mywbut.com

CHAPTER 14

Self-Organizing Networks

14. 1 INTRODUCTION

In the previous chapters we discussed a number of networks, which were trained to perform a mapping F: $\Re'' \rightarrow \Re^m$ by presenting the network 'examples' (x^p, d^p) with $d^p = F(x^p)$ of this mapping. However, problems exist where such training data, consisting of input and desired output pairs are not available, but where the only information is provided by a set of input patterns x^p. In these cases the relevant information has to be found within the (redundant) training samples x^p.

Some examples of such problems are:

- **Clustering:** the input data may be grouped in 'clusters' and the data processing system has to find these inherent clusters in the input data. The output of the system should give the cluster label of the input pattern (discrete output);
- **Vector quantisation:** this problem occurs when a continuous space has to be discretized. The input of the system is the n-dimensional vector x, the output is a discrete representation of the input space. The system has to find optimal discretization of the input space;
- **Dimensionality reduction:** the input data are grouped in a subspace, which has lower dimensionality than the dimensionality of the data. The system has to learn an optimal mapping, such that most of the variance in the input data is preserved in the output data;
- **Feature extraction:** the system has to extract features from the input signal. This often means a dimensionality reduction as described above.

In this chapter we discuss a number of neuro-computational approaches for these kinds of problems. Training is done without the presence of an external teacher. The unsupervised weight adapting algorithms are usually based on some form of global competition between the neurons.

There are very many types of self-organizing networks, applicable to a wide area of problems. One of the most basic schemes is competitive learning as proposed by Rumelhart and Zipser (1985). A very similar network but with different emergent properties is the topology-conserving map devised by Kohonen. Other self-organizing networks are ART, proposed by Carpenter and Grossberg (1987), and Fukushima (1975).

mywbut.com

14.2 COMPETITIVE LEARNING

14.2.1 Clustering

Competitive learning is a learning procedure that divides a set of input patterns in clusters that are inherent to the input data. A competitive learning network is provided only with input vectors x and thus implements an unsupervised learning procedure. We will show its equivalence to a class of 'traditional' clustering algorithms shortly. Another important use of these networks is vector quantisation.

An example of a competitive learning network is shown in Fig. 14.1. All output units o are connected to all input units i with weights w_{io}. When an input pattern x is presented, only a single output unit of the network (the winner) will be activated. In a correctly trained network, all x in one cluster will have the same winner. For the determination of the winner and the corresponding learning rule, two methods exist.

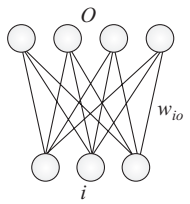

Fig. 14.1 A simple competitive learning network. Each of the four outputs o is connected to all inputs i.

Winner Selection: Dot Product

For the time being, we assume that both input vectors x and weight vectors w_o are normalized to unit length. Each output unit o calculates its activation value y_o according to the dot product of input and weight vector:

$$y_o = \sum_i w_{io} x_i = w_o^T x \qquad ...(14.1)$$

In a next pass, output neuron k is selected with maximum activation

$$\forall_o \neq k : y_o \leq y_k \qquad ...(14.2)$$

Activations are reset such that $y_k = 1$ and $y_{o \neq k} = 0$. This is the competitive aspect of the network, and we refer to the output layer as the winner-take-all layer. The winner-take-all layer is usually implemented in software by simply selecting the output neuron with highest activation value. This function can also be performed by a neural network known as MAXNET (Lippmann, 1989). In MAXNET, all neurons o are connected to other units $o0$ with inhibitory links and to itself with an excitatory link:

$$w_{o,o} = \begin{cases} -\varepsilon & \text{if } o \neq o' \\ +1 & \text{otherwise} \end{cases} \qquad ...(14.3)$$

mywbut.com

It can be shown that this network converges to a situation where only the neuron with highest initial activation survives, whereas the activations of all other neurons converge to zero. From now on, we will simply assume a winner k is selected without being concerned which algorithm is used.

Once the winner k has been selected, the weights are updated according to:

$$w_k(t+1) = \frac{w_k(t) + \gamma(x(t) - w_k(t))}{\|w_k(t) + \gamma(x(t) - w_k(t))\|} \quad ...(14.4)$$

where the divisor ensures that all weight vectors w are normalized. Note that only the weights of winner k are updated.

The weight update given in equation (14.4) effectively rotates the weight vector w_o towards the input vector x. Each time an input x is presented; the weight vector closest to this input is selected and is subsequently rotated towards the input. Consequently, weight vectors are rotated towards those areas where many inputs appear: the clusters in the input. This procedure is visualized in Fig. 14.2.

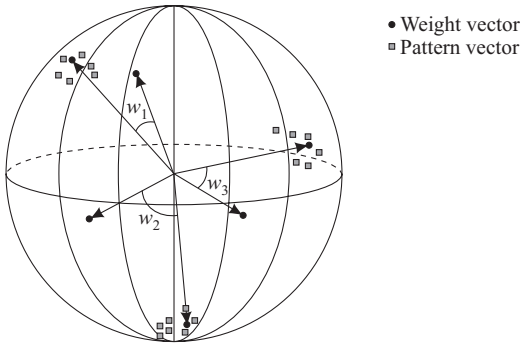

Fig. 14.2 Example of clustering in 3*D* with normalized vectors, which all lie on the unity sphere. The three weight vectors are rotated towards the centers of gravity of the three different input clusters.

