Neurals!

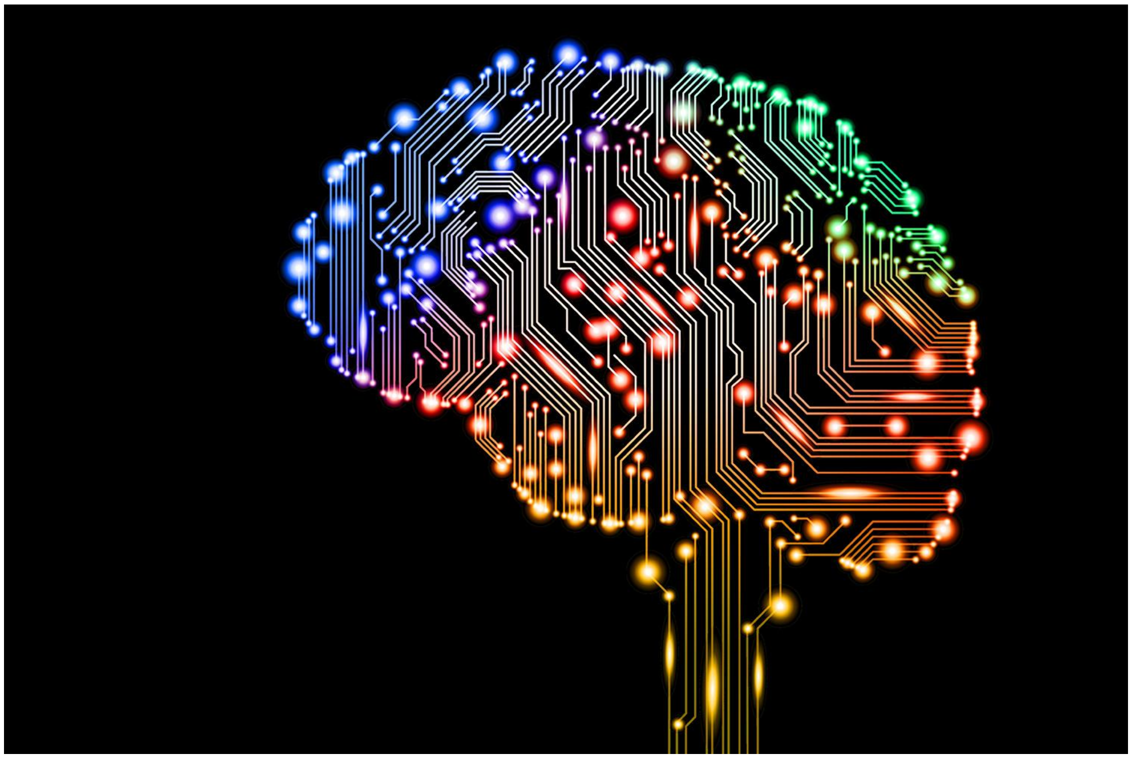
February 24, 2017

0.1 Building a Neural Network using Python(From Scratch!)

0.1.1 What we think when we hear the word Neural?

```
In [8]: from IPython.display import Image
        Image('/home/metal-machine/Desktop/ai_0.jpg')
```

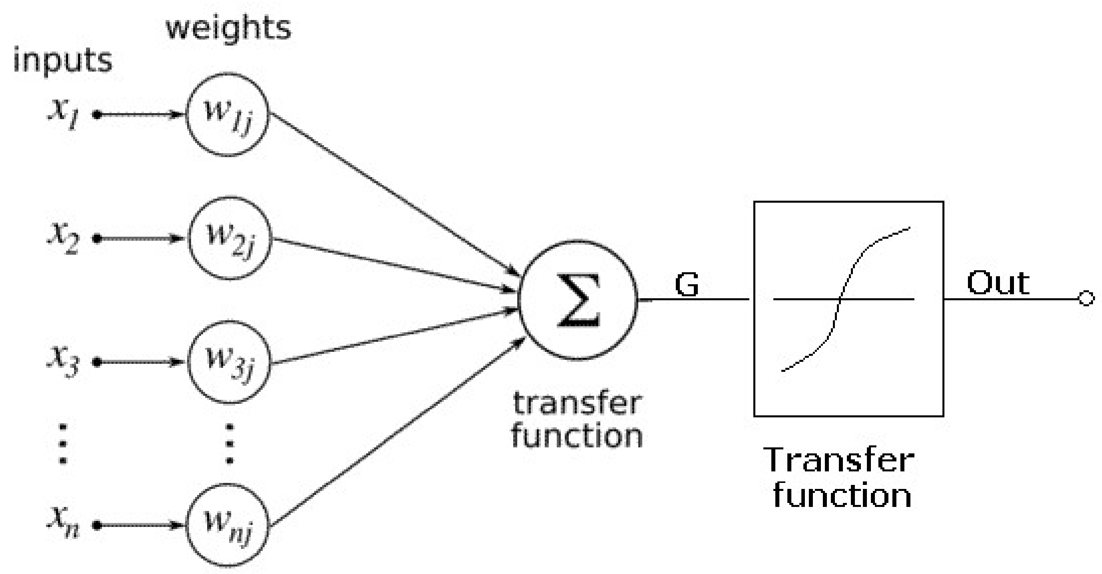
```
Out[8]:
```



0.2 What it actually is?

```
In [9]: from IPython.display import Image
        Image('/home/metal-machine/Desktop/nno.jpg')
```

```
Out[9]:
```



