ESANN'2005 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 27-29 April 2005, d-side publi., ISBN 2-930307-05-6.

Novel Algorithm for Eliminating Folding Effect in Standard SOM

Kirmene Marzouki[1] Takeshi Yamakawa[2]

1-School of Computer Science and Systems Engineering, Department of Control and Systems Engineering, Kyushu Institute of Technology,

680-4 Kawazu, Iizuka City, Fukuoka 820-8502, JAPAN

2- Graduate School of Life Science and Systems Engineering, Department of Brain Science and Engineering, Kyushu Institute of Technology

2-4 Hibikino, Wakamatsu-ku, Kitakyushu City, Fukuoka 808-0196, JAPAN

Abstract. Self-organizing maps, SOMs, are a data visualization technique developed to reduce the dimensions of data through the use of self-organizing neural networks. However, as the original input manifold can be complicated with an inherent dimension larger than that of the feature map, the dimension reduction in SOM can be too drastic, generating a folded feature map.
In order to eliminate this phenomenon, we extend the neighborhood concept to a new set of sub-neighbors, other than those introduced by Kohonen. The modified algorithm was applied to color classification and performed very well in comparison with the traditional SOM.

1 Introduction

This work is based on a former publication [1] of the same authors. In our previous work, we introduced some modifications on the standard SOM algorithm, allowing to the final map being independent from the units' weights vectors initial states. Inspired from Neural Gas Network [2], we have introduced a new type of neighbors based on weights vectors distance. We also introduced a new learning rule for the new set of neighbors using the information acquired by the net during the learning process, so that it contributes to the formation of the final map.
Although we showed we always could have the same final weight distribution independently from the initial conditions, the new learning algorithm we introduced still suffers from folding generated by the border effect, also known as boundary effect.
In this paper, we address the folding effect problem encountered in standard SOM.
In standard SOM the topology order of the prototype units is pre-determined and the learning process is to move the initialized units onto appropriate positions in the low dimensional feature map. As the original input manifold can be complicated with an inherent dimension larger than that of the feature map (usually set as 2 for visualization purpose), the dimension reduction in SOM can be too drastic, generating

a folded feature map.

Many works have addressed this issue. In [3] a spherical SOM is introduced so that boundaries are eliminated. In [4] the authors propose an Evolving SOM (ESOM), in which the map starts with zero units and as new inputs are fed to the network, new nodes and new connections are created. Fritzke [5] proposed the growing neural gas (GNG) model, which allows the neural gas model to grow by adding new nodes adaptively. Bruske and Sommer [6] presented a similar model called dynamic cell structure (DCS-GCS). Both GNG and DCS-GCS need to calculate local resources for prototypes, which introduces extra computational effort and reduces their efficiency.

All modifications and improvements made so far, have been brought out by modifying the basic structure of the unit space. We believe that the solution to the boundary effect exists in the way learning is performed while keeping the same basic structure.

Although many causes of the folding effect, caused by border effect, are cited and investigated in [2-7] in standard SOM, we believe that this problem is mainly and merely caused by the non-interactiveness of learning with the evolution of the network during the training process. In all versions of SOM, learning is performed identically from the start of the training process till its end. The same adaptation equation is used all along the process without really considering the changes of the network and the information being "*learnt*" so far.

In this work we propose to control the progress of the neighborhood function by making it consistent with learning.

2 Previous Work

Besides the traditional used topographic neighborhood function, we note *Nc*, we generate a new group of neighbors based on the weights distance as introduced in [2]. Each time an input vector is presented to the network, we investigate two groups of neighbors. The first one consists of those neighbors located within *Nc* as defined by Kohonen [7].

On the weight space, we define a detecting window allowing us to have a better view of the units space and detect those units the weights of which are very close to the BMU, but could not be selected by *Nc* and left apart because they are located beyond its borders.

These units constitute the main cause of the border effect generating folding of the feature map in standard SOM.

We note *Nw* the width of the detecting window, also we defined as the weight neighborhood function (on the weight space). Let *df* be the distance (on the units space) separating the BMU and the furthest unit the weight of which belongs to *Nw*. The new group of neighbors consists of all units situated within *df* and located outside *Nc* (Fig1).

