Implement Self-Organizing Map Using JavaScript

Zhenghao Chen

November 23, 2014

Abstract

This report will introduce the web-based implementation of Self-Organizing Map that using JavaScript to implement the neuron work of map. Different from typical approaches, this paper will illustrate an object-oriented approach that implement Self-Organizing Map algorithm by defining class SOM and Node. There are 4 main parts, in the first part object-oriented implementation, I will explain how I build my SOM class, Node class and how to implement the Euclidean distance function, decay function and neighbourhood function in my class. In second part map design, I will explain how to use hexagons to design a 2 dimensions map for rendering U-matrix and BMU on the map. In third part, some typical non-linear data like animal data set will be input in my program and then output rendering map to show the web-based application. Finally in conclusion part, I will summary how my project run and compare object-oriented way and no object-oriented way to show disadvantages and advantages of them

1 Introduction

1.1 Self-organizing map

Self-Organizing Map (SOM) or self-organizing feature map (SOFM) is an artificial neural networks invented by Prof Teuvo Kohonen in 1982. In the data-visualisation area, we normally use this methodology to visualise non-linear data like iris data set (Ronald Fisher 1936). This method is quite popular for data visualisation as it is a un-supervisor way. It has 4 main algorithms which are initialization, competition, cooperation and adaptation.

Initialization: creating a 2D-array map with nodes in the lattice and each node has arbitrary value in its different dimensions.

Competition: using Euclidean distance function to calculate the weight-distance of each node and input vector. Then define a node with minimum weight-difference as best-matching unit (BMU).

Cooperation: using Radius Decay function to locate the neighbourhoods of BMU in each iteration time.

Adaptation: training all the nodes inside neighbourhood circle, whose centre is BMU using neighbourhood function $W(t+1)=W(t)+L(t)(V(t)-W(t))$ defined by Teuvo Kohonen.

1.2 JavaScript

JavaScript is a dynamic programming language which is mainly used to web development. It is interpreted and supports object-oriented design in prototype way. In fact, using JavaScript is quite common way to implement self-organizing map in web-based application that we can find many instances in reference. Also, in my web-based SOM implementation, I try to use JavaScript to build my self-organizing map.

Most of instances implement SOM in no object-oriented way that simply training the input vector by using self-organizing map algorithms. However in my project, I define the SOM and Node using CoffeeScript and then compile it into JavaScript and rendering my map using a JavaScript library d3js.

2 Non-object-oriented Implementation study

Non-object-oriented implantation is a typical approach for JavaScript users. There are 2 main reasons. For one, JavaScript cannot support object in class way, instead user must have prototype to call their object. For another, non-object oriented design will save more memory as it does not require any memory for storing object and class. For this reason, it normally runs faster than object-oriented application.

The idea to use non-object oriented approach is quite straightforward. For instance, one of typical ideas is that users just need to create SVG node and then set attributes as its weight and position and then directly use self-organizing map algorithms to train all the SVG then append then on the screen.

Since non-object-oriented approach is not my methodology to implement SOM, hence there will not be any details explanation about this approach in this paper. Instead, I will introduce my approach in details that using object-oriented way in next part.

3 Object-oriented Implementation of SOM

Although JavaScript is an object-oriented language, it supports object-oriented design in prototype way that means we cannot use object by defining class. For this reason, most of designers used no object-oriented way that simply training the input vector by using SOM algorithms. However in my project, I define the SOM and Node class using CoffeeScript. In this part, I will illustrate how to implement 4 algorithms initialization, competition cooperation and adaptation in my SOM class as well as how to make Node class.

3.1 CoffeeScript

CoffeeScript is a little language that can compile to JavaScript. The Syntax of CoffeeScript is inspired from Ruby and Python. There are 3 general steps that I use this language that are installing, coding and compiling. Firstly I use npm and node.js to install the tool and library of CoffeeScript and then I write my syntax in CoffeeScript. Most syntax is the same or similar as JavaScript. But different from JavaScript, CoffeeScript provide a structure that we do not have to use prototype to make and call object but just need to define a class. So I simply use CoffeeScript to construct my SOM and Node class.

3.2 Yeoman

Yeoman is website helping web designer to complete and test their application. To compile my work and test whether my class can work in html file, I used its grunt server to compile my CoffeeScript file into JavaScript. By using grunt server, Yeoman compiles my CoffeeScript file into JavaScript file and test it on its web server.

3.3 Node

The idea to build the node class is that node has 2 elements that position and weight. As all the nodes are located onto a 2-dimensions array map, the position of node will compose of X- axis and Y axis. Also, I define weight as a 1 dimension array where the number of dimensions of weight is size of array.

Once a node is created, on the one hand, it will have arbitrary value for its weight which is greater than 0 and less than 1 in each dimension. On other hand, values of position will be null in the beginning and then defined in SOM class later depends on the size of SOM map. So in my constructor, I only use math random function to assign the value to elements of array weight. By this way, if we call the node class, a node object will have

Node: x: null, y: null, weight: [n_1,n_2, n_3] where n_1,n_2, n_3 are less than 1 and greater than 0.

Algorithm1: node class

Class node:

X-axis

Y-axis

array: weight

constructor: (number of dimensions: n)

```
for( i =0, i < n, i++)
        Weight[i] = random number from 0 to 1
```

3.4 SOM

The SOM class is the main class of the whole program which contains 4 algorithms initialization, competition, cooperation and adaptation and the main formulas of SOM like Euclidean distance function, radius decay function, learning rate decay function and neighbourhood training function will be implemented in this class.

3.41 Initialization

First step of self-organizing map is to create a 2-dimension map and assign the nodes in the lattice. So in the initialization part, I will do 2 things. For one thing, I will assign SOM some initial variables which are size of map (height and width) initial radius and learning rate, input dimension and max iteration time. And for training step later I create a time-constant which is iterations / log (initial radius). For another, I will create nodes where number of them equals to size of map and store them in a 1 dimension array. In this stage, I am not creating an actual 2 dimensions map, instead, I will create an “abstract map” by giving each node position values. The position values of nodeare undefined when it is created immediately. Later, by assigning X-axis and Y-axis values to them, I assume they are located in a 2 dimension map, where X-axis and Y-axis present the geometrical distribution on 2-dimension map of them. Here the X-axis is (node. index mod map. height) and Y-axis is (map.index / map.height). By that way, I implement a 2-dimension map abstract data type by using 1-dimension array.

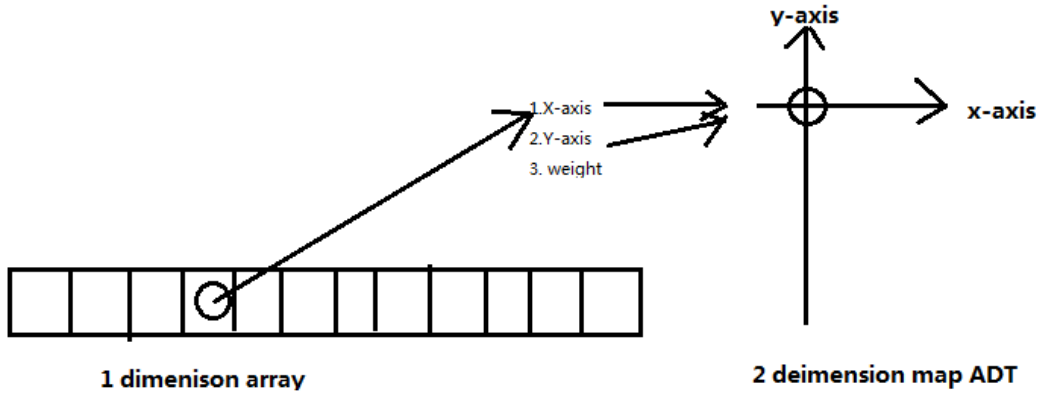


Figure 1 map ADT

Algorithm2: initialization of SOM

constructor:

input dimension

iteration

width

height

initial radius

initial learning rate

array: nodes

time-constant= iterations / log(initial radius)

for(i=0; i<mapSize; i++)

nodes[i]= new Node(Input dimension) // create a node

nodes[i].y= i/height

nodes[i].x= i%height //locate this node in 2D map

3.42 Competition

This algorithm includes 2 steps: the first step is to implement Euclidean distance formula to calculate the difference of 2 vectors. The second step is to find the best-matching unit by comparing all the nodes to the input vector and find a node with minimum weight-distance which will be defined as best-matching unit(BMU).

To implement the Euclidean distance function, I create a method call weight-distance, this method will give the absolute value sum of the difference of elements from 2 array. If 2 array, array1 and array 2 have k dimensions where array1 = $[n_1, n_2, n_3 \ldots n_k]$, array2=$[m_1, m_2, m_3 \ldots m_k]$ the weight distance will return the $\sqrt{(n_1 - m_1)^2 + (n_2 - m_2)^2 + \cdots + (n_k - m_k)^2}$and that is the weight distance we need work out from 2 vectors

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2} \qquad (1)$$

Algorithm3: competition step 1 weight-distance

weight-distance(array1 ,array2)

integer tmp=0

```
        for(i=0; i<array1.length; i++)
                tmp+=square(array[i]-array2[i])
return   square root(tmp)
```

Once we have previous method, we then, can look through whole map to find the node with minimum weight-distance from input vector, and return the position of the node we found. I assign the default minimum distance as number of input dimension as the difference of 1 dimension of 2 vectors must be less or equals to 1 assuming values of each dimension is less than 1 and greater than 0(defined in the node class).

