

# INTRODUCTION TO CONCEPTS IN ARTIFICIAL NEURAL NETWORKS

Dagmar Niebur, Member, IEEE

Jet Propulsion Laboratory,  
California Institute of Technology  
Pasadena, CA 91109, USA

**Abstract-** This introduction to artificial neural networks summarizes some basic concepts of computational neuroscience and the resulting models of artificial neurons. The terminology of biological and artificial neurons, biological and machine learning and neural processing is introduced. The concepts of supervised and unsupervised learning are explained with examples from the power system area. Finally, a taxonomy of different types of neurons and different classes of artificial neural networks is presented.

## 1. INTRODUCTION

The discipline of computational neuroscience has three goals, first the computer-aided simulation of some functionalities of the brain, second the understanding of the function of the brain in computational terms and third the application of neural concepts for innovative technical problem solving. A detailed discussion of functionalities and models in computational neuroscience as well as references concerning experimental data and theoretical models can be found in [Churchland and Sejnowski, 1992].

The theory of artificial neural networks (ANN) is mainly motivated by the second goal, i. e. the establishment of simple formal models of biological neurons and their interconnections called artificial neural networks; for an excellent introduction, see [Iertz, Krogh and Palmer, 1991]. In the power engineering domain, predominantly the third goal is, i.e. the application of already simplified tools of ANNs to technical problems remains the main objective, [Niebur *et al.*, "1993].

Although Churchland and Sejnowski [1992] give strong arguments for the validity of the simulation of complex behavior with very simple models, this tutorial is not so much concerned with the biological plausibility of the discussed artificial neural network models as with the applicability of the discovered principles to a technical task. Nature and technology have different goals, means, materials and constraints. For instance, as Churchland and Sejnowski note, nature does not start "from scratch." When "developing" birds, nature had to start from dinosaurs. The material as biological matter was given, the means were basically random mutation and the constraints had to take into account multiple criteria. Concerning the task of flying, an optimally designed bird not only had to fly fast but also to be able to reproduce. Although inspired by nature, the engineer can specify the constraints, design an object, simulate the model

on a computer, choose an appropriate material and finally build, e. g., a machine. His solution can be globally optimal in the given frame work whereas nature can only achieve a local optimum, if only the constraints of a specific task (like flying fast) without reference to the larger context are taken into account. However, when proceeding as described, it is highly unlikely that the engineer will develop a tool which flies only satisfactorily but is a magnificent swimmer and diver, as nature has been able to produce in the form of some birds.

Staying with this analogy, we understand an artificial neural network as a brain-inspired computer which may solve a similar task as the biological brain, but will not be an imitation neither in material nor means *nor* constraints. In the following text the term "neuron" will usually refer to the basic unit of an artificial neural model, and "learning" and "training" designate machine learning techniques.

## 2. BIOLOGICAL NEURONS

One of the earliest descriptions of biological neurons is due to Cajal [1894], who identified a neuron as an independent electric device transmitting and receiving electrical signals. Although at least 500 different types of biological neurons have been distinguished, many neurons have a general structure similar to that shown schematically in Fig. 1. The following description of the function of biological neurons is necessarily simplified, see [Churchland and Sejnowski, 1992] for a more detailed and very readable introduction.

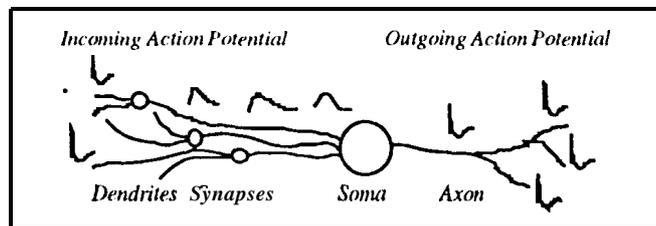


Fig. 1 Schematic drawing of a biological neuron model. We show the principal parts of the neuron, as introduced in the text, as well as schematized shapes of the neural signals. Note that this shape changes along the dendrites but remains the same when traveling along the axon

The principal components of a typical biological neuron are the cell body (or soma), the dendrites and the axon. The function of the dendrites is the collection and conduction of electric potentials which are generated at the synapses when a presynaptic neuron experiences an action potential ("spike"). If the intracellular potential in the soma exceeds a certain

value, called the threshold, an action potential is generated which travels along the axon and the intracellular voltage is reset to a value close to the so-called resting potential (the voltage obtained in the absence of synaptic inputs). The axon is connected via synapses to the dendrites of other neurons (which are called postsynaptic) and therefore the action potential will influence the voltage in these neurons. Functionally, the dendrites are closely associated with the input of the cell and the axon with the output.

There are basically two types of synapses, one called excitatory and the other inhibitory. Activation of excitatory synapses increases the voltage in the postsynaptic neuron while activation of inhibitory synapses decreases the voltage. If many excitatory synapses are activated frequently, and if only few inhibitory synapses are activated, the intracellular voltage of the neuron increases rapidly and reaches threshold fast. Therefore, the neuron will generate action potentials at a high rate, i. e., the neuron will be very active. The soma, where the instantaneous voltage is compared with the threshold and, depending on the result of this comparison, an action potential is either generated or not, corresponds most closely to the decision-making instance of the neuron.

This description of neural function is grossly simplified and there are many exceptions to this prototypical behavior. For instance, there are many classes of neurons which do not have a clear distinction between axons and dendrites. Other neurons are not even capable of generating action potentials and they communicate by other means. Also, our classification of synapses in only two types (excitatory and inhibitory) is simplified, and we have completely neglected the interior dynamics of cells, which are far more complex than just the simple summation of synaptic inputs described above. The final point we would like to make here is that recently also the notion of the spiking frequency being the only signal transmitted between neurons has been questioned, and that there is increasing evidence that the time structure of the sequences of spikes plays an important role. We have presented this simplified description of neuronal function since it is at this level that the elements of artificial neural networks are usually modeled.

### 3. THE ARTIFICIAL NEURON - A Computational Model OF THE BIOLOGICAL NEURON

McCulloch and Pitts [1943] established the first computational model of a biological neuron, by translating the biological concepts as shown in Fig. 2: incoming and outgoing action potentials and synapses are presented by in general real-valued vectors. The simulation of the soma is modeled as the weighted sum (i. e. scalar product) of incoming vector and weight vector and the amplification of the axon is modeled by a (in general non-linear) gain function. In diagrams showing artificial neurons scalar product and gain function are usually grouped to form the artificial neural unit.

