Implement SOM using Processing

Zhenghao Chen

June 2014

Abstract

Self-Organizing Map (SOM) is known as a famous approach to implement the data visualisation without supervisor while processing is a software that is developed to sketch visual arts like animation. In this paper, I will introduce how to implement Self-organizing map using processing. There are 4 parts in my paper. Firstly, I will introduce the basic ideas of self-organizing and processing. Secondly, I will explain the general approach(INTR) to implement SOM using processing. That includes how to introduce input vector, build a SOM object and Nodes object. Thirdly, 2 specific cases colour map and military power of 12 countries will be given to prove my idea. Finally, I will summary the advantages that using processing to implement SOM.

1Introduction

1.1 SOM

Self-organizing map (SOM) was invented by Dr TeuvoKohonen, also called Kohonen Network. Dr TeuvoKohonen defines it as an approach to map the input data with high dimension into a 2dimension array for data visualisation without supervisor. By using SOM we can visualize and analyse data like the similarity of some vector on the 2D map.

With a 2D map consist with nodes (neurons), and each node has its own references vector where mi=[ni1,ni2,ni3,nin] € Rn associated initially and randomly. Then we find the BMU(best matching unit) and training its neighbourhoodby the learning-rate factor. After training (can be more than thousand times) we can visualize the data and analysis it.

The main algorithm includes 4 steps are initialization, competition, cooperation, adaption that I conclude them as ICCA

i. I-initialization: construct the 2D arrays and assign the arbitrary weights for each nodes
ii. C-Competition: using ***Euclidean distance function***c=argmin{||x-mi||}to find the BMU

T

iii. C-Cooperation: using ***Decay Function*** [1]to locate the neighbourhoods of BMU
iv. A-Adaptation: training the neighbourhood of the BMU using ***neighbourhood function***[2]

1.2 Processing

Processing is a writing software developed by Dr CassyRease and Dr Benjaming Fry in MIT 2001. Currently, there are 2 version processing1 and processing2. Its motivation is to sketch the shape like circle and quadrangle by writing single line of code. The language it can support are all object-oriented language including java and C++. By using that, anything in visual- an interface, an animation can be sketched.

There are 3 main sections for the processing

i. Setup: be used to prepare everything and initially assign each value
ii. Draw: the main section of processing, used to execute the whole processing file like the main class of Java.
iii. Class: create and define the object which used to the main class.

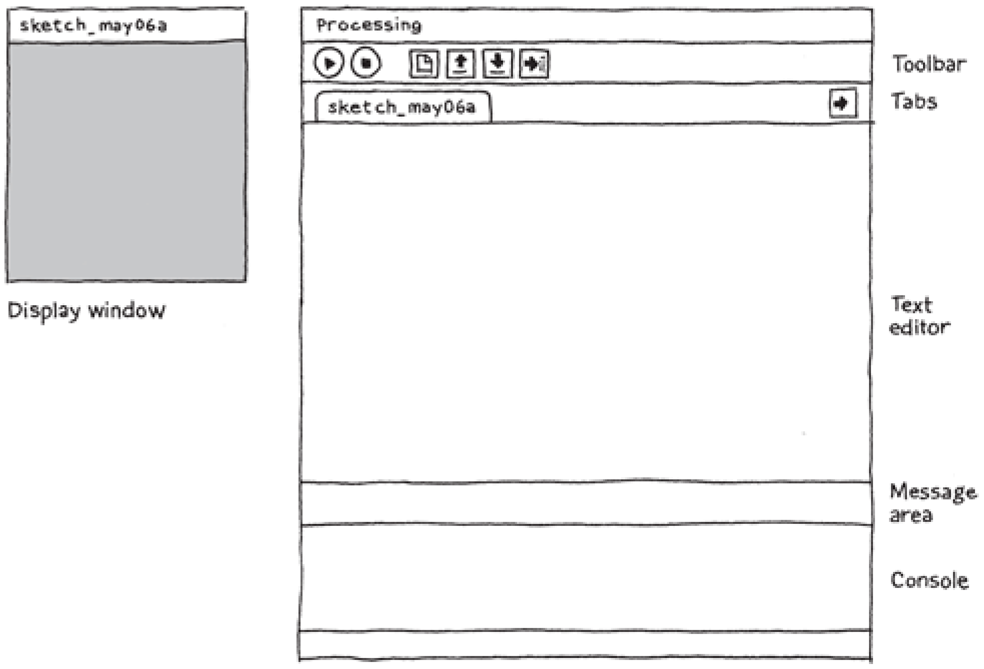


Figure 1 the basic architecture of processing

[1] Decay function:

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1,2,3\ldots$$

[2]Neighbourhood function

$$W(t+1) = W(t) + \Theta(t)L(t)(V(t) - W(t))$$

T

2Definition

2.1 Input data

Since SOM is a way to visualise the data, we need to introduce the data to the SOM, the mathematical idea is to use vector with weight in different dimensions to represent the data. In processing I use a 2D array to store the input vector, where rows is used to present the number of items and columns is to show the number of attributes

2.2Objects

Processing support Object-Oriented language, we will build class SOM and Nodes to implement Self-Organizing Map. The class SOM will allocate a 2D array as map and train the nodes using Self-Organizing Map Algorithm (ICCA). The class node is used to create the node objects with the weight in different dimensions.

