Softmax-network and S-Map – models for density-generating topographic mappings

Kimmo Kiviluoto, Erkki Oja

Abstract— **We propose a neural network model for density-generating topographic mappings. The model consists of two parts: the Softmax-network, and the S-Map. The Softmax-network implements the softmax function, so that each neuron's output is a softmax of the weighted sum of the input to that neuron and to its neighbors. The S-Map, based on the Softmax-network, utilizes a Hebbian-like learning scheme for the input-to-neuron weights to minimize the negative log likelihood error function; simulations show that a simplified version of the S-Map with fully Hebbian learning yields qualitatively similar results. The model is related both to the Generative Topographic Mapping (GTM) and the Self-Organizing Map (SOM).**

Keywords— **Softmax, S-Map, generative topographic mappings, self-organizing map**

I. Introduction

We present a model that forms a mapping from the original data space onto a neural lattice so that the topographic relations between the data vectors are preserved as well as possible. The model consists of two parts, the Softmax-network and the S-Map.

The Softmax-network determines the outputs of the network for a given input vector in a recurrent manner, based on the lateral feedback connections in the network. The output of a neuron in the Softmax-network becomes equal to the softmax function of the weighted sum of the afferent inputs to that neuron and to its neighbors. The S-Map, in turn, is a learning algorithm that, based on the Softmax-network, updates the afferent weights of the neurons to minimize the negative log likelihood error for a given data set, while topological ordering emerges.

We are not suggesting that our model would describe the information processing in real biological neurons. Neurobiology has been a notable source of inspiration, however – some of the central elements of the model have their roots in it. Therefore, we argue that both Softmax-network and S-Map can be characterized as "neural" in their philosophy, while being easily interpreted in probabilistical terms.

II. The Softmax-network

The activities of neurons in a network are often assumed to compete with each other: regardless of the initial state of the network, the neurons that receive the largest net input quickly become the most active ones, while the activities of other neurons in the network are suppressed. This kind of behavior has been suggested to occur in biological neural networks, and also many popular artificial neural network models are based on the idea that activities are determined by competition between the neurons – as a prominent example of such a model, we mention Kohonen's Self-Organizing Map [1].

The limiting case of the competition is Winner-Take-All (WTA) behavior: only the neuron receiving the largest input remains active after a transient period, and activities of all the other neurons become zero. The WTA network can suppress noise in the inputs; mathematically, it is equivalent to vector coding.

However, it may be argued that, based on a competitive network, a more effective coding scheme would be obtained if more than only one neuron were allowed to be active for each input, and if the outputs could also take on values between one and zero. This kind of "Winners-Get-Most" (WGM) behavior[1] could still suppress noise and thus improve contrast, but less information would be lost than with the WTA function, which is capable of distinguishing between only as many different types of input vectors as there are neurons in the network.

The WGM function could also help to find a probabilistical interpretation for the network outputs. From this point of view, an especially useful model for the WGM is the *softmax* output function (see e.g. [3])

$$\eta_j = \frac{\exp(\beta\alpha'_j)}{\sum_{i=1}^{K}\exp(\beta\alpha'_i)} \tag{1}$$

where α'_j is the effective input the to j^{th} neuron in an ensemble of K neurons, and η_j is its output; β is a parameter that controls the steepness, or degree of contrast-enhancement, of the softmax. The softmax function makes it possible to interpret the outputs as probabilities.

Here, we propose a simple model that implements the softmax function of the effective inputs to each neuron. The net inputs are taken to be weighted sums of the afferent inputs to each neuron and to its neighbors, the effective weights being given by a decreasing function of the distance between the neurons. The outputs of neurons located near each other thus become correlated – as later becomes obvious, this is necessary for the S-Map to form a topographic mapping in any useful sense.

A. The Softmax network model

In the Softmax network model, each neuron j is assumed to receive three kinds of input (the list of symbols used is presented in table I):

Both authors are with the Laboratory of Computer and Information Science, Helsinki University of Technology, Espoo, Finland. E-mail: Kimmo.Kiviluoto@hut.fi, Erkki.Oja@hut.fi

[1]Some authors, e.g. [2], have dubbed this kind of behavior as "Winners-Share-All". We feel that our proposal "Winners-Get-Most" might be slightly more descriptive: Winners get most but not all of the total activity of the network, and they do not share it in equal proportions.

