Combinatorial Evolution of Feedforward Neural Network Models for Chemical Processes

by

Gregor Peter Josef Schmitz

Dissertation submitted in accordance with the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ENGINEERING SCIENCE

at the

University of Stellenbosch

Supervisor: Professor C. Aldrich

November 1999

Declaration

I hereby certify that this dissertation is my own original work, except where specifically acknowledged in the text. Neither the present dissertation nor any part thereof, has previously been submitted at any other university.

Gregor Schmitz

SYNOPSIS

Neural networks, in particular feedforward neural networks architectures such as multilayer perceptrons and radial basis function networks, have been used successfully in many chemical engineering applications. A number of techniques exist with which such neural networks can be trained. These include backpropagation, k-means clustering and evolutionary algorithms. The latter method is particularly useful, as it is able to avoid local optima in the search space and can optimise parameters for which there exists no gradient information. Unfortunately only moderately-sized networks can be trained by this method, owing to the fact that evolutionary optimisation is extremely computationally intensive. In this paper, a novel algorithm called combinatorial evolution of regression nodes (CERN) is proposed for training non-linear regression models, such as neural networks. This evolutionary algorithm uses a branch-and-bound combinatorial search in the selection scheme to optimise groups of neural nodes. The use of a combinatorial search, for a set of basis nodes, in the optimisation of neural networks is a concept introduced for the first time in this thesis. Thereby it automatically solves the problem of permutational redundancy associated with the training of the hidden layer of a neural network. CERN was further enhanced by using clustering, which actively supports niches in the population. This also enabled the optimisation of the node types to be used in the hidden layers, which need not necessarily be the same for each of the nodes. (i.e. a mixed layer of different node types can be found.) A restriction that does apply is that in order to make the combinatorial search efficient enough, the output layer of the neural network needs to be linear.

CERN was found to be significantly more efficient than a conventional evolutionary algorithm not using a combinatorial search. It also trained faster than backpropagation with momentum and an adaptive learning rate. Although the Levenberg-Marquardt algorithm is nevertheless significantly faster than CERN, it struggled to train in the presence of many non-local minima. Furthermore, the Levenberg-Marquardt learning rule tends to overtrain, (see below) and requires a gradient information.

CERN was analysed on seven real world and six synthetic data sets. Oriented ellipsoidal basis nodes optimised with CERN achieved significantly better accuracy with fewer nodes than spherical basis nodes optimised by means of k-means clustering. On the test data multilayer perceptrons optimised by CERN were found to be more accurate than those trained by the gradient descent techniques, backpropagation with momentum and the Levenberg-Marquardt update rule. The networks of CERN were also compared to the splines of MARS and were found to generalise significantly better or as well as MARS. However, for some data sets, MARS was used to select the input variables to use for the neural network models. Networks of ellipsoidal basis functions built by CERN were more compact and more accurate than radial basis function networks trained using k-means clustering. Moreover, the ellipsoidal nodes can be translated into fuzzy systems. The generalisation and complexity of the resulting fuzzy rules were comparable to fuzzy systems optimised by ANFIS, but did not result in an exponential increase of the number of rules. This was caused by the grid-partitioning employed by ANFIS and for data sets with a relatively high dimensionality, in comparison with the data points, the resulting generalisation was consequently much poorer than that of the CERN models.

In summary, the proposed combinatorial selection scheme was able to make an existing evolutionary algorithm significantly faster for neural network optimisation. This made it computationally competitive with traditional gradient descent based techniques. Being an evolutionary algorithm, the proposed technique does not require a gradient and can therefore optimise a larger set of parameters in comparison to traditional techniques.

OPSOMMING

Neural netwerke, veral dié met voorwaartsvoerende argitekture soos multilaag-perseptrons en radiaalbasisfunksie-netwerke, is al suksesvol in verskeie chemiese ingenieurstoepassings aangewend. Daar bestaan 'n aantal tegnieke waarmee sulke netwerke ontwikkel kan word. Hierdie sluit truplanting, k-gemiddelde groepering en evolusionêre algoritmes in. Die laaste metode is besonder handig omdat dit in staat is om lokale minima in die soekruimte te vermy en dit kan parameters optimeer waarvoor daar geen hellinginligting bestaan nie. Ongelukkig kan slegs netwerke van matige groottes hiermee opgelei word, aangesien evolusionêre optimering uitermate berekeningsintensief is. In hierdie tesis word 'n nuwe algoritme, genaamd samevoeging evolusie van regressienodes (CERN), voorgestel om nie-lineêre regressiemodelle, soos neurale netwerke, op te lei. Hierdie evolusionêre algoritme gebruik 'n "vertakking-en-begrensing" samevoegende soektog in die uitkiesskema om groeperings van neurale nodes te optimeer. Die gebruik van 'n samevoegende soektog vir 'n versameling van basisnodes in die optimering van neurale netwerke is 'n konsep wat die eerste maal in hierdie tesis voorgestel word. Daardeur los dit die probleem van permuterende oorbodigheid, verbind met die oplei van die versteekte laag in 'n neurale netwerk, op. CERN is verder verbeter deur die gebruik van groepering, wat nisse in die populasie aktief ondersteun. Dit het die optimering van die nodetipes in die versteekte laag, wat nie noodwendig almal dieselfde hoef te wees nie, moontlik gemaak (d.w.s. 'n gemengde laag van verskillende nodetipes kan gevind word). 'n Heersende beperking is dat, ten einde die samevoegende soektog doeltreffend genoeg te maak, die uitsetlaag lineêr moet wees.

Eksperimente het aangetoon dat CERN beduidend meer doeltreffend was as 'n konvensionele evolusionêre algoritme, wat nie 'n samevoegende soektog gebruik is nie. Dit het ook vinniger opgelei as truplanting met momentum en 'n aanpassende leertempo. Alhoewel die Levenberg-Marquardt-algoritme steeds vinniger is as CERN, het dit gesukkel om op te lei in die teenwoordigheid van verskeie nie-lokale minima. Verder was die Levenberg-Marquardt-algoritme geneig om oor te pas (sien hieronder) en benodig dit hellinginligting.

CERN is ondersoek by wyse van sewe werklike en ses kunsmatige datastelle. Gerigte ellipsoïdalebasisnodes wat met CERN geoptimeer is, het beduidend beter akkuraatheid met minder nodes bereik as sferiese nodes wat geoptimeer is met k-gemiddelde groepering. Multilaag-perseptrons wat geoptimeer is met CERN, was meer akkuraat te wees as diesulkes wat met aflopende hellingtegnieke, truplanting met momentum en die Levenberg-Marquardt opdateringsreël opgelei is. Die netwerke van CERN is ook vergelyk met die latfunksies van

MARS (Eng: multi-adaptive regression splines) en het beduidend beter veralgemeen as MARS. Vir sommige datastelle was MARS egter gebruik in die keuse van insetveranderlikes vir die neurale netwerke. Netwerke bestaande uit ellipsoïdale basisfunksies wat met CERN gebou is, was meer kompak en akkuraat as radiaalbasisfunksienetwerke, wat opgelei is met k-gemiddelde groepering. Daarby kan die ellipsoïdale nodes vertaal word na wasige stelsels (Eng: fuzzy systems). Die veralgemening en kompleksiteit van die ooreenstemmende wasige reëls was vergelykbaar met wasige stelsels wat geoptimeer is met ANFIS, maar het nie op 'n eksponensiële toename in die aantal nodes uitgeloop nie. Dit is veroorsaak deur die roosterindeling wat deur ANFIS gebruik word en vir datastelle met 'n relatief hoë dimensionaliteit in verhouding tot die datapunte was die gevolglike veralgemening dus baie slegter as vir die CERN modelle.

Ter opsomming was die voorgestelde samevoegende uitkiesskema in staat om 'n bestaande evolusionêre optimeringsalgoritme van 'n neurale netwerk beduidend vinniger te maak. Dit het die uitkiesskema mededingend met tradisionele aflopende hellingtegnieke gemaak. Synde 'n evolusionêre algoritme, het die voorstelde tegniek nie 'n helling nodig nie en kan daarom 'n groter versameling van parameters optimeer in vergelyking met tradisionele tegnieke.

Acknowledgements

First of all I would like to thank Professor Chris Aldrich for the valuable guidance and support that he has given over the past years. I would also like to thank him for assisting me in communicating this work.

I furthermore wish to express my gratitude towards:

- Francois Gouws for intellectual discussions and for keeping the operating systems running.
- Dr. Derick Moolman of Crusader Systems for his motivation and interest in my work.
- Professor Jan Du Plessis for making simulation workstations available.
- The Foundation of Research and Development (FRD) for partially sponsoring this thesis.
- R. Storn and K. Price for making the source code of the DE algorithm available.
- The Delve Repository of methods and data at the University of Toronto for benchmark data sets as well as the MARS algorithm.
- UCI Machine Learning repository for keeping the Sonar and the Abalone data set available.
- My current employers Crusader Systems for allowing me to make the final alterations during work hours

I would also like to thank my mother for allowing me to focus on my research and for the excellent meals and Nelius Goosen for the spimericks and loonerisms. ☺

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Modelling in the chemical and metallurgical industry

As a result of the improvements in the technology of electronic sensors and computer networking combined with cheaper data storage, it has become possible to sample and store increasing amounts of data (Taylor 1989). The motivation for measuring and collecting these data sets is an increasing demand for more efficient processing operations (Saraiva & Stephanopoulos, 1992). These include the analysis of the dependability, i.e. availability and fault tolerance of large scale control strategies in petrochemical plants (Picciolo, 1989), neural network based control systems (Lee & Park, 1992; Chovan et al., 1996) and diagnostic systems (Basta, 1988; Calandranis et al., 1990). Further applications make use of pattern analysis by using artificial neural networks. Examples in this field are the nondestructive evaluation of hidden chemical corrosion with a high degree of accuracy, on both titanium and aluminum alloys (Bowers & Sammells, 1995) and a machine vision system that monitors the froth surface during industrial and batch flotation for decision support (Moolman et al., 1996; Aldrich et al., 1997). The process improvements are made possible through the empirical analysis of the collected process data (Saraiva & Stephanopoulos, 1992). Accurate models of the systems improve the process control strategies, as well as the diagnostic systems used to monitor plant performance. Furthermore, the diagnostic system itself can be seen as a model of the process faults that could occur. In cases where fundamental models of the plant are not available, the models need to be formed empirically using historic data.

Extended fundamental research has been undertaken regarding chemical and metallurgical processes in order to produce steady-state as well as dynamic models for processing operations. These models are developed from fundamental principles, such as mass and energy balances, usually giving rise to sets of differential equations, which require analytical or numerical solutions. Although fundamental models can be formed with minimal data and are able to extrapolate to almost any feasible condition, they require a thorough and accurate understanding of the underlying physics and chemistry specific to the process. Unfortunately, in many cases this cannot be obtained owing to the stochastic nature of the process. Examples of this include cases where turbulent flow occurs, or where complicated interfacial phenomena may depend on initial and boundary conditions, whiczh are difficult to predict. Alternatively, the task may be too time-consuming and expensive. Moreover, approximations and assumptions made when building the equations and their solutions may be invalidated by slight changes in process conditions or may simply be inadequate.

1.2 Empirical process models

In essence, empirical modelling is a search for a data-based model, that will represent the true process as well as possible. Prior information regarding the underlying process, that generated the data, may be included (Duda & Hart, 1973). In its broadest sense, empirical modelling may also include sets of exemplars and case-based reasoning (Kolodner, 1993). Other forms of empirical modelling include frames, semantic networks and sets of rules built by human experts in so-called *expert systems* (Basta, 1988). The remaining empirical models can be divided into parametric models and non-parametric models. However, this grouping is not discrete and a more realistic perspective would be to regard parametric and non-parametric models as the extreme points of a continuum. The following section discusses the advantages and disadvantages of models formed using human heuristics, which are also essentially empirical models. The aspects of parametric and non-parametric models will then be discussed.

1.2.1 Heuristic models

As indicated above, one approach is to rely on heuristics to build a model. These heuristics are typically a mixture of some fundamental principles, as well as past experience. The heuristic knowledge of process experts and human operators of the chemical process is translated into "if .. then …" type models. Such models usually take the form of fuzzy logic systems (Zadeh, 1965; 1971) or expert systems. The questions that need to be asked are:

1) Can humans express their knowledge into "if…then" types of rules?

2) Is knowledge acquisition from human experts and operators sufficient and reliable?

Muggleton (1990), and Fayyad and Irani (1992) found that humans find it difficult to express their knowledge in terms of concise, "if…then…" type rules. Furthermore, different experts exhibit cognitive biases such as overconfidence and oversimplification (Johannsen & Alty, 1991; Kattan, 1994). In addition, the knowledge human experts have is often incomplete and episodic, rather than systematic (Sun, 1994). A likely reason why operators' knowledge is incomplete, is that they work in shifts, rather than continuously. In addition, they do not pay attention to all the instrument readings. In many processes such as flotation, the resulting recovery of the precious metal for a certain time period is only known after the event and sometimes days or even weeks can elapse. By this time, the operator would have forgotten many important details of his shift. There is therefore often a need for the formation of empirical models from the data. Amongst other tests, an expert, if available, can be used to validate predictions of such a model.

1.2.2 Parametric models

Parametric models have been applied to many processing systems in industry. Linear regression (Seber 1977) is the best known technique. It is used to determine the coefficients of the model: $y = \Sigma_j w_j x_j + w_0$ for a sum of squares error measure or likelihood methods. Although the model is easily understood, it will give poor results if the process generating the data is highly non-linear. Linear logistic regression is similar to linear regression, but is used for data problems with a categorical output (i.e. for classification problems). Again, if the classes are not linearly separable, the model will not prove adequate to solve the problem. In non-linear regression (Seber & Wild, 1988; Ratkowski, 1990) a regression model involves certain non-linear functions of the input variables, for example $y = a_1x_1*x_2 + a_2x_3 + a_3\exp(a_4x_4)$. Here the values of $a_1..a_4$ can be found by using the Gauss-Newton method (Seber, 1989). Provided that the parametric model is of the correct form, non–linear parametric models possess some reasonable extrapolative properties.

However, the linear or non-linear model may not always be appropriate. When the behaviour of a system is not known in advance, the explicit specification of these model structures can be difficult. For example, unknown non-linear interactions among the influencing factors can be predominant in chemical systems, but are usually poorly understood. Sometimes, if there are more than two or three inputs, the type of non-linearity cannot be determined from the data. The sheer number of non-linear functions that can be formed with only a few variables makes it unfeasible to search through the various possibilities in an uncontrolled fashion. Even all the possible 2nd order models become too many to be attempted for problems with many inputs (e.g. $d > 10$), even when combinatorial search methods (Furnival & Wilson, 1974) are employed.

It can be attempted to determine the type of linear or non-linear relationships through extensive research. This is not always possible, for example when the properties of a given plant cannot necessarily be duplicated on a smaller scale. Experiments conducted on the plant could be expensive since they can interfere with production. An alternative to a model with fixed parameters is the use of non-parametric models.

1.2.3 Non-parametric and weakly parametric models

Non-parametric techniques, such as multi-adaptive regression splines (Friedman, 1991), artificial neural networks (Haykin, 1994), projection pursuit networks (Friedman & Stuetzle, 1981) and radial basis function networks (Powell, 1985), have been applied successfully to a multitude of non-linear process systems where other approaches have proven to be less than adequate. These models form complex non-linear relationships among the measured variables, even for multivariate problems. They are well suited to cope with noise in the data sets. Some neural networks are specially used for the purposed of eliminating noise from data set (Terry & Himmelblau, 1993). Noise in the data set is common in practice, especially in the process industries.

1.2.3.1 Non-parametric and weakly parametric regression techniques

As discussed previously, parametric models have a fixed structure and only the coefficients are determined from the data. The advantage is a more reliable model, should the underlying process have the same structure. However, if the process deviates from the assumed model structure and the process variables have different interactions, the parametric model is not sufficiently flexible to deal with interactions not specified in the model structure. Models that tend to be classified as non-parametric and weakly parametric on the other hand are more flexible and can adapt their structure and/or parameters to any data set. Many of them also have a fixed architecture, however they have many interacting parameters, which are responsible for their flexibility. The coefficients (e.g. weights) in non-parametric models also tend to have less meaning attached to them than the parameters of parametric models. Several of these techniques, (e.g. multilayer perceptrons and radial basis function networks, discussed later) belong to the class of universal approximators (Hornik et al., 1989). A universal approximator can fit any arbitrary function within an arbitrarily chosen accuracy. These models can usually be used for generalising to new unseen data. A disadvantage of the approach is that for most data sets these models cannot extrapolate as well as the parametric models. They should thus be used in conjunction with other models or heuristics for data outside their training range.

Computer resources such as storage and CPU time are becoming cheaper all the time, hence analysis of the data by computers is relatively cheap when compared with human time. It is therefore desirable to have data modelling automated as much as possible. Although many algorithms still have a lot of learning settings, most of those used in the construction of non-parametric models from the data can be fully automated, when default settings are used. Of course presenting the model-building algorithms with data already implies that some other process or a human already specified which features might be relevant. It should be noted that this is not a trivial task, since a basic understanding of the process is generally required. The algorithm presented in this thesis is a non-parametric technique, which can automatically build a non-parametric model from the data.

1.2.3.2 Non-parametric regression techniques originating from the statistical sciences

Hastie and Tibshirani (1994) have written overviews of non-parametric regression techniques. A brief summary of the most well known statistical curve fitting techniques is given below. The resulting models usually take the form of a linear combination of non-linear terms. It should be noted beforehand, that some of these techniques overlap both with connectionist techniques and/or parametric models.

Volterra series are used for non-linear system identification. Consider a discrete non-linear system of the form:

$$\mathbf{x}_{t+1} = G(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{Equation 1.1}$$

$$y_t = H(\mathbf{x}_t, u_t) \qquad \text{Equation 1.2}$$

where G and H are non-linear functions; $\mathbf{x}_t$ is the state vector; y_t the output and u_t the input. The non-linear system can also be described by a so-called NARMA (non-linear auto-regressive moving average) model (Box & Jenkins 1970), that is:

$$y_t = f(y_{t-1}, y_{t-2}, \ldots, y_{t-n}, u_{t-1}, u_{t-2}, \ldots u_{t-m}). \qquad \text{Equation 1.3}$$

A non-parametric model could now be used to approximate the unknown function f, given sufficient data points. Volterra polynomials (Schetzen,1980) are functions of the form:

$$f(\mathbf{x}_t)=w_1 + w_2 y_{t-1} + \ldots w_{n+1} y_{t-n} + w_{n+2} u_{t-1} + \ldots + w_{n+m+1} u_{t-m} + w_{n+m+2} y^2_{t-1}$$

$$+ w_{n+m+3} y_{t-1} y_{t-2} + \ldots + w_N u^1_{t-m} \qquad \text{Equation 1.4}$$

The individual terms can be seen as non-linear basis functions and can thus be used to approximate the output y_t. In a paper by Liu et al. (1998), the basis function of the polynomials are selected adaptively in order to fit the data.

Kernel-based regression or kernel smoothers (Gasser et al., 1985) smooth the output over overlapping regions. The contribution of each data point towards the function at a specific point is weighted by the distance between the points. This weighting function is usually a radially symmetric Gaussian function or any other decreasing function. This weighting function is also called the kernel. The structure of such models is the same as that of certain radial basis function networks or 0th order Sugeno Systems (Sugeno, 1988), discussed in more detail in chapter 2.

Instead of a constant contribution of each of the basis functions, each basis node can make a contribution that is a linear combination of the inputs plus a constant. The so-called local linear maps or locally weighted regressions (Cleveland, 1979), thus form a linear model at each of the overlapping receptive fields. The linear model is formed by a weighted regression for the data encircled in each of the receptive fields, with the weight of each data point in regression being the amount of the data point belonging to that receptive field. Each data point will thus belong to several receptive fields. The output at a certain point is thus a weighted average of the outputs of all the linear models, one for each receptive field. The structure in this case is the same as that of certain 1st order Sugeno Systems.

Regression splines (Schultz 1969) represent the output as a piecewise polynomial. The regions for this fit are defined by a sequence of knots or nodes. At these knots, the polynomials are forced to join continuously, up to a certain derivative (usually 2nd order). Smoothing splines have a penalty term, based on the cumulative 2nd derivative, to assist in a smoother fit to the data. A disadvantage of the spline approach is choosing a number and location of the knots. The number of points needed to define the knots will increase exponentially as the dimensions increase. Adaptive splines methods adaptively find these knots. One such algorithm is MARS (multi-adaptive regression splines) developed by Friedman (1991). A full description of it is beyond the scope of this section. Briefly, the algorithm forms a set of linear regression splines and products of linear splines. The algorithm searches through all possible knot locations and adds that spline to the set of basis functions that most reduce the error. It later prunes back some of the basis function. This is done using a generalised cross-validation error plus some ad hoc penalty term that compensates for the extra parameters. The models formed by MARS are usually fairly comprehensive and quick to build. MARS is co-ordinate sensitive, implying that a rotation of the co-ordinate system can improve or degrade the performance. This can be both an advantage and a disadvantage. It can result in better models, when the data contain many inputs that do not really contribute towards the output. The algorithm can more easily ignore the superfluous inputs than other algorithms that attempt various rotations of their basis functions. However, MARS can give poor results when the data is highly correlated such as in time series analysis. In such cases pre-processing the data by using principal component analysis can help (De Veaux & Ungar, 1994).

