Cubo Matemática Educacional
Vol. 3., N° 1, ENERO 2001

Mathematical Foundations of Neural Network Theory

Burkhard Lenze
lenze@fh-dortmund.de
Fachbereich Informatik, Fachhochschule Dortmund
Postfach 105018, 44047 Dortmund, Germany

ABSTRACT. In the following paper, we present a brief and easily accessible general survey of the theory of neural networks under special emphasis on the rôle of pure and applied mathematics in this interesting field of research.

Keywords: neural networks, applied mathematics
AMS subject classification: 92 B 20

1 Introduction

The paper gives an introduction to the most basic mathematical foundations of neural network theory and is essentially organized as follows. In Section 2 we introduce the general concept of formal neurons which lead – in case that they are combined – to the definition of formal neural networks. In Section 3, we focus our attention on special learning algorithms, which are techniques in order to make a given formal network suitable for concrete applications. We finish this survey paper with some concluding remarks in Section 4.

2 Neurons and Neural Networks

Of course, if we want to deal with a formalization of real neurons and real neural structures we must have an idea of what is basically going on there. We will sketch the most fundamental mechanism in this context quite briefly and refer to [1, 5, 9, 12, 15, 17, 20, 21, 24] for more details. As it is well-known, the basic parts of a real neuron are the cell body (or soma), the dendrites, the synapses, and the axon. Signals (may be chemically or electrically) reach a fixed

neuron through thousands of synapses located at the cell body or at dendrites connected with it. These signals coming from other neurons in the neighborhood are collected and, in case that they are sufficiently strong in total, the neuron itself fires along its single axon. Around the axon again thousands of synapses are located which collect the signal and hand it over to some other neurons, and so on. In the following, we will formalize these basic information processing steps and come to formal neurons and formal neural networks.

2.1 Formal Neurons

We start with a quite abstract definition of formal neurons from a mathematical point of view, which has first been done in a more special setting by Warren McCulloch and Walter Pitts [13] in the early forties of the last century. In the most easiest case of discrete information processing a formal neuron is nothing else but the composition of two quite special functions (comp. also Figure 1):

Definition 1 *In the discrete case, a formal neuron is a function* $\kappa : \mathbb{R}^n \to \mathbb{R}^m$, *which is given by the composition of a so-called transfer function* $T : \mathbb{R} \to \mathbb{R}$ *with a so-called activation function* $A : \mathbb{R}^n \to \mathbb{R}$ *defined as*

$$\begin{aligned} \kappa : \quad & \mathbb{R}^n \to \mathbb{R}^m , \\ & x \mapsto \big(T(A(x)), T(A(x)), \ldots , T(A(x))\big). \end{aligned}$$

Commonly used transfer functions are sigmoidal transfer functions T, which are bounded functions satisfying the limiting conditions

$$\lim_{\xi \to -\infty} T(\xi) = a \quad \text{and} \quad \lim_{\xi \to \infty} T(\xi) = b$$

with $a < b$ (example: $T(\xi) := 1/(1 + \exp(-\xi))$), moreover bell-shaped transfer functions T,which are also bounded but satisfy the limiting conditions

$$\lim_{\xi \to -\infty} T(\xi) = 0 \quad \text{and} \quad \lim_{\xi \to \infty} T(\xi) = 0$$

(example: $T(\xi) := \exp(-\xi^2)$) and, finally, the identity transfer function T, $T(\xi) := \xi$. Popular activation functions are ridge-type activation functions

$$A_{w,\Theta} : \quad x \mapsto \sum_{i=1}^{n} w_i x_i - \Theta ,$$

radial-type activation functions

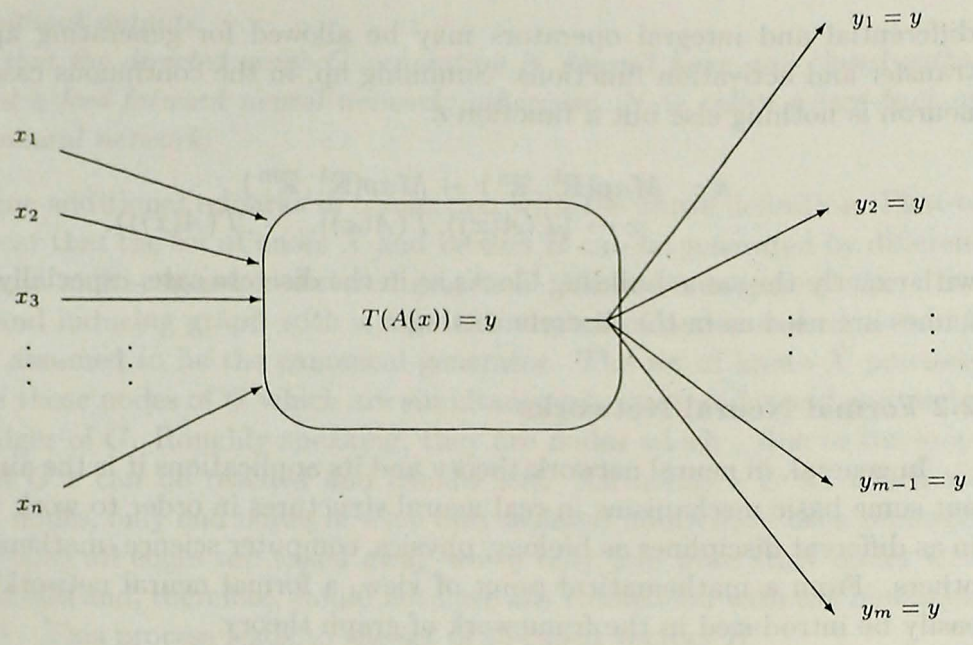

Figura 1: Sketch of a general formal neuron

$$A_{d,\rho}: \quad x \mapsto \rho \sum_{i=1}^{n} (x_i - d_i)^2 \ ,$$

hyperbolic-type activation functions

$$A_{d,\rho}: \quad x \mapsto \rho \prod_{i=1}^{n} (x_i - d_i)$$

or, in general, multi-linear sigma-pi-type activation functions

$$A_{w,\rho}: \quad x \mapsto \rho \sum_{R \subset \{1,\ldots,n\}} w_R \prod_{i \in R} x_i \ .$$

For the parameters appearing in the above definitions the following names are used in the literature: threshold Θ; weight vector or weights w; dilatation or scaling parameter ρ; translation vector or difference weights d.

