Name : Jinal Shah

UID : 2019230070

Batch: C

Course code: OE5 (Fundamentals of Computational Intelligence (FCI))

EXPERIMENT NO. 1

AIM: Experiment on Supervised Learning (Back Propagation Neural network)

THEORY:

What is Artificial Neural Networks?

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.

What is Backpropagation?

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

Why We Need Backpropagation?

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Backpropagation Key Points:

- Simplifies the network structure by elements weighted links that have the least effect on the trained network
- You need to study a group of input and activation values to develop the relationship between the input and hidden unit layers.
- It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.

•Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
•Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.

Disadvantages of using Backpropagation:

•The actual performance of backpropagation on a specific problem is dependent on the input data.
•Backpropagation can be quite sensitive to noisy data
•You need to use the matrix-based approach for backpropagation instead of mini-batch.

PROCEDURE:

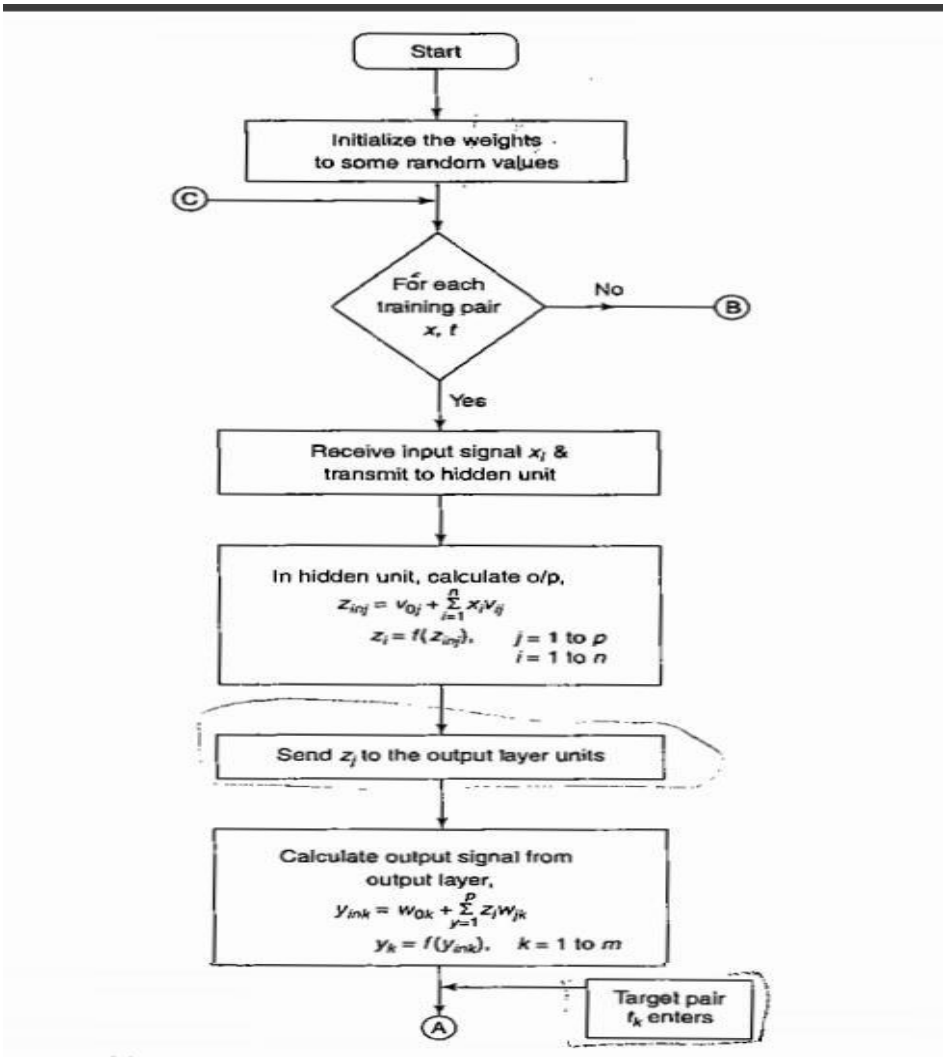


Figure 3-10 Flowchart for back-propagation network training.