CART (classification and regression trees), by Breiman et al. (1984) is a machine learning algorithm that builds a decision tree from the data. When used for regression problems, it recursively splits the data to reduce the sum-squared error. The split is on a specific threshold of an attribute. This results in a tree that can be converted to a set of crisp (if.. then..) rules, such as if $x_1 > 0.5$ and $x_2 > 0.3$ and $x_3 \leq -0.4$ then $y = 0.8$. The models formed by CART can also be seen as an adaptive splines of 0^{th} order (Friedman, 1991).

Additive models (Hastie and Tibshirani, 1990) are additions of basis functions, which are dependent on one input only. In this case the model is of the form :

$$f(x)= w_0+ \sum_i w_i\, g_i(x_i) \qquad \text{Equation 1.5}$$

The parameters of the functions usually get optimised using backfitting.

A related technique is projection pursuit, proposed by Friedman and Stuetzle (1981). It is a non-parametric method that models the output as a sum of smooth functions. The algorithm seeks projections (linear combinations of the attributes), which can be used to reduce the remaining sum of squares error. During the search, it uses a smoothing function to smooth the output along the projection tried. When a suitable projection and smoothing functions are found, this function is subtracted from the model. The algorithm continues in this fashion recursively until a certain stopping criterion is met. The model output, to a given input vector **x**, given the parameters w_m, g_m is:

$$f(\mathbf{x}) = \sum_{m=1}^{h} w_m g_m \Big(\sum_i w_{mi} x_i\Big) \qquad \text{Equation 1.6}$$

where g_m is a smoothing function to be learned from the data and h is the number of projections found.

1.2.3.3 Neural networks

When neural networks are to approximate functional relationships, they can also be seen as non-parametric modelling techniques. Many neural network architectures are similar to those of other non-parametric techniques discussed above. For example, radial basis function networks and probabilistic neural networks are strongly related to kernel-based regression. However, the training techniques of neural networks differ significantly. For some of the non-parametric techniques such as neural networks, the training methods mimic those of natural systems. Humans and animals can learn skills such as estimating the path of a flying ball without requiring a fundamental model of the process. A person learns this and other skills from practice (examples). The brain learns from each example by making incremental adjustments in order to improve the skill. Artificial neural networks, although often having very different architectures from the natural brain, are also made up of connected units that can be trained by the repeated presentation of examples. More will be said about the structure of neural networks in the next chapter.

1.3 Optimising non-parametric models

The efficiency of process operations can be directly linked to the accuracy of the models describing these systems. As a consequence, there has been a surge in techniques that can better optimise non-parametric models such as those mentioned above. Examples are evolutionary algorithms, simulated annealing and gradient descent techniques. Besides these generic optimisation techniques, algorithms have been developed specific to certain neural network architectures. For example, many non-parametric architectures originating from the statistical community, such as the regression trees build by CART or kernel based regression already have dedicated optimisations routines. Nonetheless, it is possible to optimise a model with the same structural components using more general optimisation approaches. Although these may be slower, they could conceivably find more accurate and/or more compact solutions. For example CART, MARS and the projection pursuit algorithm use greedy heuristics to build the model. This is very efficient but will not always result in optimal models. Thus these types of non-parametric models can possibly draw benefits from other optimisation techniques, such as gradient descent algorithms and evolutionary methods.

Most neural network architectures on the other hand have many different training techniques. Neural networks also have many structural parameters, which can be set. It is often difficult to design appropriate neural network models, since the basic principles governing the processing of information in neural networks are not well understood. As a result, the complex interactions among network units usually make conventional design techniques such as those based on divide-and-conquer principles inapplicable. This is especially the case when the networks continue to grow in size and complexity. Under these circumstances, the human engineering approach is inadequate and more efficient methods are required for the development of neural processing systems. It would therefore be advantageous if the optimisation algorithm could not only optimise the weights of the network, but also the structure of it. An important class of such methods is based on the use of evolutionary techniques.

For example, genetic algorithms have been successfully applied in finding the global optima of various multidimensional functions where local optima in the space of possible solutions are common, such as the optimisation of neural networks where local optima are prevalent in the parameter space of the network. Traditional optimisation techniques, in particular backpropagation, are able to optimise a number of network parameters, e.g. the connection weights of the network, but are unable to handle the optimisation of parameters for which there exists no gradient information. This problem can also be overcome with genetic algorithms. A much broader range of network parameters may therefore be optimised. For example, operations where the activation functions of nodes are adaptively changed from one type to another are possible, using genetic algorithm optimisation. However, despite the computational power of modern computers, only relatively small neural networks can be directly optimised in reasonable time with this method, as hundreds of parameters are typically required to specify even a moderately sized neural network.

The objectives of this dissertation is to improve the efficiency of evolutionary approaches when applied to optimising neural network models. In practice the improved efficiency means that the evolutionary technique is applicable to optimise models for chemical process data and other real world problems of greater size or to optimise networks of greater complexity. The resulting new algorithm should still allow the evolutionary algorithm to optimise parameters for which no gradient exists and furthermore be as capable in overcoming local minima as existing genetic algorithms. Most existing genetic algorithms encode the network parameters using the same generic manner used for other optimisation problems. The encoding does not incorporate enough information about the structure of the neural network, in particular the fact that the network consists of separate nodes. Consequently most of the other genetic operators in particular the selection scheme cannot make sufficient use of this underlying structure. This results in an inefficient selection of nodes from one generation to the next and other problems such as the permutational redundancy, discussed later in chapter 3. The improved algorithm should among other overcome such problems. The novel evolutionary optimisation algorithm called CERN (combinatorial evolution of regression nodes) is therefore proposed in this thesis. CERN is a differential evolution algorithm (Storn & Price, 1995) combined with a combinatorial search technique. The combinatorial search is employed in the selection phase of the genetic algorithm. This new selection scheme, makes extensive use of the structure of the feedforward network and therefore allows the genetic algorithm to run considerably more efficiently than a normal selection scheme. Further improvements also use the fact that the chromosomes actually describe a network that is made up of separate nodes.

The CERN algorithm can be used to evolve nodes of feedforward neural networks and certain other non-parametric models. The structure of these models and their characteristics will be discussed in the next chapter. The algorithm sequentially adds neural nodes to a network, in such a way that the remaining root mean square error (RMS) of the network model is systematically minimised. Existing techniques either optimise the whole network simultaneously or optimise only one node at a time. In contrast to this, CERN optimises a number of nodes simultaneously and therefore effectively looks a few steps ahead when adding new nodes to the network. This compromise between optimising the whole network or optimising one node at a time is a further step towards making the optimisation of neural network models with evolutionary techniques more practical.

1.4 Thesis overview

The CERN algorithm forms the main topic of the thesis. It can be applied to a certain family of connectionist architectures, which will be called regression networks. The next chapter will motivate the use of such connectionist architectures. The architecture will be described and placed in context of the larger class of non-parametric models. Chapter 3 provides a background on optimisation techniques of the aforementioned models. It also reviews the various state-of-the-art training algorithms of connectionist networks. Special attention is given to any techniques using evolutionary principles, fitting the novelties of the CERN algorithm into this framework. Chapter 4 describes the basic CERN algorithm and compares its performance on three standard benchmark data sets with that of other algorithms. The succeeding chapter describes several enhancements and extensions of the CERN algorithm, which among other allow different node types to be selected and optimised automatically. The significance of these improvements is supported by several experiments. In Chapter 6 the CERN algorithm is compared both in training speed and in generalisation capabilities to standard algorithms, such as backpropagation of error and MARS, for several real world case studies and synthetic data sets. The critical reader may wish to first take a glance at this chapter, before reading the details of the CERN algorithm in Chapters 4 and 5. Chapter 7 contains the conclusions of this thesis and closes with suggestions for future extension of the CERN algorithm.

Chapter 2 Feedforward Artificial Neural Networks

Neural networks that are used for supervised modelling belong to the larger class of non-parametric models as previously discussed in chapter 1. Three other such paradigms are statistically based algorithms, machine learning techniques and fuzzy inference systems. The basic statistical techniques were discussed in order to give a better understanding of the methodology behind certain neural network models. Also, certain neural network models such as radial basis function networks could be classified as statistical models or even fuzzy systems.

2.1 Artificial Neural Networks

Artificial neural networks (*ANN*) - also called connectionist models or parallel distributed processing models have been used successfully in many fields. In chemical engineering the techniques have been used amongst other for off-line modelling (Bandyopadnyay et al, 1996), fuzzy modelling (Schmitz & Aldrich, 1998), decision support (Joubert et al., 1996) and control (Lee & Park, 1996). A book by Bulsari (1995) contains some more applications and research results for their use in chemical engineering. As a result of its success and its fascinating background, the field has grown tremendously over the last years. The field now overlaps with fuzzy logic (Roger Jang & Sun, 1993) and statistical curve fitting techniques.

Artificial neural networks aim to achieve the best possible performance in fields such as pattern recognition and data modelling by using dense interconnections of simple computational elements. These neurons are supposed to function similar to the neurons of a brain as we understand it today. Consequently, artificial neural networks consist of numerous single processing *units or nodes*, connected to one another by *weighted* links.

Artificial neural networks are distinguished by their:

- node types
- network architecture (sometimes also called topology)
- and training or learning rules

These will be discussed next:

Each node in the network has input and output connections. Each node applies some formula on its inputs to compute its *output* or *activation*. For example, frequently used nodes, called summation nodes or inner-product nodes, form a sum of the weighted inputs and the result is passed through a non-linear function called the transfer or activation function. Other nodes have a set of reference co-ordinates and can compute the difference between these reference co-ordinates and the co-ordinates of an incoming signal.

Nodes of the same type are usually organised into *layers*. Each layer can have a specific task. For example, the input layer receives the inputs values from the data. The network architecture also determines the number of layers, the number of nodes per layer and which connections exist among the nodes of the layers.

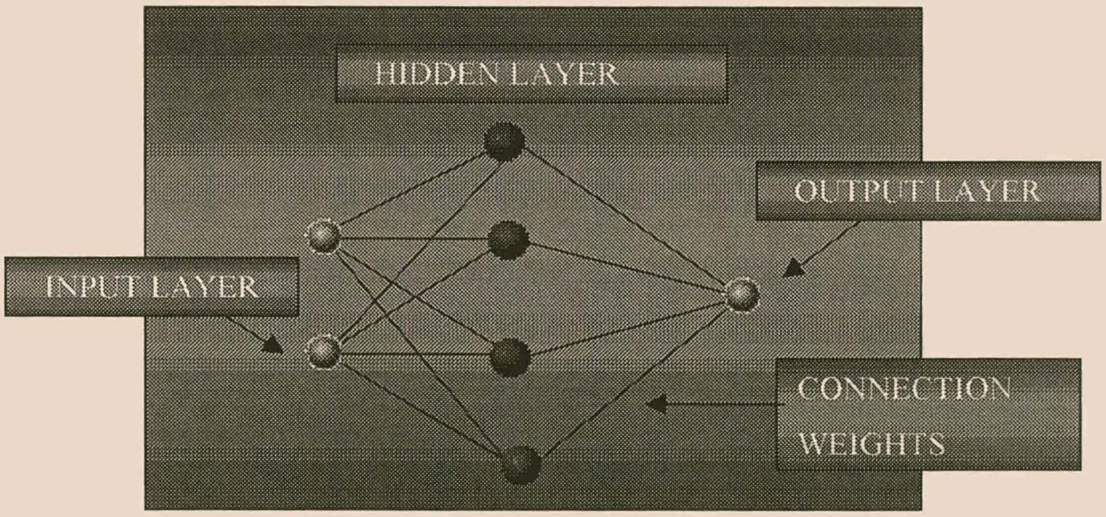

Figure 2.1 A diagrammatic representation of a multi-layer perceptron with a single hidden layer of four nodes.

2.2 Feedforward neural networks for modelling

Neural networks can amongst other tasks be employed for nonlinear empirical modelling, or for model based control. Feedforward neural networks, especially multilayer perceptrons, also sometimes referred to as backpropagation networks, are amongst the most popular techniques for these tasks. Consequently, they have also been applied for this purpose in chemical engineering and related fields. Bowers and Sammells (1995) used feedforward neural networks for the non-destructive evaluation of hidden chemical corrosion in aircrafts. The Fast Fourier Transform electrochemical impedance spectroscopy from the suspect hidden chemical corrosion sites was analysed by a neural network pattern recognition scheme. The trained networks managed to detect the presence and the extent of corrosion with a high degree of accuracy, on both titanium and aluminium alloys, currently used in military and commercial aircraft. In another pattern recognition application the gas emission in a baker's yeast manufacturing process was analysed (Mandenius et al., 1997). Fourteen different gas-sensitive semiconductor devices and an infrared gas sensor monitored the gas emission from the yeast culture. The resulting patterns were used to setup a multilayer perceptron model to estimate cell mass and ethanol concentration.

The output of trained multilayer perceptrons has also been used for control. One such example lies in the control of laser-beam welding (Farson et al., 1992). Here a multilayer perceptron monitors the laser welding process in real-time by analysing the airborne acoustic emissions of the process. A trainable fuzzy logic controller uses the neural net outputs and the degree of weld penetrations to calculate weld speed changes necessary to maintain full weld penetration. Multilayer perceptrons have also been used for model-predictive control of distributed parameter anthracane crystal growth process (Ishida & Zhan, 1995). The neural network is first trained with process input-output data to represent the process. Subsequently, using inversion of the neural network (Linden & Kindermann, 1989), the same neural network is utilised to determine the optimal values of the manipulated variables by the optimiser.

A more detailed description of multilayer perceptrons follows. Thereafter a further commonly-used feedforward network type, the radial basis function network, will be introduced. Radial basis function networks and certain multilayer perceptrons belong to the class of regression networks, the structure of which will also be explained. The algorithm proposed in this thesis can be applied to build such regression networks.

2.2.1 The multilayer perceptron

The widely used architecture of the multilayer perceptron is described below. It can also be used to illustrate the most important aspects of artificial neural networks. Figure 2.1 shows a neural network with a single layer between the input and output layer. The layers between the input and output layer are called *hidden layers*, because during their functionality they do not send or receive signals directly to structures outside the neural network.

If the node types are summation nodes, the specific artificial neural network of figure 2.1 is often referred to as a multilayer perceptron or backpropagation network. In this thesis they will be called multilayer perceptrons, as this distinguishes the learning technique from the network architecture. This will become apparent in the next section. For example, if a hyperbolic tangent (tanh) transfer function is used, the equation of the activation of input node is:

$$\phi_i(x) = \tanh(\Sigma_j w_{ij} x_j + w_{0i}) \qquad \text{Equation 2.1}$$

where **x** is the input vector to the network, W is the weight matrix of the hidden layer. The matrix holds the weight vector of each node in a row. The weights of each node can thus also be seen as a vector. w_{0i} is the weight from the bias, which is a special node that has a constant output of one. The weight w_{0i} therefore has the purpose of an adjustable offset for node i, it is also often called the bias term. The output node of the network follows the same rule as the nodes of the hidden layer, except that now the hidden layer's outputs serve as the new input. Therefore the equation of the neural network that is depicted in figure 2.1 is :

$$\begin{aligned} y(x) &= \tanh(\Sigma_i w_i \phi_j(x) + w_0) \\ &= \tanh(\Sigma_i w_i \tanh(\Sigma_j w_{ij} x_j + w_{0i}) + w_0) \qquad \text{Equation 2.2} \end{aligned}$$

Here **w** is the weight vector of the output node.

The reader can find out more about the origins and the properties of the multilayer perceptron in Haykin (1994). An important aspect of a multilayer perceptron is that it belongs to the class of universal approximators (Hornik et al., 1989). This means that it can form any, possibly unbounded, region in the space spanned by the inputs and can approximate any arbitrary function within an arbitrary chosen accuracy, provided there are enough nodes in the hidden layer. Other well-known neural networks that have this capacity (Park & Sandberg, 1991) are radial basis function networks (Powell 1985; Broomhead & Lowe 1988). Recurrent neural networks have feedback connections in time (Haykin 1994) and are especially designed for modelling of dynamic processes. An architecture proposed by Elman (1990) is perhaps the most successful recurrent network. Although recurrent networks show promising results, the feedback connections cause the networks to be even less comprehensible and predictable than feedforward networks. These, as well as other neural network architectures, are described in more detail in Haykin (1994).

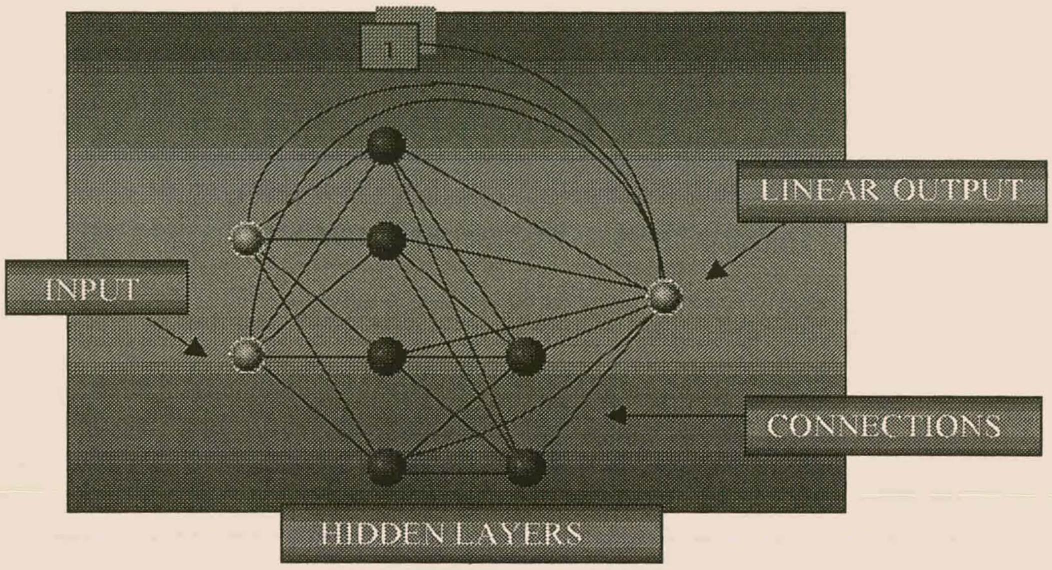

Figure 2.2 A diagrammatic representation of a regression network with two hidden layers.

In order to achieve the highest possible performance, the training rules designate an initial set of weights (or co-ordinates) to each node and specify in which way the weights (or co-ordinates) should be altered in order to learn a given task. As this thesis deals with an optimisation technique, the most well-known techniques and their principles will be discussed. First however, some more background on characteristics of neural network models will be given.

2.3 Basis and regression networks

A *basis network* is a connectionist architecture that consists of an input layer, a hidden layer of non-linear transfer functions and an output layer with summation nodes and a linear transfer function. The output of such a network is thus a linear combination of several non-linear transfer functions applied to the input. All connections are feedforward and the input layer can also be connected directly to the output layer. If the output layer uses a least squares error measure and has a linear transfer function, then the basis network was termed a *regression network* by Van der Walt (1992; Van der Walt et al., 1995), because the output layer weights can be computed by a linear regression. The basis nodes of such a network can in this case be seen as regressors (Haykin, 1994) and will be called *regression nodes*. Although the term regression networks could possibly include any network minimising least squares, this terminology will also be adopted here.

An alternative name would be basis function networks, especially when the error term is not the least squares error measure. In this case, a regression is not employed to determine the output weights. In the regression networks of Van der Walt, the regression nodes could also have feedforward connections to one another. However, the networks he used were strictly feedforward. The regression networks used in this thesis will also be feedforward only.

The general structure of the regression networks used to approximate a function *f*, is therefore:

$$y(\boldsymbol{x}) = w_0 + \sum_i w_{i1}\, \Phi_i(\Phi_{k,k<i}, \boldsymbol{x}, \boldsymbol{p}_i) + \sum_j w_{jo}\, x_j \qquad \text{Equation 2.3}$$

where $\boldsymbol{p}_i$ are the parameters of the function Φ_i. Figure 2.2 shows the architecture of such a network.

2.3.1 Radial basis function networks

If in equation 2.3 $\Phi_i(\boldsymbol{x},\boldsymbol{p}_i)$ is a function of $|\boldsymbol{x}- \boldsymbol{p}_i|$, $|\bullet|$ being the Euclidian metric, such a network is called a radial basis function network. In this case $\boldsymbol{p}_i$ is a centre of the basis function Φ_i and will be given the symbol $\boldsymbol{c}_i$. Usually a Gaussian activation function of $|\boldsymbol{x}-\boldsymbol{c}|$ is used and the network only has a single hidden layer. Hence a radial basis function can be summarised into the following two equations:

$$y(\boldsymbol{x}) = \sum_i w_i\, \Phi_i(\boldsymbol{x},c_i) + \sum_j w_{jo}\, x_j \qquad \text{Equation 2.4(a)}$$

$$\Phi_i(\boldsymbol{x},c_i) = \exp(-|\boldsymbol{x}-\boldsymbol{c}|/2s_i)\text{ ,where } s_i \text{ is the width of the i}^{\text{th}} \text{ Gaussian.} \qquad \text{Equation 2.4(b)}$$

Gaussians having an ellipsoidal shape (Saha, 1991) are also possible and will be discussed later. The structure of a radial basis function network is given in figure 2.3. Figure 2.4 shows some typical activations of the hidden layer plotted against the inputs.

