<https://7-hiddenlayers.com/wp-content/uploads/2020/06/NEURONS-IN-NUERAL-NETWORK.gif>

**Forward Propagation** is the way to move from the Input layer (left) to the Output layer (right) in the neural network. The process of moving from the right to left i.e backward from the Output to the Input layer is called the **Backward Propagation.**

Backward Propagation is the preferable method of adjusting or correcting the weights to reach the minimized loss function. In this article, we shall explore this second technique of Backward Propagation in detail by understanding how it works mathematically, why it is the preferred method. A caution, the article is going to be a mathematically heavy one but wait for the end to see how this method looks in action

## Table of Contents

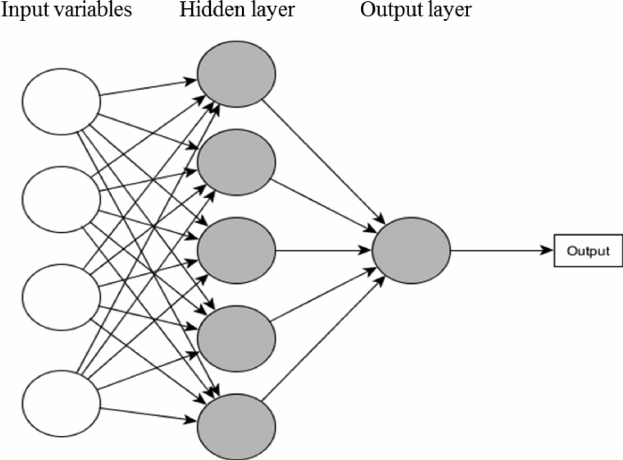
* Setting up the Base
* Contribution of each Weight and Bias on the Error
* Matrix Form of the Backward Propagation

## Setting up the Base

Let’s say we want to use the neural network to predict house prices. For our understanding purpose here, we will take a subset dummy dataset having four input variables and six observations here with the input having a dimension of 4\*5:



The neural network for this subset data looks like below:



Source: [researchgate.net](https://www.researchgate.net/profile/Vivian-Breen/publication/273773636/figure/fig2/AS:294936114155530@1447329657512/ANN-containing-one-hidden-layer-and-one-output-layer-with-5-neurons-and-1-neuron.png)

The architecture of the neural network is [4, 5, 1] with:

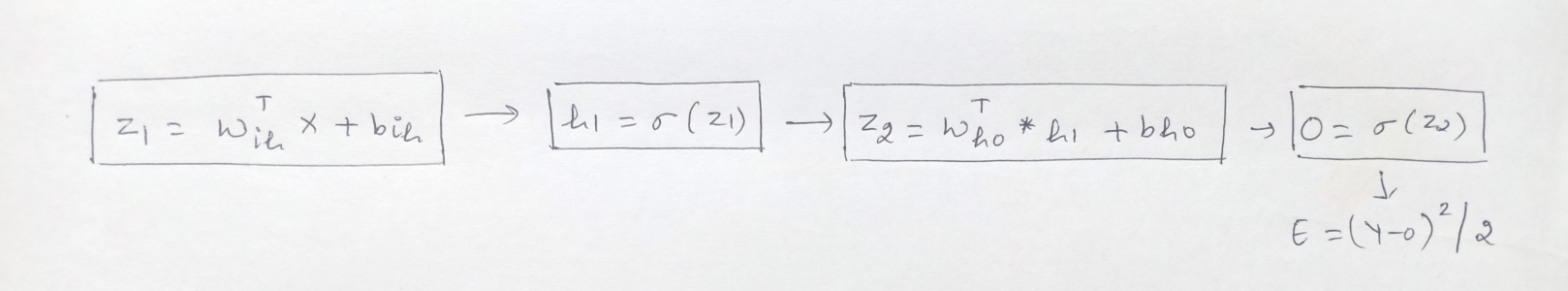
* 4 independent variables, Xs in the input layer
* 5 nodes in the hidden layer, and
* Since we have a regression problem at hand, we will have one node in the output layer.

A Neural Network operates by:

1. Initializing the weights with some random values, which are mostly between 0 and 1.
2. Compute the output to calculate the loss or the error term.
3. Then, adjust the weights so that to minimize the loss.

We repeat these three steps until have reached the optimum solution of the minimum loss function or exhausted the pre-defined epochs (i.e. the number of iterations).

