A Unified Framework for MLPs and RBFNs: Introducing Conic Section Function Networks

Georg Dorffner
Dept. of Medical Cybernetics and Artificial Intelligence
University of Vienna
and Austrian Research Institute for Artificial Intelligence
georg@ai.univie.ac.at

Abstract

Multilayer Perceptrons (MLP, Werbos 1974, Rumelhart et al. 1986) and Radial Basis Function Networks (RBFN, Broomhead & Lowe 1988, Moody & Darken 1989) probably are the most widely used neural network models for practical applications. While the former belong to a group of "classical" neural networks (whose weighted sums are loosely inspired by biology), the latter have risen only recently from an analogy to regression theory (Broomhead & Lowe 1988). On first sight, the two models—except for being multilayer feedforward networks—do not seem to have much in common. On second thought, however, MLPs and RBFNs share a variety of features, worthy of viewing them in the same context and comparing them to each other with respect to their properties. Consequently, a few attempts on arriving at a unified picture of a class of feedforward networks—with MLPs and RBFNs as members—have been undertaken (Robinson et al. 1988, Maruyama et al. 1992, Dorffner 1992, 1993). Most of these attempts have centered around the observation that the function of a neural network unit can be divided into a propagation rule ("net input") and an activation or transfer function. The dot product ("weighted sum") and the Euclidean distance are special cases of propagation rules, whereas the sigmoid and Gaussian function are examples for activation functions. This paper introduces a novel neural network model based on a more general conic section function as propagation rule, containing hyperplane (straight line) and hypersphere (circle) as special cases, thus unifying the net inputs of MLPs and RBFNs with an easy-to-handle continuum in between. A new learning rule—complementing the existing methods of gradient descent in weight space and initialization—is introduced which enables the network to make a continuous decision between bounded and unbounded (infinite half-space) decision regions. The capabilities of CSFNs are illustrated with several examples and compared with exisiting approaches. CSFNs are viewed as a further step toward more efficient and optimal neural network solutions in practical applications.

1. Introduction

Multilayer Perceptrons and Radial Basis Function Networks are both variants of feedforward neural networks, meaning that their connectivity does not contain cycles, and that they consist of groups of units (layers) with connections linking only adjacent layers. In their simplest versions both neural network types consist of three layers, an input, a "hidden," and an output layer (see fig. 1.). Although

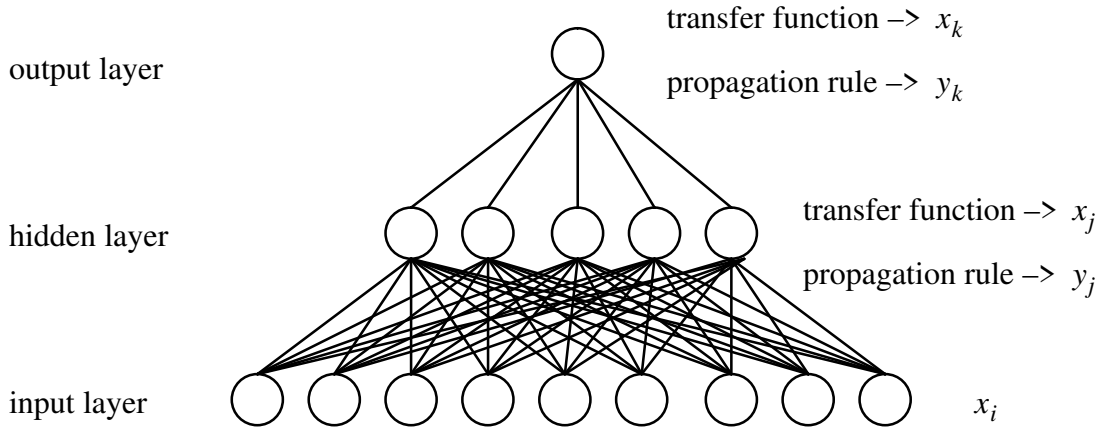

Fig.1. A generic architecture for a feedforward neural network with one hidden layer. *x* denotes activations (output values), *y* denotes "net inputs" (values computed by the propagation rule). *i* is used as index for the input layer, *j* for the hidden layer, *k* for the output layer. For simplification, throughout most of the paper only one output unit whose target is binary is assumed. All results can easily be generalized to multiple output units.

literature frequently depicts it differently, the two network types can best be compared when dividing the function of each unit into two phases—the *propagation rule* (leading to the "net input" *y* of the unit), and the *activation function* (computing the unit's activation and/or output value *x*).[1] All units of an MLP use the dot product between input and weight vectors as the propagation rule, and a sigmoid as activation function, e.g.[2]

$$y_j^{(l+1)} = \sum_{i=1}^{n} w_{ij} x_i^{(l)} - \theta_j^{(l)} \qquad (1)$$

$$x_j^{(l+1)} = \frac{1}{1 + e^{-y_j(l+1)}} \qquad (2)$$

l is an index refering to the layer of the feedforward network. To avoid having to use this superscript in the subsequent equations, we omit them and stick to the convention of letting index *i* only refer to input units, *j* only to hidden units, and *k* to output units. In contrast to (1) and (2), the units of the hidden layer of an RBFN use the Euclidean Distance between weight and input vectors as propaga-

1. We are somewhat following the terminology in Rumelhart & McClelland 1986, especially what "net input" is concerned. For greater ease and clarity in the equations, however, we are using *y* for net inputs and *x* for activations = output values of units.

2. The propagation rule here is depcited including a "threshold" θ, which — if adaptive — is equivalent to assuming an additional bias unit with constant activation, as it is done by many.

tion rule, and a Gaussian as activation function, e.g.[3]

$$y_j^{(l+1)} = \sqrt{\sum_{i=1}^{n} (x_i^{(l)} - w_{ij}^{(l)})^2} \qquad (3)$$

$$x_j^{(l+1)} = e^{\frac{-y_j^{(l+1)2}}{r_j^{(l+1)2}}} \qquad (4)$$

(where r refers to 'radius' and defines the width of the region) while the output units basically are the same as for the MLP, perhaps omitting the sigmoid (making the output layer a linear associator). The main difference between the two units is that (1) leads to decision regions that cover a whole half-space (e.g. half-plane in the case of 2 dimensions), while (3) results in bounded hyper-spherical (e.g. circular) decision regions. The main property of the latter is that the receptive field of each unit is local and restricted to only a small region in input space,[4] while the main property of the former is its coverage of an infinite portion of the space. Both properties can be advantageous or disadvantageous in terms of error in function approximation or generalization. It depends on the data distribution of a given application which of the two types of decision regions appears most appropriate. Thus it depends on the application which type of region should be chosen.

This observation, on first sight, seems to be contradictory to well-known proofs about the ability of both types of networks to be universal function approximators (Hornik et al. 1989, Cybenko 1989, Hornik 1991, Kurkova 1992, Hartman et al. 1990). However, those proofs do not touch upon issues of resources and efficiency directly (e.g. whether a reasonable number of hidden units is needed for a desired approximation) or upon issues of learning (e.g. whether an algorithm exists that finds a solution). In this paper we are more interested in such application-oriented issues and therefore investigate feedforward networks with respect to being efficient or effective in implementation. For instance, for the practitioner it is not enough to know that theoretically a certain type of network (e.g. an MLP) is sufficient to implement a desired mapping, if it can require an unpractically large number of units or tremendous training times. Other types of networks which might not be more powerful in the theoretic sense of approximation theory but which achieve generalization for some applications with less ressources or in less time are therefore desired.

Another important issue is what has been called "localized approximation" (Chui et al. 1992, Omohundro 1989). In many practical implementations of neural networks on-line adaptivity is required, i.e. the ability of adapting the network to changed conditions. If these changes occur in a localized region of the input space, then ideally only a few number of hidden units should have to be retrained

3. We depict the simplest form of an RBF unit, leading to a spherical decision region with standard deviation defined through r. Many variations in literature exist, such as one permitting elliptoidical regions by introducing an additional "weighting" (or standard deviation) parameter for each dimension in the propagation rule. The subsequent analysis will not depend on which variation is chosen. In fact, the "CSFN" — to be introduced later — can be generalized to include all those variations.

4. In the version indicated in (3), of course, the decision regions are nut truly finite, since the Gaussian extends to infinity only converging toward 0. However, at a certain distance of the center the net input is negligible. Truly bounded variations can be found in Dorffner (1992) and in the CSFN, to be introduced later. We will talk about all such spherical decision regions as "bounded."

(see also Omohundro 1989). As the above sample distributions show, this does not necessarily mean that the decision regions of each hidden unit must be bounded. If there is an infinite portion of the input space containing only cases of one class then one MLP unit can cover that subspace and changes in that region would affect only that one unit. The advantage, again, would be that localized approximation would be achieved with much fewer units than if only localized receptive fields of RBF units were used.

Apart from the type of decision regions MLPs and RBFNs implement, they usually differ in the type of learning algorithm employed to arrive at the appropriate weights. MLPs are commonly trained with a variation of gradient descent learning (usually called "backpropagation", Werbos 1976, Rumelhart et al. 1986), while the two connection layers of RBFNs are commonly "trained" differently. The input-to-hidden weights are usually set according to data points in the training set, and the hidden-to-output connections — forming a linear associator — are either set through inverting the linear mapping, or trained by using a delta rule (Broomhead & Lowe 1988, Moody & Darken 1989). Again, advantages and disadvantages abound. Gradient descent over the entire weight space is known to be extremely slow and tedious in many cases, but permits relatively subtle adaptations of the weights to a desired mapping. Initialization as done in RBFNs is quick but relatively crude (compare, e.g., Weymaere 1992). Furthermore common RBFNs have a guaranteed learning procedure (Broomhead & Lowe 1988), while backpropagation can get stuck in local minima fairly easily.

In summary, one sees that both MLPs and RBFNs appear to complement each other in certain cases and thus deserve a unified treatment in the same framework, or even a unification into a network containing both. This paper aims at introducing an approach for the latter. Before we describe this in detail, the following section gives an overview of previous attempts to a unfied treatment of MLPs and RBFNs in literature.

2. Previous attempts on unification

The most straight-forward way of treating MLPs and RBFNs in a unified framework is to consider the three main aspects propagation rule, transfer function, and learning (or "loading", Judd 1990) rule as three independent dimensions, permitting any arbitrary combination for single units. MLP units are thus a special case as the combination dot-product/sigmoid/gradient-descent, while RBFN units consist of the triple Euclidean-distance/Gaussian/initialization+delta. The fact that there are many more possible combinations, or more instantiations of each dimension, easily leads to novel types of network, or some kind of cross-fertilization between the two networks which are so frequently depicted as if they are completely different. Of course, not all combinations of instances along the three dimensions make sense but hardly any one can be rejected outrightly (Dorffner unpublished).

Important examples of such cross-fertilizations are the use of gradient descent for RBFNs (Dorffner 1992, Robinson et al. 1988, Weymaere 1992), or the use of initialization+delta for MLPs (Smyth 1992, Weymaere 1992, Dorffner 1993). The former is a viable way of fine-tuning the centers and/or widths of RBFs, while the latter has proven to improve speed (Smyth 1992) or even performance (Dorffner 1993) of MLPs (explained in more detail in section 4.).

Further extensions of the prototype networks MLP and RBFN can be achieved by alternative combinations of layers with different units, or by combining units of different type in one layer. For instance, the viablity of two RBF layers in cascade (replacing the linear associator between hidden and output layer of an RBFN with another Euclidean/Gaussian layer) has been shown (Robinson et al. 1988, Dorffner 1992, Dorffner unpublished). Also the combination of MLP and RBF units in one hidden layer can lead to improved results (Weymaere 1993). This latter extension is a way of making MLPs and RBFNs complement each other with respect to decision regions, as suggested above. A learning (or "optimization") procedure is introduced that incrementally extends the hidden layer by further MLP or RBF unit candidates until optimum performance with minimum network complexity (defined as number of degrees-of-freedom) is achieved. This procedure achieves what has been hinted upon in the previous section; it can include both bounded and unbounded decision regions and thus be more flexible with regard to input distributions. However, this approach has several disadvantages. Except for what is achieved by a prior cluster analysis, no guidance as to where and when each type of decision region is more appropriate is given. Thus the incremental procedure is rather ad hoc. Furthermore, the rule is non-local and computationally very expensive.

Several people have pointed out prinicpal equivalences of either the propagation rules or the transfer functions of MLPs and RBFNs. Maruyama (1992) and Denoeux & Lengelle (1993) have shown that by writing the term (3) of the Euclidean distance as

$$\sum_{i=1}^{n}(x_i - w_{ij})^2 = \left\|\boldsymbol{x} - \boldsymbol{w_j}\right\|^2 = \|\boldsymbol{x}\|^2 - 2\boldsymbol{x}\boldsymbol{w_j} + \left\|\boldsymbol{w_j}\right\|^2 \qquad (5)$$

and by assuming normalized input ($\|\boldsymbol{x}\| = 1$) the propagation rule of the RBFN (Euclidean distance) can in principle be mapped onto the propagation rule of an MLP unit (dot product $\boldsymbol{x} \cdot \boldsymbol{w}$ plus a threshold, in this case $\left\|\boldsymbol{w_j}\right\|^2 + 1$), with the exception of its sign. In Maruyama et al. 1992 it is shown that any MLP unit can implement RBF units this way (in that for any RBFN an MLP with the same number of units can be found that computes the same function), while RBF units can implement MLP units only under certain conditions, mainly due to the missing extra threshold parameter, for which a dummy extra input has to be introduced. The restriction of normalized input, however, can be severe in many practical applications (see Dorffner 1992). Also in Maruyama et al. (1992), as well as in Geva & Sitte (1992), ways of approximating Gaussians with sigmoids and vice versa are shown. They are based on the observation that the bell shape of a Gaussian (or better still, the positive half of it) resembles an inverted sigmoid.

An interesting way of viewing the propagation rules of MLP and RBFN and their decision regions in the same light is pointed out in Omohundro (1988). He shows that by introducing an additional dimension to the *n*-dimensional input space, and setting this *n+1*-st unit identical to the sum of the squares of the other *n* inputs, the resulting MLP units behave like RBF units with localized receptive fields (i.e. decision regions) with respect to the original *n* dimensions. Thus, an infinite decision region can be mapped to a bounded one. This will be discussed in more detail below.