Algorithm4: competition step2 best-match

```
Best_match(input vector)
        float minDistance = number of input dimension
        integer minIndex  = 0;
        float tmp = 0;
        for(i=0; i<mapSize;i++)
                 tmp =  weght_distance(nodes[i].weight, input vector.weight)
                if(tmp< minDistance)
                minDistance = tmp
                minIndex = I;
        return minIndex
```

3.43 Cooperation

Once best-matching unit (BMU) is located on the map, we will set it as centre and draw a circle called neighbourhood circle whose radius is calculated by radius decay formula in each training iteration time. In cooperation step, we need to draw the circle and search through the whole map to see if a node is inside the neighbourhood circle, if yes, and then we train this node in adaptation step. To complete this stage, there are 2 steps.

The first step is to get the geometrical distance of the particular nodes and the BMU to implement this simply find geometrical difference of the X-axis and Y-axis position of node n and BMM using the formula

$$\sqrt{(X_1 - Y_1)^2 + (X_2 - Y_2)^2} \qquad (2)$$

Algorithm5: geometrical distance

```
geo-distance(node1, node2)
        Return ((Node1.x-node2.x)^2 – (node1.y-node2.y)^2)^0.5
```

And the second step is using radius decay function to find the radius of neighborhood circle during each iteration time of training. As we can see from the result of radius formula decays in each time that means the circle shrinks during training. After we find the radius, we compare the radius and geometrical distance of BMU and node, if the geometrical distance is less than radius in other world, it is inside the circle, then we train the nodes. And the training step will be implemented in last step adaptation.

$$\sigma(t) = \sigma_0 e^{(-\frac{t}{\lambda})} \qquad t=1, 2, 3 \ldots. \qquad (3)$$

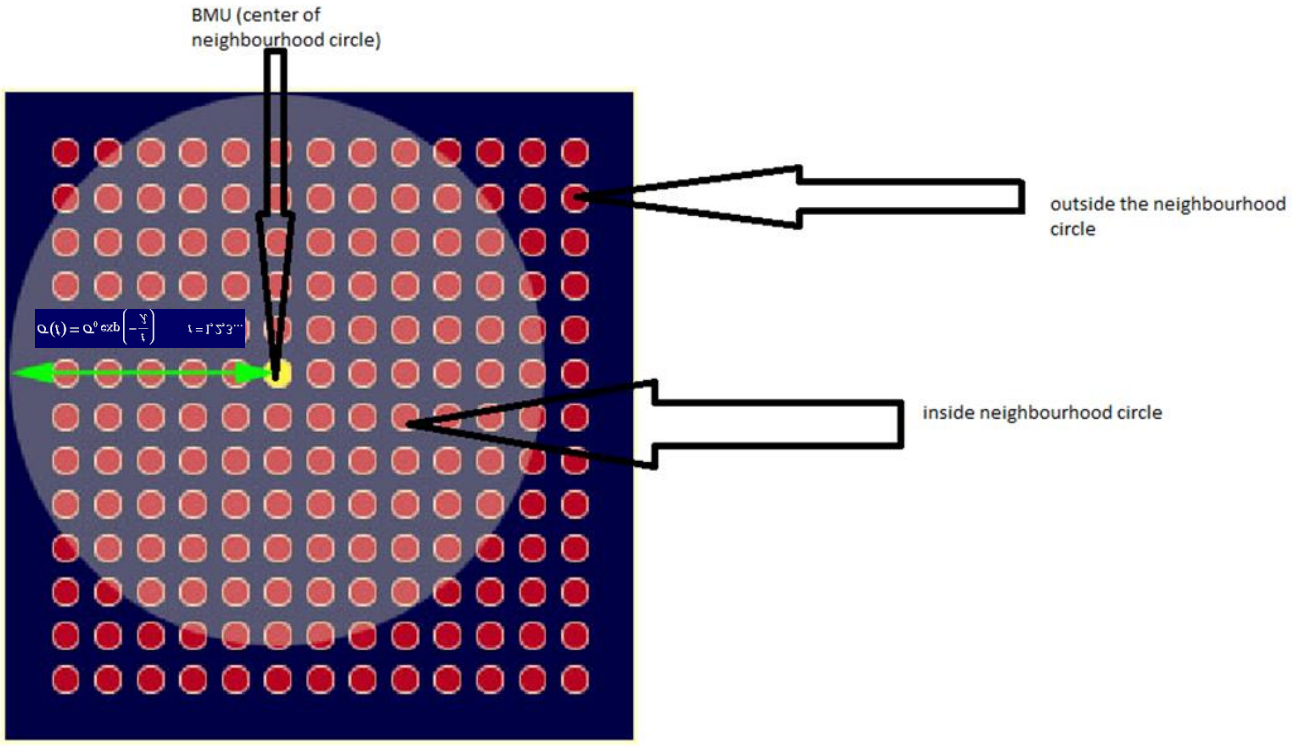


Figure 2 neighbourhood circle

3.44 Adaptation

Adaptation is the most important step of whole self-organizing map algorithms, the main idea is to train the weight of nodes to be closer to BMU and by using training formula W(t+1)=W(t)+L(t)(V(t)-W(t))to train each node inside the neighbourhood circle. Once we train all the nodes that need to be trained, one training iteration time is completed. However, before I implement this training formula, I divide this one into 2 sub-formulas that Learning Decay Formula and Neighborhood Formula.

$$W(t+1) = W(t) + L(t)(V(t)-W(t)) \qquad (4)$$

Learning decay formula is to compute the learning rate that in each training iteration time. The only factor that effects on this formula is current iteration time. From this formula, we see the learning rate decreases as the iteration time increases during training like radius decay formula.

$$L(t)=L_0 e^{(-\frac{t}{\lambda})} \qquad t= 1, 2, 3..... \qquad (5)$$

Neighborhood formula is another function that influences the weight of nodes in training step the effects of this function are the distance of nodes is currently being trained and BMU and the radius of neighborhood circle in current training iteration time.

$$\Theta(t) = e^{(-\frac{dist^2}{2\sigma^2(t)})} \qquad t=1, 2, 3... \qquad (6)$$

Algorithm6: Adaptation training

```
train (iteration i, input.weight[])
// radius of neighborhood circle
radiusDecay = initial-radius * Math.exp((-1) * i / this.timeConstant);
//learning rate of this iteration
learnDecay = initial-learnRate * Math.exp((-1) * i / this.timeConstant);
//locate BMU
BMU = best_match(input.weight)
        for(i=0;i<mapSIze;i++)
                d=geo_distance(nodes[i],BMU)
                if(d< radiusDecay)
                        // neighbourhood function
                        influence = Math.exp(((-1) * Math.pow(d, 2)) / (2 * radiusDecay * i));
                        for(k=0;k<input.dimension;k++)
                                // W(t+1)=W(t)+L(t)(V(t)-W(t)) which also equals to neighbourhood
                                function*learning rate decay formula
                                nodes[i].weight[k] += influence * learnDecay * (w[k] -
                                nodes[i].weight[k]);
```

3.45 Summary

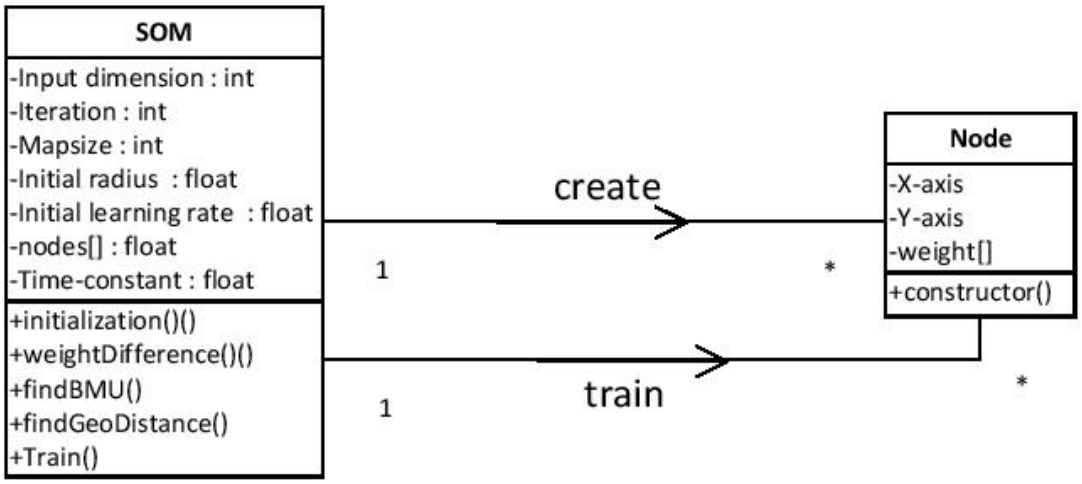


Figure 3 Structural Model -- UML class diagram of Class SOM and Class Node

The object-oriented implementation approach contains 2 class that are SOM class and Node class. In the program, we actually need only 1 SOM object and height*width Node objects to be assigned in the lattices of SOM map. Once a SOM is created, it will call the height*width nodes and gives each nodes position consists of x-position and y-position the initial weight of a node will be assigned by its constructor. Then we can start training our nodes by calling train method of SOM. It will call the best-match first to find the BMU. In this step, best-math method find the BMU by using weight-difference method to compare the weight-distance of all the nodes in map to weight of input and return the position of BMU with minimum weight difference form input vector. After we have BMU, we get the geometrical distance of BMU and all nodes in map. If the distance is less than radius of neighborhood circle we use the train function to update the weight of each nodes inside circle and we complete one training iteration. By repeating the training for the max-iteration time we set, we finish the self-organizing map algorithms.