Now, the computation graph after applying the sigmoid activation function is:



In case you are wondering how and from where this equation arrived and why there will be matrix dimensions then request you to read the [previous article](https://www.analyticsvidhya.com/blog/2021/04/estimation-of-neurons-and-forward-propagation-in-neural-net/) to understand the mechanism of how neural networks work and are estimated.

Building on this, the first step in Backward Propagation to calculate the error. In our regression problem, we shall take the loss function = (Y-Y^)2/2 where Y is actual values and Y^ is predicted values. For simplicity, replacing Y^ with O, so the error E becomes = (Y-O)2/2.

Our goal is to minimize the error that is clearly dependent on Y, which is the actual observed values, and on the output, which is further is dependent on the:

* input values
* coefficients or betas of the input variables
* biases, the [activation function](https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6?gi=2074f5569281), and
* [Optimizers](https://www.analyticsvidhya.com/blog/2021/04/is-gradient-descent-sufficient-for-neural-network/)

Now, we can neither change the input variables nor the actual Y values however, we can change the other factors. The activation function and the optimizers are the tuning parameters – and we can change these based on our requirement.

The other two factors: the coefficients or betas of the input variables (Wis) and the biases (bho, bih) are updated using the Gradient descent algorithm with the following equation:

Wnew = Wold – (α \* dE/dW)

where,

* Wnew = the new weight of Xi
* Wold = the old weight of the Xi
* α = learning rate
* dE/dW is the partial derivative of the error for each of the Xs. It is the rate of change of the error to the change in weight.

In the backward propagation, we adjust these weights or the betas in the output. The weights and biases between the respective input, hidden and output layers we have here are Wih, bih, Who, and bho:

* Wih: weight between the input and the hidden layer
* bih: bias between the input and the hidden layer
* Who: weight between the hidden and the output layer
* bho: bias between the hidden and the output layer

In the first iteration, we randomly initialize the weights. In the second iteration, we change the weights of the hidden layer that is closest to the output layer. In this case, we go from the output layer, hidden layer, and then to the input layer.

## Contribution of each Weight and Bias on the Error

Now, we have to calculate how much each of these weights (Wis) and biases (bis) contribute to the error term. For this, we need to calculate the rate of change of error to the respective weights and bias parameters.

In other words, we need to compute the terms: dE/dWih, dE/dbih, dE/dWho, and dE/dbho. This is not a direct task. It is a series of steps involving the [Chain Rule](https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html).

**The weight, Who, between the hidden and the output layer:**

From the above graph we can see that the error E is not directly dependent on the Who:

* The error term is dependent on the Output O
* Output O is further dependent on Z2, and
* Z2 is dependent on Who

Therefore we employ the chain rule to compute the rate of change in error to the change in weight Who and it becomes:

dE/dWho = dE/dO \* dO/dZ2 \* dZ2/dWho

Now, we take the partial derivatives of each of these individual terms:

E = (Y-O)2/2.

1. The partial derivative of error with respect to Output is: dE/dO = 2\*(Y-O)\*(-1)/2 = (O-Y)
2. The partial derivative of Output with respect to Z2, as output O = Sigmoid of Z2 and the derivative of sigmoid is:

dO/dZ2 = sigmoid(Z2) \*(1-sigmoid(Z2)) = O\*(1-O)

1. The partial derivative of Z2 with respect to Who is:

dZ2/dWho = d(WhoT \* h1 + bh0)/dWho

dZ2/dWho = d(WhoT \* h1)/dWho + d(bho/Who) = h1 + 0 = h1

Therefore, dE/dWho = dE/dO \* dO/dZ2 \* dZ2/dWho becomes:

**dE/dWho = (O-Y) \* O\*(1-O) \* h1**

Similarly, we will calculate the contribution for each of the other parameters in this manner.

**For the bias, bho, between the hidden and the output layer:**

dE/dbho = dE/dO \* dO/dZ2 \* dZ2/dbho

**dE/dbho = (O-Y) \* O\*(1-O) \* 1**

**The weight, Wih, between the input and the hidden layer:**

From the above graph we can see that the terms are dependent as below:

* Error term is dependent on the Output O
* Output O is dependent on Z2
* Z2 this time is dependent on h1
* h1 is dependent on Z1, and
* Z1 is dependent on Wih

dE/dWih = dE/dO \* dO/dZ2 \* dZ2/dh1 \* dh1/dZ1 \* dZ1/dWih

So, this time, apart from the initial above dE/dO, dO/dZ2, we have the partial derivatives as follow:

1. The partial derivative of Z2 with respect to h1 is:

dZ2/dh1 = d(WhoT \* h1 + bho)/dh1

dZ2/dh1 = d(WhoT \* h1)/dh1 + d(bho/h1) = Who + 0 = Who

1. The partial derivative of h1 with respect to Z1, as h1 = Sigmoid of Z1 and the derivative of sigmoid is:

dh1/dZ1 = sigmoid(Z1) \*(1-sigmoid(Z1)) = h1\* (1 – h1)

1. The partial derivative of Z1 with respect to Wih is: X

dZ1/dWih = d(WihT \* X + bih)/dWih

dZ1/dWih = d(WihT \* X)/dWih + d(bih/Wih) = X + 0 = X

Hence, the equation after plugging the partial derivative value is:

dE/dWih = dE/dO \* dO/dZ2 \* dZ2/dh1 \* dh1/dZ1 \* dZ1/dWih

**dE/dWih = (O-Y) \* O\*(1-O) \* Who \* h1(1-h1) \* X**

**The bias, bih, between the input and the hidden layer:**

dE/dbih = dE/dO \* dO/dZ2 \* dZ2/dh1 \* dh1/dZ1 \* dZ1/dbih

**dE/dbih = (O-Y) \* O\*(1-O) \* Who \* h1(1-h1) \* 1**

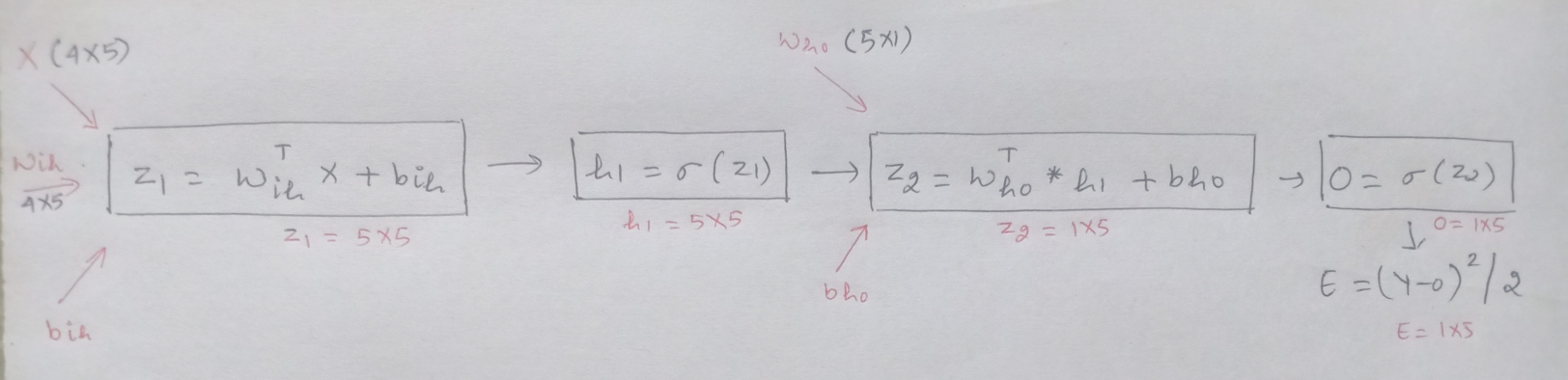
Now, that we have computed these terms we can update the parameters using the following respective update equations:

* Wih = Wih – (α \* dE/dWih)
* bih = bih – (α \* dE/dbih)
* Who = Who – (α \* dE/dWho)
* bho = bho – (α \* dE/dbho)

Now, moving to another method to perform backward propagation …

## Matrix Form of the Backward Propagation

The backward propagation can also be solved in the matrix form. The computation graph for the structure along with the matrix dimensions is:



Z1 = WihT \* X + bih

where,

* Wih is the weight matrix between the input and the hidden layer with the dimension of 4\*5
* WihT, is the transpose of Wih, having shape 5\*4
* X is the input variables having dimension 4\*5, and
* bih is a bias term, has a single value here as considering the same for all the neurons.

Z2 = WhoT \* h1 + bho

where,

* Who is the weight matrix between the hidden and the output layer with shape 5\*1
* WhoT, is the transpose of Who having a dimension of 1\*5
* h1 is the result after the applying activation function on the outcome from the hidden layer with a shape of 5\*5, and
* bho is the bias term,  has a single value here as considering the same for all the neurons.