These kind of neural networks have been used for adaptive process control of nonlinear chemical reactor models. (Pottmann & Henson, 1997) In this study the local influence of radial basis function networks was used for efficient on-line adaptation. Related to radial basis function networks are neural networks with ellipsoidal basis functions. Kavuri and Venkatasubramanian (1993) used such networks in order to obtain bounded regions for fault detection. They illustrated the suitability of such networks on the diagnosis of a reactor distillation column system. In a study by Aldrich and van Deventer (1995), a radial basis function network accurately modelled the induced aeration rate in liquids and in agitated vessels for two types of impellers. The models were based upon the Froude Number, density and viscosity of the liquid, as well as on the geometry of the draft tube.

It is recognised that radial basis function networks have a functional equivalence to certain types of fuzzy logic systems. (Roger Jang & Sun, 1993) This means that every radial basis function network can be precisely translated into a fuzzy system. It also implies that training methods available for radial basis function networks can be used for certain fuzzy logic models and vice versa. This will be discussed in more detail in chapter 6.

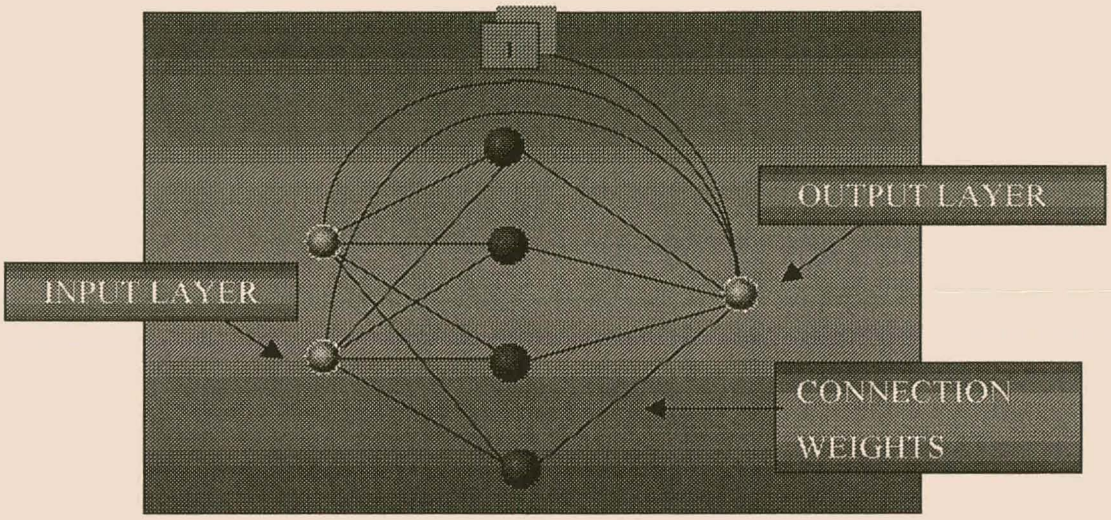

Figure 2.3 A diagrammatic representation of a radial basis function network.

2.3.2 Fuzzy systems as regression networks

Fuzzy systems (Zadeh, 1965, 1971; Wang, 1997) are knowledge-based or rule-based systems. The heart of a fuzzy system is a knowledge base consisting of (if... then) rules. Fuzzy logic is a commonly used technique, especially for control (Wang 1997). Its advantages are that the models can be non-linear and yet humans can comprehend them relatively easily. Fuzzy models were, for example, used during on-line control of a sequencing batch reactor in a sewage treatment plant (Cohen et al., 1997). The oxidation, nitrification and denitrification rate constants were estimated by fuzzy models and then used to assist control of batch reactor cycle duration. In another study by Juang and Lin (1998) *neuro-fuzzy* inference networks were used for modelling of time series data, as well as for the control of water bath temperature. Fuzzy rules have also been used in operational decision support by for example Wang et al. (1997). Their fuzzy neural network automatically generates fuzzy rules from process data. They illustrate and test their technique on a realistic simulation of a shell-and-tube heat exchanger.

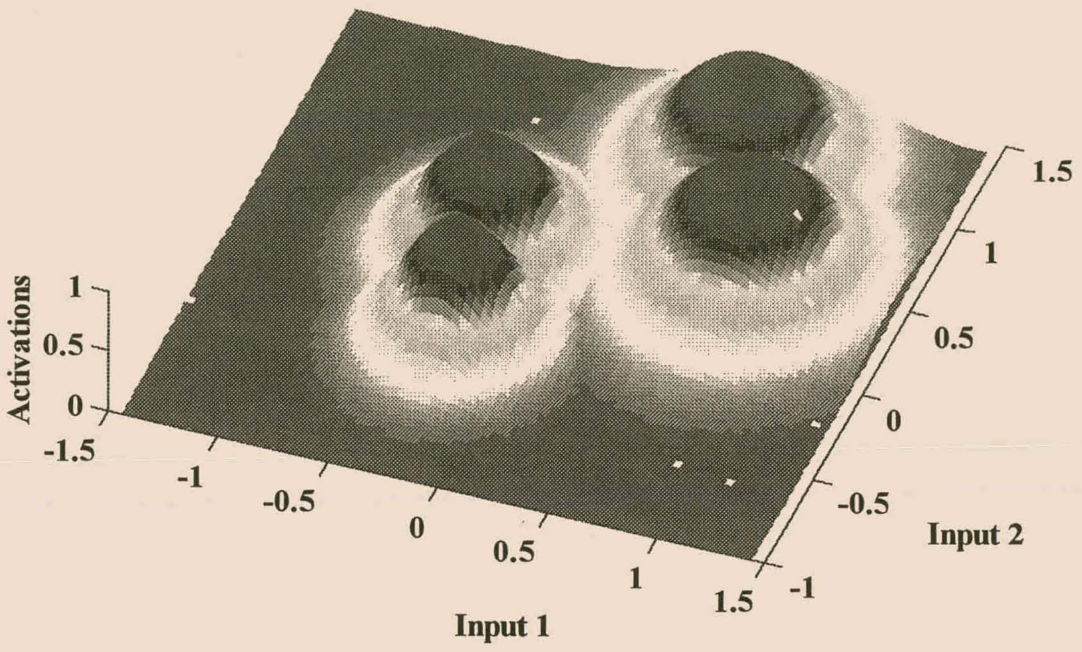

Figure 2.4 Activations of four radial basis function nodes plotted against the two inputs.

It will be shown below that the field of fuzzy logic and neural networks overlap. Certain fuzzy systems can also be seen as regression networks. The discussion of fuzzy logic will therefore focus on those systems that can be written as regression networks. This amounts to a further interpretation of regression networks.

2.3.2.1 Membership functions

In a crisp rule base system the rules are crisp and exclusive (i.e. only one rule can apply for a certain input region). In contrast to a crisp rule base the rules of a fuzzy rule base usually overlap considerably and a given input vector will usually trigger more than one rule. This is accomplished by defining for each input so-called membership functions. The membership functions are also called linguistic variables. For example, the membership *Cold* with respect to the input *temperature*, could be defined as a Gaussian function centred at -10 °C with a standard deviation of 10 °C, whereas the membership *slightly Cold* could be a similar Gaussian but centred at + 10 °C. Each membership function $\mu_m(x)$ takes on values in the interval [0,1]. The exact value depends on the input with which the fuzzy system is presented. In the example a temperature of 0 °C would result in both the temperature *Cold* and *slightly Cold* membership functions having values of 0.5. The process of converting the actual crisp input to the fuzzy membership values of each rule is called fuzzification.

Various membership functions are possible. Most of these can be found in Klir and Juan (1995) and Jang, et al. (1997). However, in this thesis only Gaussians will be used. The *Gaussian* MF is specified by two parameters c and σ. c determines the centre of the Gaussian and σ represents the Gaussians standard deviation or width. The formula for a Gaussian MF is:

$$gaussian(x; c, \sigma) = e^{-0.5\left(\frac{x-c}{\sigma}\right)^2} \qquad \text{Equation 2.5}$$

2.3.2.2 Fuzzy intersection

Fuzzy rules usually have several antecedents joined by an AND or an OR. Such as:

"If reactor temperature is *Hot* AND reactor pressure is *Medium*". The intersection needs to be defined. Usually the min or product operator is used for 'ANDING' and the max operator for '*ORING*'. Thus in the example above using the product 'ANDING' method, the membership values of Hot $_{\text{temperature}}$ and Medium $_{\text{pressure}}$ are multiplied in order to obtain the membership of the fuzzy intersection AND(Hot $_{\text{temperature}}$, Medium $_{\text{pressure}}$).

Symbolically for several antecedents, the combined intersection $\mu(\boldsymbol{x})$ under the product method is written as: $\mu(\boldsymbol{x}) = \Pi_j\mu_j(x_j)$. Figure 2.5 illustrates how the intersection of two Gaussian antecedents under the product method becomes a two-dimensional Gaussian bell function.

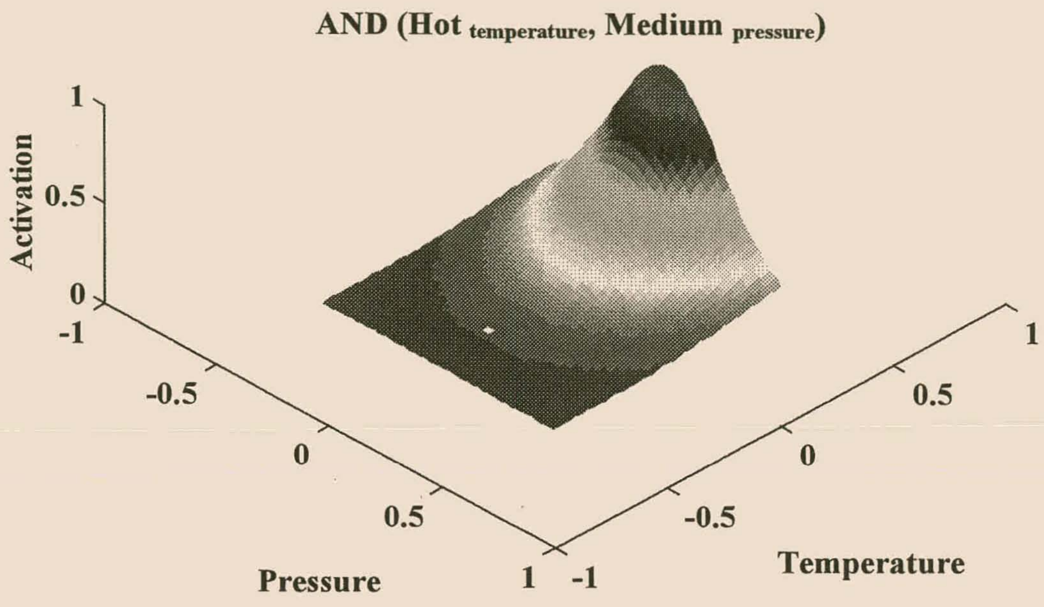

Figure 2.5 The intersection of "if temperature is hot" AND "pressure is medium" under the product method. The intersection of the two Gaussians results in a combined Gaussian activation function that increases gradually from 0 to 1.

2.3.2.3 Fuzzy rule format

As with a crisp rule, a fuzzy rule also consists of an antecedent part, or body, and a consequent. Fuzzy rules based on the Mandani fuzzy model (Mandani & Assilian 1975) have both the antecedent part and the consequent defined using linguistic variables. An example of a two-input and single-output Mandani fuzzy rule is "If reactor temperature is *Hot* and reactor pressure is *Medium* then the reaction is *Fast."* Here *Fast* is the linguistic consequent.

As with the Mandani fuzzy rule model, the Sugeno class of fuzzy rules (Takagi & Sugeno, 1985; Sugeno & Kang, 1988) has an antecedent part using linguistic variables. In contrast to the former in the Sugeno systems the consequent is some crisp function a(*x*) of the base variables used in the antecedent part of the rule.

If the function a(*x*) is a constant, the rule is called a zero-order Sugeno rule. An example of such a rule would be: "If reactor temperature is *Hot* and reactor pressure is *Medium* then the oxygen consumption reaction rate is 5.2 $m^3.h^{-1}$ " In this case the consequent is 5.2 $m^3.h^{-1}$.

If the function is a linear combination of the inputs, the rule is a 1st order Sugeno rule. An example of such a rule is: "If reactor temperature is *Hot* and reactor pressure is *Medium* then the oxygen consumption reaction rate is 0.5 $m^3.h^{-1}K^{-1}$* temperature + 0.5 $m^3.h^{-1}Pa^{-1}$* pressure + 2.2 $m^3.h^{-1}$"

2.3.2.4 Fuzzy rule system

It was mentioned above that the various fuzzy rules of one fuzzy system overlap considerably. If the various rules suggest different outcomes, how will the final outcome of such a model be decided? The process is called defuzzification. Typical defuzzification methods are the centroid method (Wang, 1997), the weighted mean or the weighted sum method. In the latter two cases the outcomes of the various rules are weighted by the weight (value) of the fuzzy membership functions ($\mu_i(\boldsymbol{x})$) of the fuzzy rules and are then averaged or added respectively.

A Sugeno system using the weighted sum defuzzifier can thus be written as:

$$y(\boldsymbol{x}) = \Sigma_i a_i(\boldsymbol{x})\mu_i(\boldsymbol{x}) \qquad \text{Equation 2.6}$$

Specifically for a zero order Sugeno system, the function $a_i(\boldsymbol{x})$ becomes a constant and the functional form is the same as that of a regression network. (equation 2.3). It can also be seen that radial basis function networks are zero order Sugeno systems using a weighted sum defuzzification and the product intersection rule (Roger Jang & Sun, 1993). In an equivalent fashion it is straightforward to show that a local linear map is a type of 1st order Sugeno system.

2.3.2.5 Fuzzy rule construction methods

Fuzzy rule construction methods for modelling purposes are very diverse. They include clustering methods, decision tree induction, genetic algorithm techniques, fuzzy neural network approaches as well as hybrid methods such as the CANFIS algorithm (Jang et al., 1997). This algorithm combines the CART decision tree algorithm (Breiman *et al.*, 1984) with ANFIS, a gradient decent algorithm, to optimise a fuzzy Sugeno system (Jang, 1993).

2.3.3 Other regression networks

Other architectures are regression networks in disguise. These include projection pursuit networks, the network of spline basis functions build by MARS and a multilayer perceptron neural network with a single hidden layer and a linear output node. All these architectures differ only with respect to the function types used in the hidden layer. The CERN algorithm presented in this thesis can be applied amongst other to such connectionist architectures as multilayer perceptrons, radial and ellipsoidal basis function networks as well as zero order Sugeno systems using weighted sum output determination. Projection pursuit regression is a special form of regression network in that the function is a smoothing function, which is data dependent and not pre-defined.

2.4 Generalisation of neural network models

Although the accuracy with which the model can describe the underlying data is salient, one of the most important criteria is to form models that can generalise well. In order to determine whether a neural network can generalise well, its error is measured on an unseen set of test data. This is important because it is possible to overtrain a neural network. This can occur when the network has been trained to respond too accurately and is uncompromising to the data in the training set. In doing so it no longer distinguishes between actual trends and noise in the data set. It is then possible that the overtrained network cannot generalise well onto unseen data. One can avoid this by using models with relatively few free parameters, i.e. models that are parsimonious in the total number of parameters that they optimise or that apply certain restrictions to the parameter values. A further technique is to enforce a smooth output surface using regularisation parameters (Girosi et al., 1993; Haykin, 1994; Girosi et al., 1995).

It is also desirable to generate connectionist networks that are relatively simple to understand. One such connectionist network architecture that lends itself to a relative easy interpretation, is the regression networks, especially in the form of the fuzzy Sugeno systems.

2.4.1 Testing and validating procedure

This section describes a procedure for training and testing neural networks. It can also be applied to other techniques. Michie et al. (1994) suggest the data set should be divided into three sets: one for training, a second for validating different networks and a third for testing the final chosen neural network. This procedure will also be followed in this thesis and is quickly explained in the following paragraph.

After a neural network has been sufficiently trained, its performance can be *validated* on 'unseen' exemplars. To do this one should set aside a certain portion of the training data, each time the network is trained. This data can be used to see how well the network has learned to generalise from the specific examples given in the training set to new exemplars not previously presented to them. If the performance on the validation data is not reliable, new network architectures, training algorithms or parameters thereof can be evaluated. When sufficiently good training parameters have been found, the neural network can ultimately be trained on all the training data and then *tested* on the testing set. The strict division of the data into one data set for training and validating and another set for testing is important when a lot of different networks and parameters are tried out. One can thus obtain a true measure of the generalisation ability of the network. Tuning the network settings on a test set biases the results on this test set.

A special methodology of verifying a neural network is called cross-validation. For k-fold cross validation, the data set is divided into k equal-sized subsets. The model is trained on the data of (k-1) of these subsets and is then validated on the remaining subset. This procedure is repeated k times until eventually each of the subsets is used as an exclusive validation set. Cross-validation can be applied for thoroughly validating the training parameters and also for testing the neural network afterwards. However, even when the method if used for testing, it is in referred to as cross-validation. Cross-validation minimises experimental bias in estimation of the accuracy. The statistic is reported in many publications to compare different models. However, it is not necessary when the data sets are very big. For typical accuracy rates (> 80%), 1000 examples for testing is extremely accurate compared to the true accuracy (Weiss & Kulikowski, 1991).

For time series data, cross validation is not applicable, because the sequence of the patterns is important. The testing data sequence should consist of data recorded some time after the training data sequence was observed. In some cases the aim is to analyse a real world data set as opposed to analysing the robustness of a method. In such cases, if k-fold cross-validation is used, the whole data analysis procedure, including selection of inputs, needs to be independently repeated k times. This will result in k models, possibly using a different selection of inputs and a different number of hidden nodes.

The selection of inputs to use is not a trivial task, or one for which a reliable algorithm exists. In addition to empirical experiments, it may also benefit from knowledge of the process. It can thus be very time consuming for k-fold cross-validation. It is also difficult for a human to strictly select the inputs for each of the validation sets *independently*, without using any information learned from the other validation sets. Furthermore, it is in certain cases not completely useful, since in the end only one model would be used to examine the properties of a plant.

2.4.2 Testing of the models on various real world data sets

In many papers the algorithms are examined on only one or two data sets. These data sets are also often very small in dimensionality. The Iris data set (Fisher, 1936) and the Mackey-Glass time series (Mackey & Glass, 1977) have four input dimensions and the Box-Jenkins gas furnace data (Box & Jenkins, 1970) usually has less than six inputs when embedded in time. These data sets not only have about the same dimensionality, are all relatively easy, (i.e. over 95% percent variance explained or correct classification), but they also have roughly the same number of data points. It is these three data sets that get examined over and over, especially in the context of fuzzy logic and radial basis function networks. Although this makes comparison to other algorithms possible, an algorithm should not be the tested on only such data sets. After many experiments, the statistical characteristics of the data become too well known. This makes it possible to construct algorithms that will do particular well only on these data sets. This need not happen consciously, but could also happen without intention. The algorithm designer will test their own and other algorithms only on these data sets. They might then adjust their algorithm, or draw conclusion about other algorithms based on results of only these data sets. Eventually the researcher could well have constructed an algorithm around these data sets. The resulting technique would probably perform well on these specific data sets but at the same time be inefficient in handling data with very different characteristics. Although everyone falls prey to such a methodology to some degree, it is especially difficult to draw conclusion about algorithms based only on result of one or two of these data sets.

The other problem encountered in many papers is that authors examine only artificial data sets. One can learn a lot about an algorithm from artificial data sets, but such data often has unrealistic properties. In many of these data sets there is either no noise present or the noise follows a perfect normal distribution. Prechelt (1996) examined more than 100 papers on neural networks and machine learning and found that in most papers the new algorithms are only examined on one or two synthetic data sets. They suggested that each algorithm should at least be tested on one real-world data set.

This thesis deals with a training algorithm. Therefore the training of the networks also needs to be compared to other algorithms. Also some of the training parameters require tuning. Consequently, it is useful to divide the data sets into 4 categories:

1. The data sets used for tuning the training parameters of CERN. In this category only the training set is used. The main interests lies not in the generalisation capabilities, but only in the learning rate of the algorithm under various circumstances. Also, it will be established which features of the algorithm cause faster learning and by how much. The learning speed will thus be compared to CERN with different settings and with various parts of the algorithm omitted. This analysis is done in Chapter 5.

2. The data sets for assessing the training speed compared to other algorithms. Here the learning rate will be compared to other better known techniques such as gradient descent. One could argue that although some algorithms train better, they might at the same time also overtrain more. Thus, for these data sets, also the performance of the networks on independent test sets will be shown. Such comparisons are made in Chapter 6.
3. In the third category the performance on an independent test set is assessed for numerous settings of the learning parameters, possibly including various network configurations. This is useful and allows the examination of the sensitivity of the algorithm to the various settings and comparisons to other algorithms. Models generated with CERN are analysed in Chapter 4 and 5 with data sets belonging to this category.
4. In Chapters 6 and 7 data sets are used to assess and compare CERN's generalisation on independent test sets to other well-know algorithms, such the backpropagation network, MARS and linear regression. In contrast to category 3, the aim here is to assess the performance of the algorithm rather than to show the influence of various parameters. Thus only one or a few network configurations should be used, on the test set. The tests do of course have to be performed independently. Results on the validation sets or other methods that do not use the test set can be used to fine-tune parameters to the specific problem. If the learning parameters are not set automatically, many benchmark data sets such as the Mackey-Glass time series do not fall into this category anymore, but should rather be seen as part of category 3. Through many studies, results on the test data set have been accumulated for a whole range of neural network configurations and this information can be used when deciding on the network complexity and the learning parameters.