The advantage of object-oriented implementation is that we can format the class SOM and Node, and by calling script file SOM.js and Node.js we can easily have self-organizing map training algorithms what we need to do is design a map to rendering the node that we train. However, compared to non-objected oriented self-organizing map implementation, there is an obvious disadvantage that it requires more memory to store the class and object we create.

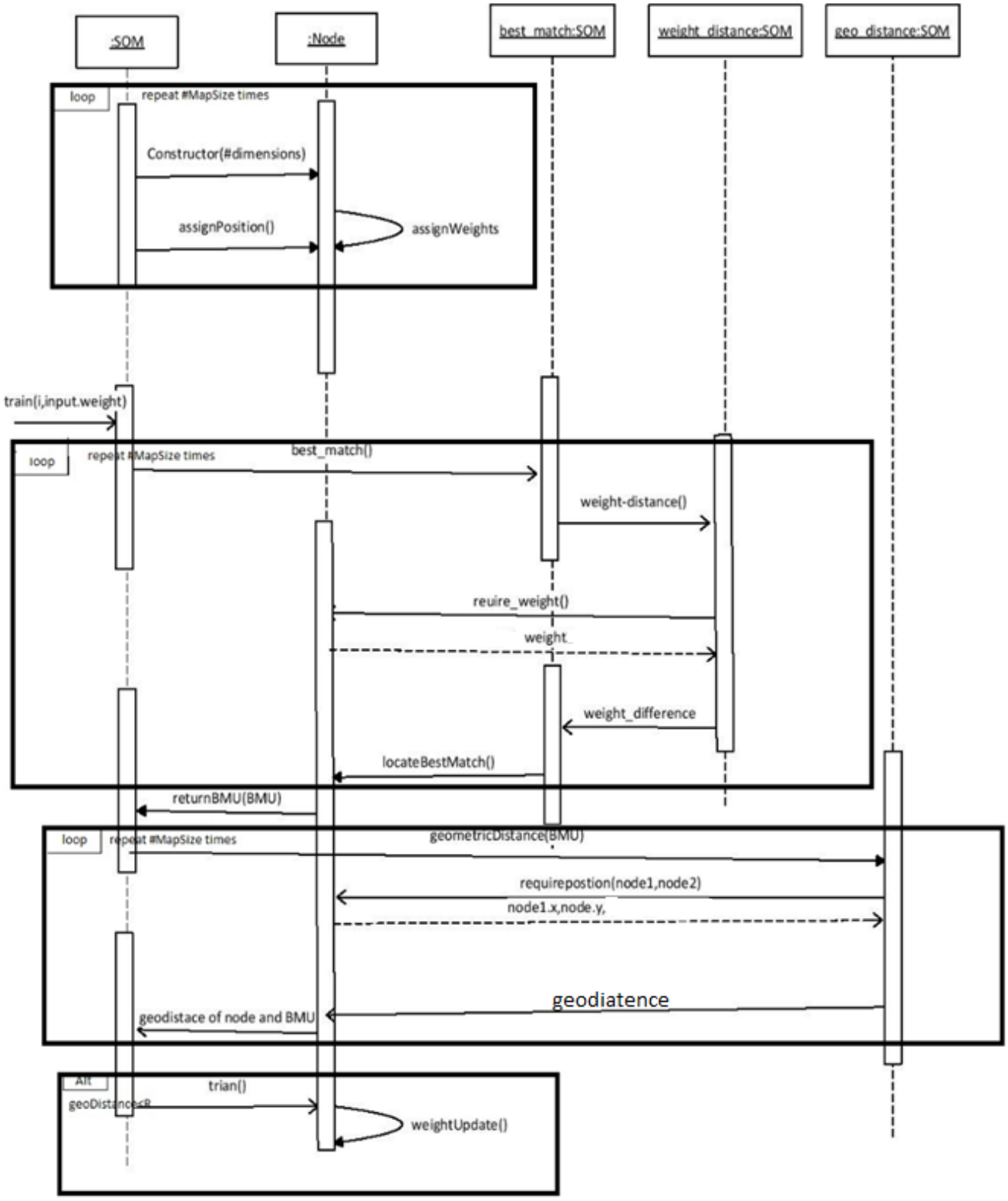


Figure 4 Behaviour Model – sequence diagram of training

4 Rendering of map

In this part, I will design a 2-dimension map to render my nodes of SOM. To achieve it, I choose hexagon map, the reason I use hexagon is because the hexagon is best geometrical shape to show the SOM as we find geometrical distance is the same of any neighbourhoods and a node.

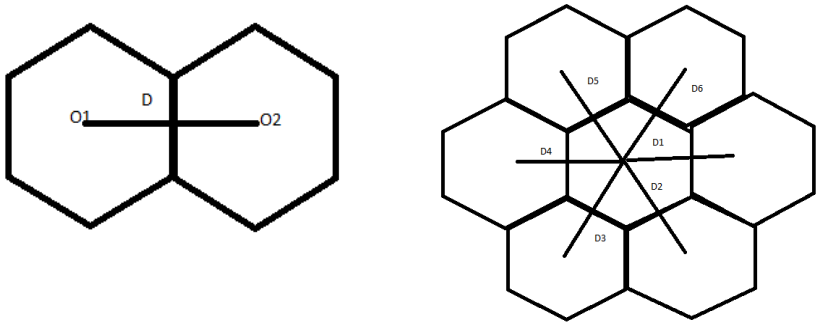


Figure 5 Hexagon distance

As shown from figure 5, if we difine the distance of 2 hexagons is O_1O_2, where O_1, O_2 are centres of 2 hexagons, we find the all the disance of neighbourhoods and a hexagon is the same(i.e. $D_1 = D_2 = D_3 = D_4 = D_5 = D_6$). For this reason, hexagon map is the best choice of SOM map.

4.1 d3js

D3js is a JavaScript library for manipulating documents based on data and drive the creation and control of dynamic and interactive graphical forms which run in web browser and widely implemented by HTML5, JavaScript and SVG creation. For easily rendering the binding data, D3js provide a lot of method in its API which enable users to complete the SVG selection and creatation and attribute completement. It contains 4 most import usabilities:

Selection: selecting particular part in html file,it can be either all html file that or some particualr part and then starting appending or adding attribute.

Data-binding: binding the data style(e.g. position or color) in the array or other data structure which will be use for node creation later.

Appending : Appending nodes as SVG or other things using binding data and API method

Attribute : Using API method or CSS to modify the attributes and completethe sytles of SVG of node which is created.

By using the method d3.hexbin() and other API methods we can create a hexagon map for our map renderring part.

