An Abstract of the Thesis of

Dietrich Wettschereck for the degree of Master of Science in Computer Science presented on June 27, 1990.

Title: An Implementation and Initial Test of Generalized Radial Basis Functions

Abstract approved: Redacted for Privacy

Thomas G. Dietterich

Generalized Radial Basis Functions were used to construct networks that learn input-output mappings from given data. They are developed out of a theoretical framework for approximation based on regularization techniques and represent a class of three-layer networks similar to backpropagation networks with one hidden layer.

A network using Gaussian base functions was implemented and applied to several domains. It was found to perform very well on the two-spirals problem and on the nettalk task.

This paper explains what Generalized Radial Basis Functions are, describes the algorithm, its implementation, and the tests that have been conducted. It draws the conclusion that network implementations using Generalized Radial Basis Functions are a successful approach for learning from examples.

An Implementation and Initial Test of Generalized Radial Basis Functions

by

Dietrich Wettschereck

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed June 27, 1990

Commencement June 1991

Approved:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented June 27, 1990

Typed by Dietrich Wettschereck

Table of Contents

List of Figures

List of Tables

An Implementation and Initial Test of Generalized Radial Basis Functions

1. INTRODUCTION

In the field of Artificial Intelligence, learning from examples [Dietterich,1990] is an extensively applied method for learning input-output mappings for unknown or intractable functions. Although performing well, many learning algorithms show a lack of theoretical justifications. In the area of neural networks, for example, algorithms such as backpropagation [Rumelhart,86] give surprisingly good results but it has been difficult to prove that these algorithms have theoretical justification.
However, Poggio and Girosi [Poggio,1989] suggested a theoretical framework for a class of networks they called Generalized Radial Basis Functions(GRBF)[1] and claimed the networks based on this approach were able to learn an input-output mapping from examples. The main goal of this project was to implement GRBF and test whether those networks can compete with already existing algorithms such as backpropagation and ID3 [Quinlan,1986].
Poggio and Girosi's main idea was to show the applicability of the results of approximation and regularization theory to the learning task. Their underlying assumption was that learning from examples can be described as hypersurface reconstruction. Given input vectors and their output values, the task is to come up with reasonable output values at points where no data were given, but this can be viewed as approximating a multidimensional function.
Therefore the well-supported results of approximation and regularization theory were used in the design of GRBF. GRBFs are a

---

[1] During this paper I will use GRBF as a synonym for the theoretical framework, the algorithm, and the implementation.

generalized form of radial basis functions which are mainly used for interpolation tasks. They represent a class of three layer networks similar to backpropagation networks with one hidden layer.
In this paper I will review some results from approximation theory; give an intuitive and theoretical explanation of what GRBFs really are; review some related work such as closest neighbor classification; describe the actual implementation of GRBF; show GRBF's performance on different domains; and discuss the practical impact of Poggio and Girosi's suggestions.

2. APPROXIMATION THEORY

The central problem addressed by approximation theory is the approximation of a unknown real valued continuous function f(x) by an known approximation function f*(A,x) which depends on a finite number of parameters A. The quality of an approximation function is measured by a distance function ∂. The function ∂ is usually a norm denoted by || ||.

The approximation problem can formally be stated as:

Approximation Problem:
Let f(x) be a given real valued continuous function defined on a set X, and let f*(A,x) be a real-valued approximation function depending continuously on x $\in$ X and on n parameters A. Given a distance function ∂, determine the parameters $A^* \in$ P such that:
$\partial[f^*(A^*,x),f(x)] <= \partial[f^*(A,x),f(x)]$ for all A $\in$ P

A solution to this problem is called "best approximation". Approximation theory provides several algorithms for finding that solution, if it exists. To approximate a function f, one has to choose an approximation function f*, choose the distance function ∂, and find the solution, e.g. the optimal values for A, if possible.
The approximation function f* is usually determined by the sum of several base functions.

Usually as many base functions as there are known data points are used. Each base function is centered at one data point, i.e. the data point's input value is subtracted from the base function's argument so that the data point is the base function's origin.

Best results are achieved by base functions h with following attributes:

$$h(0) = 1$$

$$h(-\infty) = 0 \text{ and } h(\infty) = 0$$

The base function is then scaled by the output value of its origin.

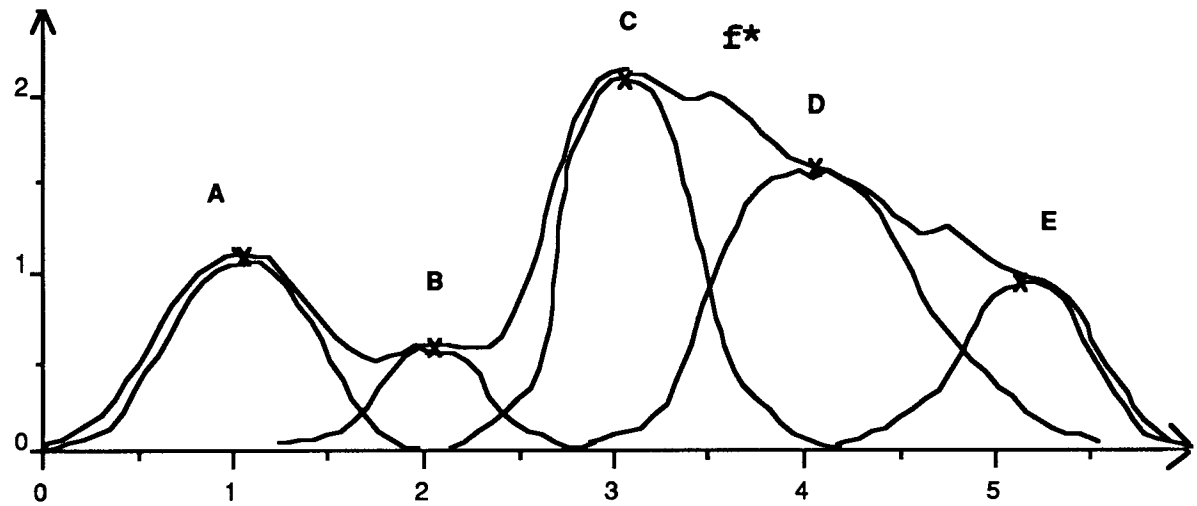

Figure 1: Approximation of a function f by an approximation function f*.

Figure 1 shows the approximation of a function of which five data points (A,B,C,D,E) are known. f* is determined by the sum of five Gaussian base functions, each centered at one of the known data points. This graph shows why Gaussian base functions are superior to many other base functions: The base function at point E does not influence f* at point A at all, but any point between A and B is, starting at A, heavily influenced by the base function at point A. With decreasing distance to B (and therefore increasing distance from A) B's influence rises while A has almost no impact on f* at B.

3. THE INTUITIVE EXPLANATION

Learning from examples is a means of generalizing from given data to an overall function describing a whole class of data points. There are several assumptions necessary to guarantee the applicability of any learning algorithm:
The examples must sufficiently cover the class to which they belong. Hence, their number has to be substantial enough and they have to be well enough distributed throughout the domain.
Furthermore, some bias must be provided to the learning algorithm [Mitchell,1980]. This bias provides a set of constraints that the unknown function is assumed to satisfy. For GRBFs, the bias asserts that the unknown function is smooth (i.e., continuously differentiable). The smoothness of the function is crucial. Otherwise an infinite number of training examples would be needed to learn the function.

These are the same assumptions that are made for interpolation or more general approximation algorithms. Using the vocabulary provided by approximation theory, the task can be described as follows:
Given:
N data points in n-dimensional space and their output values.

$$(x_1, x_2, x_3, \ldots, x_n)_i \rightarrow y_i \quad i = 1..N$$

Find:

A smooth function $f^*: \Re^n \rightarrow \Re$ such that

$\sum_{i=1}^{N} (f^*(x_i) - y_i)^2$ is minimal.

From approximation theory, it turns out that one form f^* can take, is a weighted sum of radial basis functions:

$$f^*(x) = \sum_{i=1}^{N} (c_i * h(||x-x_i||^2)) \qquad (1)$$

where h is, for example, a Gaussian base function centered at x_i.

The distance function ∂, defined by approximation theory, is the minimization of the sum of all differences between f^* and the already known output. This sum has to be minimized by finding the optimal values for the weights c_i. If the approximation is successful, f^* can later be used to determine output values for input patterns with unknown output values.
Since the number of training examples is usually very high, it becomes computationally expensive to evaluate f(x). Hence, it is desirable to limit the number of base functions. GRBFs solve this task by dividing the input space into several hyperspheres (see Figure 2). The origin of each hypersphere is called a "center". The hyperspheres are unbounded and therefore overlap each other.
The introduction of GRBFs is an approach to approximate the function f by a function f^* with a fewer number of centers (t_α) that do not necessarily coincide with any of the given examples.
The generalized form of (1) is then given as:

$$f^*(x) = \sum_{\alpha=1}^{n} (c_\alpha \; h(||x-t_\alpha||^2) \qquad (2)$$

with n: number of centers

With growing distance from their origin, the hyperspheres lose importance (strength). One can compare this with radio stations. Although with a powerful receiver it is possible to listen to every station at every point, it is usually the case that only the nearest stations are received by a normal receiver; if one is very close to a certain station, it is almost impossible to listen to another station at all.
Back to the input patterns and their output values: Any input pattern's output value is determined by every center, because it lies within every hypersphere, and those centers it is closest to have the most influence on its value.

