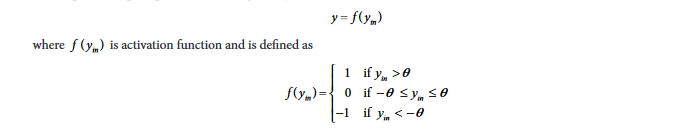
|  |  |
| --- | --- |
| **Name:** | Bodhisatya Ghosh |
| **Branch:** | CSE – Data Science |
| **Batch:** | B |
| **Course:** | Soft Computing |
| **Experiment no:** | 2 |

**Aim:** To implement a perceptron for a given problem statement and design ANN for the same using Joone Editor.

**Theory:** Perceptron networks come under single-layer feed-forward networks and are also called simple perceptrons. The perceptron network consists of three units, namely, sensory unit (input unit), associator unit (hidden unit), response unit (output unit). The sensory units are connected to associator units with fixed weights having values 1, 0 or -1, which are assigned at random. The binary step with fixed threshold q is used as activation for associator. The output of the perceptron network is given by:



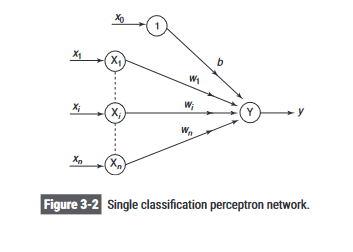


The perceptron learning rule is used in the weight updation between the associator unit and the response unit. For each training input, the net will calculate the response and it will determine whether or not an error has occurred. The error calculation is based on the comparison of the values of targets with those of the calculated outputs. The weights on the connections from the units that send the nonzero signal will get adjusted suitably. The weights will be adjusted on the basis of the learning rule if an error has occurred for a particular training pattern, i.e.



The entire loop of the training process continues until the training input pair is presented to the network. The training (weight updation) is done on the basis of the comparison between the calculated and desired output. The loop is terminated if there is no change in weight.

Single classification perceptron network:

****

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int activation\_function(float yin,float theta){

    if(yin > theta)

    {

        return 1;

    }

    else if((yin >= -1\*theta) && (yin <= theta))

    {

        return 0;

    }

    else if (yin < theta)

    {

        return -1;

    }

}

float \* step\_calc(int step\_num,int\*\* vector,int\* targets,int bias,float theta,int learning\_rate,float\* weights,int n)

{

    float yin = weights[0] \* bias;

    for (int i = 1; i < n+1; i++)

    {

        yin = yin + (vector[step\_num-1][i-1] \* weights[i]);

    }

    int y = activation\_function(yin,theta);

    // printf("Y after activation: %d\n",y);

    if(targets[step\_num-1] == y)

    {

        return weights;

    }

    // printf("Y after activation: %d\n",y);

    weights[0] = weights[0] + (learning\_rate\*targets[step\_num-1]);

    for (int i = 1; i < n+1; i++)

    {

        weights[i] = weights[i] + (learning\_rate \* targets[step\_num-1] \* vector[step\_num-1][i-1]);

    }

    return weights;

}

int main()

{

    int n,vec,epoch,bias,learning\_rate,threshhold;

    printf("Enter number of vectors: ");

    scanf("%d",&vec);

    printf("Enter number of elements per vector: ");

    scanf("%d",&n);

    int\*\* vector = malloc(vec \* sizeof(int \*));

    for (int i = 0; i < vec; i++)

    {

        vector[i] = (int\*)malloc(n \* sizeof(int));

    }

    int\* targets = malloc(vec \* sizeof(int));

    float\* weights = malloc((n+1) \* sizeof(float));

    for (int i = 0; i < vec; i++)

    {

        printf("\nFor vector %d\n",i+1);

        for(int j = 0; j < n; j++)

        {

            printf("Enter element %d: ",j+1);

            scanf("%d",&vector[i][j]);

        }

        printf("Enter target for vector %d: ",i+1);

        scanf("%d",&targets[i]);

    }

    for (int i = 0; i < n+1; i++)

    {

        printf("Enter initial weight %d: ",i);

        scanf("%d",&weights[i]);

    }

    printf("Enter threshhold: ");

    scanf("%f",&threshhold);

    printf("Enter bias: ");

    scanf("%d",&bias);

    printf("Enter learning rate: ");

    scanf("%d",&learning\_rate);

    printf("Enter number of epochs: ");

    scanf("%d",&epoch);

    for(int ep = 0;ep < epoch; ep++)

    {

        printf("\nEpoch %d\n",ep+1);

        for(int step = 1; step <= vec; step++)

        {

            weights = step\_calc(step,vector,targets,bias,0.2,learning\_rate,weights,n);

        }

        for (int i = 0; i < n+1; i++)