2.3Neurons

In Self-Organizing Map, the neurons can be defined as any regular and irregular shape for visual display, hence in processing we need to display the neuron and visualise it.

There are several way to make basic shape which are rectangle, ellipse, triangle and quad ,we can even use text to present nodes

Name	Function
Triangle	Triangle(x1,y1,x2,y2,x3,y3)
Rectangle	Rect(x,y,width,heigh)
Ellipse	Ellipse(x,y,width,height)
Quadrangle	Quad(x1,y1,x2,y2,x3,y3,x4,y,4)
text	Text(string,x,y)

Table 1 function in processing to present the shape

By using these 4 shapes we can implement the nodes on the 2D array.

3General Approach

3.1 Introduce input

There are 3 steps to define

T

Step1 Define the items and attributes from abstract data.

The self-organizing map is used to visualise the data with non-liner weight. So we need to have input data that can be anything small data or big data like human, country, animal,colour. And each data with its own attributes. For example, human have different colour of hair, different height and weight. Hence, we need to define the data

Step2 convert the data to vector and represent it in 2D array.

We can easily convert the data to the vector under the rules:

i. item = vector (number of item= number of vector)
ii. attributes = weight (number of attributes = number of dimensions)

For instance, consider there are 4 items with 5 attributes,

	a	b	c	d
Attribute1				
Attribute2				
Attribute3				
Attribute4				
Attribute5				

Table 2 example input data

We can convert these data to 4 vectors with 5 dimensions of weight.

Step3 Use a 2D array to store vectors

I use the 2D array to store these data, where column is used to store the number of weight and rows is to store the number of vector.

Using previews example table 2, I will construct a 2D array that

```
float [][] example = new float[4][5]; //4 items and 5 attributes
```

If the value of item B and its attribute 3 is 0.24, then we create the element

```
example[1][2]=0.24  // both column and rows start from 0
```

In this part there are 2 way to represent the values:

i. Binary data: this way is to represent the qualitative property like ‘good’ or ‘bad’, ’yes’ or ‘no’.

Where 1=presence

T

0= absence

ii. Real number: this way is to show the real continuous data like height of human, population of area.

3.2 Class Node

To create nodes we define a class call node class, this object has 3 components that location, dimension and weight.

1. Its location thaton x-axis, y-axis(it is on a 2D array)

i.e. The position x, y

2. The number of dimensions of weight, we will use an array to store the value of weight. And define the length of array that should equal to the number of weight. For instance, if the number dimensions is 3, we will construct an array whose length is 3. In other words we will use 3-element array to store weight in 3 dimensions

```
intweightcount =n;   //the number of weight and dimensin

Float[] weight = new float[weightcount]  //this is the weight
```

3. Random weight in each dimension

```
    for(int I=0; i<weight_count;i++){

  weight[i]= random(a, b);// assign the values from a to b

}
```

3.3 Class SOM

This class will contain the algorithm methods that defining time constant, calculating the BMU, finding the neighbourhood and training each node. Also it will have 1 constructor allocating 2D array and initializing values to neurons and render methods(in some cases can be more than 1) to show the nodes in the processing.A very important idea of this class is that we use array to store weight and train the weight by using SOM algorithm. So most parameter of methods will be array. I will introduce the each method.

1. Calculate the weight distance

This method is used to compare weight of 2 vectors and find the weight distance of them using the***Euclidean distance function***.

We need the weight of 2 vectors. So the parameters will be 2 arrays as we use array to represent the weight.

```
float x[] ,floaty[]
```

T

Then, check if dimensions of x and y is equal. If they are not equal, we cannot compare them.

i.e`.(if (x.length != y.length)){`

```
exit();

        }
```

In the final step,Apply***Euclidean distance function***.to calculate the difference

$$Dist = \sqrt{\sum_{i=0}^{i=n}(V_i - W_i)^2}$$

```
dis += sq( (x[i] - y[i]));
dis = sqrt(dis);
```

2. Find the BMU

We have to choose a random input vectors and compare it to all the nodes on the neuron. After that, we will find a node that has the smallest weight difference with the chosen vector. This node will be called ***location of the best match*** or best matching node (defined by TeuvoKohonen) that means the location "response". We call it "***Best Matching Unit***" (BMU).

Hence, this method is to look through the whole map and calculate the weight difference of all the nodes and a node with the smallest weight distance of input vector, then define it as BMU and return the position of that.