Winner selection: Euclidean distance

Previously it was assumed that both inputs x and weight vectors w were normalized. Using the activation function given in equation (14.1) gives a 'biological plausible' solution. In Fig. 14.3 it is shown how the algorithm would fail if normalized vectors were to be used. Naturally one would like to accommodate the algorithm for normalized input data. To this end, the winning neuron k is selected with its weight vector w_k closest to the input pattern x, using the Euclidean distance measure:

$$k: \|w_k - x\| \leq \|w_o - x\| \ \forall_o \quad ...(14.5)$$

It is easily checked that equation (14.5) reduces to (14.1) and (14.2) if all vectors are normalized. The Euclidean distance norm is therefore a more general case of equations (14.1) and (14.2). Instead of rotating the weight vector towards the input as performed by equation (14.4), the weight update must be changed to implement a shift towards the input:

$$w_k(t+1) = w_k(t) + \gamma(x(t) - w_k(t)) \quad ...(14.6)$$

mywbut.com

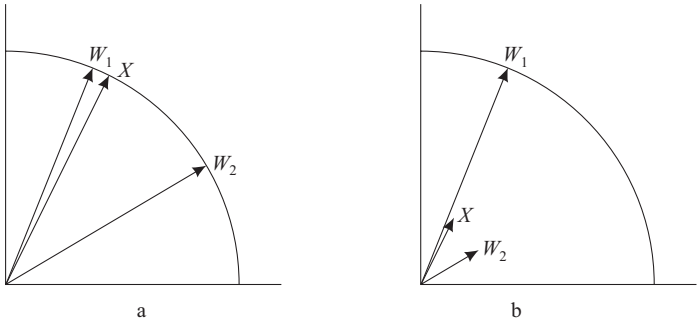

Fig. 14.3 Determining the winner in a competitive learning network. a. Three normalized vectors. b. The three vectors having the same directions as in a., but with different lengths. In *a*., vectors x and w_1 are nearest to each other, and their dot product $x^T w_1 = |x||w_1| \cos \alpha$ is larger than the dot product of x and w_2. In b., however, the pattern and weight vectors are not normalized, and in this case w_2 should be considered the 'winner' when x is applied. However, the dot product $x^T w_1$ is still larger than $x^T w_2$.

Again only the weights of the winner are updated.

A point of attention in these recursive clustering techniques is the initialization. Especially if the input vectors are drawn from a large or high-dimensional input space, it is not beyond imagination that a randomly initialized weight vector w_o will never be chosen as the winner and will thus never be moved and never be used. Therefore, it is customary to initialize weight vectors to a set of input patterns $\{x\}$ drawn from the input set at random. Another more thorough approach that avoids these and other problems in competitive learning is called leaky learning. This is implemented by expanding the weight update given in equation (14.6) with

$$w_l(t+1) = w_l(t) + \gamma'(x(t) - w_l(t)) \; \forall_l \neq k \qquad ...(14.7)$$

with $\gamma' < \gamma$ the leaky learning rate. A somewhat similar method is known as frequency sensitive competitive learning (Ahalt, Krishnamurthy, Chen, & Melton, 1990). In this algorithm, each neuron records the number of times it is selected winner. The more often it wins, the less sensitive it becomes to competition. Conversely, neurons that consistently fail to win increase their chances of being selected winner.

Cost function: Earlier it was claimed, that a competitive network performs a clustering process on the input data. i.e., input patterns are divided in disjoint clusters such that similarities between input patterns in the same cluster are much bigger than similarities between inputs in different clusters. Similarity is measured by a distance function on the input vectors, as discussed before. A common criterion to measure the quality of a given clustering is the square error criterion, given by

$$E = \sum_p \|w_k - x^p\|^2 \qquad ...(14.8)$$

where k is the winning neuron when input x_p is presented. The weights w are interpreted as cluster centres. It is not difficult to show that competitive learning indeed seeks to find a minimum for this square error by the negative gradient of the error-function.

mywbut.com

Theorem 14.1: The error function for pattern x^p

$$E^p = \sum_p \|w_k - x^p\|^2 \qquad ...(14.9)$$

where k is the winning unit, is minimised by the weight update rule in eq. (14.6).

Proof: As in eq. (3.12), we calculate the effect of a weight change on the error function. So we have that

$$\Delta_p W_{io} = -\gamma \frac{\partial E^p}{\partial w_{io}} \qquad ...(14.10)$$

where γ is a constant of proportionality. Now, we have to determine the partial derivative of E^p:

$$\frac{\partial E^p}{\partial w_{io}} = \begin{cases} w_{io} - x_i^p & \text{if unit } o \text{ wins} \\ 0 & \text{otherwise} \end{cases} \qquad ...(14.11)$$

such that

$$\Delta_p w_{io} = -\gamma(w_{io} - x_i^p) = \gamma(x_o^p - w_{io}) \qquad ...(14.12)$$

which is eq. (14.6) written down for one element of w_o.

Therefore, eq. (14.8) is minimized by repeated weight updates using eq. (14.6).

Example 14.1: In Fig. 14.4, 8 clusters of each 6 data points are depicted. A competitive learning network using Euclidean distance to select the winner was initialized with all weight vectors $w_o = 0$. The network was trained with $\gamma = 0{:}1$ and a $\gamma' = 0{:}001$ and the positions of the weights after 500 iterations are shown.

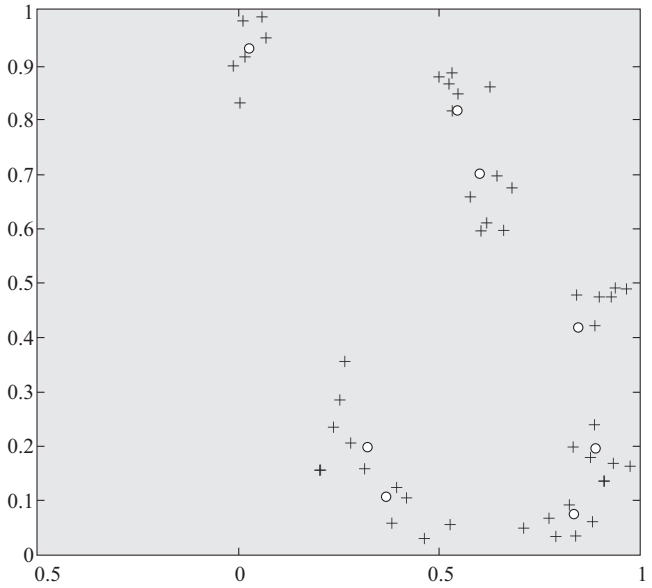

Fig. 14.4 Competitive learning for clustering data. The data are given by "+". The positions of the weight vectors after 500 iterations is given by "*o*".