To summarize, the four equations of the rate of change of error with the different parameters are:

dE/dWho = dE/dO \* dO/dZ2 \* dZ2/dWho= **(O-Y) \* O\*(1-O) \* h1**

dE/dbho = dE/dO \* dO/dZ2 \* dZ2/dbho**= (O-Y) \* O\*(1-O) \* 1**

dE/dWih = dE/dO \* dO/dZ2 \* dZ2/dh1 \* dh1/dZ1 \* dZ1/dWih = **(O-Y) \* O\*(1-O) \* Who \* h1(1-h1) \* X**

dE/dbih = dE/dO \* dO/dZ2 \* dZ2/dh1 \* dh1/dZ1 \* dZ1/dbih = **(O-Y) \* O\*(1-O) \* Who \* h1(1-h1) \* 1**

Now, lets’ see how we can perform matrix multiplication on each of these equations. For the weight matrix between the hidden and the output layer, Who.

Let us understand how the shape of this Who must be similar to that of the shape of dE/dWho, which is to used to update the weight in the following equation:

Who = Who – (α \* dE/dWho)

We saw above that dE/dWho is computed using the chain rule and is of the result:

dE/dWho = dE/dO \* dO/dZ2 \* dZ2/dWho

**dE/dWho = (O-Y) \* O\*(1-O) \* h1**

Breaking the individual components of this above equation we see each part’s dimension:

dE/dO = (O-Y) as both O and Y have the same shape of 1\*5. Hence, dE/dO is of dimension 1\*5.

dO/dZ2 = O\*(1-O) having a shape of 1\*5, and

dZ2/dWho = h1, which is of the shape 5\*5

Now, performing matrix multiplication on this equation. As we know, matrix multiplication can be done when the number of columns of the first matrix must be equal to the number of rows of the second matrix. Where this matrix multiplication rule defies, we will take the transpose of one of the matrices to conduct the multiplication.

On applying this our equation takes the form of:

dE/dWho = dZ2/dWho . [dE/dO \* dO/dZ2] T

dE/dWho = (5X5) . [(1X5) \*(1X5)]T

dE/dWho = (5X5) . (5X1) = 5X1

Therefore, the shape of dE/dWho 5\*1 is the same as that of Who 5\*1 which will be updated using the Gradient Descent update equation.

In the same manner, we can find perform the backward propagation for the other parameters using matrix multiplication and the respective equations will be:

dE/dWho = dZ2/dWho . [dE/dO \* dO/dZ2] T

dE/dbho = dZ2/dbho . [dE/dO \* dO/dZ2 ]T

dE/dWih = dZ1/dWih . [dh1/dZ1 \* dZ2/dh1. (dE/dO \* dO/dZ2)] T

dE/dbih = dZ1/dbih . [dh1/dZ1 \* dZ2/dh1. (dE/dO \* dO/dZ2)] T

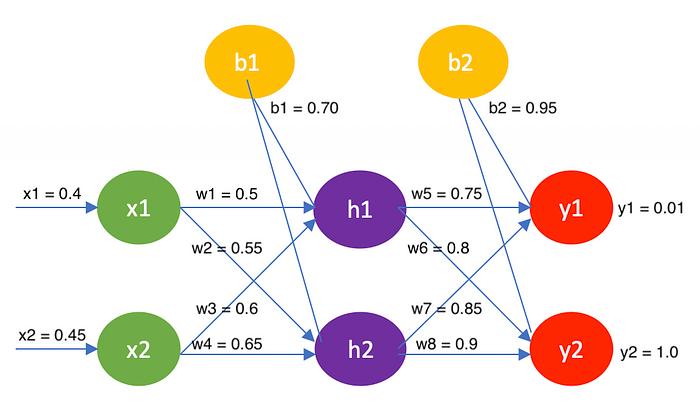
Where, (.) dot is the dot product and \* is the element wise product.

======================-=================-===============

# How does the mathematics behind Back-propagation Algorithm work? Math Time!!!

we will be discussing the step by step approach for a forward pass and backward propagation. I will be using overleaf for mathematics, as it is discussed in depth.

