Batch: SC_1 **Roll No.: 1911027**

Experiment No. : 06

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Back propagation algorithm

Objective: To Write a program to implement back propagation algorithm in MLP.

Expected Outcome of Experiment:

CO3 : Understand perceptron's and counter propagation networks

Books/ Journals/ Websites referred:

- J.S.R.Jang, C.T.Sun and E.Mizutani, "Neuro-Fuzzy and Soft Computing", PHI, 2004, Pearson Education 2004.
- Davis E.Goldberg, "Genetic Algorithms: Search, Optimization and Machine Learning", Addison Wesley, N.Y., 1989.
- S. Rajasekaran and G.A.V.Pai, "Neural Networks, Fuzzy Logic and Genetic Algorithms", PHI, 2003.
- http://library.thinkquest.org/C007395/tqweb/history.html.

Pre Lab/ Prior Concepts:

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features-

1) A set of processing units;
2) An activation state for each unit, which is equivalent to the output of the unit;
3) Connections between the units. Generally each connection is defined by a weight w_{jk} that determines the effect that the signal of unit *j* has on unit *k;*
4) A propagation rule, which determines the effective input of the unit from its external inputs;

5) An activation function, which determines the new level of activation based on the effective input and the current activation;
6) An external input (bias, offset) for each unit;
7) A method for information gathering (learning rule);
8) An environment within which the system can operate, provide input signals and, if necessary, error signals.

Implementation Details :

The Back propagation algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).

• Back propagation algorithm consists of the following two passes:

– **Forward pass**:

In this step the network is activated on one example and the error of (each neuron of) the output layer is computed.

– **Backward pass**:

In this step the network error is used for updating the weights. This process is more complex than the LMS algorithm for Adaline because hidden nodes are linked to the error not directly but by means of the nodes of the next layer. Therefore, starting at

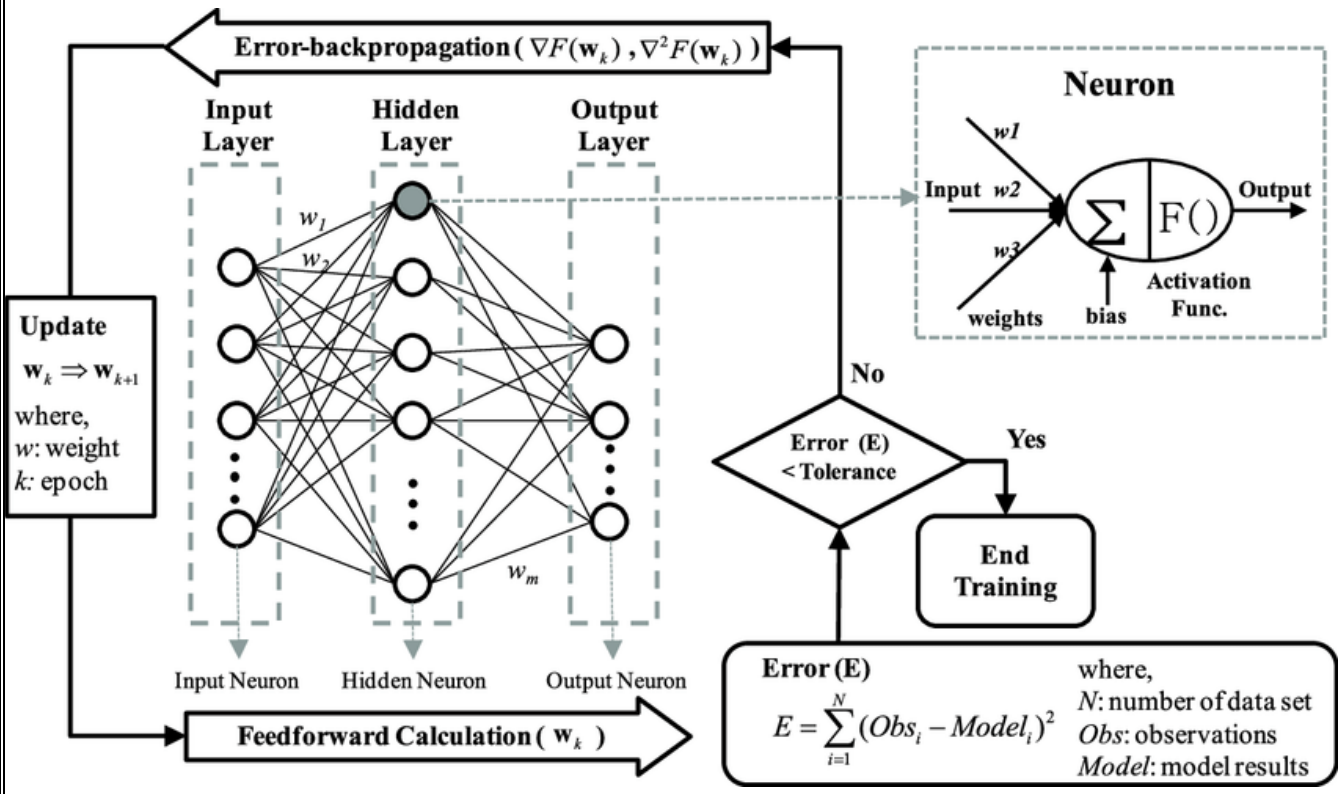
the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each weight.

