Monitoring topographic maps

Susana Vegas-Azcárate
Statistics and Decision Sciences Group
University Rey Juan Carlos
28933 Móstoles, Spain
susana.vegas@urjc.es

Temujin Gautama
Laboratorium voor Neuro- en Psychofysiologie
Katholieke Universiteit Leuven
Campus Gasthuisberg, Herestraat 49, B-3000 Leuven, Belgium
temujin.gautama@philips.com

Marc M. Van Hulle
Laboratorium voor Neuro- en Psychofysiologie
Katholieke Universiteit Leuven
Campus Gasthuisberg, Herestraat 49, B-3000 Leuven, Belgium
marc@neuro.kuleuven.ac.be

Abstract - *The neighbourhood function is crucial for preserving a good topological ordering in the map. Hence, violations on the topographic ordering may result even in the one-dimensional case if the neighbourhood function range is decreased until it vanishes. Furthermore, the rate at which the neighbourhood function range is decreased over the training cycles controls almost completely the success of the lattice disentangling phase. This parameter controls in effect the degree of smoothness or topological order— it determines the amount of quantization error minimization performed. Although it is commonly suggested that the neighbourhood range should drop to small values by the end of the run, so that the map is able to fit the data, this is not appropriate in many cases. This paper develops a novel method to obtain topology-preserved maps from all topographic map formation algorithms.*

Key words - Self-organizing maps, topology preservation.

Acknowledgement - *The authors are partially supported by grants from the local CAM Government and the European Council, as well as from Spanish and European agencies and the DMR Foundation's Decision Engineering Lab.*

1 Introduction

In the last phases of the SOM training process the zero-order topology case arises, where no neighbourhood function remains and hence no lateral control of plasticity between neurons exist [Kohonen (2001)]. As a result of it, the self-organizing process does no longer maintain the order of the weight vectors, disturbing the quality of the topographic ordering. Indeed, for any topographic map formation algorithm based on fixed-topology lattices, topological defects may occur during the final stages of training.

The decreasing range of the neighbourhood function in SOM algorithm results on at least two types of phase transitions. One of them occurs when there is a topological mismatch between lattice and data manifold [Der and Herrmann (1993)]. The other happens in maps with short neighbourhood range [Der and Herrmann (1994)]. Topology-preservation will be favoured if the neighbourhood range is large enough to permit lattice elasticity. On the contrary, if neighbourhood range is not large enough, quantization error minimization will dominate, deteriorating the quality of the topographic map. In other words, when the neighbourhood has vanished, distortion minimization is pursued instead of topology-preservation, and the optimal distortion tessellation does not necessarily correspond to a regular topographic ordering [Der and Herrmann (1994); Van Hulle (2000)]. Haese and Goodhill (2001) have pointed out that a phase transition can also be seen as follows, "the algorithm turns at the phase transition from a principal curve mapping to a mapping overfitting the input data".

To avoid these phase transitions, the neighbourhood range σ_Λ must be large enough, developing a smoothness weight distribution [Mulier and Cherkassky (1995)]. That is, σ_Λ must create a tradeoff between the approximation accuracy of weight vector distribution $p(\mathbf{w})$ and the stability ordering. The open questions are : what does 'large enough' mean in a particular data set ? How can we select the appropriate rate with which the neighbourhood function range is decreased over the training cycles ? For the Gaussian-shaped neighbourhood functions, it has been suggested to maintain certain degree of overlapping— the neighbourhood range should not fall below half the distance between neighbouring nodes. Apart from a few rules of thumb, it would be desirable to have iterative methods for making these choices.