        {

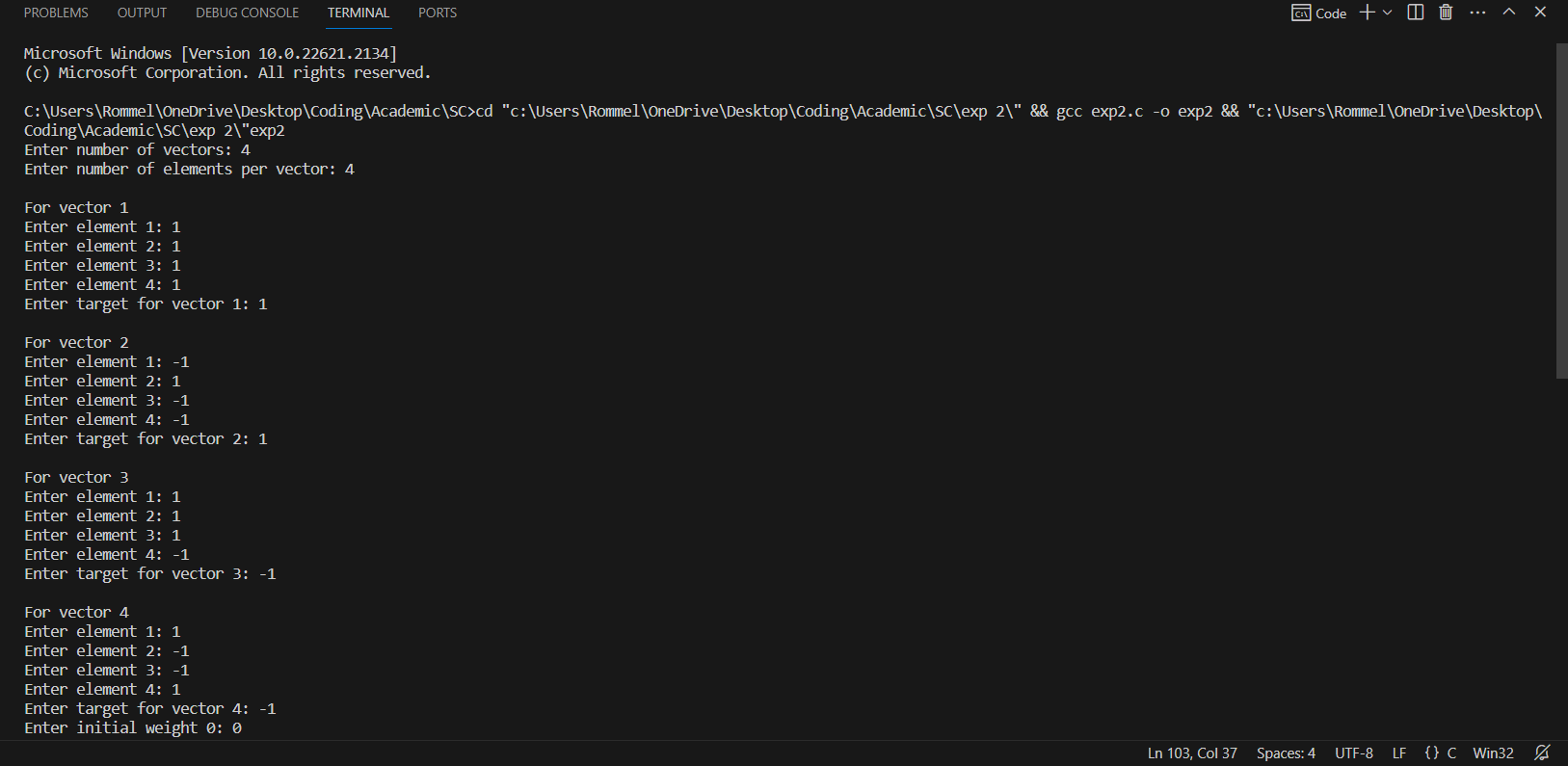
            printf("Weight %d after epoch %d: %f\n",i,ep+1,weights[i]);

