Part II
One-Layer Networks

4 Hopfield Network

learning objectives

- Hopfield networks, though extremely simple, are capable of auto-association
- how neural nets are designed (architecture)
- simple data representations: binary and bipolar
- how a simple network is trained (stabilized) by iteration
- how a network trained to recognize images can recognize them even when they are corrupted
- the number of neurons needed for a given dataset

4.1 General

In 1982, the American physicist J. J. Hopfield brought neural network research back from the anathema to which it had been pushed in the seventies and early eighties. His paper (Reference 4-1), as a milestone on the way into the new era of neural network research, introduced nonlinear transfer functions for the evaluation of the final output from neurons.

The Hopfield neural network performs one of the most interesting tasks the brain is able to do: auto-association (Figure 1-3), by means of which a stored image (or any other information representable by a multivariable vector or matrix) is regenerated from partial or corrupted data. In other words, you perform auto-association whenever you recognize a friend after seeing, say, only his/her eyes.

Such procedures are clearly desirable not only in science but in areas as diverse as art, law, and economics. In art and science, it offers the possibility of reconstructing original images from blurred copies (NASA uses "image enhancement" techniques, for example, to increase the number of pixels in astronomical photographs); it could serve a similar purpose, say, in deciphering the Dead Sea Scrolls.

A "better" auto-association procedure is one which can produce a given degree of reconstruction from a "worse" original.

Besides auto-association, the Hopfield net can solve optimization problems (such as the famous "traveling salesman problem" – see Reference 4-3). However, we will not go into that here.

4.2 Architecture

The Hopfield neural net is a one-layer neural network. It consists of as many neurons as there are input signals, each neuron having, of course, the same number of weights as there are input signals. This means that the Hopfield network for m-variate signals (group of m individual signals) is a quadratic (square) (m x m)-variate matrix of weights.

Figure 4-1 shows us a design for a Hopfield net that can learn the auto-association of (4 x 4)-pixel images. The image is input as a 16-variable array, and so the Hopfield net has 16 neurons, each having 16 weights.

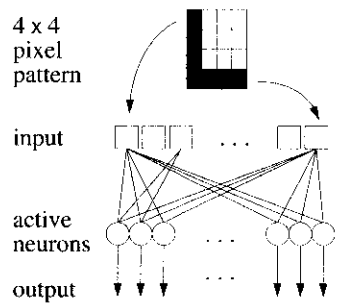

Figure 4-1: Hopfield net for learning associations of (4 x 4)-pixel images. The network has one layer with 16 neurons.

The original Hopfield neural network treats the binary signals as bipolar ones (having values of +1 or −1 only). It actually does not matter which notation is used for storing the images; however, the image should be represented in bipolar notation. In order to avoid confusion, and because computers store black and white pixels as bits (binary digits, 0 or 1), a simple transformation for each binary signal x_i can be introduced before entering the Hopfield network:

$$\boldsymbol{X}^{bipolar} = 2\boldsymbol{X}^{binary} - 1 \tag{4.1}$$

$$(x_1, x_2, ..., x_m)^{bipolar} = (2x_1 - 1, 2x_2 - 1, ..., 2x_m - 1)^{binary}$$

The transformation $2x_j - 1$ in Equation (4.1) is a programmer's trick to enlarge the interval of x_j values by a factor of 2, and to shift the entire interval one unit lower on the axis: from 0 – 1 to (−1) – (+1).

4.3 Transfer Function

Because the transfer function in the Hopfield net is a bipolar version of the hard-limiter, hl, (Equation (2.24)) the input signals must be in bipolar notation:

$$\mathrm{hl}(Net_j) = \mathrm{sign}(Net_j) = \mathrm{sign}\left(\sum_{i=1}^{m} w_{ji} x_i^{bipolar} \right) \tag{4.2}$$

where the function sign(u) means the algebraic sign of the argument u. Each term in the summation Net_j, the net input of the j-th neuron, is the product of a weight w_{ji} of the j-th neuron and the input signal $x_i^{bipolar}$ fed into it. The superscript bipolar is to remind us that the input signal must lie between +1 and −1. From now on, it will be omitted.

