WSG 36/94

Fundamentals in Neurocomputing

Manfred M. Fischer

Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie
Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Vorstand: o.Univ.Prof. Dr. Manfred M. Fischer
A - 1090 Wien, Augasse 2-6, Tel. (0222) 313 36 - 4836

Redaktion: Mag. Petra Staufer

WSG 36/94

Fundamentals in Neurocomputing

Manfred M. Fischer

WSG-Discussion Paper 36

February 1994

Gedruckt mit Unterstützung
des Bundesministerium
für Wissenschaft und Forschung
in Wien

WSG Discussion Papers are interim reports presenting work in progress and papers which have been submitted for publication elsewhere.

ISBN 3 85037 041 0

FUNDAMENTALS IN NEUROCOMPUTING

Abstract

Neurocomputing - inspired from neuroscience - provides the potential of an alternative information processing paradigm that involves large interconnected networks of relatively simple and typically non-linear processing elements, so-called (artificial) neural networks. There has been a recent resurgence in the field of neural networks, caused by new net topologies and algorithms, and the belief that massive parallelism is essential for high performance in several research areas, especially in pattern recognition. This contribution provides a brief introduction to some basic features of neural networks by defining a neural network, reflecting current thinking about the processing that should be performed at each processing element of a neural network, discussing the general categories of training that are commonly used to adjust a neural network's weight vector, and finally by characterizing the backpropagation neural network which is one of the most important historical developments in neurocomputing. The contribution concludes with pointing to some hot topics for future research. It is hoped that this contribution will stimulate the study of neural networks in quantitative geography and regional science.

INTRODUCTION

Neurocomputing, a new paradigm to information processing, has grown rapidly in popularity in the last decade. The primary information structures of interest in neurocomputing are neural networks, although other classes of adaptive information structures are considered such as genetic learning systems, simulated annealing systems and fuzzy learning systems. Several features distinguish this paradigm from algorithmic and rule-based information systems. **First**, information processing is inherently parallel. Large-scale parallelism provides a way to increase significantly the speed of information processing (**inherent parallelism**). **Second**, knowledge is encoded not in symbolic structures, but rather in patterns of numerical strengths of the connections between the processing elements of the system (**connectionist type of knowledge representation**) (Smolensky, 1989). **Third**, neural networks are extremely **fault tolerant**. They can learn from and make decisions based on incomplete, noisy and fuzzy information. **Fourth**, neurocomputing does not require algorithm or rule development and often significantly reduces the quantity of software that has to be developed.

The new paradigm to information processing shows a great potential for those types of problems, especially in the areas of pattern recognition and exploratory data analysis, for which the algorithms or rules are not known, or where they might be known, but where the software to implement them would be too expensive or inconvenient to develop. For those information processing operations suitable to neurocomputing implementation, the software to be developed is characteristically for relatively straightforward operations such as data preprocessing, data file input, data postprocessing and data file output. CASE (Computer Aided Software Engineering) tools might be used to build these routine software modules (Hecht-Nielsen, 1990).

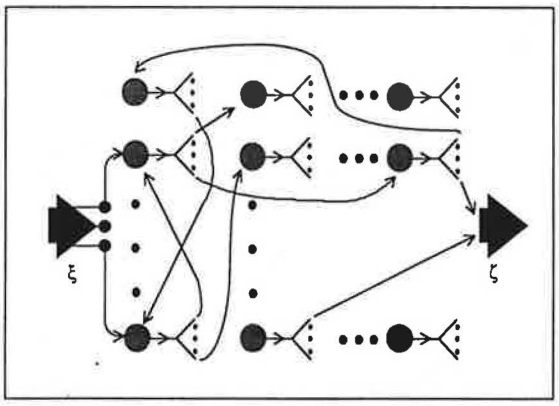
The promise of the neurocomputing paradigm and the excitement evident across various disciplines such as cognitive science, psychology, neuroscience, computer science and engineering is founded on demonstrated successes in solving a diversity of difficult problems encompassing the areas of vision, pattern recognition, process control and non-linear prediction. This contribution provides a brief introduction to some basic features of neural networks by defining a neural network, reflecting current thinking about the processing that should be performed at each processing element of a neural network, discussing the general categories of training that are commonly used to adjust a neural network's weight vector, and finally by characterizing the backpropagation neural network which is one of the most important historical developments in neurocomputing. The contribution concludes with pointing to some hot topics for future research. It is hoped that this contribution will stimulate the study of neural networks in quantitative geography and regional science.

