ECE5801: Self-Organizing Feature Map, Project 4

In this project, you will learn about and experiment with the Self-Organizing Feature Map (SOM) by writing the code for the algorithm and testing your SOM with random data distributed over a certain shape. Please read the handout for understanding of the algorithm. Below describes implementation details.

Part 1: Square Distribution

First, you will need to create a 10x10 two-dimensional array of neurons each of which has two weights (no bias terms). You may use double or float as your weight matrix:

```
double [,,] w = new double [9, 9, 1] ;
```

where w[i,j,k] denotes i-th row, j-th column, and k-th weight. The weights must be initialized with small random numbers.

The inputs are determined by the random number generator. Since the Random class can generate a number between 0 and 1, simply defining the rnd1 variable:

```
double[] x=new double[1]; // x[] is the instataneous inputs to the network
Random rnd1 = new Random();
```

and then repeatedly creating inputs using

```
x[0] = rnd1.NextDouble();
x[1] = rnd1.NextDouble();
```

will generate uniformly distributed input vectors over a square region.

Next, the weight update procedure works as follows:

Step 1: Pick a random vector $\mathbf{x}(n)$ where n denotes the present iteration number.

Step 2: Determine the winner node by finding the node with the minimum distance, i.e., min $\| \mathbf{x}(n) - \mathbf{w}_i(n) \|$ of all i.

Step 3: Determine the neighbor nodes of the winner node. Use a square shape neighbor function. We will call the nodes inside the neighbor hood as Nc.

Step 4: Update the weights of nodes insider the neighbor Nc using

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta(n)\,(\mathbf{x}(n) - \mathbf{w}_i(n)) \qquad \text{for } i \in Nc$$

Step5: Plot the network and input data distribution.

The network plot is simply connecting lines between the neighboring nodes in x and y directions, i.e.

Point(w(0,0,0), w(0,0,1)) to Point (w(0,1,0), w(0,1,1))
Point(w(0,0,0), w(0,0,1)) to Point (w(1,0,0), w(0,0,1))
…

For input data distribution plot, simply plot a dot in the square area.

Step6: Go to Step1 until the maximum number of iterations reaches.

The iteration is terminated based on a predetermined maximum number of iterations. I used 5,000 as the maximum number of iterations for the example shown in class, but I recommend you to try a bigger number such as 10,000.

The neighbor size must shrink with the progression of iterations. A simple method can be devised using a manual schedule of the neighbor size and iteration rate as a function of iteration number. Let t be the iteration index, then you could write:

```
   if (t >=0 && t<50)
   {
        Nsize = 5;
        eta = 0.25;
}
else if ( t >= 50 &&  t < 200)
{
        Nsize = 4
        eta = 0.2
}

.
.
.

}
```

In the above example, Nsize denotes the radius of the square neighbor from the winner node and eta denotes the learning rate. To be more specific, Nsize is illustrated using the following examples of neighbor patters:

```
                          * * * * *
               * * *      * * * * *
   *           * * *      * * * * *
               * * *      * * * * *
                          * * * * *

Nsize=0        Nsize=1    Nsize=2
```

The reason of defining Nsize this way is that you can now use simple for-loops for defining the nodes inside the neighbors, i.e.,

```
for (i=max(0,center_i - Nsize); i<= min(9, center_i + Nsize); i++)
{
  for (j=max(0,center_j – Nzise); j<=min(9, center_j + Nzise): j++)
  {
     ……..
  }
}
```

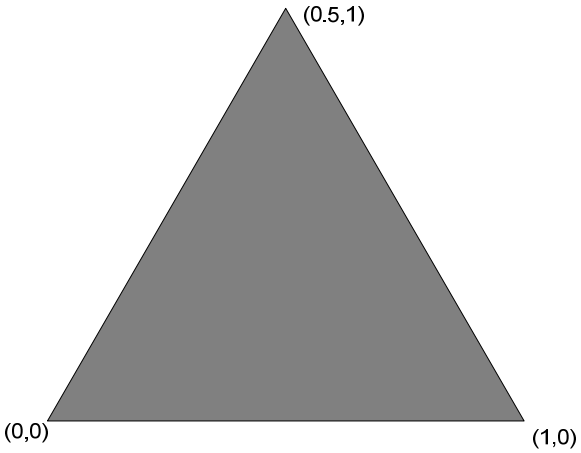
where the winner node index is (center_i, center_j) and min() and max() functions that simply return minimum or maximum of the two values your are comparing.

The manually setting the neighbor size and learning rate is referred to as SOM parameter scheduling. You will have a better understanding of the behavior of the SOM if you try various training parameter schedules.

First test is for the uniform distribution within the square area between (0,0) and (1,1).

Part 2: Triangular Distribution

For the second test, experiment with a triangular distribution of the input data. The input data points are randomly drawn out of the shape shown below.

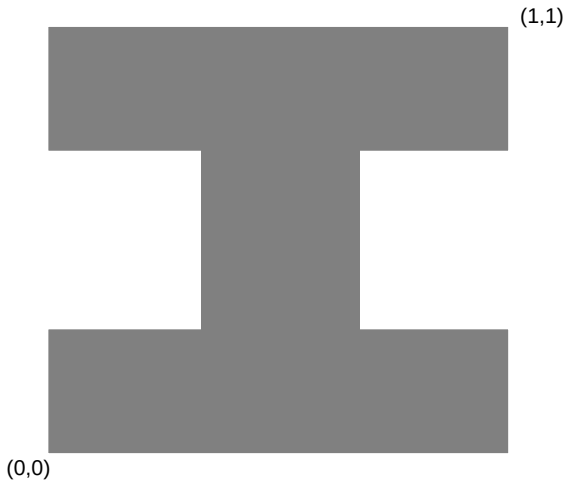


Part 3: Hour glass distribution

For the last test, experiment with an hour glass shape distribution of the input data. For this experiment, increase the weight vectors to 20x20, i.e.

double [,,] w = new double [20, 20, 1]

The input data points are evenly distributed over the region given below.



Report:

1. Objective of this project.
2. What is SOM and how does it work?
4. What are the implementation issues?
5. Provide the final plots of each case, along with your parameter schedule.
6. Point out possible applications.
7. Point out the drawbacks of SOM
8. Your conclusion
9. Attach your program listing