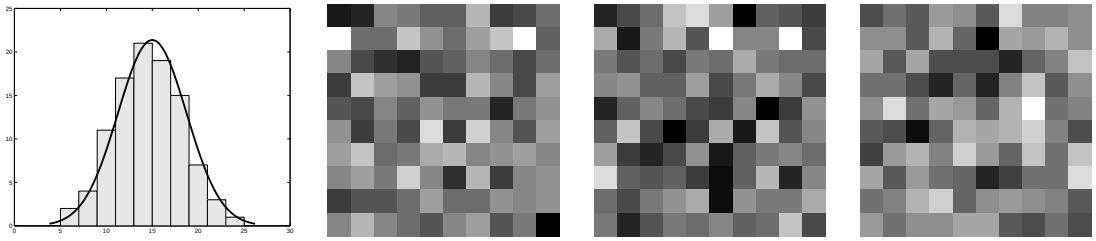
2 UDL monitoring scheme. A novel proposal

The usual topology-preservation metrics are not very sensitive to small topological defects. Furthermore, they require a lot of computational effort as a monitoring tool. In this paper we present a novel scheme that help us to find the appropriate neighbourhood range value to end with, and a monitoring process to derive 'long enough' training without human intervention. First, some functions are introduced.

The predictive log-likelihood is based on test data extracted from the same source but not used for training— a kind of cross-likelihood. This function is called here $LKtest$, that is,

$$LKtest(t) = \log \mathcal{L}(t) \tag{1}$$

where $\mathcal{L}(t)$ is the value of the likelihood on a test set in training cycle t. $LKtest$ function presents some difficulties when used as a monitoring tool during the learning process. First, a representative test set is not always easy to find in real-world examples. Second, the estimated



a b c d

FIG. 1 – (a) Data load distribution in the equiprobabilistic case : 1,500 training items projected at random onto a 10×10 map. (b,c,d) Three examples of 10×10 DL-matrices coming from a truly equiprobabilistic map.

density function at each test vector has to be computed at every cycle of training, leading to a computationally expensive monitoring process.

The dataload is the number of data vectors projecting onto each trained unit. Dataloads are the natural estimate of the probability of activation given new data generated by the sampling distribution [Bodt et al. (1997)], that is,

$$dl_i = \int_{V_i} p(\mathbf{v}) d(\mathbf{v}), \tag{2}$$

where V_i collects all input vectors which are closest to weight vector $\mathbf{w}_i$.

In the truly equiprobabilistic case the dataload vector should be distributed as a multinomial with equal component probabilities, since the pointer density in the trained equiprobabilistic map serves as an estimate of the density underlying the data— each neuron would cover about the same proportion of the data. In fact, it would follow asymptotically a joint multivariate Gaussian distribution with correlations getting weaker with increasing sample size N. For moderate N, the equiprobabilistic dataload histogram should thus correspond to a common limiting univariate Gaussian. Hence, the mean is $\frac{N}{M}$, whereas $\sqrt{\frac{N}{M}(1-\frac{1}{M})}$ is the standard deviation— it may be a bit smaller in practice due to the finite-sample correction. Note that these statistics depend only on lattice size and training sample size, they are otherwise universal over data sets. To easily visualize the dataload distribution over the map, a gray image is computed, namely, the DL-matix, where darker means higher.

A first implication of this limiting normality is that, due to the linear nature of the transformation into the gray scale, the ideal DL-matrix from Figure 1a has many more intermediate gray levels than either pure white or black cells, i.e., the distribution is not really uniform in the gray scale. Some examples are shown in Figure 1(b,c,d) : no patterns can be discerned, and just a few very light or very dark cells stand out. Of course, there are also formal tests of normality that can be applied to the DL vector directly. Yet, we believe graphical diagnostics incorporating lattice information, such as the DL-matrix, may be more suggestive for interactive analysis.

The dataload standard deviation at each iteration, namely SD function, will be the required function for our monitoring process. Unlike the $LKtest$ function, SD is cheap and easy to obtain.