WHAT IS A NEURAL NETWORK ?

A neural network may be viewed as a parallel distributed information processing structure in the form of a directed graph. The nodes of the network are called **processing elements** or **neurons** and the links **connections** which function as unidirectional signal conduction paths. Each connection has a weight associated with it that specifies the strength of the link. Each processing element can receive any number of incoming connections and can have any number of outgoing connections, but the signals in all these outgoing connections have to be the same. Thus, in effect, each processing element has a single output connection which can branch into copies to form multiple output connections, where each carrying the same signal. The signal carried by a connection can be anyone of the various mathematical data types (an integer, a real number, or a complex number). The information processing active within each processing element can be defined arbitrarily with the restriction that it has to be completely local, i.e. it has to depend only on the current values of the input signals arriving at the processing element and on values stored in the processing element local memory (Hecht-Nielsen, 1990).

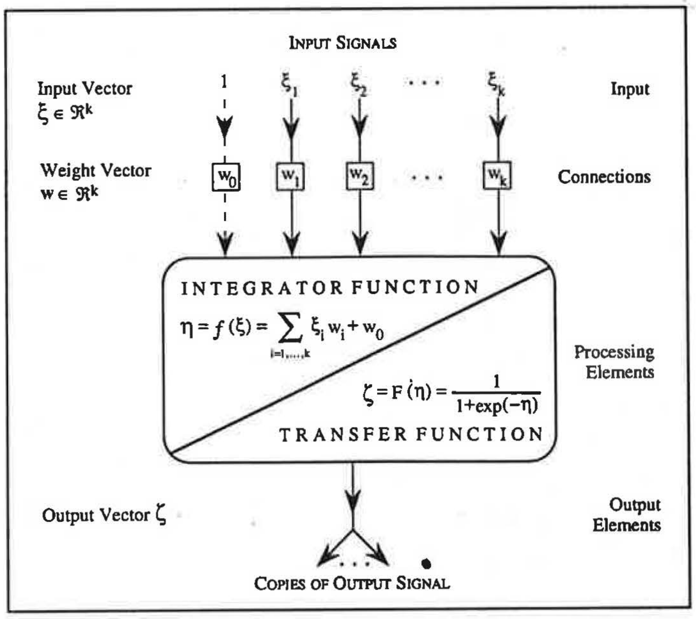
Figure 1 shows a typical neural network architecture. The input to the network considered as a data array ξ and the output of the network as a data array ζ. Viewed in this manners the general functional form of a network is similar to that of a software procedure "input $\rightarrow$ processing $\rightarrow$ output".

Figure 1: A Typical Neural Network Architecture (see Hecht-Nielsen, 1990)



Whether implemented in parallel hardware or simulated on a von Neumann computer, all neural networks consist of a collection of simple processing elements that work together to solve problems.

Figure 2: A Typical Processing Element



INPUT SIGNALS
Input Vector $\xi \in \Re^k$
1 ξ_1 ξ_2 . . . ξ_k
Input
Weight Vector $w \in \Re^k$
w_0 w_1 w_2 . . . w_k
Connections
INTEGRATOR FUNCTION
$\eta = f(\xi) = \sum_{i=1,\dots,k} \xi_i w_i + w_0$
$\zeta = F(\eta) = \frac{1}{1+\exp(-\eta)}$
TRANSFER FUNCTION
Processing Elements
Output Vector ζ
Output Elements
COPIES OF OUTPUT SIGNAL

Figure 2 reflects current thinking about information processing that should be performed at each processing element in a neural network. Characteristically two mathematical functions are active (see Bezdek 1993):

- The first mathematical function is an **integrator function**, say f, which integrates the connection weights, say $w=\{w_i\}$, with the input signals, say $\xi=\{\xi_i\}$, arriving via the incoming connections which impinge upon the processing element. The first entry in each vector in Figure 2 is shown by a dotted line to indicate the bias weight w_0 connected to a constant input $\xi_0= 1$. Typically f is an inner product, usually the euclidean dot product, say

$$\eta = f(\xi) = < \xi, \mathbf{w} > = \sum_{i=1,\dots,k} \xi_i w_i + w_0 \tag{1}$$

  where w_0 is an unknown parameter which has to be predefined or learned during training. w_0 represents the offset from the origin of $\Re^k$ to the hyperplane normal to **w** defined by f. Without loss of generality the augmented vectors $\xi=(1,\xi_1,\dots,\xi_k)^T$ and $\mathbf{w}=(w_0, w_1,\dots, w_k)^T$ may be considered as input and weight vectors, respectively, in $\Re^{k+1}$. A processing element with this type of integrator function is called **first-order processing element** because f is an affine function of its input vector ξ. When the inner product f is replaced by a more complicated function, higher order processing elements arise. For example, a second order processing element may be realized with a quadratic form, say $\xi^T \mathbf{w}\, \xi$, in ξ. It is important to note that each processing element may be viewed as having (k+1) unknowns, but only k inputs.