TABLE I

LIST OF SYMBOLS

nominal lateral weights of neuron j (strengths of lateral connections from all other neurons to neuron j)	$\boldsymbol{\nu}_j = [\nu_{1,j} \ \ldots \ \nu_{K,j}]^T$
matrix of all nominal lateral weights	$\mathbf{N} = [\boldsymbol{\nu}_1 \ \ldots \ \boldsymbol{\nu}_K]$
effective lateral weights of neuron j	$\tilde{\boldsymbol{\nu}}_j = [\tilde{\nu}_{1,j} \ \ldots \ \tilde{\nu}_{K,j}]^T = [(\mathbf{N}^{-1})_{1,j} \ldots (\mathbf{N}^{-1})_{K,j}]^T$
input vector at time t	$\boldsymbol{\xi}^t = [\xi_1^t \ \ldots \ \xi_D^t]^T$
afferent weights of neuron j, $j = 1, \ldots, K$	$\boldsymbol{\mu}_j = [\mu_{1j} \ \ldots \ \mu_{Dj}]^T$
matrix of all afferent weights	$\mathbf{M} = [\boldsymbol{\mu}_1 \ \ldots \ \boldsymbol{\mu}_K]$
afferent input to neuron j	$\alpha_j = \boldsymbol{\mu}_j^T \boldsymbol{\xi}$
vector of afferent inputs to all neurons	$\boldsymbol{\alpha} = [\alpha_1 \ \ldots \ \alpha_K]^T$
effective afferent input to neuron j	$\alpha'_j = \tilde{\boldsymbol{\nu}}_j^T \boldsymbol{\alpha}$
output (activity) of neuron j	η_j
vector of outputs from all neurons	$\boldsymbol{\eta} = [\eta_1 \ \ldots \ \eta_K]^T$

- *afferent input* $\alpha_j = \boldsymbol{\mu}_j^T \boldsymbol{\xi}$, where $\boldsymbol{\mu}_j$ and $\boldsymbol{\xi}$ are the afferent weight and input vectors, respectively – here the dot-product is used as the measure of similarity between the weight and input vectors, but also other similarity measures such as Euclidean metric could be used;
- *neuron-specific lateral input* $\boldsymbol{\nu}_j^T \mathbf{f}(\boldsymbol{\eta})$, where $\boldsymbol{\nu}_j$ is the lateral connection strength vector of the j^{th} unit and $\mathbf{f}(\boldsymbol{\eta})$ is some function of the outputs $\boldsymbol{\eta}$ of all the units; our choices for $\boldsymbol{\nu}_j$ and $\mathbf{f}$ are discussed shortly;
- *lateral input common to all neurons* γ, which serves the purpose of normalizing the total output of the network.

Networks that have this type of general structure have been considered earlier by several different authors, especially Grossberg and his co-workers, who have focused on such networks in neurophysiological context (see e.g. [4]). Also one way to motivate the Self-Organizing Map algorithm is the network architecture described above [5].

In principle, the only requirement for the lateral connection strength matrix $\mathbf{N}$ is that it must be positive definite to ensure the stability of the network. In this paper, we consider the case when the lateral connections are of the following simple form: each neuron receives positive feedback of equal strength from all neurons within a certain distance from it, and strong negative feedback from itself. In one-dimensional case, the lateral connection strength vector $-\boldsymbol{\nu}_j$ of the j^{th} neuron would then be

$$-\boldsymbol{\nu}_j = [\underbrace{0 \cdots 0}_{j-k-1 \text{ units}} \ \underbrace{1 \cdots 1}_{k \text{ units}} \ (-2k-1) \ \underbrace{1 \cdots 1}_{k \text{ units}} \ 0 \cdots 0] \tag{2}$$

where k determines the range of the explicit lateral interaction; for notational convenience, we prefer to write here $-\boldsymbol{\nu}_j$ instead of $\boldsymbol{\nu}_j$.

The functional form of the lateral feedback $\mathbf{f}(\boldsymbol{\eta})$ is here assumed be *logarithmic*. This choice gives rise to many desirable properties of the resulting network. Here, our aim is not modeling biological neurons; however, it might be interesting to note that Tal and Schwartz [6] have recently suggested that "quasi-logarithmic computations could occur in interneurons of the central nervous system".