3. Conic Section Function Networks (CSFN)

The novel unification of MLPs and RBFNs introduced in this paper is based on the observation that both straight line (the decision border of an MLP) and circle (the decision border of an RBFN)[5] are special cases of conic section functions. In between those two extremes there would be hyperbolas, parabolas and ellipses[6] (see fig. 2.). They are all valid borders for decision regions, some of them

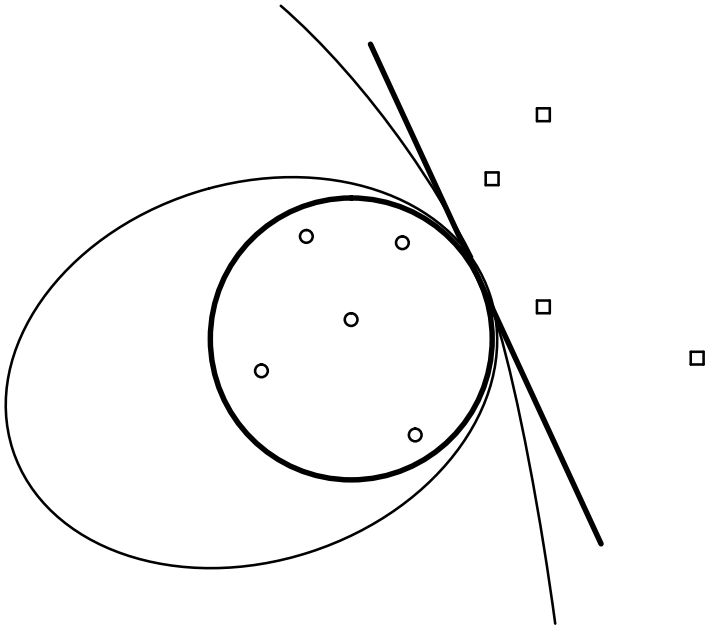

Fig.2. A (2-dimensional) input space and possible decision borders to discriminate different class samples (depicted by little circles and squares). Straight line (hyperplane) and circle (hypersphere) are the usual decision borders of MLPs and RBFNs. Intermediate types of borders (e.g. ellipses or hyperbolas) would be conceivable as well.

unbounded and infinite, some of them bounded and local. The idea therefore is to generalize the function of a unit to include all these decision regions in an easy way, building a continuum between an MLP unit and an RBF unit. For the moment, we restrict ourselves to the propagation rule and assume a threshold or Heaviside function of the form

$$x_j = \begin{cases} 1 \ \ldots \ \text{if } y_j > 0 \\ 0 \ \ldots \ \text{otherwise} \end{cases}$$

as activation function. Thus all decision regions discussed divide the space into a ‘1-’ and a ‘0-region.’

5. Even though the following analysis applies to arbitrary *n*-dimensional input spaces, for illustration purposes it is often useful to refer to the two-dimensional case. We will therefore use both forms interchangably both refering to the general *n*-dimensional case (unless otherwise specified or clear from the context). For the same reason we will sometimes leave out the suffix “hyper-” when talking about decision regions and borders.

6. In generalizations of the simple RBFN (see footnote 3) ellipses are permitted. However, their axes must be parallel to the coordinate axes, while in the continuum of fig. 2. more general ellipses occur.

One way of achieving the desired unification, of course, would be to introduce the complete second-order polynomial as propagation rule. The resulting network would then be equal to the most general second-order network (Lee et al. 1986, Taylor & Coombes 1993), as is done in Roy et al. (1993). However, this would result in a very large number of degrees of freedom ($\frac{n^2}{2} + n + 1$ per unit for n input dimensions) with no direct relationship to the weights introduced above. What we want instead is a continuum that permits a direct transition between straight line and circle via the intermediate conic section functions as in fig. 2. To visualize how this could be done we imagine a three-dimensional space in which the two-dimensional input space is embedded (Fig. 3.). Consider the

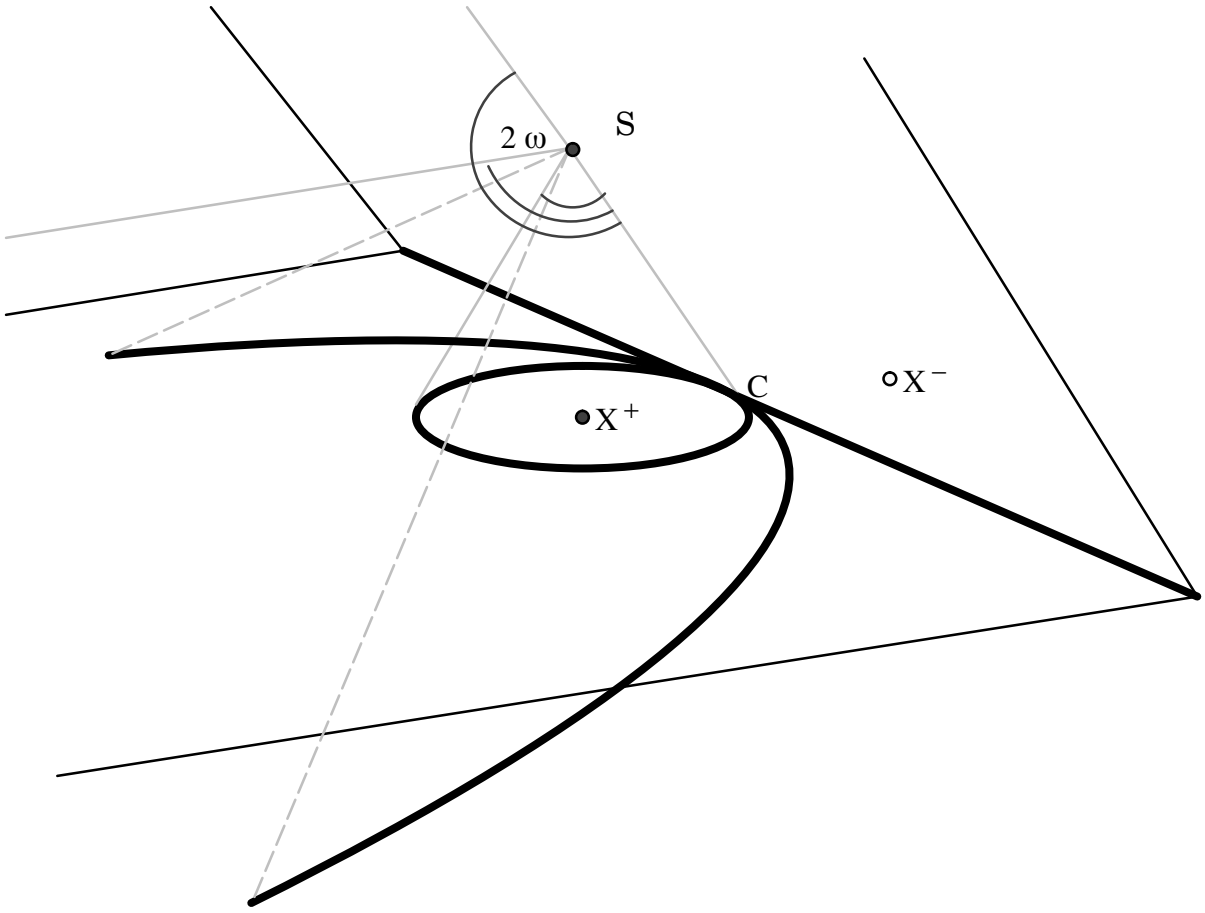

Fig. 3. A (three-dimensional) cone with tip S and opening angle 2ω intersecting the input plane along a circle, a parabola, or — when degrading itself into a plane (2ω=π) along a straight line. X+ and X– denote positive and negative class samples. These variations can be achieved by varying the opening angle and leaving the tip S and the point C fixed.

circle of an RBF unit centered around a positive input sample (leading to activation $x=1$), denoted by X^+. If somewhere above this point, on a straight line perpendicular to the plane, we locate another point we can view that point as the tip of a regular cone, also perpendicular to the plane, intersecting it by forming the given circle. Depending on how high the tip S is put we obtain a certain opening

angle 2ω. Assume that the height of S is equal to the radius of the circle; then ω becomes 45 degrees. Now, by leaving S and one point on the circle fixed, variations of ω result in variations of the section curve. By increasing ω the circle turns into an ellipsis, then into a parabola, and a hyperbola. If ω becomes 90 degrees the cone degrades into a plane intersecting the original plane in a straight line, forming a tangent of the circle we started with.

We see that by varying a single parameter (ω) we get the full continuum between the two functions formed by the propagation rule of MLP and RBF units. It is, of course, not the full spectrum of conic section functions even in the two-dimensional case, but it covers bounded and unbounded decision regions of varying size. The approach can easily be extended to more than two dimensions. Given any n-dimensional MLP- or RBF-unit, we introduce one additional dimension and a regular $n+1$-dimensional hypercone given by the opening angle 2ω, intersecting the n-dimensional space in a hypersphere, a hyperplane, or shapes in between, analogous to the two-dimensional case.

The analytic version of the propagation rule can be derived by starting with the vector equation producing a cone.

$$(\boldsymbol{x} - \boldsymbol{s})\boldsymbol{a} = \cos\omega\|\boldsymbol{x} - \boldsymbol{s}\| \qquad (6)$$

where $\boldsymbol{s}$ is the tip of the cone (defining vector), $\boldsymbol{a}$ is the unity vector defining the axis of the cone, and $\boldsymbol{x}$ defines any point on the surface of the cone. ω can be any value in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This equation expresses the fact that for each point on the surface of the cone the dot product between the vector **x**–**s** and the normalized axis vector is equal to the distance between the point X and the tip, times the cosine of the half opening angle (as given in the definition of a normalized dot product). From this we get the following form.

$$\sum_{i=1}^{n+1}(x_i - s_i)a_i = \cos\omega \sqrt{\sum_{i=1}^{n+1}\left(x_i - s_i\right)^2} \qquad (7)$$

If we set the coordinate system such that n dimensions are identical to the n dimensions of the input space, then by setting $x_{n+1} \equiv 0$ the above form becomes the equation for the intersection between the cone and the input space. For points within the cone the equation becomes an inequality with a smaller right-hand side. Thus we can subtract the right-hand side from the left-hand side and use the resulting form as the propagation rule, leading to net inputs larger than zero for points within the cone (i.e. within intersection regions), and net inputs less than zero for points outside the cone.

$$y_j = \sum_{i=1}^{n+1} (x_i - s_{ij}) a_{ij} - \cos\omega_j \sqrt{\sum_{i=1}^{n+1} \left(x_i - s_{ij}\right)^2} \qquad (8)$$

$$x_{n+1} \equiv 0$$

where *i* and *j* are again indices refering to units in the input and hidden layer, respectively. One can easily see that this formula contains two major parts, roughly corresponding to the MLP and the RBF term of (1) and (3). For this we have to identify the a_{ij} (one for each connection between units *i* and *j*) factors with the weights in an MLP, and the s_{ij} factors with the weights, or better still, *offsets* or *center coordinates*, of an RBFN. Contrary to Maruyama et al. 1992 the two sets of degrees-of-freedom are not identical but separate. This results in a doubling of resources by introducing two degrees-of-freedom per connection, certainly one major drawback of the CSFN when directly compared to either MLP or RBFN. On the other hand this means a considerably smaller complexity than the most general second order unit (see also later discussion). In addition to this, each hidden unit can have its own ω_j. In general, the idea — which will be explicated in more detail later — is the following: First a CSFN is initialized as being either an MLP or an RBFN. This fixes all a_{ij} and s_{ij} values. These values are then clamped and only all ω_js (i.e. one for each hidden unit) are changed to optimize the decision regions. Finally, all three sets of parameters might be fine-tuned.

Propagation rule (8) hides one important fact, namely the fact that all a_{ij} also depend on ω_j since the axis turns with respect to the input space and thus to the coordinate system when ω_j is changed. This is the case since when varying ω_j we leave the tipe of the cone and one point C at the intersection of the cone with the input space fixed. Fig. 4. redraws fig. 3. in frontal view such that both the input plane and the plane of the degraded cone appear as a straight line. This latter plane — the degraded cone at $\omega = \frac{\pi}{2}$ plays a special role, since it contains the points in the cone that remain fixed (in particular, S and C). This plane and the input plane (or in the general case, the two n-dimensional hyperplanes or hyperspaces) form an angle α. Theoretically this angle, or alternatively the "height" of the cone tip above the input plane $s_{n+1,j}$, could be an additional degree-of-freedom when changing the cone. For the sake of simplicity, however, we leave it fixed. Thus it will appear as a constant in the following equations.

With the help of **a** and fig. 4. we can formalize the dependency of a_{ij} on w_j. Consider the case 2, when the axis is somewhere between being perpendicular to the input plane and to the plane of the degraded cone. The best way of approaching this is by considering the fact that in the case of the degraded cone the CSFN unit should turn into an MLP unit. The decision border of that MLP unit is the intersection of the plane of the degraded cone with the input plane, i.e. the line appearing as point C in fig. 4. In general the hyperplane of an MLP is defined through propagation rule (1), whose weights can be visualized by a vector **w** which is orthonormal to the hyperplane, in this case to the

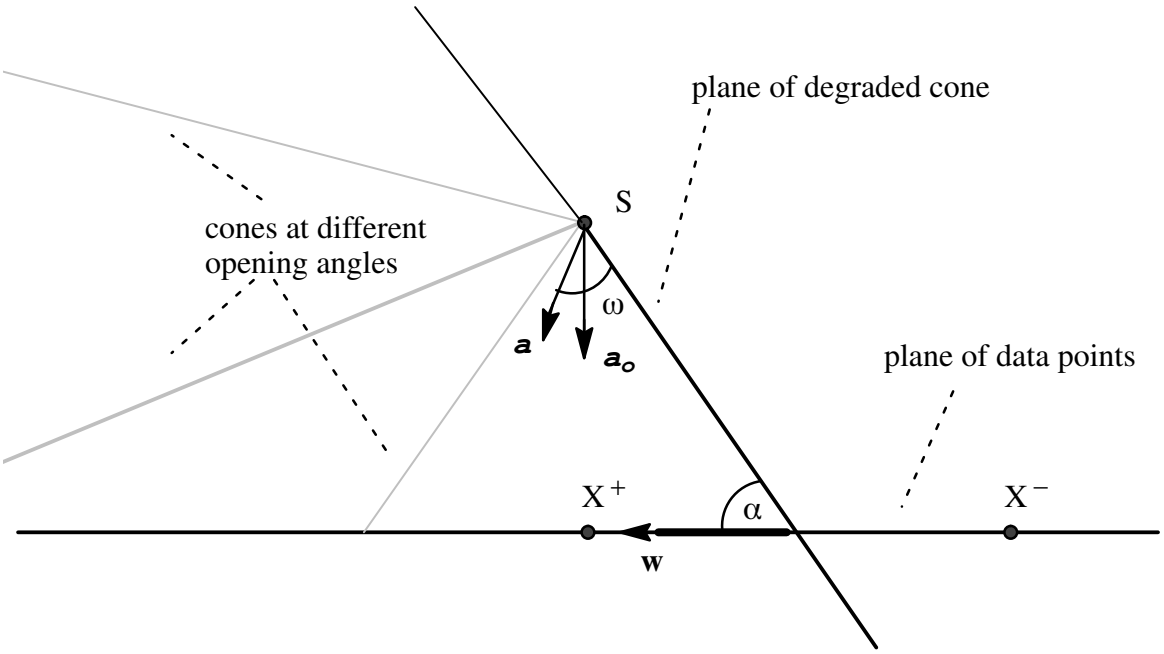

Fig.4. Frontal view of the different cones in fig.3. $\boldsymbol{a}$ is a unity vector defining the axis. The weakly drawn lines depict the silhouttes of cones at different opening angles 2ω.

line. Now one can see that the orthogonal projection of $\boldsymbol{a}_j$ onto $\mathbb{R}^n$, i.e. (a_{ij}) for $1 \leq i \leq n$, is parallel to $\boldsymbol{w}_j$. From this observation, it is easily shown that

$$\begin{aligned} a_{ij} &= w_{ij} \sin\left(\omega_j - \frac{\pi}{2} - \alpha_j\right), \quad 1 \leq i \leq n \\ &= w_{ij}\left(-\cos\left(\omega_j + \alpha_j\right)\right), \quad 1 \leq i \leq n \\ a_{n+1,j} &= -\sin\left(\omega_j + \alpha_j\right) \end{aligned} \tag{9}$$

Since according to (6) $\boldsymbol{a}_i$ has to be a unity vector, this holds only if the corresponding weight vector is normalized ($\|\boldsymbol{w}_j\| = 1$).

