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

Experiment No. 04

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

Signature of the Staff In-charge with date

Title: Delta learning rule

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

Expected Outcome of Experiment:

CO3 : To 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:

Delta learning rule : The Delta rule in machine learning and neural network environments is a specific type of backpropagation that helps to refine connectionist ML/AI networks, making connections between inputs and outputs with layers of artificial neurons. Developed by Widrow and Hoff, the delta rule, is one of the most common learning rules. It depends on supervised learning. This rule states that the modification in sympatric weight of a node is equal to the multiplication of error and the input. In Mathematical form the delta rule is as follows:

$$\Delta w = \eta (t - y) x_i$$

For a given input vector, compare the output vector is the correct answer. If the difference is zero, no learning takes place; otherwise, adjusts its weights to reduce this difference. The change in weight from ui to uj is: dwij = r* ai * ej. where r is the learning rate, ai represents the activation of ui and ej is the difference between the expected output and the actual output of uj. If the set of input patterns form an independent set then learn arbitrary associations using the delta rule. It has seen that for networks with linear activation functions and with no hidden units. The error squared vs. the weight graph is a paraboloid in n-space. Since the proportionality constant is negative, the graph of such a function is concave upward and has the least value. The vertex of this paraboloid represents the point where it reduces the error. The weight vector corresponding to this point is then the ideal weight vector. We can use the delta learning rule with both single output unit and several output units. While applying the delta rule assume that the error can be directly measured. The aim of applying the delta rule is to reduce the difference between the actual and expected output that is the error. In general, backpropagation has to do with recalculating input weights for artificial neurons using a gradient method. Delta learning does this using the difference between a target activation and an actual obtained activation. Using a linear

activation function, network connections are adjusted. Another way to explain the Delta rule is that it uses an error function to perform gradient descent learning. The delta learning rule is only valid for continuous activation functions and in the supervised training mode. The learning signal for this rule is called delta and is defined as follows : $r \triangleq [d_i - f(\mathbf{w}_i^t\mathbf{x})]f'(\mathbf{w}_i^t\mathbf{x})$.

The term $f'(\mathbf{w}_i^t\mathbf{x})$ is the derivative of the activation function f(net) computed for net = $\mathbf{w}_i^t\mathbf{x}$.

This learning rule can be readily derived from the condition of least squared error between oi and di. Calculating the gradient vector with respect to wi of the