- Each processing element typically applies a **transfer** (or activation) **function**, say F, to the value of the integrator function (or activation) on its inputs. The transfer function produces the processing element's output signal. The most common choice for F is the logistic function

$$\zeta = F(\eta) = \frac{1}{1+\exp(-\eta)} \tag{2}$$

  which scales the activation sigmoidally between 0 and 1.

TRAINING THE NETWORK

The challenge is to find a set of weights (or adaptive coefficients) and a set of N processing elements which will produce a reasonable output in response to input signals. This leads to a third important mathematical operator for a neural network, the **update function** U which performs usually local updates of the current set of weights at every processing element. The action of the update (or learning) rule may be formally written as

$$\mathbf{w}(t+1) = U(\mathbf{w}(t)) \tag{3}$$

with

$$\mathbf{w}(t) = (\mathbf{w}_1(t), \ldots, \mathbf{w}_N(t)) \tag{4}$$

denoting the network weight (connection parameter) vector, i.e. the collection of all the individual vectors at the N processing elements in the network at any time (iteration) t. The weight vector $\mathbf{w}_n$ (n=1,. . ., N) is stored in the n-th processing element's local memory.

Updating is done during training. There are various types of training processes. At the most fundamental level three categories of training are distinguished: supervised training, unsupervised training (or self-organization), and graded (or reinforcement) training. **Supervised training** implies a situation in which the neural network is operating as input-output system. In this training scheme the network is supplied with a sequence of input and desired (target) output data, and the network is, thus, precisely told what should be emitted as output. The input-target output data set has to cover all reasonable input-output types, and contain enough noise to encourage generalization. An adaptation algorithm automatically adjusts the weights so that the output responses to the input patterns will be as close as possible to their respective desired (target) responses. There is a variety of supervised neural network training (or parameter adaptation) algorithms. The concept underlying these algorithms is the minimal disturbance principle which suggests to adapt the weights to reduce the output error for the current training pattern, with minimal disturbance to responses already learned (Widrow and Lehr, 1990). Supervised trained networks may be viewed as input-output models or as non-linear regression models with a quite specific form.

In some cases there is no teacher who can provide all the desired output values for a particular input. In such cases **graded** (reinforcement) **training** is a useful training scheme. Graded training is similar to supervised training except that the network receives only a grade or numeric score, i.e. a value of some network performance measurement function measured over a time period, which tells how well the network has done over the time period encompassing a sequence of input-output trials. Graded training networks are usually less capable and less generally applicable than supervised training networks. They were applied to control and process-optimization problems up to now where there is no way to know what the target outputs should be and where the network necessary to solve the problem at hand is relatively small.

The third category of training is **unsupervised training** (or self organization) It is a training scheme in which the network is given only input data, and is expected to modify itself in response to it. Unsupervised learning is about discovering structure in the input data. The term training no longer refers to an input-output framework. Many unsupervised training schemes allow the network to change as the character of the input changes. Some techniques are available to monitor this change and assist to prevent the neural system from totally losing its knowledge of earlier input examples when exposed to a new type (Padgett and Karplus, 1993). Examples of the use of unsupervised training include competitive learning algorithms as used, for example, in Kohonen's self-organizing feature maps (Kohonen, 1984).

SUPERVISED TRAINING BY THE STOCHASTIC LEARNING RULE

The backpropagation technique by Rumelhart, Hinton and Williams (1986) has unquestionably been the most influential development in the field of neural networks during the past decade. Amari (1993) showed that this technique is a version of the stochastic descent method for parametrized networks, an idea which has been known since the 1960s and will be developed in the sequel.

