Batch: A2 **Roll No.: 1911027**

Experiment No. 05

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

Signature of the Staff In-charge with date

Title: HEBBS learning rule

Objective: To write a program to implement HEBBS learning rule.

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:

Hebbian learning rule :

Hebbian learning rule is one of the earliest and the simplest learning rules for the neural networks. It was proposed by Donald Hebb. Hebb proposed that if two interconnected neurons are both "on" at the same time, then the weight between them should be increased. Hebbian network is a single layer neural network which consists of one input layer with many input units and one output layer with one output unit. This architecture is usually used for pattern classification. The bias which increases the net input has value 1. Hebbian Learning Rule, also known as Hebb Learning Rule, was proposed by Donald O Hebb. It is one of the first and also easiest learning rules in the neural network. It is used for pattern classification. It is a single layer neural network, i.e. it has one input layer and one output layer. The input layer can have many units, say n. The output layer only has one unit. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

Hebbian Learning Rule Algorithm : 1) Set all weights to zero, wi = 0 for i=1 to n, and bias to zero.

2) For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.

3) Set activations for input units with the input vector Xi = Si for i = 1 to n.

4) Set the corresponding output value to the output neuron, i.e. y = t.

5) Update weight and bias by applying Hebb rule for all i = 1 to n:

$$w_i\,(\text{new}) = w_i\,(\text{old}) + x_i\,y$$
$$b\,(\text{new}) = b\,(\text{old}) + y$$

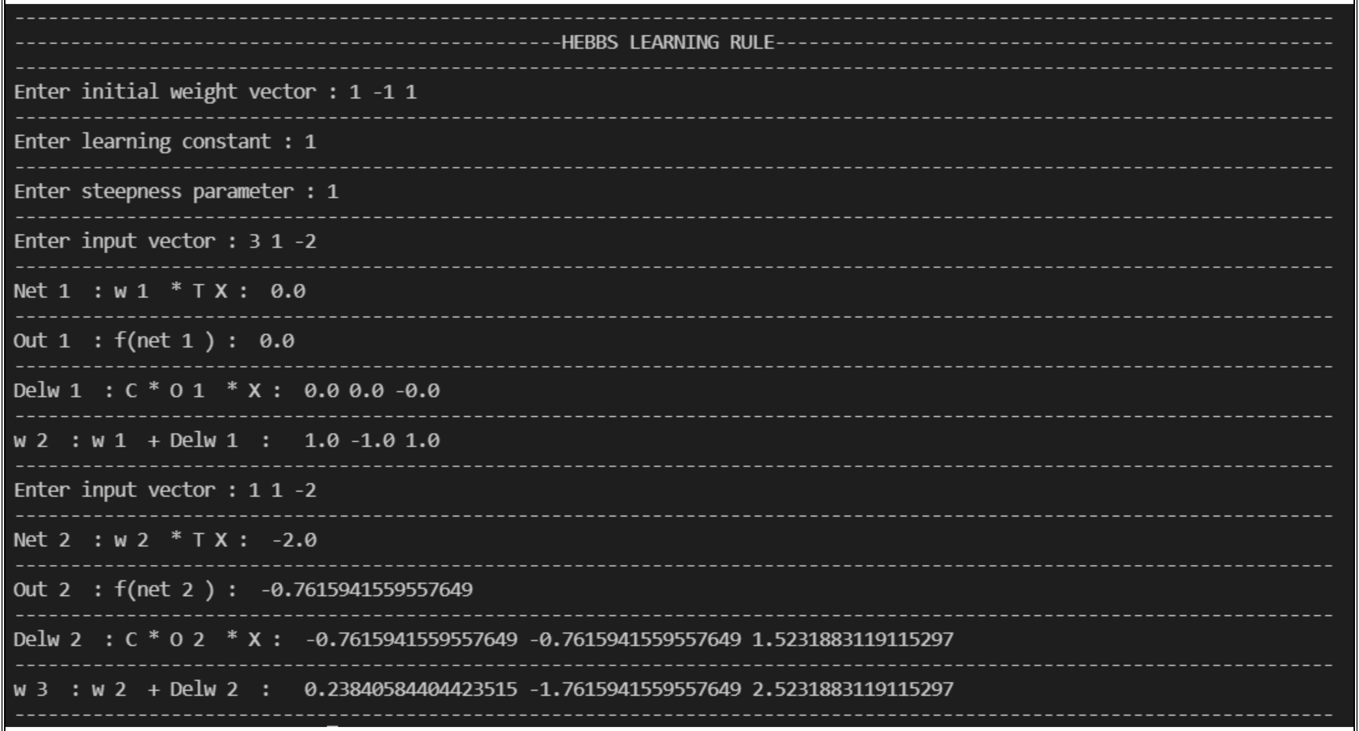
Hebbian learning is primarily used, many applications would require a highly customized learning approach. Typically, the learning of dynamical systems is accomplished without a teacher. The adjustment of system parameters, which are functions of patterns or associations to be learned, does not depend on the difference between the desired and actual output value of the system during the learning phase. According to Hebb's rule, the weights are found to increase proportionately to the product of input and output. It means that in a Hebb network if two neurons are interconnected then the weights associated with these neurons can be increased by changes in the synaptic gap. This network is suitable for bipolar data. The Hebbian learning rule is generally applied to logic gates.