mywbut.com

14.2.2 Vector Quantisation

Another important use of competitive learning networks is found in vector quantisation. A vector quantisation scheme divides the input space in a number of disjoint subspaces and represents each input vector x by the label of the subspace it falls into (i.e., index k of the winning neuron). The difference with clustering is that we are not so much interested in finding clusters of similar data, but more in quantising the entire input space. The quantisation performed by the competitive learning network is said to 'track the input probability density function': the density of neurons and thus subspaces is highest in those areas where inputs are most likely to appear, whereas a more coarse quantisation is obtained in those areas where inputs are scarce. An example of tracking the input density is sketched in Figure 14.5. Vector quantisation through competitive learning results in a more fine-grained discretization in those areas of the input space where most input have occurred in the past.

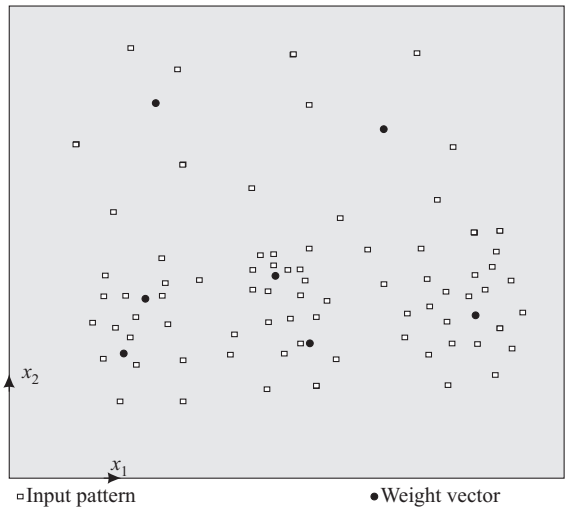

Fig. 14.5 This figure visualizes the tracking of the input density. The input patterns are drawn from $\Re^2$; the weight vectors also lie in $\Re^2$. In the areas where inputs are scarce, the upper part of the figure, only few (in this case two) neurons are used to discretized the input space. Thus, the upper part of the input space is divided into two large separate regions. The lower part, however, where many more inputs have occurred, five neurons discretized the input space into five smaller subspaces.

In this way, competitive learning can be used in applications where data has to be compressed such as telecommunication or storage. However, competitive learning has also be used in combination with supervised learning methods, and be applied to function approximation problems or classification problems. We will describe two examples: the "counter propagation" method and the "learning vector quantisation".

14.2.3 Counter Propagation

In a large number of applications, networks that perform vector quantisation are combined with another type of network in order to perform function approximation. An example of such a network is given in

mywbut.com

Fig. 14.6. This network can approximate a function $f: \Re^n \rightarrow \Re^m$ by associating with each neuron o a function value $[w_{1o}; w_{2o}, \ldots, w_{mo}]^T$ which is somehow representative for the function values $f(x)$ of inputs x represented by o. This way of approximating a function effectively implements a 'look-up table': an input x is assigned to a table entry k with $\forall_o \neq k$: $||x - w_k|| \leq ||x - w_o||$, and the function value $[w_{1k}; w_{2k}, \ldots, w_{mk}]^T$ in this table entry is taken as an approximation of $f(x)$. A well-known example of such a network is the Counter propagation network (Hecht-Nielsen, 1988).

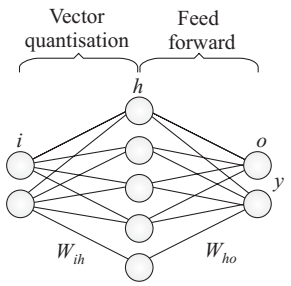

Fig. 14.6 A network combining a vector quantisation layer with a 1 layer feed forward neural network. This network can be used to approximate functions from $\Re^2$ to $\Re^2$, the input space $\Re^2$ is discretized in 5 disjoint subspaces.

Depending on the application, one can choose to perform the vector quantisation before learning the function approximation, or one can choose to learn the quantisation and the approximation layer simultaneously. As an example of the latter, the network presented in Fig. 14.6 can be supervisedly trained in the following way:

1. Present the network with both input x and function value $d = f(x)$;
2. Perform the unsupervised quantisation step. For each weight vector, calculate the distance from its weight vector to the input pattern and find winner k. Update the weights w_{ih} with equation (14.6);
3. Perform the supervised approximation step:

$$w_{ko}(t+1) = w_{ko}(t) + \gamma(d_o - w_{ko}(t)) \qquad ...(14.13)$$

This is simply the δ – rule with $y_o = \sum_h y_h w_{ho} = w_{ko}$ when k is the winning neuron and the desired output is given by $d = f(x)$.

If we define a function $g(x, k)$ as:

$$g(x, k) = \begin{cases} 1 & \text{if } k \text{ is winner} \\ 0 & \text{otherwise} \end{cases} \qquad ...(14.14)$$

It can be shown that this learning procedure converges to

$$w_{ho} = \int_{\Re^n} y_o g(x, h) dx \qquad ...(14.15)$$

mywbut.com

i.e., each table entry converges to the mean function value over all inputs in the subspace represented by that table entry. As we have seen before, the quantisation scheme tracks the input probability density function, which results in a better approximation of the function in those areas where input is most likely to appear.

Not all functions are represented accurately by this combination of quantisation and approximation layers. e.g., a simple identity or combinations of sines and cosines are much better approximated by multilayer back-propagation networks if the activation functions are chosen appropriately. However, if we expect our input to be (a subspace of) a high dimensional input space $<n$ and we expect our function f to be discontinuous at numerous points, the combination of quantisation and approximation is not uncommon and probably very efficient. Of course this combination extends itself much further than the presented combination of the presented single layer competitive learning network and the single layer feed-forward network. The latter could be replaced by a reinforcement learning procedure (see chapter 15).