The simplest formal model, the logic threshold unit is shown in Fig. 3. It consists of a two-dimensional binary input vector  $x$ , the gain function of the neural unit being the threshold function  $g(x)$  with a given threshold  $\theta$  as given in (1),

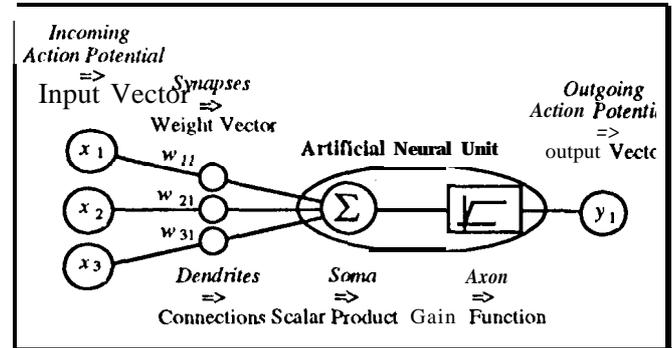


Fig. 2: Schematic drawing of the artificial neuron model. Terms in italic denote the biological parts Their formal components are denoted in normal type.

and a one-dimensional output vector. The synaptic weights are modeled by two real numbers  $w_k, k = 1, 2$ . The neuron processes the binary input components  $x_k, k = 1, 2$  as follows:

$$y = g\left(\sum_{k=1}^2 w_k x_k\right) \text{ with } g(h) = \begin{cases} 1 & \text{for } h \leq \theta \\ 0 & \text{for } h > \theta \end{cases} \quad (1)$$

Although extremely simple, this artificial neuron can calculate Boolean functions. For example for the Boolean values  $1 = \text{TRUE}$  and  $-1 = \text{FALSE}$ , for the given synaptic weights  $w_1 = w_2 = 1$  and a threshold value  $\theta = -0.5$ , the neural net computes the output  $y$  as the Logical OR between  $x_1$  and  $x_2$ . For the same weights and a threshold value  $\theta = +0.5$ , the neural net calculates the Logical AND.

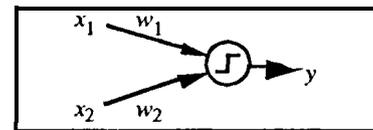


Fig. 3 The logic threshold unit with threshold  $\theta$  being part of the gain function

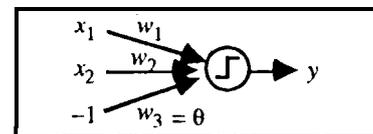


Fig. 4 The logic threshold unit with fixed third input and with threshold  $\theta$  being part of the weight vector

A straightforward calculation shows that the gain function  $g$  can be chosen as the sign function if the dimension of the

input vector and weight vector is augmented and their values are clamped to -1 and  $\theta$  respectively.

$$y = g\left(\sum_{k=1}^3 w_k x_k\right) \text{ with } g(h) = \begin{cases} 1 & \text{for } h \geq \theta \\ 0 & \text{for } h < \theta \end{cases} \quad (2)$$

In the following we will therefore model any threshold  $\theta$  by an extra synaptic weight. Fig. 4 shows the architecture of the equivalent unit.

#### 4. CHARACTERIZATION OF ARTIFICIAL NEURAL NETWORKS

So far we have defined the architecture of a simple neuron. We have also seen how the neuron processes input vectors, provided the weights are known. In section 7. we will discuss strategies, commonly referred to as learning, how to determine the weight vectors for a given set of input vectors,

The described simple model can be generalized in many ways. Every artificial neural network model can be characterized by the following features: its *architecture*, its *processing* algorithm and its *training* algorithm,

Combining several neurons so that the output  $y_j$  of neuron  $j$  serves as input to one or several neurons leads to networks of artificial neurons. The *architecture* specifies the arrangement of neural connections as well as the type of units characterized by its gain function.

For a given architecture the neural network is used in two different modes, the *processing* mode and the *training* mode.

In the *processing* mode, the *processing* algorithm specifies how the neural unit for a given set of weights calculates the output vector  $y$  for any input vector  $x$ . The type of processing further depends on the type of the gain function.

The *training* algorithm specifies how the neural network adapts its weights for all  $M$  given input vectors  $x$ , called training vectors, from a set of given vectors, the training set.

Let us now examine these three characteristics, architecture, processing and training in more detail.

#### 5. ARCHITECTURE OF ARTIFICIAL NEURAL NETWORKS

In general the architecture, see Fig. 5, consists of three parts, the  $n$ -dimensional *input layer* where the input data is fed in, the *neural network layer* consisting of  $N$  neurons interconnected in various ways and the  $m$ -dimensional *output layer*. The  $n$  input vector and  $m$  output vector components are either binary or real numbers. Each input and output vector component can be connected with each neuron through a *synaptic weight*, which is a real number. We therefore have two *weight matrices*, one  $m \times N$ -dimensional matrix for the input layer - neural net layer connection and one  $N \times n$ -

dimensional matrix for the neural net layer - output layer connections. The way the neurons are connected in the neural net layer is specific for the different existing models as for example the *multi-layer perception*, *Kohonen's self-organizing feature map* and the *Hopfield model*. Examples of architectures will be explained in the following section. There exist a variety of other types of neural architectures, see [Krogh, et al., 1991].

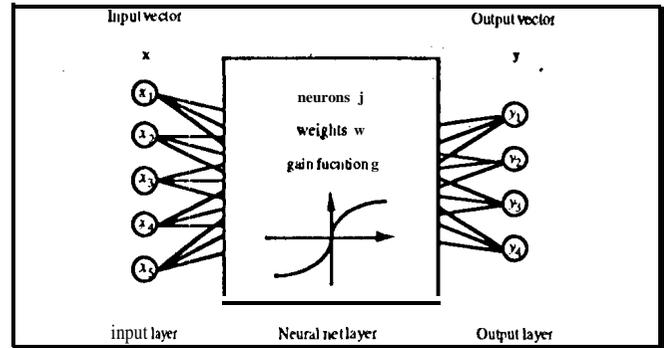


Fig. 5 General neural network architecture.

With respect to the architecture four main types of neural networks can be distinguished:

- 1) *Layered feed-forward neural networks*, where a layer of neurons receives input only from previous layers as for instance the *multi-layer perception (MLP)* shown in Fig. 6. The functional relation  $y = f(x)$  between input and output is usually not given in an analytical form.

The flow of information for the processing of input vectors with fixed weight vectors is in one direction only. Input units feed the input values directly to the hidden neurons whereas hidden and output units process their input through a non-linear gain function.

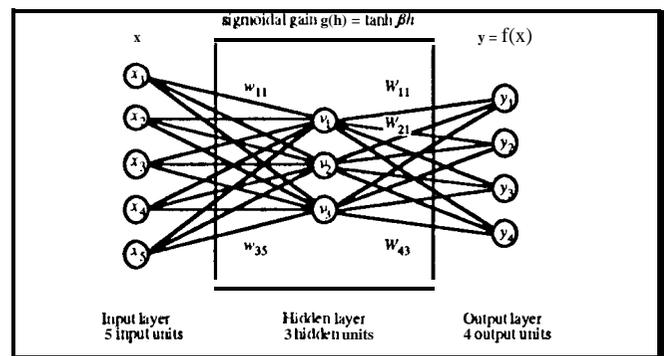


Fig. 6: Architecture of a multi-layer perceptron containing 5 input, 3 hidden and 4 output units.

