3. NEURAL NETWORK MODELS

3.1 Early Approaches

The first neural network models go back to the 1940s. Around this time, two mathematicians, McCulloch and Pitts (1943) suggested the description of a neuron as a *logical threshold element* with two possible states. Such a threshold element has L input channels (*afferent axons*) and one output channel (*efferent axon*). An input channel is either active (input 1) or silent (input 0). The activity states of all input channels thus encode the input information as a *binary sequence* of L bits. The state of the threshold element is then given by linear summation of all afferent input signals x_i and comparison of the sum with a threshold value s. If the sum exceeds the threshold value, then the neuron is *excited*; otherwise, it is in the quiescent state. The excited and quiet state should correspond to the firing or not firing of an action potential of biological neurons and are represented in the model by the binary values 1 and 0 for the activity of the output channel. Excitatory and inhibitory input signals are modulated by "synaptic strengths" $w_i = \pm 1$. The output signal y of a neuron is thus given by

$$y = \theta\left(\sum_i w_i x_i - s\right). \tag{3.1}$$

where $\theta(x) = 1$ for $x \geq 0$ and $\theta(x) = 0$ for $x < 0$. McCulloch and Pitts demonstrated that any arbitrary logical function can be constructed by an appropriate combination of such elements. Their proof is based on the observation that, in particular, AND-gates and inverters can be realized as special cases of (3.1); therefore, any other logical function can be constructed from these. The model of McCulloch and Pitts for the first time suggested how neurons might be able to carry out logical operations. Their idea of the neuron as a logical threshold element was a fundamental contribution to the field

and has found entrance into numerous later models, albeit often in modified form.

However, the theory of McCulloch and Pitts failed in two important respects. Firstly, it did not explain how the necessary interconnections between neurons could be formed, in particular, how this might occur through *learning*. Secondly, such networks depended on error-free functioning of all their components and did not display the (often quite impressive) error tolerance of biological neural networks.

The psychologist Hebb (1949) suggested an answer to the first question in his now famous book *Organization of Behaviour* (Hebb 1949). According to his suggestion, the connection between two neurons is *plastic* and changes in proportion to the activity correlation between the presynaptic and the postsynaptic cell.

This *Hebb hypothesis* has survived up until today in various mathematical formulations as the essential feature of many network models with learning ability, although its experimental verification remains in dispute. One of its simplest mathematical formulations is

$$\Delta w_i = \epsilon \cdot y(\mathbf{x}) \cdot x_i \tag{3.2}$$

for the change in the synaptic strengths w_i $(i = 1, 2, \ldots, n)$ of a neuron receiving an input $\mathbf{x} = (x_1, x_2, \ldots, x_n)T$ when x_i is the input at the ith synapse. $y(\mathbf{x})$ denotes the excitation of the neuron and $\epsilon > 0$ is a parameter measuring the size of a single learning step. The quantities $y(\mathbf{x})$ and w_i can also be considered as continuous.

With the advent of the computer, it became possible to simulate in more detail the learning capacity of networks made of neurons subject to rules of the above kind and to demonstrate practical applications of such systems.

3.2 The Perceptron

The *perceptron* proposed by Rosenblatt (1958) constituted an important step in this direction. It consists of a fixed number N of elements, each of which is supplied with an "input pattern" through L channels. Each of the input patterns is described by an L-component feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_L)T$

and belongs to one of N " pattern classes." The classification of the input patterns and the required number and the interpretation of the components x_i depends on the application; the x_i might, for example, describe gray levels of image pixels or quantities of a more complex feature extracted from the input pattern by some preprocessing stage. The perceptron shall learn the correct classification of the pattern vectors using known classification examples during a "training phase." For the classification of an input pattern $\mathbf{x}$, each element r computes a binary output value y_r according to

$$y_r = \theta(\sum_{i=1} L w_{ri} x_i). \tag{3.3}$$

The coefficients w_{ri}, $i = 1, 2, \ldots, L$ determine the behavior of the element r. The absence of an "excitation threshold" in (3.3) does not imply a loss of generality. The action of such a threshold can be taken into account without changing the general form of (3.3) by agreeing on a constant input signal $x_1 = 1$. The threshold is then given by the value $-w_{r1}$.

During a training phase, each element adjusts its coefficient w_{ri} in such a way that it only reacts to the input patterns of "its" class C_r with an output value $y_r = 1$. For this to be possible, the *existence of a solution* must first be guaranteed, *i.e.*, there must exist weights w^*_{ri} for which (3.3) correctly solves the classification problem. The satisfaction of this condition depends both on how the problem is posed and on the coding chosen for the pattern vector $\mathbf{x}$. This can be illustrated as follows: Within a particular choice of coding, *i.e.*, an assignment of "features" $x_1, x_2, \ldots, x_L$ to each pattern, each pattern corresponds to a point $\mathbf{x}$ in a "feature space"(possibly of very high dimension). The individual classes C_r can be considered as subsets of points in this space. Each element must assign its output values y_r to points in such a way that the spatial region belonging to the output value $y_r = 1$ includes the points of the class C_r and excludes the points of all other classes C_s, $s \neq r$. However, the flexibility of the separation afforded by the threshold elements of the form (3.3) is limited: geometrically, every choice of weights w_{ri} corresponds to a separation of the feature space by an $L-1$-dimensional "hyperplane" into two regions, one with $y_r = 1$ and the other with $y_r = 0$. If the classes in the space of pattern vectors $\mathbf{x}$ are arranged in a manner which is too "convoluted," then the desired separation by hyperplanes cannot be achieved, and the perceptron algorithm is doomed to fail from the outset. Figure 3.1 offers a simple example. We assume $L = 2$, *i.e.*, each pattern is

characterized by a two-dimensional "feature vector," and only two classes are considered. In this case, the available "hyperplanes" are lines ($L-1=1$), by means of which a complete separation of the classes C_1 and C_2 is evidently impossible.

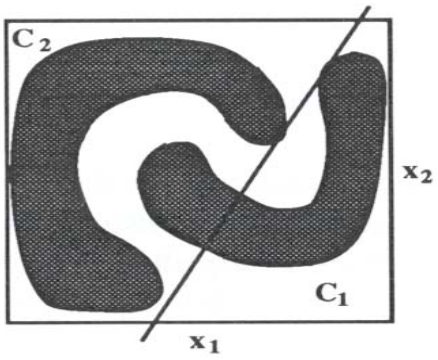

Abb. 3.1: Example of two pattern classes C_1 and C_2 in a two-dimensional feature space of the variables x_1 and x_2 which are not linearly separable.

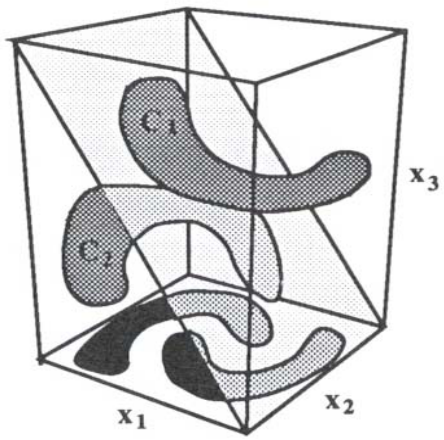

Abb. 3.2: The addition of a further feature x_3 leads to a higher-dimensional feature space, in which the two classes may be linearly separable. Projection onto the x_1-x_2-plane leads back to the nonseparable situation of Fig. 3.1

A way out of such a situation can often be found by appropriate extension of the feature vectors by additional feature variables. These can increase the distinguishability of the classes to the extent that a separation by hyperplanes becomes possible. If, for example, in addition to C_1 and C_2 a further feature variable x_3 can be found that differs sufficiently from C_1 and C_2, the situation shown in Fig. **??** may occur in the resulting $L = 3$-dimensional feature space. A separation of C_1 and C_2 by a plane is now possible. This geometric property of two classes is called *linear separability.*

Linear separability of each class from the union of all the other classes thus

guarantees that the perceptron of Eq.(3.3) can correctly classify all pattern instances, provided its weights w_{ri} are chosen appropriately. The task of finding such a set of weights remains. An attractive approach makes use of a number of *classification examples*, *i.e.*, vectors $\mathbf{x}$ together with a specification of their respective classes. These constitute the input to the perceptron in a "training phase." Every time an element r provides an incorrect output value $y_r \neq y_r(corr)$ in response to some input $\mathbf{x} \in C_s$, its coefficients w_{ri}, $i = 1, 2, \ldots, L$, are changed by an amount

$$\Delta w_{ri} = \epsilon \cdot \left(y_r^{(\mathrm{corr})} - y_r \right) \cdot x_i \tag{3.4}$$

This no longer exactly corresponds to Hebb's rule, but rather the postsynaptic activity y_r in (3.2) is replaced by the difference $d_r = y_r^{(corr)} - y_r$ between the correct output value and the perceptron's current output value y_r. The factor ϵ in front of the expression determines the "learning step size" and must be positive. Eq. (3.4) then acts as an *error correction rule*: In case of a correct answer from the element r, $d_r = 0$, and all of the weights w_{ri} remain unchanged. In case of an incorrect output value, $d_r = \pm 1$ and Eq. (3.4) implies a change in the sum $\sum_i w_{ri} x_i$ by $\pm\epsilon \sum_i x_i 2$. If the output value was too small, this causes an increase in the sum ($d_r = 1$) if the input pattern $\mathbf{x}$ is repeated, and a reduction ($d_r = -1$) if the output was too large. This procedure is known as the *perceptron algorithm*, for which the following *convergence theorem* holds: *Perceptron Convergence Theorem:* (Rosenblatt 1961; Block 1962; Minsky and Papert 1969) Let the classification problem be solvable with appropriate weights w_{ri}^* by means of the perceptron *ansatz* (3.3), and suppose that the feature vectors $\mathbf{x}$ are all bounded, *i.e.*, there exists a constant M, such that $\|\mathbf{x}\| < M$ is always satisfied. Then, with the choice

$$\epsilon_r = 1/\|\mathbf{x}\| \tag{3.5}$$

the perceptron algorithm (3.5) will always find a solution after finitely many adaptation steps for the weights w_{ri} (only the "true" modification steps, *i.e.*, steps with $d_r \neq 0$, are counted).

In the following, we give a proof of this theorem for the mathematically interested reader. However, for an understanding of what follows, there is no harm in skipping the mathematical derivation.