$$E \triangleq \frac{1}{2}(d_i - o_i)^2$$

which is equivalent to

$$E = \frac{1}{2}\left[d_i - f(\mathbf{w}_i^t\mathbf{x})\right]^2$$

we obtain the error gradient vector value

$$\nabla E = -(d_i - o_i)f'(\mathbf{w}_i^t\mathbf{x})\mathbf{x}$$

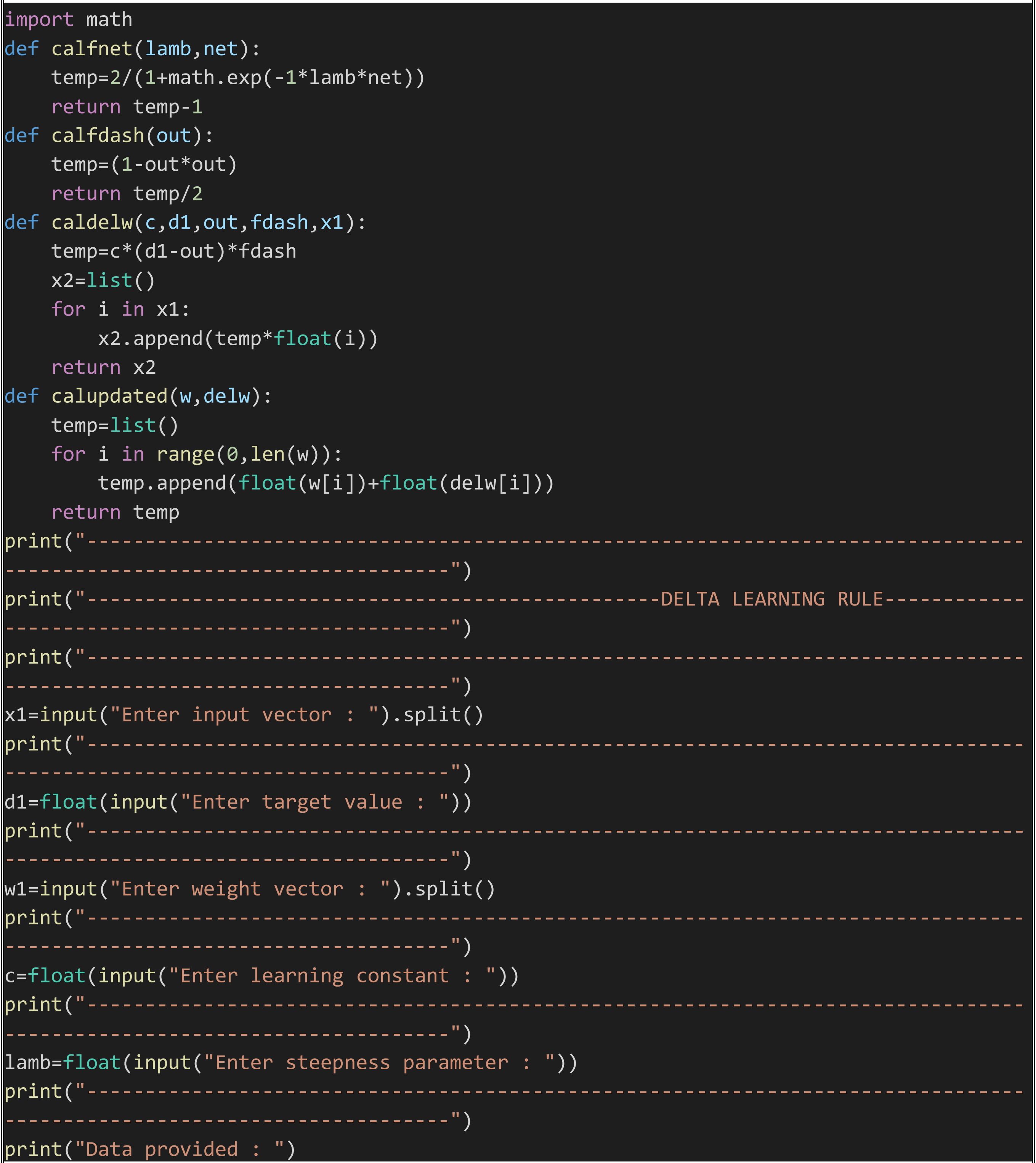
Since the minimization of the error requires the weight changes to be in the negative gradient direction, we take $\Delta\mathbf{w}_i = -\eta\nabla E$

