NEURAL NETS: SURVEY AND APPLICATION TO WAVEFORM PROCESSING

INFORMATICA 1/91

Ivan Bruha
Dept. Computer Science and Systems,
McMaster University, Hamilton, Ont.
Canada L8S4K1

Keywords: neural nets, waveform processing

Abstract

This paper surveys Artificial Neural Nets as essential tools for pattern recognition. The author concentrates on the characteristics of well-known types of currently exploited neural nets and implementation of a multilayer perceptron. Aspects, specification, and comparison of various types of neural nets will be presented, as well.

Furthermore, a decision-supporting system for neurological diagnosis will be described. The actual input to our system is an evoked potential waveform. It is analyzed by a syntax pattern recognition algorithm based on a regular attributed grammar. Its semantic functions return a list of numerical features. The second step of the waveform processing includes a two-layer perceptron which processes the above list of numerical features. The rules of thumb for optimal adjustment of perceptron's parameters will be discussed, too.

1. Pattern Recognition: A Survey

Pattern recognition is one of the oldest and most widely investigated areas of artificial intelligence (AI). Although some researchers do not recognize pattern recognition as a part of AI, there are some others (including the author of the paper) who accept Minsky's definition of AI as 'the science of making machines do things that would require intelligence if done by men' and consider artificial intelligence as a 'club' of several theoretical and experimental disciplines, including pattern recognition. Pattern recognition as a discipline has been thoroughly described, analyzed, and surveyed in a great number of books and monographs. For examples the reader might look at e.g. [Du73], [Fu82], [Tou74], [Wa85], or [Bru80]. Nevertheless, in this section we present the necessary terminology.

The principal function of a pattern recognition system is to make a decision concerning the class membership of the patterns received as input. *Patterns* are facts, observable statements, situations, events, or objects of any type that are to be recognized. The patterns are grouped into two or more sets, so-called *classes*, according to the given problem. Any such system (usually a computer) cannot, of course, observe real-world objects, since it is only able to read in the data of its world, i.e. numbers and symbols. Therefore, before the actual classification, we have to represent patterns by a suitable *formal description*. There are in fact two distinct possibilities for pattern representation:

(a) numerical (statistical) description: a pattern is represented by a sequence of numbers, called features;
(b) symbolic description: a pattern is described either by a string of (terminal) symbols, or by more general structures such as relational structures, semantic nets, frames, formulas of predicate calculus, etc.

Perceptrons and other neural nets we discuss in this paper have strictly numerical character; therefore, we will focus on the numerical approach only. A pattern is represented by so-called *feature vector*

$$\mathbf{x} = [\, x_1 , x_2 , \dots , x_N \,]$$

of numbers, called *features*. The number of features is usually fixed, and therefore all feature vectors form an N-dimensional space, called the *feature space*.

Let a pattern recognition problem comprise R classes; the r-th class will be designated by the symbol z_r , $r = 1,\dots,R$. Then, a pattern recognition system with numerical representation of patterns (called *numerical classifier*) is a device with N inputs (one for each feature) and one output, which classifies an input feature vector $\mathbf{x}$ to one of the given classes, i.e. it yields one of the symbols z_r . The relation between its input and output is called *decision rule* and can be written as the following function

$$z = d(\mathbf{x})$$

The feature space is divided by the decision rule into R disjoint subsets R_r (called *decision regions*) where R_r contains the features $\mathbf{x}$ such that $z_r = d(\mathbf{x})$. The boundaries between decision regions are called *decision boundaries*. Their adjustment is the principle problem of a classifier design.

There are several ways of describing decision boundaries. Among them, a most general approach uses so-called *discriminant* (or *decision*) *functions*. The R discriminant functions $g_r(\mathbf{x})$ have to be selected so that

$$g_s(\mathbf{x}) > g_r(\mathbf{x}) \quad , \quad r = 1,\dots,R \; , \; r \neq s$$

for all $\mathbf{x} \in R_s$. Consequently, the decision boundary between adjacent regions R_r and R_s is defined by the equation

$$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0$$

The decision rule for a classifier with discriminant functions thus has the form: for a given $\mathbf{x}$ compute all $g_r(\mathbf{x})$, $r = 1,\dots,R$ and classify $\mathbf{x}$ to the class z_s for which

$$g_s(\mathbf{x}) = \max_r \, g_r(\mathbf{x}) \qquad (1)$$

see Fig. 1.

The most common type of a numerical classifier is the one with linear discriminant functions:

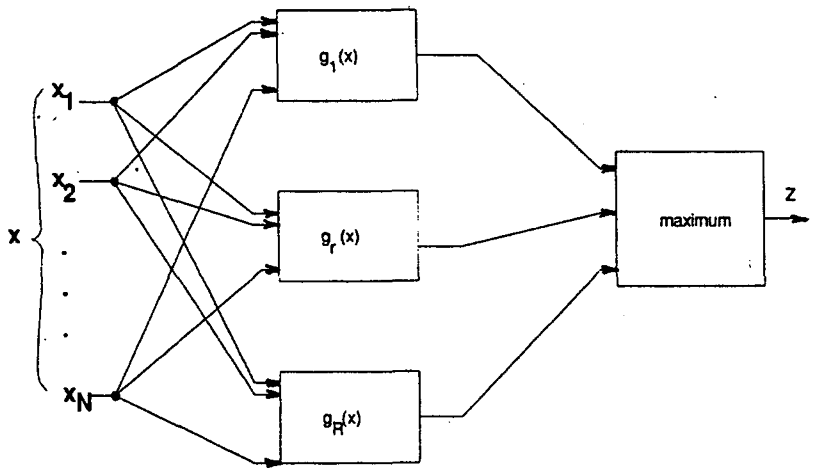


Fig. 1. Classifier with discriminant functions

$$g_r(x) = \sum_{n=0}^{N} w_{nr} x_n \quad , \quad r = 1,\dots,R \tag{2}$$

where w_{nr} , $n = 0,1,\dots,N$, $r = 1,\dots,R$, are so-called *weights*, and $x_0 = 1$. This latter condition allows w_{0r} to be interpreted as a *threshold* of the discriminant function. The structure of a linear classifier is illustrated in Fig. 2. Decision boundaries formed by a linear classifier are parts of hyperplanes (see Fig. 3).

The problem of finding decision boundaries is thus reduced to that of finding optimal values for all the weights of the linear classifier. One possible and often used way of a classifier adjustment is to utilize a *learning (training)* process. In such a case, a teacher (a designer of the classifier) has to collect a reasonable set (*training set*) of typical, representative examples (*training patterns*) including the correct class membership of each training pattern (*desired class*). During the learning process, training patterns with their desired classes are presented (usually many times) by the teacher to the classifier which performs as a student, i.e. it modifies (utilizing the information involved in the training set) its weights according to an appropriate learning algorithm. Among a large collection of learning algorithms for linear classifiers, we introduce the simplest one, called *Rosenblatt's algorithm* (also known as *perceptron* algorithm, *fixed-increment*, *delta*, or *LMSE* algorithm):

1. Initialize weight vectors

   $w_r = [\, w_{0r} \, , \, w_{1r} \, , \dots , \, w_{Nr} \,] \quad , \quad r = 1,\dots,R$

   arbitrarily.

2. For each training pattern $x = [\, x_0, x_1 ,\dots, x_N \,]$ (where we added $x_0 = 1$ for a convenience) with its desired class $Z = z_s$ do:

   2.1. Compute $g_r(x)$, $r = 1,\dots,R$ according to (2).

   2.2. If the equation (1) is satisfied, i.e. the pattern x is correctly classified, do nothing. However, if

   $$g_s(x) < g_r(x) \quad \text{for some } r \tag{3}$$

   then change the weights vectors as follows (c is a positive constant):

   $w_s \mathrel{+}= c\, x$

   $w_r \mathrel{-}= c\, x \quad$ for all r satisfying (3)

   (where A += B means: add B to A ; similarly -= relates to a decrement).

3. If the weights were modified for at least one training pattern, return to the step 2, i.e. repeat the training for the entire training set. If not, save the weights and terminate the learning process.

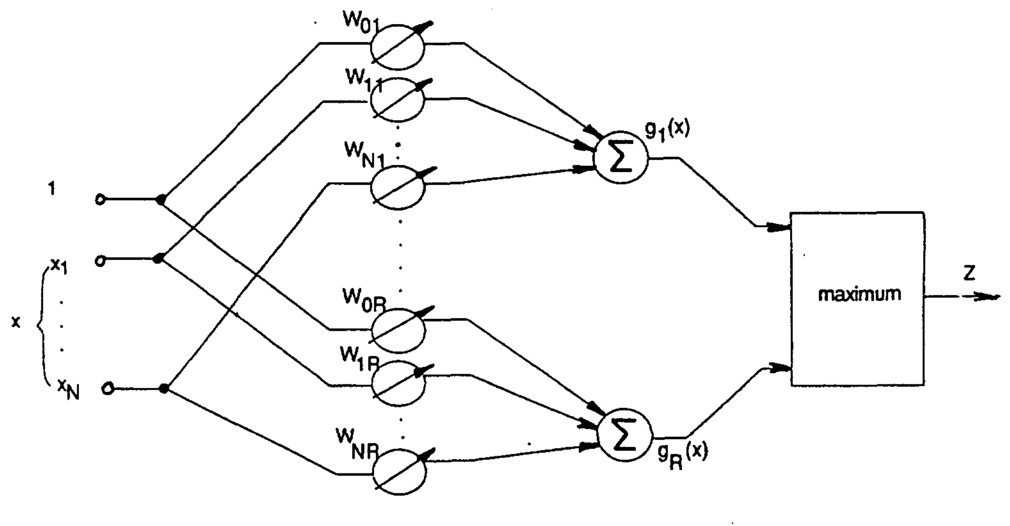


Fig. 2. Linear classifier

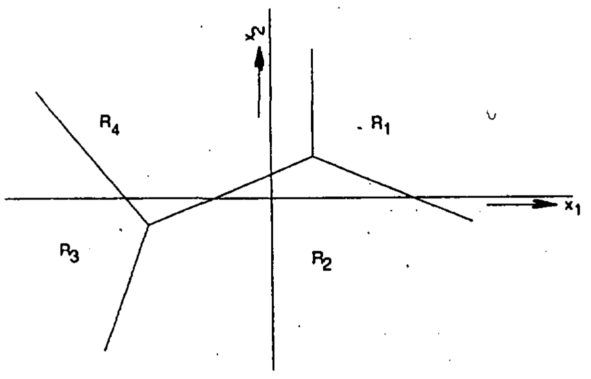


Fig. 3. Decision boundaries of a linear classifier

A simple model of a learning linear classifier was developed by Widrow and Hoff [Wid60]. Their *adaline* (adaptive linear neuron) is able to recognize $R = 2$ classes only (using a single discriminant function) where features can be only equal to +1 or -1, but otherwise its structure and learning is equivalent to those presented above. Adaline has found many useful application in technical areas.

2. Rosenblatt's Perceptron

Development, investigation, and theory of neural nets is not a new discipline at all. The first attempts to model the behaviour of biological nerve cells were realized in 1940's. Their results encouraged the foundation of cybernetics, a discipline (or a 'club' of disciplines), which attempted to combine concepts from biology, psychology, engineering, and mathematics. Nevertheless, more serious research and experimentation with neural nets started in the 1960's with the publication of Rosenblatt's book [Ros61].