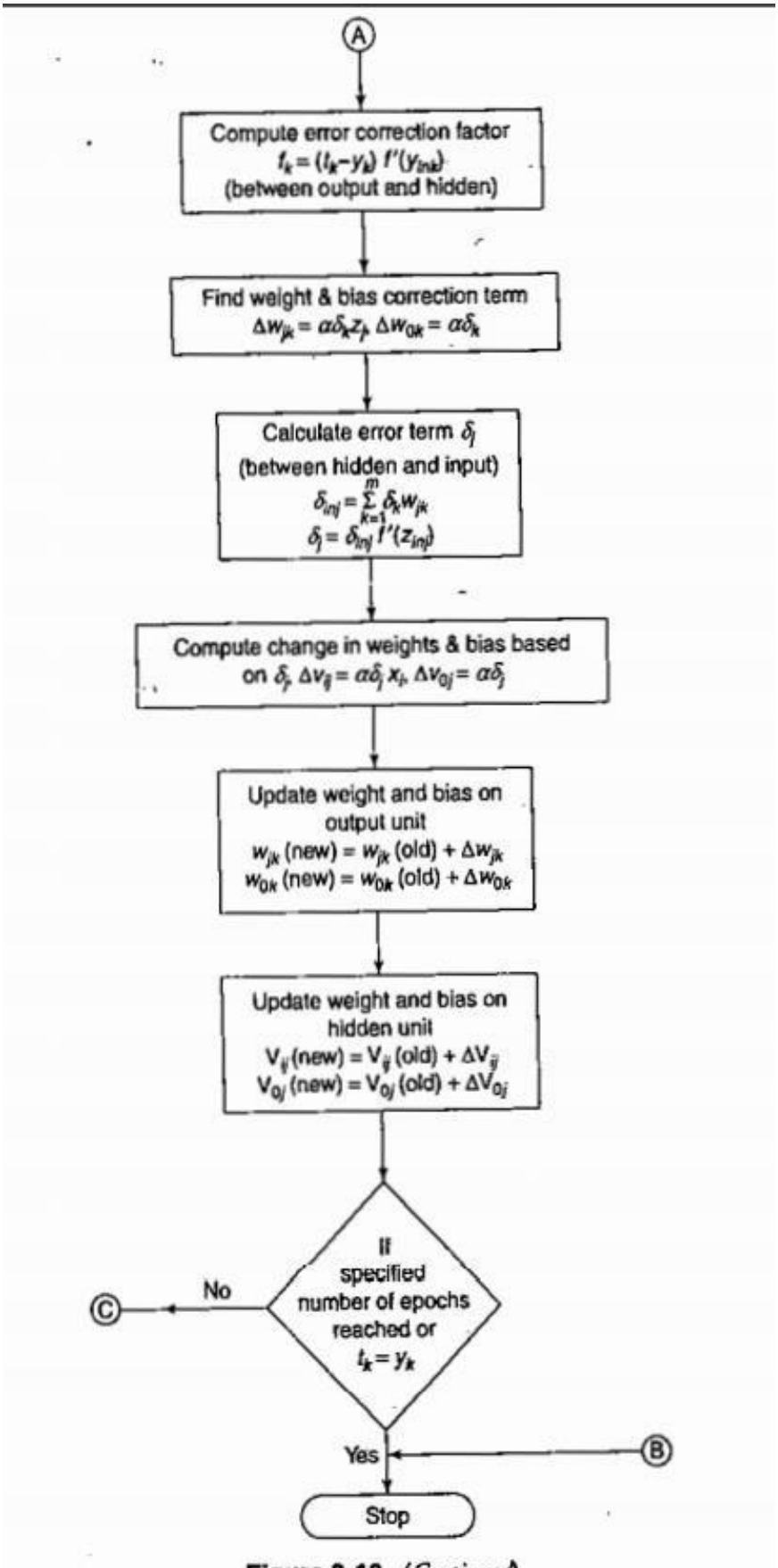
