C2.1 Feedforward models

Michel Verleysen

Abstract

Feedforward unsupervised models cover a wide range of neural networks with various applications. In this section, we discuss three widely used models. (i) Kohonen's self-organizing map, also called the *Kohonen network*, the *self-organizing feature map*, or the *topological map*, is intended to map a high-dimensional space into a one- or two-dimensional space, preserving the topology of the input space; it has a strong biological plausibility and is basically intended to be used in applications where preserving the topology between input and output spaces is important (e.g. control, inverse mapping, image compression). It is an unsupervised model, but can be extended to a supervised one by adding a supplementary layer. In addition to the topology-conserving property, the Kohonen model also acts as a vector quantizer. (ii) The neural gas is another vector quantization algorithm that may be considered as a neural network method because it relies on the same principle of adaptation, may be represented in the form of a feedforward graph, and may be described by the same formalism as used in many other neural models. It is different from the Kohonen map in the sense that it does not have the topology preserving property but it generally performs better giving a smaller final distortion error. (iii) The neocognitron is a complex feedforward model formed by several layers each containing a large number of neurons. Its goal is to automatically detect features in two-dimensional arrays of points through self-organization and reinforcement principles. The network is built to be insensitive to shifts in position of the patterns or of small parts of them, thus also allowing for distorted patterns. The network is primarily intended to be used in feature extraction and pattern recognition tasks, for example in OCR (optical character recognition).

C2.1.1 Kohonen's self-organizing map

C2.1.1.1 Introduction

The self-organizing feature map (SOFM) has been developed by Teuvo Kohonen (Helsinki University of Technology, Laboratory of Computer and Information Sciences, FIN-02150 Espoo 15, Finland). While it has been described in several research papers, an interesting and self-contained description of the model, its biological background, its implementation, and possible applications can be found in one of the three editions of Kohonen's seminal book *Self-Organization and Associative Memory* (Kohonen 1989).

The SOFM is presented as a biologically plausible network, which takes its inspiration from the fact that some regions of the cortex 'map' either physical locations of sensory neurons, or the ordering of some physical properties like the acoustic frequencies in the auditory cortex.

C2.1.1.2 Purpose of the model

The purpose of the self-organizing feature map is basically to map a continuous high-dimensional space into a discrete space of lower dimension (usually 1 or 2). This goes through two more or less independent properties obtained through the topology and the learning of the network.

First, as in adaptive vector quantization methods like the *LVQ algorithms*, the input space is quantized by assigning each neuron to a defined region in the input space. The number of (external) inputs to each neuron being equal to the dimension of the input space, the weight vector of each neuron can be interpreted as a location in this space. The region then assigned to a neuron is the set of locations nearer to the corresponding neuron than to any other one; this is called the 'Voronoi tessellation' of the input space (see figure C2.1.1 for a Voronoi tessellation of a two-dimensional space). C1.1.5

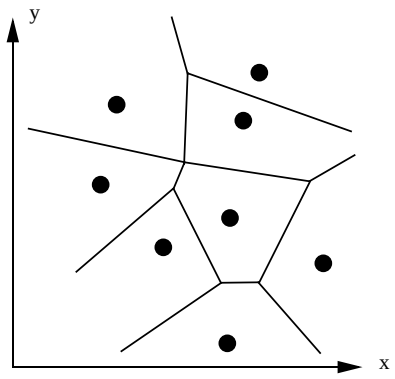

Figure C2.1.1. Voronoi tessellation of a two-dimensional space.

In figure C2.1.1, the eight neurons are represented by using their weight vector (of two components, the input space being of two dimensions) as coordinates in a two-dimensional space; this is the common representation standard for Kohonen maps.

The second property concerns the preservation of the topology. The principle is that the network organizes its topology in the following way. Inherently, all neurons of the network, regardless of their weight vectors, are arranged according to a defined topology. For example, they can be arranged on a one-dimensional string or a two-dimensional grid, as shown in figure C2.1.2.

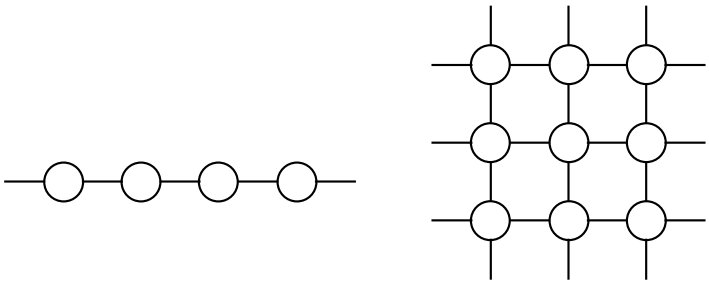

Figure C2.1.2. One-dimensional string and two-dimensional grid.

The 'dimension' d_{out} of the string (one) or of the grid (two) is usually referred to as the output dimension; it is absolutely independent of the dimension of the input space d_{in}, which is determined by the number of external inputs to each neuron.

Having defined these two dimensions, the second property of self-organizing feature maps can now be explained as follows. After learning, that is, after the network has converged to an 'ordered' topology, two input vectors (of dimension d_{in}), one close to the other (according to the definition of a distance measure in the input space), will be projected on two close neurons in the output space; close neurons means here that they are close on the string or on the grid, depending on the output dimension.

Of course, a good topology preservation is not always possible if the input and output dimensions are different; for example, if the dimension of the input space is three, and the neurons are arranged on a two-dimensional grid, and if the input space is uniformly filled, it is not possible to find a projection from the three-dimensional space to the two-dimensional one that will respect the topology (the neighborhood property) at all locations.

Nevertheless, the property of the Kohonen map is that the learning is targeted to the 'best' weight vector for each neuron in order to best match the neighborhood property; this will be detailed in a later section, mainly through examples.

C2.1.1.3 Biological origin

It is well known now that the organization of cells in the brain, and more especially in the cortex, is not a result of chance. Without doubt, a large part of this organization is determined genetically, while it is also proven that learning plays an important role. For example, persons having had a sensory organ amputated will develop different sensory sites than other people, through a rearrangement of dedicated sites in the brain.

Moreover, the locations of nervous cells assigned to sensory or motor tasks in the brain are, at least in several known parts of it, arranged in a way such that they map either the physical locations of the sensory or motor organs themselves, or some of their physical properties: receptive fields are arranged in the cortex according to the sensory organs, neighboring locations in the visual cortex correspond to neighboring locations in the retina, close neurons in the auditory cortex are activated by close frequencies, and so on.

