Institute of Phonetic Sciences,
University of Amsterdam,
Proceedings 15 (1991), 1-25.

ASPECTS OF NEURAL NETS

David Weenink

Abstract

In this paper the relation between the topology of linear supervised feedforward neural nets and their classification possibilities is investigated. Starting with the classification possibilities of one single node (the building block of neural nets) we will then describe the effect of adding nodes and additional layers of nodes. It will be shown that the classification possibilities of one-layer nets are restricted to a problem space that is linearly separable. Two-layer nets can combine the hyperplanes, formed by the first layer, to form cells in the input space. However, not all possible combinations of cells can be selected by a two-layer net. Only nets with three or more layers can make all possible combinations of cells in the input space.

Criteria are given in order to optimize the topology of neural nets for specific tasks. In addition, some aspects of the effects of varying weights on the decision regions are studied.

1 Introduction

1.1 Context and outline

In recent years there has been much interest in the use of neural nets in speech research. Neural nets have been used a.o. for vowel-classification tasks (McCulloch and Ainsworth, 1988; Hult, 1889; Kamm et al., 1989; Morin and Nusbaum, 1990; Watrous, 1991), speaker-dependent classification of consonants (Waibel et al., 1989), isolated-word recognition (Kämmerer and Küpper, 1990) and speech analysis (Elman and Zipser, 1988). In the studies cited above no explicit motivations were given concerning the topology of the neural nets used. Topology in this context refers to the architecture of the net in terms of number of layers and number of nodes in each layer. Most of the time, topologies were determined by 'trial and error'. In the present study, one of the main topics will be to give some theoretical background on the link between the topology of a neural net and its 'classification capabilities'. One of the questions we want to answer is the following: Given a certain classification problem, what are the minimum bounds on the number of layers and the number of nodes in each layer, to perform this task? When a neural net is used for classification it has to specify to which of M classes a given input belongs. We will concentrate on the classification capabilities of supervised auto- and hetero-associative neural nets with analog inputs, i.e. the inputs are real-valued.

The general outline of this paper will be as follows. After a brief introduction, explaining some of the terminology used in the field, we will start by examining the classification possibilities of a single node, the building block of a neural net. Next we will investigate the possibilities of one layer of these building blocks: one-layer nets.

We will show that when we add another layer and make a two-layer net, the classification possibilities increase significantly. Adding yet another layer gives us three-layer nets. It will be shown that a three-layer net can meet all 'possible' classification tasks. The last section will cope with some other aspects of neural nets such as the function of the nonlinearity. Discussion and conclusions end the paper.

1.2 Terminology

In the literature, several other names for neural nets are commonly used: connectionist models, parallel distributed processing models (PDP), artificial neural nets (ANN), multi-layer perceptrons (MLP) and Boltzmann machines. We prefer the term neural net.

In a neural net, the basic building block is called a node. Other denominations are processing element, processing unit and sometimes McCulloch-Pitts unit. In a neural net many of these nonlinear nodes are connected together and working in parallel. The architecture of such a net resembles the patterns that neurons make in the nervous system, hence the name neural net. The nodes are usually organized into a sequence of layers and connected to each other with connections of variable strength. These connections contain the 'knowledge' or the 'memory' of the net. In general, it is not possible to decide which specific connection is responsible for what classification decision. 'Knowledge' is distributed in the net.

In the operation of a neural net two phases can be distinguished: the *learning* or *training* phase and the *regeneration* or *recall* phase. During the regeneration phase the net has to show what it has learned, it processes a given input to generate an output. During the learning phase a node has the ability to modify its connection strengths depending on the input signals which it receives and the associated teacher signals or error signals; this is called *supervised* learning. A neural net is called *hetero-associative* when teacher signal and input signal are different, and *auto-associative* when they are the same. The teacher or error signal is not provided in some cases where a node modifies its weights depending only on its state and input signal. This is the case of *unsupervised* learning, and such a learning scheme is sometimes called *self-organization*. For further details on these types of learning, we refer to the papers of Lippmann (1987) and Amari (1990).

1.3 Topology

First some remarks concerning notation. We will write down the topology of a network, the number of layers and the number of nodes per layer, in the following way:

$$(Xn_1,\ldots,Xn_j,\ldots,Xn_k),$$

where X is a representation of the information at a particular layer. X is either the symbol A or B, meaning respectively that the representation is analog or binary. n_1 is the number of inputs, n_k is the number of outputs and n_j is the number of nodes in hidden layer j–1 (j>1). Hidden units are units that are neither input nor output. We do not count the inputs as a separate layer, so the number of layers of a neural net is always one less than the number of elements between parentheses. In the case above the number of layers is k–1. The term 'inputs' can refer to two things: the input for the net or the input for a particular node. It will be obvious from the context which input is meant. Examples of some topologies:

$(A2, B4)$	one-layer net with 2 analog inputs followed by a layer with 4 binary outputs.
$(A2, A2, B3)$	two-layer net with 2 analog inputs followed by a layer with 2 hidden analog nodes and a layer with 3 binary outputs.
$(B4, B2, B3, B4)$	three-layer net with 4 binary inputs, followed by a layer with 2 binary hidden nodes, followed by a layer with 3 binary hidden nodes and a layer with 4 binary outputs.

This notation is only suitable for feedforward nets. In a feedforward net there are no feedback connections, information travels in one direction. Only during the learning phase information flows from the output of the net to the input. We will restrict ourselves to feedforward neural nets.

2 Capabilities of one node

The basic building block of a neural net is the node. A node is an element that receives a number of inputs, weighs them and then calculates an output. This output, y, is defined by:

$$y = f\left(\sum_{i=1}^{N} w_i x_i - \Theta\right) \qquad (1)$$

Here, the x_i $(i=1,\ldots N)$ denote the N input values, the w_i $(i=1,\ldots N)$ are real numbers that weigh each input x_i, the Θ is the threshold or bias term and f is an arbitrary function. In (1), the inputs are linearly weighed. The argument of the function f is a linear combination of the inputs x_i. When the argument is equated to a constant, e.g. zero, it forms the equation of a hyperplane in a N-dimensional space.

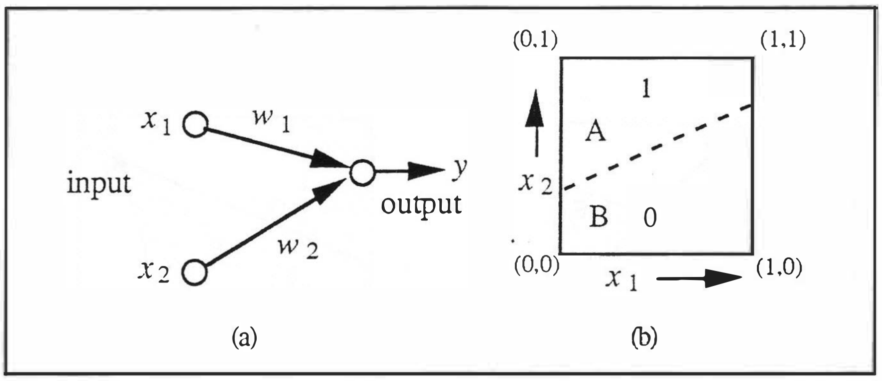

Fig.1. (a) Example of a net with topology $(A2, B1)$. (b) The dotted line separates the input space into two regions A and B.

This hyperplane separates the input space into two regions, one on either side of the hyperplane. Essential of f is that it can be chosen nonlinear in such a way that different values on either side of the hyperplane result. Two popular forms of this function are the sigmoid function or logistic function, and the hard limiting nonlinearity, the Heaviside function. In the sequel, until explicitly notified, we limit the discussion to a nonlinearity f of the Heaviside form. This means that the value of the function f attains distinct values on both sides of the hyperplane: e.g. 1 whenever its argument is greater than zero, and 0 whenever its argument is less than or equal to zero. Sometimes, a node with the Heaviside nonlinearity is called a McCulloch-Pitts unit.

We will give an example of the two-dimensional case with topology ($A2$, $B1$). In figure 1a the full topology is drawn and in figure 1b, the output of the (binary) node as a function of both inputs. The equation for the hyperplane dividing the space follows from the argument of formula (1) and boils down to the equation of a line in the two-dimensional case:

$$w_1x_1+w_2x_2-\Theta=0$$

This line divides the plane in two parts A and B. The exact partitioning depends on the values of the weights w_1, w_2 and the bias Θ.

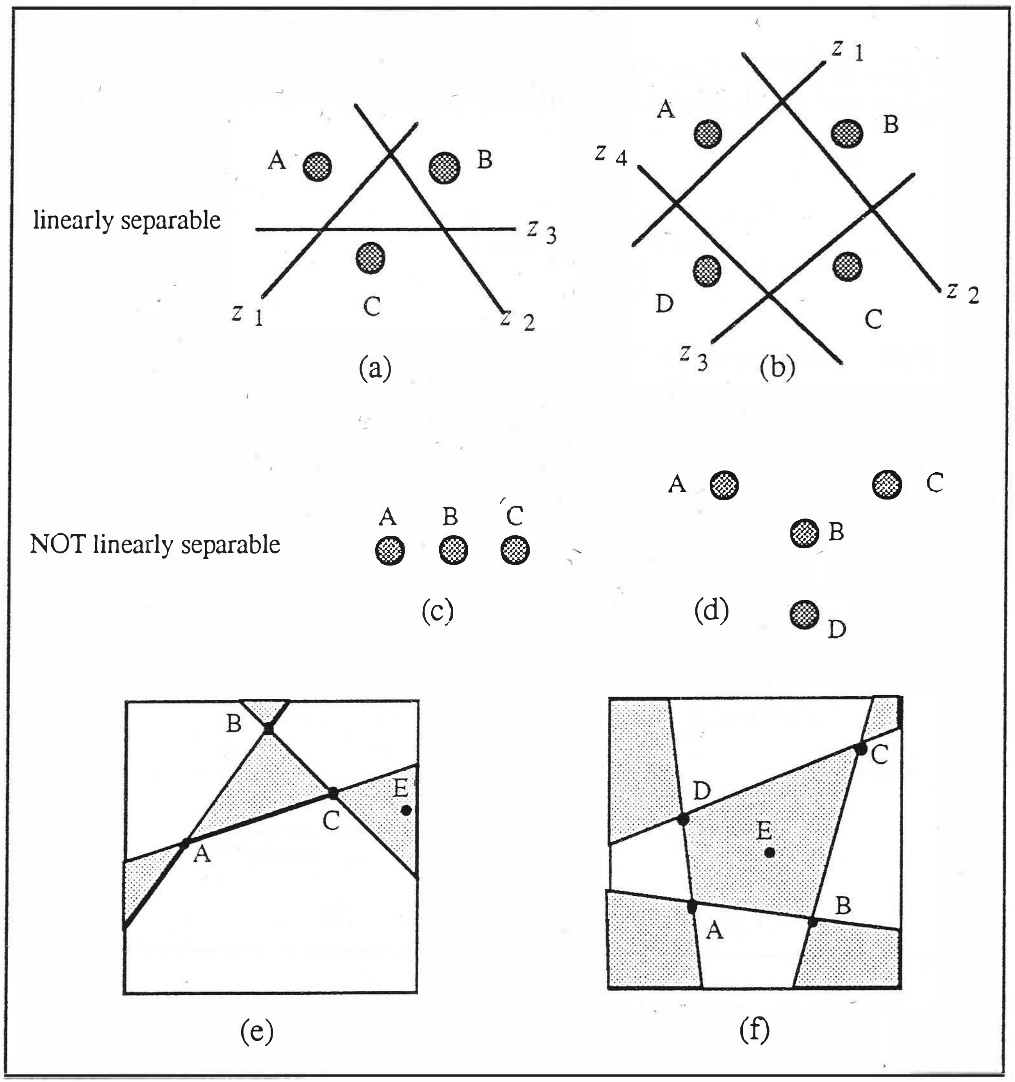

Fig.2. Linear separability in two dimensions. The sets in figures (a) and (b) are linearly separable with a one-layer net, the sets in figures (c) and (d) are not. In (e) and (f) the sets $\{A,B,C\}$ and $\{A,B,C,D\}$ are linearly separable. Adding a new point E, in the shaded area, makes the new set not linearly separable.

3 Capabilities of one-layer nets

In this paragraph we will discuss the classification capabilities of one-layer neural nets. The one-layer neural net represents the most simple neural net. Its topology is (A_N, B_p), N inputs and p outputs, each input being connected to p outputs. We consider the N-dimensional input space divided into p subspaces by p hyperplanes. Classifying in this context means that whenever the input is in subspace k, the output of node k should give a 1 and the output of the other nodes should give a 0. A single node, as we have seen in the previous paragraph, separates the input space into two regions by means of a hyperplane. Each node in the layer can classify according to one distinct hyperplane. Consequently, the classifying capabilities of a one-layer net are restricted to problems which are *linearly separable*. A set S, with N elements A_i, is linearly separable if for all A_i there exists a hyperplane that separates A_i from S–A_i .

As an example, consider a set S with three elements A, B and C. Linear separability means that A can be separated from $B \cup C$ (the union of B and C), B can be separated from $A \cup C$ and C can be separated from $A \cup B$. In figure 2 some examples are given of sets in a twodimensional space that are linearly separable (2a and 2b) and some sets that are not (2c and 2d).

A question related to linearly separable sets is the following: Given a set S of p points uniformly distributed in N-dimensional space. What is the probability that S is linearly separable? First we note that when $p \leq N+1$ the set is in general linearly separable. This question can be solved in a recursive manner: the probability that p points are linearly separable is equal to the probability that p–1 points are linearly separable times the probability that the p-th point is situated such that it is linearly separable from the other p–1 points. What is the probability that, given a set S' of p–1 points that is linearly separable, the p-th point, randomly chosen from a uniform distribution, is linearly separable? We will try to make it plausible that when p/N -> ∞ the probability that the set S is linearly separable tends to zero. In the one-dimensional case it is obvious that when $p > 2$, S is not linearly separable. In a two-dimensional space spanned by [0,1]x[0,1], three randomly chosen points in this space are always linearly separable. What is the probability that a fourth, randomly chosen, point is linearly separable? In figure 2e we have drawn three points A, B and C. We can see from this figure that when a fourth point E is situated in the shaded region, the set of four points is not linearly separable. When four points A, B, C and D are not linearly separable in two-dimensional space, you always can draw a triangle with corners on three of the four points such that the point not used to form the triangle lies in its interior (in the figure the triangle ADB encloses point C). The probability that the four points form a linearly separable set is equal to the ratio of the nonshaded area to the total area. It will be approximately 0.5. The situation for p=5 in two dimensions is depicted in figure 2f. The total probability for this set with p=5 in two dimensions is the product of the probabilities for p=3 and p=4. Every time adding a new point decreases this probability. This argument makes it plausible that the probability, that a set of p randomly chosen points is linearly separable, goes to zero when p/N goes to infinity.

not on a hyperplane

We will show in the next section that whenever a set S is not linearly separable, at least one additional layer is needed in order to be able to classify its elements.

$$P(p,N) = \prod_{i=1}^{p} p(i,N) < (1-\varepsilon)^p \; ; \; p(i,N) \leq 1 \;\wedge\; p(i+1,N) < p(i,N)$$

$< \varepsilon < 1$

$$\lim_{p\to\infty} (1-\varepsilon)^p = 0$$

4 Capabilities of two-layer nets

4.1 Introduction

In the preceding paragraphs we have seen that each node in the first layer is capable of separating the input space into two regions by means of a hyperplane. Addition of a second layer will appear to enrich the classifying potential enormously. The first layer forms hyperplanes in the input space. The second layer makes logical combinations with these hyperplanes from the first layer. A logical combination of hyperplanes boils down to forming so-called cells in the input space. A cell is the smallest region in the input space that is bounded by one or more hyperplanes. Before going into the question about the classification capabilities of two-layer neural nets, which is a question about the possible combinations of cells in the input space that can be selected by a node in the second layer, we first want to answer another question. How many nodes in the first hidden layer do we need to be able to classify M cells in the input space?

4.2 Number of cells formed by a two-layer net

The question stated above can be translated into the following problem: Given a N-dimensional space S. How many hyperplanes, of dimension N–1, do we need to divide space S into M subspaces? Mirchandani et al. (1989) and Makhoul et al. (1989) have found the answer to a related question: What is the maximum number of subspaces in which p hyperplanes can divide a N-dimensional space?

Let $C(p, N)$ be the number of cells, formed by p hyperplanes in N-dimensional space. In general the following recursive relation holds (Schläfli) :

$$C(p, N) \leq C(p-1, N) + C(p-1, N-1) \tag{2}$$

With equality if the hyperplanes are in a general position, i.e. no more than N planes intersect at the same point and no hyperplanes are parallel to each other.
Starting with initial conditions:

$$C(0, N) = 1 \text{ and } C(p, 0) = 1$$

it follows that:

$$C(p, N) = \sum_{i=0}^{N} \binom{p}{i} \tag{3}$$

Whenever $p \leq N$, (3) simplifies to:

$$C(p, N) = 2^p$$

Formula (3) gives the theoretical upper bound on the number of cells that can be formed by p hyperplanes in N-dimensional space. The number of cells grows exponentially whenever the number of hyperplanes, p, does not exceed the dimension of the space, N. If $p > N$ the exponential growth changes into polynomial growth, however, the degree of the polynomial increases with the dimension N. Table 1 shows $C(p, N)$ for a number of values for p and N.

Table 1. Values of $C(p, N)$, the maximum number of cells in a N-dimensional space formed by p hyperplanes of dimension N-1 in general position. The upper row shows the dimension N, while the first column shows the number of hyperplanes p. The relation $C(p, N) = C(p-1, N) + C(p-1, N-1)$, is illustrated for p=4 and N=3.

	Dimension of input space									
p	1	2	3	4	5	6	7	8	9	10
1	2	2	2	2	2	2	2	2	2	2
2	3	4	4	4	4	4	4	4	4	4
3	4	7	8	8	8	8	8	8	8	8
4	5	11	15	16	16	16	16	16	16	16
5	6	16	26	31	32	32	32	32	32	32
6	7	22	42	57	63	64	64	64	64	64
7	8	29	64	99	120	127	128	128	128	128
8	9	37	93	163	219	247	255	256	256	256
9	10	46	130	256	382	466	502	511	512	512
10	11	56	176	386	638	848	968	1013	1023	1024
15	16	121	576	1941	4944	9949	16384	22819	27825	30827
20	21	211	1351	6196	21700	60460	137980	263950	431910	616666
50	51	1276	20876	251176	$2.4\ 10^{6}$	$1.8\ 10^{7}$	$1.2\ 10^{8}$	$6.5\ 10^{8}$	$3.2\ 10^{9}$	$1.3\ 10^{10}$
100	101	5051	166751	$4.1\ 10^{6}$	$7.9\ 10^{7}$	$1.3\ 10^{9}$	$1.7\ 10^{10}$	$2.0\ 10^{11}$	$2.1\ 10^{12}$	$1.9\ 10^{13}$
1000	1001	500610	$1.7\ 10^{8}$	$4.2\ 10^{10}$	$8.3\ 10^{12}$	$1.4\ 10^{15}$	$2.0\ 10^{17}$	$2.4\ 10^{19}$	$2.7\ 10^{21}$	$2.7\ 10^{23}$

$C(p, N)$ serves as an indication of the minimum number of binary nodes in the first layer necessary for classifying M sets in N-dimensional input space. We consider two cases:

1. $M \leq 2^N$
The minimum number of nodes, p, is given by:

$$p = \lceil {}^{2}\log M \rceil,$$

where the number between the 'square' parentheses is rounded to the nearest higher integer. For example when M=3 and N=2 we have p=2, because $1 < {}^{2}\log 3 < 2$.

2. $M > 2^N$
The problem is to find for given M a minimal p such that for a given dimension N:

$$C(p, N) = \sum_{i=0}^{N} \binom{p}{i} \geq M \tag{4}$$

Equation (4) is polynomial in p of order N and hard to solve. Generally it may be impossible to find an analytical solution for $N > 5$. For small p and N a table lookup gives us the answer we need. However, when $p \gg N$ we can make the following estimation for the polynomial $C(p, N)$:

$$C(p, N) \geq \binom{p}{N} \geq \frac{(p-N+1)^N}{N!}$$

which yields:

$$p \geq \sqrt[N]{MN!} + N - 1$$

In practice, we have to be careful what number to take for N, the dimension of the input space. When the inputs are mutually 'orthogonal', the dimension of the input space is simply equal to the number of inputs. However, in many occasions the dimension of the input space can be substantially smaller than the number of input units. This occurs when the inputs are correlated. An indication of the actual dimension of the input space can then be given for example by a principal component analysis.
A concrete example: We want to classify the twelve Dutch vowels, each vowel specified by sixteen bandfilter values. The minimum two-layer net topology that could perform this task is ($A16$, Bh, $B12$), sixteen analog inputs, twelve binary outputs and h hidden binary nodes, where h depends on the real dimensionality of the vowel space. From table 1 we observe that, in order to make twelve cells, we need $h = 4$ when the dimension of the input space is greater than two. Should the real dimension of the problem be $N = 2$, we look up $h = 5$.

4.3 Permissible logical combinations of two-layer nets

We consider two-layer nets with topology (Ai, Bj, Bk), i analog inputs, j binary hidden nodes and k binary outputs. Each output k can form logical combinations with its inputs, i.e. the output of the hidden nodes j that form hyperplanes in the input space. Potentially each output k can form 2^j distinct logical combinations with the outputs of j hidden nodes. A logical combination of hyperplanes forms cells in the input space. The question which presents itself now is: which cells, or, which combination of cells, can be selected by a particular node in the second layer, the output? Such a combination of cells will be called a decision region. Before we can answer this question we, first of all, have to know what logical combinations of hyperplanes are permissible with a two layer net. In this paragraph we will prove that of all possible logical combinations only certain subsets of a restricted form are permissible with a two-layer net.

All possible logical combinations can be formed with the operators $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT) and $\oplus$ (XOR: eXclusive OR). Table 2 shows the outputs of these operators on two binary inputs z_1 and z_2.
We can interpret this table graphically by representing z_1 and z_2 in the two-dimensional plane. In figure 3a we notice that one line separates the points (1, 1) from the three points (0, 0), (0, 1) and (1, 0). This line functions as the decision boundary for the Boolean AND. Likewise the line in figure 3b functions as the decision boundary for the Boolean OR. However, no single line, functioning as the decision boundary for the XOR, exists that separates the points (0, 1) and (1, 0) together from the two other points. Therefore, a two-layer neural net cannot have decision regions selected by an XOR. On the basis of this conclusion (and at that time, the lack of a suitable training algorithm) it was once suggested that neural nets were not very interesting objects to investigate (Minsky and Papert, 1969).

The only logical operators that can be used in forming logical expressions, learnable with a two-layer net, therefore are: $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). We will prove that the only permissible logical combinations of a node with N binary inputs are of the following two forms: 7 met behan

$$(z_1 \vee \ldots \vee z_p) \wedge (z_{p+1} \wedge \ldots \wedge z_{p+k}) \quad p, k \geq 0 \text{ and } p+k \leq N \qquad (6)$$
$$(z_1 \vee \ldots \vee z_p) \vee (z_{p+1} \wedge \ldots \wedge z_{p+k}) \quad p, k \geq 0 \text{ and } p+k \leq N \qquad (7)$$

Table 2. Truth table of the Boolean operators ¬ (NOT), ∧ (AND), ∨ (OR), en ⊕ (XOR).

z_1	z_2	$\neg z_1$	$z_1 \wedge z_2$	$z_1 \vee z_2$	$z_1 \oplus z_2$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

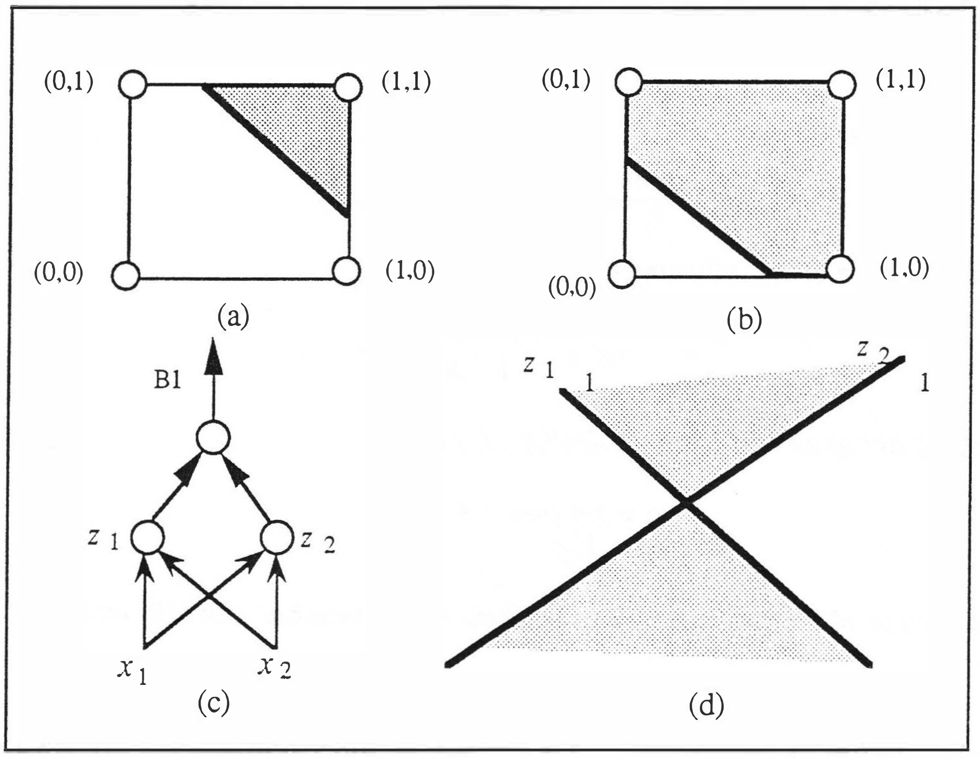

Fig 3. Graphical representation of boolean functions AND (a) and OR (b) of a neural net (c) with topology $(A2, B2, B1)$. The XOR function cannot be made. No straight line separates (0,1) en (1,0) from the other two points. In (d) the XOR-decision region in the input space is shown shaded.

In these logical expressions the z_i are either 0, 'FALSE' or 'off', or 1, 'TRUE' or 'on'. We implicitly assume that each z_j, is used only once in the particular expression. We will prove these formulae by constructing an explicit decision function for each one of them.
A general decision function of a node, with N inputs z_i, has the form:

$$\alpha_1 z_1 + \alpha_2 z_2 + \ldots \alpha_N z_N > c$$

Where the α_i are the weights and c is the bias. If coefficients α_i and bias c can be found for expressions (6) and (7) then these forms are learnable.
We first note that, without loss of generality, we can choose the decision function for these logical expressions in such a way that only two different coefficients α_i are needed:

$$1 \cdot (z_1 + \ldots + z_p) + \alpha \cdot (z_{p+1} + \ldots + z_{p+k}) > c \tag{8}$$

The logical expression (6) imposes the following conditions on the decision function above: First, at least one of the z_i ($i \leq p$) must be on, together with all the z_j ($j \geq p$) in order for (8) to hold. Second, all the z_i ($i \leq p$) may be on, but, whenever one of the z_j ($j \geq p$) is off, (8) does not hold. These conditions can be mathematically stated as:

$$\begin{array}{rcl} 1 + \alpha \cdot k & > c & \quad k > 1 \\ p + \alpha \cdot (k-1) & \leq c & \quad p > 1 \\ \alpha \cdot k & \leq c & \end{array} \qquad (9)$$

From the first and the third expression in (9) it follows that

$$\alpha \cdot k \leq c < \alpha \cdot k + 1 \qquad (10)$$

While all three combined give:

$$\frac{c-1}{k} < \alpha \leq \min \left(\frac{c-p}{k-1}, \frac{c}{k} \right)$$

Which gives rise to the following condition:

$$\frac{c-1}{k} - \min \left(\frac{c-p}{k-1}, \frac{c}{k} \right) \leq 0$$

This gives rise to two different possibilities for c:

$$\begin{array}{l} c > k \cdot (p-1) + 1 \\ c > k \cdot p \end{array}$$

The minimum of the two expressions is taken, which together with (10) makes:

$$c = k \cdot (p-1) + 1 + \varepsilon \qquad 0 < \varepsilon < 1 \; ? \qquad (11)$$

The value for α can be taken halfway between the values on the right-hand and left-hand side of equation (10) and it becomes:

$$\alpha = p - 1 + \frac{\varepsilon \cdot (2k-1)}{2k \cdot (k-1)} \qquad 0 < \varepsilon < 1 \qquad (12)$$

Equation (11) and (12) are the values for the bias and weights of the decision function that proves equation (6).

