Self-Organizing Map in Matlab: the SOM Toolbox

Juha Vesanto, Johan Himberg, Esa Alhoniemi and Juha Parhankangas
Laboratory of Computer and Information Science
Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland
Juha.Vesanto@hut.fi

March 10, 2000

Abstract

The Self-Organizing Map (SOM) is a vector quantization method which places the prototype vectors on a regular low-dimensional grid in an ordered fashion. This makes the SOM a powerful visualization tool. The SOM Toolbox is an implementation of the SOM and its visualization in the Matlab 5 computing environment. In this article, the SOM Toolbox and its usage are shortly presented. Also its performance in terms of computational load is evaluated and compared to a corresponding C-program.

Notice: This document is an updated version of the paper published in Matlab-DSP 1999 Conference [1].

1 General

This article presents the (second version of the) SOM Toolbox [2], hereafter simply called the Toolbox, for Matlab 5 computing environment by MathWorks, Inc. The SOM acronym stands for Self-Organizing Map (also called Self-Organizing Feature Map or Kohonen map), a popular neural network based on unsupervised learning [3]. The Toolbox contains functions for creation, visualization and analysis of Self-Organizing Maps. The Toolbox is available free of charge under the GNU General Public License from `http://www.cis.hut.fi/projects/somtoolbox`.

The Toolbox was born out of need for a good, easy-to-use implementation of the SOM in Matlab for research purposes. In particular, the researchers responsible for the Toolbox work in the field of data mining, and therefore the Toolbox is oriented towards that direction in the form of powerful visualization functions. However, also people doing other kinds of research using SOM will probably find it useful — especially if they have not yet made a SOM implementation of their own in Matlab environment. Since much effort has been put to make the Toolbox relatively easy to use, it can also be used for educational purposes.

The Toolbox — the basic package together with contributed functions — can be used to preprocess data, initialize and train SOMs using a range of different kinds of topologies, visualize SOMs in various ways, and analyze the properties of the SOMs and data, e.g. SOM quality, clusters on the map and correlations between variables. With data mining in mind, the Toolbox and the SOM in general is best suited for data understanding or survey, although it can also be used for classification and modeling.

2 Self-organizing map (SOM)

A SOM consists of neurons organized on a regular low-dimensional grid, see Figure 1. Each neuron is a d-dimensional weight vector (prototype vector, codebook vector) where d is equal to the dimension of the input vectors. The neurons are connected to adjacent neurons by a neighborhood relation, which dictates the topology, or structure, of the map. In the Toolbox, topology is divided to two factors: local lattice structure (hexagonal or rectangular, see Figure 1) and global map shape (sheet, cylinder or toroid).

The SOM can be thought of as a net which is spread to the data cloud. The SOM training algorithm moves the weight vectors so that they span across the data cloud and so that the map is organized: neighboring neurons on the grid get similar weight vectors. Two variants of the SOM training algorithm have been implemented in the Toolbox. In the traditional sequential training, samples are presented to the map one at a time, and the algorithm gradually moves the weight vectors towards them, as shown in Figure 2. In the batch training, the data set is presented to the SOM as a whole, and the new weight vectors are weighted averages of the data vectors. Both algorithms are iterative, but the batch version is much faster in Matlab since matrix operations can be utilized efficiently.

For a more complete description of the SOM and its implementation in Matlab, please refer to the book by Kohonen [3], and to the SOM Toolbox documentation.

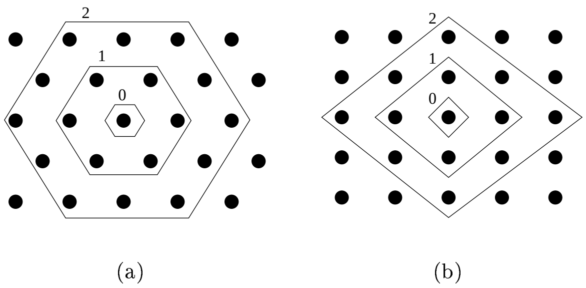

Figure 1: Neighborhoods (0, 1 and 2) of the centermost unit: hexagonal lattice on the left, rectangular on the right. The innermost polygon corresponds to 0-, next to the 1- and the outmost to the 2-neighborhood.