The quantisation layer can be replaced by various other quantisation schemes, such as Kohonen networks or octree methods (Jansen, Smagt, and Groen, 1994). In fact, various modern statistical function approximation methods (Breiman, Friedman, Olshen, and Stone, 1984; Friedman, 1991) are based on this very idea, extended with the possibility to have the approximation layer influence the quantisation layer (e.g., to obtain a better or locally more fine-grained quantisation).

14.2.4 Learning Vector Quantisation

It is an unpleasant habit in neural network literature, to also cover Learning Vector Quantisation (LVQ) methods in chapters on unsupervised clustering. Granted that these methods also perform a clustering or quantisation task and use similar learning rules, they are trained supervisedly and perform discriminant analysis rather than unsupervised clustering. These networks attempt to define 'decision boundaries' in the input space, given a large set of exemplary decisions (the training set); each decision could, e.g., be a correct class label.

A rather large number of slightly different LVQ methods is appearing in recent literature. They are all based on the following basic algorithm:

1. With each output neuron o, a class label (or decision of some other kind) y_o is associated;
2. A learning sample consists of input vector x^p together with its correct class label y_o^p;
3. Using distance measures between weight vectors w_o and input vector x^p, not only the winner k_1 is determined, but also the second best k_2:
$$||x_p - w_{k1}|| < ||x^p - w_{k_2}|| < ||x^p - w_i|| \; \forall_o \neq k_1, k_2$$
4. The labels $y_{k_1}^p$, $y_{k_2}^p$ are compared with d^p. The weight update rule given in equation (6.6) is used selectively based on this comparison.

An example of the last step is given by the LVQ2 algorithm by Kohonen (1977), using the following strategy:

- if $y_{k_1}^p \neq d^p$ and $d^p = y_{k_2}^p$
- and $||x^p - w_{k_2}|| - ||x^p - w_{k_1}|| < \varepsilon$
- then $w_{k_2}(t+1) = w_{k_2}(t) + \gamma(x - w_{k_2}(t))$
- and $w_{k_1}(t+1) = w_{k_1}(t) + \gamma(x - w_{k_1}(t))$

mywbut.com

i.e., w_{k_2} with the correct label is moved towards the input vector, while w_{k_1} with the incorrect label is moved away from it.

The new LVQ algorithms that are emerging all use different implementations of these different steps, e.g., how to define class labels y_o, how many 'next-best' winners are to be determined, how to adapt the number of output neurons i and how to selectively use the weight update rule.

14.3 KOHONEN NETWORK

The Kohonen network (1982, 1984) can be seen as an extension to the competitive learning network, although this is chronologically incorrect. Also, the Kohonen network has a different set of applications.

In the Kohonen network, the output units in S are ordered in some fashion, often in a two-dimensional grid or array, although this is application-dependent. The ordering, which is chosen by the user, determines which output neurons are neighbours.

Now, when learning patterns are presented to the network, the weights to the output units are thus adapted such that the order present in the input space $\Re^2$ is preserved in the output, i.e., the neurons in S. This means that learning patterns which are near to each other in the input space (where 'near' is determined by the distance measure used in finding the winning unit) must be mapped on output units, which are also near to each other, i.e., the same or neighboring units. Thus, if inputs are uniformly distributed in $\Re^N$ and the order must be preserved, the dimensionality of S must be at least N.

The mapping, which represents a discretization of the input space, is said to be topology preserving. However, if the inputs are restricted to a subspace of $\Re^N$, a Kohonen network can be used of lower dimensionality. For example: data on a two- dimensional manifold in a high dimensional input space can be mapped onto a two-dimensional Kohonen network, which can for example be used for visualization of the data.

Usually, the learning patterns are random samples from $\Re^N$. At time t, a sample $x(t)$ is generated and presented to the network. Using the same formulas as in section 6.1, the winning unit k is determined. Next, the weights to this winning unit as well as its neighbours are adapted using the learning rule

$$w_o(t+1) = w_o(t) + \gamma g(o, k)(x(t) - w_o(t)) \qquad ...(14.16)$$

Here, $g(o, k)$ is a decreasing function of the grid-distance between units o and k, such that $g(k, k) = 1$. For example, for $g(\ )$ a Gaussian function can be used, such that (in one dimension!) $g(o, k) = \exp(-(o-k)^2)$ (see Fig. 14.7). Due to this collective learning scheme, input signals, which are near to each other, will be mapped on neighbouring neurons. Thus the topology inherently present in the input signals will be preserved in the mapping, such as depicted in Fig. 14.8.

If the intrinsic dimensionality of S is less than N, the neurons in the network are 'folded' in the input space, such as depicted in Fig. 14.9.

The topology-conserving quality of this network has many counterparts in biological brains. The brain is organized in many places so that aspects of the sensory environment are represented in the form of two-dimensional maps. For example, in the visual system, there are several topographic mappings of visual space onto the surface of the visual cortex. There are organized mappings of the body surface

mywbut.com

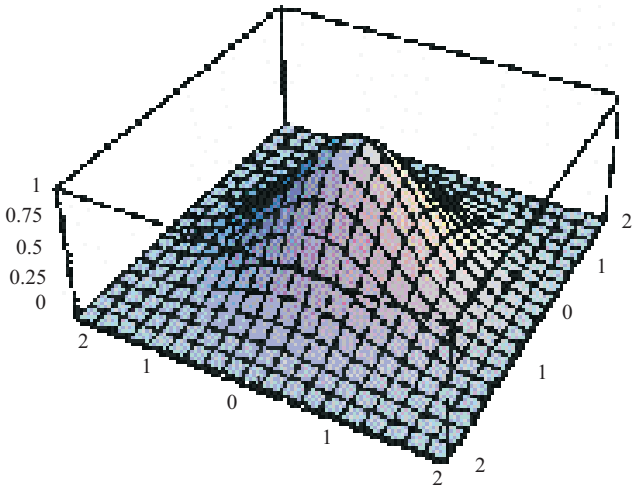

Fig. 14.7 Gaussian neuron distance function g( ). In this case, g( ) is shown for a two dimensional grid because it looks nice.