Code:

```python
import math
def calnet(x1,w1):
    net=0
    for i,j in zip(x1,w1):
        net=net+float(i)*float(j)
    return net

def calout(net,lamb):
    temp=2/(1+math.exp(-1*lamb*net))
    return temp-1

def caldelw(c,out,x1):
    temp=c*out
    lst=list()
    for i in x1:
        lst.append(temp*float(i))
    return lst

def calwup(w1,delw):
    temp=list()
    for i,j in zip(w1,delw):
        temp.append(float(i)+float(j))
    return temp

print("-----------------------------------------------------------------------------------------------------------------------")
print("-------------------------------------------------HEBBS LEARNING RULE-----------------------------------------------------")
print("-----------------------------------------------------------------------------------------------------------------------")
```

```
w1=input("Enter initial weight vector : ").split()
print("-------------------------------------------------------------------------------
--------------------------------------")
c=float(input("Enter learning constant : "))
print("-------------------------------------------------------------------------------
--------------------------------------")
lamb=float(input("Enter steepness parameter : "))
print("-------------------------------------------------------------------------------
--------------------------------------")
for i in range(0,2):
    x1=input("Enter input vector : ").split()
    print("---------------------------------------------------------------------------
------------------------------------------")
    net=calnet(x1,w1)
    print("Net",i+1," : w",i+1," * T X : ",net)
    print("---------------------------------------------------------------------------
------------------------------------------")
    out=calout(net,lamb)
    print("Out",i+1," : f(net",i+1,") : ",out)
    print("---------------------------------------------------------------------------
------------------------------------------")
    delw=caldelw(c,out,x1)
    print("Delw",i+1," : C * O",i+1," * X : ",*delw)
    print("---------------------------------------------------------------------------
------------------------------------------")
    wupdated=calwup(w1,delw)
    print("w",i+2," : w",i+1," + Delw",i+1," :  ",*wupdated)
    print("---------------------------------------------------------------------------
------------------------------------------")
    w1=wupdated
```