Thus we could replace a_{ij} by weights w_{ij} and rewrite (8) as

$$y_j = \bar{y}_j\left(-\cos\left(\omega_j + \alpha_j\right)\right) + s_{n+1,j} \sin\left(\omega_j + \alpha_j\right) - \bar{\bar{y}}_j \cos\omega_j \tag{10}$$

where

$$\bar{y}_j = \sum_{i=1}^{n} (x_i - s_{ij}) w_{ij}$$

$$\bar{\bar{y}}_j = \sqrt{\sum_{i=1}^{n+1} \left(x_i - s_{ij}\right)^2}$$

$\bar{y}$ and $\bar{\bar{y}}$ denote the terms in the two major parts of the propagation rule (loosely called the "hyperplane-" and the "hypersphere part") which are independent of ω. We will use (8) and (10) interchangeably, the latter mainly when the dependence on ω is crucial, the former for ease of writing. Now it also becomes obvious how the two special cases—MLP and RBFN—are derived from (8). The former is given if $\omega = \frac{\pi}{2}$, since then the second term of (8) is identical to 0. In that case (8) reduces to

$$\begin{aligned} y_j &= \sum_{i=1}^{n+1} (x_i - s_{ij}) a_{ij} \\ &= \sum_{i=1}^{n} x_i a_{ij} - \sum_{i=1}^{n+1} s_{ij} a_{ij} \end{aligned} \tag{11}$$

The first part is the weighted sum, while the second part is the threshold of the MLP unit. The scaling of the weights w_{ij} even in this case (with: $-\cos\left(\frac{\pi}{2} + \alpha_j\right) = \sin\left(\alpha_j\right)$, see (9)), as compared to a "true" MLP with that hyperplane can be explained through the fact that the degraded cone is not perpendicular to the input space. The RBFN unit is given if $\omega + \alpha = \frac{\pi}{2}$, since then all a_{ij}, for *i=1 .. n,* are identical to zero, $a_{n+1,j} = 1$, and (8) reduces to

$$\begin{aligned} y_j &= s_{n+1,j} - \cos\left(\frac{\pi}{2} - \alpha_j\right) \sqrt{\sum_{i=1}^{n+1} \left(x_i - s_{ij}\right)^2} \\ &= s_{n+1,j} - \sin\alpha_j \sqrt{\sum_{i=1}^{n} \left(x_i - s_{ij}\right)^2 + s_{n+1,j}^2} \end{aligned} \tag{12}$$

This has the major shape of the propagation rule in (3), but differs in several interesting ways. First of all, the sign is inverted such that large Euclidean distances lead to small net inputs. Secondly, a

constant ($s^2_{i,n+1}$) is added to the squared distance before the square root is applied. This term, even though disturbing at first, does not change the major hyperspherical characteristic. Thirdly, the net input is 0 at the decision border (the surface of the hypersphere) and negative outside. Thus, as opposed to the regular RBFN, these decision regions are "truly" finite and bounded. However, this is not a relevant distinction, either. Both $s_{i,n+1}$ and $\sin\alpha_j$ together define the radius.

4. Initializing CSFNs

The essence of a CSFN is that the cone of each hidden unit can be adapted — or, as it will be called, *folded* or *unfolded* — so as to make an automatic decision upon the most appropriate region boundary. In particular, the network is made to decide between bounded and unbounded decision regions based on the distribution of the data. We have seen, that basically this can be achieved by varying the parameter ω between $\frac{\pi}{2}$ (the MLP case) and $\alpha - \frac{\pi}{2}$ (the RBFN case). The great advantage of the above form of the propagation rule — as opposed to, say, the complete polynomial of second order — is that there is a single additional parameter per unit to be varied to go through the continuous spectrum between the two special cases. Thus for the adaptation of the decision regions the number of degrees of freedom is relatively small, which has considerable effect on the learning algorithm. Of course, due to the small number of degrees of freedom, on the other hand, success of automatic adaptation of the decision regions (henceforth called cone folding) depends on an appropriate initialization of the fixed elements of the cone. In essence, these elements are the hyperplane and the hypersphere of the MLP and RBFN part, respectively. We therefore assume cone folding to set in after a conventional MLP or RBFN has been trained or initialized appropriately. Thus, cone folding is used to improve or prune a network starting off as one of the two conventional types.

We begin by describing a CSFN that starts off as an MLP.

4.1. Starting with an initialized MLP

As introduced in Smyth (1992), Weymaere (1993), and Dorffner (1993), multilayer perceptrons can be initialized by setting decision boundaries between data points or clusters, in analogy to the initialization of RBFNs. One can start by picking positive and negative exemplars of each output class at random, or by picking positive and negative centers after cluster analysis. Assume that we have a pair of points X_j^+ and X_j^-, which are a positive and a negative exemplar (or cluster center) for a desired output class, respectively. Then a hyperplane can be put on the mid-point of the line connecting the two points, and orthogonal to this line, optimally seperating the two points. This is achieved by setting the weights and the thresholds of a hidden unit according to Smyth (1992), Weymaere (1993), or Dorffner (1992)

$$w_{ij} = x_{ji}^+ - x_{ji}^- \tag{13}$$

$$\theta_j = \frac{1}{2}\sum_{i=1}^{n}\left(x_{ji}^{+\,2} - x_{ji}^{-\,2}\right)$$

In terms of computational geometry, this procedure defines hyperplanes so as to form a Voronoi tesselation of the given training points. This is a direct analogy to initializing RBFNs by setting decision regions (in this case hyperplanes instead of hyperspheres) based on training points (in this case pairs instead of single points). Since the number of possible pairs is rather large ($\frac{n(n-1)}{2}$) usually it is reduced by either picking pairs randomly or by pruning methods (Smyth 1992). This procedure fixes the input-to-hidden weights. In a further analogy to RBFNs the hidden-to-output weights can be set using a delta rule (for the linear associator between hidden and output layers), or the whole network can be trained by regular backpropagation using the pre-set weights as (near-to-optimal) starting points.

It can be shown that for many applications this initialization can lead to surprising results as compared to backpropagation while being much more efficient what computing time and convergence is concerned (Dorffner 1993). At the same time, since hyperplanes and positive training points (or even cluster centers) have been devised, this initialization procedure lends itself perfectly as a starting point for cone folding. For this, the a_{ij}-values are computed as in (9), using the weights in (13). In a practical implementation it is advisable at this point to leave the weights after normalization unchanged and compute the product at each update cycle, since the factor changes with ω. The s_{ij} values ($i = 1..n$) are set identical to the coordinates of X^+ as the positive center. The additional constant $s_{n+1,j}$ is computed according to

$$\begin{aligned} s_{n+1,j} &= d\left(X_j^+, \mathcal{H}_j\right) \tan\alpha_j \qquad (14) \\ &= y_j^M(\boldsymbol{x} = X_j^+) \tan\alpha_j \\ &= \left[\sum_{i=1}^{n} \bar{w}_{ij} x_i - \bar{\theta}_j\right] \tan\alpha_j \end{aligned}$$

where $d\left(X_j^+, \mathcal{H}_j\right)$ refers to the distance of the j-th positive sample X^+ to the hyperplane $\mathcal{H}_j$ of the j-th MLP unit, which is identical to the propagation rule (net input) of the initialized MLP applied to point X^+, denoted by $y_j^M(\boldsymbol{x})$. $\bar{w}_{ij}$ and $\bar{\theta}_j$ are weights and threshold from (13) after normalization:

$$\bar{w}_{ij} = \frac{w_{ij}}{\|\boldsymbol{w}_j\|}, \quad \bar{\theta}_j = \frac{\theta_j}{\|\boldsymbol{w}_j\|}$$

4.2. Starting with a pretrained MLP

Assume a regular three-layer MLP (i.e. an MLP with one hidden layer) has been trained by backpropagation according to Werbos 1976, Rumelhart et al. 1986. To initialize a corresponding CSFN one must again start by setting the a- and s-values of all hidden units. Since all decision boundaries are

hyperplanes we set ω to $\frac{\pi}{2}$. The a values are initialized as in section 4.1., this time using the weights derived from training.

For initializing the s-values one has to choose a good class representative X^+ around which the decision boundary is to bend. This is, of course, not trivial. There are several possibilities:

– choose a random training sample as X^+. This strategy does not require much computation, but, of course, will most likely lead to non-optimal results. Figure 5.shows the potential influence of

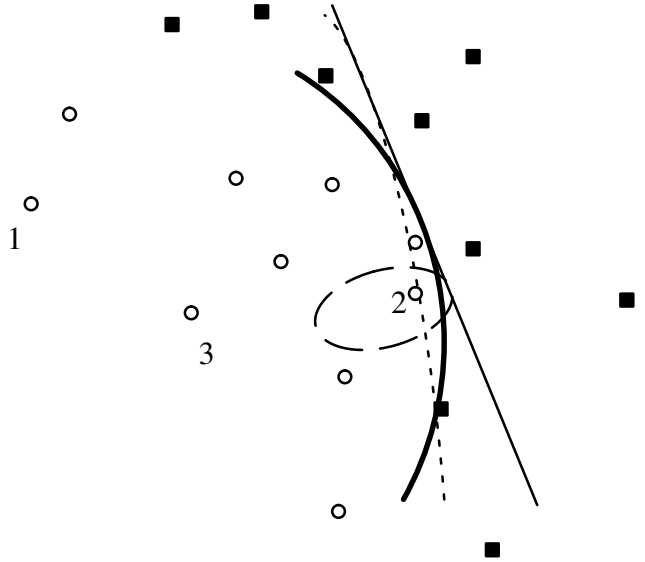

Fig. 5. An input space of positive (white circles) and negative class samples (black squares). Assume a unit of a pre-trained MLP with the straight line as decision border. For adapting the cone so as to change (optimize) the decision region a positive data point has to be picked. If it is too far from the center of the class (point 1) a change in the decision border (hyperbola) tends to exclude true positives early on (dotted line). If it is too close to the straight line, (point 2) the possible variations are too restrictive (dashed line). Point 3 (cluster center or a point close-by) would be ideal in this case (bold line).

choosing a disadvantageous point. It turns out that optimum results can most likely be expected from choosing cluster centers, or at least points close to one.

– If previously a cluster analysis has been performed, choose the closest cluster center as X^+. This promises much more success but can fail if the hyperplane has not succeeded to separate the prior clusters.

– perform a cluster analysis after training on all correctly separated data points and choose the closest cluster center. This strategy requires the most computation (since for each training point it has to be first decided whether it is classified correctly by the given hyperplane) but promises the greatest improvements since the likelihood of an appropriate center to bend the decision region around is increased, as compared to the above two strategies.

In all cases an additional decision must be made as to whether the hidden unit j is a positive or negative "detector." In other words, after training a hidden unit might also have learned to positively re-

spond to the complementary class (all training pairs with negative or zero target output). This decision can be made easily by looking at the weight w_{jk} leading to the (k–th) output unit. If that weight is positive, the hidden unit is a positive detector, and analogously for a negative weight.

After choosing a class representative X_j^+ the values s_{ij} for $i=1..n$ can be set identical to x_{ij}^+. $s_{n+1,j}$ can be computed like in (14), replacing weights and thresholds gained from initialization by those computed through the preceding training.

4.3. Starting with an RBFN

So far we have described the initialization of a CSFN which starts off with only hyperplanes as decision regions ($\omega_j = \frac{\pi}{2}$). For those initializations, the adaptation of the cone can achieve the closing of initially unbounded decision regions. The opposite case is to start off with a conventional RBFN and permit the hyperspherical decision regions to open where appropriate. Given the propagation rule of an RBFN (3), now the s_{ij} ($i=1..n$) values can easily be derived by setting them identical to w_{ij} in (3).[7] $s_{n+1,j}$ is set identical to the radius of the hypersphere given through the RBFN, multiplied by $\tan a$. Thus the hyperspherical part of the CSFN becomes identical to the hypersphere of the RBF unit.

Similar to the previous case (starting with a pre-trained MLP), the hyperplane part is more difficult to obtain appropriately. Here, a suitable hyperplane — which could be any hyperplane tangent to the hypersphere of the RBF unit — must be chosen. A possible strategy would be the same used to initialize an MLP (section 4.1.), i.e. by computing a Voronoi tesselation between positive and negative training samples. By doing this, however, the original radius of the hypersphere is ignored, since the distance of the hyperplane from the data sample (and thus the true radius for the CSFN) is given through the distance between the positive and negative sample. Two strategies are possible: Either (a) shift the hyperplane on the connecting line to be tangent to the hypersphere of the originally given RBFN. Or (b) use the new radius (half the connecting stretch) instead (fig. 6.). In both cases the a values can be computed as in (9) and (13). In case (a) $s_{n+1,j}$ is computed as in (14) and (13). In case (b) the new $s_{n+1,j}$ is computed according to

$$s_{n+1,j} = r_j \tan \alpha_j \tag{15}$$

where r_j is the radius of the hypersphere of the given RBFN unit. Since a regular RBFN region — as opposed to the hyperspherical part of the CSFN — is not truly bounded (see footnote 4), a certain point at the Gaussian, usually multiples of the standard deviation, has to be chosen to define the radius. As discussed in footnote 3, many variations of algorithms to choose the actual size of the decision region exist (e.g. Weymaere & Martens 1991, Mel & Omohundro 1991, Leonard et al. 1992). Thus it further depends on the chosen method, as to what the radius is to be used in (15).