Since all elements operate independently of one another, in the proof of the preceding theorem it suffices to consider a single element, and in the following

we can thus suppress the index r. We denote by $\mathbf{w}^* = (w_1^*, w_2^*, \ldots, w_L^*)T$ a weight vector for which the perceptron solves the classification problem correctly (the existence of a vector with this property is required by the theorem). Hence, there exists a constant $\delta > 0$, such that [1]

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{x} &> \delta, \qquad \text{if } y^{(\mathrm{corr})}(\mathbf{x}) = 1 \\ \mathbf{w}^* \cdot \mathbf{x} &< -\delta, \qquad \text{if } y^{(\mathrm{corr})}(\mathbf{x}) = 0. \end{aligned} \tag{3.6}$$

Here, $y^{(corr)}(\mathbf{x})$ designates that output value which corresponds to a correct classification of the input $\mathbf{x}$ by the element under consideration. Let $\mathbf{w}(t)$ denote the weight vector of the perceptron obtained after t modification steps from some (arbitrary) starting value. For $\mathbf{w}(t)$, the next modification step, *i.e.*, $d = y^{(\mathrm{corr})} - y = \pm 1$, yields

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \epsilon \cdot d \cdot \mathbf{x}, \tag{3.7}$$

and thus

$$\begin{aligned} \mathbf{w}^2(t+1) &= \mathbf{w}^2(t) + 2\epsilon d \cdot (\mathbf{w}(t) \cdot \mathbf{x}) + \epsilon 2 d^2 \mathbf{x}^2 \\ &\leq \mathbf{w}^2(t) + \epsilon^2 d^2 \mathbf{x}^2 = \mathbf{w}^2(t) + 1. \end{aligned} \tag{3.8}$$

In (3.8), we have made use of the relations $d \cdot (\mathbf{w} \cdot \mathbf{x}) = (y^{(corr)} - y)(\mathbf{w} \cdot \mathbf{x}) \leq 0$ and $d2\epsilon 2 = 1/\|\mathbf{x}\|2$. From Eq.(3.8), we obtain an upper bound for the increase in the length $\|\mathbf{w}(t)\|$ with the number of modification steps that have occurred

$$\|\mathbf{w}(t)\| \leq \sqrt{\|\mathbf{w}(0)\|^2 + t}. \tag{3.9}$$

On the other hand, at each modification step, the scalar product $\mathbf{w} \cdot \mathbf{w}^*$ satisfies

$$\begin{aligned} \mathbf{w}(t+1) \cdot \mathbf{w}^* &= \mathbf{w}(t) \cdot \mathbf{w}^* + \epsilon \cdot d \cdot (\mathbf{x} \cdot \mathbf{w}^*) \\ &\geq \mathbf{w}(t) \cdot \mathbf{w}^* + \epsilon \cdot \delta. \end{aligned} \tag{3.10}$$

The last step makes use of Eq.(3.6). Therefore, $\mathbf{w}(t) \cdot \mathbf{w}^*$ grows at least linearly with the number of modification steps

$$\mathbf{w}(t) \cdot \mathbf{w}^* \geq \mathbf{w}(0) \cdot \mathbf{w}^* + t \cdot \delta / M. \tag{3.11}$$

[1] For classes with infinitely many elements, there could always be vectors demanding arbitrarily small δ. In this case, we consider Eq. (6) as a more precise statement of the requirement that a solution exist. Equation (6) implies that this solution is insensitive to sufficiently small perturbations of the weights $\mathbf{w}^*$.

With the help of (3.10), (3.11) and the Cauchy-Schwarz inequality, we thus have

$$\begin{aligned} \mathbf{w}(0) \cdot \mathbf{w} *^{+} t \cdot \delta/M &\leq \mathbf{w}(t) \cdot \mathbf{w} *^{\leq} \|\mathbf{w}(t)\| \cdot \|\mathbf{w}^*\| \\ &\leq \|\mathbf{w}^*\| \sqrt{\|\mathbf{w}(0)\|^2 + t} \end{aligned} \tag{3.12}$$

Since the left side of this inequality is linear and thus grows faster with t than the right side, t cannot become arbitrarily large, *i.e.*, only a finite number of modification steps can occur. This concludes the convergence proof for the perceptron. For a thorough discussion and other approaches to a proof, see for example (Minsky and Papert 1969).

A more flexible variant of the perceptron results if the individual elements do not work completely independently of one another, but rather compete with one another for the correct classification. Equation (3.3) is then replaced by

$$y_r = \begin{cases} 1 & : \quad \mathbf{w}_r \cdot \mathbf{x} > \mathbf{w}_s \cdot \mathbf{x} \quad \forall \quad s \neq r \\ 0 & : \quad \text{else.} \end{cases} \tag{3.13}$$

A learning step occurs every time the index r of the element with $y_r = 1$ deviates from the correct classification s of the input vector. In this case, the weight vectors of the two elements r and s are changed according to (3.4). The resulting learning rule is (note $y_r^{(\text{corr})} = 0$ and $y_s^{(\text{corr})} = 1$)

$$\begin{aligned} \Delta \mathbf{w}_s &= \epsilon \, \mathbf{x}, \\ \Delta \mathbf{w}_r &= -\epsilon \, \mathbf{x}. \end{aligned} \tag{3.14}$$

Here, the procedure also finds a solution after a finite number of modification steps, as long as a solution exists. The greater flexibility of this approach stems from the fact that now each element r can close off for its class not just a half-space, but a conical region bounded by those hyperplanes where $\mathbf{w}_r \cdot \mathbf{x} = \mathbf{w}_s \cdot \mathbf{x}$.

Aside from the convergence theorems, numerous other interesting and important statements can be derived with mathematical rigor for the perceptron (Minsky and Papert 1969). This is possible because the individual elements "interact" either not at all or at most in a very simple way. In spite of this relative simplicity, the insights gained from the perceptron illustrate many typical problems posed by parallel and adaptive systems. In particular, the perceptron permits a relatively far-reaching analysis of its performance and

also of its limitations. The convergence theorem guarantees that whenever a perceptron solution exists, the learning algorithm will find it. Similarly far-reaching results are usually not known for learning rules in more complex, multilayered networks. On the other hand, it soon became evident that for a whole series of practical classification problems, the requirement of the existence of a perceptron solution, *i.e.*, of appropriate weights w_{ri}, is not satisfied, and that the perceptron thus cannot solve such problems. Moreover, other problems do admit a solution in principle but demand that the weights be maintained with a precision growing exponentially with the size of the problem, a requirement which in practice cannot be satisfied (Minsky and Papert 1969). Such limitations of the perceptron have been overcome recently by means of multilayer network models. We will return to this in Section 3.9.

3.3 Associative Memory

Hebb's learning rule in the form of Eq. 3.2) led to another class of models, those of *associative memory*. One of the remarkable properties of the human brain is its ability to draw inferences or *associations* between all types of mental images. For a long time the location of this mental capacity and the respective mode of information storage posed a great puzzle. Although Hebb proposed the synapse as a relevant storage element, it was not compatible with neurophysiological experiments to ascribe to individual synapses a role requiring a degree of reliability similar to that of storage locations in a conventional computer. On the contrary, information storage in neural networks turned out to be remarkably robust with respect to loss or malfunction of some limited portion of the neurons. One possible explanation was provided by storage concepts in which each newly arriving piece of information is stored in a way that is "distributed" over many storage elements. Thus, loss or malfunction of a portion of the memory causes merely a general degrading of the quality of the stored information but not the total loss of individual entries.

A proposal for a network model with such properties was made by Willshaw, Bunemann, and Longuet-Higgins (1969), whose network uses threshold value elements of the type (3.1). The information to be stored is presented in the form of *training pairs* $(\mathbf{x}, \mathbf{y})$. $\mathbf{x}$ plays the role of the *input pattern*, $\mathbf{y}$ that

of the *output pattern*, and both are represented as binary vectors for which the components take values 0 or 1. $\mathbf{x}$ plays the role of the "key" for the associated pattern $\mathbf{y}$. Proper operation of the network model of Willshaw et al. requires that $\mathbf{x}$ be a vector with a large number of components, whereas $\mathbf{y}$ may be of low dimension. The dimension N of $\mathbf{y}$ determines the number of threshold value elements required. Each threshold value element computes, as in the perceptron, a single component y_r of the output pattern. It is given by

$$y_r = \theta\left(\sum_{i=1} L w_{ri} x_i - s_r\right). \tag{3.15}$$

Here, s_r is the threshold of element r and $\theta(.)$ is again the step function defined in connection with (3.1). The right side of (3.15) can be evaluated by N McCulloch-Pitts neurons, which receive the input pattern $\mathbf{x}$ through N common input channels. Information storage occurs in the matrix of the $L \times N$ "synaptic strengths" w_{ri}. These are to be chosen in such a way that (3.15) assigns the correct output pattern $\mathbf{y}$ to each input pattern $\mathbf{x}$.

Willshaw et al. considered the case where p training pairs $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(p)}, \mathbf{y}^{(p)})$ are to be stored, whose input patterns of 1s and 0s each contain the same number k of 1s, *e.g.*, $\mathbf{x} = 010100100$ for $k = 3$. Suppose that the positions of these 1s are not correlated with each other and that their number k is small compared to the total number L of components of a pattern. The input patterns thus consist almost completely of 0s. On the other hand, the number of 1s in the output patterns $\mathbf{y}^{(\nu)}$ need not be restricted in any way.

The storage of a training pair $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{(\nu)}, \mathbf{y}^{(\nu)})$ consists of setting all weights w_{ri} satisfying both $x_i = 1$ and $y_r = 1$ to the value one (all the weights are zero before storage of the first pattern). The remaining weights remain unchanged. This is illustrated in Fig. 3.3a–c for the example of the storage of three training pairs. The horizontal input lines carry the input pattern, the vertical output lines the output pattern. A value $w_{ri} = 1$ is designated by a mark at the intersection of input line i and output line r. Those weights that have been changed for the storage of the most recently offered pattern pair are identified by open circles.

After all of the p training pairs have been stored, the resulting memory matrix becomes

$$w_{ri} = \max_{\nu=1,\ldots,p} \left(y_r^{(\nu)} x_i^{(\nu)}\right). \tag{3.16}$$

The threshold is set to the value $s_r = k - 1/2$.

