Digit Recognition in C Language

<u>Structure of Neural Network</u>
Input Layer - 50+1
Hidden Layer - 15+1
Output Layer - 10

10 samples for each digit (7 for training and 3 for testing).

Code:

```
/******************************************************************************/

#include <bits/stdc++.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fcntl.h>
#define NUMPAT 70
#define NUMIN  50
#define NUMHID 15
#define NUMOUT 10
#define rando()((double)rand()/((double)RAND_MAX+1))

using namespace std;
int main()
{
   int i,j,k,p,np,op,ranpat[NUMPAT+1],epoch;
   int NumPattern=NUMPAT,NumInput=NUMIN,NumHidden=NUMHID,NumOutput=NUMOUT;
   double Input[NUMPAT+1][NUMIN+1] =

{{7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7}
,

{0,0,0,0,0,0,7,7,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,7,7,7,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,7,7,7,0,0,7,0,0,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,0,7,0,0,7,7,7,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,7,7,7,7,7,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,0,0,7,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,0,0,7,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,7,7,7,7,7,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,7,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,7,7,7,7,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,7,7,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,7,0,0,7,0,7,0,0,0,0,7,0,0,0,7,0,0,7,7,0,0,0,7,7,7,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,0,0,0,7,0,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,0,0,7,7,0,0,0,0,0,7,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,7,7,7,7,0,0,0,0,0},

{0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,0,0,0,7,0,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,0,0,7,7,7,0,0,0,0,0,0,0,0,0,0},

{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7},

{0,0,0,0,0,0,0,7,7,0,0,7,0,0,7,0,0,0,0,7,0,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,7,0,0,7,0,0,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,0,7,0,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0},

{7,7,7,7,7,7,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,7,7,7,7,7},

{0,7,7,7,0,7,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0},
{0,0,7,7,0,0,7,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,7,0,0,7,0,0,7,7,7,0,0,0,0,0},

{0,0,7,0,0,0,7,0,7,0,7,0,0,7,0,0,0,7,0,0,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7},

{0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,7,0,7,0,0,7,0,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,7,7,0,0,7,0,7,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7.7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0},

{0,0,0,0,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,7,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,7,0,0,0,7,7,0,0,7,0,7,0,7,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,0,7,0,7,0,0,7,0,0,7,7,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0},

{7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7},

{7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,7,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},

{7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,0,0,0,0,0,7,0,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,7,0,7,0,0,0,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,7,7,7,7,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,7,7,7,7,0,0,0,0,7,0,7,7,7,7,0,0,0,0,0,0,0,0,0,0},

{7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,7,7,7,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,0},

{7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7},

{0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0},

{0,0,7,7,0,0,7,0,0,7,0,7,0,0,7,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,7,7,7,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,7,7,0,7,7,0,0,7,0,7,0,0,7,0,0,7,7,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,7,7,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,7,0,7,0,0,7,0,0,7,7,0,0,0,0,0,0},

{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7},

{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7},

{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,7,7,7,7,7,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,0},

{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0},

{0,0,7,7,7,0,7,0,0,7,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,7,7,7,7,7,0,0,7,0,0,0,0,7,0,0,0,7,0,7,7,7,7,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0},

{7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7},

{0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0},

{0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0},

