In [1]:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]))
y = np.array(([92], [86], [89]))

y = y/100 # max test score is 100



def sigmoid(x):
    return 1/(1 + np.exp(-x))


def derivatives_sigmoid(x):
    return x * (1 - x)


epoch=10000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1


wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bias_hidden=np.random.uniform(size=(1,hiddenlayer_neurons))
weight_hidden=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bias_output=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp= hinp1 + bias_hidden
    hlayer_activation = sigmoid(hinp)

    outinp1=np.dot(hlayer_activation,weight_hidden)
    outinp= outinp1+ bias_output
    output = sigmoid(outinp)


    EO = y-output

    outgrad = derivatives_sigmoid(output)

    d_output = EO * outgrad

    EH = d_output.dot(weight_hidden.T)

    hiddengrad = derivatives_sigmoid(hlayer_activation)
    d_hiddenlayer = EH * hiddengrad


    #update the weights
    weight_hidden += hlayer_activation.T.dot(d_output) *lr
    bias_hidden += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

    wh += X.T.dot(d_hiddenlayer) *lr
    bias_output += np.sum(d_output, axis=0,keepdims=True) *lr

print("Input: \n" + str(X))
```

```
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[2 9]
 [1 5]
 [3 6]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89338515]
 [0.87968619]
 [0.89543673]]
```

In [ ]: