**Practical : 9**

**Topic :**

Implementation of SOM based procedures for dataset analysis.

**Implementation :**

1. A teacher wants to form the clusters of 4 students based on their performance in 4 subjects: English, Hindi, Maths, and Science respectively. The performance of the students is judged by the parameters high and low. Consider low as 0 and high as 1 as given in the matrix. Student A has a low performance in English and Hindi whereas high in Math and Science. The performance of student B is only high in English. Student C was recorded high in Hindi and Math with a low score in English and Science. Lastly, the performance of student D was only observed high in Science. Assume that there are two clusters formed and the initial learning rate is 0.5. Construct KSOM for the above situation with the following matrix.

#include <iostream>

#include <conio.h>

#include <iomanip>

#include <cmath>

using namespace std;

const int maxClusters = 2;

const int vectors = 4;

const int vectorLen = 4;

const double decayRate = 0.96;

const double minAlpha = 0.01;

double alpha = 0.5;

double d[maxClusters];

double w[maxClusters][vectorLen] = {{0.2, 0.6, 0.5, 0.9},

{0.8, 0.4, 0.7, 0.3}

};

int pattern[vectors][vectorLen] = {{1, 1, 0, 0},

{0, 0, 0, 1},

{1, 0, 0, 0},

{0, 0, 1, 1}

};

int tests[vectors][vectorLen] = {{0, 0, 1, 1},

{1, 0, 0, 0},

{0, 1, 1, 0},

{0, 0, 0, 1}

};

int minimum(double valueA, double valueB) {

if (valueA > valueB) {

return 1;

} else {

return 0;

}

}

void computeInput(int vectorNumber) {

d[0] = 0.0;

d[1] = 0.0;

for (int i = 0; i <= (maxClusters - 1); i++) {

for (int j = 0; j <= (vectors - 1); j++) {

d[i] += pow((w[i][j] - tests[vectorNumber][j]), 2);

}

}

}

void training() {

int iterations = 0;

int dMin = 0;

do {

iterations += 1;

for (int vecNum = 0; vecNum <= (vectors - 1); vecNum++) {

computeInput(vecNum);

dMin = minimum(d[0], d[1]);

//Update the weights on the winning unit.

for (int i = 0; i <= (vectors - 1); i++) {

w[dMin][i] = w[dMin][i] + (alpha \* (pattern[vecNum][i] -

w[dMin][i]));

}

}

//Reduce the learning rate.

alpha = decayRate \* alpha;

} while (alpha > minAlpha);

cout << "Iterations: " << iterations << "\n\n";

}

void showResult() {

int dMin;

//Print clusters created.

cout << "-------------------------------------------------------------\n";

cout << "Clusters for training input:" << endl;

for (int vecNum = 0; vecNum <= (vectors - 1); vecNum++) {

computeInput(vecNum);

dMin = minimum(d[0], d[1]);

cout << "\nVector (";

for (int i = 0; i <= (vectors - 1); i++) {

cout << pattern[vecNum][i] << ", ";

}

cout << ") fits into cluster " << dMin << endl;

}

//Print weight matrix.

cout << "-------------------------------------------------------------\n";

for (int i = 0; i <= (maxClusters - 1); i++) {

cout << "Weights for Node " << i << " connections:" << endl;

for (int j = 0; j <= (vectorLen - 1); j++) {

cout << w[i][j] << ", ";

}

cout << "\n\n";

}

cout << "-------------------------------------------------------------\n";

}

int main() {

cout << fixed << setprecision(3) << endl;

training();