Output:



```
----------------------------------------------------------------------------------------------------
---------------------------------------------HEBBS LEARNING RULE------------------------------------
----------------------------------------------------------------------------------------------------
Enter initial weight vector : 1 -1 1
----------------------------------------------------------------------------------------------------
Enter learning constant : 1
----------------------------------------------------------------------------------------------------
Enter steepness parameter : 1
----------------------------------------------------------------------------------------------------
Enter input vector : 3 1 -2
----------------------------------------------------------------------------------------------------
Net 1  : w 1  * T X :  0.0
----------------------------------------------------------------------------------------------------
Out 1  : f(net 1 ) :  0.0
----------------------------------------------------------------------------------------------------
Delw 1  : C * O 1  * X :  0.0 0.0 -0.0
----------------------------------------------------------------------------------------------------
w 2  : w 1  + Delw 1  :   1.0 -1.0 1.0
----------------------------------------------------------------------------------------------------
Enter input vector : 1 1 -2
----------------------------------------------------------------------------------------------------
Net 2  : w 2  * T X :  -2.0
----------------------------------------------------------------------------------------------------
Out 2  : f(net 2 ) :  -0.7615941559557649
----------------------------------------------------------------------------------------------------
Delw 2  : C * O 2  * X :  -0.7615941559557649 -0.7615941559557649 1.5231883119115297
----------------------------------------------------------------------------------------------------
w 3  : w 2  + Delw 2  :   0.23840584404423515 -1.7615941559557649 2.5231883119115297
----------------------------------------------------------------------------------------------------
```

Conclusion: Thus, we have successfully understood Hebbian learning algorithm of Neural Network. Also implemented the same using python programming language.

Post Lab Descriptive Questions :

1. Explain different types of learning method.

ANS) One of the most impressive features of artificial neural networks is their ability to learn. Although simplified, artificial neural networks can model this learning process by adjusting the weighted connections found between neurons in the network. This effectively emulates the strengthening and weakening of the synaptic connections found in our brains. This strengthening and weakening of the connections is what enables the network to learn. Learning algorithms are

