```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

X = X / np.amax(X, axis=0)  # maximum of X array longitudinally
y = y / 100

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

# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Variable initialization
epoch = 5000  # Setting training iterations
lr = 0.1  # Setting learning rate
inputlayer_neurons = 2  # number of features in the data set
hiddenlayer_neurons = 3  # number of hidden layer neurons
output_neurons = 1  # number of neurons at the output layer

# weight and bias initialization
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))  # 2,3
bh = np.random.uniform(size=(1, hiddenlayer_neurons))  # 1,3
wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))  # 3,1
bout = np.random.uniform(size=(1, output_neurons))  # 1,1

for i in range(epoch):
    # Forward Propagation
    hinp = np.dot(X, wh) + bh
    hlayer_act = sigmoid(hinp)  # HIDDEN LAYER ACTIVATION FUNCTION
    outinp = np.dot(hlayer_act, wout) + bout
    output = sigmoid(outinp)
    outgrad = derivatives_sigmoid(output)
    hiddengrad = derivatives_sigmoid(hlayer_act)

    EO = y - output  # ERROR AT OUTPUT LAYER
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)  # ERROR AT HIDDEN LAYER (TRANSPOSE => COZ
REVERSE(BACK))
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) * lr  # REMEMBER WOUT IS 3*1 MATRIX
    wh += X.T.dot(d_hiddenlayer) * lr
```

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