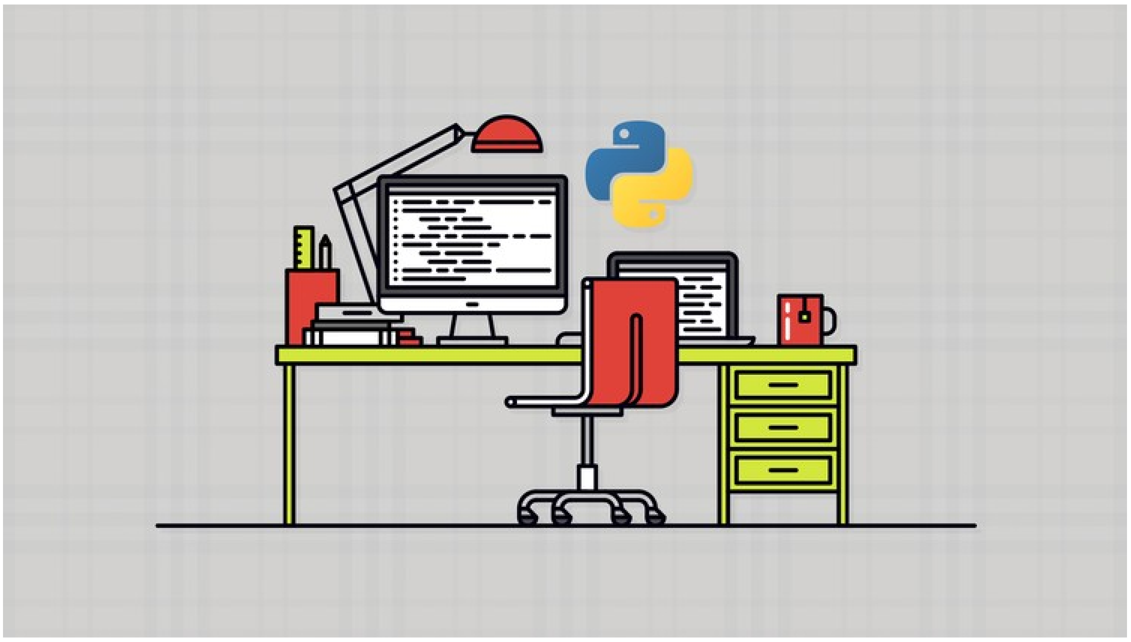
0.3 Simple steps when we build a Neural network

1. Get ready with your input and output DataSets.
2. Choose Your Activation Function.
3. Calculate Error
4. Minimise your Error

0.4 What we will be doing?

1. Create four samples one output Neuron Each.
2. Using Numpy to create Array(Matrix) of Random weights
3. Using Sigmoid as activation Function.

```
In [46]: from IPython.display import Image
         Image('/home/metal-machine/Desktop/658286_99b2_2.jpg')

Out[46]:
```



0.5 At first only understand following two things:

A neural Network is collection of neurons with synapses connecting them. The collection contains three parts: One input one output and one is hidden. Hidden part can have ‘n’ number of layers.

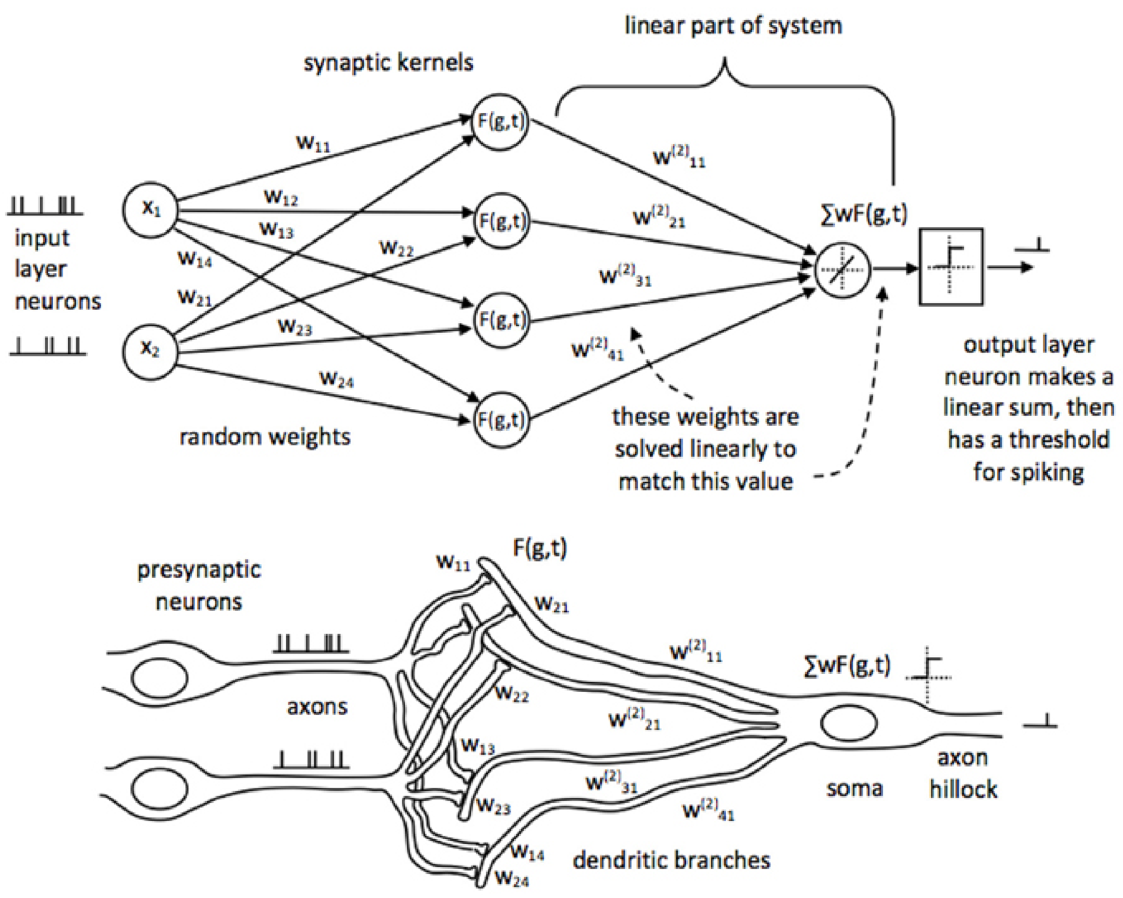
0.5.1 what is Synanpses?

Synapses are used to conneccting the Neurons(A bridge for Input-Middle-Output layer)

0.5.2 What is activation Function?

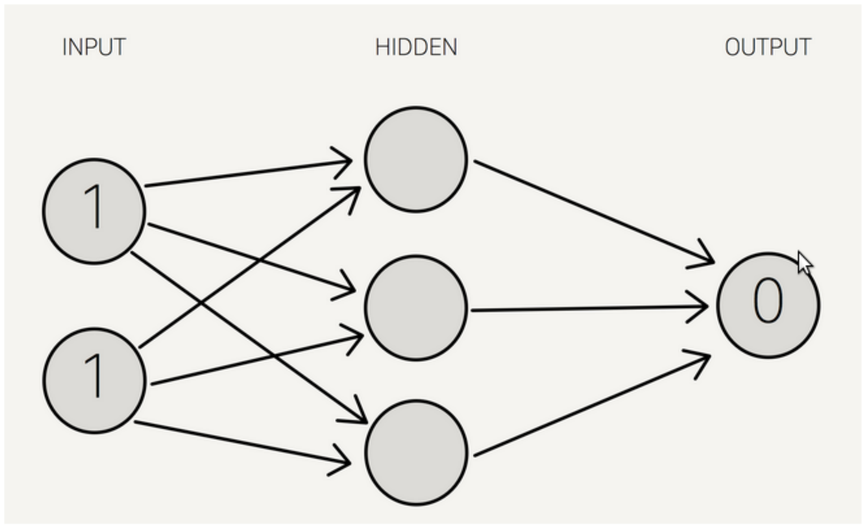
It could be really Tricky, For now just remember that Activation Function is used to convert numbers(any number) into probabilities between range 0 and 1. By limiting the range of possibilites we decrease the deviation(spread-out) of data so finding predictions are more easy.

```
In [35]: from IPython.display import Image
         Image('/home/metal-machine/Desktop/fnins-07-00153-g002.jpg')

Out[35]:
```



0.5.3 Circles represent the neurons and lines represent the synapse.

```
In [47]: import os
         from IPython.display import display,Image
         Image('/home/metal-machine/Downloads/neurals/imag1.png')

Out[47]:
```



Note that we use a single hidden layer with only three neurons for this example.

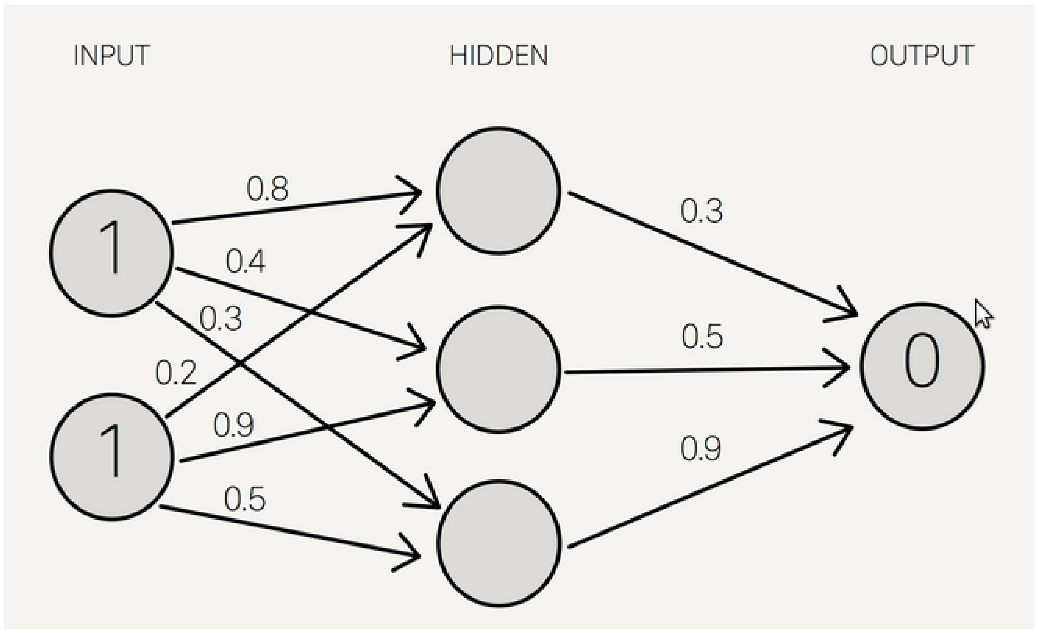
0.5.4 What term 'weight' signifies in Neural Network?

The strength of input in Determining the output. Weights are 'strength' of input by determining the output- means how input and output effect each other.

0.6 We have assigned Random weights betwwen Input – |Hidden| and Output – |Hidden|

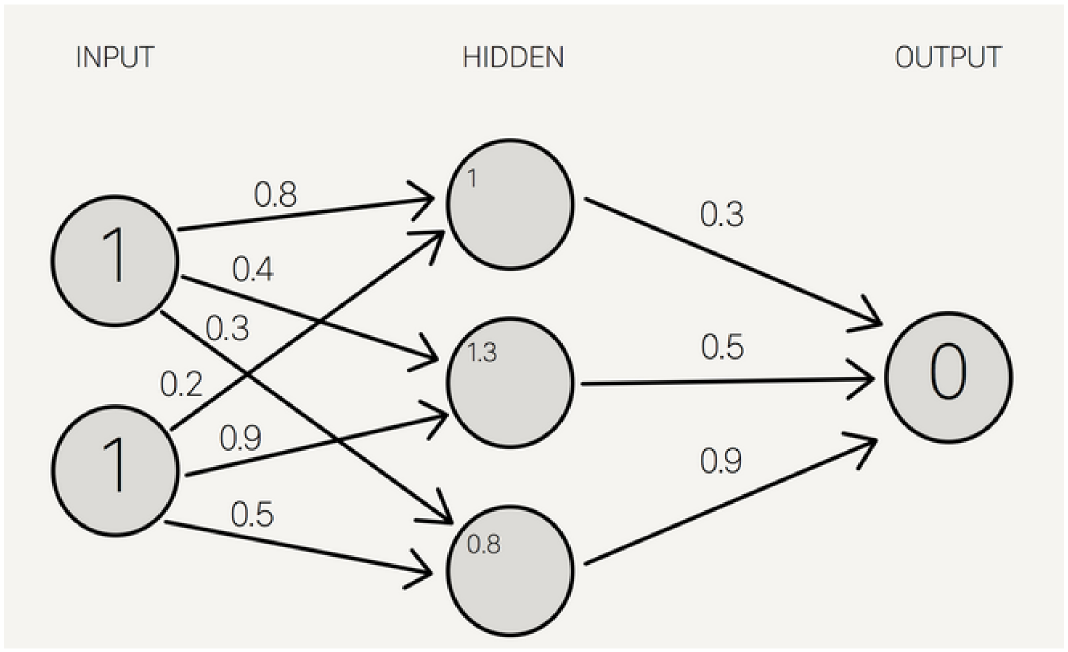
a = 1 * 0.8 + 1 * 0.2
b = 1 * 0.4 + 1 * 0.9
c = 1 * 0.3 + 1 * 0.5

```
In [12]: Image('/home/metal-machine/Downloads/neurals/image2.png')

Out[12]:
```



```
In [13]: Image('/home/metal-machine/Downloads/neurals/image3.png')
```

```
Out[13]:
```


We put these sums smaller in the circle, because they're not the final value:
INPUT
HIDDEN
OUTPUT
1
1
0.8
0.4
0.3
0.2
0.9
0.5
1
1.3
0.8
0.3
0.5
0.9
0

To get the final value, we apply the activation function to the hidden layer sums. The purpose of the activation function is to transform the input signal into an output signal and are necessary for neural networks to model complex non-linear patterns that simpler models might miss.

There are known three types of activation functions: ### linear ### sigmoid ### hyperbolic tangent

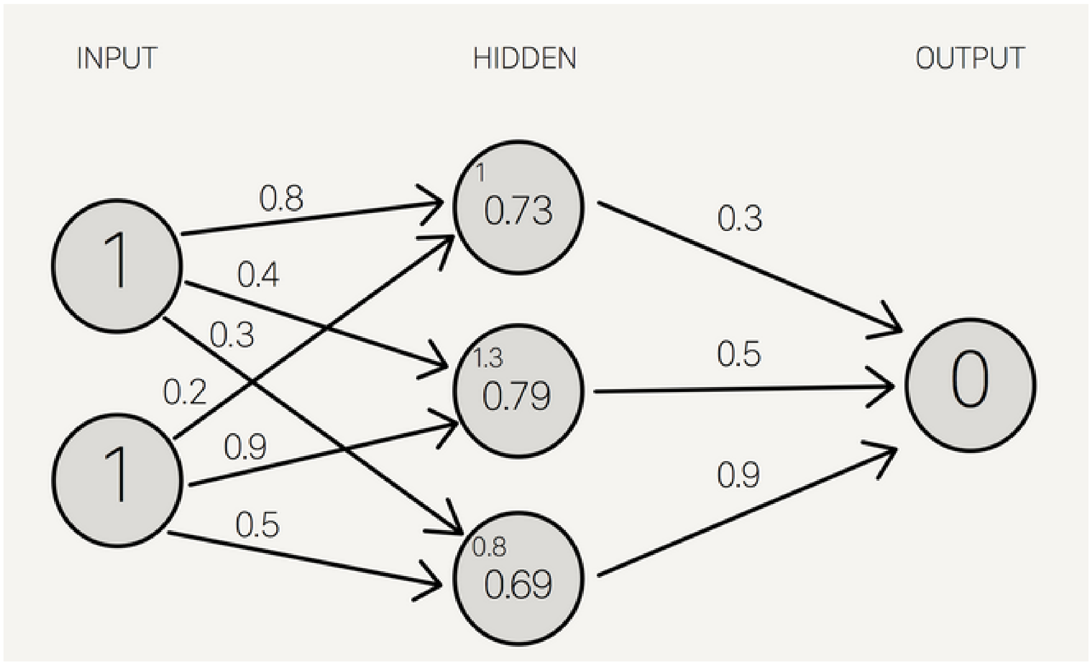
0.7 After getting results by adding random weights we apply those results to Activation function. S represents sigmoid function.

