Fast Self-Organizing of n-dimensional Topology Maps

Karin Haese, Heinz-Dieter vom Stein
University of the Federal Armed Forces, Hamburg
Signal Processing
Holstenhofweg 85, D - 22043 Hamburg
Tel.: (040)-65412462 Fax: (040)-6530413 Email: e_haese@unibw-hamburg.de

ABSTRACT
The self-organizing algorithm, proposed in this contribution, is more efficient than the original one, because it starts at a coarse lattice and refines the lattice of the map using spline interpolation at well determined learning steps until a quantization criteria is reached. Therefore the feature map becomes self-growing.
The proposed algorithm also includes a hierarchical search for the nearest neighbour. All these enhancements lead to a time complexity of order $O(logN)$ of the self-organizing algorithm for a $n_{\mathcal{A}}$-dimensional map with N neurons in each dimension.

1 Introduction

The *self-organizing topological map* (SOM) of Kohonen [2] [3] is an important tool to analyse high dimensional data sets $\mathcal{M}$ of unknown density distributions. These high dimensional data $\boldsymbol{m} \in \mathcal{M}$ are mapped onto a low $n_{\mathcal{A}}$-dimensional discrete lattice of units (*neurons*). The number $N^{n_{\mathcal{A}}}$ of neurons on the lattice has to be greater than 1000 in case of real problems. Thoroughly then Kohonens algorithm becomes unfeasible, especially because the nearest neighbour $\boldsymbol{w_{r'}}$ to the input $\boldsymbol{m}$

$$||\boldsymbol{w_{r'}}(j) - \boldsymbol{m}(j)|| = \min_{\boldsymbol{r}\in\mathcal{A}} ||\boldsymbol{w_r}(j) - \boldsymbol{m}(j)|| \quad (1)$$

has to be found at each time step j among all $N^{n_{\mathcal{A}}}$ weights $\boldsymbol{w_r}$.
The time complexity C_{search} is of order $O(N^{n_{\mathcal{A}}})$.
Afterwards the weights $\boldsymbol{w_r}$ are updated according to the well known learning rule

$$\begin{aligned} \boldsymbol{w_r}(j) &= \boldsymbol{w_r}(j-1) + \Delta w_{\boldsymbol{r}}(j) && (2) \\ \Delta w_{\boldsymbol{r}}(j) &= \epsilon(j) \cdot h_{\boldsymbol{rr'}}(j) \cdot [\boldsymbol{w_r}(j-1) - \boldsymbol{m}(j)]\,, && (3) \end{aligned}$$

with the neighbourhood function

$$h_{\boldsymbol{rr'}}(j) = exp\left(\frac{||\boldsymbol{r}-\boldsymbol{r'}||^2}{2\cdot\varsigma^2(j)}\right), \quad (4)$$

where

$$\begin{aligned} ||\boldsymbol{r}-\boldsymbol{r'}|| &= \sqrt{(r_1-r_1')^2 + \cdots (r_{n_{\mathcal{M}}} - r'_{n_{\mathcal{M}}})^2}, && (5) \\ \varsigma(j) &= \varsigma_0\, exp\left(-\mu_0 \left(\frac{j}{j_{max}}\right)^2\right) && (6) \end{aligned}$$

and

$$\epsilon(j) = \epsilon_0\, exp\left(-\nu_0 \left(\frac{j}{j_{max}}\right)^2\right). \quad (7)$$

The time complexity C_{update} of weight update is independent of N. Therefore Kohonens algorithm has time complexity $C_{SOM} = C_{search} + C_{update} = O(N^{n_{\mathcal{A}}})$.

First approaches to reduce the time complexity of this algorithm have been made by LAMPINEN and OJA [4] for small numbers $N^{n_{\mathcal{A}}} < 1000$ and high input dimensions $n_{\mathcal{M}} > 8$. The latest proposal in 1992, published by JUN ET. AL.[1], reaches time complexity $O(\log N)$ for a two-dimensional map, but needs more learning steps than the here proposed algorithm, named *Growing-Quick-SOM.* This advantage is achieved mainly by starting the self-organization process on a lattice with few neurons, refining the map when it is well ordered using spline interpolation and implementing a hierachical search for the nearest neighbour.

