Nonvectorial kmer and topology preservation

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 - *In nonvectorial topographic maps the data sequences are not previously converted into histogram vectors, thus avoiding the shortcomings associated to these representations. Like in standard vectorial topographic maps, in nonvectorial learning algorithms the optimal speed of shrinking of the neighbourhood range should be experimentally determined. This paper shows how UDL monitoring scheme can be extended to the case of the kernel-based Maximum Entropy Learning Rule (kMER).*

Key words - Topographic maps, monitoring studies, mixture models.

Acknowledgement - *SVA gratefully acknowledge the financial support for this research given by MCYT, CICYT, Autonomous Community of Madrid, URJC, Complutense University of Madrid, DMR Consulting Foundation, European Science Foundation and TED Program.*

1 Introduction

Similarity and distance measures have been routinely used to compare two biological sequences, such as proteins or nucleic acids. The basis of such comparisons is the information from the biochemist as to the linear sequence of elements comprising such molecules [Smith and Waterman (1981)]. Similarity measures such as Smith-Waterman, BLAST or FASTA, are appropriate for clustering large protein sequence databases with topographic maps [Somervuo and Kohonen (2000)]. In nonvectorial topographic maps, unlike the previous vectorial ones, the data sequences are not converted into histogram vectors in order to perform the clustering.

2 The nonvectorial SOM

Kohonen and Somervuo (2002) have shown how to implement the SOM algorithm principle to nonvectorial data in the case of fixed-size standard maps. Interestingly, they have illustrated their method by using protein sequences as basic items and FASTA scores [Pearson and Lipman (1988); Pearson (1999)] as similarity values. Specifically, if x and y are any entities, a sufficient condition for them to be mapped into a SOM diagram is that some kind of symmetric distance function, $d(x, y)$, is definable for all pairs (x, y).

Furthermore, Kohonen and Somervuo (2002) have shown how this extension of the SOM, called here SOM-nv (see Algorithm 1), can be used for the clustering, organization and visualization of large databases of nonvectorial items such as protein sequences. The new method, originally suggested in Kohonen (1996), allows the construction of the SOM when only a similarity measure is defined for pairs of items. Hence, a vectorial representation is not really needed, avoiding the important drawbacks and limitations typically derived from the vectorial representation of biological data. To define an ordered projection, it will be sufficient to compare the pairwise distances or dissimilarities among items [Kohonen and Somervuo (2002)].

The nonvectorial SOM is based on the batch-learning version of the SOM, and it requires the computation of the *generalized median* of symbol strings [Kohonen (1985, 1995)]. Here, the way a winning neuron is selected is

$$i_n^* = \arg\min_i \{d[x_n, z_i]\} \tag{1}$$

where $d(\cdot, \cdot)$ is the underlying pseudo-distance measure. Notice that $\mathbf{v}_n$ and $\mathbf{w}_i$ were used to define input vectors and weight vectors, respectively, living in $\mathbb{R}^d$. Now, x_n and z_i term the items and the pointers, respectively, living in the symbolic (e.g. protein) space.

The generalized median is defined as follows [Kohonen (1985, 1995)]. Let $\Upsilon = \{x_n\}$ be a set of items, and let $d[x_n, x_{n'}]$ be some distance, pseudo-distance or dissimilarity measure between x_n and $x_{n'} \in \Upsilon$. The generalized median m over Υ is defined as the item that minimizes the sum of distances to all other items in Υ,

$$m = \arg\min_{x_n \in \Upsilon} \sum_{x_{n'} \in \Upsilon : n \neq n'} d[x_n, x_{n'}]. \tag{2}$$

This way, if the input samples had been real scalars and the distance measure were the absolute value of their difference, the generalized median would coincide with the usual arithmetic median.

Algorithm 1
The nonvectorial version of SOM algorithm (SOM-nv).

Initialize the map (see below).
repeat for each iteration, t,
  for each input sequence, x_n, **do**
    Find the best matching unit for x_n, see Equation 1.
  end for.
  Recollect in Z_i (see Equation 3) the input items associated to pointer z_i.
  Store in Ω_i (see Equation 4) the items associated to each pointer in its neighbourhood N_i.
  Update each z_i as the generalized median (see Equation 2) of Ω_i.
until $Z_i^t = Z_i^{t-1}$, $\forall i$.