where n is a positive constant. By combining above equations

$$\Delta\mathbf{w}_i = \eta(d_i - o_i)f'(net_i)\mathbf{x}$$

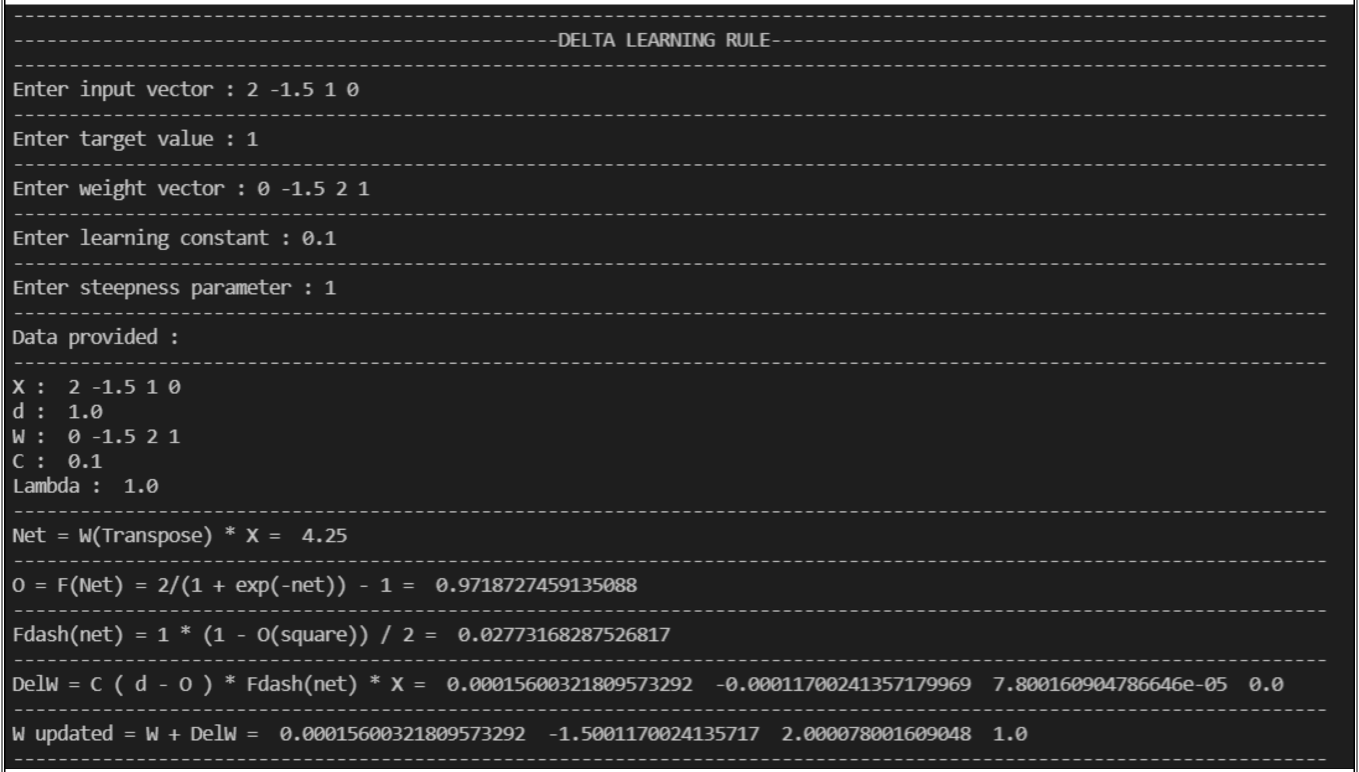
The delta rule was introduced only recently for neural network training (McClelland and Rumelhart 1986). This rule parallels the discrete perceptron training rule. It also can be called the continuous perceptron training rule. The delta learning rule can be generalized for multilayer networks

CODE :



```
import math
def calfnet(lamb,net):
    temp=2/(1+math.exp(-1*lamb*net))
    return temp-1
def calfdash(out):
    temp=(1-out*out)
    return temp/2
def caldelw(c,d1,out,fdash,x1):
    temp=c*(d1-out)*fdash
    x2=list()
    for i in x1:
        x2.append(temp*float(i))
    return x2
def calupdated(w,delw):
    temp=list()
    for i in range(0,len(w)):
        temp.append(float(w[i])+float(delw[i]))
    return temp
print("--------------------------------------------------------------------------------
--------------------------------------")
print("-------------------------------------------------DELTA LEARNING RULE------------
--------------------------------------")
print("--------------------------------------------------------------------------------
--------------------------------------")
x1=input("Enter input vector : ").split()
print("--------------------------------------------------------------------------------
--------------------------------------")
d1=float(input("Enter target value : "))
print("--------------------------------------------------------------------------------
--------------------------------------")
w1=input("Enter weight vector : ").split()
print("--------------------------------------------------------------------------------
--------------------------------------")
c=float(input("Enter learning constant : "))
print("--------------------------------------------------------------------------------
--------------------------------------")
lamb=float(input("Enter steepness parameter : "))
print("--------------------------------------------------------------------------------
--------------------------------------")
print("Data provided : ")
print("----------------------------------------------------------------------------------------------------------------------")
print("X : ",end=" ")
for i in x1:
    print(i,end=" ")
print("\nd : ",d1)
print("W : ",end=" ")
for i in w1:
    print(i,end=" ")
print("\nC : ",c)
print("Lambda : ",lamb)
print("----------------------------------------------------------------------------------------------------------------------")
net=0
for i in range(0,len(x1)):
    net=net+float(x1[i])*float(w1[i])
print("Net = W(Transpose) * X = ",net)
print("----------------------------------------------------------------------------------------------------------------------")
out=calfnet(lamb,net)
print("O = F(Net) = 2/(1 + exp(-net)) - 1 = ",out)
print("----------------------------------------------------------------------------------------------------------------------")
fdash=calfdash(out)
print("Fdash(net) = 1 * (1 - O(square)) / 2 = ",fdash)
print("----------------------------------------------------------------------------------------------------------------------")
delw=caldelw(c,d1,out,fdash,x1)
print("DelW = C ( d - O ) * Fdash(net) * X = ",end=" ")
for i in delw:
    print(i,end="  ")
print("\n----------------------------------------------------------------------------------------------------------------------")
w_updated=calupdated(w1,delw)
print("W updated = W + DelW = ",end=" ")
for i in w_updated:
    print(i,end="  ")
print("\n----------------------------------------------------------------------------------------------------------------------")
```