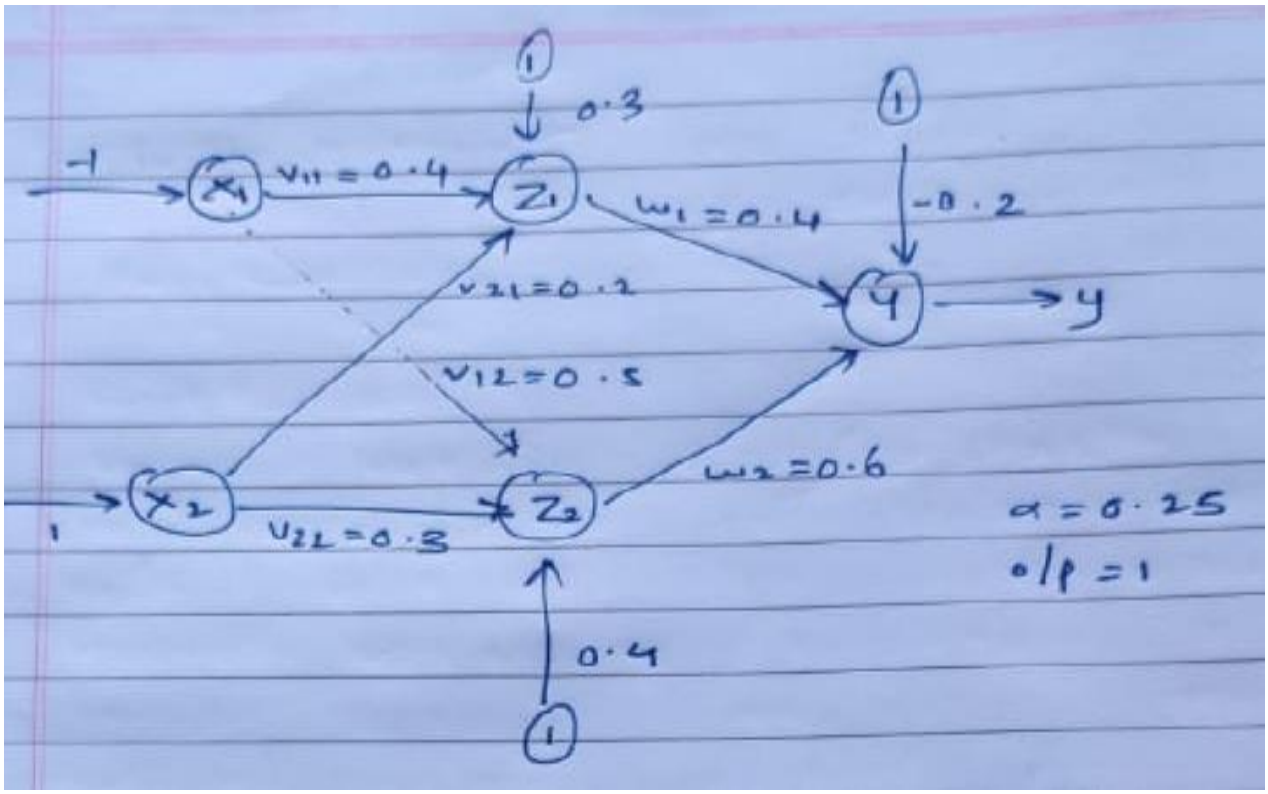