Algorithm 2
Initialization method for SOM-nv in the case of proteins.

Convert input sequences into 400-dimensional dipeptide histogram vectors.
Provide each map neuron with a 400-dimensional vector.
repeat,
  Train a SOM-batch cycle,
until neurons are 2D ordered.
Label neurons by those proteins that represent the generalized medians of the sequences associated to them.

The main features of SOM-nv are now highlighted. To initialize the algorithm, auxiliary vectorial pointers are introduced. Indeed, the convergence of this algorithm is significantly faster and safer if the initial pointers are already two-dimensionally ordered [Kohonen and Somervuo (2002)]. In the case of proteins, these vectorial pointers can be selected as the usual 400-dimensional dipeptide histogram vectors [Ferrán and Ferrara (1991)]. Thus, each map node is provided with a 400-dimensional vector, each component of which is initialized with a random value between zero and unity— the whole vector is finally normalized to unit length. The standard SOM-batch algorithm is then trained with the dipeptide vectors, and the final pointers obtained are recoded to get nonvectorial SOM initialized. Specifically, for each vectorial pointer the usual subset of input items (including all items having that pointer as winner in the vectorial sense) is associated to it, and the corresponding nonvectorial pointer is chosen as the generalized median of that subset. With this labelling, a 2D set of relatively ordered input sequences is achieved, so that the nonvectorial SOM can proceed. From this point on, all vectorial representations are dropped. This initialization method for SOM-nv is summarized in Algorithm 2.

For each pointer z_i, two sets are then defined. First, one would recollect in Z_i the input items associated to it, i.e., the input items that have z_i as its best-matching unit. Winning neurons could be determined as usual according to the FASTA method, but note that an

input item could then have exactly the same distance to two or more pointers. Therefore, in order to make the winner unique in this case, one would ask the winner to minimize the sum of distances from the input to all pointers in a small neighbourhood around the winner candidate i, say N_i. This neighbourhood includes all pointers within a certain radius from node i on the grid. Like in the traditional SOM, this radius can shrink monotonically with time. Mathematically, $x_n \in Z_i$ if and only if

$$z_i = \arg\min_l \sum_{k \in N_l} d[x_n, z_k]. \tag{3}$$

Recollect now in Ω_i the input items associated to each pointer in N_i in the previous sense, that is,

$$\Omega_i = \bigcup_{k \in N_i} Z_k, \tag{4}$$

and update each z_i as the generalized median of Ω_i. Thus, this is called the adaptation process. For each new pointer z_i, recollect in Z_i the new input items associated to it as before. If the old Z_i, say Z_i^{t-1}, coincide with the new Z_i^t for all i, then the process has converged. If not, continue with the adaptation process. When convergence is reached, pointers approximate the input items in an orderly fashion, since each pointer coincides with the generalized median of the input items mapped onto its neighbourhood.

In this context, El Golli et al. (2004a,b) have proposed an extension of the standard Kohonen learning rule that can also handle symbolic data. Specifically, they have presented an adaptation of the SOM-batch to dissimilarity data. As in Kohonen and Somervuo (2002) work, the main difference with traditional SOM is that El Golli et al. (2004a) are not working on $\mathbb{R}^d$ but on an arbitrary set on which a dissimilarity is defined. The experiments in El Golli et al. (2004a,b) show the usefulness of their method applied to symbolic data.

3 The nonvectorial kMER

The nonvectorial kMER algorithm is based on the batch-learning version of kMER [Gautama and Van Hulle (2005)]. kMER-batch update rule is illustrated in Algorithm 3. As in the SOM-batch algorithm, in kMER-batch a learning rate function is no longer required, and training only depends on the neighbourhood range, σ_Λ.

As Gautama and Van Hulle (2005) have shown, the nonvectorial kMER, called here kMER-nv, can be derived from the kMER-batch learning rule by means of the generalized median of Equation 1, that is, the input item for which the distance to the other inputs items is the smallest (see Algorithm 4). Following the notation previously introduced, x_n and z_i are used for nonvectorial items and pointers, respectively, while $\mathbf{v}_n$ and $\mathbf{w}_i$ are used for vectorial data and weights, respectively, living in $\mathbb{R}^d$.

