Assignment#7 Veera Venakata Sathya Bhargav Nunna

a), b) & c)

Self-Organising Feature Maps (SOM)

Self-Organising Feature Maps (SOM) is extension of the competitive learning in several ways Competitive neurons are arranged in some geometry – typically a 2dimensional grid or a line During learning not only the weights of the winner are updated but also the weights of its neighbors. A neighboring function defines the size and shape of the neighborhood around the winner. As the learning progresses, both the size of the neighborhood and the learning

SOM - Algorithm

Initialization: Selection of the number of weight vectors; at least 1 competitive neuron corresponds to one cluster and Initialize weights to small random values

Distance calculation: Distance between the input and each output neuron j is calculated as $d_j=\sum(x_i(t)-w_{ij}(t))^2$, Select the winner neuron, the output node with minimum distance

Weights updating for winner neuron and its neighborhood neurons : $W(n+1) = W_j(n) + \eta\, h_{j,i(x)}(n)(x-W_j(n))$ (for the winner neuron the h function is 1)

Stopping criteria: Maximum number of epochs is reached or weight vectors stop to move

SOM was implemented and may experiments were carried out with changing the value of n.

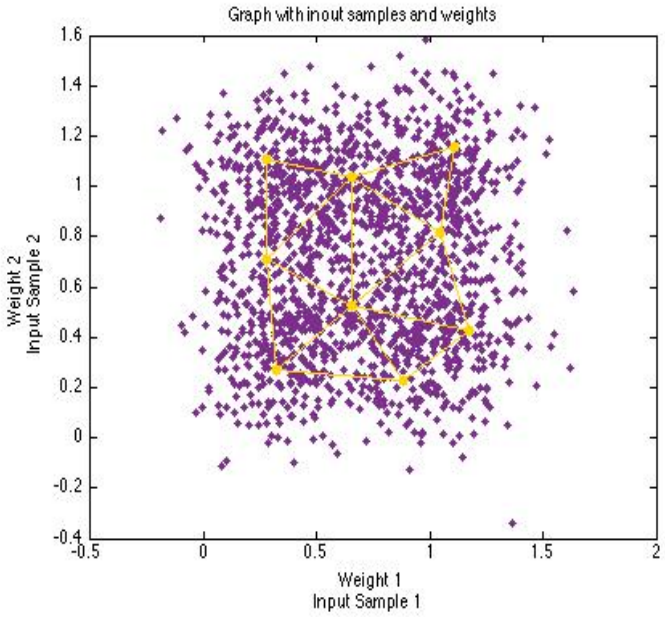
The sigma and the learning rate values decay as the number of the iteration increases.

$Sigma_n=sigma*exp(-n/t1)$

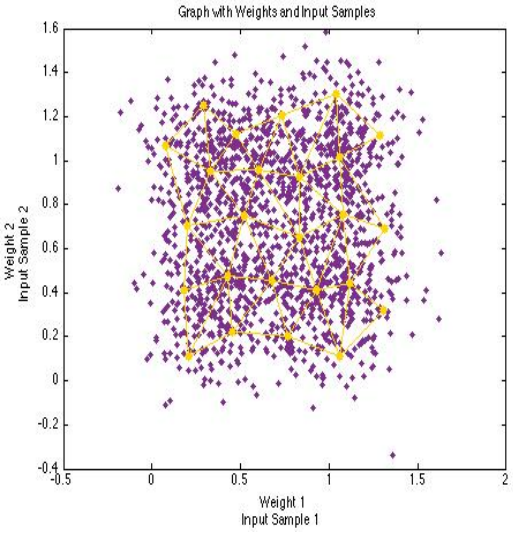
$\eta_n = \eta *exp(-n/t2)$

The following are the plot for various n x n grid neurons of SOM:

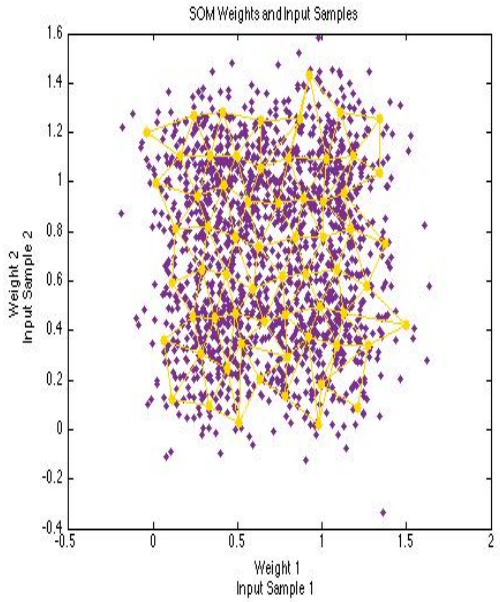
Plot for n=3:



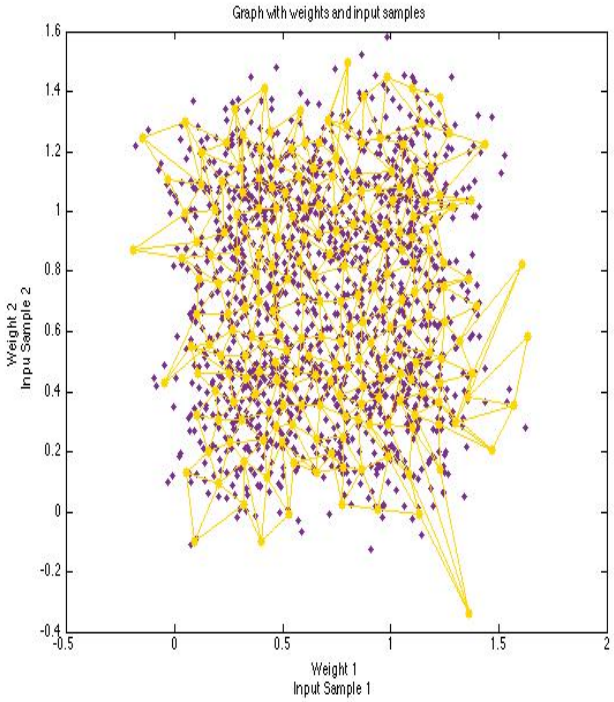
Plot for n=5:



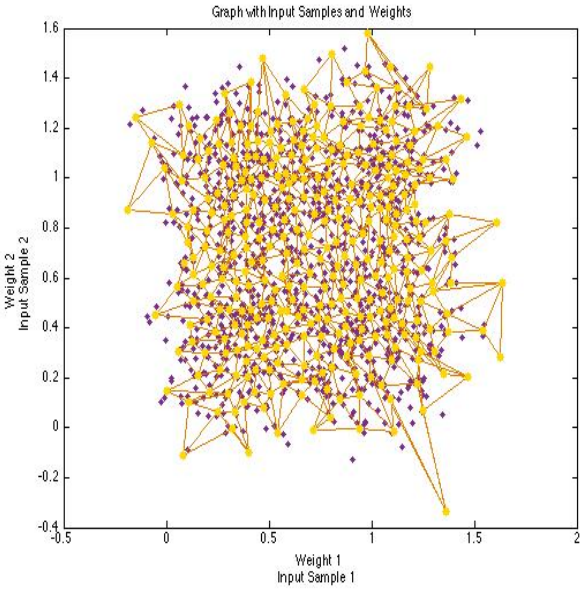
Plot for n=9:



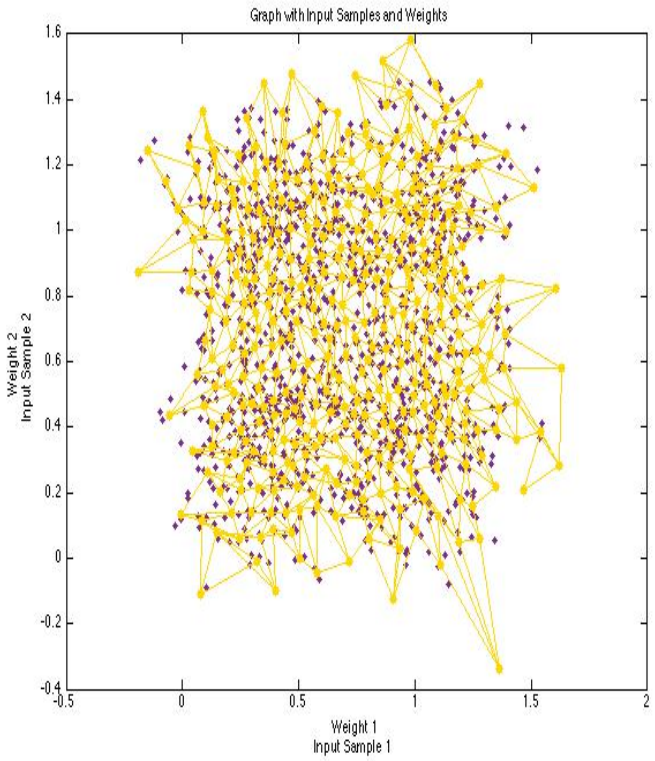
Plot for n=15



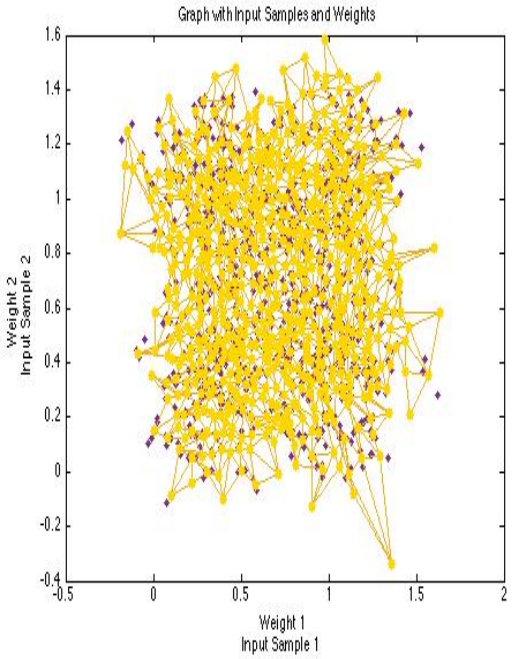
Plot for n=19:



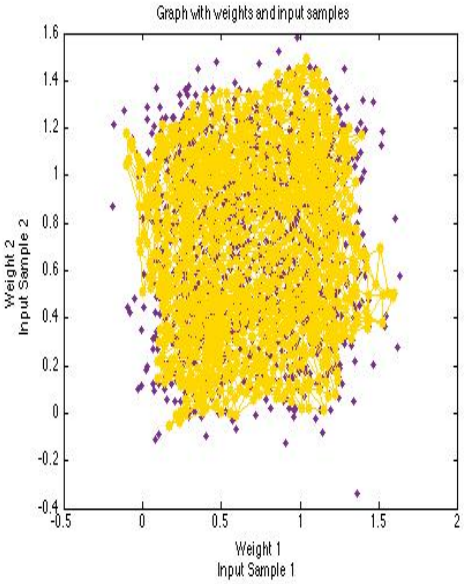
Plot for n=21;



Plot for n=27

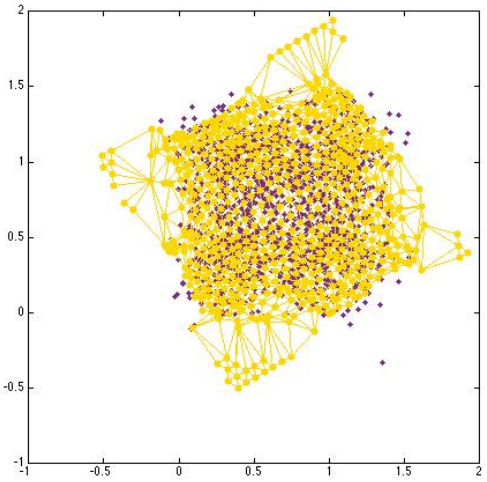


Plot for n=33

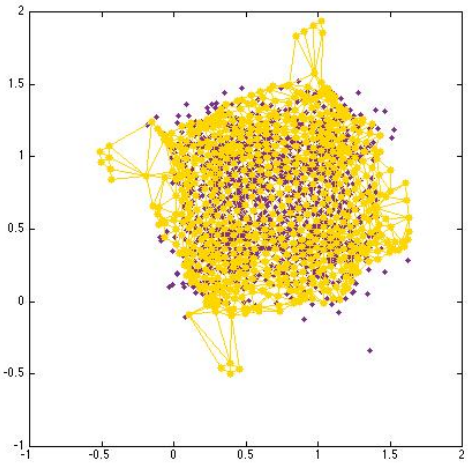


The following are the plots for weigth tansistion in n^{th} iteration:

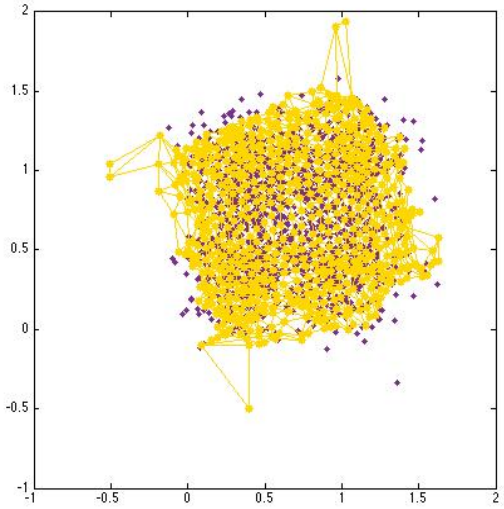
Grid with 27 Neurons and 1^{st} iteration



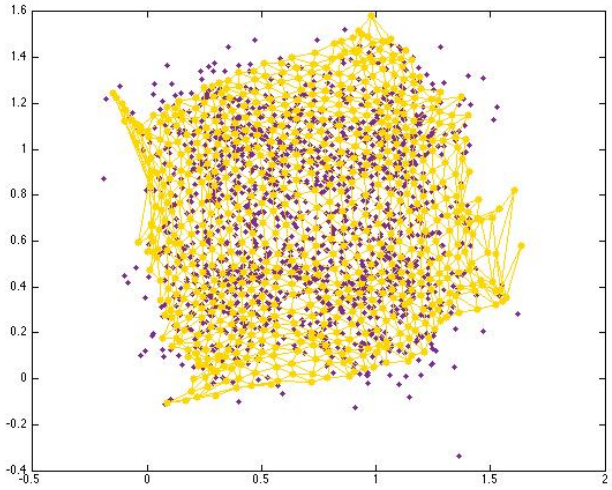
Grid with 27 Neurons and 2nd iteration



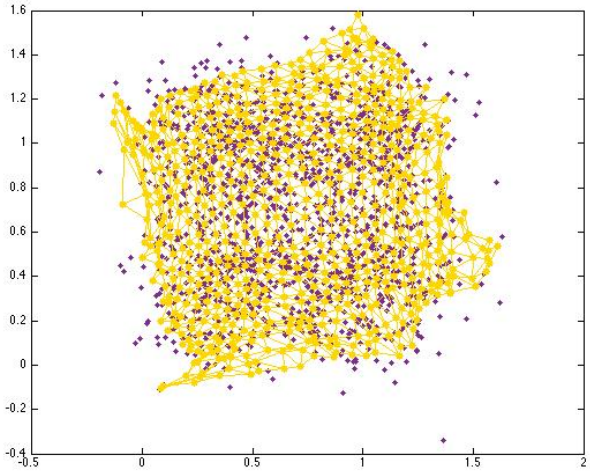
Grid with 27 Neurons and 3rd iteration



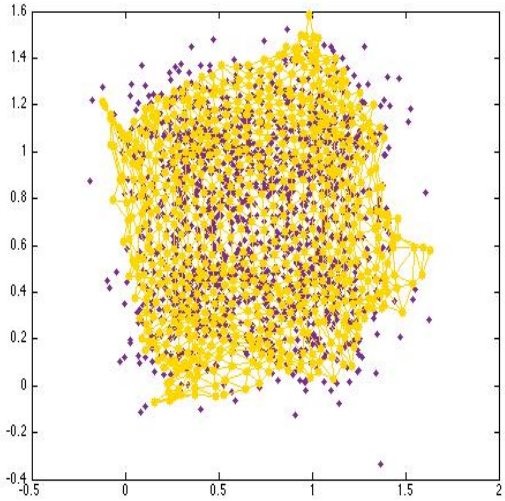
Grid with 27 Neurons and 4th iteration



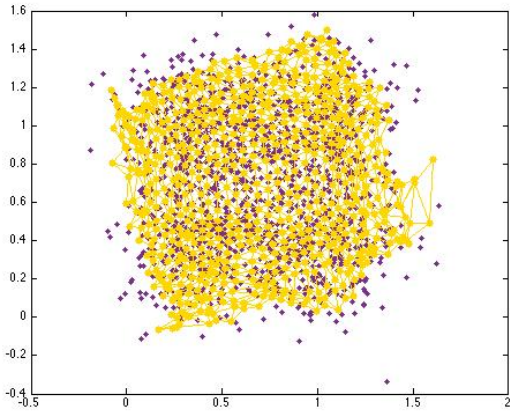
Grid with 27 Neurons and 5th iteration



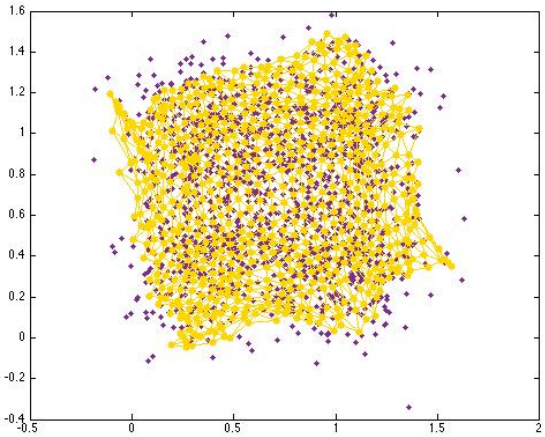
Grid with 27 Neurons and 6th iteration



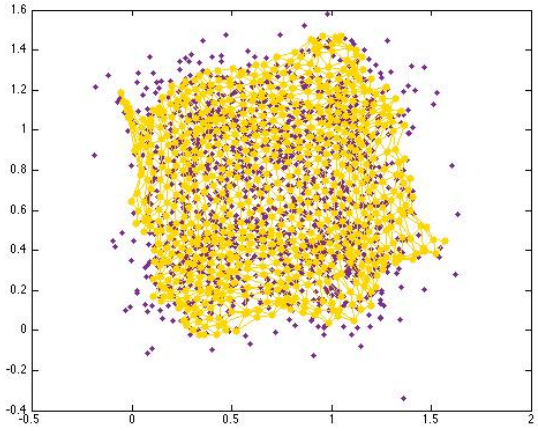
Grid with 27 Neurons and 7th iteration



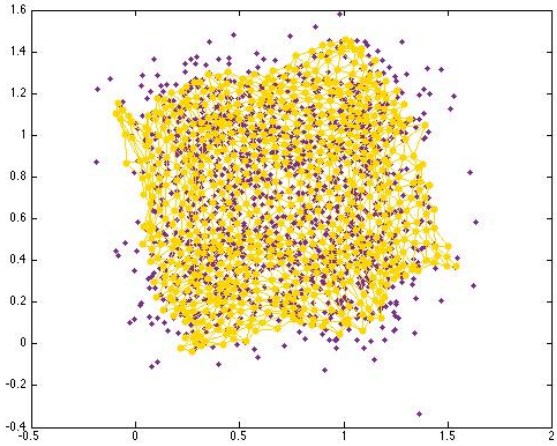
Grid with 27 Neurons and 8th iteration



Grid with 27 Neurons and 9th iteration



Grid with 27 Neurons and 10th iteration



Appendix:

```
#include<iostream.h>
#include<math.h>
#include<fstream.h>
#include<string.h>

using namespace std;

double cal_dist(double x1,double y1,double x2,double y2)
{
        double val=(x2-x1)*(x2-x1);
        val=val+(y2-y1)*(y2-y1);
        val=sqrt(val);


        return val;
}

double activation(double d,double sig)
{
        double val=exp(-((d*d)/2*(sig*sig)));

        return val;

}
double fRand(double fMin, double fMax)
{
    double f = (double)rand() / RAND_MAX; return fMin + f * (fMax - fMin);
}

int main()
{
        int k=0;
        cout<<"Enter the grid value..\n";
        cin>>k;


        double wx[k][k], wy[k][k],deltawx[k][k],deltawy[k][k];
  double d[k][k], act[k][k];
```

```
        double smalld,n=0.1,x,y,no;
   int a=0;
        int smalli=0,smallj=0,run=0;
        double sig,sigo;
        double inputx1[1600]={//Inpute of x1};
