CLASS: B.E. E &TC **SUBJECT: ML**
EXPT. NO.: 3 **DATE: 17/05/21**
ROLL NO.: 42428

TITLE: Implement and test MLP trained with back- propagation algorithm

---

CO 3: Apply the fundamentals of Artificial Neural Network (ANN) to design and implement back propagation algorithm for various applications like classification, Self-Organizing Feature Maps, Learning vector quantization and Radial Basis Function networks. Using a substantial training and test data set, validate the efficacy of designed Neural Networks in terms of Confusion Matrix, accuracy, recognition rate, sensitivity and specificity.

CO 5: Carry out experiments as an individual and in a team, comprehend and write a laboratory record and draw conclusions at a technical level.

AIM:

1. To implement an MLP (Multi-Layer Perceptron)
2. To train the MLP with Backpropagation Algorithm
3. To test the MLP

SOFTWARES REQUIRED: MATLAB 7.0 or Python

THEORY:

Introduction:

- Initially, Neural Networks were not capable of doing anything. They were just another concept in the Machine Learning.
- The initial demonstration of possible applications of Neural Networks was shown by Geoffrey Hinton.
- Geoffrey Hinton et al from the University of Toronto, developed a model which was capable of Recognizing and Classifying pictures, using Neural Nets.

- The model classified 1.2 million high resolution images in the ImageNet LSVRC- 2010 contest into the 1000 different classes.
- When an image was fed to the model, it produced the Top-5 probabilities of an image belonging to a certain class.
- The neural network, of the model, which had 60 million parameters and 650,000 neurons, consisted of five deep layers, some of which were followed by max- pooling layers, and three fully-connected layers with a final 1000-way Softmax.

Neuron:

A basic Biological Neuron inspired the concept of Neural Networks, in Machine Learning.

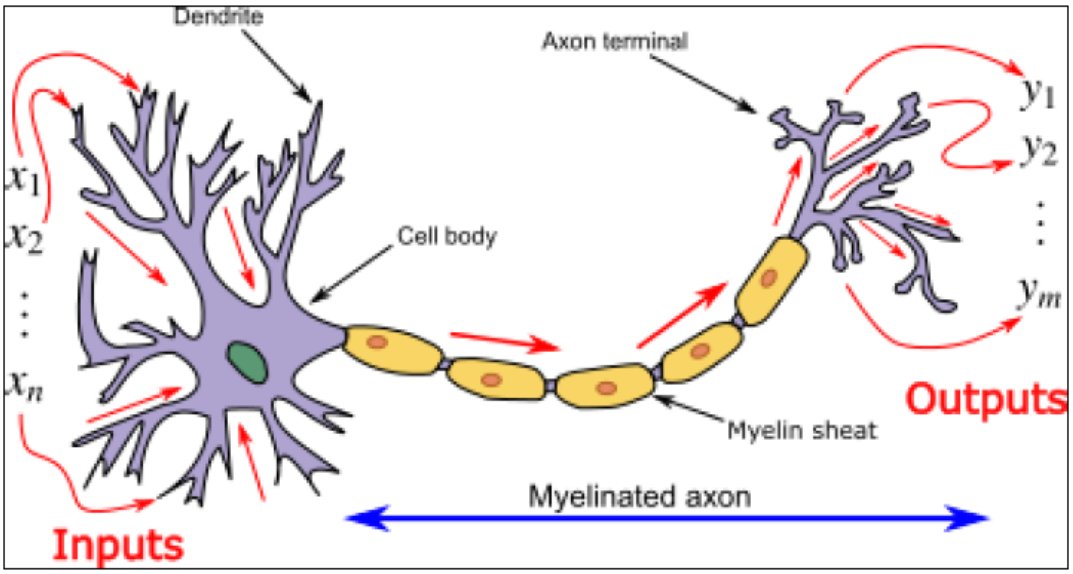


Fig. 1: Biological Neuron

- A Biological Neuron consists of a Cell Body with a Nucleus at its centre.
- The Cell Body spreads in either direction; Along one direction, Axon is obtained, which has a Pre-Synaptic thickening at its end. This consists of Villi which are used to interact with other neurons by supplying Blood. Along the other end, branches emerge, which are known as Dendrites, and act as inputs to the Neuron.
- The Villi transport the blood to these Dendrites (which are part of the Dendritic Tree).
- The Axon acts as a transmission line.

- If there is enough stimulation from the Dendritic Tree, then a spike will go down the Axon, which will act as a Carrier.
- Note that the Neurons are separated by the Synaptic Gaps.

MLP (Multi-Layer Perceptron):

- An MLP is same as an Artificial Neural Network (ANN).
- It consists of an Input and an Output layer, as well as one or more Hidden Layers.
- The Hidden layers consist of different linear combinations of weights and inputs fed to them.

Neural Network:

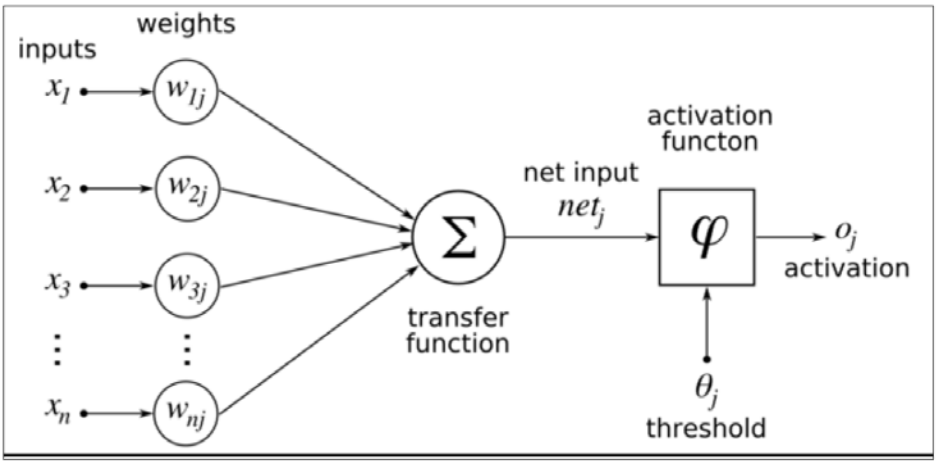


Fig. 2: Basic Neural Network

- The Inputs of the Neural Network (Dendrites), xi, for integral values of i in the interval [1,n], have corresponding weights. The Output from the Summing Block is given as follows:
- Considering the Activation Function to be a Step Function, with a Threshold, allows a Binary Output in conjunction with a Binary Input.
- The final output, oj is a function of input vector x, weight vector w, and threshold, θj.
- The features of a Neural Network Model can be:
  1. All or None (Binary)
  2. Cumulative Influence
  3. Synaptic Weights

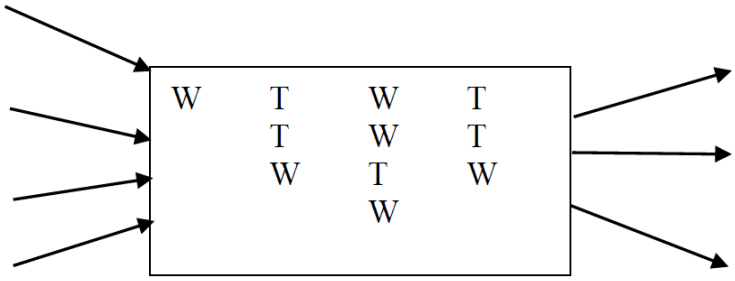


Fig. 3: Neural Network- A Black Box

- A Neural Network (generally, a combination of at least 2 Neurons) can be considered as a Black Box with n Inputs and m Outputs, with respect to a User.
- But, as far as "training a Neural Net" is concerned, it refers to adjusting Weights and Thresholds so that what we obtain is what we desire.
- In a broader perspective, a Neural Net can be considered as a Function Approximator.

Let z be the obtained output and d be the desired output. Assuming vector x to represent the inputs, we can observe a function, g such that:

$$\boldsymbol{d} = g(\boldsymbol{x})$$

Multiple Performance Metrics are possible for a Neural Net. But most of those Metrics have some drawbacks which cater themselves unsuitable. For example:

1. Hill-Climbing approach can be employed over lesser number of weights. But, over a huge number of parameters, it is deemed computationally intractable.
2. Considering the performance metric to be Euclidean Distance between d and z, we obtain a function which is not differentiable at x = 0.

So, considering a performance metric, P as follows:

$$\boldsymbol{P} = \left(\frac{1}{2}\right) * (\boldsymbol{d_x} - \boldsymbol{o_x})^2$$

allows us to get rid of Non-Differentiability issues.

- The above equation is also referred to as Gradient Descent Function.

- The reason for the above choice of Performance Metric is that this formula has some convenient properties. The formula yields a minimum at o = d and monotonically (smoothly) increases as o deviates from d.
- But, Gradient Descent requires a continuous function, but our threshold is a Step function.
- In order to account for the Threshold (which is extra baggage), we employ a scheme similar to the following figure:

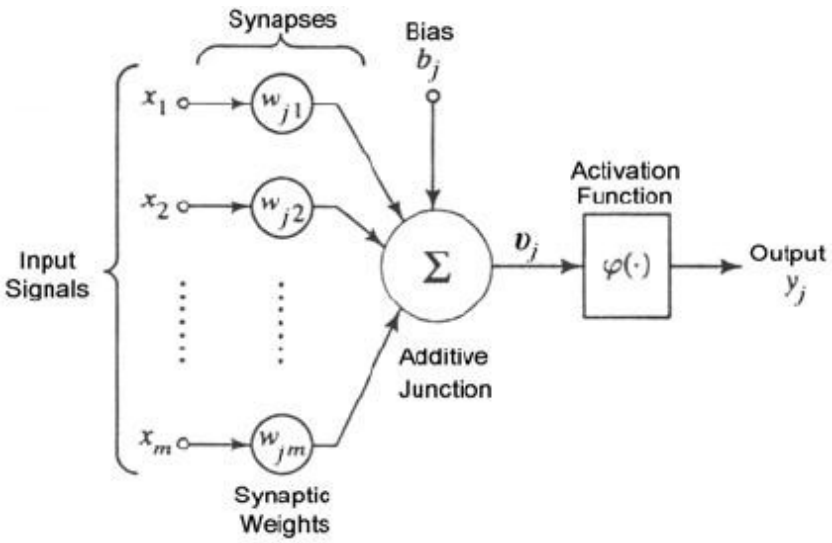


Fig. 4: Neural Net with a Unipolar Activation Function

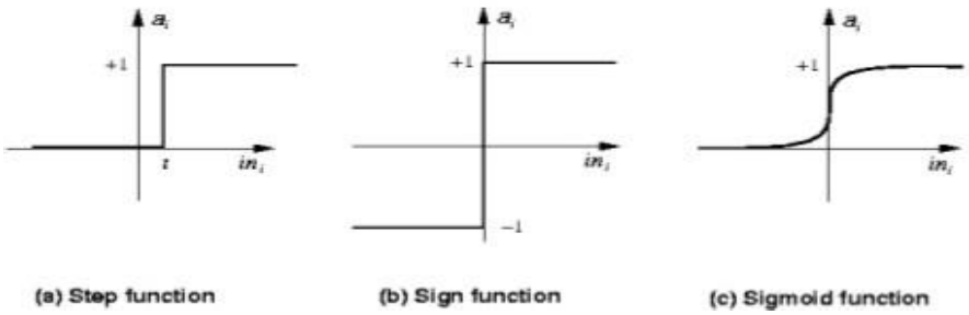
- In the above figure, Bias, bj, can be equated to -1 and an extra weight (w0) equal to the Threshold can be used over the Bias.
- This implies that the Activation function is now a Unipolar Function, but still the function is discontinuous at x = 0.
  1. Using a smoother Thresholding/Activation Function like Sigmoid function, resolves most of the issues. It is given as follows:

$$y = f(x) = \frac{1}{1 + e^{-x}}$$

$$
\begin{aligned}
\frac{dy}{dx} &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -1 \times (1+e^{-x})^{-2} \times e^{-x} \times -1 \\
&= \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \times \frac{1+e^{-x}-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \times \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= y(1-y)
\end{aligned}
$$

Moreover, a Sigmoid function, when differentiated obtains a result in terms of itself!

2. Sigmoid Function is smooth (continuous) and obtains the results in terms of Probability of occurrence of a particular class as Output, as far as a Classification problem is concerned.



(a) Step function (b) Sign function (c) Sigmoid function

Fig. 5: Step function to Sigmoid Function (From Discontinuous to Continuous Activation Function)

Backpropagation:

- In Backpropagation, the weights are initially randomly assigned.
- The output is computed based on these random weights (Feed Forward Path), i.e., the flow is from input to output.

- This output is compared with the desired output and the weights are adjusted so as to minimize the difference (or a fraction of squared difference) between the obtained and desired outputs. Here, the flow is from the output to input (Backward propagation).
- Backpropagation is a specialized case of Gradient Descent. We are trying to find the minimum of a Performance Metric, P, by changing the weights associated with neurons, in order to move in the direction of the gradient in a space that gives P as a function of the weights, w.
- That is, we move in the direction of most rapid descent if we take a step in the direction with components governed by the following formula, which shows how much to change a weight, w, in terms of a partial derivative:

$$\Delta w \propto \frac{\partial P}{\partial w}$$

- The actual change in the weights is also influenced by Learning Rate/Rate Constant, α; Accordingly, the new weight, w', is given as:

$$w' = w + \alpha * \frac{\partial P}{\partial w}$$

Simplest Neural Net:
Consider the simplest possible Neural Net: one input, one output, and two neurons - the left neuron and the right neuron. A net with two neurons (Fig. 7) is the smallest that illustrates how the derivatives can be computed layer by layer.

Note that the subscripts indicate layer. Thus, il, wl, pl, and ol are the input, weight, product, and output associated with the Neuron on the Left (l) while ir, wr, pr, and or are the input, weight, product, and output associated with the neuron on the Right (r). Of course, ol = ir.

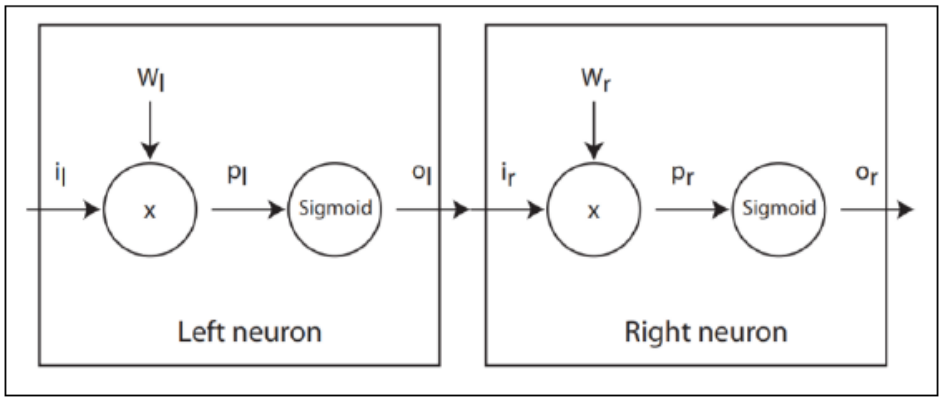


Fig. 6: Simplest Neural Network

Let us assume that the output of Right Neuron, or to decide the Performance Metric, P. The partial derivative of the performance metric, P, with respect to both weights, obtains a measure of the enhancement in Performance with adjustment in weights.

Using the Chain Rule of Differentiation, we have:

$$\frac{\partial P}{\partial w_r} = \frac{\partial P}{\partial o_r} * \frac{\partial o_r}{\partial p_r} * \frac{\partial p_r}{\partial w_r}$$

Using the previous results, we have:

$$\frac{\partial P}{\partial w_r} = [(d - o_r)] * [o_r(1 - o_r)] * i_r$$

Repeating the same analysis with respect to wl, we have:

$$\begin{aligned}\frac{\partial P}{\partial w_l} &= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial w_l} \\ &= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial w_l} \\ &= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial w_l} \\ &= \frac{\partial P}{\partial o_r} \times \frac{\partial o_r}{\partial p_r} \times \frac{\partial p_r}{\partial o_l} \times \frac{\partial o_l}{\partial p_l} \times \frac{\partial p_l}{\partial w_l} \\ &= [(d - o_r)] \times [o_r(1 - o_r)] \times [w_r] \times [o_l(1 - o_l)] \times [i_l]\end{aligned}$$

So, we have:

$$\frac{\partial P}{\partial w_r} = (d - o_r) \times o_r(1 - o_r) \times i_r$$

$$\frac{\partial P}{\partial w_l} = (d - o_r) \times o_r(1 - o_r) \times w_r \times o_l(1 - o_l) \times i_l$$

Note that in the above equations, the first three terms on the RHS are repetitive and need not be computed again.

If the Neural Nets are two or more Neurons deep, then the number of paths are going to blow up, which would make this problem of finding the Partial Derivatives seem a computationally hard problem. But, in fact, most of those terms in the results of Partial Derivatives are redundant and instead of computing all over, can be reused. “What’s computed is computed and need not be recomputed!”

Adjustment of Weights:

Assuming Sigmoid Activation Function for the Neural Net, whose inputs and weights are expressed in terms of vectors, x and w, respectively, we have output, z, as follows:

$$z = \frac{1}{1 + e^{-(wx - T)}}$$

$$= \frac{1}{1 + e^{-wx + T}}$$

Based on w & T, the standard Sigmoid curve is shrunk (expanded) and/or shifted.

Based on the Target value and Obtained output, the Sigmoid curve gets adjusted (and thereby the weights) accordingly so as to get the Obtained outputs as close as possible to Target values, with respect to Gradient Descent Algorithm and its associated Learning Rate (α). Obtaining a Sigmoid curve, which obtains all (or most) of the output values similar to the desired values for a given set of input values, is the essence of the Backpropagation Method.

CONCLUSION:

The operations of the Backpropagation neural networks can be divided into two steps: feedforward and Backpropagation. In the feedforward step, an input pattern is applied to the input layer and its effect propagates, layer by layer, through the network until an output is produced. The network's actual output value is then compared to the expected output, and an error signal is computed for each of the output nodes. Since all the hidden nodes have, to some degree, contributed to the errors evident in the output layer, the output error signals are transmitted backwards from the output layer to each node in the hidden layer that immediately contributed to the output layer. This process is then repeated, layer by layer, until each node in the network has received an error signal that describes its relative contribution to the overall error.

Once the error signal for each node has been determined, the errors are then used by the nodes to update the values for each connection weights until the

network converges to a state that allows all the training patterns to be encoded. The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

The network behaviour is analogous to a human that is shown a set of data and is asked to classify them into predefined classes. Like a human, it will come up with "theories" about how the samples fit into the classes. These are then tested against the correct outputs to see how accurate the guesses of the network are. Radical changes in the latest theory are indicated by large changes in the weights, and small changes may be seen as minor adjustments to the theory.

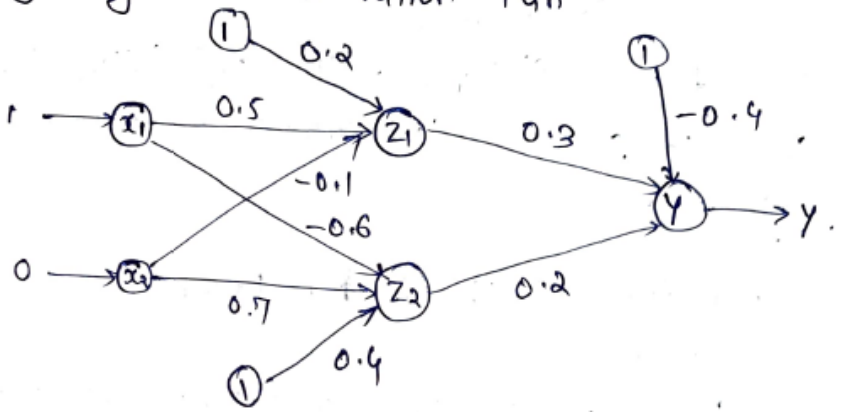
REFERENCES:

i. Laurene Fausett, “Fundamentals of Neural Networks: Architectures, AlgorithmsAnd Applications”, Pearson Education, Inc, 2008.
ii. Winston, Patrick Henry. “Artificial Intelligence” 3rd Edition, Addison-Wesley, 1992.
iii. Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, “ImageNet Classification with Deep Convolutional Neural Networks”, University of Toronto

(Course Teacher)

Solved Problem :

Using Back propogation network, find the new weights for network shown. The network is presented with the i/p pattern [1,0] & the target o/p 1. Use learning rate of $\alpha = 0.3$ & binary signoidal activation fun^n.



Feed forward stage-

Net i/p (HL)-

$Z_{in_1} = 1 \times 0.2 + 1 \times 0.5 + 0 \times -0.1 = 0.7$

$Z_{in_2} = 1 \times 0.4 + 1 \times -0.6 + 0 \times 0.7 = -0.2$.

Apply activation fun^n (HL)

$Z_1 = f(Z_{in_1}) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-0.7}} = 0.66818$

$Z_2 = f(Z_{in_2}) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{0.2}} = 0.45016.$

Net i/p (OL)

$y_{in} = 1 \times 0.4 + 0.66818 \times 0.3 + 0.45016 \times 0.2$

$y_{in} = -0.109514$

Apply activation fun^n (OL)-

$y = f(y_{in}) = \frac{1}{1+e^{0.109514}} = 0.472648$

Back Propogation of error-

Error at o/p layer -

$$\delta_k = (t_k - y_k) f'(y_{ink}) = (t_k - y_k)(f(y_{in})(1 - f(y_{in})))$$

$$\delta_k = (1 - 0.472648)(0.472648(1 - 0.472648))$$

$$\delta_k = 0.13144$$

Error portion at the hidden layer-

Net i/p error-

$$\delta_{in1} = 0.3 \times 0.13144 = 0.03943$$

$$\delta_{in2} = 0.2 \times 0.13144 = 0.026288$$

Net o/p error

$$\delta_1 = \delta_{in1} f'(z_{in1}) = \delta_{in1}(f(z_{in1})(1 - f(z_{in1})))$$

$$\delta_1 = 0.03943[0.66818(1 - 0.66818)]$$

$$\delta_1 = 0.0087422$$

$$\delta_2 = \delta_{in2} f'(z_{in2}) = \delta_{in2}[f(z_{in2})(1 - f(z_{in2}))]$$

$$\delta_2 = 0.026288[0.45016(1 - 0.45016)]$$

$$\delta_2 = 0.006483$$

Weight Updation -

O/P layer -

$W_{0new} = W_{0old} + \Delta W_0 = W_{0old} + \alpha \delta_k Z_0$

$= -0.4 + 0.3 \times 0.13144 \times 1$

$= -0.360568$

$W_{1new} = W_{1old} + \Delta W_1 = W_{1old} + \alpha \delta_k Z_1$

$= 0.3 + 0.3 \times 0.13144 \times 0.66818$

$= 0.326347$

$W_{2new} = W_{2old} + \Delta W_2 = W_{2old} + \alpha \delta_k Z_2$

$= 0.2 + 0.3 \times 0.13144 \times 0.45016$

$= 0.21775$

Hidden layer -

$V_{01new} = V_{01old} + \Delta V_{01} = V_{01old} + \alpha \delta_1 x_0$

$= 0.2 + 0.3 \times 0.0087422 \times 1 = 0.202622$

$V_{02new} = V_{02old} + \Delta V_{02} = V_{02old} + \alpha \delta_2 x_0$

$= 0.4 + 0.3 \times 0.006482 \times 1 = 0.401945$

$V_{11new} = V_{11old} + \Delta V_{11} = V_{11old} + \alpha \delta_1 x_1$

$= 0.5 + 0.3 \times 0.0087422 \times 1 = 0.502622$

$V_{12new} = V_{12old} + \Delta V_{12} = V_{12old} + \alpha \delta_2 x_1$

$= -0.6 + 0.3 \times 0.006482 \times 1 = -0.598054$

$V_{21new} = V_{21old} + \Delta V_{21} = V_{21old} + \alpha \delta_1 x_2$

$= -0.1 + 0.3 \times 0.0087422 \times 0 = -0.1$

$V_{22new} = V_{22old} + \Delta V_{22} = V_{22old} + \alpha \delta_2 x_2$

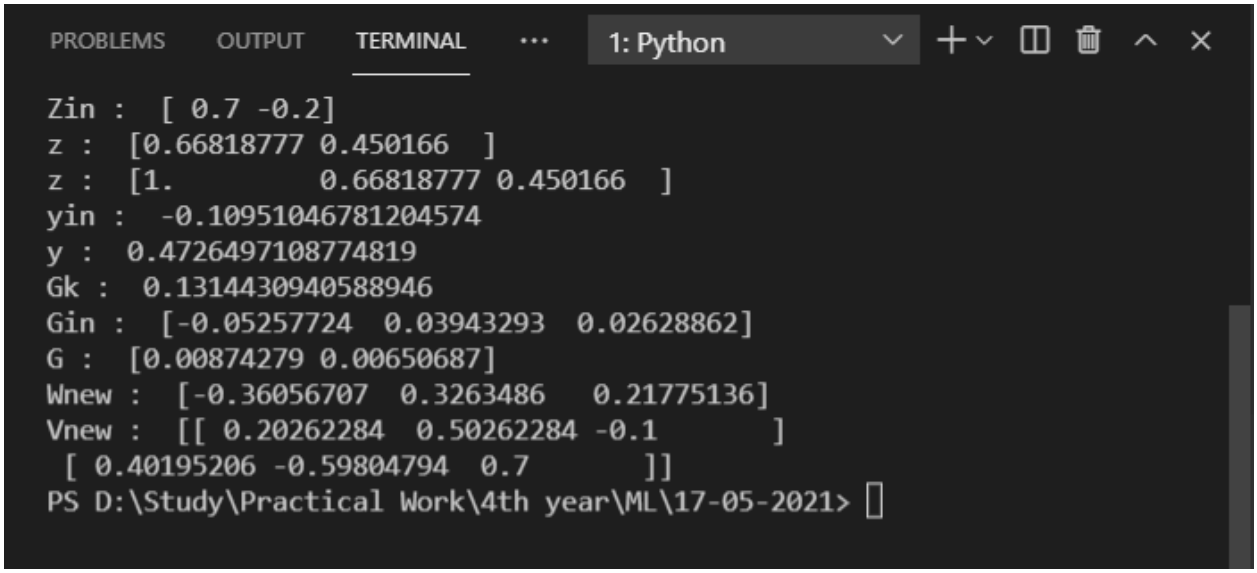
$= 0.7 + 0.3 \times 0.006482 \times 0 = 0.7$

Code :

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

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

x = np.array([1, 1, 0])

v = np.array([
    [0.2, 0.5, -0.1],
    [0.4, -0.6, 0.7]
])

w = np.array([-0.4, 0.3, 0.2])

Ze = np.array([1])

tk = 1

alpha = 0.3

#Feed Forward Stage
#Net Input(HL)
Zin = np.dot(v, x)
print("Zin : ",Zin)

#Apply Activation Function(HL)
z = sigmoid(Zin)
print("z : ",z)

z = np.append(Ze,z)
print("z : ",z)

#Net Input(OL)
yin = np.dot(z, w)
print("yin : ",yin)

#Apply Activation Function(OL)
```

```
y = sigmoid(yin)
print("y : ",y)

#Back Propogation of Error
#Error at output layer
Gk = (tk - y)*(y * (1 - y))
print("Gk : ",Gk)

#Error portion in hidden layer
Gin = w * Gk
print("Gin : ",Gin)

l = np.arange(0, Ze.size, 1)
G = np.delete(Gin * (z * (1 - z)),l)
print("G : ",G)

#Weight Updation at output layer
Wnew = w + (alpha * Gk * z)
print("Wnew : ",Wnew)

#Weight Updation at Hidden Layer
Vnew = []
for i in range(0, v.shape[0]):
    b = v[i] + (alpha * G[i] * x)
    Vnew.append(b)
Vnew = np.array(Vnew)
print("Vnew : ",Vnew)
```

OUTPUT :


PROBLEMS
OUTPUT
TERMINAL
1: Python
Zin : [0.7 -0.2]
z : [0.66818777 0.450166]
z : [1. 0.66818777 0.450166]
yin : -0.10951046781204574
y : 0.4726497108774819
Gk : 0.1314430940588946
Gin : [-0.05257724 0.03943293 0.02628862]
G : [0.00874279 0.00650687]
Wnew : [-0.36056707 0.3263486 0.21775136]
Vnew : [[0.20262284 0.50262284 -0.1]
[0.40195206 -0.59804794 0.7]]
PS D:\Study\Practical Work\4th year\ML\17-05-2021>