6 Kohonen Network

learning objectives

- concept of "topology"
- "mapping" a dataset from a space of high dimension to a space of lower dimension
- how a neural network (the Kohonen net) can do mapping
- how "unsupervised" learning is used when you don't have a set of "correct" answers
- how topology is maintained in mapping, as in representing the relationships between continents on a globe, on a planar map

6.1 General

When we think of "data", we ordinarily think of values, magnitudes, signs, etc.; this is an **algebraic** view of a dataset. In addition, there is an **information science** view, which focuses on the relationships among data items. These relationships may exist entirely within the given dataset, or may involve data in other datasets as well.

Complex information is always incomplete, to some extent. In fact, we may **choose** to reduce the dataset, as when digital images are compressed to reduce storage requirements. In dealing with missing data, we must not forget their possible relationships. When we focus on the relationships among data, rather than their algebraic attributes, we say that we are dealing with the *topology* of the information. Figure 6-1 contrasts the concept of topology with the concept of numerical values.

Efficiency is obviously crucial in handling large amounts of information; for a given level of hardware competence, efficiency is generally achieved by compressing the data. *Compression* may be thought of as a process of *mapping* a multidimensional input into an output space of significantly smaller dimension (Figure 6-2).

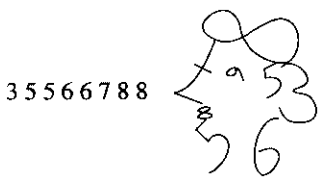

Figure 6-1: A completely new level of information can be revealed by taking into account the topology between the numerals (b) instead of considering their numerical values (a) only.

Obviously, we want a maximum of compression and a minimum of information loss; this is one of the basic problems in information and computer science in general, and in artificial intelligence and neural network research in particular. There are, of course, many questions involved in this problem. For example: how can we make the trade-off between reduction and preservation when the information has not yet been processed? Can we map information onto a two-dimensional array of neurons? How can such a mapping be performed or learned? How can the retained knowledge be retrieved from the mapped information?

Teuvo Kohonen has introduced the very interesting concept of *self-organized topological feature maps*, which are maps that preserve the topology of a multidimensional representation within the new one- or two-dimensional array of neurons (Reference 6-1). We will discuss Kohonen's approach to neural networks, which attempts to preserve the topology of the input information while mapping it into the neural array.

The concept of topology (or better still, the concept of "preservation of topology") has become the essential feature of the Kohonen approach in neural network research. As he has pointed out in his book (Reference 6-2) the mapping of multidimensional information into a two-dimensional plane of neurons "seems to be a fundamental operation in the formation of abstractions too!" He argued that topological relations should be preserved in this mapping.

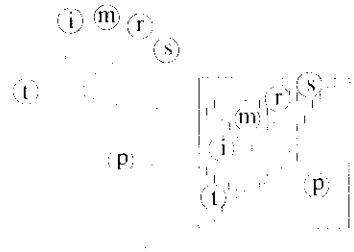

Figure 6-2: The topology of the five fingers (t, i, m, r, s) and the palm (p) (above) is preserved or "mapped" onto a square plane of (13 x 13) neurons (below) using the Kohonen algorithm, which will be explained later.

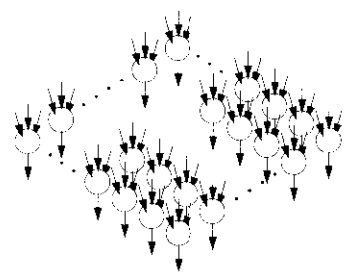

Figure 6-3: Two-dimensional layout of the Kohonen neural network.

6.2 Architecture

The Kohonen network is probably the closest of all artificial neural network architectures and learning schemes to the biological neuron network. As a rule, the Kohonen type of network is based on a single layer of neurons arranged in a one-dimensional array or in a two-dimensional plane having a well defined topology (Figure 6-3 and Figure 6-4).

A defined topology means that each neuron has a defined number of neurons as nearest neighbors, second-nearest neighbors, etc. To be in accordance with the previous cases the neurons for the Kohonen layer are visualised by circles (Figure 6-3). However, from the didactic as well as from the mathematical point of view the visualisation of neurons in the Kohonen network is more natural in the form of columns (Figure 6-4, above). The advantage of a scheme with

column-like neurons is its clear presentation of the weights in individual neurons and how the weights handling the same input variable are connected together in the network. It can easily be seen that weights affected by each variable are lying on a single and well-defined level of weights. Each set of weights affected either by the first, the second, or by the third input variable are forming a separate level of weights. The levels of weights are superimposed onto each other in a one-to-one-correspondence, hence the weights of each neuron are obtained by looking at the weights in all levels that are exactly aligned in a vertical column. There are as many weight levels in each Kohonen network as there are input variables describing the objects for which the network is designed. Because the input vector consists of three variables, there are also three levels of weights in the case shown on Figure 6-4.

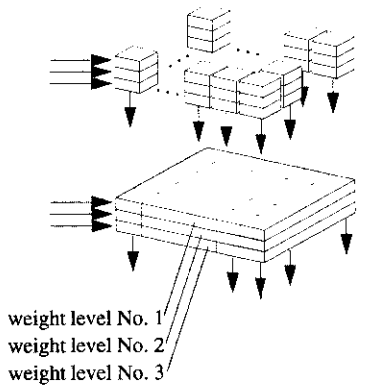

Figure 6-4: Neurons can as well be drawn as little boxes (above). The three inputs are coming from the side to all neurons at the same time. Schematically all neurons (small boxes) can be "packed" into a larger box or "brick" in which neurons are represented as columns (below).