Figure 3-10 (*Continued*).

REPRESENTATION THROUGH AN EXAMPLE



CODE:

```
import numpy as np
from prettytable import PrettyTable

def initialise_network(input_layer,hidden_layer,output_layer):

    #weights between hidden and input layer
    V = np.array([])
    for i in range(hidden_layer):
        for j in range(input_layer):
            node = float(input('V'+str(i+1)+str(j+1)+': '))
            V = np.append(V,node)
    V = V.reshape(hidden_layer,input_layer)

    #bias weights on hidden layer
    B1 = np.array([])
    for i in range(hidden_layer):
        node = float(input('B1'+str(i+1)+': '))
        B1 = np.append(B1,node)
    B1 = B1.reshape(hidden_layer,1)

    #weights between hidden and output layer
```

```
    W = np.array([])
    for i in range(output_layer):
        for j in range(hidden_layer):
            node = float(input('W'+str(i+1)+str(j+1)+': '))
            W = np.append(W,node)
    W = W.reshape(output_layer,hidden_layer)

    #bias weights on output layer
    B2 = np.array([])
    for i in range(output_layer):
        node = float(input('B2'+str(i+1)+': '))
        B2 = np.append(B2,node)
    B2 = B2.reshape(output_layer,1)

    return V,B1,W,B2

def forward_propagation(X,V,B1,W,B2):
    #calculating the z1 of hidden layer units
    Z1 = np.dot(V.T, X) + B1
    A1 = sigmoid(Z1)

    #calculating the predicted output using activation function
    Z2 = np.dot(W, A1) + B2
    Y = sigmoid(Z2)

    return Z1,A1,Z2,Y

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

#derivative of activation function
def sigmoid_derivative(z):
    return sigmoid(z)*(1-sigmoid(z))

def cost(Y, y):
    #computing the error function
    cost = (y-Y)*Y*(1-Y)

    output_values.append(list(Y)[0])
    error_values.append(list(cost)[0])

    return cost

def back_propagation(cost,V,W,B1,B2,Z1,Z2,A1,X,alpha):

    #calculating change in weights between hidden layer and output layer
    delta_W = (np.multiply(alpha*cost,A1)).T
```

```
    #calculating change in bias weights on output layer
    delta_B2 = alpha*cost

    #calculating change in weights between input and hidden layer
    X = X.T
    X = np.concatenate([X,X], axis=0)

    delta_h=np.multiply((cost*W).T,A1*(1-A1))

    delta_V = np.multiply(alpha*X,delta_h).T

    #calculating change in bias weights on hidden layer
    delta_B1 = alpha*np.multiply((cost*W).T,A1*(1-A1))

    #updating the new weights
    new_V = V + delta_V
    new_W = W + delta_W
    new_B1 = B1 + delta_B1
    new_B2 = B2 + delta_B2

    return new_V,new_B1,new_W,new_B2

if __name__ == "__main__":

    input_layer=int(input("Input Units: "))
    hidden_layer=int(input("Hidden Units: "))
    output_layer=int(input("Output Units: "))

    output_values = []
    error_values = []

    #input for input matrix X
    X = np.array([])
    for i in range(input_layer):
        p = float(input('X'+str(i+1)+': '))
        X = np.append(X,p)
    X = X.reshape(input_layer,1)

    #input for target value
    y = int(input('Target Value: '))

    #input for the learning rate
    learning_rate=float(input('Learning rate: '))

    #initializing the network
    V,B1,W,B2 = initialise_network(input_layer,hidden_layer,output_layer)
    Y = np.array([0])

    n = 1
```

```
while(n<=500):
    #finding the predicted output value
    Z1,A1,Z2,Y = forward_propagation(X,V,B1,W,B2)
    c = cost(Y, y)
    #Calculating the updated weights
    V,B1,W,B2 = back_propagation(c,V,W,B1,B2,Z1,Z2,A1,X,learning_rate)

    if (n==1):
        print('\nEpoch 1: ')
        print('\nV: ',V)
        print('B1: ',B1)
        print('W: ',W)
        print('B2: ',B2)
        print('Z1: ',Z1)
        print('Z2: ',Z2)

    if (n==2):
        print('\nEpoch 2: ')
        print('\nV: ',V)
        print('B1: ',B1)
        print('W: ',W)
        print('B2: ',B2)
        print('Z1: ',Z1)
        print('Z2: ',Z2)

    if (n==3):
        print('\nEpoch 3: ')
        print('\nV: ',V)
        print('B1: ',B1)
        print('W: ',W)
        print('B2: ',B2)
        print('Z1: ',Z1)
        print('Z2: ',Z2)

    if (n==4):
        print('\nEpoch 4: ')
        print('\nV: ',V)
        print('B1: ',B1)
        print('W: ',W)
        print('B2: ',B2)
        print('Z1: ',Z1)
        print('Z2: ',Z2)

    n = n+1

print('\nEpoch 500: ')
print('\nV: ',V)
print('B1: ',B1)
```

```
print('W: ',W)
print('B2: ',B2)
print('Z1: ',Z1)
print('Z2: ',Z2)

print('\nEpoch Table: ')


#Output in tabular form
table = PrettyTable(['Epoch', 'Output', 'Error Values'])
for i in range(len(output_values)):
    if(i>3 and i<495):
        continue
    table.add_row([(i+1),output_values[i],error_values[i]])

print(table)
```

