|  |  |
| --- | --- |
| **Name:** | Bodhisatya Ghosh |
| **Branch:** | CSE – Data Science |
| **Batch:** | B |
| **Course:** | Soft Computing |
| **Experiment no:** | 4 |

**Aim:** To implement KSOFM for a given pattern classification problem.

**Theory:** Feature mapping is a process which converts the patterns of arbitrary dimensionality into a response of one- or twodimensional arrays of neurons, i.e., it converts a wide pattern space into a typical feature space. The network performing such a mapping is called feature map. Apart from its capability to reduce the higher dimensionality, it has to preserve the neighbourhood relations of the input patterns, i.e., it has to obtain a topology preserving map. For obtaining such feature maps, it is required to find a self-organizing neural array which consists of neurons arranged in a one-dimensional array or a two-dimensional array. At the time of self-organization, the weight vector of the cluster unit which matches the input pattern very closely is chosen as the winner unit. The closeness of weight vector of cluster unit to the input pattern may be based on the square of the minimum Euclidean distance. The weights are updated for the winning unit and its neighboring units. It should be noted that the weight vectors of the neighboring units are not close to the input pattern and the connective weights do not multiply the signal sent from the input units to the cluster units until dot product measure of similarity is being used.

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void print\_array(float\* arr,int num,char\* c){

    printf("%s",c);

    for(int i = 0; i < num ; i++){

        printf("%f ",arr[i]);

    }

    printf("\n");

}

void print\_array\_int(int\* arr,int num,char\* c){

    printf("%s",c);

    for(int i = 0; i < num ; i++){

        printf("%d ",arr[i]);

    }

    printf("\n");

}

void print\_weights(float\*\* arr,int row, int col){

    for(int i = 0; i < row ; i++){

        for(int j = 0; j < col; j++){

            printf("%f ",arr[i][j]);

        }

        printf("\n");

    }

    printf("\n");

}

float\* allocate\_1Dmem(float\* arr,int size,char\* desc){

    printf("Allocating memory for %s\n",desc);

    arr = (float\*)malloc(size \* sizeof(float));

    return arr;

}

int\* allocate\_1Dmem\_int(int\* arr,int size,char\* desc){

    printf("Allocating memory for %s\n",desc);

    arr = (int\*)malloc(size \* sizeof(int));

    return arr;

}

float\*\* fill\_weights(float\*\* weights,int i,int j,char\* desc){

    printf("Allocating %s\n",desc);

    weights = (float\*\*)malloc(i \* sizeof(float\*));

    for(int x = 0; x<j ; x++){

        weights[x] = (float\*)malloc(j \* sizeof(float));

    }

    for(int a = 0; a < i; a++){

        for(int b = 0; b < j; b++){

            printf("Enter weight for cluster %d to vector %d: ",a+1,b+1);

            scanf("%f",&weights[a][b]);

        }

    }

    printf("\n");

    return weights;

}

float\*\* allocate\_2Dmem(float\*\* vector,int i,int j,char\* desc){

    printf("Allocating memory for %s\n",desc);

    vector = (float\*\*)malloc(i \* sizeof(float\*));

    for(int x = 0; x<j ; x++){

        vector[x] = (float\*)malloc(j \* sizeof(float));

    }

    for(int a = 0; a < i; a++){

        printf("For vector %d\n",a+1);

        for(int b = 0; b < j; b++){

            printf("Enter value %d: ",b+1);

            scanf("%f",&vector[a][b]);

        }

    }

    printf("\n");

    return vector;

}

int calc\_winning\_cluster(float\* vector, float\*\* weights, int clusters, int vect\_cnt){

    float dist[] = {0,0};

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

        for(int j = 0; j < vect\_cnt; j++){

            dist[i] = dist[i] + pow((weights[i][j]-vector[j]),2);

        }

    }

    if(dist[0] >= dist[1]){

        return 1;

    }

    else{

        return 0;

    }

}

float\* update\_weights(float\* weights, float\* vector,float learning\_rate, int vector\_len){

    print\_array(vector,vector\_len,"Vector: ");

    for(int i = 0; i < vector\_len; i++){

        weights[i] = weights[i] + learning\_rate \* (vector[i] - weights[i]);