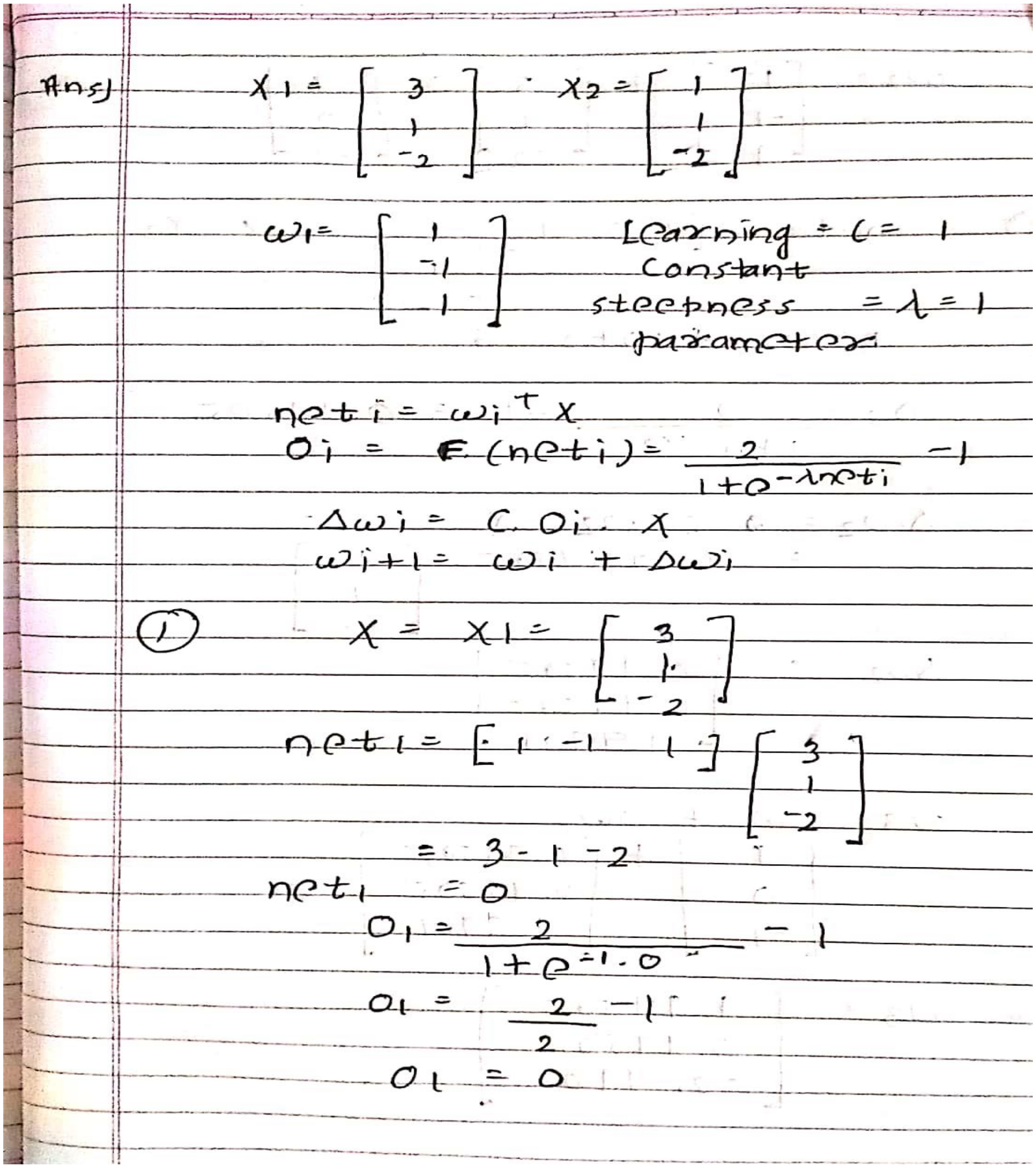
extremely useful when it comes to certain problems that either can’t be practically written by a programmer or can be done more efficiently by a learning algorithm. Facial recognition would be an example of a problem extremely hard for a human to accurately convert into code. A problem that could be solved better by a learning algorithm, would be a loan granting application which could use past loan data to classify future loan applications. There are many different algorithms that can be used when training artificial neural networks, each with their own separate advantages and disadvantages. The learning process within artificial neural networks is a result of altering the network's weights, with some kind of learning algorithm. The objective is to find a set of weight matrices which when applied to the network should - hopefully - map any input to a correct output.
Supervised learning: The learning algorithm would fall under this category if the desired output for the network is also provided with the input while training the network. By providing the neural network with both an input and output pair it is possible to calculate an error based on its target output and actual output. It can then use that error to make corrections to the network by updating it weights. Approaches and algorithms: Analytical learning, Artificial neural network, Backpropagation, Boosting (meta-algorithm), Bayesian statistics, Case-based reasoning, Decision tree learning, Inductive logic programming, Gaussian process regression, Genetic programming, Group method of data handling, Kernel estimators, Learning automata, Learning classifier systems, Minimum message length (decision trees, decision graphs, etc.), Multilinear subspace learning, Naive Bayes classifier, Maximum entropy classifier, Conditional random field, Nearest neighbor algorithm, Probably approximately correct learning (PAC) learning, Ripple down rules, a knowledge acquisition methodology, Symbolic machine learning algorithms, Subsymbolic machine learning algorithms, Support-vector machines, Minimum complexity machines (MCM), Random forests, Ensembles of classifiers, Ordinal classification, Data pre-processing, Handling imbalanced datasets, Statistical relational learning, Proaftn, a multicriteria classification algorithm .
Applications: Bioinformatics, Cheminformatics, Quantitative structure–activity relationship, Database marketing, Handwriting recognition, Information retrieval, Learning to rank, Information extraction, Object recognition in computer vision, Optical character recognition, Spam detection.
Unsupervised learning: In this paradigm the neural network is only given a set of inputs and it’s the neural network’s responsibility to find some kind of pattern within the inputs provided without any external aid. This type of learning paradigm is often used in data mining and is also used by many recommendation algorithms due to their ability to predict a user's preferences based on the preferences of other similar users it has grouped together. Approaches: Common families of algorithms used in unsupervised learning include: (1) clustering, (2) anomaly detection, (3) neural networks (note that not all neural networks are unsupervised; they can be trained by supervised, unsupervised, semi-supervised, or reinforcement methods), and (4) latent variable models. Clustering methods include hierarchical clustering, k-means, mixture models, DBSCAN, and OPTICS algorithm. Anomaly detection methods include Local Outlier Factor, and Isolation Forest

