

Kohonen Self-Organized Maps

K. Ming Leung

Abstract: The Kohonen Self-Organized Maps for clustering a set of continuous input vectors is discussed. This NN is trained by unsupervised learning.

Directory

- [Table of Contents](#)
- [Begin Article](#)

Table of Contents

1. Topology of neurons within a layer
2. Kohonen Self-Organizing Maps
 - 2.1. Example: SOM
3. Another example
4. Remarks

1. Topology of neurons within a layer

In the nets we have studied so far, we have ignored the geometrical arrangements of the output neurons. Each neuron in a given layer has identical behavior in that each one receives the signals from the input layer and reacts individually in a similar way. The behavior of one neuron is independent of the behaviors of other neurons in its neighborhood within the layer. We are now going to take into consideration that physical arrangement (the topology) of these nodes. Nodes that are "close" together are going to interact differently than nodes that are "far" apart. In the brain, neurons tend to cluster in groups. The interactions of the neurons within the group are much greater than those outside the group. The Kohonen self-organizing maps are neural networks that try to mimic this feature in a simple way.

Kohonen self-organizing maps (SOM) are also known as the topology-preserving maps, since a topological structure of the output neurons are assumed, and this structure is maintained during the training process. Each output neuron is referred to as a cluster unit. Typically

two types of topologies are considered for the output layer: linear and two-dimensional arrays. The following are examples of such geometries.

An example of a linear output layer:

```
* * * 2 1 0 1 2 * *
```

An example of a two-dimensional array of output neuron arranged on a square lattice:

```
* * * * * * * * * *
* * * 2 2 2 2 2 * *
* * * 2 1 1 1 2 * *
* * * 2 1 0 1 2 * *
* * * 2 1 1 1 2 * *
* * * 2 2 2 2 2 * *
* * * * * * * * * *
```

and on a hexagonal lattice:

```
*   *   *   *   *
  *  2   2   2   *
*   2   1   1   2
  2   1   0   1   2
*   2   1   1   2
  *   2   2   2   *
*   *   *   *   *
```

In SOM, learning occurs unsupervised (that is why it is called self-organizing), that means that targeted outputs are not known or given. At each step of the training process, there are competitions among the output neurons (cluster units) with a resulting single winner, denoted by "0" in the above illustrations. The weights connected to the winning neuron, as well as the weights connected to the winning neuron's first and second neighborhoods are updated in a similar fashion. Neurons in its first and second neighborhoods are labeled

here by "1" and "2" respectively. All the other neurons are labeled by "*".

If a winning neuron is located close to the edge of the grid, then some neighborhoods may have fewer neurons. Neighborhoods do not "wrap around" from one side of the grid to the other; missing neurons are simply ignored in the updating process.

2. Kohonen Self-Organizing Maps

Kohonen SOM is designed to group a set of Q continuous-valued vectors $\mathbf{s}^{(q)} = \begin{bmatrix} s_1^{(q)} & s_2^{(q)} & \dots & s_N^{(q)} \end{bmatrix}$, $q = 1, \dots, Q$, into $M (< Q)$ clusters. The SOM thus consists of an input layer having N neurons, and an output layer having M neurons (cluster units) arranged in some predetermined fashion. Each neuron in the input layer is connected to a neuron in the output layer. Thus output neuron (cluster units) j is connected to each of the input neuron through weights w_{ij} , $i = 1, \dots, N$, which is referred to as the j -th weight vector. In vector or matrix notation, we denote the j -th weight vector by $\mathbf{W}_{.j} = \begin{bmatrix} w_{1j} & w_{2j} & \dots & w_{Nj} \end{bmatrix}$. It is given by the j -th column of

the weight matrix. There is a total of M such weight vectors, one for each cluster units. Each one of these weight vectors serves as an exemplar of the input patterns associated with that cluster. Unless some prior information is known about the clusters, these weight vectors are typically initialized to some random values.

During the training process, each training vector is cyclically or randomly selected and presented to the network. The cluster unit j' whose weight vector $\mathbf{W}_{.j'}$ matches the input pattern \mathbf{x} the most closely is chosen as the winner. Here two vectors are considered closest if the square of the Euclidean distance between them, $\|\mathbf{x} - \mathbf{W}_{.j'}\|^2$ is the smallest. The winning unit and its neighboring units (those located in its first and second neighborhoods) then update their weight vectors according to the Kohonen rule. This process is continued until the weight vectors change by less than a preset amount.

Unless the dot-product is used to measure the closeness of two vectors, the input vector does not get multiplied with the weight vectors, as we have been doing so far. Instead it is the square of the Euclidean

distance between the input vector and each of the weight vectors

$$d_j = \|\mathbf{x} - \mathbf{W}_{.j}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{W}_{.j}\|^2 - 2\mathbf{x} \cdot \mathbf{W}_{.j}$$

that is computed. Therefore for a given input vector \mathbf{x} , the weight vector j' whose $d_{j'}$ is the smallest is always the one which has the largest dot-product with \mathbf{x} only if the variation in the magnitudes of the weight vectors can be ignored. This happens if the weight vectors are constrained to have the same magnitude.