3 Performance

The Toolbox can be downloaded for free from **http://www.cis.hut.fi/projects/somtoolbox/**. It requires no other toolboxes, just the basic functions of Matlab (version 5.2 or later). The total diskspace required for the Toolbox itself is less than 1 MB. The documentation takes a few MBs more.

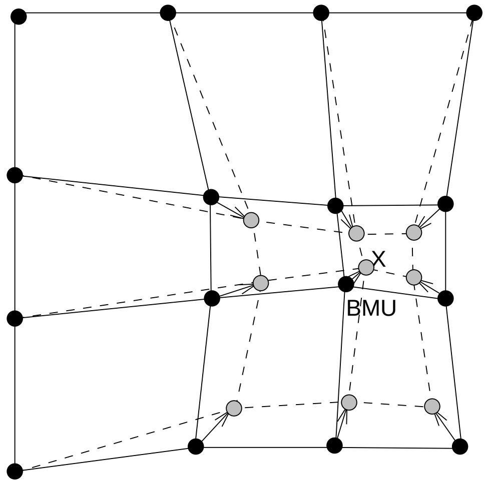

Figure 2: Updating the best matching unit (BMU) and its neighbors towards the input sample marked with `x`. The solid and dashed lines correspond to situation before and after updating, respectively.

The performance tests were made in a machine with 3 GBs of memory and 8 250 MHz R10000 CPUs (one of which was used by the test process) running IRIX 6.5 operating system. Some tests were also performed in a workstation with a single 350 MHz Pentium II CPU, 128 MBs of memory and Linux operating system. The Matlab version in both environments was 5.3.

The purpose of the performance tests was only to evaluate the computational load of the algorithms. No attempt was made to compare the quality of the resulting mappings, primarily because there is no uniformly recognized "correct" method to evaluate it. The tests were performed with data sets and maps of different sizes, and three training functions: `som_batchtrain`, `som_seqtrain` and `som_sompaktrain`, the last of which calls the C-program `vsom` to perform the actual training. This program is part of the SOM_PAK [4], which is a free software package implementing the SOM algorithm in ANSI-C.

Some typical computing times are shown in Table 1. As a general result, `som_batchtrain` was clearly the fastest. In IRIX it was upto 20 times faster than `som_seqtrain` and upto 8 times faster than `som_sompaktrain`. Median values were 6 times and 3 times, respectively. The som_batchtrain was especially faster with larger data sets, while with a small set and large map it was actually slower. However, the latter case is very atypical, and can thus be ignored. In Linux, the smaller amount of memory clearly came into play: the marginal between batch and other training functions was halved.

The number of data samples clearly had a linear effect on the computational load. On the other hand, the number of map units seemed to have a quadratic effect, at least with som_batchtrain. Of course, also increase in input dimension increased the computing times: about two- to threefold as input dimension increased from 10 to 50. The most suprising result of the performance test was that especially with large data sets and maps, the som_batchtrain outperformed the C-program (vsom used by som_sompaktrain). The reason is probably the fact that in SOM_PAK, distances between map units on the grid are always calculated anew when needed. In SOM Toolbox, all these are calculated beforehand. Likewise for many other required matrices.

Indeed, the major deficiency of the SOM Toolbox, and especially of batch training algorithm, is the expenditure of memory. A rough lower bound estimate of the amount of memory used by som_batchtrain is given by: $8(5(m+n)d + 3m^2)$ bytes, where m is the number of map units, n is the number of data samples and d is the input space dimension. For [3000 x 10] data matrix and 300 map units the amount of memory required is still moderate, in the order of 3.5 MBs. But for [30000 x 50] data matrix and 3000 map units, the memory requirement is more than 280 MBs, the majority of which comes from the last term of the equation. The sequential algorithm is less extreme requiring only one half or one third of this. SOM_PAK requires much less memory, about 20 MBs for the [30000 x 50] case, and can operate with buffered data.