The model for the output of the j^{th} neuron can be expressed as the differential equation

$$\dot{\eta}_j = \left[\alpha_j - \sum_{i=1}^{K} \nu_{i,j} \ln \eta_i + (1-\gamma) \right] \eta_j \tag{3}$$

Therefore, the differential equation for all outputs, written in vector form, is

$$\dot{\boldsymbol{\eta}} = \text{diag}[\boldsymbol{\alpha} - \mathbf{N} \ln \boldsymbol{\eta} + (1-\gamma)\mathbf{1}]\boldsymbol{\eta} \tag{4}$$

where logarithm of a vector is taken elementwise so that $\ln \boldsymbol{\eta} = [\ln \eta_1 \ \ldots \ln \eta_K]^T$, $\text{diag}(\boldsymbol{\alpha})$ denotes a matrix with elements $[\alpha_1 \ \ldots \ \alpha_K]^T$ on the main diagonal and zeros elsewhere, and $\text{diag}(\mathbf{1}) = \mathbf{I}$, the unit matrix.

The Softmax-network was greatly inspired by the model proposed by Fukai and Tanaka [2]; the main difference between our model and their neuroecological equation is that our model employs logarithmic feedback whereas in the neuroecological equation the feedback is linear. Note that both of these models can be interpreted as special cases of the general class of Cohen-Grossberg competitive dynamical systems [7].

B. The stable states of the Softmax-network

Let us calculate the stable states of the differential equations (4). These occur at the points in which the time derivatives for the neuron outputs vanish:

$$\text{diag}[\boldsymbol{\alpha} - \mathbf{N} \ln \boldsymbol{\eta} + (1-\gamma)\mathbf{1}]\boldsymbol{\eta} = 0 \tag{5}$$

The non-trivial solutions for (5) are

$$\boldsymbol{\alpha} - \mathbf{N} \ln \boldsymbol{\eta} + (1-\gamma)\mathbf{1} = \mathbf{0} \tag{6}$$

$$\Leftrightarrow \ln \boldsymbol{\eta} = \mathbf{N}^{-1}[\boldsymbol{\alpha} + (1-\gamma)\mathbf{1}] \tag{7}$$

If the lateral weights $\boldsymbol{\nu}_j$ are chosen as in (2), the columns of the inverse $\mathbf{N}^{-1}$ sum to one, and we may further simplify (7) to

$$\begin{aligned} \boldsymbol{\eta} &= \exp(1-\gamma) \exp(\mathbf{N}^{-1}\boldsymbol{\alpha}) \\ &= \gamma' \exp(\mathbf{N}^{-1}\boldsymbol{\alpha}) \end{aligned} \tag{8}$$

The output of each neuron j is thus proportional to the exponential function of the weighted sum of afferent inputs to that neuron and its neighbors, with weights given by $\tilde{\boldsymbol{\nu}}_j = [(\mathbf{N}^{-1})_{1,j} \ldots (\mathbf{N}^{-1})_{K,j}]^T$. A comparison of our choice for the "nominal" lateral weights $-\boldsymbol{\nu}_j$ and the resulting "effective" lateral weights $\tilde{\boldsymbol{\nu}}_j$ is depicted in figure 1.

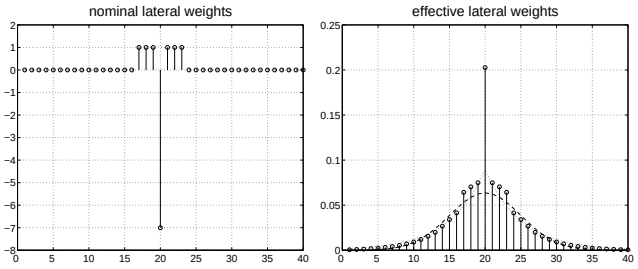

Fig. 1. The nominal lateral weights $-\boldsymbol{\nu}_j$ and the corresponding effective lateral weights $\tilde{\boldsymbol{\nu}}_j$ in the case of one-dimensional neuron lattice. For comparison, in the same picture with the effective weights, also Gaussian (dashed line) and Laplacian (dotted line) functions are shown.

The output of a single neuron can therefore be written as

$$\eta_j = \gamma' \exp(\tilde{\boldsymbol{\nu}}^T \boldsymbol{\alpha}) = \gamma' \exp\left[\left(\sum_{i=1}^{K} \tilde{\nu}_{i,j} \boldsymbol{\mu}_i\right)^T \boldsymbol{\xi}^t\right] \tag{9}$$

Contrasting this to (1), it is clear that with a suitable choice for γ', *the network outputs equal a softmax of the* effective *afferent inputs* $\alpha'_j = \tilde{\boldsymbol{\nu}}_j^T \boldsymbol{\alpha}$, *which are weighted sums of the original afferent inputs* $\boldsymbol{\alpha}$. Some possible choices for γ' are discussed below.