As we aim to find a smallest weight distance of input vector, the first step is to compare weight distance of input vector and the nodes on the whole map. Hence, the parameter should be the weight of input vector i.e. the array `w[]`

Then we will calculate the weight distance of input vector and all nodes on the map applying the method weight_distance.

```
    for (inti = 0; i<mapHeight; i++) {

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

floatdis = weight_distance(nodes[i][j].w, w);// apply weight-distance

                                                    //method

    }

    }
```

Finally, asidea for the code is to find the BMU, so we need to return the position of BMU in x-axis and y-axis, I use an index number to represent the BMU where first 16 bits is the x-axis ,last 16bits is y-axis

```
if (dis<minDist) {
minDist = dis;
minIndex = (i<< 16) + j; // use an index number to store the
                                    // position of BMU
       }
```

T

3. Define the time constant
   We know the time constant λ is quite important number for the learning rate function, radius decay function. So we need to define it before train the node.

   (1). Apply the functionλ = (numIterations/mapRadius)

```
timeConstant = iterations/log(radius);
```

4. Training

   Training stage is the most important stage of Self-Organizing map that is also defined as learning step by TeuvoKohonen, he said "those nodes that are topographically close in the array up to a certain geometric distance will activate each other to learn something from the same input x". Hence, to achieve this goal, there are 2 steps which are locating the neighbourhoods of BMU and training all the nodes that are defined as BMU.

   i. Locate the neighbourhood

   we will use a circle to locate the neighbourhood whose central is BMU and the initial radius isthe width of the map.

```
(som.mapWidth + som.mapHeight) / 2;
```

   Then, radius of circle will decay by each iteration(i.e. shrinks by each time), hence, we use the decay function to redraw the circle each time Based on this function, we find that the radius will be approaching to 0 when t increase.

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1, 2, 3...$$

```
radiusDecay = radius*exp(-1*i/timeConstant);
```

   In order to know if the nodes are inside the circle, we need to use function

```
float d = dist(nodes[x][y].x, nodes[x][y].y, nodes[a][b].x,
nodes[a][b].y);
```

By this function we can calculate the geometrical distance of nodes and BMU.

If the distance is less than radius (inside the circle) then trained it.

```
(d <radiusDecay)
```

   All the nodes inside the circle will be trained.

   ii. Training all the nodes

T

Since all the neighbourhoods are located now we start training them by"neighbourhoodfunction" Hic(t)=h(||rc-ri||,t)

The only parameter we need for this function is t, the number of iteration. But,beforehand ,we need to defined 2 sub-function ,hence ,there are 3 steps to achieve this function.

Step1define influence function

$$\Theta(t) = \exp\left(-\frac{dist^2}{2\sigma^2(t)}\right) \qquad t = 1,2,3\ldots$$

To get this function, obviously, we need the geometrical distance of BMU and nodes and radius of neighbourhood circle which we have got in the first stage.

Then we applied it in processing,

```
float influence = exp((-1*sq(d)) / (2*radiusDecay*i));
```

Step 2 define learning rate function

$$L(t) = L_0 \exp\left(-\frac{t}{\lambda}\right) \qquad t = 1,2,3\ldots$$

This function is to calculate a very important scalar value called "learning rate factor"(defined by TeuvoKohonen). It has the same characteristic as radius function that they both decrease by the time. Also, the learning rate factor is always between 0 and 1.

To apply this function in processing

```
learnDecay = learnRate*exp(-1*i/timeConstant);
```

Step 3define the neighbourhood function

Once we get these two sub function we can now apply the main function which is

$$W(t+1) = W(t) + \Theta(t)L(t)(V(t) - W(t))$$

```
float influence = exp((-1*sq(d)) / (2*radiusDecay*i));
```

Then we train the weight of all the nodes within circle by neighbourhood function

```
nodes[a][b].w[k] += influence*learnDecay*(w[k] - nodes[a][b].w[k]);
```

5. Construction

T

We need to assign and define initial values before we start to call SOM class, this constructor will allocate a 2D array serves as our map and define lattice of 2D array as nodes and assign the dimensions and weight to each nodes.

We need to do 2 things:

i. Assign the value of the map: the width,the height and the input dimension and initial radius of neighbourhood circle.

```
     mapWidth = w;
     mapHeight = h;
inputDimension = n;
radius = (h + w) / 2;
```

ii. Assign the arbitrary values to the different weight of different dimensions of each node, and allocate them on the map.

```
nodes = new Node[h][w];
for(inti = 0; i< h; i++){
for(int j = 0; j < w; j++) {
nodes[i][j] = new Node(n, h, w); // assign the nodes to lattice
nodes[i][j].x = i;
nodes[i][j].y = j;
      }
   }
```

6. Render

In order to implement the SOM by processing, we need methods called render. As mentioned in introduction, Self-Organizing Map needs some shape to display the nodes to show the process of result of training while in processing we can use several objects to represent nodeslike ellipse and rectangle or even texts. In some cases, we even need more than 1 renders to show different things. Since we can have different way to render our map depend on different cases, there is no fixed codes for renders. However, there are 2 general steps.

i. calculate the number of the nodes on the row or column

```
int X = screenW / mapWidth;
int Y = screenH / mapHeight;
```

ii. display each node to like rectangles or ellipses in processing that users can visualise them on the screen. And then transfer the weight to the visualised value. For instance if what you input is colour ,you need to transfer the weight of your vector to real RGB values and use rectangle to represent the node

T

3.3 Summary

The general idea of implementing SOM using SOM, includes 4 main steps that are introduce input, use SOM class to allocate 2D map consists each node with arbitrary weight, train the weight of nodes by completing SOM algorithm and render nodes on the map. These 4 step I conclude as input, map, train and render (INTR), we need to use very import class that class SOM and class Node to implement these 4 steps and complete SOM.