Some restrictions do fall onto these categories:

The learning parameters were tuned using data sets from category 1. Thus data sets from category 1 should not be used afterwards in category 2, because the parameters could have been tuned to these specific problems. The data sets used for category 4 are used to assess the data set on previously unseen test patterns. Consequently, no point of the test set may have been used previously in either of category 1, 2 or 3. (However some of the training data can be used beforehand in other categories, e.g. parts of it for training and other parts if desired for validation.). Appendix A contains a description of each data set used in this dissertation. The Table A.1 of Appendix A is a summary of all the data sets. The table also shows in which categories a given set is used and gives a list of chapters in which the data sets were used.

2.4.3 Performance measures used

For classification problems the percentage of cases classified correctly is generally used. For data with a continuous output, various error measures exist. In this thesis the accuracies are given in terms of percentage of variance explained, or percentage of variance unexplained. The percentage of variance explained is also called the coefficient of determination, given the symbol R^2. It is defined by the formula:

$$R^2 = 1-\Sigma(y_p-y_t)^2/\Sigma(y_t-y_{avg})^2, \qquad \text{Equation 2.7}$$

where y_p is the predicted value of the outcome, y_t is the target value of the outcome and y_{avg} is the average target value of the outcomes (Myers, 1989). The percentage of unexplained variance is just $1-R^2$. It should be noted that the R^2 formula used here is not just the Pearson's correlation coefficient squared, but will always be lower or at best as high. In some literature it is stated that R^2 should not be used for non-linear models (Ratkowski, 1989). That applies if the Pearson's correlation coefficient is simply squared, as the symbol might suggest. In fact this should not even be used for all types of linear models. For linear or non-linear models, the formula $R^2 = R.R$ only applies on the training data when the output coefficients and the constant are determined by a regression on the training data.

Chapter 3 Optimisation of a neural network

Extensive surveys in the field of optimisation techniques for neural networks have already been published and therefore only the main features of the chief algorithms will be discussed here. See for example Haykin (1994; 1999), and Hertz et al. (1991), for details on non-evolutionary methods. Evolutionary methods are discussed by Whitley et al. (1990), Schaffer et al. (1992) and Yao (1993). A good summary of evolutionary and non-evolutionary methods is given by Shang and Wah (1996).

There are two fundamental ways of optimising a neural network. One is called unsupervised optimisation and the other is known as supervised optimisation. The more frequently used supervised learning method requires exemplars of the relationship which the network has to represent. This method is used to construct prediction, classification and time series models. It is referred to as supervised learning, for during training, the method is able to assess the relation between the predicted outputs of the network and the actual or desired outputs. It can then regulate the model accordingly. The actual outputs are therefore used to supervise the network. Unsupervised learning does not have identified solutions to train the model with, but instead constructs its own interpretation and verification of the data. Data are for example clustered by using unsupervised learning.

3.1 Unsupervised optimisation

Parts of certain networks discussed in the previous chapter can be trained in an unsupervised fashion. Consider for example the radial basis function network. Owing to the fact that the training of the connection weights (w) can be done by iterative or direct numerical solutions to the least squares problem, the most difficult task is finding a good set of basis functions. This is usually done in an unsupervised phase that clusters the input data for the selection of the centre co-ordinates (Moody & Darken, 1989). The widths of the basis functions are computed from the distances to the other cluster centres and the norm is taken to be the constant

Euclidean norm. Wettschereck and Dietterich (1992) found that using a supervised technique to train the hidden layer of a radial basis function network was superior to the above-mentioned unsupervised clustering technique. Nevertheless, owing to its efficiency and simplicity, training the hidden layer unsupervised is still very popular. It could also be used to initialise the parameters of the hidden layer of a radial basis function network, in order to train them in a supervised fashion later on. An improvement to clustering the inputs can be made by clustering the input data augmented with the output data (Saha & Keeler 1990).

Various fuzzy rule construction algorithms also use clustering methods to form the membership functions of the fuzzy rules (Wang & Mendel, 1992). Abe and Thawonmas (1997) formed elliptical clusters using a special clustering algorithm designed for fuzzy systems. After the clustering phase they fine-tuned the parameters of the fuzzy system with a gradient descent algorithm as described in more detail below.

3.2 Supervised optimisation techniques

The *linear* parameters or weights are those parameters, which have a linear effect on the output of the model. In a regression network they are weights that lead to the output node. These linear parameters can be trained by solving the linear least squares problem iteratively or by matrix inversion. For the optimisation of those parameters, which have a non-linear effect on the output (non-linear parameters), there is no algorithm, besides an exhaustive search, that can in general guarantee to find the optimal solution. In practice one is often satisfied with finding a near-optimal or a relatively good solution. In this case one can improve greatly on the exhaustive search technique or on a random search technique by using a non-linear optimisation algorithm.

Supervised techniques can be classified as either local or global techniques. Local minimisation methods use the information of their immediate surrounding to determine their next step. These algorithms, such as gradient descent, are usually fast, but are prone to converge quickly to a local minimum. In contrast, global optimisation techniques cover many different disjoint parts of the search. Consequently, they do not get stuck as easily in local minima, but also converge slower. Rapid increases in computer speed, available computer storage, as well as fast object orientated languages such as C++, Java and Oberon have made many existing and new optimisation techniques feasible for the training of neural networks. A thorough discussion of these is beyond the scope of this thesis. Various local and global optimisation techniques can be found in an article by Shang and Wah (1996). The basic techniques and their application to training artificial neural networks will be discussed below.

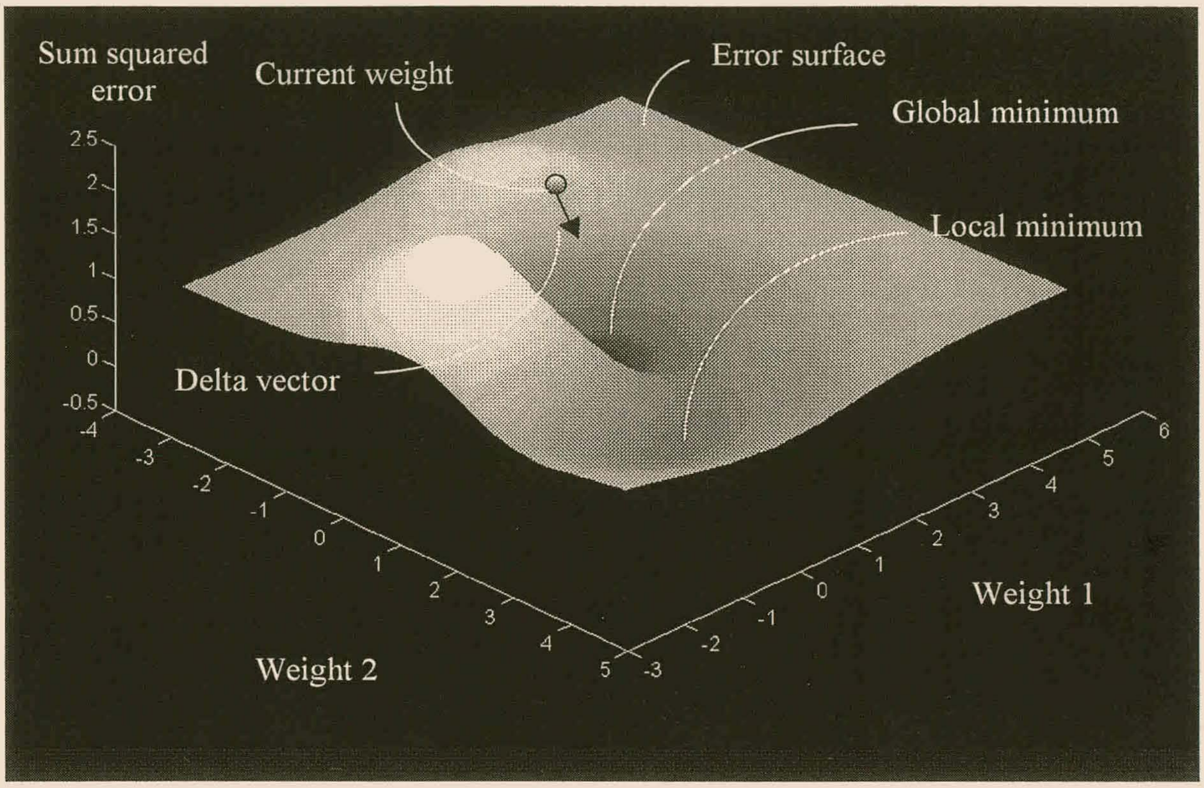

Figure 3.1 Illustration of the gradient descent technique by a ball rolling down a slope. The point of the ball touching the error surface always indicates the current weight vector.

3.2.1 Gradient descent algorithm

The most well known algorithm for feedforward neural networks is the delta learning rule of Werbos (1974). The delta learning rule begins by giving each connection weight a small random value. After this the input of training patterns are presented to the input layer and the desired target outputs to the output layer of the network. For each pattern the neural network computes the output in a feedforward manner. It then computes the difference of its output to the actual target value and makes a step change to its connection weights in order to reduce this difference. This step follows along the steepest vector of the error surface and is consequently also known as a gradient descent or steepest descent method. Intuitively it can be seen as letting a ball roll down a surface. Within the vector space of the adjustable weights, the rule attempts to move towards the global minimum. The error surface itself can take on many shapes and is seldom smooth. Indeed, in most problems, the solution space is quite irregular with numerous 'pits', 'hills' and 'plateaus', as is depicted in the figure 3.1. The pits and plateaus may cause the network to settle down in a local minimum. This is a serious drawback of gradient descent techniques. Consequently, researchers have developed heuristics to alleviate the problem. For example, more advanced learning rules such as the delta-bar-delta method (Jacobs 1988) and quickprop (Fahlman 1988) have built-in mechanisms to control the step size of the delta rule. For example, in the generalised delta rule (Haykin, 1994), a *momentum* term helps the network to overcome obstacles (local minima) in the error surface and to enhance convergence. This is sometimes called the heavy ball mechanism.

When the delta rule is applied to a multilayer perceptron, the error is propagated backwards (Rumelhart et al., 1986), in the same way that the input is propagated forwards, and the technique is called backpropagation of error or just backpropagation. The network is then also often referred to as a backpropagation network. The batch backpropagation algorithm (Rumelhart et al., 1986), with or without momentum, is the classic training algorithm for multi-layer perceptrons. Its success can be linked to its relative simplicity, as well as the small amount of computer memory required.

3.2.2 Second order optimisation techniques

The delta rule is essentially a first order or steepest descent algorithm. Besides the heuristics that can be added to make it more efficient, second order methods can speed-up training considerably. These include the conjugate gradient algorithm devised by Fletcher and Reeves (1964). Others are discussed in Battiti (1992) and Van der Smagt (1994). These methods are classed as second order methods, because they exploit the second order derivative of the information of the error surface to generate the search direction. The algorithms can either be of matrix or non-matrix (vector) type. The former are typically two orders of magnitude faster than the plain steepest descent algorithm, but require the storage of the Hessian matrix (McLoone et al., 1998). The vector method requires less storage but is only about one order of magnitude faster.

Derivative-based algorithms are able to optimise a number of network parameters, such as the connection weights of the network, but are unable to handle the optimisation of parameters for which there is no gradient information. This problem can be overcome with optimisation methods that do not depend on gradient information, such as stochastic search techniques.

3.2.3 Genetic algorithms

Stochastic search methods such as tabu search (Battiti, R. & Tecchiolli, G. 1994), simulated annealing (Kirkpatrick et al., 1983; Otten & van Gineken, 1989) and genetic algorithms (GA's) (Goldberg, 1989; Holland, 1975) offer an alternative to first- and second order derivative-based training algorithms. These methods, which are derivative free, are characterised by their stochastic search elements, which allow the algorithm to escape from local minima. Genetic algorithms and simulated annealing can be seen as special cases of population-based optimisation algorithms. Genetic algorithms are based on the concepts of natural selection and evolutionary processes.

Their main drawback is that, like other global optimisation techniques, they converge very slowly. However they do have the following advantages:

- They are parallel search procedures that can be implemented easily on parallel processing machines for speeding up the algorithm by orders of magnitude.
- GA's are applicable to both continuous and discrete optimisation problems.
- Owing to their stochastic character, they are less likely to get stuck in local minima.
- They can be used for both structure and parameter tuning in a wide range of models, such as fuzzy systems, controllers and neural networks.

A genetic algorithm can thus in principle be used to minimise any objective function. In the case of a neural network, this would usually be the sum of the squared errors for a given data set. The parameters to be optimised would be the weights of the neural network, although many other features of the neural network can also be identified with the help of a genetic algorithm.

The algorithm starts with a population of randomly configured parameters. For a multilayer perceptron this would be the weight matrices of all the layers. This is the phenotype representation. Each member of the population is encoded into a string or chromosome, also referred to as the phenotype representation. The chromosome can be of a binary type or it can consist of real numbers. At each generation certain individuals of the population are selected to produce offspring for the next generation. Analogous to survival of the fittest in natural selection individuals, those that are 'fit' have the highest chance to be selected. The genetic algorithm needs to assign a fitness score to each member of the population. This is usually the output of the objective function evaluated for the specific phenotype of a member (e.g. the sum of the squared errors of the particular neural network configuration). The process of selecting the fittest individuals, recombining their genes and evaluating the objective function of the new individuals is repeated through a user-defined number of generations. The following genetic operators are used in the recombination of the individuals:

- *Crossover*: To exploit the potential of the current gene pool, the crossover operator generates a new individual by combining the chromosomes of two or more existing individuals. Various schemes of crossover exist. (Goldberg, 1989).
- *Mutation*: This operator randomly changes *bits* of the chromosomes and thereby introduces new variations in the gene pool and helps individuals to escape from local minima.
- *Selection*: The individuals get chosen probabilistically in proportion to their fitness scores. Tournament selection is a special selection method in which the individuals compete in pairs (for binary tournament selection) or larger groups. Of two randomly chosen individuals, the one with the higher fitness score is allowed to recombine its chromosomes.

CERN is an evolutionary algorithm that uses a combinatorial search in the selection scheme. It is thus in the selection scheme that the CERN algorithm of this thesis differs fundamentally from other evolutionary approaches. Variations of the genetic algorithm and other algorithms, which are tailored more specifically to the task of optimising neural networks will be discussed in the next section. As was shown in chapter 2, neural network techniques overlap with non-parametric statistical modelling and fuzzy logic rule systems. Hence some construction algorithms for these kinds of models will also be discussed.

3.2.4 Combinations of optimisation techniques

The methods discussed in the previous section can be seen as the basic training techniques for optimising a neural network. There are many more specific optimisation algorithms, but most of them can be grouped into either the local search technqiues viz 1st or 2nd order derivative based learning or the global optimisation algorithms such as evolutionary algorithms and simulated annealing. Some researchers have adapted methods such as genetic algorithms to the specific task of training neural networks. The training time could thereby be reduced significantly. For specific types of neural networks, such as the radial basis function networks or recurrent networks, it is possible to use further features of the specific network to be trained. This can make the algorithm more efficient and/or result in smaller models that can generalise more accurately.

Many other techniques can be seen as a combination of several such techniques. For example, the EPNeT algorithm (Yao & Liu, 1997) is an evolutionary algorithm that uses simulated annealing, as well as gradient descent. The reason for using gradient descent in the EPNeT algorithm is to accelerate learning. Indeed, for large neural networks, evolutionary methods and also other techniques can take very long to find good or near optimal solutions. The reasons for this, as well as possible solutions to the problem, will be discussed next.

3.3 Considerations regarding the solution space

3.3.1 Moving target problem

An important reason why simultaneous optimisation of the parameters of the entire network can fail, is that the solution space to be searched becomes too large and consequently the nodes become involved in a *moving target problem* (Fahlman & Lebiere, 1990), where each hidden node sees a constantly moving environment. Furthermore, in techniques with a fixed number of neural nodes, the choice of the total number of nodes initially specified in the network architecture is critical. If too few nodes are used, the network will underfit the data and if too many nodes are used, the network is easily over-trained. For example, approaches where neural nodes are added dynamically to the network are able to solve the Double-Spiral problem (Appendix A) with ease and with few training epochs. Examples are the cascade correlation algorithm (Fahlman and Lebiere, 1990) which uses a hybrid of gradient descent and a genetic algorithm for optimisation, and the growing cell structure (Fritzke, 1994). In contrast, other methods that simultaneously optimise the network, for example by gradient descent techniques, sometimes fail to solve the Double-Spiral problem and others require specialised architectures or learning methods, as well as many more training epochs in order to be successful (Fahlman & Lebiere, 1990).

3.3.2 Representational redundancy and recombination

If used to optimise more than one node at a time, all the basic optimisation approaches discussed in the previous chapter have the problem of permutational redundancy (Radcliffe 1990; 1993; Belew et. al 1990). This is caused by the arbitary labelling of topologically equivalent neurons. For example, in a simple neural network with h units in the hidden layer there is a potential redundancy of a factor $h!$ associated with the fact that identical neural networks with relabelled units cannot be distinguished from one another. A simple example will clarify this: Whether one calls a certain node in the hidden layer "node 3" or "node 4" will not result in a different network. However, a simple representation that encodes each node and places the encoded nodes one after another in a string already makes a distinction among the two neural networks. By distinguishing between what are actually identical networks, the encoding enlarges the search space enormously.

While the number of optima usually also increases by a comparable factor, the global nature of the search tends to make the navigation through the search spaces very difficult (Radcliffe, 1990; 1993; Belew et al., 1990). Simulated annealing and genetic algorithms are sensitive to the potential of redundant representation in a way that most gradient based approaches are not. The reasons for this are that the local techniques tend to make relatively small moves in the search space and they usually only maintain a single solution rather than a population of solutions. It can also be seen that the tabu search for example, that labels certain regions of the spaces it has evaluated previously as 'tabu', might also examine the same areas of the search space repeatedly under a redundant encoding. Such an encoding is especially detrimental to evolutionary techniques (Radcliffe, 1990; 1991; 1993; Belew et al., 1990). This is because two very similar neural networks can be combined to generate offsprings dissimilar to either parent. In fact, it might very possibly inherit the same node twice in the same layer. A node that is identical or nearly identical to another node in the same hidden layer is usually completely useless. It will do nothing but be a nuisance and cause numerical instability in case the output layer is trained by a direct solution to the least mean squares problem.

At the same time, consider what happens when both of the individuals hold a beneficial node at the same locations in their genes. The clash will ensure that no matter how many times these two individuals recombine, they will never produce an individual that will have inherited both of the beneficial nodes. Radcliffe (1993) presented a solution to this problem for node connectivity, but his solution does not guarantee that the two beneficial nodes will be inherited. It only makes it possible. The CERN algorithm proposed in Chapter 4 solves the problem of permutational redundancy and guarantees that two beneficial nodes belonging to different individuals will always be chosen. That is if they can be used to construct a better neural network in terms of minimising the error. It is also possible to use a genetic algorithm where each node is encoded as a separate individual. This will cause different problems that require solutions, as described below.

3.3.3 The niche and credit apportionment problem

It is possible to focus the search for a solution by letting each individual of the genetic algorithm population represent only a single node of the network. The population can then either consist of only one neural network (Odri et al., 1993; Whitehead & Choate, 1996) or of several networks (Smalz & Conrad, 1994). The performance of the whole network is maximised by selecting a good set of individuals. The genetic algorithm therefore requires a formula to assign a fitness value to each individual of the neural network. This is known in the evolutionary algorithm literature as the *credit apportionment problem* (Holland, 1975). The other problem that needs to be solved is that different nodes must evolve to do different tasks, also know as the *niche problem* (De Jong, 1975; Deb & Goldberg, 1989). This can be understood intuitively, either directly in a radial basis function network where the different nodes need to cover different regions of the input space, or indirectly in a single hidden layer perceptron where different nodes need to combine to explain the total variation of the output.

3.4 Approaches of training algorithms to manage the solution space

Especially for algorithms that use a global search, for example genetic algorithms, the search space needs to be reduced or covered more rapidly in order to make the method feasible for training neural networks. For this reason and also because CERN is an evolutionary approach, the following section will focus specifically on algorithms using evolutionary principles.

If the output layer is linear, the solution space can be reduced by solving the least squares problem directly, because the output nodes weights need not be optimised anymore. This can be used for both gradient based searches and for population based methods. Along a similar vein, an evolutionary method can traverse great parts of the solution space by making use of gradient descent. Another way to keep the solution space small is to use a representation that is not redundant with respect to permutations of the labels of the nodes or otherwise. A combination of these methods is also possible.

A different approach is to divide the problem up and reduce the number of nodes that are simultaneously optimised. This can be done effectively by, for example, only optimising and adding one node or groups of nodes at a time to the network. A disadvantage is that for some problems the nodes optimised initially can commit or get stuck in sub-optimal solutions and the best solutions can no longer be found. This method will therefore not only affect the time to find good solutions, but will usually also change the types of solutions that are found. This suggests that algorithms can be divided according to their basic strategy of building the neural network model. On the one end of the continuum the entire network structure and attendant parameters are optimised simultaneously. On the other end, methods involve the sequential addition and optimisation of new nodes to the existing network.

Below various algorithms will be grouped by this basic building strategy. If an algorithm uses one of the other methods to reduce the solution spaces or to speedup the search, it will be indicated where applicable. Among these methods, CERN's strategy to reduce the solutions will be motivated.