Table 1: Typical computing times. Data set size is given as [n x d] where n is the number of data samples and d is the input dimension.

data size	units	batch	seq	sompak
IRIX				
[300×10]	30	0.2 s	3.1 s	0.9 s
[3000×10]	300	7 s	54 s	17 s
[30000×10]	1000	5 min	19 min	9 min
[30000×50]	3000	27 min	5.7 h	75 min
Linux				
[300×10]	30	0.3 s	2.7 s	1.9 s
[3000×10]	300	24 s	76 s	26 s
[30000×10]	1000	13 min	40 min	15 min

4 Use of SOM Toolbox

4.1 Data format

The kind of data that can be processed with the Toolbox is so-called spreadsheet or table data. Each row of the table is one data sample. The columns of the table are the variables of the data set. The variables might be the properties of an object, or a set of measurements measured at a specific time. The important thing is that every sample has the same set of variables. Some of the values may be missing, but the majority should be there. The table representation is a very common data format. If the available data does not conform to these specifications, it can usually be transformed so that it does.

The Toolbox can handle both numeric and categorial data, but only the former is utilized in the SOM algorithm. In the Toolbox, categorial data can be inserted into labels associated with each data sample. They can be considered as post-it notes attached to each sample. The user can check on them later to see what was the meaning of some specific sample, but the training algorithm ignores them. Function som_autolabel can be used to handle categorial variables. If the categorial variables need to be utilized in training the SOM, they can be converted into numerical variables using, e.g., mapping or 1-of-n coding [5].

Note that for a variable to be "numeric", the numeric representation must be meaningful: values 1, 2 and 4 corresponding to objects A, B and C should really mean that (in terms of this variable) B is between A and C, and that the distance between B and A is smaller than the distance between B and C. Identification numbers, error codes, etc. rarely have such meaning, and they should be handled as categorial data.

4.2 Construction of data sets

First, the data has to be brought into Matlab using, for example, standard Matlab functions `load` and `fscanf`. In addition, the Toolbox has function `som_read_data` which can be used to read ASCII data files:

```
sD = som_read_data('data.txt');
```

The data is usually put into a so-called data struct, which is a Matlab struct defined in the Toolbox to group information related to a data set. It has fields for numerical data (`.data`), strings (`.labels`), as well as for information about data set and the individual variables. The Toolbox utilizes many other structs as well, for example a map struct which holds all information related to a SOM. A numerical matrix can be converted into a data struct with: `sD = som_data_struct(D)`. If the data only consists of numerical values, it is not actually necessary to use data structs at all. Most functions accept numerical matrices as well. However, if there are categorial variables, data structs has be used. The categorial variables are converted to strings and put into the `.labels` field of the data struct as a cell array of strings.

4.3 Data preprocessing

Data preprocessing in general can be just about anything: simple transformations or normalizations performed on single variables, filters, calculation of new variables from existing ones. In the Toolbox, only the first of these is implemented as part of the package. Specifically, the function `som_normalize` can be used to perform linear and logarithmic scalings and histogram equalizations of the numerical variables (the `.data` field). There is also a graphical user interface tool for preprocessing data, see Figure 3.

Scaling of variables is of special importance in the Toolbox, since the SOM algorithm uses Euclidean metric to measure distances between vectors. If one variable has values in the range of [0,...,1000] and another in the range of [0,...,1] the former will almost completely dominate the map organization because of its greater impact on the distances measured. Typically, one would want the variables to be equally important. The standard way to achieve this is to linearly scale all variables so that their variances are equal to one.

One of the advantages of using data structs instead of simple data matrices is that the structs retain information of the normalizations in the field `.comp_norm`. Using function `som_denormalize` one can reverse the normalization to get the values in the original scale: `sD = som_denormalize(sD)`. Also, one can repeat the exactly same normalizations to other data sets. All normalizations are single-variable transformations. One can make one kind of normalization to one variable, and another type of normalization to another variable. Also, multiple normalizations one after the other can be made for each variable. For example, consider a data set `sD` with three numerical variables. The user could do a histogram equalization to the first variable, a logarithmic scaling to the third variable, and finally a linear scaling to unit variance to all three variables:

```
sD = som_normalize(sD,'histD',1);
sD = som_normalize(sD,'log',3);
sD = som_normalize(sD,'var',1:3);
```

The data does not necessarily have to be preprocessed at all before creating a SOM using it. However, in most real tasks preprocessing is important; perhaps even the most important part of the whole process [5].