        // printf("Weight from vector %d is %f:",i,weights[i]);

        // printf("\n");

    }

    print\_array(weights,vector\_len,"Updated cluster weight: ");

    return weights;

}

float\*\* KSOFM(float\*\* vectors, float\*\* weights, int\* winning\_cluster, int num\_of\_vectors, int num\_of\_clusters, float learning\_rate, int vect\_len){

    for (int i = 0; i < num\_of\_vectors; i++)

    {

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

        winning\_cluster[i] = calc\_winning\_cluster(vectors[i], weights, num\_of\_clusters, num\_of\_vectors);

        weights[winning\_cluster[i]] = update\_weights(weights[winning\_cluster[i]],vectors[i],learning\_rate,vect\_len);

    }

    return weights;

}

void main()

{

    int\* winning\_cluster;     //winning cluster info

    float\*\* vectors;       //input vectors

    float\*\* weights;      //vector to cluster weight

    int vect\_len,input,clusters;      // vector len, vector num, cluster num

    float learning\_rate;     //learning rate and target

    printf("Enter number of vectors: ");

    scanf("%d",&input);

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

    scanf("%d",&vect\_len);

    vectors = allocate\_2Dmem(vectors,input,vect\_len,"vectors");

    printf("Enter number of clusters: ");

    scanf("%d",&clusters);

    winning\_cluster = allocate\_1Dmem\_int(winning\_cluster,input,"winning cluster info");

    weights = fill\_weights(weights,clusters,input,"weights");

    printf("Enter learning rate: ");

    scanf("%f",&learning\_rate);

    printf("\n");

    weights = KSOFM(vectors, weights, winning\_cluster, input, clusters, learning\_rate, vect\_len);

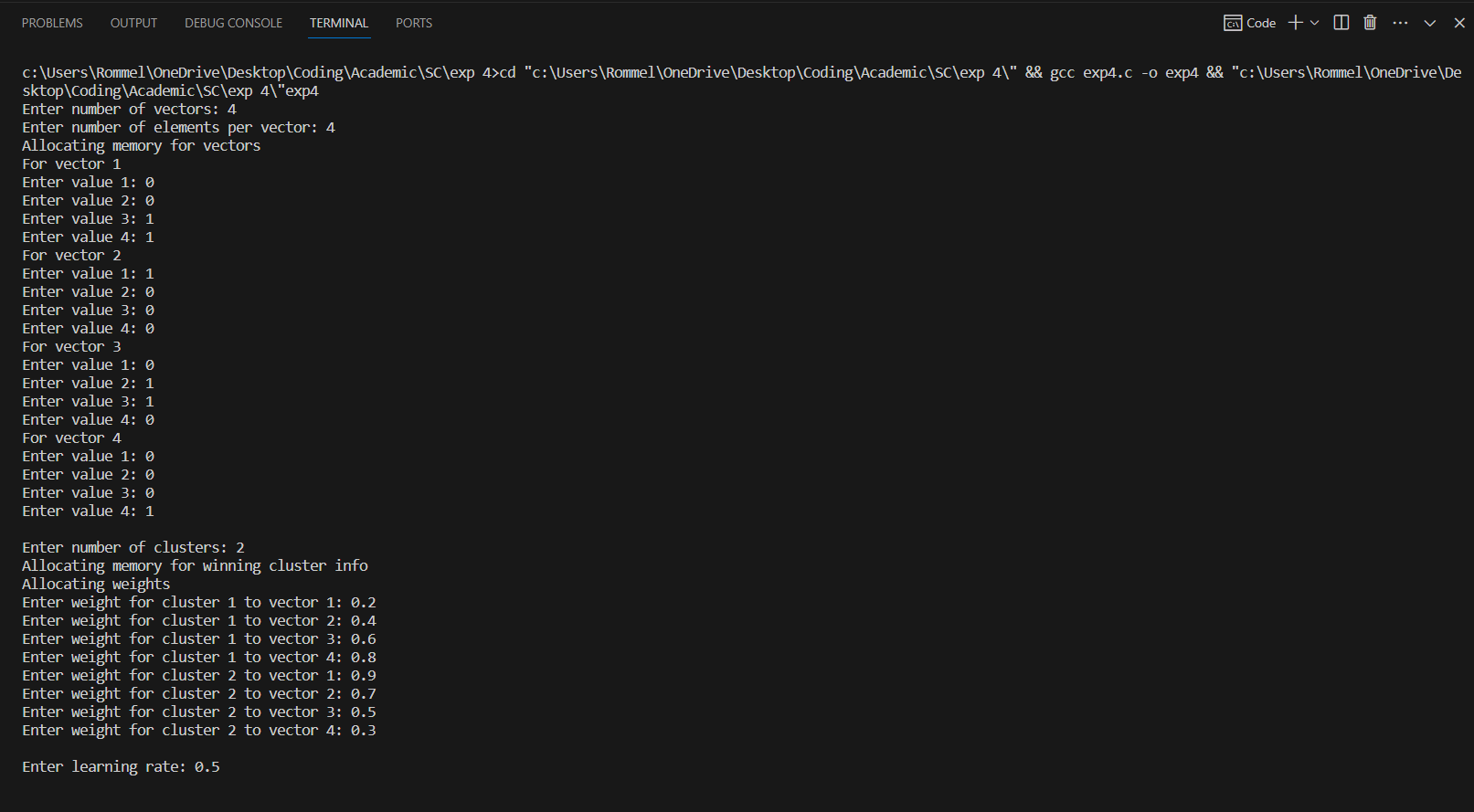
    printf("\nFinal weights are : \n");