4.2 design

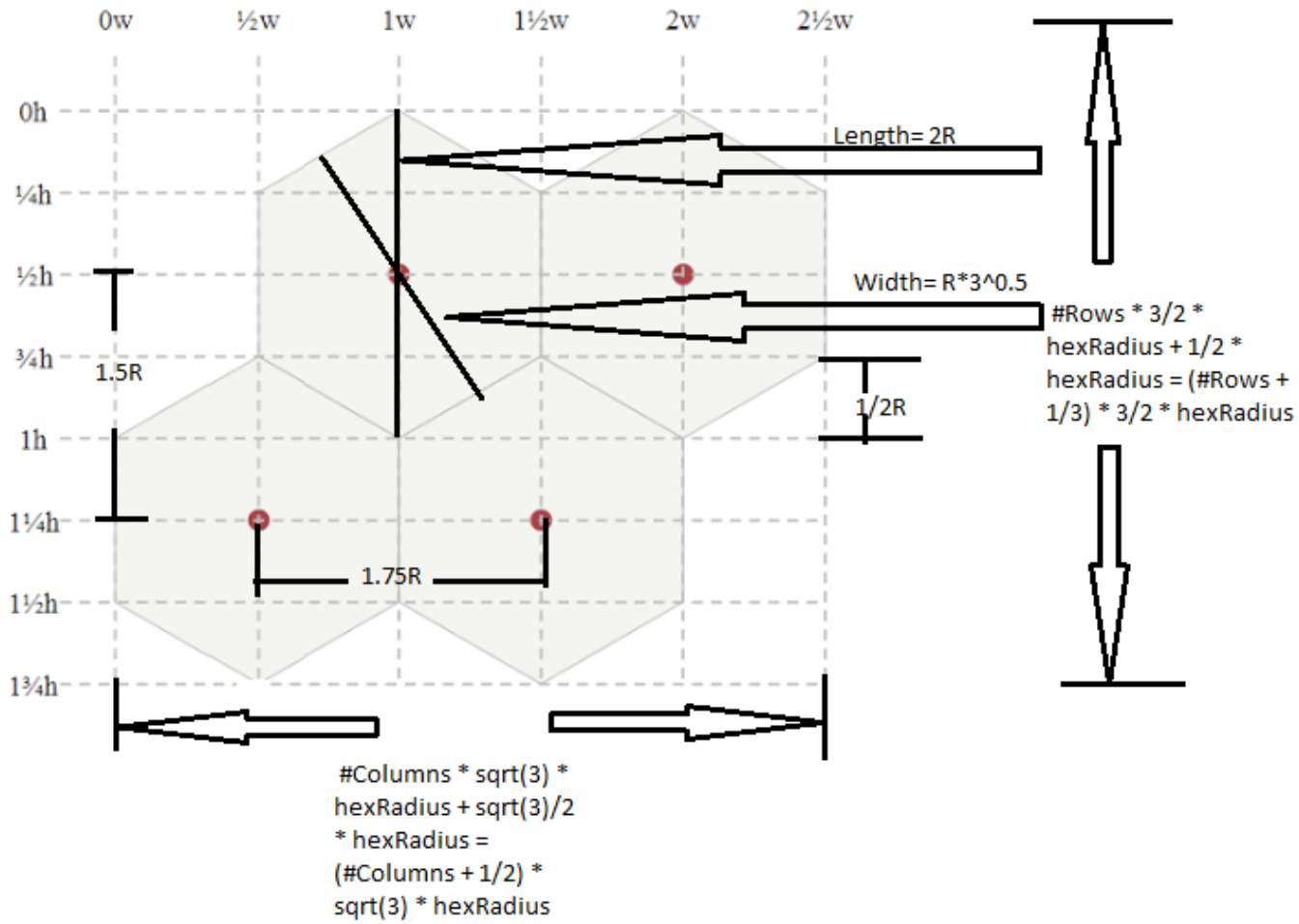


Figure 6 Hexagon map analysis

For desgining the hexagon map, we need 3 compolsory parametters for each hexagon that are radius, x-axis position, y-axis postion. For getting these 3 parametters, we need to assign the width of map, height of map, number of columns and number of rows.

We will calculate the height and width of hexagon first, we get

Equation 1: height=hexRadius * 2

Equaiton 2: width= hexRadius * $\sqrt{3}$

From figure6, we can find the realationship between height and width of map and radius and by using equation 1 and equation 2 , we find

Equation 3: length of map = #Columns * width + width/2

Resorting eqution 3 by subtituted width for hexagon raidus we get

Equation 3.1: (#Columns + 1/2) * $\sqrt{3}$ * hexRadius

And we get:

Equation4: #Rows * 3/4 * height + 1/4 * height

Resorting eqution 4 by subtituted height for hexagon raidus we get

Equation 4.1: = (#Rows + 1/3) *3/2 * hexRadius

Using Eqution 3.1 and 4.1 and we assume that maximum of radius can still fill the screen, we get the radius of hexgon is

$$min(width/((MapColumns + 0.5) * \sqrt{3}), height/((MapRows + 1/3) * 1.5)) \quad (7)$$

Then we find that the honrizontal postion x-axis and vertical postion y-axis:

Where X=Radius*x*1.75 x=0.1.2.3….MapColumn

Y=Radius*y*1.5 y=0.1.2.3…MapRows

By binding data of horizontal postion X and vertical postion Y, I can assign the hexagon on my map. My hexagon map looks like as follow

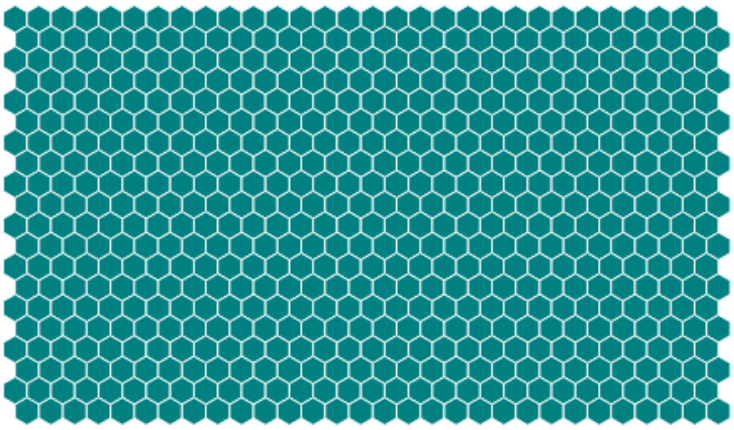


Figure 7 Hexgon map

4.3 U-matrix rendering – colour

To see the unified distance matrix, or U-Matrix, we need to find the sum of weight-distance of neighbourhoods of each node on map, hence in this stage, color will be a very important factor to analysis as I use color to present the similarity and difference of data using U-matrix. Here, I firstly create a 1 dimension array whose size is exactly the same as array nodes. Then I implement the weight-distance method of SOM to find the sum of weight-distance of 6 neightbourhoods of a node and then I use rainbow color table to salce the color to 0 to 180 and store this color value in an array whith the same index number as this node. By storing all the sum of weight-distance of 6 neighbourhoods of nodes as colour value in this array, the colour data is binding and will be used to be attributes of the hexagon map we just create. However, there are two thing we need note are

that firstly the index number set of neighbourhood of a node in hexgon map are different depends on the node is in even row or odd row. Specifically, if node.y is odd then, then index number set of its neighbourhood is

[i-#Columns -1, i-#Columns, i-1, i+1, i+#Columns -1,i+#Columns].

And if the node.y is even the index number set of its neighbor will be

[i-#Columns, i-#Columns+1, i-1, i+1, i+#Columns , i+#Columns+1].

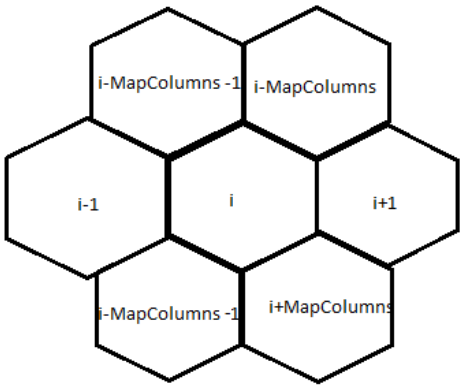


Figure 8 Index number set of neighbourhood when node.y is odd

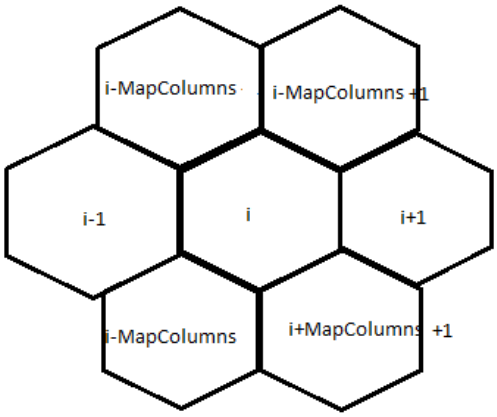


Figure 9 Index number set of neighbourhood when node.y is even

The second thing is that to use colour value in most effcient way we need to do the linear scaler which is to set the maximun value to upper limit and minimun value as lower limt to restrict the color scaler and in other word we reset all color value colour[i] to be (Colour[i]-minColor)/(maxColour-minColor)

Algorithm 7: Colour Rendering

---

array colors[]

```
for (i=0; i < mapSize; i++)
  if nodes[i].y is odd
        neighbourWeight=0
        array neighbours[]
        neighbours[0] = i-#Columns -1
        neighbours[1] = i-#Columns
        neighbours[2] = i-1
        neighbours[3] = i+1
        neighbours[4] = i+#Columns -1
        neighbours[5] = i+#Columns
        for(var j=0; j < 6; j++)
              if(0=<neighbours[j] < mapSize) //  neighbour is valid node
                    k = neighbours[j]
                    neighbourWeight +=weight_distance(nodes[i].weight,som.nodes[k].weight)
                    //sum up weight of all the neighbours
              colors[i] = neighbourWeight  // store the clour values in array
    else
        repeat what we do when nodes[i].y is odd, except the neighbours[i] is being
         [i-#Columns, i-#Columns+1, i-1, i+1, i+#Columns, i+#Columns+1]
]
// findind maximun value and minimun value for color
maxColor=0;
minColor=1
for(i=0; i < mapSize;i++){
if(neighbours[i]< minColor)
        minColor=colors[i]
if(neighbourt[i]> maxColour)
        maxColor=colors[i]
//reset color value by linear scaler
for(i=0; i <mapSize;i++)
         hue= round (colors[i]-minColor)/(maxColor-minColor)*180)
```

colors[i] = d3.hsl(hue,0.6,0.6); // store the new color vlaue

4.4 BMU rendering – text

Normally, we need to locate the BMU of a input and render it as name of input on the map to show the visualization, in order to achieve that, we create an array as the same size of nodes and implement the method best-match methd to find the index i of BMU and set the element of String array whose index is i to be name of the input and bind this array as data. After that we create SVG text using the same postion binding data for postion as hexagon map and then append the String array as the content of text on my SVG. The attributes of text like size and color only need to be modified by CSS.

Algorithm 8: String render

String[] A= new String[nodes.size]

i=best-Match(input)

A[i]=input.name

4.5 Summary

By using d3js API methods d3.hexbin() for creating hexagon map and d3.hsl() or d3.rgb() for colour rendering, it is not hard to render the whole map. And we actually have 2 types of SVG. The first one is hexgon and the second one is text. The data they bind for postion are the same that x postion and y postion calculated by the hexagon radius while the content data they bind are different. For hexagons, it uses color which is calculated by weight-distance and show as U-matrix setting as its attibute. However, for text, it uses content which is a String array whose element is set by method best-match and show as BMU for visualization and style is only modified by CSS.

