Neural Computation no. 2391R

Kernel-based topographic map formation by local density modeling

Marc M. Van Hulle

K.U.Leuven, Laboratorium voor Neuro- en Psychofysiologie
Campus Gasthuisberg, Herestraat 49, B-3000 Leuven, BELGIUM
E-mail: marc@neuro.kuleuven.ac.be

June 3, 2002

Rather than developing topographic maps with disjoint and uniform activation regions (Voronoi tessellation), such as in the case of the popular Self-Organizing Map (SOM) algorithm (Kohonen, 1995), and its adapted versions, algorithms have been introduced that can accommodate neurons with overlapping activation regions, usually in the form of kernel functions, such as radially-symmetric Gaussians. For these *kernel-based* topographic maps, as they are called, several learning principles and -schemes have been proposed (for a review, see Van Hulle, 2000). One of the earliest examples is the elastic net of Durbin and Willshaw (1987), which can be viewed as a Gaussian mixture density model, fitted to the data points by a penalized maximum likelihood term. The standard deviations of the Gaussians ("radii") are all equal, and are gradually decreased over time. More recently, Bishop *et al.* (1998) introduced the Generative Topographic Map, which is based on *constrained* Gaussian mixture density modeling, constrained since the Gaussians cannot move independently from each other (the map is topographic by construction). Furthermore, all Gaussians have an equal and fixed radius. Sum and co-workers (1997) maximized the correlations between the activities of neighboring lattice neurons.

The radii of the Gaussians are under external control, and are gradually decreased over time. Graepel and co-workers introduced the Soft Topographic Vector Quantization (STVQ) algorithm, and showed that a number of probabilistic, SOM-related topographic map algorithms can be regarded as special cases (Graepel *et al.*, 1997). The Gaussian kernel represents a fuzzy membership (in clusters) function, and its radius, which is equal for all neurons, is again under external control (deterministic annealing). In the kernel-based Maximum Entropy learning Rule (kMER) (Van Hulle, 1998), the outputs of the Gaussians are thresholded, and the radii individually adapted so as to make the neurons (supra-threshold) active with equal probabilities (equiprobabilistic maps). In the Kernel-based Soft Topographic Mapping (STMK) algorithm (Graepel *et al.*, 1998), a non-linear transformation is introduced that maps the inputs to a high-dimensional "feature" space, and that, in addition, admits a kernel function — an idea borrowed from Support Vector Machines (SVMs) (Vapnik, 1995; Schölkopf *et al.*, 1999). The kernel operates in the original input space but its parameters are not modified by the algorithm. This connection with SVMs was taken up again by András (2001), but with the purpose of linearizing the class boundaries. The kernel radii are adapted individually so that the map's classification performance is optimized (using supervised learning). Yin and Allinson (2001) proposed an algorithm aimed at minimizing the Kullback-Leibler divergence (also called relative- or cross-entropy) between the true and the input density estimate obtained from the individually-adapted Gaussian kernels in the topographic map. In this contribution, we propose a still different approach: the Gaussian kernels are adapted individually, and in an unsupervised manner, but in such a way that their centers and radii correspond to those of the assumed Gaussian local input densities.

Let A be a lattice of N formal neurons and $V \subseteq \mathcal{R}^d$ the input space. To each

neuron $i \in A$ corresponds a weight vector $\mathbf{w}_i = [w_{i1}, \ldots, w_{id}] \in V$ and a lattice coordinate $\mathbf{r}_i \in V_A$, with V_A the lattice space (we assume discrete lattices with regular topologies). As an input to lattice space transformation, we take one which admits a kernel function: $\langle \Psi(\mathbf{v}), \Psi(\mathbf{w}_i) \rangle = K(\mathbf{v}, \mathbf{w}_i)$, with $\mathbf{v} \in V$, Ψ the transformation, and $\langle ., . \rangle$ the internal product operator in V_A. We prefer to use Gaussian kernels:

$$K(\mathbf{v}, \mathbf{w}_i, \sigma_i) = \exp\left(-\frac{\|\mathbf{v} - \mathbf{w}_i\|^2}{2\sigma_i^2}\right). \tag{1}$$