2 Lattice Refinement

At first a feature map with $N_0^{n_{\mathcal{A}}}$ neurons on a discrete lattice is trained using an exponentially decaying learning coefficient ϵ and standard deviation ς. When $\varsigma(j)$ is smaller than the distance ς_{min_0} between two neighbours on the map, the feature map is well ordered. Then the lattice with constant c_0 is refined to a lattice with $c_1 = c_0/2$ (see figure 1). This can be done by interpolating the neurons weight vectors in each dimension upon the lattice with constant c_1 using spline functions. Here spline functions of 4th order are used. After the interpolation process new learning parameters have to be calculated in the way, that there is continuity at the interpolation time step j_1. Therefore the equation

$$\varsigma_{min_0} = \varsigma_1\, exp\left(-\mu_1 \left(\frac{j_1}{j_{max}}\right)^2\right) \quad (8)$$

has to be solved for ς_1. For the following example the choice of $\mu_1 = \mu_0/\varsigma_0$ is appropriate and for all following

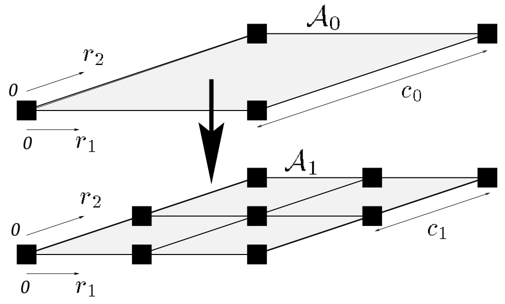

Figure 1: Splineinterpolation of a twodimensional lattice

interpolation time steps j_l $\mu_l = \mu_{l-1}/2$ $(l \geq 2)$ is used. Then j_l can be calculated by:

$$j_1 = \sqrt{\frac{j_{max}^2}{\mu_0} \ln \frac{\varsigma_0}{\varsigma_{min_0}}}, \quad l = 1 \tag{9}$$

$$j_l = \sqrt{j_{l-1}^2 + \frac{j_{max}^2}{\mu_{l-1}} \ln \frac{\varsigma_{min_l}}{\varsigma_{min_{l-1}}}}, \quad l \geq 2. \tag{10}$$

In the same way ϵ_l is determined. Figure 3 and 4 show typical learning parameter sequences with the time steps of interpolation signed.
Lattice refinement continues until a desired quantization error q is reached. This error is defined by the mean Euclidean distance of all input vectors to their nearest neighbour weight vectors:

$$q = \frac{1}{N_{\mathcal{M}}} \sum_{i=1}^{N_{\mathcal{M}}} \| \boldsymbol{w}_{\boldsymbol{r}'_i} - \boldsymbol{m}_i \|. \tag{11}$$

Its implementation makes the algorithm self-growing.
Using the mechanism of lattice refinement during the learning process a hierarchical search algorithm can be used to speed up the self-growing fast algorithm.

3 Hierarchical Search Algorithm

The hierarchical search can be applied the first time after the first interpolation time step j_1, because then it can be assumed that the topology of the input is preserved in the map. The hierarchical search begins among those neurons defining a coarse lattice with constant c_0. On this lattice a first approximation of the nearest neighbour to an input vector can be found. Imagine this first approximation is found at the place $\boldsymbol{r}'_0$ on the coarse lattice, then a second, better approximation is likely to be found in a surrounding of $\boldsymbol{r}'_0$ (light gray rectangle in figure 2), because of the topology preservation. The minimal surrounding contains 8 neurons. Among these neurons the search leads to a second better approximation at place $\boldsymbol{r}'_1$. This procedure is repeated until the lattice of search reaches constant c_l.