4.4 Initialization and training

There are two initialization (random and linear) and two training (sequential and batch) algorithms implemented in the Toolbox. By default linear initialization and batch training algorithm are used. The simplest way to initialize and train a SOM is to use function `som_make` which does both using automatically selected parameters:

```
sM = som_make(sD);
```

The training is done is two phases: rough training with large (initial) neighborhood radius and large (initial) learning rate, and finetuning with small radius and learning rate. If tighter control over the training parameters is desired, the respective initialization and training functions, e.g. `som_batchtrain`, can be used directly. There is also a graphical user interface tool for initializing and training SOMs, see Figure 4.

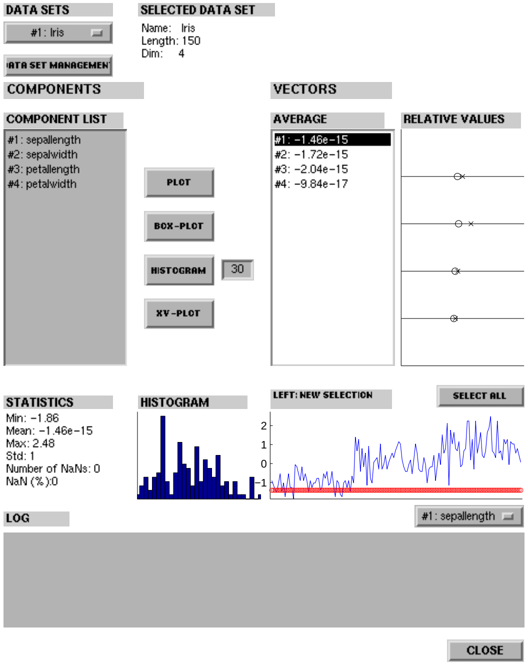

Figure 3: Data set preprocessing tool.

Figure 4: SOM initialization and training tool.

4.5 Visualization and analysis

There are a variety of methods to visualize the SOM. In the Toolbox, the basic tool is the function `som_show`. It can be used to show the U-matrix and the component planes of the SOM:

```
som_show(sM);
```

The U-matrix visualizes distances between neighboring map units, and thus shows the cluster structure of the map: high values of the U-matrix indicate a cluster border, uniform areas of low values indicate clusters themselves. Each component plane shows the values of one variable in each map unit. On top of these visualizations, additional information can be shown: labels, data histograms and trajectories.

With function `som_grid` much more advanced visualizations are possible. The function is based on the idea that the visualization of a data set simply consists of a set of objects, each with a unique position, color and shape. In addition, connections between objects, for example neighborhood relations, can be shown using lines. With `som_grid` the user is able to assign arbitrary values to each of these properties. For example, x-, y-, and z-coordinates, object size and color can each stand for one variable, thus enabling the simultaneous visualization of five variables. The different options are:

- the position of an object can be 2- or 3-dimensional
- the color of an object can be freely selected from the RGB cube, although typically indexed color is used
- the shape of an object can be any of the Matlab plot markers ('.','+', etc.)
- lines between objects can have arbitrary color, width and any of the Matlab line modes, e.g. '-'
- in addition to the objects, associated labels can be shown

For quantitative analysis of the SOM there are at the moment only a few tools. The function som_quality supplies two quality measures for SOM: average quantization error and topographic error. However, using low level functions, like som_neighborhood, som_bmus and som_unit_dists, it is easy to implement new analysis functions. Much research is being done in this area, and many new functions for the analysis will be added to the Toolbox in the future, for example tools for clustering and analysis of the properties of the clusters. Also new visualization functions for making projections and specific visualization tasks will be added to the Toolbox.