By using the Processing, we can store weight of node in a 1D array where the length of array equals to the number of dimensions. And the weight is the thing we actually need to train. As Processing support object-oriented language, we can create the object SOM and nodes. In addition, we can easily render the map by using different visual arts in Processing since it is a sketching –program. The values we need are

i. dimensions and weight of input
ii. size of map, height and width
iii. iteration time t

The structure of idea is

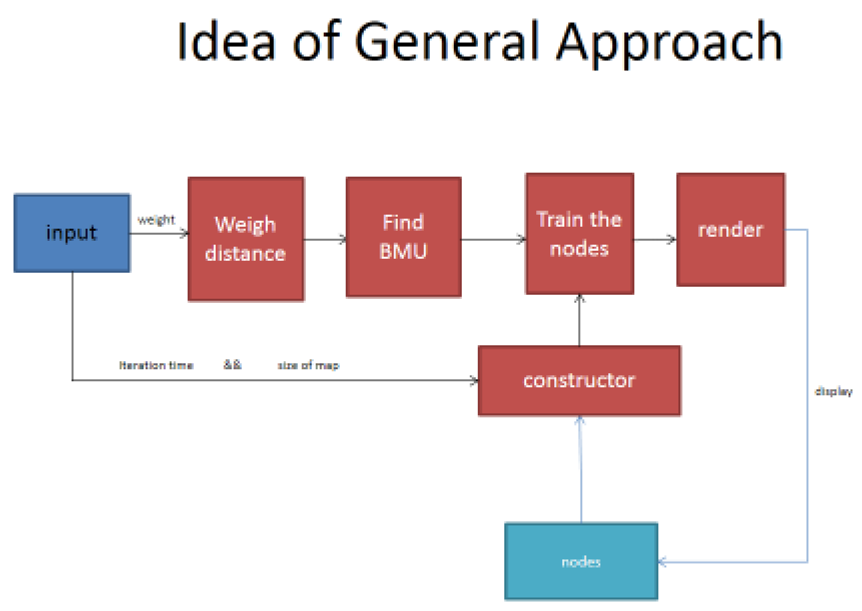


Figure 2 structure of general approach

4Specific case study

To show how I implement my general idea input ,map ,train and render (INTR) to real case that how to use Processing to show Self-Organizing Map, I prove 2 specific case which are colour map and military power of 12 countries. For both cases, I used exactly the same class SOM and class Node,

T

and implement general idea that introducing input, allocating map, training and rendering. I mentioned before. The only different things are the input and renders. For colour map, the map will show the similarity of 10 colours that I input. For military power, the map will show the difference and similarity of military power of 12 countries.

4.1 Colour Map

As we know that colour is present by its 3 basic elemental colour which are red, green and blue (RGB). So I will use 3 dimensions input which consists RGB. The 10 different colours will have different value in their RGB. And to render each nodes I use rectangle (rect(x, y ,l,w)) and colour(fill(r,g,b)) in Processing. The map will show the similarity of 10 different colours.

1. Input

I introduce 10 different colours and then implement them on a 2 dimension map, they are all 3 dimensions. In Processing 2, the range of value of single colour is form 0 to 255

	Red	Green	Blue
Colour 1	255	255	255
Colour 2	0	0	0
Colour 3	255	0	255
Colour 4	255	0	0
Colour 5	0	255	0
Colour 6	0	0	255
Colour 7	255	255	0
Colour 8	0	255	255
Colour 9	255	102	102
Colour 10	63.75	63.75	63.75

Table 3 input of 10 different colours

Based on table 3, I store input to 2 dimensions array for store the 10 input colours. Where the row is the number of input colour and the column is the weight of input colour in the 3 dimensions. I use the range for 0 to 1. (i.e. RGB/255)

```
float[][] rgb= new float[10][3] ;
```

2. Map

I will train them for 5000 times in 600*600 screen

```
intmaxIters = 5000;
intscreenW = 600;
intscreenH = 600;
```

And I allocate a 40*40 map and define the number of dimensions is 3 as there are 3 attributes RGB.

