Self-organizing Map – simple example

This is a partial listing of the Java code for the Self-organizing map program working on a simple set of data that contains examples of ten different types of data.

It includes all the code from the **Initialize** and **Evolution** sections, but does not include the variables' descriptions or the display section. The specification of the variables, for example the type of each variable and reasonable dimensions for the arrays, should be clear from the context.

Initialization section

```
// Set up initial parameters

mapsize=70;
cycle=0;
adjustment=1.0;
neighbourhood=15;
autoscaleplot=true;
showlabel=true;
labelcounter=0;

/* The parameter ledge determines whether the "normal" SOM will be displayed, or
whether, instead, the display will be coloured in proportion to the difference between
nearby nodes. */

ledge=false;

/* Create initial weights and data points. */

for (i=0; i<mapsize; i++)
{
   for (j=0; j<mapsize; j++)
   {
      for (k=0; k<2; k++)
      {
         weights[i][j][k]=Math.random();
      }
   }
}

for (i=0; i<10; i++)
{
   xpoint[i]=i*i;
   ypoint[i]=5.0+i;
```

```
}

// oldbesti and oldbestj store the positions for drawing the labels

for (i=0; i<10; i++)
{
   oldbesti[i]=5;
   oldbestj[i]=5;
}
```

Evolution section

```
cycle++;

/* Select a random point and add some noise. */

pointchoice=(int)(10.0*Math.random());
xp=xpoint[pointchoice]+Math.random();
yp=ypoint[pointchoice]+Math.random();

// besti and bestj are the coordinates of the BMU

besti=0;
bestj=0;
double diff,bestdiff;
bestdiff=9999.9;

/* Find the BMU */

for (i=0; i<mapsize; i++)
{
   for (j=0; j<mapsize; j++)
   {
      diff=Math.abs(xp-weights[i][j][0])+Math.abs(yp-weights[i][j][1]);
      if (diff<bestdiff)
      {
         bestdiff=diff;
         besti=i;
         bestj=j;
      }
   }
}

// Update labelling

labelcounter++;
if (labelcounter>9) labelcounter=0;
oldbesti[labelcounter]=besti;
```

```
oldbestj[labelcounter]=bestj;

if (showlabel)
{
   switch(labelcounter)
   {
      case 0: label0=" " + pointchoice; break;
      case 1: label1=" " + pointchoice; break;
      case 2: label2=" " + pointchoice; break;
      case 3: label3=" " + pointchoice; break;
      case 4: label4=" " + pointchoice; break;
      case 5: label5=" " + pointchoice; break;
      case 6: label6=" " + pointchoice; break;
      case 7: label7=" " + pointchoice; break;
      case 8: label8=" " + pointchoice; break;
      case 9: label9=" " + pointchoice;
   }
}
else
{
   label0=label1=label2=label3=label4=label5=label6=label7=label8=label9="";
}

// update weights at the winning node and its neighbours

int istart,iend,jstart,jend;  // Neighborhood runs from istart to iend
istart=besti-neighbourhood;
if (istart<0) istart=0;
iend=besti+neighbourhood;
if (iend>mapsize) iend=mapsize;
jstart=bestj-neighbourhood;
if (jstart<0) jstart=0;
jend=bestj+neighbourhood;
if (jend>mapsize) jend=mapsize;
double dist,distance;

for (i=istart; i<iend; i++)
{
   for (j=jstart; j<jend; j++)
   {
      distance=Math.sqrt((double)((i-besti)*(i-besti)+(j-bestj)*(j-bestj)));
      if(distance<=neighbourhood)  // node is within neighborhood
      {
         dist=0.1*(neighbourhood-distance);
         weights[i][j][0]=(weights[i][j][0]*(1.0-dist)+dist*xp);
         weights[i][j][1]=(weights[i][j][1]*(1.0-dist)+dist*yp);
      }
   }
}
```

```
// Use the weights to generate a z value for the display

for (i=0; i<mapsize; i++)
{
   for (j=0; j<mapsize; j++)
   {
      z[i][j]=0.0;
      if (!ledge) // Display regular SOM
      {
         if (multiplyweights)
         {
            z[i][j]=Math.pow(weights[i][j][0],0.5);
         }
         else
         {
            z[i][j]=weights[i][j][1]+weights[i][j][0];
         }
      }
      else  // Display boundaries map
      {
         istart=i-nhood;  // Boundary extends nhood units in either direction
         iend=i+nhood;
         jstart=j-nhood;
         jend=j+nhood;
         if (istart<0) istart=0;
         if(iend>mapsize) iend=mapsize;
         if(jstart<0) jstart=0;
         if(jend>mapsize)jend=mapsize;
         for (int is=istart; is<iend; is++)
         {
            for (int js=jstart; js<jend; js++)
            {
               z[i][j]=z[i][j]+Math.abs(weights[i][j][0]-
weights[is][js][0])+Math.abs(weights[i][j][1]-weights[is][js][1]);
            }
         }
      }
   }
}

if (cycle%250 == 0)  // Every 250 cycles, squeeze the neighborhood down…
{
   neighbourhood--;
   if (neighbourhood<3) neighbourhood=3;  // …but don't make it too small
}
```