Finally, let us mention that in the continuous case (in contrast to the discrete case) simply the input vectors $x \in \mathbb{R}^n$ have to be substituted by vector-valued input functions $x : \mathbb{R}^k \to \mathbb{R}^n$ (accordingly, the output consisting of m identical values $y \in \mathbb{R}$ with $y := T(A(x))$ has to be substituted by m identical output functions $y : \mathbb{R}^k \to \mathbb{R}$ with $y(t) := T(A(x(t)))$, $t \in \mathbb{R}^k$) and, moreover,

differential and integral operators may be allowed for generating appropriate transfer and activation functions. Summing up, in the continuous case a formal neuron is nothing else but a function κ,

$$\begin{aligned}\kappa: \quad & Map(\mathbb{R}^k, \mathbb{R}^n) \to Map(\mathbb{R}^k, \mathbb{R}^m)\,, \\ & x \mapsto \big(T(A(x)), T(A(x)), \ldots, T(A(x))\big),\end{aligned}$$

with exactly the same building blocks as in the discrete case; especially, the same names are used as in the discrete setting.

2.2 Formal Neural Networks

In general, in neural network theory and its applications it is the aim to figure out some basic mechanisms in real neural structures in order to work with them in as different disciplines as biology, physics, computer science, mathematics, and others. From a mathematical point of view, a formal neural network can most easily be introduced in the framework of graph theory.

Definition 2 *Let $G := (X, H, \gamma)$ be a loop-free and multi-edges-free directed graph with a finite non-empty set of edges X, a finite set of nodes H with $H \cap X = \emptyset$ and an incidence mapping $\gamma : H \to X \times X$. Moreover, for all edges $v \in X$ let $\delta^+(v)$ denote the out-degree and $\delta^-(v)$ the in-degree of v. If we now define the sets $\tilde{X} \subset X$ and $\tilde{H} \subset H$ as*

$$\begin{aligned}\tilde{X} &:= X \setminus \{v \in X \mid \delta^+(v) \cdot \delta^-(v) = 0\}, \\ \tilde{H} &:= H \setminus \{h \in H \mid \gamma(h) \in (X \setminus \tilde{X}) \times (X \setminus \tilde{X})\},\end{aligned}$$

and assume that $\tilde{X} \neq \emptyset$, then N,

$$N := (X, \tilde{X}, H, \tilde{H}, \gamma),$$

is called (formal) neural network. Moreover, we have the following notations:

- *All elements $v \in \tilde{X}$ are called knots of N.*
- *All elements $h \in \tilde{H}$ are called vectors of N.*
- *All knots $v \in \tilde{X}$, for which we have $w \in X \setminus \tilde{X}$ and $h \in \tilde{H}$ with $\gamma(h) = (w, v)$ are called input knots of N. In this case, the vector $h \in \tilde{H}$ is called input vector of N. A neural network N without input knots and input vectors is called a network without inputs.*
- *All knots $v \in \tilde{X}$, for which we have $w \in X \setminus \tilde{X}$ and $h \in \tilde{H}$ with $\gamma(h) = (v, w)$ are called output knots of N. In this case, the vector $h \in \tilde{H}$ is called output vector of N. A neural network N without output knots and output vectors is called a*

network without outputs.

- *In case that the directed graph G generating N doesn't have any closed paths, N is called a feed-forward neural network; otherwise, N is called a feed-back or recursive neural network.*

Some additional remarks in connection with the above definition: First of all, it is clear that the set of knots $\tilde{X}$ and vectors $\tilde{H}$ can be generated by different loop-free and multi-edges-free directed graphs G. However, in general there is a near at hand inducing graph with a minimal number of nodes and edges which is usually assumed to be the canonical generator. The set of knots $\tilde{X}$ precisely consists of those nodes of G which are simultaneously start nodes and end nodes of some edges of G. Roughly speaking, they are nodes which – due to the loop-freeness of G – can be reached and can be left. All nodes $v \in X$, which are only start nodes, only end nodes or even only isolated nodes have been removed. Going on, also all edges are taken away which only join pure start nodes with pure end nodes and, therefore, would not have any connection with the remaining knots in $\tilde{X}$. This process leads to the set of so-called vectors $\tilde{H}$.

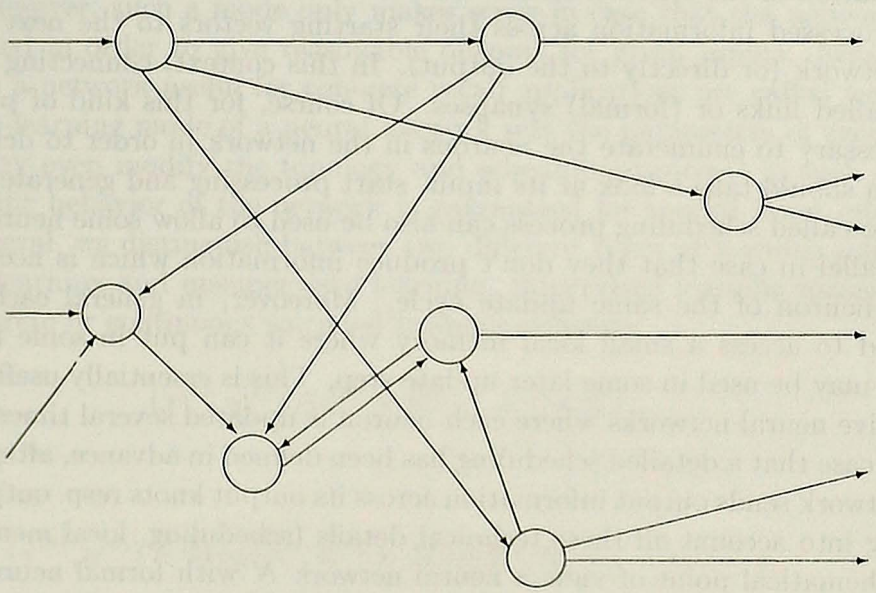

Figura 2: Topology of a general neural network

In the following, we will motivate why some nodes and edges of the given graph G are dropped in order to come to the concept of formal neural networks.