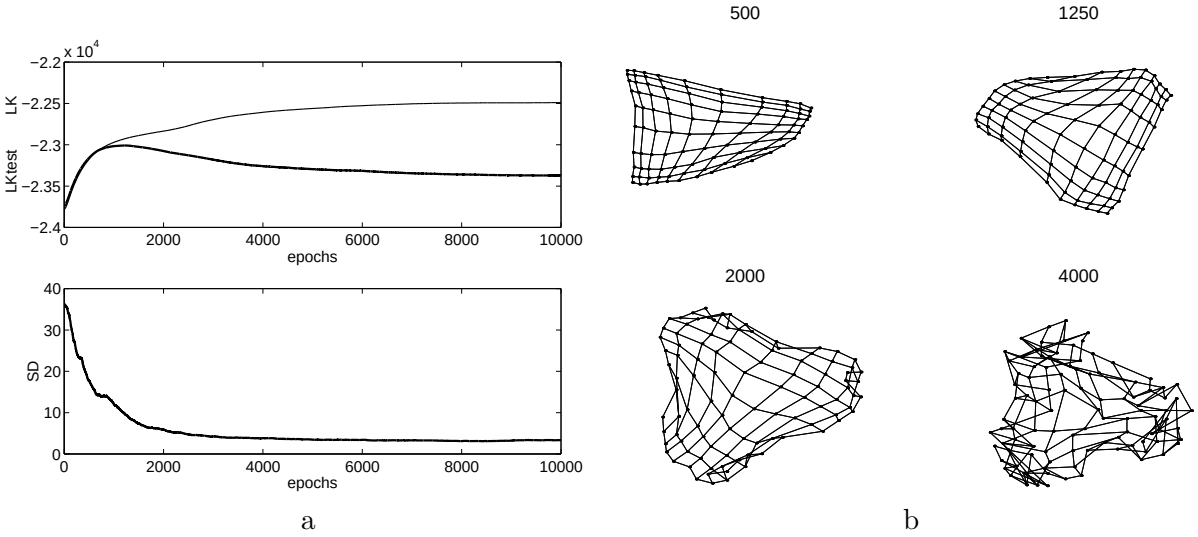


FIG. 2 – kMER on 3M-10D data set with a long training of 10,000 epochs. (a) 10×10 map showing LKtest (thick solid line on the top plot), LKtrain (thin solid line on the top plot) and SD (bottom plot) functions; (b) 10×10 Sammon's projected map on the iterations number 500, 1250 (the highest LKtest value), 2000 and 4000.

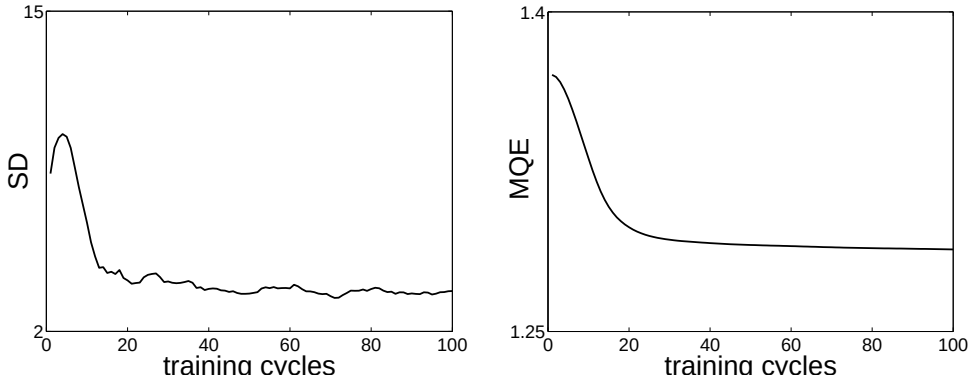
2.1 A stopping policy

When training the map, there is a moment in which neurons begin to learn more about the concrete train set, adding noise to the final estimated distribution. Thus, overtraining problems have to be avoided. Therefore, the aim of the stopping process is to find the map with maximum predictive likelihood in which overtraining has not appeared yet and the resulting neurons' distribution provides the map with the highest likelihood. This map will be the one that explains in the best way the real population.

At this moment of maximum predictive likelihood, topographic order is maintained. Furthermore, the worse the topographic order the lower the predictive likelihood. In Figure 2 overtraining problems in kMER learning rule are illustrated. The data set considered here consists of three Gaussians living in a 10 dimensional space— a data set referred to as 3M-10D. In this data set 10,000 cycles of training are supposed to be long enough.