Approaches for learning latent variable models include expectation–maximization algorithm, the method of moments, and blind signal separation techniques (principal component analysis, independent component analysis, non-negative matrix factorization, singular value decomposition) Neural network methods include autoencoders, deep belief networks, Hebbian learning, generative adversarial networks (GANs), and self-organizing maps. he classical example of unsupervised learning in the study of neural networks is Donald Hebb's principle that "neurons that fire together wire together".[9] In Hebbian learning, the connection is reinforced irrespective of an error, but is exclusively a function of the coincidence between action potentials between the two neurons.[10] A similar version that modifies synaptic weights takes into account the time between the action potentials (spike-timing-dependent plasticity or STDP). Hebbian Learning has been hypothesized to underlie a range of cognitive functions, such as pattern recognition and experiential learning. Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used in unsupervised learning algorithms. The SOM is a topographic organization in which nearby locations in the map represent inputs with similar properties. The ART model allows the number of clusters to vary with problem size and lets the user control the degree of similarity between members of the same clusters by means of a user-defined constant called the vigilance parameter. ART networks are used for many patterns recognition tasks, such as automatic target recognition and seismic signal processing. This algorithms are less in practical use as some of the information is not available.

Reinforcement learning: Reinforcement learning is similar to supervised learning in that some feedback is given, however instead of providing a target output a reward is given based on how well the system performed. The aim of reinforcement learning is to maximize the reward the system receives through trial-and-error. This paradigm relates strongly with how learning works in nature, for example an animal might remember the actions it’s previously taken which helped it to find food (the reward). Types: Associative reinforcement learning, Deep reinforcement learning, Inverse reinforcement learning, Safe Reinforcement Learning and Partially Supervised Reinforcement Learning (PSRL). Applications of reinforcement learning were in the past limited by weak computer infrastructure. However, as Gerard Tesauro’s backgamon AI superplayer developed in 1990’s shows, progress did happen. That early progress is now rapidly changing with powerful new computational technologies opening the way to completely new inspiring applications. Training the models that control autonomous cars is an excellent example of a potential application of reinforcement learning. In an ideal situation, the computer should get no instructions on driving the car. The programmer would avoid hard-wiring anything connected with the task and allow the machine to learn from its own errors. In a perfect situation, the only hard-wired element would be the reward function.

2. Take any example and show the different steps of Hebb's Algorithm.

ANS)



Ans) $X_1 = \begin{bmatrix} 3 \\ 1 \\ -2 \end{bmatrix}$ $X_2 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}$

$w_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$ Learning constant $= c = 1$