In an analogous manner the conditions imposed by logical expression (7) on the decision function (8) are the following:

$$\begin{array}{r} 1 > c \\ 0 \leq c \\ \alpha \cdot k > c \\ \alpha \cdot (k-1) \leq c \end{array}$$

From the equations above we deduce:

$$0 \leq c < 1$$
$$\alpha = \frac{c \cdot (2k-1)}{2k \cdot (k-1)} \qquad (13)$$

These expressions for α and c prove (7). We now have shown that it is possible to form decision functions for the logical expressions (6) and (7).

To show that these expressions are the only ones permissible we further have to prove that no decision function can be constructed for all other possible expressions. Exactly two other 'basic' expressions exist that have a form analogous to (6) or (7): the and-of-ors (14) and the or-of-ands (15). These forms are given by:

$$(z_1 \vee \ldots \vee z_p) \wedge (z_{p+1} \vee \ldots \vee z_{p+k}) \qquad p, k \geq 2 \text{ and } p+k \leq N \qquad (14)$$
$$(z_1 \wedge \ldots \wedge z_p) \vee (z_{p+1} \wedge \ldots \wedge z_{p+k}) \qquad p, k \geq 2 \text{ and } p+k \leq N \qquad (15)$$

In these expressions the minimum values of p and k are 2 since smaller values would reduce them to one of the forms (6) or (7). The following conditions can be imposed by the logical and-of-ors expression on the decision function (8):

$$\begin{aligned} 1 + \alpha &> c \\ 2 &\leq c \\ 2\,\alpha &\leq c \end{aligned} \qquad (16)$$

Combining the last two expressions in (16) and dividing by 2:

$$1 + \alpha \leq c$$

This expression contradicts the first expression of (9). This proves that expression (14) is not a permissible one in our context.

The following conditions on p and α lead to a contradiction for expression (15):

$$\begin{aligned} p &> c \\ k \cdot \alpha &> c \\ 1 < p-1 &\leq c \\ p - 1 + (k-1) \cdot \alpha &\leq c \end{aligned}$$

$c \geqslant 1$, $\frac{1}{k} < \alpha \leq \frac{1}{k-1}$, $p < 1 + \frac{k}{k-1} < 3$
$\Rightarrow p = 2$
$1 \leqslant c < 2$
$\frac{c}{k} < \alpha < \frac{c-1}{k-1} \Rightarrow c > k \geqslant 2$ $\left.\begin{matrix} c < \\ c \geqslant \end{matrix}\right\}$

We have now proved that expressions (6) and (7) are valid expressions and (14) and (15) are not. To complete the proof that (6) and (7) are the only valid expressions we still have to show that all expressions that have (14) or (15) as subexpression are not permissible. This proof is trivial.

We can summarize the permissible logical expressions of a node in the following way: all permissible logical expressions are composed of either a series of simple AND's combined with possibly one series of simple OR's, or, a series of simple OR's combined with possibly one series of simple AND's.

4.4 Decision regions of two-layer nets

Now we are ready to tackle the classifying capabilities of two-layer neural nets. We are familiar with the number of cells formed in the input space by a logical combination of the hyperplanes of the hidden layer, the fractions of bounded and unbounded cells and the permissible logical combinations that a two-layer net can make.

The decision regions of an output node in a two-layer net with topology $(AN, Bp, B1)$, N analog inputs, p hidden binary nodes and 1 binary output, constitute those combinations of cells in the input space that are permissible. The number of possible decision regions is $2^{C(p,N)} - 1$, where $C(p,N)$ is the number of cells in the input space. Because of the form of the permissible expressions (6) and (7), a combination of maximally p simple terms which each can have the value 0 or 1, the output node can maximally form 2^p different combinations. The number of different combinations at the output layer is in general much smaller than the number of possible decision regions in the input space.

In figure 3d an example is given of a decision region that could not be formed with a two-layer net, one that amounts to an XOR. We will show that, despite the fact that not all possible logical combinations are permissible, very many interesting decision regions are possible with two-layer neural nets.

Lippmann (1987, page 16) states that the decision regions of two-layer neural nets are either bounded or unbounded convex regions in the input space. We will demonstrate that this statement is too conservative. Decision regions do not need to be convex: non-convex and 'hollow' regions are possible too. The definition of a convex region is that a straight line between any two points in the region lies totally within the region. Figure 4a shows four possible decision lines, z_1, z_2, z_3 and z_4 from the four hidden nodes from a net with topology $(A2, B4, B1)$. We will show that the output node can form the non-convex decision region, which is shown shaded in this figure, by a specific combination of the four decision lines. A 1 denotes that side of the line where the hidden node generates a 1, the other side of the line gets the 0. It is not essential which side of the line is chosen. All eleven cells are labeled with the quadruple (z_1, z_2, z_3, z_4), giving the relative location respective to the four decision lines. We have to prove that we can find a linear combination of z_1, z_2, z_3 and z_4 such that when the value of this linear combination exceeds the bias, precisely the shaded area, a combination of three cells in the input space is selected. The Boolean function characterizing the desired decision region is:

$$(z_1 \vee z_2) \wedge z_3 \wedge z_4 \qquad (17)$$

The decision function for this logical expression can easily be found with the help of the rules we gave in the preceding paragraph. The weights and bias can be chosen according to formulae (11) and (12). Because (17) is a simple expression we will deduce the decision function in yet another way. Column *B1* in table 3 contains the value of this Boolean expression, it gives a 1 for each cell in the desired region and a 0 for all others. The table as well as the figure show that the combinations of z_1, z_2, z_3 and z_4 that select the desired decision region all contain at least three 1's. Therefore the sum function (column *Sum* in the table) $z_1 + z_2 + z_3 + z_4$ with a bias of 3 almost selects the region. A small modification of the sum function is needed to weigh z_3 and z_4 somewhat more than z_1 and z_2 to deselect two unwanted regions with a sum of 3.

It appears that the following function, implemented in the output node $B1$, selects the correct region:

$$z_1 + z_2 + (1 + \varepsilon)(z_3 + z_4) > 3 + 1.5\,\varepsilon \qquad (18)$$

The exact value of ε is not important in this context, it can be any value between zero and one. We have now shown that the function (18) can select the shaded region of figure 4a.

A two-layer neural net is also capable of making 'hollow' decision regions illustrated in figure 4b. It is the simplest 'hollow' decision region that can be constructed in two dimensions. We will prove that the output node of a neural net with topology $(A2, B6, B1)$ can already generate this decision region. As before, $z_1, ..., z_6$ are decision lines corresponding to the six hidden nodes. The shaded area is given by the Boolean expression:

$$z_1 \wedge z_2 \wedge z_3 \wedge \neg (z_4 \wedge z_5 \wedge z_6)$$

which can be brought into the standard form (6) as:

$$(\neg z_4 \vee \neg z_5 \vee \neg z_6) \wedge z_1 \wedge z_2 \wedge z_3 \tag{19}$$

Disconnected decision regions are possible too with a two-layer net. We will give an example in one dimension. Given that $a < b$ and the decision boundaries $z_1 = x<a$ and $z_2 = x>b$, then $z_1 \vee z_2$ selects the simplest disconnected decision region possible.

Table 3. Column *B1* shows values of the Boolean expression $(z_1 \vee z_2) \wedge z_3 \wedge z_4$ for all possible combinations of z_1, z_2, z_3 and z_4. Column *Sum* shows the value of $z_1 + z_2 + z_3 + z_4$. Because of the chosen topology not all possible combinations of z_1, z_2, z_3 and z_4 are associated with real cells in figure 4a. The combinations that are not present in the figure are associated with virtual cells and are shown in column *Present* with a minus-sign.

z_1	z_2	z_3	z_4	$B1$	Sum	Present
0	0	0	0	0	0	–
0	0	0	1	0	1	+
0	0	1	0	0	1	+
0	0	1	1	0	2	+
0	1	0	0	0	1	–
0	1	0	1	0	2	+
0	1	1	0	0	2	–
0	1	1	1	1	3	+
1	0	0	0	0	1	–
1	0	0	1	0	2	–
1	0	1	0	0	2	+
1	0	1	1	1	3	+
1	1	0	0	0	2	+
1	1	0	1	0	3	+
1	1	1	0	0	3	+
1	1	1	1	1	4	+

5 Three-layer nets

Three-layer nets are capable of implementing the whole set of Boolean functions, AND, OR, XOR and NOT. We have seen that the Boolean expressions that a two-layer net could form were of the restricted forms (6) and (7). A three-layer net imposes no restrictions on Boolean expressions. To prove this, we first use the fact that the Boolean operators $\wedge$, $\vee$ and $\neg$ form a complete set, i.e. every possible Boolean expression can be constructed with only these operators. It then suffices to show that with a three-layer net, for every possible combination of these operators, a decision function can be constructed. Every Boolean expression can be broken down into simple expressions that can be handled by a two-layer net, combined with $\wedge$ or $\vee$. These notions above conclude the 'proof'. We will just give a simple example how combinations of expressions can be implemented. The most simple Boolean expression that can not be implemented in a two-layer net is the XOR. We can write $z_1 \oplus z_2$ in two possible ways either as an *and-of-ors* or as an *or-of-ands*:

$$(\neg z_1 \wedge z_2) \vee (z_1 \wedge \neg z_2) \tag{20}$$
$$(z_1 \vee z_2) \wedge (\neg z_1 \vee \neg z_2) \tag{21}$$

The two forms above suggest two possible implementations of $\oplus$ with a net of topology ($B2, B2, B1$), two binary inputs, two hidden binary units and one binary output. (Beware: the given topology is that of a binary two-layer net and not that of a three-layer net. However, in general, we are not interested in binary inputs but in analog inputs, we want to combine hyperplanes in the input space. Therefore the inputs should be considered as a layer that receives input from a preceding analog layer.)