Each group of neighbors follows its own learning rule.

$$W_i(t) = W_i(t-1) + \alpha(t)[x(t) - W_i(t-1)] \tag{1}$$

for units selected by *Nc* (same as in standard SOM) , and

$$W_j(t) = W_j(t-1) + \alpha(t)[x(t) - W_j(t-1)] + \alpha(t)(t/T)[A_j(t) - x(t)] \quad (2)$$

for all units *j* belonging to the new group of neighbors, where $W_j(t)$ is the newly updated weight, $W_j(t-1)$ is the old weight before update, $\alpha(t)$ is the learning rate, x(t) is the input, t is the current time epoch, T is the total number of iterations, and $A_j(t)$ is the average of the weights of the immediate neighbors of unit *j*.
For more details about how we generate (2), please refer to [1].

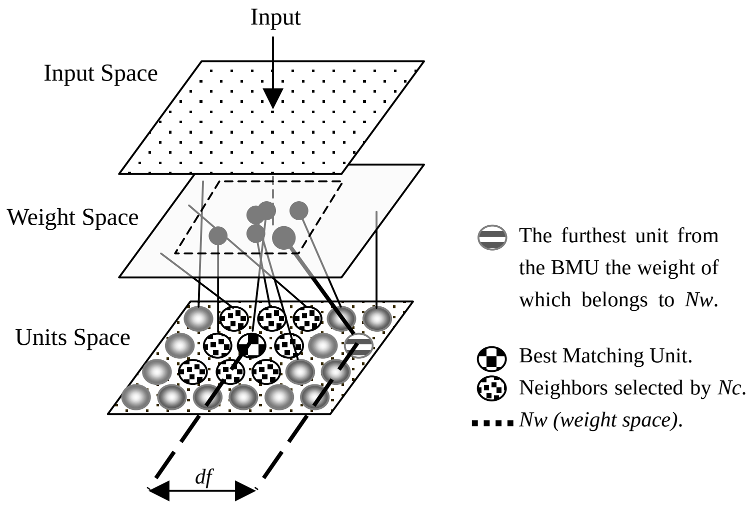

Input
Input Space
Weight Space
Units Space
df
The furthest unit from the BMU the weight of which belongs to *Nw*.
Best Matching Unit.
Neighbors selected by *Nc*.
Nw (weight space).

Fig. 1. The two neighborhood functions and the new sets of neighbors.

For this new type of neighbors, updating is made according to a new learning algorithm, in which not only the input information is considered, but also the information already existing on the map is used and takes part of the training process [1].

3 Interactive Learning

In all versions of SOM, the neighborhood function shrinks at a constant speed independently from learning quality. In this paper we propose to use *df* to control the shrinking of the neighborhood function.

If the learning process is going properly, the neighborhood function *Nc* should return a value very near to *df*. But since *Nc* is computed with respect to the constantly narrowing immediate surroundings of the BMU, after few iterations from the start of learning, it'll start returning values very different from *df*. The key idea of our algorithm consists of matching *Nc* and *df* in order to make *Nc*'s evolution interactive with training. For a matter of simplicity the two neighborhood functions we used have the following form

$$Nc = Po(1 - t/T) \cdot R_i \quad (3)$$

$$Nw = do(1 - t/T) \tag{4}$$

where *Po* and *do* are the values of *Nc* and *Nw* at t=0 respectively, R_i is a resizing factor initially set to 1, *t* is the current time, e.g. current iteration number, and *T* is the total time, e.g. Total number of iterations.

Let *i* be the BMU at time epoch *t*. From (3) and (4) the two neighborhood functions will return $Nc_i(t)$ and $Nw_i(t)$. If we force $Nc_i(t)$ to be equal to df_i, we'll have

$$df_i(t) = Nc_i\,(t) = Po(1 - t_x/T) \tag{5}$$

$$t_x = T(1 - df_i(t)/Po) \tag{6}$$

Since *t* is the only one variable, a new time variable t_x comes out. t_x represents the time epoch at which learning would be if Nc_i would have returned df_i (Fig. 2).

