**EXPERIMENT 3** Date:

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

**Packages Used**: None

**Dataset Used:** None

**Theory:**

A **Perceptron** is one of the simplest types of artificial neural networks used for binary classification. It consists of a single neuron and is used for linearly separable data. The perceptron takes a set of inputs, assigns weights to them, applies a weighted sum, and then applies a step activation function to predict the output.

The **Perceptron Learning Rule** is a supervised learning algorithm used to train the perceptron by adjusting the weights iteratively based on the error between the predicted and actual output. The goal is to minimize the error and correctly classify the inputs over several iterations.

**Perceptron Structure:**

The perceptron consists of:

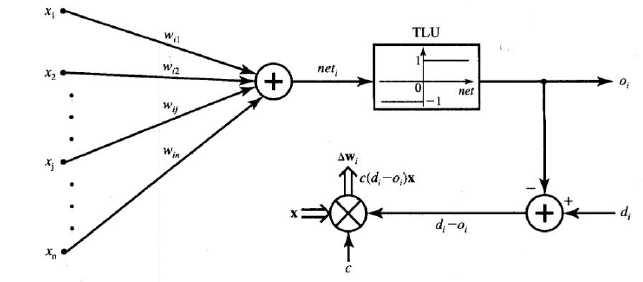
1. **Inputs** (x1,x2,...,xnx\_1, x\_2, ..., x\_nx1​,x2​,...,xn​) – Features that describe the data.
2. **Weights** (w1,w2,...,wnw\_1, w\_2, ..., w\_nw1​,w2​,...,wn​) – Weights assigned to each input.
3. **Bias** (bbb) – A threshold value that helps adjust the decision boundary.
4. **Activation Function** – Determines whether the neuron should fire or not based on the weighted sum of inputs.

**Perceptron Function:**

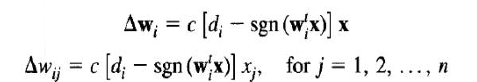
For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response (Rosenblatt 1958). Thus, learning is supervised and the learning signal is equal to



where oi = sgn(wit \* x) and di is the desired response as shown in below figure.



Weight adjustments in this method, DeltaWi and Delta Wij are obtained as follows



Note that this rule is applicable only for binary neuron response, and the above relationships express the rule for the bipolar binary case. Under this rule, weights are adjusted if and only if oi is incorrect. Error as a necessary condition of learning is inherently included in this training rule. Obviously, since the desired response is either 1 or -1, the weight adjustment equation reduces



where a plus sign is applicable when di = 1 , and sgn(wt \* x) = - 1 , and a minus sign is applicable when di = -1 , and sgn(wt \* x) = 1

**Code:**

#include <stdio.h>

int main()

{

float input1[] = {1.0, -2.0, 0.0, -1.0};

float input2[] = {0.0, 1.5, -0.5, -1.0};

float input3[] = {-1.0, 1.0, 0.5, -1.0};

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

float c = 0.1;

float d1=-1, d2=-1, d3=1;

//first iteration

float net = 0.0;

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

net = (input1[i]\*weights[i]) + net;

}

printf("net = %f\n", net);

float sgn = net>0 ? 1.0:-1.0;

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

weights[i] = weights[i] + c\*(d1-sgn)\*input1[i];

}

printf("w1 = [ ");

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

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

}

printf("]");

//second iteration

net = 0.0;

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

net = (input2[i]\*weights[i]) + net;

}

printf("\n\nnet = %f", net);

sgn = net>0 ? 1.0:-1.0;

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

weights[i] = weights[i] + c\*(d2-sgn)\*input2[i];

}

printf("\nw2 = [ ");

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

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

}

printf("]");

//third iteration

net = 0.0;

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

net = (input3[i]\*weights[i]) + net;

}

printf("\n\nnet = %f", net);

sgn = net>0 ? 1.0:-1.0;

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

weights[i] = weights[i] + c\*(d3-sgn)\*input3[i];

}

printf("\nw3 = [ ");

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

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

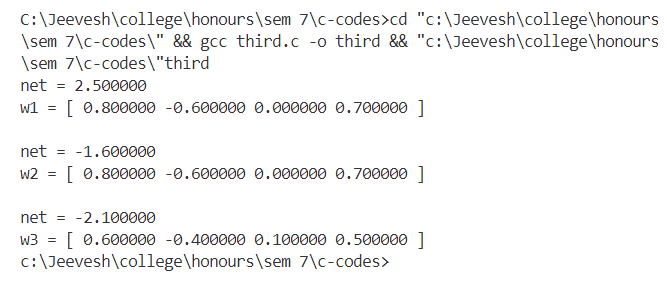
}

printf("]");

return 0;

}

**Output:**

****

**Conclusion:**

Implemented the Perceptron Learning Rule successfully.