4.6 Example

Here is a simple example of the usage of the Toolbox to make and visualize a SOM of a data set. As the example data, the well-known Iris data set is used [6]. This data set consists of four measurements from 150 Iris flowers: 50 Iris-setosa, 50 Iris-versicolor and 50 Iris-virginica. The measurements are length and width of sepal and petal leaves. The data is in an ASCII file, the first few lines of which are shown below. The first line contains the names of the variables. Each of the following lines gives one data sample beginning with numerical variables and followed by labels.

```
#n sepallen sepalwid petallen petalwid
5.1 3.5 1.4 0.2 setosa
4.9 3.0 1.4 0.2 setosa
...
```

The data set is loaded into Matlab and normalized. Before normalization, an initial statistical look of the data set would be in order, for example using variable-wise histograms. This information would provide an initial idea of what the data is about, and would indicate how the variables should be preprocessed. In this example, the variance normalization is used. After the data set is ready, a SOM is trained. Since the data set had labels, the map is also labeled using som_autolabel. After this, the SOM is visualized using som_show. The U-matrix is shown along with all four component planes. Also the labels of each map unit are shown on an empty grid using som_show_add. The values of components are denormalized so that the values shown on the colorbar are in the original value range. The visualizations are shown in Figure 5.

```
%% make the data
sD = som_read_data('iris.data');
sD = som_normalize(sD,'var');

%% make the SOM
sM = som_make(sD);
sM = som_make(sD,'munits',30);
sM = som_autolabel(sM,sD,'vote');

%% basic visualization
som_show(sM,'umat','all','comp',1:4,...
         'empty','Labels','norm','d');
som_show_add('label',sM,'subplot',6);
```

From the U-matrix it is easy to see that the top three rows of the SOM form a very clear cluster. By looking at the labels, it is immediately seen that this corresponds to the Setosa subspecies. The two other subspecies Versicolor and Virginica form the other cluster. The U-matrix shows no clear separation between them, but from the labels it seems that they correspond to two different parts of the cluster. From the component planes it can be seen that the petal length and petal width are very closely related to each other. Also some correlation exists between them and sepal length. The Setosa subspecies exhibits small petals and short but wide sepals. The separating factor between Versicolor and Virginica is that the latter has bigger leaves.

Component planes are very convenient when one has to visualize a lot of information at once. However, when only a few variables are of interest scatter plots are much more efficient. Figures 6 and 7 show two scatter plots made using the som_grid function. Figure 6 shows the PCA-projection of both data and the map grid, and Figure 7 visualizes all four variables of the SOM plus the subspecies information using three coordinates, marker size and marker color.

5 Conclusions

In this paper, the SOM Toolbox has been shortly introduced. The SOM is an excellent tool in the visualization of high dimensional data [7]. As such it is most suitable for data understanding phase of the knowledge discovery process, although it can be used for data preparation, modeling and classification as well. In future work, our research will concentrate on the quantitative analysis of SOM

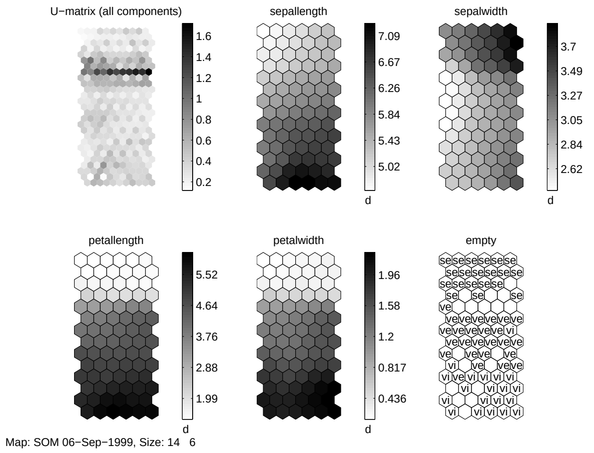

Figure 5: Visualization of the SOM of Iris data. U- matrix on top left, then component planes, and map unit labels on bottom right. The six figures are linked by position: in each figure, the hexagon in a certain position corresponds to the same map unit. In the U-matrix, additional hexagons exist between all pairs of neighboring map units. For example, the map unit in top left corner has low values for sepal length, petal length and width, and relatively high value for sepal width. The label associated with the map unit is 'se' (Setosa) and from the U-matrix it can be seen that the unit is very close to its neighbors.