OUTPUT:

```
Input Units: 2
Hidden Units: 2
Output Units: 1
X1: -1
X2: 1
Target Value: 1
Learning rate: 0.25
V11: 0.4
V12: 0.5
V21: 0.2
V22: 0.3
B11: 0.3
B12: -0.4
W11: 0.4
W12: 0.6
B21: -0.2

Epoch 1:

V:  [[0.39726237 0.49623263]
 [0.20273763 0.30376737]]
B1:  [[ 0.30273763]
 [-0.39623263]]
W:  [[0.41440796 0.6097249 ]]
B2:  [[-0.17255518]]
Z1:  [[ 0.1]
 [-0.6]]
Z2:  [[0.22259789]]
```

```
  Epoch 2:

  V:  [[0.39450522 0.49249721]
   [0.20549478 0.30750279]]
  B1:  [[ 0.30549478]
   [-0.39249721]]
  W:  [[0.42847476 0.61925176]]
  B2:  [[-0.14586432]]
  Z1:  [[ 0.1082129]
   [-0.5886979]]
  Z2:  [[0.26348029]]

  Epoch 3:

  V:  [[0.39173569 0.48879805]
   [0.20826431 0.31120195]]
  B1:  [[ 0.30826431]
   [-0.38879805]]
  W:  [[0.44220067 0.62857838]]
  B2:  [[-0.11992175]]
  Z1:  [[ 0.11648434]
   [-0.57749164]]
  Z2:  [[0.3034638]]



Epoch 4:

V:  [[0.38896033 0.48513894]
 [0.21103967 0.31486106]]
B1:  [[ 0.31103967]
 [-0.38513894]]
W:  [[0.45558738 0.63770357]]
B2:  [[-0.09471887]]
Z1:  [[ 0.12479292]
 [-0.56639414]]
Z2:  [[0.3425455]]

Epoch 500:

V:  [[0.11572776 0.16848587]
 [0.48427224 0.63151413]]
B1:  [[ 0.58427224]
 [-0.06848587]]
W:  [[1.31288807 1.28358863]]
B2:  [[1.29342925]]
Z1:  [[0.95239805]
 [0.39405228]]
Z2:  [[3.0064453]]
```

Epoch Table:

Epoch	Output	Error Values
1	[0.55542082]	[0.10977928]
2	[0.56549163]	[0.10676342]
3	[0.57528905]	[0.10377029]
4	[0.58480872]	[0.10081155]
496	[0.95263717]	[0.00213699]
497	[0.95269428]	[0.00213197]
498	[0.95275119]	[0.00212697]
499	[0.95280792]	[0.00212199]
500	[0.95286446]	[0.00211704]

CONCLUSION:

In this experiment I learnt about back propagation algorithm using python. This algorithm updates the weights of the neural network by calculating the derivative of error multiplied by learning rate and helps in reducing the cost function. It thus proves beneficial in minimizing the error occurred and how every node is responsible for the amount of loss that occurs throughout the epochs. Its efficiency makes it feasible to use this algorithm for training multi-layer networks.