To do this, we take a look at a fixed knot v in N. Since $\delta^+(v) \cdot \delta^-(v) \neq 0$, we may assume that in v we have n vectors ending and m vectors starting. Making the assumption that it is possible to carry some information over the given vectors we can think of incoming information over the n vectors ending on v, some information processing in v and, finally, sending the processed information over the m vectors starting on v. Obviously, this is precisely identical with the general definition of a formal neuron given in Subsection 2.1. Therefore, it should be no surprise that the step from the pure topology of a neural network N to a dynamic information processing tool consists in placing formal neurons in each knot v. At this point, it also becomes clear why we assumed the directed graph G to be free of loops and multi-edges: free of loops because a formal neuron only has free inputs and outputs; free of multi-edges because a formal neuron only produces m identical output signals and it therefore doesn't make any sense to connect two formal neurons with more than one connection vector in each direction. Summing up, the complete dynamic behavior of such a neural network can be described as follows: At the beginning, only the input vectors of N hand over some information to the input knots of N (from now on also called input neurons), where the information may be discrete or continuous. The input neurons process the given information in the sense of formal neurons as defined in Subsection 2.1 and feed the processed information across their starting vectors to the next neurons in the network (or directly to the output). In this context, connecting vectors are also called links or (formal) synapses. Of course, for this kind of processing it is necessary to enumerate the neurons in the network in order to define when a neuron should take a look at its input, start processing and generate its output. This so-called scheduling process can also be used to allow some neurons to work in parallel in case that they don't produce information which is needed by any other neuron of the same update cycle. Moreover, in general each neuron is allowed to access a small local memory where it can put in some information which may be used in some later update step. This is essentially useful in case of recursive neural networks where each neuron is updated several times. Summing up, in case that a detailed scheduling has been defined in advance, after some time the network sends output information across its output knots resp. output vectors. Taking into account all these technical details (scheduling, local memory), from a mathematical point of view a neural network N with formal neurons in each knot is nothing else but a special realization of a function $\aleph$,

$$\aleph : \mathbb{R}^n \to \mathbb{R}^m,$$

in the discrete case, respectively,

$$\aleph : Map(\mathbb{R}^k, \mathbb{R}^n) \to Map(\mathbb{R}^k, \mathbb{R}^m),$$

in the continuous case, which again is simply called the neural network. In view of the above definition we assume that the network N has precisely n input vectors and m output vectors and that the network function $\aleph$ depends on the parameters of each neuron and the scheduling process in a quite complicated form. Let us emphasize that in the literature often no difference is made between the topology N of a neural network and its final applicable realization in the sense of the mapping $\aleph$. Both are called neural network or shortly network and the context has to be taken into account in order to figure out in which sense the term is used. Moreover, in many cases it is quite difficult to give a compact closed form of the mapping $\aleph$; in these cases, the output behavior of the network is calculated step by step following the given scheduling details. Finally, let us note that in contrast to the deterministic dynamics sketched above also so-called probabilistic neural networks are discussed in the literature. For networks of this kind, the update rules described above are modified or even completely substituted by probabilistic components, and, of course, the whole analysis of such a network has to be done in a probabilistic framework.

Up to now, we only have sketched the so-called recall mode of a neural network. However, such a mode only makes sense in case that the network has been designed in order to give reasonable outputs for given inputs. Strategies which make a network useful for concrete recall applications are called learning modes. The learning mode of a neural network sets the parameters of each neuron (and may even modify the topology and scheduling process) in such a way that a specific behavior of the network is guaranteed for special input information. In general, we distinguish between two different types of learning schemes: supervised learning and unsupervised learning. Supervised learning means that a set of discrete or continuous so-called training elements

$$(x^{(s)}, y^{(s)}) \in \mathbb{R}^n \times \mathbb{R}^m \ , \quad 1 \leq s \leq t \ ,$$

respectively,

$$(x^{(s)}, y^{(s)}) \in Map(\mathbb{R}^k, \mathbb{R}^n) \times Map(\mathbb{R}^k, \mathbb{R}^m) \ , \quad 1 \leq s \leq t \ ,$$

with correct input-output behavior are known and may be used to configure the network. In detail, in case of supervised learning the parameters determinating the network function $\aleph$ should be set in such a way that the errors

$$(y^{(s)} - \aleph(x^{(s)})) \ , \quad 1 \leq s \leq t \ ,$$

become as small as possible with respect to some given norm. Concrete realizations of supervised learning are – for example – the Hebb learning rule, the delta learning rule, the perceptron learning rule, the backpropagation learning rule, the hyperbolic learning rule, and, finally, the so-called learning vector quantization. All these rules are discussed in more detail in Subsections 3.1–3.6.

In contrast to supervised learning a so-called unsupervised learning scheme only has access to some discrete or continuous input training elements

$$x^{(s)} \in \mathbb{R}^n \ , \quad 1 \leq s \leq t \ ,$$

respectively,

$$x^{(s)} \in Map(\mathbb{R}^k, \mathbb{R}^n) \ , \quad 1 \leq s \leq t \ ,$$

without any information in view of proper corresponding outputs. Therefore, a reasonable strategy to adapt the network parameters is to make the network produce similar outputs for similar inputs (whatever that means in detail). Roughly speaking, after unsupervised learning the network should act on inputs like

$$x \approx \tilde{x} \implies \aleph(x) \approx \aleph(\tilde{x})$$

or even

$$x \approx \tilde{x} \implies \aleph(x) = \aleph(\tilde{x}) \ .$$

Popular examples of unsupervised learning schemes are the Kohonen learning scheme, adaptive resonance theory and the Oja learning scheme. These learning schemes are discussed in some detail in Subsections 3.7–3.9.

Let us finally mention that for both, supervised and unsupervised learning, again also probabilistic variants are discussed in the literature and should be treated from a probabilistic point of view.

3 Special Learning Algorithms

In this section, we take a brief look at some very popular and intensively studied learning rules in neural network theory. We start with some examples of supervised learning (Subsections 3.1–3.6) and end up with some unsupervised learning schemes (Subsections 3.7–3.9). Of course, we don't claim any completeness and refer to the literature for further learning schemes and extensions.

3.1 Supervised Learning: Hebb Learning Rule

The Hebb learning rule is based on a neuro-biological proposition for the process of learning formulated by Donald Hebb [4] in 1949, which now is the basis of a large number of similar learning schemes in neural network theory. Roughly speaking, Hebb's idea of how learning takes place may be formulated as follows: *In case that two neurons which are coupled by a synaptic link are often active simultaneously, then the pre-synaptic neuron will get a stronger and stronger influence on the behavior of the post-synaptic neuron. In other words, during learning synaptic links of simultaneously active neurons are getting stronger.*

In the following, we will give a formalization and a concrete implementation of the general idea of Hebb in connection with discrete two layer feed-forward neural networks with ridge-type activation and identical transfer function in the output neurons (comp. Figure 3 in view of the general topology of such a network): In case that we present t training pairs $(x^{(s)}, y^{(s)}) \in \mathbb{R}^n \times \mathbb{R}^m$, $1 \leq s \leq t$, we set $\Theta_j := 0$, $1 \leq j \leq m$, and define

$$w_{ij} := \sum_{s=1}^{t} x_i^{(s)} y_j^{(s)}$$

for $1 \leq i \leq n$ and $1 \leq j \leq m$. For pairwise orthonormal training inputs $x^{(s)}$, $1 \leq s \leq t$, the neural network now performs perfectly on the training set, which means

$$\sum_{i=1}^{n} w_{ij} x_i^{(s)} = y_j^{(s)}$$

for $1 \leq j \leq m$ and $1 \leq s \leq t$. A neural network of this kind is called linear associator (trained by Hebb) and it reflects Hebb's original idea of learning in the following sense: Since in recall mode for given $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ the network calculates the corresponding output vector $y = (y_1, \ldots, y_m) \in \mathbb{R}^m$ via

$$y_j = \sum_{i=1}^{n} w_{ij} x_i \,, \quad 1 \leq j \leq m \,,$$

the weight w_{ij} determines how strong x_i can affect the output y_j. In case that for the s-th training pair $(x^{(s)}, y^{(s)})$ the so-called simultaneous activity $x_i^{(s)} y_j^{(s)}$ in the