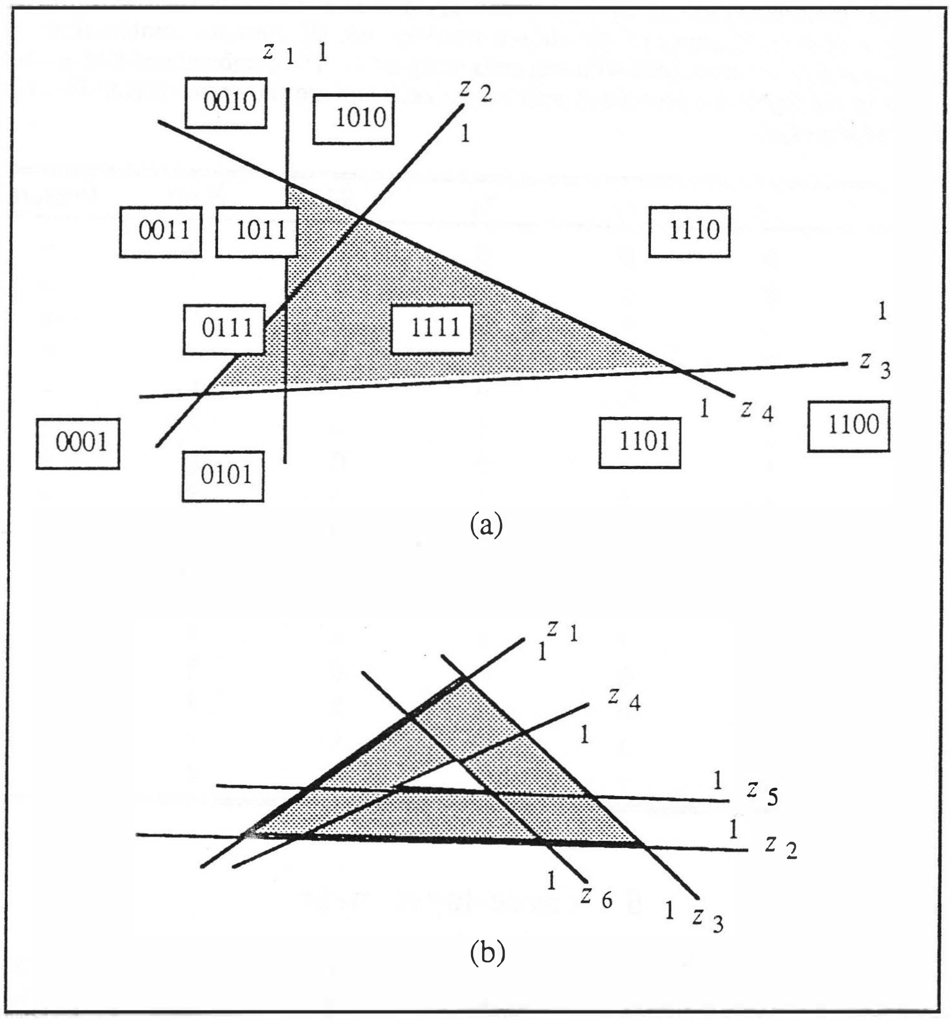

Fig 4. Two possible decision regions for two-layer nets. (a) Most simple concave decision region of net with topology ($A2$, $B4$, $B1$). In figure (b) the most simple hollow decision region of a net, with the topology ($A2$, $B6$, $B1$) is shown. That side of the line where output of the hidden node is 1 is marked.

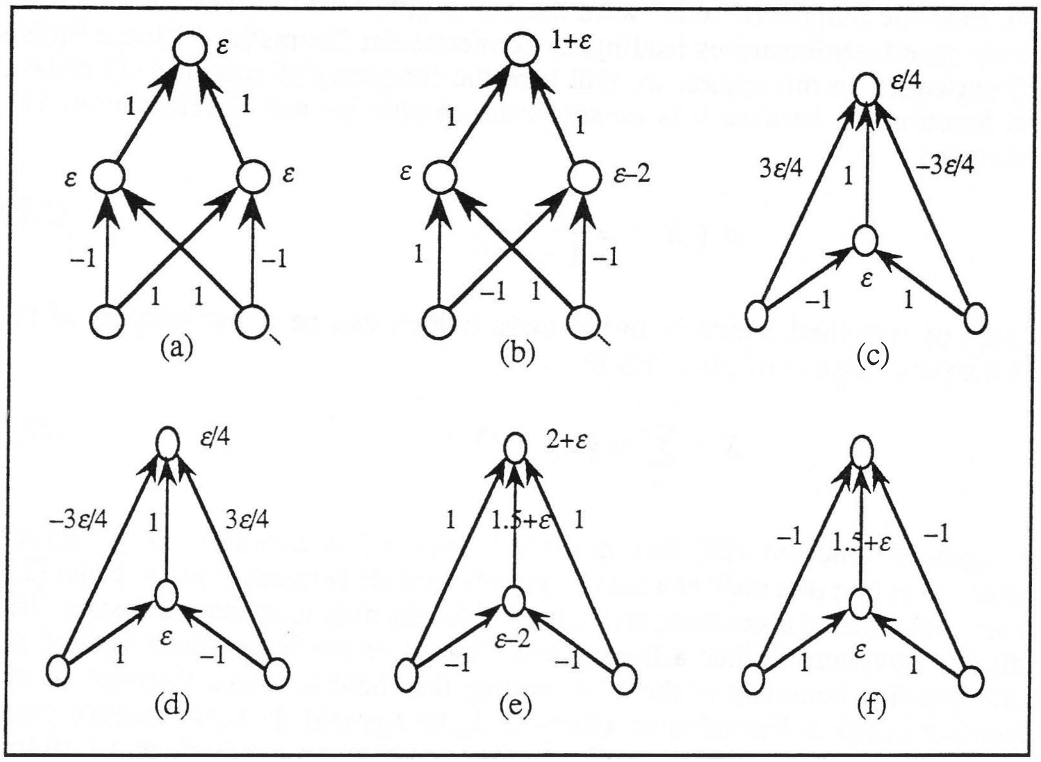

Fig. 5. Implementations of the XOR-function with two different topologies. In all figures $0 < \varepsilon < 1$.

The first implementation is suggested by (20): the two expressions within parentheses are implemented in either hidden node and the output combines them with $\vee$. The biases and weights can be calculated with the help of the formulas of section 4.4. The second implementation can in an analogous manner be derived from (21). These two possible implementations of the $\oplus$ are shown in figure 5a and 5b.

A careful look at equations (20) and (21) suggests another topology which also implements the $\oplus$: one output node with three input connections, two directly connected to the inputs and the third to the output of a hidden node which is also connected to these inputs. The four possible combinations of weights and biases correspond to the four possible choices for the Boolean function of the one hidden node: each one of the expressions between brackets in (20) and (21). Figure 5c, d, e, and f show these combinations. Of course the possible ways to implement the $\oplus$ are not exhausted yet: with only 4 constraints on 9 parameters for a net with a topology like the one in fig. 5a (6 weights and 3 biases) there is a lot of freedom. Likewise there are 7 parameters to be calculated for a net with a topology as is shown in fig. 5c (5 weights and 2 biases).

6 Other aspects of neural nets

6.1 The nonlinearity

The nonlinearity f guarantees that the capabilities of multi-layer neural nets are essentially different from those of one-layer nets. Without the nonlinearity the function of a multi-layer net could be reduced to that of an equivalent one-layer net. Different nonlinearities have been used in the literature, depending on the problem. Up till now, in all the derivations of the preceding paragraphs, the nonlinearity was a Heaviside function. Possible outputs of nodes were limited to either 0 or 1. Jones et al. (1990) use locally tuned nonlinearities leading to Connectionist Normalized Linear Spline (CNLS) networks. In the sequel, we will take the function *f* of equation (1) to be a sigmoid function, σ, because it is monotonous, continuous and differentiable. The function σ reads:

$$\sigma (X) = \frac{1}{1 + e^{-X}} \tag{22}$$

X is a sum of weighted inputs x_k to the node (which can be either outputs of the preceding layer or inputs) minus a bias Θ:

$$X = \sum_k w_k x_k - \Theta \tag{23}$$

The sigmoid function (22) has domain $(-\infty, +\infty)$ and range $(0, 1)$ which means that the output of a node can take all possible values between 0 and 1. From (22) we can see that when X is constant, the output of the sigmoid function is constant. The equation $X = constant$ defines a hyperplane parallel to the hyperplane defined by $X = 0$, the decision boundary of the hard limiting threshold function. Contrary to the latter function which is discontinuous from 0 to 1, the sigmoid changes smoothly from 0 to 1. Both these extremes are approximated when the argument tends to ± infinity. As shown in figure 6, we can define a transitional area where:

$$\delta < \sigma (X) < 1-\delta \qquad 0 \leq \delta \leq 1/2$$

On the input side this defines a symmetrical region around $X = 0$. The input range ΔX that corresponds with it is:

$$\Delta X(\delta) = 2 \ln \frac{1-\delta}{\delta}$$

We can interpret $\Delta X(\delta)$ as that part of the domain of X where no clear decisions are yet made, a transitional area. 'Domain' in this context must be understood as the actual domain i.e. the range of values of X in the problem at hand and not the domain in the mathematical sense which is always $(-\infty, +\infty)$. The ratio R between this transitional area and the actual 'domain' of X is very important because it gives us an indication of the relative sharpness of the decision boundary. This ratio is:

$$0 < R = \frac{\Delta X(\delta)}{\text{domain}(X)} < 1$$

This ratio R is a measure of the resolution we wish, it determines the width of a decision boundary. If we want sharp boundaries this ratio has to be much smaller than one. Given a certain decision boundary, the only way in which a node can 'sharpen' it is by enlarging all the weights and the bias with the same factor.