7. Note that in (3) the center coordinates were also depicted as "weights" in a direct comparison with MLPs. Now, in a CSFN, the roles of the parameters in the two network types obtain a different, complementary role (expressed as a and s values).

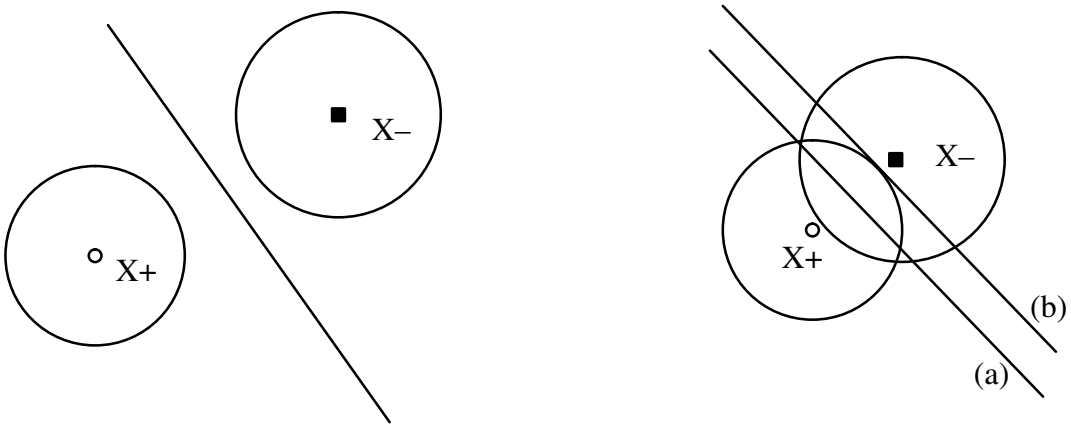

Fig. 6. Two cases of choosing a hyperplane between two cluster centers of an RBFN. If the clusters do not intersect (left) the hyperplane can simply be chosen as the Voronoi tesselation. If they do intersect, there are two choices (right) — either Voronoi (case (a)) or a tangent to the positive cluster circle (case (b)). The size of the circles is given through the RBFN initialization procedure.

5. Learning by adapting the cone ("cone folding")

5.1. Gradient descent

No matter how the cone has been initialized, adapting the parameter ω to fold or unfold the cone in order to improve the corresponding decision regions can be done relatively easily. Consider figs. 3. and 4. Even though only the n-dimensional subspace is relevant for the network, the $n+1$-dimensional cone can be taken as a decision criterion whether a point is either a false positive (FP, i.e. not a member of the class classified as being a member) or a false negative (FN, i.e. a member classified as not being a member). If every net input greater than zero is taken as positive, then according to (8) every point within the cone is positive, while all points outside are negative. To push a FP outside the cone, ω simply has to be decreased to make the cone narrower. Analogously, to pull a FN inside the cone ω has to be increased to make the cone flatter, all within the limits set by the MLP and RBFN ends of the spectrum.

A successful strategy to arrive at a learning rule to adapt ω is to apply gradient descent so as to make each unit an "optimal separator" of the training data. This can be interpreted in several ways. One of them is to maximize a "goodness of separation." This goodness should have a large value if as many as possible negative points are far outside the unit's decision region and at the same time as many as possible positive points far inside. This loosely described property can be formally expressed by summing the distances between all positive points and the decision border (ideally all positive) and subtracting the sum of the distances between the negative points and the border (ideally all negative, if distance is defined as a directed measure). Now it turns out that the net input, even though not identical (as in the MLP case), is proportional to such a distance measure. For this we take a look at figure 7., which — similarly to fig. 4. — shows a cross-section of the cone with a plane

containing the axis ($\boldsymbol{a}$) and a data point X.[8] In this view, d is the desired distance to the decision border. Now recall that the net input was originally defined as (compare (6))

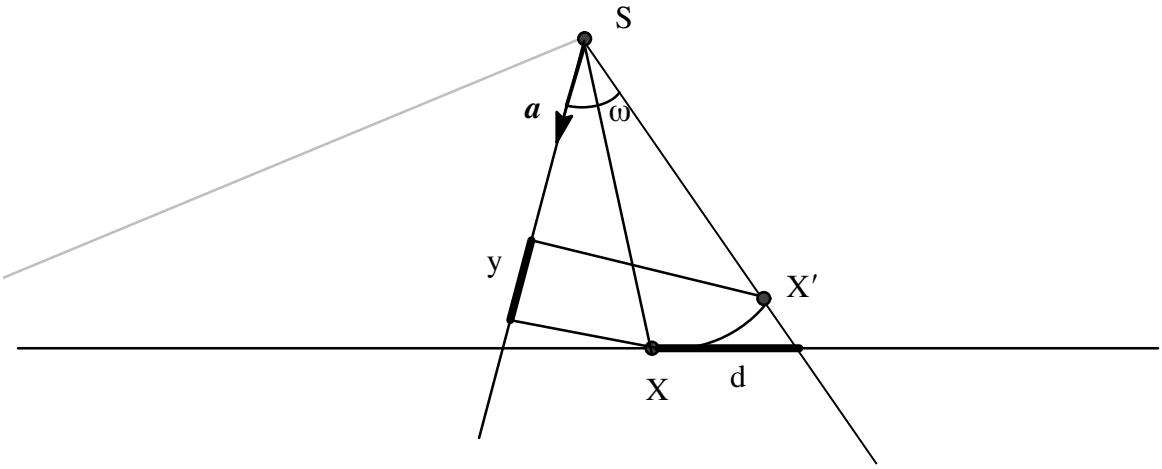

Fig. 7. A geometrical interpretation of the net input y. It is given as the difference between two orthogonal projections ont the axis of the cone; namely the projections of SX and SX'. This figure shows that y is approximately proportional to the distance d between X and the decision border.

$$y = (\boldsymbol{x} - \boldsymbol{s})\boldsymbol{a} - \cos\omega\|\boldsymbol{x} - \boldsymbol{s}\| \qquad (16)$$

This expression can be interpreted geometrically as the difference between the projection of the strectch SX onto the axis and the projection of the stretch SX', where X' is the point on the surface of the cone with the same distance to S as X. Now one can easily see that y is approximately proportional to d, positive for points inside the cone, and negative for points outside, as desired. The closer the point is to the decision border, the better is the approximation (i.e. close to the border the relationship between d and y is approximately linear, farther away it is sub-linear). Since it is these points which should be emphasized most (since they are the most critical ones with respect to separation), y indeed can be used as an estimate for the distance in the goodness value g. Hence, we arrive at a goodness of separation given by the following expression

$$g_j = \left[\sum_{\{l|t_k^l>0\}}^{n} y_j^l - \sum_{\{l|t_k^l<0\}}^{n} y_j^l \right] sign\left(w_{jk}\right) \qquad (17)$$

where l is an index over all training patterns and $t^l{}_k$ is the target value for the l-th pattern on output unit k. w_{jk} is again the weight between the j-th hidden unit and the output unit k, whose sign expresses

8. Note that this cross-sectional plane — as opposed to the view in fig. 4. — in general is not orthogonal to the plane (hyperspace) of the input data.

whether that hidden unit is a positive or negative "detector" (see section 4.2.). The advantage of using y instead of the true distance lies in the much greater ease of computation (see also below).

Now, maximizing g by changing ω can be simply devised as a gradient descent rule by defining

$$\Delta\omega_j \propto \frac{\partial g_j}{\partial \omega_j} \tag{18}$$

which, by using (17) and the fact that y_I is independent from ω_J for all $I \neq J$, reduces to

$$\Delta\omega_j \propto \frac{\partial y_j}{\partial \omega_j}\ sign\left(w_{jk}\right) \tag{19}$$

Some additional restrictions, however, are in place. First of all, ω can only reasonably be changed between the limits $\frac{\pi}{2}$ and $\frac{\pi}{2} - \alpha$, since outside this range the decision regions would degrade to either a tiny ellipsoidic region, or a sub-space that tends to cover most of the input space. Secondly, as a consequence, it is only reasonable to consider points which are on the relevant side of the hyperplane part (the degraded cone at $\omega = \frac{\pi}{2}$). All points at the other side cannot be affected by changing ω. This can easily be decided by looking at the sign of $\bar{y}$ in (10). Finally, it might be a good strategy to exclude points from influencing the decision border which are too far away. This can be accounted for by introducing a threshold θ_ω as an additional parameter. Given all this, we arrive at the learning rule:

$$\Delta\omega_j = \begin{cases} \eta\delta \text{ ... if } y_j > 0 \text{ and } w_{jk} > 0 \text{ and } t_k = 0 \\ \quad \text{or if } y_j < 0 \text{ and } w_{jk} < 0 \text{ and } t_k = 1 \\ -\eta\delta \text{ ... if } y_j < 0 \text{ and } w_{jk} < 0 \text{ and } t_k = 1 \\ \quad \text{or if } y_j < 0 \text{ and } w_{jk} < 0 \text{ and } t_k = 1 \\ 0 \text{ ... otherwise} \end{cases} \tag{20}$$

$$\delta_j = \bar{y}_j \sin(\omega + \alpha) + s_{i,n+1} \cos(\omega + \alpha) + \bar{\bar{y}}_j \sin\omega_j$$

$$\omega_j := \begin{cases} \omega_j + \Delta\omega_j \text{ ... if } \Delta\omega_j > \theta_\omega \\ \omega_j \text{ ... otherwise} \end{cases}$$

δ_j is the first derivative of y_j, as written in (10), t_k is the target of the k-th output unit, and η is a learning rate. The sign of w_{jk} in (17) and (19) is explicitly accounted for by making a case distinction.

Learning rule (20) would only be correct if the update of ω were done in a "batch mode" (compare Rumelhart et al. 1986), i.e. only after all derivatives computed for all training data have been summed up (i.e. the sum in (17)). Experience shows that updating ω after each presentation of single data points can also lead to satisfying results.

5.2. Maximally excluding false positives

Gradient descent as defined above might not be the optimal strategy. Like in many applications of RBFNs, an alternative strategy might be to cover the positive samples through decision regions as well as possible, excluding false positives. Such a strategy — henceforth called 'remove FP' — can easily be derived from (20) by only allowing changes in the case of false positives. Thus the learning rule reduces to

$$\Delta\omega_j = \begin{cases} \eta\delta \ \ldots \text{ if } y_j > 0 \text{ and } w_{jk} > 0 \text{ and } t_k = 1 \\ -\eta\delta \ \ldots \text{ if } y_j < 0 \text{ and } w_{jk} < 0 \text{ and } t_k = 1 \\ 0 \ \ldots \text{ otherwise} \end{cases} \qquad (21)$$

This applies only to CSFNs that start with unbounded regions (e.g. MLPs) that can be closed. For CSFNs starting with bounded regions, this rule would have to be adapted appropriately.

5.3. Mixing training strategies

We have seen that cone folding is most appropriate for choosing optimal decision borders based on a previously trained or initialized conventional network. However, since in (8) the two major parts are essentially independent from each other, learning strategies like backpropagation (gradient descent on the a or w–values, respectively) can also be applied after cone folding. The main purpose could be fine-tuning of near-optimal decision regions. The effect is to slightly alter the hyperplane part of the cone while leaving the tip fixed.

Another way of mixing strategies could be to use gradient descent for more than one set of degrees of freedom (e.g. a and ω, or even a, s, and ω). However, due to the computational complexity of such a technique (which, even more than regular backpropagation, would be prone to get stuck in local minima) it would again only be advisable to use it for fine-tuning. The major progress in improvements should come from the relatively little complex process of cone folding.

6. Transfer functions

So far we have only considered the propagation rule of a CSFN, i.e. the rule replacing the dot product in an MLP and the Euclidean distance in an RBFN. The second important part of the computation in a neural network unit – the transfer function (see section 1.) – has been ignored, or better still, has been implicitly assumed as being a threshold function.

Even though a CSFN unit equally includes both the MLP and the RBFN propagation rule, the main overall characteristics is more similar to the MLP case. With this we mean that "better" input points

— i.e. points which are farther away from the decision border — lead to larger net input y_j. Therefore a monotonically increasing transfer function, such as the sigmoid $\frac{1}{1 + e^{-x}}$, is most appropriate to lead to large responses for points that are "farther within or outside" the respective decision regions. In the RBFN, inversion has to take place, since large responses are desired for *small* Euclidean distances (i.e. small net inputs). This inversion is already implicitly contained in the CSFN (see (12)).

In the spirit of the above mentioned alternative combinations of the dimensions defining well-known feedforward networks (see section 1.), a Gaussian could also be used as transfer function, leading to linear, spherical or intermediate hyper-"ridges" as decision regions, which could be useful for some given data sets. In summary, we see that — against implicit wide-spread belief — the type of transfer function (e.g. sigmoid vs. Gaussian) is not so much tied to the general characteristics of the propagation rule (e.g. hyperplane vs. hypersphere). In the case of a CSFN, the sigmoid appears most suitable, while the propagation rule includes both major decision regions. As a result, in the unification of the two conventional networks presented here no approximation of one transfer function by the other has to be sought (as was done, for instance, in Maruyama et al. 1992, Geva & Sitte 1992).

Practical experience has shown that best results come from choosing a "steep sigmoid", i.e. a sigmoid , $\frac{1}{1 + e^{-cy}}$ in which y is scaled with large c so as to approach the function toward a heaviside function.

7. Pruning

Given the larger number of parameters than in any of the conventional network types, methods to reduce unnecessary complexity of a CSFN are desirable. There are two major ways how such pruning can be achieved.

First of all, cone folding will not in every case lead to units with decision regions which are intermediate between hyperspheres and hyperplanes. Therefore, for all units j whose ω_j is within an error tolerance ε of either $\frac{\pi}{2}$ or $\alpha - \frac{\pi}{2}$, the s values (a values) can be discarded turning those units esentially into MLP (RBF) units. The result is a mixed hidden layer with units of three types (MLP, RBF, intermediate) – compare Weymaere (1993).

Secondly, based on a suggestion in Smyth (1992), hidden units with redundant decision regions can be discarded. A decision region is redundant if (within a certain tolerance) it is subsumed by another decision region. In a CSFN this can, for instance, happen when among two bounded regions one opens so as to include the second one. Two strategies to make such a decision can be conceived.

– probing on all training points whether two units are completely correlated with respect to the sign or their net response. Of course, this strategy is open to errors for sparse training sets.