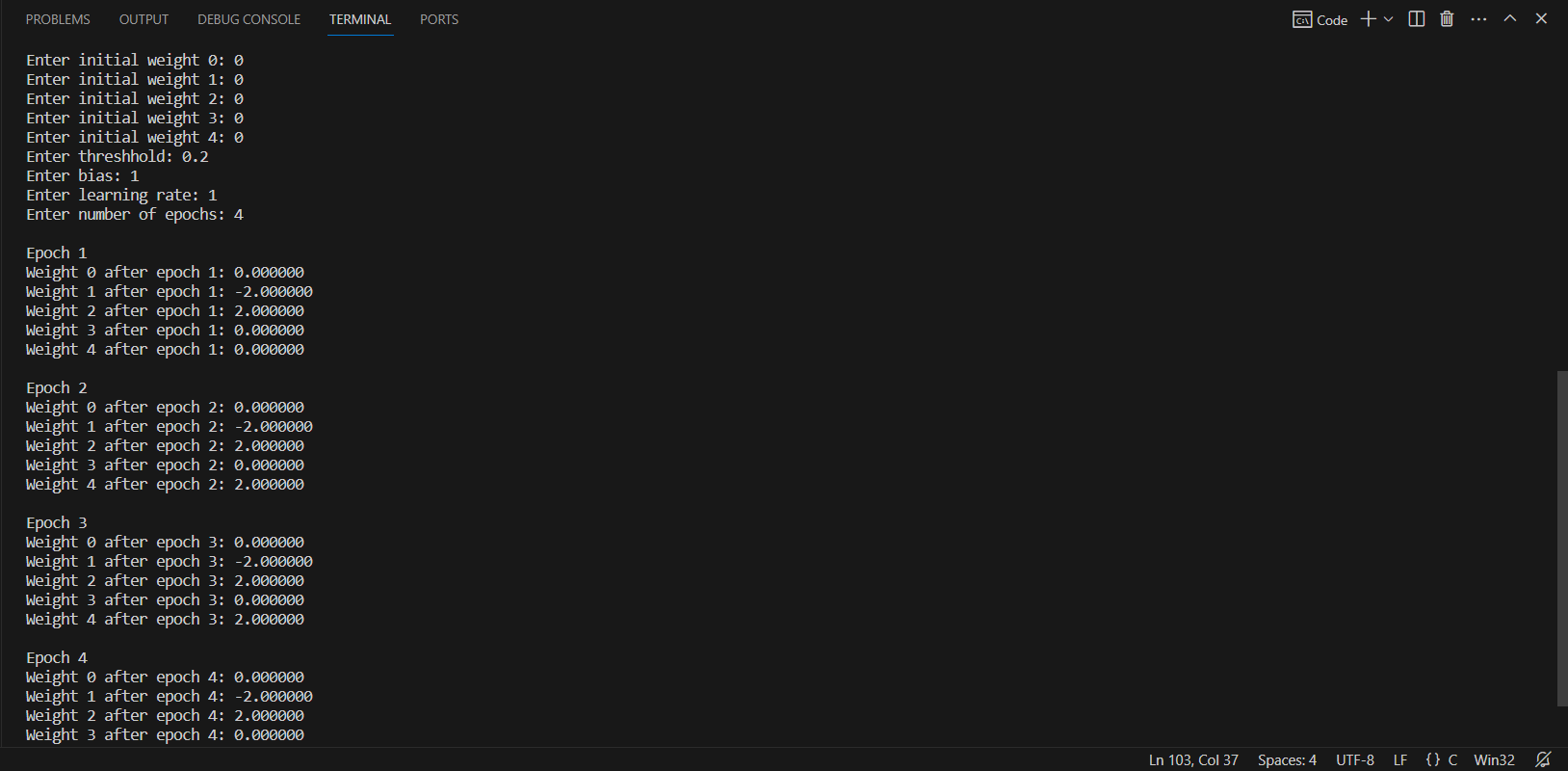
        }

    }

}

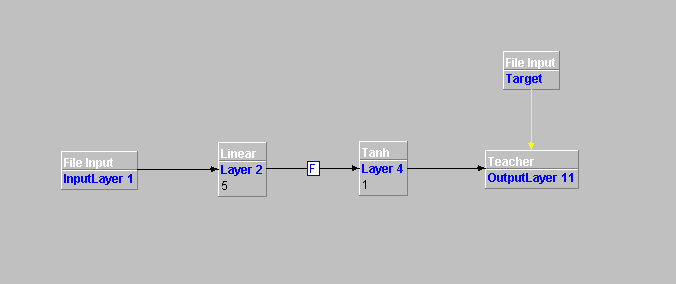
**Results:**



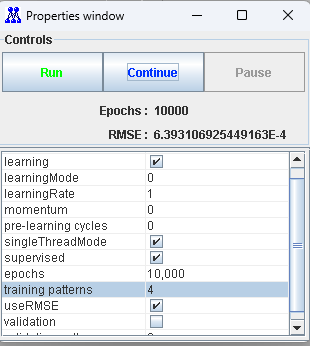
****

**Joone Editor:-**

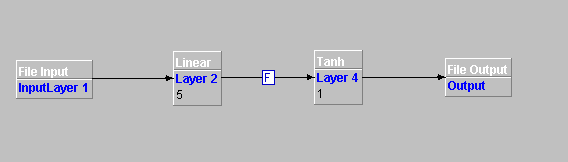
Step 1: Draw a network with a teacher in it.



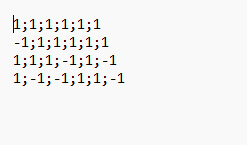
Step 2: Go to the control panel and do the required changes.



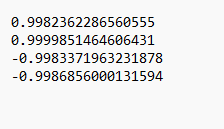
Step 3: After training the layers we test it by forming the model.



Step 4: Give the input file.



Step 5: Make changes in the control panel run the model and then store the output in the file.



Here we observe that our Joone model is correct and works perfectly. It adjusted its weights so as to meet the target values.

**CONCLUSION: -** In this experiment we studied about the perceptron model and joone editor and how to use it.