Rosenblatt developed a cognitive model of the brain, which was called the *perceptron*. Its structure is shown in Fig. 4. It is very similar to a linear classifier (Fig. 2); however, the features are not input directly to its weights, but instead are preprocessed by a so-called Φ-processor which is represented by M functions $\Phi_m(x) = \Phi_m(x_1,...,x_N)$, $m = 1,...,M$. The Φ-processor transforms the original features to the new ones x'_m , $m = 1,...,M$, which are then processed by a linear classifier. Hence, the discriminant functions of Rosenblatt's perceptron are

$$g_r(x) = w_{0r} + w_{1r}\,\Phi_1(x) + ... + w_{Mr}\,\Phi_M(x) \quad , r = 1,...,R \qquad (4)$$

Consequently, the decision boundaries of the original feature space need not be just hyperplanes but may consist of more general shapes that are transformed by the Φ-processor to linear ones. Another difference between the perceptron and a linear classifier is that both original and new features of the perceptron can only equal to 0 or 1 . Rosenblatt's algorithm is used for the training.

The functions of the Φ-processor can be of any shape, but the following cases are the most interesting ones:

- The Φ-processor consists of quadratic functions; thus decision boundaries are quadratic surfaces.
- The functions are boolean ones; the case with randomly arranged connections (bonds) and boolean gates was extensively studied by Rosenblatt's group since they had expected that this would be the best model of the brain.
- If the Φ-processor is an identity (or is eliminated) we get the 'standard' linear classifier or the Widrow's adaline.

The perceptron is a technical model for the following hypothesis of a nerve system's performance. This hypothesis states that the nerve system of a living organism is more less chaotically (randomly) organized when it arises. Its organization takes place during the life of the organism as a consequence of its adaptation to its environment and learning. The system creates suitable bonds (connections) within its Φ-processor and modifies its weights. As a metaphor one could say that a living organism is a general computer whose program is formed during its life. After the program has been created (the system has learned), a negligible amount of computer hardware is utilized; this is the cost of the versatility of the computer.

Perceptrons were applied to solve many problems in many technical disciplines. In some cases they were 100% successful, in others, however, they did not yield satisfactory results. The fundamental disadvantage of a perceptron consists in its low ability to generalize acquired knowledge; that is the principle difference between perceptrons and living organisms. For instance, a perceptron trained to recognize squares and circles at a certain place on its input retinal matrix, is not able to satisfactorily recognize the same squares and circles situated at another place on the matrix. Precise theoretical investigation of perceptron's limitations were presented by Minsky and Papert [Min69]. Analysis of the perceptron's behaviour revealed that the above hypothesis is not quite correct. Some bonds already seem to be determined when the organism arises, i.e. in our metaphorical example, the designer of a computer does not propose only its hardware, but also some parts of its software as well. In spite of these limitations, the perceptron became one of the fundamental models of neural nets, has had many technical applications, and, at present, has successful successors.

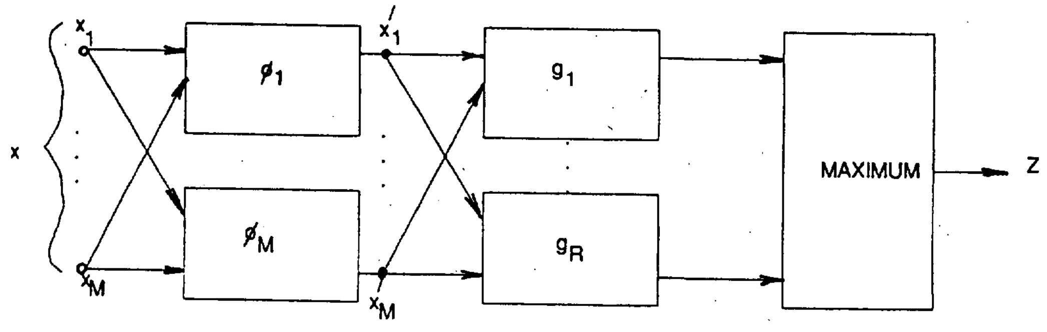


Fig. 4. Perceptron. g_1 to g_R are linear discriminant functions

3. Neural Nets

3.1. Paradigm of Connectionism

Minsky and Papert's criticism of perceptrons strongly diminished interest in their investigation, and consequently research in the field of perceptrons, their applications, and generalization was neglected for many years. Moreover, research activities in 1970's were focused on symbolic approaches and knowledge-intensive modelling. However, a few research centres continued to investigate the neural (or connectionist) approach and parallel processing systems, especially in computer vision; see e.g. Hough, Duda and Hart [Du72], Grossberg [Gro78], Anderson [And72], Kohonen [Koh77], and others (more references are in [Lip87], [Mat87], [Om87]).

After more than a decade of intensive research, however, the apparent limitations of the knowledge-based and symbolic approaches to artificial intelligence have been realized, returning the interest of AI researchers to neural net modelling. Consider just the following. The standard serial von Neumann computers are excellent in multiplication; they can multiply two large numbers in a fragment of one microsecond, but - even in late 80's - any computer with the best available knowledge base and the best available algorithm needs hours to recognize objects in a picture. That is evidently one of the reasons for the recent explosion of interest in parallel distributed processing and neural network models. Neural nets have exhibited promising results for a number of problems such as pattern recognition, speech processing, computer vision, association etc. (see e.g. Kohonen [Koh84], Sejnowski and Rosenberg [Sej86], Hopfield and Tank [Hop86], Rummelhart and McClelland [Ru86a], Hinton and Sejnowski [Hin86], Carpenter and Grossberg [Car86], etc.). A fair survey of types of neural nets and related problems can be found in [Gro88], [Lip87], [Sim90].

The paradigm of connectionism can be specified as follows: *Intelligence is an emergent property of a large parallel net of simple uniform nodes.* Indeed, although there are various types of neural nets, they all attempt to get fast and perfect behaviour by massive interconnections among their elementary units (nodes), and by exploring promising cases (hypotheses) simultaneously, in parallel. An elementary unit (node) usually has a large number of inputs, either from the environment that is to be processed or from the outputs of other nodes. The input signals to a unit are usually weighted and their sum is processed by a nonlinearity whose output is either a binary number (0 or 1) or a real number in a certain interval. The weights of nodes are usually adjustable by a learning (training) algorithm.

In the following, we will discuss all relevant aspects of neural nets and their specifications. We will then survey a few famous types of neural nets, and focus on the multilayer perceptron, one of the best models for the pattern recognition purposes (particularly, it is the best model for our task of doing waveform analysis).

3.2. Aspects of Neural Nets

Hecht-Nielson [He88] defines a neural network as a parallel, distributed information processing structure consisting of processing elements interconnected together with connections; each processing element has a *local* memory, a single output, and carries out *localized* information processing operations. Another definition can be found in [Sim90]: a neural net is a nonlinear directed graph with weighted edges that is able to *store* patterns by changing the edge weights, and is able to *recall* patterns from incomplete and unknown data.

Particularly, the latter definition reflects not only the internal structure of a neural net, but also the procedures (actions) that are carried out by the net. Now, following the paradigm of connectionism and the above definitions, we can observe the following aspects of any artificial neural net.

(1) Set of processing elements (neurons, units, nodes)

A neural net consists of a large numbers of simple elements (neurons, units, nodes) of the same type. Each neuron n has its *activity (state, output)* x_n. It is either a real number, usually from the close interval $< 0; 1>$, or an integer from the set $\{ 0,1,\dots,K \}$, or a binary number 0, 1, or +1, -1 .

(2) Topology of neural nets

Neurons are connected within a net by means of a large number of connections (synapses). Each connection is accompanied by a certain strength or weight. Therefore, the topology of a neural network is characterized by connections, their weights, and so-called interconnection schemes. We will briefly discuss each of these factors.

(a) Connections (synapses). Neurons of a net are connected together by oriented synapses. We can observe a neural net as a graph with oriented edges. There are several types of connections (graphs), among them the most typical ones are:

- total connection: each node is connected with all nodes;
- uniform local connection: each neuron is connected with its neighbours only;
- layered networks: neurons are grouped into layers which are ordered; neurons can be connected to neurons of the same layer, those of the next 'higher' layer, or any 'higher' layer, or 'lower' layer etc.; some of the possible layered types are on Fig. 5.

(b) Weights (connection strengths). If the neuron m is connected to the neuron n, then the corresponding connection (synapsis) has a certain strength or weight w_{mn}. A weight is usually characterised by a real number from a certain interval. All weights of a neural net form a matrix $w = [w_{mn}]$ which is called *(long-term) memory* of the net.

Weights can be either positive numbers (in such a case the connection is called *excitatory*), or negative (*inhibitory* connection). If a weight has zero value, then we have to distinguish either *static* zero (there exists no connection between the two nodes at all) or *dynamic* zero (the initial zero value of the weight has not been modified yet).

If $w_{mn} = w_{nm}$ for all connections, then the neural net is called symmetrical. If $w_{mn} * w_{nm} = 0$ for all connections, then we have one-way connection network. Otherwise, the net is of a general asymmetrical type.

(c) Interconnection schemes. We distinguish feedforward and feedback schemes. If information (signals) flows in one direction only, the net exhibits so-called *feedforward* scheme. If information flows in either direction and/or information flows recursively the net has the *feedback* scheme. Most feedback systems exploit recursive information flow with the following *stopping condition*: the information flows until the output of the network ceases to change.

(3) Environment

A neural net is working in a certain environment, processes environmental stimuli and returns its responses (output signals). The nodes which are directly connected to the environment (i.e. they process the environmental inputs, stimuli) are called *input* nodes. The nodes whose outputs (activities) are returned to the environment are called *output* nodes. Other nodes (i.e. those whose inputs and/or output are not directly connected to the environment) are called *hidden* ones; see Fig. 6.

Consequently, we can characterize a neural net as a system with the mapping $\mathbf{b} = S(\mathbf{a})$ where $\mathbf{a}$ is a vector of all environmental inputs, $\mathbf{b}$ a vector of outputs (activities) of all output nodes (i.e. the response of the neural system to the given environmental input).

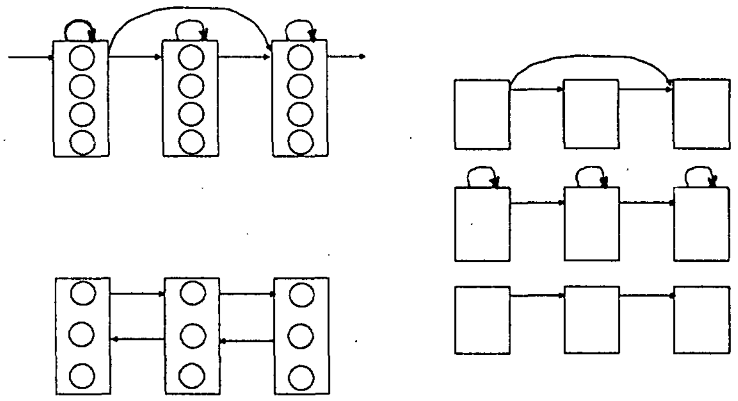


Fig. 5. Layered networks

(4) Time

A neural network can work either in continuous or discrete time. The neural systems we will present here are working in discrete time only. However, most of them have corresponding continuous variant.