The SOM algorithm is:

- 1 Initialize M weight vectors. Set topological neighborhood parameters and learning rate, $\alpha (< 1)$.
- 2 For step $k = 1, 2, \dots$, do steps a - d by cycling through training set until weight vectors converge
 - a Set input vector $\mathbf{x} = \mathbf{s}^{(q)}$, one of the training vectors.
 - b Compute for each cluster unit $j = 1, \dots, M$ the Euclidean distance

$$d_j = \sum_{i \neq 1}^N (x_i - w_{ij}(k))^2.$$

- c Find the index j' such that $d_{j'}$ is a minimum.
- d For all cluster units j within the specified neighborhoods of j' , update the weight vectors

$$w_{ij}(k+1) = w_{ij}(k) + \alpha [x_i - w_{ij}(k)], \quad i = 1, \dots, N.$$

- e May reduce the learning rate.
- f May reduce the radii that define the topological neighborhoods.

The above updating rule moves the weight vectors for the winning neuron and those in its neighborhood towards the input vector. The amount of change is proportional to α . In the extreme limit of $\alpha = 1$, all these weight vectors are set to the input vector.

During training, the learning rate can be decreased linearly, that is

$$\alpha(k) = \frac{\alpha(1)}{k}$$

where $k = 1, 2, \dots$ is the iteration counter. Geometric decrease of α :

$$\alpha(k+1) = f\alpha(k)$$

where $0 < f < 1$, also works. In general convergence may required many iterations through the training set.

2.1. Example: SOM

We now consider an example where 4 input vectors

$$\mathbf{s}^{(1)} = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{s}^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{s}^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{s}^{(4)} = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$$

are to be grouped into 2 clusters. It is clear that $N = 4$, $M = 2$ and $Q = 4$.

We will be using matrix notation for convenience. However, the input vectors are row vectors but the weight vectors are column vectors. To make them compatible we will transpose the weight vectors to form row vectors.

Suppose the initial learning rate is $\alpha(1) = 0.6$, and we use a geometric schedule with $f = 0.5$ for decreasing α . For simplicity we shrink the topological radii to zero so that only the winning neuron has its weight vector updated. We assume that the initial weight matrix is given by

$$\mathbf{W} = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}.$$

During training, the input vectors are presented to the NN one at

a time in the given order. For $\mathbf{x} = \mathbf{s}^{(1)}$, we have

$$\begin{aligned} d_1 &= \left\| \begin{bmatrix} 0.2 & 0.6 & 0.5 & 0.9 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} -0.8 & -0.4 & 0.5 & 0.9 \end{bmatrix} \right\|^2 = 1.86. \end{aligned}$$

and

$$\begin{aligned} d_2 &= \left\| \begin{bmatrix} 0.8 & 0.4 & 0.7 & 0.3 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} -0.2 & -0.6 & 0.7 & 0.3 \end{bmatrix} \right\|^2 = 0.98. \end{aligned}$$

Since d_2 is the smallest, the winning neuron is $j' = 2$. So we update the second weight vector

$$\begin{aligned} \mathbf{W}_{.2} &= \begin{bmatrix} 0.8 & 0.4 & 0.7 & 0.3 \end{bmatrix} + \\ &\quad \alpha(1) \left(\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0.8 & 0.4 & 0.7 & 0.3 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.92 & 0.76 & 0.28 & 0.12 \end{bmatrix}. \end{aligned}$$

The weight matrix is now given by

$$\mathbf{W} = \begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}.$$

Then we present $\mathbf{x} = \mathbf{s}^{(2)}$. we find

$$\begin{aligned} d_1 &= \left\| \begin{bmatrix} 0.2 & 0.6 & 0.5 & 0.9 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} 0.2 & 0.6 & 0.5 & -0.1 \end{bmatrix} \right\|^2 = 0.66. \end{aligned}$$

and

$$\begin{aligned} d_2 &= \left\| \begin{bmatrix} 0.92 & 0.76 & 0.28 & 0.12 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} 0.92 & 0.76 & 0.28 & -0.88 \end{bmatrix} \right\|^2 = 2.2768. \end{aligned}$$

Since d_1 is the smallest, the winning neuron is $j' = 1$. So we update the first weight vector

$$\begin{aligned} \mathbf{W}_{.1} &= \begin{bmatrix} 0.2 & 0.6 & 0.5 & 0.9 \end{bmatrix} + \\ &\quad \alpha(1) \left(\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.2 & 0.6 & 0.5 & 0.9 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.92 & 0.76 & 0.28 & 0.12 \end{bmatrix}. \end{aligned}$$

The weight matrix is then given by

$$\mathbf{W} = \begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}.$$

We continue this process for the remaining two training vectors in the data set to obtain the weight matrix

$$\mathbf{W} = \begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.048 \end{bmatrix}$$

after the first epoch. Next we reduce the learning rate by a multiplicative factor $f = 0.5$ and repeat another round through the training set.