The *neighborhood* of a neuron is usually arranged either in squares or in hexagons, which means that each neuron has either four or six nearest neighbors. The concept of "nearest neighbors" needs some elaboration, especially for those who have studied crystals or coordination chemistry. For example, the square neighborhood is often regarded as having eight and not four nearest neighbors (Figure 6-5). Certainly, the corner points in the rectangular grid are further away from the central point compared to the actual first neighbors; but we are interested in the topology, that is, the **connections** and not the actual **distances**. In the Kohonen conception of neural networks, the signal similarity is related to the spatial (topological) relations among the neurons in the network.

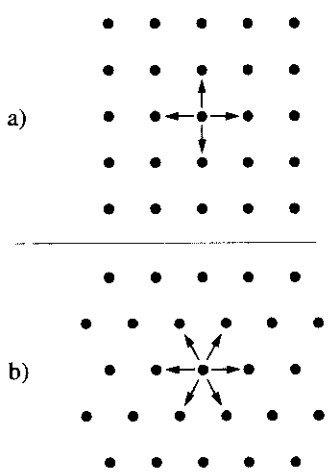

Figure 6-5: Square (a) and hexagonal (b) layout of neighbors.

The Kohonen concept tries to map the input so that similar signals excite neurons that are very close together (in terms of spatial distance); this similarity-to-distance relationship should be generalized to include the entire range of similarity relations between different signals as well. Kohonen learning represents an attempt to fit the signal space onto the neural network by a kind of "smoothing" or "reshaping" procedure.

> The aim of Kohonen learning is to map similar signals to similar neuron positions.

A practical point: we need to ensure that each neuron in the net has the same number of first-, second-, etc. neighbors (Figure 6-6); but the

net is a finite object, with edges. What about the neurons at the edges? Well, this wouldn't be a problem if the net were, say, a torus.

Figure 6-7 shows how a plane can be wrapped into a toroid: first, the upper edge (or the row of neurons located there) wraps around and links to the lower edge, and then the left edge joins the right one. Of course, we don't actually manipulate the net; we simply convert the indices of edge neurons so that they appear to wrap around (Figure 6-8).[1]

The coordinates (indices) of the neurons located in the k-th neighboring ring (Figure 6-6) around a particular neuron will be labeled $(x_{m1},y_{m2})_k$. The following example shows how, for an (11 x 20)-neuron plane, we would calculate the coordinate pairs of all neurons in the forth neighboring ring around neuron $(3, 2)_c$. ("c" is used to designate the central neuron:

(In applications, the following algorithm for finding the neighbors at a certain topological distance from the center may be executed more efficiently because the entire neighborhood can be found by going through these loops.)

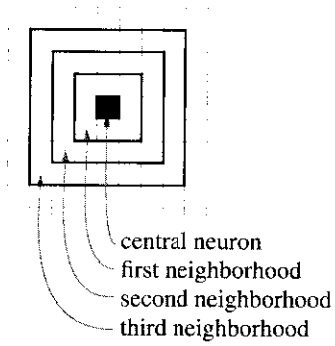

Figure 6-6: The square neighborhood is usually regarded as having 8, 16, 24, etc. neighbors in concentric neighborhoods.

1. The *modulus* (written MOD(A, M) in some programming languages, or "*a mod m*" in ordinary mathematical notation) is a function of two parameters a and m. It represents the remainder after division of the first parameter by the second. For example, 8 *mod* 3 (read "eight modulo three") = 2. Usually, the length or width of the neural network layout appears as the parameter m. Using the modulo function, any index no matter how large can be converted into what the position would be if the corresponding axis wrapped around to form a loop.

```
p = 11
q = 20
x =  3
y =  2
r =  4
for m1 = −r to r by 1
    for m2 = −r to r by 1
        xm1 = mod(x + m1 + p − 1, p) + 1
        ym2 = mod(y + m2 + q − 1, q) + 1
        if (m1 = r  or m2 = r) then
            the element (xm1, ym2) is in the r-th ring
        else
            the element is in the neighborhood ring
            determined by max(xm1, ym2)*
        end if
    next m2
next m1

*max(a, b) is the larger of the values a and b
```

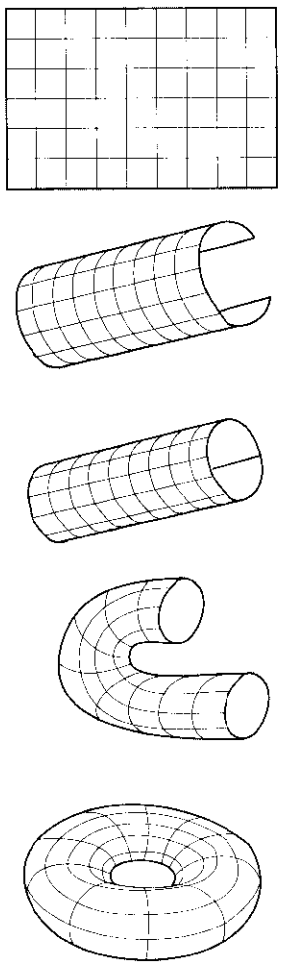

Figure 6-7: Wrapping a two-dimensional plane into a toroid.

In a Kohonen network, as always, we can speak of two layers: the input and the output layers. Only one layer is active. This active layer is usually arranged as a two-dimensional grid or "brick" of neurons (Figure 6-3 and Figure 6-4), but can also be arranged as a linear array (Figure 6-9). The toroidal wrapping in one-dimension is even simpler compared to the two-dimensional wrapping of neurons (Figure 6-7): the linear array becomes a circle.

