**EXPERIMENT 4** Date:

**Problem Definition:** To implement the Delta Learning Rule in C

**Packages Used**: None

**Dataset Used:** None

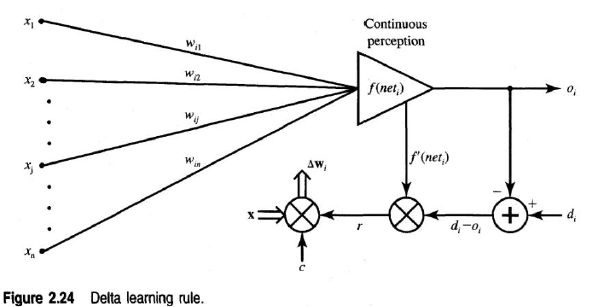
**Theory:**

The Delta Learning Rule, also known as the Widrow-Hoff Rule, is a fundamental algorithm used in artificial neural networks for adjusting the weights of the connections between neurons. This learning rule is primarily associated with single-layer networks, such as the Perceptron, and it is designed to minimize the error in predictions by adjusting the weights based on the difference between the predicted output and the actual target output.

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



The term f'(wti/x) is the derivative of the activation function f(net) computed for net = wti x. The explanation of the delta learning rule is shown in the below figure



Perceptron rule finds a successful weight vector when the training examples are linearly separable, but it can fail to converge if the examples are not linearly separable.

A second training rule, called the delta rule, is designed to overcome this difficulty.

If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.

The key idea behind the delta rule is to use **gradient descent** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

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 squared error defined as



which is equivalent to



we obtain the error gradient vector value



The components of the gradient vector are



Since the minimization of the error requires the weight changes to be in the negative gradient direction, we take



where η is a positive constant. We then obtain from Eqs. (2.38a) and (2.39)



or, for the single weight the adjustment becomes



Note that the weight adjustment as in the above equations is computed based on minimization of the squared error. Considering the use of the general learning rule and plugging in the learning signal as defined in (2.36), the weight adjustment becomes



**Code:**

#include <stdio.h>

#include <math.h>

int main()

{

// Define the inputs as a 2D array (matrix)

float inputs[3][4] = {

{1.0, -2.0, 0.0, -1.0}, // input1

{0.0, 1.5, -0.5, -1.0}, // input2

{-1.0, 1.0, 0.5, -1.0} // input3

};

float weights[] = {1.0, -1.0, 0.0, 0.5};

float l = 1.0;

float c = 0.1;

// Store targets in an array for easy access

float targets[3] = {-1.0, -1.0, 1.0};

// Iterate over each input

for (int iteration = 0; iteration < 3; iteration++) {

float net = 0.0;

// Calculate net input for the current iteration

for (int i = 0; i < 4; i++) {

net += inputs[iteration][i] \* weights[i];

}

printf("\nnet for input%d = %f", iteration + 1, net);

//calculataion of f(net)

float o = (1 - exp(-l \* net))/(1 + exp(-l \* net));

printf("\nf(net) for input%d = o%d = %f", iteration + 1,iteration + 1, net);

//deriavtive of f(net)

float der = 0.5 \* (1.0 - (o\*o));

printf("\nderivative(f(net)) for input%d = %f",iteration + 1, net);

// Update weights according to the perceptron learning rule

for (int i = 0; i < 4; i++) {

weights[i] += c \* (targets[iteration] - o) \* der \* inputs[iteration][i];

}

// Print updated weights

printf("\nw%d = [ ", iteration + 1);

for (int i = 0; i < 4; i++) {

printf("%f ", weights[i]);

}

printf("]");

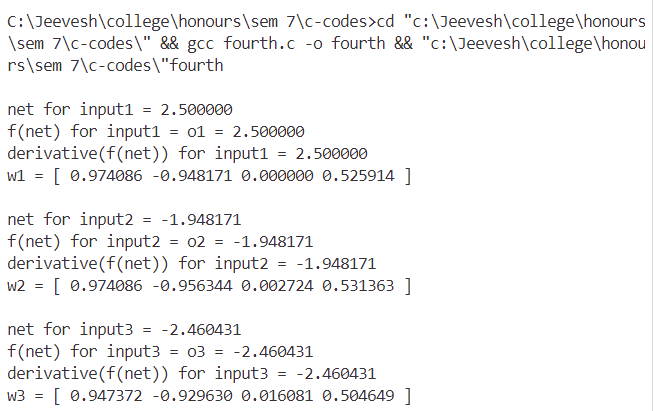
printf("\n");

}

return 0;

}

**Output:**

****

**Conclusion:**

Implemented the Delta Learning Rule successfully.