T

```
som = new SOM(40, 40, 3);
som.initTraining(maxIters);
iter = 1;
```

3. Training

What we need to train is the weight that is the colour in this case, so we pick up a random input colour and use its weight to find BMU and locate the neighbourhood and training them each time, that is in different train time, SOM will randomly pick up a colour and train its weight until we manually stop it or it complete 5000 times for training.

```
if (iter<maxIters&&bGo){
som.train(iter, rgb[t]);
iter++;
  }
```

4. Render

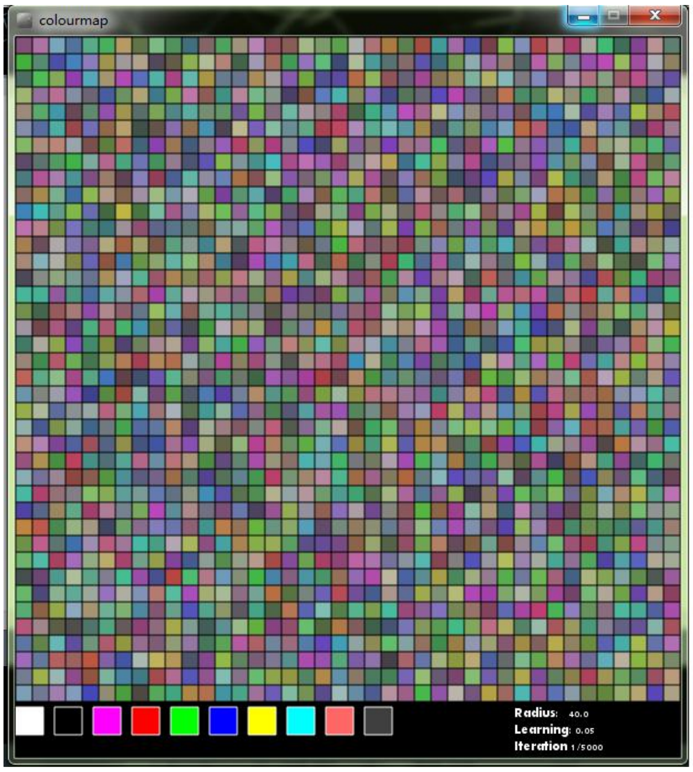
In this case, I only use 1 render to display the node as rectangle filling corresponding colour after training. As I use the range from 0 to 1 for input 2D array and real range of colour in Processing is from 0 to 255. So I multiply 255 for the results.

```
int r = int(nodes[i][j].w[0]*255);
int g = int(nodes[i][j].w[1]*255);
int b = int(nodes[i][j].w[2]*255);
fill(r, g, b);
stroke(0);
rectMode(CORNER);
rect(i*pixPerNodeW, j*pixPerNodeH, pixPerNodeW, pixPerNodeH);
```

5. Results

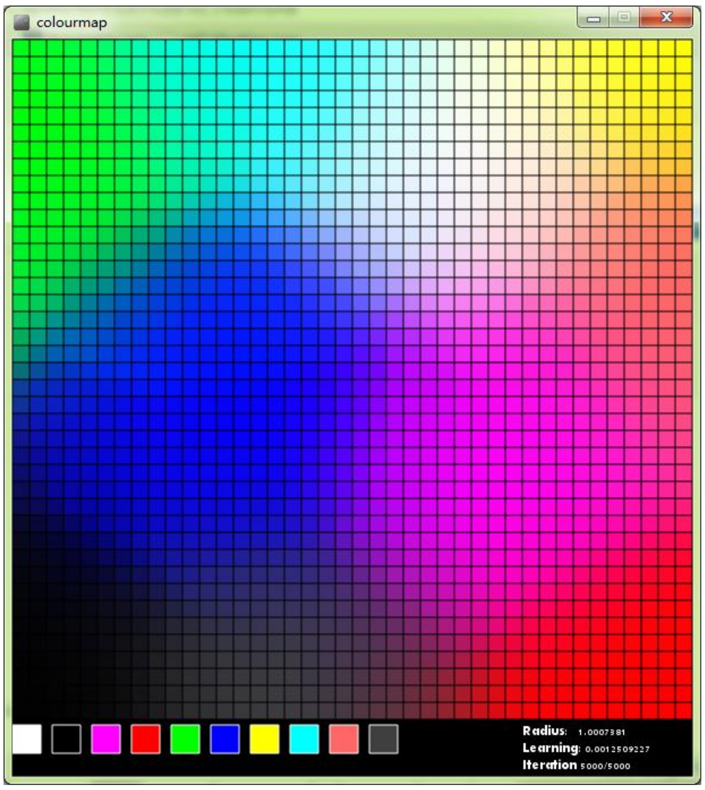
Initially, random colours in each rectangle will show on the screen and after 500 times training, the map will show the similarity of 10 different colours

T



```
Radius:   40.0
Learning: 0.05
Iteration 1/5000
```

Figure 3 The initial random colour



```
Radius:   1.0007381
Learning: 0.0012509227
Iteration 5000/5000
```

Figure 4 similarity of 10 colours after 5000times train

T

4.2 military power of 12 countries

In this case, I visualise the data that military power of 12 countries which are Japan, United States, Russia, Korea, Russia, China, Germany, England, India, France, Turkey and Canada. The attributes includes available man power, number of tanks, aircraft and aircraft carrier and strength of naval Also whether the country has nuclear weapon, and annual budget are considered. This SOM program will show the similarity of military power of different countries. Also, this SOM program will show the difference of military power by calculating the weight difference of neighbourhood of each nodes and present weight distance by using colour.