$$X' = \beta X = \sum_k \beta w_k x_k - \beta \Theta = \sum_k w_k' x_k - \Theta', \qquad \beta > 1 \tag{24}$$

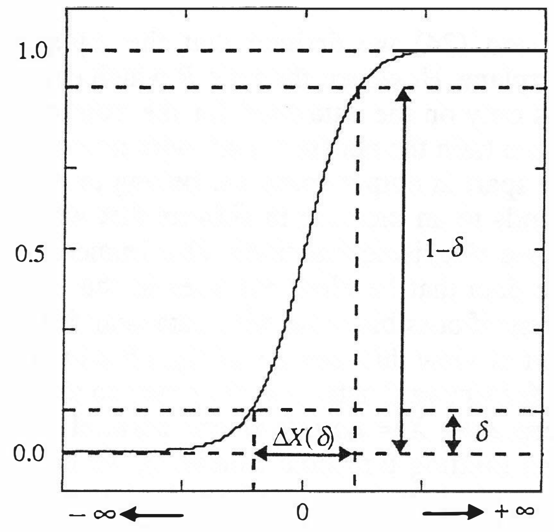

Fig 6. The sigmoid function.

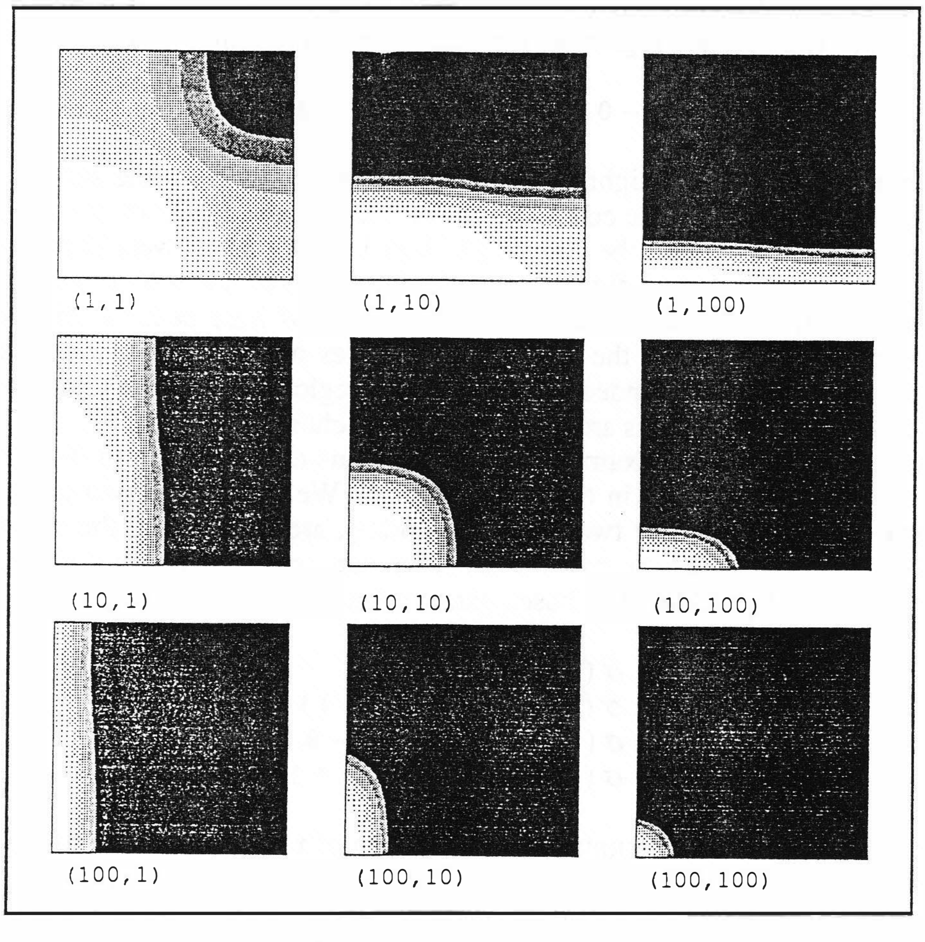

Fig. 7. Some of the possible decision regions of a net with topology ($A2$, $A2$, $A1$). The decision regions are given by the formula: $\sigma\,(A\cdot\sigma(10(x-0.5)) + B\cdot\sigma(10(y-0.5)) - 2)$. The values of the weights A and B that were used, are indicated underneath each figure as the pair (A, B).

From equations (23) and (24) we deduce that the equations $X' = 0$ and $X = 0$ describe the same hyperplane. However, the ratio R which determines the sharpness of the transitions depends only on the data used for the training. The weights and bias should increase, when we train the net (see later) with inputs that lie close together in the input space and far apart in output space i.e. belong to different decision regions. This increase corresponds to an increase in β (note that when β goes to infinity the sigmoid approximates the Heaviside function). This immediately suggests a possible way to train a net: use data that lie close together in the input space but belong to different decision regions: if possible, train with data near decision region boundaries. From a theoretical point of view this may be all right but in practice it turns out to be nearly impossible with feedforward nets. We will return to this point in the discussion.

Because the hyperplanes $X = constant$ are parallel to the plane $X = 0$ (the hyperplane for the hard limiting threshold function), we conclude that the decision regions for a neural net with sigmoids as nonlinearities, can approximate the decision regions limited by hyperplanes as close as is necessary by increasing β. However, the decision regions can substantially change when the ratio of R changes. In general when R decreases, hypersurfaces gradually degenerate to hyperplanes. As a first example of this matter, we have plotted in figure 7 some of the possible decision regions for the output of a net with topology $(A2, A2, A1)$. For simplicity, we have taken the two decision regions of the two nodes in the first layer to be the right half and the upper half of the plane. The decision regions of the output were determined by the formula:

$$\sigma\ (\ A\cdot\sigma\ (10\cdot(x - 0.5)) + B\cdot\sigma\ (\ 10\cdot(y - 0.5)\ ) - 2\ )\ ,$$

where the values of the weights A and B have been taken from the set $\{1, 10, 100\}$. For each of the nine possible combinations of A and B, the output of this equation was calculated with the input in the domain [0,1]x[0,1]. Grey levels were used to represent the output. Underneath each figure the particular values of A and B are indicated as (A, B). Only positive values of A and B were considered here because different signs merely rotate the forms in the figures in multiples of 90 degrees. We note that all decision regions are unbounded i.e. all decision regions start and end at the borders. No bounded decision regions are possible with the chosen topology.

In figure 8 we give an example of what happens to the form of a decision region when the only variation is in the scale factor β. We consider a two-layer net with topology $(A2, A4, A1)$. The two inputs, x_1 and x_2, are confined to the interval [0,1]. With four decision lines, z_1, z_2, z_3 and z_4, we can construct a non-convex decision region such as in figure 4a. The chosen parametrizations are:

$$\begin{aligned} z_1 &: \sigma\ (\ \beta_1\cdot(\ x_1 - 0.5\ )\ ) \\ z_2 &: \sigma\ (\ \beta_2\cdot(\ x_1 - x_2 - 0.1\ )\ ) \\ z_3 &: \sigma\ (\ \beta_3\cdot(\ -0.3\ x_1 + x_2 - 0.1\ )\ ) \\ z_4 &: \sigma\ (\ \beta_4\cdot(\ -1.14\ x_1 - x_2 + 1.34\ )\ ) \end{aligned} \tag{25}$$

The decision region of the output node, a function of x_1 and x_2, can be calculated with the help of equation (18) in which $\varepsilon = 0.5$ was chosen:

$$z_5 : \sigma\ (\ \beta_5\cdot\ (\ z_1 + z_2 + 1.5\ z_3 + 1.5\ z_4\ - 3.75)\ )$$

The β's can help us to influence the steepness of the sigmoids because the functions (25) are invariant under multiplication with a scale factor. We have chosen all $\beta_i = \beta$ (i=1,...5), and, for β=1, 10, 100 and 1000, plotted in figures 8a, b, c and d respectively the value of z_5 as a function of x_1 and x_2 with grey levels.