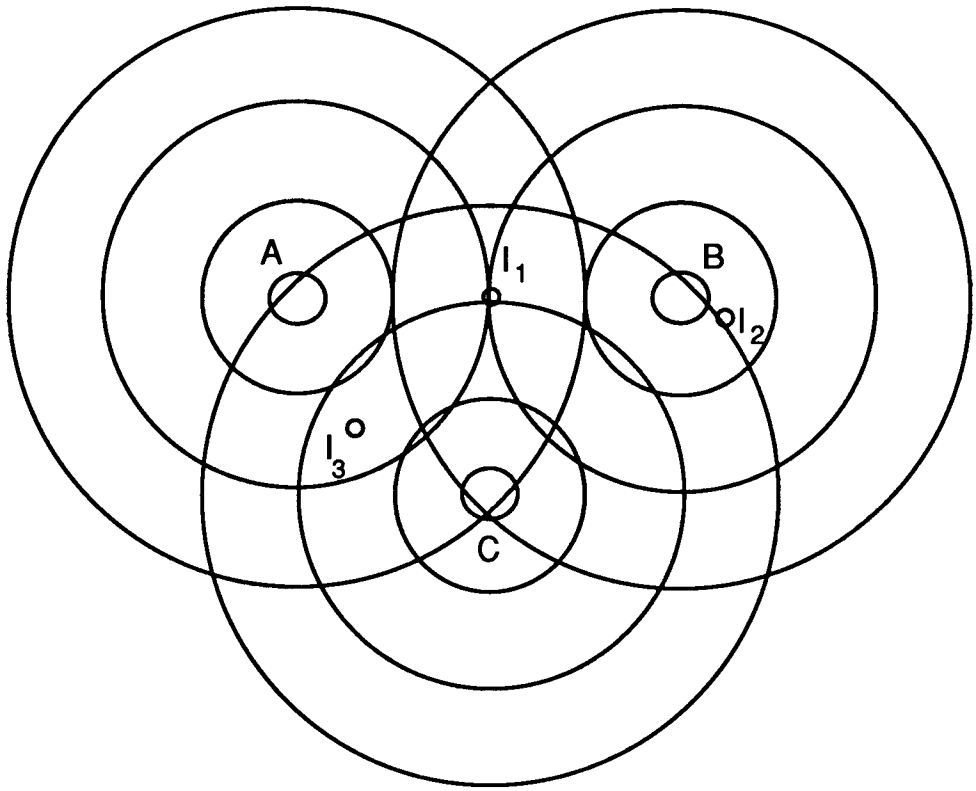

Figure 2: Three centers A,B,C and three data points I_1, I_2, and I_3.

I_1 is equally influenced by A,B,and C (neglecting the center's weights for the moment)
I_2 is almost only described by B
I_3 is determined by A and C, but only slightly by B

In addition to the weights of the basis functions that determined f* in the radial basis function approach, it is now necessary to find the optimal number of centers and their locations.
GRBFs are implemented through a network. The network architecture is described by Figure 3. During each iteration of the training phase, the difference between the network's output for each input pattern and the desired output is computed. These error terms are then used to update the network. This is done by moving the centers to different locations and adjusting their weights. The weights of the centers are used to scale the output. The network's output is determined by the sum of the outputs of every center scaled by each center's weight to that particular output unit. Moving the centers is

the fundamental point of the whole algorithm. Initially, the centers are chosen as a subset of the given input data, but during training, they are moved to different locations so that they represent the data in the best possible way (see Figure 4).

GRBFs represent a class of three layer networks with one hidden layer. The hidden units are called centers. Each center is fully connected to each input unit. A base function h is centered at each center's coordinates. Each center t is connected with a certain weight c to each output unit.

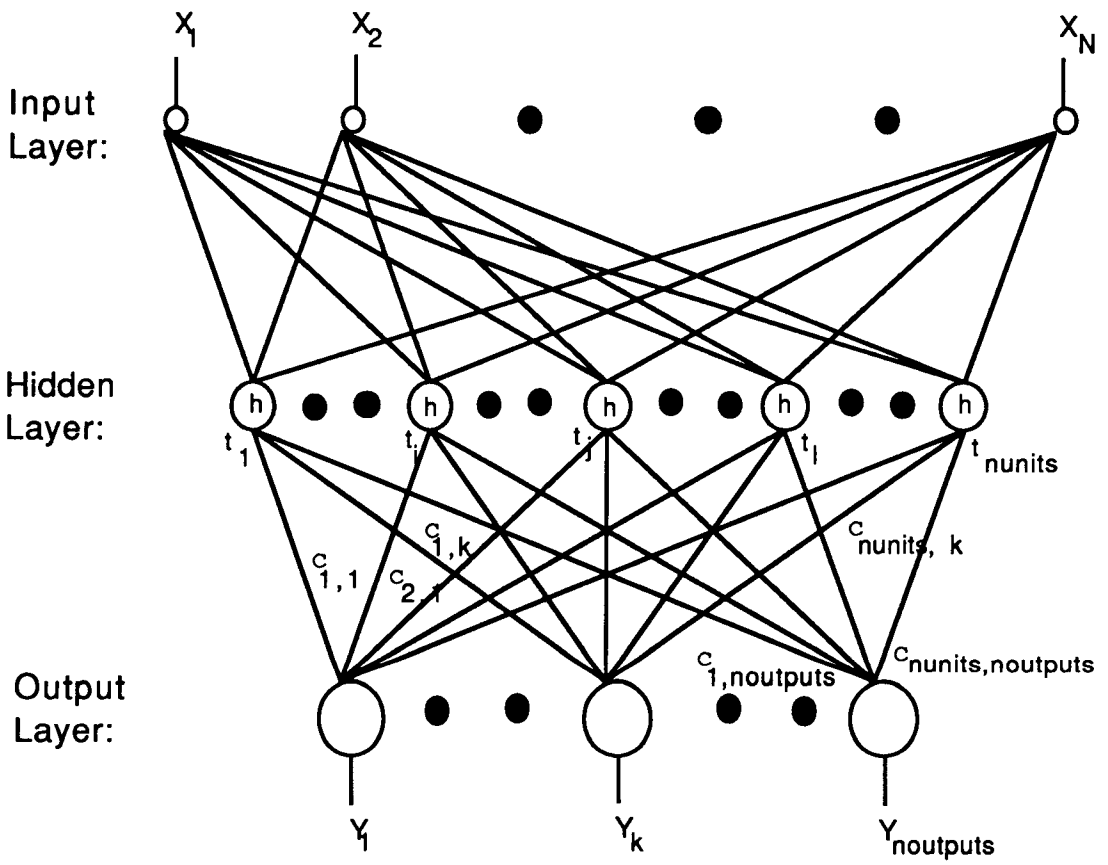

Figure 3: The GRBF network architecture.

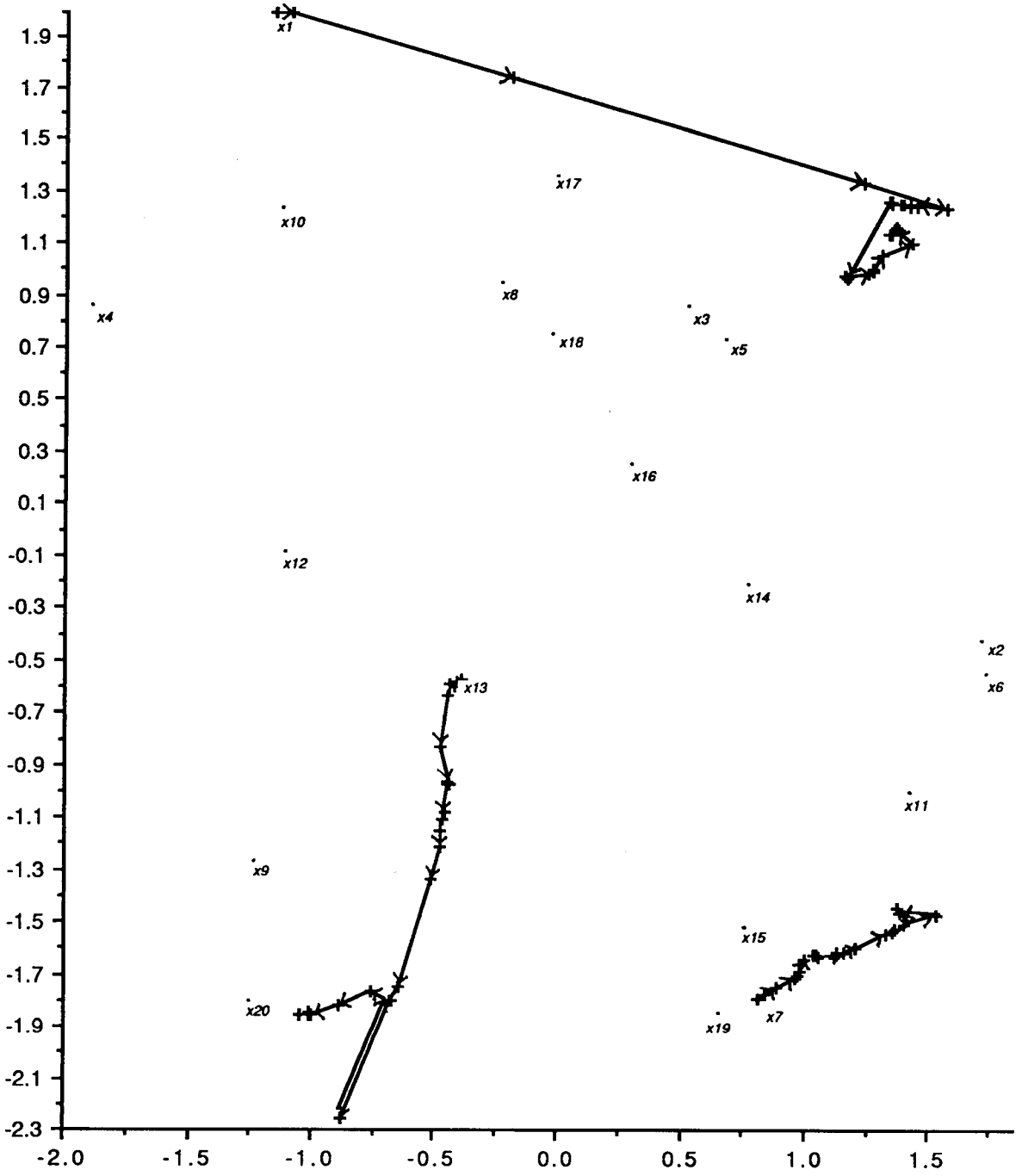

Figure 4: Training phase

Figure 4 shows how GRBF moved the centers to different locations during a training phase. The points x1 to x20 represent the training data. Three centers were initialized to the data points x1, x7, and x13. During training these centers were moved to the points indicated by the arrows.