3.5 Simultaneous optimisation of all network parameters

The backpropagation and related first- or second-order gradient descent algorithms are usually employed to optimise all the parameters of a network simultaneously. When used in conjunction with a momentum term, these algorithms are usually sufficiently fast to optimise networks for medium-sized data sets. However, the algorithms can still get stuck in local minima or on a plateau where the gradients are near zero. For example, they struggled to optimise the multilayer perceptron on the very non-linear Double-Spiral data, as was discussed in section 3.3. Furthermore, many of these algorithms perform poorly on training radial basis function networks, (McLoone et al., 1998) as these are especially ill-conditioned (McLoone, 1996).

A promising alternative is to combine gradient-based optimisation of the weights that have a non-linear effect on the output (non-linear weights) and singular value decomposition for the linear weights in one integrated routine (McLoone et. al, 1998). This method is especially effective for training networks with many linear weights relative to the non-linear weights, such as occurs in local linear maps. The algorithm was also more efficient for training radial basis function networks than pure gradient based techniques, even for those that are of second order.

3.5.1 Simultaneous evolution of the entire network

In evolutionary strategies aimed at optimisation of networks in total, each individual in the population represents a complete neural network. One approach that has been investigated (Schiffman et al., 1991; Bornhold & Graudenz, 1992; Hancock, 1992; Maniezzo, 1994) is to let individuals encode only architectural specifications of the network, i.e. in general any parameters that can describe the neural network, excluding the values of the connection weights. The weights are then trained with other methods, such as gradient descent. The approach is inherently computationally demanding because the complete training of the weights is required simply to evaluate the fitness of a single chromosome. Clearly the weight training algorithm still has to solve the problem of local optima. Studies in this class include those of Whitley et al. (1990), Harp et al. (1989) and Hancock (1990).

Another approach is to let the individuals represent the architecture, as well as the weights of the neural network (Kitano, 1990; Fogel, 1993; Angeline et al., 1994). In this case the solution space that must be searched by the optimisation algorithm becomes extremely large, but can be reduced slightly by, for example, not optimising the values of the final connections to the output nodes. As was mentioned earlier, these could be trained afterwards by a least mean square algorithm (McDonnel & Waagen, 1993; Whithead & Choate, 1994; 1996; Billings & Zheng, 1995). Algorithms where the structure, as well as the weights of the network are evolved usually involve evolutionary programming (Koza, 1992). In the context of neural networks, evolutionary programming techniques differ from genetic algorithms, in that they use tree-structured variable length individuals, instead of fixed-length chromosomes, to represent the neural network architecture and connection weights. The evolutionary operators, such as mutation, crossover, parameter training and pruning, work directly on these tree structures. Yao and Liu (1997) argue that the evolutionary programming approach is better than using a genetic algorithm for evolving neural networks, as a stronger behavioural link can be maintained by evolutionary programming between parents and offsprings. This link can be used to improve the network evolution efficiency as in Yao and Liu's algorithm, EPNet. Unlike other evolutionary programming algorithms (Fogel et al., 1990; Fogel, 1993; Angeline et al., 1994), EPNet partially trains the weights after each architectural mutation, starting from the parent's weights, instead of randomly initialised weights, as one of its techniques to better maintain the above-mentioned behavioural link. It also uses simulated annealing, deletion and addition of connections, as well as node addition as further adaptation steps. Node deletion is preferred to node addition to encourage parsimony. For a series of problems this algorithm was able to find neural network architectures that were very parsimonious with respect to the total number of connections. It remains unclear from the article, which adaptation steps of the algorithms result in the biggest improvements of the trained networks.

Another algorithm that optimises a fuzzy system simultaneously is FuGeNeSys, proposed by by Russo (1998). It can be used for the construction of zero order Sugeno fuzzy systems. Further constraints included the specifications that the antecedent part of each rule contains at most one condition for each attribute and that the membership functions are one-dimensional Gaussians. The structure is therefore similar to that of an ellipsoidal basis function network. FuGeNeSys uses a genetic algorithm that makes use of subpopulations or so-called demes. During the evolution the individuals are selected by a fitness-proportional elitist scheme. A further hill-climbing operator is used by FuGeNeSys for quicker parameter optimisation. This operator is activated during the evolution process whenever an individual is generated with a fitness value greater than the best one obtained so far. The objective function used by FuGeNeSys to evaluate the fitness of an individual is a heuristic formula that takes into account the prediction error of the current individual, the ratio between the number of antecedents and consequents, the significance of each fuzzy rule and a value proportional to the number of attributes used. Russo (1998) compares his algorithm with a number of other techniques on two modelling problems, as well as on a control problem. He also gives references to six other successful applications of the algorithm.

3.5.2 Simultaneous evolution of all the nodes where each node is encoded as an individual

Recall that evolutionary algorithms that encode each neural node as an individual, required a method to assign credit to the individual nodes. The credit assignment or sharing process used in an algorithm of Smalz and Conrad (1994) is fairly complex: The population consists of networks, identically structured, but having different parameters. The neurons are grouped in classes, that "*compete for their right to copy - with variation - their parameters to other neurons in the class.*" In each neuron class and for each input the similarity of the neurons to that neuron belonging to the best network for that specific input pattern is determined. This similarity is measured in how much the activations of the two neurons being compared are alike. The idea is to identify those neurons, which "*are most compatible with all of the networks that perform the best with respect to a specific external input*". The algorithm performs better than a standard genetic algorithm and can also be used on recurrent neural networks. However, it was tested only on artificial data sets. The credit of a neuron depends on the activations of neurons from a specific network, which is selected anew for every data pattern. This is a very interesting and original procedure and it is difficult to compare it to existing techniques or to give further comments on it.

Whitehead and Choate (1996) proposed an algorithm that awards credit to each node in proportion to the node's contribution to the overall output of the network. The niche problem is solved by varying the intensity of the competition among the nodes, based on the degree of overlap of the radial basis functions. This enables the evolution of so-called *cooperative-competitive* nodes. Although the niche problem is solved, to apportion credit to non-orthogonal basis functions is a complex task. One cannot simply use the size of outgoing weights or the degree of correlation of a given basis function correlated with the output. Whitehead and Choate (1996) used a formula that can only approximate the actual contribution of a given basis node. After this paper, Whitehead (1996) tried a different credit assignment formula. However, in this algorithm it is wrongfully assumed that the orthogonal least squares (OLS) algorithm will be able to pick the best *m* nodes of a larger set of nodes: "*Given a finite set* $\underline{\Phi} \subset \Psi$ *of basis functions sufficient but larger than necessary to approximate f to the desired accuracy* (e.g. , a radial basis function centered on each training example) *OLS rank ordering of* $\underline{\Phi}$ *selects* $\Phi \subset \underline{\Phi}$ *of size m which maximises* $||g_{\Phi}||^2$ "* Unfortunately this is not the case and is sometimes stated incorrectly, e.g. in Liu et al. (1998). To support his statement, Whitehead also referred to a paper by Chen et al. (1991), which correctly states that the OLS algorithm is much more efficient than a random selection, but they do not state more. A reason why the OLS algorithm does not necessarily select the best subset is given in the next section. See also a paper by Sherstinsky and Picard (1996) that addresses the efficiency of the OLS algorithm.

* (In the above, $||g_{\Phi}||^2$ is the squared length of the projection of *f* onto the space spanned by Φ, i.e. the R^2 statistic or the proportion of variance explained).

3.6 Evolving one node at a time

3.6.1 Adding one node while keeping remaining nodes fixed

As mentioned in section 3.3, a number of algorithms have been proposed which attempt to reduce the error of a neural network by sequentially adding nodes, one node at a time. These algorithms belong to the larger class of constructive algorithms. A comparison of constructive algorithms can be found in Fiesler (1994) and in Beiu (1996). The best known algorithms are the cascade correlation algorithm mentioned previously, a variant of the cascade correlation algorithm (Littmann & Ritter, 1992) and the orthogonal least squares algorithm (Chen et al., 1991). The difference between the cascade correlation algorithm and its variants is that the former successively adds a node that maximises the correlation with the error, and the latter adds a new node to reduce the error at each node-addition step. The OLS algorithm is used to select basis vectors for radial basis function networks. It also adds a single node to the model at each particular stage of the search. The co-ordinates of one of the remaining subset of data points are chosen as the next basis node's centre. At each step the OLS algorithm selects the basis function that will maximise the increment in the explained variance of the output. Small and Judd's algorithm (1998) adds a new basis function at each step that best fits the residual. However, they consider many more parameters such as the width of the basis functions, possible cylindrical basis functions, as well as a parameter determining the shape of the transfer function. The minimum description length principle (Rissanen, 1978; 1989) is used as a stopping criterion to prevent overfitting. The projection pursuit algorithm summarised in Section 1.3.3 also counts as an algorithm that adds one node at a time to a network.

An advantage of dynamically allocating new nodes is that it is possible to halt optimisation when a certain level of accuracy has been reached. A more sophisticated method would stop when the minimum description length - MDL (Rissanen, 1978; Leonardis & Bischof, 1998) or prediction sum of squares - PRESS statistic (Myers 1989; Schaal & Atkeson, 1998) flattens out or starts increasing. (The PRESS statistic is simply the sum of squares error computed on a n-fold cross validation, where n is the number of data points.) This is not so simple when the whole network is optimised concurrently. Moreover, if the network's size were evolved dynamically, it would be possible to remove the nodes one at a time from the final network, starting at the node which was added last and moving back to the one that was added first. One may therefore determine how much of the total variance of the output function is explained by the remaining nodes of the network. This implies that pruning the network to obtain either a simpler model or better generalisation capabilities may be accomplished by simply removing the nodes one at a time in the reverse order in which they were originally added to the network. This would continue until a certain statistic is minimised, such as the minimum description length (Rissanen, 1978). If the whole network were optimised by the algorithm in one step, it would become difficult to prune (Haykin, 1994), since the nodes would not be ranked in any order of importance.

One problem with these algorithms is that it cannot be guaranteed that successively maximising the explained variance by choosing or optimising one unit at a time will lead to the best possible solution after several such steps. This issue is discussed in detail by Sherstinsky and Picard, (1996) and they show that the OLS algorithm is not optimal in the selection of significant basis functions.

3.6.2 Adding nodes one at a time while keeping remaining nodes pliable

Certain incremental constructive algorithms overcome this problem, as they sequentially add receptive (or local) units, but at the same time still allow the remaining nodes to adapt. The previously placed nodes will usually only adapt gradually, otherwise this scheme would not be any different from learning all parameters simultaneously.

The backfitting algorithm (Hastie & Tibshirani, 1990) will add one node at a time, while keeping the previous nodes fixed. It then adjusts the parameters of one previous node. This node is fine-tuned, while the remaining nodes are fixed. The procedure cycles in this fashion through all the nodes several times, including the new one, until the parameters converge. This algorithm can be used to tune the parameters for any problem, where the parameters can be broken up into different subsections. Of course, if the parameters of different nodes interact, as is almost always the case, optimising them separately can prevent the algorithm from finding the best global fit.

Another algorithm that belongs to this class is the resource allocation algorithm, RAN (Platt, 1991). It appends new radial basis nodes to the network according to two criteria. The error of the training sample has to be large and no RBF node has to be activated by more than a certain threshold value. The width of the new node is chosen to be the initial distance to the neighbouring RBF nodes. After this the centres and outgoing weights are optimised by a second order gradient descent algorithm. Thus RAN inserts a new node based on a single poorly mapped pattern. This can lead to a huge number of basis nodes (Fritzke, 1994).

The growing cell structure (GCS) algorithm (Fritzke, 1994) dynamically adds to and removes units from a connected structure of basis nodes. When a certain resource, such as the sum of the squared errors, accumulates at a receptive field (or basis node), the basis node splits and a new node is created. The algorithm can also be used to build local linear maps (Fritzke 1995). Brüske and Sommer (1995) combined Fritzke's ideas with the slightly superior growing neural gas algorithm (Martinetz and Schulten, 1994) to accomplish a more flexible topographic structure than the original GCS algorithm. Unfortunately all these algorithms were designed with spherical basis nodes or receptive fields in mind. High-dimensional data or data with superfluous attributes could side-track them very easily.

A more recent algorithm (Schaal & Atkeson 1998), receptive field weighted regression (RFWR), overcomes this problem by adding ellipsoidal receptive fields of local linear models. The algorithm incrementally minimises the locally weighted leave-one-out cross validation error by using a gradient descent technique on the parameters. They do this by minimising the formula used for calculating the PRESS statistic for a linear model, that is the prediction sum of squares for regression (Myers, 1990). In doing this they treat the hidden nodes as the regressors. A new node is added whenever a new training sample does not activate any existing receptive field by more than a threshold. Similar to the growing cell structure and the growing neural gas, the RFWR algorithm is capable of adjusting the parameters as the input and output distribution changes over time. RFWR's error function, the leave one-out-cross-validation error reduces over-fitting, but does not remove it. This might seem like a paradox and can be clarified as follows:

One could argue whether minimising the PRESS formula for linear regression is strictly the same as minimising the leave-one-out cross-validation error, as Schaal and Atkeson (1998) call it. For a leave-one-out cross-validation error, the test data point must be totally removed from the data set. The model is then formed on the other data points and tested on the validation or test point. Accordingly, there is a different model for each test point. Given some *fixed* regressors (independent variables), the formula for the PRESS error is in fact equal to the leave-one-out cross-validation sum of squared error computed explicitly (Myers, 1990). However, in the RFWR algorithm or in any regression network the regressors are adjusted systematically using all the training data points to minimise some error criterion. Let us make this error function the leave-one-out cross-validation error. Then for each validation set a data point should be left out completely and only be used for validation after all iterations are finished. In the algorithm it is likely to affect the parameters of the regressors from one iteration to the next. The validation or test point thus still participates in the training routine and is not really left out. This argument asserts that one cannot minimise the leave one-out-cross-validation error directly, but only compute it. In this thesis the leave one-out-cross validation error is also computed. It will be used to select input variables for linear models and also in order to help assess at what point to cease addition of new nodes to a neural network. The results, discussed later, show that for non-linear models such an error function must be used with caution.

It would be crucial to study whether the RFWR (like the RAN algorithm's criteria for adding new receptive fields) would not add too many receptive fields for high-dimensional data, especially when there is noise present. Therefore Vijayakumar and Schaal (1997) incorporated local dimensionality reduction as a preprocessing step in every receptive field. The criteria for adding new nodes in the GNG and the GCS algorithms, as well as Small and Judd's approach (1998) are possibly more reliable, because they do not add a new basis node for every single distant data point.

3.7 Evolving groups of nodes at a time

A greedy search method involves the addition of only one node at a time to a neural network. At each step the newly-added node is optimised to explain as much of the remaining output variance as possible, without considering the subsequent addition of nodes. The chance of finding sub-optimal solutions can be decreased by adding groups of nodes, instead of just a single node (Mézard & Nadal, 1989; Saha et al., 1991; Lange et al., 1994). Saha et al.'s network of *orientated nonradial-basis functions* (ONRBF) starts with a few widely tuned receptive nodes and repeatedly collects the points where a high error is recorded. New orientated ellipsoidal nodes are then placed in locations chosen randomly from this set. The algorithm subsequently optimises the parameters of all the nodes in the network with a gradient descent algorithm. The ONRBF nodes perform notably better than RBF nodes. Unfortunately the paper on the algorithm is very brief and the experimental section only includes a two- and a four-dimensional problem, both artificial, without the presence of noise.

The *tiling algorithm* of Mézard and Nadal (1989) and the *task decomposition by correlation measures* (TACOMA) of Lange et al. (1994) both sequentially add new layers of nodes to the existing network. In the tiling algorithm the number of nodes of each successive layer is always decreasing. TACOMA on the other hand uses a mapping procedure, which gives the maxima of the mapping of the output unit's residual error to the input space. The number of units placed in the new hidden layer by the algorithm is the number of maxima, which are above a certain threshold. Every time a new layer is added, its nodes are optimised using correlation measures. Algorithms such as these, that add more than one node to the network at each step, are effectively looking a number of steps ahead in the search for the optimal solution. Given a solution space that can be thoroughly searched and a certain error function, an algorithm which adds groups of nodes should always find a solution that is at least as good as one obtained by an algorithm that adds nodes one at a time. An algorithm that adds only one node at a time will not easily find a solution that requires a difference between two nodes for example. This is similar to a stepwise forward linear regression method that could overlook those models, to which the difference of two variables contributes, but where each of the two variables on their own do not make a significant contribution.

It therefore stands to reason that the more nodes that are added and optimised at a given stage, the better the solution is that can be found, but at the same time a larger part of the solution space has to be searched. Therefore an algorithm that adds groups of nodes can achieve a compromise between the two extremes of sequentially adding single nodes to the network and optimising a fixed size network. Such a method will avoid the moving target problem and at the same time will not be as easily diverted by structural local optima caused by adding nodes sequentially. This is the approach taken by the algorithm proposed in this thesis, which uses an evolutionary algorithm for optimisation. Similar to the techniques described in section 2.1, the solution space can be decomposed further by using groups of competing nodes. It is then possible to use an encoding scheme where an individual represents only one node. This representational format is employed by the CERN algorithm, described in the following chapter. Instead of looking for a set of parameters that describes the whole group of nodes, the algorithm searches for combinations of individuals, each of which describes a single node only. In the algorithm the nodes are bundled into groups of nodes. It will be shown that in such a group the ordering of the nodes is irrelevant (hence the group is actually a set). Thus it will not lead to the problem of permutational redundancy associated with the arbitrary ordering of the nodes. The CERN algorithm also guarantees that when two such sets of nodes are recombined to form a new set of the same size, the set resulting in the lowest RMS error will always be found. The algorithm adds new nodes until some stopping criterion is met.

Chapter 4 The CERN algorithm

In this chapter the concept of a combinatorial selection scheme as well as the other steps of the CERN algorithm (Schmitz & Aldrich, 1999) will be explained in more detail. An algorithm that constructs a neural network by the stepwise addition of groups of nodes will be described first. Next an outline of the differential evolution (DE) algorithm will be given. It will be explained how it could be used as part of the technique that constructs the neural network. However, the differential evolution algorithm also has some consequential shortcomings for this task and the proposed solution of these problems leads to the CERN algorithm.

CERN's training curve will be compared to the training curve of the DE algorithm in some controlled experiments. The data sets examined are the so-called Double-Spiral classification problem, the Mackey-Glass time series (Mackey & Glass, 1977) as well as a data set on the prediction of the age of Abalone (Nash, et al., 1994) from their physical characteristics. Further details on these data sets are provided in Appendix A. The test results on the data sets will also be compared with those obtained with standard algorithms. These include radial basis function nodes optimised by the k-means clustering algorithm and multilayer perceptrons optimised with gradient descent techniques. In this chapter CERN evolved networks of orientated ellipsoidal basis nodes as well as multilayer perceptrons. In chapter 5 the algorithm will be extended that will among other allow the simultaneous evolution of nodes of different types (e.g. RBF nodes and MLP nodes) in the same hidden layer.

4.1 The steps of the CERN algorithm

4.1.1 Node allocation

The proposed algorithm for dynamically adding groups of nodes will be described for a single output only. This approach is not necessarily restrictive, since multiple output systems can be accommodated by a separate neural network for each output. The method of evolving the network used in this algorithm is similar to the one employed by Littmann and Ritter (1992), except that groups of nodes are added at each step. This concept is illustrated in Figure 4.1.

As indicated in figure 4.1, all new nodes can, but need not be, connected to all the existing nodes of the previous hidden layers and of the input layer and are always directly connected to the output node. The input layer can optionally be connected directly to the output layer. In addition to this, there is a bias connected to the output node that is permanently set to one. The cost function that is minimised at each step of the algorithm is the sum of the squared residuals remaining after the addition of the new nodes. The weights of the output node remain flexible. The existing network nodes of the hidden layers are left unchanged during the search for the best new group of nodes to add to the network. One can therefore see that at a given stage of network evolution the nodes that were added first will usually represent or explain the global information of the actual mapping between the inputs and output of the training data. The nodes added to the network at later stages will represent or explain the finer details of this mapping or function that the network is trying to approximate. A hybrid of the differential evolution algorithm (Storn & Price, 1995) and the "*Regression by Leaps and Bounds*" algorithm (Furnival & Wilson, 1974) is used to optimise the nodes which are added to the network at any given stage.

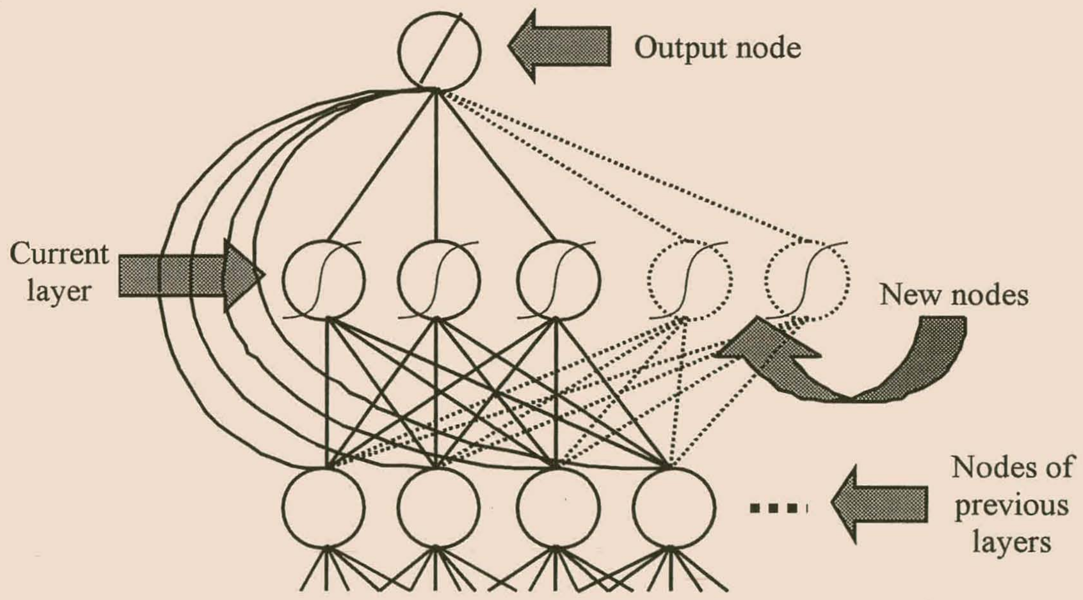

Figure 4.1 Stepwise addition of groups of nodes to the existing network.