Back-propagation training algorithm :

Backpropagation (backward propagation) is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. Essentially, backpropagation is an algorithm used to calculate derivatives quickly. Artificial neural networks use backpropagation as a learning algorithm to compute a gradient descent with respect to weights. Desired outputs are compared to achieved system outputs, and then the systems are tuned by adjusting connection weights to narrow the difference between the two as much as possible. The algorithm gets its name because the weights are updated backwards, from output towards input. The difficulty of understanding exactly how changing weights and biases affects the overall behavior of an artificial neural network was one factor that held back wider application of neural network applications. Because backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient, it is usually classified as a type of

supervised machine learning. The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

Network Architecture :



Algorithm: (Backpropogation)

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

   ErrorB= Actual Output – Desired Output
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

6. Keep repeating the process until the desired output is achieved

Implementation details:

Code:

```
import math
def calnetinp(b,x1,v1,x2,v2):
    return b+x1*v1+x2*v2

def calfnet(zin):
    return 1/(1+math.exp(-1*zin))

def calfdashnet(y):
    return y*(1-y)

def calerrpor(t,y,f):
    return (t-y)*f

def updatew(alp,th,z):
    return alp*th*z

def caldelinptohid(dele,wei):
    return dele*wei

print("------------------------------------------------------------------------------------
-------------------------------------------------------------------")
print("-------------------------------------------------------------------------EBPTA--------
-------------------------------------------------------------------")
print("------------------------------------------------------------------------------------
-------------------------------------------------------------------")
ip=input("Enter input: ").split()
print("------------------------------------------------------------------------------------
-------------------------------------------------------------------")
inp=[]
for i in ip:
    inp.append(float(i))
alpha=input("Enter learning rate: ")
print("------------------------------------------------------------------------------------
-------------------------------------------------------------------")
alpha=float(alpha)
weights=input("Enter weights associated with neuron 1 (from input to hidden v11 v21
v0): ").split()
```

```
print("-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
v1=list()
for i in weights:
    v1.append(float(i))
weights=input("Enter weights associated with neuron 2 (from input to hidden layer v12
v22 v1): ").split()
print("-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
v2=list()
for i in weights:
    v2.append(float(i))
weights=input("Enter weights associated with output neuron (from hidden to output layer
w1 w2 w0): ").split()
print("-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
w=list()
for i in weights:
    w.append(float(i))
t=float(input("Enter target: "))
print("-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
zin=list()
v=list()
v.append(v1)
v.append(v2)
for i in range(0,len(inp)):
    temp=calnetinp(v[i][2],inp[0],v[i][0],inp[1],v[i][1])
    zin.append(temp)
print("Net input for hidden layer neurons: ",end=" ")
for i in zin:
    print(i,end=" ")
print("\n-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
z=list()
for i in zin:
    temp=calfnet(i)
    z.append(temp)
print("Output from hidden layer neuron: ",end=" ")
for i in z:
    print(i,end=" ")
```

```
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
yin=calnetinp(w[2],z[0],w[0],z[1],w[1])
print("Net input for output layer: ",yin)
print("-------------------------------------------------------------------------------
-------------------------------------------------------------------")
y=calfnet(yin)
print("Output from output layer: ",y)
print("-------------------------------------------------------------------------------
-------------------------------------------------------------------")
fdashyin=calfdashnet(y)
delerr=calerrpor(t,y,fdashyin)
print("Error (between output to hidden layer): ",delerr)
print("-------------------------------------------------------------------------------
-------------------------------------------------------------------")
delw=list()
for i in range(0,len(w)):
    if(i==len(w)-1):
        temp=updatew(alpha,delerr,1)
    else:
        temp=updatew(alpha,delerr,z[i])
    delw.append(temp)
print("Weight change (between output and hidden layer (delw1 delw2 delw0)): ",end=" ")
for i in delw:
    print(i,end=" ")
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
delhidinp=list()
for i in range(0,len(inp)):
    temp=caldelinptohid(delerr,w[i])
    delhidinp.append(temp)
final_del=list()
for i in range(0,len(delhidinp)):
    temp=calfdashnet(z[i])
    temp2=delhidinp[i]*temp
    final_del.append(temp2)
print("Error (Between hidden layer and input layer): ",end=" ")
for i in final_del:
    print(i,end=" ")
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
delv1=list()
```

```
for i in range(0,len(v1)):
    if(i==len(v1)-1):
        temp=updatew(alpha,final_del[0],1)
    else:
        temp=updatew(alpha,final_del[0],inp[i])
    delv1.append(temp)
print("Weight change (between hidden and input layer (delv11 delv21 delv0)): ",end=" ")
for i in delv1:
    print(i,end= " ")