4. THE ALGORITHM

In this section I will develop the theoretical framework for GRBFs: Given $S= \{(x_i;y_i) \in \Re^n \times \Re;\ i = 1,..N\}$ a set of data points that is to be approximated by means of a function f. The regularization approach consists of looking for the function f that minimizes the functional

$$H[f] = \sum_{i=1}^{N} (y_i - f(^*x_i))^2 + \lambda \, ||Pf||^2 \qquad (3)$$

Where P is a constraint operator (for example a Laplacian operator) and the regularization parameter λ is directly related to both the degree of enforced generalization and to an estimate of the noise and is set to zero because we assume there is no noise in the data. Poggio and Girosi proved in their paper [Poggio,1989] that the solution to that equation has the form:

$$f^*(x) = \sum_{i=1}^{N} (c_i \ ^* \ G(x;t_i)) \qquad (4)$$

where G(x) is the Green's function of the self-adjoint differential operator $\hat{P}P$, $\hat{P}$ being the adjoint operator of P, and the coefficients c_α satisfy a linear system of equations that depend on the N examples. In this paper I assume that G is radial ($G(x;t) = h(||x-t||^2)$) and therefore equation (4) can be written as:

$$f^*(x) = \sum_{i=1}^{N} (c_i \ ^* \ h(||x-t_i||^2)) \qquad (1)$$

This is the sum of radial functions each centered on a distinct data point t_α. There are as many radial functions as data points (training examples).

The introduction of GRBFs is an approach to approximate the unknown function f by a function f^* with a fewer number of centers (t_α) that do not necessarily coincide with any of the given examples. The generalized form of (1) is then given as:

$$f^*(x) = \sum_{\alpha=1}^{n} (c_\alpha \; h(||x-t_\alpha||^2) \qquad (2)$$

with n: number of centers

Now the problem is to find the coefficients c_α and the centers t_α. To minimize the functional $H[f^*]$ the necessary conditions

$$\frac{\partial H[f^*]}{\partial c_\alpha} = 0 \qquad \text{and} \qquad \frac{\partial H[f^*]}{\partial t_\alpha} = 0 \qquad (5)$$

are imposed. Those equations can be used to find the initial values for the weight vector C:

$C = (G^tG + \lambda g)^{-1}G^t Y$ (see 6.1. The Initialization Procedures)
with G an N×n matrix $G_{i\alpha} = G(x_i;t_\alpha)$, g a n×n matrix $g_{\alpha\beta} = G(t_\alpha,t_\beta)$.

Given that G is radial ($G(x;t) = h(||x-t||^2)$), $\lambda = 0$, and defining the error as
$\Delta_i = y_i - f^*(x_i)$
with $f^*(x_i)$ = the network's output for the ith example, the gradient terms can be written as:

$$\begin{aligned}\frac{\partial H[f^*]}{\partial c_\alpha} &= \frac{\partial}{\partial c_\alpha}\left(\sum_{i=1}^{N} (y_i - c_\alpha * h(||x_i-t_\alpha||^2))^2\right) \\ &= 2 * \sum_{i=1}^{N} (y_i - c_\alpha * h(||x_i-t_\alpha||^2)) * (- h(||x_i-t_\alpha||^2)) \\ &= -2 * \sum_{i=1}^{N} \Delta_i * h(||x_i-t_\alpha||^2) \qquad (6)\end{aligned}$$

$$\frac{\partial H[f^*]}{\partial t_\alpha} = \frac{\partial}{\partial t_\alpha}\left(\sum_{i=1}^{N}(y_i - c_\alpha * h(||x_i-t_\alpha||^2))^2\right)$$

$$= 2 * \sum_{i=1}^{N}(y_i - c_\alpha * h(||x_i-t_\alpha||^2)(-c_\alpha) * h'(||x_i-t_\alpha||^2) * (-2(x_i-t_\alpha)))$$

$$= 4 * c_\alpha * \sum_{i=1}^{N}(\Delta_i * h'(||x_i-t_\alpha||^2) * (x_i-t_\alpha)) \qquad (7)$$

The network is trained by an iterative process. During each iteration the gradient terms were used to find better values for the t_α and c_α. Although gradient descent and momentum term methods work quite well, the conjugate gradient method suggested by Powell [Powell,1977] was used to find the optimal set of values for the weights c_α and the centers t_α. Note that in addition to the original interpolation task of finding the optimal values for the weights $\{c_\alpha\}$, the centers $\{t_\alpha\}$ were moved around to best approximate the hypersurface.

Any input pattern's output value is computed as:

$$f^*(x) = \sum_{\alpha=1}^{n} c_\alpha * h(||x-t_\alpha||^2) \qquad (8)$$

where $h(||x-t_\alpha||)$ is an appropriate radial basis function. In this thesis I mainly used the well known Gaussian base functions ($h(x) = \exp(-(x/\sigma)^2)$) which offers the advantage of being already smooth enough so that λ can be set to zero in (3).
Equation (8) can be generalized to a d-dimensional output vector as follows:

$$f^*(x_1,x_2,\ldots,x_n) = (\Sigma\, c_{\alpha 1} * h(||x-t_\alpha||^2), \Sigma\, c_{\alpha 2} * h(||x-t_\alpha||^2), \ldots, \Sigma\, c_{\alpha d} * h(||x-t_\alpha||^2))$$

This shows that d different input-output mappings were actually learned ($c_{\alpha 1}$-$c_{\alpha d}$) with the restriction that all d functions use the same centers, but different weights (C).

5. RELATED WORK

5.1. Pattern Classification, Parzen Windows, Nearest Neighbor Estimation

Duda and Hart [Duda,1973] describe several approaches for finding a density function p(x). This density function is used to determine the probability of a given pattern belonging to a specific class. In particular they mentioned two nonparametric methods for pattern recognition that work without knowing the underlying densities. Those methods are Parzen Windows [Parzen, 1962] and the k_n-nearest-neighbor-estimation method.

5.1.1. Parzen Windows

A Parzen window is a d-dimensional solid (in the following discussion I assume that it is a hypercube) centered at a point x.
Let V_n be the volume and h_n the length of an edge of that hypercube, k_n the number of samples falling in it, and $p_n(x)$ the n-th estimate for p(x), then

$$p_n(x) = \frac{k_n/n}{V_n} \qquad (9)$$

A window function Φ defines a solid of dimension d centered at the origin as follows:

$$\Phi(u) = \begin{cases} 1 & |u_j| <= 1/2 \qquad j = 1..d \\ 0 & otherwise \end{cases}$$

Therefore $\Phi\left(\frac{x-x_i}{h_n}\right)$ equals 1 if x_i lies within the hypercube centered at x. The number of examples falling in that hypercube is then given by:

$$k_n = \sum \Phi\left(\frac{x-x_i}{h_n}\right) \qquad (10)$$

When substituting (10) into (9) we obtain:

$$p_n(x) = \frac{1}{n} * \sum \frac{1}{V_n} * \Phi\left(\frac{x-x_i}{h_n}\right)$$

Given an infinite number of examples ($n \to \infty$), p_n converges to p(x) while V_n (one choice for h_n might be $h_n = 1/n$) converges to 0. In other words, if it is possible to decrease the size of the window without eventually obtaining an empty window, then it is possible to approximate the density function p(x) arbitrarily closely.
In practical applications with a finite number of (training) patterns, the choice of h_n (and therefore V_n) determines $p_n(x)$ such that if V_n is too large, $p_n(x)$ will be averaged over too many patterns, and if V_n is too small, it will result in a p_n that depends too much on the particular examples.

5.1.2. The k_n-nearest-neighbor-estimation Method

While the size of Parzen windows is independent from the data, the k_n-nearest-estimation method makes every window, any of which are still centered at x, so large that it captures exactly k_n example patterns. Therefore the windows will be small in areas with many patterns and huge if the density is low.
Taking $p_n(x) = \frac{k_n/n}{V_n}$ and letting $\lim_{n\to\infty} k_n \to \infty$ and $\lim_{n\to\infty} \frac{k_n}{n} \to 0$ assures that, although an infinite number of patterns lies within V_n, V_n will be very small because k_n is negligible in comparison to the total number of examples.
This method suffers, as well as Parzen Windows, from the fact that it is only asymptotically correct.
Both methods are similar to GRBF in the sense that they assume the existence of an underlying density function. Also, the windows can be viewed as the hyperspheres which are used by GRBF. The main difference is that Parzen windows are bounded by size and the k_n-nearest-estimation method's windows are bounded by the number of examples falling in each window, while GRBF's hyperspheres are theoretically unbounded.

5.2 Exemplar Learning Algorithms

Several different approaches for learning concepts from examples have been proposed. Some of them simply store all examples or a subset of the examples. A new instance is then classified by comparing it with all stored examples and assigning it the value of the example to which it is most similar (i.e. k_n-nearest-neighbor with $k_n = 1$).

Smith and Medin suggested the proximity model [Smith,1981] which stores all training instances. Kibler and Aha [Kibler,1987] developed the growth and shrink algorithms. The growth algorithm does not add instances that are already classified correctly. The shrink algorithm removes examples that would be classified correctly by the other examples.

These algorithms are similar to GRBF in that the number of examplars is less than the number of training examples. However, in GRBF the centers can correspond to "idealized" prototypes rather than to particular training examples. Also, as with Parzen windows and k_n-nearest-neighbor, exemplar methods do not allow distinct exemplars to influence the estimated value of the function.

6. THE IMPLEMENTATION

To implement GRBF, software originally written by McClelland and Rumelhart [Rumelhart,1988] was enhanced by one more program. The design, and in particular the user-interface, were left unchanged and therefore I had only to be concerned about the code that actually deals with GRBF. The idea is that it should be possible to run the program in the same environment as the PDP software.
There are three new files that I have written. Init.c deals with the network initialization. Net.c takes care of saving an already trained network's parameters to a file and reading them back in, when issued. Lastly grbf.c is the main file that does the work for training the network.