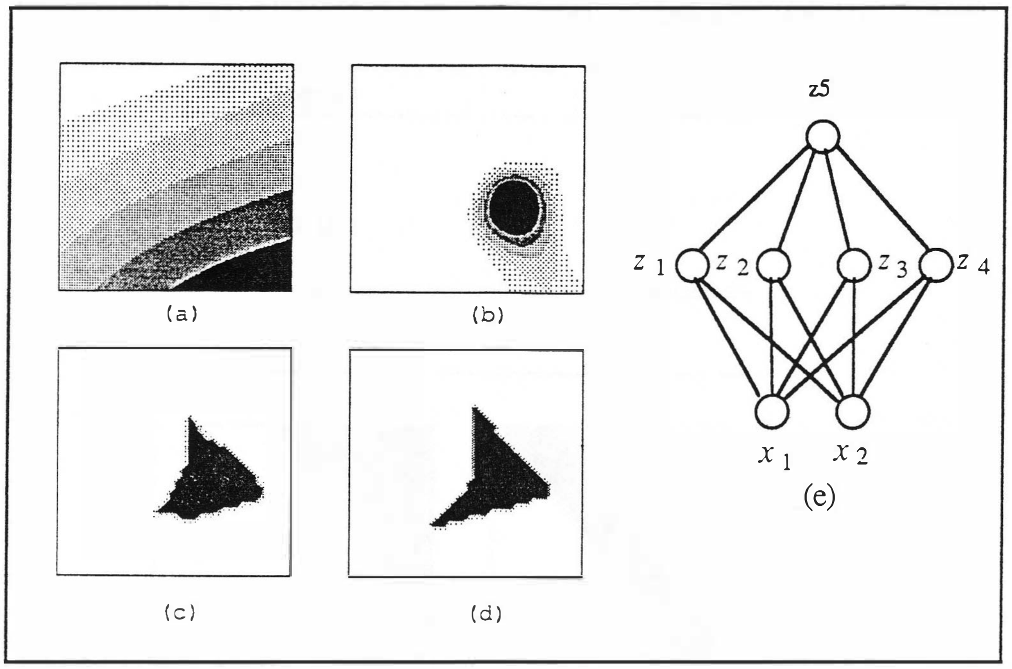

Fig. 8. The output of net (e), with topology ($A2, A4, A1$) as a function of the steepness of its sigmoids: (a) β=1, (b) β=10, (c) β=100, (d) β=1000. Further explanation is given in the text.

Every figure is scaled individually, the larger z_5 the darker the grey. The figures clearly show that the decision region only possesses sharp decision boundaries when the β's are sufficiently large. For moderate values of β, e.g. β=10, the transition area of the sigmoid plays an important role and the decision boundary diverges from the one intended. For β=1 the decision region is nowhere near the intended one. The figures 8a, b, c, and d make clear that by varying the weights and the biases, the decision regions of the sigmoid function can be made very diverse and that these decision regions are not limited to ones that are bounded by hyperplanes, as was the case for the hard limiting threshold function but by hypersurfaces. The corners of the cell can become rounded.

6.2 Level coding

The output of the sigmoid function can take all values between zero and one. This means that, in principle, more information than just 'on' and 'off' can be coded. Whenever this is the case we speak of *level coding*.

An example of level coding is the following problem. We want to solve the input-output relations of the table above with a multiplexer network of topology ($B2, A1, Ax, B4$), where x should be the smallest value possible. To solve the input-output relations of this table the net first has to represent the four possible input values as four different output values of the first hidden node. It makes the problem a one-dimensional one: find a separation of four classes in one dimension. This problem is not linearly separable which means that after the first hidden node we need one extra level of hidden nodes. Rumelhart et al. (1989, their figure 8) give a example with x=4. We have found a solution to the multiplexer problem with x=2. The decision regions of this net are displayed in figure 9. The inputs are taken as analog nodes for drawing purposes. The figure demonstrates that the desired function of the net, multiplexing, is performed correctly: each decision region encloses a separate corner of the square.

Table 4. Inputs and outputs for a multiplexer neural net with 2 inputs and 4 outputs.

Input	Output
00	1000
01	0100
10	0010
11	0001

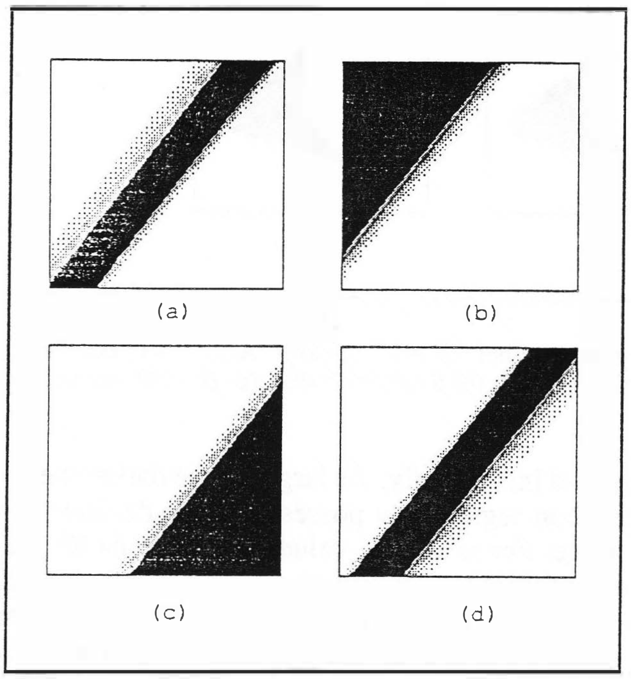

Fig. 9. Decision regions for each one of the outputs of a multiplexer net with topology ($A2$, $A1$, $A2$, $B4$). Lower left corner of each square is (0, 0), upper right corner is (1, 1).

We note that for two possible input combinations, (0, 0) and (1, 1), the functioning of this multiplexer is sensitive to noise: a small deviation from 0 or 1 of the second input activates the wrong output.

We can go further to show that x=2 solves any multiplexer problem of topology (Bn, $A1$, $A2$, $B2^n$): n inputs coding 2^n possible outputs can be multiplexed over one hidden node followed by a layer with two hidden nodes. It suffices to show that the two nodes in the second layer can map the 2^n different outputs of the single node in the first layer to a curved line in two dimensions. Any point on a line with curvature in the same 'direction' is linearly separable.

Let y be the output of the single node in the first layer and z_1 and z_2 the outputs of the two nodes in the second layer. It follows from (22) and (23) that:

$$\begin{aligned} z_1 &= (\ 1 + e^{-w_1 y + \Theta_1}\ )^{-1} \\ z_2 &= (\ 1 + e^{-w_2 y + \Theta_2}\ )^{-1} \end{aligned} \tag{26}$$

Let the weight and bias of the second node be twice those of the first, it follows that:

$$z_1 = (1 + e^{-t})^{-1} \tag{27}$$
$$z_2 = (1 + e^{-2t})^{-1}$$

Solving for t and expressing z_2 as a function of z_1 yields:

$$z_2 = \frac{2\, z_1 - 1}{z_1{}^2} \tag{28}$$

According to (27) the value for z_1 is: $0 < z_1 < 1$ and (28) describes a curved line. By arranging the weights and biases in (26) the points (z_1, z_2) can always be positioned on that part of the line described by equation (28) where the curvature has the same direction. This concludes the 'proof' that the two are sufficient to solve the multiplexer problem. Probably it will be very difficult to train a multiplexer like this when the number of inputs is larger than 2 because ever smaller differences among the output values of the singleton node must be distinguished.

6.3 Training a neural net

During the training of a neural net weights and biases are iteratively updated until the error is smaller than some threshold. Normally during training a desired output is compared with the actual output of the net. The size of the difference, the error, is an indication for the amount of change necessary for the weights and biases. Training a one-layer net is very simple since the desired output is known and the only weights are those between the outputs and the inputs. This means that weights can be gradually updated until the error is sufficiently small. However, this procedure is not directly applicable to a net with hidden nodes because most of the time one does not know what the hidden nodes should represent, there is no desired output from the hidden nodes. Nevertheless this objection, different training procedures for nets with hidden layers have been developed. These minimalization procedures can roughly be divided into two types: simulated annealing (Prager et al., 1986) and gradient descent (Rumelhart et al., 1986). An everlasting problem with minimalization procedures is that the minimizing procedure could get stuck into a local minimum. During simulated annealing the system always has a chance to escape from a local minimum. This procedure is often used with a neural net called a Boltzmann machine but it takes very long computing times to train a net with simulated annealing. Therefore, the second method based on gradient descent is often more attractive.

The gradient descent used in conjunction with feedforward neural nets is called the back propagation training algorithm (Rumelhart et al., 1986) because the errors at the outputs are propagated back into the net and are used to update the weights and biases of the nodes in the hidden layers. The error between the desired output and the estimated output is minimized in a quadratic sense. It is essential that the nonlinearity is monotonous, continuous and differentiable (the sigmoid is such a function). Nodes adapt their weights according to what is called a generalized delta rule and this updating is done *locally*.

7 Discussion

7.1 Possible decision regions and topology

In paragraph 4.5 we have proven that a two-layer neural net could, in principle, make non-convex decision regions: weights and biases were explicitly given for the most simple non-convex region bounded by four lines. However, when we tried to *learn* this decision region to a net with topology ($A2$, $A4$, $A1$) it turned out that much more iterations were necessary than when we used the topology ($A2$, $A4$, $A2$, $A1$). Investigating the weights and biases of the latter net showed that it only had to make decisions based on AND's. This should suggest that although non-convex and 'hollow' decision regions can be formed with a two-layer net, adding an extra layer simplifies the decision making. This would indicate that in each particular layer conclusions should only be based on a combination of AND's since this simplifies the decision making process: all nodes make simple decisions. The benefit of adding nodes could involve a decrease in training time. This is in accordance with a notion of Rumelhart et al. (1986) in which they state that occurrences of the net getting stuck in local minima always involved networks that had just enough connections to perform the task and that adding a few more connections created extra dimensions in weight space to provide paths around the barriers that create local minima in the lower dimensional subspaces.

