Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)      # X = (hours
sleeping, hours studying)
y = np.array(([92], [86], [89]), dtype=float)            # y = score on
test

# scale units
X = X/np.amax(X, axis=0)         # maximum of X array
y = y/100                        # max test score is 100
class Neural_Network(object):
    def __init__(self):
                              # Parameters
        self.inputSize = 2
        self.outputSize = 1
        self.hiddenSize = 3
                               # Weights
        self.W1 = np.random.randn(self.inputSize,
self.hiddenSize)         # (3x2) weight matrix from input to hidden
layer
        self.W2 = np.random.randn(self.hiddenSize,
self.outputSize)        # (3x1) weight matrix from hidden to output
layer

    def forward(self, X):
                               #forward propagation through our network
        self.z = np.dot(X, self.W1)                # dot product of X
(input) and first set of 3x2 weights
        self.z2 = self.sigmoid(self.z)             # activation function
        self.z3 = np.dot(self.z2, self.W2)         # dot product of
hidden layer (z2) and second set of 3x1 weights
        o = self.sigmoid(self.z3)                  # final activation
function
        return o

    def sigmoid(self, s):
        return 1/(1+np.exp(-s))      # activation function

    def sigmoidPrime(self, s):
        return s * (1 - s)           # derivative of sigmoid
```

```
    def backward(self, X, y, o):
                                        # backward propgate through the
network
        self.o_error = y - o        # error in output
        self.o_delta = self.o_error*self.sigmoidPrime(o) # applying
derivative of sigmoid to
        self.z2_error = self.o_delta.dot(self.W2.T)    # z2 error: how
much our hidden layer weights contributed to output error
        self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) #
applying derivative of sigmoid to z2 error
        self.W1 += X.T.dot(self.z2_delta)       # adjusting first set
(input --> hidden) weights
        self.W2 += self.z2.T.dot(self.o_delta)  # adjusting second set
(hidden --> output) weights

    def train (self, X, y):
        o = self.forward(X)
        self.backward(X, y, o)
NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
    print ("\nInput: \n" + str(X))
    print ("\nActual Output: \n" + str(y))
    print ("\nPredicted Output: \n" + str(NN.forward(X)))
    print ("\nLoss: \n" + str(np.mean(np.square(y -
NN.forward(X)))))     # mean sum squared loss)
    NN.train(X, y)
```