When performing topographic map formation, we require that the weight vectors are updated so as to minimize the expected value of the squared Euclidean distance $\|\mathbf{v} - \mathbf{w}_i\|^2$ and, hence, following our transformation Ψ, we instead wish to minimize $\|\Psi(\mathbf{v}) - \Psi(\mathbf{w}_i)\|^2$, which we will achieve by performing gradient descent with respect to $\mathbf{w}_i$. We first determine the gradient:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}_i}\|\Psi(\mathbf{v}) - \Psi(\mathbf{w}_i)\|^2 &= \frac{\partial}{\partial \mathbf{w}_i}K(\mathbf{w}_i, \mathbf{w}_i, \sigma_i) + \frac{\partial}{\partial \mathbf{w}_i}K(\mathbf{v}, \mathbf{v}, \sigma_i) - 2\frac{\partial}{\partial \mathbf{w}_i}K(\mathbf{v}, \mathbf{w}_i, \sigma_i) \\ &= -2\frac{\partial}{\partial \mathbf{w}_i}\exp\left(-\frac{\|\mathbf{v} - \mathbf{w}_i\|^2}{2\sigma_i^2}\right). \end{aligned} \tag{2}$$

Hence, the learning rule for the kernel centers $\mathbf{w}_i$ becomes (without neighborhood function — see further):

$$\Delta \mathbf{w}_i = \eta_w \frac{(\mathbf{v} - \mathbf{w}_i)}{\sigma_i^2} K(\mathbf{v}, \mathbf{w}_i, \sigma_i), \tag{3}$$

with η_w the learning rate. The equilibrium weight vector can be written as:

$$\mathbf{w}_i^{eq} = \frac{\int_V \mathbf{v}\ K(\mathbf{v}, \mathbf{w}_i^{eq}, \sigma_i)\ p(\mathbf{v})\ d\mathbf{v}}{\int_V K(\mathbf{v}, \mathbf{w}_i^{eq}, \sigma_i)\ p(\mathbf{v})\ d\mathbf{v}}, \quad \forall i \in A, \tag{4}$$

i.e., the center of gravity of $K(.)\,p(.)$, or, when we define the product of the input density and the Gaussian kernel as a new, local input density $p^*(\mathbf{v}) = K(\mathbf{v}, \mathbf{w}_i^{eq}, \sigma_i)\,p(\mathbf{v})$ — note that p^* is normalized by the denominator in eq. (4) —, then we can re-write the equilibrium weight vector as:

$\mathbf{w}_i^{eq} = \langle \mathbf{v} \rangle_{p^*}$. In order to achieve a topology-preserving mapping, we multiply the right hand side of eq. (3) with a neighborhood function $\Lambda(i, i^*)$, with $i^* = \arg\max_{\forall i \in A} (K(\mathbf{v}, \mathbf{w}_i, \sigma_i))$, *i.e.*, an *activity*-based definition of "winner-takes-all", rather than a minimum *Euclidean distance*-based definition ($i^* = \arg\min_i \|\mathbf{w}_i - \mathbf{v}\|$), which is usually adopted in topographic map formation. The equilibrium weight vector now becomes:

$$\mathbf{w}_i^{eq} = \frac{\int_V \mathbf{v}\ \Lambda(i, i^*)\ K(\mathbf{v}, \mathbf{w}_i^{eq}, \sigma_i)\ p(\mathbf{v})\ d\mathbf{v}}{\int_V\ \Lambda(i, i^*)\ K(\mathbf{v}, \mathbf{w}_i^{eq}, \sigma_i)\ p(\mathbf{v})\ d\mathbf{v}}, \quad \forall i \in A. \tag{5}$$

We now derive the learning rule for the kernel radii σ_i. We consider two steps. First, we perform gradient descent on $\|\Psi(\mathbf{v}) - \Psi(\mathbf{w}_i)\|^2$, but now with respect to σ_i, so that, after some algebraic manipulations:

$$\Delta\sigma_i \propto \frac{\|\mathbf{v} - \mathbf{w}_i\|^2}{\sigma_i^3} K(\mathbf{v}, \mathbf{w}_i, \sigma_i), \quad \forall i \in A. \tag{6}$$

Second, we wish each radius σ_i to correspond to the standard deviation of a d-dimensional Gaussian input distribution centered at the (equilibrium) weight vector of neuron i so that, together with the latter, the assumed Gaussian local input density is modeled. Now since the expected value of the Euclidean distance to the center of a d-dimensional Gaussian can be approximated as $\sqrt{d}\sigma$ for d large (Graham *et al.*, 1994),[1] the radius update rule becomes (without neighborhood function):

$$\Delta\sigma_i = \eta_\sigma \frac{1}{\sigma_i} \left(\frac{\|\mathbf{v} - \mathbf{w}_i\|^2}{\sigma_i^2} - \rho d \right) K(\mathbf{v}, \mathbf{w}_i, \sigma_i), \quad \forall i \in A, \tag{7}$$

with η_σ the learning rate. The scale factor ρ (a constant) is designed to relax the local Gaussian (and d large) assumption in practice. The equilibrium condition for the kernel radii can be written as:

$$\sigma_i^{2eq} = \frac{1}{\rho d} \frac{\int_V \|\mathbf{v} - \mathbf{w}_i\|^2\ K(\mathbf{v}, \mathbf{w}_i, \sigma_i^{2eq})\ p(\mathbf{v})\ d\mathbf{v}}{\int_V K(\mathbf{v}, \mathbf{w}_i, \sigma_i^{2eq})\ p(\mathbf{v})\ d\mathbf{v}}, \quad \forall i \in A, \tag{8}$$