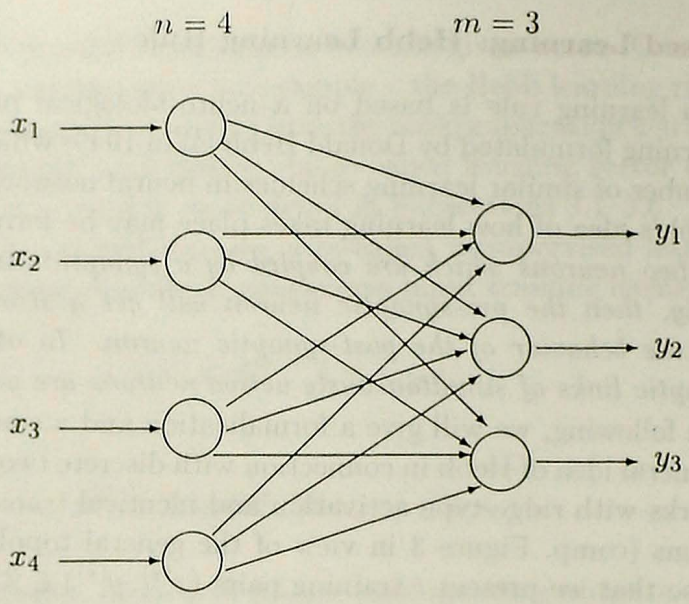

Figura 3: Topology of a two layer feed-forward neural network

sense of a product is large, then the synaptic coupling w_{ij} should increase appropriately; in case of being small the weight should be modified only moderately. In this sense, the formalized Hebb learning rule basically reflects the original idea of Hebb.

3.2 Supervised Learning: Delta Learning Rule

The delta learning rule (also called Widrow-Hoff learning rule) has been introduced by Bernard Widrow and Marcian Hoff [23] in the late fifties of the last century and may be seen as a special case of backpropagation learning for two layer feed-forward neural networks with identical transfer function. In the following, we will sketch the general idea of delta learning in context of discrete two layer feed-forward neural networks with ridge-type activation and identical transfer function in the output neurons:

In case that we present t training pairs $(x^{(s)}, y^{(s)}) \in \mathbb{R}^n \times \mathbb{R}^m$, $1 \leq s \leq t$, the weights $w_{ij} \in \mathbb{R}$, $1 \leq i \leq n$, $1 \leq j \leq m$, and the thresholds $\Theta_j \in \mathbb{R}$, $1 \leq j \leq m$, should be fixed in such a way that for all $j \in \{1, \ldots, m\}$ and for all $s \in \{1, \ldots, t\}$ the squared errors

$$\left(y_j^{(s)} - (\sum_{i=1}^{n} w_{ij} x_i^{(s)} - \Theta_j) \right)^2$$

become as small as possible. Defining t partial differentiable error functions

$$F^{(s)} : \mathbb{R}^{nm} \times \mathbb{R}^m \longrightarrow \mathbb{R} , \quad 1 \leq s \leq t ,$$

by

$$F^{(s)}(.., w_{ij}, .., \Theta_j, ..) := \sum_{j=1}^{m} \left(y_j^{(s)} - (\sum_{i=1}^{n} w_{ij} x_i^{(s)} - \Theta_j) \right)^2 ,$$

a gradient search algorithm for finding a local (or, fortunately, even a global) minimum leads to the following update steps, where $\lambda > 0$ is still a free learning parameter to be set appropriately:

1. weights w_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$:

$$w_{ij}^{(new)} := w_{ij} - \lambda F_{w_{ij}}^{(s)}(.., w_{ij}, .., \Theta_j, ..) \quad \text{resp.}$$

$$w_{ij}^{(new)} := w_{ij} + 2\lambda \left(y_j^{(s)} - (\sum_{k=1}^{n} w_{kj} x_k^{(s)} - \Theta_j) \right) x_i^{(s)} .$$

2. thresholds Θ_j, $1 \leq j \leq m$:

$$\Theta_j^{(new)} := \Theta_j - \lambda F_{\Theta_j}^{(s)}(.., w_{ij}, .., \Theta_j, ..) \quad \text{resp.}$$

$$\Theta_j^{(new)} := \Theta_j - 2\lambda \left(y_j^{(s)} - (\sum_{k=1}^{n} w_{kj} x_k^{(s)} - \Theta_j) \right) .$$

Of course, $F_{w_{ij}}^{(s)}$ and $F_{\Theta_j}^{(s)}$ denote the partial derivatives of $F^{(s)}$ with respect to w_{ij} and Θ_j. Applying the above strategy for all error functions $F^{(s)}$, $1 \leq s \leq t$, and iterating this procedure is called delta learning rule or Widrow-Hoff learning rule. In more detail, the above strategy is called on-line delta learning rule since each error function $F^{(s)}$, $1 \leq s \leq t$, is tried to be minimized, separately. In contrast, we could also sum over all error functions to come to a kind of total error function, $F := \sum_{s=1}^{t} F^{(s)}$, and try to minimize this function by means of a gradient descent method. In the literature, this approach is called off-line or batch-mode delta learning rule. The on-line mode has the advantage that no weight or threshold corrections have to be stored and that a non-deterministic presentation of the training pairs is possible (stochastic learning). Its disadvantage is that after one complete learning cycle with all t training pairs it can not be guaranteed that the total error F did really decrease (even for small λ); in each step, $F^{(s)}$ in general becomes smaller but the other errors $F^{(r)}$, $r \neq s$, may increase. Although, this is a serious problem, on-line learning has proved to be quite successful and is usually preferred to avoid the heavy computation and storage costs of off-line learning.