1. **Input**

The input data in this case are 12 countries with 7 dimensions of weight that are 12 attributes with 7 items that I show in the follow table.

	Japan	United States	Australia	Korea	Russia	China
Manpower	53,608,446	145,212,012	10,500,000	25,609,290	69,117,271	749,610,775
Aircraft	1,595	13,683	59	1,393	3,082	2,788
Tanks	767	8,325	59	2,346	15,500	9,150
Aircraft Carrier	1	10	0	0	1	1
Nuclear Power	0	1	0	0	1	1
Finance	49,100,000,000	612,500,000,000	1,870,000,000	33,700,000,000	76,600,000,000	126,000,000,000
Naval	131	473	53	166	352	520

	Germany	India	France	England	Canada	Turkey
Manpower	36,417,842	615,201,057	28,802,096	29,164,233	15,786,816	41,637,773
Aircraft	710	1,785	1,203	908	404	989
Tanks	408	3,569	423	407	201	3,657
Aircraft Carrier	0	2	1	1	0	0
Nuclear Power	0	0	1	1	0	0
Finance	45,000,000,000	46,000,000,000	43,000,000,000	53,600,000,000	18,000,000,000	18,185,000,000
Naval	53	184	120	66	67	115

Table 4 input of military power of 12 countries

Based on table 4, I store input in a 2D array countries.

```
float[][] country = new float[12][7] ;
```

T

However, there is an obvious difference from case 1 that is the range of each dimension is way different from others. For instance, the range of finance is 0 to 100,000,000,000 and the range of number of aircraft carrier is only from 0 to 10. Hence, to solve this problem make all the range to be [0,1]. I modify the 2D array to let each value divide the biggest value in its dimension to make sure every elements can between 0 and 1. For instance the manpower of Korea is 25,609,290 and the max manpower among these 12 countries is China with 749,610,775. So the values for the manpower part of Korea should be 25,609,290/749,610,775=0.034.

To apply this, modify all the elements by this loop.

```
float[] max=new float[7];               //define max number to each

                                        //dimensions.

for(int k=0; k<7;k++){

max[k]=0; for(inti=0;i<12;i++){

if(country[i][k]>=max[k]){

max[k]=country[i][k];              //find the max element in each

                                     //dimension.

}

}

for(inti=0;i<12;i++){

country[i][k]=country[i][k]/max[k];     //let every elements divide the

                                          //max element.

}

}
```

The idea of this this loop is to find the max value in each dimension and let very elements divide the max values to make the range of each dimension is [0,1].

2. Map

I use exactly the same shape of screen as case 1 colour map that the screen is 600*600 and the iteration time is 5000.

```
intmaxIters = 5000;
intscreenW = 600;
intscreenH = 600;
```

The map is 40*40 as case 1 but in this case the dimension number should be 7 as there are 7 attributes.

```
som = new SOM(40, 40, 7);

som.initTraining(maxIters);

iter = 1;
```

T

3. Train

To train the weight of each dimensions of vectors, I set the iteration time to be 5000, and program will pick up a random country and train its weight. The train step is exactly the same as case 1.

```
int t = int(random(12));

if (iter<maxIters&&bGo){

som.train(iter, country[t]);

iter++;

  }
```

4. Render

In this case I want to show the similarity of military power of 12 countries[3] and use the colour to present their difference, to achieve this, I use 2 render methods.

The first 1 is to find the BMU node, locate them and display as text as corresponding country name, I applying best-match method which to find BMU in this render

```
intndxComposite = bestMatch(w);

int x = ndxComposite>> 16;

int y = ndxComposite& 0x0000FFFF;

for(inti = 0; i<mapWidth; i++) {

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

if(i==x && j==y){


if(t==0){

fill(255);

textSize(20);

text("JPN",i*pixPerNodeW, j*pixPerNodeH);

}


. . . // there should be 10 countries here which I leave out


if(t==11){

fill(255);

textSize(20);

text("TUR",i*pixPerNodeW, j*pixPerNodeH);
```