For the MLP we will show later that the flow of information during training is in two directions, forwards to calculate the actual output and backwards in order to back-propagate the error for the correction of the weights.

2) *Recurrent neural networks*, where the inputs to a neuron are the net's previous outputs as well as inputs from external sources which are input  $x_i$  and bias (equivalent to a shift of the threshold)  $I_i$ . The fully connected *Hopfield net* [Hopfield, 1982] shown in Fig. 7 is an example for this type of architecture. Here, the neurons process their input through a threshold function.

During processing the Hopfield net will feedback its output, during training however only one feed-forward step is used.

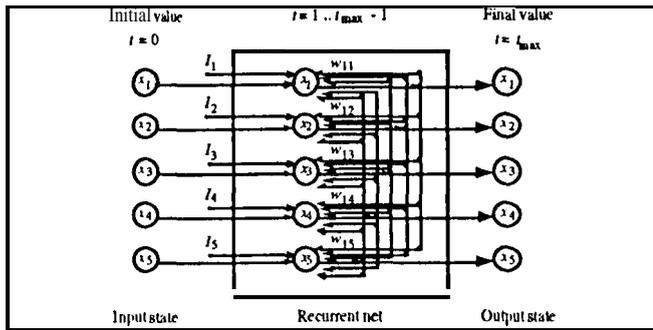


Fig. 7: Architecture of the Hopfield network containing 5 input, 5 recurrent and 5 output units. Note that, in contrast to the MLP, the input and output units correspond to states, not to physical neurons

3) *Laterally connected neural networks* consisting of feed-forward input units and a lateral layer consisting of  $m$  neurons, which are laterally connected to their neighbors. The *Kohonen network* shown in Fig. 8 is an example for this type of network. It will be discussed in detail in the tutorial chapter [Niebur, 1996],

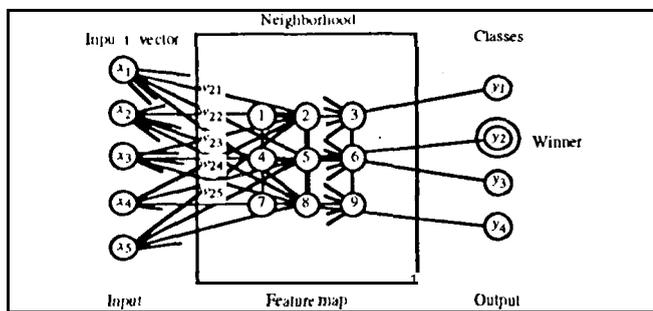


Fig. 8: Architecture of the Kohonen network containing 5 input units and 9 laterally connected units. The number of output classes depends on the characteristics of the training set, and is at most equal to the number of neurons.

The number of the output classes depends on the characteristics of the training set and can thus not be considered as part of the architecture which is specified in advance. In the original model training as well as processing results in a recurrent dynamical process involving all laterally connected neurons. The most commonly used simplification, however, applies a winner-take-all feed-forward strat-

egy where only one neuron and its close neighbors are stimulated by the input,

4) *Hybrid networks* combine two or three of the above features. For example, the *Boltzmann machine* has a hidden layer with recurrent connections. The two neural layers of the Counter-propagation network consist of a Kohonen layer and a feed-forward layer,

## 6. PROCESSING OF INFORMATION WITH ARTIFICIAL NEURONS

In biological neural networks, the incoming action potential will excite different neurons to a different degree. Depending on the size of the stimulus the neuron will then amplify or inhibit the incoming signal. In artificial neural networks the degree of the excitation is usually measured as the similarity between input vector and weight vector. Higher similarity usually results in a larger output. A saturated gain function ensures that this process stays bounded.

Similarity of two vectors can be defined as generalized *colinearity*. In this case the angle formed by two vectors serves as a measure of similarity. Out of all weight vectors  $w_i, i = 1, \dots, m$ , the weight vector  $w_i^*$  is the most similar to an input vector  $x$ , if their scalar product (i. e. weighted sum of components) takes its maximum

$$\begin{aligned} \text{sim}(x, w) &:= \langle x, w_i^* \rangle \\ &= \max \{ \langle x, w_i \rangle \mid i = 1, \dots, m \} \end{aligned} \quad (3)$$

A second concept of similarity uses the Euclidean distance of two vectors as a measure. Here  $w_i^*$  is most similar to  $x$  if

$$\begin{aligned} \text{sim}(x, w) &:= \Pi x - w_i^* \Pi \\ &= \min \{ \|x - w_i\| \mid i = 1, \dots, m \} \end{aligned} \quad (4)$$

Applying the parallelogram equation it can be easily shown that these two concepts are equivalent for normalized vectors.

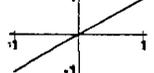
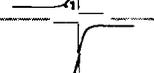
The gain function  $g$  further determines the way how the neuron amplifies its output  $y$  for a given weight vector  $w$  and a given input vector  $x$ . The sign function will either produce a TRUE or FALSE answer for all output neurons. The winner-take-all function will produce a TRUE output only for the most stimulated neuron and FALSE for all others. A linear threshold function will produce saturated responses for very large and very small stimulation only.

The processing algorithm for one neuron in general is:

$$y = g(\text{sim}(x, w)) \quad (5)$$

The most commonly used similarity measure is the scalar product of  $x$  and  $w$  for  $x, w \in \mathbb{R}^m$ .

TAXONOMY <sup>TABLE 1</sup> FOR ONE FORMAL NEURON,

Neuron:	Gain Function $g: \mathfrak{R} \rightarrow \mathfrak{R}$	Input/Output Vector Components
Threshold unit	 $\text{sign}(h)$	$x \in \{-1, 1\}^m$ $y \in \{-1, 1\}$
Linear unit	 $\text{identity}$	$x \in \mathfrak{R}^m$ $y \in [-1, 1]$
Non-linear unit (sigmoid unit)	 $\tanh(\beta h)$	$x \in \mathfrak{R}^m$ $y \in [-1, 1]$
Non-linear unit (sigmoid unit)	 $(1 + \exp(-2\beta h))^{-1}$	$x \in \mathfrak{R}^m$ $y \in [0, 1]$
Radial basis function unit	 $\exp(-(h-i^*)^2/2\beta^2)$ $i^*$ denotes most stimulated unit	$x \in \mathfrak{R}^m$ $y \in [0, 1]$
Winner-take all unit	 $g(h) = \begin{cases} 1 & \text{for most stimulated unit } i^* \\ 0 & \text{elsewhere} \end{cases}$	$x \in \mathfrak{R}^m$ $y \in \{0, 1\}$

The logic threshold model can be easily extended for  $n$ -dimensional real-valued input vectors. Depending on the task to be solved, subsequent models replaced the threshold function by a

- 1) linear or linear-saturated gain function (linear perception, adaptive linear element or "adaline"),
- 2) non-linear gain function, usually a sigmoid function (non-linear perceptron)
- 3) the winner-take-all function (self-organizing feature map neuron)
- 4) Gaussian function (radial basis function neuron).