Let us consider a network which receives a vector input signal ξ and emits an output signal ζ. The system includes feedforward, but not feedback connections. Let the system include a number of modifiable parameters $\mathbf{w} = (w_1,...,w_j,...,w_N)$ with $w_j = (w_{j1},...,w_{ji},...,w_{jk})$ wihch specify the network, where w_{ji} denotes the weight associated with the link from processing element i to processing element j. Then the network defines a mapping from the set $\Xi = \{\xi\}$ of input signals to the set $Z = \{\zeta\}$ of output signals. The network is determined as a function of input ξ and the parameter values $\mathbf{w}$ as

$$\zeta = G\ (\xi;\mathbf{w}) \tag{5}$$

We assume that G is differentiable with respect to $\mathbf{w}$. When an input signal ξ is processed by a network specified by $\mathbf{w}$, an error is caused because the network might not be optimally tuned. Let us assume that a desired output ϑ accompanies ξ. Then the loss (error) may be written as $L(\xi,\ \vartheta;\ \mathbf{w})$, denoting the error when ξ with a desired (target) signal ϑ is processed by the network specified by $\mathbf{w}$.

Let us assume that ξ is generated subject to a fixed, but unknown probability distribution $p(\xi)$ each time independently. The associated desired output ϑ is usually a function of $\xi(\vartheta{=}\vartheta_d(\xi))$, sometimes disturbed by noise. Then ϑ is generated subject to the conditional probability $p(\vartheta|\xi)$ and the expectation of ϑ:

$$\vartheta_d(\xi) = E\,[\vartheta|\xi] = \int \vartheta\, p(\vartheta|\xi)\, d\vartheta \tag{6}$$

where $E[\vartheta|\xi]$ is the conditional expectation of ϑ under the condition that the input ξ is the desired signal (White, 1989). The risk $R(\mathbf{w})$ of using a network specified by $\mathbf{w}$ is given by the expectation of the error

$$R(\mathbf{w}) = E\,[L\,(\xi,\vartheta,\mathbf{w})] = \int L(\xi,\vartheta,w)\;\; p(\xi)\;\; p(\vartheta|\xi)\;\; d\xi\;\; d\vartheta \tag{7}$$

By receiving an input- (desired) output pair $(\xi(t),\ \vartheta(t))$ at time (iteration) t the stochastic descent rule updates the current parameter $\mathbf{w}(t)$ to

$$\mathbf{w}(t+1) = \mathbf{w}(t) - c(t)\; A\; \frac{\partial L\;(\xi(t),\vartheta(t);\ \mathbf{w}(t))}{\partial \mathbf{w}} \tag{8}$$

where c(t) is a positive constant which may depend on t, A is a positive-definite matrix, and $\partial/\partial w$ is the gradient operator. Since

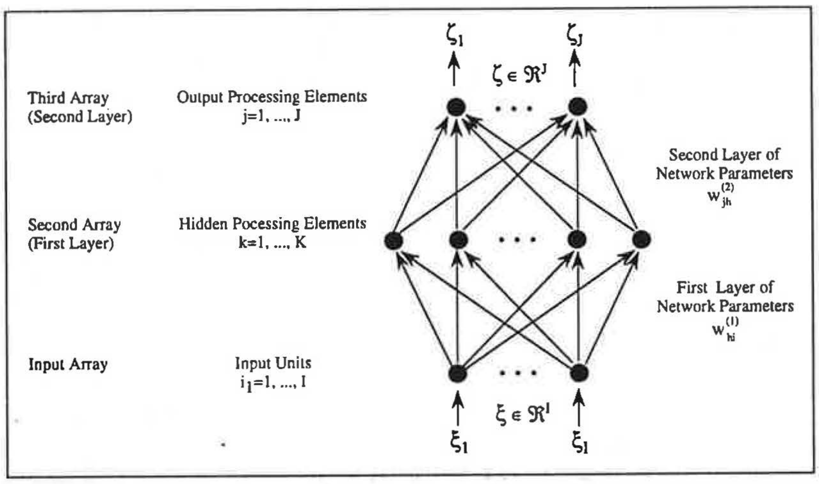
$$E\left[\frac{\partial L(\xi,\vartheta;\ \mathbf{w})}{\partial \mathbf{w}}\right] = \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}} \tag{9}$$

(8) adapts the current parameter, $\mathbf{w}(t)$, in the direction of decreasing $R(\mathbf{w})$ on average depending on the randomly generated (ξ,ϑ). This is the reason why this rule is called stochastic descent rule. It is worthwhile to mention that in the simplest case, c(t) equals a constant c and A is a unit matrix.