This choice of a memory matrix and threshold guarantees that all output lines that were active during the storage of a training pair $\left(\mathbf{x}^{(\nu)}, \mathbf{y}^{(\nu)}\right)$ are reactivated if the input pattern $\mathbf{x}^{(\nu)}$ is presented alone. Formally, this results from the relation

$$\sum_i w_{ri} x_i^{(\nu)} = \sum_i w_{ri} \cdot y_r^{(\nu)} x_i^{(\nu)} = \sum_i y_r^{(\nu)} x_i^{(\nu)} = \sum_i x_i^{(\nu)} = k > s_r, \qquad (3.17)$$

which holds provided $y_r^{(\nu)} = 1$.

Figure 3.3d offers an illustrative example. Here, the memory array formed after the steps shown in Fig. 3.3a–c is presented with the input pattern of the training pair stored in step (b). In this case, $k = 2$, and all thresholds take the value $s_r = 3/2$. Each output line r active in storage step (b) has left exactly k weights $w_{ri} = 1$ (r fixed) in the memory matrix; these are highlighted in Fig. 3.3d by open circles. These weights "belong" precisely to those k input lines which together were active in step (b), and, if the same input lines are again activated, they produce the sum $\sum_i w_{ri} x_i = k > s_r$ and thus the excitation of all those output lines r that were active during storage step (b).

Note that Fig. 3.3d also shows that the output pattern can contain, in addition to the correct 1s, a few erroneous 1s. Such an "erroneous 1" is present on the second output line of Fig. 3.3d and is designated by an $(*)$. As illustrated by the example, such errors occur whenever many training pairs activate the same output line r and the opposite output value $y_r = 0$ is to be assigned to an input pattern which agrees partly with many of these pairs. However, it can be shown that for pattern vectors with a sufficiently small proportion of 1s (this assumption is strongly violated in the example of Fig. 3.3), these errors occur with very low probability.

For a statistical estimate of the error probability of 0s, we introduce the additional assumption that every output pattern contains a fixed fraction f' of 1s. However, we make this assumption for convenience of the mathematical discussion only; it is not important for the function of the memory. We denote the fixed fraction of 1s in each input pattern by $f = k/L$.

We consider a pattern ν and a component r of the output, to which the value $y_r^{(\nu)} = 0$ is assigned for pattern ν. What is the probability for equation (3.15) to provide the wrong output value $y_r = 1$ if altogether p patterns are stored?

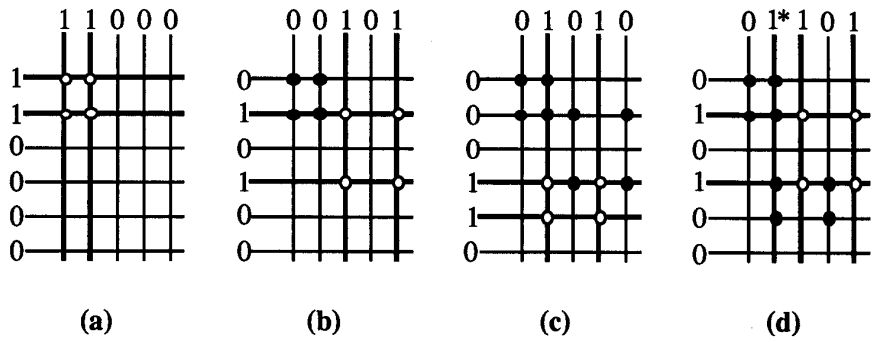

Abb. 3.3: Pattern storage in an associative memory matrix. The input information is provided by the horizontal lines as a 0-1-pattern $\mathbf{x}$ of "activities." The output information $\mathbf{y}$ is the 0-1-pattern on the vertical lines. Each intersection of an output line r and an input line i is assigned a "weight" w_{ri} (marked by a symbol), which is set to the value one if both lines are active simultaneously. The associations between the input and output patterns are stored in the matrix determined in this manner. Fig. 3.3a–c show the implications of this rule for three consecutive training pairs. The output $\mathbf{y}$ corresponding to the input pattern $\mathbf{x}$ can be approximately reconstructed for each output line by summation and thresholding, $\mathbf{y}_r = \theta(\sum_r w_{ri}x_i - s_r)$ (Fig. 3.3d). In Fig. 3.3d, the output pattern reconstructed in this way deviates from the correct pattern in the component designated by an $(*)$.

An incorrect value $y_r = 1$ always occurs if

$$\sum_i w_{ri} x_i^{(\nu)} > s_r. \tag{3.18}$$

Because of the choice $s_r = k - 1/2$ for the threshold, the left side of (3.18) would have to assume its maximal value k. This can only occur if all of the k 1s of the input pattern $\mathbf{x}^{(\nu)}$, *i.e.*, $x_i(\nu)$, coincide with elements $w_{ri} = 1$. Since $y_r^{(\nu)} = 0$, these elements can only come from training pairs $\mu \neq \nu$. Those k values of the index i for which $w_{ri} = 1$ or $x_i = 1$ are therefore uncorrelated with one another. The probability for all k input 1s to coincide with elements $w_{ri} = 1$, and thus the probability for the occurrence of an error in the output value y_r, is therefore

$$P \approx q^k. \tag{3.19}$$

Here, q is the fraction of all weights which have been set to the value 1 during the storage procedure. Since a weight w_{ri} keeps its initial value 0 if and only if it "avoids" the coincidence of $x_i = 1$ (probability f) and $y_r = 1$ (probability f') for all p training pairs (probability each time $1 - ff'$), q is thus given by

$$q = 1 - (1 - ff')^p \approx 1 - \exp(-pff'). \tag{3.20}$$

Equations (3.19) and (3.20) thus yield the estimate for the probability of a "false 1" ($f = k/L$)

$$P \approx (1 - \exp(-pf'k/L))^k . \tag{3.21}$$

The average fraction γ of "false 1s" in the output pattern then becomes

$$\gamma = P(1 - f')/f'. \tag{3.22}$$

For fixed f', γ depends only on k, the ratio p/L of the number of stored patterns, and on the input dimension L. A convenient parameter for discussing the behavior of P is $\alpha = f'p/L$. Figure 3.4 shows the behavior of the error probability P with the number k of 1s per input pattern for several values of α. Below $\alpha \approx 0.1$, the error probability falls off rapidly with decreasing α. There is always an optimal value $k_{opt} = \ln 2/\alpha$ of k that minimizes the error probability for fixed α. The minimum is given by $P_{min} = 2^{-k_{opt}} \approx 0.618^{1/\alpha}$.

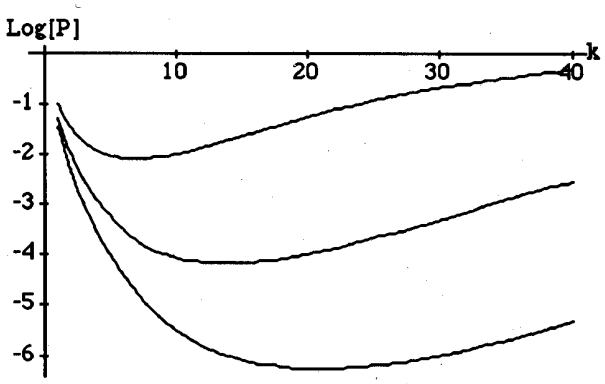

Abb. 3.4: Dependence of error probability (logarithmic scale) P on the number k of 1s of the input pattern in the model of Willshaw et al. for parameter values $\alpha = 0.1$ (upper graph), $\alpha = 0.05$ (middle graph) and $\alpha = 0.025$ (lower graph). For every α, there is an optimal value of k minimizing the error probability. For $\alpha << 1$, errors rapidly become very rare

The choice $s_r = k - 1/2$ for the thresholds "just barely" enables the activation of an output line. If any 1s at all are lacking from the input pattern, the value zero results in all output lines. Similarly, even a single missing "synapse" ($w_{ri} = 0$ instead of $w_{ri} = 1$) prevents the output value $y_r = 1$. Thus, the system has practically no error tolerance with respect to failure of a few "synapses" or a few missing input bits. However, the choice of a lower threshold s_r gives rise to such an error tolerance. This raises the error probability P to

$$P = \sum_{\nu > s_r}^{k} q^{\nu}(1-q)^{k-\nu}\binom{k}{\nu} \tag{3.23}$$

(for $s_r = k - 1/2$ this reduces to the simpler expression (3.21)), but this worsening can be compensated by choosing a correspondingly smaller value for the ratio p/L , *i.e.*, by a storage of a smaller number of patterns. This corresponds to utilization of a smaller fraction of the "storage capacity." The advantage compared to the choice of threshold $s_r = k - 1/2$, however, is that the error rate P is now robust with respect to a limited fraction of missing synapses or with respect to the absence of some percentage of the input 1s. The tolerable fraction of such errors can be estimated as follows: since $s_r < k$ correct input 1s of an input pattern already suffice for activation of all the correct output 1s, even if a relative fraction $\kappa \approx 1 - s_r/k$ of all input 1s were absent, an incorrect output pattern would not result. Here, it is of no consequence whether these errors occur in the input pattern itself or whether they arise due to the lack of a corresponding fraction of the "synapses" $w_{ri} = 1$. In this way, the model offers a way to realize a distributed pattern storage. A similar kind of storage is believed to be realized in the brain.

An especially interesting feature is that one can obtain the matrix w_{ri} using Hebb's Rule (3.2) by adding the condition that the value of a weight must be strictly increasing and be bounded from above by the maximal value one. Beginning with the initial values $w_{ri} = 0$, during a training phase one "forces" the input and output lines to take on successively the binary values of all training pairs $(\mathbf{x}^{(1)}, y^{(1)})$, $(\mathbf{x}^{(2)}, \mathbf{y}^{(2)})$, ..., $(\mathbf{x}^{(L)}, \mathbf{y}^{(L)})$ which are to be stored. For each pair (3.2) is applied with $\epsilon = 1$. Thus, one sets $w_{ri} = 1$ in the first training pair satisfying both $x_i^{(\nu)} = 1$ and $y_r^{(\nu)} = 1$ simultaneously. All w_{ri} for which this never happens remain zero, which finally results in (3.14).

The nonlinear threshold operation by means of the function $\theta(.)$ defined in (3.5) does not allow a mathematical analysis. However, a matrix memory

can also be realized by means of a *linear ansatz*. In this case, the threshold operation does not occur, and the binary values may be replaced by continuous variables. The resulting *linear associative memory* has been investigated by Kohonen (1972, 1984a) and forms the subject of the following section.