OUTPUT :



```
-----------------------------------------------------------------------------------------------------------
-------------------------------------------DELTA LEARNING RULE---------------------------------------------
-----------------------------------------------------------------------------------------------------------
Enter input vector : 2 -1.5 1 0
-----------------------------------------------------------------------------------------------------------
Enter target value : 1
-----------------------------------------------------------------------------------------------------------
Enter weight vector : 0 -1.5 2 1
-----------------------------------------------------------------------------------------------------------
Enter learning constant : 0.1
-----------------------------------------------------------------------------------------------------------
Enter steepness parameter : 1
-----------------------------------------------------------------------------------------------------------
Data provided :
-----------------------------------------------------------------------------------------------------------
X :  2 -1.5 1 0
d :  1.0
W :  0 -1.5 2 1
C :  0.1
Lambda :  1.0
-----------------------------------------------------------------------------------------------------------
Net = W(Transpose) * X =  4.25
-----------------------------------------------------------------------------------------------------------
O = F(Net) = 2/(1 + exp(-net)) - 1 =  0.9718727459135088
-----------------------------------------------------------------------------------------------------------
Fdash(net) = 1 * (1 - O(square)) / 2 =  0.02773168287526817
-----------------------------------------------------------------------------------------------------------
DelW = C ( d - O ) * Fdash(net) * X =  0.00015600321809573292  -0.00011700241357179969  7.800160904786646e-05  0.0
-----------------------------------------------------------------------------------------------------------
W updated = W + DelW =  0.00015600321809573292  -1.5001170024135717  2.000078001609048  1.0
-----------------------------------------------------------------------------------------------------------
```

Conclusion: Thus, we have successfully implemented Delta learning algorithm of Neural Network using python programming language. Also understood the theoretical part of delta learning rule. Solved examples related to delta learning rule to gain in depth knowledge.

Post Lab Descriptive Questions :

1. Explain different types of unsupervised techniques.

ANS) Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition. Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction. Below we'll define each learning method and highlight common algorithms and approaches to conduct them effectively.
Clustering: Clustering is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects

into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

Exclusive and Overlapping Clustering: Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. This can also be referred to as “hard” clustering. The K-means clustering algorithm is an example of exclusive clustering.

- **K-means clustering:** It is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group’s centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression.

Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. “Soft” or fuzzy k-means clustering is an example of overlapping clustering.

Hierarchical clustering: Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a “bottoms-up approach.”

Four different methods are commonly used to measure similarity:

- **Ward’s linkage:** This method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged.
- **Average linkage:** This method is defined by the mean distance between two points in each cluster
- **Complete (or maximum) linkage:** This method is defined by the maximum distance between two points in each cluster
- **Single (or minimum) linkage:** This method is defined by the minimum distance between two points in each cluster

Probabilistic clustering: A probabilistic model is an unsupervised technique that helps us solve density estimation or “soft” clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution.

- **Gaussian Mixture Models:** This models are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately.

Association Rules: An association rule is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines.

