 PROGRAM 5 : Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

|  |
| --- |
| **import** **numpy** **as** **np**  *# X = (hours studying, hours sleeping), y = score on test, xPredicted = 4 hours studying & 8 hours sleeping (input data for prediction)*  X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)  y = np.array(([92], [86], [89]), dtype=float)  xPredicted = np.array(([4,8]), dtype=float)  X = X/np.amax(X, axis=0)  xPredicted = xPredicted/np.amax(xPredicted, axis=0)  y = y/100 *# max test score is 100*  **class** **Neural\_Network**(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** forward(self, X):  self.z = np.dot(X, self.W1)  self.z2 = self.sigmoid(self.z) *# activation function*  self.z3 = np.dot(self.z2, self.W2) *# dot product of hidden layer (z2)andsecondsetof 3x1 weights*  o = self.sigmoid(self.z3) *# final activation function*  **return** o  **def** sigmoid(self, s):  **return** 1/(1+np.exp(-s))  **def** sigmoidPrime(self, s):  **return** s \* (1 - s)    **def** backward(self, X, y, o):  self.o\_error = y - o *# error in output*  self.o\_delta = self.o\_error\*self.sigmoidPrime(o) *# applying derivative of sigmoid to error*  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)    **def** saveWeights(self):  np.savetxt("w1.txt", self.W1, fmt="**%s**")  np.savetxt("w2.txt", self.W2, fmt="**%s**")  **def** predict(self):  print("Predicted data based on trained weights: ")  print("Input (scaled): **\n**" + str(xPredicted))  print("Output: **\n**" + str(self.forward(xPredicted)))  NN = Neural\_Network()  **for** i **in** range(1000): *# trains the NN 1,000 times*  print("# " + str(i) + "**\n**")  print("Input (scaled): **\n**" + str(X))  print("Actual Output: **\n**" + str(y))  print("Predicted Output: **\n**" + str(NN.forward(X)))  print("Loss: **\n**" + str(np.mean(np.square(y - NN.forward(X))))) *# mean sum squared loss*  print("**\n**")    NN.train(X, y)  NN.saveWeights()  NN.predict() |