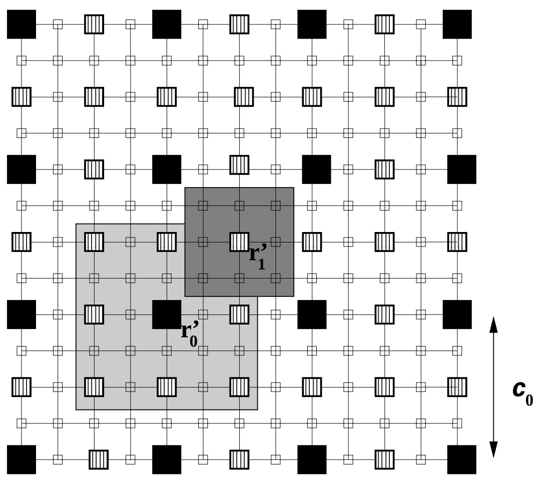

Figure 2: Hierarchical search for the nearest neighbour

4 Time Complexity

The time complexity of an algorithm is the average of the time for one learning step. In case of "Growing-Quick-SOM" this time complexity is the time for all searches of nearest neighbours T_{search} and the time for the spline interpolations T_{spline} during the whole learning process normalized on the maximal number of learning steps of the process:

$$C_{opt} = \frac{T_{search} + T_{spline}}{j_{max}}. \tag{12}$$

The time T_{search}, needed during the learning process, is:

$$\begin{aligned} T_{search} &= N_0^{n_{\mathcal{A}}} \, T_d \, j_{max} \\ &+ (N_u^{n_{\mathcal{A}}} - 1) T_d \, l_{max} (j_{max} - j_{l_{max}}) \\ &+ (N_u^{n_{\mathcal{A}}} - 1) T_d \sum_{l=1}^{l_{max}-1} (j_{l+1} - j_l) l, \end{aligned} \tag{13}$$

with

T_d	time needed to calculate the Euclidean distance between the input and one synaptic weight,
l	number of interpolations
j_l	time steps of interpolation,
$N^{n_{\mathcal{A}}}$	number of neurons on the lattice at j_{max},
$N_0^{n_{\mathcal{A}}}$	number of neurons on the initial lattice,
and	
$(N_u^{n_{\mathcal{A}}} - 1)$	number of neurons in the surroundings of an approximation to the nearest neighbour on coarser lattice

it follows

$$T_{search} = N_0^{n_{\mathcal{A}}} \, T_d \, j_{max}$$

$$+ \quad (N_u^{n_{\mathcal{A}}}-1)T_d \,\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)\left(j_{max}-j_{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)}\right)$$
$$+ \quad (N_u^{n_{\mathcal{A}}}-1)T_d \sum_{l=1}^{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)-1} (j_{l+1}-j_l)\, l. \qquad (14)$$

The term $S = \left(\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)\left(j_{max}-j_{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)}\right) + \sum_{l=1}^{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)-1} (j_{l+1}-j_l)\, l \right)$ can be approximated by its worst case using $l = \mathrm{ld}\left(\frac{N-1}{N_0-1}\right)$:

$$\begin{aligned} S &\le \mathrm{ld}\left(\frac{N-1}{N_0-1}\right)\left(j_{max}-j_{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)}\right) \\ &\quad + \mathrm{ld}\left(\frac{N-1}{N_0-1}\right) \sum_{l=1}^{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)-1} (j_{l+1}-j_l) \\ &= \mathrm{ld}\left(\frac{N-1}{N_0-1}\right)(j_{max}-j_1). \end{aligned}$$

Now, replacing j_1 by equation (9), the sum S can be estimated as follows:

$$S \le \mathrm{ld}\left(\frac{N-1}{N_0-1}\right) j_{max}\left(1-\frac{1}{\sqrt{\mu_0}}\sqrt{\ln\frac{\varsigma_0}{\varsigma_{min_0}}}\right).$$

This leads to the limit for the time complexity of distance calculation in equation (15).

$$\begin{aligned} C_{search} &\le T_d N_0^{n_{\mathcal{A}}} \\ &+ (N_u^{n_{\mathcal{A}}}-1)T_d\left(1-\frac{1}{\sqrt{\mu_0}}\sqrt{\frac{\ln(2N_0)}{c_0}}\right)\mathrm{ld}\left(\frac{N-1}{N_0-1}\right) \qquad (15) \end{aligned}$$

Equation (15) shows C_{search} is of time complexity $\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)$.
The other time complexity of spline calculation is determined as follows:

$$C_{spline} = \frac{n_{\mathcal{M}}T_{sp}}{j_{max}} \sum_{l=1}^{\mathrm{ld}\left(\frac{N-1}{N_0-1}\right)} \left(\frac{N-1}{2^l}\right)^{n_{\mathcal{A}}}. \qquad (16)$$

Using the geometrical series $\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}$ with $a = \frac{1}{2}$, C_{spline} is always lower than $\frac{n_{\mathcal{M}}T_{sp}}{j_{max}}(N-1)^{n_{\mathcal{A}}}$. Because $j_{max} \sim N_{\mathcal{M}} \sim N^{n_{\mathcal{A}}}$ the complexity of the spline approximation is constant. This leads to $O(\log(N))$ for the whole time complexity C_{opt} of "Growing-Quick-SOM".