To use the binary representation, Equation (4.2) should be slightly modified:

$$\mathrm{hl}(Net_j) = \mathrm{sign}\left[\sum_{i=1}^{m} w_{ji} \left(2x_i^{binary} - 1 \right) \right] \tag{4.3}$$

4.4 Weight Matrix

How is the association learned in a Hopfield network? Very simply. Assume that each image is stored in the computer as an m-variable vector $\boldsymbol{X}$ of bipolar values (pixels):

$$\boldsymbol{X} = (x_1, x_2, ..., x_i, ..., x_m) \tag{4.4}$$

obtained by substituting white and black pixels by −1 and +1, respectively, and by taking the 4-pixel lines in sequence from the top row to the bottom row of the (4 x 4) image.

Now, first, we have to calculate the weights so that they will be appropriate for the image patterns that are to be learned by the Hopfield network. Say there are p images; the weights w_{ji} are calculated as follows:

$$\begin{aligned} w_{ji} &= \sum_{s=1}^{p} x_{sj} x_{si} \quad \text{if} \quad j \neq i \\ &\text{and} \\ w_{ji} &= 0 \qquad\qquad \text{if} \quad j = i \end{aligned} \tag{4.5}$$

Analyzing the value of w_{ji}, we see that it increases by 1 if the j-th and i-th pixel in a given pattern s are equal (i.e. both are white or both

black), and it decreases by 1 if they are different. The more identical pairs of pixels j and i exist in the p patterns, the larger is the weight; the maximum absolute value that each w_{ji} can reach is p, the number of patterns in the set.

Hence, a **new** pattern is learned (that is, one cycle of weight modification takes place) simply by adding or subtracting 1 from each w_{ji}, depending on whether the j-th and the i-th pixel are equal or different. (It doesn't get much easier than that!)

Let us calculate a simple example. Figure 4-2 gives us four 16-variable patterns, actually (4 x 4)-pixel images, showing four arbitrary patterns. We would like these four patterns to be learned by the Hopfield net (Figure 4-3) by association.

The reader is encouraged to go through this example, or similar ones, by himself. To this purpose, the program HOPF is provided on the website of this book

http://www2.ccc.uni-erlangen.de/ANN-book/

On this site, also the datafiles for the patterns used in the example discussed here can be found. See Appendix for further details.

Equation (4.5) "stores" all of the patterns in the weight matrix $\boldsymbol{W}$, in the sense that **each** element of $\boldsymbol{W}$ is actually influenced by **all** patterns $\boldsymbol{X}_s$ $(x_{s1}, x_{s2}, \ldots, x_{si}, \ldots, x_{sm})$. (Since the weight matrix $\boldsymbol{W}$ "contains" all patterns, then presumably the patterns can be "retrieved" from it!)

A new pattern $\boldsymbol{X}^{(new)}$ can be added to the matrix by simply adding the product of the corresponding components of $\boldsymbol{X}^{(new)}$ to each element of the weight matrix $\boldsymbol{W}$:

$$\text{Updated } \boldsymbol{W} \leftarrow w_{ji}^{(new)} = w_{ji}^{(old)} + x_j^{(new)} x_i^{(new)}$$

It is evident from Equation (4.5) that the matrix of weights is quadratic (because both i and j run from 1 to m), and symmetric over the main diagonal, which is set to zero. You should carry out the calculation of the weight matrix (or at least try to evaluate a few weights), and then compare your results with the complete weight matrix shown below. See also Figure 4-3; note that the first index, j, of w_{ji} is associated with the output and the second, i, refers to the input.

As an example, $w_{1,4}$ is calculated to be

$$w_{1,4} = (-1)(+1) + (-1)(-1) + (+1)(+1) + (+1)(-1) = 0$$

pattern no. 1:
$\boldsymbol{X}_1$=(0,0,1,1,0,0,1,1,1,1,0,0,1,1,0,0)

pattern no. 2:
$\boldsymbol{X}_2$=(0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0)

pattern no. 3:
$\boldsymbol{X}_3$=(1,1,1,1,0,0,0,1,0,0,0,1,1,1,1,1)

pattern no. 4:
$\boldsymbol{X}_4$=(1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,1)

Figure 4-2: (4 x 4)-pixel images to be memorized by a Hopfield net can be represented as 16-variable vectors, given in binary notation. (Before entering the Hopfield net they are converted into bipolar representation using the transformation of (4.1).)