3.4 Linear Associative Memory

An important difference between nonlinear and linear systems is the validity of the *superposition principle* in the latter. The linear superposition of several input patterns yields the same superposition of the corresponding output pattern. Whether or not this is a desired property depends on the intended application. However, in general this circumstance does imply a limitation of linear models compared to nonlinear models: only in the latter case can a linear combination of input patterns be associated with an independent output pattern.

In the following, we consider the linear *ansatz*

$$y_r = \sum_{i=1} L w_{ri} x_i. \tag{3.24}$$

Like Eq.(3.5), Eq. (3.24) can be interpreted as the transformation of an input signal $\mathbf{x}$ by a number of "neurons," now assumed linear, into an output signal $\mathbf{y}$.

We are again interested in the use of a system described by Eq.(3.24) as a *memory* for a number of given "training pairs" $(\mathbf{x}^{(\nu)}, \mathbf{y}^{(\nu)})$, $\nu = 1, 2, \ldots, p$. In contrast to the previous section, the components $x_i^{(\nu)}$ and $y_i^{(\nu)}$ can now take arbitrary continuous values. For example, $\mathbf{x}^{(\nu)}$ might be an array of pixel intensities of a gray-level image, and $\mathbf{y}^{(\nu)}$ might contain some information which is to be "associated" with this image. In particular, $\mathbf{y}^{(\nu)}$ may even coincide with $\mathbf{x}^{(\nu)}$. In this case, one has a so-called *autoassociative memory* . At first glance the association of a pattern with itself seems to promise little new information. However, a useful effect results if the association succeeds even in the case of an erroneous or incomplete input pattern. Autoassociation leads in this case to elimination of errors and/or to completion of incomplete input data.

The requirement that the p training pairs $(\mathbf{x}^{(\nu)}, \mathbf{y}^{(\nu)})$ be stored constitutes a condition on the $N \times L$-matrix $\mathbf{W}$ of weights w_{ri}. The simplest approach consists in minimizing the squared error $E[\mathbf{W}]$, averaged over all input patterns of the matrix, which is dependent on the matrix $\mathbf{W}$:

$$E[\mathbf{W}] = \sum_{\nu=1} p \sum_{r=1} N\Big(y_r^{(\nu)} - \sum_{i=1} L w_{ri} x_i^{(\nu)}\Big) 2 = \text{ Minimum.} \tag{3.25}$$

Several solution strategies are possible for minimizing $E[\mathbf{W}]$. The three most important are: (i) exact algebraic minimization by means of the so-called *pseudoinverse*, (ii) application of an iterative *gradient-descent procedure* for stepwise minimization of $E[\mathbf{W}]$, and (iii) use of the *correlation matrix* of training pairs as an approximate solution for the weight array $\mathbf{W}$.

The approaches (i) and (ii) lead essentially to the same solution and maximize the achievable "storage capacity." However, as a solution technique, (i) has the disadvantage of requiring a completely new computation of all w_{ri} for each newly arriving pattern. Hence, this approach is unrealistic, at least as far as applications to neural models are concerned. On the other hand, method (ii) can be formulated as an iterative "learning rule" which, for sufficiently frequent sequential "presentation" of the training pairs to be stored, gradually produces the optimal weights w_{ri}. However, the change of a weight w_{ri} in a learning step also depends on all the other weights w_{rj}, $j \neq i$. In this sense, alternative (iii) is still simpler. Moreover, the required correlation matrix is easy to compute, and its formulation as an iterative "learning rule" takes the form of Hebb's rule. However, in general (iii) does not yield the minimum of $E[\mathbf{W}]$ and hence its utilization of storage capacity is worse than that of the optimal techniques (i) and (ii). This disadvantage is only avoided in the case of pairwise orthogonal pattern vectors, *i.e.*, $\mathbf{x}^{(\nu)} \cdot \mathbf{x}^{(\mu)} = 0$ for $\mu \neq \nu$. In this case, all three techniques are equally good, and (i) and (ii) reduce to (iii).

Following this survey, we now discuss approaches (i), (ii) and (iii) in more detail.

3.5 The Pseudoinverse as a Memory Array

The average error $E[\mathbf{W}]$ is a quadratic polynomial in the weight variables w_{ri}. Minimality of $E[\mathbf{W}]$ demands the vanishing of all first derivatives with

respect to weight variables w_{ri}, *i.e.*, the existence of the $L \times p$ equations

$$\frac{\partial E}{\partial w_{ri}} = 2 \cdot \sum_{\nu=1} p \left(\sum_j w_{rj} x_j^{(\nu)} - y_r^{(\nu)} \right) \cdot x_i^{(\nu)} = 0, \tag{3.26}$$

or, in matrix notation,

$$\mathbf{WXXT} = \mathbf{YXT}. \tag{3.27}$$

Here, $\mathbf{W}$ is the $N \times L$-matrix of weights w_{ri}, $\mathbf{X}$ is a $L \times p$-matrix, whose elements $X_{i\nu}$ are given by the components of the input vector $\mathbf{x}_i^{(\nu)}$, and $\mathbf{Y}$ is a $N \times p$-matrix with elements $Y_{r\nu} = y_r^{(\nu)}$.

Equation (3.26) is a *linear equation* for the weight array $\mathbf{W}$. Comparison with the original "storage condition" (3.24), which in matrix notation takes the form

$$\mathbf{WX} = \mathbf{Y}, \tag{3.28}$$

shows that (3.27) results from (3.28) after "right-multiplication" of both sides by the matrix $\mathbf{X}^T$. If the square matrix $\mathbf{XX}^T$ is invertible, one can solve (3.27) for $\mathbf{W}$, and one obtains

$$\mathbf{W} = \mathbf{YX}^T(\mathbf{XX}^T)-1. \tag{3.29}$$

However, the invertibility of $\mathbf{XX}^T$ requires the presence of N linearly independent input pattern vectors $\mathbf{x}^{(\nu)}$, which is usually not satisfied. For example, the number p of input vectors might be smaller than N; or, although $p > N$, the dimension of the space spanned by the input vectors might be lower than N.

The noninvertibility of the matrix $\mathbf{XX}^T$ indicates that (3.28) possesses a whole family of solution matrices $\mathbf{W}$ forming an affine space. However, the uniqueness of the minimal solution can be restored by imposing an additional requirement. An appropriate condition is the minimization of the squared sum $\sum_{ri} w_{ri}2$ of all weights w_{ri}. This requirement can be incorporated easily into the original *ansatz* by minimizing the new functional $E[\mathbf{W}] + \alpha \sum_{ri} w_{ri}2$ instead of $E[\mathbf{W}]$. This, besides measuring error, also measures the magnitude of an average weight. Here, α is a positive constant, and we take the limit of α approaching zero at the end in order to recover the original minimization problem.

This leads to the new minimization condition

$$\mathbf{W}(\mathbf{X}\mathbf{X}^T + \alpha\mathbf{1}) = \mathbf{Y}\mathbf{X}^T. \tag{3.30}$$

For every $\alpha > 0$, the matrix $\mathbf{X}\mathbf{X}^T + \alpha\mathbf{1}$ has a well-defined inverse (because $\mathbf{u}T(\mathbf{X}\mathbf{X}^T+\alpha\mathbf{1})\mathbf{u} \geq \alpha\|\mathbf{u}\|2$, all its eigenvalues are positive). Hence, combining this with the limit $\alpha \to 0$, we obtain the closed expression

$$\mathbf{W} = \lim_{\alpha\to 0} \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \alpha\mathbf{1})-1 \equiv \mathbf{Y}\tilde{\mathbf{X}} \tag{3.31}$$

for a minimal solution of $E[\mathbf{W}]$. The matrix

$$\tilde{\mathbf{X}} = \lim_{\alpha\to 0} \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \alpha\mathbf{1})-1 \tag{3.32}$$

is known as the *pseudoinverse* or *Moore-Penrose inverse* of $\mathbf{X}$.

Whereas the inverse $\mathbf{X}-1$ of a matrix $\mathbf{X}$ arises in solving the matrix equation $\mathbf{W}\mathbf{X}-\mathbf{Y} = 0$ for given $\mathbf{X}$ and $\mathbf{Y}$ in terms of the variables $\mathbf{W}$, (and exists if and only if there is a unique solution $\mathbf{W}$, which is then given by $\mathbf{W} = \mathbf{Y}\mathbf{X}-1$), the pseudoinverse $\tilde{\mathbf{X}}$ arises in the present, more general problem of minimizing the squared sum $E[\mathbf{W}]$, Eq. (3.25), of the matrix elements of the difference matrix $\mathbf{W}\mathbf{X} - \mathbf{Y}$. In contrast to the stronger condition $\mathbf{W}\mathbf{X} - \mathbf{Y} = 0$, this problem always has at least one solution, which can be expressed in terms of the pseudoinverse $\tilde{\mathbf{X}}$ in the form $\mathbf{W} = \mathbf{Y}\tilde{\mathbf{X}}$. If more than one solution exists, the pseudoinverse chooses the one with the smallest possible sum of the squares of the matrix elements. Unlike the ordinary inverse, which is defined only for quadratic, nonsingular matrices $\mathbf{X}$, the pseudoinverse exists for any matrix (hence in particular for rectangular matrices), and it coincides with the inverse $\mathbf{X}-1$ whenever the inverse is defined.

3.6 Gradient Descent for the Computation of the Memory Matrix

Frequently, it is desired to "memorize" new patterns and/or to change already stored patterns adaptively — while the memory is being used — without having to carry out a completely new computation of the weight array $\mathbf{W}$ every time. This seems to be an important property for a neural model as well.

Such requirements can be taken into account by an iterative procedure for minimization of $E[\mathbf{W}]$. Each iteration step consists of changing all the weights w_{ri} in the direction of the negative gradient of $E[\mathbf{W}]$, or, in matrix notation,

$$\Delta\mathbf{W} = \epsilon \sum_{\nu=1}^{p} \left(\mathbf{y}^{(\nu)} - \mathbf{W}\mathbf{x}^{(\nu)}\right)\left(\mathbf{x}^{(\nu)}\right)^T, \qquad 0 < \epsilon << 1. \tag{3.33}$$

Since $E[\mathbf{W}]$ is a quadratic function of the matrix elements w_{ri}, this procedure leads to a monotonic decrease and eventually to the global minimum of E. In the case of a family of minimal solutions, the asymptotic solution depends on the initial value of $\mathbf{W}$, and, in contrast to the solution using the pseudoinverse, it is generally not characterized by having the smallest possible sum of the $w_{ri}2$.