Of course, because of the complex physical structure of the brain and cortex, these mappings are highly nonlinear; their main property is that 'close' elements before the projection will correspond to 'close' elements after the projection. It is this specificity that researchers such as Cristoph Von der Malsburg (1973) and Teuvo Kohonen (1989) tried to model through topological maps.

C2.1.1.4 Topology

A self-organizing feature map contains one layer of neurons, but two layers of connections, as illustrated in figure C2.1.3. For the purpose of simplicity, it will be assumed in the following that the output dimension d_{out} is two, i.e. that neurons are arranged on a two-dimensional grid. Each neuron has d_{in} external connections, to the d_{in} inputs. In addition, each neuron is laterally connected to its neighbors (on the grid), up to a certain distance; for example, neuron n_{43} in the grid can be connected to its four nearest neighbors n_{33}, n_{42}, n_{44}, and n_{53}, or to its eight nearest neighbors (the four preceding ones plus n_{32}, n_{34}, n_{52}, and n_{54}), and so on.

Computations are feedforward in the first layer of connections: the network computes the scalar product between the input vector $\boldsymbol{x}$ and each of the neuron weight vectors $\boldsymbol{w}$. The self-organizing feature map usually supposes that input and weight vectors are normalized; the scalar product can thus be considered as a distance measure between the two vectors $\boldsymbol{x}$ and $\boldsymbol{w}$ (it is the angle formed by these two vectors on a unit circle).

The second layer of connections acts as a *recurrent* excitatory/inhibitory network, whose aim is to B2.3 reinforce the activation values of 'strong' neurons and to decrease the activation of the 'weak' ones. The values of the lateral connections are fixed (they are not changed during the learning), and are only dependent on the physical distance between neurons on the grid. A typical function representing the values

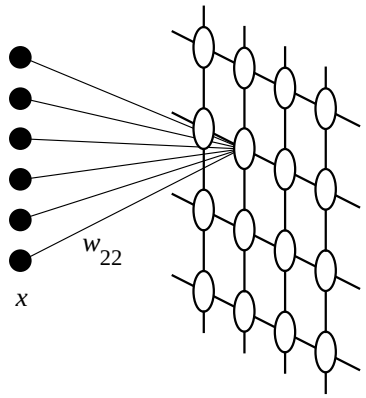

Figure C2.1.3. Self-organizing feature map.

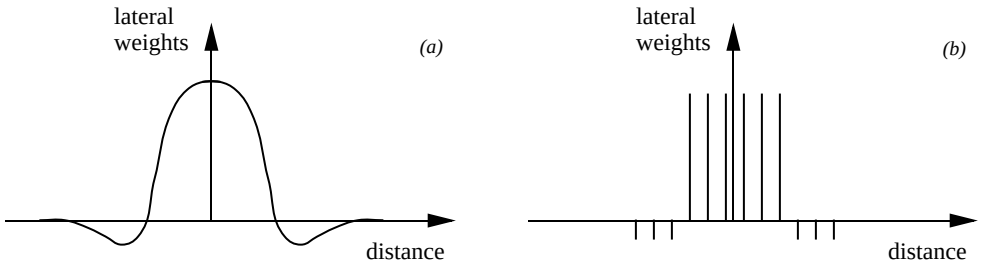

Figure C2.1.4. Lateral connections between neurons.

of the lateral weights versus the distance between neurons on the grid is plotted in figure C2.1.4(a), while practically used values (at discretized locations since there is a discontinuous set of distances between neurons) are shown in figure C2.1.4(b).

It can be seen from figure C2.1.4 that connections for close neurons are positive (excitatory synapses), while they are negative for more distant ones (inhibitory synapses); they tend to zero for larger values of the distance between neurons.

While the connections in the first layer (connections to input vectors) are strictly feedforward, the lateral ones between neurons are bidirectional: there is obviously the same connection between neurons n_{ij} and n_{kl} as between neurons n_{kl} and n_{ij}, since the value of the weight only depends on the distance between the neurons (measured as $|k-i|+|l-j|$).

C2.1.1.5 Operation of the network

The behavior of the network is formally described by

$$a_{ij} = \sigma\left[\sum_{k=1}^{d_{\text{in}}} w_{ijk}x_k + \sum_{\mathrm{n}_{mn}\in N(\mathrm{n}_{ij})} v_{mnij}a_{mn}\right] \tag{C2.1.1}$$

where a_{ij} is the activation of neuron n_{ij}, w_{ijk} the weight between neuron n_{ij} and the kth component x_k of the input vector $\boldsymbol{x}$, $N(\mathrm{n}_{ij})$ the neighborhood of neuron n_{ij} as defined above, v_{mnij} the lateral weight between neurons n_{mn} and n_{ij}, and $\sigma[\cdot]$ a standard sigmoid-type nonlinearity. In this equation and in the following equations, neurons (and related values as activations) have been numbered by two indices, corresponding to their location on the Kohonen grid ($d_{\text{out}} = 2$), and layer indices have been omitted for simplicity; lateral fixed weights between neurons are denoted by v to avoid the confusion with adaptable weights w.

Equation (C2.1.1) forms a complex system of coupled nonlinear equations that must be solved to find activation values a_{ij}. However, a simple interpretation of the behavior of the lateral connection layer is that it reinforces the activity of neurons in the areas of the map where strong activations already exist through the first summing term of equation (C2.1.1), while it decreases the activations of neurons in other areas. This usually leads to the formation of an 'activity bubble' in the map, as shown in figure C2.1.5, where the neuron activities in a 15×15 neuron map are represented by their gray level.

With this property, Kohonen showed that it is possible to calculate the activations in a simpler and much more straightforward way. Since the activity bubble in the map will be located where the activations due to the first summing term of equation (C2.1.1) are the most important ones, Kohonen proposes to determine the center of this bubble by choosing the 'best match' between the input vector $\boldsymbol{x}$ and the weight vectors $\boldsymbol{w}_{ij}$, under a defined distance measure. To avoid the necessity of normalization with the scalar product, the Euclidean distance can be chosen. The best-match neuron n_{ab} will thus be chosen as:

$$\|\boldsymbol{x} - \boldsymbol{w}_{ab}\| \leq \|\boldsymbol{x} - \boldsymbol{w}_{ij}\| \qquad \forall 1 \leq i \leq N_i, \forall 1 \leq j \leq N_j \tag{C2.1.2}$$

where N_i and N_j are, respectively, the number of neurons in the x and y directions of the map ($N = N_i N_j$ is the total number of neurons). The selection of the best-match neuron will be used in the learning process of the network.