[1] Note that the Euclidean distance distribution is chi-squared with $\theta = 2$ and $\alpha = \frac{d}{2}$ degrees of freedom.

i.e., the moment of inertia of p^* (without the $\frac{1}{\rho d}$ term), or for short: $\sigma_i^{2eq} = \frac{1}{\rho d}\langle \|\mathbf{v} - \mathbf{w}_i\|^2 \rangle_{p^*}$. Finally, similar to the weight update rule, we multiply the right hand side of eq. (7) with our neighborhood function $\Lambda(.)$. The corresponding equilibrium radius then becomes:

$$\sigma_i^{2eq} = \frac{1}{\rho d} \frac{\int_V \Lambda(i, i^*)\ \|\mathbf{v} - \mathbf{w}_i\|^2\ K(\mathbf{v}, \mathbf{w}_i, \sigma_i^{2eq})\ p(\mathbf{v})\ d\mathbf{v}}{\int_V \Lambda(i, i^*)\ K(\mathbf{v}, \mathbf{w}_i, \sigma_i^{2eq})\ p(\mathbf{v})\ d\mathbf{v}}, \quad \forall i \in A. \tag{9}$$

Equations (5) and (9) are in the form in which the so-called *iterative contraction mapping* (fixed-point iteration), used for solving non-linear equations, can be directly applied (thus without learning rates).[2] Such an iterative process is, at least for the weights, reminiscent of the so-called "Batch Map" version of the SOM algorithm (Kohonen, 1995; Mulier and Cherkassky, 1995). This observation leads us to the Soft Topographic Vector Quantization (STVQ) algorithm (Graepel *et al.*, 1997), a general model for probabilistic, SOM-based topographic map formation, since several algorithms can be considered as special cases, including Kohonen's Batch Map version. In Appendix 1, we detail the correspondence of our learning scheme, which we further call the Local Density Estimation (LDE) algorithm, with the STVQ, the Batch Map, and also the Kernel-based Soft Topographic Mapping (STMK) algorithm (Graepel *et al.*, 1998).

As an example, consider the standard case of a 10×10 lattice, trained on samples drawn from the uniform distribution in the unit square $(0, 1]^2$. The weights are initialized by sampling the same distribution; the radii are initialized by randomly sampling the $(0, 1]$ range. We use a Gaussian neighborhood function: $\Lambda(i, i^*, \sigma_\Lambda) = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_{i^*}\|^2}{2\sigma_\Lambda^2}\right)$, with σ_Λ the neighborhood radius,

[2] Note that, strictly speaking, since we opt for a "winner-takes-all" rule, for computational reasons, we should integrate over the subspace of V that leads to an update of neuron i's kernel parameters.

and $\mathbf{r}_i$ neuron i's lattice coordinate. We adopt the following neighborhood cooling scheme: $\sigma_\Lambda(t) = \sigma_{\Lambda 0} \exp\left(-2\sigma_{\Lambda 0}\frac{t}{t_{max}}\right)$, with t the present and t_{max} the maximum number of time steps, and $\sigma_{\Lambda 0}$ the radius at $t = 0$. We take $t_{max} = 500,000$, $\sigma_{\Lambda 0} = 5$, $\eta_w = 0.01$ and $\eta_\sigma = 0.02\eta_w$, and $\rho = 0.4$. The results are shown in Fig. 1. We observe that, prior to the lattice disentangling phase, the kernel radii grow rapidly in size and span a considerable part of the input distribution. Furthermore, for larger ρ, the lattice disentangles more rapidly, and the radii at convergence are larger. Hence, what happens in case the radii are forced to stay small: will the lattice still disentangle? This is exemplified in Fig. 2 for the same initial weight distribution, but with the radii kept constant. The lattice disentangles albeit at a much slower rate ($t_{max} = 2M$).