Equation (3.33) can be regarded as the result of the superposition of p "learning steps," where each learning step corresponds to a term in the ν-summation and can be interpreted as a change of the weights during a "presentation" of the training pair $(\mathbf{x}^{(\nu)}, \mathbf{y}^{(\nu)})$. If every training pair occurs with the same probability and the "learning step size" ϵ is sufficiently small, then on the average (3.33) corresponds to the simpler prescription

$$\Delta\mathbf{W} = \epsilon'(\mathbf{y}^{(\nu)} - \mathbf{W}\mathbf{x}^{(\nu)})(\mathbf{x}^{(\nu)})^T, \qquad \epsilon' = \epsilon/p. \tag{3.34}$$

Comparison of 3.34 and (3.4) shows that (3.34) is nothing more than a variant of the perceptron rule discussed above. The present derivation augments the previous discussion of the perceptron rule by showing us that this rule can be interpreted as a gradient descent procedure for the average squared response error.

3.7 The Correlation Matrix Memory

The perceptron rule (3.34) requires that a matrix multiplication $\mathbf{W}\mathbf{x}^{(\nu)}$ is carried out for each learning step. Hence, the change of any single weight w_{ri} involves the values of all the remaining weights w_{rj}, $j = 1, \ldots, L$. In order to carry out the procedure in practice, for example in very large scale integrated (VLSI) circuits, it would be desirable to have a simpler rule that would work

without this dependence. In fact, in many cases one can do without the term $\mathbf{W}\mathbf{x}^{(\nu)}$ in (3.34). This leads to a rule of the form

$$\Delta\mathbf{W} = \epsilon\Big(\mathbf{y}^{(\nu)}(\mathbf{x}^{(\nu)})^T - \mathbf{W}\Big), \qquad \epsilon > 0. \tag{3.35}$$

Here, as opposed to (3.34), an additional decay term $-\mathbf{W}$ has been introduced for the sole purpose of automatically normalizing $\mathbf{W}$; it can be left out if the normalization is otherwise guaranteed.

In the limit of small step size ϵ the matrix $\mathbf{W}$ converges by means of (3.35) to the *correlation matrix* $\langle \mathbf{y}\mathbf{x}^T \rangle$, where $\langle . \rangle$ denotes averaging over the presented pattern pairs. A matrix memory based on this choice of $\mathbf{W}$ is therefore known as a *linear correlation matrix memory* . If the training pairs all occur with equal frequency, one has

$$\mathbf{W} = \frac{1}{p} \sum_{\nu=1}^{p} \mathbf{y}^{(\nu)} \left(\mathbf{x}^{(\nu)}\right)^T . \tag{3.36}$$

For pairwise orthogonal pattern vectors $\mathbf{x}^{(\nu)}$, one easily sees that (3.36) leads to a memory matrix with the desired property $\mathbf{W}\mathbf{x}^{(\nu)} = \|\mathbf{x}^{(\nu)}\|2 \cdot \mathbf{y}^{(\nu)}$. In this case, the correlation matrix memory evidently works in an error-free manner, (the factor in front, $\|x^{(\nu)}\|2$, can be regarded as an "intensity normalization" and disappears if normalized input vectors $\mathbf{x}\nu$ are used). In particular, a maximum of $p = N$ such pattern vectors can be stored in this manner.

Deviations from pairwise orthogonality lead to " cross-talk" between different patterns and, thus, to a decreased storage capacity: for an input pattern $\mathbf{x}^{(\nu)}$, the resulting output signal is

$$\mathbf{y} = ||\mathbf{x}^{(\nu)}||^2 \left(\mathbf{y}^{(\nu)} + \sum_{\mu \neq \nu} \mathbf{y}^{(\mu)} \; \frac{\mathbf{x}^{(\mu)} \cdot \mathbf{x}^{(\nu)}}{||\mathbf{x}^{(\nu)}||^2} \right). \tag{3.37}$$

Equation (3.37) shows that, superimposed on the correct output pattern $\mathbf{y}^{(\nu)}$, there are contributions from all the remaining patterns μ, $\mu \neq \nu$ for which the scalar product $\mathbf{x}^{(\nu)} \cdot \mathbf{x}^{(\mu)}$ with the input pattern $\mathbf{x}^{(\nu)}$ does not vanish. A proper functioning of the linear correlation matrix memory thus requires that these scalar products be small and, hence, that the patterns be at least approximately orthogonal.

As a consequence, the operation of a linear correlation matrix memory can be significantly improved by a previous *orthogonalization* of the input patterns

$\mathbf{x}^{(\nu)}$. A simple and often appropriate procedure (*e.g.*, for many kinds of image data) is a *high-pass filtering* of the input signal. The slowly varying parts of signals are then suppressed, and only the "high-frequency" (in space or time) part of the signal is kept. Subtracting from each component, its average value can be regarded as the simplest version of such *high-pass filtering*.

The models discussed so far have no *feedback*. Feedback is present if some of the input is provided by the output lines. This situation, which complicates a theoretical analysis considerably, is almost always found in real nerve systems. The smallest degree of complication occurs in the case of the linear matrix memory, considered here. A qualitative summary of its behavior when feedback is present can easily be given. If the dimensions of the input and output vectors agree, $(L = N)$, the process of repeatedly feeding the output back into the input is equivalent to replacing the memory matrix $\mathbf{W}$ by the matrix taken to some higher power. After t loops, the initial vector $\mathbf{x}(0)$ becomes

$$\mathbf{x}(t) = \mathbf{W}^t \mathbf{x}(0). \tag{3.38}$$

For a diagonalizable memory matrix $\mathbf{W}$, $\mathbf{x}(t)$ converges (up to a normalization factor) for large t to its projection on the eigenspace corresponding to the eigenvalue of $\mathbf{W}$ with the largest absolute value. The components of $\mathbf{x}(0)$ along the eigenvectors with small eigenvalues of $\mathbf{W}$ fall off most rapidly. If, for example, $\mathbf{W}$ has been determined using (3.36), and if p stored patterns are approximately orthogonal to each other, then the eigenvalues of $\mathbf{W}$ may form two "clusters", one cluster consisting of p eigenvalues of nearly the same magnitude near $1/p$, the other consisting of $N-p$ eigenvalues near zero. The eigenvectors corresponding to the latter eigenvalues are approximately orthogonal to the stored patterns. Hence, an input vector is generally "driven" in the direction of the "most similar pattern" among the p stored patterns. Competition between the stored patterns eventually occurs after many iterations, and $\mathbf{x}(t)$ converges to the eigenvector whose eigenvalue has the largest absolute value. Since $\mathbf{W}$ is nearly degenerate in the space spanned by the stored patterns, this eigenvector need not necessarily agree with any of the stored patterns.

Since the eigenvalue of greatest magnitude will usually differ from unity, the norm of $\mathbf{x}(t)$ gradually tends either to zero or infinity. Hence, for a realistic model, the introduction of nonlinearities is unavoidable, at least for stabilization. One of the earliest suggestions of this kind goes back to Anderson

et al. (1977) and is known as the "Brain State in a Box" ("BSB" model), since an appropriate nonlinearity constrains $\mathbf{x}(t)$ within a multidimensional box.

The mathematical analysis of systems with feedback of this type turns out to be much more difficult than in the linear case. In particular, it seemed quite hopeless for a long time to go much beyond computer simulations for nonlinear threshold value models with McCulloch-Pitts neurons. This changed in 1982 due to an important idea of Hopfield (1982), which forms the subject of the following section. For other important contributions related to these questions see, for example, the papers by Grossberg (1976ab,1978), Kohonen (1984a), as well as Cohen and Grossberg (1983).

3.8 The Hopfield Model

If a portion of the output lines is fed back to the inputs, the corresponding portion of the output patterns $\mathbf{y}$ can contribute to the input pattern $\mathbf{x}$. An especially interesting case arises for $\mathbf{y} = \mathbf{x}$, *i.e.*, every input pattern is associated with itself as output (*autoassociation*). If one presents an incomplete input pattern to a recurrent system in which such training pairs are stored, then at first a correspondingly incomplete output pattern results. However, if output is fed back, the intact portion may be sufficient for reconstruction of part of the missing input data. The system may react to the improved input pattern with an improved output, which in turn reconstructs even more of the input pattern, etc. until finally the system winds up in a state in which the input pattern is completely restored.

Such feedback mechanism can enable recall of the complete pattern on the basis of an *incomplete input fragment*. Such a capability of pattern restoration is an important requirement for high-performance data processing and a prominent feature of biological nervous systems, which are highly optimized in the processing of incomplete information from their natural environment.

Due to feedback, every neuron affects the inputs to all the other neurons. The behavior of such a system is generally quite difficult to analyze. However, by exploiting an analogy to interacting many-particle systems from statistical physics, Hopfield (1982) was able to characterize the behavior of an interesting class of such systems in an elegant model.

Hopfield's original model employs McCulloch-Pitts neurons. In the following, we give a version with "±1-neurons". Because of the feedback, now the input pattern of each neuron i is constructed from the states y_j of all the other neurons. The state of neuron i at the latest time step is determined according to

$$y_i^{(\text{new})} = \text{sgn}\left(\sum_{j,j\neq i} w_{ij} y_j^{(\text{old})}\right). \tag{3.39}$$

Here, sgn(x) deno tes the " sign function", *i.e.*, is equal to +1 for $x \geq 0$ and -1 otherwise. The "update-steps" (3.39) are carried out " asynchronously," *i.e.*, the state of a neuron is updated at discrete times chosen to be uncorrelated among the neurons.

The solvability of the models follows from the requirement of *symmetric matrix elements*, *i.e.*, $w_{ij} = w_{ji}$ for all index pairs (i, j). In this case, (3.39) describes the stochastic dynamic of a *physical spin system* with an " energy function"

$$H(\mathbf{y}) = -\frac{1}{2}\sum_{i,j} w_{ij} y_i y_j, \tag{3.40}$$

where $w_{ii} = 0$.