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
delv2=list()
for i in range(0,len(v2)):
    if(i==len(v2)-1):
        temp=updatew(alpha,final_del[1],1)
    else:
        temp=updatew(alpha,final_del[1],inp[i])
    delv2.append(temp)
print("Weight change (between hidden and input layer (delv12 delv22 delv1)): ",end=" ")
for i in delv2:
    print(i,end= " ")
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
final_v1=list()
for i in range(0,len(delv1)):
    temp=v1[i]+delv1[i]
    final_v1.append(temp)
print("Weight updation (between input and hidden layer (v11 v21 v0))(vnew = vold +
delv): ",end=" ")
for i in final_v1:
    print(i,end= " ")
print("\n-------------------------------------------------------------------------------
-------------------------------------------------------------------")
final_v2=list()
for i in range(0,len(delv2)):
    temp=v2[i]+delv2[i]
    final_v2.append(temp)
print("Weight updation (between input and hidden layer (v12 v22 v1))(vnew = vold +
delv): ",end=" ")
for i in final_v2:
    print(i,end= " ")
```

```
print("\n-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
final_w=list()
for i in range(0,len(delw)):
    temp=w[i]+delw[i]
    final_w.append(temp)
print("Weight updation (between hidden and output layer (w1 w2 w0))(wnew = wold +
delw): ",end=" ")
for i in final_w:
    print(i,end= " ")
print("\n-------------------------------------------------------------------------------------
-------------------------------------------------------------------")
```

Output:

```
-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------EBPTA-------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------
Enter input: 0 1
-------------------------------------------------------------------------------------------------------------------------------------------
Enter learning rate: 0.25
-------------------------------------------------------------------------------------------------------------------------------------------
Enter weights associated with neuron 1 (from input to hidden v11 v21 v0): 0.6 -0.1 0.3
-------------------------------------------------------------------------------------------------------------------------------------------
Enter weights associated with neuron 2 (from input to hidden layer v12 v22 v1): -0.3 0.4 0.5
-------------------------------------------------------------------------------------------------------------------------------------------
Enter weights associated with output neuron (from hidden to output layer w1 w2 w0): 0.4 0.1 -0.2
-------------------------------------------------------------------------------------------------------------------------------------------
Enter target: 1
-------------------------------------------------------------------------------------------------------------------------------------------
Net input for hidden layer neurons:  0.19999999999999998 0.9
-------------------------------------------------------------------------------------------------------------------------------------------
Output from hidden layer neuron:  0.549833997312478 0.7109495026250039
-------------------------------------------------------------------------------------------------------------------------------------------
Net input for output layer:  0.09102854918749158
-------------------------------------------------------------------------------------------------------------------------------------------
Output from output layer:  0.5227414361305817
-------------------------------------------------------------------------------------------------------------------------------------------
Error (between output to hidden layer):  0.11906781576358075
-------------------------------------------------------------------------------------------------------------------------------------------
Weight change (between output and hidden layer (delw1 delw2 delw0)):  0.01636688327313882 0.02116280109894083 0.029766953940895187
-------------------------------------------------------------------------------------------------------------------------------------------
Error (Between hidden layer and input layer):  0.011788503071235473 0.002446847273398785
-------------------------------------------------------------------------------------------------------------------------------------------
Weight change (between hidden and input layer (delv11 delv21 delv0)):  0.0 0.0029471257678088682 0.0029471257678088682
-------------------------------------------------------------------------------------------------------------------------------------------
Weight change (between hidden and input layer (delv12 delv22 delv1)):  0.0 0.0006117118183496963 0.0006117118183496963
-------------------------------------------------------------------------------------------------------------------------------------------
Weight updation (between input and hidden layer (v11 v21 v0))(vnew = vold + delv):  0.6 -0.09705287423219114 0.30294712576780886
-------------------------------------------------------------------------------------------------------------------------------------------
Weight updation (between input and hidden layer (v12 v22 v1))(vnew = vold + delv):  -0.3 0.4006117118183497 0.5006117118183497
-------------------------------------------------------------------------------------------------------------------------------------------
Weight updation (between hidden and output layer (w1 w2 w0))(wnew = wold + delw):  0.41636688327313887 0.12116280109894084 -0.17023304605910483
-------------------------------------------------------------------------------------------------------------------------------------------
```