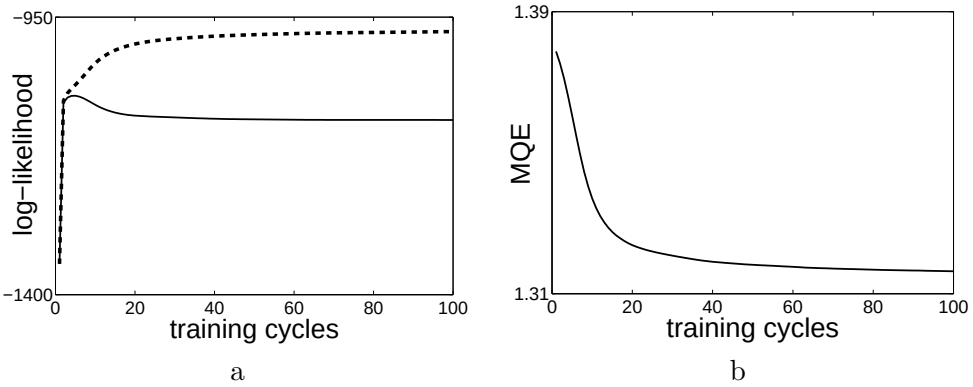
Figure 2a top plot highlights how, while log-likelihood over the train set LK (thin solid line) is a monotonically increasing function, the log-likelihood over a test set $LKtest$ (thick solid line) has a completely different behaviour. $LKtest$ is an increasing function in its initial stages, reaches a maximum, and begin to decrease until the end of training. Sammon projected maps at different moments of training are shown in Figure 2b. At cycle $t = 500$ the map is well organized but it does not reflect properly the underlying distribution (note that at $t = 500$, LK-like functions have lower values). At cycle $t = 1250$, which corresponds to the maximum predictive likelihood, the map is completely disentangled. From this moment on, overtraining problems arise, since the learning process stresses local regions instead of global relationships, and this is reflected on both the loss of organization and a lower predictive likelihood value.

Hence, Figure 2 shows the link between maximum $LKtest$ and topology preservation.



a b

FIG. 3 – Learning trajectories of GTM on 1M-50D data set. (a) SD ; (b) MQE.



a b

FIG. 4 – GTM on 2M-10D data set. (a) Log-likelihood using training data (dotted line) and test data (solid line) ; (b) MQE.

Consequently, the neighbourhood value at where the maximum of $LKtest$ function is reached is the one desired to end the training process with. This value, called here σ_{Λ}^{lkt}, is the optimal one, provided that an infinitely slow training (that begins with an unfolded map) is developed. Algorithm 1 shows this stopping policy.

As pointed before, evaluating $\log \mathcal{L}$ at each cycle of training is enormously time consuming, doing it a hard task with low dimensional data and impossible in higher dimensions—as real cases use to be. At this point, a heuristic is needed.

3 UDL stopping policy

Here we propose a novel monitoring criterion based on the uniformity of the dataload vector, a criterion called UDL— Uniform DataLoad vector [Vegas-Azcárate and Muruzábal (2005); Muruzábal and Vegas-Azcárate (2005)]. Specifically, our UDL criterion looks for the moment at which the speed of decrease of the dataload standard deviation function is nearly zero.

As highlighted before, pointer density in the trained equiprobabilistic map serves as an

Algorithm 1
Ideal stopping policy.

Obtain a disentangled lattice to be used to initialize neurons' map— train a random set of neurons over a finite number of cycles with a fixed range.
repeat for each iteration, t,
    Train the map.
    Evaluate $\log \mathcal{L}$ and store its value in $LKtest(t)$.
until t is long enough.
Determine the number of epochs, t_{lkt}, and the corresponding range, σ_{Λ}^{lkt}, for which $LKtest(t)$ reaches its maximum.
Select the map obtained at t_{lkt}.

Algorithm 2
The new proposed UDL stopping policy.