Both mentioned layouts of neurons in a Kohonen layer – the one-dimensional (array) and the two-dimensional (plane) – are not the only possible solutions in this type of neural network application. Clearly, the layouts can be generalised to three- and higher-dimensional structures of neurons provided that the results from such complex networks can explain or solve the problem investigated in a more effective or plausible way.

All neurons in the active layer obtain the same multidimensional input. The most characteristic feature of the active layer in a Kohonen network is that it implements only a *local feedback*; that is, the output of each neuron is not connected to all other neurons in the plane (as in

the case of Hopfield network), but only to a small number that are topologically close to it. Such local feedback of possible corrections has the result that topologically close neurons behave similarly when similar signals are input.

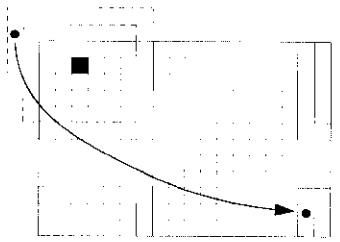

Figure 6-8: Elements of the fourth neighboring ring for the third neuron in the second row; ring segments found by "wrapping" (modulus function) are unhatched.

6.3 Competitive Learning

In *competitive learning*, only **one** neuron from those in the active layer is selected after input occurs; no matter how close the other neurons are to this best one, they are left out of that cycle. (This is also referred to as the "winner takes it all" method.)

The network selects the winner "*c*" (for "central") according to one of two criteria; *c* is the neuron having either:

The largest output in the entire network:

$$out_c \leftarrow \max(out_j) = \max\left(\sum_{i=1}^{m} w_{ji} x_{si}\right) \qquad (6.1)$$

$$j = 1, 2, \ldots, n$$

or the weight vector $\boldsymbol{W}_j (w_{j1}, w_{j2}, \ldots, w_{jm})$ most similar to the input signal $\boldsymbol{X}_s (x_{s1}, x_{s2}, \ldots, x_{sm})$:

$$out_c \leftarrow \min\left\{\sum_{i=1}^{m} (x_{si} - w_{ji})^2\right\} \qquad (6.2)$$

$$j = 1, 2, \ldots, n$$

(The index *j* refers to a particular neuron; *n* is the number of neurons; *m* is the number of weights per neuron; *s* identifies a particular input.) See also Section 7.4.

Index *j* that specifies the actual neuron in the Kohonen layer depends on the layout of the network. There is no problem if the neurons are arranged in a one-dimensional array consisting of *n* neurons. In such a case, the index *j* simply runs from 1 to *n*, while the closest neighbors to the selected neuron *c*, are the neurons with indices $j = c - 1$ and $j = c + 1$, the neurons of the second neighborhood have indices $j = c - 2$ and $j = c + 2$, and so on. In the case of a two-dimensional layout of the Kohonen network, index *j* has to be understood as describing the location of the particular neuron in a

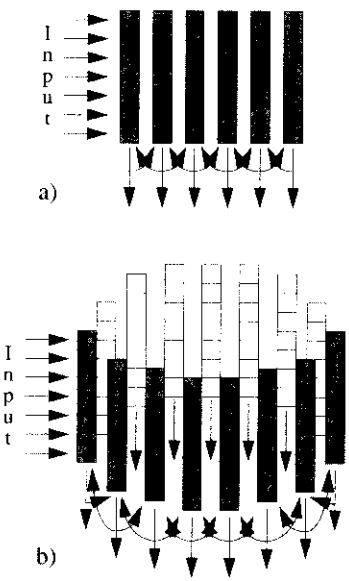

Figure 6-9: Kohonen network represented as a linear array of neurons (a). The local feedback connections are clearly seen: only the two neurons that are closest to each other receive the feedback when it occurs. The toroidal condition makes the array a circle (b).

two-dimensional plane. Usually, a two-dimensional location of neurons in the network is described by two indices: one describing the abscissa and the other one the ordinate axis of the position of the neuron. This means that the neuron j in a two-dimensional Kohonen layout having n neurons (ordered into an $n1 \times n1 = n$ network with $n1$ rows and $n1$ columns) can be found in the

$m1 = \lceil (j-1) / (n2) \rceil + 1$ column (x coordinate) runs from 1 to $n2$

and in the

$m2 = j - (m1n2)$ row (y coordinate) runs from 1 to $n1$

The mathematical expression $\lceil a \rceil$ means the largest integer not exceeding the value of a.

From now on, in all equations where index j occurs and whenever the two-dimensional Kohonen network architecture is applied (which is true in all examples) the index j will actually define the position of the j-th neuron in the $m1$-th column and $m2$-th row within the Kohonen network having $n = n1 \times n1$ neurons.

It must be added that rectangular neuron layouts with $n1 \neq n2$ are seldom used. In most cases the Kohonen layers are quadratic, i.e., $n1 = n2$. The quadratic layout minimizes the distortion of the 2D projection space that always occurs when projection from the multi-dimensional space of input vectors is applied.

After finding the neuron c, that best satisfies the selected criterion, its weights w_{ci} are corrected to make its response larger and/or closer to the desired one. This means that if a certain signal x_i coming to the weight w_{ci} has produced too large an output, the weight should be diminished, and vice versa.