S(1.0) = 0.73105857863
S(1.3) = 0.78583498304
S(0.8) = 0.68997448112

```
In [15]: Image('/home/metal-machine/Downloads/neurals/image4.png')
```

```
Out[15]:
```

We add that to our neural network as hidden layer *results*:



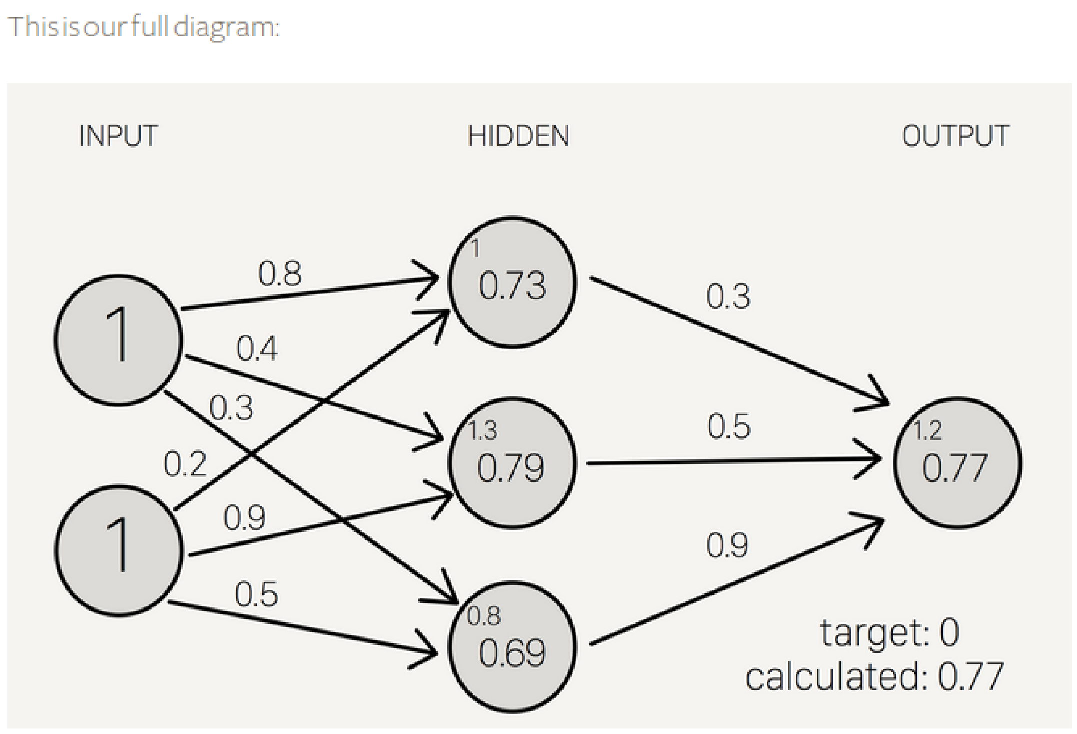
Then, we sum the product of the hidden layer results with the second set of weights (also determined at random the first time around) to determine the output sum.

0.7.1 Again Multiply the values of neurons to Random weights and applying activation Function.

0.73 * 0.3 + 0.79 * 0.5 + 0.69 * 0.9 = 1.235 ### We calculated 0.77(approx) S(1.235) = 0.7746924929149283

```
In [42]: Image('/home/metal-machine/Downloads/neurals/image6.png')
```

```
Out[42]:
```



