Ann 3

import numpy as np

# X = (hours sleeping, hours studying), y = test score of the student

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)

y = y/100

class NeuralNetwork(object):

def \_\_init\_\_(self):

self.inputSize = 2

self.outputSize = 1

self.hiddenSize = 3

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 feedForward(self, X):

self.z = np.dot(X, self.W1)

self.z2 = self.sigmoid(self.z)

self.z3 = np.dot(self.z2, self.W2)

output = self.sigmoid(self.z3)

return output

def sigmoid(self, s, deriv=False):

if (deriv == True):

return s \* (1 - s)

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

def backward(self, X, y, output):

self.output\_error = y - output

self.output\_delta = self.output\_error \* self.sigmoid(output, deriv=True)

self.z2\_error = self.output\_delta.dot(self.W2.T)

self.z2\_delta = self.z2\_error \* self.sigmoid(self.z2, deriv=True)

self.W1 += X.T.dot(self.z2\_delta)

self.W2 += self.z2.T.dot(self.output\_delta)

def train(self, X, y):

output = self.feedForward(X)

self.backward(X, y, output)

NN = NeuralNetwork()

for i in range(1000):

if (i % 100 == 0):

print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))

NN.train(X, y)

print("Input: " + str(X))

print("Actual Output: " + str(y))

print("Loss: " + str(np.mean(np.square(y - NN.feedForward(X)))))

print("\n")

print("Predicted Output: " + str(NN.feedForward(X)))