Although not explicitly shown here, the steepness parameter of the softmax, denoted with β in (1), can be incorporated into the model by appropriate scaling of the afferent weight vectors $\boldsymbol{\mu}_j$, which results in a corresponding scaling of the afferent inputs $\alpha_j = \boldsymbol{\mu}_j^T \boldsymbol{\xi}$.

C. Constraining the total activity of the network

The choice for γ' depends on the desired normalization – should the network outputs sum up to some constant, or should the length of the network output vector $||\boldsymbol{\eta}||$ be normalized? The normalization can be performed according to the scheme introduced in [8]. In difference equation form, the approximation for the Softmax-network outputs can thus be written as

$$\boldsymbol{\eta} := \boldsymbol{\eta} + \delta \,\mathrm{diag}(\boldsymbol{\alpha} - \mathbf{N} \ln \boldsymbol{\eta} + \mathbf{1}\boldsymbol{\eta}^T \mathbf{N} \ln \boldsymbol{\eta})\boldsymbol{\eta} \tag{10}$$

if the outputs are to sum up to some constant (δ is a step size parameter), and as

$$\boldsymbol{\eta} := \boldsymbol{\eta} + \delta \,\mathrm{diag}[\boldsymbol{\alpha} - \mathbf{N} \ln \boldsymbol{\eta} + \mathbf{1}\boldsymbol{\eta}^T \,\mathrm{diag}(\mathbf{N} \ln \boldsymbol{\eta})\boldsymbol{\eta}] \tag{11}$$

if the network output vector should be of constant length.

D. Simulating the Softmax-network

In figure 2, simulation results with a one-dimensional lattice consisting of 1,000 neurons are shown. The initial outputs of the network were random in the first simulation and ordered in the second. For comparison, also shown is a softmax function of the inputs that is calculated explicitly using Gaussian effective weights.

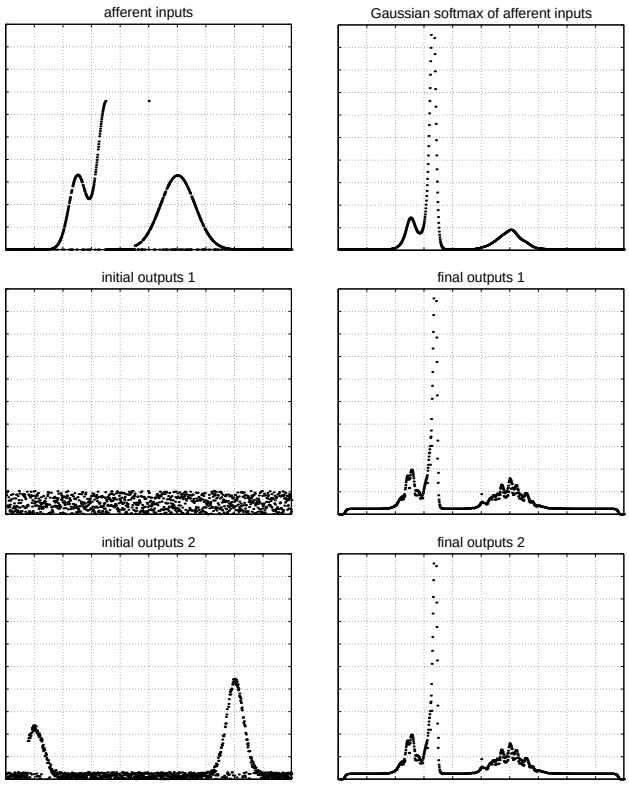

Fig. 2. Simulations with a one-dimensional Softmax-network. Network units are evenly spaced along the x-axis; the input to or output from the i^{th} unit corresponds to the y-coordinate of the dot placed at $x = i$. The afferent inputs are shown on top left, an explicitly calculated softmax with Gaussian effective weights on top right; the random initial outputs of the Softmax-network on middle left, and the corresponding final outputs on middle right; the non-random initial outputs of the Softmax-network on bottom left, and the corresponding final outputs on bottom right.

III. The S-Map

The S-Map [9] is a network model with a learning algorithm for the afferent weight updates. It is based on the Softmax-network in the sense that it utilizes the Softmax-network for determining the outputs of the neurons for a given input vector.