Obtain a disentangled lattice to be used to initialize neurons' map.
repeat for each iteration, t,
    Train the map.
    Obtain the dataloads and store its standard deviation in $SD(t)$.
until t is long enough.
Determine the number of epochs, t_{udl}, and the corresponding range, σ_{Λ}^{udl}, for which $SD(t)$ reaches its stability.
Select the map obtained at t_{udl}.

estimate of the density underlying the data. Thus, each neuron would cover about the same proportion of the data, leading to a uniform dataload vector. It appears that the stochastic Gaussian behaviour in the equiprobabilistic case can be approximately detected when it is first reached [Vegas-Azcárate and Muruzábal (2005); Muruzábal and Vegas-Azcárate (2005)]. Hence, the associated UDL stopping rule could be stated as follows : quit as soon as the trained map shows the first signs of having reached the reference Gaussian DL distribution— a moment referred to as the UDL stage. In other words, quite as soon as SD function reaches its stability.

Furthermore, minor gains in quantization error brought by training beyond the UDL stage seem to enforce the loss of useful organization, implying fuzzier displays for analysis. Indeed, the UDL stage also signals approximately the beginning of the fine-tuning phase in quantization error [Vegas-Azcárate and Muruzábal (2005); Muruzábal and Vegas-Azcárate (2005)]. An example is given in Figure 4, where the test log-likelihood function reaches its optimum at a very early stage of training. Notice that the MQE trajectory in Figure 4b is closely related to the training log-likelihood trajectory. In particular, it can be appreciated how most progress in quantization error has been done at a relatively early stage of training. GTM learning trajectories on 1M-50D (see Figure 3) show that SD and MQE functions settle around the same reference value.

Algorithm 3
UDL monitoring scheme.

Train the map with a constant neighbourhood range.
Store the obtained disentangled lattice, Q^0.
Starting from Q^0, perform one complete training with $\gamma^1 = 1$.
Determine the number of epochs, t^1_{udl}, and the corresponding range, $\sigma^1_{\Lambda,udl}$, for which the speed of decrease of SD^1 function is nearly zero.
repeat, for each monitoring run, $j > 1$,
Perform a new training, starting from Q^0, with $T^j = 2 \cdot t^{j-1}_{udl}$ and

$$\gamma^j = -\frac{\ln \frac{0.9 \cdot \sigma^{j-1}_{\Lambda,udl}}{\sigma_{\Lambda(0)}}}{2 \cdot \sigma_{\Lambda(0)}} \tag{3}$$

to cool at a slower rate, but only run the simulation as long as necessary. Determine the epoch, t^j_{udl}, and the range, $\sigma^j_{\Lambda,udl}$, for which the speed of decrease of SD^j function is nearly zero.
until $\sigma^j_{\Lambda,udl} \simeq \sigma^{j-1}_{\Lambda,udl}$.
Do a complete run with $T_{udl} = t^j_{udl}$ and $\gamma_{udl} = \gamma^j$.

Simulation experiments of Figures 5 and 6 reflect the concordance between σ^{lkt}_Λ and the neighbourhood range value at which SD reflects some kind of stability, called σ^{udl}_Λ. Then, the cycle at which the speed of decreasing of SD function goes to zero, t_{udl}, seems to be a good, easy and cheap heuristic estimate of t_{lkt}, and so σ^{udl}_Λ can be treated as a heuristic of σ^{lkt}_Λ. Algorithm 2 illustrates our UDL stopping policy.

Note that during the learning process, the best matching unit of each datum is calculated, so the overhead in obtaining the dataloads and its standard deviation is minimal. The problem now is to obtain t_{udl} or σ^{udl}_Λ, since a complete long enough training has to be done. Indeed, this is not possible with real-world data where 'long enough' is not known.