Table I shows examples of commonly used neural units characterized by their gain functions.

### 7. LEARNING IN NEURAL NETWORKS

Neural networks are commonly used for tasks like pattern recognition, content addressable memories, approximation, classification, parameter estimation and control. We will now discuss strategies for the determination of weights in order to achieve the desired objective. These strategies are usually called learning or training.

#### 7.1 Hebb's rule - a biological learning hypothesis

In the example of the logic threshold unit, the synaptic weights were assumed fixed and known in advance. In biological terms this would correspond to the synaptic

strength being genetically pre-determined. Already from the estimated number of neurons and synapses in the human central nervous system (on the order of  $10^{12}$  neurons and  $10^{15}$  synapses) it is, however, clear, that not all synaptic connections can be pre-coded. It is a biological fact that structural and functional changes in the nervous system, usually called plasticity, can result from experience or damage. In psychological terms, plasticity is at the base of changes in behavior due to experience (learning). Engineers usually prefer the term "training" which does not imply any "intelligence" of the learning individual respectively computer. Therefore this term is less prone to philosophical discussions about whether a machine can be intelligent.

In 1949, the psychologist D. Hebb [Hebb, 1949] formulated the hypothesis that synapses change in efficacy according to the following principle: The strength increases when both pre- and postsynaptic elements are active simultaneously (learning). The synapse may decrease in strength if there is a presynaptic activity without concurrent postsynaptic activation (forgetting).

Formulated in terms of artificial neurons Hebb's learning rule can be stated as follows:

For the weight vector  $w$  of an artificial neuron, given input vector  $x$  and output  $y$ , synaptic learning can be expressed as follows:

$$\Delta w = \eta y x \quad \eta > 0 \quad (6)$$

There are models which also take forgetting into account by adding a negative term proportional to the output and the strength of the synapses, i. e. the weight.

$$\Delta w = \eta y x - \alpha y w = \alpha y ((\eta/\alpha)x - w) \quad (7)$$

with  $\eta, \alpha > 0$

The learning or adaptation rate  $\eta$  and the forgetting factor  $\alpha$  are often chosen to be time-dependent, e. g.  $\eta(t) \propto 1/t$ .

Synapses of Hebbian type as well as of non-Hebbian type have been identified physiologically, but the details of the implementation of biological plasticity are still open. Nevertheless, most artificial neural networks base their learning algorithm on Hebb's generalized learning principle (7).

## 7.2 Machine learning

For artificial neural networks, the neural net objectives, for example pattern classification, have to be defined for a set of examples called the *training set*.

The training algorithm specifies how the neural network adapts its weights for all given input vectors  $x$ , called training vectors, from a set of given vectors, the training set. If for every input vector  $x$ , the desired output vector  $y_{\text{target}}$  is given, and the weights are adapted in order to produce the desired output, the training process is called *supervised learning*. If only the input vector is given and the structure of the data is discovered autonomously, the training is called *unsupervised learning*.

### 7.2.1 Learning as an optimization task

Let us go back to our most simple example of a neural net, the Logic Threshold Unit shown in Fig. 3, which calculates the Logic AND function. We will now determine the weights of the unit in order to solve this task.

Defining TRUE as 1 and FALSE as -1 the training set is given by 4 input vectors:

$$x^1 = (-1, -1), x^2 = (-1, 1), x^3 = (1, -1), \text{ and } x^4 = (1, 1)$$

and their corresponding target output

$$y_{\text{target}}^1 = -1, y_{\text{target}}^2 = -1, y_{\text{target}}^3 = -1, \text{ and } y_{\text{target}}^4 = 1.$$

For the moment, let us assume a linear gain function. We now want to determine the weights such that the calculated output is equal to the target output

$$y(w) = \langle x^\mu, w \rangle = y_{\text{target}}^\mu \quad \text{for all } \mu = 1, \dots, 4. \quad (8)$$

Since this problem is overdetermined we will formulate it as a least square error minimization problem. Note, we are further defining the threshold  $\theta$ , as part of the weight vector as outlined in (2) and shown in Fig. 4. We therefore added a fixed third component to all input vectors.

$$E(w) = \frac{1}{2} \sum_{\mu=1}^4 (y^\mu(w) - y_{\text{target}}^\mu)^2$$

$$= \frac{1}{2} \sum_{\mu=1}^4 \left( \left( \sum_{i=1}^3 w_i x_i \right) - y_{\text{target}}^\mu \right)^2 = \min!$$

or in its equivalent matrix form (9)

$$E(w) = 1/2 (X w - y_{\text{target}})^T (X w - y_{\text{target}}) = \min!$$

where

$$X^T = \begin{bmatrix} x^1 & x^2 & x^3 & x^4 \\ -1 & -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

and  $y_{\text{target}} = [-1, -1, -1, 1]^T$  and  $w = [w_1, w_2, \theta]^T$ .

Because  $E$  is continuously differentiable with respect to the weights, one solution of the least square minimization problem can be obtained by solving

$$\text{grad } E(w) = X^T (X w - y) = 0 \quad (10)$$

For non-singular  $X^T X$ , called pseudo-inverse, the solution of (10) can be expressed as the

$$w = (X^T X)^{-1} X^T y_{\text{target}} \quad (11)$$

(For singular pseudo-inverse the solution exists but has to be calculated as a limiting sequence.) In our particular example the Logic Threshold Unit will perform the Logic AND with  $w = (0.5, 0.5, 0.5)^T$ .

Geometrically our solution represents a hyperplane  $H$  which divides the training set into two classes. Fig. 9 shows another possible solution of this task.

$$H = \{x \in \mathcal{R}^2 \mid \langle x^\mu, w \rangle - \theta = 0\}$$

such that (12)

$$y_{\text{target}}^\mu = \text{sign}(\langle x^\mu, w \rangle - \theta), \quad \text{for } \mu = 1, \dots, 4$$

We therefore have formulated a supervised learning task as an optimization task. An excellent discussion of neural networks within the framework of optimization and signal processing can be found in [Cichocki and Unbehauen, 1993].

