Name: Apeksha Chavan
Roll No: 2017130013
Batch: C (FCI)
Course Code: OE5 (Fundamentals of Computational Intelligence(FCI))

Experiment No: 1
Name of the Experiment: Supervised Learning (Back Propagation Neural Networks)

Theory: The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to other cell's dendrites. The principle of the back propagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state. Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. Backpropagation can be used for both classification and regression problems, but we will focus on classification in this tutorial. In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as [1, 0] and [0, 1] for A and B respectively.
This is called a one hot encoding.

Procedure:

1) **Forward Propagation**
    a. Initialize the Input Vector, Target output vector, Learning rate parameter, Bias on jth hidden unit and kth output unit, Weights for jth hidden unit and kth output unit.

b. Calulate net input for the hidden layer using the formula:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

c. Apply binary sigmoidal activation function to zinj to get z
d. Calulate net input for the output layer using the formula:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

e. Apply binary sigmoidal activation function to yink to get y

2) Backward Propagation

a. Get the error portion Delta k using the formula:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

where, $f'(y_{in}) = f(y_{in})[1 - f(y_{in})]$

b. Find the changes in weight between hidden and output layer using:

Delta wk = Learning Rate * Deltak * zk

c. Compute the error portion Deltaj between the input and hidden layer using:

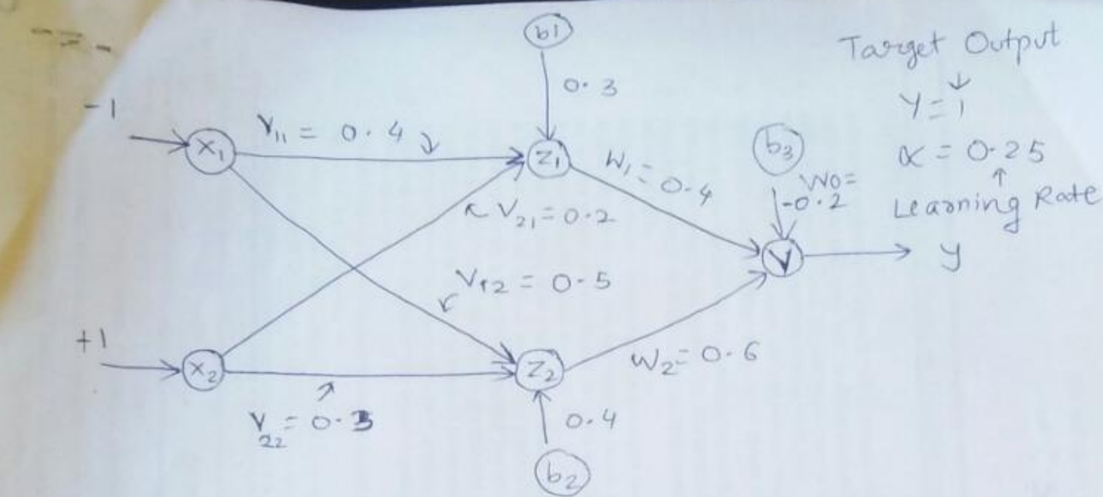
$$\boxed{\delta_j = \delta_{inj} f'(z_{inj})}$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

d. Finally find the changes in weights between input and hidden layer using:
Delta weight = learning rate * error * input (for bias there is no input part)

3) Updating Weights

a. Update the old weights by adding the change found in Step 2d to them so as to adjust them according to the current Epoch.



$x \rightarrow$ Input layer, $z \rightarrow$ hidden layer, $v \rightarrow$ output layer

Output at Hidden layer:

$$z_1 = x_1 v_{11} + x_2 v_{21} + b_1$$

Activation function is sigmoid $= \dfrac{1}{1+e^{-x}} = f(x)$

so at $z_1 = \dfrac{1}{1+e^{-z_1}}$

Example:

Input values:

$x_1 = -1$ $\quad$ $b_1 = 0.3$

$x_2 = -1$ $\quad$ $b_2 = 0.4$

Initial weights:

$V_{11} = 0.4$ $\quad$ $V_{21} = 0.2$

$V_{12} = 0.5$ $\quad$ $V_{22} = 0.3$

Implementing forward Pass:

$$Z_1 = x_1 \times V_{11} + x_2 \times V_{21} + b_1$$

$$= (-1) \times 0.4 + (+1) \times 0.2 + 0.3$$

$$\boxed{Z_1 = \cancel{0.3}\ 0.1}$$

$$Z_2 = x_1 v_{12} + x_2 v_{22} + b_2$$
$$= -1 \times 0.5 + 1 \times 0.3 + 0.4$$
$$\therefore \boxed{Z_2 = 0.2}$$

For getting the output applying ~~an~~ activation func. on Hidden layers

$$Z_1 = \frac{1}{1+e^{-z_1}} = \frac{1}{1+e^{-0.1}} = 0.525$$

$$Z_2 = \frac{1}{1+e^{-z_2}} = \frac{1}{1+e^{-0.2}} = 0.549$$

Now, to calculate the net input entering the output layer:

$$y_{in} = Z_1 W_1 + Z_2 W_2 +$$
$$= 0.525 \times 0.4 + 0.549 \times 0.6 + (-0.2)$$
$$= 0.21 + 0.329 - 0.2$$
$$\boxed{y_{in} = 0.339}$$

Calculating Total Error:

$$f'(y_{in}) = f(y_{in}(1 - f(y_{in})))$$

$$f(y_{in}) = \frac{1}{1+e^{-y_{in}}} = \frac{1}{1+e^{-0.339}} = 0.584$$

$$\therefore f'(y_{in}) = 0.584(1 - 0.584) = 0.242$$

$$\delta_1 = 0.242(1 - 0.584) \quad \left[\because \delta = f'(y_{in})(\text{Target} - f(y_{in}\right.$$

$$\cancel{\boxed{f'(y_{in}) = 0.101}}$$

$$\boxed{\delta_1 = 0.101} \rightarrow \text{Error}$$

Changes in weights in hidden layers:

$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.101 \times 0.525 = 0.0132$

$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.101 \times 0.549 = 0.0138$

$\Delta w_0 = \alpha \delta_1 = 0.25 \times 0.101 = 0.0252$

Error Between input & Hidden layer:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$\cancel{\delta_{in1} = \delta_1}$ $\delta_{in1} = \delta_1 w_{11} = 0.101 \times 0.4 = 0.0404$

$\delta_{in2} = \delta_1 w_{22} = 0.101 \times 0.6 = 0.0606$

Error $\delta_1 = \delta in_1 \, f'(z_{in1})$

$f'(z_{in_1}) = f(z_{in1})[1 - f(z_{in1})]$

$= 0.525 [1 - 0.525]$

$= 0.249$

$\delta_1 = \delta_{in_1} f'(z_{in1}) = 0.0404 \times 0.249 = \underline{\underline{0.01}}$

$\delta_2 = \delta_{in_2} f'(z_{in_2})$

$f'(z_{in_2}) = 0.549 (1 - 0.549) = 0.247$

$\delta_2 = \delta_{in_2} \times f'(z_{in2}) = 0.0606 \times 0.247 = \underline{\underline{0.014}}$

Changes in weight Between input & Hidden layer:

$\Delta v_{11} = \alpha \delta_1 x_1 = -0.0025$

$\Delta v_{21} = \alpha \delta_1 x_2 = 0.0025$

$\cancel{\Delta v_{01} = \alpha \delta_2 x_1 =}$ $\Delta b_1 = \alpha \delta_1 = 0.0025$

$\Delta v_{12} = \alpha \delta_2 x_1 = -0.0035$

$\Delta v_{22} = \alpha \delta_2 x_2 = 0.0035$

$\Delta b_2 = \alpha \delta_1 = 0.0035$

Final weights of network:

$V_{11}(new) = V_{11} + \Delta V_{11} = 0.4 - 0.0025 = 0.3975$

$V_{12}(new) = V_{12} + \Delta V_{12} = 0.5 - 0.0035 = 0.4963$

$V_{21}(new) = V_{21} + \Delta V_{21} = 0.2 + 0.025 = 0.2025$

$V_{22}(new) = V_{22} + \Delta V_{22} = 0.3 + 0.0035 = 0.4132$

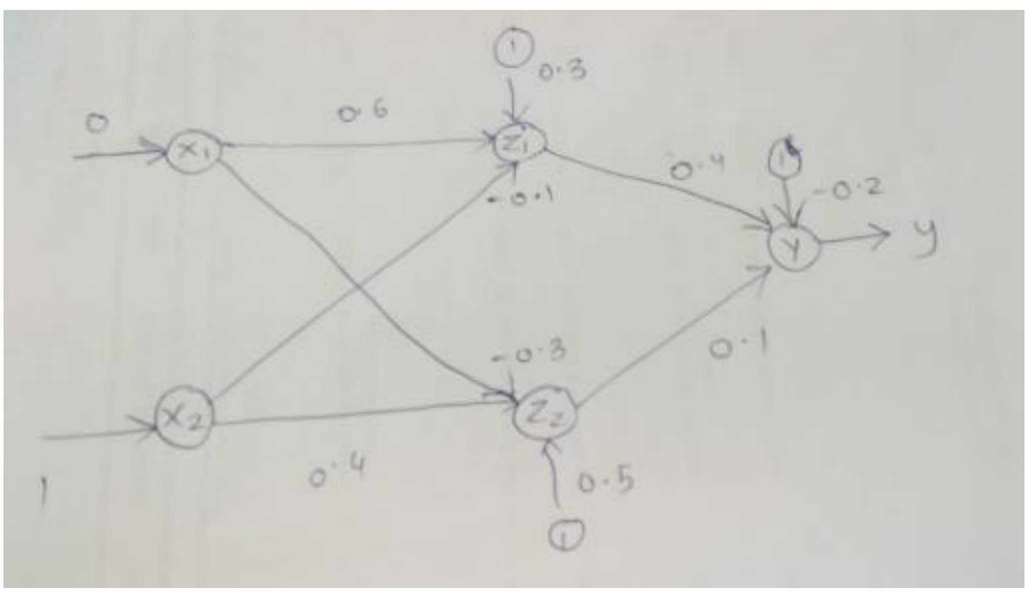
$W_1(new) = W_1 + \Delta W_1 = 0.4 + 0.0132 = 0.4132$

$W_2(new) = W_2 + \Delta W_2 = 0.6 + 0.0138 = 0.6138$

$b_1(new) = b_1 + \Delta b_1 = 0.4 + 0.0025 = 0.4025$

$b_2(new) = b_2 + \Delta w_0 = -0.2 + 0.0252 = -0.1748$

Example Network used in the code:



Code:

```
# -*- coding: utf-8 -*-
"""fci_exp1.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/
1PfsY_fbt9JOPyoDixCL9FAZ_rqWMGFXP
""" import numpy

as np

def sigmoid(x):
  return 1/(1+np.exp(-x))

alpha = 0.25  # Learning Rate
tk = 1  # Target Value
```

```
x    = np.reshape(np.array([0, 1]), (1,2))  # Input Vector
i. e. [x1, x2] v = np.array([[0.6, -0.3], [-0.1, 0.4]])  #
Weight Vector [ v11, v12], [v21, v22] b =
np.reshape(np.array([0.3, 0.5]), (1,2))  # Bias Vector
[v01, v02] wk = np.reshape(np.array([0.4, 0.1]), (1,2))  #
Weight Vect or for Output [w1, w2] w0 = -0.2 # bias on
output neuron

# Running the Algorithm for 1000 Epochs
for i in range(0, 1000):
  zin = x @ v + b # Net Input to Hidden Layer
  z = sigmoid(zin)  # Activation to calculate the Output  (
1x2)

  yin = z @ np.transpose(wk) + w0  # Net Input to Output La
yer
y    = sigmoid(yin) # Activation to calculate Output (1x1)

  df_yin = y @ (1-y)  # f'(yin)
  dk = (tk-
y) @ df_yin  # Error Portion (Since we have only 1 output n
euron so k=1)

  # Changes in weight between hidden and output layer
  dw = (alpha*dk) @ z
  dw0 = alpha*dk

  # Error portion delta j between input and hidden layer (
dj = din_j * f'(zinj) )
```

```
  din_j = dk @ wk # summation of dk @ wk but since k=1
henc e we ignore it

  df_zin = z * (1-z)  # error
  dj = din_j * df_zin

  # Changes in weight between input and hidden layers
dv = alpha*np.transpose(dj) @ x
  dv0 = alpha*dj

  # Computing the final weights of the network
v_new = dv + np.transpose(v)
  wk_new = dw + wk
  b_new = b + dv0
  w_new = w0 + dw0

  # Printing the Weights after every 50 Epochs
if i%50 == 0:
    print('EPOCH ', i, '-----------------------------------
-------------------------------')
    print('New Weight between Input and Hidden layers')
    print('[v11, v12] = [{}, {}]'.format(round(v_new[0]
[0], 5), round(v_new[1][0], 5)))      print('[v21, v22]
= [{}, {}]'.format(round(v_new[0]
[1], 5), round(v_new[1][1], 5)))

    print('\nNew Weight between Hidden and Output layers')
    print('[w1, w2] = [{}, {}]'.format(round(wk_new[0]
[0], 5), round(wk_new[0][1], 5)))

    print('\nNew Bias for Hidden Layer Neuron')
    print('[v01, v02] = [{}, {}]'.format(round(b_new[0]
[0], 5), round(b_new[0][1], 5)))
```

```
    print('\nNew Bias for Output Layer Neuron')
    print('[w0] = [{}]'.format(round(w_new[0][0], 5)))
    print('Y=', y)
    print('\n')
  v = v_new
  wk = wk_new
  b = b_new
  w0 = w_new
```

Output:

```
EPOCH  0 ------------------------------------------------------------------
New Weight between Input and Hidden layers
[v11, v12] = [0.6, -0.3]
[v21, v22] = [-0.09705, 0.40061]

New Weight between Hidden and Output layers
[w1, w2] = [0.41637, 0.12116]

New Bias for Hidden Layer Neuron
[v01, v02] = [0.30295, 0.50061]

New Bias for Output Layer Neuron
[w0] = [-0.17023]
Y= [[0.52274144]]
```

.
.
.

```
EPOCH  950 ------------------------------------------------------------------
New Weight between Input and Hidden layers
[v11, v12] = [0.6, -0.122]
[v21, v22] = [0.06194, 0.61323]

New Weight between Hidden and Output layers
[w1, w2] = [1.35037, 1.32364]

New Bias for Hidden Layer Neuron
[v01, v02] = [0.63994, 0.71323]

New Bias for Output Layer Neuron
[w0] = [1.45084]
Y= [[0.9676771]]
```

Observation Table:

EPOCH NUMBER	Output Value (Y)
0	0.522741
50	0.817311
100	0.878694
150	0.905063
200	0.920149
250	0.930103
300	0.937255
350	0.94269
400	0.946991
450	0.950497
500	0.953422
550	0.955909
600	0.958055
650	0.95993
700	0.961586
750	0.963062
800	0.964388
850	0.965588
900	0.966679
950	0.967677

We can see that the Y value is approaching/converging to **1** which is our target value.

For Target = 0.88

EPOCH 0 -- New Weight between Input and Hidden layers
[v11, v12] = [0.6, -0.3]
[v21, v22] = [-0.09779, 0.40046]

New Weight between Hidden and Output layers
[w1, w2] = [0.41225, 0.11584]

New Bias for Hidden Layer Neuron [v01, v02] = [0.30221, 0.50046]

New Bias for Output Layer Neuron
[w0] = [-0.17772]
Y= [[0.52274144]]

EPOCH 950 -- New Weight between Input and Hidden layers
[v11, v12] = [0.6, -0.2078]
[v21, v22] = [-0.02547, 0.49146]

New Weight between Hidden and Output layers
[w1, w2] = [0.95486, 0.83123]

New Bias for Hidden Layer Neuron [v01, v02] = [0.46673, 0.59146]

New Bias for Output Layer Neuron
[w0] = [0.8093]
Y= [[0.88199559]]

Conclusion:

1) We successfully implemented the Back Propagation algorithm for feed forward networks to optimize the weights of the network so that the network effectively produces the target output with almost negligible error
2) The backpropagation algorithm normally converges reasonably fast.
3) The biggest drawback of the Backpropagation is that it can be sensitive for noisy data.
4) From our implementation we can conclude that gradient descent is taking place and moving obtained output towards expected output.