3.1 UDL monitoring scheme

To solve the 'long enough' problem, a scheme similar to the one presented in [Van Hulle (2000)] is developed here. In it, the rate at which the neighbourhood function range decreases is adjusted during training, ensuring that the final range value is the one desired, σ^{udl}_Λ. Hence, in a finite and affordable number of cycles the same optimal map as the one achieved in an infinitely slow training is obtained. The neighbourhood cooling scheme is developed following,

$$\sigma_\Lambda(t) = \sigma_\Lambda(0) \cdot \exp\left(-2 \cdot \sigma_\Lambda(0) \cdot \frac{t}{T_{udl}} \cdot \gamma_{udl}\right), \tag{4}$$

where γ_{udl} and T_{udl} provide the map that has a range value of σ^{udl}_Λ. Algorithm 3 reflects the proposed UDL monitoring scheme.

The monitoring processes converge, since in a slow enough training the neighbourhood range value at which SD function reaches its stability does exist [Vegas-Azcárate and Mu-

ruzábal (2005); Muruzábal and Vegas-Azcárate (2005)]. Hence, if the training algorithm employs a neighbourhood function to ensure topographic ordering, as in the case of SOM-like and kMER algorithms, it is possible to use the new UDL monitoring scheme presented in this paper.

4 A new way to select the point of SD stability

At each monitoring run, say j, we need to estimate t^j_{udl} and $\sigma^j_{\Lambda,udl}$ values, for which the speed of decrease of SD^j function is nearly zero. Note that T_{udl} and the corresponding σ^{udl}_{Λ} values can be selected just looking to SD function plot. An accepted value for T_{udl} is the one for which $SD(T_{udl}) \simeq SD(T_{udl-1})$, or another criterion of stability. However, no human guidance is sometimes preferred, so as in the previous case in which several runs are performed. Any procedure to detect function stability can be used at this point. Here is presented a novel methodology that estimates t^j_{udl} value during the monitoring run j automatically, providing satisfactory results [Vegas-Azcárate and Muruzábal (2003, 2005); Muruzábal and Vegas-Azcárate (2005)]. Note that knowing t^j_{udl} value, $\sigma^j_{\Lambda,udl}$ value is obtained following Equation 4.

First, plot SD function in a 'normalized' way— that its shape does not depend on nothing but general values, such as N or M. For example, with X-axis from 0 to T and Y-axis from 0 to the maximum possible value of any SD function. This value can be calculated as follows. Chose a large and fixed value for the neighbourhood range ; hence, during the training all neurons will be updated in the same way, collapsing in one concrete weight vector— let $\mathbf{w}_1$ be that one.

The dataload vector at convergence will be a $M \times 1$ vector, $DL = (dl_1, ..., dl_M)^t$, with $dl_1 = N$ and the rest of the $M - 1$ elements set to zero. The mean of the dataload vector is

$$\overline{DL} = \frac{\sum_{i=1}^{M} dl_i}{M} = \frac{N}{M} \tag{5}$$

and its quasi-standard deviation

$$qsd(DL) = \sqrt{\frac{\sum_{i=1}^{M} \left(dl_i - \overline{DL}\right)^2}{M-1}} = \sqrt{\frac{\left(N - \frac{N}{M}\right)^2 + (M-1)\left(\frac{N}{M}\right)^2}{M-1}} \leq \frac{N\sqrt{M}}{M}. \tag{6}$$

Then, SD never reaches a value larger than $\frac{N\sqrt{M}}{M}$. This way, Y-axis takes values from 0 to $\frac{N\sqrt{M}}{M}$, and X-axis from 0 to T, as pointed before. The bisector line from the origin to $(T, \frac{N\sqrt{M}}{M})$ will be called UDL-line. Algorithm 4 summarizes the presented procedure.