The weights w_{ji} of neighboring neurons must be corrected as well. These corrections are usually scaled down, depending on the distance from c; for this reason, the scaling function is called a topology dependent function:

$$\mathrm{a}(\cdot) = \mathrm{a}(d_c - d_j) \tag{6.3}$$

where $d_c - d_j$ is the *topological* distance between the central neuron c and the current neuron j, while the extent of the stimulation depends on the function a(·). Figure 6-10 shows some of the forms that this function can take; besides decreasing with increasing d_c, it decreases with each iteration cycle of the Kohonen learning process (see box on the opposite page explaining the algorithm). In Kohonen learning one can distinguish two different cases. The first one occurs when the number of objects is so large that each object $\boldsymbol{X}_s$ enters the Kohonen

network only once and probably many more do not enter the learning procedure at all, while in the second case the number of objects for training the network is small, hence, it is necessary to input the entire set of objects again and again into the network, before it is properly trained.

In real world applications the second case occurs much more often. In order to describe the number of training cycles necessary for handling all objects by the network exactly once, the term called "one epoch" of training has been defined. Therefore, the duration of training is usually expressed in terms of epochs, meaning the number of times all objects have been processed by the network.

In view of this explanation and assuming the network is improving during the learning procedure Equation (6.3) is multiplied by another monotonically decreasing function $\eta(t)$:

$$f = \eta(t)\, a\,(d_c - d_j) \tag{6.4}$$

where t is the number of objects entered into the training process (if the number of objects is very large) or the number of epochs. The parameter t can easily be associated with time, since the time used for training is proportional to both - to the number of objects entering the network or to the number of epochs. $\eta(t)$ can be expressed as:

$$\eta(t) = (a_{max} - a_{min}) \frac{t_{max} - t}{t_{max} - 1} + a_{min} \tag{6.5}$$

where t_{max} is either the total number of objects that will be input into the network until learning is completed or the maximum number of epochs predefined at the beginning of training. The two constants a_{max} and a_{min} define the upper and lower limit between which the correction $\eta(t)$ is decreasing from the beginning to the end of training.

Due to its destimulation of the border of the selected neighborhood, the "Mexican hat" function (Figure 6-10c)) enhances the "contrast" on the borders of the developing regions in the output plane. This makes it very useful, but if too much contrast is applied "empty" spaces ("no man's land") can develop in the map on the borders between the categories.

The size of a neighborhood for the scaling function need not be permanent: it may well be changed during the learning period. Usually, it shrinks, which means that fewer neurons will have their

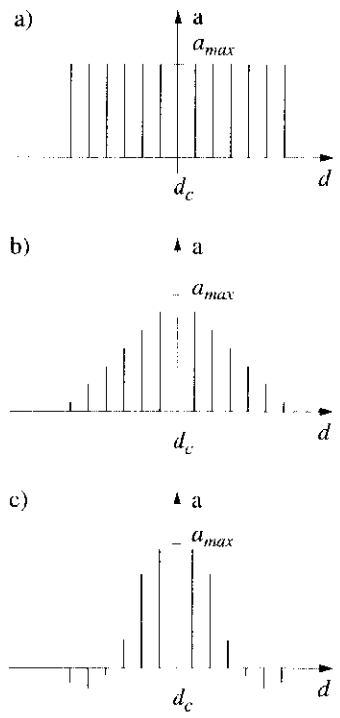

Figure 6-10: Typical functions for scaling corrections on neighbor-weights: constant (a), triangular (b), and Mexican hat (c).

In summary, the algorithm for one cycle of Kohonen learning is as follows:

- an m-dimensional object $\boldsymbol{X}_s$ enters the network;
- the responses of all neurons (each having m weights) are calculated;
- the position c is found for the neuron whose output is largest or most similar to the input;
- the weights of neuron c are corrected to improve its response for the same input $\boldsymbol{X}$ on the next cycle;
- the weights of all neurons in the (arbitrarily defined) neighborhood of the c-th neuron are corrected by an amount that decreases with increasing topological distance from c;
- the next m-variate object $\boldsymbol{X}_s$ is input and the process repeated.

weights corrected as the process goes on. Additionally, the maximum value of the scaling constant can be lowered.

The corrections of the weights w_{ji} of the j-th neuron lying within the region defined by the function f depend on the criterion used to select the central neuron c. We will describe in detail how to correct weights for one of these (Equation (6.2)):

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta\,(t)\,\mathrm{a}\,(d_c - d_j)\left(x_i - w_{ji}^{(old)}\right) \tag{6.6}$$

(Here, x_i is a component of the input $\boldsymbol{X}_s$; the central neuron is designated c, and the one being corrected is j; a particular weight of neuron j (and a particular input) is designated by i; t is (related to) which iteration cycle this is.)

Whether the difference $x_i - w_{ji}^{(old)}$ is positive or negative, i.e. whether x_i is greater or smaller than the weight $w_{ji}^{(old)}$, $w_{ji}^{(new)}$ will be closer to x_i than $w_{ji}^{(old)}$ was.

The correction function for the maximum signal criterion (Equation (6.1)) is evaluated similarly.

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta\,(t)\,\mathrm{a}\,(d_c - d_j)\left(1 - x_i w_{ji}^{(old)}\right) \tag{6.7}$$

After the corrections have been made using Equation (6.6) or (6.7), the weights should be normalized to a constant value, usually 1:

$$\sqrt{\sum_{i=1}^{m} w_{ji}^2} = 1 \tag{6.8}$$

Because of the specific architecture and learning algorithm of Kohonen networks, the outputs do not play as significant a **quantitative** role as in other networks. If the only significance of the output is to locate (topologically) the neuron with the largest output, then the actual magnitude of the output does not matter very much. Generally, our only concern is to keep the outputs within given limits in order to preserve the resemblance to actual biological neurons.