;double inputx2[1600]={//Inpute of x2};
        double meanx=0,meany=0;
        double diffx,diffy,sqdiffx,sqdiffy,sum;

        for(int i=0;i<k;i++)
        {
                for(int j=0;j<k;j++)
                {
                //      cout<<"Enter weight coordinate for "<<i<<" "<<j<<"\n";
                        //cin>>wx[i][j];
                        //cin>>wy[i][j];
                         wx[i][j] = fRand(0,0.2);
         wy[i][j] = fRand(0,0.6);
        // cout<<"Enter weight coordinate for "<<i<<" "<<j<<" "<<wx[i][j]<<" "<<wy[i][j]<<"\n";
       meanx=  wx[i][j]+meanx;
       meany=wy[i][j]+meany;
                }
        }
        meanx=meanx/k*k;
        meany=meany/k*k;

        for(int i=0;i<k;i++)
        {
        for(int j=0;j<k;j++)
        {
          diffx=wx[i][j]-meanx;
          sqdiffx=diffx*diffx;
          diffy=wy[i][j]-meany;
          sqdiffy=diffy*diffy;
          sum=sum+sqdiffx+sqdiffy;
        }
   }
   cout<<"sum: "<<sum;
   sum=sum/k*k;
   sum=sqrt(sum);
```

```
   cout<<"Sq sum: "<<sum;
   sig=sum;


int c=0;
//cin>>c;
while(run<=1)
{
            a=0;

          while(a<=1600)
          {
              //cout<<a;
          x=inputx1[a];
          y=inputx2[a];

          for(int i=0;i<k;i++)
          {
                    for(int j=0;j<k;j++)
                    {
                              d[i][j]=cal_dist(wx[i][j],wy[i][j],x,y);
                    }
          }

          smalld=d[0][0];

          for(int i=0;i<k;i++)
          {
                    for(int j=0;j<k;j++)
                    {
                              if(smalld > d[i][j])