6.1. The Initialization Procedures

Two things have to be done to get started:
First choose the centers, and second set their initial weights to all output units.
In choosing the centers, if the number of centers is assumed to be n, and there are N input patterns, then the centers are simply selected as a subset of all input patterns by making every (N mod n)th input pattern a center.
There might be better ways of choosing the centers such as the k-means method by MacQueen [MacQueen,1967] or some other set covering algorithms. I did not implement any of those more sophisticated algorithms, because I made the assumption that the input patterns are presented in such a way that it is unlikely to choose specific subsets by this algorithm without covering at least a few examples of any subset.
For example, the data might be randomly distributed or there might be a total ordering. In the case of the total ordering, the centers would be ordered totally also. Here I assumed that a suboptimal

starting position would not prevent the algorithm from learning, but simply would result in a few more iterations. Therefore I traded programming time for running time. A fact supporting this point of view is that the network learns very fast at the beginning and slows down when coming close to the optimum. However, section 8.1 shows that the performance of GRBF depends on good choice of the initial centers.

The weights are initialized according to:

$$f^*(x) = \sum_{\alpha=1}^{n} c_\alpha * G(x;t_\alpha)$$

$$\begin{aligned} H[f] &= \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda * ||Pf||^2 \quad (\text{set } \lambda = 0) \\ &= \sum_{i=1}^{N} (y_i^2 - 2 * f(x_i) * y_i + f(x_i)^2) \\ &= Y * Y - 2 * \sum_{\alpha=1}^{n} (c_\alpha * G(x_i;t_\alpha)) * Y + (\sum_{\alpha=1}^{n} (c_\alpha * G(x_i;t_\alpha)))^2 \\ &= Y * Y - 2 * C * G^t * Y + (C * G)^2 \\ &= Y * Y - 2 * C * G^t * Y + C * G^t * G * C \end{aligned} \qquad (11)$$

with $G = G(x_i;t_\alpha)$ $Y = (y_1,...,y_N)$, $C = (c_1,..,c_n)$, G^t: transpose of G

The functional H has to be minimized, giving the condition:

$$\begin{aligned} &\frac{\partial H[f^*]}{\partial c_\alpha} = 0 \\ &\Rightarrow -2\, G^tY + 2G^tGC = 0 \\ &\Leftrightarrow \quad G^tGC = G^tY \\ &\Leftrightarrow (G^tG)^{-1} * G^tGC = (G^tG)^{-1} * G^tY \\ &\Leftrightarrow \quad C = (G^tG)^{-1} * G^tY \end{aligned} \qquad (12)$$

That means, after choosing the centers the matrix G has to be computed and the equation (12) can be used to initialize the weights. One problem occurred during inverting the matrix G^tG: Especially for the nettalk data this was a very huge matrix (up to 500 * 500) with many small entries. Using an algorithm with Pivot search, integer overflows occurred for n×n matrixes with n > 60. To avoid this, I multiplied the whole matrix by a factor of 10, inverted the matrix and multiplied the inverted matrix again with the same factor, thereby bringing most elements close to 1.

6.2. Checking for Collapsing Centers

There are no constraints that limit the movements of the centers. Hence two or more centers may move towards each other until they coincide. I wrote a function that checks the distance between any two centers after each iteration. Surprisingly, the case of collapsing centers never occurred, and therefore the current implementation does not check the distance between the centers anymore.

6.3. Optimizations

6.3.1. General optimizations

The formulas for computing the gradient for the centers (T) and their weights (C) were on page 11 given as follows:

$$\frac{\partial H}{\partial c_\alpha} = -2 * \sum_{i=1}^{N} (\Delta_i * h(||x_i - t_\alpha||^2)) \qquad (6)$$

$$\frac{\partial H}{\partial t_\alpha} = 4 * c_\alpha * \sum_{i=1}^{N} (\Delta_i * h'(||x_i - t_\alpha||^2) * (x_i - t_\alpha)) \qquad (7)$$

Since this implementation deals with multidimensional outputs these formulas looked actually like this:

$$\frac{\partial H}{\partial c_{\alpha j}} = -2 * \sum_{i=1}^{N} (\Delta_{ij} * h(||x_i - t_\alpha||^2)) \qquad (13)$$

$$\frac{\partial H}{\partial t_\alpha} = 4 * \sum_{j=1}^{noutputs} (c_{aj} * \sum_{i=1}^{N} (\Delta_{ij} * h'(||x_i - t_\alpha||^2) * (x_i - t_\alpha))) \qquad (14)$$

Now (14) can be changed to:

$$\frac{\partial H}{\partial t_\alpha} = 4 * \sum_{i=1}^{N} (\sum_{j=1}^{noutputs} (c_{\alpha j} * \Delta_{ij}) * h'(||x_i - t_\alpha||^2) * (x_i - t_\alpha)) \qquad (15)$$

Thereby replacing one level of loop nesting by two parallel loops. The complexity was reduced from
(nunits * npatterns * noutputs * ninputs) to
(nunits * npatterns * (noutputs + ninputs)).
Note that (x - t) is a vector of dimensionality 'ninputs'.

6.3.2. Nettalk Optimizations

For the nettalk data, even further optimizations were possible, since the nettalk data had the characteristic that every input vector consist of exactly seven 1's and 196 0's.
Therefore equation (15) can be written as:

$$\frac{\partial H}{\partial t_\alpha} = 4 * \sum_{i=1}^{N} \left(\sum_{j=1}^{noutputs} (c_{\alpha j} * \Delta_{ij}) * h'(||x_i - t_\alpha||^2) * (-t_\alpha) \right)$$
$$+ 4 * \sum_{i=1}^{N} \left(\sum_{j=1}^{noutputs} (c_{\alpha j} * \Delta_{ij}) * h'(||x_i - t\alpha||^2) * (x_{ik}) \right)$$
$$\text{for all } k: x_{ik} = 1, \quad k:1..7 \qquad (16)$$

resulting in a complexity of (nunits * npatterns * (noutputs + 7))
For the same reason, the complexity of the function that computes the distance of every input pattern to every center could be reduced from (nunits * ninputs * npatterns)
to (nunits * (ninputs + 7 * npatterns)).
This was simply done be using the following:
The square of the euclidian distance between two n dimensional points can be computed as follows:

$$|| a - b ||^2 = \sum_{i=1}^{n} (a_i - b_i)^2$$

If it is known that b is '0' most of the time and '1' otherwise, this equation can be changed to:

$$\sum_{i=1}^{n} (a_i^2 - 2a_i b_i + b_i^2)$$

$$\sum_{i=1}^{n} a_i^2 + \sum_{i_k=1}^{7} (1 - 2\, a_{ik}) \qquad \text{since } b_{ik} = 1,\ k = 1..7$$

$$i^k \in 1..n\ ,\ 0 \text{ otherwise}$$

7. RUNNING THE PROGRAM

The design of the program is kept close to the underlying implementation of the backpropagation algorithm by McClelland and Rumelhart [Rumelhart,1988].
I will here just give a brief overview of the rudimentary commands and files that are needed to run GRBF. For further details refer the handbook written by McClelland and Rumelhart [Rumelhart,1986].
Four files should be provided when starting the program:
- A network file containing the network description.
- A pattern file with the training data.
- A command file.

The command file may contain any legal commands. It is very convenient to incorporate here the commands that initialize the network such as loading the network definition file and setting the number of iterations for example. This file is also necessary to run the program in the background.
- A template file (optional).

The template file might be used to display some information during a program run.
The program is started by the command:

gr [<template-file>| -] <command-file>

A pattern file has to be loaded using the 'get patterns <pattern-file>' command.
Training is started by the command 'strain' and after training, the centers and their weights can be saved to a file using the 'save' command.
Example files are included in the appendix.

8. THE INITIAL TESTS

Several different experiments were carried out to show the correctness of both the implementation and the underlying theory. Furthermore the representational strength of GRBF was compared to already existing algorithms performing on the same task.

8.1. Correctness

The implementation is assumed to be correct if it is possible to train the network on some training set so that, after a reasonable amount of iterations and with an appropriate number of centers, the network is able to compute output values for the given training data close to the desired output values. If one could construct the centers and their weights before training and the algorithm would come up with similar parameters, that would be a very strong argument for the correctness of the algorithm. Therefore I created test and training sets by first choosing the centers, their weights, and the input data randomly. Then the output values for all input patterns were determined by using equation (8).
Then I trained the network several times, each time with a different number of centers. The most interesting case was, of course, when the network was initialized with the same number of centers as were used to determine the training and test data.

Figure 5 shows the final locations of the centers for each experiment. The original centers that were used to determine the training data are O,1, O,2, and O,3. The other points are the learned centers, where labeling a,b means: centers number b out of a centers. In the experiments illustrated by Figure 5, GRBF achieved almost a perfect fit on the training set. In these experiments, GRBF moved the centers in fact very close to the original centers. GRBF also assigned similar weights which are not shown by the graph.

However, the second original center was located in between the other two centers and therefore has less representational power. GRBF located the corresponding center, in the case of three centers, at a different location so that these three centers build a triangle together. This result is very encouraging in the sense that the algorithm not only learns some representation for the training data but a very general representation.