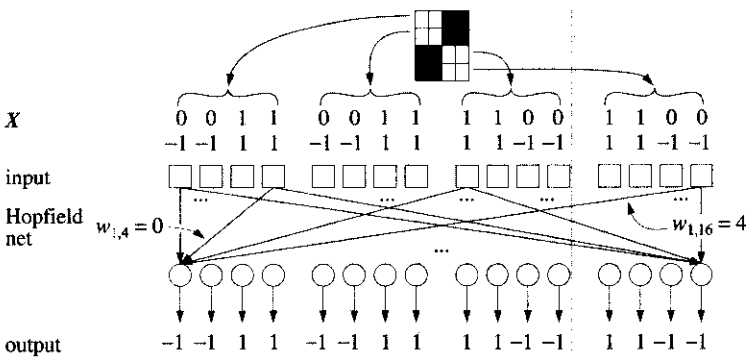

Figure 4-3: The Hopfield net shown in conventional notation. Each neuron has 16 weights ("synapses") monitoring the 16 input signals.

What are some of the things $\boldsymbol{W}$ can tell us about the learned images? For one thing, the value −4 for $w_{10,1}$ indicates that in all images, the first and tenth pixels are different from each other. Similarly, the value of $w_{1,16} = 4$ indicates that all four (1,16)-pairs of pixels have the same color. Figure 4-2 shows that the first and last pixel in patterns $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are white, while they are both black in $\boldsymbol{X}_3$ and $\boldsymbol{X}_4$.

The (16 x 16) matrix $\boldsymbol{W}$, below, represents 16 neurons, each having 16 weights. Because it is symmetric across the main diagonal, it does not matter whether we imagine the organization of the neurons to be by column or by row; that is, each row or column of the matrix is a set of 16 weights which belong to one neuron.

Once the patterns to be stored in a Hopfield network are known, "learning" consists simply of calculating the weight matrix $\boldsymbol{W}$ according to Equation (4.5). A very interesting thing about the particular $\boldsymbol{W}$ shown above is that each of the four patterns $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$ or $\boldsymbol{X}_4$ (Figure 4-2) from which $\boldsymbol{W}$ was generated, are recognized exactly when put back into the net; that is, each one produces a 16-variable signal identical to the input. This, of course, is what we hoped for, but it is surprising that it was achieved all at once. With larger Hopfield networks (that can learn hundreds of different images), exact recognition of the "training" patterns comes only after a number of iterations. We will now examine the process of iteration during learning.

$$W = \begin{array}{c} \begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{array} \\ \left[\begin{array}{cccccccccccccccc} 0 & 2 & 0 & 0 & 2 & 0 & -2 & 0 & -2 & -4 & -2 & 0 & 0 & 2 & 4 & 4 \\ 2 & 0 & 2 & 2 & 0 & 2 & 0 & 2 & -4 & -2 & 0 & 2 & -2 & 0 & 2 & 2 \\ 0 & 2 & 0 & 4 & -2 & 0 & 2 & 4 & -2 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 4 & 0 & -2 & 0 & 2 & 4 & -2 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\ 2 & 0 & -2 & -2 & 0 & 2 & 0 & -2 & 0 & -2 & 0 & -2 & -2 & 0 & 2 & 2 \\ 0 & 2 & 0 & 0 & 2 & 0 & 2 & 0 & -2 & 0 & 2 & 0 & -4 & -2 & 0 & 0 \\ -2 & 0 & 2 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & -2 & -2 & 0 & -2 & -2 \\ 0 & 2 & 4 & 4 & -2 & 0 & 2 & 0 & -2 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\ -2 & -4 & -2 & -2 & 0 & -2 & 0 & -2 & 0 & 2 & 0 & -2 & 2 & 0 & -2 & -2 \\ -4 & -2 & 0 & 0 & -2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 & -2 & -4 & -4 \\ -2 & 0 & -2 & -2 & 0 & 2 & 0 & -2 & 0 & 2 & 0 & 2 & -2 & -4 & -2 & -2 \\ 0 & 2 & 0 & 0 & -2 & 0 & -2 & 0 & -2 & 0 & 2 & 0 & 0 & -2 & 0 & 0 \\ 0 & -2 & 0 & 0 & -2 & -4 & -2 & 0 & 2 & 0 & -2 & 0 & 0 & 2 & 0 & 0 \\ 2 & 0 & 2 & 2 & 0 & -2 & 0 & 2 & 0 & -2 & -4 & -2 & 2 & 0 & 2 & 2 \\ 4 & 2 & 0 & 0 & 2 & 0 & -2 & 0 & -2 & -4 & -2 & 0 & 0 & 2 & 0 & 4 \\ 4 & 2 & 0 & 0 & 2 & 0 & -2 & 0 & -2 & -4 & -2 & 0 & 0 & 2 & 4 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array}$$