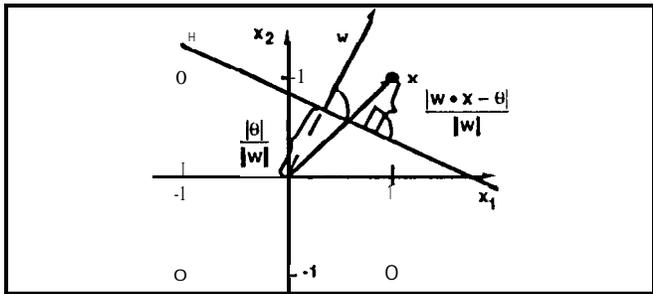


Fig. 9 The Logic AND problem as a linear separability task. The black, the white and the two shaded dots denote the four training vectors. Values above the hyperplane like the black dot are classified as TRUE, values below as FALSE.

### 7.2.2. Reflections on learning paradigms

The following re-evaluation of the learning technique proposed in 7.2.1 apply to supervised as well as unsupervised learning techniques. In this simple example we have implicitly made four severely restrictive assumptions.

- 1) We have assumed that our learning task is defined by a continuously differentiable error function to which efficient optimization techniques can be applied.

In the case of self-organizing feature maps, a winner-take-all net trained in an unsupervised manner, only a non-differentiable error function exists for training vectors distributed with a discrete probability. This case will be discussed in the next tutorial chapter.

- 2) We have assumed that the solution of our learning task exists. This is only the case if the training set can be linearly separated (*i. e.* by hyperplanes) into the target output classes.

If the training set is not linearly separable, the Least Square Minimization problem does not have non-zero solutions. For example the Logic XOR cannot be calculated with the Logic Threshold Unit. In Fig. 9 this task would require to place a hyperplane, *i.e.* straight line, such that the shaded dots lie on one side and the black dots on the other side of the line. The interested reader may verify that for  $y_{\text{target}} = (-1, 1, 1, -1)^T$ ,  $w = (0, 0, 0)^T$  is the only solution.

In the next tutorial chapter we will see that replacing the gain function by a sigmoid function provides a powerful remedy to this problem. For the multi-layer perceptron and a sigmoid form of the gain function, the partial derivatives of the error with respect to the weights can easily be computed numerically. The error function is usually minimized with a stochastic gradient search technique which is known as the back-propagation algorithm and which was independently developed by [Werbos, 1974] and [Rumelhart *et al.*, 1986].

- 3) In order to establish matrix X, we have assumed that all training patterns are known beforehand.

In many adaptive tasks, however, optimization has to be adjusted to new incoming patterns. This task can usually not be solved by otherwise powerful optimization tools like the calculation of the pseudo-inverse or Quasi-Newton techniques. Furthermore for large training sets the inversion of the pseudo-inverse  $X^T X$  represents a heavy computational burden, especially in the case of singular pseudo-inverses. We will present an iterative and adaptive algorithm in the next section.

- 4) Last, but not least we did not use a biologically plausible learning concept.

In general we are not overly concerned if our machine learning task achieves the same objective as biological learning, by using different means. However, by neglecting biological learning paradigms like simple computation, local adaptation, and robustness concerning failing neurons, researchers would have missed the opportunity to develop robust, simple, efficient and highly parallel hardware circuits, which are quite different from the techniques employed by conventional super-computers [Mead *et al.*, 1989]. Neural-bawd hardware has recently been introduced for power system security assessment, [Cornu *et al.*, 1994].

### 7.2.3. Supervised learning - learning for parameter estimation

Supervised learning techniques, also commonly referred to as *learning by example* or *learning with a teacher* fall in the same class of tasks as regression analysis and parameter estimation.

Note that in this framework, classification tasks can be formulated as the task of finding a regression model for the function which maps an input vector  $x$  onto its class label, for example TRUE or FALSE coded with binary numbers. The non-linear gain function like the sign function will map real-valued weighted sums onto binary outputs,

In the following, we will define a local learning rule which solves the logical AND problem while respecting the biological learning paradigms cited above. It is based on a simple iterative optimization algorithm, the steepest gradient descent technique. It converts the task of finding the zero of the function  $\text{grad } E(w)$  into the task of finding the fixpoint of a related function  $G$ . Under certain conditions, the latter can be solved by a simple iterative method. Thus

$$\text{grad } E(w^*) = 0 \iff G(w^*) = w^* \quad (13)$$

with

$$G(w) := w - \eta(t) \text{grad } E(w) \quad \text{and} \quad \eta(t) \text{ bounded}$$

The iterative procedure of steepest descent is shown in (14). The expression for the gradient of the error function  $E$  was obtained based on  $y^{\mu}$  given in (7) and  $E$  given in (8)

$$\begin{aligned} w(t+1) &= w(t) - \eta(t) \text{grad } E(w(t)) \\ &= w(t) + \eta(t) \sum_{\mu=1}^4 (y^{\mu}(w(t)) - y_{\text{target}}^{\mu}) x^{\mu} \end{aligned} \quad (14)$$

So far we have replaced the inversion of the pseudo-inverse by an iterative procedure. This procedure still needs the knowledge over the whole training set.

However instead of minimizing the error globally we can now try to minimize the error locally by randomly taking one training example  $(x^{\mu}, y_{\text{target}}^{\mu})$  at a time

$$\begin{aligned} w(t+1) &= w(t) + \eta(t) (y^{\mu}(w(t)) - y_{\text{target}}^{\mu}) x^{\mu} \\ &= w(t) + \eta(t) \delta^{\mu} x^{\mu} \end{aligned} \quad (15)$$

This stochastic updating or learning rule is commonly referred to as the *LMS rule*, the *Widrow-Hoff rule*, or the *delta rule*. It is one of the earliest adaptive "neural units, called *adaptive linear unit* or *ADALINE* and was used for adaptive control, [Widrow and Hoff, 1960].

The iterative process converges stochastically to the minimum of the error function, if the so-called "Robbins-Monro" conditions hold for the learning rate  $\eta$ , [Duda and Hart, 1972]

$$\sum_{t=0}^{\infty} \eta(t) < +\infty \quad \text{and} \quad \sum_{t=0}^{\infty} \eta(t)^2 < \infty \quad (16)$$

Although only applicable to linearly separable learning tasks, see remarks 7.2.3, the delta rule fulfills several of the biological paradigms. It is computationally simple, robust with respect to noisy input data as well as numerical rounding errors and it is a local adaptation scheme learning one example at a time.

It further obeys a generalization of Hebb's learning principle (8). For a fixed input and target output, the weight changes depend on two terms only, the correlation of input  $x$  and calculated output  $y$ , and a constant stimulus, the product of input and target output.

There are other supervised learning rules based on Hebb's principle like the perceptron rule and the generalized delta rule, introduced in the next tutorial chapter. Other types of neural networks trained with a different type of supervised learning like the Functional-Link Net are discussed in [Pao, 1989].