In this simplified model, much more convenient for practical computations, the activities of the neurons are computed by:

$$a_{ij} = \|\boldsymbol{x} - \boldsymbol{w}_{ij}\| . \tag{C2.1.3}$$

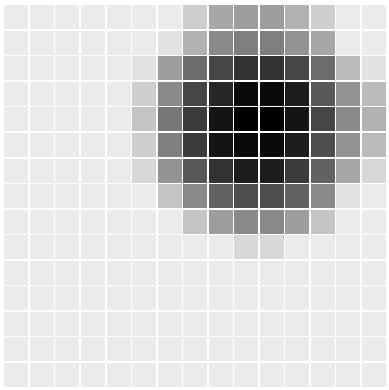

Figure C2.1.5. Activities and 'bubble' in a 15×15 neuron Kohonen map.

While the values of the activities will obviously not be the same in the original and simplified models, the same phenomena of bubble formation and of topology preservation will be encountered in both models, through appropriate learning.

C2.1.1.6 Learning

In order to ensure the above described property of topology conservation between the input space and the locations of 'winning' (best-match) neurons on the grid, appropriate values of the weights w and v in the two layers of connections must be chosen.

Values of lateral connection weights v are chosen according to some Mexican-hat-like function as illustrated in figure C2.1.4. The width of the function (i.e. the distance between neurons above which connection weights are null) is initially chosen as 'reasonable' according to the size of the map, and then decreased during learning.

Values of weights w between the neurons and the input vector $\boldsymbol{x}$ are the result of an adaptive process. Learning in a Kohonen map is a typical example of *unsupervised learning*: input vectors are presented to the network, and the weights are adapted without any knowledge of a 'desired output value' to the network. During learning, the values of the weights are adapted according to

B3.1

$$\boldsymbol{w}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) + \alpha(t)(\boldsymbol{x}(t) - \boldsymbol{w}_{ij}(t)) \qquad \text{if} \quad \mathrm{n}_{ij} \in N(\mathrm{n}_{ab}) \tag{C2.1.4}$$

$$\boldsymbol{w}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) \qquad \text{if} \quad \mathrm{n}_{ij} \notin N(\mathrm{n}_{ab}) \tag{C2.1.5}$$

where $\boldsymbol{w}_{ij}(t)$ is the weight vector between neuron n_{ij} and input vector $\boldsymbol{x}(t)$ at time step t of the learning, $\boldsymbol{x}(t)$ is the input vector presented to the network at time step t, $\alpha(t)$ is a learning factor that decreases with time to ensure convergence to fixed states, n_{ab} is the best-match neuron according to equation (C2.1.2), and $N(\mathrm{n}_{ab})$ is a neighborhood of the best-match neuron n_{ab}, that also decreases with time.

In some other versions of this simplified model, a weaker adaptation can be chosen for neurons 'far' from the best-match one n_{ab}, by adding a multiplying term in the equation

$$\begin{aligned} &\boldsymbol{w}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) + \alpha(t)\beta(|i-a| + |j-b|)(\boldsymbol{x}(t) - \boldsymbol{w}_{ij})(t) \qquad \text{if} \quad \mathrm{n}_{ij} \in N(\mathrm{n}_{ab}) \\ &\boldsymbol{w}_{ij}(t+1) = \boldsymbol{w}_{ij}(t) \qquad \text{if} \quad \mathrm{n}_{ij} \notin N(\mathrm{n}_{ab}) \end{aligned} \tag{C2.1.6}$$

where $\beta(\cdot)$ is a monotonically decreasing function of the distance between neurons n_{ij} and n_{ab} in the map.

The choice of the adaptation factor $\alpha(t)$ and the neighborhood domain $N(\mathrm{n}_{ab})$ as a function of time is important, but not critical. Usually, $\alpha(t)$ is chosen according to the Robbins–Monro conditions (Robbins

and Monro 1951), that is,

$$\lim_{t\to\infty}\alpha(t)=0 \qquad \sum_{t=0}^{\infty}\alpha(t)=\infty \qquad \sum_{t=0}^{\infty}\alpha^2(t)=\infty\,. \tag{C2.1.7}$$

The learning algorithm of Kohonen maps in natural language representation is:

(i) compute the Euclidean distance between the input vector and the weight vector associated with the first neuron in the map;
(ii) repeat step (i) for all neurons in the map;
(iii) determine the best-match neuron, the neuron whose distance computed at step (i) is minimum;
(iv) determine a topological neighborhood of the best-match neuron in the Kohonen map;
(v) update the first weight associated with the first neuron in that topological neighborhood by adding a fraction of the difference between the input vector and the weight of this neuron;
(vi) repeat step (v) for all neurons in the neighborhood determined at step (iv).

The learning rule is basically local: the operation at each neuron only requires the knowledge of the weight associated with that neuron and of the input vector. However, a nonlocal function (but implementable as a tree structure) is needed to choose the best-match neuron. Adaptation is then again local, as soon as a neighborhood of the best-match neuron is chosen.

C2.1.1.7 Convergence of the algorithm

As explained by Marie Cottrell (Cottrell *et al* 1994) in her review paper about the different proofs of convergence of the Kohonen algorithm in specific situations,

'Despite the large use and the different implementations in multi-dimensional settings,
the Kohonen algorithm is surprisingly resistant to a complete mathematical study'.

In high dimensions, two obstacles prevent a thorough study of the convergence of the algorithm. First, while ordering is an obvious concept in one dimension, it is difficult to know what is a correctly ordered situation in a dimension greater than one; secondly, it is proven that it is impossible to associate a global decreasing potential function with the algorithm in the general case (Erwin *et al* 1992). In that situation, convergence results of the algorithm are only partial or apply in specific situations. Details of the actual state of the research in that domain may be found in the article by Cottrell *et al* (1994); here are, very briefly, some of these results:

- The self-organization property and the convergence are proved in the one-dimensional case for a Kohonen string, for a large class of input distributions and neighborhood functions.
- Self-organization in dimension 1 is proven for a large class of neighborhood functions when the input and the weights are quantized.
- A potential function (from which the Kohonen algorithm is a stochastic *gradient descent* function) can be found when the input values belong to a finite discrete set; this potential function is not differentiable, but the convergence to a stationary point can be proven under some conditions on the adaptation parameter and the neighborhood function. B5.2.2
- The '0-neighbor' algorithm (Kohonen procedure with neighborhood restricted to the best-match vector only) always corresponds to a gradient descent procedure.
- Results in higher dimensions are only partial.