(5) Rule of activity

A detailed structure of a node (neuron) is on Fig. 7. The neuron n has N inputs $a_1, a_2, \dots, a_N$, which are either outputs of some neurons or environmental inputs (stimuli). All inputs are weighted by weights $w_{1n}, w_{2n}, \dots, w_{Nn}$, summed, and led through a nonlinearity function (or threshold function) f. The common types of the threshold function are also shown on Fig. 7. A neuron usually has a threshold which symbolizes the level of the summed input signals above which the neuron is 'activated'. To simplify the formula for the neuron's output we consider its threshold as a weight w_{0n} and formally introduce the 0-th input a_0 identically equal to 1. Hence, the output (state, activity) of the neuron n is

$$x_n = f\left(\sum_{m=0}^{N} w_{mn}\, a_m \right)$$

To incorporate discrete time we should write rather

$$x_n(t+1) = f\left(\sum_{m} w_{mn}\, a_m(t) \right)$$

which indicates that the input signals at time t form the output for time t+1.

The neuron can process its own output x_n as one of its inputs, i.e. a feedback loop can be incorporated. Fig. 7 depicts this situation by means of the dotted connection with the weight w_{nn}.

(6) Learning

Learning is characterized by any change of neural net's memory w (i.e. matrix of all its weights). Methods of learning can be classified along several different dimensions. However, the presence or absence of a teacher seems to be a most important attribute. We will thus distinguish two types of learning for neural systems: supervised learning (with a teacher) and unsupervised one (without a teacher).

Supervised learning consists in that the teacher supplies so-called training set of typical exemplars (representatives, prototypes, etalons) $a_1, a_2, \dots, a_K$ with the desired output behaviour of the net. One of the simplest, but commonly used method of supervised learning is so-called *error-correction learning* (or *Widrow-Hoff*, or *delta*). A change of the weight w_{mn} between the neuron m and n is done by the formula

$$\Delta\, w_{mn} = \eta\, x_m\, (X_n - x_n)$$

where x_m is the input from the neuron m, x_n is the actual output of the neuron n, X_n is the desired output of the neuron n, provided by the teacher, and η is a parameter, called learning rate. The Rosenblatt's algorithm, discussed in Section 1, is a generalization of the above.

The simplest method of *unsupervised learning* (without a teacher) is *Hebbian learning*. A change of the weight w_{mn} between the neuron m and n is done by the formula

$$\Delta\, w_{mn} = \eta\, x_m\, x_n$$

When studying learning algorithms we should also distinguish competitive and cooperative learning. The *competitive* learning uses

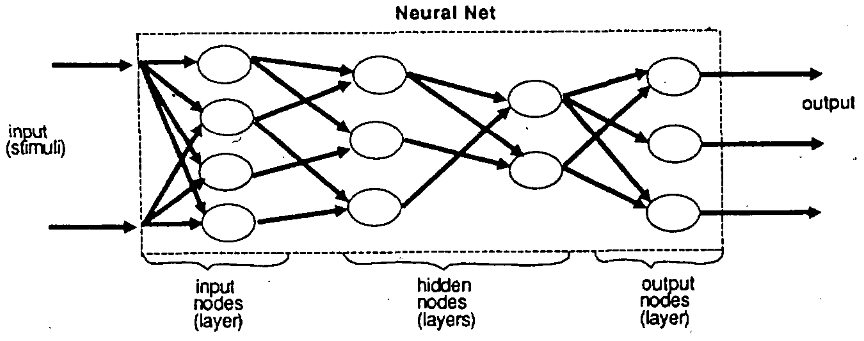
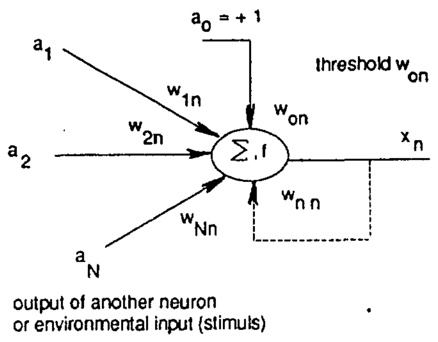


Fig. 6. A neural net in an environment



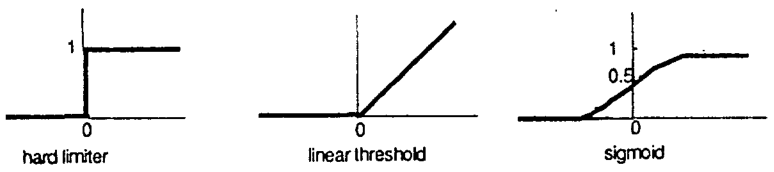


Fig. 7. Rule of activity and threshold functions

neighbour-inhibiting strategy, i.e. activities of all neighbour neurons of a given neuron are lessened, and the activity of the given neuron is increased. One simple competitive learning method is called *winner-take-all*; we will discuss it in the Section 3.5. The *cooperative* learning, on the other hand, uses neighbour-exciting strategy, i.e. activities of all neighbours including the given neuron are reinforced.

(7) Mapping mechanism

A neural network can be observed as a certain type of an associative memory. We distinguish two types of associativity:

- *autoassociative* neural net: its memory w stores input patterns (environmental stimuli) $a_1, a_2, ..., a_K$;
- *heteroassociative* neural net: its memory w stores input as well as output pattern pairs $(a_1, b_1), ..., (a_K, b_K)$.

(8) Rule for changing topology

This optional attribute grants a neural net to modify its topology, which is usually done by adding new connections between existing nodes, or even by creating a new node and its new connections to existing nodes. Carpenter-Grossberg classifier (Section 3.6) is one example of the neural net which is able to modify its topology.

3.3. Specification of neural nets

We observe from the above aspects of neural nets that there exist two (or three) interacting dynamic regimes: activities (outputs) of nodes are changed, the weights (connection strengths) are adjusted, and, optionally, the net's topology is modified. Since the weight adjustments and topology modifications are usually much slower processes than the activity changes, we can specify the above processes separately.

(1) Active regime is specified by

- the space X of state vectors $x = [\, x_1, x_2, ..., x_N \,]$, where x_n is the state (activity, output) of the neuron n , $n=1,2,...,N$;
- the activity function

  $$x(t+1) = F(x(t), w(t))$$

  which determines the state vector for the time t+1 if the state vector and the net's memory are given for the time t ;
- the energy function $E(x)$.

Its motion consists in that, for given external (environmental) input **a**, the state vector is initialized to

$x(0) = a$	for input nodes,
$x(0) = \mathbf{0}$ (or a random vector),	otherwise

and the net is trying to find a stable state $x(t^*)$ for which the energy function E reaches its minimum (so-called *stability* of the net). The over-all output (i.e. the response of the neural network to the given environmental input) is

$$b = x(t^*) \qquad \text{for output nodes.}$$

A typical active regime is called *nearest-neighbour* one: for a given unknown input pattern a , it finds the stored input pattern (exemplar, etalon) a_k which most closely matches a , see Section 3.4 for details.

(2) Training (learning) regime is specified by

- the space W of the weight matrices $w = [\, w_{mn} \,]$;
- the training function (learning algorithm)

  $$\Delta\, w = G(w,a)$$

  which determines how the weight matrix w is changed according to the training example a ;
- the error function $J(w)$.

Its motion consists in that, for a given training set of training examples $a_1, a_2,..., a_K$ and $w(0) = \mathbf{0}$ (or random), it finds an optimal weight matrix w^* for which the error function reaches its minimum (so-called *convergence* of learning).

(3) Optional *configuration regime* could be considered as a special type of the training regime. If the modification of the weight matrix is not efficient for some reasons then the configuration (topology) of the network could be optionally changed: either new connections are added or new nodes are incorporated to the existing network. Specification of such a regime is not, however, so uniform as that of the other regimes. We have to specify first of all the conditions under which the training mode has to be replaced by the configuration regime. Second, rules for adding new connections and/or nodes and their connections have to be specified as functions of existing topology and training examples.

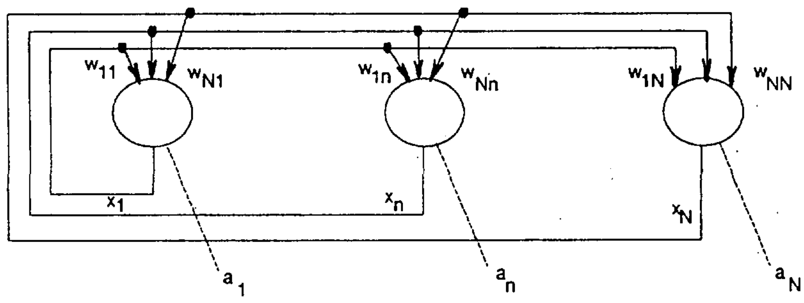


Fig. 8. Hopfield net

3.4. Hopfield net

The *Hopfield net* [Hop82] (see Fig. 8) is a single-layer, symmetric neural network, with the feedback scheme, working in discrete time. It uses the nearest neighbour active regime and Hebbian algorithm for (unsupervised) learning. It is a simple but very efficient network exhibiting the autoassociative mapping mechanism. Therefore, it is also called *autocorrelator*, or just *autoassociative memory*. Its input is binary only.

Training (learning) of the Hopfield net looks as follows. Given R training examples $\mathbf{a}_r = [\, a_{r,1}, ..., a_{r,N} \,]$, r=1,...,R, which are to be stored in the memory of the network, the weight from the node m to n is simply given by

$$w_{mn} = \sum_{r=1}^{R} a_{r,m}\, a_{r,n} \;, \quad m,n = 1,...,N \;,\; r = 1,...,R$$

However, if we prefer sequential formulas for (incremental) learning we can simply derive Hebbian learning algorithm:

$$w_{mn}(0) = 0$$
$$\Delta w_{mn} = a_{r,m}\, a_{r,n}$$

for the r-th example on the input, r=1,...,R

In the active regime, an unknown (new) pattern $\mathbf{a} = [a_1,...,a_N]$ is imposed onto the net's input at time 0 and the node states (outputs) are changed in the feedback scheme according to the following formula (Fig. 8):

$$\mathbf{x}(0) = \mathbf{a}$$
$$x_n(t+1) = f\,(\,\sum_{m=1}^{N} w_{mn}\, x_m(t)\,) \;, \quad n = 1,...,N$$

until the states (outputs) no longer change on successive iterations (i.e. stopping condition of the active regime is reached). The pattern $\mathbf{x}(t^*)$ specified by the node outputs after the stability is achieved represents the exemplar $\mathbf{a}_r$ which best matches the unknown pattern $\mathbf{x}$ (i.e. the nearest-neighbour active regime).