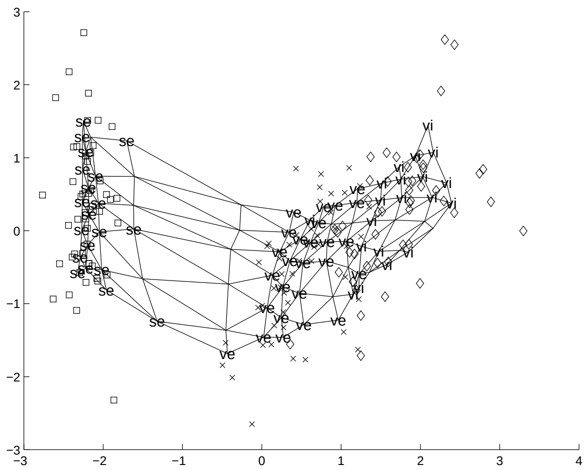

Figure 6: Projection of the IRIS data set to the subspace spanned by its two eigenvectors with greatest eigenvalues. The three subspecies have been plotted using different markers: □ for Setosa, × for Versicolor and ◊ for Virginica. The SOM grid has been projected to the same subspace. Neighboring map units are connected with lines. Labels associated with map units are also shown.

mappings, especially analysis of clusters and their properties. New functions and graphical user interface tools will be added to the Toolbox to increase its usefulness in data mining. Also outside contributions to the Toolbox are welcome. It is our hope that the SOM Toolbox promotes the utilization of SOM algorithm — in research as well as in industry — by making its best features more readily accessible.

Acknowledgments

This work has been partially carried out in 'Adaptive and Intelligent Systems Applications' technology program of Technology Development Center of Finland, and the EU financed Brite/Euram project 'Application of Neural Network Based Models for Optimization of the Rolling Process' (NEUROLL). We would like to thank Mr. Mika Pollari for implementing the initialization and training GUI.

References

[1] Juha Vesanto, Johan Himberg, Esa Alhoniemi, and Juha Parhankangas. Self-organizing map in matlab: the som toolbox. In *Proceedings of the Matlab DSP Conference 1999*, pages 35–40, Espoo, Finland, November 1999.

[2] Juha Vesanto, Esa Alhoniemi, Johan Himberg, Kimmo Kiviluoto, and Jukka Parviainen. Self-Organizing Map for Data Mining in Matlab: the SOM Toolbox. *Simulation News Europe*, (25):54, March 1999.

[3] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences.* Springer, Berlin, Heidelberg, 1995.

[4] Kohonen T., Hynninen J., Kangas J., Laaksonen J. SOM_PAK: The Self-Organizing Map Program Package, Technical Report

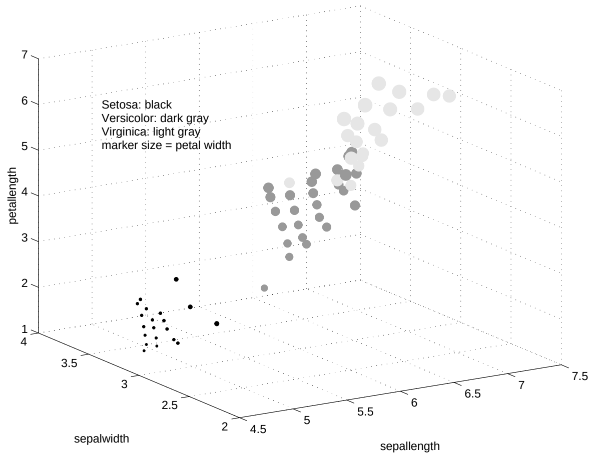

Figure 7: The four variables and the subspecies information from the SOM. Three coordinates and marker size show the four variables. Marker color gives subspecies: black for Setosa, dark gray for Versicolor and light gray for Virginica.

A31, Helsinki University of Technology, 1996, `http://www.cis.hut.fi/research/som_lvq_pak.shtml`

[5] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, 1999.

[6] Anderson E. The Irises of the Gaspe Peninsula. Bull. American Iris Society; 1935;59:2-5.

[7] Juha Vesanto. SOM-Based Data Visualization Methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

Address for correspondence.
SOM Toolbox team
Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland
somtlbx@mail.cis.hut.fi
http://www.cis.hut.fi/projects/somtoolbox/