Details and references on actual results can be found in the article by Cottrell *et al* (1994).

C2.1.1.8 Examples of results

The Kohonen map is basically aimed to project a continuous high-dimensional space onto a discrete one- or two-dimensional one. For illustration purposes, we will show by a few examples the projection of a two-dimensional space onto a one- or two-dimensional one. Of course, the real interest of Kohonen maps

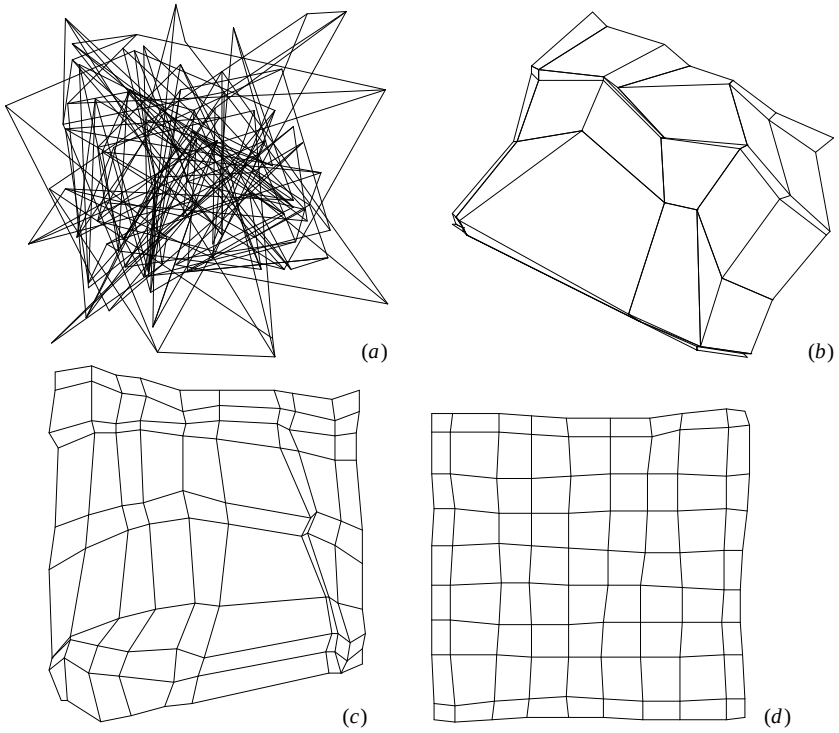

Figure C2.1.6. Locations and neighboring relations between neurons after (*a*) 0 iterations, (*b*) 100 iterations, (*c*) 1000 iterations, and (*d*) 10 000 iterations, for a uniform distribution of inputs.

is rarely found in such low-input-dimension examples; the idea of organization principles is, however, easily expandable to higher dimensions.

The way to easily represent a Kohonen map (in two dimensions) is to locate in the input space each neuron by the coordinates formed by its weight vector. Moreover, to represent the neighborhood relations in the output space, neighboring neurons are linked in the representation. Before learning, there is no ordering since the weights are randomly initialized, and the network resembles a dish of spaghetti. Progressively, however, a local ordering relationship develops in the map, so that two neurons which are neighbors in the map will be close by their locations in the input space. Figure C2.1.6 shows the locations and neighboring relations of a 10×10 Kohonen map, respectively, after 0, 100, 1000, and 10 000 learning iterations; the simulation was carried out with a uniform distribution of input patterns in a square area. The final distribution of neurons is effectively quite uniform, except on the sides of the map where a border effect can be noticed.

Of course, mapping a two-dimensional uniform distribution onto a two-dimensional square map is not the most useful application of a Kohonen map. Figure C2.1.7 shows the mapping of a uniform triangular distribution onto a square Kohonen map; the mapping cannot be perfect because of the border and corner effects, but one can see that the *vector quantization* property is well achieved, while the topological neighborhood property is maintained at least locally. C1.1.5

Figure C2.1.8 shows the mapping of a nonuniform distribution (left-hand side of the figure) by a 10×10 Kohonen map. As expected, the density of neurons approximates the density of points in the distribution, which proves the vector quantization effect.

Finally, figure C2.1.9 shows how a three-dimensional distribution of points which are effectively arranged on a two-dimensional surface maps onto a two-dimensional Kohonen grid. From such an example, it can be concluded that the nonlinear projection realized by a Kohonen map will be optimal when the dimension of the map is approximately equal to the intrinsic dimension of the data.

C2.1.1.9 Kohonen map and principal component analysis

The last example above clearly shows the projection property of Kohonen maps: points in a high-dimensional space are projected onto a low-dimensional (usually one- or two-dimensional) space. If

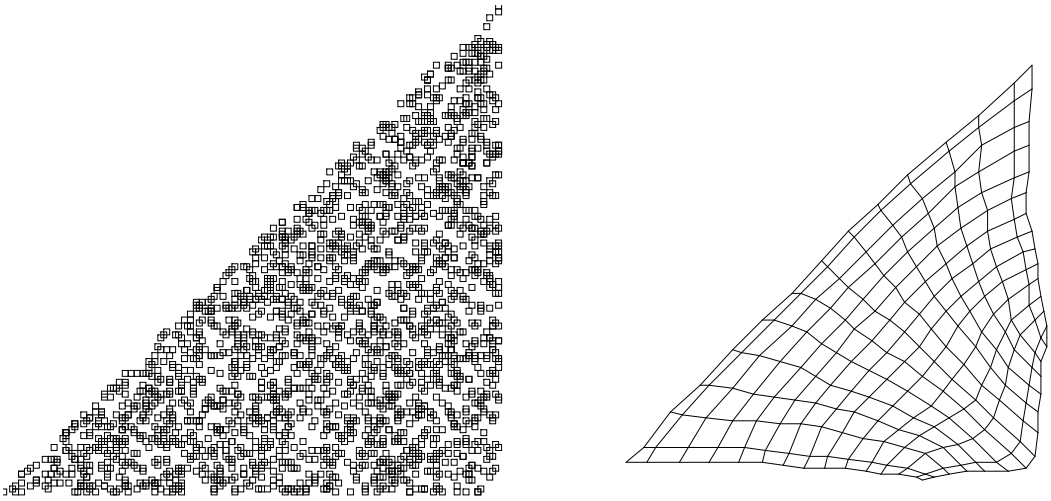

Figure C2.1.7. Final locations and neighboring relations between neurons for a uniform distribution of inputs in a triangle.