The energy function $E(\mathbf{x})$ used for the Hopfield net is the Lyapunov function

$$E(\mathbf{x}) = -\sum_{m,n} w_{mn}\, x_m\, x_n$$

It reflects a disharmony (chaos) of the system. Learning can be portrayed as a decrease of the energy function, i.e. 'relaxation' of the system. Stability and capacity (i.e. number of exemplars stored in the network) is discussed e.g. in [Sim90]. [Lip87] states that $R < 0.15\ N$. Here we just mention that local minima of the energy function are either the right ones (i.e. correspond to stored exemplars), or spurious ones, so-called phantoms. If a phantom φ is identified then it can be 'unlearned' by

$$\Delta\, w_{mn} = -\, b\, \varphi_m\, \varphi_n$$

where b is a positive constant.

3.5. Hamming net

The *Hamming net* (Fig. 9) is a two-layer neural network, with the feedback scheme, working in discrete time. Its input is binary only. Learning regime of the Hamming net is very simple, because it is in fact done by these assignment statements for the weights of the first layer:

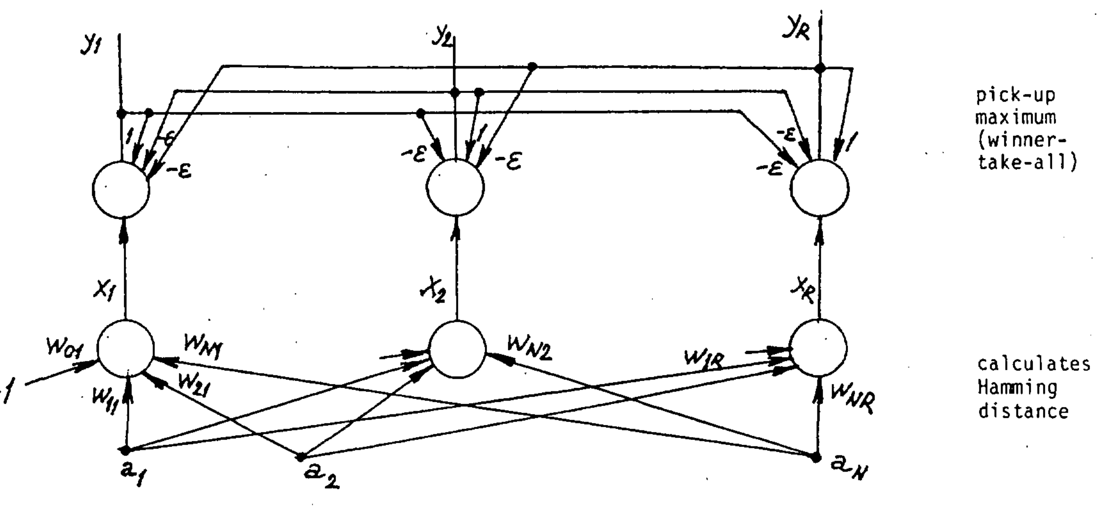


Fig. 9. Hamming net

$$w_{nr} = \frac{1}{2} a_{r,n} \quad , \quad r = 1,\dots,R \ , \ n = 1,\dots,N$$
$$w_{0r} = \frac{1}{2} N$$

where $a_r = [\ a_{r,1} ,\dots, a_{r,N}\ ]$, r=1,...,R, are training examples. The second layer has predefined weights, see Fig. 9.

In the active regime, an unknown pattern $a = [\ a_1 ,\dots, a_N\ ]$ is imposed onto the net's input layer and is propagated through the first layer. The outputs of the first-layer nodes are (see Fig. 9):

$$x_r = f\,(\ \sum_{n=0}^{N} w_{nr}\, a_n\ ) \quad , \quad r = 1,\dots,R$$

where f is the linear-threshold nonlinearity. The first layer in fact calculates *Hamming distance* for each training example stored in the network, which is defined as a number of (binary) features of the input pattern **a** which do not match the corresponding components (features) of the stored example $\mathbf{a}_r$.

The second layer of the Hamming net depicts the *winner-take-all* technique which picks up the maximum value among its inputs $x_1 ,\dots, x_R$. This is done in a feedback regime according to the following iterative formulas:

$$y_r(0) = x_r \quad , \quad r = 1,\dots,R$$
$$y_r(t+1) = f\,(\ y_r(t) - \varepsilon \sum_{s \neq r} y_s(t)\ )$$

where $0 < \varepsilon < 1/R$ is a priori given constant. The above formula is executed until the outputs y_r no longer change. At that time only one output is positive and the others are zero. This positive output indicates the 'winner'; it is the training example which is the closest to the input pattern a , i.e. whose Hamming distance to the input pattern is minimum. If the Hamming net is used for classification then the input pattern will be classified to the class of the winner. Other attributes are discussed e.g. in [Lip87].

3.6. Carpenter-Grossberg classifier

The *Carpenter-Grossberg classifier* [Car86] (Fig. 10) is a two-layer neural network, with the feedback regime, working in a discrete time. It uses the nearest neighbour active regime and competitive unsupervised learning. Its input is binary only. This net achieves a clustering algorithm which is similar to the traditional sequential leader clustering algorithms (see e.g. [Har75]).

As we can see on Fig. 10, the network consists of both 'bottom-up' connections with weights w_{nr} and 'top-down' connections with weights w'_{rn} . The net of 'bottom-up' connections is equivalent to the Hamming net (section 3.5), the 'top-down' connections propagate so-called matching exemplar to the input nodes where it is compared with the input pattern. In the following we will give a flow-chart of the entire procedure. Let there be N input nodes and R output nodes. Assume the all weights have been adjusted according to previous learning sweeps. Let a new input pattern be $a = [\ a_1 ,\dots, a_N\ ]$. Then its processing by the Carpenter-Grossberg classifier is outlined as follows:

1. Present an unknown input pattern $a = [\ a_1 ,\dots, a_N\ ]$ to the input nodes. Let the set of allowable winners comprise all output nodes.

2. Send signals through w_{nr} to these output nodes y_r which are members of the set of allowable winners.

3. Find y_s with the maximum value among y_r , r=1,...,R, using the winner-take-all strategy (equivalent to the Hamming net).

4. The winning output node y_s sends the top-down signal along the weights w'_{sn} back to the input nodes, forming the matching exemplar $a' = [\ w'_{s1}\, a_1 ,\dots, w'_{sN}\, a_N\ ]$.

5. The input pattern is compared with the matching exemplar by so-called *vigilance*

$$\rho = \frac{|a|}{|a'|}$$

where the norm |a| of the pattern a equals to the sum of its components (features) a_n .

5.1. If $\rho \geq \rho_{max}$, where ρ_{max} is a vigilance threshold (a number between 0 and 1), then the winning node y_s represents the proper class of the given input pattern a , and a is *merged* into this class, i.e. all w_{ns} and w'_{sn} of the winning matching exemplar are modified:

$$w'_{sn}(t+1) = w'_{sn}(t)\ a_n$$
$$w_{ns}(t+1) = \frac{w'_{sn}(t)\ a_n}{0.5 + |a'|}$$

and the learning sweep for the given input pattern is terminated.

5.2. However, if $\rho < \rho_{max}$ then the winning node y_s does not represent the proper class of the input pattern a , and y_s is removed from the set of allowable winners. If there are still some allowable winners, go to Step 2. Otherwise the pattern a forms a new class, i.e. a new node y_{R+1} is created and a is encoded to it.

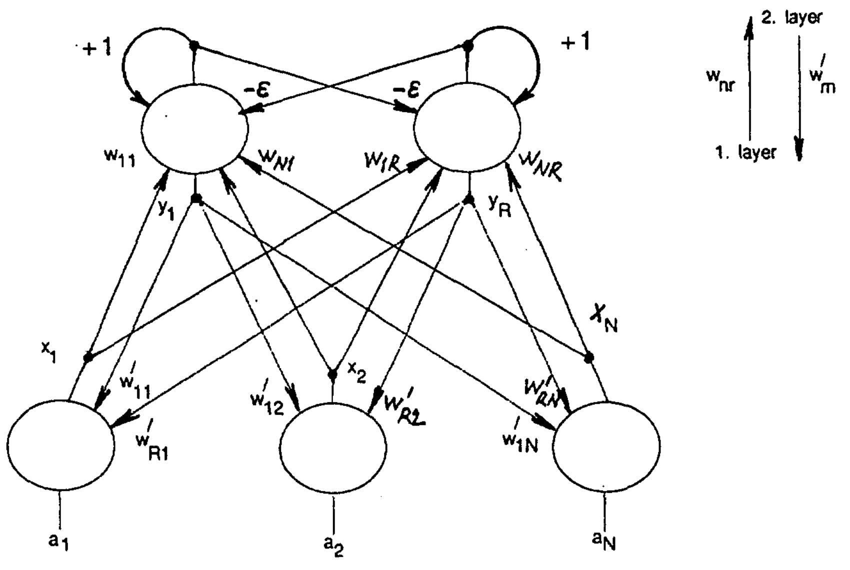


Fig. 10. Carpenter-Grossberg classifier

With no noise, the vigilance threshold can be set such that two patterns which are most similar are considered different. In noise, however, this level may be too high and the number of new stored exemplars can rapidly grow. One possibility is changing vigilance threshold during training, see e.g. [Car86].

3.7. Kohonen's self-organizing feature maps

The *Kohonen's self-organizing feature map* [Koh84] is a one-layer neural network with the feedforward scheme, working in discrete time. It exploits the nearest-neighbour active regime and cooperative learning. Its behaviour is similar to K-means clustering algorithm. It incorporates a kind of space topology (Fig. 11), since all its nodes are organized into a two-dimensional array. Particularly, neighbourhood of a node is taking into consideration.

Training (learning) of Kohonen's feature map looks as follows. Let there be R nodes in the two-dimensional topology, and each pattern have N features. For a training example a = $[a_1,...,a_N]$ the following distances between a and the weight vectors w_r for each node are computed

$$d_r = | \mathbf{a} - \mathbf{w}_r |^2 = \sum_{n=1}^{N} (a_n - w_{nr})^2 \quad , \quad r = 1,...,R$$

Then the output s with minimum distance is selected among r=1,...,R . Afterwards, the weights of this winner and its neighbours are updated:

$$w_{nr}(t+1) = w_{nr}(t) + \eta(t)\,(a_n - w_{nr}(t)) \qquad \text{for } r \in N_s(t)$$

where $N_s(t)$ is the neighbourhood of the node s at time t , $\eta(t)$ is the learning rate at time t . The other weights are not modified. The important fact is that both the size of any neighbourhood and the learning rate decrease in time. E.g. if the node s is a winner for the time t then the weights of all 25 nodes in its neighbourhood (including the winner itself) are modified at the time t , only 9 nodes at the time t+1 , and just the winner itself at time t+2 , see Fig. 11.

After enough training patterns have been presented, the weights will specify cluster or pattern (vector) centers that sample the feature space such that the density function of the pattern centers tends to approximate the probability density function of the training patterns. The weights will be organized such that topologically close nodes are sensitive to the input patterns that are physically similar. The algorithm performs relatively well in noise. Other aspects as well as areas of applications can be found in [Sim90].

3.8. Boltzmann machine

The *Boltzmann machine* [Hin86] is a two-layer network with feedforward scheme and binary inputs, working in discrete time. It exploits the nearest-neighbour active regime and a combination of Hebbian learning and stochastic learning.

The entire principle of the Boltzmann machine is described in detail e.g. in [Sim90]. Here we highlight the idea of *stochastic learning* only. A learning sweep looks as follows:

1. Add a new training example to the network by applying the Hebbian learning formula.
2. Make a random weight change.
3. Determine the change of energy function ΔE after the weight was randomly changed.