We will now explore the density estimation performance of our LDE algorithm by using samples drawn from the distribution shown in Fig. 3A (quadrimodal product distribution). We will also use this case as a benchmark for comparing the performances of two modified versions of our algorithm, and of four other kernel-based topographic map formation algorithms. The analytic equation of the product distribution is, in the first quadrant, $(-\log v_1)(-\log v_2)$, with $(v_1, v_2) \in (0, 1]^2$, and so on. Each quadrant is chosen with equal probability. The resulting asymmetric distribution is unbounded and consists of four modes separated by sharp transitions (discontinuities), which makes it difficult to model. The support of the distribution is bounded by the unit square $[-1, 1]^2$. We take a 24×24 lattice, and train it in the same way as in the previous example, but with $t_{max} = 1,000,000$, $\eta_w = 0.01$ and $\eta_\sigma = 0.1\eta_w$. The density estimate is obtained by taking the sum of all (equal-volume) Gaussian kernels at convergence, normalized so as to obtain a unit-volume density estimate. The result is shown in Fig. 3B. The Mean Squared Error (MSE) between the estimated and the theoretical distribution is 6.23×10^{-2}.

In order to explore the contribution of our activity-based competition rule, $i^* = \arg\max_i y_i$, we also train our lattice using the classic Euclidean distance-based rule, $i^* = \arg\min_i \|\mathbf{w}_i - \mathbf{v}\|$, but with the radii adapted as before, using eq. (7) ("LDE min Eucl" case in Table 1). The MSE is now 6.77×10^{-2}, which is inferior to our initial result. Furthermore, in order to see the effect of individually adapting the kernel radii, we also consider the case where the radii are equal and kept constant during learning ("LDE fixed radii" case). But here the following problem arises, as in other algorithms that do not adapt the kernel radii: how to choose this radius, since it directly determines the smoothness of the density estimate? In order not to have to rely on a separate optimal smoothness procedure, which could bias our results, we determine the (common) kernel radius for which the MSE between the estimated and the true, theoretical distributions is minimal. In this way, we at least know that a better MSE result cannot be obtained. The "optimal" MSE found is 9.78×10^{-2} for $\sigma_{\text{opt}} = 0.195$ (optimized in steps of 5×10^{-3}). The result is shown in Fig. 3C. This clearly shows the advantage of adapting the kernel radii individually to the local input density.

For the sake of comparison, we also consider four other kernel-based topographic map algorithms, provided that their kernels specify a density estimate in the input space directly. We use the following algorithms: the kernel-based Maximum Entropy learning Rule (kMER) (Van Hulle, 1998), the algorithm of Yin and Allinson (2001), and the STVQ and Soft-SOM (SSOM) algorithms (Graepel *et al.*, 1997). All simulations are run on the same input data, using the same cooling scheme (except for the SSOM and STVQ algorithms, see further), and the same number of iterations, as before. For the STVQ algorithm, we take for the neighborhood function radius $\sigma_\Lambda = 0.5$, and for the equal and constant kernel radii ("temperature parameter") $\frac{1}{\beta} = 0.01$, as suggested in

(Graepel *et al.*, 1997).[3] We also adopt these parameter values for the SSOM algorithm, since it is in fact a limiting case of the STVQ algorithm (see Appendix 1). Furthermore, again for the SSOM and STVQ algorithms, since they do not adapt their kernel radii, we look for the (common) kernel radius that optimizes the MSE between the estimated and the theoretical distributions, as explained above. The result for the STVQ algorithm is shown in Fig. 3D. Finally, we also consider the original SOM algorithm, add Gaussian kernels at the converged weight vectors, and look for the (common) kernel radius that minimizes the MSE. The results for these algorithms are summarized in Table 1, together with the parameters used.

The density estimation facility described above can be used for visualizing clusters in data distributions by performing density-based clustering in *lattice* space. For each neuron in the lattice, the density estimate at the neuron's weight vector is determined, and the result displayed graphically, on a relative scale, in the lattice. Clusters then correspond to high-density regions in the lattice, and they can be found and demarcated by a procedure such as hill-climbing (see caption of Fig. 3E,F). An efficient, tree-based hill-climbing algorithm is given in Appendix 2. This procedure could be regarded as an alternative to the gray level clustering procedure that has been devised for the SOM, and used for data mining purposes (Kohonen, 1995; Lagus and Kaski, 1999). Here, for each neuron, the average distance to the weight vectors of the neuron's lattice neighbors is determined, rather than a local estimate of the input density. Finally, density-based clustering itself can be regarded as an alternative to the (usually Euclidean) distance-based similarity criterion, that in most cases is adopted for clustering with competitive learning and topographic

[3] Note that our input distribution also spans the unit square $[-1, 1]^2$, as in (Graepel *et al.*, 1997).

maps (Luttrell, 1990; Rose *et al.*, 1990; Graepel *et al.*, 1997): distance-based clustering assumes that the cluster shape is hyperspherical, at least for the Euclidean case, whereas density-based clustering does not make any assumptions about the cluster shape.