4.5 Iteration

When, during training, the network produces an output that is not equal to the input (which is what usually happens), this output is input again. The input in the next iteration, $X^{(t+1)}$, is taken from the output of the present one, $Out^{(t)}$:

$$X^{(t+1)} = Out^{(t)}$$

Generally, a pattern more similar to the input will emerge. This procedure is repeated (without changing the weights) until identical outputs are produced in two consecutive steps (Figure 4-4).

$$Out^{(t)} = Out^{(t-1)} = X^{(t)}$$

NOTE: While the outputs are converging towards **something**, there is no guarantee that this "something" is the signal which was input to the net at the beginning (the original). As a matter of fact, the iterative procedure can have many different outcomes:

1) the final output is identical to the **initial** input;

2) the final output is identical to some **other** pattern stored in the net;

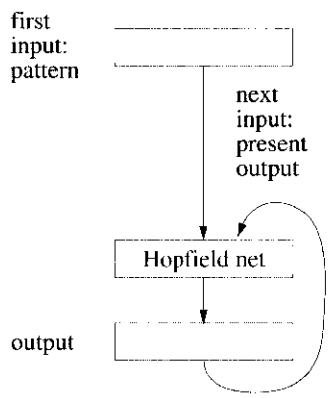

Figure 4-4: Iteration. Cycling of a multivariate signal from the net's output to the input, then through the net again until an output identical to the input is obtained.

3) the final output is not equal to any of the patterns used in the evaluation of weights;

4) the final output is as described in 1), 2) or 3), but with the pixels color-inverted (black for white, white for black) – that is, a **negative** image. This can happen, if the input signal has **more than** 50% wrong pixels from one of the originals.

Or, if convergence is not achieved:

5) the final output may **oscillate** between two or more patterns (possibly inverted) which are not equal to any of the original patterns.

If the Hopfield network produces the **original** input signal, we refer to it as a *stable* or *stabilized* network.

> Note that these cycles do not change the weights!

In order to appreciate how difficult it is to get a "balanced" set of inputs to form a stabilized Hopfield net, you should program your own small nets and test them with different images. The main trick is to select images with approximately the same number of black and white pixels, as evenly distributed over the entire area as possible.

Clearly, only outcome 1) is of much practical use. But, you may ask, what good is a device for reproducing the same image that was input to it? The reproduction of all originals is only an initial test, determining whether the net is stabilized.

The idea of the Hopfield net is that the stabilizing effect of weights will force the net to produce the original image even from a corrupt, incomplete, dim, or blurred image. How far such an association of an incomplete input with the original one will go is hard to predict; in some cases, even an image with more than 30% of the pixels corrupted or missing will produce the original.

The necessary condition for a successful association is that the network be stable from the beginning.