Whenever (3.39) leads to a change $\Delta y_i \neq 0$ of y_i, it can be written in the form

$$\Delta y_i = 2 \cdot \text{sgn}\left(\sum_{j,j\neq i} w_{ij} y_j\right) \tag{3.41}$$

By symmetry of the w_{ij}, the corresponding change ΔH of H is then

$$\begin{aligned} \Delta H &= -\Delta y_i \cdot \sum_j w_{ij} y_j \\ &= -2 \cdot \left\| \sum_j w_{ij} y_j \right\| \leq 0, \end{aligned} \tag{3.42}$$

i.e., H decreases until either the quantities $\sum_j w_{ij} y_j$ all vanish (an exceptional situation arising only for "pathological" choices of w_{ij}), or the adaptation rule (3.39) does not yield any further change in state. In this (common) case, the system reaches a stationary " fixed point."

This allows a rather clear interpretation of the time evolution of the neural activities described by (3.39). $H(\mathbf{y})$ defines a " potential surface" on the

state space of all possible binary vectors $\mathbf{y}$. Starting from an initial state $\mathbf{y}(0)$, the system moves downhill along the gradient of $H(\mathbf{y})$ until it comes to rest at a local minimum (a "perpetual" descent is impossible, since only finitely many states are available to the system). If the input pattern defines the initial state $\mathbf{y}(0)$, the minimum attained by the network is the output associated with the input. Every minimum in the potential surface is the lowest point of a "basin" or "sink" surrounding it. All the input patterns within this basin are attracted to the basin minimum by the system dynamics and, thus, yield the same output pattern. Hence, one also refers to *basins of attraction* surrounding the local minima.

By an appropriate choice of w_{ij}, one can "mold" the potential surface and, in particular, place local minima at desired target patterns $\xi\nu$. The system dynamics will then be able to restore a fragmentary input pattern to that target pattern $\xi\nu$, whose basin of attraction encloses the input pattern. The completion of fragmentary information in the Hopfield model thus is obtained through gradient descent on a potential surface.

The choice of the w_{ij} is based on the specified target patterns to be stored. For uncorrelated binary patterns consisting of equally many positive and negative elements an appropriate choice is

$$w_{ij} = \frac{1}{N} \sum_{\nu=1}^{p} \xi_i^\nu \cdot \xi_j^\nu. \tag{3.43}$$

Here, N is the number of neurons, p is the number of patterns and ξ_i^ν the ith component of the νth pattern vector ξ^ν, $\nu = 1, 2, \ldots, p$.

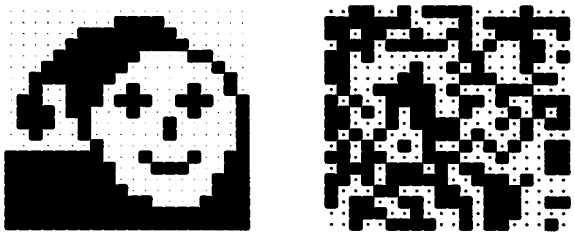

Abb. 3.5: (left) This pattern is stored together with 19 others in a Hopfield model consisting of 400 neurons. (right) All of the other 19 patterns are "random patterns" of the type shown; 50% of randomly chosen pixels are black.

In the following, we present an example of a simulation for a network consisting of 400 neurons. In this case, 20 patterns are stored according to the prescription (3.43). The first two of these patterns are shown in Figure 3.5. Each -1 is represented by a white pixel, each +1 by a black pixel. Only the first pattern represents a recognizable motif (Fig. 3.5a); all of the remaining 19 patterns are "random patterns," each consisting of 50 percent randomly chosen white and black pixels; a representative example is shown in Figure 3.5b.

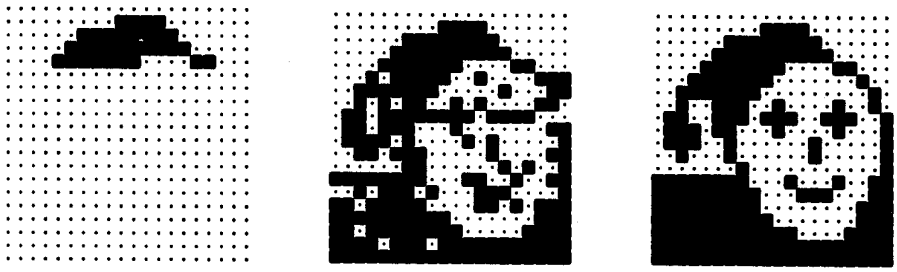

Abb. 3.6: Completion of a fragmentary input pattern. Only the upper 25% of pattern 1 is presented to the network (left). After one timestep, the complete pattern can already be recognized (middle); two steps later, the pattern has been correctly completed (right).

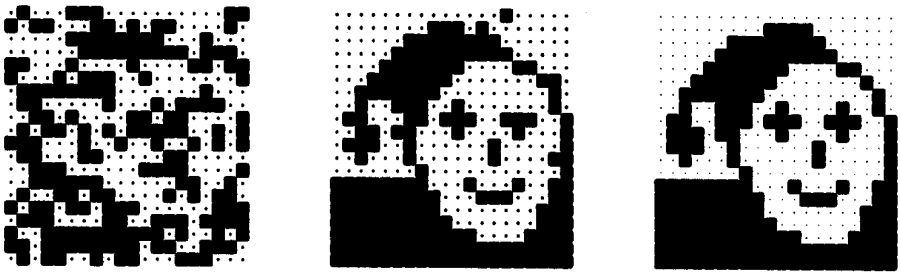

Abb. 3.7: Reconstruction of a noisy input pattern. This time, the input is the complete pattern 1, but, with a probability of P=0.3, every pixel of the image has been changed (left). After only one timestep, nearly all of the errors are eliminated (middle), and after an additional step the correct pattern 1 is restored (right).

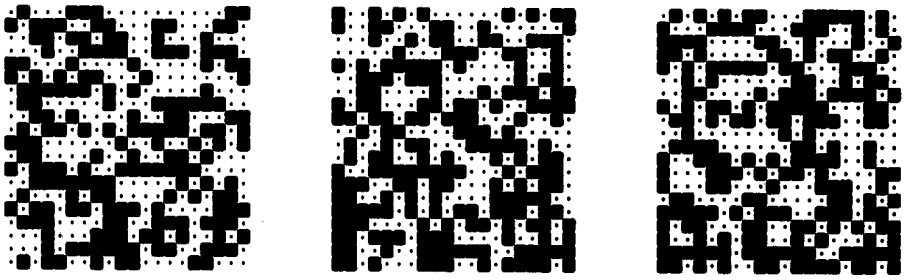

Abb. 3.8: Like the preceding sequence of images, but for P=0.4. In this case, the network is no longer able to restore the original pattern, and it converges to one of the 19 other random patterns.

In Figure 3.6, we see the reaction of the network, if just the upper quarter of pattern 1 is presented as input. In the course of a few timesteps (each timestep includes update steps for all neurons) the pattern is correctly completed.

Figure 3.7 shows a similar simulation. This time, pattern 1 is corrupted by changing each pixel of the image with a probability P=0.3. This corresponds to the presence of intense " signal noise." Although the original motif is hardly recognizable, within a few time steps all of the "errors" have been corrected.

Figure 3.8 shows a repetition of this simulation, but this time with P=0.4. In this case, the network is no longer able to restore the original motif, and the output pattern converges to one of the other stored random patterns.

The weight choice (3.43) is sufficient for pattern recall, provided that the number p of stored patterns is not too large. If the number of patterns is increased beyond a critical threshold, the character of the potential surface changes, and the system no longer functions as a memory for the specified input patterns. This can be qualitatively understood as follows. If all of the neurons are in a pattern state, for example $\mathbf{y} = \xi^1$, then

$$\begin{aligned} \sum_j w_{ij} y_j &= \frac{1}{N} \sum_j \left(\xi_i^1 \cdot (\xi_j^1)^2 + \sum_{\nu=2}^{p} \xi_i^\nu \xi_j^\nu \xi_j^1 \right) \\ &= \xi_i^1 + \frac{1}{N} \sum_{j,\nu>1} \xi_i^\nu \xi_j^\nu \xi_j^1. \end{aligned} \tag{3.44}$$

After separation of the terms with $\nu = 1$, the remaining summation on the right side consists of $N \cdot (p-1)$ uncorrelated terms of value ± 1 and with average value zero. Hence, the sum is itself again a random variable with average value zero, but with variance $\sqrt{N(p-1)}$, and we can write (3.44) approximately as

$$\sum_j w_{ij}\xi_j^1 = \xi_i^1 + \eta \cdot \sqrt{\frac{p-1}{N}}, \tag{3.45}$$

where η is a normally distributed random variable with variance one. The second term in (3.45) shows that the stored patterns act like " Gaussian noise" superimposed on the currently active pattern. Nevertheless, provided $p << N$, the first term in (3.45) dominates, and the system is immune to the noise, since in this case (3.39) does not lead to a change in any neuron states. However, if p gets to be of order N, the influence of the noise becomes comparable to the effect of the currently active pattern itself. In that case, we can no longer expect to find stability for any stored pattern. A more precise computation shows that the critical transition occurs at $p \approx 0.146N$. In the analogous physical spin system, one encounters at this value a *phase transition* to a so-called *spin glass state* (Amit et al. 1985).

The choice (3.43) for the storage of given patterns is not the only possible one. By means of more general procedures, for example iterative methods, or by a more sophisticated coding of the patterns, on can store a larger number of patterns. However, for an estimate of the *storage capacity* of a network, the mere number of patterns that can be stored is not the only important variable. In addition, one has to consider the *information content* per pattern as well as the information contained in the synaptic strengths w_{ij}. For a thorough discussion of these interesting questions, the reader is referred to the literature (see Palm 1980, 1981; Amit et al. 1985, 1987; Gardner and Derrida 1988; Buhmann et al. 1989).

The Hopfield model of associative memory is less than optimal in many respects. For example, it is ill suited for the storage of correlated patterns. Another problem arises in connection with invariance: the model judges the similarity of patterns exclusively according to the number of pixels that coincide. Hence, it is unable to recognize the equivalence of patterns that differ only by a simple transformation, such as, by a translation.

Nonetheless, the model is of great conceptual significance, since it constitutes a fully connected neural network for which many questions can be given an

analytical answer. In particular, the Hopfield model initiated the use of many highly developed mathematical methods of statistical physics and thus made important new tools available for the field of neural computation. Therefore, it formed the basis for numerous new developments and motivated important new questions, and it was thus a forceful stimulus for the major upsurge in " neural computing" at the beginning of the eighties.