– computing analytically whether one decision region is subsumed by another. Given the large number of possible pairs and the effort involved in making each single decision, this strategy can be very computation-intensive (however, it is error-safe).

8. Examples

8.1. XOR

In a trivial way the effects of cone folding can be demonstrated with the famous XOR-problem. Fig. 8. shows the four points of the problem, black dots indicating a target output of 1, white dots

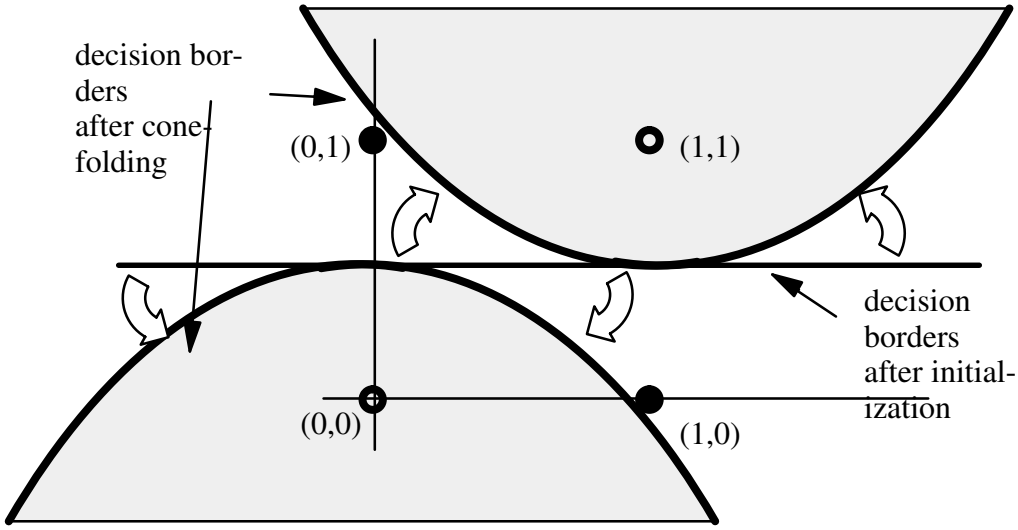

Fig. 8. The four points of the XOR problem and the solution achieved by initialization and subsequent cone folding. MLP Initialization according to (13) on the pairs (0,0)/(1,0) and (1,1)(0,1), respectively, in both cases leads to the straight line as decision border for two hidden units. They differ only in that for the first decision region the positive half-space is below the line, while for the second one it is above. Cone folding, in a few steps changes the lines to two hyperbolas (bold, with grey areas as positive regions) leading to a solution of the problem.

a target of 0. If a 2-2-1 MLP is initialized according to the above method (13) by taking the two input pairs (0,0)/(0,1) and (1,0)/(1,1) one twice obtains the straight line y=0.5, once with a positive region above that line, once below. Of course, this is not a solution to the problem. However, if a corresponding CSNF is inialized with (1,0) and (0,1) as the positive exemplar for hidden unit one and two, respectively, learning rule (20) can quickly change the decision regions to obtain a solution. For hidden unit one and its half-plane above the line the point (1,1) is an FP. Therefore, the corresponding ω_1 is decreased and the line turns into a hyperbola. This is done until the point is pushed outside that decision region (it could even happen in one step). The same happens for the other line (which was identical with the first one to begin with. Thus cone folding very quickly changes the decision regions so as to achieve the desired XOR output (see fig. 8.).

8.2. A complex 2-dimensional problem

To illustrate and test the capabilities of cone folding in a CSFN, a two-dimensional data distribution has been created (figure 9.). The points in this data set are divided into two classes (call them 'class

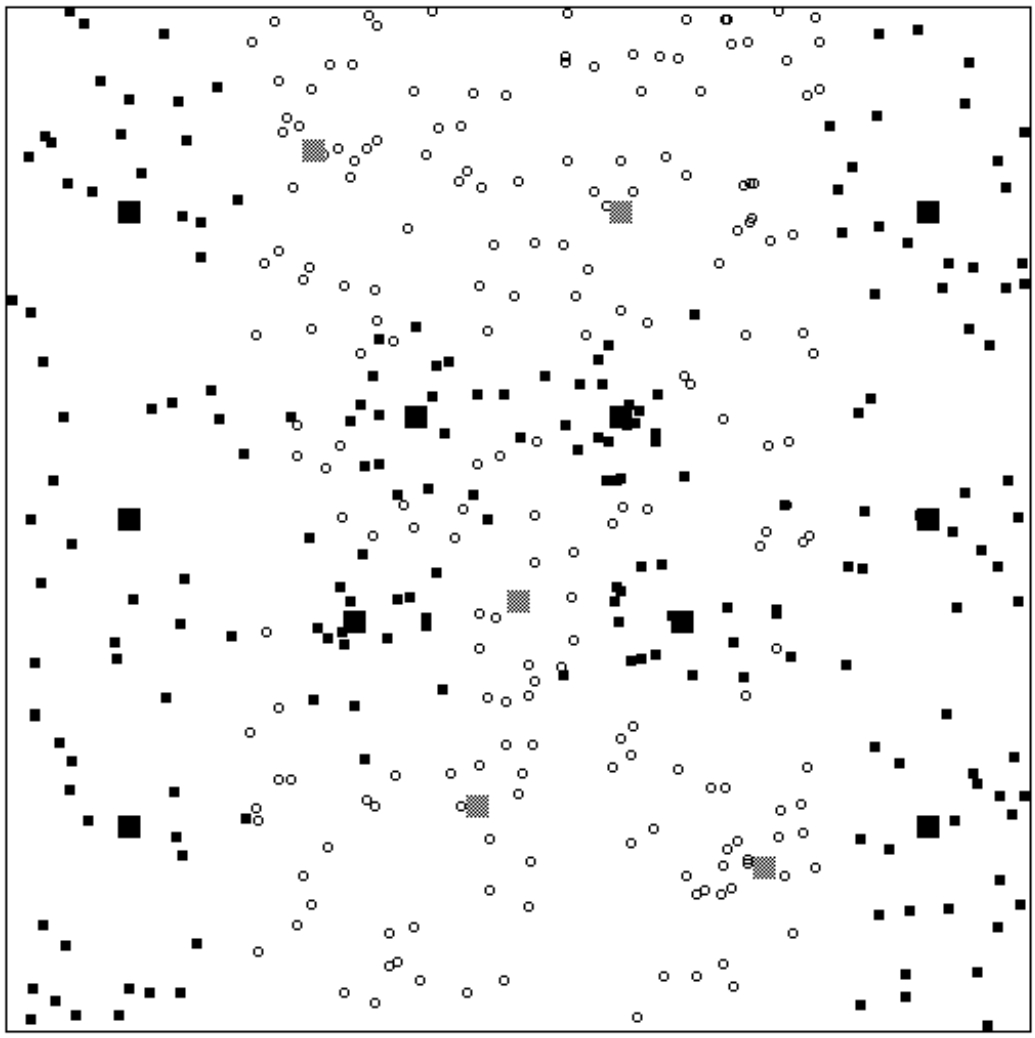

Fig. 9. A complex 2-dimensional data distribution with points from two classes (class 1, depicted as little black squares, and class 0, depicted as white circles). Input x_1 is drawn along the horizontal axis (ranging from –2.5 to 2.5), and x_2 is drawn on the vertical axis (also ranging from –2.5 to 2.5). The origin of the coordinate system is the center o the depicted overall region. Large black squares depict cluster centers derived from an ideal cluster analysis of all points in class 1, whereas grey squares depict clusters of class 0.

1' — depicted as black squares — and 'class 0' — depicted as white circles). Class 1 was defined by four Gaussian distributions (namely around the points (0.5,0.5), (0.8,–0.5), (–0.5,0.5), and (–0.8, –0.5) with standard deviations of 0.3, 0.5, 0.35, and 0.4, respectively) and two even distributions (defined by $x_1 > 1.5$ and $x_1 < -1.3$ where x_1 is the activation of the first input unit — depicted on a horizontal axis in fig. 9.). Points of class 0 are evenly distributed elsewhere (i.e. for

$-1.3 \leq x_1 \leq 1.5$ and more than the standard deviation away from the positive Gaussian clusters). 400 such points have been randomly created as training set (fig. 9.) and 400 more as test set. For both inputs only values between –2.5 and 2.5 have been computed (and depicted). Note, however, that the definition of the data distribution applies to all points in $\mathbb{R}^2$and thus defines unbounded regions in both classes.

An idealized cluster analysis was assumed which identifies the clusters of the four Gaussian distributions and three equidistant points in each of the even distributions. Similarly, five cluster centers for class 0 were chosen (also depicted in fig. 9.). Note that this is of course the optimal case which cannot be guaranteed in real applications. This case was chosen to illustrate the effects of cone folding. We will, however, discuss non-optimal cases below.

For the following experiments, 10 or 20 hidden units were chosen and the method of initializing the CSFN based on an initialized MLP was applied (section 4.1.). In the case of 10 hidden units, for each hidden unit one of the ten clusters of class 1 was chosen with the closest cluster of class 0 as the pair X_j^+ and X_j^-. In the case of 20 hidden units, each positive cluster center formed a pair with each of its two closest negative clusters. The learning rate η was set to 1, and the threshold θ_ω was varied between 0.1 and 1.0 (the latter meaning that ω was changed every time). Each training step consisted of the presentation of one randomly chosen input vector from the training set, update of hidden and output units (regular MLP or CSFN) and one learning step with the following delta rule between hidden and output layer (learning rate η=0.1, no transfer function at the output units):

$$\Delta w_{jk} = \eta x_j (t_k - y_k) \tag{22}$$

where t_k is the target of the k-th output unit.

Figures 10., 11., and 12. show the results, plotted as learning curves (summed squared error over training steps) for a CSFN with varying number of cone foldings, as compared to a regularly initialized MLP (using the same cluster pairs for initialization). The dotted line on top always depicts the regular MLP case. Thus, in each case the CSFN outperformed the MLP considerably (increasing dash size in the learning curve indicates increasing number of cone foldings). It should be noted that for each experiment — to be fair — the regular MLP was initialized and its hidden-to-output connections trained by the delta rule again. Thus, the dotted curves indicate several independent training runs (the paramters varied in figs. 10. through 12. do not affect the MLP) with slight variations in performance. Nevertheless, the worst result from the CSFN was always better than the best result from the MLP. In all cases, regular backpropagation with randomly initialized weights was also performed. In none of the experiments, however, it achieved performance below a summed squared error of about 100.0, in the time constraint of 2000 training steps. This is due to the fact that the learning problem is hard given the complex data distribution. Thus both initialized MLP and CSFN considerably outperformed regular backpropagation in terms of learning speed. It is interesting to note that in the case of 10 hidden units the CSFN always starts off better than the MLP, while in the case

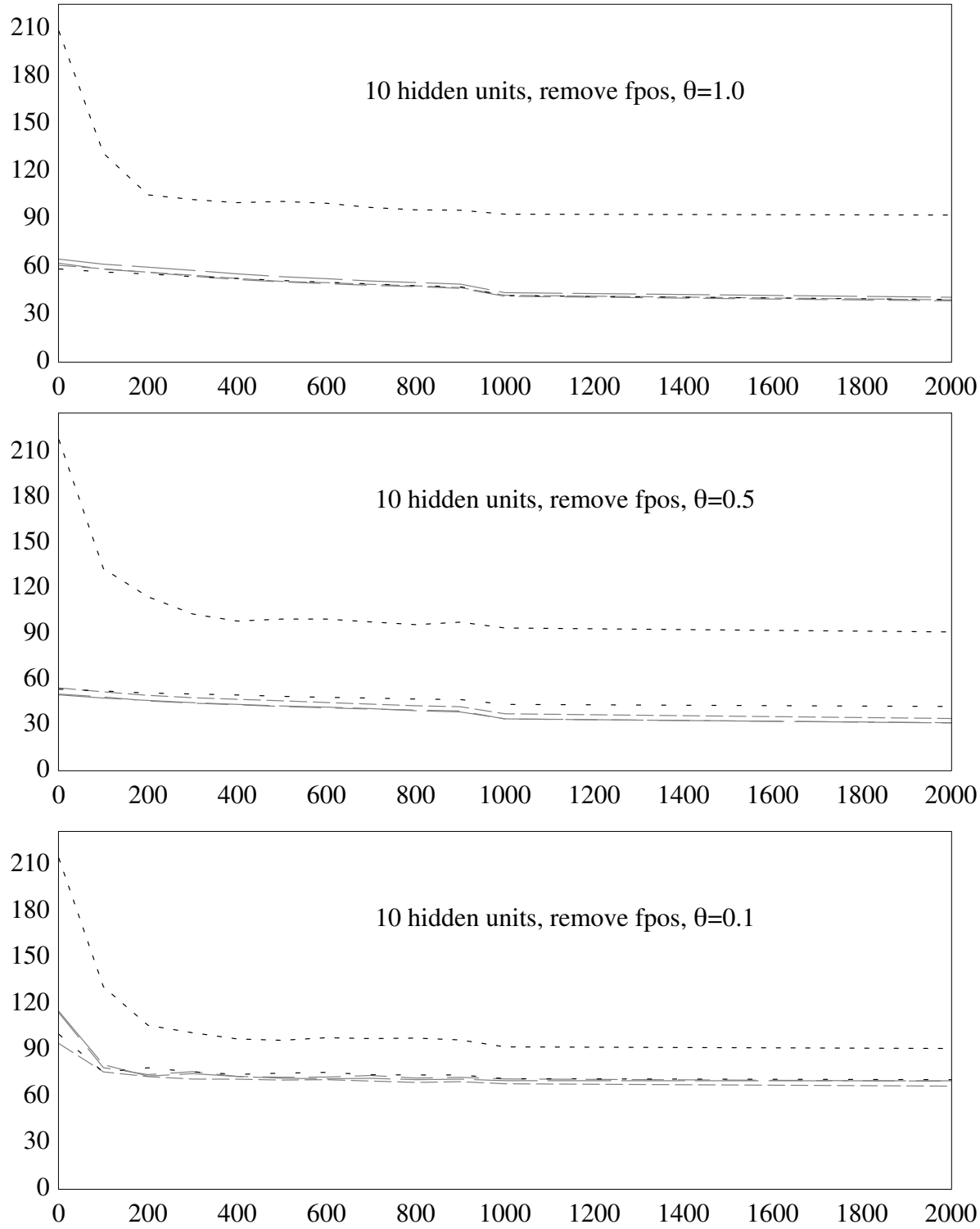

Fig. 10. Learning curves (summed squared error over learning steps) of the **'remove FP' strategy** with varying number of cone folding steps and threshold θ. The dotted line on top is the learning curve of regular MLP initialization + delta rule. Increasing dash size indicates increasing number of cone foldings.