3.3 Supervised Learning: Perceptron Learning Rule

The perceptron learning rule has been introduced by Frank Rosenblatt [18, 19] in the late fifties of the last century to improve some shortcomings of the Hebb learning rule (see also [14] for a serious critical analysis of perceptron-type strategies). Roughly speaking, the perceptron learning rule may be interpreted as a kind of delta learning rule for non-differentiable transfer functions. In the following, we sketch the general idea of the perceptron learning rule in connection with discrete two layer feed-forward neural networks with ridge-type activation and non-differentiable sigmoidal transfer functions in the output neurons:

Let $T : \mathbb{R} \to \{0,1\}$ with $T(\xi) := 0$ for $\xi < 0$ and $T(\xi) := 1$ for $\xi \geq 0$ be the transfer function for the output neurons. In case that t training pairs $(x^{(s)}, y^{(s)}) \in \mathbb{R}^n \times \{0,1\}^m$, $1 \leq s \leq t$, are presented to the network, the weights $w_{ij} \in \mathbb{R}$, $1 \leq i \leq n$, $1 \leq j \leq m$, and the thresholds $\Theta_j \in \mathbb{R}$, $1 \leq j \leq m$, should be fixed in such a way that for all $j \in \{1, \ldots, m\}$ and for all $s \in \{1, \ldots, t\}$ the squared errors

$$\left(y_j^{(s)} - T(\sum_{i=1}^{n} w_{ij} x_i^{(s)} - \Theta_j) \right)^2$$

become as small as possible. To reach this goal, the perceptron learning rule is defined as follows, where $\lambda > 0$ is still a free learning parameter to be fixed appropriately (sometimes set to 1 in the literature):

1. weights w_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$:

$$w_{ij}^{(new)} := w_{ij} + \lambda \left(y_j^{(s)} - T(\sum_{k=1}^{n} w_{kj} x_k^{(s)} - \Theta_j) \right) x_i^{(s)}.$$

2. thresholds Θ_j, $1 \leq j \leq m$:

$$\Theta_j^{(new)} := \Theta_j - \lambda \left(y_j^{(s)} - T(\sum_{k=1}^{n} w_{kj} x_k^{(s)} - \Theta_j) \right).$$

Applying the above procedure for all $s \in \{1, \ldots, t\}$ and iterating as usual is called perceptron learning rule. It can be shown that after a finite number of iterations the above algorithm yields a neural network which performs perfectly on the training set in case that for all $j \in \{1, \ldots, m\}$ the two subsets

$$A_j := \{x^{(s)} \mid 1 \leq s \leq t \ \wedge \ y_j^{(s)} = 0\} \ ,$$
$$B_j := \{x^{(s)} \mid 1 \leq s \leq t \ \wedge \ y_j^{(s)} = 1\} \ ,$$

of $\mathbb{R}^n$ can be strongly separated by an affine hyperplane (convergence theorem of perceptron learning). Of course, for practical purpose it is quite difficult to figure out in advance whether a set of training pairs is strongly linear separable or not. Therefore, this result is mainly of theoretical interest.

3.4 Supervised Learning: Backpropagation Learning Rule

Backpropagation learning is a learning rule based on the gradient descent method and has first been used in connection with neural networks by Paul Werbos [22] in 1974. In the following, we sketch the general idea of backpropagation learning in context of discrete three layer feed-forward neural networks with ridge type activation in the hidden neurons (comp. Figure 4 in view of the general topology of such a network):

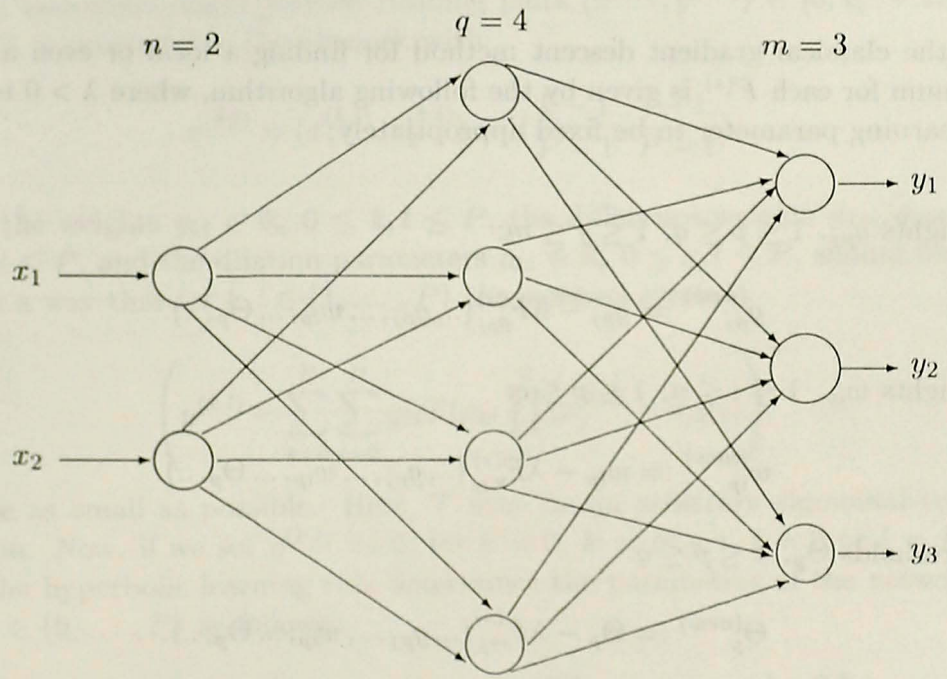

Figura 4: Topology of a three layer feed-forward neural network

In case that a set of t training pairs $(x^{(s)}, y^{(s)}) \in \mathbb{R}^n \times \mathbb{R}^m$, $1 \leq s \leq t$, should be represented by the network, the weights $g_{pj} \in \mathbb{R}$, $1 \leq p \leq q$, $1 \leq j \leq m$, and $w_{ip} \in \mathbb{R}$, $1 \leq i \leq n$, $1 \leq p \leq q$, and, moreover, the thresholds $\Theta_p \in \mathbb{R}$, $1 \leq p \leq q$, should be fixed in such a way that for all $j \in \{1, \ldots, m\}$ and for all $s \in \{1, \ldots, t\}$ the squared errors

$$\left(y_j^{(s)} - \sum_{p=1}^{q} g_{pj} T(\sum_{i=1}^{n} w_{ip} x_i^{(s)} - \Theta_p) \right)^2$$

become as small as possible. In case that the transfer function T is continuously differentiable and t partial differentiable error functions

$$F^{(s)} : \mathbb{R}^{qm} \times \mathbb{R}^{nq} \times \mathbb{R}^q \longrightarrow \mathbb{R}\,, \quad 1 \leq s \leq t\,,$$

are defined as

$$F^{(s)}(.., g_{pj}, .., w_{ip}, .., \Theta_p, ..) := \sum_{j=1}^{m} \left(y_j^{(s)} - \sum_{p=1}^{q} g_{pj} T(\sum_{i=1}^{n} w_{ip} x_i^{(s)} - \Theta_p) \right)^2 ,$$

then the classical gradient descent method for finding a local or even a global minimum for each $F^{(s)}$ is given by the following algorithm, where $\lambda > 0$ is still a free learning parameter to be fixed appropriately:

1. weights g_{pj}, $1 \leq p \leq q$, $1 \lesssim j \leq m$:

$$g_{pj}^{(new)} := g_{pj} - \lambda F_{g_{pj}}^{(s)}(.., g_{pj}, .., w_{ip}, .., \Theta_p, ..).$$

2. weights w_{ip}, $1 \leq i \leq n$, $1 \lesssim p \leq q$:

$$w_{ip}^{(new)} := w_{ip} - \lambda F_{w_{ip}}^{(s)}(.., g_{pj}, .., w_{ip}, .., \Theta_p, ..).$$

3. thresholds Θ_p, $1 \leq p \leq q$:

$$\Theta_p^{(new)} := \Theta_p - \lambda F_{\Theta_p}^{(s)}(.., g_{pj}, .., w_{ip}, .., \Theta_p, ..).$$

Of course, the abbreviations $F_{g_{pj}}^{(s)}$, $F_{w_{ip}}^{(s)}$ and $F_{\Theta_p}^{(s)}$ denote the partial derivatives of $F^{(s)}$ with respect to g_{pj}, w_{ip} and Θ_p. Applying the above steps to all error

functions $F^{(s)}$, $1 \leq s \leq t$, and iterating in cycles is called backpropagation learning or backpropagation algorithm (the name is motivated by the fact that the errors $F^{(s)}$, $1 \leq s \leq t$, are propagated back through the network and are used in order to update the network parameters appropriately). The algorithm and its various modifications is one of the most popular and successful training tools in neural network design. Of course, in view of on-line and off-line implementations the same remarks hold as in case of the delta learning rule.

3.5 Supervised Learning: Hyperbolic Learning Rule

Hyperbolic learning applies to three layer feed-forward neural networks which possess hidden neurons with hyperbolic activation functions and can be used in case that the given training set comes from a multi-dimensional regular grid. After training the network is able to interpolate or at least approximate the given information on the respective grid. The learning scheme has been introduced about 1990 by the author (cf. [11]; also [12], for more details). In the following, we sketch the general idea in the most easiest two-dimensional case:

Let a three layer feed-forward neural network with hyperbolic transfer functions in the hidden neurons be given and assume that a set of P^2 ($P \in \mathbb{N}$, $P \geq 2$) two-dimensional gridded training pairs $(x^{(k,l)}, y^{(k,l)}) \in [0,1]^2 \times \mathbb{R}$, $1 \leq k,l \leq P$, is presented to the network with

$$x^{(k,l)} = (x_1^{(k,l)}, x_2^{(k,l)}) := (\frac{k-1}{P-1}, \frac{l-1}{P-1}).$$

Then, the weights $g_{kl} \in \mathbb{R}$, $0 \leq k,l \leq P$, the difference weights $d_{1kl}, d_{2kl} \in \mathbb{R}$, $0 \leq k,l \leq P$, and the dilation parameters $\rho_{kl} \in \mathbb{R}$, $0 \leq k,l \leq P$, should be fixed in such a way that for $\tilde{k},\tilde{l} \in \{1,\ldots,P\}$ the squared errors

$$\left(y^{(\tilde{k},\tilde{l})} - \sum_{k=0}^{P} \sum_{l=0}^{P} g_{kl} T(\rho_{kl} \prod_{j=1}^{2} (x_j^{(\tilde{k},\tilde{l})} - d_{jkl})) \right)^2$$

become as small as possible. Here, T may be an arbitrary sigmoidal transfer function. Now, if we set $y^{(k,l)} := 0$, for $k = 0$, $k = P+1$, $l = 0$ or $l = P+1$, then the hyperbolic learning rule determines the parameters of the network for all $k,l \in \{0,\ldots,P\}$ as follows:

$$\begin{aligned} \rho_{kl} &:= (P-1)^2\,, \quad d_{1kl} := \frac{k-0.5}{P-1}\,, \quad d_{2kl} := \frac{l-0.5}{P-1}\,, \\ g_{kl} &:= \frac{1}{2}(y^{(k,l)} - y^{(k+1,l)} - y^{(k,l+1)} + y^{(k+1,l+1)})\,. \end{aligned}$$

Obviously, this learning rule is not iterative, which means that it is a so-called one-shot learning scheme (real-time-processing). Moreover, in case of a sigmoidal transfer function $T : \mathbb{R} \to \mathbb{R}$ satisfying $T(\xi) = 0$ for $\xi \leq -1/4$ and $T(\xi) = 1$ for $\xi \geq 1/4$ the hyperbolic learning scheme fixes the network parameters in such a way that the errors on the training set vanish (interpolation). For general sigmoidal transfer functions satisfying only $\lim_{\xi \to -\infty} T(\xi) = 0$ and $\lim_{\xi \to \infty} T(\xi) = 1$ we cannot guarantee interpolation any longer but only get an approximation of the given data set by the induced neural network.

3.6 Supervised Learning: Learning Vector Quantization

Learning vector quantization (for short: LVQ) has been introduced in the literature by different authors about the late seventies of the last century (comp. [17, 21, 24] for more detailed historical information) and is now used in a huge number of quite different variants. By nature, it is intimately connected with Kohonen learning (cf. Subsection 3.7) where the essential difference is that Kohonen learning is an unsupervised learning scheme while LVQ is supervised. In the following, we explain the basic idea of LVQ by considering a simple example (discrete case):

Let us assume that we have given a finite set of t vectors $x^{(s)} \in \mathbb{R}^n$, $1 \leq s \leq t$, a finite set of j cluster vectors $w^{(i)} \in \mathbb{R}^n$, $1 \leq i \leq j$, and, finally, a classification function $f : \{1, \ldots, t\} \to \{1, \ldots, j\}$, which maps each vector $x^{(s)}$ identified by its index s on a cluster vector $w^{(f(s))}$ identified by its index $f(s)$. Here, in general j is significantly smaller than t and, obviously, f is used to identify for each vector $x^{(s)}$ a class which it belongs to, where the class itself is represented by a cluster vector. During learning the cluster vectors are now modified depending on the given vectors to be classified. In the simplest case, we have the following update steps, where $\lambda \in (0, 1)$ is still a free learning parameter to be set appropriately: In learning step s ($1 \leq s \leq t$) used to update $w^{(f(s))}$ a measure of the distances of $x^{(s)}$ from all cluster vectors $w^{(i)}$, $1 \leq i \leq j$, has to be calculated (for example, by means of angles, Euclidean distances, or something else). In case that $w^{(f(s))}$ belongs to the set of cluster vectors with minimal distance from $x^{(s)}$ (consistent classification), substitute $w^{(f(s))}$ by

$$w^{(f(s))} + \lambda(x^{(s)} - w^{(f(s))}) .$$

All other cluster vectors are not modified. In case that $w^{(f(s))}$ does not belong to the set of cluster vectors with minimal distance from $x^{(s)}$ (inconsistent classification), substitute $w^{(f(s))}$ by

$$w^{(f(s))} - \lambda(x^{(s)} - w^{(f(s))}) .$$

Again, all other cluster vectors are not modified. Iterate this procedure several times, make λ smaller and smaller step by step, and terminate the algorithm, for example, in case that the maximal distance between each vector to be classified and its respective cluster vector is less than some given limit or a predefined number of iterations have been performed.