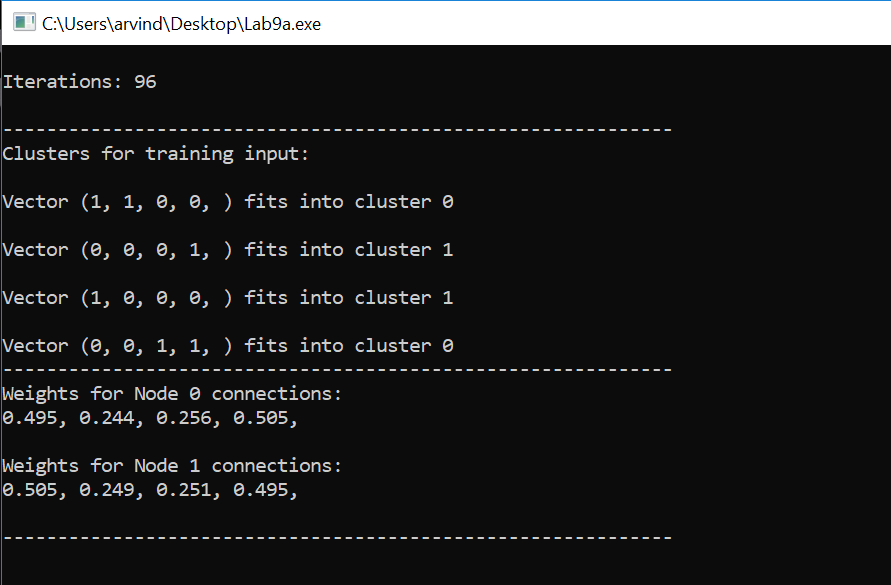
showResult();

getch();

return 0;

}

Output :



1. Problem described in (a) extended to ‘n’ number of students .

#include <iostream>

#include <conio.h>

#include <iomanip>

#include <cmath>

#include<vector>

using namespace std;

#define vint vector<int>

const int maxClusters = 2;

const int vectorLen = 4;

const double decayRate = 0.82;

const double minAlpha = 0.01;

double alpha = 0.5;

int vectors = 4;

double d[maxClusters];

double w[maxClusters][vectorLen] = {

{0.2, 0.4, 0.7, 0.3}, {0.4, 0.6, 0.5, 0.9}

};

int pattern[10][10];

int tests[10][vectorLen] = {{0, 0, 0, 1},

{0, 0, 0, 0},

{0, 0, 1, 1},

{0, 0, 1, 0},

{0, 1, 0, 0},

{0, 1, 0, 1},

{0, 1, 1, 0},

{1, 0, 0, 1},

{0, 1, 1, 0},

{0, 0, 1, 1}

};

int minimum(double valueA, double valueB) {

if (valueA > valueB) {

return 1;

} else {

return 0;

}

}

void computeInput(int vectorNumber) {

d[0] = 0.0;

d[1] = 0.0;

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

for (int j = 0; j < vectors; j++) {

d[i] += pow((w[i][j] - tests[vectorNumber][j]), 2);

}

}

}

void training() {

int iterations = 0;

int dMin = 0;

do {

iterations += 1;

for (int vecNum = 0; vecNum <= (vectors - 1); vecNum++) {

computeInput(vecNum);

dMin = minimum(d[0], d[1]);

//Update the weights on the winning unit.

for (int i = 0; i <= (vectors - 1); i++) {

w[dMin][i] += (alpha \* (pattern[vecNum][i] - w[dMin][i]));

}

}

//Reduce the learning rate.

alpha = decayRate \* alpha;

} while (alpha > minAlpha);

cout << "Iterations: " << iterations << "\n\n";

}

void showResult() {

int dMin;

//Print weight matrix.

cout << "-------------------------------------------------------------\n";

for (int i = 0; i <= (maxClusters - 1); i++) {

cout << "Weights for Node " << i << " connections:" << endl;

for (int j = 0; j <= (vectorLen - 1); j++) {

cout << w[i][j] << ", ";

}

cout << "\n\n";

}

//Print clusters created.

cout << "-------------------------------------------------------------\n";

cout << "Clusters for training input:" << endl;

for (int vecNum = 0; vecNum <= (vectors - 1); vecNum++) {

computeInput(vecNum);

dMin = minimum(d[0], d[1]);

cout << "\nVector (";

for (int i = 0; i <= (vectorLen - 1); i++) {

cout << pattern[vecNum][i] << ", ";

}

cout << ") fits into cluster " << dMin << endl;

}

cout << "-------------------------------------------------------------\n";

}

int main() {

cout << "Enter Number of inputs :";

cin >> vectors;

//pattern = vector<vint> (vectors, vint(vectorLen, 0));

cout << "Enter " << vectors << " input patterns :\n";

for (int i = 0; i < vectors; ++i)

{

for (int j = 0; j < vectorLen; ++j)

{

cin >> pattern[i][j];

}

}

cout << "-------------------------------------------------------------\n";

cout << fixed << setprecision(3) << endl;

training();

showResult();

getch();

return 0;

}

Output :