Acknowledgments

M.M.V.H. is supported by research grants received from the Fund for Scientific Research (G.0185.96N), the National Lottery (Belgium) (9.0185.96), the Flemish Regional Ministry of Education (Belgium) (GOA 95/99-06; 2000/11), the Flemish Ministry for Science and Technology (VIS/98/012), and the European Commission, 5th framework programme (QLG3-CT-2000-30161 and IST-2001-32114).

References

András, P. (2001). Kernel-Kohonen networks. *Int. J. Neural Systems*, submitted.

Bishop, C.M., Svensén, M., and Williams, C.K.I. (1998). GTM: The generative topographic mapping. *Neural Computat.*, **10**, 215-234.

Durbin, R., and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, **326**, 689-691.

Graepel, T., Burger, M., and Obermayer, K. (1997). Phase transitions in stochastic self-organizing maps. *Physical Rev. E*, **56**(4), 3876-3890.

Graepel, T., Burger, M., and Obermayer, K. (1998). Self-organizing maps: Generalizations and new optimization techniques. *Neurocomputing*, **21**, 173-190.

Graham, R.L., Knuth, D.E., and Patashnik, O. (1994). Answer to problem 9.60 in *Concrete Mathematics: A Foundation for Computer Science.* Reading, MA: Addison-Wesley.

Kohonen, T. (1995). *Self-organizing maps.* Heidelberg: Springer.

Lagus, K., and Kaski, S. (1999). Keyword selection method for characterizing text document maps. *Proc. ICANN99, 9th Int. Conf. on Artificial Neural Networks*, IEE: London, Vol. 1, pp. 371-376.

Luttrell, S.P. (1990). Derivation of a class of training algorithms. *IEEE Trans. Neural Networks*, **1**, 229-232.

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

Rose, K., Gurewitz, E., and Fox, G.C. (1990). Statistical mechanics and phase transitions in clustering. *Phys. Rev. Lett.*, **65**(8), 945-948.

Schölkopf, B., Burges, C.J.C., and Smola, A.J. (1999). *Advances in kernel methods. Support vector learning.* Cambridge, MA: MIT Press.

Sum, J., Leung, C.-S., Chan, L.-W., and Xu, L. (1997). Yet another algorithm which can generate topography map. *IEEE TNNS*, **8**(5), 1204-1207.

Van Hulle, M.M. (1998). Kernel-based equiprobabilistic topographic map formation. *Neural Computat.*, **10**(7), 1847-1871.

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

Vapnik, V.N. (1995). *The nature of statistical learning theory.* New York: Springer-Verlag.

Yin, H., and Allinson, N.M. (2001). Self-organizing mixture networks for probability density estimation. *IEEE Trans. Neural Networks*, **12**, 405-411.

Appendix 1: Correspondence with STVQ, SOM and STMK algorithms

The Soft Topographic Vector Quantization (STVQ) algorithm (Graepel *et al.*, 1997) performs a fuzzy assignment of data points to clusters, whereby each cluster corresponds to a single neuron. It also serves as a general model for probabilistic, SOM-based topographic map formation. The weight vectors represent the cluster centers, and they are determined by iterating the following equilibrium equation (put into our format):

$$\mathbf{w}_i^{eq} = \frac{\int_V \mathbf{v} \sum_j \Lambda(i,j) P(\mathbf{v} \in \mathcal{C}_j)\ p(\mathbf{v})\ d\mathbf{v}}{\int_V \sum_j \Lambda(i,j) P(\mathbf{v} \in \mathcal{C}_j)\ p(\mathbf{v})\ d\mathbf{v}}, \quad \forall i \in A, \tag{10}$$

with $P(\mathbf{v} \in \mathcal{C}_j)$ the assignment probability of data point $\mathbf{v}$ to cluster $\mathcal{C}_j$ (*i.e.*, the probability of "activating" neuron j), which is given by:

$$P(\mathbf{v} \in \mathcal{C}_j) = \frac{\exp\left(-\frac{\beta}{2} \sum_k \Lambda(j,k)\ \|\mathbf{v} - \mathbf{w}_k\|^2\right)}{\sum_l \exp\left(-\frac{\beta}{2} \sum_k \Lambda(l,k)\ \|\mathbf{v} - \mathbf{w}_k\|^2\right)}, \tag{11}$$

with β the inverse temperature parameter, and $\Lambda(i,j)$ the transition probability of the noise-induced change of data point $\mathbf{v}$ from cluster $\mathcal{C}_i$ to $\mathcal{C}_j$. A number of topographic map algorithms can be considered as special cases by putting $\beta \to \infty$ in eq. (11), and $\Lambda(i,j) = \delta_{ij}$ in eq. (11) and/or eq. (10). For example, the Soft-SOM (SSOM) algorithm (Graepel *et al.*, 1997) is obtained by putting $\Lambda(i,j) = \delta_{ij}$ in eq. (11), but not in eq. (10). Kohonen's Batch Map version (Kohonen, 1995; Mulier and Cherkassky, 1995) is obtained for $\beta \to \infty$ and $\Lambda(i,j) = \delta_{ij}$ in eq. (11), but not in eq. (10), and for $i^* = \arg\min_j \|\mathbf{v} - \mathbf{w}_j\|^2$ (*i.e.*, distance-based "winner-takes-all" rule).