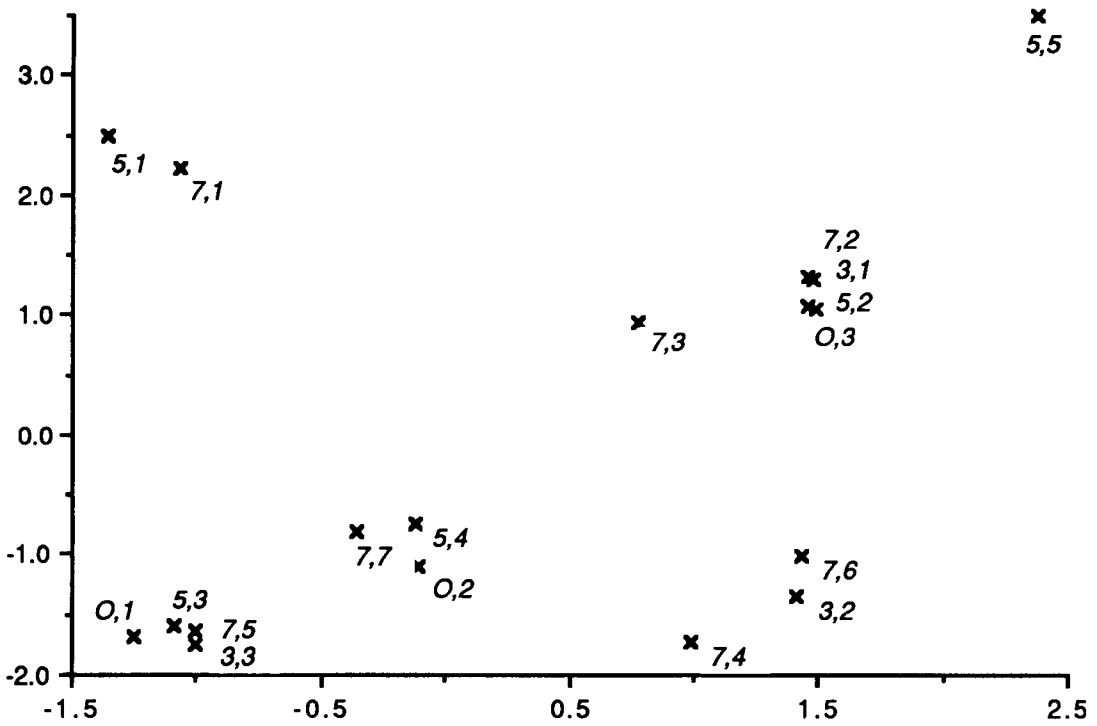

Figure 5: Locations of centers after learning

The experiment was repeated several times. The number of centers and their initial locations was varied. Figure 6 shows the correlation between GRBF's performance on the training set and the test set. The training set consists of 20 patterns and the test set was a 2000 point grid over the area [-2.5:2.5,-2.0:2.0]. Each point shown by the graph has the total summed squared error (tss) as its first coordinate and the tss over the test set as its second coordinate.

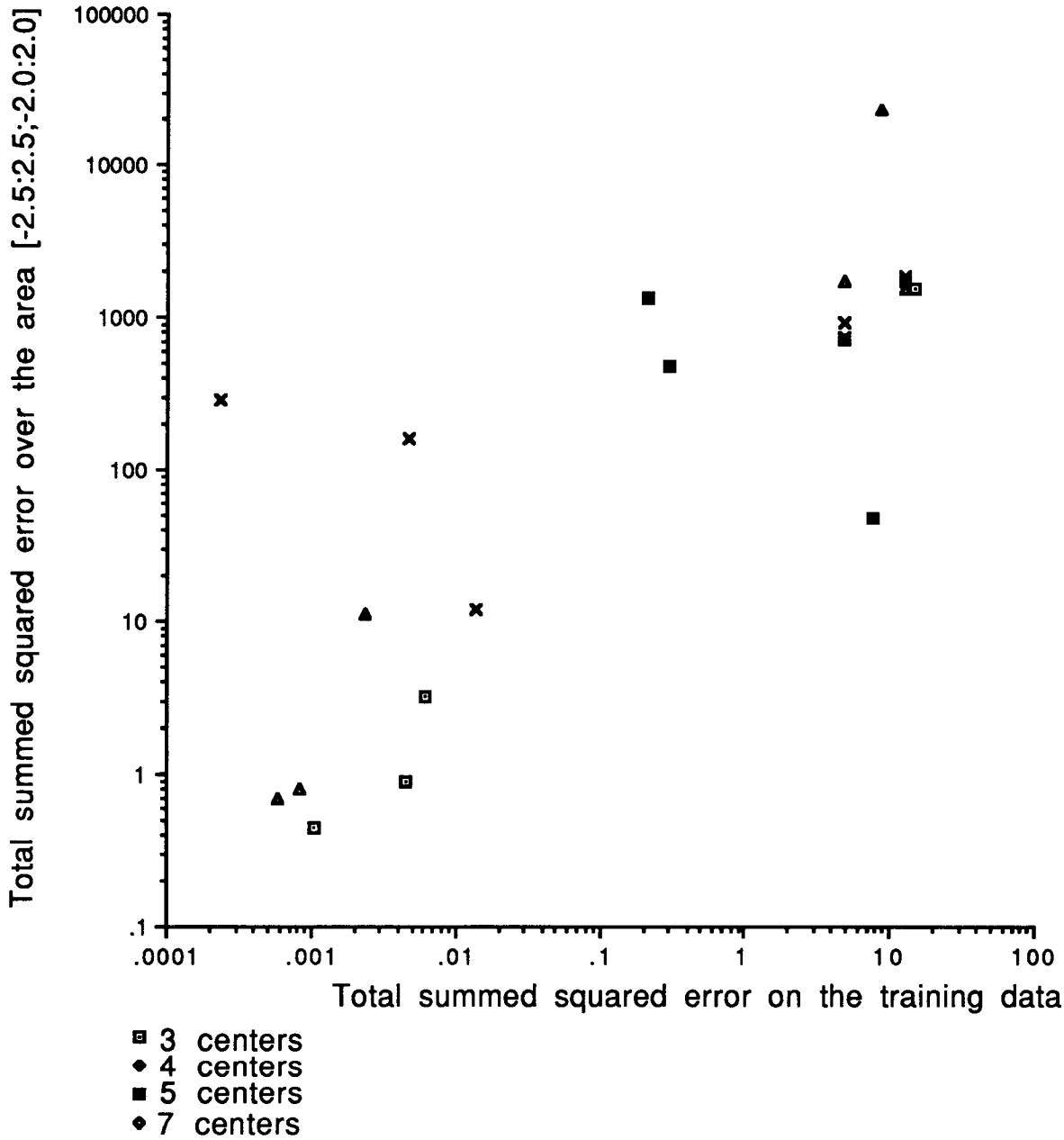

Figure 6: Correlation between performance on the training and the test set.

According to Figure 6, GRBF was not always able to achieve a perfect fit on the training data and there exists a strong correlation between performance on the training and the test set.

Two conclusions can be drawn from these experiments:
First, the initialization of the centers influences GRBF's ability to learn. Second, if GRBF is able to achieve a good fit on the training data then it also performs well on the test set.

8.2. Representational Strength

The representational strength of a network can be determined by its performance on the test data and in some sense by the number of centers required to learn the mapping. To test the representational strength of GRBF, the program was applied to different domains.

8.2.1. The two-spirals Problem

The two-spirals problem is an extremely hard problem for all algorithms based on backpropagation [Rumelhart,1988]. The task is to distinguish between two interlocking spirals. All points on one spiral have positive output values, while all points on the other spiral have negative output values. The training set, which was also used as the test set, consisted of 194 points.
GRBF was able to distinguish between the two spirals without any misclassification when 50 centers were used. It ran for about 20 minutes or 600 epochs. Figure 7 shows the locations the centers were moved to by GRBF. Different weights of the centers are indicated by different symbols. The weights range from -6 to +7 and darker symbols indicate more negative values.
Given 25 centers, it misclassified, after 2000 epochs, 6 data points. In comparison, only modified versions of backpropagation were able to solve this task at all. These versions required 8000 to 200,000 epochs and used several hidden layers, each layer containing about five units [Lang,1988].

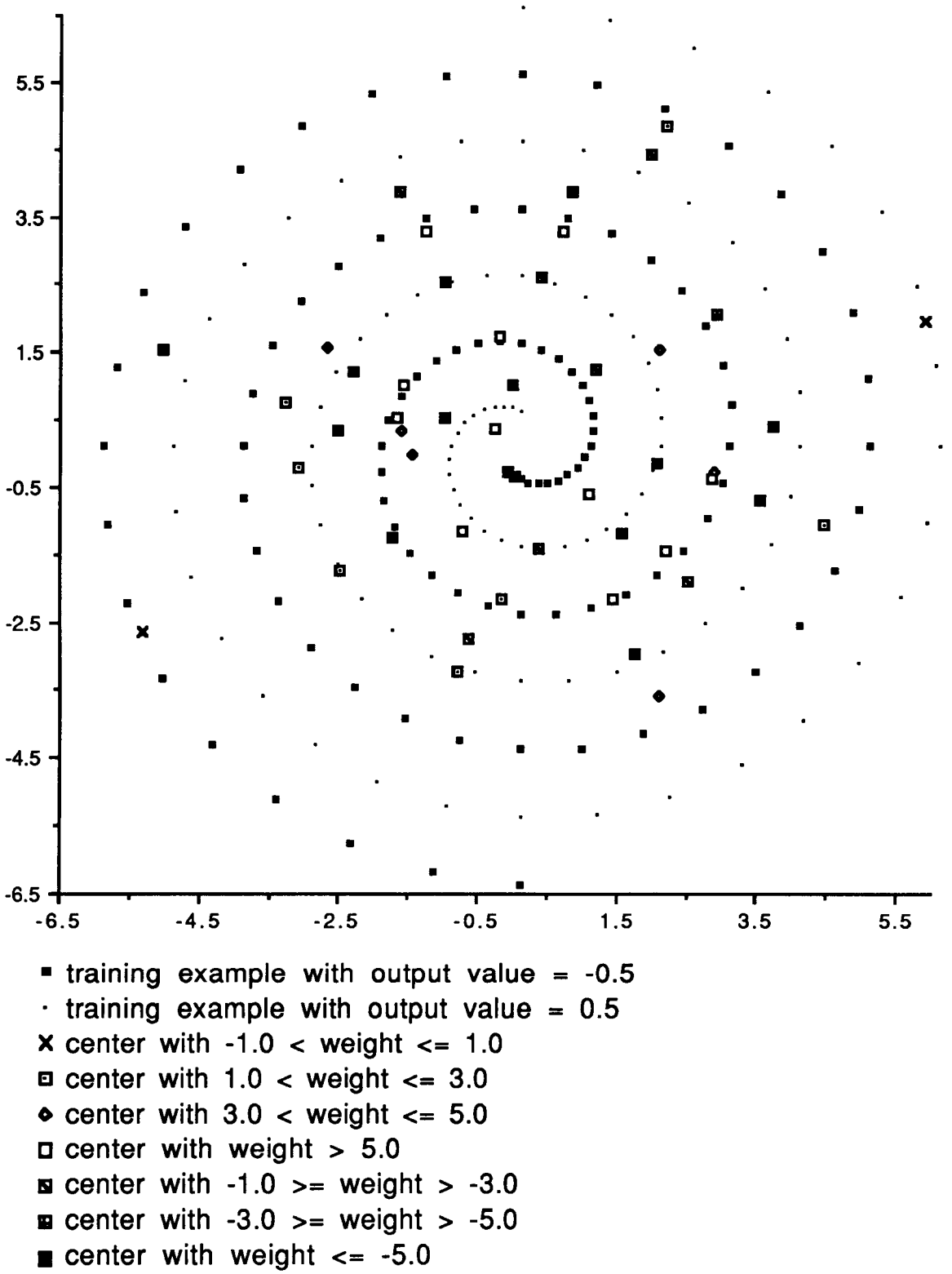

Figure 7: The two-spirals Problem

8.2.2. Random Functions with Unimportant Features

An interesting question was how GRBF would deal with input patterns where some dimensions were of no importance at all, e.g. the output values were only determined by some dimensions. The hypothesis was that GRBF would not be able to distinguish between meaningful information and useless noise. To test this hypothesis, I generated a training and a test set with twenty input dimensions of which only two were important. The remaining dimensions were random. Then I trained the network on that data several times, each time decreasing the number of input dimensions by eliminating irrelevant dimensions.