	Hexagon(u matrix)	Text(BMU)
Selection	Chart	Text
Data Binding(postion)	x,y postion caluculated by hexagon method	
Data Binding(content)	array of color values	Array of String
Appending	d3.hexbin()	Append text
attribute	d3.rbg or d3.hsl and CSS	CSS

table 1 implemtation of renderring map

5 Typical Instance Data

To prove that my class SOM and calss Node are working, I will use some typical non-linear data as input vectors to show self-organizing map. There are 4 case studies which are hello world, animal data set, iris data set and power of countries. In hello world, I will input 10 different and show similarity of them. In animal data, 16 animals will be considered as input vetcors with 16 dimensions that present their features. A different analysis approach component plane study will be introduced in iris data. Lastly, I make some research about power of countries that I use 12 countries as my input vectors and have 2 big features which are soft power and hard power. Where there are 6 sub-attributes for both soft power and hard power.

5.1 Hello World

Hello world is the data set that normally be used to test the work of SOM. The input data includes 10 different colors that differ by 3-dimension input vector RGB, for rendering the map I d3js API method d3.rgb() to show color of each nodes.

Data

10 different colours 3 dimensions data size3*10

	Red	Green	Blue
Color1	255	0	0
Color2	0	0	255
Color3	127.5	0	1
Color4	0	0	255
Color5	255	0	127.5
Color6	255	127.5	0
Color7	127.5	127.5	0
Color8	255	63.75	255
Color9	63.75	0	255
Color10	0	0	0

Table 2 color input

Map

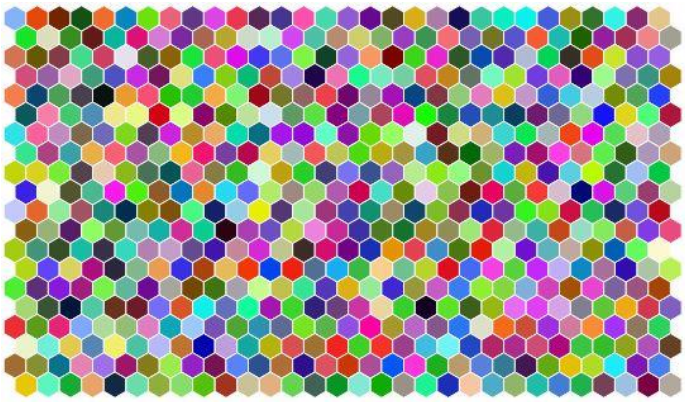


Figure 10 initial random color

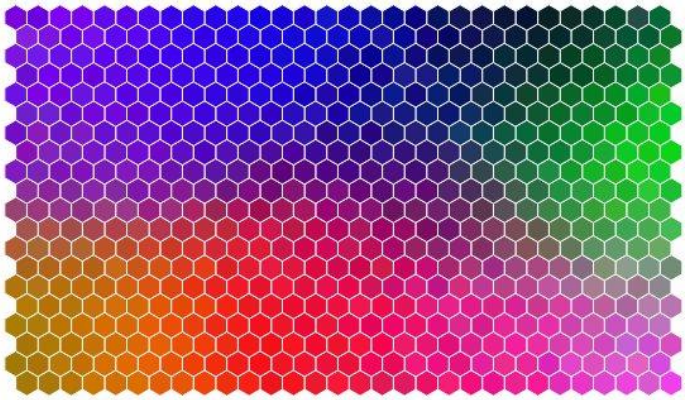


Figure 11 100 times traning

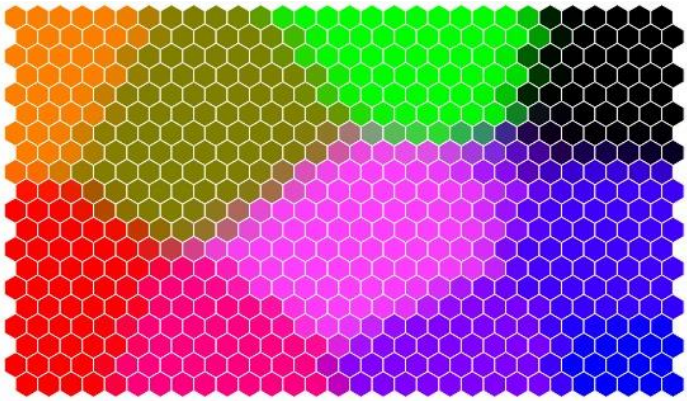


Figure 12 1000 times trainig

Sumary

As we can see, in 100 training time, we can find the difference and similiraty of 10 colorsbut it is not vey obvious while in 1000times trainig it is quite obvious to tell this 10 different colors. In that instance, we can say the SOM program works pretty good and it can train nodes on the map properly.

5.2 animal data

Animal data is another very classical and typical non-linear data set from Teuvo Kohonen ,Self-Organizing Map 2001. It illustrates the similarity of 16 animals which are dove, hen, duck, goose, owl, hawk, eagle, fox, dog, wolf, cat, tiger, lion, horse, zebra and cow by using their features(atrributes as input vector) that inludes 16 dimensions like size, feathers and so on. The aim of this data visualization is to visualise the similarity of these 16 animals. To visualize it, I use d3js API method d3.hsl() to scale the weight-difference as color to render U-matrix on the map and use text as name of animals to present the BMU on the map

Data

From Teuvo Kohonen ,Self-Organizing Map 2001

16 animals 16 dimensions data size 16*16

		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 3 animal data input

Map

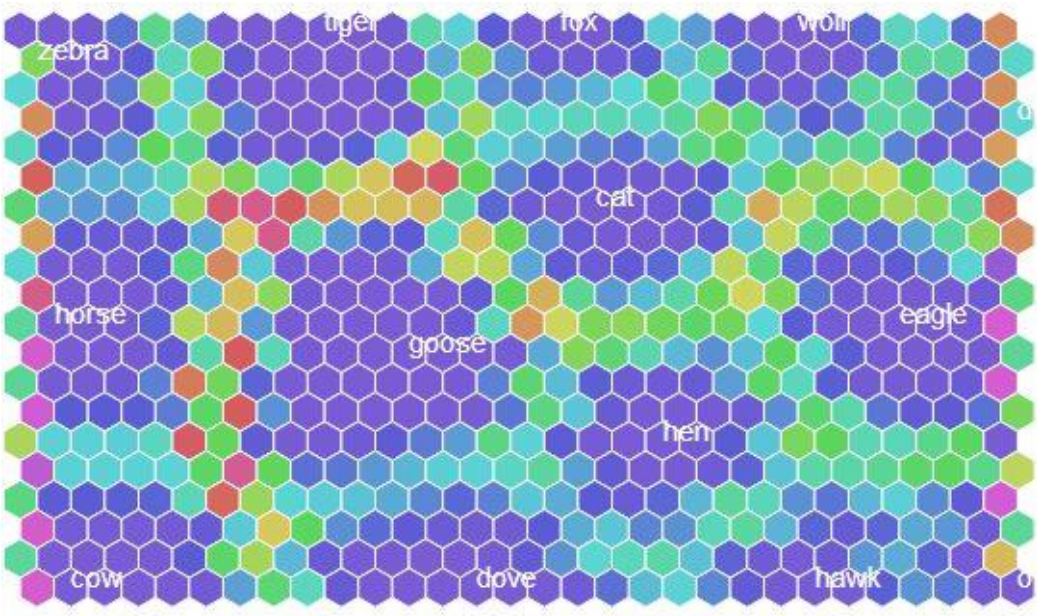


Figure 13 animal data map

Summary

Based on this U-matrix map we can see that 16 animals differ by 16 different obvious clusters where strongest difference occurs among goose, horse and tiger as we see the color. Then difference among goose, cat, hen and eagle is obvious as well but not as strong as previous one. Generally, this map show the good work of rendering.

5.3 iris data

The Iris data set or Fisher's iris dataset is multivariate data set introduced by Sir Ronald Fisher in 1936 as an example of discriminant analysis. The data consists of 50 samples from each of three species of Iris flowers which are Iris Setosa, Iris Virginica and Iris Versicolour. With 4 dimensions for its features that are sepal width, sepal length, petal length and petal width, I introduce 150*4 2-dimension vector as input data in this case study to show my map. In this case I do not only aims to see the similarity of 4 dimensions input instead, another analysis approach component plane analysis which considers the effects of each components will be implemented. Using this methodology, we need to render U-matrix of each component plane individually.

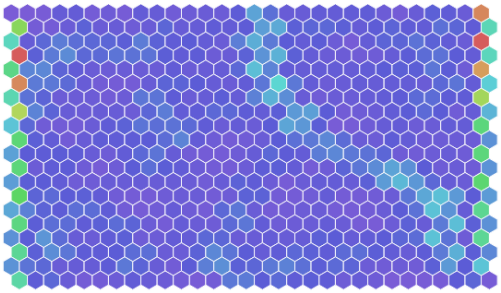
Data

From iris data set, Ronald Fisher 1936,

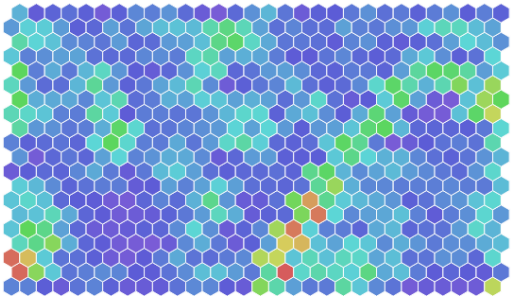
150 iris 4 dimensions sepal width, sepal length, petal length and petal width, size 150*4

As these data is big so I do not show the data set here instead I will attach it in the appendix part.