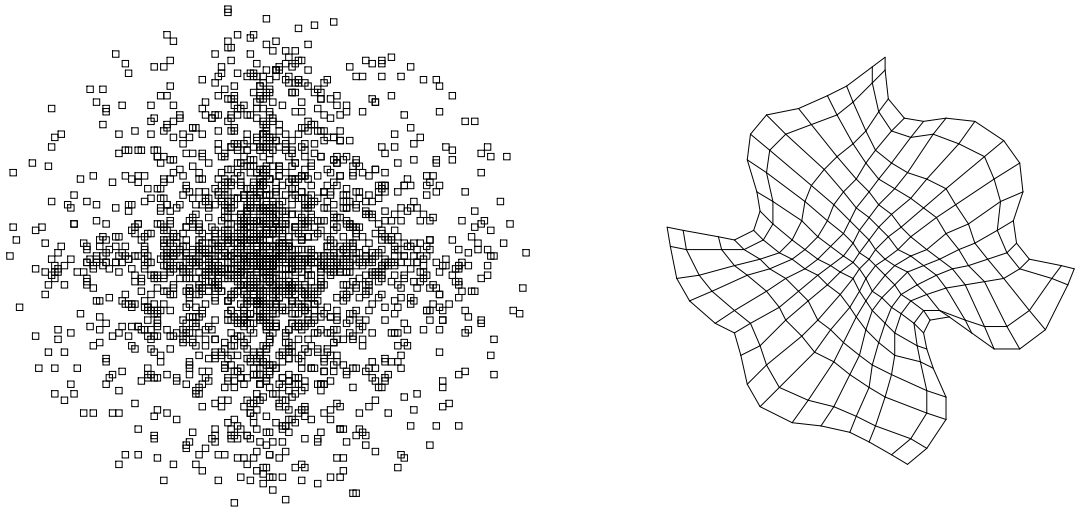

Figure C2.1.8. Final locations and neighboring relations between neurons for a nonuniform distribution of inputs.

we do not take into account the quantization property of Kohonen maps (which can be 'bypassed' by some linear or nonlinear interpolation between adjacent nodes in the map if necessary) the projection is similar to the one obtained with principal component analysis (PCA) according to the fact that in both cases the projected space is chosen to best fit the initial distribution.

The main difference, of course, relies on the nonlinear projection in the case of the Kohonen map, while PCA is purely linear; this can be a strong advantage in many situations.

C2.1.1.10 Related neural network models

The principle of auto-organization, which is the key aspect of the self-organizing feature map, is at the basis of many neural network models, more or less derived from Kohonen's one. Among these extensions, Fritzke's *growing cell structures* are of primary interest. The organization principles of Fritzke's network C2.4 are similar to those of the Kohonen map; the main difference relies, however, on the fact that the map in Fritzke's network is not fixed *a priori* as in Kohonen's model, but is built during learning. The dimension of the map d_{out}, its number of neurons, and the connectivity between them are progressively adapted to form a structure more adapted to the input distribution. Other differences with respect to Kohonen's model

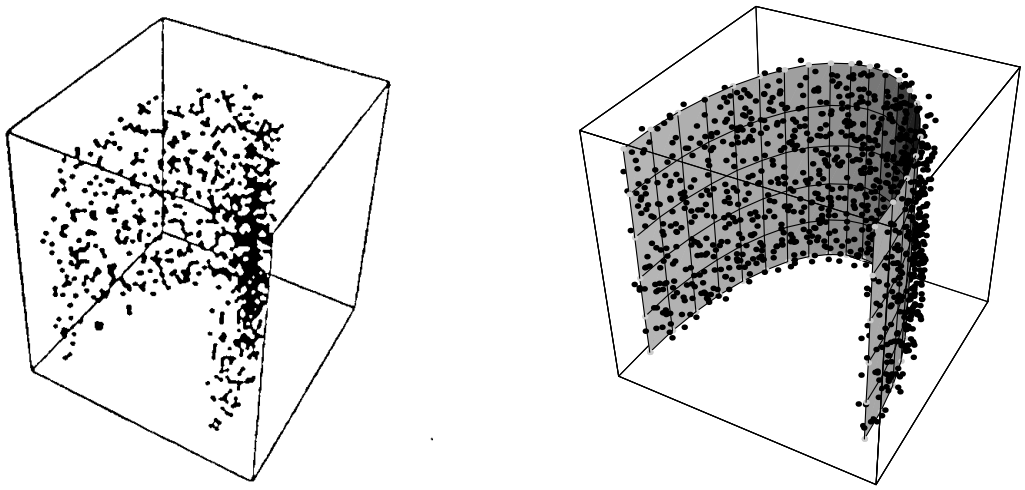

Figure C2.1.9. Final locations and neighboring relations between neurons for three- to two-dimensional mapping of inputs arranged on a two-dimensional surface.

are a fixed neighborhood limited to the direct neighbors, and fixed adaptation parameters. The principle of the network growth is to add cells in regions where neurons are most frequently chosen as the best-match ones, in order to obtain similar firing rates for all neurons, which is a characteristic of good quantization.

The Kohonen self-organizing map is also related to many adaptive models of vector quantization, such as K-means algorithm, maximum-entropy clustering, and neural gas, and the like. The principle of any vector quantization method is to approximate a continuous generally nonuniform distribution of vectors (or a discrete distribution with a large number of vectors) by another distribution being formed by a much smaller number of units of the same dimension. If we consider the weight vector of each neuron as a unit in the input space, a Kohonen network implements this vector quantization scheme. The above-mentioned methods of vector quantization find their place in the field of artificial neural networks (ANNs): they are adaptive, may be represented in the form of a feedforward graph, and may be described by the same formalism as used in many other neural models, including Kohonen's. For this reason, in the next section we will describe the 'neural-gas' method, a vector quantization method known in the field of neural networks as a powerful algorithm that quickly converges to low distortion errors, generally reaches a final distortion under that obtained by other algorithms, and obeys a gradient descent on an energy function surface, like the K-means clustering, but unlike Kohonen's algorithm.