As in the nonvectorial SOM, all pointers coincide with actual input items, and the map is trained by means of only the matrix of pairwise dissimilarity measure. Notice that, while in SOM-batch a neuron is updated as the mean of its Voronoi region, in nonvectorial SOM and nonvectorial SOTA each neuron is updated by means of the generalized median of its neighbourhood.

Algorithm 3
The kMER batch version (kMER-batch).

Initialize the weight vectors, $\mathbf{w}_i^0$, and the radii, σ_i^0, of the receptive fields.
repeat for each iteration, t,
  for each data point, $\mathbf{v}_n$, **do**
    Find its best matching unit,
  end for.
  Update the weight vectors, $\mathbf{w}_i^t$, with the median of all data points, $\mathbf{v}_n$, weighted by

$$\sum_{j=1}^{M} \Lambda_{ij}^t \cdot \Xi_j(\mathbf{v}_n). \tag{5}$$

  Update the radii, σ_i^t, with the distance between $\mathbf{w}_i^t$ and the $\frac{\rho N}{M}$-th nearest data point— being ρ the scale factor.
  Decrease the width of the neighbourhood function.
until t reaches T.

Algorithm 4
The nonvectorial version of kMER algorithm (kMER-nv).

Initialize the pointers, z_i^0, and the radii, σ_i^0, of the receptive fields.
repeat for each iteration, t,
  for each data point, x_n, **do**
    Find its best matching unit, see Equation 1,
  end for.
  Update the weight vectors, z_i^t, with the generalized median (see Equation 2) of all items x_n, weighted by Equation 5.
  Update the radii, σ_i^t, with the distance between z_i^t and the $\frac{\rho N}{M}$-th nearest input item.
  Decrease the width of the neighbourhood function.
until t reaches T.

4 The UDL stopping policy

UDL criterion looks for the moment at which the speed of decrease of the dataload standard deviation function is nearly zero. Pointer density in the trained equiprobabilistic map serves as an 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

Algorithm 5
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

$$\gamma^j = -\frac{\ln \frac{0.9 \cdot \sigma^{j-1}_{\Lambda,udl}}{\sigma_{\Lambda(0)}}}{2 \cdot \sigma_{\Lambda(0)}} \tag{6}$$

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 $\gamma_{udl} = \gamma^j$.

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)]. 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. 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,

$$\sigma_\Lambda(t) = \sigma_\Lambda(0) \cdot \exp\left(-2 \cdot \sigma_\Lambda(0) \cdot \frac{t}{T_{udl}} \cdot \gamma_{udl}\right), \tag{7}$$

where γ_{udl} and T_{udl} provide the map that has a range value of σ^{udl}_{Λ}. Algorithm 5 reflects the proposed UDL monitoring scheme.

Algorithm 6
UDL monitoring scheme for nonvectorial kMER. Initialization.

Convert input sequences into auxiliar vectorial representation.
Provide each map neuron with the same vectorial representation.
repeat,
    Train the map with the vectorial kMER-batch and a constant range.
until neurons are 2D ordered.
Store the obtained disentangled lattice, say Q^0.
Label pointers, z_i, by those proteins that represent the generalized medians of the sequences associated to them.
Initialize the radii, σ_i^t, with the distance between z_i^t and the $\frac{\rho N}{M}$-th nearest input item.

Algorithm 7
UDL monitoring scheme for nonvectorial kMER. First run.

Select Q^0 as the initial neurons configuration.
repeat for each iteration, t,
    for each input sequence, x_n, **do**
        Find the best matching unit for x_n,
    end for
    Select the neighbourhood range function to be

$$\sigma_\Lambda(t) = \sigma_\Lambda(0) \cdot \exp\left(-2 \cdot \sigma_\Lambda(0) \cdot \frac{t}{T^1} \cdot \gamma^1\right), \tag{8}$$

    where $\gamma^1 = 1$ and $T^1 = \#neurons$.
    Update the pointers, z_i^t, with the generalized median of all input items x_n weighted by Equation 5.
    Update the radii, σ_i^t, with the distance between z_i^t and the $\frac{\rho N}{M}$-th nearest input item.
    Obtain the dataloads and store their standard deviation in $SD^1(t)$.
until $t = T^1$.
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.