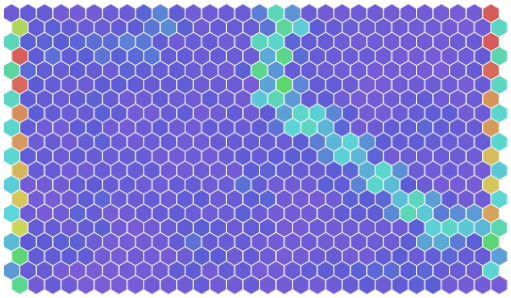
Map



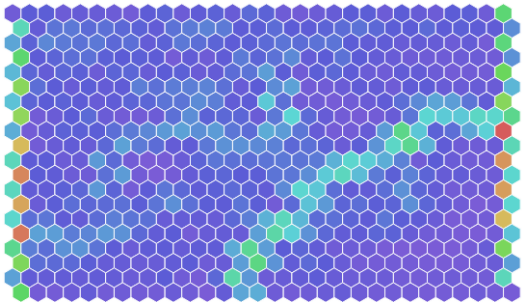
Component plane1 Sepal length



Component plane2 Sepal Width



Component plane3 Petal length



Component plane4 Petal width

Figure 14 component planes of Iris data

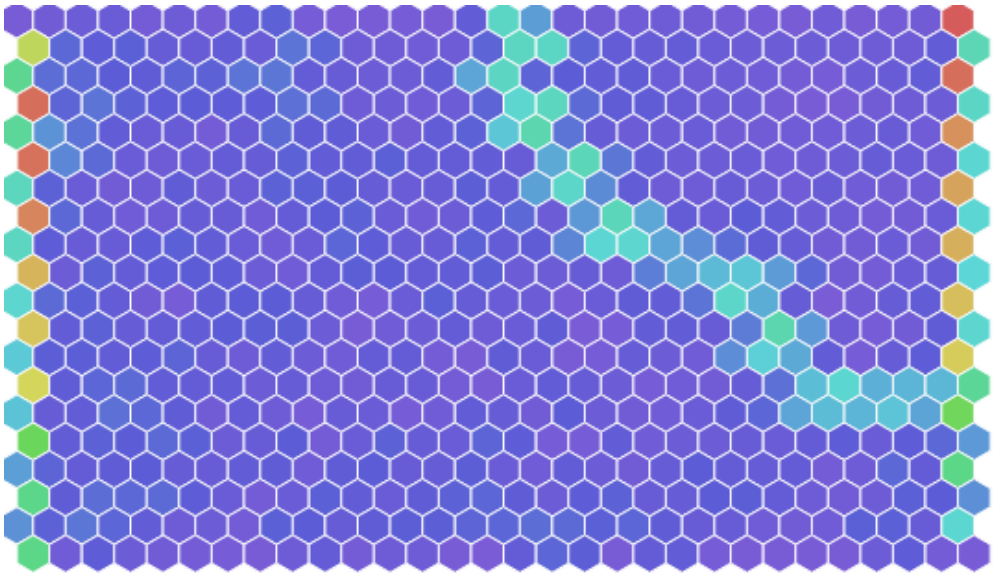


Figure 15 combined Iris data

Summary

As seen from Figure 14, the component plane2 sepal width shows very obviously different from other 3 componets. Where in component plan 1,2 and 4 we can easily distinguish 2 clusters dicreactly from map however it seems unreable from component plane 2. Assuming that both the self-organizing map training algorithms U-matrix renderring are correct, I make assumption that component plane 2 Sepal Width has different effect than other 3 components. Howvever, this assumption needs to be proved futher with more sufficient evidence.

5.4 Power of Countries

When we say wheter a country is powerful or not, we normally consider its hard power and soft power. Where hard power includes militray power, science, technology and so forth while soft power consists of economics, eduation, agriculture and so on. In this case study, I will visualise the similarity of power of 12 countries that are Japan, Unitied States, Australia, Korea, Russia, China, Germany, India, France, England, Canada and Turkey.

5.41 Soft Power

In the case of soft power visualization, I choose the typical soft power that economics, the data is oringinated from World Bank Database where I choose 7 attributes that are GDP, GDP(by PPP method), GNI, GNI(by Atlanta method) and GNI(by PPP method) , population and foreign investment as input dimesions. To see the visualtion, I render 2 things which are U-matrix and BMU of 12 countries.

Data

From http://data.worldbank.org/ world bank database 2012

12 countries, 7 dimensions, size 12*7

	GDP	GDP(PPP)	GNI	GNI(Alt)	GNI(PPP)	POP	For investment
JPN	4901530	4624359	5875019	46140	37630	122332.4	3,715,043,055
USA	16800000	16800000	16967740	53670	53960	316128.8	235,867,000,000
AUS	1560597	1007353	1515558	65520	42540	23130.9	49,396,157,289
KOR	1304554	1664259	1301575	25920	33440	50219.67	12,220,700,000
RUS	2096777	3461259	1988216	13860	23200	143499.9	70,653,718,709
CHN	9240270	16157704	8905336	6560	11850	1357380	347,848,740,397

FRA	2734949	2436930	2789619	42250	37580	66028.47	6,480,400,817
GER	3634823	3493479	3716838	46100	44540	80621.79	32,627,493,896
ENG	2521381	2320104	2508965	39140	35760	64097.09	48,314,454,024
CAN	1826769	1520493	1835383	52200	42610	35158.3	67,581,373,072
IND	1876797	6774441	1960072	1570	5350	1252140	28,153,031,270
TUR	820207	1421881	820622	10950	18760	74932.64	12,868,000,000

Table 4 economics of 12 countries

Map

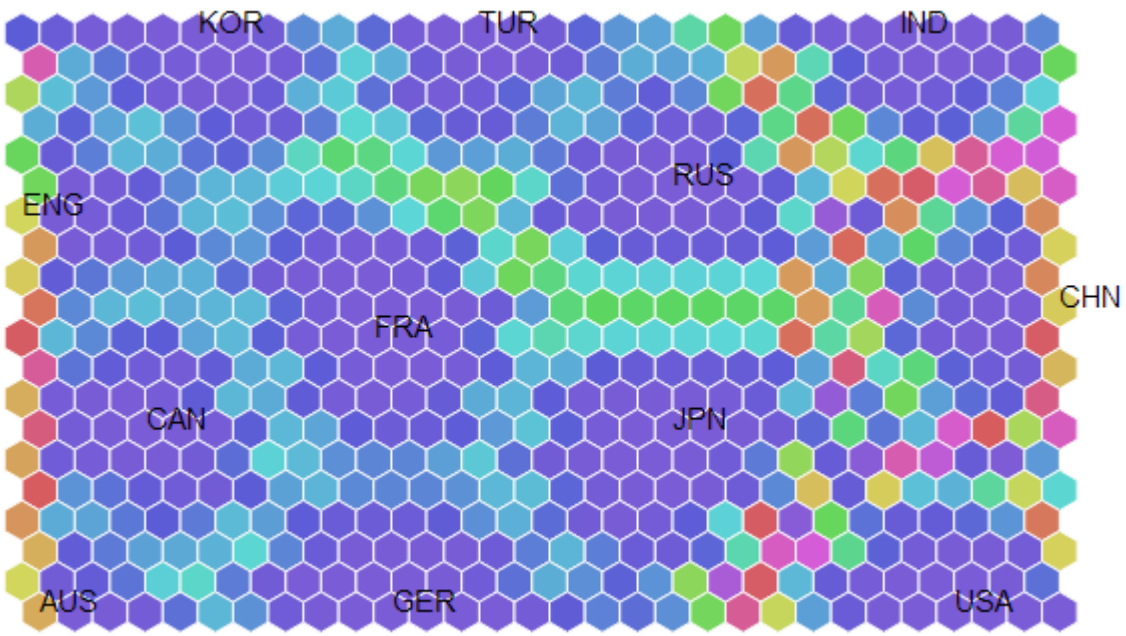


Figure 16 soft power among 16 countries

comment:

As we can tell from the map, on the one hand,India, China and USA have most different from other 9 countries which is very obvious. On the other hand, we cannot know what reasons contribute those difference as it can be difference of either value or structure.

5.42 Hard Power

I use the military that is most typical effect of hard power of a country to visualise the hard power similiarity of 12 countries. The attributes are avialible manpower, quatity of aircaft, tanks, naval and aircraft carrier as well as wether a country has nuclear weapon and finace support of army in each country. As what I did for previous instance I use U-matirx and BMU to visualise the map.