4.1.2 The differential evolution algorithm

The differential evolution (DE) algorithm (Storn & Price, 1995; Price & Storn 1997; Storn & Price, 1997) is a global optimisation algorithm. The individuals of the population that it evolves are real-valued vectors with each co-ordinate of a vector representing a parameter that must be optimised. The DE algorithm was examined on a test-bed ranging from 2 to 30 dimensional problems including the DE Jong test suite (Storn & Price, 1995). It converged faster and with more certainty than the Annealed Nelder & Mead (ANM) approach (Press et al., 1992) as well as Adaptive Simulated Annealing (ASA) (Ingber, 1993), that are both known to be very powerful. The DE algorithm also finish 3rd at the First International Contest on Evolutionary Computation (1st ICEO) which was held in Nagoya, May 1996. DE turned out to be the best evolutionary algorithm for solving the real-valued test function suite of the 1st ICEO. (The first two places were given to non-GA type algorithms, that are not universally applicable.)

DE generates new vectors by adding the scaled difference vector of two randomly chosen individuals to a randomly chosen third individual. These three individuals will be called the generating individuals. The degree of scaling is set by a user-defined control parameter F. This procedure is similar to the concept of the mutation operator found in conventional genetic algorithms, but differs from the typical mutation operator in that it is not entirely random. Thus let $\boldsymbol{v}$ be the newly formed parameter vector and V be the parameter matrix of the population of size N_P, then:

$$\boldsymbol{v} = \mathbf{V}_{r1} + F(\mathbf{V}_{r2} - \mathbf{V}_{r3}), \qquad \text{Equation 4.1}$$

with $r1, r2, r3 \in [1, N_P]$, integer and mutually different, $F>0$.

In the DE algorithm a more greedy method to form the new vector parameters also exists. In that particular strategy $r3$ is the best individual from the population (Storn & Price, 1995). This strategy is called DE2 and will be considered in the next chapter. In this chapter only the more conservative strategy, DE1, will be used. A one way crossover of parameters (using a user-defined probability CR) occurs between the newly formed vector or individual and a fourth randomly chosen member of the old population, the latter termed the parent individual. The crossover occurs for each parameter of each individual with a probability of 1-CR, where $0 < CR \leq 1$. Thus if CR is close to 0, most of the parameters of the existing parent will be copied into the newly formed vector. The crossover used in the DE algorithm is thus actually a copying of parameters. Nevertheless, it serves the same purpose as crossover does in genetic algorithms.

The copying action yields a new individual, termed the trial individual. If this trial individual is calculated to be fitter than the parent individual, it replaces the parent in the original population.

This procedure is repeated for a specified number of iterations, after which the individual with the highest fitness is chosen as the final solution to the optimisation problem. Apart from setting the number of the iterations for which the algorithm must run, the size of the population must also be defined by the user. Note that the DE algorithm only uses the fitness of an individual to select the better of two individuals when evolving the population, or to select the best individual from the population after all iterations have been completed. An absolute fitness value, as found with conventional genetic algorithms, is therefore not required. A selection criterion with which to select the better of two individuals or the best individual from the entire population is used instead.

If, in the DE algorithm, each vector of the population were to parameterise a whole group of g nodes, evolution of the network would proceed very slowly. After a few iterations the population members will have moved away from those regions of the solution space with very low fitness values and improvements would be more difficult to achieve. If for instance the crossover rate were set moderately high, many parameters of the new competing trial member would differ from those of the old population member. It is likely that a few encoded nodes of the group will have improved through the crossover of parameters, while most of the other nodes of the group will remain in a state that is less useful to the neural network. The newly formed trial vector will be discarded even though some nodes have actually improved. If one were to circumvent this problem by setting the crossover rate very low, so that on average only the parameters of one node are adapted each iteration, convergence would be very slow. A similar argument will hold for conventional genetic algorithms. Furthermore, the differential evolution algorithm would suffer from the representational redundancy (as described in detail in section 3.3.2). As a result, direct application of the DE algorithm will not provide very efficient optimisation of the nodes.

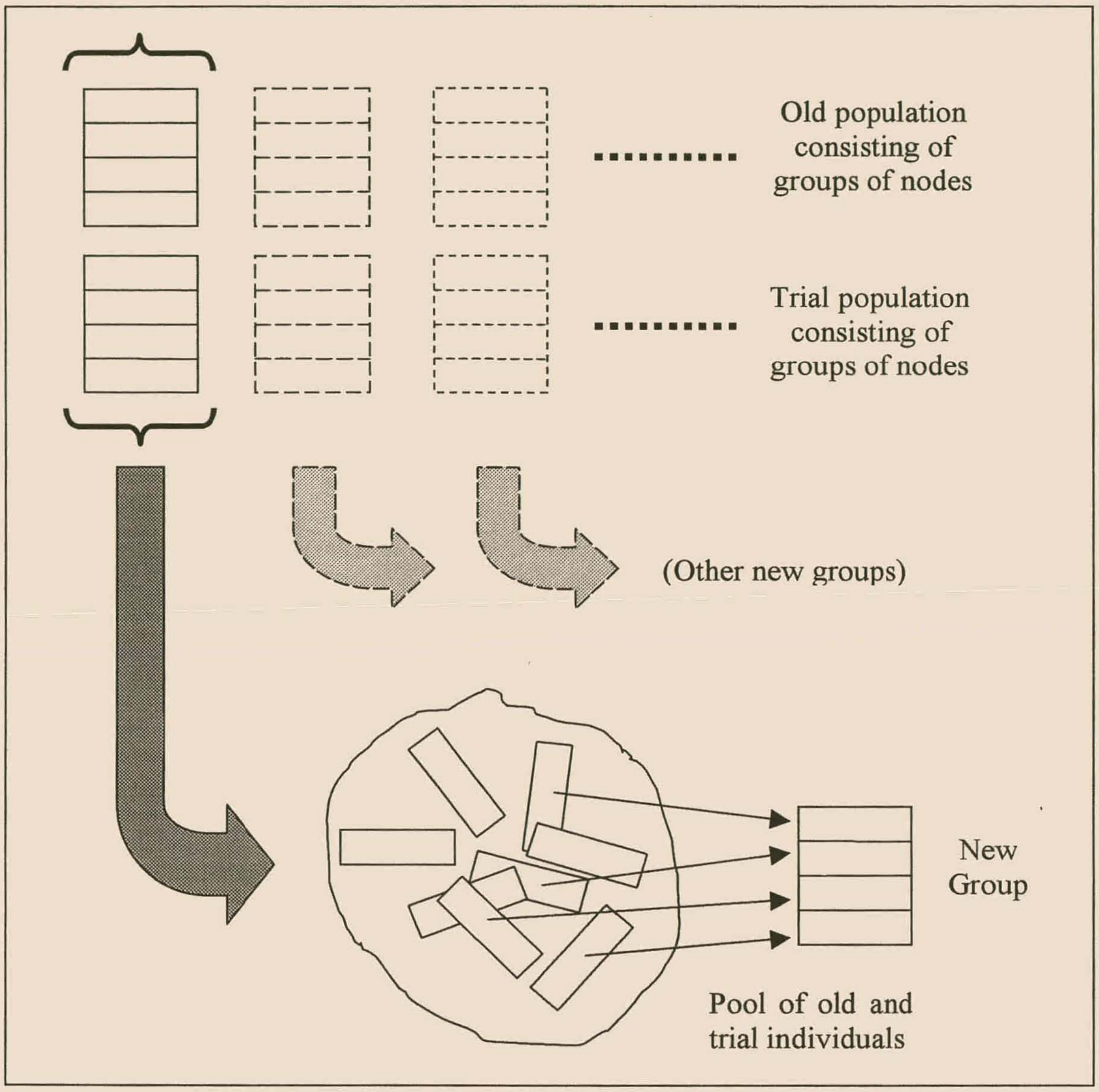
Old population consisting of groups of nodes
Trial population consisting of groups of nodes
(Other new groups)
New Group
Pool of old and trial individuals

Figure 4.2 The selection of the individuals of the new groups in the CERN algorithm.

4.1.3 Individuals in the CERN algorithm

The CERN algorithm will attempt to utilise the fact that the neural network is made up of separate nodes. It was therefore decided that each individual of the DE algorithm population will represent a single neural node. These individuals are bundled together in groups of g individuals each. The most basic genetic individuals are therefore single nodes and no longer groups of nodes. In other words, the original individuals representing a group of g nodes was split up into g individuals, each representing a node. The total number of individuals (nodes) in the population is therefore a multiple of g.

After each iteration a newly formed trial population of nodes is formed in the usual manner of differential evolution. Each individual can still adapt and crossover with any other individual in the population. After this adaptation each individual in an original group of individuals has a corresponding partner in a new group of individuals. Using the standard DE algorithm with these individuals might solve some of the problems associated with representational redundancy. Nevertheless, good nodes could still be lost when using a standard selection scheme. A better selection scheme is proposed below.

4.1.4 The combinatorial selection scheme

Instead of selecting either of the old or the new group of nodes collectively, the best possible combination of individuals (nodes) to survive to the next iteration is now chosen from a pool containing both the original group of individuals and the new group of individuals. Such an iteration of the CERN algorithm is illustrated in figure 4.2. The search required to find the best combination of individuals has to be performed several times for each iteration of the algorithm and this is normally computationally infeasible. The next section will show how it can be done using a special form of combinatorial search. After all the iterations have been completed, the group of individuals (nodes) is chosen which, after being included in the existing architecture of the neural network, best improves the prediction capability of the network. More specifically, the group of nodes is chosen which will give the newly adapted network the smallest residual sum of squares errors (RSS) after the least mean squares (LMS) algorithm had been applied to weights of the output node. In this regression all the incoming weights of the output node are still treated as if they are flexible. Since the combinatorial search procedure described below does not explicitly compute the weights of the output node, but only the RSS error, the weights are calculated after the last iteration by the LMS algorithm.

4.1.5 Combinatorial search procedure

Consider a group of g nodes that share a common input vector from the underlying network. Each node will extract a different feature from the input vector and together all these features must combine to explain as much of the variance of the output as possible. At a given iteration of the CERN algorithm this group of nodes needs to compete with its offspring, in other words, a group of g nodes must be selected from a combined pool of $2g$ nodes. The only method that will guarantee that the best possible combination of nodes chosen is one that considers all possible combinations of nodes. It is clear that there are in total (2g)!/(2g-g)!g! possible solutions, all of which must be evaluated.

There is only one principle by which the search for the best combination of nodes may be guided. If the output weights of the adapted neural network are found by a least squares method, then the RSS resulting from the newly added group of nodes is greater than or equal to the RSS of the containing set of nodes. In other words, $\mathrm{RSS}(B) \geq \mathrm{RSS}(A)$ for $B \subseteq A$.

The same principle was exploited by Furnival and Wilson (1974) to develop the previously mentioned methodology called "*Regression by Leaps and Bounds*", which selects the best possible regression variables from a pool of variables. The reader is referred to Furnival & Wilson (1974) and Seber (1977) for a more detailed description of this particular algorithm. What is relevant to this discussion is that the covariance matrix of a chosen set of regression variables holds all the information required to compute all the possible regression models. A new regression model is evaluated by Gaussian elimination on the covariance matrix, which requires few floating point operations. To compute the R^2 for all possible combinations of regressors out of a maximum of $2g$ regressors, the maximum number of floating point multiplication and division operations which are required is $\{6 \times 2^{2g} - g(2g + 7) - 6\}$ (Furnival & Wilson, 1974). Furthermore, we are only interested in finding the best g out of $2g$ regressors. In this case the maximum number of operations would be about half. When using the branch and bound search, this formula would be the worst case scenario and on average the number of floating point operations is orders of magnitude smaller (Furnival & Wilson, 1974). The search time is independent of the number of training data and this method is orders of magnitude faster than computing the error directly from the data for each possible combination of regression variables. Besides the algorithm of Furnival and Wilson (1974), it is possible to use other subset algorithms, that are discussed by Miller (1990). For example, an algorithm used by Narula and Wellington (1977, 1979) and Armstrong et al. (1984) uses the weighted sum of absolute errors as the criterion instead of the least squares criterion.

The niche problem, mentioned in chapter 3.3.3, is overcome in each group of nodes, since all possible combinations of the parent nodes and their offspring are considered during selection. The nodes finally chosen, are those that work the best to collectively fit the output function. Therefore nodes that specialise by extracting useful features from the input, different from those modelled by other nodes, will survive. The same applies to nodes, that best extract a feature, such as a separation boundary between two classes in the training data. Thus the individuals that find a niche will survive, although specialising sub-populations are not appointed for a specific niche. This will be considered in the next chapter.

The credit apportionment problem is not solved directly, as the DE algorithm does not require a fitness value for each individual (where credit must be apportioned to each individual). Only a selection criterion is required, which is furnished by the combinatorial search. This selection criterion considers all possible selections. One could thus say the credit apportionment problem is solved by brute, but effective computational force.

In order to apply this methodology to the particular problem of neural network optimisation, the final activation function to the output node needs to be a linear transfer function. In this case all the nodes connected to the output node can be thought of as regressors. The limitation of only using linear transfer functions in the output nodes is not particularly restrictive, as these functions are used almost exclusively for radial basis function networks. For problems with continuous outputs a linear output node is also believed to be the most suitable choice for multilayer perceptrons (MLP) (Hrycej, 1992). Note too that the MLP will remain a universal approximator (Hornik et al., 1989), even if linear transfer functions to output nodes are used.

The regression by leaps and bounds procedure can find the best *n* of *m* regressors for a R^2 error criterion. A further detail that still remains to be explained is how the fixed nodes of figure 4.1 are accommodated in this search procedure. They are fixed and will require no extra combinations to be examined. Their corresponding outgoing weights to the output nodes are to be kept flexible. Hence they need to be taken into account during the search procedure. This can be done effectively by performing the symmetric sweep operator (Seber 1977, p.354) on the covariance or product matrix. Thus first all the regressors that are to be made a permanent part of all the regressions are "swept" into the regression and then their corresponding rows and columns are deleted from the covariance (or product matrix).

4.2 Applying CERN to build specific neural network types

4.2.1 Radial basis function networks

Radial basis function networks (Powel, 1987; Broomhead & Lowe, 1988) have been applied successfully in many fields as good function approximators. This is owing to the fact that they have the capability to represent highly non-linear relationships, are resistant to noise in the data and yet yield relatively clear and well-behaved models.

Consider a set of n data points $\boldsymbol{x}_j \in \Re^d$, $j = \{1,...,n\}$ and the value of an unknown function f: $\Re^d \rightarrow \Re$ at each of the data points. The general structure of the radial basis function network, used to approximate the function f, is $y(\boldsymbol{x}) = w_0 + \Sigma_i w_i \Phi_i(\boldsymbol{x},c_i)$ where $c_i \in \Re^d$ is the centre of basis node i. $\Phi_i(x,c_i)$ is usually a Gaussian of the form $\exp(-|x-c|/2s)$, where $|\circ|$ is the Euclidean norm defined in $\Re^d$ and s is the width of the basis node.

Training the radial basis function network implies finding the set of basis nodes and weights such that the RMS error over the training data is a minimum. Owing to the fact that the training of the connection weights ($\boldsymbol{w}$) can be done by iterative or direct numerical solutions to the least squares problem, the most difficult task is finding a good set of basis functions. This is usually done in an unsupervised phase that clusters the input data for the selection of the centre co-ordinates (Moody & Darken, 1989). The widths of the basis functions are computed from the distances to the other cluster centres and the norm is taken to be the constant Euclidean norm. Although this method is computationally very efficient, it suffers from three serious drawbacks. First, the number of clusters needs to be specified in advance, which leads to similar problems as those encountered when choosing the initial number of hidden nodes in multilayer perceptrons. Second, clustering algorithms place most centres at those locations where many input vectors can be found, instead of reserving them for those regions that show great variations in the output, e.g. the class boundaries of a classification problem. Finally this method suffers from the curse of dimensionality, i.e. to achieve the same descriptive resolution in higher-dimensional problems an exponential increase in the number of hidden nodes is required.

These problems may be alleviated if one allows the network to adapt the number of nodes, as well as the norm of each of these nodes. A more general norm, other than the Euclidean norm, can be defined for each Gaussian node (Haykin, 1994). For example more general hyperellipsoidal Gaussians can be used instead of hyperspherical Gaussians (Pietruschka & Kinder, 1995). These nodes can have a high resolution in certain directions, i.e. dimensions, and lower resolutions in other dimensions. If required they can also be orientated in space to adjust the alignment of the nodes to the natural contours of the output function. CERN is used to evolve such ellipsoidal basis function networks that will be termed *oriented ellipsoidal basis functions*, or OEBF. Note that these are more specific than the oriented, non-radial basis functions proposed by Saha et al. (1991) in that only hyperellipsoids are taken into account and other second order polynomials such as hyperparaboloids or hyperbolics are not considered. The next section gives details about the translation of chromosomes into their hyperellipsoidal basis function phenotypes.

4.2.2 Chromosomal encoding of ellipsoidal basis functions

The DE algorithm and thus the CERN algorithm works exclusively with real-valued parameter vectors, whose components can each take on any value $\in \Re$. Therefore any real-valued parameter vector (chromosome) needs corresponding parameters specifying an ellipsoidal node (phenotype).

The nodes of an RBF network each has a centre, a generalised metric, as well as a width of the Gaussian, associated with the node. The information of each node is encoded in a chromosome. The metric and the width can be combined into a single positive, semidefinite, symmetric matrix A (Duda & Hart, 1973), which in symbolic form can be written $\|\mathbf{x}\|^2/s = \mathbf{x}^T A \mathbf{x}$. Such a matrix can always be written in terms of a positive diagonal matrix D and an orthogonal matrix Q (Watkins, 1991), where $A = Q^T D Q$.

The orthogonal matrix Q can be seen as an identity matrix on which $d \times (d\text{-}1)/2$ Givens rotations, also known as Jacobian rotations (Watkins, 1991), are applied. The matrix D accounts for the ellipsoidal shape of a given Gaussian, while the angles of the Givens rotations are responsible for rotating the hyperellipsoid in hyperspace. As a result of symmetry the optimal orientation of a given hyperellipsoid can always be found by rotating the initial axis-parallel hyperellipsoid through at most $\pi/4$ radians. The real-valued co-ordinates of the parameter vector which represent the rotations are therefore scaled by a suitable transfer function, (the scaled hyperbolic tangent function) to give an angle between $-\pi/4$ and $\pi/4$ radians. In the CERN algorithm the angles were allowed to increase by another ϕ radians so that an hyperellipsoid near a more suitable solution can rotate past $\pi/4$ by ϕ radians. It should be noted that the representation used here is not 1 to 1 i.e. several different chromosomes can actually be mapped to the same phenotype. A 1 to 1 representation can be formed using the Cholesky decomposition (Watkins 1991), for example.

The elements of the diagonal matrix D need to stay nonnegative. Furthermore, all very large values for these elements would translate into very narrow Gaussians. These are usually not optimal in reducing the RMS error and are also undesirable. In order to confine the algorithm to the more feasible solutions, the real-valued parameters are scaled to yield diagonal values of D

lying between zero and a positive number λ_{max}. This results in a Gaussian with a standard deviation that is bound in the lower limit by $\sigma_{min} = (\lambda_{max})^{-0.5}$ and unbound in the upper limit in every dimension. The centre co-ordinates ($\boldsymbol{c}$) of each Gaussian were simply attached to one end of the chromosome. Figure 4.3 shows such a chromosome for an input space of d dimensions. As shown in this figure, the centre co-ordinates, the scaling parameters of the hyperellipsoids and the Given's rotation angles are all encoded sequentially on the chromosome (parameter vector). For axis parallel ellipsoids the rotation angles are not present. Thus the matrix A is equal to the diagonal matrix D.

Although the centre of a Gaussian can in theory take any set of positional co-ordinates, it is limited to any of the existing co-ordinates of the training data points. It has been shown (Chen et al., 1990) that the resulting networks are sufficient to capture the essential features of the data. In the CERN algorithm the centres are allowed to move a distance δ away from the nearest training point, as measured in the RBF node's particular metric.