THE BACKPROPAGATION NEURAL NETWORK

The backpropagation neural network is by far the most popular and pervasive neural information system. It is a powerful mapping network that has been applied to a wide variety of problems ranging from image compression over credit applications to interregional telecommunication flow modelling in geography (see, for example, Fischer and Gopal, 1993). The backpropagation neural network has a particular neural network architecture characterized by a hierarchical design consisting of an array of input nodes, at least one layer of intermediate (hidden) typically non-linear nodes and an output layer. Such networks are called multilayer networks. In Figure 3 a two layer feedforward network is displayed. The information processing operation that backpropagation networks are intended to carry out is the approximation of a bounded mapping from a compact subset $\{\xi_1, ..., \xi_I\}$ of I-dimensional euclidean space to a compact subset $\{\zeta_1, ..., \zeta_J\}$ f J-dimensional euclidean space by means of supervised training, using the backpropagation procedurte as update rule.

Figure 3: The 2-Layer Feedforward Network



Third Array (Second Layer)
Output Processing Elements j=1, ..., J
Second Array (First Layer)
Hidden Pocessing Elements k=1, ..., K
Input Array
Input Units i1=1, ..., I
Second Layer of Network Parameters $w_{jh}^{(2)}$
First Layer of Network Parameters $w_{hi}^{(1)}$
ζ_1
ζ_J
$\zeta \in \Re^J$
ξ_1
ξ_I
$\xi \in \Re^I$

Consider a multilayer feedforward network: Let $\xi_i^{(s-1)}$ be the i-th input to the processing elements of the s-th layer (s=1, ..., S, in Figure 3: S=2). The j-th processing element of the s-th layer integrates the connection weights from the (s-1)-th to the s-th layer, $w_{ji}^{(s)}$, with the input signals as

$$\eta_j^{(s)} = \sum_i \xi_i^{(s-1)} w_{ji}^{(s)} \tag{10}$$

and emits the output

$$\xi_j^{(s)} = F(\eta_j^{(s)}) \tag{11}$$

where F is the logistic transfer function (2). $\xi_j^{(s)}$ becomes the j-th input to the next (s+1)-th layer. The overall input to the network is given by

$$\xi_i = \xi_i^{(0)} \qquad \text{for } i=1,\ldots, I \tag{12}$$

and the overall output of the network is given from the S-th layer by

$$\zeta_j = \xi_J^{(S)} = F(\eta_j^{(S)}) \qquad \text{for } j=1, \ldots, J \tag{13}$$

These equations recurrently describe the input-output relation (5) where the parameter **w** contains all the $w_{ji}^{(s)}$. When the squared error is used as error function

$$L(\xi,\vartheta;W) = \tfrac{1}{2}\sum_{j} (\zeta_j - \vartheta_j)^2 \tag{14}$$

the stochastic gradient rule yields the following learning rule

$$\Delta\, w_{ji}^{(s)} \;=\; -\,c\,\frac{\partial L(\xi,\vartheta,w)}{\partial w_{ji}} \;=\; c\; l_j^{(s)}\; \xi_i^{(s-1)} \tag{15}$$

where $l_j^{(s)}$ is the error (loss) signal of the j-th processing element of the s-th layer. The error signals are given recurrently by

$$l_j^{(S)} = F'\,(\eta_j^{(S)}) \tag{16}$$

for the connections from the (S-1)-th to the S-th layer, and for the other connections

$$l_j^{(s)} = F'\,(\eta_j^{(s)}) \sum_{i} w_{ji}^{(s)}\, l_i^{(s+1)} \qquad \text{for } s = 1, \ldots, S\text{-}1 \tag{17}$$

Since the error signals $l_j^{(s)}$ can be recurrently computed from the last layer by backpropagating them the procedure is termed backpropagation (see Rumelhart, Williams and Hinton 1986). The generally good performance found for the backpropagation algorithm is somewhat surprising considering that it is a gradient search procedure which may find a local minimum in the error function instead of the global minimum. Suggestions to improve performance and to increase rates of convergence include heuristic modifications of the backpropagation algorithm (Jacobs 1988), making many training runs starting with different sets of random weights, and adopting a more complex error function (Weigend, Rumelhart and Huberman 1991).

