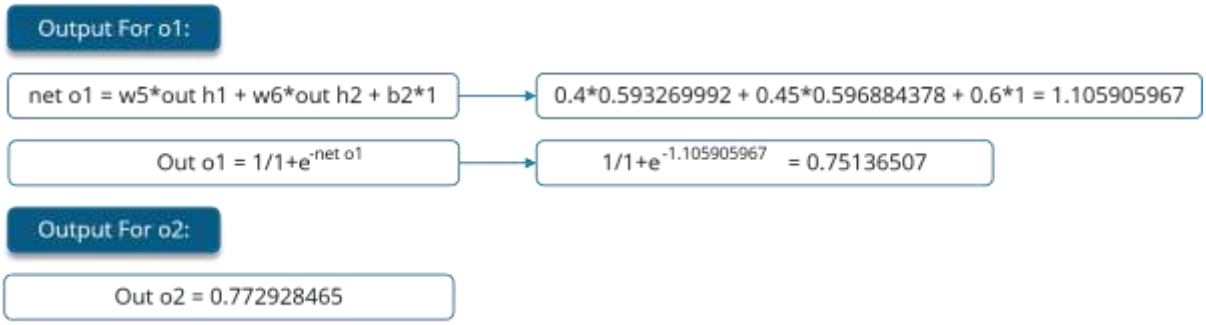
Below are the steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step – 3: Putting all the values together and calculating the updated weight value

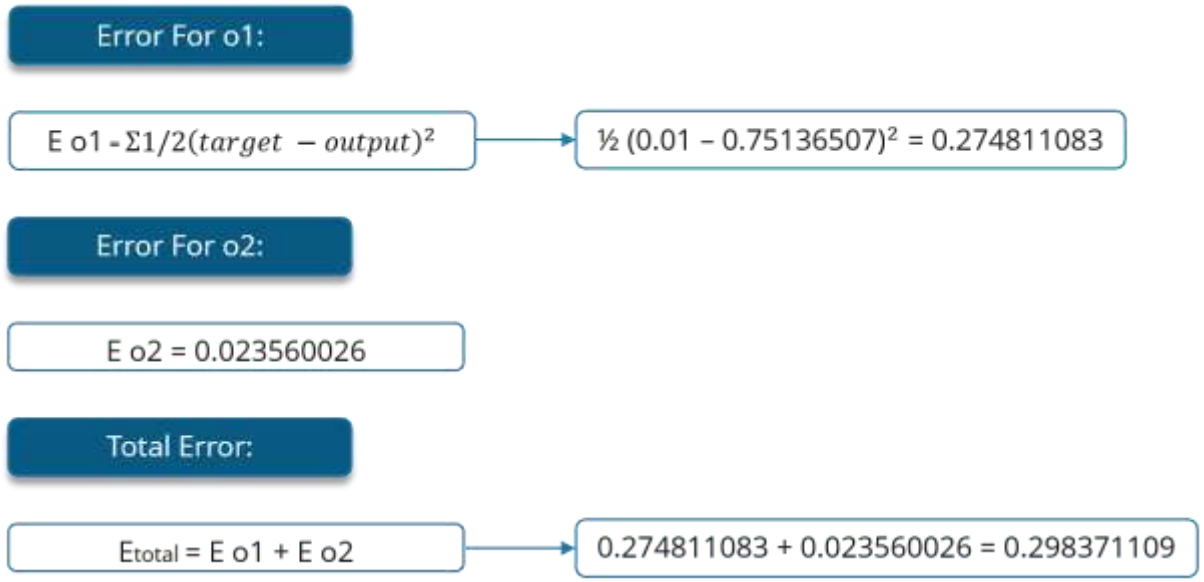
Step – 1: Forward Propagation



We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.



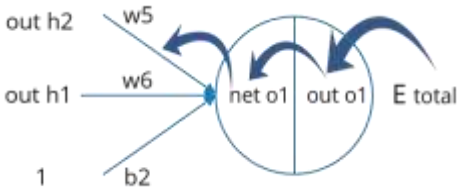
Now, let's see what is the value of the error:



Step – 2: Backward Propagation
Now, we will propagate backwards. This way we will try to reduce the error by changing the values of weights and biases.

Consider W5, we will calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



Since we are propagating backwards, first thing we need to do is, calculate the change in total errors w.r.t the output O1 and O2.

$$E_{total} = 1/2(target\ o1 - out\ o1)^2 + 1/2(target\ o2 - out\ o2)^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(target\ o1 - out\ o1) = -(0.01 - 0.75136507) = 0.74136507$$

Now, we will propagate further backwards and calculate the change in output O1 w.r.t to its total net input.

$$out\ o1 = 1/1+e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = out\ o1\ (1 - out\ o1) = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

Let's see now how much does the total net input of O1 changes w.r.t W5?

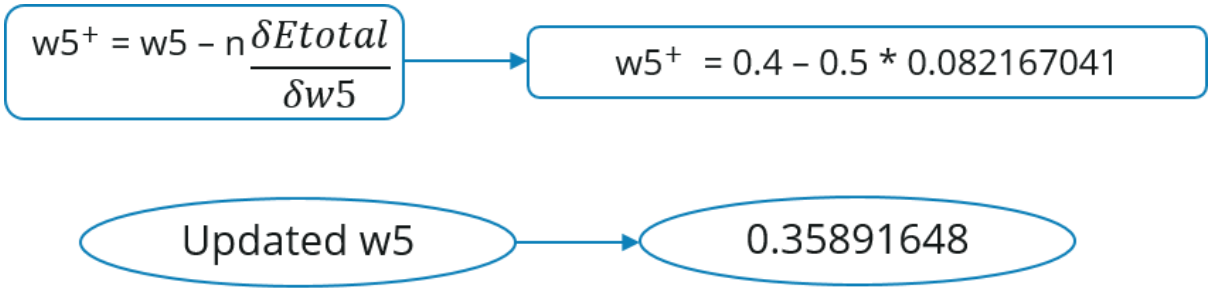
$$net\ o1 = w5 * out\ h1 + w6 * out\ h2 + b2 * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * out\ h1\ w5^{(1-1)} + 0 + 0 = 0.593269992$$

Step – 3: Putting all the values together and calculating the updated weight value
Now, let's put all the values together:

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5} \rightarrow 0.082167041$$

Let's calculate the updated value of W5:



- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum.

Backpropagation Algorithm:
initialize network weights (often small random values)

do

forEach training example named ex

prediction = neural-net-output(network, ex) *// forward pass*

actual = teacher-output(ex)

compute error (prediction - actual) at the output units

compute {displaystyle Delta w_{h}} for all weights from hidden layer to output layer *// backward pass*

compute {displaystyle Delta w_{i}} for all weights from input layer to hidden layer *// backward pass continued*

update network weights *// input layer not modified by error estimate*

until all examples classified correctly or another stopping criterion satisfied

return the network