We chose ϕ to be $\pi / 20$, i.e. 10% of the maximum of the used range used in the rotation angles. The value used for δ was 0.25, which means that the minimum activation of the closest point to the centre of the Gaussian is at least $\exp(-0.5 \times 0.25) = 0.8825$. It is obvious that the value of λ_{max} and therefore also σ_{min}, the minimum width of the Gaussians in the input space is linked to possible scaling of the input data. In some preliminary experimental runs a value of 150 for λ_{max} proved to be sufficiently robust to deal with problems that had input variables in the range of 0 to 1. This value for λ_{max} was also used in the experiments below. Consequently the spread of input data of the problems below should be similar to that of the preliminary data set used. To conform to a measure that does not depend on single data points such as the range, the standard deviation is in some cases a better measure of the distribution of the data. The input features for these preliminary problems had standard deviations that ranged between 0.2 and 0.4. Therefore the scaling of the input data will be made so that the inputs have a standard deviation around 0.3 and is given in the Appendix for each problem, where local basis nodes are used.

In the DE algorithm the chromosomes get initialised with random numbers that are distributed uniformly throughout a predefined range that is likely to contain the best solution. The same was done for CERN. For ellipsoidal nodes the elements of the diagonal metric D were initially set to lie randomly between 0.5 and 50. The centre took on any randomly chosen data point and the angles were set between $-\pi/4$ and $\pi/4$.

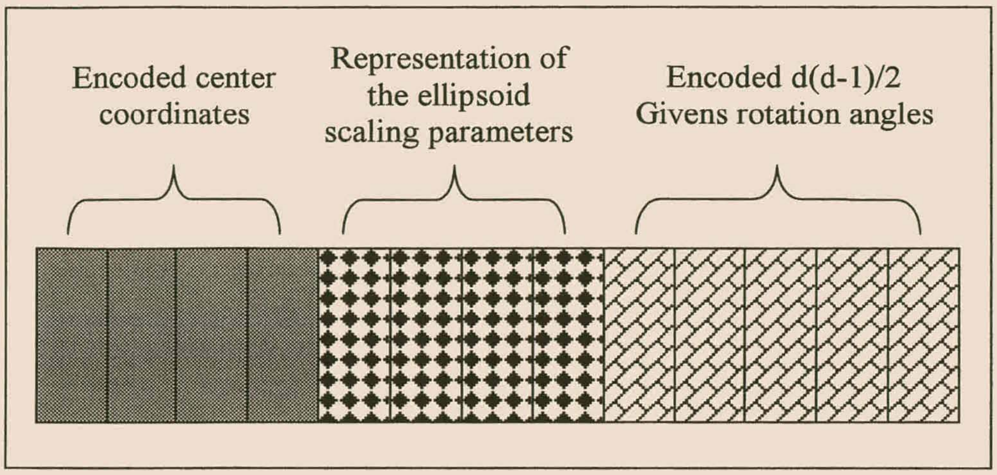
Encoded center coordinates
Representation of the ellipsoid scaling parameters
Encoded d(d-1)/2 Givens rotation angles

Figure 4.3 Chromosomal encoding of the ellipsoidal basis nodes.

4.2.3 The multilayer perceptron

In general a multilayer perceptron consists of a layer of input nodes, one or more hidden layers of inner product nodes and an output layer. Each layer is completely connected to the previous layer by a set of weights. Input signals propagate through the network from the input layer onwards to the output layer in a feedforward manner (Haykin, 1994).

The multilayer perceptron is usually trained by gradient descent methods (Werbos, 1974; Rumelhart & McClelland, 1986), in which the error is propagated backwards through the network. Despite the many local optima typically associated with the weight space of two networks, the gradient descent techniques have been applied successfully to numerous data sets. A problem is that the backpropagation algorithm sometimes requires several attempts to find the global or a near global optimum. Furthermore, the number of hidden nodes needs to be set in advance and the optimal number of nodes can usually only be found through extensive cross-validation.

Figure 4.1 shows that in addition to network weights connecting a given layer to the layer directly preceding it, the nodes evolved by CERN are also allowed to connect to the nodes of any other preceding layers. Moreover, the output node is always connected to all hidden nodes and optionally to the input nodes. The chromosomal encoding of the MLP inner product nodes for use by CERN is straightforward. The parameter vector is simply the weight vector of the incoming nodes, including the bias. The hyperbolic tangent function was chosen as the transfer function for each hidden node in all experiments reported in this article, although any other suitable transfer function could have been selected. For the summation nodes, also known as the tanh or inner product nodes, the size of the weight components can be limited to a number w_{max}. The same applies to the bias, which limited in size by b_{max}. The reason for this is that large weights, similar to sharp Gaussians, can result in a very rough output surface, that will not generalize well. It was shown empirically by Hinton (1987), MacKay (1991) as well as Hertz and Krogh (1992) that networks with smaller weights result in better generalization. In the experiments discussed in this chapter, w_{max} and b_{max} were both set to ∞. In subsequent chapters w_{max} was set to 30 and b_{max} to 8. Another important reason for restricting the weights is that activation of the hidden layer nodes can become saturated. For example the hyperbolic tangent transfer function with a weight component of 30 does not cause a transfer function that differs appreciably from one with a weight component of for example 300 (if the other components are relatively small). Weights as large as 300 can according to equation 4.1 result in other chromosomes getting weights of sizes in the range of 300 F. F is usually in the range of 0.25 and 0.75 and this will therefore disturb the evolutionary process unnecessarily. Limiting the weights will keep the values closer to the responsive region. The disadvantage is that very sharp peaks cannot be fitted when using this restriction. (This is similar to a fuzzy logic system where the designer usually does not use very sharp membership functions and also cannot get very sharp peaks in the output surface.) In real world problems the occurrence of such sudden peaks are unusual.

The ranges for the initial weight elements were [–1.5; 1.5] for the weights and [-1;1] for the bias.

4.3 Computational resources

4.3.1 Floating point operations

The extra time spent by the CERN algorithm to select the best combination is independent of the number of data points n and the number of dimensions d. For $n > 2000$ and $g < 9$, the biggest extra computational burden is to compute a larger product matrix for every regression. The impact this will have on the total number of floating point operations will be estimated below.

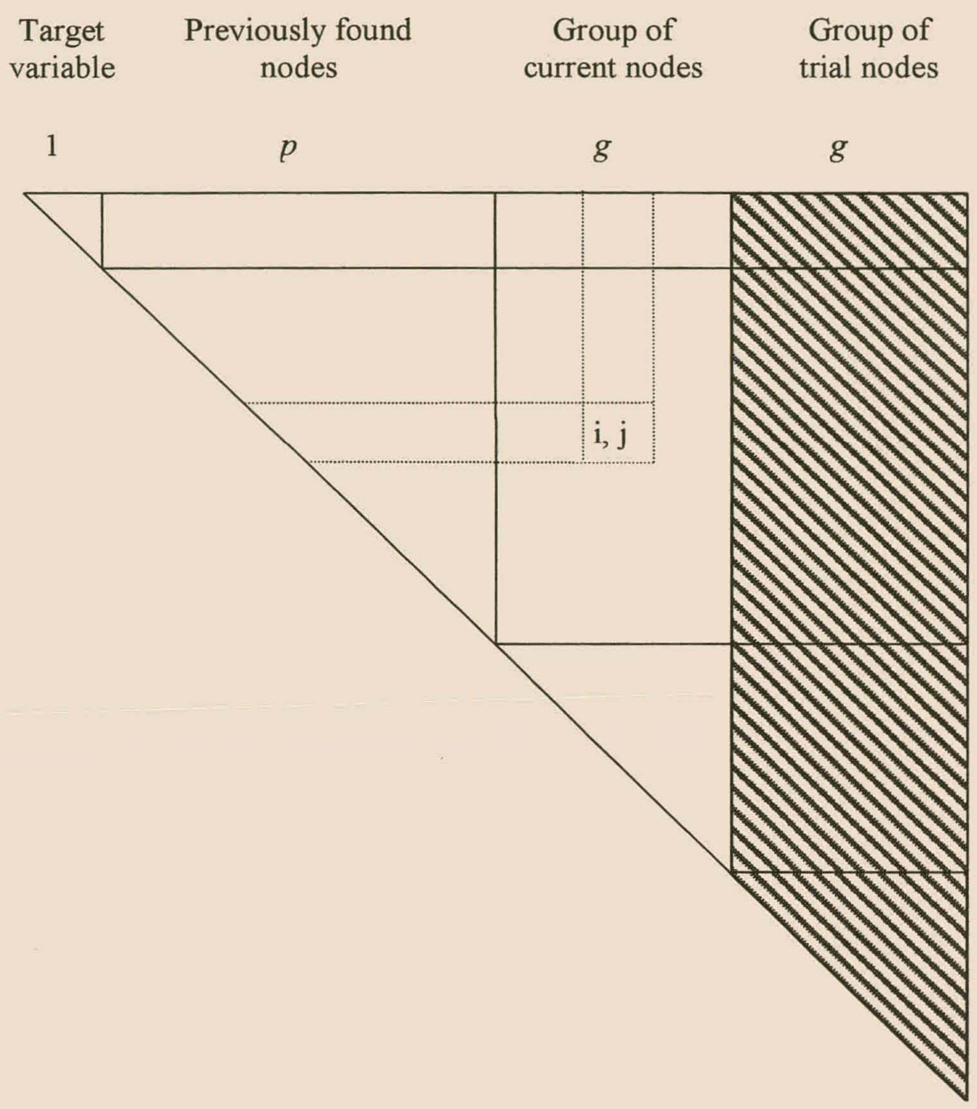

Figure 4.4 The sections of the symmetric product matrix. For example element i, j of the matrix is indicated by the middle square.

Each element of the matrix is a vector product and requires n multiplications *and*[*] additions. As the matrix is symmetric only the computation of the elements in the upper triangle of the matrix are required. For each iteration the program only updates the necessary fields of the product matrix. The matrix is resorted after every iteration to reuse the previously computed vector products. If a number of p previous nodes were already found and added to the hidden layer, the number of fields in the product matrix that need to be computed are: $g\{1+p+½(g+1) +g\}$. This is illustrated by the shaded area of the product matrix shown in figure 4.4. On the other hand, the number of elements to be computed for the DE algorithm at every iteration are $g\{1+p+½(g+1)\}$.

[*] In this section where applicable "*and*" will mean "as well as" or "plus" and not "including."

This is $g \times g$ vector products less. The DE algorithm does not need the vector products of the new trial nodes with the old trial nodes.

The computation of the activation of the various nodes requires the following number of operations for each data record:

- Summation nodes requires d multiplications *and* $d+1$ additions for $\boldsymbol{wx}+b$.
- Each basis node requires d additions for $\boldsymbol{r} = \boldsymbol{x}\text{-}\boldsymbol{c}$.
- On top of this the spherical Gaussian require d multiplications *and* d additions to compute $|\boldsymbol{r}| = \boldsymbol{r}' \bullet \boldsymbol{r}$. One more multiplication is needed for $|\boldsymbol{r}|/s$, where s is the width.
- axis parallel basis function use $2d$ multiplications *and* d additions for $||\boldsymbol{r}|| = \boldsymbol{r}'\mathrm{D}\boldsymbol{r}$, where it is used that D is diagonal.
- using the fact that A is symmetric, orientated basis functions need $\frac{1}{2}d(d\text{-}1)$ 2 multiplications *and* additions for the off-diagonal elements of A. To add the contribution of the diagonal elements to $||\boldsymbol{r}|| = \boldsymbol{r}'\mathrm{A}\boldsymbol{r}$ another $2d$ multiplications *and* d additions are employed.

The transfer functions such as the Gaussians and the hyperbolic tangent functions can be kept in a lookup table and require a few extra operations that are independent of d. The programming overhead required to keep track of the computations is closely proportional to the number of operations required. Furthermore, one notices that for all the computations the number of multiplications is almost equal to the number of additions (except the spherical basis nodes require d + 1 multiplications *and* $2d$ additions). Thus for g nodes and n data records the total number of operations multiplications or additions is:

- Summation node: $2ngd + ng$
- Spherical basis function: $3ngd + ng$
- Axis parallel basis function: $4ngd$
- Orientated basis function: $ngd^2 + 3ngd$

To compare to this the operations required to compute the product matrix are:

- DE $2ng\{1+p+{}^1/_2(g+1)\} = 3ng + 2ngp + ng^2$
- CERN $2ng\{1+p+{}^1/_2(g+1) + g\} = 3ng + 2ngp + 3ng^2$

The number of operations required to compute the combinatorial selection scheme is data dependent, but is typically very much smaller than $\{6 \times 2^{2g} - g(2g+7) - 6\}$. (Furnival &

Wilson, 1974). This is the worst case scenario. The authors reported that the relative time saved by using the bounded search, increases with the number of total regressors (k) to be considered. For example for some specific problem and $k = 15$ they counted 11000 operations. For $k = 20$ it was 66766. They noted that this number may vary by a factor of 2 in either direction. As only the division and multiplication were counted we shall multiply their counts by a factor of 2 to get a better approximation of the total number of operations as was done above. Furthermore, the combinatorial scheme has a relatively high amount of overhead compared to the straightforward computations of the activations of the nodes. Thus in total one could multiply the number of operations they counted by 8 to get a pessimistic approximation of the relative time used up by the search. We will later see that this time will not make a big difference to the total computation time required for a data sets of size at least 10 by 2000 provided that $g \leq 10$.

To best illustrate these equations one can plot the number of operations, for various values of g, p, d and n. To make this possible let g be fixed at 8. Increasing the total number of records (n) from 200 to 25000 is sufficient to display their influence. To show the effect of d, the dimensions of the data, we assume to be dealing with problems of 5, 10 and 20 dimensions. The number of previously found nodes, was set equal to $d + g$. From figure 4.1 one sees that this would correspond to a neural network where the input nodes are directly connected to the output nodes and an additional g nodes had already been optimized and added to the hidden layer. For such a network the floating point operations are plotted that are required every iteration for each group when an additional g nodes are optimized. The resulting total number of operations required for the various computations are plotted in figure 4.5.

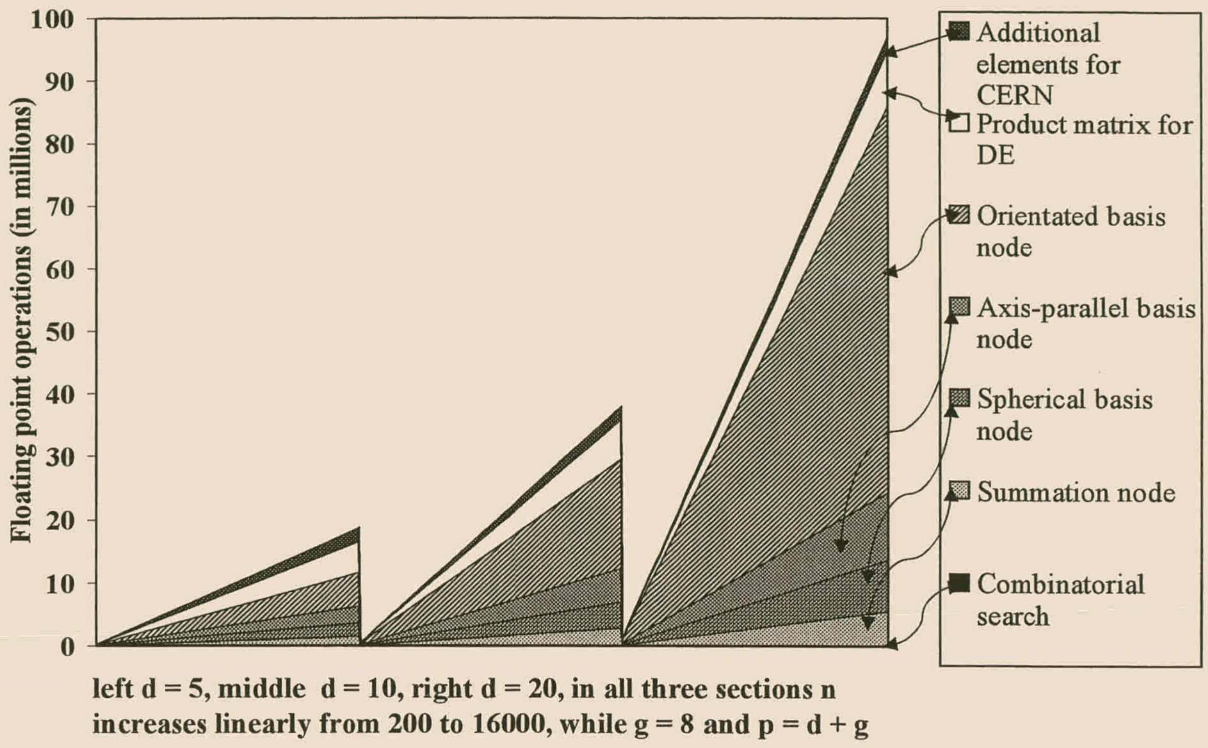

Figure 4.5 The total number of floating point operations required every iteration for each of the N_g groups of the CERN algorithm.

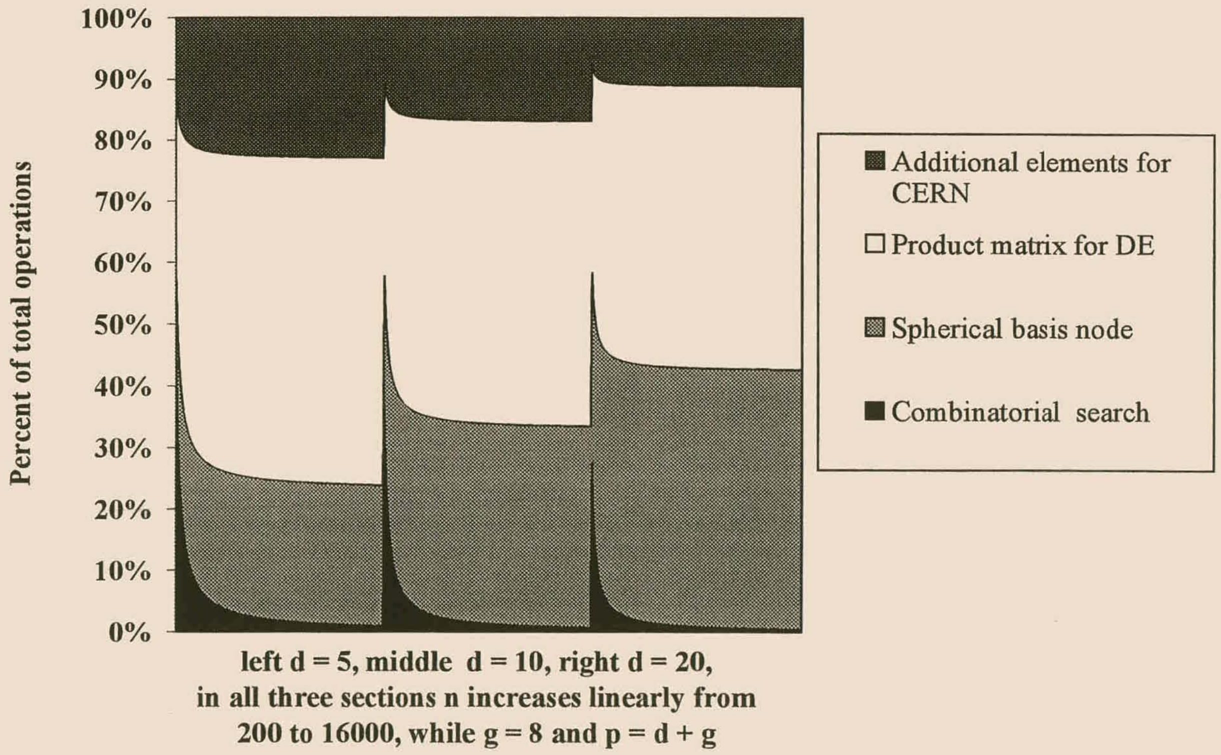

Figure 4.6(a) The percentage of total operations required for the different subroutines of the CERN algorithm with radial basis nodes.

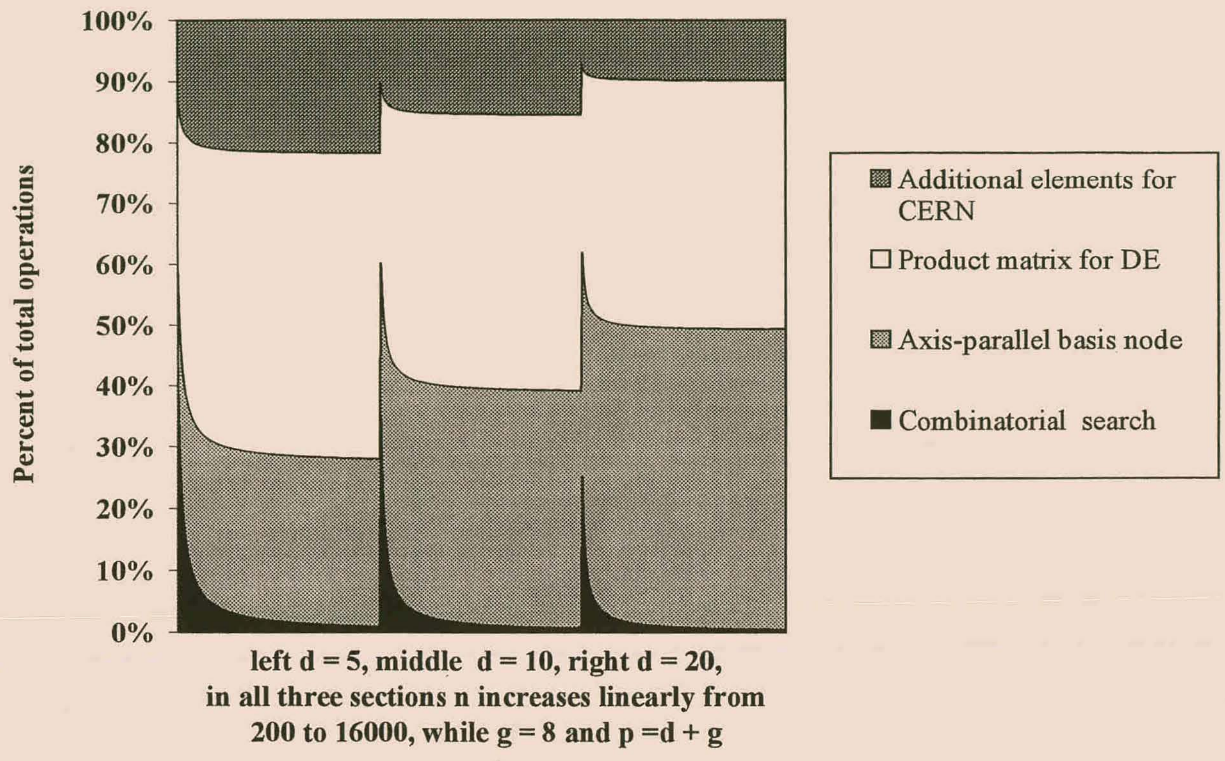

Figure 4.6(b) The percentage of total operations required for the different subroutines of the CERN algorithm with axis-parallel ellipsoidal basis nodes.