(In the case of criterion (6.2) and the associated correction (6.6), normalization does not improve the quality of the results.)

If the quantitative size of the output has little or no influence on the performance of the net, then normalizing the weights only disturbs the corrections. Besides, since the weights are corrected directly by comparison with the input signals (which presumably are normalized, or at least scaled to some reasonable values), the weights will be corrected to match them, and so will end up adjusted to normalized values anyway.

In any case, some kind of precaution must be taken to prevent the network output from "exploding". In our experience, providing an initial random distribution of weights (within the interval –0.1 to +0.1 or $-1/m$ to $+1/m$, where m is the number of weights) and scaling the input signals (and hence, the outputs) between –1 and +1 offers sufficient guarantee.

It must be clear from all of this that scaling or normalizing inputs must be performed very carefully.

> In most cases, it should be the nature of the problem rather than the method of solving it that dictates the criteria for normalizing (scaling and/or other transformations) of the input variables.

Though at first scaling the input may seem to be harmless at worst, this is not necessarily so. Improper scaling, especially across different variables, can change their internal relations and strongly influence the final results. Before applying normalization of any kind, check

thoroughly that the transformed variables will still adequately describe your problem.

Always try the simplest transformations first, for example, simply dividing all the variables by a value approximately equal to the total system output; something like this may do the job quite as well as more complex procedures.

> Only one thing is more important than a complete understanding of the method you are using for handling data – a thorough knowledge and understanding of your data!

6.4 Mapping from Three to Two Dimensions

To provide you with a feeling for mapping from a higher to a lower dimension (what it looks like and how it can be done), we will work out a very simple example in detail: topological mapping will be used to transfer the entire surface of a three-dimensional sphere onto a square, planar (15 x 15) matrix. The problem is schematically shown in Figure 6-11.

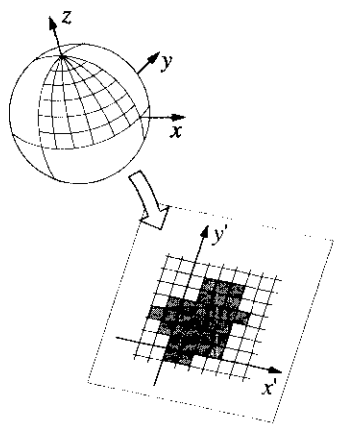

Figure 6-11: Topological mapping from a three- into a two-dimensional space.

A three-dimensional sphere of radius 1 is drawn around the coordinate system and divided into 8 spherical triangles (1 to 4 in the upper half and 5 to 8 in the lower half of the globe). 2000 points are generated randomly on the sphere's surface (approximately 250 on each triangular area) and labeled according to the eight possible combinations of coordinate signs (Table 6-1).

The Kohonen network used for this application is composed of three neurons in the input layer (one for each coordinate) and 225 neurons in the active output layer, arranged in a (15 x 15) matrix. Each of the three input neurons is connected to all 225 neurons on the Kohonen layer, which means that altogether 675 weights have to be trained for proper mapping. No bias weights are used.

Because the Kohonen network is square, the selection of square neighborhood rings (Figures 6-4 and 6-5) is the most natural choice. At the beginning, a neighborhood comprises seven rings of neighbors around the selected (central) neuron, which ensures that the neighborhoods are adjacent (no "no man's land")[1]. The number of

coordinates			label
x	y	z	
+	+	+	1
−	+	+	2
−	−	+	3
+	−	+	4
+	+	−	5
−	+	−	6
−	−	−	7
+	−	−	8

Table 6-1: Labeling eight triangular areas on the sphere.

corrections declines linearly from the inner to the outer rings of the neighborhood. The maximum correction at the central neuron, $a^{(0)}$ (Equation (6.4)) is 0.3, while in the 7th layer it is 0.5% of the central value (= 0.005 x 0.3 = 0.0015). No decrease of the central value $a^{(0)}$ is made during the training; in other words, $\eta(t) = constant$.

During training, the outer border of the neighborhood is reduced by one ring of neighbors every time a fifth (400) of the total points has entered the net. As before, the more distant neighbors are corrected proportionally less.

The selection of the central neuron is made by the criterion of Equation (6.2) (the neuron whose weights are most similar to the coordinates of the input points); hence, Equation (6.6) is used to correct the weights.

Remember, that the initial weights are random values between –0.1 and +0.1.

The result after entering 2000 points is shown in Figure 6-12. All the triangle labels on the sphere cluster into irregular areas of the net ("4" is shaded). These can be thought of as groups of neurons that "specialize" in different labels; in other words, the weights have evolved and differentiated so that, for example, all points from area number "3" produce their best matches in the area numbered "3" in the planar pattern in Figure 6-12.

You can get a feeling for how the weights change in the Kohonen network by referring to Figure 6-13. Let's say that a certain neuron has weights of (0.5, –0.7, 0.9); after the signal from one point (input:

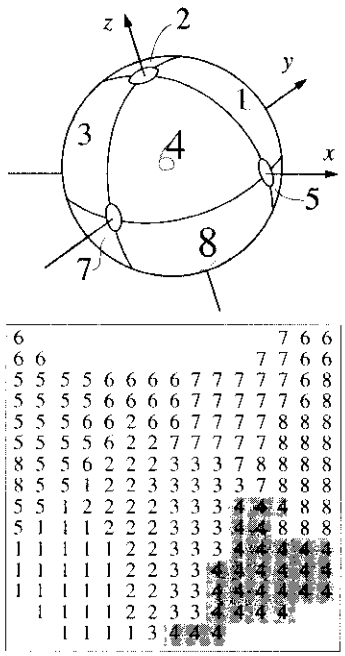

Figure 6-12: Mapping a sphere into two dimensions (see text).