A. S-Map learning algorithm

The S-Map learning algorithm is used to tune the afferent weights. The updates are proportional to the activity of the unit that the weight is associated with, and the activities of the neighboring units. Using the "dot-product metric" to measure the similarity between the afferent weights $\boldsymbol{\mu}_j$ and afferent inputs $\boldsymbol{\xi}$, the learning rule can be written as

$$\boldsymbol{\mu}_j^{t+1} := \boldsymbol{\mu}_j^t + \delta^t \left(\sum_{i=1}^{K} \phi_j^i \eta_i^t\right)(\mathbf{I} - \boldsymbol{\mu}_j^t \boldsymbol{\mu}_j^{tT})\boldsymbol{\xi}^t \tag{12}$$

In words, the stronger the activities η_i that are given by (9) near unit j, where "nearness" is determined by the weighting factors ϕ_j^i, the more the afferent weight vector $\boldsymbol{\mu}_j$ turns toward the present afferent input vector $\boldsymbol{\xi}^t$. The matrix in the second parentheses keeps the weight vectors normalized

to unit length, assuming a small value for the step size parameter δ^t [8]. Allowing the weight vector lengths to take on also other values besides unity corresponds to varying the steepness of the softmax (1).

The S-map learning algorithm (12) can be further simplified to employ pure Hebbian learning, so that the afferent weight updates are proportional to the unit's own activity only:

$$\boldsymbol{\mu}_j^{t+1} := \boldsymbol{\mu}_j^t + \delta^t \eta_j^t (\mathbf{I} - \boldsymbol{\mu}_j^t \boldsymbol{\mu}_j^{tT}) \boldsymbol{\xi}^t \tag{13}$$

B. Probabilistic interpretation of the S-Map

Let us assume that the afferent weight vector lengths are normalized to some constant that is common to all neurons – i.e., we may write the afferent weights as $\beta \boldsymbol{\mu}_j$, $||\boldsymbol{\mu}_j|| = 1$ for all j. Then, it is possible to make the following interpretation: *each neuron j of the network generates a density in the input space:*

$$p(\boldsymbol{\xi}|j;\mathbf{M},\beta) = \\ = \begin{pmatrix}\text{normalizing}\\ \text{constant}\end{pmatrix}^{-1} \times \exp\left[\beta \left(\sum_{i=1}^K \tilde{\nu}_{i,j} \boldsymbol{\mu}_i\right)^T \boldsymbol{\xi}\right] \tag{14}$$

where β is the inverse of the variance of the density. Note that the density generated by neuron j is centered at the *image* of that neuron in the input space, which is given by $\sum_{i=1}^K \tilde{\nu}_{i,j} \boldsymbol{\mu}_i$. The interpretation bears similarities to that introduced in the GTM algorithm of Bishop et al. [10].

Summing up all the densities (14) and normalizing, a mixture density is obtained. Assuming equal prior probabilities for all units, the posterior probability that the neuron j were "responsible" for the data vector $\boldsymbol{\xi}^t$ is then given by

$$p(j|\boldsymbol{\xi}^t;\mathbf{M},\beta) = \frac{p(\boldsymbol{\xi}^t|j;\mathbf{M},\beta)}{\sum_{j'=1}^K p(\boldsymbol{\xi}^t|j';\mathbf{M},\beta)} \\ = \frac{\exp\left[\beta \left(\sum_{i=1}^K \tilde{\nu}_{i,j} \boldsymbol{\mu}_i\right)^T \boldsymbol{\xi}^t\right]}{\sum_{j'=1}^K \exp\left[\beta \left(\sum_{i=1}^K \tilde{\nu}_{i,j'} \boldsymbol{\mu}_i\right)^T \boldsymbol{\xi}^t\right]} \tag{15}$$

which can be equated with (9) because of the assumption made in the beginning of this subsection. We may therefore write

$$\eta_j^t = p(j|\boldsymbol{\xi}^t;\mathbf{M},\beta) \tag{16}$$

– the outputs of the neurons for a given input vector are equal to the probabilities that they were "responsible" for generating that input vector.

C. Properties of the S-Map

The S-Map algorithm has been shown to minimize the negative log likelihood for a given data set [9], provided that the weights ϕ_j^i in (12) equal the effective weights $\tilde{\nu}_{i,j}$ in (14) and (15).

The S-Map has many properties in common with the SOM and GTM algorithms – it might even be characterized as a crossbreed of those two algorithms, combining the computational simplicity and the ability to self-organize of the SOM with the probabilistic framework of GTM. A more thorough discussion of the S-Map, SOM and GTM can be found in [9].

