1 The Self-Organizing Map (SOM)

Teuvo Kohonen

1.1 Introduction

The SOM is a new, effective software tool for the visualization of high-dimensional data. It implements an orderly mapping of a high-dimensional distribution onto a regular low-dimensional grid. Thereby it is able to convert complex, nonlinear statistical relationships between high-dimensional data items into simple geometric relationships on a low-dimensional display. As it compresses information while preserving the most important topological and metric relationships of the primary data items on the display, it may also be thought to produce some kind of abstractions. These two aspects, visualization and abstraction, can be utilized in a number of ways in complex tasks such as process analysis, machine perception, control, and communication.

The SOM usually consists of a two-dimensional regular grid of nodes. A *model* of some observation is associated with each node (cf. Fig. 1).

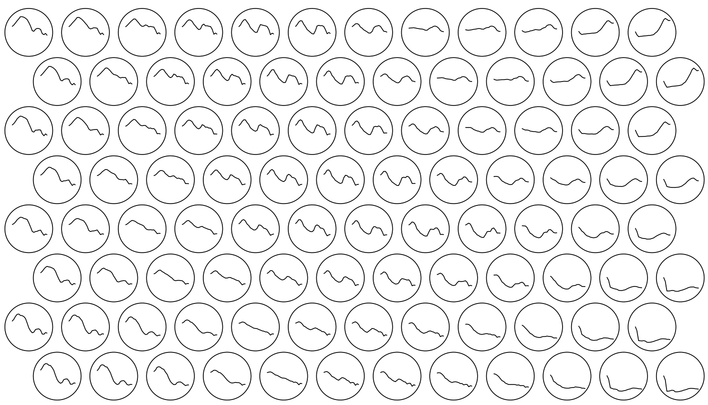

Figure 1: In this exemplary application, each processing element in the hexagonal grid holds a model of a short-time spectrum of natural speech (Finnish). Notice that neighboring models are mutually similar.

The SOM algorithm computes the models so that they optimally describe the domain of (discrete or continuously distributed) observations.

The models are automatically organized into a meaningful two-dimensional order in which similar models are closer to each other in the grid than the more dissimilar ones. In this sense the SOM is a similarity graph, and a clustering diagram, too. Its computation is a nonparametric, recursive regression process.

1.2 The incremental-learning SOM algorithm

Regression of an ordered set of model vectors $\mathbf{m}_i \in \Re^n$ into the space of observation vectors $\mathbf{x} \in \Re^n$ is often made by the following process:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(\mathbf{x}),i}(\mathbf{x}(t) - \mathbf{m}_i(t)) \ , \tag{1}$$

where t is the index of the regression step, and the regression is performed recursively for each presentation of a sample of $\mathbf{x}$, denoted $\mathbf{x}(t)$. The scalar multiplier $h_{c(\mathbf{x}),i}$ is called the *neighborhood function*, and it is like a smoothing or blurring kernel over the grid. Its first subscript $c = c(\mathbf{x})$ is defined by the condition

$$\forall i, \ \ \|\mathbf{x}(t) - \mathbf{m}_c(t)\| \leq \|\mathbf{x}(t) - \mathbf{m}_i(t)\| \ , \tag{2}$$

that is, $\mathbf{m}_c(t)$ is the model (called the *"winner"*) that matches best with $\mathbf{x}(t)$. The comparison metric is usually selected as Euclidean; for other metrics, the forms of (1) and (2) will change accordingly. If the samples $\mathbf{x}(t)$ are stochastic and have a continuous density function, the probability for having multiple minima in (2) is zero. With discrete-valued variables, multiple minima may occur; in such cases one of them should be selected at random for the winner.

The neighborhood function is often taken to be the Gaussian

$$h_{c(\mathbf{x}),i} = \alpha(t) \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma^2(t)}\right) \ , \tag{3}$$

where $0 < \alpha(t) < 1$ is the learning-rate factor, which decreases monotonically with the regression steps, $\mathbf{r}_i \in \Re^2$ and $\mathbf{r}_c \in \Re^2$ are the vectorial locations on the display grid, and $\sigma(t)$ corresponds to the width of the neighborhood function, which is also decreasing monotonically with the regression steps.

A simpler definition of $h_{c(\mathbf{x}),i}$ is the following: $h_{c(\mathbf{x}),i} = \alpha(t)$ if $\|\mathbf{r}_i - \mathbf{r}_c\|$ is smaller than a given radius from node c (whereupon this radius is a monotonically decreasing function of the regression steps, too), but otherwise $h_{c(\mathbf{x}),i} = 0$. In this case we shall call the set of nodes that lie within the given radius the *neighborhood set* N_c.

Due to the many stages in the development of the SOM method and its variations, there is often useless historical ballast in the computations.

For instance, an old ineffective principle is random initialization of the model vectors $\mathbf{m}_i$. Random initialization was originally used to show that there exists a strong self-organizing tendency in the SOM, so that the order can even emerge when starting from a completely unordered state, but this need not be demonstrated every time. On the contrary, if the initial values for the model vectors are selected as a regular array of vectorial values that lie on the subspace spanned by the eigenvectors corresponding to the two largest principal components of input data, computation of the SOM can be made orders of magnitude faster, since (i) the SOM is then already approximately organized in the beginning, (ii) one can start with a narrower neighborhood function and smaller learning-rate factor.