3.9 The Back-Propagation Algorithm

In the Hopfield model, every neuron is connected to every other neuron. Hence, with respect to its connections, the model has no "internal structure" and is "homogeneous." However, neural networks are usually structured. A structure encountered frequently results from connecting several layers of neurons in series. The first layer is usually reserved for input patterns. Every neuron of this layer sends out connections to every neuron of the next layer. This continues until the last layer has been reached, whose activity pattern constitutes the output.

Each individual layer can perform a partial transformation of the activity pattern of the preceding layer. For the perceptron — corresponding essentially to a single layer — we saw that a serious limitation of the possible transformations between input and output occurs. Hence, an important question concerns how to overcome the limitations of the perceptron by connecting several layers in series and thus concatenating their transformations.

In contrast to the perceptron and to nets of the Hopfield type, a layered feed-forward network contains *hidden units* that are not directly connected to input or output lines. Therefore, the activity state of these neurons cannot be affected directly by the "outside world," but can only be influenced indirectly through the internal circuitry of the network. The perceptron convergence theorem described in Section 3.2 guarantees that the weights of a network with a single layer of units can be trained with a finite number of adaptation steps. However, this theorem cannot be generalized to feed-forward networks with hidden units. Because the hidden units are only indirectly affected by input signals, the following problem arises: if the given task has been performed badly, it is not clear which of the weights are responsible for the bad result and how they have to be changed. This problem is known as the *credit assignment problem* and was one of the reasons which led to the demise

of the perceptron and its multilayer successors in the 1960s. The *back-propagation algorithm* (Werbos 1974; Rumelhart, Hinton and Williams 1986) is an interesting approach to solve this problem. We describe this procedure for a network which consists of three layers: an input layer, a "hidden layer," and an output layer, as shown schematically in Fig. 3.9.

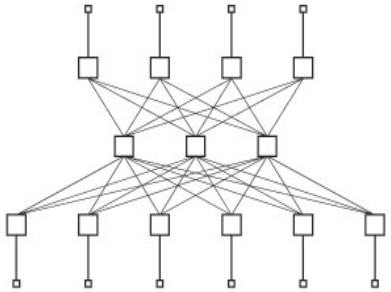

Abb. 3.9: Three-layer neural net. Each layer sends connections to the layer just above it. The input pattern is applied to the neurons of the bottom layer, while the neuron activities of the top layer constitute the output pattern.

We designate neurons of the output layer, the hidden layer, and the input layer by denoting indices i, j and k, respectively. In contrast to the earlier models, here each neuron has a continuous output activity between zero and one. The activity s_j of a neuron j of the hidden layer is given by

$$s_j = \sigma\left(\sum_k w_{jk} s_k\right). \tag{3.46}$$

Here, s_k are the activities of the neurons k in the input layer, *i.e.*, we identify s_k with the components x_k of the input vector. $\sigma(x)$ is a *sigmoid function*, *i.e.*, $\sigma(x)$ is nonnegative, everywhere monotonically increasing, and approaches the asymptotic saturation values zero or one, respectively for $x \to \pm\infty$. $\sigma(x)$ describes the response of a neuron to a total synaptic input x. A frequent choice for $\sigma(x)$ is the " Fermi function"

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{3.47}$$

presented in Fig. 3.10.

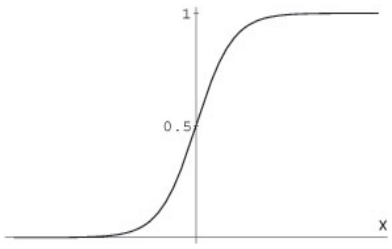

Abb. 3.10: Graph of the Fermi function $\sigma(x) = (1+\exp(x))^{-1}$, a typical choice for the response function σ of a neuron

The activities of the neurons of the output layer are

$$s_i = \sigma\left(\sum_j w_{ij} s_j\right), \tag{3.48}$$

and provide the output values $y_i \equiv s_i$. According to (3.46) and (3.48), for each input pattern $\mathbf{x}$ an output pattern $\mathbf{y}$ is assigned. This assignment depends on the values of the synaptic strengths w_{ij} from the hidden layer to the output layer and on the synaptic strengths w_{jk} from the input layer to the hidden layer. [2]Equations (3.46) and (3.49) do not contain explicit "excitation thresholds". These can be taken into account in the form of synaptic strengths w_{i0} and w_{j0} by taking for each layer $s_0 = -1$.

We now seek w_{ij} and w_{jk} such that the network maps some given number of input patterns $\mathbf{x}^\nu$ onto given output patterns $\mathbf{y}^\nu$, $\nu = 1, 2, \ldots, p$. A measure of how well the network performs this task is the sum of the squared errors over all training pairs $(\mathbf{x}^\nu, \mathbf{y}^\nu)$

$$E = \frac{1}{2}\sum_{\nu=1}^{p}\sum_i \left(y_i^\nu - s_i(\mathbf{x}^\nu)\right)^2. \tag{3.49}$$

For a set of given, fixed training pairs, E is a function of all the synaptic strengths w_{ij} and w_{jk}. Here, the w_{ij} and w_{jk} are optimally chosen if the error E is minimized. The determination of appropriate synaptic strengths is hence equivalent to the problem of minimizing the function E. The gradient descent procedure offers the simplest way of doing this. The back-propagation

2 †

algorithm is a parallelized computational scheme for carrying out an approximate gradient descent for E.

To do this, all of the w_{ij} and w_{jk} are modified iteratively according to $w_{ab}^{new} = w_{ab}^{old} + \Delta w_{ab}$ where

$$\Delta w_{ab} = -\alpha \cdot \frac{\partial E}{\partial w_{ab}}. \tag{3.50}$$

For sufficiently small $\alpha > 0$, one will then move along the direction of steepest descent of E. The change of E during such an iteration step is approximately

$$\Delta E = \sum_{ab} \frac{\partial E}{\partial w_{ab}} \Delta w_{ab} = -\alpha \sum_{ab} \left(\frac{\partial E}{\partial w_{ab}} \right)^2 \leq 0. \tag{3.51}$$

The derivatives $\partial E / \partial w_{ab}$ are obtained using the chain rule. For the connections w_{ij} between the hidden layer and the output layer we have

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{\nu} (y_i^{\nu} - s_i(\mathbf{x}^{\nu})) \cdot \sigma'(\sum_{j'} w_{ij'} s_{j'}) \cdot s_j; \tag{3.52}$$

and for the connections w_{jk} from the input layer to the hidden layer we have

$$\begin{aligned} \frac{\partial E}{\partial w_{jk}} &= -\sum_{\nu} \sum_{i} (y_i^{\nu} - s_i(\mathbf{x}^{\nu})) \cdot \sigma'(\sum_{j'} w_{ij'} s_{j'}) \cdot w_{ij} \cdot \frac{\partial s_j}{\partial w_{jk}} \\ &= -\sum_{\nu} \sum_{i} (y_i^{\nu} - s_i(\mathbf{x}^{\nu})) \cdot \sigma'(\sum_{j'} w_{ij'} s_{j'}) \cdot w_{ij} \\ &\quad \times \sigma'(\sum_{k'} w_{jk'} s_{k'}) \cdot s_k. \end{aligned} \tag{3.53}$$

Both expressions consist of sums over contributions from specified input training pairs $(\mathbf{x}^{\nu}, \mathbf{y}^{\nu})$. If α is sufficiently small, it makes little difference if just one ν-term of (3.52) or (3.53) is taken into account at each iteration (3.50), provided that every term is on the average included equally often. This leads to the update rules

$$\begin{aligned} \Delta w_{ij} &= \alpha \cdot \epsilon_i^{\nu} \cdot s_j s_i (1 - s_i), \\ \Delta w_{jk} &= \alpha \cdot \sum_i \epsilon_i^{\nu} \cdot s_k s_i (1 - s_i) \cdot w_{ij} \cdot s_j (1 - s_j) \end{aligned} \tag{3.54}$$

for the w_{ij} and w_{jk} connecting hidden and output layer, and connecting input and hidden layer, respectively. Here, we have defined $\epsilon_i^{\nu} = y_i^{\nu} - s_i(\mathbf{x}^{\nu})$ for

the ith "output error" in the νth input pattern, and we have employed the expression $\sigma'(x) = \sigma(x)(1-\sigma(x))$ which is valid for the Fermi function (3.47).

Expressions (3.54) can easily be generalized to the case of more than one hidden layer. In the following, let a and b designate arbitrary neurons of two consecutive layers, and let b lie in the layer preceding a. The change Δw_{ab} of the weight w_{ab} under an iteration with the specified input training pair ν is a summation over contributions D_{γ_i}. Each contribution belongs to one neuron i of the output layer and to a sequence γ_i of connections leading from neuron a to neuron i. The summation is to be performed both over all the different sequences of this kind, visiting each layer between a and the output layer only once and over all possible choices of the output neuron i, *i.e.*,

$$\Delta w_{ab} = \sum_i \sum_{\gamma_i} D_{\gamma_i}. \tag{3.55}$$

Each contribution D_{γ_i} consists of a product of factors along the "connecting path" γ_i. The individual factors are obtained according to the following rules:

1. For each "visited" neuron n along the path γ_i, one obtains a factor $s_n(1 - s_n)$, where s_n is the activity of neuron n.

2. For each connection between two consecutive neurons n, n' along the path γ_i, one obtains a factor $w_{nn'}$.

3. Additionally, one has a factor $\alpha \cdot \epsilon_i^\nu \cdot s_b$. Here, ϵ_i^ν is the output error of the neuron i at the end of the connecting path.

Equations (3.54) allows the following interpretation: for each iteration, a training pair $(\mathbf{x}^\nu, \mathbf{y}^\nu)$ is selected and the activities of the neurons in the input layer are set to values $\mathbf{x}^\nu$. On the basis of the resulting neuron activities in the remaining layers and the error ϵ_i^ν occurring at the output layer, the network carries out a "learning step" such that the output error for pattern ν is decreased.