Let us explore this feature of the Hopfield net using our recent example. Instead of inputting the original four patterns, let's corrupt the originals by 2, 5 and 13 randomly selected pixels and put it into the Hopfield network (in an image with 16 pixels, these represent errors of 13%, 31% and 81%: the proportions of the 16 pixels whose colors are opposite of what they should be). The numbers of iterations required

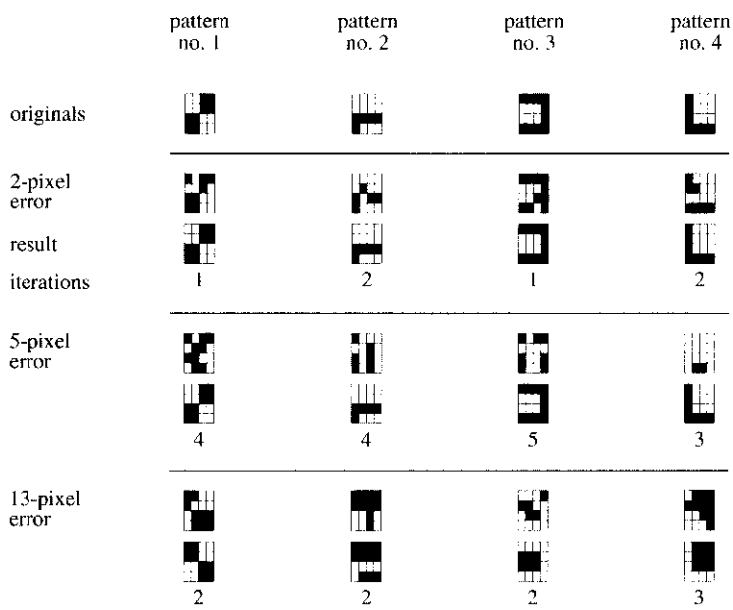

Figure 4-5: Outputs obtained by the Hopfield network of Figure 4-3 from inputs with 2, 5, and 13 errors, compared to the original patterns (top row). Numbers of iterations needed for each stable output are given below each output.

to get a stable output in each of these cases are given in Figure 4-5. However, two images with the same percentage of corrupted pixels may not require the same number of iterations, nor will they necessarily converge towards the same pattern. The larger the degree of corruption, the more unpredictable the outcome.

Figure 4-6 shows unpredictable output for the middle case (31% errors). You might think the stability of the output would get worse as the proportion of errors gets larger, but this is not the case.

If more than 50% of the pixels are corrupted, the retrieved image is likely to become the negative of the original (a 95% corrupt picture is actually the negative of the original corrupted by 5%). Therefore, it is not hard to understand the bottom row of Figure 4-5, in which an 81% (13 pixels) corrupted image produces a negative of the original.

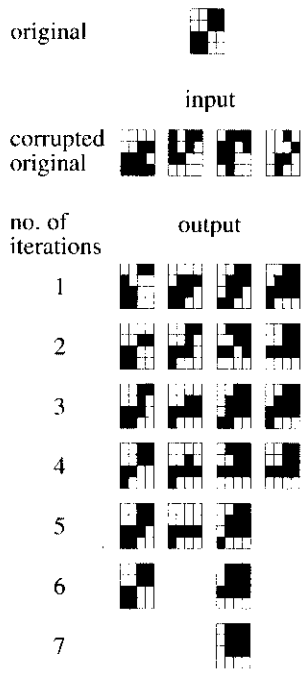

Figure 4-6: Some possible outputs (last pattern in each column) if the original pattern has 5 corrupted pixels. The outcome is unpredictable: it may be the original (first column), some other pattern (second column), the negative of an actual pattern (third column), or it may oscillate between two patterns (fourth column). Note that the number of iterations required differs from case to case.

4.6 Capacity of the Hopfield Network

The Hopfield net may seem very attractive for some applications, but there are severe restrictions on the number of images (patterns) that such a network is able to learn. In general, the net should contain about seven neurons for each pattern to be learned (and recognized):

$$N^{neurons} = 7N^{images} \tag{4.6}$$

(In our preceding example, the number of neurons was only four times the number of images. This will be explained at the end of this section; for details, see Reference 4-12.)

Equation (4.6) has two consequences. First, it shows that the minimum resolution of the images depends on how many there are. In terms of the previous example, Equation (4.6) requires that each of one hundred 2-dimensional images, which would require 700 neurons, be represented on a grid of at least 700 pixels, or, for a square image, a grid of about (27 x 27) pixels.