Our LDE algorithm is different from the STVQ algorithm in three ways. First, in the STVQ algorithm, the "kernel" $P(.)$ in eq. (11) represents a *fuzzy membership (in clusters) function*, *i.e.*, the softmax function, normalized with respect to the other neurons in the lattice, with the degree of fuzzification depend-

ing on β. In our case, the kernel $K(.)$ operates in the input space, instead of the (discrete) lattice space, and represents a *local density estimate*.[4] Our algorithm is also different from Kohonen's Batch Map by the definition of the kernel, which is in Kohonen's case the neighborhood function Λ, whereas in our case we have both $K(.)$ and $\Lambda(.)$, and by the definition of the winner i^* (distance- *vs.* activity-based). Second, instead of using "kernels" $P(.)$ with equal radii $\frac{1}{\beta}$, with β externally controlled (deterministic annealing or "cooling"), our radii differ from one another and are individually adapted. Third, the kernels also differ conceptually since, in the STVQ algorithm, the kernel radii are related to the magnitude of the noise-induced change in the cluster assignment (thus, in lattice space), whereas in our case, they are related to the standard deviations of the local input density estimates (thus, in input space).

In the Kernel-based Soft Topographic Mapping (STMK) algorithm (Graepel *et al.*, 1998), a non-linear transformation Ψ, from the original input space V to some "feature" space $\mathcal{F}$, is introduced that admits a kernel function: $\langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle = K(\mathbf{x}, \mathbf{y})$, with $K(.)$, *e.g.*, a Gaussian, and with $\langle ., . \rangle$ the internal product operator in $\mathcal{F}$-space. The topographic map's parameters ("weights") are expressed in this feature space, as linear combinations of the transformed inputs: $\mathbf{w}_i = \sum_{\mu=1}^{M} a_{\mu i} \Psi(\mathbf{v}^\mu)$, given a batch of M input samples $\{\mathbf{v}^\mu\}$. The coefficients $a_{\mu i}$ are determined by iterating an equilibrium equation. Furthermore, as in the STVQ algorithm, soft assignments of the inputs to clusters are made, $P(\mathbf{v}^\mu \in \mathcal{C}_j)$.

Our algorithm differs in several ways from the STMK algorithm. First, the topographic map's parameters $a_{\mu i}$ are developed in the feature space $\mathcal{F}$, rather than in the input space V, as in our LDE algorithm. Second, the STMK algo-

[4] Technically, the "kernel" in the STVQ algorithm represents a probability distribution, whereas our kernel represents a probability density.

rithm does not update the kernels' parameters. In fact, when initializing the STMK algorithm, the kernel-transformed data $\{K(\mathbf{v}^\mu, \mathbf{v}^\nu)\}$ are determined, and they replace the original input data (see p. 182 in Graepel *et al.*, 1998). In our LDE algorithm, both the kernel centers and the kernel radii are updated during learning. Third, the (transformed) inputs are assigned in probability to clusters in the STMK algorithm, whereas in our case, the (original) inputs are assigned with an activity-based "winner-takes-all" rule.

Appendix 2: Tree-based hill-climbing algorithm

Conventions: ID= lattice number of a neuron; Top= lattice number of the neuron which has the highest density estimate of all k+1 neurons in the current hypersphere; Ultimate Top= lattice number of the neuron which represents a local maximum in the input density.

```
/* Three types of nodes:
leaves :  (0, ID, successor, top)
nodes :  (predecessor, ID, successor, top)
tops :  (predecessor, ID, ID, ID)/*

/* Initialize all neurons, by labeling them all as leaves */
for(i ⇐ 1; i ≤ N; i ⇐ i + 1)
    label[i] ⇐ (0, i, 0, 0)
/* Search for all neurons the top of the cluster they belong to */
/* and with respect to the k nearest neighbors (parameter)*/
for(i ⇐ 1; i ≤ N; i ⇐ i + 1)
    if(label[i].successor == 0)
        /* Neuron pointer is not processed until now */
        pointer ⇐ i
        /* pointer will point to the neuron currently being processed */
        while(label[pointer].ID != label[pointer].successor)
                top ⇐ MaxNeighbor(pointer, k)
                label[pointer].successor ⇐ top
                if(top != pointer)
                   /* pointer is not ultimate top-neuron of cluster */
                   label[top].predecessor ⇐ pointer
                   pointer ⇐ top
```

```
else
    label[top].top ⇐ top
if(label[pointer].successor != 0)
    /* Current neuron is processed and leads to a top */
    /* No further processing is needed ⇒ quit while-loop */
    /* First give all parsed neurons ID of ultimate top */
    top ⇐ label[pointer].top
    while(label[pointer].predecessor != 0)
            pointer ⇐ label[pointer].predecessor
            label[pointer].top ⇐ top
    break
```