OUTLOOK

During the last few years, neural network techniques have proven to be capable of solving a number of difficult problems better than conventional methods. While for many problems an adequate neural network solution may be implementable in software running on a conventional microprocessor, a large and increasing class of problems exists now where microprocessors are orders-of-magnitude too slow, or too expensive to offer an appropriate solution. This is especially true for on-line learning in the context of environmental monitoring and management tasks using huge masses of satellite data. For these applications a new hardware is required.

Several scholars are currently investigating to explore ways and means of building fuzzy neural networks, by incorporating the notion of fuzziness into a neural network framework, especially in the area of pattern recognition. There are several ways to incorporate fuzziness into a backpropagation network. One way is to incorporate fuzziness by arranging the integrator / transfer functions at each processing element to perform some sort of fuzzy aggregation (fuzzy union, weighted mean or intersection) on the numerical information arriving at each processing element. Another way to introduce fuzziness into the neural network framework is through the input data itself which may be "fuzzified" in one of several ways (Bezdek 1993). Fuzzy systems techniques may be also used to interpret the neural network output and to guide the training and validation procedures (Kosko 1992).

Another hot topic refers to the integration of neural networks and genetic algorithms to automate the design of neural networks through genetic search. Genetic algorithms use global search, recombination and other optimization techniques motivated by evolutionary biology (Goldberg 1989). The integration of neural and fuzzy systems and genetic algorithms will be perhaps the most important horizon for research in the next decade.

Acknowledgement: *The author is grateful for the funding provided by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung (P 09972-TEC). Moreover, he would like to thank Petra Staufer for her valuable assistance in the preparation of the illustrations and manuscript.*

REFERENCES

Amari, S.-I. (1993): Backpropagation and stochastic gradient descent method. In: **Neurocomputing**, vol. 5, pp. 185-196.

Bezdek, J.C. (1993): Pattern recognition with fuzzy sets and neural nets. In: **Tutorial Texts, International Joint Conference on Neural Networks IJCNN'93**, pp. 169-206.

Fischer, M.M. and Gopal, S. (1993): Artificial neural networks, A new approach to modelling interregional telecommunication flows. In: **Journal of Regional Science** (in press).

Goldberg, D.E. (1989): **Genetic Algorithms in Search, Optimization and Machine Learning**. Reading (Ma.): Addison-Wesley.

Gopal, S. and Fischer, M.M. (1994): Learning in single hidden layer feedforward network models. Backpropagation and a real world application. Submitted to: **Geographical Analysis.**

Hecht-Nielsen, R. (1990): **Neurocomputing.** Reading (Ma.): Addison-Wesley.

Jacobs, R.A. (1988): Increased rates of convergence through learning rate adaptation. In: **Neural Networks**, vol. 1, pp. 295-307.

Kohonen, T. (1984): **Self-Organization and Associative Memory**. Berlin: Springer.

Kosko, B. (1992): **Neural Networks and Fuzzy Systems**. Englewood Cliffs: Prentice-Hall.

Lippmann, R.P. (1987): An introduction to computing with neural networks, **IEEE ASSP Magazine**, April 1987, pp. 4-22.

Padgett, M.L. and Karplus, W.J. (1993): Neural network basics: Applications, examples and standards, **Tutorial Texts, IJCNN'93**, Nagoya, Japan, pp. 383-412.

Rumelhart, D.E., Hinton, G.E., and R.J. Williams (1986): Learning internal representations by error propagation. In Rumelhart, D.E., McClelland, J.L. and the PDP Research Group (eds.): **Parallel Distributed Processing. Explorations in the Microstructures of Cognition. Volume 1: Foundations**, pp. 318-362. Cambridge (Ma.) and London (England): The MIT Press.

Smolensky, P. (1988): On the proper treatment of connectionism. In: **Behavioral and Brain Sciences**, vol. 11, pp. 1-74.

Weigend, A.S., Rumelhart, D.E. and Huberman, B.A. (1991): Back-propagation, weight-elimination and time series prediction, in Touretzki, D.S., Elman, J.L., Sejnowski, T., and Hinton, G.E. (eds.): **Connectionist Models. Proceedings of the 1990 Summer School**, pp.105-116. San Mateo (CA), Morgan Kaufmann Publishers.

White, H. (1989): Learning in artificial networks: A statistical perspective. In: **Neural Computation**, vol. 1, pp. 425-464.

Widrow, B. and Lehr, M.A. (1990): 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. In: **Proceedings of the IEEE**, vol. 78, pp. 27-53.