The hope is to gradually reduce the error E to zero or at least to negligibly small values for all specified input patterns, provided sufficiently many learning steps are made. However, the problem of *local minima* can arise. As a rule, E is an extremely complicated function of all the synaptic strengths w_{ab} and, hence, it can have numerous local minima. Depending on the initial

values specified for the w_{ab}, the gradient-descent method always leads to the nearest minimum, independently of how far it lies above the absolute minimum. Thus, the learning procedure can get "stuck" prematurely although the network has not yet solved the problem. Whether or not a good minimum is attained depends in a generally unpredictable way on the initial values for the synaptic strengths and on the (generally unknown) form of the " error surface" E. A further difficulty is caused by parameter regions, for which the height of the error surface hardly varies with w_{ab}. There, the gradient is very small and, thus, the adaptation steps (3.55) yield negligible changes. This difficulty is the price for a completely "general" learning algorithm, which is supposed to solve a given problem without any a priori information.

In spite of these problems, the back-propagation algorithm represents a significant step forward. In particular it allows the solution of problems that cannot be solved with a single-layer perceptron. One problem of this kind that is frequently considered is the logical " exclusive-or-gate," which assigns the output value 1 to an input if and only if one input is equal to 0 and the other is equal to 1.

In the back-propagation algorithm, the solution of such problems becomes possible because, in contrast to the perceptron, the system has access to additional, " hidden" neurons. The activity of these neurons provides an *internal representation* of the input patterns. By evolving appropriate connections w_{jk}, the system can develop internal representations that make the problem solvable for the following layer. We demonstrate this by means of a simulation example, the so-called " encoder problem" (Rumelhart et al. 1986).

We consider a network consisting of three layers. The input and output layers each contain N neurons, numbered from one to N. The middle layer contains $M < N$ neurons. The learning task of the network is to respond to the activation of a single neuron n in the input layer with the activation of a single neuron n in the output layer, *i.e.*, activation of that neuron whose index coincides with the index of the activated input neuron.

If the hidden layer also had N neurons, the solution would be simple and evident: each input neuron would be connected directly via one of the hidden neurons to "its" output neuron. However, since the layer in between possesses *less than* N neurons, it constitutes a "bottleneck" for the transfer of information, and the network must find some way of getting the information

through this bottleneck. One possibility consists in discovering an appropriate data coding — hence the name " encoder problem" — which can make do with M elements for the representation of the information.

Figure 3.11 shows the result of a computer simulation for a network with $N = 8$ and $M = 3$ after 10,000 learning steps with a step size of $\alpha = 0.25$. The initial values of all connections were chosen to be pseudo-random numbers in the interval $[-2,\ 2]$. Each of the eight specified input patterns $\mathbf{x}^\nu$ was given by $x_k^\nu = 0.1 + 0.8\delta_{k\nu}$, *i.e.*, the νth neuron received the input 0.9, all others 0.1. The inputs 0.9 and 0.1 for "active" and "inactive" were used to avoid convergence difficulties, since the Fermi function $\sigma(x)$ produces binary outputs zero and one only for $x = \pm\infty$ (this would require infinitely large weights w_{ab}).

Figure 3.11 shows for each of the eight possible input patterns the resulting neuron activities. The bottom row of each picture shows the input layer, the middle row consists of the three "hidden" neurons, and the upper row shows the output layer. Each square symbol stands for one neuron, whose activity is indicated by the size of the square.

One sees that the network has solved the problem successfully. A look at the activity patterns of the middle layer reveals the solution strategy developed by the network. The neurons of this layer have organized their connections in such a way as to assign to each of the eight input patterns a 3-bit binary code, thus enabling the transfer of the required information through the "bottleneck."

This example shows how a network can "discover" interesting *internal data representations*, in this case the binary code. This is relevant to a key question of neural computing: What internal data representations are required in order to solve a given problem by means of a massively parallel network? With the back-propagation algorithm one has a method to construct networks performing some desired task. One then can analyze the structure of the networks thus obtained in order to gain new insights how parallel networks can solve various computational tasks. This approach, occasionally termed "neurophysiology in the computer," may also help to interprete neurophysiological findings about "real" neural networks and to guide new experiments. For instance, interesting parallels have been noted between the response of "neurons" in the computer and neurons in biological networks (Sejnowski and Rosenberg 1987; Zipser and Andersen 1988).

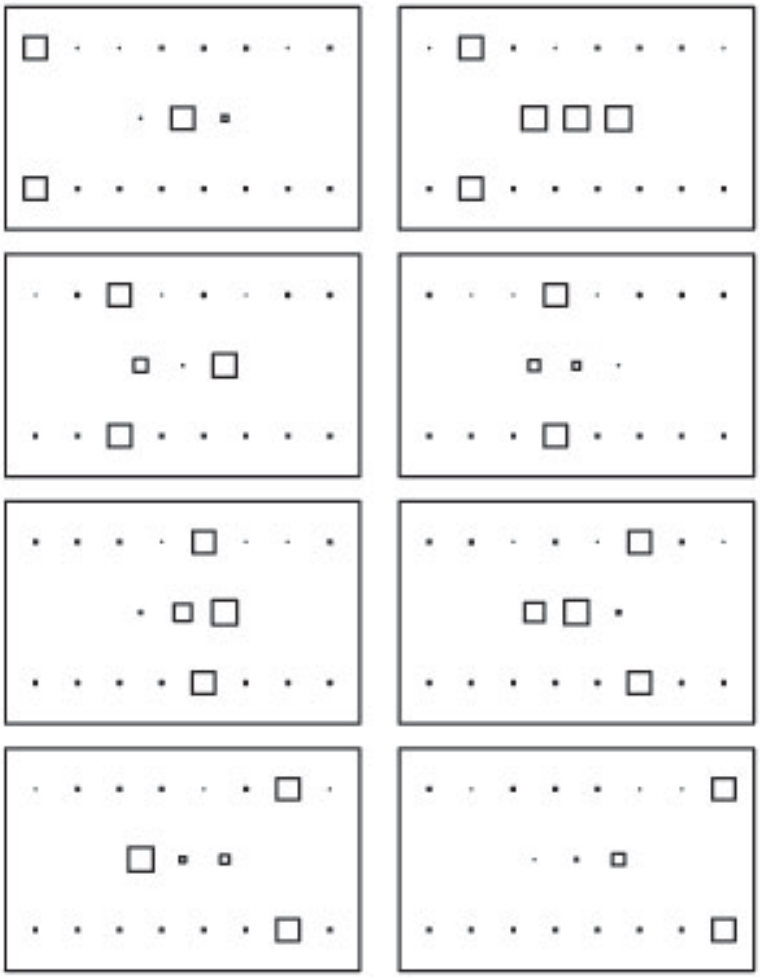

Abb. 3.11: Internal data coding found by the back-propagation algorithm for the "encoder problem." In each picture, one of the eight lower input neurons is activated. The task of the network is the activation of the corresponding output neuron in the upper layer. The input neurons cannot reach the output neurons directly, but only by imposing an "intermediate coding pattern" on the three neurons of the middle layer. From their activities it is evident that the network figurehas "discovered" in essence a binary coding of the eight input patterns.

3.10 Self-Organizing Maps

In all of the previous models, a key role was played by the connections between neurons. In the Hopfield model, every neuron was connected to every

other one, and the only (but, from a biological point of view rather restrictive) constraint was the symmetry $w_{ij} = w_{ji}$. The feed-forward nets of the previous chapter were already organized into a number of layers connected in a fixed order. However, thus far the *location of each neuron within a layer* has played no role for the outgoing or incoming connections. This was a direct consequence of connecting every neuron of one layer to every neuron of the subsequent layer. With *self-organizing maps*, one deals with models in which the ordering of the neurons, *i.e.*, within a layer structure, plays an important role. One is concerned with the question of how the neurons should organize their connectivity in order to optimize the *spatial distribution of their responses within the layer*. Here, the purpose of the optimization is to convert the *similarity of signals* into *proximity* of excited neurons. Neurons with similar tasks can thus communicate over especially short connection paths. This is a very important property for a massively parallel system. A further consequence of such optimization is the formation of *topographic maps* of the input signals, in which the most important similarity relationships among the input signals are converted into *spatial relationships* among the responding neurons. This conversion can be viewed as a process of *abstraction*, suppressing trivial details and mapping the most important properties or features along the dimensions of the map; this is once again relevant to the important problem of the construction of internal data representations. An important special case of such maps is the occurrence of topographically organized projections from a receptor layer to a layer of sensory neurons. This corresponds to the occurrence of simple maps, representing on the neuron layer a (distorted) image of the receptor layer. A theory of the formation of such projections on the basis of synaptic plasticity was suggested by von der Malsburg and Willshaw (Willshaw and von der Malsburg 1976; von der Malsburg and Willshaw 1977). These authors consider a neural layer A, from which output nerve fibers are supposed to grow into a second layer B such that the neighborhood relationships in A are preserved under this " projection." To this end they postulate in layer A the presence of at least two different " marker substances" $i = 1, 2, \ldots$, with concentration gradients such that their local concentrations $c_i(\mathbf{r})$ uniquely determine the position $\mathbf{r}$ everywhere in A. The nerve fibers leaving $\mathbf{r}$ are assumed to transport these marker substances in a mixing ratio characteristic of their origin $\mathbf{r}$ and to give them off at all points in B with which they make synaptic contact. In this way, position-dependent concentrations $c'_i(\mathbf{r}')$ of the marker substances are formed in B as well. The evolution of the strength of each synapse in

B is then determined by two competing contributions. One contribution is a decay term driving the strength of a synapse slowly to zero. The other is a growth term. The better the agreement between the mixing ratios of the marker substances present at the position of the synapse in B and the mixing ratio of the markers given off by the synapse itself, the larger is this growth term. This favors establishment of neighboring synaptic contacts in B for nerve fibers that originate from neighboring positions in A and, as demonstrated by von der Malsburg and Willshaw, leads to the formation of a topographically ordered projection between A and B. By means of computer simulations, von der Malsburg and Willshaw were able to demonstrate a good agreement between the properties of this model and experimentally known findings. In a series of papers, this model was elaborated further in various directions, and new variations were proposed that were capable of explaining additional details, such as the formation of cortical microcolumns (Takeuchi and Amari 1979).

These models often endeavored to be more or less faithful to biological details. A more abstract, and at the same time more general, approach was subsequently suggested by Kohonen (1982a) for the formation of self-organizing sensory maps. We will discuss his model for the formation of such maps in more detail in the subsequent chapters of this book. The Kohonen maps offer a very good point of departure for presenting how a multitude of data processing problems can be solved by means of a small number of powerful basic principles. At the same time, the simplifications and computational savings due to abstraction from biological details not only allow computer simulations of interesting applications, but also are suited for a far-reaching mathematical analysis.