7.2 Coding the inputs

Input consists in most cases of real numbers. We note that, theoretically, it makes no sense to scale the inputs to a certain range when the scaling algorithm is linear. In principle, the weights and biases of the next layer could perform this scaling. In practical situations, however, scaling the inputs beforehand to the range (0,1) does make sense. Whenever the absolute value of some, or most of the inputs is substantially larger than one, the weighted sum of the inputs, which forms the input to the sigmoid nonlinearity, can be a relatively big number. As a consequence the training algorithm starts when the sigmoid functions are at a position where the derivative is extremely small, and, since the speed of updating weights is a function of this derivative, hardly any training of the net is the effect. In order to effectively train the net, the input to a sigmoid must not start to far of from zero. This argument can also be applied to the training of a net with data near decision boundaries to show why such a training is very difficult if not impossible. In any case it is better to scale the inputs to the interval [-1, 1] or [0, 1] if one uses the tanh or the sigmoid nonlinearity (Elman and Zipser, 1988). As a scaling function, the sigmoid function performs well, since it accepts input values in the range $(-\infty, +\infty)$.

7.3 Coding the outputs

Two possible output units exist: analog and binary output units.

In case of an identity mapping, analog output units are used because the input and output signals are conditioned to be the same. This mapping of a signal onto itself is obtained through one or more hidden layers.

When the net is used for classification purposes, the outputs make binary decisions: output unit i is trained to give a value near 1 whenever the input belongs to decision region M_i. This presupposes M output units when M different decision regions need to be classified. More 'economical' coding of the outputs (e.g. ${}^2\log M$ output units could code M output classes) are not very efficient because part (or all) of the hidden nodes in the layer just before the outputs are simply used as a binary decoder network instead of being used for the task at hand. One further remark concerning output coding: whenever the outputs need to distinguish between elements of two different input sets at the same time, sets with K_1 and K_2 mutual orthogonal elements respectively, two equivalent possibilities exist for the number of outputs. First we could have $K_1 + K_2$ output units of which two units at the same time are active, one from K_1 and one from K_2. In the second case we have $K_1 \cdot K_2$ outputs and only one of them is active at the same time. Both codings are equivalent since both have $K_1 \cdot K_2$ different outputs. An example: classification of twelve vowels while at the same time membership of one of three speaker categories (Male, Female or Child) should be given. Possible codings: $12 + 3 = 15$ outputs, with always two active, one of 12 and one of 3, or, $12 \cdot 3 = 36$ outputs with only one active at the same time.

The number of outputs, M, gives us an indication for the minimum number of units needed in the preceding hidden layer: ${}^2\log M$. When the number of units in the hidden layer is below this minimum, some units are forced to level coding. This may be what we want for some kinds of problems where we are interested in the coding per se, like the multiplexer problem. Whenever we want to detect 'features' we have to stay above this limit.

7.4 Why neural nets?

Using neural nets for classification has a number of advantages as compared to 'classical' statistical methods. Probably the greatest advantage is that there is no need for assumptions on the underlying distribution of the inputs. In the derivation of the classification possibilities of neural nets, we never had to make any assumption concerning the distribution of the input data. When input data cannot be described well by a distribution function, neural net classifiers outperform statistical classifiers (Niles et al., 1989). Another pro is that neural nets can be made adaptive. The net can be in the learning mode permanently and adapt itself to 'new' inputs. Since in a neural net the 'knowledge' is distributed over many connections, they can be made very robust. The performance will not stop but rather decrease only gradually with an increasing number of connections that become disabled. The net also shows its robustness when the input data are not complete (or noisy), because, in many cases, it is able to give the correct output. Neural nets can be implemented on any computer and very good learning algorithms do exist. The topology of the net is only limited by the computing power of the computer. Nowadays, special parallel hardware exists for the implementation of neural nets.

Unfortunately, neural nets also have some serious drawbacks. It is not yet clear how we can add explicit 'knowledge' to a neural net. The only influence we can exercise is on the topology of the net. We do not know how to modify weights and biases when only 'knowledge' and no algorithm is present, except for some trivial problems such as the construction of a gaussian classifier with a neural net when means and variances of the inputs are known (Lippmann, 1987). Likewise, it can be a tedious task to extract 'rules' or 'knowledge' from a neural net. The interpretation of the weights and biases can be difficult, especially when we are confronted with a large net. In this case it will be favourable to design a modularized net, with each module having a specific subtask and the output of each module being integrated afterwards.

Although excellent training algorithms exist, they require a great deal of training data and training time for reasonable accuracy. This can be a serious problem when many data are not available. Special net topologies and nonlinearity functions have been developed to speed up training time in order to reduce the amount of data needed (Jones et al.,1990). However, once a neural net is trained, its response is very fast.

Neural nets are indeed very promising but they are not a panacea for just any problem. In general, when an algorithmic solution exists, this algorithm constitutes the preferred solution. Some problems that are very easy to solve with an algorithm, are notably difficult to solve with a neural net. The outstanding example is the parity problem: the net has to decide whether (a possible transformation of) the input has even or odd parity. This is a difficult problem for a neural net because the larger the input numbers, the smaller the relative differences between succeeding numbers. In order to solve this problem, weights can become prohibitively large. Many other problems e.g. the validation whether a decision region is connected or not, can be reduced to the parity problem (Minsky and Papert, 1960).

8 Conclusions

A number of aspects of neural nets have been discussed. Despite their simple topology, the powerful combinatorial possibilities in the decision regions are impressing. In the field of automatic speech recognition (ASR) they can potentially rival the successful Hidden Markov Models (HMM). Bourlard and Wellekens (1989) show that compared to HMM, neural nets have advantages: a HMM needs more decisions beforehand, like the number of states, the distribution functions, the permitted transitions and the transition rules. Another disadvantage of HMM is their weak discriminating power. During training the probability of the optimum hypothesis is maximized without minimizing alternative hypotheses. We have seen that during training of a neural net, in conjunction with optimizing the wanted hypothesis, the incorrect ones are being minimized. Neural nets show increasing performance in the ASR field (Waibel, 1989; Watrous, 1991). Even hybrid systems which use a neural net as the front end and a HMM doing the time modelling, are being developed.

Our belief is that neural nets not only are valuable as a research tool, but will find growing application in many fields where no exact formalisms exist and decisions are made which are based on experience, however vague this term may be.

Acknowledgements

Our thanks to Louis ten Bosch for stimulating discussions and helpful comments, Sander Kloosterman for training the multiplexer nets and Paul Boersma for help with plotting of figures involving grey scales.

References

Amari, S. (1990): "Mathematical foundations of neurocomputing", *Proceedings of the IEEE* **78**: 1443-1463.

Bourlard, H. and Wellekens, C.J. (1989): "Speech pattern discrimination and multilayer perceptrons", *Computer Speech and Language* 3: 1-19.

Elman, J.L. and Zipser, D. (1988): "Learning the hidden structure of speech", *Journal of the Acoustical Society of America* **83**: 1615-1626.

Hult, G. (1989): "Some vowel recognition experiments using multilayer perceptrons", *STL-QPSR* **1**: 125-130.

Jones, R.D., Lee, Y.C., Qian,S., Barnes, C.W., Bisset, K.R., Bruce, G.M., Flake, G.W., Lee, K., Lee, L.A., Mead, W.C., O'Rourke, M.K., Poli, I.J. and Thode, L.E. (1990): "Nonlinear adaptive networks: a little theory, a few applications", *Los Alamos National Laboratory Rpt.* No. LA-UR-91-273.

Kamm, C.A., Streeter, L.A. Kane-Esrig, Y. and Burr, D.J. (1989): "Comparing performance of spectral distance measures and neural network methods for vowel recognition", *Computer Speech and Language* **3**: 21-34.

Kämmerer, B.R. and Küpper, W.A. (1990): "Experiments for isolated-word recognition with single- and two-layer perceptrons", *Neural Networks* **3**: 693-706.

Lippmann, R.P. (1987): "An introduction to computing with neural nets", *IEEE Magazine*, April 1987.

Makhoul, J., Schwartz, R. and El-Jaroudi, A. (1989): "Classification capabilities of two-layer neural nets", *Proceedings ICASSP*, Glasgow: 635-638.

McCulloch, N. and Ainsworth, W.A. (1988): "Speaker independent vowel recognition using a multi-layer perceptron", *Proceedings of Speech '88*, Edinburgh: 851-857.

Minsky, M. and Papert, S. (1969): *Perceptrons: An introduction to computational geometry*, MIT Press.

Mirchandani, G., Cao, W. and Bosworth, B. (1989): "Efficient implementation of neural nets using an optimal relationship between number of patterns, input dimension and hidden nodes", *Proceedings ICASSP*, Glasgow: 2521-2523.

Morin, T.M. and Nusbaum, H.C. (1990): "Perceptual learning of vowels in a neuromorphic system", *Computer Speech and Language* **4**: 79-126.

Niles, L., Silverman, H., Tajchman, G. and Bush, M. (1989): "How limited training data can allow a neural network to outperform an 'optimal' statistical classifier", *Proceedings ICASSP*, Glasgow: 17-20.

Prager, R.W., Harrison, T.D. and Fallside, F. (1986): "Boltzmann machines for speech recognition", *Computer Speech and Language* **1**: 3-27.

Rosenblatt, R. (1959): *Principles of neurodynamics*, Spartan Books, New York.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986): "Learning internal representations by error propagation", reprint in: *Neurocomputing: Foundations of research*, Anderson, J.A. and Rosenfeld, E. (eds.): MIT Press, Cambridge: 675-695.

Schläfli, L. (1814-1895): *Gesammelte Mathematische Abhandlungen*, vol. 1, Birkhäuser, Basel, 1950: 209-212.

Waibel, A. Hanazawa, T., Hinton, G., Shikano, K. and Lang, K.J. (1989): "Phoneme recognition using time-delay neural networks", *IEEE Trans. on ASSP* **37**: 328-339.

Watrous, R.L. (1991): "Context-modulated vowel discrimination using connectionist networks", *Computer Speech and Language* **5**: 341-362.