                              {
                                        smalld=d[i][j];
                                        smallj=j;
                                        smalli=i;
                                        //cout<<"Smalld: "<<smalld<<"\n";
                              }

                    }
          }
```

```
        for(int i=0;i<k;i++)
        {
        for(int j=0;j<k;j++)
        {

                d[i][j]=cal_dist(wx[smalli][smallj],wy[smalli][smallj],wx[i][j],wy[i][j]);
        }
}

        for(int i=0;i<k;i++)
        {
                for(int j=0;j<k;j++)
                {

       sigo=sig*exp(-((double)a/1000));   //tau 1
       sigo=sig;
                        act[i][j]=activation(d[i][j],sigo);


                }
        }

        act[smalli][smallj]=1;
        for(int i=0;i<k;i++)
        {
                for(int j=0;j<k;j++)
                {




                }
}


        for(int i=0;i<k;i++)
        {
                for(int j=0;j<k;j++)
                {
         no=n*exp(-((double)a/1000));  //tau 2
```

```
            deltawx[i][j]=(no*act[i][j]*(x-wx[i][j]));
            deltawy[i][j]=(no*act[i][j]*(y-wy[i][j]));
                                wx[i][j]=wx[i][j]+(no*act[i][j]*(x-wx[i][j]));
                                wy[i][j]=wy[i][j]+(no*act[i][j]*(y-wy[i][j]));
                        }
                }
//Write the new weights to an output file.
            cout<<"\n";
for(int i=0;i<k;i++)
{
        for(int j=0;j<k;j++)
        {
         cout<<wx[i][j]<<"\t"<<wy[i][j]<<"\n";
        }
}
 a++;
   }


            run++;
}

            return 0;
}
```