Second, Equation (4.6) determines the size of the weight matrix. Since a Hopfield network generates a quadratic matrix, learning (recognizing, associating) 100 images requires a matrix containing $700^2 = 490{,}000$ weights.

This brings up an obvious problem: since each number in a computer is represented by four bytes (floating points) or two bytes (short integers), a Hopfield net for recalling 100 images (on a (27 x 27) grid) will require one or two Mbyte of memory space for the weight matrix. However, there is an even worse aspect of this problem. Recalling any image requires as many as 2 million multiplications and 2 million additions (performed sequentially, on most computers) for **each** iteration cycle! And the space and time requirements grow quadratically as the size of the problem grows!

Equation (4.6) is valid for a random sample only; as a consequence, the number of patterns to be stored in a Hopfield net need not depend only on the number of neurons. By a careful arrangement or selection of patterns, we can increase the capacity of the Hopfield network above 0.14 images/neuron, up to maximum of about 0.25. Our example, storing four images on a grid of 16 pixels, is about the maximum we can achieve on such a grid.

4.7 Essentials

- the number of input data is equal to the number of output data
- there is no refinement of weights in the Hopfield network; the weights are calculated from the patterns
- the stored patterns are retrieved from the network by a "circular" flow of signals; the output becomes the next input
- the network is able to "retrieve" the uncorrupted patterns even if the inputs are corrupted
- the number of patterns that can be stored simultaneously in the Hopfield network is relatively small

• **binary to bipolar vector conversion**

$$(x_1, x_2, ..., x_m)^{bipolar} = (2x_1 - 1, 2x_2 - 1, ..., 2x_m - 1)^{binary} \tag{4.1}$$

• **Hopfield network:**

weights

$$w_{ji} = \sum_{s=1}^{p} x_{sj} x_{si} \quad \text{if} \quad j \neq i$$

and

$$w_{ji} = 0 \quad \text{if} \quad j = i \tag{4.5}$$

output

$$\text{hl}(Net_j) = \text{sign}(Net_j) = \text{sign}\left(\sum_{i=1}^{m} w_{ji} x_i^{bipolar} \right) \tag{4.2}$$

• **requirement or capacity**

$$N^{neurons} = 7N^{images} \tag{4.6}$$

4.8 References and Suggested Readings

4-1. J. J. Hopfield, "Neural Networks and Physical Systems With Emergent Collective Computational Abilities", *Proc. Natl. Acad. Sci. USA* **79** (1982) 2554 – 2558.

4-2. J. J. Hopfield, "Neurons With Graded Response Have Collective Computational Abilities", *Proc. Natl. Acad. Sci. USA* **81** (1984) 3088 – 3092.

4-3. J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems", *Biol. Cybern.* **52** (1985) 141 – 152.

4-4. J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model", *Science* **233** (1986) 625 – 633.

4-5. J. J. Hopfield and D. W. Tank, "Simple Neural Optimization Networks", *IEEE Trans CS*, **CAS-33,** 533 – 541.

4-6. R. P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987, 4 – 22.

4-7. R. J. McEliece, E. C. Posner, E. R. Rodemich and S. S. Venkatesh, "The Capacity of Hopfield Associative Memory", *IEEE TIT*, **IT-33** (1987) 461 – 482.

4-8. J. J. Hopfield, D. I. Feinstein and R. G. Palmer, "Unlearning has a Stabilizing Effect in Collective Memories", *Nature* **304** (1988) 158 – 159.

4-9. J. A. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, USA, 1988.

4-10. M. Tusar and J. Zupan, "Neural Networks", in *Software Development in Chemistry* ***4***, Ed.: J. Gasteiger, Springer Verlag, Berlin, FRG, 1990, pp. 363 – 376.

4-11. H. Ritter, T. Martinetz and K. Schulten, *Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*, Addison-Wesley, Bonn, FRG, 1990.

4-12. D. J. Amit, H. Gutfreund and M. Sompolinski, "Storing Infinite Number of Patterns in a Spin-glass Model of Neural Networks", *Phys. Rev. Lett.* **55** (1985) 1530 – 1533.