Many computational aspects like this and the selection of proper parameter values have been discussed in the software package SOM_PAK [1], as well as the book [2].

1.3 The batch version of the SOM

Another remark concerns faster algorithms. The incremental regression process defined by (1) and (2) can often be replaced by the following batch computation version which is significantly faster and does not require specification of any learning-rate factor $\alpha(t)$.
Assuming that the convergence to some ordered state is true, we require that the expectation values of $\mathbf{m}_i(t+1)$ and $\mathbf{m}_i(t)$ for $t \to \infty$ must be equal, even if $h_{ci}(t)$ were then selected nonzero. In other words, in the stationary state we must have

$$\forall i, \quad \mathrm{E}_t\{h_{c(\mathbf{x}),i}(\mathbf{x} - \mathbf{m}_i^*)\} = 0 \,. \tag{4}$$

In the special case where we have a finite number (batch) of the $\mathbf{x}(t)$ with respect to which (4) has to be solved for the $\mathbf{m}_i^*$, and $h_{c(\mathbf{x}),i}$ represents the kernels used during the last phases of the learning process, we can write (4) as

$$\mathbf{m}_i^* = \frac{\sum_t h_{c(\mathbf{x}),i}\mathbf{x}(t)}{\sum_t h_{c(\mathbf{x}),i}} \,. \tag{5}$$

This, however, is not yet an explicit solution for $\mathbf{m}_i^*$, because the subscript $c(\mathbf{x})$ on the right-hand side still depends on $\mathbf{x}(t)$ *and all the* $\mathbf{m}_i^*$. The way of writing (5), however, allows us to apply the *contractive mapping method* known from the theory of nonlinear equations: starting with even coarse approximations for the $\mathbf{m}_i^*$, (2) is first utilized to find the indices $c(\mathbf{x})$ for all the $\mathbf{x}(t)$. On the basis of the approximate $h_{c(\mathbf{x}),i}$ values, the improved approximations for the $\mathbf{m}_i^*$ are computed from (5), which are then applied to (2), whereafter the computed $c(\mathbf{x})$ are substituted to (5), and so on. The optimal solutions $\mathbf{m}_i^*$ are usually obtained in a few iteration cycles, after the discrete-valued indices $c(\mathbf{x})$ have settled down and are no longer changed in further iterations. This procedure is called the *Batch Map* principle.
An even simpler Batch Map principle is obtained if $h_{c(\mathbf{x}),i}$ is defined in terms of the neighborhood set N_c. Further we need the concept of the *Voronoi set*. It means a domain V_i in the $\mathbf{x}$ space, or actually the set of those samples $\mathbf{x}(t)$ that lie closest to $\mathbf{m}_i^*$. Let us recall that we defined N_i as the set of nodes that lie up to a certain radius from node i in the array. The union of Voronoi sets V_i corresponding to the nodes in N_i shall be denoted by U_i. Then (5) can be written

$$\mathbf{m}_i^* = \frac{\sum_{\mathbf{x}(t)\in U_i} \mathbf{x}(t)}{n(U_i)} \,, \tag{6}$$

where $n(U_i)$ means the number of samples $\mathbf{x}(t)$ that belong to U_i.
Notice again that the U_i depend on the $\mathbf{m}_i^*$, and therefore (6) must be solved iteratively. The procedure can be described as the following steps:

1. Initialize the values of the $\mathbf{m}_i^*$ in some proper way. (Even random values for the $\mathbf{m}_i^*$ will usually do.)

2. Input all the $\mathbf{x}(t)$, one at a time, and list each of them under the model $\mathbf{m}_i^*$ that is closest to $\mathbf{x}(t)$ according to (2).

3. Let U_i denote the union of the above lists at model $\mathbf{m}_i^*$ and its neighbors that constitute the neighborhood N_i. Compute the means of the vectors $\mathbf{x}(t)$ in each U_i, and replace the old values of $\mathbf{m}_i^*$ by the respective means.

4. Repeat from 2 a few times until the solutions can be regarded as steady.

A further acceleration of computation results if one notes that for the different nodes i, the same addends occur a great number of times. Therefore it is advisable to first compute the mean $\bar{\mathbf{x}}_j$ of the $\mathbf{x}(t)$ in each Voronoi set V_j and then weight it by the number n_j of samples in V_j and the neighborhood function. Now we obtain

$$\mathbf{m}_i^* = \frac{\sum_j n_j h_{ji} \bar{\mathbf{x}}_j}{\sum_j n_j h_{ji}} \; , \tag{7}$$

where the sum over j is taken for all units of the SOM. For the case in which neighborhood sets N_i are used,

$$\mathbf{m}_i^* = \frac{\sum_{j \in N_i} n_j \bar{\mathbf{x}}_j}{\sum_{j \in N_i} n_j} \; . \tag{8}$$

A convergence and ordering proof of the Batch Map has been presented in [3].
There is a Matlab SOM Toolbox program package available in the Internet at the address http://www.cis.hut.fi/projects/somtoolbox/, which makes use of the Batch Map method.