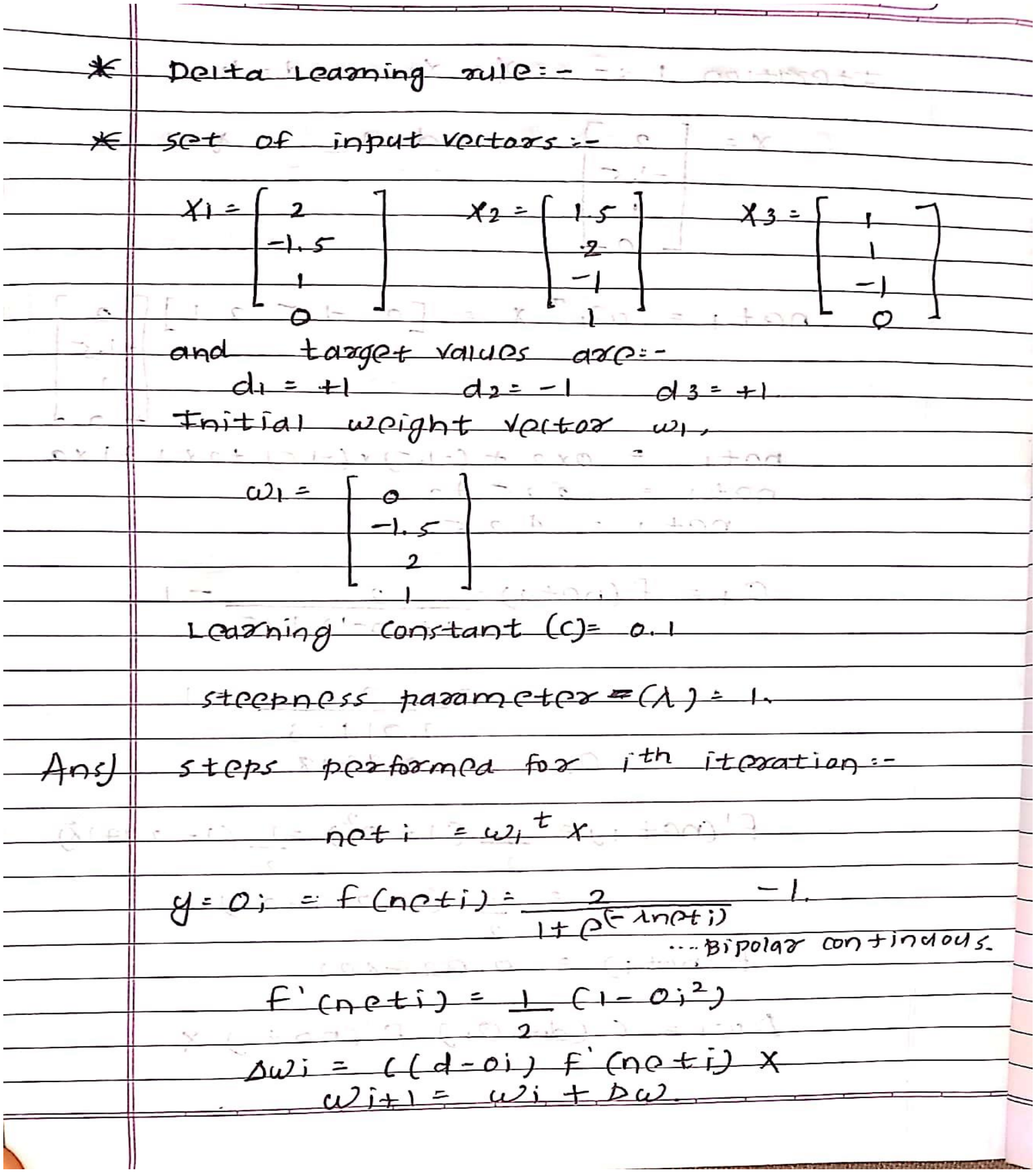
Dimensionality reduction: While more data generally yields more accurate results, it can also impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets. Dimensionality reduction is a technique used when the number of features, or dimensions, in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible.

Principal component analysis: Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component.

Singular value decomposition: Singular value decomposition (SVD) is another dimensionality reduction approach which factorizes a matrix, A, into three, low-rank matrices. SVD is denoted by the formula, A = USVT, where U and V are orthogonal matrices.

2. Take any example and show the different steps of Delta Algorithm.

ANS)



* Delta Learning rule:-

* set of input vectors :-

$$X_1 = \begin{bmatrix} 2 \\ -1.5 \\ 1 \\ 0 \end{bmatrix} \quad X_2 = \begin{bmatrix} 1.5 \\ 2 \\ -1 \\ 1 \end{bmatrix} \quad X_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \end{bmatrix}$$

and target values are:-

$d_1 = +1 \quad d_2 = -1 \quad d_3 = +1$

Initial weight vector w_1,

$$w_1 = \begin{bmatrix} 0 \\ -1.5 \\ 2 \\ 1 \end{bmatrix}$$

Learning constant $(c) = 0.1$

steepness parameter $= (\lambda) = 1$.