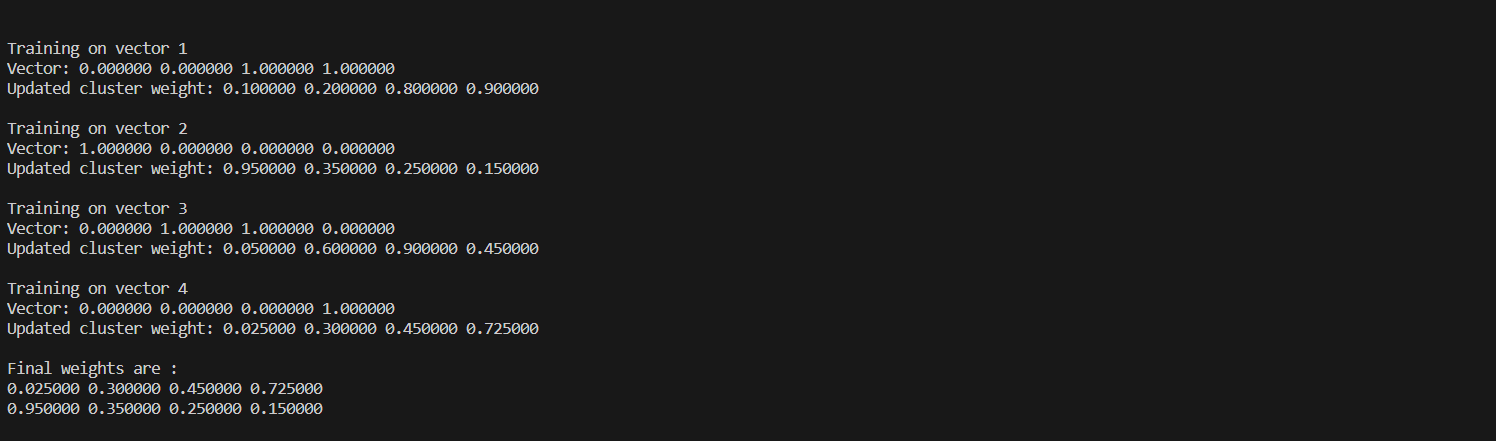
    print\_weights(weights,clusters,input);

}

//4 4 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 1 2 0.2 0.4 0.6 0.8 0.9 0.7 0.5 0.3 0.5

**Results:**





**CONCLUSION: -** In this experiment we studied about unsupervised learning using Kohonen Self-Organising Feature Maps