For any dimension above 7, GRBF was able to fit the training data but performed poorly on the test data. To achieve good performance on the test data, I had to decrease the dimension of the input to three.

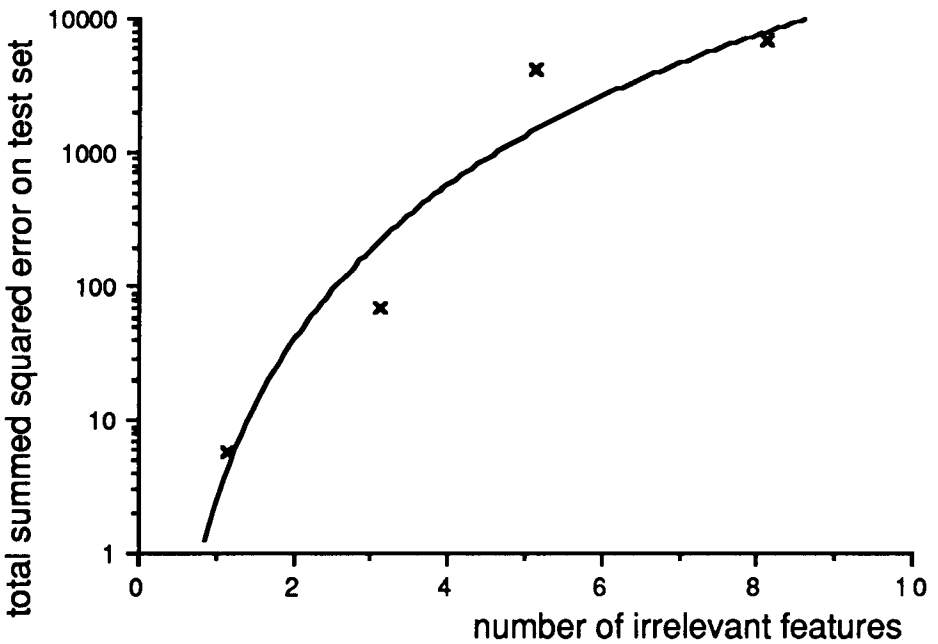

Figure 8: Correlation between irrelevant features and performance.

This experiment showed another interesting result:
The number of centers was crucial to the ability of GRBF to learn. A higher number of centers enabled the network either to learn faster or better, where better means down to a lower remaining error. However, the number of centers did not influence performance on the test set at all. One might be tempted to think that a smaller number of centers makes training harder and therefore the network is 'smarter' after learning. The experiments did not support this, although more tests should be carried out to verify that.

8.2.3. The nettalk Task

The nettalk task is to learn how to pronounce english words from their spelling. It is also known as the Text-to-Speech problem. This task was first introduced by Sejnowski and Rosenberg [Rosenberg,1987]. They provided us with a dictionary of 20,003 words and their corresponding phonemes and stress strings. This dictionary was randomly divided into disjoint subsets. A 1000-word subset was used as a test set and the remaining 19003 words were partitioned into training sets of different sizes. Since the data was coded in a specific way (see [Dietterich,1990] for further details), a 50-word set, for example, contains about 350 patterns.
I trained the network using two different base functions (1/cosh(x) and $\exp(-x^2/\sigma^2)$). For each base function, the number of centers and epochs was varied. I also varied the size of the training set. Specifically, training sets of size 50, 100, and 1000 words were studied. The resulting networks were then evaluated against the 1000-word test set.

Table 1: Performance on the 50-word training set, base function 1/cosh(x)

#centers		WORDS	LETTERS	(PHON/STRESS)		BITS
300	TRAIN:	100.0	100.0	100.0	100.0	100.0
	TEST:	0.0	6.0	6.1	62.9	89.4
120	TRAIN:	100.0	100.0	100.0	100.0	100.0
	TEST:	0.5	25.0	27.5	68.4	91.6

Only two experiments were carried out because performance on the test set was so poor.

Table 2: Performance on the 50-word training set, base function $\exp(-x^2/\sigma^2)$

#centers	#epochs		WORDS	LETTERS	(PHON/STRESS)		BITS
150	250	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	1.5	43.1	52.5	69.8	93.3
150	500	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	1.1	45.3	55.4	69.8	93.5
150	1500	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	0.8	46.7	57.1	70.8	93.6
120	500	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	1.4	49.5	59.2	71.3	94.1
100	200	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	1.6	47.8	55.8	74.1	94.2
100	300	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	1.5	48.6	57.0	73.9	94.2
80	180	TRAIN:	78.0	97.1	99.5	97.6	99.9
		TEST :	2.4	47.2	56.5	73.3	93.9
60	1200	TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	0.9	49.6	61.4	70.9	94.0
40	3000	TRAIN:	96.0	99.5	99.5	100.0	100.0
		TEST :	0.8	49.3	61.5	69.7	93.8
25	7000	TRAIN:	96.0	99.5	99.5	100.0	100.0
		TEST :	0.8	46.7	58.5	69.0	93.4
For comparison, Backprop with 120 hidden units, 30 epochs:							
legal		TRAIN:	82.0	97.4	98.2	99.2	99.9
		TEST :	1.8	48.4	59.4	72.9	93.5
ID3 legal		TRAIN:	100.0	100.0	100.0	100.0	100.0
		TEST :	0.8	41.5	60.5	60.1	93.1

On the 50 word training set it was always possible to achieve a perfect fit, but performance on the test set varied widely. Especially the difference between the two base functions is interesting since they are very similar functions. I assume that the scaling factor σ makes the Gaussian base functions more powerful. The bigger σ is, the further away from its center each hypersphere influences the output. Theoretically the hyperspheres are unbounded, but in practice the exponential function approaches zero very fast. Figure 5 shows how a bigger σ results in more overlapping base functions, thereby making every input pattern's output value dependent on more centers.

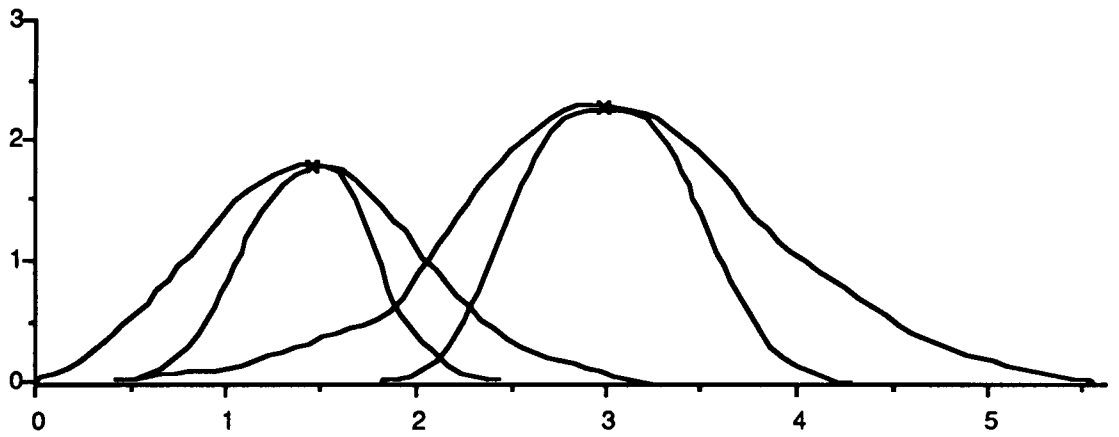

Figure 9: Two base functions with different scaling factors σ.

Given only 25 centers and a Gaussian base function, GRBF was still able to learn the training set reasonably well, but the number of words of the test set that were classified correctly reduced to about half of what they were when using 120 centers.
The number of epochs is not really comparable to the number of epochs used by backpropagation, because GRBF completes each epoch much faster. To train the network on the 100-word training set, given 100 centers, GRBF needed about 20 cpu minutes per 100 epochs on a sparc sun 4 workstation; on the 1000-word training set with 80 centers, it ran for about 3 hours per 100 epochs.