Ans) steps performed for i^{th} iteration :-

$$net_i = w_1^t x$$

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

... Bipolar continuous.

$$f'(net_i) = \frac{1}{2}(1 - O_i^2)$$

$$\Delta w_i = c(d - O_i) f'(net_i) X$$

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

Iteration 1 :- $X = X_1$, $d = d_1$.

$X = \begin{bmatrix} 2 \\ -1.5 \\ 1 \\ 0 \end{bmatrix}$ $d = 1$ $\lambda = 1$

$net_1 = w_1^T X = \begin{bmatrix} 0 & -1.5 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ -1.5 \\ 1 \\ 0 \end{bmatrix}$

$net_1 = 0 \times 2 + (-1.5) \times (-1.5) + 2 \times 1 + 1 \times 0$

$net_1 = 2.25 + 2$

$net_1 = 4.25$

$O_1 = f(net_1) = \dfrac{2}{1 + e^{-1 \times 4.25}} - 1$

$= \dfrac{2}{1.01426} - 1$

$O_1 = 0.9718.$

$f'(net_1) = \frac{1}{2}(1 - O_1^2) = \frac{1}{2}(1 - 0.9718^2)$

$= \frac{1}{2}(0.055604)$

$f'(net_1) = 0.027802$

$\Delta w_1 = c(d_1 - O_1) f'(net_1) X$

$= 0.1 \times (1 - 0.9718) \times 0.027802 \times X$

$$= 0.00007840 \times \begin{bmatrix} 2 \\ -1.5 \\ 1 \\ 0 \end{bmatrix}$$

$$\Delta w_1 = \begin{bmatrix} 0.0001568 \\ -0.0001176 \\ 0.00007840 \\ 0.0 \end{bmatrix}$$

$w_2 = w_1 + \Delta w_1$

$$= \begin{bmatrix} 0 \\ -1.5 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.0001568 \\ -0.0001176 \\ 0.00007840 \\ 0.0 \end{bmatrix}$$

$$w_2 = \begin{bmatrix} 0.0001568 \\ -1.5001176 \\ 2.00007840 \\ 1.0 \end{bmatrix}$$

Iteration 2 :- $X = X_2 \quad d = d_2$.

$$X = \begin{bmatrix} 1.5 \\ 2 \\ -1 \\ 1 \end{bmatrix} \qquad d_0 = -1$$

$net_2 = w_2^T X = [~~0.00015~~]$

$= [0.0001568 \quad -1.5001176 \quad 2.0007840 \quad 1] \begin{bmatrix} 1.5 \\ 2 \\ -1 \\ 1 \end{bmatrix}$