4a. If $\Delta E < 0$ then keep the change.

4b. If $\Delta E > 0$ select a random number p and calculate
$$P = e^{-\Delta E/T(t)}$$
where T(t) is the *temperature* of the Boltzmann process. If p < P then accept the weight change; otherwise return to the original value.

We can observe that larger T(t) causes the random weight change would be accepted more likely. The temperature T(t) decreases in time according to

$$T(t) = T_0 \,/\, (1 + \log t)$$

The authors of this method have proven that the random character of the weight change allows to escape local energy minima and reach thus the absolute minimum of the energy function.

3.9. Multilayer perceptron

As we have already mentioned, a linear classifier, i.e. Rosenblatt's perceptron without its Φ-processor (or: a *single-layer perceptron*, in the neural net terminology) can only create linear decision boundaries (Fig. 12a). A *two-layer* perceptron can form convex decision boundaries (see Fig. 12b) while a *three-layer* one can generate boundaries of any shape (Fig. 12c) [Lip87]. This means that no more than three layers are required because a three-layer perceptron can generate arbitrarily complex decision regions.

The structure of a three-layer perceptron is shown in Fig. 12c; it has two hidden layers and one output layer. A two-layer

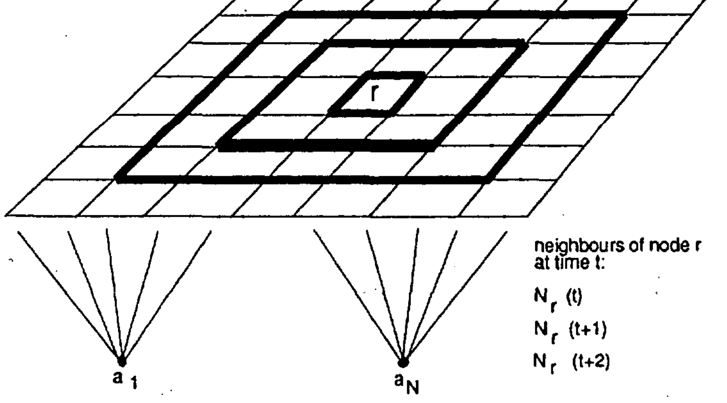


Fig. 11. Kohonen's self-organizing feature map

perceptron has only one hidden layer and, of course, one output layer (Fig. 12b). Thanks to the hidden layer(s), the multilayer perceptrons overcome many limitations found in a single-layer perceptron. However, they were not generally used in the past because effective learning algorithms that would adjust their weights in an optimal way were not available. Three years ago, the so-called back propagation learning algorithm for multilayer perceptrons was discovered [Ru86b]. Although it cannot be proven that this learning algorithm generally converges (as with the single-layer perceptron), it has been proven to be useful and efficient for many pattern recognition problems.

Since the structure and the back propagation learning algorithm are almost identical for two- and three-layer perceptron, we will only describe the structure and behaviour of the latter. An input pattern of a multilayer perceptron is represented by a N-dimensional feature vector

$$x = [\, x_1, \ldots, x_N \,]$$

where x_n , $n = 1,\ldots,N$ are features, i.e. numbers in the range 0 to 1 .

The feature vector is input to each of M nodes of the first hidden layer so that the linear weighted sum of all features is computed and the sigmoid nonlinearity

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}}$$

converts it to an output x'_m , $m = 1,\ldots,M$, that is thus between 0 and 1 . Hence (see Fig. 12c)

$$x'_m = f\Big(\sum_{n=0}^{N} w^{(1)}_{nm}\, x_n \Big) \;, \quad m = 1,\ldots,M \qquad (5)$$

where f is the above nonlinearity, $w^{(1)}_{nm}$ is the weight connecting the n-th input to the m-th node of the first hidden layer, $x_0 = 1$ allows the weight $w^{(1)}_{0m}$ to be considered as a threshold of the m-th hidden node.

In the same way, the second hidden layer has H nodes and the output x''_h of the h-th node of the second hidden layer is

$$x''_h = f\Big(\sum_{m=0}^{M} w^{(2)}_{mh}\, x'_m \Big) \;, \quad h = 1,\ldots,H \qquad (6)$$

where $w^{(2)}_{mh}$ is the weight connecting the m-th node of the first hidden layer to the h-th node of the second hidden layer, $x'_0 = 1$ processes the weights $w^{(2)}_{0h}$ as thresholds.

Similarly, the output layer has R nodes and the outputs x''_h of the second hidden layer are linearly weighted and the sum is converted by the sigmoid nonlinearity. The output signal of the r-th output node is thus

$$y_r = f\Big(\sum_{h=0}^{H} w^{(3)}_{hr}\, x''_h \Big) \;, \quad r = 1,\ldots,R \qquad (7)$$

where $w^{(3)}_{hr}$ is the weight connecting the h-th node of the second hidden layer to the r-th output node, $x''_0 = 1$ processes the weights $w^{(3)}_{0r}$ as thresholds.

The multilayer perceptron, when used for classification, can classify input patterns to R classes z_r , $r = 1,\ldots,R$. The discriminant functions are given by formulas (7) and the decision rule of the classifier is equivalent to (1): the pattern (more precisely, its feature vector) x is classified to the class z_s iff the output y_s is the maximum among y_r , $r = 1,\ldots,R$.

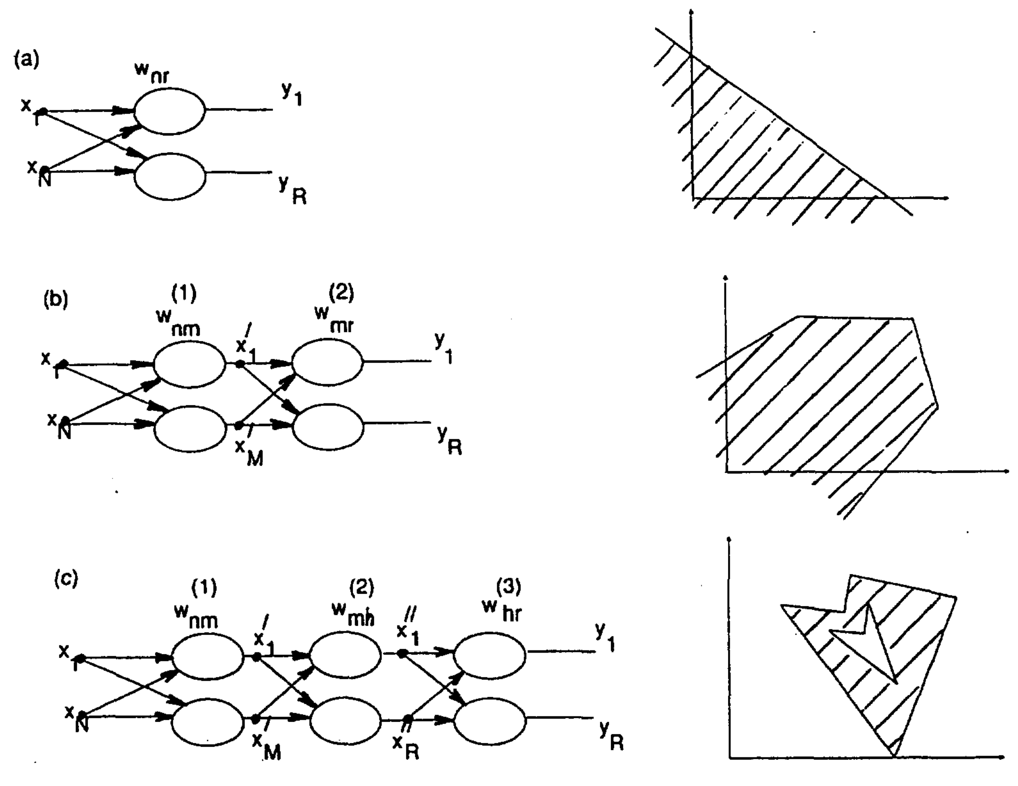


Fig. 12. Structure and shape of decision regions for (a) single-layer perceptron, (b) two-layer perceptron, (c) three-layer perceptron. The arrows depict weights, a circle represents the sum including nonlinearity

Since we want the multilayer perceptron to have the optimal classification performance we have to adjust its weights according to a criterion. As in case of a single-layer perceptron, one possible and commonly used way of such an optimal adjustment is to use a learning (training) process. As we have already stated one of the popular learning algorithm for multilayer perceptrons is the *back propagation training algorithm.* It propagates an output error signal back through the network and modifies its weights accordingly.

The *error signal* is defined as

$$\text{eps} = \sum_{r=1}^{R} (y_r - Y_r)^2 \tag{8}$$

where Y_r is the desired output of the r-th output node.

For the purposes of classification, if a training pattern x belongs to the desired class $Z = z_s$ then

$$\begin{aligned} Y_s &= 1 \\ Y_r &= 0 \quad \text{for} \quad r \neq s \end{aligned} \tag{9}$$

Hence, the error signal for the training pattern x of the (desired) class $Z = z_s$ is

$$\text{eps} = \sum_{r \neq s} y_r^2 + (1 - y_s)^2 \tag{10}$$

Training patterns are presented several times to the training algorithm until the error signal for all training patterns is less than an a priori given maximum epsmax . This maximum must be below 0.5 , since it is the error signal value of the worst case: if e.g. a training pattern belongs to $Z = z_1$ and if

$$y_1 = 0.5 \ , \ y_2 = 0.5 \ , \ y_r = 0 \ \text{ for } \ r \neq 1, 2$$

then

$$\text{eps} = 0.5^2 + (1 - 0.5)^2 = 0.5$$

The flow chart of the back propagation algorithm is given below (written in the same fashion as Rosenblatt's learning algorithm):

1. Initialize all weights to random values in the range -Winit to +Winit where Winit is a parameter of the learning algorithm that shrinks or expands the range of the initial values of weights.
2. Set Number_of_learning_sweeps to 0 .
3. For each training pattern $x = [x_1 ,..., x_N]$ with its desired class $Z = z_s$ do:
   - 3.1. Compute y_r , $r = 1,...,R$ for the given pattern according to (5) to (7).
   - 3.2. Compute the error signal eps according to (8) and (9).
   - 3.3. If eps > epsmax (a given maximum) then modify the weights as follows: compute

     $$\delta_r^{(3)} = y_r (1 - y_r) (Y_r - y_r) \ , \ r = 1,...,R \tag{11}$$

     and change the weights of the output layer:

     $$w_{hr}^{(3)} \mathrel{+}= \eta \ \delta_r^{(3)} \ x''_h \ , \ h = 0,1,...,H \ , \ r = 1,...,R \tag{12}$$

     where η is the *learning rate* (*gain term*), another parameter of the learning algorithm.