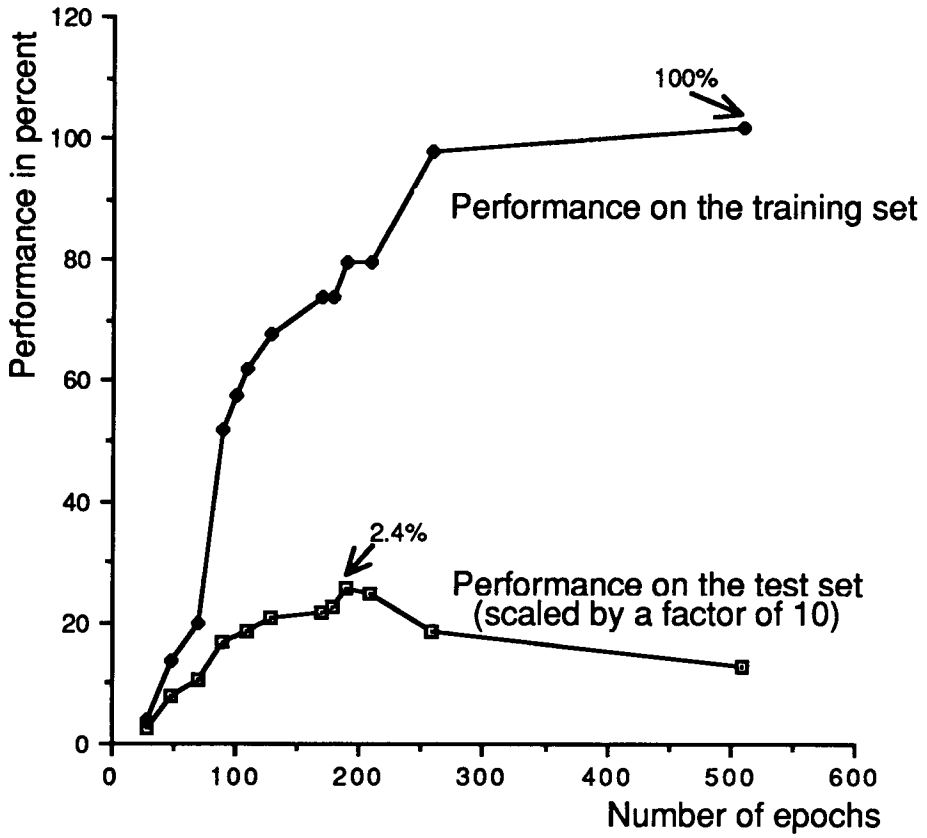

Figure 10: Correlation between performance and the number of training epochs. (50 word training, 1000 word test set, 80 centers)

The main problem was that GRBF overfitted the data when trained too long. As Figure 10 shows, a perfect fit on the training set decreases performance on the test set. To avoid overtraining the network, it is necessary to stop training before maximum performance on the training set is reached. This saves a lot of computation time since the network has to be trained for only about 20% of the epochs that would be needed to achieve a perfect fit. However, in practice the problem is to find the proper cutoff without evaluating the network's performance too often.

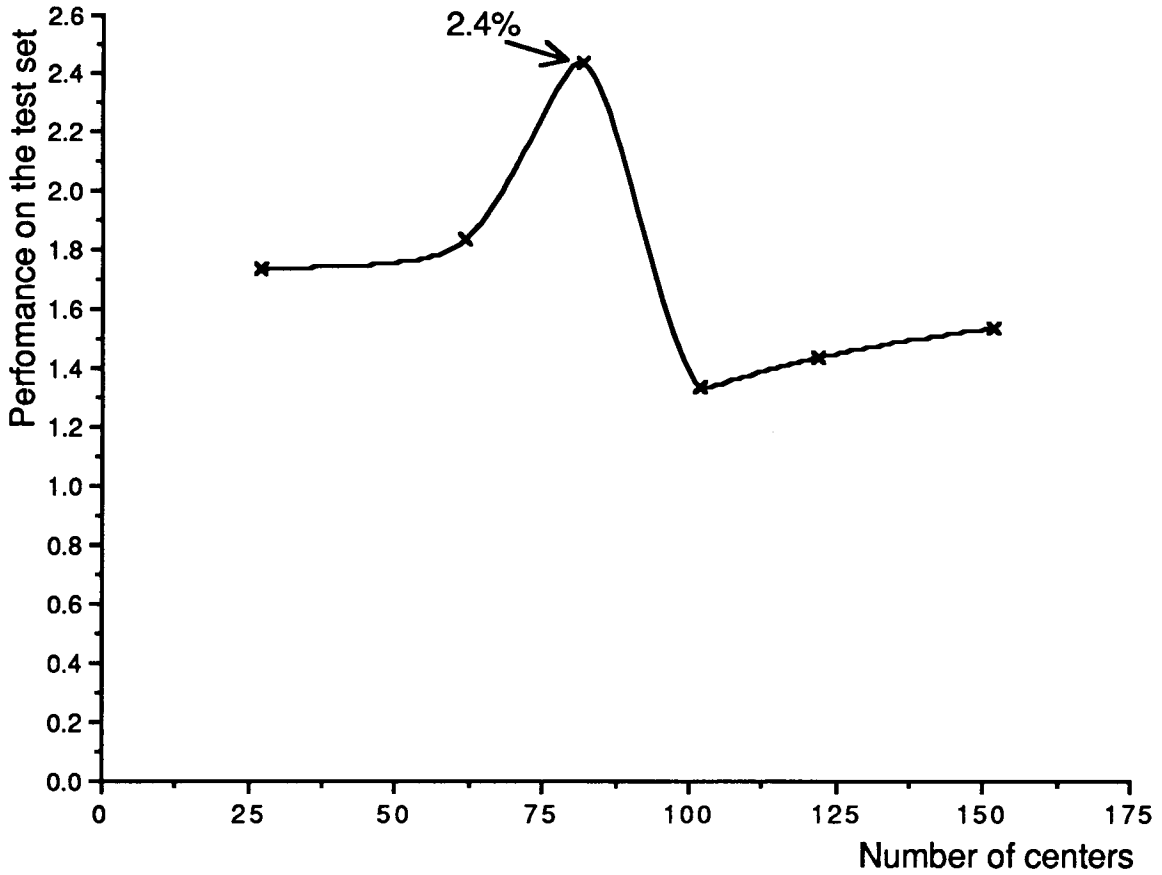

Figure 11: Correlation between performance and the number of centers. (50 word training, 1000 word test set)

The proper choice of the number of centers also influences GRBF's performance on the test set (see Figure 11). Too few centers hinder GRBF, while too many centers do not enforce enough generalization.

Table 3: Performance on the 100-word training set, base function $\exp(-x^2/\sigma^2)$

#centers	#epochs		WORDS	LETTERS	(PHON/STRESS)		BITS
80	4000	TRAIN:	97.0	99.6	99.7	99.9	100.0
		TEST :	2.7	55.1	65.9	73.0	94.5
100	100	TRAIN:	26.0	80.6	86.1	90.3	98.3
		TEST :	3.2	51.9	61.3	76.6	94.8
100	300	TRAIN:	52.0	92.0	96.5	94.7	99.5
		TEST :	3.9	55.3	65.1	76.2	94.9
100	500	TRAIN:	72.0	95.6	98.4	96.8	99.8
		TEST :	3.5	55.6	66.3	74.8	94.8
100	2000	TRAIN:	98.0	99.7	99.9	99.9	100.0
		TEST :	2.7	55.8	67.5	72.9	94.7
150	100	TRAIN:	53.0	91.3	94.4	95.9	99.4
		TEST :	3.6	53.0	62.3	76.4	94.8
150	300	TRAIN:	79.0	96.9	98.4	98.5	99.8
		TEST :	3.7	54.8	64.3	75.6	94.8
150	500	TRAIN:	96.0	99.5	99.7	99.7	100.0
		TEST :	3.9	54.7	65.0	74.1	94.7
150	1000	TRAIN:	98.0	99.7	99.9	99.9	100.0
		TEST :	3.1	54.2	65.3	73.3	94.6
For comparison:							
ID3 (no CHI^2)		TRAIN:	97.0	99.6	99.7	99.9	100.0
		TEST :	2.0	47.3	64.1	65.8	94.0
BP		TRAIN:	80.0	96.4	97.3	98.9	99.7
		TEST :	3.7	55.2	66.1	75.5	94.4

Table 4: Performance on the 1000-word training set, base function $\exp(-x^2/\sigma^2)$

#centers	#epochs	WORDS	LETTERS	(PHON/STRESS)	BITS
80	700	TRAIN: 8.2	65.0	72.8 83.8	96.5
		TEST : 6.0	60.5	69.6 81.2	95.9
100	800	TRAIN: 29.7	81.9	89.0 89.4	98.4
		TEST : 15.7	71.3	80.7 83.0	97.0
150	600	TRAIN: 38.2	85.1	91.6 91.0	98.8
		TEST : 17.8	72.6	82.1 83.3	97.1
For comparison:					
ID3 (no CHI^2)		TRAIN: 96.1	99.4	99.7 99.7	100.0
nearest ph/str		TEST : 9.6	65.6	78.7 77.2	96.4
BP (30 epochs)		TRAIN: 64.4	93.8	96.8 96.5	99.5
		TEST : 14.7	70.9	81.1 81.4	96.6

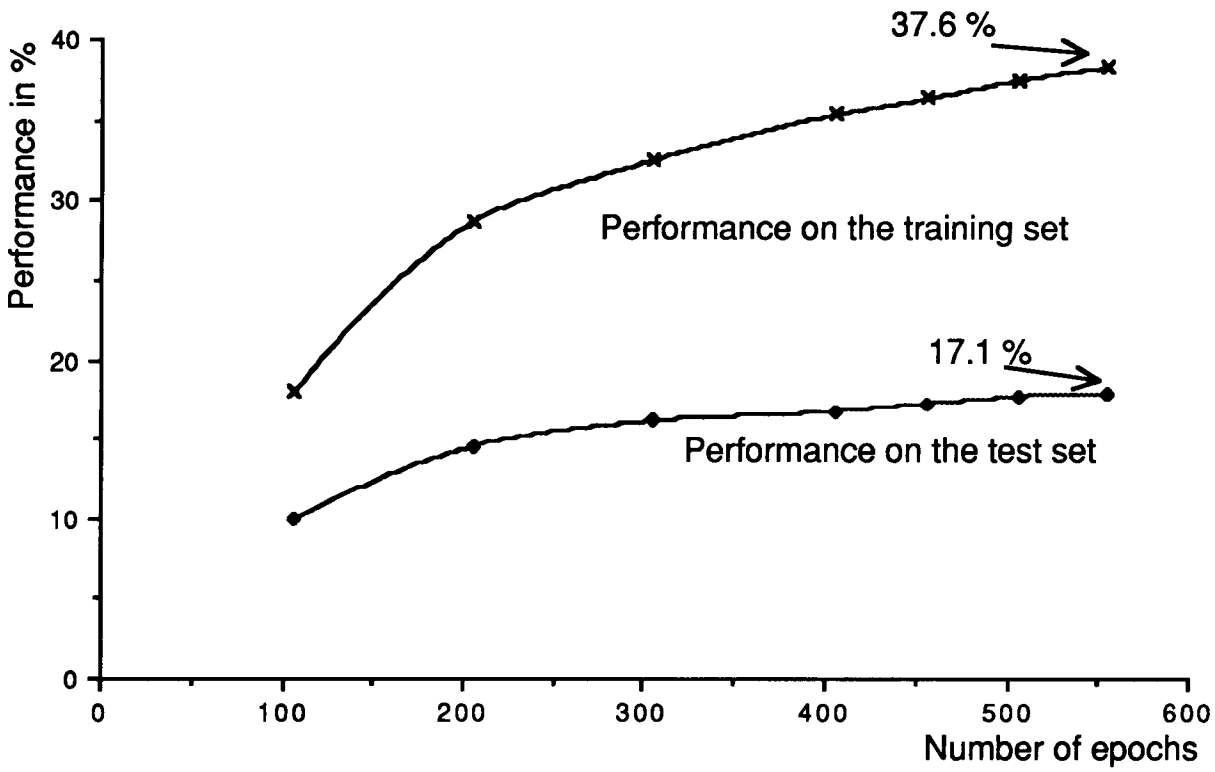

Figure 12:Correlation between the number of epochs and performance (1000 word training, 1000 word test set).