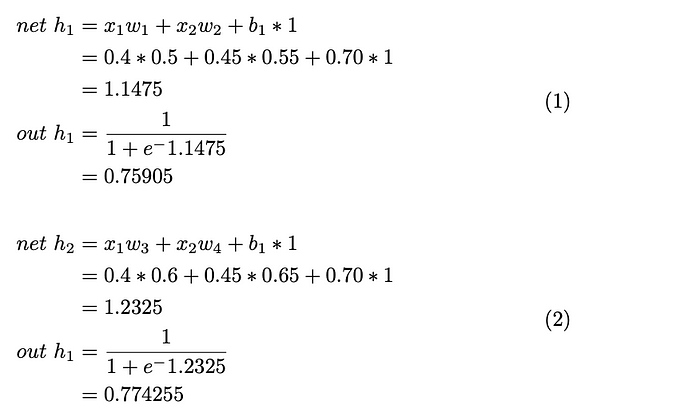
A 3 layer neural network will help us understand the propagation in a simplified manner.



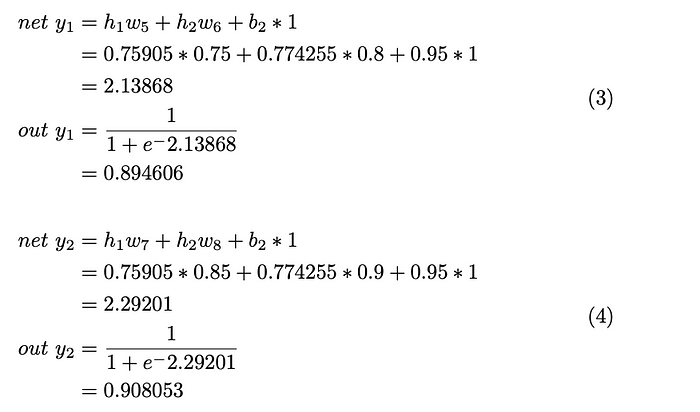
3-layer Neural Network

Let’s start by calculating the values of the hidden layer and the output layer.

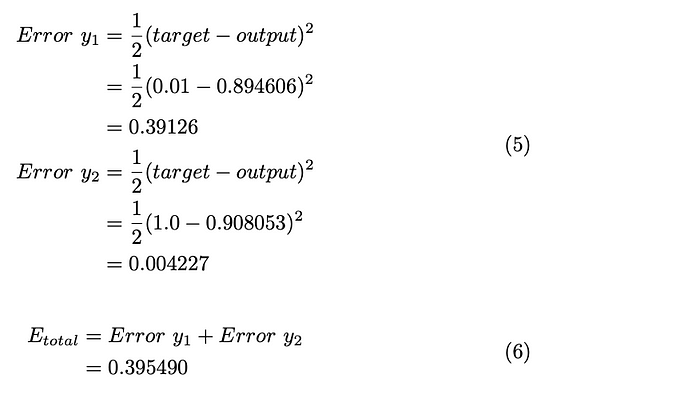
**Hidden layer: h1 and h2**



**Output layer: y1 and y2**



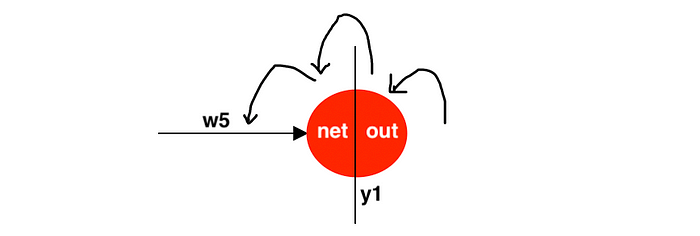
**Error in the node y1 and y2, add up to get the total error.**

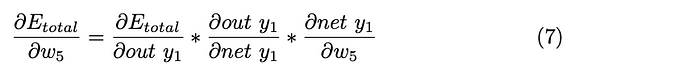


# Backward propagation

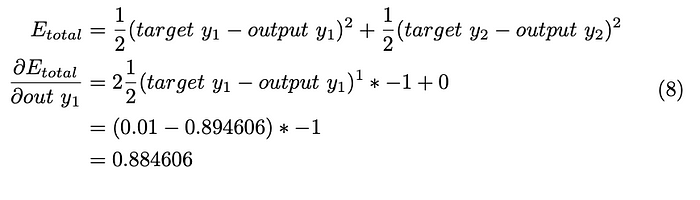
There is a difference between the values of the output layer (y1=0.01 and y2=1.0) and the calculated values (y1 = 0.894606 and y2 = 0.908053). To reduce the error, we have to adjust the weights accordingly.