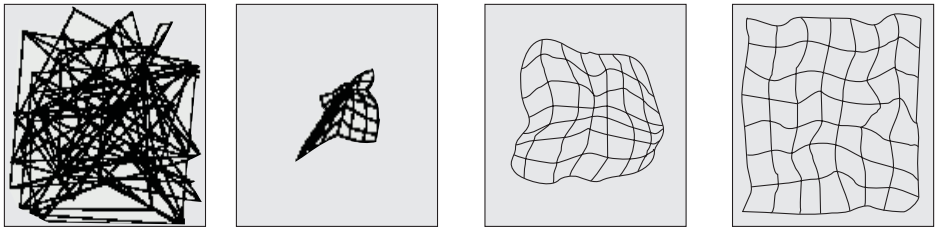

Fig. 14.8 A topology conserving map converging. The weight vectors of a network with two inputs and 8 x 8 output neurons arranged in a planar grid are shown. A line in each figure connects weight $w_{i,(o_1 o_2)}$ with weights $w_{i,(o_1+1, o_2)}$ and $w_{i,(i_1 i_2+1)}$. The leftmost figure shows the initial weights; the rightmost when the map is almost completely formed.

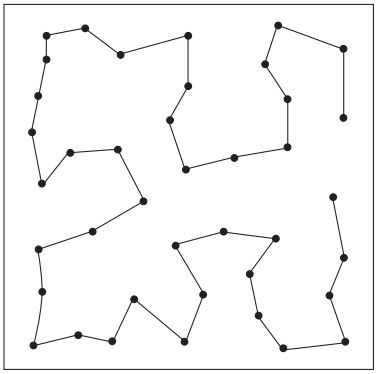

Fig. 14.9 The mapping of a two dimensional input space on a one dimensional Kohonen network.

mywbut.com

onto the cortex in both motor and somatosensory areas, and tonotopic mappings of frequency in the auditory cortex. The use of topographic representations, where some important aspect of a sensory modality is related to the physical locations of the cells on a surface, is so common that it obviously serves an important information processing function.

It does not come as a surprise, therefore, that already many applications have been devised of the Kohonen topology-conserving maps. Kohonen himself has successfully used the network for phoneme-recognition (Kohonen, Makisara, and Saramaki, 1984). Also, the network has been used to merge sensory data from different kinds of sensors, such as auditory and visual, 'looking' at the same scene (Gielen, Krommenhoek, and Gisbergen, 1991).

To explain the plausibility of a similar structure in biological networks, Kohonen remarks that the lateral inhibition between the neurons could be obtained via efferent connections between those neurons. In one dimension, those connection strengths form a 'Mexican hat' (see Figure 14.10).

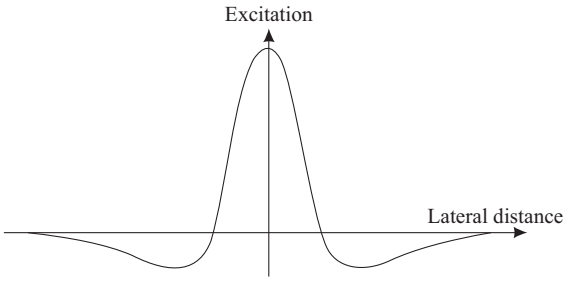

Fig. 14.10 Mexican hat. Lateral interaction around the winning neuron as a function of distance: excitation to nearby neurons, inhibition to farther off neurons.

14.4 PRINCIPAL COMPONENT NETWORKS

The networks presented in the previous sections can be seen as (nonlinear) vector transformations, which map an input vector to a number of binary output elements or neurons. The weights are adjusted in such a way that they could be considered as prototype vectors (vectorial means) for the input patterns for which the competing neuron wins.

The self-organizing transform described in this section rotates the input space in such a way that the values of the output neurons are as uncorrelated as possible and the energy or variances of the patterns is mainly concentrated in a few output neurons. An example is shown in Figure 14.11.

The two dimensional samples (x_1, x_2) are plotted in the figure. It can be easily seen that x_1 and x_2 are related, such that if we know x_1 we can make a reasonable prediction of x_2 and vice versa since the points are centered around the line $x_1 = x_2$. If we rotate the axes over $\pi/4$ we get the (e_1, e_2) axis as plotted in the figure. Here the conditional prediction has no use because the points have uncorrelated coordinates. Another property of this rotation is that the variance or energy of the transformed patterns is maximized on a lower dimension. This can be intuitively verified by comparing the spreads (d_{x_1}, d_{x_2}) and (d_{e_1}, d_{e_2}) in the figures. After the rotation, the variance of the samples is large along the e_1 axis and small along the e_2 axis.

mywbut.com

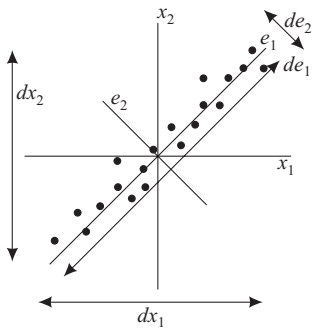

Fig. 14.11 Distribution of input samples.

This transform is very closely related to the eigenvector transformation known from image processing where the image has to be coded or transformed to a lower dimension and reconstructed again by another transform as well as possible.

The next section describes a learning rule which acts as a Hebbian learning rule, but which scales the vector length to unity. In the subsequent section we will see that a linear neuron with a normalised Hebbian learning rule acts as such a transform, extending the theory in the last section to multi-dimensional outputs.