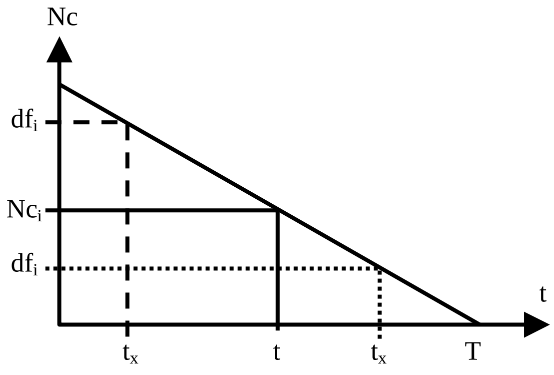


Fig 2. Learning Progress and different case figures of Nc_i and df_i.
If $df_i \geq Nc_i$, it means that learning is proceeding naturally. Conversely, in the case of $df_i < Nc_i$, learning is proceeding faster than the shrinking of Nc_i. In this case training should be slowed down.

In order to make the neighborhood shrinking speed interactive with training, one can replace *t* by t_x in all involved equations, that is either we step back in time, or make a jump into the future. This is neither plausible nor affordable.

Making a jump into the future means that we have to ignore the over all status of the net for a certain number of iterations, which is very risky.

On the other hand, stepping back in time needs to restore the learnt information to the one acquired up to t_x, which is almost impossible to achieve, because even though we manage to memorize every amount of update of all winning units at each time step, we also need to memorize each amount of update of all neighbors, and their numbers, of each winning unit at the same time epoch. Obviously both operations are impossible to achieve.

We choose to assign to each unit *i* its own neighborhood function Nc_i with a resizing factor, R_i, reflecting the delay between *t* and t_x. R_i is used to control the shrinking speed of the neighborhood function whenever it is necessary, as it is explained in the next section.

4 Algorithm

Step0: Input a sample vector from the training data set.

Step 1: Find the best matching unit to the input vector. Let *i* be the BMU.

Step 2: From (3) and (4) we investigate Nc_i and Nw_i, and we pick up df_i, the distance separating the BMU and the furthest units the weight of which belongs to Nw_i. At this point the two groups of neighbors are very well identified.

Step 3: Update the two groups of neighbors. Two case figures should be considered:

Case1: $df_i < Nc_i$ ($t_x > t$)
In this case, training is proceeding faster than Nc_i shrinking, which means that either we have to narrow Nc_i, or to inhibit the update.
Narrowing Nc_i means that we have to ignore the topographic neighbors returned by Kohonen's neighborhood function, which may encourage the creation of some *"isolated islands"* on the net. Therefore, the only one solution is to temporarily stop updating using (1), and use only (2) for all units included within Nc_i. In this case, the two sets of neighbors are processed identically.
Using (2) means that updating is more likely an operation of neighborhood strengthening rather than a real operation of update with regard to the input.
After update, we set

$$R_i=1 \tag{7}$$

that is we keep Nc_i's shrinking speed as it is for the next selections of the same BMU unit *i*, but slow down the updating with regard to the input.

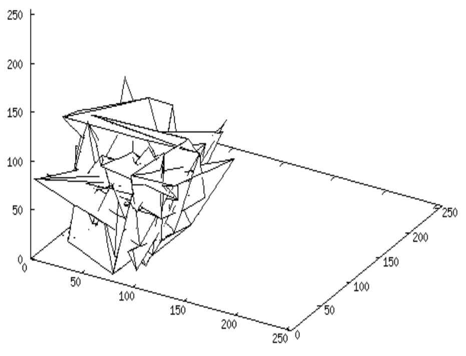
Case2: $df_i \geq Nc_i$ ($t_x < t$)
In this case, training is proceeding normally, but we detect some units that may cause folding. Updating is made according to (1) for units selected by Nc_i (SOM), and according to (2) for the new set of neighbors, respectively.
After update, we define the resizing factor as

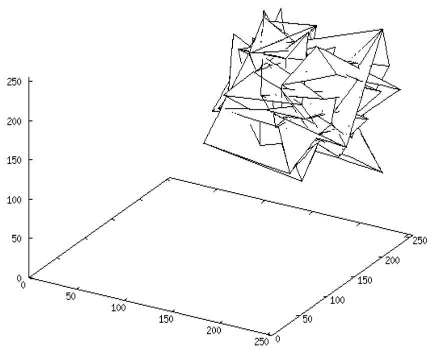
$$R_i= (T- t)/(T- t_x) \tag{8}$$

that is in the case that this same unit *i* is selected as the BMU in the future, we force its relative Nc_i to be less than df_i so that we keep the whole process going naturally, while keeping an eye on those suspicious units which may cause the boundary effect. Then we multiply R_i by the expression of Nc_i (3), to be considered in future selections. It is very important to point out that R_i is to be used in the future, and not in the following immediate steps.
Step 4: Input a new sample vector, if still available, and go to step 1. If not End of algorithm.