$net_2 = 0.0002352 - 3.0002352 - 2.0007840 + 1$

$net_2 = -4.000784.$

$O_2 = f(net_2) = \frac{2}{1+e^{+1 \times 4.000784}} - 1$

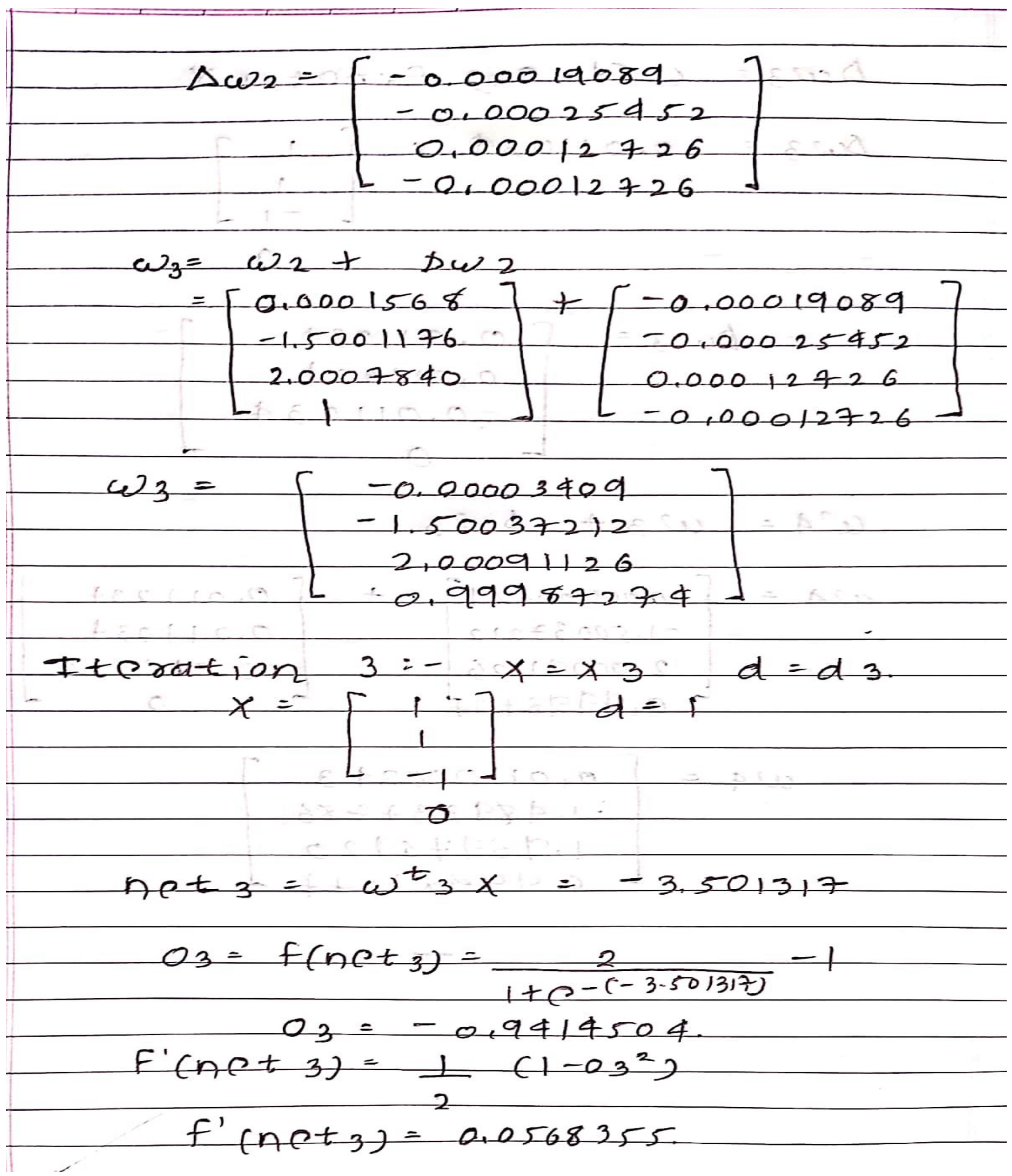
$O_2 = -0.964$

$f'(net_2) = \frac{1}{2}(1-0.964^2)$

$f'(net_2) = 0.035352.$

$\Delta w_2 = 0.1(-1-(-0.964)) \times 0.035352 \times X$

$\Delta w_2 = -0.00012726 \times \begin{bmatrix} 1.5 \\ 2 \\ -1 \\ 1 \end{bmatrix}$



$$\Delta w_2 = \begin{bmatrix} -0.00019089 \\ -0.00025452 \\ 0.00012726 \\ -0.00012726 \end{bmatrix}$$

$$w_3 = w_2 + \Delta w_2$$

$$= \begin{bmatrix} 0.0001568 \\ -1.5001176 \\ 2.0007840 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.00019089 \\ -0.00025452 \\ 0.00012726 \\ -0.00012726 \end{bmatrix}$$

$$w_3 = \begin{bmatrix} -0.00003409 \\ -1.50037212 \\ 2.00091126 \\ 0.99987274 \end{bmatrix}$$

Iteration 3 :- $x = x_3$ $d = d_3$.

$$x = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \end{bmatrix} \qquad d = 1$$