Figure captions

Fig. 1: Evolution of a 24 $\times$ 24 lattice as a function of time. The circles demarcate the areas spanned by the standard deviations ("radii") of the corresponding Gaussian kernels. The boxes correspond to the range spanned by the uniform input density. The values given below the boxes represent time.

Fig. 2: Evolution when the radii are kept constant at a value of 0.1. Same conventions as in Fig. 1.

Fig. 3: (A-D) Density estimation with kernel-based topographic maps. Two-dimensional quadrimodal product distribution (A), and the density estimates obtained with our learning algorithm, when adapting the kernel radii (B), and when keeping them fixed (C), and the estimate obtained with the STVQ algorithm (D). The size of the lattices is 24$\times$24 neurons. Abbreviation: pd = probability density.

(E,F) Hill-climbing procedure applied in lattice space on the density estimate shown in (B). We first determine the density estimates at the neurons' weight vectors, since hill-climbing is performed on them. For each neuron i, we look for the neuron with the highest density estimate among itself and its k-nearest lattice neighbors. When this is neuron i itself, it is called a "top" neuron, since it represents a local density maximum; when it is another neuron j, then the procedure is repeated for neuron j, and we say that neuron i "points" to neuron j. All neurons that eventually point to the same "top" neuron, receive the same cluster label. The result is represented as a cluster map (E). In order to motivate the choice of the parameter k in the hill-climbing procedure, the number of clusters found as a function of k is plotted (F). The long plateau indicates that there are 4 clusters. The cluster map corresponding to $k = 100$ is shown in (E).

Table caption

Table 1: Density estimation performance, expressed in Mean Squared Error (MSE), in the case of the product distribution example shown in Fig. 3A, when using our learning algorithm (LDE), and two of its modified versions, and four other kernel-based topographic map formation algorithms. The parameters used in these algorithms are also listed. See text.

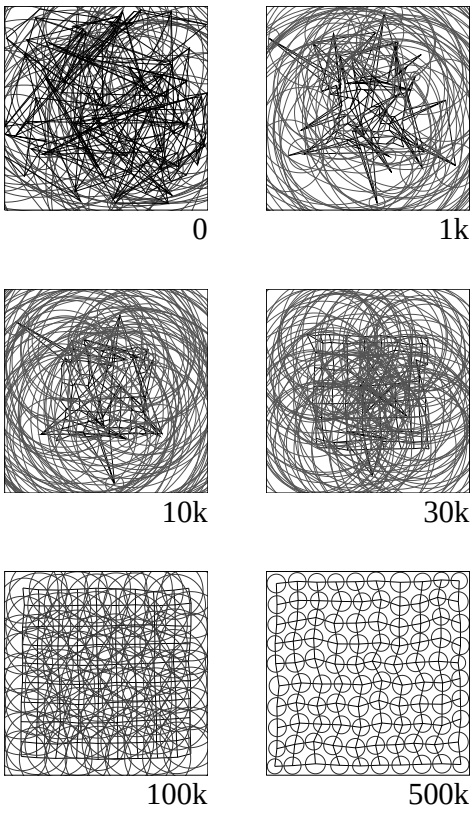

Figure 1:

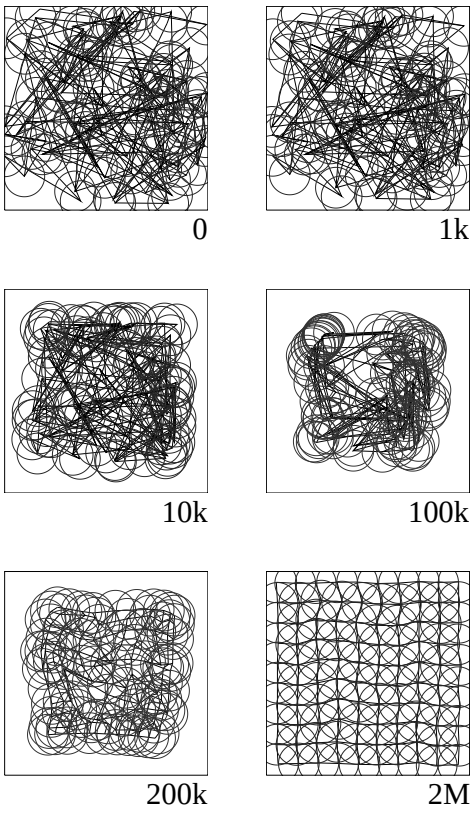

Figure 2:

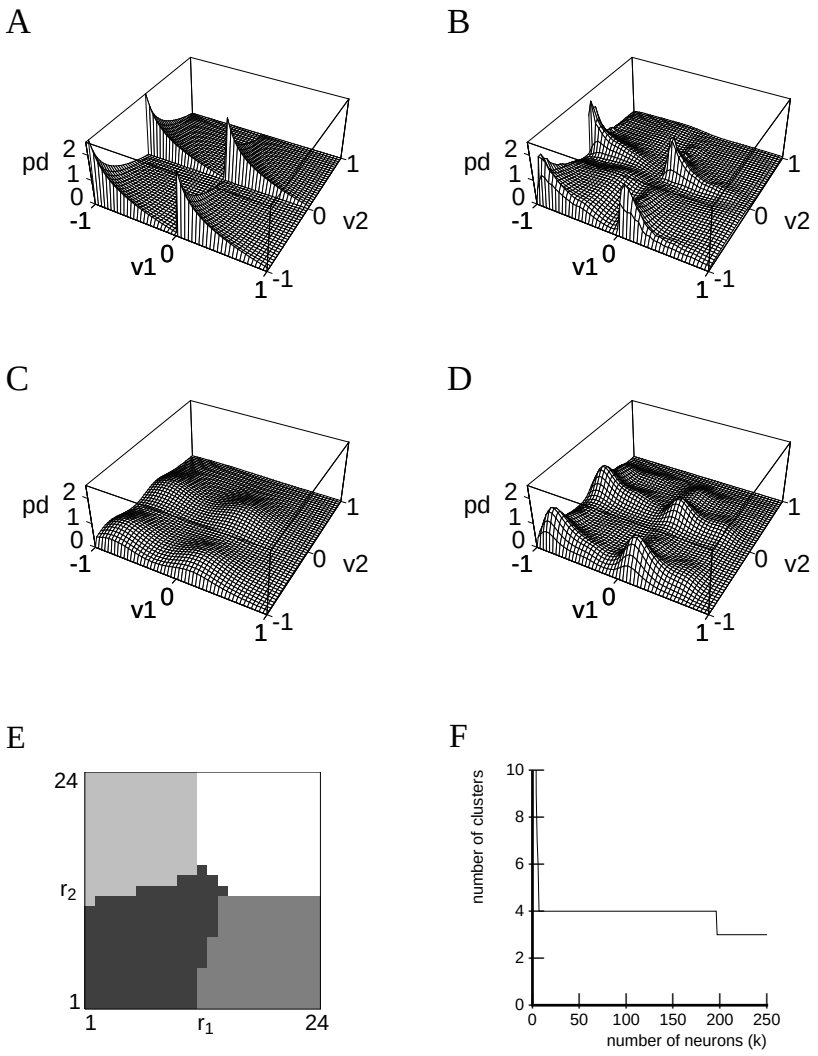
A
B
C
D
pd
2
1
0
-1
v1
v2
1
0
-1
E
F
24
r_2
1
r_1
number of clusters
10
8
6
4
2
0
0
50
100
150
200
250
number of neurons (k)

Figure 3:

Table 1:

algorithm	parameters	MSE
LDE	$\eta_w = 0.01$, $\eta_\sigma = 0.1\eta_w$	6.23×10^{-2}
LDE (min Eucl)	$\eta_w = 0.01$, $\eta_\sigma = 0.1\eta_w$	6.77×10^{-2}
LDE (fixed radii)	$\eta_w = 0.01$	9.78×10^{-2}
kMER	$\rho_r = 1$, $\rho_s = 2$, $\eta = 0.001$	6.71×10^{-2}
Yin & Allinson	$\eta = 0.01$	7.16×10^{-2}
STVQ	$\frac{1}{\beta} = 0.01$, $\sigma_\Lambda = 0.5$, $\sigma_{\text{opt}} = 0.0905$	7.48×10^{-2}
SSOM	$\frac{1}{\beta} = 0.01$, $\sigma_\Lambda = 0.5$, $\sigma_{\text{opt}} = 0.0925$	7.28×10^{-2}
SOM	$\eta = 0.015$, $\sigma_{\text{opt}} = 0.15$	1.21×10^{-1}