14.4.1 Normalized Hebbian Rule

The model considered here consists of one linear neuron with input weights w. The output $y_o(t)$ of this neuron is given by the usual inner product of its weight w and the input vector x:

$$y_o(t) = w(t)^T x(t) \qquad ...(14.17)$$

As seen in the previous sections, all models are based on a kind of Hebbian learning. However, the basic Hebbian rule would make the weights grow uninhibitedly if there were correlation in the input patterns. This can be overcome by normalising the weight vector to a fixed length, typically 1, which leads to the following learning rule

$$w(t+1) = \frac{w(t) + \gamma y(t) x(t)}{L(w(t) + \gamma y(t) x(t))} \qquad ...(14.18)$$

where $L(\bullet)$ indicates an operator which returns the vector length, and γ is a small learning parameter. Compare this learning rule with the normalized learning rule of competitive learning. There the delta rule was normalized, here the standard Hebb rule is.

Now the operator which computes the vector length, the norm of the vector, can be approximated by a Taylor expansion around $\gamma = 0$:

$$L(w(t)) + \gamma y(t)\, x(t) = 1 + \gamma \left.\frac{\partial L}{\partial \gamma}\right|_{\gamma=0} + O(\gamma^2) \qquad ...(14.19)$$

mywbut.com

When we substitute this expression for the vector length in equation (6.18), it resolves for small $\gamma(t^2)$.

$$w(t+1) = (w(t) + \gamma y(t)\, x(t)) \left(1 - \gamma \frac{\partial L}{\partial \gamma}\bigg|_{\gamma=0} + o(\gamma^2) \right) \qquad ...(14.20)$$

Since $\frac{\delta L}{\delta \gamma}\bigg|_{\gamma=0} = y(t)^2$ discarding the higher order terms of γ leads to

$$w(t+1) = w(t) + \gamma y(t)\, x(t)(x(t) - y(t) w(t)) \qquad ...(14.21)$$

which is called the 'Oja learning rule' (Oja, 1982). This learning rule thus modifies the weight in the usual Hebbian sense, the first product terms is the Hebb rule $y_o(t)\, x(t)$, but normalizes its weight vector directly by the second product term $-y_o(t)\, x(t)\, w(t)$. What exactly does this learning rule do with the weight vector?

14.4.2 Principal Component Extractor

Remember probability theory? Consider an N-dimensional signal $x(t)$ with

- Mean $\mu = E(x(t))$;
- Correlation matrix $R = E((x(t) - \mu)\, (x(t) - \mu)^T)$.

In the following we assume the signal mean to be zero, so $\mu = 0$.

From equation (6.21) we see that the expectation of the weights for the Oja learning rule equals

$$E(w(t+1)|w(t)) = w(t) + \gamma(Rw(t) - (w(t)^T R w(t)) w(t)) \qquad ...(14.22)$$

which has a continuous counterpart

$$\frac{d}{dt} w(t) = Rw(t) - (w(t)^T R w(t))\, w(t) \qquad ...(14.23)$$

Theorem 14.2: Let the eigenvectors e_i of R be ordered with descending associated eigenvalues λ_i such that $\lambda_1 > \lambda_2 > ... > \lambda_N$. With equation (6.23) the weights $w(t)$ will converge to $\pm e_1$.

Proof: 1 Since the eigenvectors of **R** span the N-dimensional space, the weight vector can be decomposed as

$$w(t) = \sum_i^N \beta_i(t) e_i \qquad ...(14.24)$$

Substituting this in the differential equation and concluding the theorem is left as an exercise.

14.4.3 More Eigenvectors

In the previous section it was shown that a single neuron's weight converges to the eigenvector of the correlation matrix with maximum eigenvalue, i.e., the weight of the neuron is directed in the direction

mywbut.com

of highest energy or variance of the input patterns. Here we tackle the question of how to find the remaining eigenvectors of the correlation matrix given the first found eigenvector.

Consider the signal x which can be decomposed into the basis of eigenvectors e_i of its correlation matrix **R**,

$$x = \sum_{i}^{N} \alpha_i e_i \qquad ...(14.25)$$

If we now subtract the component in the direction of e_1, the direction in which the signal has the most energy, from the signal x

$$\tilde{x} = x - \alpha_1 e_1 \qquad ...(14.26)$$

we are sure that when we again decompose ~x into the eigenvector basis, the coefficient $\alpha_1 = 0$, simply because we just subtracted it. We call $\tilde{x}$ the deflation of x.

If now a second neuron is taught on this signal $\tilde{x}$, then its weights will lie in the direction of the remaining eigenvector with the highest eigenvalue. Since the deflation removed the component in the direction of the first eigenvector, the weight will converge to the remaining eigenvector with maximum eigenvalue. In the previous section we ordered the eigenvalues in magnitude, so according to this definition in the limit we will find e_2. We can continue this strategy and find all the N eigenvectors belonging to the signal x.

We can write the deflation in neural network terms if we see that

$$y_o = w^T x = e_1^T \sum_{i}^{N} \alpha_i e_i = \alpha_i \qquad ...(14.27)$$

since

$$w = e_1 \qquad ...(14.28)$$

So that the deflated vector $\tilde{x}$ equals

$$\tilde{x} = x - y_o w \qquad ...(14.29)$$

The term subtracted from the input vector can be interpreted as a kind of a back-projection or expectation. Compare this to ART described in the next section.

14.5 ADAPTIVE RESONANCE THEORY

The last unsupervised learning network we discuss differs from the previous networks in that it is recurrent; as with networks in the next chapter, the data is not only fed forward but also back from output to input units.

14.5.1 Background: Adaptive Resonance Theory

In 1976, Grossberg introduced a model for explaining biological phenomena. The model has three crucial properties:

mywbut.com

1. A normalization of the total network activity. Biological systems are usually very adaptive to large changes in their environment. For example, the human eye can adapt itself to large variations in light intensities;
2. Contrast enhancement of input patterns. The awareness of subtle differences in input patterns can mean a lot in terms of survival. Distinguishing a hiding panther from a resting one makes all the difference in the world. The mechanism used here is contrast enhancement.
3. Short-term memory (STM) storage of the contrast-enhanced pattern. Before the input pattern can be decoded, it must be stored in the short-term memory. The long-term memory (LTM) implements an arousal mechanism (i.e., the classification), whereas the STM is used to cause gradual changes in the LTM.