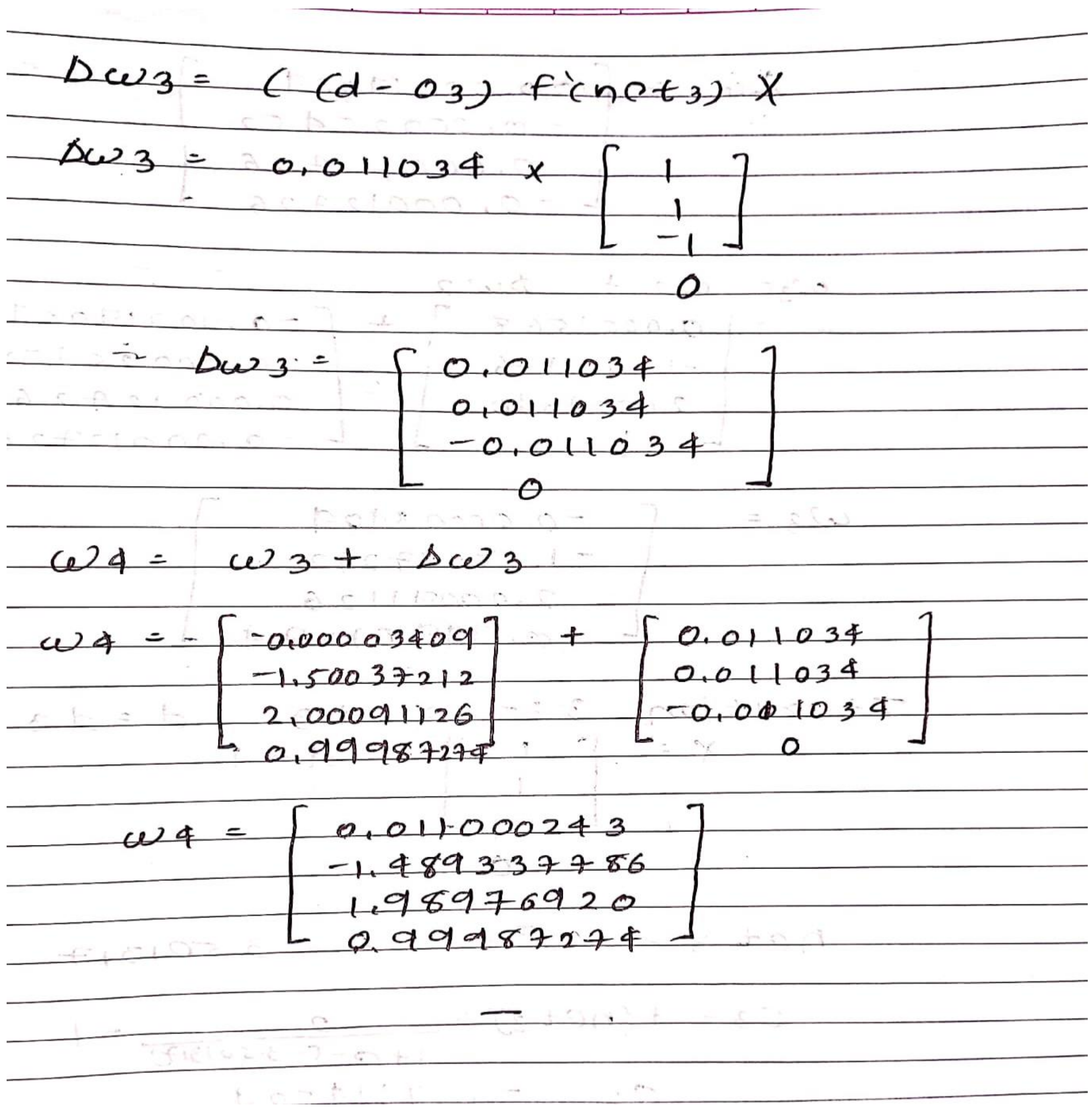
$$net_3 = w^t_3 x = -3.501317$$

$$O_3 = f(net_3) = \frac{2}{1+e^{-(-3.501317)}} - 1$$

$$O_3 = -0.9414504.$$

$$F'(net\ 3) = \frac{1}{2}(1 - O_3^2)$$

$$f'(net_3) = 0.0568355.$$



$$\Delta w_3 = c\,(d - o_3)\, f'(net_3)\, X$$

$$\Delta w_3 = 0.011034 \times \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \end{bmatrix}$$

$$\therefore \Delta w_3 = \begin{bmatrix} 0.011034 \\ 0.011034 \\ -0.011034 \\ 0 \end{bmatrix}$$

$$w_4 = w_3 + \Delta w_3$$

$$w_4 = \begin{bmatrix} -0.00003409 \\ -1.50037212 \\ 2.00091126 \\ 0.99987274 \end{bmatrix} + \begin{bmatrix} 0.011034 \\ 0.011034 \\ -0.011034 \\ 0 \end{bmatrix}$$

$$w_4 = \begin{bmatrix} 0.011000243 \\ -1.489337786 \\ 1.989769920 \\ 0.99987274 \end{bmatrix}$$

Date: 24 / 9 / 2021 **Signature of faculty in-charge**