The LVQ-prototype sketched above has been improved and modified in several ways during the last decades. We only mention canceling of old or generation of new cluster vectors or a cluster-dependent non-global adaption of λ. Finally, let us note that it is obviously no problem to implement the above algorithm in a neural network context. The main problem, which has to be solved, is to calculate the given distance measure by means of appropriate formal neurons. Of course, for the measurement of angles ridge-type activation functions together with sigmoidal transfer functions are proper choices (remember that angles between vectors are essentially determined by the scalar product of their normalizations) while for dealing with Euclidean distances radial-type activation functions together with bell-shaped transfer functions should be used. We omit the technical details of a complete definition of networks of such type and leave their concrete design to the reader.

3.7 Unsupervised Learning: Kohonen Learning Rule

Kohonen learning has been intensively studied by Teuvo Kohonen [9] in the late seventies of the last century, but had been present in literature even earlier in quite similar formulations. As already mentioned, Kohonen learning is intimately connected with learning vector quantization (see the remarks in Subsection 3.6). Moreover, there is a connection to adaptive resonance theory in the sense that the latter is an extension of Kohonen learning in order to solve the so-called stability-plasticity dilemma (cf. Subsection 3.8 for details). In the following, we sketch the basic idea of Kohonen learning by considering an easy discrete classification problem:

A finite set of t vectors $x^{(s)} \in \mathbb{R}^n$, $1 \leq s \leq t$, should be classified, which means that they should be put into one of j so-called cluster sets, where j is usually significantly smaller than t. For this purpose, first of all a set of cluster vectors $w^{(i)} \in \mathbb{R}^n$, $1 \leq i \leq j$, is randomly generated with each cluster vector representing one specific cluster. Now, the modification of the cluster vectors depending on the vectors to be classified is done in the following way, where we sketch the most easiest strategy with $\lambda \in (0, 1)$ a free learning parameter to be

set appropriately: In step s $(1 \leq s \leq t)$ to classify $x^{(s)}$ a measure for the distance of $x^{(s)}$ from all cluster vectors $w^{(i)}$, $1 \leq i \leq j$, is calculated, for example, by means of an angle or by means of Euclidean distance. Put $x^{(s)}$ into the cluster which is represented by a cluster vector with minimal distance from $x^{(s)}$. In case that there are several cluster vectors with minimal distance to $x^{(s)}$, choose the cluster with minimal index. In case that the cluster vector representing the proper cluster in the above sense has index i, substitute it by

$$w^{(i)} + \lambda(x^{(s)} - w^{(i)}) \,,$$

i.e., by a convex combination of the old cluster vector and the new vector assigned to the cluster; all other cluster vectors are not modified. Iterate this procedure several times, make λ smaller and smaller step by step, and terminate the algorithm, for example, in case that the maximal distance between each vector to be classified and its respective cluster vector is less than some given limit or a predefined number of iterations have been performed.

The above strategy is only the most simplest prototype of Kohonen learning and has been modified during the time in several ways. We only mention the idea to modify more cluster vectors in each step depending on a so-called neighboring function and the idea of not always updating a cluster vector in case that it quite often realizes minimal distance (Kohonen learning with conscience).

3.8 Unsupervised Learning: Adaptive Resonance Theory

Adaptive resonance techniques have first been studied – among others – by Stephen Grossberg and Gail Carpenter [3, 2] in the late seventies of the last century. In this theory, which is shortly called ART, the neural networks should be constructed in such a way that they are able to classify given input information by its own. ART-strategies are intimately connected with Kohonen learning with the most essential difference that Kohonen learning is set up using a fixed number of clusters chosen in advance while ART is able to adapt the number of clusters according to the given classification problem. This special feature of ART makes a neural network solve the so-called stability-plasticity problem: The network puts new vectors to be classified into proper clusters in case that they exist (stability feature) and it generates new clusters in case that the given ones are not sufficiently suited (plasticity feature). In detail, we sketch the ART-idea in context of a most easy application (discrete variant):

A finite set of t vectors $x^{(s)} \in \mathbb{R}^n$, $1 \leq s \leq t$, should be classified in the sense already discussed in Subsections 3.6 and 3.7. Therefore, step by step so-called cluster vectors $w^{(i)} \in \mathbb{R}^n$, $i \in \mathbb{R}$, are generated in some clever way. For example, in case of given $\epsilon > 0$ and $\lambda \in (0, 1)$ this may be done as follows: In the first step $(s = 1)$ define $w^{(1)} := x^{(1)}$ and $j := 1$. In step s $(1 < s \leq t)$ to classify

$x^{(s)}$ a measure of the distance of $x^{(s)}$ to all already defined cluster vectors $w^{(i)}$, $1 \leq i \leq j$, has to be calculated (for example, via angle or Euclidean distance). In case that the smallest distance calculated in this way is smaller than ϵ then $x^{(s)}$ is put into the respective cluster. In case that there a several cluster vectors with minimal distance take the one with minimal index. In case that the index of the cluster vector fixed in this way is i, substitute it by

$$w^{(i)} + \lambda(x^{(s)} - w^{(i)}) ,$$

i.e., by a convex combination of the old cluster vector and the new vector assigned to this cluster; all other cluster vectors are not modified (stability effect). On the other hand, in case that the distance of the vector to be classified is larger or equal than ϵ for all cluster vectors, then add a new cluster vector $w^{(j+1)} := x^{(s)}$ to the set of cluster vectors and increase the index j by 1 (plasticity effect). Iterate this procedure several times, make ϵ and/or λ smaller and smaller step by step, and terminate the algorithm, for example, in case that the maximal distance between each vector to be classified and its respective cluster vector is less than some given limit or a predefined number of iterations have been performed.

Of course, during the years several modifications and improvement have been introduced similar to those for LVQ and Kohonen learning. We omit the details.