The system consists of two layers, F_1 and F_2, which are connected to each other via the LTM (see

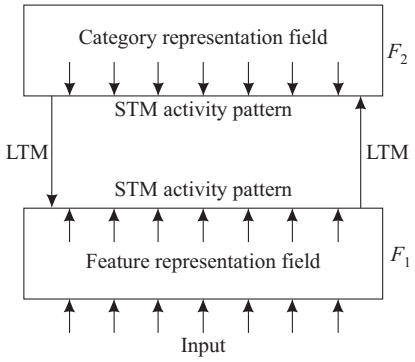

Fig. 14.12 The ART architecture.

Fig. 14.12). The input pattern is received at F_1, whereas classification takes place in F_2. As mentioned before, the input is not directly classified. First a characterization takes place by means of extracting features, giving rise to activation in the feature representation field.

The expectations, residing in the LTM connections, translate the input pattern to a categorization in the category representation field. The classification is compared to the expectation of the network, which resides in the LTM weights from F_2 to F_1. If there is a match, the expectations are strengthened otherwise the classification is rejected.

14.5.2 ART1: The Simplified Neural Network Model

The ART1 simplified model consists of two layers of binary neurons (with values 1 and 0), called F_1 (the comparison layer) and F_2 (the recognition layer) (see Fig. 14.13). Each neuron in F_1 is connected to all neurons in F_2 via the continuous-valued forward long term memory (LTM) W^f, and vice versa via the binary-valued backward LTM W^b. The other modules are gain 1 and 2 (G_1 and G_2), and a reset module.

Each neuron in the comparison layer receives three inputs: a component of the input pattern, a component of the feedback pattern, and a gain G_1. A neuron outputs a 1 if and only if at least three of these inputs are high: the ‘two-thirds rule’.

mywbut.com

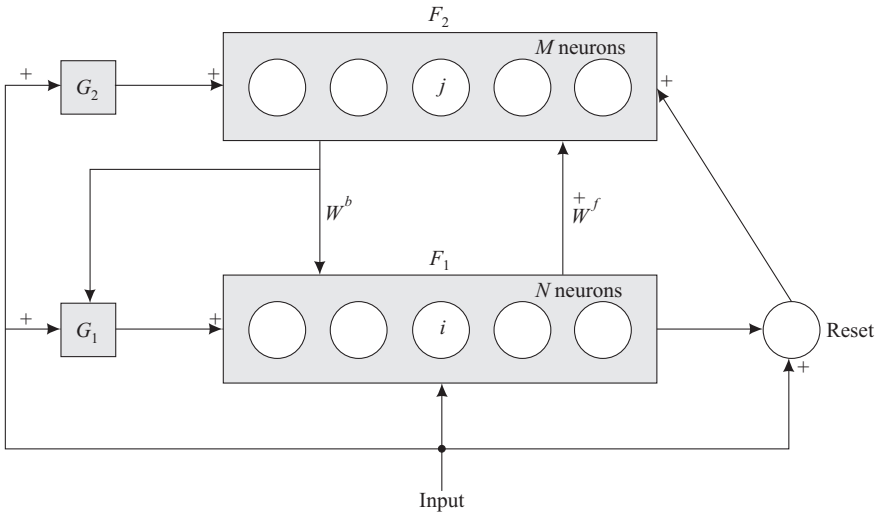

Fig. 14.13 The ART 1 neural network.

The neurons in the recognition layer each compute the inner product of their incoming (continuous-valued) weights and the pattern sent over these connections. The winning neuron then inhibits all the other neurons via lateral inhibition.

Gain 2 is the logical 'or' of all the elements in the input pattern x.

Gain 1 equals gain 2, except when the feedback pattern from F_2 contains any 1; then it is forced to zero.

Finally, the reset signal is sent to the active neuron in F_2 if the input vector x and the output of F_1 differ by more than some vigilance level.

14.5.3 Operation

The pattern is sent to F_2, and in F_2 one neuron becomes active. This signal is then sent back over the backward LTM, which reproduces a binary pattern at F_1. Gain 1 is inhibited, and only the neurons in F_1 which receive a 'one' from both x and F_2 remain active.

If there is a substantial mismatch between the two patterns, the reset signal will inhibit the neuron in F_2 and the process is repeated.

Instead of following Carpenter and Grossberg's description of the system using differential equations, we use the notation employed by Lippmann (1987):

1. Initialization:

$$w_{ji}^{b}(0) = 1$$

$$w_{ji}^{f} = \frac{1}{1+N}$$

mywbut.com

where N is the number of neurons in F_1, M the number of neurons in F_2, $0 \le i < N$, and $0 \le j < M$. Also, choose the vigilance threshold ρ, $0 \le \rho \le 1$;

2. Apply the new input pattern x;
3. Compute the activation values of the neurons in F_2:

$$y_i' = \sum_{j=1}^{N} w_{ij}^f(t) x_1 \qquad ...(14.30)$$

4. Select the winning neuron $k (0 \le k < M)$;
5. Vigilance test: if

$$\frac{w_k^b(t) \circ x}{x \circ x} > \rho \qquad ...(14.31)$$

where o denotes inner product, go to step 7, else go to step 6. Note that $w_k^b \circ x$ essentially is the inner product $x^* \circ x$, which will be large if x^* and x are near to each other;

6. Neuron k is disabled from further activity. Go to step 3;
7. Set for all l, $o \le l < N$:

$$w_{kl}^b(t+1) = w_{kl}^b(t) x_l$$

$$w_{lk}^f(t+1) = \frac{w_{kl}^b(t) x_l}{\frac{1}{2} + \sum_{i=1}^{N} w_{ki}^b(t) x_i}$$

8. Re-enable all neurons in F_2 and go to step 2.

Fig. 14.14 shows exemplar behaviour of the network.