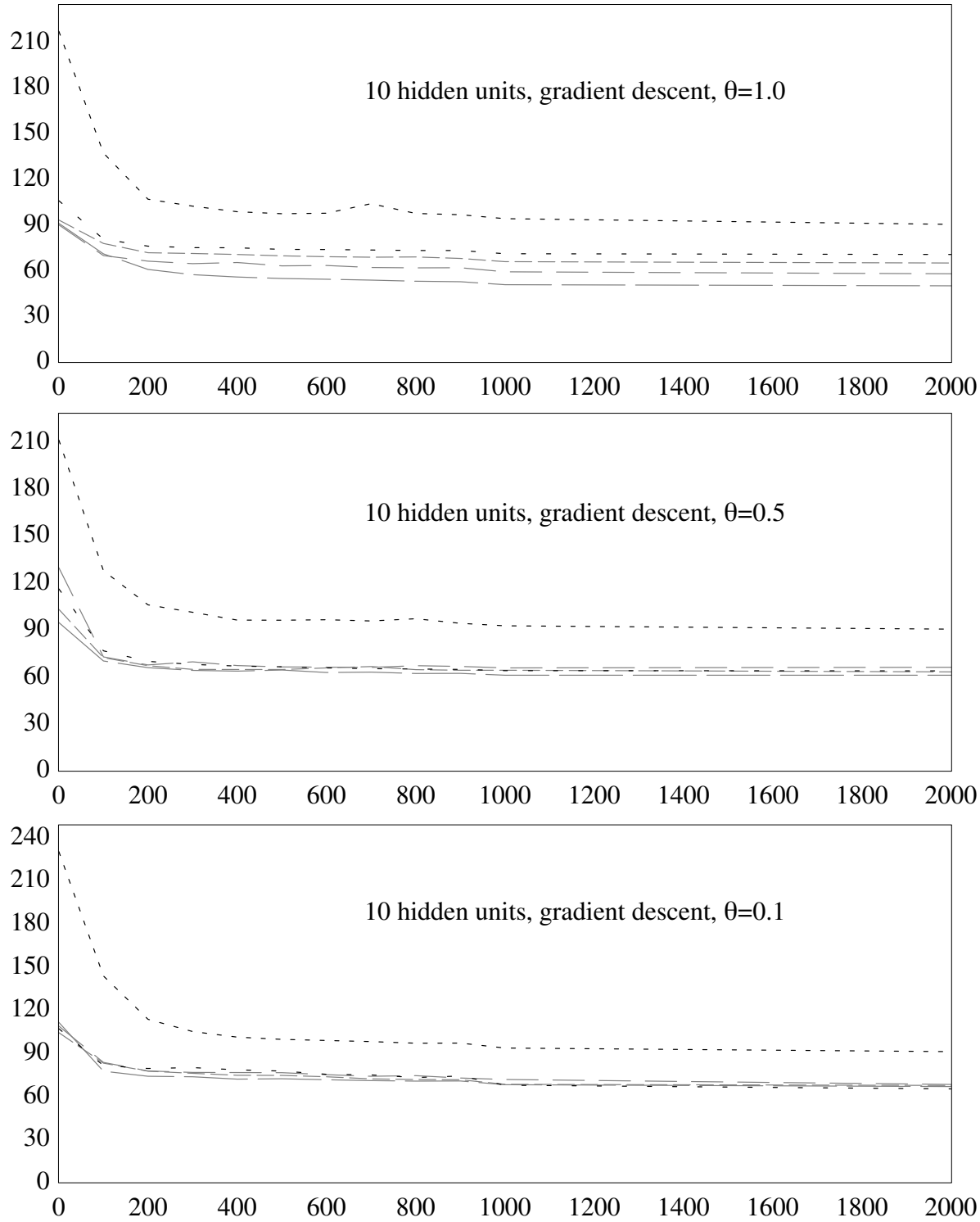

Fig. 11. Learning curves (summed squared error over learning steps) of the **gradient descent strategy** with varying number of cone folding steps and threshold θ. The dotted line on top is the learning curve of regular MLP initialization + delta rule. Increasing dash size indicates increasing number of cone foldings.

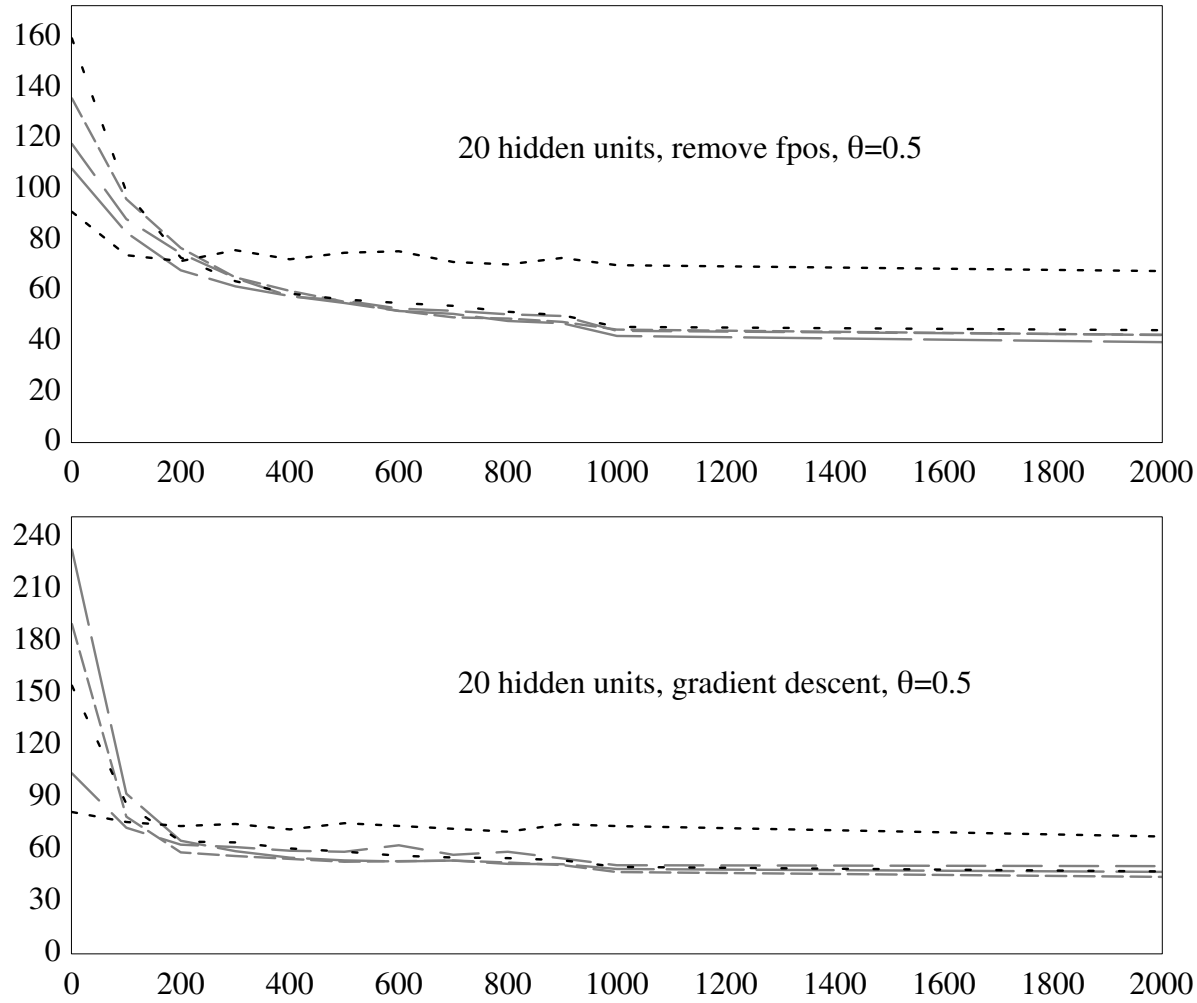

Fig. 12. Learning curves (summed squared error over learning steps) of the gradient descent and 'remove FP' strategies **for a network with 20 hidden units**, with varying number of cone folding steps and threshold θ. The dotted line on top is the learning curve of regular MLP initialization + delta rule. Increasing dash size indicates increasing number of cone foldings.

of 20 hidden units its performance is worse initially (which changes after a few 100 training steps, though).

The figures show that convergence of cone folding (in terms of improvement) is reached very quickly with sometimes even one or two cone foldings. This quick learning over as many parameters as there are hidden units must be contrasted with the learning over all weights in the regular backpropagation. The choice of θ_ω seems to be of little effect, at least in this learning problem.

Fig. 13. depicts the resulting decision regions of the best CSFN and the best MLP (the grey area corresponds to the region where the network responds with positive output (classifies the input as class 1). One can see that the CSFN indeed very closely captures the structure of the input space while the MLP can give only a crude approximation. In particular, the two unbounded regions at the left

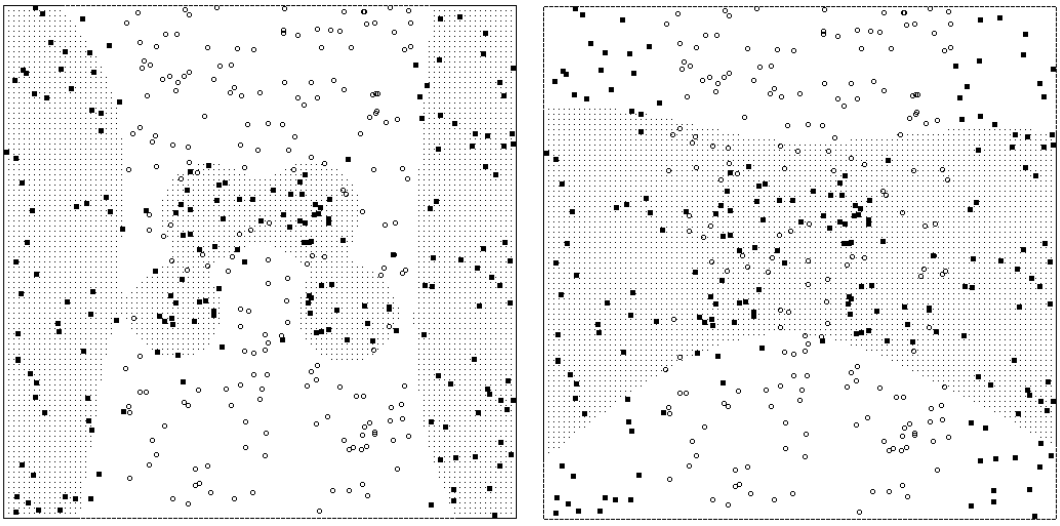

Fig. 13. The final decision regions of a CSFN (left) and an initialized MLP learning with the delta rule (right). Points with positive responses are highlighted as a grey area.

and right edges of the depicted sub-space (given through the above definition) are nicely discovered by the CSFN, while the MLP would make more "incorrect" extrapolations. Of course, this need not always be desirable or optimal in every application. But through the tendency to leave decision regions unbounded in lack of evidence against it, and to close them if such evidence is encountered the CSFN seems to be capable of more optimal extrapolation for many complex real-world problems. No doubt that the MLP can achieve the same decision region (see again Hornik 1991, Hornik et al. 1989) but with more hidden units and perhaps more training data.

Fig. 14. depicts the change of the decision regions of two particular hidden units when cone folding is done with the 'remove FP' strategy. On can see that indeed it closes over bounded regions of positive points and remains unbounded if no evidence of a false positive point in its decision region exists. This leads to the above generalizations to regions outside the depicted sub-space. In the case of gradient descent (fig. 15.) decision regions stop changing when a balance between positives and negatives (given to the goodness-of-separation in (17)) is reached. For this classification problem this gives slightly worse results. However, in real applications it might never be the method of choice since the 'remove FP' strategy can be too restrictive when the preceding cluster analysis did not cover all input space equally.

An alternative to using cluster center (which — as can easily be seen — can be very disadvantageous when cluster analysis fails to cover all the input space) is choosing random training samples for initialization. In the case of the two-dimensaional problem, 20 positive and 20 negative samples were taken (distributed across the clusters according to the number of occurences) randomly and chosen as pairs for initalizing a CSFN, again starting off as an MLP, with 20 hidden units. Fig. 16. shows the results. One can clearly see that the gradient descent strategy can still outperform the initialized MLP, while the 'remove FP' strategy does not lead to improvements at all. This is interesting with at least two respects. First, it shows great promise for the CSFN for practical applications, even if

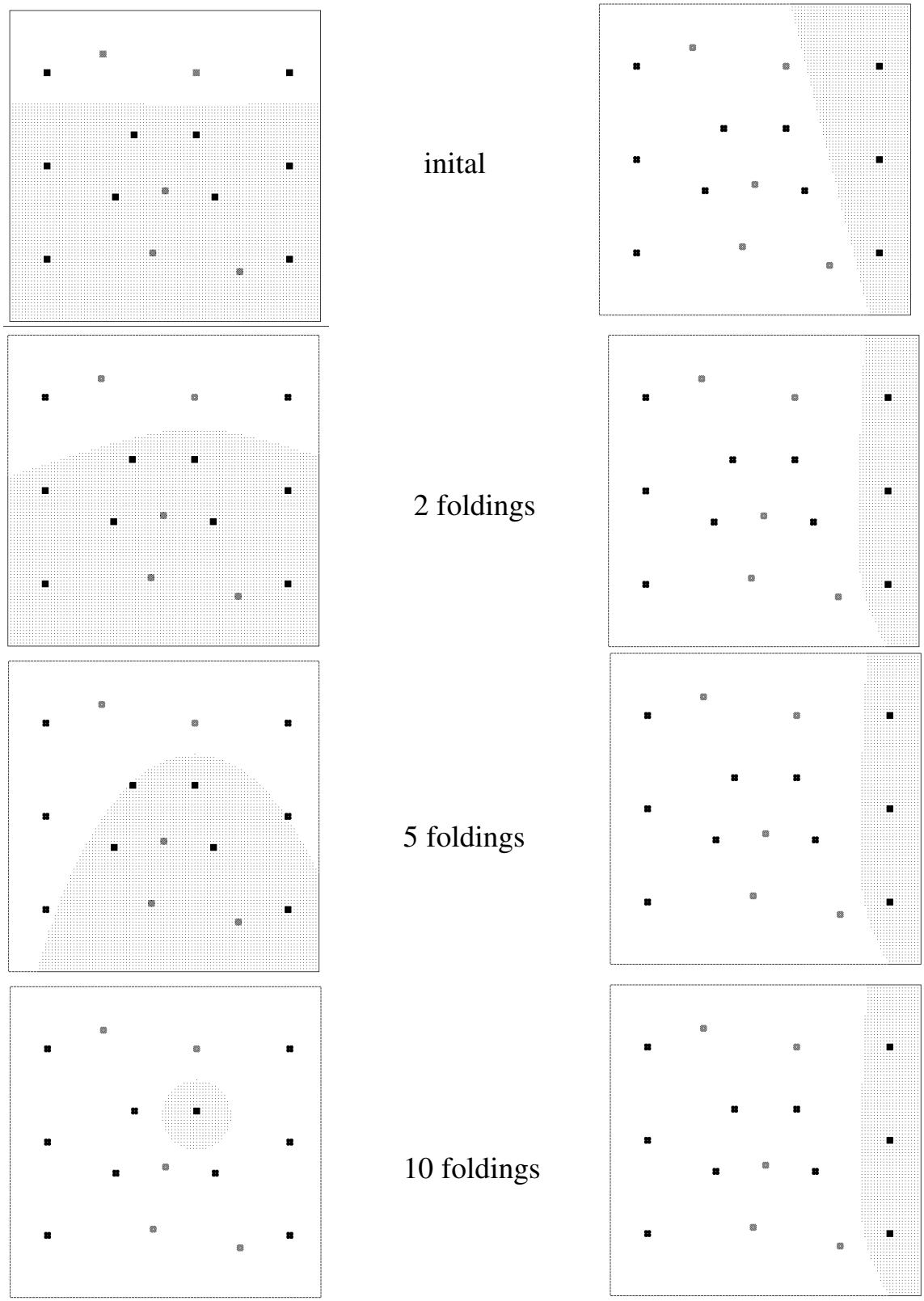

Fig. 14. The changes in the decision regions (grey area for positive response) of two hidden units — a unit restricting itselg to a bounded region (left), and a unit remaining open since it encounters no evidence against an unbounded region