#### 7.2.4. Unsupervised learning - learning for data reduction

In unsupervised learning the input vectors of the training set are given, but the corresponding target outputs are not

specified. Unsupervised neural nets fall into the same class of tools as statistical non-parametric data analysis, clustering algorithms and encoding or decoding techniques.

Their main goal consists in data reduction. The reduction of the data set of input vectors can be achieved in two different ways: either by *reducing the dimensionality* of the input vector, or by *reducing the number* of input vectors.

The simplest neural network for unsupervised learning consists of a layer of feed-forward winner-take-all units. For each input vector only one such unit will respond, namely the unit characterized by the maximum output, respectively minimum distance, for this input vector  $x$ . The units of the network are thus competing for selection. Only the weights of the winner will be adapted. All input vectors responding to the same unit are said to form a class and the weight vector of this unit is called the class "prototype." Here, the gain function is defined to yield one for the maximum respectively minimum output, and zero otherwise.

"Winner-take-all" units are related to "grandmother cells" because they are responsible for selecting one specific feature, e. g. the feature presenting the stereotypical grandmother. Note that this representation is not robust because when one unit is removed (or one cell dies in a biological brain), all information concerning the corresponding class would be lost.

A solution to this dilemma is proposed by Kohonen's self organizing feature map where (ideally) neighboring neurons classify neighboring features and thus the loss in one neuron will result in a decrease of accuracy but not in a complete loss of information.

Let us briefly introduce the main concepts used in some types of unsupervised networks. For more detailed information see [Krogh *et al.*, 1991; Haykin, 1995]. Figs. 10, 11 and 12 show schematically how the data reduction of randomly distributed data is achieved using 3 different types of unsupervised networks.

The first unsupervised approach for the *reduction of the dimension* of the input vector falls in the class of subspace techniques where the input vector is projected on a linear subspace presenting the most salient features. Statistical principal component analysis chooses the subspace spanned by the eigenvectors of the correlation matrix of the input vector. The standard deviation of the input vectors take their maximal and minimal values along the eigenvectors corresponding to maximal and minimal eigenvalues. A simple example is shown in Fig. 10 where the data variation along the horizontal axis is more prominent than the one along the vertical axis.

A non-competitive unsupervised network for principal component analysis based on Hebb's learning rule was proposed by Oja [1989] and generalized by Sanger, [1989];

for details and references to this work see also [Haykin *et al.*, 1995].

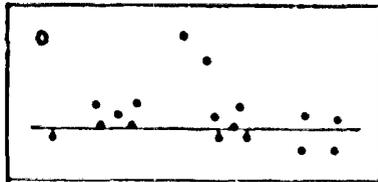


Fig. 10 Data projection onto a one-dimensional hyperplane. Each data point will be represented by its lower-dimensional projection onto the straight line. The shaded circle denotes a new input vector for which the projection exists, although the classification error will be large.

The second unsupervised approach for the reduction of the number of input vectors is based on clustering techniques. In order to reduce this number, the neural net categorizes the training vectors into classes or clusters based on the concept of similarity introduced in section 6. For the examples we will use the Euclidean distance between two vectors as a measure of similarity.

In classical clustering techniques, such as the ISODATA algorithm, [Duda and Hart, 1973], clusters are formed by computing the distance between an input vector and already existing clusters. If the distance between the input vector and the reference vector of an existing cluster is smaller than a previously defined threshold, the new input vector is grouped with this cluster; otherwise, a new cluster is formed. Functionally, a spherical neighborhood is formed around the reference vector of each new cluster. Note that the diameter of the sphere is predetermined, whereas the number of clusters is not. An example of this type of clustering is presented in Fig. 11. A similar objective is achieved by the Adaptive Resonance Theory (ART) networks [Carpenter and Grossberg, 1987].

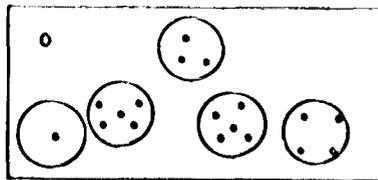


Fig. 11 Clustering of data into a variable number of classes of fixed diameter. The center of the circles, not presented in this figure present the class prototypes. The shaded circle denotes a new input vector which does not fall into any of the trained classes.

In vector quantization techniques based on the LBG algorithm [Linde *et al.*, 1980] or the *k-means* clustering, like the Kohonen network, [Kohonen, 1989], the maximal number of clusters is determined by the number of neurons in the map. The weight vectors are the reference vectors or prototypes of the class. On the other hand, the distance around the reference vector of a cluster is not predetermined and the region is, in general, not spherical. Instead, the clusters are large in the regions where the density of

probability of the input vectors is small, and vice-versa, as shown in Fig. 12.

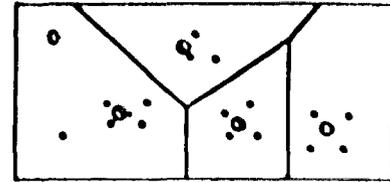


Fig. 12 Tessellation of data into a fixed number of classes of variable diameter. The striped circles represent the class prototypes. The shaded circle denotes a new input vector which does not fall into one of the trained classes although the classification error will be large.

In the case of simple vector quantization, that is for a Kohonen network with winner-take-all units and no neighbor stimulation, the network minimizes the average distortion error between the input vectors and their reference vector. The regions themselves correspond to the Voronoi tessellation, and boundaries of the regions around a cluster are hyperplanes. More details will be presented in the chapter on Kohonen networks. Important results and references can be found in [Ritter *et al.*, 1992].

### 7.3 Purpose of training in power systems

Let us illustrate the concepts of supervised and unsupervised learning for a very simple power system shown in Fig. 13, consisting of two generation buses *a*, *b*, one load bus, *c*, and three lines *ab*, *ac*, *bc*, whose active power flows  $p_{ab}$ ,  $p_{ac}$ , and  $p_{bc}$  are limited by the maximal active line powers, i. e.  $p_{ab \max}$ ,  $p_{ac \max}$  and  $p_{bc \max}$ .

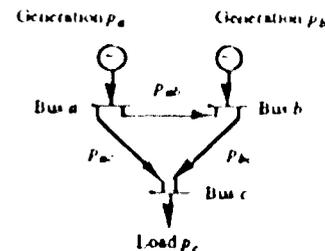


Fig. 13 A 3-bus-3-line linear power system model.

The operating vector can be chosen to consist of the active line powers  $(p_{ab}, p_{ac}, p_{bc})^T$ . In this case the secure operating space is defined by a parallelepiped whose boundaries are determined by  $p_{ab \max}$ ,  $p_{ac \max}$  and  $p_{bc \max}$ , see Fig. 14. For simplicity we will throughout this work refer to this parallelepiped as the security "cube". Operating points inside the shaded cube are secure, points inside but at the border are critical and operating point outside the shaded cube are insecure because they violate at least one constraint on the maximum admissible line powers.