14.5.4 ART 1: The Original Model

In later work, Carpenter and Grossberg (1987) present several neural network models to incorporate parts of the complete theory. We will only discuss the first model, ART 1.

The network incorporates a follow-the-leader clustering algorithm (Hartigan, 1975). This algorithm tries to fit each new input pattern in an existing class. If no matching class can be found, i.e., the distance between the new pattern and all existing classes exceeds some threshold, a new class is created containing the new pattern.

The novelty in this approach is that the network is able to adapt to new incoming patterns, while the previous memory is not corrupted. In most neural networks, such as the back- propagation network, all patterns must be taught sequentially; the teaching of a new pattern might corrupt the weights for all previously learned patterns. By changing the structure of the network rather than the weights, ART1 overcomes this problem.

mywbut.com

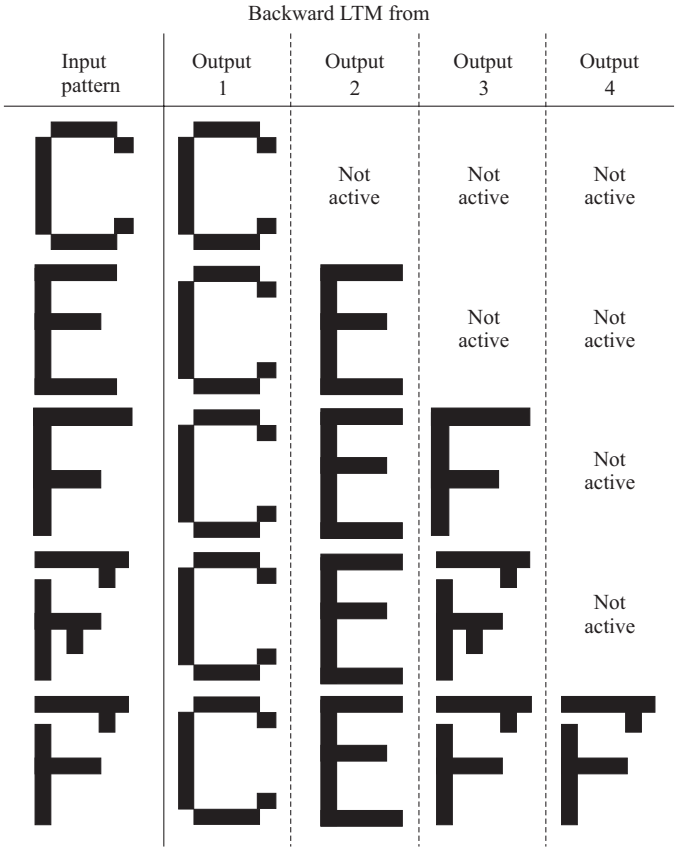

Fig. 14.14 An example of the behaviour of the Carpenter Grossberg network for letter patterns. The binary input patterns on the left were applied sequentially. On the right the stored patterns (i.e., the weights of W^b for the first four output units) are shown.

14.5.5 Normalization of the Original Model

We will refer to a cell in F_1 or F_2 with k.

Each cell k in F_1 or F_2 receives an input s_k and respond with an activation level y_k.

In order to introduce normalization in the model, we set $I = \sum s_k$ and let the relative input intensity $\Theta_k = s_k I^{-1}$.

So we have a model in which the change of the response y_k of an input at a certain cell k

- depends inhibitorily on all other inputs and the sensitivity of the cell, i.e., the surroundings of each cell have a negative influence on the cell $-y_k \sum_{l \neq k} s_l$;
- has an excitatory response as far as the input at the cell is concerned $+Bs_k$;

mywbut.com

- has an inhibitory response for normalization $-y_k s_k$;
- has a decay $-Ay_k$.

Here, A and B are constants. The differential equation for the neurons in F_1 and F_2 now is

$$\frac{dy_k}{dt} = -Ay_k + (B - yk)s_k - y_k \sum_{l \neq k} s_l \qquad ...(14.32)$$

with $0 \leq y_k(0) \leq B$ because the inhibitory effect of an input can never exceed the excitatory input.

At equilibrium, when $\frac{dy_k}{dt} = 0$, and with $I = \sum s_k$ we have that

$$y_k(A + 1) = Bs_k \qquad ...(14.33)$$

Because of the definition of $\Theta_k = s_k I^{-1}$ we get

$$y_k = \Theta_k \frac{BI}{A + I} \qquad ...(14.34)$$

Therefore, at equilibrium y_k is proportional to Θ_k, and, since

$$\frac{BI}{A + I} \leq B \qquad ...(14.35)$$

The total activity $y^{\text{total}} = \sum y_k$ never exceeds B: it is normalized.

14.5.6 Contrast Enhancement

In order to make F_2 react better on differences in neuron values in F_1 (or vice versa), contrast enhancement is applied: the contrasts between the neuronal values in a layer are amplified. We can show that eq. (14.32) does not suffice anymore. In order to enhance the contrasts, we chop off all the equal fractions (uniform parts) in F_1 or F_2. This can be done by adding an extra inhibitory input proportional to the inputs from the other cells with a factor C:

$$\frac{dy_k}{dt} = -Ay_k + (B - y_k)s_k - (y_k + C) \sum_{l \neq k} s_l \qquad ...(14.36)$$

At equilibrium, when we set $B = (n - 1)\, C$ where n is the number of neurons, we have

$$y_k = \frac{nCI}{A + I}\left(\Theta_k - \frac{1}{n}\right) \qquad ...(14.37)$$

Now, when an input in which all the s_k are equal is given, then all the y_k are zero: the effect of C is enhancing differences. If we set $B \leq (n - 1)C$ or $C/(B + C) \geq 1/n$, then more of the input shall be chopped off.

The description of ART1 continues by defining the differential equations for the LTM. Instead of following Carpenter and Grossberg's description, we will revert to the simplified model as presented by Lippmann.