0.8 Forward Propagation Ends here for Single node.

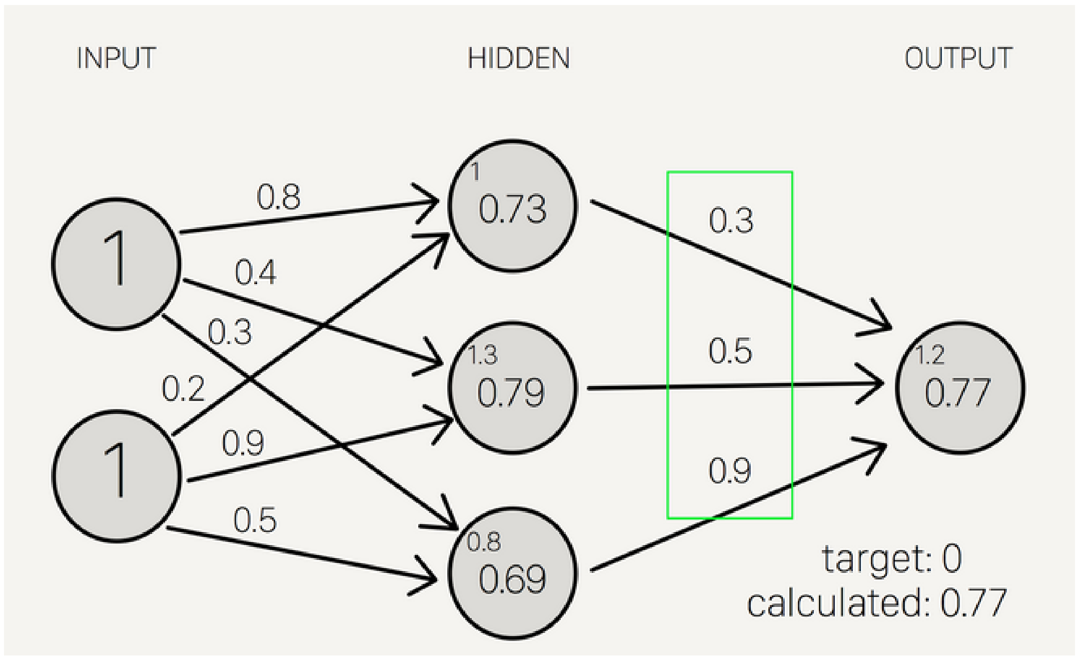
0.9 Back Propagation

To improve our model, we first have to quantify just how wrong our predictions are. Then, we adjust the weights accordingly, So that the margin of errors are decreased.

Similar to forward propagation, back propagation calculations occur at each "layer". We begin by changing the weights between the hidden layer and the output layer.

```
In [16]: Image('/home/metal-machine/Downloads/neurals/image8.png')
```

```
Out[16]:
```



Target = 0, Calculated = 0.77, Target - calculated = -0.77

0.9.1 S(sum) = result(that we got 0.77)

So the derivative of sigmoid, also known as sigmoid prime,will give us the rate of change (or "slope") of the activation function at the output sum.

Apply derivative both sides. ### S'(sum) = D(sum)/D(result)

Now change in the result is calculated as: ### Target-calculated So if we multiply (Target-calculated) on both sides of our above expression we will be able to find 'rate of change' ### delta(S) = D(sum)x(Target-calculated)/D(result) Remember carefully that (Target-Calculated) is Error in our Network.

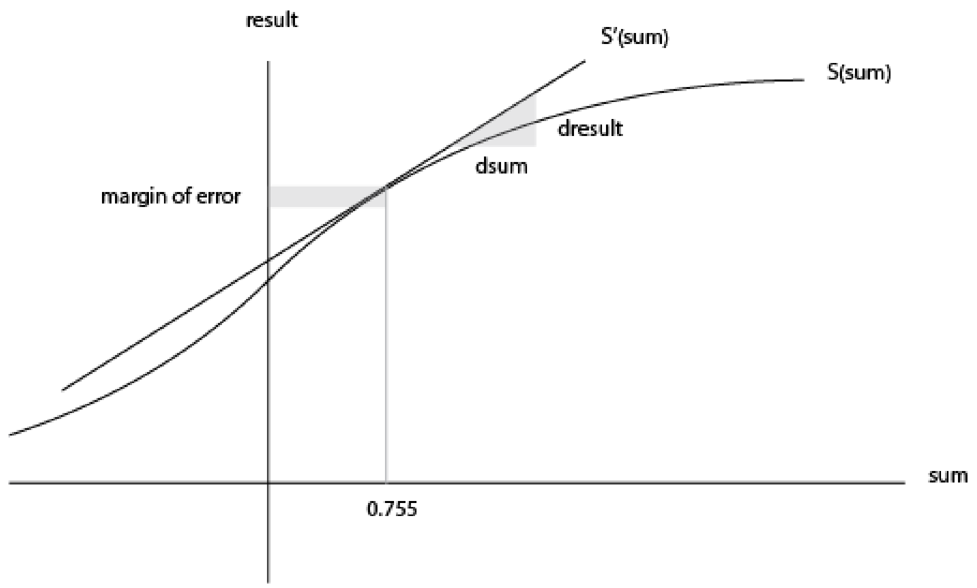
Delta output sum = S'(sum) * (output sum margin of error)

Delta output sum = S'(1.235) * (-0.77)

Delta output sum = -0.13439890643886018

```
In [17]: from IPython.display import Image
         Image('/home/metal-machine/Desktop/grPH.png')
```

Out[17]:



Now we need to understand Process of Back-Propagation. but before that we need to know how we are going to change the weights in the backpropagation.

As a reminder, the mathematical definition of the output sum is the product of the 'hidden layer result' and the 'weights' between the hidden and output layer:

result(hidden-layer) x weights = sum(output we calculated)

sum(output we calculated)/weights = result(hidden-layer)