steepness parameter $= \lambda = 1$

$net_i = w_i^T X$

$O_i = f(net_i) = \dfrac{2}{1+e^{-\lambda net_i}} - 1$

$\Delta w_i = c . O_i . X$

$w_{i+1} = w_i + \Delta w_i$

(1) $X = X_1 = \begin{bmatrix} 3 \\ 1 \\ -2 \end{bmatrix}$

$net_1 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ -2 \end{bmatrix}$

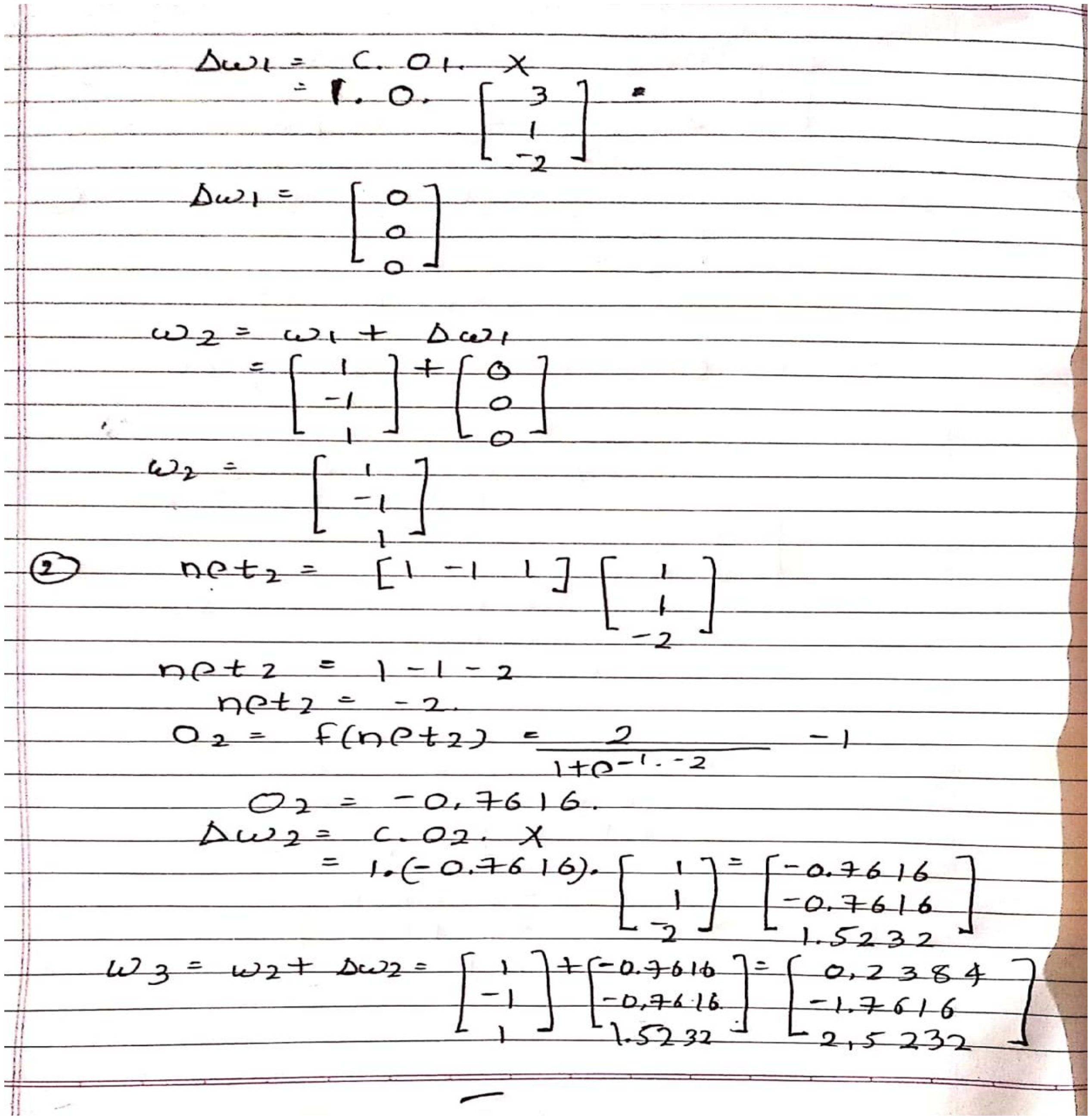
$= 3 - 1 - 2$

$net_1 = 0$

$O_1 = \dfrac{2}{1+e^{-1.0}} - 1$

$O_1 = \dfrac{2}{2} - 1$

$O_1 = 0$



$\Delta w_1 = c.\, o_1.\, X$

$= 1.\, 0.\, \begin{bmatrix} 3 \\ 1 \\ -2 \end{bmatrix} =$

$\Delta w_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$w_2 = w_1 + \Delta w_1$

$= \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$w_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$

(2) $net_2 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}$

$net_2 = 1 - 1 - 2$

$net_2 = -2.$

$O_2 = f(net_2) = \frac{2}{1+e^{-1.-2}} - 1$

$O_2 = -0.7616.$

$\Delta w_2 = c.\, o_2.\, X$

$= 1.(-0.7616).\begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} -0.7616 \\ -0.7616 \\ 1.5232 \end{bmatrix}$

$w_3 = w_2 + \Delta w_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.7616 \\ -0.7616 \\ 1.5232 \end{bmatrix} = \begin{bmatrix} 0.2384 \\ -1.7616 \\ 2.5232 \end{bmatrix}$

Date: 16 / 10 / 2021 **Signature of faculty in-charge**