This example is based on several simplifications. Only active powers have been considered. In the general case the cube has

to be replaced by a non-linear manifold. Furthermore, not all vectors of the three-dimensional power system operating space shown in Fig. 14 represent feasible operating states, since Volt- $g$ -VAR constraints and Kirchhoff's laws apply for each bus and each line. Nevertheless, the example illustrates conveniently the differences between supervised and unsupervised learning.

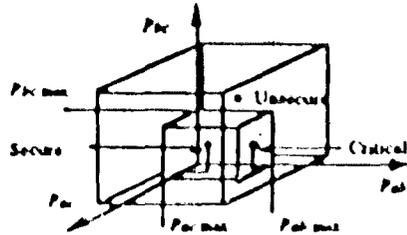


Fig. 14 The operating space of the 3-bus 3-line linear power system model

*Supervised training* approximates the boundaries of the operating space for the training set and interpolates in between known data points. It basically constructs separating hyperplanes (manifolds in the non-linear case) corresponding to the surfaces of the shaded secure cube in Fig. 14. An example for this technique as well as several enhancements are discussed in [El-Sharkawi et al., 1991].

However, because in the general case the dimension of the operating space is very high (in the order of 500 for a medium sized power system at the transmission level), it is not feasible to generate a set of operating points which is densely distributed in the operating space and to analyze the operating points with multiple contingency analysis off-line. In order to overcome this "curse of dimensionality", *unsupervised learning* tackles the dimensionality problem first based on two different approaches

- a) Subspace techniques
- b) Quantization techniques.

The simplest *subspace technique* is the conventional contingency ranking techniques. If for example the outage of line  $ab$  is selected as the most important contingency, the operating space of the linear model is projected to a two-dimensional subspace as illustrated in Fig. 15.

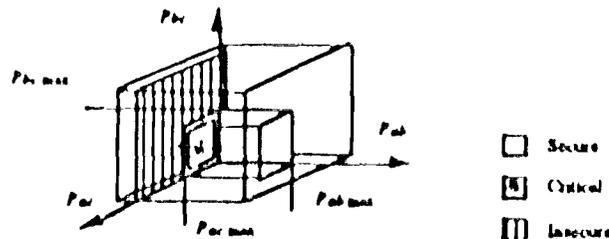


Fig. 15 Limitation of the number of contingencies

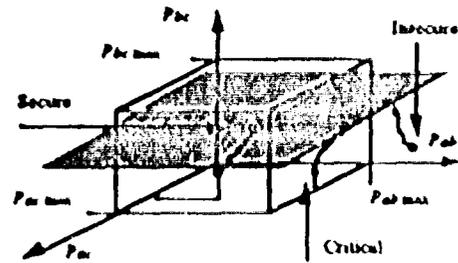


Fig. 16 Reduction of the dimension of the operating vector

Conventional load flow analysis examines the projection of the base case onto this subspace. Supervised techniques are also applied to construct the boundaries of the projected reduced, security cube, see [El-Sharkawi et al., 1991].

Fig. 16 shows an more general example of the reduction of the operating space by a lower-dimensional manifold. Depending on the projection used for reduction, the manifold may be a linear or even orthogonal subspace.

In [Weerasooriya and El-Sharkawi, 1991] the principal component analysis method (also called Karhunen-Löve expansion) is used to reduce the dimensionality of the training vectors and construct the eigenspace corresponding to the most significant components of the input vector. The researchers implemented their approach in a conventional algorithmic manner instead of using Oja's and Sanger's neural net approach. The second class of unsupervised approaches encountered in power system security assessment are *quantization techniques*. Fig. 17 shows an example of the quantization of the operating space into classes of typical states. Depending on the distance measure used for classification, classes may be hyperboxes, spheres or in the case of the self-organizing feature map, of a more general form because of the arrangement of neurons on a grid. The classes usually do not divide the cube crisply in secure and insecure areas, but may contain critically high loaded as well as slightly overloaded cases.

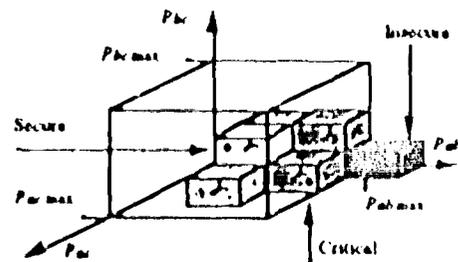


Figure 17: Quantization of the operating space.

The two different clustering approaches discussed in section 7.3 have been applied to security assessment.

In the case of a small space station transmission system, Sobajic *et al.* [1990] quantize the operating space into a variable number of hyperspheres of fixed radius using an unsupervised ART2-like ANN algorithm.

In [Niebur and Germond, 1991] Kohonen's self-organizing feature map is used for the quantization of the operating space. The maximal number of classes is given by the number of neurons whose weight vector represents typical operating states. The size of each class depends on the density of the probability distribution of the training vectors. The operating space is represented on the two-dimensional feature map by secure and insecure regions. This case will be discussed in more detail in the following chapter.

#### 7.4 Comparison of supervised and unsupervised learning

Although usually discussed on equal terms, there is an important difference between supervised and unsupervised learning. Unsupervised learning helps to organize complex features into classes whereas supervised learning will then calculate follow-up features for specific classes.

Unsupervised networks can therefore be viewed as a data pre-processing step which reduces the size of the data set before learning the data's characteristics with supervised learning. The *Functional Link Net* (FLN) is often used in combination with the ART2 network [Sobajic and Pao, 1988]. Other ANNs combining an unsupervised and a supervised step are the *Counter-Propagation Network* (CPN) [Hecht-Nielsen, 1988], and the *Radial Basis Functions Network* (RBF) [Moody and Darken, 1989]. The CPN combines a Kohonen map layer with a feed forward layer. In the case of the RBF, clustering can be achieved by any unsupervised learning or the *k*-means algorithm, and the neurons of the hidden layer are represented by these means. The architecture of the supervised part is a linear feed-forward layer. In contrast to the winner-take all scheme in the Kohonen network, Gaussian activation functions stimulate several neurons at the same time and the output of the network is a weighted sum of these activations.

For security assessment, the combination of an unsupervised step for operating space reduction and a supervised step for operating state classification has been applied by several researchers including [Sobajic and Pao, 1990; El-Sharkawi *et al.*, 1991; Ranaweera and Karady, 1994]

Another example in power systems, where supervised and unsupervised networks are employed for data clustering and estimation is the area of load forecasting [Hsu and Yang 1991]. A Kohonen network separates the forecasting data into representative classes like summer, winter, autumn and spring and further into weekdays and holidays (see also [Macabrey *et al.* 1991]). For each class of data a supervised network is then used for load prediction for the classes data points. For a similar purpose Ranaweera *et al.* [1995] apply the RBF network in the area of load forecasting. Further

detailed examples will be discussed in the other tutorial chapters.