C2.1.2 Neural gas

C2.1.2.1 Introduction

It is a tricky question to know whether adaptive vector quantization techniques find their place in the frame of ANNs or not. We do not pretend to give an answer to this question, but would like to point out some common points between methods which are usually classified as 'artificial neural networks' and vector quantization techniques. Neural networks are adaptive techniques, which can usually be represented in the form of a graph of computational units, linked by synaptic weights. Neural networks have a biological origin, although this is not acknowledged in all models of learning. Neural networks usually realize some kind of function or distribution approximation, or classification.

In that order of ideas, many adaptive methods of vector quantization resemble neural networks. Algorithms like K-means clustering or any derived method such as maximum-entropy clustering and frequency-sensitive learning are also adaptive, can be represented as a similar graph of computational units, and realize some kind of distribution (or probability density) estimation. It is not our intention here, however, to describe all vector quantization methods, since it would greatly exceed the scope of a handbook of neural networks. However, it would not be fair to fully omit and completely dissociate

them from the field of neural networks; their common characteristics, and the advantages that both vector quantization and neural networks can take from a common approach, fully justify their insertion here.

Vector quantizers are feedforward unsupervised models: they have no 'teacher', and there is no feedback between the positioning of the neurons and the distribution of input vectors. Rather than going into the details of many similar (while different) vector quantization algorithms, we will focus our discussion on one model, the neural gas, which seems to have three advantages with respect to most other methods (Martinetz *et al* 1993):

- it converges quickly
- it reaches a lower distortion error after convergence than with other methods
- it obeys a gradient descent on an energy function surface, which is not the case for example with the Kohonen maps.

The neural-gas algorithm has been developed by Thomas Martinetz and Klaus Schulten and a good description of the model and its applications can be found in the article by Martinetz *et al* (1993).

C2.1.2.2 Purpose of the model

The purpose of the neural-gas model is to quantize multidimensional vectors, that is, to transform an initial large set of d_{in}-dimensional vectors into a reduced set of N vectors (neurons) of the same dimension, with a minimal mean distortion between each of the vectors in the initial set and its best-match neuron, the best-match neuron being defined as the vectors in the final set closest to the input vector. Each neuron will thus define in the input space a so-called Voronoi region, being the set of locations closer to this neuron than to any other one; this property is identical to the vector quantization property of Kohonen's algorithm, while the topology conservation property does not hold in the neural-gas model. In comparison with other adaptive vector quantization techniques, the neural-gas algorithm is fast, converges to low average distortions, and obeys a gradient descent on a known energy function surface.

C2.1.2.3 Topology

A neural-gas network has one layer of neurons and one layer of connections. As in the Kohonen model, each d_{in}-dimensional neuron is connected to the input vector $\boldsymbol{x}$ of the same dimension, through a weight vector $\boldsymbol{w}$. There is no connection between units in the neuron layer.

C2.1.2.4 Learning

The neuron activations in the neural-gas model may be defined as the distance between the input vector $\boldsymbol{x}$ and weight $\boldsymbol{w}_i$ associated with neuron n_i:

$$a_i = \|\boldsymbol{w}_i - \boldsymbol{x}\| \tag{C2.1.8}$$

where a_i is the activation of neuron n_i.

Learning, i.e. adaptation of weights $\boldsymbol{w}_i$ to a distribution of input vectors, goes through the selection of the best-match neuron n_b (of weight $\boldsymbol{w}_b$) at each presentation of an input vector $\boldsymbol{x}(t)$ to the network:

$$\|\boldsymbol{x}(t) - \boldsymbol{w}_b\| \leq \|\boldsymbol{x}(t) - \boldsymbol{w}_i\| \qquad \forall 1 \leq i \leq N\,. \tag{C2.1.9}$$

After selection of the winner, the weights of several neurons are adapted, as in the Kohonen map. However, the neurons to adapt are not selected according to a topological relation with the best-match neuron n_b, but are selected according to the rank they have in the ordered list of distances between their weights and the input vector.

Each time an input vector $\boldsymbol{x}(t)$ is presented to the network, all neurons are 'ranked' according to their distance from the input; $\boldsymbol{w}_{b_0}$ is the weight of the closest neuron to $\boldsymbol{x}$ (according to the Euclidean distance), $\boldsymbol{w}_{b_1}$ is the second-closest neuron to $\boldsymbol{x}$, and so on. In other words, the ranks b_i, $1 \leq i \leq N-1$, are determined according to

$$\|\boldsymbol{x} - \boldsymbol{w}_{b_0}\| \leq \|\boldsymbol{x} - \boldsymbol{w}_{b_1}\| \leq \ldots \leq \|\boldsymbol{x} - \boldsymbol{w}_{b_{N-1}}\|\,. \tag{C2.1.10}$$

After ranking, neuron weights are adapted according to

$$\boldsymbol{w}_i(t+1) - \boldsymbol{w}_i(t) = \alpha\beta(j)(\boldsymbol{x} - \boldsymbol{w}_i(t)) \tag{C2.1.11}$$

where α is an adaptation factor, j is chosen so that $b_j = i$, and $\beta(j)$ is a monotonic decreasing function. Usually, function $\beta(j)$ is chosen according to

$$\beta(j) = \mathrm{e}^{-j/\lambda} \tag{C2.1.12}$$

where λ is a decay constant. For $\lambda \to 0$, the algorithm becomes equivalent to the K-means method.

It can be proven that the neural-gas algorithm obeys a gradient descent on the surface defined by the energy function

$$E(\boldsymbol{w}, \lambda) = \frac{1}{2C(\lambda)} \sum_{i=1}^{N} \int P(\boldsymbol{x}(t)) \beta j (\boldsymbol{x} - \boldsymbol{w}_i)^2 \, \mathrm{d}x \tag{C2.1.13}$$

where $P(\boldsymbol{x})$ is the distribution of input vectors $\boldsymbol{x}(t)$, j is chosen so that $b_j = i$, the integral is taken over the whole input space, and

$$C(\lambda) = \sum_{j=0}^{N-1} \beta(j) \, . \tag{C2.1.14}$$

A proof of this can be found in the article by Martinetz *et al* (1993). As a stochastic gradient descent algorithm on a bounded function, the neural-gas algorithm will converge, but may be trapped in one of the local minima of the energy function (C2.1.13). It must also be noticed that the parameter λ usually decreases with time, a large parameter corresponding to a smooth energy function, while when $\lambda \to 0$ the energy function (C2.1.13) becomes equivalent to the energy function of the K-means algorithm. Reducing the value of λ with time thus reduces the risk of being trapped in a 'bad' local minimum of function (C2.1.13).