     Similarly, change the weights of the second hidden layer:

     $$\delta_h^{(2)} = x''_h (1-x''_h) \sum_{r=1}^{R} \delta_r^{(3)} w_{hr}^{(3)} \tag{13}$$

     $$w_{mh}^{(2)} \mathrel{+}= \eta \ \delta_h^{(2)} \ x'_m \ , \ m = 0,1,...,M \ , \ h = 1,...,H \tag{14}$$

     and those of the first hidden layer:

     $$\delta_m^{(1)} = x'_m (1-x'_m) \sum_{h=1}^{H} \delta_h^{(2)} w_{mh}^{(2)} \tag{15}$$

     $$w_{nm}^{(1)} \mathrel{+}= \eta \ \delta_m^{(1)} \ x_n \ , \ n = 0,1,...,N \ , \ m = 1,...,M \tag{16}$$
4. If we have modified the weights for at least one training pattern, increment Number_of_learning_sweeps by 1 and return to the step 3, i.e. repeat the training for the entire training set.
   If not, save all weights, print Number_of_learning_sweeps and terminate the learning process.

Convergence is sometimes faster and the danger of oscillation is diminished if a so-called *momentum term* α is added to the formulas for changing weights [Lip87].

The two-layer perceptron has just one hidden layer with weights $w_{nm}^{(1)}$ and the output layer with weights $w_{mr}^{(2)}$. Otherwise, the structure is identical to the three-layer perceptron, and the formulas (11) to (16) have to be slightly changed.

4. Application: waveform processing

4.1. Introduction

Multilayer perceptrons have found many useful and efficient applications in pattern recognition. Our research group is using a multilayer perceptron as one component of a larger decision-supporting system for neurological diagnoses. The actual input to our system is an evoked potential waveform. It is processed by these subsystems [Bru88], [Bru89] (see Fig. 13 for illustration):

(1) Filter preprocessing. The evoked potential waveform is preprocessed by a digital filter. This eliminates noise and allows the use of simple recognition grammars in the syntax analysis stage.

(2) Extracting a string of symbols. The potential waveform is segmented and each segment is described by one (terminal) symbol. Thus, the entire input waveform is formally described by a string of (terminal) symbols [Mad86].

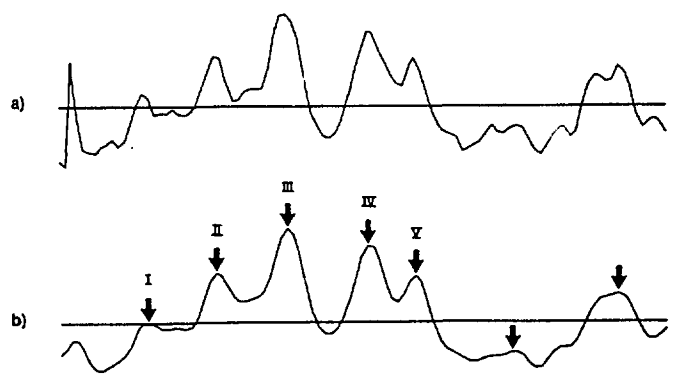
(3) Syntax analysis. An attributed regular grammar with semantic functions [Fu82] is used for the syntactic analysis of an input waveform represented by a string of symbols. It recognizes the start of the relevant waveform as well as each peak of the waveform. It returns through its semantic functions the latency of the beginning of the relevant waveform, number of hills, and the peak latency of each hill found (see [Bru88] for details).

(4) Numerical classification. The features extracted by the attributed grammar are further processed by a two-layer perceptron that classifies the given input waveforms (using the above features) into two classes: normal and abnormal. Note the semantic functions of our attributed grammar form an interface between strictly syntax subsystem (a grammar) and strictly numerical processing (a multilayer perceptron).

(5) The knowledge-based subsystem. We analyze methods of incorporating a knowledge-based subsystem into our decision-supporting system in order to obtain more reliable results. The need for such a knowledge-intensive device is discussed in the conclusion of this paper.

After analyzing a large number of experiments that have been conducted to obtain the performance characteristics of a multilayer perceptron, we have found out that the three-layer perceptron is quite unstable for this recognition task. Therefore, we have focused on a two-layer perceptron. Similar conclusion can be found in e.g. [Wie87].

The training method used in our experiments is the back propagation training algorithm. By analyzing the above algorithm one can easily discover that the following five parameters strongly affect both learning and classification performance of the multilayer perceptron:



c) $x^{10}u^{4}$ $f^{1}d^{2}f^{3}d^{1}u^{5}d^{4}f^{1}u^{7}$ $d^{6}f^{1}u^{6}f^{1}d^{4}u^{3}d^{9}$ $f^{1}u^{2}f^{2}u^{2}f^{1}d^{3}f^{1}u^{3}f^{2}u^{6}$ $d^{5}f^{1}u^{2}$

d) 11 14.5 27 39 46.5 54 70.5 89

Fig. 13. Waveform processing: (a) evoked potential waveform, (b) filtered data, (c) string of symbols (*u* upward slope, *d* downward slope, *f* flat line, *x* irrelevant data), (d) list of features

- the range of initial weights Winit ,
- the learning rate (gain term) eta ,
- the momentum term alpha ,
- the maximum allowed value of the error signal eps ,
- the number of hidden nodes M .

However, there is no formal method for obtaining optimal values of the above parameters. Therefore, we have carried out a large number of experiments, analyzed them, and obtained a few heuristics (thumb-rules) for optimal adjustments of the above parameters. The following sections discuss both experiments and the heuristics obtained.

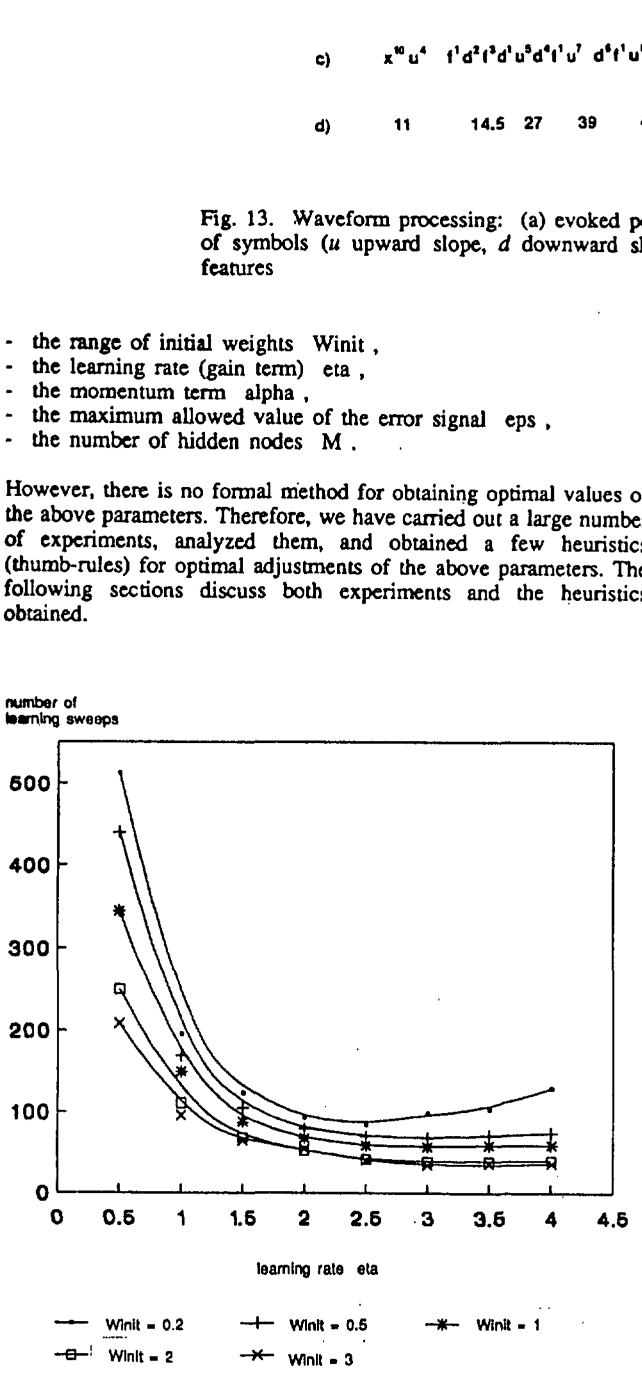


Fig. 14. Number of learning sweeps versus the learning rate (gain term) eta and the initial weight range Winit (M = 9 , eps = 0.30 , alpha = 0)

4.2. Optimal adjustment of parameters: experiments

A two-layer perceptron emulator used in the experiments has been implemented in C because of its portability and flexibility. The emulator has been tested under BSD 4.3, SunOS, and DOS using MS-DOS Microsoft C and Turbo C.

We have used a training set of 50 evoked potentials (training patterns) of normals (class z_1) and 50 potentials of abnormals (class z_2) for training the two-layer perceptron. Additional set of 40 patterns with unknown classes has been used for testing. The number of features (inputs) has always been $N = 4$, and the number of classes $R = 2$. We have chosen the maximum error signal eps = 0.3 for most experiments since it corresponds to the reasonable difference of 0.4 between the actual output (y_r) and the desired one (Y_r) which has been accepted in many experiments (see e.g. [Bur88], [Bru89]); if a training pattern belongs to the class say z_1 then we allow

$$y_1 = 0.6 \quad , \quad y_2 = 0.4$$

thus approximately

$$\text{eps} = (1 - 0.6)^2 + 0.4^2 = 0.3$$

We have executed a large number of experiments for various sets of parameters. The efficiency of the learning algorithm has been symbolized by the following factors [Bur88]:

1. the number of learning sweeps,
2. the accuracy in classifying unknown (testing) patterns, measured as percentage of correct classifications.

Following the Kolmogorov's mapping theorem and its interpretation to the neural nets [He87], we have set the number of hidden nodes to

$$M = 2 * N + 1 = 9$$

Afterwards, we have carried out four sets of experiments, running the back propagation algorithm for the given set of 100 training patterns (50 normals and 50 abnormals). To achieve comparable statistical results, we have made 200 runs for each configuration.

The first set of our experiments has been run to find optimal values of the learning rate eta . Therefore, we have run the back propagation algorithm for the two-layer perceptron with eps = 0.30, M = 9 , alpha = 0 , for various learning rates (gain terms) eta (from 0.5 to 4.0) and various initial weight range Winit (from 0.2 to 3.0). The results are in Fig. 14. We observe that the learning algorithm has required *larger* number of learning sweeps for *smaller* values of eta , since less information is learned during each sweep. The performance has not changed substantially for learning rate eta between 2.5 and 4.0 . It is also seen that *the larger* the range of initial weights, *the faster* the training is completed.

Therefore, we have chosen Winit = 3.0 and eta = 2.5, 3.0, 4.0 as promising values for the second set of experiments that were to reveal an optimal number of hidden nodes M . Again, eps = 0.30 , alpha = 0 , but the number of hidden nodes has been changed from 4 to 25 . The results are in Fig. 15a and 15b. The figures indicate that the number M = 9 chosen according to Kolmogorov's mapping theorem yields the optimal performance for the number of learning sweeps only. On the other hand, the accuracy of classifying unknown patterns (evoked potential waveforms) is satisfactory for M > 9 . Therefore, we conclude that the *number obtained by Kolmogorov's theorem* should be considered as the *lower bound* of the recommended number of hidden nodes.

The objective of the third set of experiments has been to confirm the previous two tests that Winit equal 3.0 was an optimal parameter. We have set eps = 0.30 , eta = 4.0 , alpha = 0 , M has been changed from 7 to 16 . The results are presented in Fig. 16a and 16b; they confirm the predicted optimal value for Winit as well as the above thumb-rule for M .