Data

From http://www.globalfirepower.com/ 2013

12 countries, 7 dimension, size 12*7

	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 5 militrary of 12 countries

Map

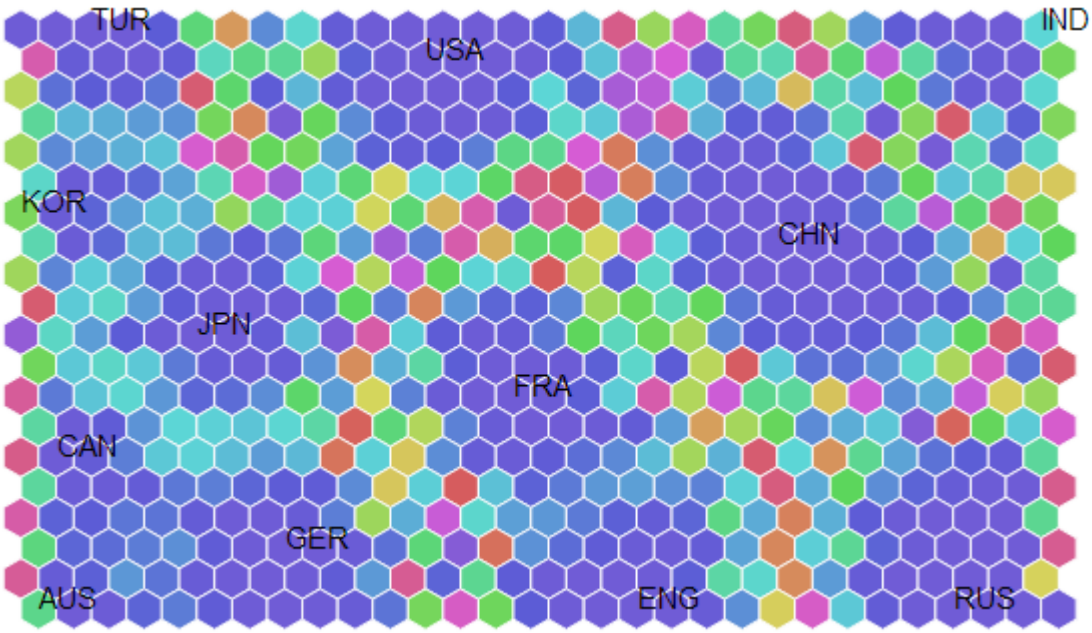


Figure 17 hard power among 12 countries

Comment

As we can see from the map, clusters are quite elegant. For Canada, Australia and Germany there is no much difference as well as France and England. Where Japan, Turkey and Korea get some difference. The most obvious differnce show on USA, China, Russian and India from the rendering of U-matrix.

5.43 Comprehensive power

To see the similarity comprehensive power among these 12 countries, I combine all these 14 dimensions together to render the map

Data

From table4 and table 5

12 countries, 14 dimensions size12 *14

Map

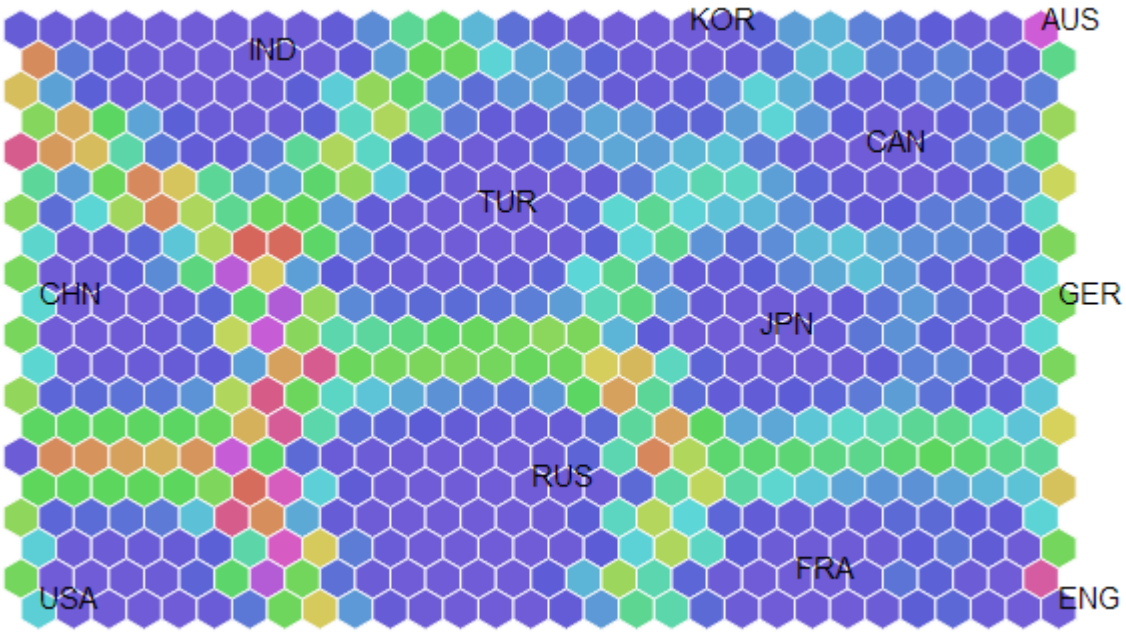


Figure 18 comprehensive power among 12 countries

Comment

The result of comprehensive power is actually in my predication based on the results from previous 2 instances. There are 4 countries shows mostly different that are USA, China, India and Russia. Where they are outstanding in the soft power and hard power visualisation as well.

5.44 Summary

This case study is inspired from iris data set for its component study. Here I use soft power and hard power as components for studying the similaity of comprehensive power among 12 countries. The result of comprehensive power is in my expect which I make predication based on soft power and hard power. From the result I also make an assumption that hard power is more important factor to difference of power among countries than soft power. My assumption comes from analysis of hard powermap and soft power map as we can see the difference of hard power are more obvious than soft power. However, this assumption is not proved in my paper as the input vector for both soft power and hard power are way to small no matter the quantity of both input and dimensons. Another reason is as nodes are randomly initialized so the result of rendering will be different and I have no evidence to say the result is always ture even though it will be similar every time. Hence I cannot find sufficient evidence to say my assumption is correct.

6 Conclusion

My project is compledby 3 stages, firstly use CoffeeScript to build class SOM and calss Node, in this stage I create the node by defining its positon and weight and implement all the self-organing map alorithms initialization, competition, cooeration and adaptation in SOM calss. Second step is to use Yeoman to compile my CoffeeScript file to JavaScript. In last step I use d3js.org library to render my map and show self-organizing map by trainig the node using input data that color, animal iris and country data.

My project has already shown that Object-Oriented approach to implementation can also work properly for self-organizing map as well as its advantages are quite obvious that we can use SOM class(SOM.js) and Node class (Node.js) to implement self-oragizing map algorithm and renderit as different ways. However, to make the program run faster non-object-oriented implemtation is a better choice. It is hard to say which approach is better for implelementing self-organinzing map. The choice depends on cases we actually meet.

6.1 Futher work

As mentioned prevously, 4 improvements can still be made for my project. Firstly, I may need more references to compare non-object oriented approach and non-object-oriented approch. Secondly, for iris data I would try other renderring way to see its component effect and see the effect of component plane 2 sepal width and study what effect it actually make for the iris data. Thirdly, I will continuemy power of countries study to make the more countries(increase number of input) and more attributes(enhance quantity of dimensions) to see prove my assumption. Lastly, I might try to enhance my algotihtm which can improve the speed of my program and makes good running time as well as save memory to make sure it can work effieciently.

7 Acknowledgements

I would like to thank my supervisor Prof Masahiro Takatsuka for being my supervisor and allowing me to take this project and for his continued spport and encouragement. Also, I would like to thank Dr Florence Wang, Dr Jina Chung, Dr John Stavrakakis, Benjamin Brent for their kindness help.

8 Reference

[1] Kohonen, T. (1998). The self-organizing map. Neurocomputing, 21(1-3), pp.1-6.

[2] Kohonen, Teuvo. "The self-organizing map." Proceedings of the IEEE 78.9 (1990): 1464-1480

[3] Next-Generation User-Centered Information Management Jing Li

[4] Self Organizing Maps: Fundamentals Introduction to Neural Networks : Lecture 16 John A. Bullinaria, 2004
[5] Pollock, J. (2010). JavaScript. New York: McGraw Hill.

[5] Bostock, M. (2014). D3.js - Data-Driven Documents. [online] D3js.org. Available at: http://d3js.org [Accessed 23 Nov. 2014].

[6] CoffeeScript.org, (2014). CoffeeScript. [online] Available at: http://JavaScript.org [Accessed 23 Nov. 2014].

[7] Hudson, E. (2013). Smashing CoffeeScript. Chichester, West Sussex, U.K.: Wiley.

[8] MacCaw, A. and Ashkenas, J. (2012). The little book on CoffeeScript. Beijing: O'Reilly Media.

[9] Bremer, N. and profile, V. (2013). Self Organizing Maps - Creating hexagonal Heatmaps with D3.js | Data Sparkle. [online] Nbremer.blogspot.com.au. Available at: http://nbremer.blogspot.com.au/2013/07/self-organizing-maps-creating-hexagonal.html [Accessed23 Nov. 2014].

[10] Bl.ocks.org, (2013). Self Organizing Map - Hexagonal Heatmap - Lines. [online] Available at: http://bl.ocks.org/nbremer/6052681 [Accessed 23 Nov. 2014].

[11] Bl.ocks.org, (2013). SOM Animation with d3.js. [online] Available at: http://bl.ocks.org/e-/5244131 [Accessed 23 Nov. 2014].

[12] D3.js: Colors! (D3.js Color Tutorial) http://old.schneidy.com/Tutorials/ColorTutorial.html

[13] Using Self-Organizing Feature Maps (Kohonen Maps) in MetaTrader 5

Mql5.com,. 2014. 'Using Self-Organizing Feature Maps (Kohonen Maps) In Metatrader 5'. Accessed November 23 2014. https://www.mql5.com/en/articles/283.

[14] d3/d3-plugins

GitHub,. 2014. 'D3/D3-Plugins'. Accessed November 23 2014. https://github.com/d3/d3-plugins/tree/master/hexbin.