After 100 iterations, the learning rate decreases to 0.006, and the weight matrix becomes

$$\mathbf{W} = \begin{bmatrix} 1.61e - 4 & 0.99984 \\ 2.0e - 16 & 0.49376 \\ 0.50616 & 5.65e - 4 \\ 0.99992 & 2.42e - 4 \end{bmatrix}$$

and appears to be approaching

$$\mathbf{W} = \begin{bmatrix} 0 & 1 \\ 0 & 0.5 \\ 0.5 & 0 \\ 1 & 0 \end{bmatrix}.$$

Notice that the weight vector for cluster 1, $\mathbf{W}_{.1}$, is exactly at the half way point between $\mathbf{s}^{(2)}$ and $\mathbf{s}^{(4)}$:

$$\frac{1}{2} \left(\mathbf{s}^{(2)} + \mathbf{s}^{(4)} \right) = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 1 \end{bmatrix} = \mathbf{W}_{.1},$$

and the weight vector for cluster 2, \mathbf{W}_{*2} , is exactly at the half way point between $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(3)}$:

$$\frac{1}{2} \left(\mathbf{s}^{(1)} + \mathbf{s}^{(3)} \right) = \begin{bmatrix} 1 \\ 0.5 \\ 0 \\ 0 \end{bmatrix} = \mathbf{W}_{*2}.$$

These are the expected results since $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(3)}$ are closest together (the square of the Euclidean distance between them is 1, the smallest) and together they form cluster 2. Similarly, $\mathbf{s}^{(2)}$ and $\mathbf{s}^{(4)}$ are closest together (the square of the Euclidean distance between them is 1, the smallest) and together they form cluster 1.

$\ \mathbf{s}^{(i)} - \mathbf{s}^{(j)}\ ^2$	$\mathbf{s}^{(1)}$	$\mathbf{s}^{(2)}$	$\mathbf{s}^{(3)}$	$\mathbf{s}^{(4)}$
$\mathbf{s}^{(1)}$	0	3	1	4
$\mathbf{s}^{(2)}$	3	0	2	1
$\mathbf{s}^{(3)}$	1	2	0	3
$\mathbf{s}^{(4)}$	4	1	3	0

3. Another example

We consider grouping 6 training vectors into 3 clusters. The training vectors all have unit magnitude and so they lie on the unit circle. Although we are not supposed to know at the outset, these vectors actually can be separated into 3 pairs. The first pairs are located at

$\pm 11^\circ$ from 90° , the second pair at $\pm 11^\circ$ from 0° , and the third pair at $\pm 11^\circ$ from -135° . Thus the training set is

$$\left\{ \mathbf{s}^{(q)} = \begin{bmatrix} \cos(\theta^{(q)}) & \sin(\theta^{(q)}) \end{bmatrix}, \quad q = 1, 2, \dots, 6 \right\}$$

where $\theta^{(q)}$ specifies the angles of the training vectors in radians. Thus we have $N = 2$, $M = 3$, and $Q = 6$.

The weight vectors for the 3 clusters are initially taken to be unit vectors at -45° , 45° , and 180° , respectively. Therefore the initial weight matrix is

$$\mathbf{W} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -1 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

Notice that these initial exemplar vectors are not pointing at any of the clusters. This choice of \mathbf{W} was chosen intentionally to provide a stringent test of the ability of our NN to cluster the given input vectors. In general, the initial exemplar vectors for each of the clusters should be chosen randomly.

In reality, of course we do not know how to separate the input

vectors into separate clusters. That is we have no idea which vector belong to which of the cluster, nor do we know the number of vectors in each of the clusters. All that will be determined by the NN at the end. The only thing we know at the beginning is that we want to separate the input vectors into 3 clusters.

Notice that although the training vectors and the weight vectors are all unit vectors, Kohonen's updating rule does not preserve the magnitude of the weight vectors. Of course it can be modified to do just that if we want to. In that case, for a given input vector, the winning neuron can be chosen to be the cluster unit whose weight vector has the largest dot-product with the input vector.

For the present problem, it is obvious that the correct final orientations for the weight vectors are at 90° , 0° , and -135° (the ordering is totally irrelevant). We find that a constant learning rate of about 0.05 works very well for the present problem.

4. Remarks

1. During the clustering process, it may be a good idea to gradually reduce the topological neighborhood of each of the output neurons.
2. The weight vector forming the prototype of a cluster may invade the territory of a nearby weight vector and as a result upset the current clustering scheme.
3. A cluster unit's initial weight vector may be located so far from any input vector that it may never be chosen as the winning neuron. Such neurons are dead and should be pruned.

References

- [1] See Chapter 4 in Laurene Fausett, "Fundamentals of Neural Networks - Architectures, Algorithms, and Applications", Prentice Hall, 1994.