1. Each ring is two neurons wider than the previous one. Hence, the width of n neighborhoods is $2n+1$, in this case 2 x 7 + 1 = 15, the (topological) width of the network.

(0.8, –0.5, 0.33)) is received, the weight corrections are calculated assuming that this is a central neuron and that $a^{(0)}$ (given by (6.1) with $d_c - d_j = 0$) equals 0.2. The changes of the weights are calculated from the differences (input minus old weights: (0.3, 0.2, –0.57)) and the scaling value $a^{(0)}$ to give: 0.06, 0.04, and –0.11, respectively; hence, the new weights are (0.56, –0.66, 0.79). It is obvious that all three weights are closer to the input values, and that the changes are proportional to the error. The largest correction is calculated for the third weight, where the error is the largest of the three.

The *Kohonen map* shown in Figure 6-14 indicates that the topology of the surface of the sphere is preserved in the planar map. Figure 6-12 clearly shows that area 4 is adjacent to 1, 3 and 8, and shares corners with 5, 7, 2; it has no contact with 6, which is on the opposite "side" of the sphere. These features also appear in Figure 6-14 (if you remember to wrap it so that opposite edges come together).

The circles in Figure 6-14 correspond to the points where the axes penetrate the globe (Figure 6-13); the circle at the bottom of Figure 6-14 is the mapping of the north pole.

We can present our results more clearly by actually extending the (15 x 15) network with redundant neurons (instead of "wrapping" the array 1, 2, 3, ..., 15, we write it out as 1, 2, 3, ..., 15, 1, 2, 3, ..., 15, etc.). That is, we will use the original (15 x 15) neural network (Figure 6-14) as a tile to cover an area nine times larger than before (Figure 6-15). (Close examination of Figure 6-15 shows that the pattern repeats every 15 units.)

The topological relations (links) among the numbered areas can be expressed as a connection table or as a connectivity matrix. Both forms are shown below in Table 6-2. (The row or column for area 4 shows that it is adjacent to 1, 3 and 8.)

By comparing the topologies found on the sphere and on the (15 x 15) neural network, we can see that they are identical, whichever presentation form we use.

In addition to the longer borders between the numbered areas, shorter borders (as short as one or two neurons) can be observed as well. A closer look at the regions where such short borders are found shows that at these locations **four** areas converge in all cases. Such areas correspond to the places where the coordinate axes enter or leave the sphere (white circles, Figures 6-11 and 6-14). As expected, there are six such areas in the obtained map.

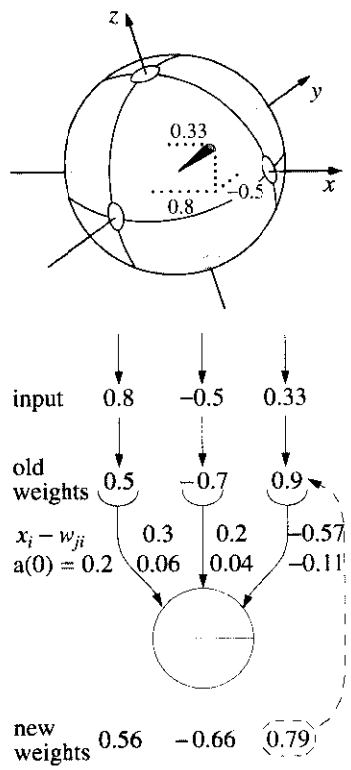

Figure 6-13: Correction of weights in the Kohonen network (cf. Equation (6.6)).

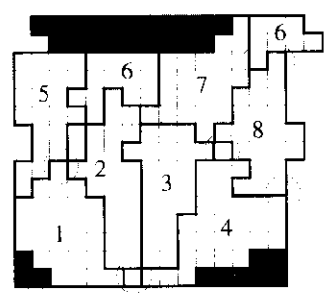

Figure 6-14: Topology of the Kohonen map of the sphere, indicating the border lines and common points of the spherical triangles.

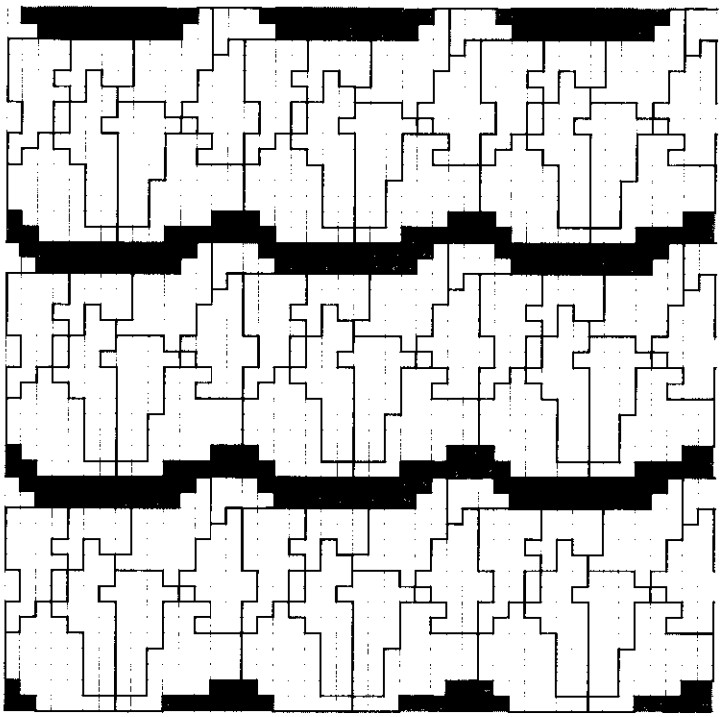

Figure 6-15: The plane can be tiled by repetition of patterns obtained on the toroid. In a periodic pattern the most important points can be seen more easily.