[15] rainbow color table

Ncl.ucar.edu,. 2014. 'Rainbow Color Table'. Accessed November 23 2014. https://www.ncl.ucar.edu/Document/Graphics/ColorT

[16] Colorpicker.com,. 2014. 'Colorpicker.Com - Quick Online Color Picker Tool'. Accessed November 23 2014. http://colorpicker.com/.

[17] Li, J. (0). Information Visualization with Self-Organizing Maps.

[18] jquery.org, jQuery. 2014. 'Jquery'. Jquery.Com. Accessed November 23 2014. http://jquery.com/.

[19] Pedrycz, W., Succi, G., Musílek, P., & Bai, X. (2001). Using self-organizing maps to analyze object-oriented software measures. Journal of Systems and Software. doi:10.1016/S0164-1212(01)00049-8

[20] Yeoman.io,. 2014. 'The Web's Scaffolding Tool For Modern Webapps | Yeoman'. Accessed November 23 2014. http://yeoman.io/.

[21] Yeoman.io,. 2014. 'Let's Scaffold A Web App With Yeoman | Yeoman'. Accessed November 23 2014. http://yeoman.io/codelab.html.

[21] Dashingd3js.com,. 2014. 'SVG Basic Shapes And D3.Js | Dashingd3js.Com'. Accessed November 23 2014. https://www.dashingd3js.com/svg-basic-shapes-and-d3js.

[22] Dashingd3js.com,. 2014. 'SVG Text Element | Dashingd3js.Com'. Accessed November 23 2014. https://www.dashingd3js.com/svg-text-element.

[23] Vesanto, J., & Alhoniemi, E. (2000). Clustering of the self-organizing map. IEEE Transactions on Neural Networks. doi:10.1109/72.846731

[24] CoffeeScriptcookbook.com,. 2014. 'CoffeeScript Cookbook » Home'. Accessed November 23 2014. http://CoffeeScriptcookbook.com/ .

[25] Craig Sketchley, November 26, 2013 A Self-Organising Map Application built with Dart

[26] Data.worldbank.org,. 2014. 'Data | The World Bank'. Accessed November 23 2014. http://data.worldbank.org/.

[27] Globalfirepower.com,. 2014. 'Global Firepower Military Powers Ranked For 2014'. Accessed November 23 2014. http://www.globalfirepower.com/

9 Appendix

9.1 Class SOM (CoffeeScript)

```
class SOM

    constructor:(@inputDimension,@iterations) ->

        @height= 30

        @width= 20

        @radius= 20*0.8

        @timeConstant = @iterations/Math.log(@radius)

        @nodes =  []

        @learnRate= 0.05

        for i in [0..(@height*@width-1)] by 1

            @nodes[i] = new Node(@inputDimension)
```

```
                @nodes[i].y = (i / @width | 0)
                @nodes[i].x = i % @width

        train : (i, w) ->
                radiusDecay = @radius*Math.exp(-1*i/@timeConstant)
                learnDecay = @learnRate*Math.exp(-1*i/@timeConstant)
                BMU = @best_match(w)
                for a in [0..(@height*@width-1)] by 1
                        d = @geo_Distance(@nodes[a], @nodes[BMU])
                        if (d < radiusDecay)
                                influence = Math.exp(((-1) * Math.pow(d,2)) / (2 * radiusDecay * i))
                                for k in [0..(@inputDimension-1)] by 1
                                        @nodes[a].weight[k] += influence*learnDecay*(w[k] -
@nodes[a].weight[k])
                return

        geo_Distance : (node1 ,node2) -> Math.sqrt(Math.pow((node1.x - node2.x),2) +
Math.pow((node1.y - node2.y),2))

        weight_distance:(x ,y) ->
                tmp = 0
                for i in [0..(x.length-1)] by 1
                        tmp += Math.pow((x[i] - y[i]),2)
                tmp = Math.sqrt(tmp)
                return tmp

        best_match : (w) ->
                minDist = @inputDimension
                tmp = 0
                minIndex = 0
                for i in [0..(@height*@width-1)] by 1
                        tmp = @weight_distance(@nodes[i].weight, w)
```

```
                if tmp < minDist
                        minDist = tmp
                        minIndex = i
        return minIndex
```

9.2 class node

```
Class Node
    Constructor: (d) ->
        @weight = []
        @x=null
        @y=null
        for i in [0..(d-1)]
            @weight[i] =Math.random()
```

9.3 Iris data

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
```

```
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.2,3.4,1.4,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
4.8,3.1,1.6,0.2,Iris-setosa
5.4,3.4,1.5,0.4,Iris-setosa
5.2,4.1,1.5,0.1,Iris-setosa
5.5,4.2,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.0,3.2,1.2,0.2,Iris-setosa
5.5,3.5,1.3,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
4.4,3.0,1.3,0.2,Iris-setosa
5.1,3.4,1.5,0.2,Iris-setosa
5.0,3.5,1.3,0.3,Iris-setosa
4.5,2.3,1.3,0.3,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
5.0,3.5,1.6,0.6,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
6.3,3.3,4.7,1.6,Iris-versicolor
4.9,2.4,3.3,1.0,Iris-versicolor
6.6,2.9,4.6,1.3,Iris-versicolor
5.2,2.7,3.9,1.4,Iris-versicolor
5.0,2.0,3.5,1.0,Iris-versicolor
5.9,3.0,4.2,1.5,Iris-versicolor
6.0,2.2,4.0,1.0,Iris-versicolor
6.1,2.9,4.7,1.4,Iris-versicolor
5.6,2.9,3.6,1.3,Iris-versicolor
6.7,3.1,4.4,1.4,Iris-versicolor
5.6,3.0,4.5,1.5,Iris-versicolor
5.8,2.7,4.1,1.0,Iris-versicolor
6.2,2.2,4.5,1.5,Iris-versicolor
5.6,2.5,3.9,1.1,Iris-versicolor
5.9,3.2,4.8,1.8,Iris-versicolor
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
6.1,2.8,4.7,1.2,Iris-versicolor
6.4,2.9,4.3,1.3,Iris-versicolor
6.6,3.0,4.4,1.4,Iris-versicolor
6.8,2.8,4.8,1.4,Iris-versicolor
```

6.7,3.0,5.0,1.7,Iris-versicolor
6.0,2.9,4.5,1.5,Iris-versicolor
5.7,2.6,3.5,1.0,Iris-versicolor
5.5,2.4,3.8,1.1,Iris-versicolor
5.5,2.4,3.7,1.0,Iris-versicolor
5.8,2.7,3.9,1.2,Iris-versicolor
6.0,2.7,5.1,1.6,Iris-versicolor
5.4,3.0,4.5,1.5,Iris-versicolor
6.0,3.4,4.5,1.6,Iris-versicolor
6.7,3.1,4.7,1.5,Iris-versicolor
6.3,2.3,4.4,1.3,Iris-versicolor
5.6,3.0,4.1,1.3,Iris-versicolor
5.5,2.5,4.0,1.3,Iris-versicolor
5.5,2.6,4.4,1.2,Iris-versicolor
6.1,3.0,4.6,1.4,Iris-versicolor
5.8,2.6,4.0,1.2,Iris-versicolor
5.0,2.3,3.3,1.0,Iris-versicolor
5.6,2.7,4.2,1.3,Iris-versicolor
5.7,3.0,4.2,1.2,Iris-versicolor
5.7,2.9,4.2,1.3,Iris-versicolor
6.2,2.9,4.3,1.3,Iris-versicolor
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
7.6,3.0,6.6,2.1,Iris-virginica
4.9,2.5,4.5,1.7,Iris-virginica
7.3,2.9,6.3,1.8,Iris-virginica
6.7,2.5,5.8,1.8,Iris-virginica
7.2,3.6,6.1,2.5,Iris-virginica
6.5,3.2,5.1,2.0,Iris-virginica
6.4,2.7,5.3,1.9,Iris-virginica
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
6.4,3.2,5.3,2.3,Iris-virginica
6.5,3.0,5.5,1.8,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
6.0,2.2,5.0,1.5,Iris-virginica
6.9,3.2,5.7,2.3,Iris-virginica
5.6,2.8,4.9,2.0,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.7,3.3,5.7,2.1,Iris-virginica
7.2,3.2,6.0,1.8,Iris-virginica
6.2,2.8,4.8,1.8,Iris-virginica
6.1,3.0,4.9,1.8,Iris-virginica
6.4,2.8,5.6,2.1,Iris-virginica
7.2,3.0,5.8,1.6,Iris-virginica
7.4,2.8,6.1,1.9,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
6.4,2.8,5.6,2.2,Iris-virginica
6.3,2.8,5.1,1.5,Iris-virginica
6.1,2.6,5.6,1.4,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
6.3,3.4,5.6,2.4,Iris-virginica
6.4,3.1,5.5,1.8,Iris-virginica

```
6.0,3.0,4.8,1.8,Iris-virginica
6.9,3.1,5.4,2.1,Iris-virginica
6.7,3.1,5.6,2.4,Iris-virginica
6.9,3.1,5.1,2.3,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
6.8,3.2,5.9,2.3,Iris-virginica
6.7,3.3,5.7,2.5,Iris-virginica
6.7,3.0,5.2,2.3,Iris-virginica
6.3,2.5,5.0,1.9,Iris-virginica
6.5,3.0,5.2,2.0,Iris-virginica
6.2,3.4,5.4,2.3,Iris-virginica
5.9,3.0,5.1,1.8,Iris-virginica
```