[3]The data is from the http://www.globalfirepower.com/

T

```
}
}
}
```

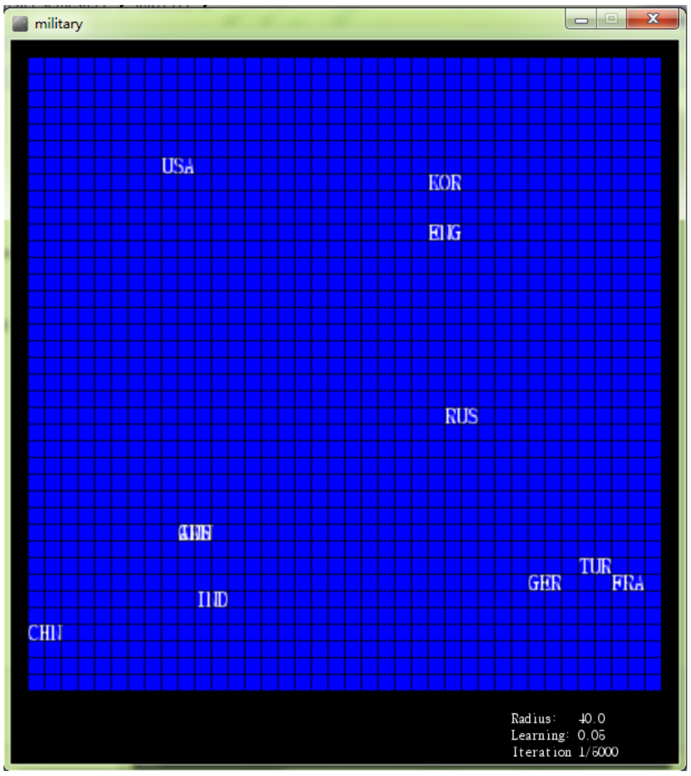
The second one is to show the weight different of neighbourhood and present it as colour blue in each node. Since I use rectangle to present each node, each node has 8 neighbourhood nodes, so I calculate the sum of weight distance and use this value to show the colour blue, in this render I apply the weight-distance method to find the weight different.

```
for(inti = 1; i< mapWidth-1; i++) {
for(int j = 1; j < mapHeight-1; j++) {
for(int z = i-1;z<i+2;z++){
for(int k = j-1;k<j+2;k++){
if(!(z==i)&&(k==j)){
floatneighbouWeight=0;
float colour;
neighbouWeight=neighbouWeight+weight_distance(nodes[i][j].w,nodes[z][k].w);
colour=neighbouWeight*255*10;
fill(0,0,colour);
stroke(0);
rectMode(CORNER);
rect(i*pixPerNodeW, j*pixPerNodeH, pixPerNodeW, pixPerNodeH);
}
}
}
}
}
```

5. Result

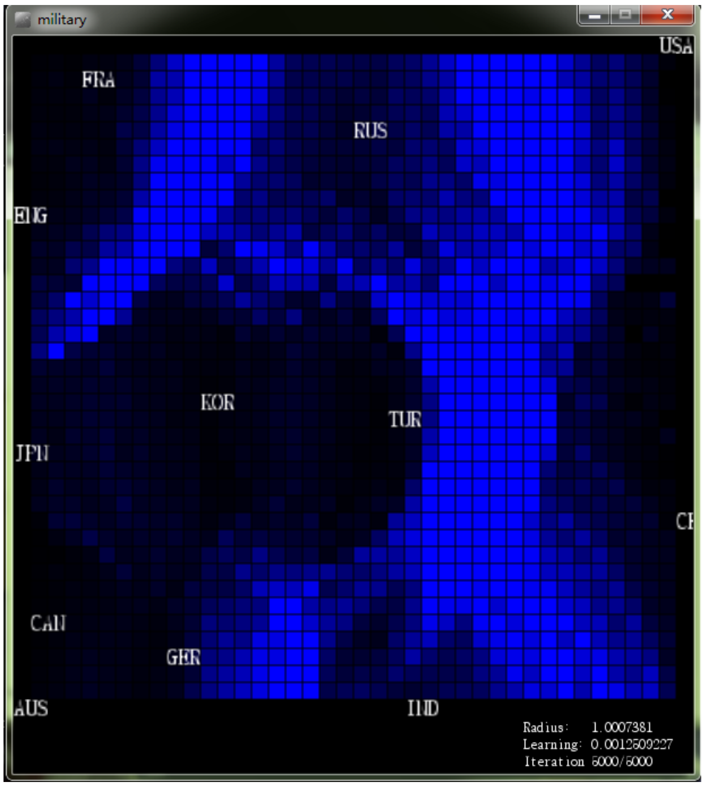
Initially, as every nodes assigned random values, the country will distributed randomly, and all nodes will display blue as they have obvious weight-distance of their neighbourhood. After 500 times training the map wills how the similarity of military power and show the difference of them.

T



Radius: 40.0
Learning: 0.05
Iteration 1/6000

Figure 5 the initial map of military power



Radius: 1.0007381
Learning: 0.0012509227
Iteration 6000/6000

Figure 6 the similarity of military power after 500 times train

From the figure 6, we find that there are 5 area which are area1 France and England, area2 Russia, area3 Korea, Turkey, Japan, Canada, Australia and Germany, area4 United States and China and

T

area5 India. This results shows, the military power of those countriesthat in the same area should be similar. Also, if the geometrical distance of two country is close, means the military power of these 2 country will be similar. For instance, Canada and Australia is in the same area and pretty close means these 2 countries is very similar. United States and China is in the same area but the geometrical distance is large so there are some difference of military power between these 2 countries.

However this result is not completely as the number of dimensions is too small and the data cannot be guarantee to be accurate.