area	neighboring area no.			area	1	2	3	4	5	6	7	8
1	2	4	5	1	0	1	0	1	1	0	0	0
2	1	3	6	2	1	0	1	0	0	1	0	0
3	2	4	7	3	0	1	0	1	0	0	1	0
4	1	3	8	4	1	0	1	0	0	0	0	1
5	1	6	8	5	1	0	0	0	0	1	0	1
6	2	5	7	6	0	1	0	0	1	0	1	0
7	3	6	8	7	0	0	1	0	0	1	0	1
8	4	5	7	8	0	0	0	1	1	0	1	0

Table 6-2: The topological relations among the numbered areas shown in Figure 6-12 can be recorded as a connection table (left) or as a connectivity matrix (right).

Another notable feature of Figure 6-12 is the "empty region", which separates the vertex where areas 1, 2, 3 and 4 come together (the "north pole" of Figure 6-12) from the areas numbered 5, 6, 7 and

8, which are in the "southern hemisphere". This same region can be thought of as separating the south pole from the triangles in the northern hemisphere; that is, there is only one such "empty region", a fact which is evident from the extended (tiled) representation of Figure 6-15 (see Reference 6-9).

The actual results of the Kohonen topology preserving mapping procedure always depend on the initial choice of weights (chosen randomly in the region from –0.1 to 0.1 for this experiment), the selected neighborhood, the correction function $a(d_c - d_j)$, and the initial value of correction $a^{(0)}$. However, the topology of the obtained map (as expressed above) will generally be preserved, i.e., the procedure is very robust or stable regarding small changes in the procedure.

6.5 Another Example

The mapping of two- and three-dimensional objects into a two-dimensional plane of neurons is very instructive, because you can visualize the results. However, in physics, chemistry, technology, sociology, economics, and other disciplines, we have data sets composed of more than three variables.

For example, wine producers analyze each wine for at least ten components, and they have thousands of sets of data. Engineers monitor a given technological process for even larger numbers of parameters (temperature, pressure, flow-rate, etc.).

All these sets of data can be regarded as m-tuplets or m-element vectors, each component in a given set being the value of one variable describing a certain wine, say, or a technological process. Such data can be analyzed in many ways and by many methods. If large datasets must be processed, Kohonen mapping can turn out to be quite effective for at least the prescreening of data. For more information on mapping, see Section 9.4.

In Part IV of this book, where some examples are worked out in detail, a set of a few hundred chemical bonds is described using seven different variables: electronic, energy, etc. (Chapter 11); that is, each chemical bond in this collection is represented in a seven-dimensional space.

There are quite a number of things that chemists would like to know about these chemical bonds, such as whether or under what conditions the bond is breaking.

This is a typical problem for complex statistical analysis, cluster analysis, or any other standard data processing method. Unfortunately, most of these standard methods are too inefficient for handling tens of thousands of datasets. Kohonen's competitive learning method offers a very simple and efficient (though maybe not well understood) method that can at least shed some light on the problem.

For the next example, we have selected 200 chemical bonds (seven-element vectors). For 94 of these bonds we know that they either break very easily (58) under specified conditions or with great difficulty or not at all (36), and the rest are bonds for which we do not have the relevant information. The questions we would like to answer are:

1) Can the easily breakable bonds and those that are hard to break be separated by this method?
2) If yes, how many variables are really needed to achieve this?
3) What can be said about the bonds that will overlap or "excite" the same neuron even though they have different representations?
4) How stable or robust is the method?

There are of course many more questions that can be raised, but not all of them are as important as those above. The Kohonen map of this dataset is shown in Figure 6-16.

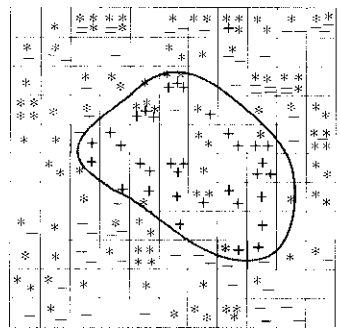

Figure 6-16: Kohonen network of (11 x 11) neurons that clusters breakable and non-breakable bonds on the basis of seven-dimensional data vectors.

Because the map shown in Figure 6-16 has only 121 neurons, it is clear that more than one of the 200 bonds must excite the same neuron. At the same time, some neurons that are not excited at all will co-exist with them in the network. The neurons excited by bonds that are "easy" or "difficult" (to break) are marked as "+" and "–", respectively, while the neurons excited by the bonds for which we did not have data are marked with asterisks (*).

As can be seen, only two neurons are excited by all three types of bonds; a detailed analysis has shown that the parameters of these bonds are actually very similar. There are of course many more interesting details in this study, which may be found in Chapter 11.

6.6 Remarks

The Kohonen network has a very serious computational drawback that affects the performance of large scale applications running on parallel (but not serial) computers. In order to find out which neuron

(and neighborhood) is to be stimulated, the program must check **all** n neurons; this is a serious restriction when large nets are to be trained. Even on a parallel computer, this involves $n/2$ parallel comparisons, which requires at least $\log_2 n$ steps; the relative advantage of (expensive) parallel computers is thus compromised. On ordinary machines, all calculations have to be done sequentially anyway, and keeping track of the largest output does not affect the overall performance very much.