The momentum term alpha may speed up the convergence of the back propagation learning algorithm. It is expected to smooth the weight changes during the learning. Therefore, we have run the last set of experiments in order to observe the effect of the momentum term to the perceptron's performance. The results are in Fig. 17a and 17b. Fig. 17a indicates that as the value of the momentum factor *increases*, the optimal value for the learning rate (gain term) *decreases*. However, the accuracy of classification *decreases* with *larger* momentum factor (Fig. 17b).

We have run the same set of experiments for a three-layer perceptron for the same learning rates, initial weights range, and maximum error signal. As for the number M of nodes in the first hidden layer, we have again incorporated Kolmogorov's theorem, and have followed the advice [6] that there should be more than three times as many nodes in the second hidden layer as in the first one, i.e. H >= 3 * M . However, we have discovered that the back propagation learning algorithm does not converge over a great number of runs, and the weights oscillate, especially in the second hidden layer.

4.3. Optimal Adjustment of Parameters: Heuristics

This section discusses the heuristics (thumb-rules) for optimal adjustment of multilayer perceptron's parameters we have observed after analyzing the results of our experiments.

- The initial weight range Winit should be a larger value, greater than 1.0 . Our recommendation is Winit = 3.0 .
- The learning rate (gain term) eta should be a larger value, greater than 1.0 . Intuitively, a larger momentum term together with a larger learning rate would speed up the learning and avoid the oscillation. However, our experiments have revealed that the optimal learning rate for minimum number of learning sweeps becomes *smaller* for *larger* momentum factor. Our recommendation is to choose eta between 1.0 and 4.0 and alpha = 0 as a starting point. If the learning does not converge, one should increase alpha and decrease eta .
- The number obtained by Kolmogorov's theorem should be considered as the lower bound of the recommended number of hidden nodes. However, [Guy89] introduces another thumb-rule for an optimal number of hidden nodes:

M = (N + R) / 2

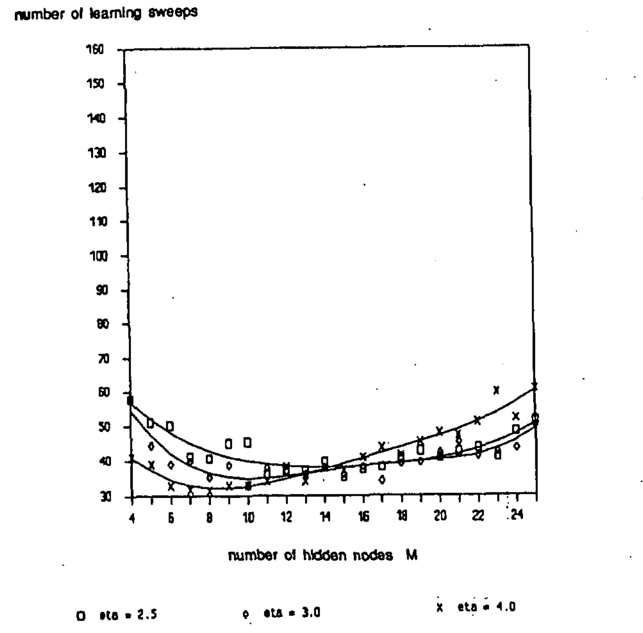


Fig. 15a. Number of learning sweeps versus number of hidden nodes M (eta = 2.5, 3.0, 4.0 , Winit = 3 , eps = 0.30 , alpha = 0)

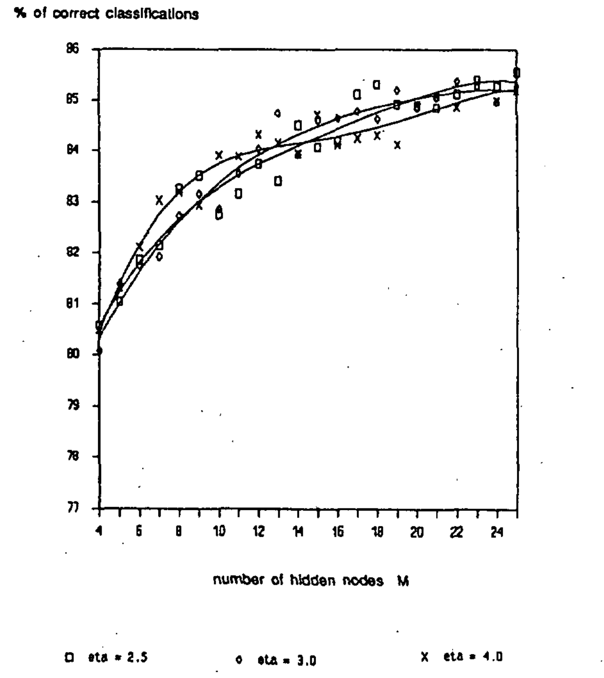


Fig. 15b. Percent correct recognitions versus number of hidden nodes M (eta = 2.5, 3.0, 4.0 , Winit = 3 , eps = 0.30 , alpha = 0)

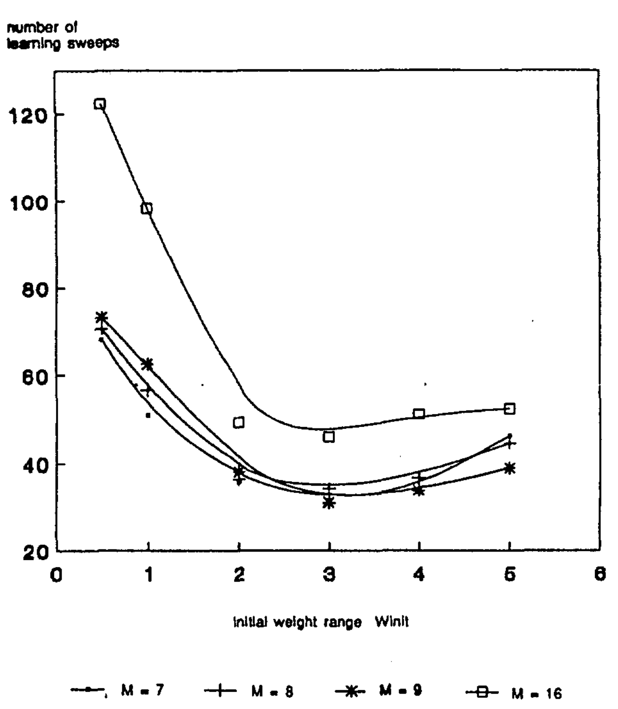


Fig. 16a. Number of learning sweeps versus initial weight range Winit and the number of hidden nodes M (eta = 4.0 , eps = 0.30 , alpha = 0)

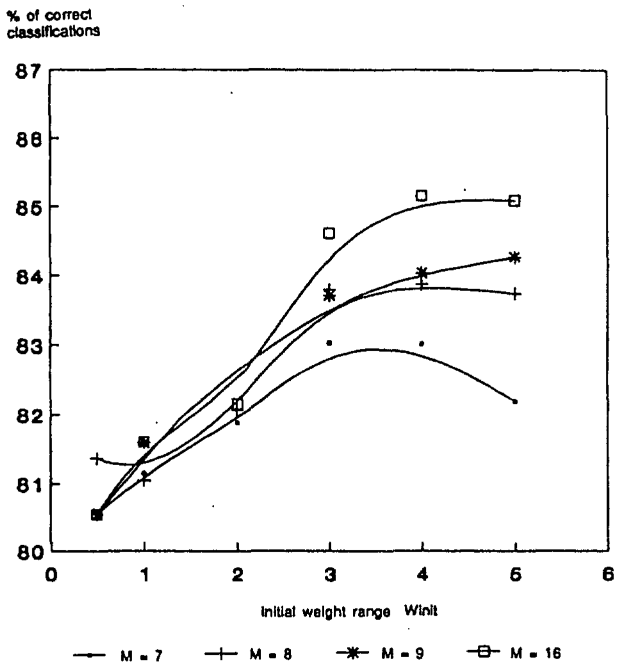


Fig. 16b. Percent correct recognitions versus initial weight range Winit and the number of hidden nodes M (eta = 4.0 , eps = 0.30 , alpha = 0)

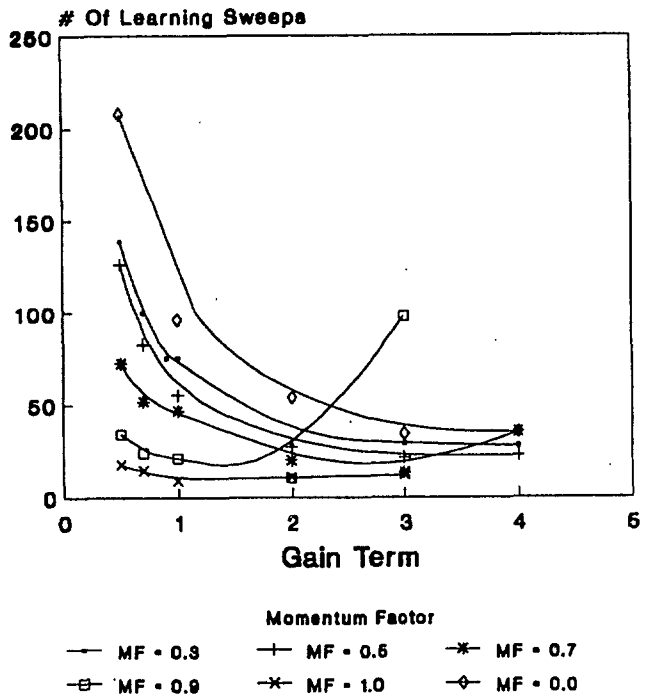


Fig. 17a. Number of learning sweeps versus learning rate (gain term) eta and momentum factor alpha (M = 9 , Winit = 3 , eps = 0.30)

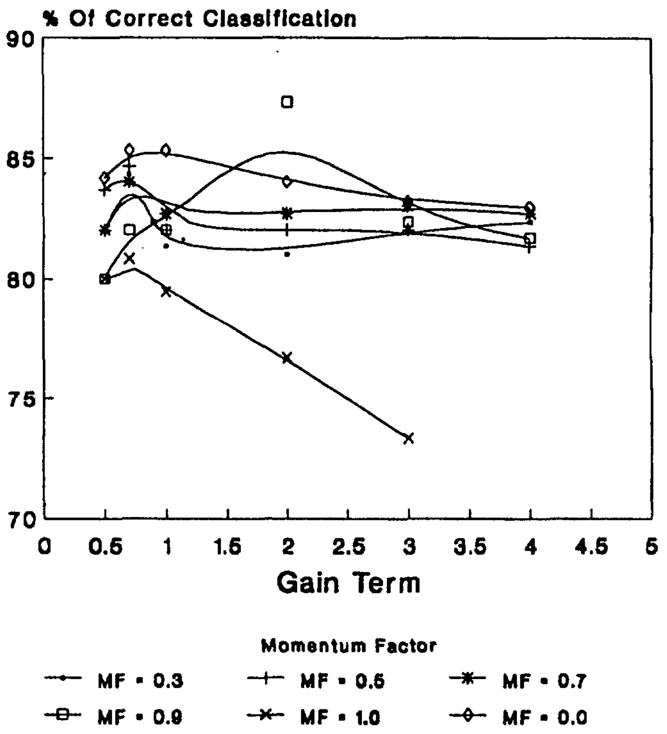


Fig. 17b. Percent correct recognitions versus learning rate (gain term) eta and momentum factor alpha (M = 9 , Winit = 3 , eps = 0.30)

where N is the number of features (input nodes) of the perceptron, R is the number of output nodes (i.e. classes in a classification problem). Our recommendation is to try Kolmogorov's mapping theorem for smaller number (units) of features, and the above formula for larger number (tens) of features[1].