D. Simulations

In this section, we present results of training the S-map and its simplified version with an artificial data set. The data consists of 500 points from a uniform random distribution in the unit square. Initially, the afferent weight vectors were set to random values. The initial and final configurations of the maps plotted on top of the data are shown in figure 3. A batch version of the algorithms was used here: the weight vectors were updated only after going through all the data vectors. Moreover, we used an Euclidean metric version of the algorithms, in which equations (12) and (13) become

$$\boldsymbol{\mu}_j^{t+1} := \boldsymbol{\mu}_j^t + \delta^t \left(\sum_{i=1}^K \tilde{\nu}_{i,j} \eta_i^t\right) (\boldsymbol{\xi}^t - \boldsymbol{\mu}_j) \tag{17}$$

and

$$\boldsymbol{\mu}_j^{t+1} := \boldsymbol{\mu}_j^t + \delta^t \eta_j^t (\boldsymbol{\xi}^t - \boldsymbol{\mu}_j) \tag{18}$$

respectively.

The simulations show that both the S-Map and the simplified S-Map produce a topologically ordered mapping even from random initialization.

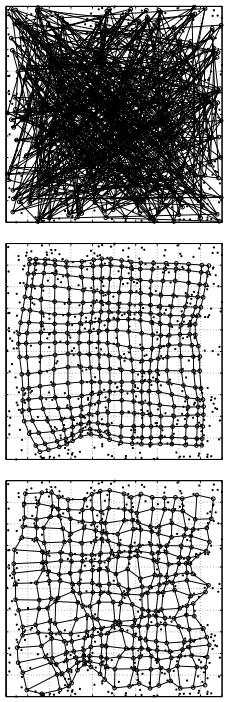

Fig. 3. Random initialization (top), the trained S-Map (middle) and the trained simplified S-Map (bottom)

IV. Discussion

Much of the work done in neural modelling at the early phases of artificial neural network research was bottom-up in the sense that the models started from a minimum of functional principles realizable in parallel neural networks, biological or artificial. The emphasis was on the exact network architectures and the dynamics of activation and neural learning. This made many of the models hard to analyze rigorously, however, although their validity could often be substantiated by extensive simulations.

In the present day research, more and more emphasis is put on "principled" approaches, in which the goals of neural learning are given explicitly, often using Bayesian statistics, and the algorithms are derived from the top-down criteria by conventional numerical methods. Little concern is devoted to whether the algorithms could actually be realized in biological or even artificial neural circuits.

We have here attempted to formulate a neural network for self-organization that is based on a simple feedback architecture and activation functions, yet can be shown to be equivalent to a rigorous probabilistic model in which the neuron activities are equal to well-defined probabilities. By experiments, the model was shown to be able to self-organize in the same way as the well-known Self-Organizing Map.

References

[1] T. Kohonen, *Self-Organizing Maps*. Springer Series in Information Sciences 30, Berlin Heidelberg New York: Springer, 1995.

[2] T. Fukai and S. Tanaka, "A simple neural network exhibiting selective activation of neuronal ensembles: from winner-take-all to winners-share-all," *Neural Computation*, vol. 9, pp. 77–97, 1997.

[3] J. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, architectures and applications* (F. Fogelman Soulié and J. Hérault, eds.), (New York), Springer-Verlag, 1990.

[4] S. Grossberg, "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, no. 23, pp. 121–134, 1976.

[5] T. Kohonen, "The 'neural' phonetic typewriter," *Computer*, vol. 21, no. 3, pp. 11–22, 1988.

[6] D. Tal and E. L. Schwartz, "Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication," *Neural Computation*, vol. 9, pp. 305–318, 1997.

[7] M. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, pp. 815–826, 1983.

[8] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267–273, 1982.

[9] K. Kiviluoto and E. Oja, "S-map: A network with a simple self-organization algorithm for generative topographic mappings," in *Advances in Neural Information Processing Systems* (M. I. Jordan, M. J. Kearns, and S. A. Solla, eds.), vol. 10, MIT Press, 1998. To appear.

[10] C. M. Bishop, M. Svensen, and C. K. I. Williams, "GTM: A principled alternative to the self-organizing map," in *Advances in Neural Information Processing Systems* (M. C. Mozer, M. I. Jordan, and T. Petche, eds.), vol. 9, MIT Press, 1997.