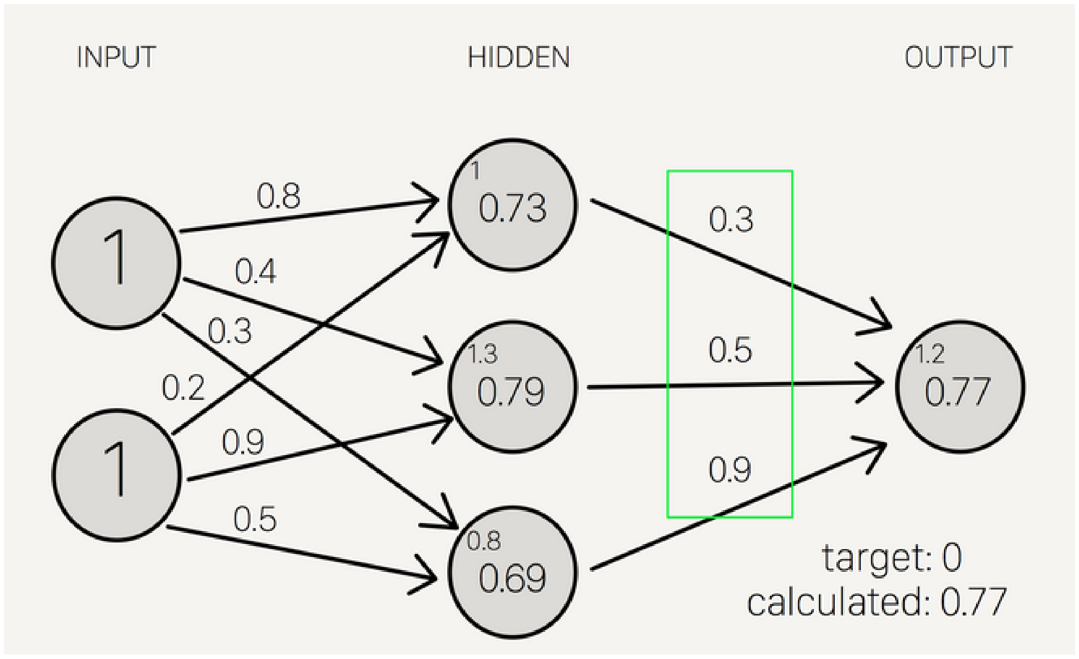
0.9.2 Applying derivative in left side

D-sum(output we calculated)/D-weights = result(hidden-layer)

0.9.3 We can also represent that

D-weights = D-sum(output we called)/result(hidden-layer)

```
In [19]: ## Now Let's again take a look at final Values again:
         from IPython.display import Image
         Image('/home/metal-machine/Downloads/neurals/image8.png')

Out[19]:
```



```
In [1]: ### Generating weights between hiddenlayer and Output layer

In [ ]: # from the ablve image we can do math as follows:
        hidden layer result1 = 0.73
        hidden layer result2= 0.79
        hidden layer result3 = 0.69

        delta-output-sum or D-sum(output-we-called) = -0.13439890643886018

        # Result We calculated: D-weights = D-sum(output we called)/result(hidden-l
        # let's insert values:

        D-weights = -0.13439890643886018/[0.73,0.79,0.69]

        D-weights = [-0.1838, -0.1710, -0.1920]

        new w7 = 0.3-0.1838=0.1162
        new w8 = 0.5-0.1710=0.329
        new w9 = 0.69-0.1920=0.708

In [2]: ## Determine weights between input and hidden layer

In [ ]: #Same as we did last time but we will use Old weights
```

result(hidden-layer) x weights = sum(output we calculated) result(hidden-layer)/sum(output we calculated) = 1/weights # derivative up and down on left side

D-result(hidden-layer)/D-sum(output we calculated) = 1/weights

D-result(hidden-layer) =D-sum(output we calculated)/weights —-Equation 1

1 Activation function

S'(hidden-sum) = D-hiddenSum/D-Hidden-result ——— Equation 2

2 Multiply Equation 1 and 2

D-result(hidden-layer) x D-hiddenSum/D-hidden-result = D-sum(output we called)/weights x D-hiddenSum/D-Hidden-result

3 D-result(hidden-layer) and D-hidden-result are same so got canceled.

D-hiddenSum = D-Sum(output we called)/Weights x S'(hidden-sum) # S'(hidden-sum) activation applied on hidden sum

3.1 Mathematical calculations

Delta hidden sum = delta output sum / hidden-to-outer weights * S'(hidden sum)

Delta hidden sum = -0.1344 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])

Delta hidden sum = [-0.448, -0.2688, -0.1493] * [0.1966, 0.1683, 0.2139]

Delta hidden sum = [-0.088, -0.0452, -0.0319]

```
In [3]: # Now we have find the delta of hidden sun now we will use this delta to fi
```

4 Same as always we will start from following equation:

weights x input = output-result

input = output-result/weights # doing derivative on neum and denom

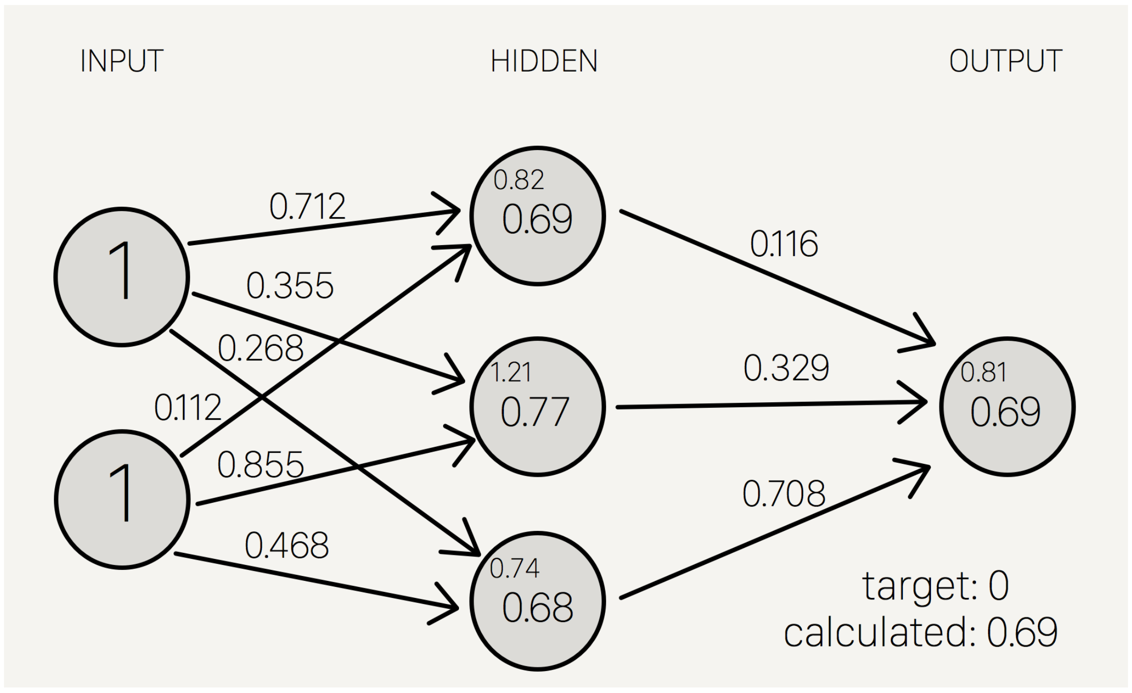
input = D(output-results)/D(weights)