The Kohonen network or a Kohonen layer can be built into a more complex network as one of its constituent layers (Reference 6-4) or implemented in combination with some other techniques (Reference 6-5). As we will see in the next chapter, Kohonen competitive learning can be combined with the counter-propagation corrections to form multilevel networks (Reference 6-7).

6.7 Essentials

- the main goal of Kohonen neural networks is to map objects from m-dimensional into n-dimensional space
- this mapping preserves the essential **topological** features of the data
- the primary neuron for weight modification is chosen by competition ...
- ... the algorithm modifies the weight of the neuron with the most intense output, or whose weights are most similar to the input signal ...
- ... and smooths the map by also making modulated changes to neurons in a defined "neighborhood" of that one
- the Kohonen learning procedure is unsupervised learning
- the topology of the planar network can vary considerably from application to application and there are different types of neighborhood relations in the network

• **finding the best neuron**

$$out_c \leftarrow \max(out_j) = \max\left(\sum_{i=1}^{m} w_{ji} x_{si}\right) \qquad (6.1)$$

$$j = 1, 2, ..., n$$

$$out_c \leftarrow \min\left\{\sum_{i=1}^{m} (x_{si} - w_{ji})^2\right\} \qquad (6.2)$$

$$j = 1, 2, ..., n$$

• **corrections of weights**

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta(t)\, a(d_c - d_j)\left(x_i - w_{ji}^{(old)}\right) \qquad (6.6)$$

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta(t)\, a(d_c - d_j)\left(1 - x_i w_{ji}^{(old)}\right) \qquad (6.7)$$

6.8 References and Suggested Readings

6-1. T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", *Biol. Cybern.* **43** (1982) 59 – 69.

6-2. T. Kohonen, *Self-Organization and Associative Memory*, Third Edition, Springer Verlag, Berlin, FRG, 1989.

6-3. R. Hecht-Nielsen, "Counterpropagation Networks", *Appl. Optics* **26** (1987) 4979 – 4984; D. G. Stork, "Counter-propagation Networks: Adaptive Hierarchical Networks for Near Optimal Mappings", *Synapse Connection* **1** (1988) 9 – 17.

6-4. P. D. Wasserman and T. Schwartz, "Neural Networks, Part 2: What are They and Why is Everybody so Interested in Them Now?", *IEEE Expert*, Spring 1988, 10 – 15.

6-5. J. Zupan, *Algorithms for Chemists*, Paragraph 11.1.6: "Neural Network Algorithm A30 for Kohonen Learning", John Wiley, Chichester, UK, 1989, pp. 257 – 261.

6-6. H. Ritter, T. Martinetz and K. Schulten, *Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*, Addison-Wesley, Bonn, FRG, 1990.

6-7. J. Dayhoff, *Neural Network Architectures, An Introduction*, Van Nostrand Reinhold, New York, USA, 1990.

6-8. K. L. Peterson, "Classification of Cm Energy Levels Using Counterpropagation Neural Networks", *Phys. Rev. A* **41** (1990) 2457 – 2461.

6-9. X. Li, J. Gasteiger and J. Zupan, "On the Topology Distortion in Self-Organizing Feature Maps", *Biol. Cybern.* **70** (1993) 189 – 198.

6-10. J. Gasteiger and J. Zupan, "Neuronale Netze in der Chemie", *Angew. Chem.* **105** (1993) 510 – 558; J. Gasteiger and J. Zupan, "Neural Networks in Chemistry", *Angew. Chem. Int. Ed. Engl.* **32** (1993) 503 – 527.

6-11. J. Zupan, "Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them", *Acta Chim. Slov.* **41** (1994) 327 – 352.

6-12. J. Lozano, M. Novic, F. X. Rius and J. Zupan, "Modelling Metabolic Energy by Neural Networks", *Chemom. Intell. Lab. Syst.* **28** (1995) 61 – 72.

6-13. J. Novic and J. Zupan, "Investigation of IR Spectra-Structure Correlation Using Kohonen and Counterpropagation Neural Network", *J. Chem. Inf. Comp. Sci.* **35** (1995) 354 – 466.

6-14. N. Majcen, K. Rajer-Kanduc, M. Novic and J. Zupan, "Modeling of Property Prediction from Multicomponent Analytical Data Using Different Neural Networks", *Anal. Chem.* **67** (1995) 2154 – 2161.

6-15. J. A. Remola, J. Lozano, I. Ruisanchez, M. S. Larrechi, F. X. Rius and J. Zupan, "New Chemometric Tools to Study the Origin of Amphorae Produced in the Roman Empire", *Tr. Anal. Chem.* **15** (1996) 137 – 151.

6-16. I. Ruisanchez, P. Potokar, J. Zupan and V. Smolej, "Classification of Energy Dispersion X-Ray Spectra of Mineralogical Samples by Artificial Neural Networks", *J. Chem. Inf. Comput. Sci.* **36** (1996) 214 – 220.

6-17. J. Zupan, M. Novic and I. Ruisanchez, "Tutorial: Kohonen and Counter-propagation Artificial Neural Networks in Analytical Chemistry", *Chemom. Intell. Lab. Syst.* **38** (1997) 1 – 23.

6-18. I. Ruisanchez, J. Lozano, M.S. Larrechi, F.X. Rius and J. Zupan, "On-line Automated Analytical Signal Diagnosis in Sequential Injection Analysis Systems Using Artificial Neural Networks", *Anal. Chim. Acta* **348** (1997) 113 – 128.