Conclusion: Thus, we have successfully understood back propagation algorithm. Also implemented the same using python programming language.

Post Lab Descriptive Questions :

1. Derive the weight change equations for EBPA by using tangent activation function.

ANS)

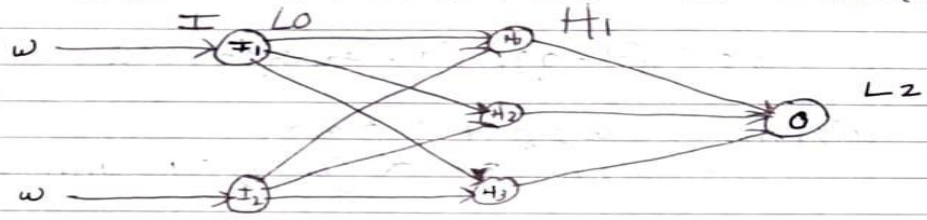
Ans) tangent activation function • $f(x) = \tanh(x)$.

$f'(x) = 1 - f(x)^2$ ... by property.

α = learning rate.

d = required output.

Consider neural network with 1 hidden layer.



Let w_1 = weights from I to H_1.

w_2 = weights from H_1 to O.

α_i = activation function for layer L_i,

z_i = net input for layer L_i.

forward propogation :-

for hidden layer,

$$z_1 = w_1^T \alpha_0$$

$$\alpha_1 = F(w_1^T \alpha_0) \quad \ldots (1)$$

for output layer,

$$z_2 = w_2^T \alpha_1$$

$$z_2 = w_2^T f(w_1^T \alpha_0) \text{ from } (1).$$

$$\alpha_2 = f(w_2^T f(w_1^T \alpha_0)).$$

Back propogation :-

(output) Error :- $d - O$ $(O = output)$.
layer

Weight change for $w_2 = \Delta w_2 = \alpha \delta_0 Z_1$.

$\delta_0 = \text{Error} \cdot f'(z_2)$.

$= (d-O) \cdot f'(z_2)$.

$\delta_0 = (d-O) \cdot (1 - \tanh(z_2)^2)$.

$w_{2new} = w_{2old} + \Delta w_2$.

$= w_{2old} + \alpha [(d-O)(1 - \tanh(z_2)^2)] z_1$

(hidden) error :- $w_{2new}^T \delta_0$.
layer

$\text{Error} = L_1 = w_2^T [(d-O)(1 - \tanh(z_2)^2]$

$\therefore \delta_1 = \text{Error} \cdot f'(z_1)$

$\therefore \delta_1 = \text{Error} (1 - F(z_1)^2)$.

$\therefore \Delta w_1 = \alpha \delta_1 z_0$

$= \alpha [\text{Error} \cdot (1 - f(z_1)^2)] \cdot z_0$

$\therefore w_{1new} = w_{1old} + \Delta w_1$.

$\therefore w_{1new} = w_{1old} + \alpha [\text{Error} \cdot (1 - f(z_1)^2)] \cdot z_0$

—

Date: 28 / 10 / 2021 **Signature of faculty in-charge**