{0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0,0,7,7,0,0,7,0,0,7,0,7,0,0,0,7,0,7,7,7,0},

{0,0,0,0,0,0,0,7,0,7,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,7,0,0,7,7,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,7,0,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0},

{0,0,0,0,0,0,0,7,7,7,0,7,0,0,7,0,7,0,0,7,0,7,7,7,7,0,7,0,0,7,0,7,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},

{7,7,7,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7},

{0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,7,7,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},

{0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,0,7,0,0,7,7,7,0,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0},
{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7},

{0,0,0,0,0,0,0,0,0,0,0,0,7,7,0,0,7,0,0,7,0,7,0,0,7,0,0,7,7,7,0,0,0,0,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0}}
;

   double Target[NUMPAT+1][NUMOUT+1]=
   {{1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {1,0,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,1,0,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0,0},
   {0,0,0,0,0,1,0,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,1,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,1,0,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,1,0,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,1,0},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1},
    {0,0,0,0,0,0,0,0,0,1}};

    double
SumH[NUMPAT+1][NUMHID+1],WeightIH[NUMIN+1][NUMHID+1],Hidden[NUMPAT+1][NUMHI
D+1],WeightHO[NUMHID+1][NUMOUT+1],SumO[NUMPAT+1][NUMOUT+1],Output[NUMPAT+
1][NUMOUT+1];
    double DeltaO[NUMOUT+1],SumDOW[NUMHID+1],DeltaH[NUMHID+1];
    double DeltaWeightIH[NUMIN+1][NUMHID+1],DeltaWeightHO[NUMHID+1][NUMOUT+1];
    double Error,eta=0.5,alpha=0.55,smallwt=0.555;
```

```
for(j=1;j<=NumHidden;j++)
{
   for(i=0;i<=NumInput;i++)
   {
      DeltaWeightIH[i][j]=0.0;
      WeightIH[i][j]=2.0*(rando()-0.5)*smallwt;
   }
}
for(k=1;k<=NumOutput;k++)
{
    for(j=0;j<=NumHidden;j++)
   {
      DeltaWeightHO[j][k]=0.0;
      WeightHO[j][k]=2.0*(rando()-0.5)*smallwt;
   }
}
for(epoch=0;epoch<67;epoch++)
{
   for(p=1;p<=NumPattern;p++)
   {
       ranpat[p]=p;
   }
   for(p=1;p<=NumPattern;p++)
   {
      np=p+rando()*(NumPattern+1-p);
      op=ranpat[p];
      ranpat[p]=ranpat[np];
      ranpat[np]=op;
   }
   Error=0.0;
   for(np=1;np<=NumPattern;np++)
   {
      p=ranpat[np];
      for(j=1;j<=NumHidden;j++)
      {
         SumH[p][j]=WeightIH[0][j] ;
         for(i=1;i<=NumInput;i++)
         {
            SumH[p][j]+=Input[p][i]*WeightIH[i][j];
         }
         Hidden[p][j]=1.0/(1.0+exp(-SumH[p][j]));
      }
      for(k=1;k<=NumOutput;k++)
```

```
        {
            SumO[p][k]=WeightHO[0][k];
            for(j=1;j<=NumHidden;j++)
            {
                SumO[p][k] += Hidden[p][j]*WeightHO[j][k];
            }
            Output[p][k] = 1.0/(1.0 + exp(-SumO[p][k]));
            Error += (0.5*Target[p][k]-Output[p][k])*(Target[p][k]-Output[p][k]);
            DeltaO[k] = (Target[p][k]-Output[p][k])*Output[p][k]*(1.0-Output[p][k]);
        }
        for(j=1;j<=NumHidden;j++)
        {
            SumDOW[j]=0.0;
            for(k=1;k<=NumOutput;k++)
            {
                SumDOW[j]+=WeightHO[j][k]*DeltaO[k];
            }
            DeltaH[j]=SumDOW[j]*Hidden[p][j]*(1.0-Hidden[p][j]);
        }
        for(j=1;j<=NumHidden;j++)
        {
            DeltaWeightIH[0][j]=eta*DeltaH[j]+alpha*DeltaWeightIH[0][j];
            WeightIH[0][j]+=DeltaWeightIH[0][j];
            for(i=1;i<=NumInput;i++)
            {
                DeltaWeightIH[i][j]=eta*Input[p][i]*DeltaH[j]+alpha*DeltaWeightIH[i][j];
                WeightIH[i][j]+=DeltaWeightIH[i][j] ;
            }
        }
        for(k=1;k<=NumOutput;k++)
        {
            DeltaWeightHO[0][k]=eta*DeltaO[k]+alpha*DeltaWeightHO[0][k] ;
            WeightHO[0][k]+=DeltaWeightHO[0][k] ;
            for(j=1;j<=NumHidden;j++)
            {
                DeltaWeightHO[j][k]=eta*Hidden[p][j]*DeltaO[k]+alpha*DeltaWeightHO[j][k] ;
                WeightHO[j][k]+=DeltaWeightHO[j][k] ;
            }
        }
    }
    fprintf(stdout,"\nEpoch %-5d:Error = %f",epoch,Error) ;
    if( Error<0.0004 )
    {
            fprintf(stdout, "\nEpoch %-5d:Error = %f",epoch,Error);
            break;
        }
    }

double TESTINTPUT[31][NumHidden+1]=
{{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0}
,
{0,0,0,0,0,0,0,0,0,0,7,7,7,0,7,7,0,7,7,7,0,0,7,7,7,0,0,7,7,7,7,0,7,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,7,7,7,0,7,7,0,7,7,7,7,0,7,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,0,7,7,0,7,7,7,0},