5 Experimental Results

For a matter of clarity, the chosen application to test the new algorithm consists of color mapping.
As SOM produces a mapping of classified data, it is then easy to evaluate how good a map is and how strong the similarities between objects are. In the case of color classification, it is very easy to check the resulted map and make comparisons between the two algorithms.
Simulations were performed with different network topologies with different number of training data and different iteration numbers. The training data and initial states were three-dimensional vectors on the RGB scale having the form $X_1=(R_1,G_1,B_1)$, $X_2=(R_2,G_2,B_2)$… $X_N=(R_n,G_n,B_n)$, where $R_m\in[0,255]$, $G_m\in[0,255]$ and $B_m\in[0,255]$. At each simulation, the initial states were drawn at random.
The shown results were performed on a 10x10 net lattice using 10000 training data for 5000 iterations. *Po* and *do* of (3) and (4) are set to 13 and 400, respectively. These are the maximum Euclidian distances separating two units on the unit space (10x10), and two weights on the weight space (0-255), respectively. Representative examples of the used initial data sets are plotted in Fig.3.



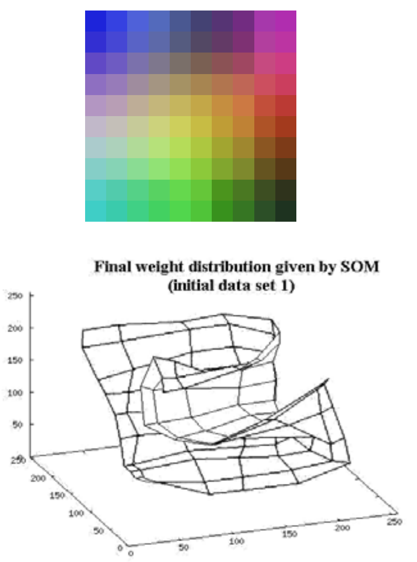


(a) (b)

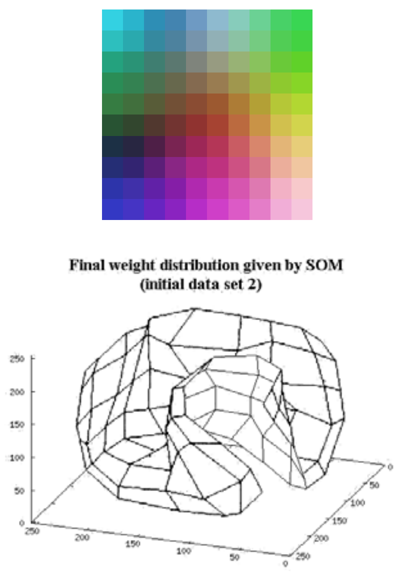
Fig. 3. Two different unit space surfaces relative to the used two random initial weight distributions, for two different simulations (a) and (b) respectively. X, Y and Z axis are scaled [0-255] on the RGB scale.

Simulation results given by the standard SOM algorithm are shown in Fig.4. (a') and (b') show the obtained colored maps and their relative unit space surfaces.
The folding of the obtained final unit space surface, in both simulations, indicates the occurrence of the border effect. This can be easily checked on the obtained color maps, by the presence of similarly colored different regions e.g. two purple and two black shades on the first, and two green and two blue shades on the second, respectively.

Simulation results given by the proposed algorithm are shown in Fig.5. We can clearly see that in the obtained color map all colors and their different shades are very



(a')



(b')

Fig. 4. Obtained results by the standard SOM from the two different sets of initial data (a) and (b) respectively. The colored maps represent the final weight distributions, and the graphs represent their unit space surfaces.