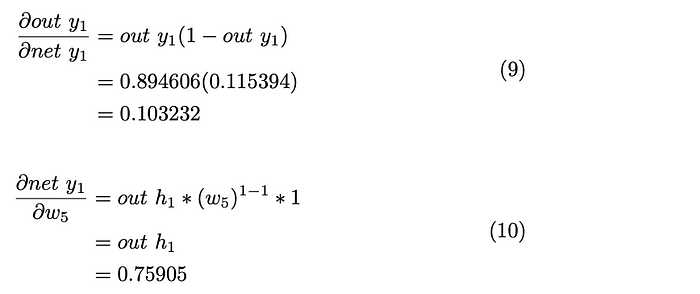
*Consider the weight w5 and apply the chain rule.*



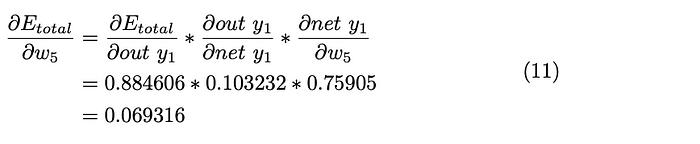


It gets complicated if we tackle all the partial differentiation at the same time. So, we will solve them one by one.





Substituting the equations 8,9,10 in eq.7

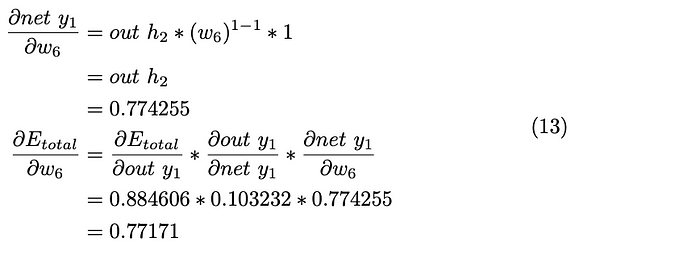


Finally, the weight w5 is updated. Here the learning rate is assumed to be η=0.4(It can vary from 0 to 1).



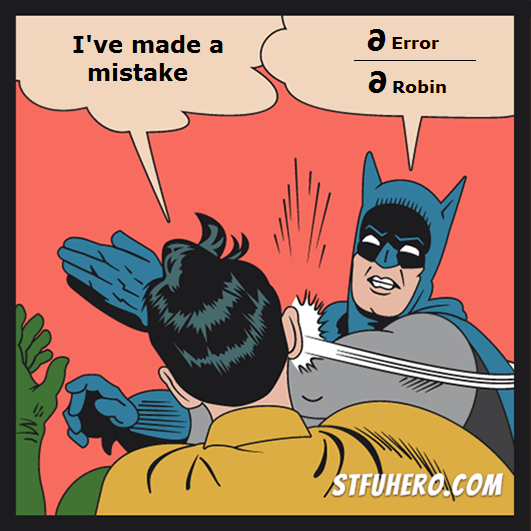
*Now, consider wight w6.*

We already have the values for the first two parts of the equation in equations 8 and 9. Solving for the last part of the equation and substitute back into equation 7.



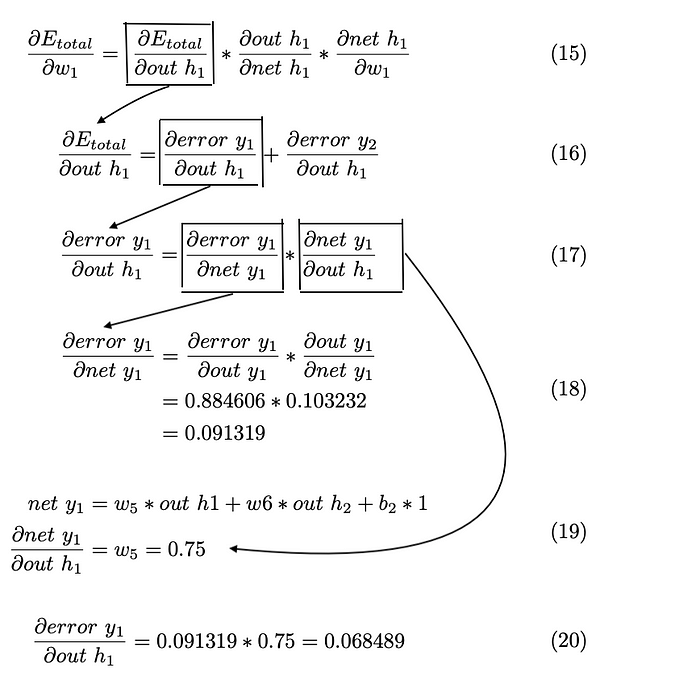
In the similar manner, we update the weights w7 and w8.