5 Examples of simulation

In order to demonstrate that the proposed fast algorithm reaches the same topology preserving quality as the original algorithm a set of two dimensional input vectors with uniform distribution on a square is trained using the fast and the original algorithm.
Starting the fast algorithm on a lattice of 5x5 neurons a desired maximal quantization error of $q = 3$ is reached with two interpolations during the learning process (figure 9). Right before the interpolation steps the fast algorithm converges to a well organized map (figure 5 to 7). At the end of the learning process the map with 17x17 neurons has the same quality as a map of same size obtained using the original algorithm (figure 10). But the "Growing-Quick-SOM" is superior in speed. It is 7 times faster than the original one.

6 Summary

This paper proposes a self-growing fast algorithm for training $n_{\mathcal{A}}$-dimensional self-organizing feature maps. It shows that the time complexity of the original algorithm ($O(N^{n_{\mathcal{A}}})$) is reduced to $O(\log N)$. This advantage is achieved by starting on a coarse lattice, refine the lattice during the learning process until a desired quantization error is reached. In addition to that a hierarchical search algorithm is applied, so that a map of 17x17 neurons can be trained in about 1/7 of the time needed by the original algorithm.

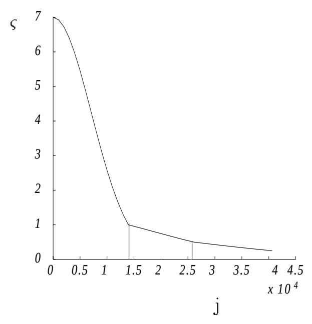

Figure 3: Standard deviation $\varsigma(j)$

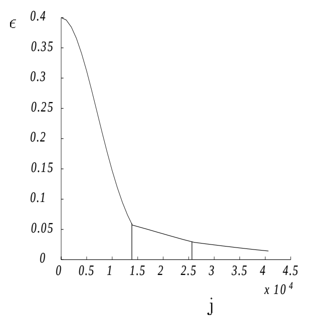

Figure 4: Learning coefficient $\epsilon(j)$

References

[1] Jun,Y. P.; Yoon, H. ; Cho, J. W. L*-Learning: A Fast Self-Organizing Feature Map Learning Algorithm Based on Incremental Ordering. *IEICE Trans. on Information & Systems*, E76-D(6):698–706, June 1993.

[2] Kohonen, T. Self-organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

[3] Kohonen,T. *Self-Organization and Associative Memory.* Springer Series in Information Sciences 8, Heidelberg, 1984.

[4] Lampinen, J.; Oja, E. Fast Self-Organizing by Probing Algorithm. *Proc. IEEE International Joint Conference on Neural Networks*, II:503–507, June 1989.

