Batch: A1 Roll No.: 16010120015

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:

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 wjk 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.

Back-propagation training algorithm :

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:

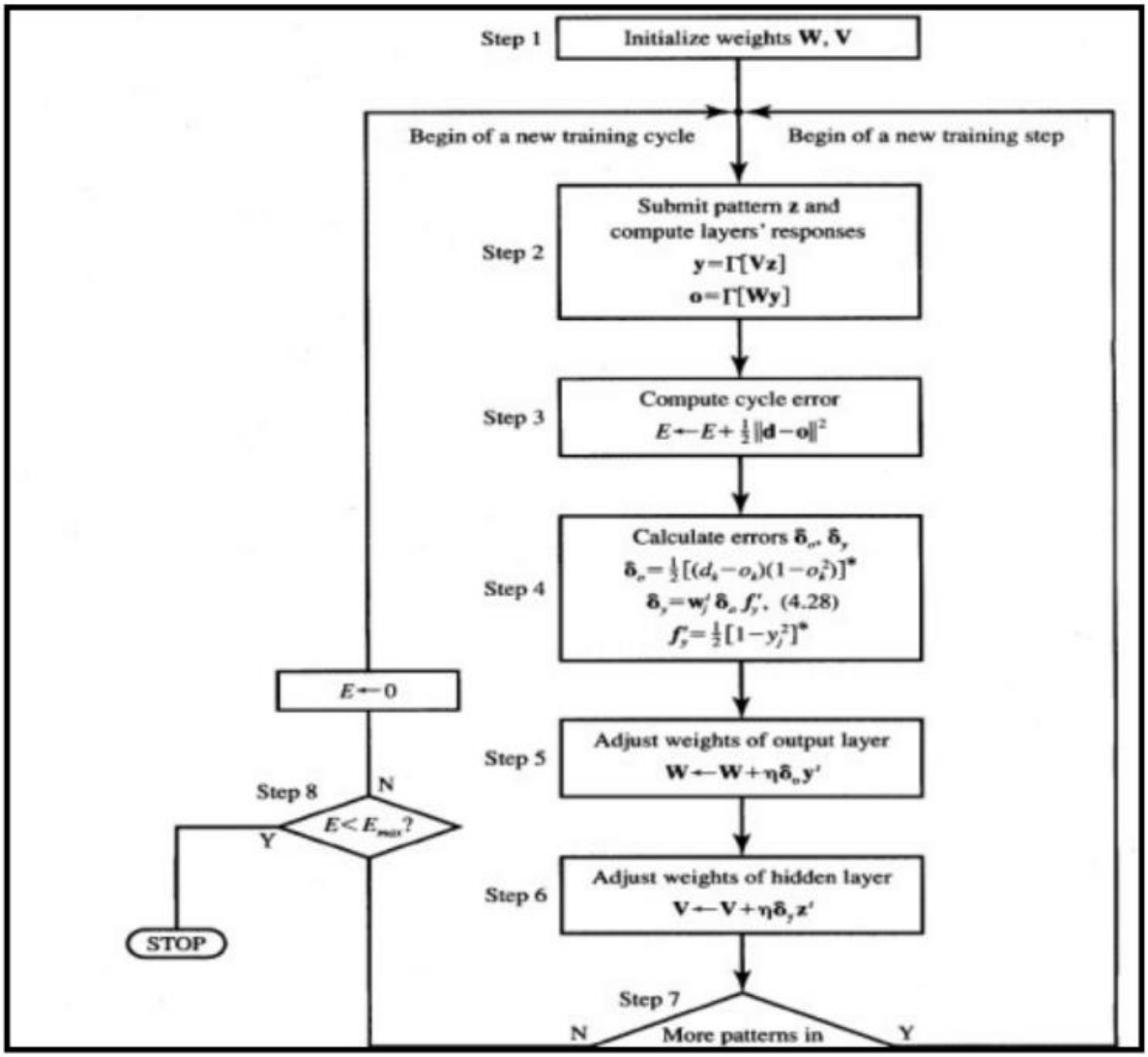
– **Forward pass**:

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

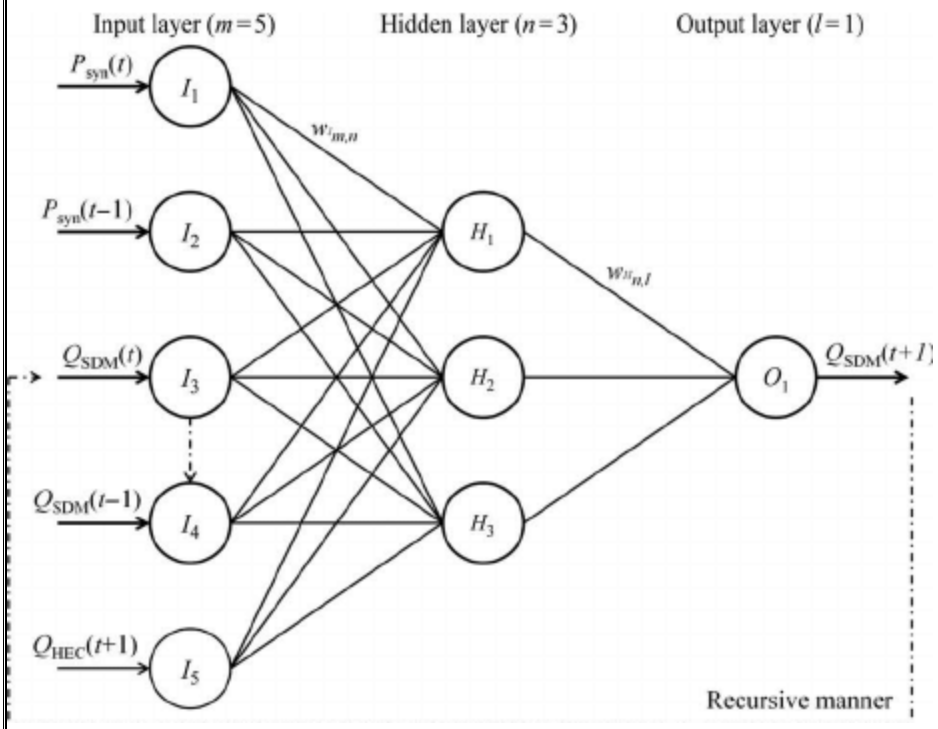
– **Backward pass**:

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.



Network Architecture :



Implementation Details :

To implement EBPT algorithm to find the new weights for the network given for given input output pair for certain no. of epochs.

```
import random
import numpy as np

# weights from input to hidden layer
w11 = float(input("\tEnter weight11: "))
w12 = float(input("\tEnter weight12: "))
w21 = float(input("\tEnter weight21: "))
w22 = float(input("\tEnter weight22: "))

# weights from hidden layer to output layer
v13 = float(input("\tEnter weight11: "))
v23 = float(input("\tEnter weight12: "))

# bias weights
```

```
b1 = float(input("\tEnter weight13: "))
b2 = float(input("\tEnter weight23: "))
b3 =float(input("\tEnter weight23: "))

x1 = 0.35
x2 = 0.9

target = 0.5
learning_rate = 1

def net(w11, x1, w12, x2, b1):
  net1 = w11 * x1 + w12 * x2 + 1 * b1
  return net1

def out(net1):
  out1 = 1.0/(1.0+np.exp(-net1))
  return out1

def computing_err(out):
  E_tot = 0.5 * (target - out)**2
  return E_tot

#for outer layer
def delta_outlayer(out):
    return out*(1-out)*(target-out)

#for hidden layers
def delta_hiddden(out1,out2,delta_out):
    delta_h1 = out1*(1-out1)*v13*delta_out
    delta_h2 = out2*(1-out2)*v23*delta_out
    return delta_h1,delta_h2

epochs = int(input("Enter number of epochs: "))

for i in range(epochs):
  print(f"____________Epochs{i+1}_____________")
  neu1 = net(w11, x1, w21, x2, b1)
  print("h1 net:", neu1)
  out1 = out(neu1)
  print("h1 output:", out1)
  neu2 = net(w12, x1, w22, x2, b2)
  print("h2 net:", neu2)
```

```
out2 = out(neu2)
print("h2 output:", out2)

h1 = net(v13, out1, v23, out2, b3)
print("o net:", h1)
oh1 = out(h1)
print("output layer output", oh1)

err = computing_err(oh1)
print("Total error: ", err)


if target == oh1:
  print("Got the Desired output")

else:

  delta_out = delta_outlayer(oh1)
  print("delta from output layer: ",delta_out)

  delta_h1,delta_h2 = delta_hiddden(out1,out2,delta_out)
  print("Delta for h1: ",delta_h1)
  print("Delta for h2: ",delta_h2)

  v13 += learning_rate*delta_out*out1
  v23 += learning_rate*delta_out*out2
  b3 += learning_rate*delta_out*1

  w11 += learning_rate*delta_h1*x1
  w21 += learning_rate*delta_h1*x2
  w12 += learning_rate*delta_h2*x1
  w22 += learning_rate*delta_h2*x2
  b1 += learning_rate*delta_h1*1
  b2 += learning_rate*delta_h2*1

  print("Updated Weights: ")
  print("w11: ",w11,"w12: ",w12,"w21: ",w21,"w22: ",w22,"v13: ",v13,"v23: ",v23)

  print("Updated Biases: ")
  print("b1: ",b1,"b2: ",b2,"b3: ",b3)
```