Let us summarize the entire procedure of finding optimal values of the parameters:

(1) The user of a multilayer perceptron should firstly select a relatively small subset of representative training examples (patterns). The error signal should be set to a larger value (less than 0.5), say eps = 0.49 . The first heuristic should be applied for the initial weight range.

(2) Following the second and the third heuristics, the user should run a few experiments (using the above small representative set of training patterns) in order to find the optimal values of eta , alpha and M .

(3) As our experiments have revealed, the above optimal values can be afterwards used for the entire training set, with the error signal set to e.g. eps = 0.30 , which will guarantee more precise results of learning.

5. Conclusion: the need for a knowledge-based subsystem

This paper surveys the fundamental tools for pattern recognition: perceptrons and neural nets. We have exhibited that neural nets are powerful tools for image, speech, and waveform processing, as well as general recognition systems. The greatest potential of neural nets is in the high-speed processing that can be achieved by means of parallel VLSI implementations. Massively parallel hardware is evidently one of the reasons why so many researchers are investigating this field so broadly at present. Current research is aimed at analyzing new types of neural net structures and their learning (or self-learning) algorithms. Thus after twenty years the enthusiasm of artificial intelligence research and development has returned to connectionism. Nevertheless, we should bear in mind that the marvellous powers of the brain emerge not from any single, uniformly structured connectionist network but from highly evolved arrangements of smaller, specialized networks which are interconnected in very specific ways [Min88].

Furthermore, the paper introduces our decision-supporting system for neurological diagnoses comprised of several subsystems, with all but last one having been developed, implemented, and thoroughly tested. Using attributed grammars in our recognition system seems to be quite adequate because they provide a simple way of expressing semantic results of the syntactic recognition. The features extracted by the attributed grammar are further processed by a multilayer perceptron. A large number of experiments have revealed that the three-layer perceptron is quite unstable. Therefore, we have focused on the two-layer perceptron, and indicated four parameters of the learning algorithm that have to be adjusted by the user. We have found that the ranges of these parameters depend on the classification problem. Thumb-rules have been derived from these results.

After considerable experience with combined syntax and neural net classification, we have found that 'well-behaved' waveform processing reaches acceptable rates of correct classification. By 'well-behaved', we mean waveforms whose peaks are identifiable after the initial filter processing. However, there are quite a few clinical cases which are not 'well-behaved'. For example, the even peaks of waveforms may be difficult to detect even in some normal subjects [Chi83]. Therefore, we have decided to incorporate a knowledge-based subsystem to our decision-supporting system, since our present combined syntax and neural net system would not be able to handle such situations in an elegant way. The knowledge-based subsystem will be able:

(a) to handle 'non-well-behaved' data as mentioned above,
(b) to combine different modalities of evoked potential waveforms (to improve its behaviour the system will have to process and analyze three or more separate types of evoked potential waveforms),
(c) to incorporate other clinical information (for a complete diagnostic system, rules for incorporating other patient related factors such as age, sex, etc. will have to be developed).

There are three possible ways of incorporating a knowledge-based subsystem into our decision-supporting system:

(i) A knowledge-based system as a 'high-level' processing subsystem can be placed at the end of the entire evoked potential processing. In this traditional case, the knowledge-based subsystem does not influence the 'low-level' processing at all. However, this configuration fails if the feature list represents a 'non-well-behaved' situation (pattern) which would be processed by the neural net in an improper fashion and should, therefore, be recognized in advance by the knowledge-based subsystem.

(ii) A knowledge-based subsystem can be placed before the neural net, recognize 'non-well-behaved' cases (patterns) and process them separately; the 'well-behaved' cases could be directly processed by the neural net. We are going to investigate this promising configuration thoroughly but its success will depend on the knowledge acquisition, i.e. co-operation with human experts in the field of neurology. As for the knowledge representation we are going to use a rule-based model with uncertainties, the expert system environment called McESE [Fr89].

(iii) A knowledge-based subsystem can co-operate with the neural net so that it can call the multilayer perceptron and continue the processing according to its results. This feed-back loop configuration seems to be the most promising model. However, we cannot directly use the conventional representation of knowledge as a set of production rules, which is the case in the model (ii), since the rule-based system has to modify its inference process according to the results of the neural net.

References

[And72] J. Anderson, "A simple neural network generating an interactive memory", Math. Biosciences, 14, pp. 197-220, 1972

[Bru80] I. Bruha, J. Jelinek, Z. Kotek, Adaptive and Learning Systems, SNTL Prague, 1980 (In Czech)

[Bru88] I. Bruha and G.P. Madhavan, "Use of attributed grammars for pattern recognition of evoked potentials", IEEE Trans. System, Man and Cyber., Dec. 1988

[Bru89] I. Bruha and G.P. Madhavan, "Combined syntax - neural net method for pattern recognition of evoked potentials", Proc. International Conference Computing and Information, North Holland, May 1989

[Bur88] D.J. Burr, "Experiments on neural net recognition of spoken and written text", IEEE Trans. Acoustics Speech Signal Proc., Vol. 36, No. 7, pp. 1162-8, July 1988

[Car86] G.A. Carpenter and S. Grossberg, "Neural dynamics of category learning and recognition: attention, memory consolidation, and amnesia", in: J. Davis, R. Newburgh, and E. Wegman (eds.), Brain Structure, Learning, and Memory, 1986

[Chi83] K.H. Chiappa, "Evoked potentials in clinical medicine", Raven Press, New York, 1983

[Du72] R.O. Duda and P.E. Hart, "Use of Hough transformation to detect lines and curves in pictures", Communications of the ACM, 15, 1972

[Du73] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973

[1] based on results of experiments done with another set of real data [Ho89]

[Fr89] F. Franek and I. Bruha, "McESE - McMaster Expert System Environment", International Conference Computing and Information, North Holland, May 1989

[Fu82] K.S. Fu, "Syntactic Pattern Recognition and Applications", Prentice-Hall, New Jersey, 1982

[Gro78] S. Grossberg, "Adaptive pattern classification and universal recording: parallel development and coding of neural feature detectors", 3rd European Conference Cybernetics and Systems Research, Halstead Press, 1978

[Gro88] S. Grossberg, "Nonlinear neural networks: principles, mechanisms, and architectures", Neural Networks, Vol. 1, pp. 17-61, 1988

[Guy89] I. Guyon, "Neural Network Systems", Proc. of INME Symposium, Lausanne, Sept. 1989

[Har75] J.A. Hartigan, Clustering Algorithms. John Wiley, New York, 1975

[He87] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem", IEEE International Conference Neural Networks, San Diego, 1987

[He88] R. Hecht-Nielson, "Applications of counterpropagation networks", Neural Networks, 1, pp. 131-140, 1988

[Hin86] G.E. Hinton and T.J. Sejnowski, "Learning and relearning in Boltzmann machines", in [Ru86a]

[Ho89] R. Ho, "A Neural Network System for Recognition of Evoked Potentials", M.Sc. Thesis, Dept. Computer Science and Systems, McMaster Univ., 1989

[Hop82] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proc. National Academy of Sciences, 79, 2554-8, 1982

[Hop86] J.J. Hopfield and D.W. Tank, "Computing with Neural Circuits: a model", Science, Vol. 233, pp. 625-633, 1986

[Koh77] T. Kohonen, Associate memory - a system theoretical approach, Springer Verlag, New York, 1977

[Koh84] T. Kohonen, Self-organization and associative memory, Springer Verlag, Berlin, 1984

[Lip87] R.P. Lippman, "An introduction to computing with neural nets", IEEE ASSP magazine, pp. 4-22, April 1987

[Mad86] G.P. Madhavan, H. de Bruin, A.R.M. Upton, M.E. Jernigan, "Classification of Brain-stem Auditory Evoked Potential by Syntactic Methods", Electroencephalography and Clinical Neurophysiology, 65, 289-296, 1986

[Mat87] C.J. Matheus and W.E. Hohensee, "Learning in artificial neural systems", Techn. Report 87-1394, Dept. Computer Science, Univ. Illinois, Dec. 1987

[Min69] M. Minsky and S. Papert, The perceptrons, MIT Press, 1969

[Min88] M. Minsky and S. Papert, The perceptrons: expanded edition, MIT Press, 1988

[Om87] S.M. Omohundro, "Efficient algorithms with neural network behaviour", Techn. Report 87-1331, Dept. Computer Science, Univ. Illinois, Dec. 1987

[Ros61] R. Rosenblatt, Principles of neurodynamics, Spartan Books, Washington, 1961

[Ru86a] D.E. Rummelhart and J.L. McClelland, Parallel distributed processing: exploration in the microstructure of cognition, MIT Press, 1986

[Ru86b] D.E. Rummelhart, G.E. Hinton, and R.J. Williams, "Learning internal representation by error propagation", in [Ru86a]

[Sej86] T. Sejnowski and C.R. Rosenberg, "NETtalk: a parallel network that learns to read aloud", Johns Hopkins Univ., Techn. Report EECS-86/01, 1986

[Sim90] P.K. Simpson, Artificial Neural Systems. Pergamon Press, 1990

[Tou74] J.T. Tou and R.C. Gonzalez, Pattern Recognition Principles, Addison-Wesley, London, 1974

[Wa85] S. Watanabe, Pattern Recognition: Human and Mechanical, Wiley, New York, 1985

[Wid60] B. Widrow and M.E. Hoff, "Adaptive switching circuits", IRE WESCON Convention Record, Part 4, pp. 96-104, 1960

[Wie87] A. Wieland, R. Leighton, "Geometric analysis of neural network capabilities", IEEE International Conference Neural Networks, San Diego, 1987

Nevronske mreže - pregled in uporaba v procesiranju signalov

V prispevku je podan pregled umetnih nevronskih mrež kot bistvenih orodij za razpoznavanje vzorcev. Avtorjeva pozornost gre predvsem značilnostim dobro znanih tipov trenutno uporabljanih nevronskih mrež in implementaciji večnivojskega perceptrona. Podane so specifikacije, različni vidiki in primerjave različnih vrst nevronskih mrež. Opisan je tudi sistem za podporo pri odločanju za nevrološko diagnostiko. Vhod v ta sistem je signal evociranega potenciala, ki ga analizira algoritem za sintaktično razpoznavanje vzorcev. Algoritem temelji na atributni regularni gramatiki in njegove semantične funkcije izračunajo seznam njegovih numeričnih značilk. Druga faza procesiranja signala vključuje dvonivojski perceptron, ki procesira omenjene numerične značilke. Nekaj besed bomo namenili tudi empiričnim pravilom za optimalno nastavitev parametrov pri perceptronu.