The neural-gas learning algorithm in natural language representation is:

(i) compute the Euclidean distance between the input vector and the weight vector associated with the first neuron in the map;
(ii) repeat step (i) for all neurons in the map;
(iii) rank the neurons by their respective Euclidean distances to the input neuron;
(iv) update the first neuron in the network by adding a fraction of the difference between the input vector and the weight of the neuron, multiplied by a factor depending on its rank;
(v) repeat step (iv) for all neurons in the network.

The learning rule is basically local: the operation at each neuron only requires the knowledge of the weight associated with that neuron and of the input vector. However, a nonlocal function (but implementable as a tree structure) is needed to rank the neurons according to their distance from the input vector.

C2.1.2.5 Examples of results

An illustration of the neural-gas algorithm behavior in two dimensions is given in figure C2.1.10. The input distribution (left-hand part of the figure) is formed by three Gaussian distributions, of which two overlap. The right-hand side of the figure illustrates the positions of the neurons after convergence. It can be seen that more neurons are concentrated in the center of the Gaussian functions than in the tails; this was expected as more points from the input distribution are concentrated in these regions too.

The neurons seem also to be concentrated in the tails of the distribution which is in contradiction to the vector quantization principle. In fact, this visual effect is due to the fact that the Voronoi regions of the neurons in the tails of the Gaussian functions are very narrow but long. The (large) size of the Voronoi regions associated with these neurons, compared to the regions associated with neurons in the centers of the distribution, is thus in accordance with the (low) density of points in the tails.

C2.1.2.6 Related neural network models

As already mentioned above, the neural-gas algorithm is closely linked to Kohonen's self-organizing feature maps, because of its vector quantization property. While the neural gas does not have the topology-preservation property of Kohonen maps, it achieves a similar quantization of input data. In that case, the neural-gas algorithm is also usually faster and more accurate (lower final mean-square error) than other vector quantization methods.

Figure C2.1.10. Input distribution formed by three Gaussian functions in two dimensions, and the corresponding neuron locations after convergence of the neural-gas algorithm.

Many other vector quantization methods exist which are more or less similar to the neural gas; including them in the neural network field or not is a question of personal feeling. We can, however, mention that most of them are derived from or have links to the K-means Lloyd and MacQueen algorithm, which uses the same principle as the above-described neural-gas method, except that only the best-match neuron is moved at each presentation of an input vector; the ranking procedure is thus not used. Several methods also try to avoid confinement to local minima of the distortion function, by adding some probabilistic term in the best-match neuron choice, or by moving several neurons at each presentation of an input vector, depending on the distance between the corresponding weights and this input, and so on.

C2.1.3 Neocognitron

C2.1.3.1 Introduction

The neocognitron has been developed by Kunihiko Fukushima (Osaka University, Department of Biophysical Engineering, Toyonaka, Osaka 560, Japan). It has been described in many research papers and conference communications since 1980; a good description of the network and of the learning process can be found in the article by Fukushima (1988). In this section, we will follow Fukushima's description from that reference.

The neocognitron is presented as a biologically plausible network. It can indeed be proved that in the visual cortex and in the higher areas there are cells which respond to very simple patterns like segments of lines or curves at an early stage, and then respond to more complex figures (squares, circles, and the like) at a higher stage, and so on. The neocognitron is based on the same hierarchical structure; cells in the first layers extract simple patterns, while they become progressively more complex up to the last layers of the network.

The neocognitron is devoted to pattern extraction, in applications where this property of extracting features of increasing complexity may be exploited, for example in optical character recognition, where simple features are combined to recognize more complex characters.

C2.1.3.2 Purpose of the model

As will be described in the next section the neocognitron is formed by an even number of layers, odd layers having adaptable input connections and even layers having fixed input ones. Each odd layer is intended to detect features in the plane formed by the preceding layer; in the first stages of the neocognitron, the detected features are very simple, such as line segments, circles, and angles. Each neuron in an odd layer will be adapted to detect a particular feature in a particular location of the preceding layer. In the even layers, fixed connections between each neuron in that layer and a set of close neurons in the previous one are intended to cancel the effect of small shifts in position of the features detected in the previous layer.

The neocognitron is thus intended to detect complex patterns in a plane of cells, through the progressive detection of simple to elaborated features from the first to the last layers. One of its applications, largely described by the author of the model, is *optical character recognition* (OCR), where simple features (strokes) may be combined to detect more complex patterns (the characters). G1.2, G1.3

C2.1.3.3 Topology

The topology of a neocognitron is described in figure C2.1.11. It contains an even number of layers (the input plane U_0, having only the task of redistributing the inputs on the different neurons of the first layer U_{S1}, is not counted here as a 'layer'). Layers U_{Si} and U_{Ci} alternate in the network, their functionality being different. Neurons used in the S-layers (odd layers) are typical nonlinear neurons with excitatory and inhibitory inputs, and with positive outputs. Neurons in C-layers (even layers) act as OR functions (they fire if one of their inputs fires).

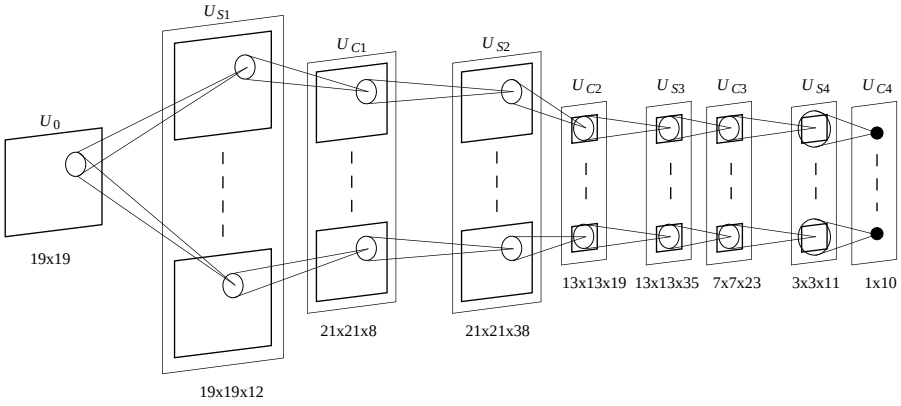

Figure C2.1.11. Layer structure of the neocognitron (according to Fukushima 1988).