5 Conclusion

As these 2 specific cases show, the general idea that to implement Self–Organizing Map using Processing can be applied to both cases without changing the class SOM and node, and use exactly 4 steps to achieve that. The only difference in these 2 cases are only different input and renders. Hence, I prove that 4 steps that input, map,train and render (INTR) can be applied in most cases and the class SOM and class node can be used in most SOM program.

There are 2 advantages that using Processing to implement SOM. For one thing, Processing is a sketching program that developed to design the visual arts like animation and painting and provide users an easy way to develop visually oriented applications. So users can do not need to display map by writing extra class and spend extra work to display the nodes ,instead , users can simply display the node with any shape like rectangle and colour by single line of command. For another thing, Processing support all Object-Oriented language including Java, C++ and Python which enables users to create the objects SOM and node to implement Self-Organizing map. Also, since Processing provide these programing languages, users can just choose one that they are familiar with that will make users use it conveniently. For example, users can just store the weight in a 1D array as what they do in Java.

6 Acknowledge

I would like to thank Professor MasahrioTakasuk for being supervisor of me, letting me take this project and giving me a lot of help and suggestions. Also I would like to thank Dr Florence Wang and Dr JINA CHUNG for their very good and useful suggestion.

7 Appendix

These 2 class is originated from class made by others, which give me the idea. Then I change and modify the class by deleting some unnecessary codes and adding some new codes to make it can work properly and applied in Processing2

T

7.1 class node codes

```
class Node
{
int x, y;
intweightCount;
float [] w;
Node(int n, int X, int Y)
{
    x = X;
    y = Y;
weightCount = n;
    w = new float[weightCount];

    // initialize weights to random values
for(inti = 0; i<weightCount; i++)
    {
w[i] = random(0, 1);
    }
  }
}
```

7.2 class SOM codes(without render)

```
class SOM
{
intmapWidth;
intmapHeight;
Node[][] nodes;
float radius;
floattimeConstant;
floatlearnRate = a;  //define this number later
intinputDimension;

SOM(int h, int w, int n)
 {
mapWidth = w;
mapHeight = h;
radius = (h + w) / 2;
inputDimension = n;

nodes = new Node[h][w];
   // create nodes and initilize map
for(inti = 0; i< h; i++){
for(int j = 0; j < w; j++) {
nodes[i][j] = new Node(n, h, w);
nodes[i][j].x = i;
nodes[i][j].y = j;
     }
   }

 }

voidinitTraining(int iterations)
 {
timeConstant = iterations/log(radius);
 }

void train(inti, float w[])
```

T

```
 {
radiusDecay = radius*exp(-1*i/timeConstant);
learnDecay = learnRate*exp(-1*i/timeConstant);

   //get best matching unit
intndxComposite = bestMatch(w);
int x = ndxComposite>> 16;
int y = ndxComposite& 0x0000FFFF;

//scale best match and neighbors...
for(int a = 0; a <mapHeight; a++) {
for(int b = 0; b <mapWidth; b++) {


float d = dist(nodes[x][y].x, nodes[x][y].y, nodes[a][b].x, nodes[a][b].y);

float influence = exp((-1*sq(d)) / (2*radiusDecay*i));

if (d <radiusDecay)
for(int k = 0; k <inputDimension; k++)
nodes[a][b].w[k] = influence*learnDecay*(w[k] - nodes[a][b].w[k])+
nodes[a][b].w[k];

     }
   }

 }

intbestMatch(float w[])
 {
floatminDist = sqrt(inputDimension);
intminIndex = 0;

for (inti = 0; i<mapHeight; i++) {
for (int j = 0; j <mapWidth; j++) {
floatdis = weight_distance(nodes[i][j].w, w);
if (dis<minDist) {
minDist = dis;
minIndex = (i<< 16) + j;// fisrt 16 bits for x last 16 bits for y
       }
     }
   }


returnminIndex;
 }

floatweight_distance(float x[], float y[])
 {
if (x.length != y.length) {
println ("array length don't match");
exit();
    }
floatdis = 0.0;
for(inti = 0; i<x.length; i++)
dis = sq( (x[i] - y[i]))+dis;
dis = sqrt(dis);
returndis;
 }
}
```

T

8 Reference

1. Kohonen, Teuvo. *Self-organizing maps*. Vol. 30. Springer, 2001.
2. Reas, Casey, and Ben Fry. *Getting Started with Processing*. " O'Reilly Media, Inc.", 2010.
3. Reas, Casey, and Ben Fry. *Processing: a programming handbook for visual designers and artists*. Vol. 6812. Mit Press, 2007.
4. *Next-Generation User-Centered Information Management* **Jing Li**
5. http://www.globalfirepower.com/ military power data
6. http://www.processing.org/ processing tutorial
7. http://www.ai-junkie.com/ann/som/som1.html tutorials of SOM
8. http://www.cs.bham.ac.uk/~jxb/NN/l16.pdf foundation of SOM
9. http://davis.wpi.edu/~matt/courses/soms/applications of SOM
10. http://www.jjguy.com/som/SOM codes

T