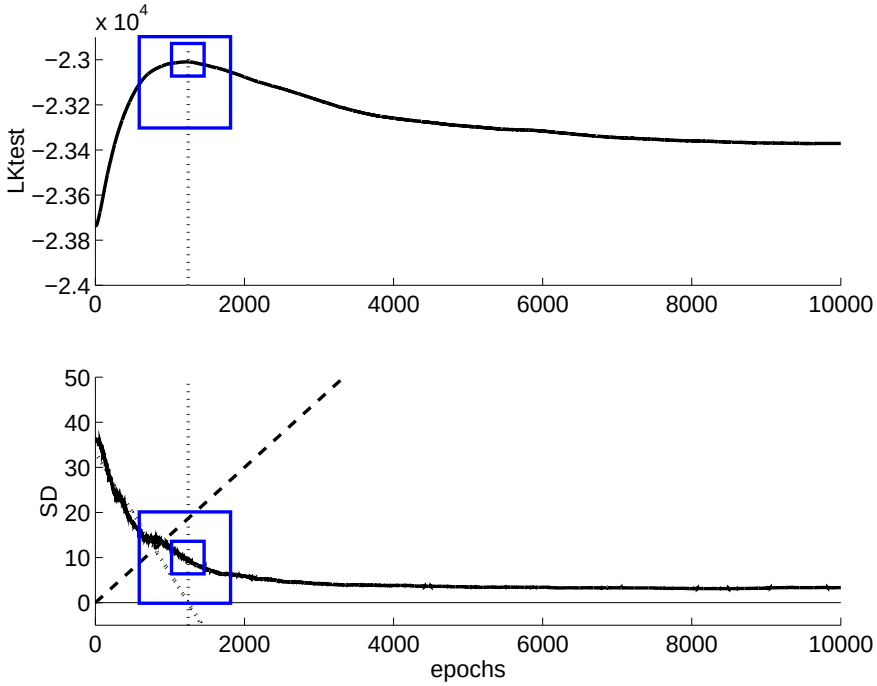
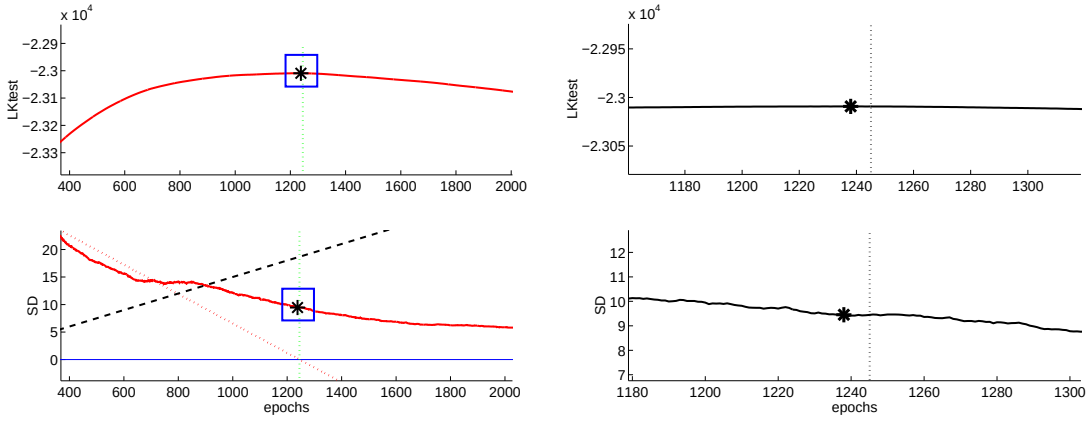


FIG. 5 – kMER on 3M-10D data set with a long training of 10,000 epochs. LKtest (top) and SD (bottom) with UDL-line (dashed), SD-line (thick dotted) and X-axis (thin solid).



a b

FIG. 6 – kMER on 3M-10D data set with a long training of 10,000 epochs. (a) Zooms of the biggest square area of Figure 5. (b) Zooms of the smallest square area of Figure 5, highlighting the maximum of $LKtest$ (asterisks).

Algorithm 4
A new way to select the point of SD stability.

Calculate UDL-line, from $(0,0)$ to $(T^j, \frac{N\sqrt{M}}{M})$.
Find the training cycle at which UDL-line intersects SD function, say t_{prev}.
Find the regression line of $\{(t, SD(t)) : t \in (1, ..., t_{prev})\}$, say SD-line.
The cycle of training where SD-line intersects X-axis is t^j_{udl}.