Output:

```
PS D:\Semester 5\SC> & "C:/Program Files/Python310/python.exe" "d:/Semester 5/SC/ebpta.py"
_____EPOCH1_____
h1 net: 1.355
h1 output: 0.7949458649106351
h2 net: 1.48
h2 output: 0.8145725807070178
o net: 1.3715990821095065
output layer output 0.7976383864867824
Total error:  0.044294304555227634
delta from output layer:  -0.048042225944951424
Delta for h1:  -0.0023493648261001423
Delta for h2:  -0.006530844932930901
Updated Weights:
w11:  0.09917772231086495 w12:  0.3977142042734742 w21:  0.7978855716565099 w22:  0.5941222395603621 v13:  0.2618090311439584 v23:  0.8608661200291113
Updated Biases:
b1:  0.5976506351738998 b2:  0.7934691550670692 b3:  0.3519577740550486
_____EPOCH2_____
h1 net: 1.3504598524735614
h1 output: 0.7942047982821733
h2 net: 1.4673791421671112
h2 output: 0.8126587016702905
o net: 1.2594781062379854
output layer output 0.7789362535923094
Total error:  0.03890271678405658
delta from output layer:  -0.04803130724946256
Delta for h1:  -0.002055307379446115
Delta for h2:  -0.006295087031083746
Updated Weights:
w11:  0.09845836472805881 w12:  0.3955109238125949 w21:  0.7960357950150083 w22:  0.5884566612323868 v13:  0.22366233645866992 v23:  0.8218330602402363
Updated Biases:
b1:  0.5955953277944537 b2:  0.7871740680359854 b3:  0.303926466805586
_____EPOCH3_____
h1 net: 1.346487970962782
h1 output: 0.7935548612936107
h2 net: 1.4552138864795419
h2 output: 0.8107995595048461
```

. After 20 epochs:

```
_____EPOCH20_____
h1 net: 1.3382100817468938
h1 output: 0.7921954359337233
h2 net: 1.3655300588613228
h2 output: 0.796657007277045
o net: 0.11127327404606868
output layer output 0.5277896507375869
Total error:  0.0003861323440585315
delta from output layer:  -0.006925951718436975
Delta for h1:  0.00015305551329491949
Delta for h2:  -0.0005146896592323027
Updated Weights:
w11:  0.09701270441239916 w12:  0.37908791585159096 w21:  0.7923183827747408 w22:  0.5462260693326627 v13:  -0.13972642232557358 v23:  0.45322110476408795
Updated Biases:
b1:  0.5914648697497117 b2:  0.7402511881474032 b3:  -0.15476599814813258
```

Conclusion: Thus, we have understood the concept and successfully implemented back propagations algorithm.

Post Lab Descriptive Questions :

1. What is meant by local and global minima?

Ans. In general, solvers return a local minimum (or optimum). The result might be a global minimum ,but this result is not guaranteed.

- A *local* minimum of a function is a point where the function value is smaller than at nearby points, but possibly greater than at a distant point.
- A *global* minimum is a point where the function value is smaller than at all other feasible points.

2. What are factors that can improve the convergence of learning in Back propagation learning algorithm

Ans. The factors that improve the convergence of Error Back Propotion Algorithm(EBPA) are called as learning factors they are as follows:

1. Initial weights
2. Steepness of activation function
3. Learning constant
4. Momentum
5. Network architecture
6. Necessary number of hidden neurons.