Algorithm 8
UDL monitoring scheme for nonvectorial kMER. Monitoring runs.

repeat for each monitoring run, $j > 1$,
  Select Q^0 as the initial neurons configuration.
  repeat for each iteration, t,
    for each input sequence, x_n, **do**
      Find the best matching unit for x_n,
    end for.
    Select the neighbourhood range function to be

$$\sigma_\Lambda(t) = \sigma_\Lambda(0) \cdot \exp\left(-2 \cdot \sigma_\Lambda(0) \cdot \frac{t}{T^j} \cdot \gamma^j\right), \tag{9}$$

    where $T^j = 2 \cdot t_{udl}^{j-1}$ and

$$\gamma^j = -\frac{\ln \frac{0.9 \cdot \sigma_{\Lambda,udl}^{j-1}}{\sigma_{\Lambda(0)}}}{2 \cdot \sigma_{\Lambda(0)}}. \tag{10}$$

    Update the pointers, z_i^t, with the generalized median of all input items, x_n, weighted by Equation 5.
    Update the radii, σ_i^t, with the distance between z_i^t and the $\frac{\rho N}{M}$-th nearest input item.
    Obtain the dataloads and store its standard deviation in $SD^j(t)$.
  until $t = T^j$
  Determine the epochs, t_{udl}^j, and the range, $\sigma_{\Lambda,udl}^j$, for which the speed of decrease of SD^j function is nearly zero.
until $\sigma_{\Lambda,udl}^j \simeq \sigma_{\Lambda,udl}^{j-1}$.

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 Muruzábal (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 UDL monitoring scheme.

5 UDL monitoring scheme for nonvectorial kMER

This section shows how the novel UDL monitoring scheme can also be applied to nonvectorial algorithms, such kMER-nv. The UDL-monitored algorithms presented in this section have been successfully tested on the data sets considered in Vegas-Azcárate and Muruzábal (2005); Muruzábal and Vegas-Azcárate (2005). A similar approach can be derived to control the degree of topology preservation on nonvectorial kMER (see Algorithms 6, 7 and 8). This way, the goodness derived from UDL monitorized nonvectorial SOM is added to the advantages of training with kMER learning rule.

6 Discussion

A variety of neural networks have been used to organize protein sequences into clusters or families according to their sequence homologies. However, since the number and composition of the families are not known, the use of unsupervised learning algorithms, such as the SOM type algorithms, seems indeed very appropriate. The corresponding topological maps so obtained should be very useful in organizing large protein or DNA databases and for rapidly classifying new sequences. In contrast to earlier works, the extension of the SOM batch allows for the use of any similarity measure in sequences. The combination of the nonvectorial topographic maps with the previously presented UDL monitoring ideas is expected to be a helpful tool to deal with biological sequences.

Références

El Golli, A., Conan-Guez, B. and Rossi, F. (2004a). Self organizing map and symbolic data, *Journal of Symbolic Data Analysis*, **2**.

El Golli, A., Conan-Guez, B. and Rossi, F. (2004b). A self organizing map for dissimilarity data, *Classification, Clustering, and Data Mining Applications (Proceedings of IFCS 2004)*, pp. 61–68.

Ferrán, E. A. and Ferrara, P. (1991). Topological maps of protein sequences, *Biological Cybernetics*, **65** : 451–458.

Gautama, T. and Van Hulle, M. (2005). Bacth map extensions of the kernel-based maximum entropy learning rule, *To appear in IEEE Transactions on Neural Networks*.

Kohonen, T. (1985). Median strings, *Pattern Recognition Letters*, **3** : 309–313.

Kohonen, T. (1995). *Self-Organizing Maps*, Berlin : Springer-Verlag.

Kohonen, T. (1996). Self-organizing maps of symbol strings, in *Technical Report A42*, Laboratory of Computer and Information Science, Helsinki University of Technology, Finland.

Kohonen, T. and Somervuo, P. (2002). How to make large self-organizing maps for nonvectorial data, *Neural Networks*, **15** : 945–952.

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

Pearson, W. (1999). The FASTA program package, `ftp ://ftp.virginia.edu/pub/fasta`.

Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence comparison, in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, pp. 2444–2448.

Smith, T. and Waterman, M. (1981). Comparison of biosequences, *Advances in Applied Mathematics*, **2** : 482–489.

Somervuo, P. and Kohonen, T. (2000). Clustering and visualization of large sequence databases by mean of an extension of the self-organizing map, in *Proceedings of the Discovery Science*, edited by Arikawa, S. and Morishita, S., pp. 76–85, Berlin : Springer.

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