9. FUTURE WORK

9.1. Work Beyond the Initial Tests

When testing GRBF, I have carried out experiments with different numbers of centers and trained the network down to different remaining total summed squared errors. These experiments justify only assumptions about the algorithm's general behavior.
Future work should include a more rigorous testing of the interdependence of performance and the number of centers that were used to train the network. Another interesting question is, whether it is necessary to train the network as well as possible or if a cutoff can be used to shorten training time. Both questions address general problems with any kind of neural networks and are not specific to GRBF. See [Lang,1990] for suggestions how to deal with these tasks.

9.2. Improvements / Enhancements of the Algorithm

The current implementation is based on two underlying assumptions. First, we deal only with correct data. The regularization parameter λ was assumed to be related to an estimate of the noise and therefore set to zero. Given a nonzero regularization parameter, the choice of the constraint operator P would influence the algorithm and result in different gradient terms.
Secondly, only Gaussian base functions were used to train the network, although any function satisfying Michelli's condition [Michelli,1986] can be used. Most of these functions require the addition of a polynomial to achieve differentiability (smoothness). We used only Gaussian base functions, because they are easier to handle than the more general functions that satisfy Michelli's condition, and there is no reason to believe that these functions would give better results than radial Gaussian base functions.
To enhance the implementation with a nonzero regularization parameter λ and non-Gaussian base functions, only the gradient terms have to be changed.
Since some experiments showed that GRBF is sensitive to the proper initialization of its centers, nontrivial initialization methods should be used to find the initial locations for the centers.

9.2.1. Generalizing GRBFs to HyperBFs

Poggio and Girosi [Poggio,1990] suggested a more general form of GRBF. A GRBF network uses the same base function for every center. HyperBFs are the superposition of different types of base functions, They parameterize the base functions of every center. For example, in the current implementation I used the base function

$$\exp(-x^2/\sigma^2), \text{ with } \sigma = \sqrt[2]{0.05}.$$

A HyperBF implementation would also parameterize the factor σ. HyperBFs require learning three different types of parameters: The center coordinates, their weights to the output units, and the scaling of the base functions.

10. CONCLUSIONS

During recent years, many researchers have shifted their focus of research from symbolic computation to neural networks. Development of algorithms of the backpropagation family showed that neural networks are an alternative to the long followed path of symbolic computation.

I showed in this research that Generalized Radial Basis Functions invent a new class of networks which are superior in performance to backpropagation. GRBFs have some advantages over backpropagation as well as some disadvantages. They share with every neural network implementation the problem of finding the optimal number of centers and training epochs.

The way a GRBF network is trained shows the main advantage of this approach: Every step makes intuitive sense and an outside observer could follow the algorithm's actions and interpret the results in a meaningful way. We assume that the centers represent prototypes of the data after training. Although any center might not resemble a real data point, it captures information about its vicinity.

Since GRBFs are developed out of a theoretical framework based on regularization theory, research about further developments as well as about the limitations of this approach can be based on the underlying theory. This well-studied theory should help GRBF to achieve broad acceptance in the scientific world.

Although the current implementation is only a first attempt and further work might improve performance, the tests I have carried out gave encouraging results. GRBF outperformed backpropagation on the two-spiral problem and on the nettalk task.

A GRBF specific problem is the high number of centers required to achieve good performance. It takes GRBF about twice as long as backpropagation to train a network. A fundamental flaw of GRBF is its inability to distinguish between important and unimportant features of the data. Every input dimension is equally important to the algorithm, and GRBF has no mechanism to recognize and ignore useless features.

In summary, I think the invention of the Generalized Radial Basis Functions offers a worthwhile alternative to other neural network algorithms such as backpropagation.

REFERENCES

[Braess,1986] Braess,D. "Nonlinear Approximation Theory", Springer Verlag, Berlin, 1973.

[Dietterich,90] Dietterich, T.G. "Machine Learning", Annu. Rev. Comput.Sci. 4:255-306, 1990.

[Dietterich,90] Dietterich, T.G., Hild, H. and Bakiri, G. "A Comparative Study of ID3 and Backpropagation for English Test-to-Speech Mapping", Tech. Report, Oregon State University, Comp. Sc. Dept., 1990.

[Dreyfus,1986] Dreyfus, H.L. and Dreyfus, S.E. "Mind over Machine", The Free Press, New York, 1986.

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

[Fahlman,1990] Fahlman, S.E. and Lebiere, C. "The Cascade-Correlation Learning Architecture", Tech Report CMU-CS-90-100, Carnegie Mellon University, Dept. of CS,1990.

[Lang,1988] Lang, K.J. and Witbrock, M.J. "Learning to Tell Two Spirals Apart" in Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann,1988.

[Lang,1990] Lang, K.J., Witbrock, M.J. and Hinton, G.E. "A time-delay neural network architecture for isolated word recognition", Neural Networks,3:23-43,1990.

[Lorentz,1986] Lorentz, G.G "Approximation of Functions", Chelsea Publishing Co., New York, 1976.

[MacQueen,67] MacQueen, J. "Some methods of classification and analysis of multivariate observations." In LeCam, L.M. and Neyman, J., editors, Proc. 5th Berkeley Symposium on Math., Stat., and Prob., pg 281. University of California Press, Berkeley, CA, 1967.

[Minsky,1969] Minsky, M.L. and Papert, S. "Perceptrons", MIT Press, Cambridge, MA, 1969.

[Mitchell,1980] Mitchell, T.M., "The need for biases in learning generalizations", Report No. CBM-TR-117, Rutgers University, Dept. of CS, 1980.

[Parzen,1962] Parzen, E. "An estimation of a probability density function and mode.", Ann. Math. Statis., 33:1065-1076,1962.

[Poggio,1989] Poggio, T. and Girosi, F. "A Theory of Networks for Approximation and Learning.", A.I. Memo No. 1140, C.B.I.P. Paper No.31, Massachusetts Institute of Technology,1989.

[Poggio,1990] Poggio, T. and Girosi, F. "Regularization Algorithms for Learning That Are Equivalent to Multilayer Networks.", Science, 247:978-982,1990.

[Powell,1977] Powell, M.J.D. "Restart Procedures for the Conjugate Gradient Method", Mathematical Programming, 12:241-254, 1977.

[Quinlan,1986] Quinlan, J.R. "Introduction to Decision Trees.", Machine Learning,1:81-106,1986.

[Rosenberg,87] Rosenberg, C.R. and Sejnowski, T.J. "Parallel networks that learn to pronounce english text.", Complex Systems, 1:145-168,1987.

[Rumelhart,86] Rumelhart, D.E. and McClelland, J.L. "Learning internal representations by error propagation." Parallel Distributed Processing, MIT Press, Cambridge, MA,1986.

[Rumelhart,88] Rumelhart, D.E. and McClelland, J.L. "Explorations in Parallel Distributed Processing", MIT Press, Cambridge, MA, 1988.

APPENDIX

APPENDIX

Network file formats:

A network description file ending with the suffix .net specifies the network architecture:

```
definitions:
nunits 80                {the network has  80 centers      }
ninputs 203              { "      "     "  203 input units }
noutputs 26              { "      "     "  26 output  "    }
end
```

A start-up file (.str) contains commands that initialize the network configuration and set the values of various parameters of the network:

```
get network t50.net          {read in the network definitions      }
get patterns corpus.50.pat   {load the pattern file                }
set ecrit .001               {set the exit criterium               }
set dlevel 3                 {set display level                    }
set slevel 1                 {set log level                        }
set lflag 1                  {set learning flag to true            }
set mode lgrain epoch        {train the network in epochs          }
set nepochs 100              {every epoch runs for 100 iterations  }
tall                         {test all patterns                    }
```

A template file (.tem) is used to specify the appearance of the display screen and the way in which various display objects, called templates, will appear on the screen:

```
define: layout
                                pname ipatterns   tpatterns (scaled by 10)
epoch  $      tss    $          $     $          $

cpname $      pss    $

centers(scaled by 100):
 $

output(scaled by 100):
$
weights(scaled by 100):$
 $                              $
end
epochno     variable    1     $     n     epochno     5 1.0
tss         floatvar    1     $     n     tss         7 1.0
cpname      variable    2     $     5     cpname      -5  1.0
pss         floatvar    2     $     n     pss         7 1.0
env.pname   vector      0     $     2     pname       v 4  0 0  5
env.ipat    matrix      0     $     n     ipattern    h 3  10.0 0  5 0 2
env.tpat    matrix      0     $     n     tpattern    h 3  10.0 0  5 0 1
centers     vector      3     $     7     weights     h 6  100.0 0  5
output      vector      3     $     n     activation  h 3  100.0 0  26
weights     vector      3     $     n     weights     h 4  100.0 0  10
```

The pattern file (.pat) format:

A pattern file contains for each pattern a pattern name followed by 'ninput' input values and optional 'noutputs' output values.

The weight file (.wts) format:

A weight file contains the coordinates of the centers and their weights to the output units. The format is:
'nunits' weights to each output unit are followed by the coordinates of each center. All values are written as floating point numbers and separated by a space.