All neuron layers are divided into 'cell planes', being represented as squares in figure C2.1.11; each cell plane contains neurons which detect similar features at different locations in the previous layer; similarly, neurons at corresponding locations in the different cell planes of a layer respond to different features at the same location of the previous layer. The numbers of cells indicated at the bottom of figure C2.1.11 are given as an example (these values are those given in the application example described by Fukushima 1988).

Each S-layer contains S-cells and inhibitory V-cells. The S-cells are feature extracting cells; after learning, they respond to specific features at specific locations in the preceding layer. In general, features extracted at lower stages of the network are elementary (line segments, strokes, branchings, and the like), while those extracted at upper stages are more complex (e.g. characters).

C-cells in the C-layers are inserted in the network to allow for shifts in the feature detection in the preceding S-layers. Because of their (local) OR function, C-cells fire when one of the S-cells in a local neighborhood of the location corresponding to the C-cell, but in the previous layer, is activated; this functionality makes the network less sensitive to shifts in the position of the detected features. Since C-layers are each inserted between two S-layers, this property applies to the features detected at all stages of the network; simple features in one layer of the network being elements of more complex features in the subsequent layers, this property also allows for some insensitivity to distortion of complex features.

Subsidiary V-cells in the S-layers have another functionality. These cells have fixed connections to the same neurons in the preceding layer as their associated S-cells (there are as many V-cells as S-cells in S-layers) have variable connections to. The outputs of the V-cells are inhibitory connections to their associated S-cells. Their objective is mainly a winner-take-all functionality, to allow only one neuron firing at a time in one cell plane.

C2.1.3.4 Learning

Learning in a neocognitron is primarily unsupervised: neurons in S-layers find themselves the patterns they have to extract, according to a competitive adaptive scheme. Some supervised learning can, however, be added in the unsupervised scheme to have more control over the kind of feature that will be extracted by each S-layer.

Two principles are at the basis of learning in the neocognitron. The first one concerns the reinforcement of maximum-output cells, the second one the development of iterative connections.

The first principle consists in reinforcing the variable connections (to the *S*-layers) under two conditions:

(i) the cell receiving the connection has a stronger activation than other cells in its neighborhood;

(ii) the cell sending out the connection has a nonzero activation value.

The weight change at this connection is then proportional to the activation of the neuron sending out the connection.

Connections to the *S*-cells are thus adapted so that the *S*-cells will match the template presented at the previous layer. Inhibitory *V*-cells whose outputs are also connected to the *S*-cells fire when the pattern presented in the previous layer at the location scanned by the *S*- and *V*-cells does not match the feature already learned by the cell; the *V*-cells thus watch for irrelevant features, and increase the ability of the network to differentiate between different features. This differentiation property is also due to the first condition mentioned above, that a single cell in a small area will have its connections reinforced when a pattern is presented: the connections leading to this cell already match more or less the presented feature (since it is the neuron with the largest activation) and are even reinforced in that direction because of the proportionality between the reinforcement and the feature.

The other principle leading to self-organization consists in selecting seed cells. In each hypercolumn, defined as the group of *S*-cells in a layer detecting features at approximately the same location in the previous layer, the cell that fires most is chosen as a candidate for seed cells. However, only one seed cell can be selected in each cell plane; if two *S*-cells are selected as candidates in the same cell plane, only the one having the greatest activation will be selected.

Having selected (maximum) one seed cell in each cell plane, all other *S*-cells in that plane grow in order to have the spatial distribution of their inputs identical to that of the seed cell. This ensures that all cells in one cell plane will respond to the same feature, but of course at different locations. In contrast, all cells in the same hypercolumn will respond to different features at the same locations.

In order to guarantee that the auto-organization principle of the network can take place, small random values must be attributed as initial conditions for the adaptable weights; if all weights were null at the beginning of the process, no selection of winner could occur, and the competitive process could not be initiated.

The cooperative and simultaneous action of these two principles is at the basis of the auto-organization of the network, where the *S*-cells will progressively adapt their connections to extract the most frequent features detected when input patterns are presented.

Let us finally mention that supervised learning can easily be inserted into the network, by replacing the automatic selection of seed cells by a teacher's choice. This is possible at any stage of the network, and can greatly help in order to add available knowledge in the network (such as conventional similarity between visually different characters in OCR).

C2.1.3.5 Related neural network models

The neocognitron is not 'similar' to other widely used models in the neural network field. Basically, it is much more complex, has more layers, more weights, and more neurons than most other networks. The learning process is also complex, and not obvious to implement. In fact, an implementation of the neocognitron requires a lot of 'fine-tuning' effort; the size of the network is obviously dependent on the application, and some tricks can be given to guide the choice of the number of layers, neurons, cell planes, and so on. There are also parameters in the equations of the network (both use and learning) which are not obvious to choose, such as the exact form of the activation functions, the neuron thresholds, and so on. This is why we did not go into the details of the equations, since this would have largely exceeded the scope of this description.

However, in the opinion of the author, the network has remarkable properties of adaptation to the examples given to the network. Mixing unsupervised and supervised learning can also be a determining advantage, to benefit from the inherent properties of the network when no supplementary information is available, but also to allow for the insertion of knowledge into the network when this knowledge is available.

References

Cottrell M, Fort J C and Pages G 1994 Two or three things that we know about the Kohonen algorithm *Proc. Eur. Symp. on Artificial Neural Networks* (Brussels: D facto) pp 235–44

Erwin E, Obermayer K and Shulten K 1992 Self-organizing maps: ordering, convergence properties and energy functions *Biol. Cyber.* **67** 47–55

Fukushima K 1988 Neocognitron: a hierarchical neural network capable of visual pattern recognition *Neural Networks* **1** 119–30

Kohonen T 1989 *Self-Organization and Associative Memory* (Berlin: Springer)

Martinetz T M, Berkovich S G and Schulten R J 1993 'Neural-Gas' network for vector quantization and its application to time-series prediction *IEEE Trans. Neural Networks* **4** 558–69

Robbins H and Monro S 1951 A stochastic approximation method *Ann. Math. Stat.* **22** 400–7

von der Malsburg C 1973 Self-organization of orientation sensitive cells in the striate cortex *Kybernetik* **14** 85–100