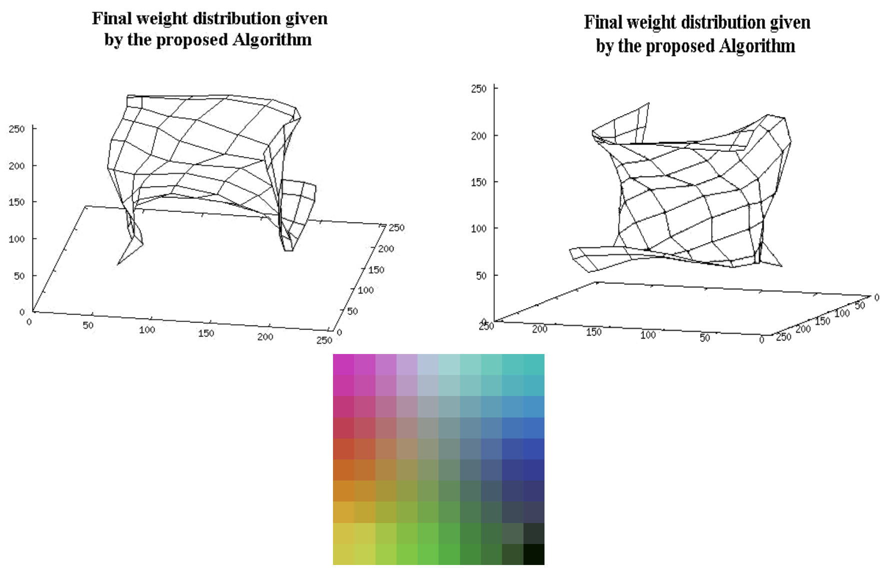


Fig. 5. Final weight distribution (colored map) and its unit space surface (front view and back view) obtained from the proposed algorithm.

well arranged. To confirm the consistency of the classification given by our algorithm,

the front view and the back view of the obtained unit space are shown. It is very easy to see that no boundary effect-like folding is present.
Besides, for all performed simulations, we obtained the same final weight distribution, independently from the initial states of weight vectors, as achieved in [1].

6 Conclusion

In this paper we proposed modifications on the traditional SOM algorithm, in order to eliminate folding generated by border effect, also known as boundary effect. Folding effect phenomenon consists of the appearance of two similar regions at different locations on the unit space.

The main reason of the occurrence of such phenomenon is attributed to the dimension reduction realized by standard SOM. As the original input manifold can be complicated with an inherent dimension larger than that of the feature map, the dimension reduction in SOM can be too drastic, generating a folded feature map.

To cope with this problem, we introduced a control mechanism through which the neighborhood function shrinks in an interactive manner with the training process by the use of a resizing factor.

Besides, we also introduced a new group of neighbors, other than the topographic neighbors, based on weight neighborhood function, so that two kinds of neighbors are processed. For this new type of neighbors, updating is made according to a new learning algorithm, in which not only the input information is considered, but also the information already existing on the map is used.

The proposed algorithm was tested on a three dimensional color classification application, and exhibited very stable behavior.
In all performed simulations, and independently from the initial states of weight vectors, we obtained the same final weight distribution free of border effect-like folding of the unit space.

References

[1] K. Marzouki and T. Yamakawa . Novel Learning Algorithm Aiming at Generating a Unique Units Distribution in Standard SOM. Being submitted to ICANNGA 2005, 21st -23rd March, Coimbra Portugal. Paper Accepted on November 15th.

[2] Martinetz, T., &Schulten, K. (1991). “Neural Gas” network learns topologies. In T.Kohonen et al. (Eds). Artificial neural networks (Vol I,pp 397-402), Amsterdam, North Holland.

[3] H. Ritter, Self-organizing maps on non-euclidean spaces, in: S. Oja, E. Kaski (Eds.), Kohonen Maps, Elsevier, Amsterdam, 1999, pp. 97-110.

[4] D. Deng, and N. Kasabov, ESOM: An algorithm to evolve self-organizing maps from on-line data streams, Proc. of IJCNN 2000, (IEEE Press, 2000), VI:3-8.

[5] B. Fritzke, Growing cell structures – a self-organizing network for unsupervised and supervised learning. Neural Networks 7 (1994) 1441-1460.

[6] J. Bruske, and G. Sommer, Dynamic cell structure learns perfectly topology preserving map, Neural Computation 7 (1995) 845-865.

[7] T.Kohonen, Self Organizing Maps. Springer-Verlag, Berlin, Germany, 1995.