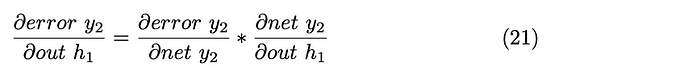


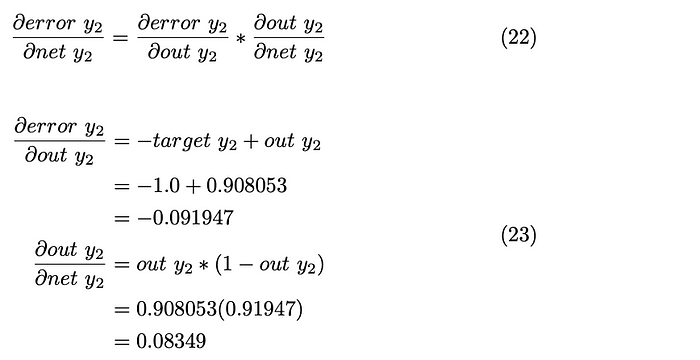
After solving the layer between the hidden and output layers, we will be updating the weights between the hidden and input layers.

Weight w1 can be updated through a series of equations. As it can be complicated, I have used arrows to indicate the continue chains of the equations.

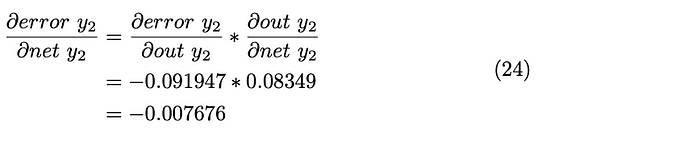


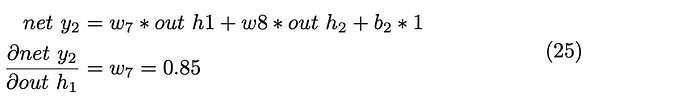
In the same way, the error in the out node y2 because of the hidden node h1 in eq.16 is calculated.



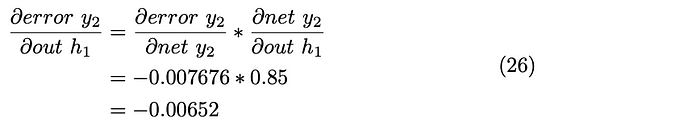


Substitute eq.23 in eq.22

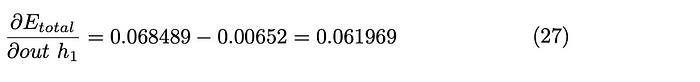


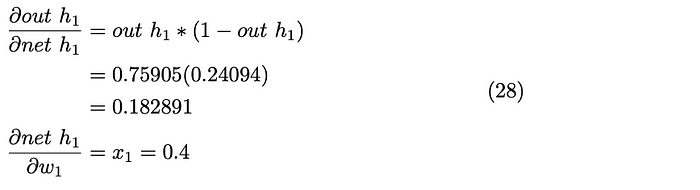


Now substitute eq.24 and 25 in eq.21

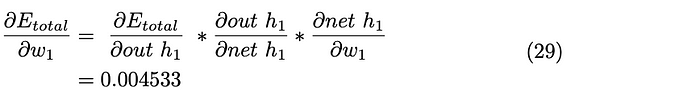


We now have the first part of the equation. By substituting the equations 26 and 20 into eq.16





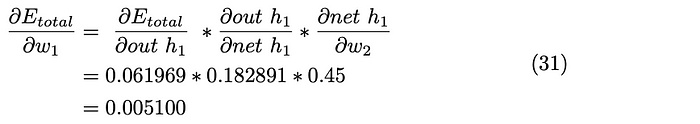
We get the final result by substituting the eq.27 and 28 into eq.15



Weight w1 is updated by multiplying the error with the learning rate.



Now, calculate the updated weight w2 as follows. But it is quicker as we already have the values from eq.27 and 28. The only change is the last part of the equation. i.e the weight.





Follow the same methods to calculate the updated weights w3 and w4.



This completes the first iteration. And be iterated until the error is minimized .