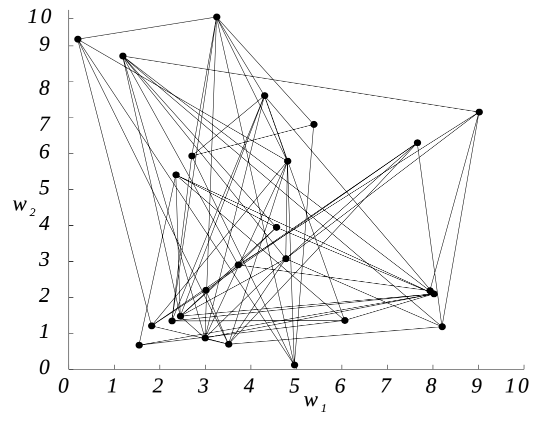

Figure 5: Initial map

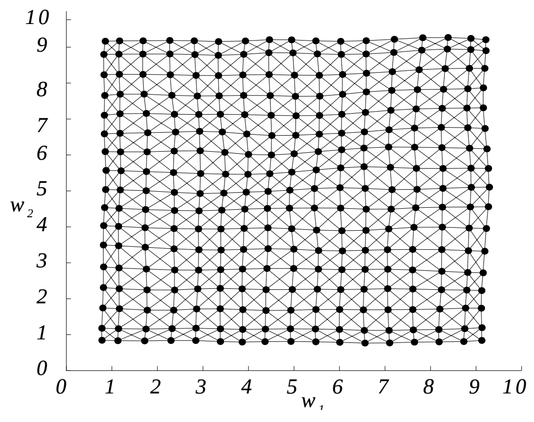

Figure 8: Map after $j = 40626$ learning steps with "Growing-Quick-SOM"

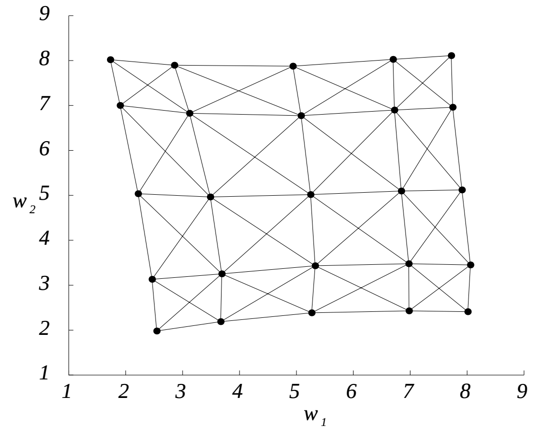

Figure 6: Map after $j = 13349$ learning steps with "Growing-Quick-SOM"

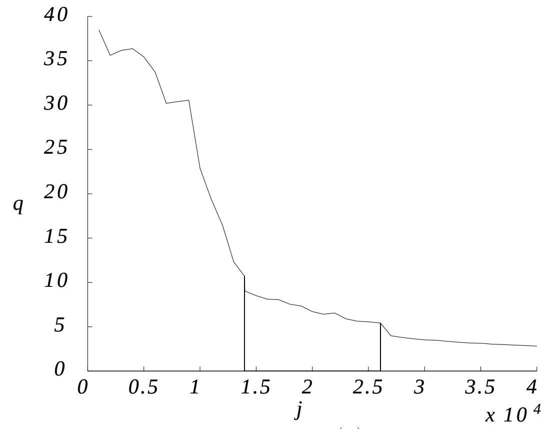

Figure 9: Error of quantization $q(j)$ during the learning process (figure 5 to 8)

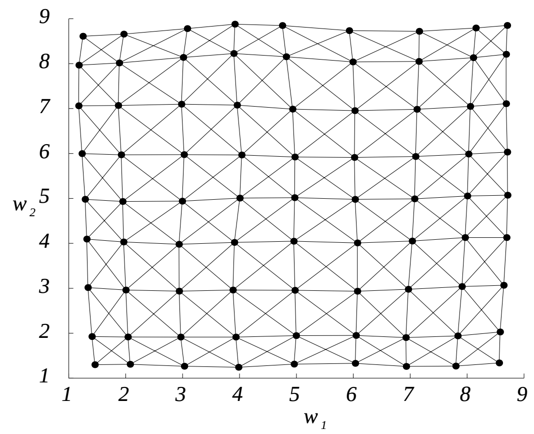

Figure 7: Map after $j = 26075$ learning steps with "Growing-Quick-SOM"

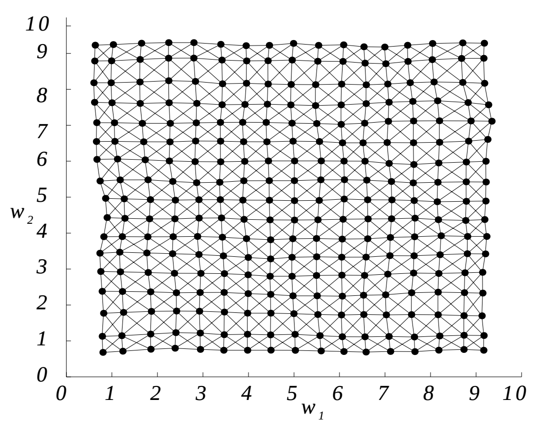

Figure 10: Map after $j = 40626$ learning steps with the original algorithm