3.9 Unsupervised Learning: Oja learning Rule

This learning rule has been introduced by Erkki Oja [16] in the early eighties of the last century. Roughly speaking, Oja learning is a special case of Kohonen learning where only one cluster vector, the so-called principal component vector, has to be calculated and, moreover, a gradient descent method is used instead of simple convex combinations. In the following, the general idea of Oja learning is sketched in context of a simple discrete classification problem:

For a finite set of t vectors $x^{(s)} \in \mathbb{R}^n \setminus \{0\}$, $1 \leq s \leq t$, a vector $w \in \mathbb{R}^n \setminus \{0\}$ should be calculated with minimal averaged distance from all given vectors. Here, the distance should be given by the non-orientated angle between the straight lines induced by w and $x^{(s)}$ going through the origin. This means, that the vector $w \in \mathbb{R}^n \setminus \{0\}$ has to be chosen in such a way that for all $s \in \{1, \ldots, t\}$ the ratios

$$\frac{(w \cdot x^{(s)})^2}{(w \cdot w)(x^{(s)} \cdot x^{(s)})} = \frac{\left(\sum_{i=1}^{n} w_i x_i^{(s)}\right)^2}{\left(\sum_{i=1}^{n} (w_i)^2\right)\left(\sum_{i=1}^{n} (x_i^{(s)})^2\right)}$$

become as large as possible. Defining t partial differentiable functions

$$Q^{(s)} : \mathbb{R}^n \setminus \{0\} \longrightarrow \mathbb{R}\,, \quad 1 \leq s \leq t\,,$$

via

$$Q^{(s)}(w) \;:=\; \frac{(w \cdot x^{(s)})^2}{(w \cdot w)(x^{(s)} \cdot x^{(s)})},$$

a gradient search for a (local) maximum of the functions $Q^{(s)}$ leads to the following iteration, where $\lambda > 0$ is a so-called learning parameter which can be set appropriately:

$$w^{(new)} := w + \lambda \,\mathrm{grad}\, Q^{(s)}(w)$$

with $\mathrm{grad}\, Q^{(s)}(w)$ given by

$$\frac{2(w \cdot x^{(s)})}{(w \cdot w)(x^{(s)} \cdot x^{(s)})}\Big(x^{(s)} - (w \cdot x^{(s)})(w \cdot w)^{-1} w\Big).$$

Applying the above update steps to all functions $Q^{(s)}$, $1 \leq s \leq t$, and doing this iteratively is called Oja learning rule. Moreover, in case that w is normalized to Euclidean length 1 after each iteration step, which implies $(w \cdot w) = 1$, and in case that the factor $2(x^{(s)} \cdot x^{(s)})^{-1}$ is ignored and thought to be covered by means of the learning parameter λ, then a simple form of Oja learning is obtained which is often discussed in the literature. Finally, since Oja learning again is based on a gradient descent method the remarks concerning on-line and off-line implementations hold in the same sense as already discussed in connection with the delta rule or backpropagation learning.

4 Concluding Remarks

In this survey paper, a brief overview has been given concerning the mathematical foundations and the most classical strategies in the theory of neural networks. Of course, being limited by about twenty pages does not allow to cover all relevant and interesting topics. For example, Hopfield and Kosko networks [6, 7, 10] (see also [8] for a more detailed treatment of networks of this kind) as most popular realizations of recursive neural networks could not be discussed. Moreover, all recent studies in view of radial basis function networks and the most interesting, but quite technical implementations of continuous information

processing neural networks could not be included. In view of this and other missing topics the reader may take a look at one of the special and more complete text and research books included in the reference list given below.

Referencias

[1] **P. C. Bressloff and D. J. Weir**, *Neural Networks*, GEC Journal of Research 8, 151–169, 1991.

[2] **G. A. Carpenter and S. Grossberg**, *ART 2: Self-organization of stable category recognition codes for analog input patterns*, Applied Optics 26, 4919–4930, 1987.

[3] **S. Grossberg**, *Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors*, Biological Cybernetics 23, 121–134, 1976.

[4] **D. O. Hebb**, *The Organization of Behaviour*, John Wiley & Sons, New York, 1949.

[5] **R. Hecht-Nielsen**, *Neurocomputing*, Addison-Wesley, Reading, Massachusetts, 1990.

[6] **J. J. Hopfield**, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences (USA) 79, 2554–2558, 1982.

[7] **J. J. Hopfield and D. W. Tank**, *Neural computations of decisions in optimization problems*, Biological Cybernetics 52, 141–152, 1985.

[8] **Y. Kamp and M. Hasler**, *Recursive Neural Networks for Associative Memory*, John Wiley & Sons, Chichester, 1990.

[9] **T. Kohonen**, *Self-Organization and Associative Memory*, Springer Verlag, Berlin, 1984.

[10] **B. Kosko**, *Bidirectional associative memories, IEEE Transactions on Systems*, Man, and Cybernetics 18, 49–60, 1988.

[11] **B. Lenze**, *How to make sigma-pi neural networks perform perfectly on regular training sets*, Neural Networks 7, 1285–1293, 1994.

[12] **B. Lenze**, *Einführung in die Mathematik neuronaler Netze*, Logos Verlag, Berlin, 1997.

[13] **W. S. McCulloch and W. Pitts**, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 115–133, 1943.

[14] **M. Minsky and S. Papert**, *Perceptrons*, MIT Press, Cambridge, Massachusetts, 1969, 4th extended edition 1990.

[15] **B. Müller and J. Reinhardt**, *Neural Networks*, Springer Verlag, Berlin, 1991.

[16] **E. Oja**, *A simplified neuron model as a principal component analyzer*, Journal of Mathematical Biology 15, 267–273, 1982.

[17] **R. Rojas**, *Theorie der neuronalen Netze*, Springer Verlag, Berlin, 1993.

[18] **F. Rosenblatt**, *The perceptron: a probabilistic model for information storage and organization in the brain*, Psychological Review 65, 386–408, 1958.

[19] **F. Rosenblatt**, *Principles of Neurodynamics*, Spartan Books, Washington, 1962.

[20] **D. E. Rumelhart and J. L. McClelland**, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. I & II, MIT Press, Cambridge, Massachusetts, 1986.

[21] **R. J. Schalkoff**, *Artificial Neural Networks*, McGraw-Hill, New York, 1997.

[22] **P. J. Werbos**, *Beyond regression: New tools for prediction and analysis in behavioral sciences*, Ph. D. Thesis, Harvard University, 1974.

[23] **B. Widrow and M. E. Hoff**, *Adaptive switching circuits*, 1960 IRE WESCON Convention Record, New York, 96–104, 1960.

[24] **A. Zell**, *Simulation Neuronaler Netze*, Addison-Wesley, Bonn, 1994.