## 8. SUMMARY

We have presented an overview over different types of neural units characterized by their input, output, weight vector, gain function, architecture, processing and learning algorithm.

Tables II-V give a short overview on the different characteristics of neural networks and a non-exhaustive list of examples.

TABLE II

Neural net parameters	
Input vector $x$	Number of neurons
Output vector $y$	Gain function $g(h)$
Weight vector $w$	Learning rate $\eta(t)$

TABLE III

Architecture	Examples
Layered	Multi-layer perceptron
Fully connected	Hopfield
Lateral connections	Kohonen
Hybrid networks	Radial Basis Functions net Counter-Propagation net Boltzmann machine

TABLE IV

Processing ( $x, w$ given, calculate $y$ )	Examples
Feed-forward, feed $x$ once to get $y$	Adaline Multi-layer perceptron Kohonen
Recurrent, iterate $x$ to get $y$	Hopfield Diagonally recurrent ANN

TABLE V

Training ( $x$ given, calculate $w$ )	Examples
Supervised learning ( $y$ given)	Delta rule Back-propagation
Unsupervised learning (no $y$ given)	Principal Component Analysis Self-organization

## 9. ACKNOWLEDGMENT

The work described in this paper was started at the Swiss Federal Institute of Technology, Lausanne (EPFL), sponsored by EPFL, and was completed at the Jet Propulsion Laboratory, California Institute of Technology sponsored by the U.S. Department of Energy through an agreement with the National Aeronautics and Space Administration.

Reference herein to any commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

## 10. REFERENCES

- Cajal, S. R., "La fine structure des centres nerveux," *Procs. Roy. Soc. Lond.* 55, 1894, 444-468.
- Carpenter, O. A. and Grossberg, S., "ART2: Self-organization of stable category recognition category of analog input patterns," *Applied Optics* 26, 1987, 4919-49.
- Churchland, P. S. and Sejnowski, T. J., *The Computational Brain*, The MIT Press, Cambridge MA, 1992.
- Cornu, T., Ienne, P., Niebur, D. and Vireliz, M., "A systolic accelerator for power system security assessment," *Fifth International Symposium on Intelligent System Applications to Power Systems*, Montpellier, France, September 94, 431-438.
- Duda, R. O. and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley and Sons, Inc., New York, 1973.
- El-Sharkawi, M. A. et al., "Artificial neural networks as operating aid for an on-line static security assessment of power systems," *Procs. of the Tenth Power System Computation Conference*, Graz, 1990, 895-901.
- El-Sharkawi, M. A. et al., "Neural networks and their application to power engineering," *Control and Dynamic Systems* 41, Academic Press, 1991.
- Haykin, S., *Neural Networks, A Comprehensive Foundation*, IEEE Press #PC04036, Macmillan College Publishing Company, Inc. Englewood Cliffs, NJ, 1994.
- Hertz, J., Krogh, A. and Palmer, R. G., *Introduction to the Theory of Neural Computing*, Addison Wesley, Reading, Ma, 1991.
- Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities," *Procs. Nat. Acad. Sc.*, 79, 1982, 2554-2558.
- Hsu, Y.-Y., Yang, C.-C., Design of Artificial Neural Networks for Short Term Load Forecasting. Part I: Self-Organizing Feature Map for Type Identification, Part II: Multi-Layer Feed-Forward Networks for Peak Load and Valley Forecasting, *IEE Proceedings-C*, vol. 138, no. 5, September 1991, pp. 407-418.
- Kohonen, T., *Self-organization and Associative Memory*, Third edition, Springer-Verlag, Berlin, 1989.
- Linde, Y., Buzo, A. and Gray, R.M., "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. COM-28, 1980, 84-95.
- McCulloch, W. S. and Pitts, W. H., "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, 1943, 115-133.
- Mead, C., *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA, 1989.
- Moody, J. and Darken, C., "Fast learning in networks of locally tuned processing units," *Neural Computation* 1, 1989, 281-294.
- Niebur, D., "An Example of unsupervised networks - Kohonen's self-organizing feature map," *IEEE\_PES Tutorial on Artificial Neural Networks for Power Systems*, to be published, 1996.
- Niebur, D. and Germond, A. J., "Power system static security assessment using the Kohonen neural network classifier," *Procs of the 7th Power Industry Computer Applications Conference*, Baltimore, May 1991a, also in *IEEE Transaction on Power Systems*, vol. PWRS-7, no. 2, May 1992, 865-872.
- Niebur, D. et al., "Neural network applications in power systems," *Int. Journal of Engineering Intelligent Systems*, vol 1 no 3, December 1993, 133-158.
- Oja, E., "Neural networks, principal components and subspaces," *Int. J. of Neural Systems*, vol. 1, no. 1, 1989, 61-68.
- Pao, Y. H., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA, 1989.
- Ranaweera, D. K. and Karady, G., "Power system security analysis using radial basis function neural network," *Procs. of the 4th Symposium on Expert System Application to Power Systems*, Melbourne, 1993, 272-274.
- Ranaweera, D. K., Hubels, N., and Papalexopoulos, A., "Applications of radial basis function neural network model for short-term load forecasting," *IEE Proc.-Generation, Transmission and Distribution*, vol. 142, no. 1, January 1995, 45-50.
- Ritter, H., Martinetz, T. and Schulten, K., *Neural Networks*, Addison Wesley, 1992.
- Rumelhart, D. E., Hinton, G. E. and Williams, R. J., "Learning internal representations by error propagation," in Rumelhart, D. E. and McClelland, J. L. (eds.), *Parallel distributed processing: Explorations in the Microstructure of Cognition*, 1, 318-362, MIT Press, Cambridge, MA, 1986.
- Sobajic, D. J., Pao, Y.-H. and Dolce, J., "Real-time security monitoring of electric power systems using parallel associative memories," *IEEE International Symposium on Circuit and Systems*, New Orleans, Louisiana, May 1-3, 1990, 2929-2932.
- Weerasooriya, S. and El-Sharkawi, M. A., "Use of Karhunen-Loeve expansion in training neural network for static security assessment," *First International Forum on Applications of Neural Networks to Power Systems*, Seattle, WA, July 23-26, 1991, 59-64.
- Werbos, P. J., Beyond regression: New tools for prediction and analysis in the behavioral sciences, *Doctoral Dissertation*, AI. Math., Harvard University, November 1974.
- Widrow, B. and Hoff, M. E., "Adaptive switching circuits," *IRE WESCOMN Convention Record*, 1960, 96-104.