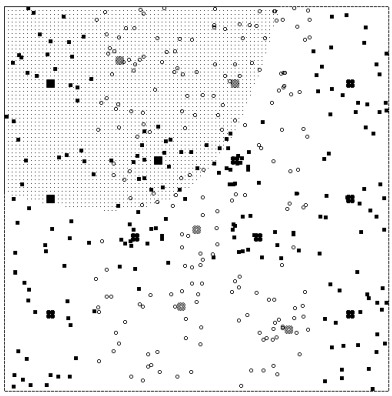

Fig. 15. The final decision region of a unit learning with gradient descent over ω. The folding of the cone stops when there is a balance between positive points inside and negative ones outside (only for points on the "relevant side" of the cone).

successful cluster analysis cannot be guaranteed. Here, cone folding does not lead to a perfect approximation of the input distribution as in fig. 13. but instead to an optimization of all decision regions with respect to separation. Secondly, it shows that the two strategies (gradient descent and 'remove fpos') can complement each other. Data analysis or experimentation will help in finding out which one to apply for a given application.

8.3. Detecting Coronary Artery Disease in heart scans

The above examples mainly served to prove and to illustrate the capabilities of CSFNs and cone folding. Apart from that, CSFNs have also been successfully employed to improve the results of several real-world applications currently under development. One of them, (described in more detail in Dorffner & Porenta 1993) is the detection of so-called coronary artery disease (CAD) in thallium-201 scintigrams (scans) of the human heart. The input were several coarsely segmented scans from different views and at different points in time, plus washout values. This resulted in an input vector of 45 values, scaled as to roughly lie in the range [0 .. 1] (however, there were a number of cases lying considerably outside). The target output was obtained by angiography (an invasive method) as gold standard. Different networks, namely MLPs with backpropagation, RBFNs, and CSFNs — all of them with 15 hidden units and one output unit indicating presence of CAD — were trained. Five (almost) independent different training set, which had the disadvantage of being rather small and unbalanced, were used and thus lead to five different comparisons. The results showed that standard backpropagation was consistently outperformed, while in 3 out of 5 cases CSFNs lead to the most improvements. Fig. 17. shows the results from two of the best of these cases, plotting a false positive over false negative curve for the four different networks (MLP with backprop, initialized MLP, RBFN, and CSFN). This type of curve is of vital importance in medical applications indicating how well the system can detect pathological cases ("sensitivity") while still maintaining a high rate of correctly classified normal cases ("specificity"). The results showed that CSFN could help in making this curve flatter (meaning a higher sensitivity for high specificities), making this neural-

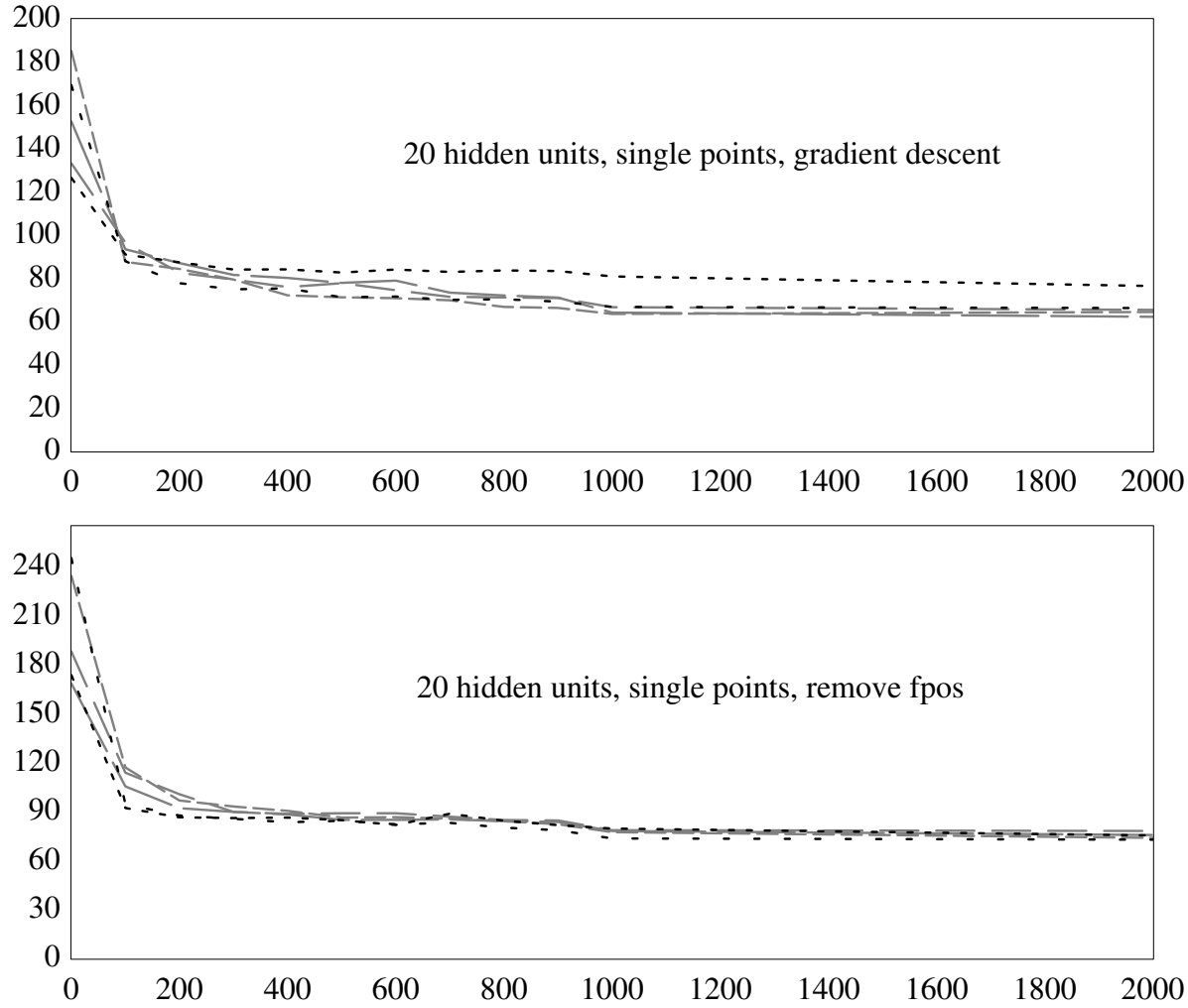

Fig. 16. Learning curves (summed squared error over learning steps) of the gradient descent and 'remove FP' strategies for a network with 20 hidden units, **initialized by single randomly chosen training points**, with varying number of cone folding steps. The dotted line on top of the above diagram is the learning curve of regular MLP initialization + delta rule. Increasing dash size indicates increasing number of cone foldings.

network based detection clinically useful (previously, using only MLPs with backpropagation, the results on the networks trained with the data from angiography, failed to be useful).

9. Discussion

As the examples have shown, a CSFN is indeed capable of making automatic decisions with respect to bounded and unbounded decision regions. This way, it can better approximate the function given by the desired classification. Since both MLPs and RBFNs have been proven to be universal approximators (Hornik et al. 1989, Hartman et al. 1990), the actual advantage of CSFNs can come from one of the following aspects:

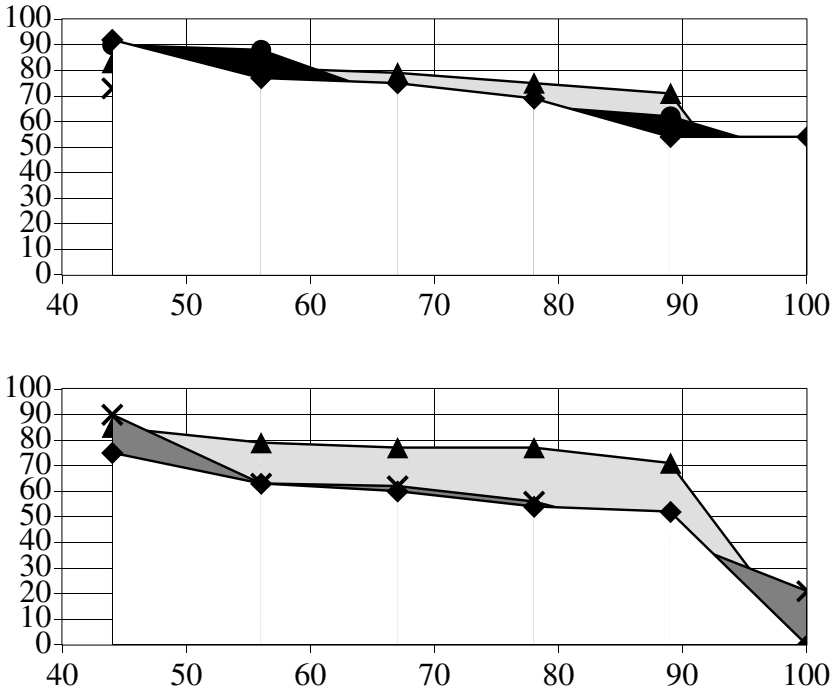

Fig.17. Comparative results from applying four networks (MLPs with backprop — white/diamonds, RBFNs — black, initialized MLPs — dark grey/crosses, and CSFNs — light grey, triangles) to the problem of detecting CAD in heart scans. The curves show sensitivity values (correct positives) on the vertical axis, and specificity values (correct negatives) on the horizontal axis, both in percent. In both cases, CSFN help to make the curve considerably flatter, as is desired in medical diagnosis. For the above curve, CSFNs and MLPs were initialized by single training point pairs, for the lower curve cluster centers were used.

– Learning to within the same error as a plain MLP can be quicker, since cone folding involves fewer degrees of freedom in a smaller search space (the space of possible omegas).

– Learning to within the same error as a plain MLP or RBFN can be achieved by a smaller number of hidden units, since decision regions are more complex.

– Generalization can be better since the adaptive shape of decision regions can better account for peculiar shapes in the input space.

In this sense, CSFNs cannot be more powerful in the theoretic sense (in that they can approximate more function classes than MLPs or RBFNs) but they might offer a number of advantages in practical applications concerning efficiency, speed, and generalization. Of course, given an application, one cannot say whether CSFNs will lead to improvements or not. This points to a need for extensive data analysis before CSFNs are applied. However, this need is even prevalent — although largely overlooked by practitioners — when applying the standard networks MLP or RBFN. As we have argued in the beginning, some data distributions might be more approrpiate for bounded decision regions, others for undbounded ones. Furthermore, for initialization — which can be extremely powerful also for the MLP, as has been shown — knowledge about clusters and their distribution

is a must. Since it unifies all these frameworks, a CSFN has the potential to provide a decisive step forward to more optimal neural network solutions with respect to a given application. Its special appeal comes from the ability to make some of the decisions about decision regions automatically. Further research on optimizing the procedure of finding good initializations will, however, still be needed.

Another interesting aspect worth mentioning is the idea of localized adaptation (Chui et al. 1992, Omohundro 1989), briefly discussed in section 2. The notion of 'localized' in this context need not be refined to bounded decision regions. In the examples above we have seen that unbounded regions, when covered by one or a few hidden units would also qualify as localized with respect to needed changes. Thus CSFNs might have the capability of being more easily adaptible in changing environments. This will have to be evaluated in the future.

Apart from those practical aspects, CSFN provide for one coherent way of depicting the widely–used MLPs and RBFNs in the same framework, as aimed at in the beginning of this paper. No normalization of the input vectors has to be assumed (as in Maruyama et al. 1992, Denoeux & Lengelle 1993) to map the two networks onto each other. Instead of showing their principal equivalence, CSFNs give a generalization containing both networks as special cases, each with their respective advantages. It is interesting to take a look at an intuitive reason behind the required normalization in Maruyama et al. 1992. When all input vectors are normalized to constant length, the inputs space is restricted to a subspace of finite size (i.e. points on a hypersphere). It is obvious that every half-hyperspace in a finite space (which itself is no longer infinite but finite) can be approximated by hyperspheres reasonably well, while in an infinite space an infinite number of hyperspheres might be needed to include all possible input points. Along those lines, someone might argue that "truly infinite" decision regions are not needed (apart from the advantage of using fewer hidden units in many cases). CSFNs can work with both types of decision regions in a potentially infinite (unnormalized) input space. During cone folding a region can remain unbounded (or open itself) as long as there is no counterexample lying on the main direction of opening (the axis of the intermediate hyperbolas). Thus they can generalize to any point in the infinite region, given the lack of contradictory evidence in the training set.

This aspect is even more important since normalization often is not desirable or even possible in practical applications (see also Dorffner 1992, Dorffner & Wiklicky 1992). Normalization can change or even erase potentially relevant information in the input data. It can also compress relevant regions into tiny transformed regions, possibly too small for being separated by a practically implemented network (due to reasons of accuracy of data or algorithms).