1.4 Learning Vector Quantization (LVQ)

If each of the sample vectors $\mathbf{x}(t)$ is known to belong to some predefined class, and the model vectors $\mathbf{m}_i(t)$ are labeled by symbols corresponding to the predefined classes, too, then a supervised-learning algorithm can be used to fine tune the model vectors [2]. The basic LVQ1 algorithm can be written in a compressed form as

$$\begin{aligned}
\mathbf{m}_i(t+1) &= \mathbf{m}_i(t) + \alpha(t)s(t)\delta_{ci}[\mathbf{x}(t) - \mathbf{m}_i(t)] \; , \\
\text{where } s(t) &= +1 \;\; \text{if } \mathbf{x} \text{ and } \mathbf{m}_c \text{ belong to the same class,} \\
\text{but } s(t) &= -1 \;\; \text{if } \mathbf{x} \text{ and } \mathbf{m}_c \text{ belong to different classes} \; .
\end{aligned} \tag{9}$$

Here $\alpha(t)$ is the scalar-valued *learning-rate factor*, $0 < \alpha(t) < 1$, and δ_{ci} is the Kronecker delta ($= 1$ for $c = i$, $= 0$ for $c \neq i$) ; usually $\alpha(t)$ is initially of the order of a couple percent and decreases monotonically with time. The index c labels the winner according to (2). Notice that the neighborhood set around the winner now consists of the winner itself only.

Batch-LVQ1

The LVQ1 algorithm, like the SOM, can be expressed as a batch version. In a similar way as with the Batch Map (SOM) algorithm, the equilibrium condition for the LVQ1 is expresssed as

$$\forall i, \quad \mathrm{E}_t \left\{ s(t)\delta_{ci}(\mathbf{x} - \mathbf{m}_i^*) \right\} = 0 \ . \tag{10}$$

The computing steps of the so-called *Batch-LVQ1* algorithm (in which at steps 2 and 3 the class labels of the nodes are redefined dynamically) can then be expressed, in analogy with the Batch Map, as follows:

1. For the initial reference vectors take, for instance, those values obtained in the preceding unsupervised SOM process, where the classification of $\mathbf{x}(t)$ was not yet taken into account.
2. Input the $\mathbf{x}(t)$ again, this time listing the $\mathbf{x}(t)$ *as well as their class labels* under each of the corresponding winner nodes.
3. Determine the labels of the nodes according to the majorities of the class labels of the samples in these lists.
4. Multiply in each partial list all the $\mathbf{x}(t)$ by the corresponding factors $s(t)$ that indicate whether $\mathbf{x}(t)$ and $\mathbf{m}_c(t)$ belong to the same class or not.
5. At each node i, take for the new value of the reference vector the entity

$$\mathbf{m}_i^* = \frac{\sum_{t'} s(t')\mathbf{x}(t')}{\sum_{t'} s(t')} \ , \tag{11}$$

   where the summation is taken over the indices t' of those samples that were listed under node i.

6. Repeat from 2 a few times.

Comment 1. For stability reasons it may be necessary to check the sign of $\sum_{t'} s(t')$. If it becomes negative, no updating of this node is made.
Comment 2. Unlike in usual LVQ, the labeling of the nodes was allowed to change in the iterations. This has sometimes yielded slightly better classification accuracies than if the labels of the nodes were fixed at first steps. Alternatively, the labeling can be determined permanently immediately after the SOM process.

1.5 Further remarks

Finally it should be taken into account that the purpose of the SOM is usually visualization of data spaces. For an improved quality (isotropy) of the display it is then advisable to select the grid of the SOM units as hexagonal; the reason is similar as when using a hexagonal screen for images, say, in color television.
The above algorithms can be generalized, e.g., by defining various generalized matching criteria.
The following categories of similarity graphs, computed by the SOM, have already been used in many practical applications:

1. State diagrams for processes and machines
2. Data mining applications: similarity graphs for
   - statistical tables
   - full-text document collections

A list of 3043 research papers from very different application areas of the SOM and its variations is presented in [4].

References

[1] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J. (1996). SOM_PAK: The self-organizing map program package. Report A31. Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland. Also available in the Internet at the address http://www.cis.hut.fi/nnrc/nnrc-programs.html.

[2] Kohonen, T. (1995). *Self-Organizing Maps*. Series in Information Sciences, Vol. 30. Springer, Heidelberg. Second ed. 1997.

[3] Cheng, Y. (1997). Convergence and ordering of Kohonen's batch map. *Neural Computation*, Vol. 9, pp. 1667-1676.

[4] Kangas, J. and Kaski, S. (1998) 3043 works that have been based on the self-organizing map (SOM) method developed by Kohonen. Report A50. Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland. Also available in the Internet at the address http://www.cis.hut.fi/nnrc/refs/references.ps.