D(weights) = D(output-results)/input

Once we arrive at the adjusted weights, we start again with forward propagation. When training a neural network, it is common to repeat both these processes thousands of times (by default, Mind iterates 10,000 times).

```
In [4]: from IPython.display import Image
        Image('/home/metal-machine/Desktop/mm.png')
```

```
Out[4]:
```



```
In [ ]: input 1 = 1
        input 2 = 1

        Delta weights = delta hidden sum / input data
        Delta weights = [-0.088, -0.0452, -0.0319] / [1, 1]
        Delta weights = [-0.088, -0.0452, -0.0319, -0.088, -0.0452, -0.0319]

        old w1 = 0.8
        old w2 = 0.4
        old w3 = 0.3
        old w4 = 0.2
        old w5 = 0.9
        old w6 = 0.5

        new w1 = 0.712
        new w2 = 0.3548
        new w3 = 0.2681
        new w4 = 0.112
        new w5 = 0.8548
        new w6 = 0.4681
```

```
In [ ]: l2_error = y - l2
        l2_delta = l2_error x derivative(S())

        l1_error = l2_delta x derivative(S(l1))
        l1_delta = l1
```

4.1 Use this Python Code and Play with Neural!

```
In [20]: import numpy as np

         def nonlin(x,deriv=False):
                 if(deriv==True):
                     return x*(1-x)

                 return 1/(1+np.exp(-x))

         X = np.array([[0,0,1],
                     [0,1,1],
                     [1,0,1],
                     [1,1,1]])

         y = np.array([[0],
                                 [1],
                                 [1],
                                 [0]])

         np.random.seed(1)

         # randomly initialize our Random-weights with mean 0
         syn0 = 2*np.random.random((3,4)) - 1
         syn1 = 2*np.random.random((4,1)) - 1

         for j in xrange(60000):

         # feeding Data to Layers, l0=input, l1=hidden , l2=Output

             l0 = X # l0 is input layer :)
             l1 = nonlin(np.dot(l0,syn0)) # here we are only calculating probabilit

             #np.dot multiplication of matrices(input_layer and weights)
             l2 = nonlin(np.dot(l1,syn1))

             # how much did we miss the target value? One interesting thing about r
             l2_error = y - l2

             if (j% 10000) == 0:
                 print "Error:" + str(np.mean(np.abs(l2_error)))
                 #print syn1
                 print syn0
         ### Here the Back-propagation Starts
```

```
            # in what direction is the target value? '''''''''''''Revese directi

            # were we really sure? if so, don't change too much.

            l2_delta = l2_error*nonlin(l2,deriv=True)  # calculate the delta

            # we are applying derivative=True here because now we need to get the

            # how much did each l1 value contribute to the l2 error (according to
            l1_error = l2_delta.dot(syn1.T)

            # in what direction is the target l1?
            # were we really sure? if so, don't change too much.
            l1_delta = l1_error * nonlin(l1,deriv=True)

            syn1 += l1.T.dot(l2_delta)
            syn0 += l0.T.dot(l1_delta)

Error:0.496410031903
[[-0.16595599  0.44064899 -0.99977125 -0.39533485]
 [-0.70648822 -0.81532281 -0.62747958 -0.30887855]
 [-0.20646505  0.07763347 -0.16161097  0.370439  ]]
Error:0.00858452565325
[[ 4.05751199  3.77592492 -6.07774883 -3.91512755]
 [-1.92447764 -5.52099478 -6.39028245 -3.19429746]
 [ 0.57447316 -1.83960582  2.41541223  5.23881505]]
Error:0.00578945986251
[[ 4.28478854  3.93902693 -6.17252302 -4.02803994]
 [-2.19378519 -5.64013849 -6.48112119 -3.3961458 ]
 [ 0.73577658 -1.91725339  2.46249844  5.48719508]]
Error:0.00462917677677
[[ 4.40687311  4.02794026 -6.22460222 -4.09173523]
 [-2.3423839  -5.70613025 -6.52994552 -3.50615882]
 [ 0.82632219 -1.95916572  2.48799085  5.62311269]]
Error:0.00395876528027
[[ 4.48957285  4.08879708 -6.26041202 -4.13605829]
 [-2.44446319 -5.75170217 -6.56306645 -3.58137163]
 [ 0.88902782 -1.98777554  2.50547158  5.71628306]]
Error:0.00351012256786
[[ 4.55174021  4.13491648 -6.2876362  -4.16999257]
 [-2.52190782 -5.78644401 -6.58800254 -3.63829223]
 [ 0.93683818 -2.00944394  2.51876888  5.78693669]]
```