An example of how the monitoring process works is presented in Figures 5 and 6. Figure 5 top plot shows $LKtest$ function (solid line) of kMER learning rule ; a vertical (thin dotted) line identifies the number of epochs at which the speed of decrease of SD function is nearly zero (bottom plot), and two squares (the biggest and the smallest) indicate the area that will be highlighted on top plots of Figures 6a and 6b, respectively.

Figure 5 bottom plot displays SD function (thick solid line), UDL-line (that goes from $(0,0)$ to $(10000, 150)$, since $N = 1500$ and $M = 100$, in dashes), SD-line (thick dotted), X-axis (thin solid line), the vertical (thin dotted) line identifies the intersection between the SD-line and X-axis (a heuristic of the number of epochs at which the speed of decrease of SD function is nearly zero) and the two squares indicate the areas that will be zoomed on the bottom plots of Figures 6a and 6b, respectively.

An asterisk is placed on $LKtest(t_{lkt})$, that is, on the epoch of the highest predictive likelihood or maximum of $LKtest$ function. Figure 6b shows that $t_{lkt} = 1239$ and $t_{udl} = 1246$, hence t_{lkt} and t_{udl} are nearly the same value, with only 7 out of 10,000 epochs of difference. This way, the concordance between σ^{lkt}_{Λ} and σ^{udl}_{Λ} is straightforward. Moreover, the map recollected at cycle t_{udl} provides an unfolded map with the highest predictive log-likelihood. This way, an appropriate heuristic of the neighbourhood range value at where the predictive log-likelihood reaches it maximum is obtained following our UDL monitoring scheme.

5 Discussion

This paper present a novel monitoring approach founded on the tenets that the optimal map is the one obtained in an infinitely slow training— although good approximations can be done with large cycles of training when working with synthetic data, in higher real spaces this long training is not recommended, due to the enormous time consumed by the algorithms and the impossibility to know when a long enough training is achieved. Moreover, the neighbourhood function, specially the rate at which neighbourhood range is decreased over the finite simulation time, is crucial in the formation of topology-preserving maps— if the final value of this range is too large, neurons will not properly span within the input data set, but if it is too small, violations in the topographic order will occur. To the best of our knowledge, the new proposed UDL monitoring scheme is the first methodology that estimates the decrementing schedules for the neighbourhood range function during the training automatically.

Références

Bodt, E., Verleysen, M. and Cottrell, M. (1997). Kohonen maps versus vector quantization for data analysis, in *European Symposium on Artificial Neural Networks, ESANN'97*, pp. 211–220, Brussels : D-Facto publications.

Der, R. and Herrmann, M. (1993). Phase transitions in self-organizing feature maps, in *Proc. ICANN'93 (Amsterdam, The Netherlands)*, pp. 597–600, New York : Springer.

Der, R. and Herrmann, M. (1994). Inestabilities in self-organized feature maps with short neighborhood range, in *Proc. Eur. Symp. on Artificial Neural Networks - ESANN'94 (Brussels, Belgium)*, edited by Verleysen, M., pp. 271–276.

Haese, K. and Goodhill, G. J. (2001). Auto-SOM : Recursive parameter estimation for guidance of self-organizing feature maps, *Neural Computation*, **13** : 595–619.

Kohonen, T. (2001). *Self-Organizing Maps*, Berlin : Springer-Verlag, 3rd extended ed.

Mulier, F. and Cherkassky, V. (1995). Self-organization as an iterative kernel smoothing process, *Neural Computation*, **7** : 1165–1177.

Muruzábal, J. and Vegas-Azcárate, S. (2005). On equiprobabilistic maps and plausible density estimation, in *5th Workshop On Self-Organizing Maps, Paris*.

Van Hulle, M. M. (2000). *Faithful representations and topographic maps : From distortion- to information-based self-organization*, New York : Wiley.

Vegas-Azcárate, S. and Muruzábal, J. (2003). On the use of the GTM algorithm for mode detection, in *LNCS*.

Vegas-Azcárate, S. and Muruzábal, J. (2005). On cluster analysis via neuron proximity in monitored self-organizing maps, in *Workshop on Biosignal Processing and Classification, Barcelona, Spain*.