{0,0,0,0,0,0,0,7,7,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,7,0,0,0,0,0,0,0},
{0,0,0,0,0,7,0,0,0,0,0,7,0,0,0,0,7,7,0,0,0,7,7,0,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,7,0,0,0,0,7,0,0,0,0,0},
{0,0,0,0,0,0,0,0,7,0,0,0,7,7,0,0,7,7,0,0,0,7,7,0,0,7,7,0,0,0,7,7,0,0,7,7,0,0,0,7,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0,0,7,0,7,7,7,0,0,7,7,0,0,7,0,0,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,7,0,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7},
{0,0,7,7,0,0,7,0,0,7,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,0,7,0,0,0,0,0,7,0,0,0,7,0,0,7,7,0,0,0,0,0,0,0},
{7,7,7,7,7,0,0,0,7,0,0,0,7,0,0,0,7,7,0,0,0,0,0,7,0,0,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,0,0,0,0,7,0,0,7,7,0,0,0,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,0,7,0,0,7,0,7,0,0,7,0,0,7,7,0,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,7,0,7,7,7,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,7,0,0,7,0,7,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,7,7,7,7.7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0},
{0,0,0,0,0,0,7,7,7,0,0,7,0,0,0,0,7,0,0,0,0,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,7,7,7,7,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,7,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,0,7,7,7,0,0,0,0,0,7,0,0,0,0,7,7,7,7,7,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,0,0,0,0,7,0,7,7,7,7,7,0,0,7,0,7,7,7,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,7,7,7,7,7,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,7,7,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,7,7,7,7,7,0,0,7,7,0,0,0,7,0,7,7,7,0,0,0,0,0,0},
{0,7,7,7,7,7,0,0,7,7,7,0,0,7,0,0,0,7,7,0,0,0,7,0,0,0,0,7,0,0,0,7,7,0,0,7,0,0,0,0,7,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,7,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,7,7,7,7,7,0,0,0,0,7,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,0,0,0,0,0,0,0},
{0,0,7,7,0,0,7,0,0,0,7,0,0,0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0,0,7,0,7,0,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0},
{0,0,0,0,0,0,7,7,0,0,7,0,0,0,0,7,0,0,0,7,7,0,0,7,0,7,7,7,0,0,0,7,7,0,0,7,0,0,7,0,7,0,0,0,7,0,7,7,7,0},
{0,0,0,0,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,7,7,7,0,7,0,0,0,7,7,0,0,0,7,7,0,0,7,0,0,7,7,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,7,0,0,7,0,0,7,0,7,0,0,7,0,0,7,7,7,7,0,0,0,7,0,7,0,7,0,0,0,7,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,7,7,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,7,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,7,0,0,7,7,0,0},
{0,0,7,7,0,0,7,0,0,7,7,0,0,0,7,7,0,0,0,7,0,7,0,7,0,0,0,7,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0,0,0,0,7,0};

    double x2[30][NumHidden];
    for(i=1;i<=30;i++)
    {
        for(j=1;j<=NumHidden;j++)
        {
            x2[i][j]=SumDOW[j];
            for(int k=1;k<=NumInput;k++)
            {
                x2[i][j]+=(TESTINTPUT[i][k]*WeightIH[k][j]);
            }
        }
    }
    double ans[30][NumOutput];
    for(p=1;p<=30;p++)
    {
        for(k=1;k<=NumOutput;k++)
        {
            ans[p][k]=0.0;
            for(j=1;j<=NumHidden;j++)
            {
                ans[p][k]+=x2[p][j] * WeightHO[j][k];
            }
        }
    }
    cout<<endl;
    int tmp[10];
    int mans[30][10];
    int m=INT_MIN,pos;
    for(i=1;i<=30;i++)
    {
        m=INT_MIN,pos=0;
        for(j=1;j<=10;j++)
        {
            if(ans[i][j]>m)
            {
                m=ans[i][j];
                pos=j;
            }
        }
        for(int g=1;g<=10;g++)
        {
            if(g==pos)
            {
                mans[i][g]=1;
            }
            else
            {
                mans[i][g]=0;
            }
        }
    }
    //Print Output Matrix
    for(p=1;p<=70;p++)
    {
        for(i=1;i<=10;i++)
        {
            cout<<Output[p][i]<<" ";
        }
        cout<<endl;
    }
    //Print Target Matrix
    for(p=1;p<=70;p++)
    {
        for(i=1;i<=10;i++)
        {
            cout<<Target[p][i]<<" ";
        }
        cout<<endl;
    }
    //Making confusion matrix
    int res[10][10];
    int c=0;
    for(p=1;p<=30;p++)
    {
        for(j=1;j<=10;j++)
        {
            c+=mans[p][j];
        }
        cout<<c<<endl;
    }
    return 0;
}
/******************************************************************************/
```

```
//Confusion Matrix
```

```
{{2,0,0,0,0,0,0,1,0,0},
{0,2,0,0,1,0,0,0,0,0},
{0,0,2,0,0,0,0,0,1,0},
{0,0,0,1,1,0,0,0,1,0},
{0,1,0,0,1,0,0,0,1,0},
{0,0,0,1,0,1,0,0,1,0},
{0,0,0,0,1,0,2,0,0,0},
{0,0,0,0,1,0,0,2,0,0},
{0,0,0,1,0,0,1,0,1,0},
{0,0,0,0,1,0,0,0,0,2}};
```