An aspect with similar consequences is scaling of the input data. In many practical applications, training data with strongly differing ranges are scaled so as to lie in equal intervals (usually [0..1] or [–1..1]). If this can be done completely for all data, the input space would again be finite.[9] However, in many applications the size of ranges (and therefore the scaling transformation) cannot be foreseen. Thus, scaling might still be useful, but data points can also lie outside of the intervals. This

9. In fact, the input space would already be finite before scaling, simply through the assumption that each input dimension has a maximum and a minimum value.

is another example where truly infinite decision regions are needed (or can at least be a helpful concept).

As mentioned in section 2., Omohundro (1989) has suggested another way of viewing hyperplanes and hyperspheres as equivalent decision borders. He suggested to introduce an additional dimension (just like it was done here) and to use the sum of the squares of the other *n* input dimensions as the additional input. This can be interpreted geometrically as introducing a hyper–paraboloid (figure 18.) which intersects the hyperplane of the hidden unit. Now if one draws the projection of the inter-

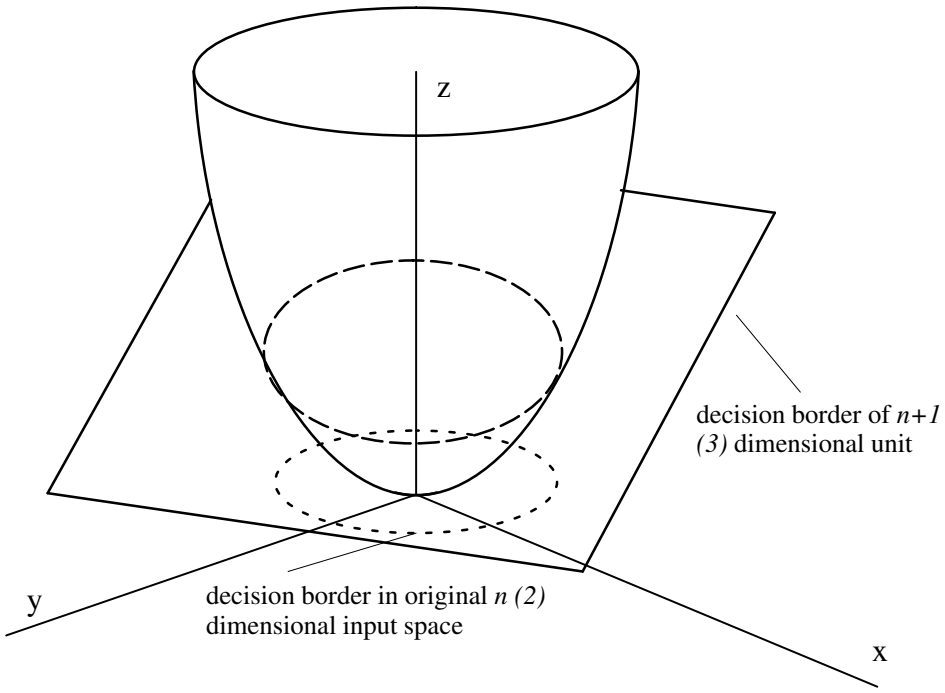

Fig. 18. A geometrical interpretation of Omohundro's suggestion (redrawn from Omohundro, 1988) of using an additional input dimension, which is identical to the sum of squares of the other n dimensions. In the 2-dimensional case this creates a paraboloid wich, when intersected by the plane of the original MLP unit (dashed ellipsis) and projected onto the original space, transforms an infitie half-plane into a finite circular region (dotted circle).

section onto the original n-dimensional space, every point that lies one one side of the hyperplane comes to lie within the projected hypersphere, every point on the other side comes to lie outside. This is how an infinite half-space is projected onto a finite hypersphere. The general idea behind this is very reminiscent of the technique employed in this paper to arrive at a CSFN unit (although developed independently). However, there are important differences.

First of all, the propagation rule of this Omohundro unit can bewritten as

$$y_j = \sum_{i=1}^{n+1} x_i w_{ij} \tag{23}$$

$$= \sum_{i=1}^{n} x_i w_{ij} + w_{n+1,j} \sum_{i=1}^{n} x_i^2$$

which, adding an arbitrary constant as threshold, turns into the exact same form as used in Maruyama et al. (1992) (equation (5)) to show the equivalence between MLP and RBFN. Therefore, the two approaches can be viewed as identical.

Secondly, the Omohundro unit was not suggested as combining both types of decision regions (as is done in the CSFN) but rather as mapping one of them onto the other (again in the same spirit as in Maruyama et al., 1992, or Denoeux & Lengelle, 1993). Nevertheless, the propagation rule in (23) could be extended to provide another such unification (Omohundro, Smolensky, personal communication). By introducing a scaling factor $\beta_j = w_{n+1,j}$ in (23) one could make the influence of the hyperspherical part variable.

By changing β_j, the impact of the paraboloid is varied. If $\beta_j = 0$, only the hyperplane remains. The major difference to the CSFN is that there is no smooth transition between the two cases but an abrupt one. As long as $\beta_j > 0$ the resulting decision region in the original n-dimensional space is exactly hyperspherical, only at $\beta_j = 0$ it degrades to being the linear halfspace. In other words, all decision regions for $\beta_j > 0$ are bounded (albeit arbitrarily large, see fig. 19.). Also, the mappings between errors, distances and the defining parameter (β in this case, instead of ω) is not as obvious as before. Nevertheless, the network as suggested by Omohundro might be able to similar approximations as the CSFN in selected applications.

A comparison with other approaches to include more than one type of decision regions in one hidden layer (such as Weymaere 1993, Roy et al. 1993) reveals that the advantage of CSFNs is its localized learning rule and lesser dependence on heuristic design algorithms. Nevertheless, a combination of all these ideas might improve neural network based classifiers even further.

10. Conclusion

In this paper we have introduced a framework which unifies and generalizes the well-known concepts of multilayer perceptrons (MLPs) and radial basis function networks (RBFNs). This framework was introduced in the form of a novel network, called conic section function network (CSFN), which was designed so as to form a continuum between the two special cases MLP and RBFN, what their propagation rules are concerned. This continuum is controled by a single parameter which can be changed (i.e. the cone can be “folded” or “unfolded”)automatically, depending on the training data distribution, through simple learning rules. We have demonstrated the capabilities of CSFNs

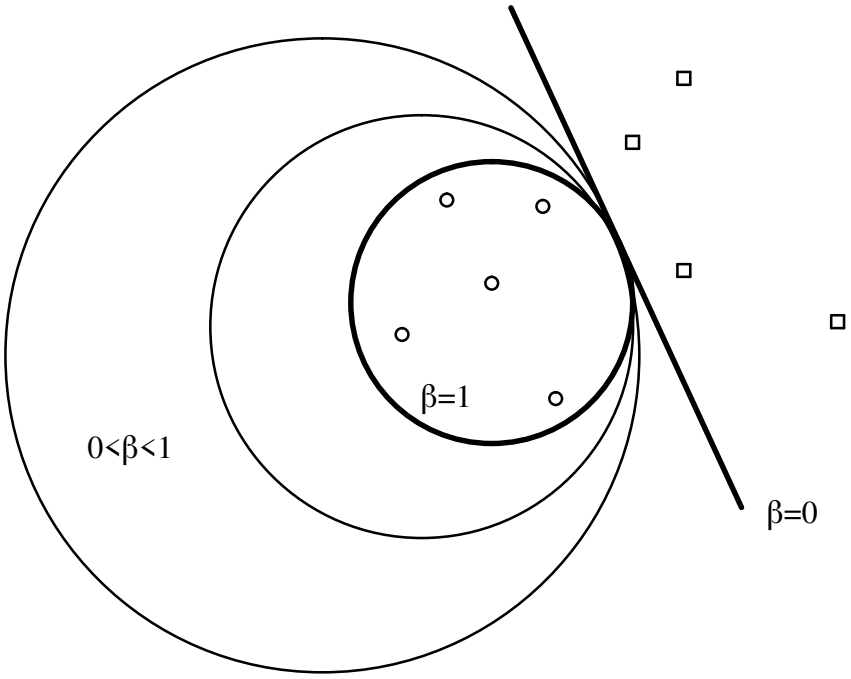

Fig.19. Different decision regions in the original n-dimensional space when varying β ($w_{n+1,j}$ in (23)). All regions are circles except when $\beta = 0$ (compare with fig. 2.)

in several examples. The power comes from permitting both bounded and unbounded decision regions on a continuous spectrum and thus from being able to approximate input distributions more smoothly. In many cases this can have decisive advantages with respect to generalization, efficiency and learning speed. Thus CSFNs are a further step toward more optimized practical neural network solutions.

Acknowledgments

This work has been done as part of the EC-funded ESPRIT-II project 5433 "NEUFODI—Neural Networks for Forecasting and Diagnosis Applications." The Austrian contribution to this project has been funded by the Austrian Industrial Research Promotion Fund. I wish to express my gratefulness to Prof. Robert Trappl for his encouragment and inspiration, as well as support for this work. I further thank my colleagues from the Neural Network Group at the Austrian Research Institute for Artificial Intelligence. I am indebted to Herbert Wiklicky for tremendous mathematical support and for pointing me in the right direction to begin with; furthermore Erich Prem, Claudia Ulbricht and Günter Linhart, for fruitful discussions and many hints that have pushed this research beyond a possible standstill. People like Thierry Denoeux, Stephane Canu, Nico Weymaere, Paul Smolensky, and Stephen Omohundro have all contributed to the outcome of this research, be it through helpful comments or invaluable hints. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Science and Research.

References

/1/ Broomhead D.S., Lowe D.: Multivariable Functional Interpolation and Adaptive Networks, Complex Systems, 2,321−355, 1988.

/2/ Chen S., Cowan C.F.N., Grant P.M.: Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks, IEEE Transactions on Neural Networks, 2(2)302−309, 1991.

/3/ Chui C.K., Li X., Mhaskar H.N.: Neural networks for localized approximation; Texas A&M University, unpublished manuscript.

/4/ Cybenko G.: Approximation by Superpositions of a Sigmoidal Function, Math Control Signals Syst, 2:303−314, 1989.

/5/ Denoeux T., Lengelle R.: Initializing Backpropagation Networks with Prototypes, Neural Networks, (to appear), 1993.

/6/ Dorffner G., Wiklicky H.: Reanalysing similarity measures in neural networks and their practical consequences; Austrian Research Institute for Artificial Intelligence, Report No. OeFAI−92−15, 1992.

/7/ Dorffner G.: Euclid Net − A multilayer feedforward network using the Euclidean distance as propagation rule; in: Aleksander I., Taylor J. (eds.): Artificial Neural Networks 2, Vol.2, pp. 1633−1636, North−Holland, 1992.

/8/ Dorffner G., Porenta G.: On Using Feedforward Neural Networks for Clinical Diagnostic Tasks; Austrian Research Institute for Artificial Intelligence, Report No. OeFAI−93−24, 1993. Also submitted for publication.

/9/ Dorffner G.: Novel extensions of radial basis function networks and multilayer perceptrons; unpublished manuscript.

/10/ Geva S., Sitte J.: A Constructive Method for Multivariate Function Approximation by Multilayer Perceptrons, IEEE Transactions on Neural Networks, 3(4)621−624, 1992.

/11/ Girosi F., Poggio T.: Networks and the Best Approximation Property, Biological Cybernetics, 63,169−176, 1990.

/12/ Hartman E.J., Keeler J.D., Kowalski J.M.: Layered Neural Networks with Gaussian Hidden Units as Universal Approximations, Neural Computation 2, 210−215, 1990.

/13/ Hornik K., Stinchcombe M., White H.: Multi−layer Feedforward Networks are Universal Approximators, Neural Networks, Vol. 2, pp.359−366, 1989.

/14/ Hornik K.: Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4,251−257, 1991.

/15/ Judd J.S.: Neural Network Design and the Complexity of Learning; MIT Press, 1990.

/16/ Kurkova V.: Kolmogorov's Theorem and multilayer neural networks; Neural Networks 5, 501−506, 1992.

/17/ Lee C., Doolen G., Chen H.H., Sun G.Z., Maxwell T., Lee H.Y., Giles C.L.: Machine learning using a higher order correlation network; Physica 22D, 276−306, 1986.

/18/ Leonard J.A., Kramer M.A., Ungar L.H.: Using Radial Basis Functions to Approximate a Function and Its Error Bounds, IEEE Transactions on Neural Networks, 3(4)624−627, 1992.

/19/ Maruyama M., Girosi F., Poggio T.: A Connection between GRBF and MLP, Massachusetts Institute of Technology, Cambridge, MA, AI Memo No. 1291, 1992.

/20/ Mel B.W., Omohundro S.M.: How Receptive Field Parameters Affect Neural Learning, in Lippmann R.P., et al.(eds.), Advances in Neural Information Processing 3, Morgan Kaufmann, San Mateo, CA, pp.757− 766, 1991.

/21/ Moody J., Darken C.: Fast learning in networks of locally-tuned processing units; Neural Computation 1, 281−294, 1989.

/22/ Omohundro S.: Geometric Learning Algorithms, International Computer Science Institute (ICSI), report, 1989.

/23/ Robinson A.J., Niranjan M., Fallside F.: Generalising the Nodes of the Error Propagation Networks, Cambridge University Engineering Department, Cambridge, UK, CUED/F−INFENG/TR 25, 1988.

/24/ Roy A., Kim L.S., Mukhopadhay S.: A polynomial time algorithm for the construction and training of a class of multilayer perceptrons; Neural Networks 6, 535−545, 1993.

/25/ Rumelhart D.E., Hinton G.E., Williams R.J.: Learning internal representations by error back propagation; in: Rumelhart D.E., McClelland J.L. (eds.): Parallel Distributed Processing, Vol1. I, MIT Press, 1986.

/26/ Rumelhart D.E., McClelland J.L. (eds.): Parallel Distributed Processing, Vol1. I, MIT Press, 1986.

/27/ Smyth S.G.: Designining Multilayer Perceptrons from Nearest-Neighbor Systems, IEEE Transactions on Neural Networks, 3(2)329−333, 1992.

/28/ Taylor J.G., Coombes S.: Learning Higher Order Correlations; Neural Networks 6, 423−427, 1993.

/29/ Werbos P.: Beyond regression: New tools for prediction and analysis in the behavioral sciences; Harvard University, Ph.D. Dissertation, 1974.

/30/ Weymaere N., Martens J.−P.: A Fast and Robust Learning Algorithm for Feedforward Neural Networks, Neural Networks, 4,361−369, 1991.

/31/ Weymaere N.: Progress Report − Task 402, Esprit−II project 5433 NEUFODI, Commission of the European Community, 1992.