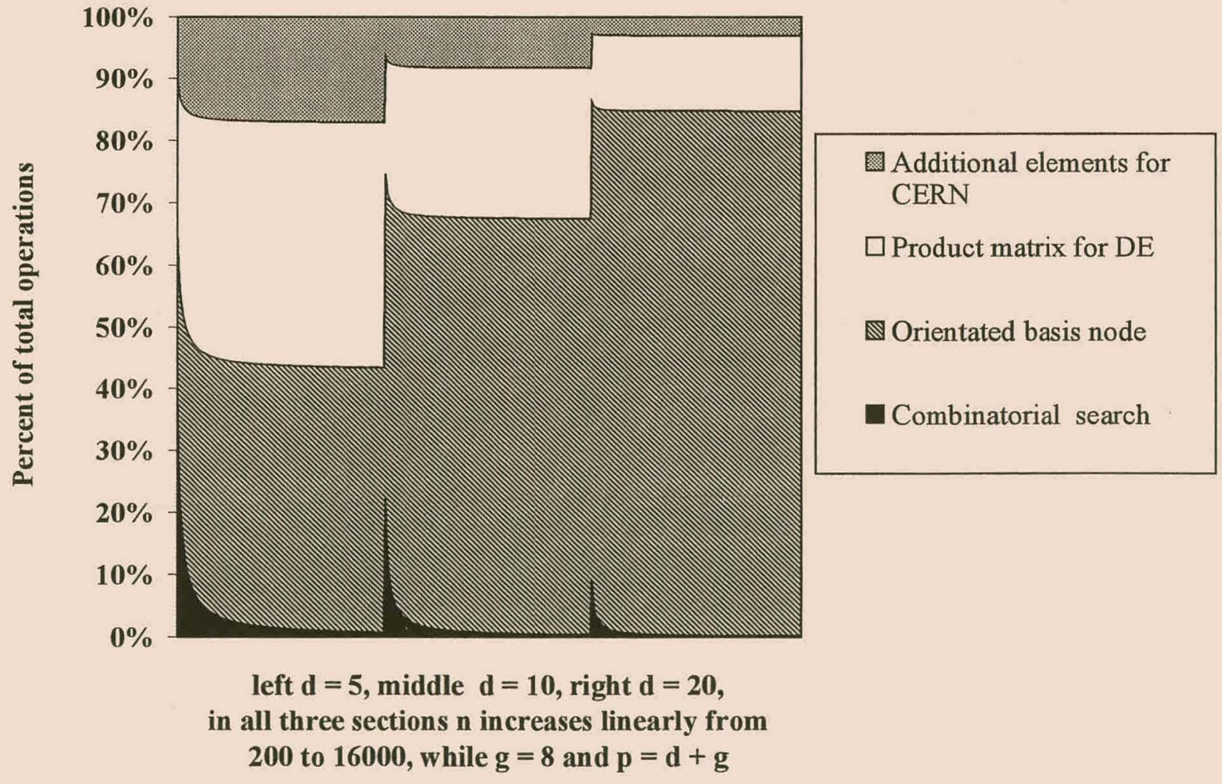

Figure 4.6(c) The percentage of total operations required for the different subroutines of the CERN algorithm with orientated ellipsoidal basis nodes.

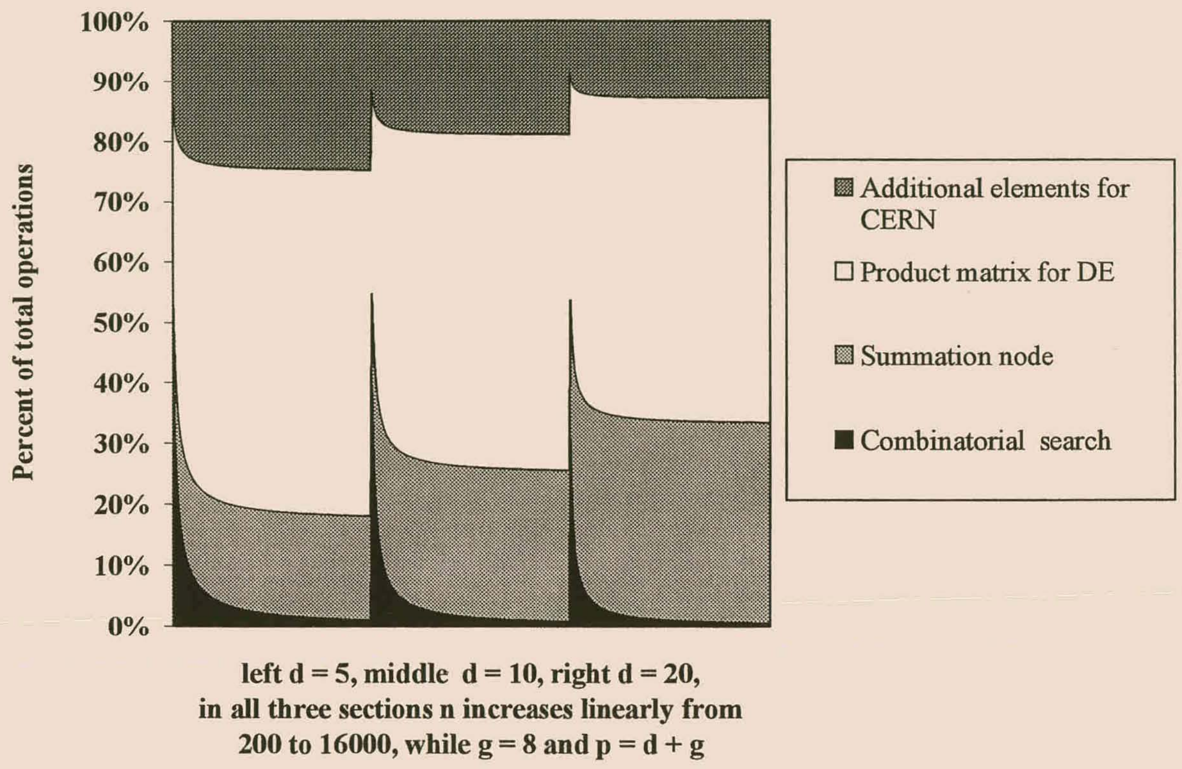

Figure 4.6(d) The percentage of total operations required for the different subroutines of the CERN algorithm with summation nodes.

These are the number of operations required every iteration for each group of nodes. We see that especially for the 20-dimensional problem the orientated basis function would use by far the most number of operations. The combinatorial search time can hardly be noticed.

To get a clearer view, one can plot the percentage of time taken by the different computation routines. This is done for each node type in figures 4.6 (a)-(d). Thus for example in figure 4.6(a) the percentages are given for the case where we are optimizing $g = 8$ radial basis nodes simultaneously. They are located in a network having d input nodes, while $d + g$ nodes are already connected to the output node. The number of data points n in the training set changes on the graphs. One can thus clearly see that the combinatorial search time is very small. For the summation nodes and 1500 data points it requires only a mere 10% of the total number of floating point operations and decreases further with n. For the other node types it is an even smaller percentage. The extra amount of time required to compute the additional elements of the product matrix for the CERN algorithm is also not appreciable. It ranges between 2% and 24% of the total number of operations and decreases with d.

4.3.2 Memory requirements

To be able to make use of the algorithm as described above one needs to store the activations of all the nodes for every data point. If not, the activation of each node needs to be computed again. Figures 4.5(a)–(c) show that this would typically increase the number of operations by 50%. The amount of memory required is $2ng \times N_G$ times 8, if double precision storage is used. N_G is the number of groups in the population. Thus for a data set of n = 100 000, a population of size N_G = 100 with g = 8 nodes each, would require 1.22 gigabytes of memory. Although some of it can be stored in virtual memory, this would become a problem for a typical PC. If no more memory is available, either less nodes need to be allocated, g decreased or the activations of the current population must be recalculated with every iteration. To use single point precision throughout the program is not advisable. Rounding errors can accumulate during the operations on the product matrix.

4.3.3 Numeric stability

The *regression by leaps and bounds* algorithm can become unstable for ill-conditioned matrices. A regressor was removed if it was too linearly dependent on the other regressors. In the working program a pivot tolerance of p_t had to be exceeded, when pivoting on the variable during the formation of the inverse matrix. The algorithm was made significantly more stable by adding a ridge of size k_r to the regression (Seber 1977). A ridge as small as k_r = 0.002 and value of $p_t = 1*10^{-15}$ were used in over a 1000 experiments run with CERN and the regressions never become unstable.

4.4 Experimental set-ups to test the learning of CERN

The efficiency of the CERN algorithm can be illustrated by means of the Double-Spiral problem and the Mackey-Glass time series problem (Mackey & Glass, 1977). Both these problems have been considered extensively by other investigators, which can facilitate the interpretation of results. Experiments were also conducted on a real world data set where the task was the prediction of the age of abalone (Nash et al., 1994). See Appendix A for a summary of the data sets. For various optimisation problems with the DE algorithm Storm and Price (1995) used various values of the crossover rate (*CR*) that ranged between 0.2 and 1, and scaling factors (*F*) between 0.4 to 1. After some experiments on the training data sets, it was found that the CERN algorithm converged more rapidly with slightly lower values of CR and for F, than typically used in the DE algorithm. Consequently thereafter a value of 0.4 was used for both *CR* and the *F*. Just like for the DE algorithm these values are not necessarily optimal for all problems. Two of the data sets have relatively few data points. Thus the combinatorial search time could take a considerable fraction of time. It was decided to select the group size in each experiment, so that the combinatorial search did not have a large impact on the total computation time of each run. For problems with a continuous output, the results of this and subsequent chapters are given in terms of percentage of variance explained, or percentage of variance unexplained, see section 2.4.3.

4.4.1 Comparison of learning speed in CERN and the DE algorithm

In order to determine the increase in efficiency of CERN over the conventional DE algorithm, both algorithms were used to optimise six axis-parallel ellipsoidal nodes, which were added to a basis function network. The network's input nodes were connected to the output node. The training data were the Mackey Glass time series. The networks were not evaluated on an independent test set during this experiment, as we were only interested in learning speed.

Figure 4.7 shows the effect of various settings of CR and F on the decrease in the unexplained variance during evolution of the two algorithms. After 25 iterations and a population of 60×6 nodes the CERN algorithm consistently outperforms the DE algorithm by a margin of 4% unexplained variance. Furthermore, for all settings CERN reached an error of $\approx$ 15% within 25 iterations, while the DE algorithm with the best setting required about 200 iterations to obtain such a result. Not shown on the graph are some further DE runs with F = 0.5, CR = 0.25; F = 0.5, CR = 0.02 and F = 0.9, CR = 0.05, none of which resulted in better learning than those DE runs shown. It was also examined whether the encoding of the nodes as single individuals or the more advanced selection scheme was responsible for the improvements of CERN over the DE algorithm. Thus for this experiment the CERN algorithm's selection scheme was replaced with the conventional selection scheme. In other words, it was to pick either of the groups of nodes rather than the best possible combination. The results are also given in figure 4.7, and are termed "DE with nodes as individuals". It can be seen that this algorithm does not fair appreciably better than the DE algorithm in optimising the network parameters. One can conclude that it is the combinatorial selection scheme that is mainly responsible for the fast learning of the CERN algorithm.

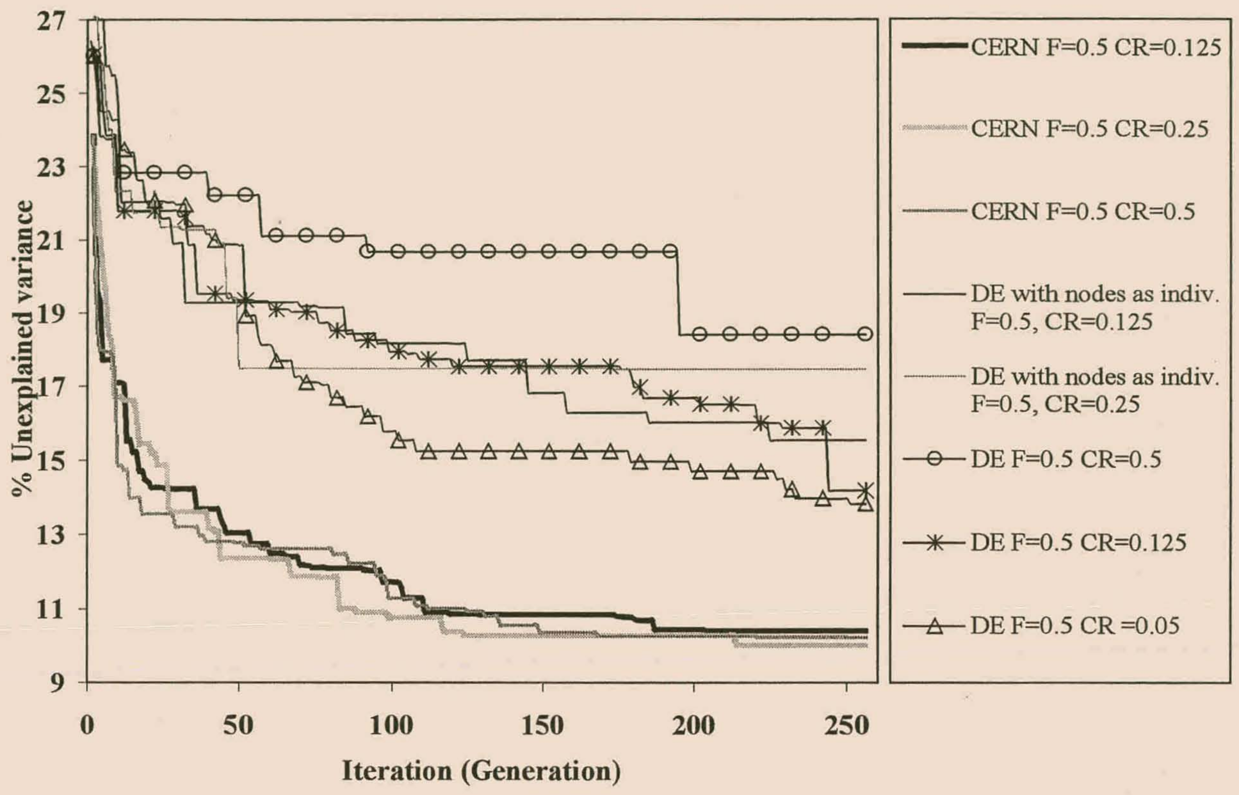

Figure 4.7 The learning error of the CERN and the DE algorithm when optimising six axis-parallel ellipsoidal nodes for the Mackey-Glass time series. The learning curves are stochastic and the differences owing to the CR settings might not be significant and are not the focus of this graph. The effect of CR on CERN is examined in more detail in chapter 5.

CERN was also compared to the DE algorithm on the more difficult Spiral classification task. Both algorithms were used to optimise seven orientated ellipsoidal nodes, which were added to basis function network trained on the Double-Spiral problem. Figure 4.8 shows the effect of some settings of *CR* and *F* on the decrease in the RMS error during evolution of the two algorithms. For both algorithms a population of 180 groups was used. It is clear that CERN significantly outperforms the conventional DE algorithm.

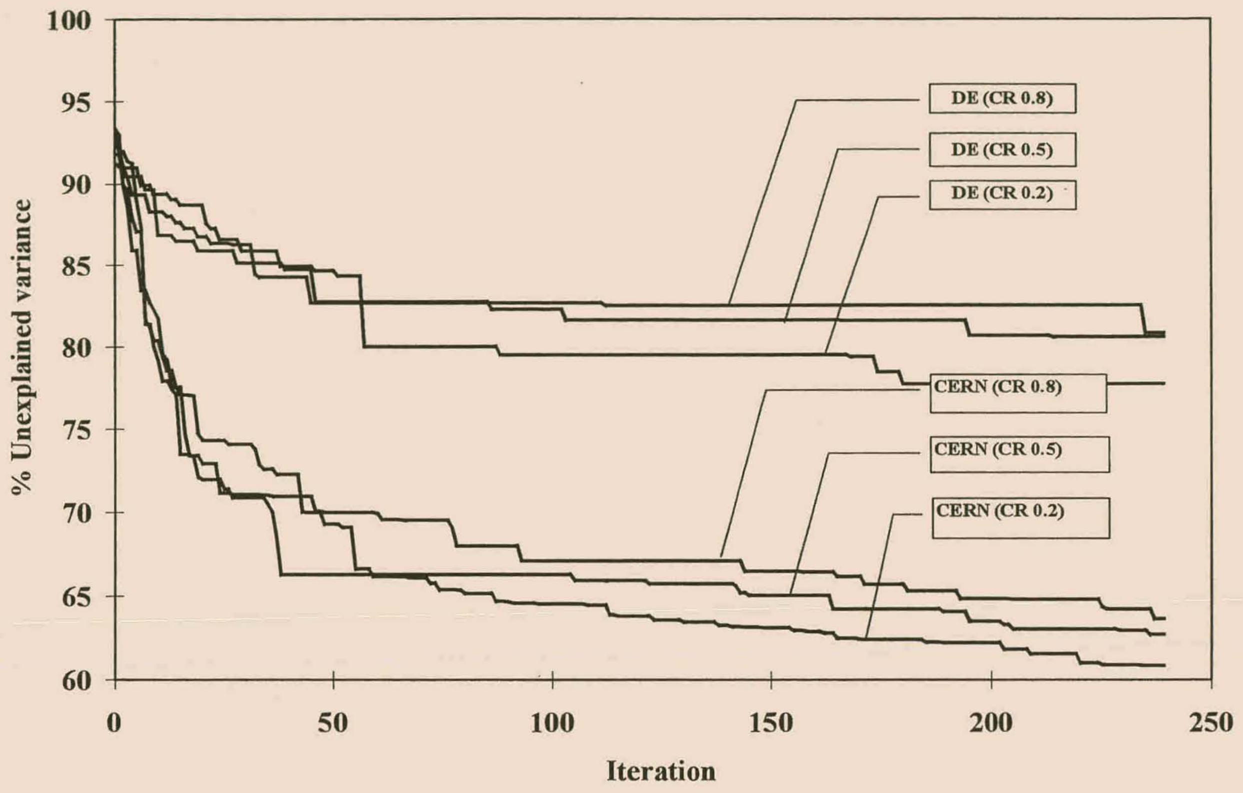

Figure 4.8 The training error vs. iteration number of the CERN and the DE algorithm when optimising 7 orientated ellipsoidal nodes for the spiral data set. The learning curves are stochastic and the differences owing to the CR settings might not be significant and are not the focus of this graph. The effect of CR on CERN is examined in more detail in chapter 5.

4.4.2 Mackey Glass time series

Two networks were evolved for this problem using CERN. First, an MLP network with a single hidden layer and additional connections from the input layer to the output node was evolved. After each step of evolution the best group of seven nodes found by the hybrid DE algorithm was allocated to the network. The population was initialised to 1400 individuals (200 × 7 groups of nodes) at each stage and these were allowed to evolve over 300 generations. The normalised error (the quotient of the RMS error and the standard deviation of the target output), denoted NRMS, of the current network on the test set was monitored throughout. This error measure is used frequently for this data set. The network obtained a minimum error of 0.072 after the addition of 98 nodes. Thereafter the network error increased slightly to a final value of 0.073 after all 112 hidden nodes had been allocated to the network.

Second, an OEBF network was evolved. Initially groups of five nodes were allocated to the network. The number of nodes in each group gradually decreased to one as the network evolved. At each step of evolution the population size was initialised to $250 \times g$ individuals. The number of iterations at each step of evolution was the rounded integer result of $140 \times g^{0.5}$. The final number of nodes in the hidden layer was 64. Figure 4.9 shows the NRMS error of the network on the test set during the experiment. For comparative purposes the performance of an RBF network that initialised the nodes of the hidden layer with conventional *k*-means clustering is also shown. Owing to the fact that hyperellipsoidal nodes have more free parameters than hyperspherical nodes, a further comparison of the two approaches is given in figure 4.10. Here the NRMS error is plotted against the number of free parameters used in the given model. Note that figures 4.9 and 4.10 are presented on a log-log scale.

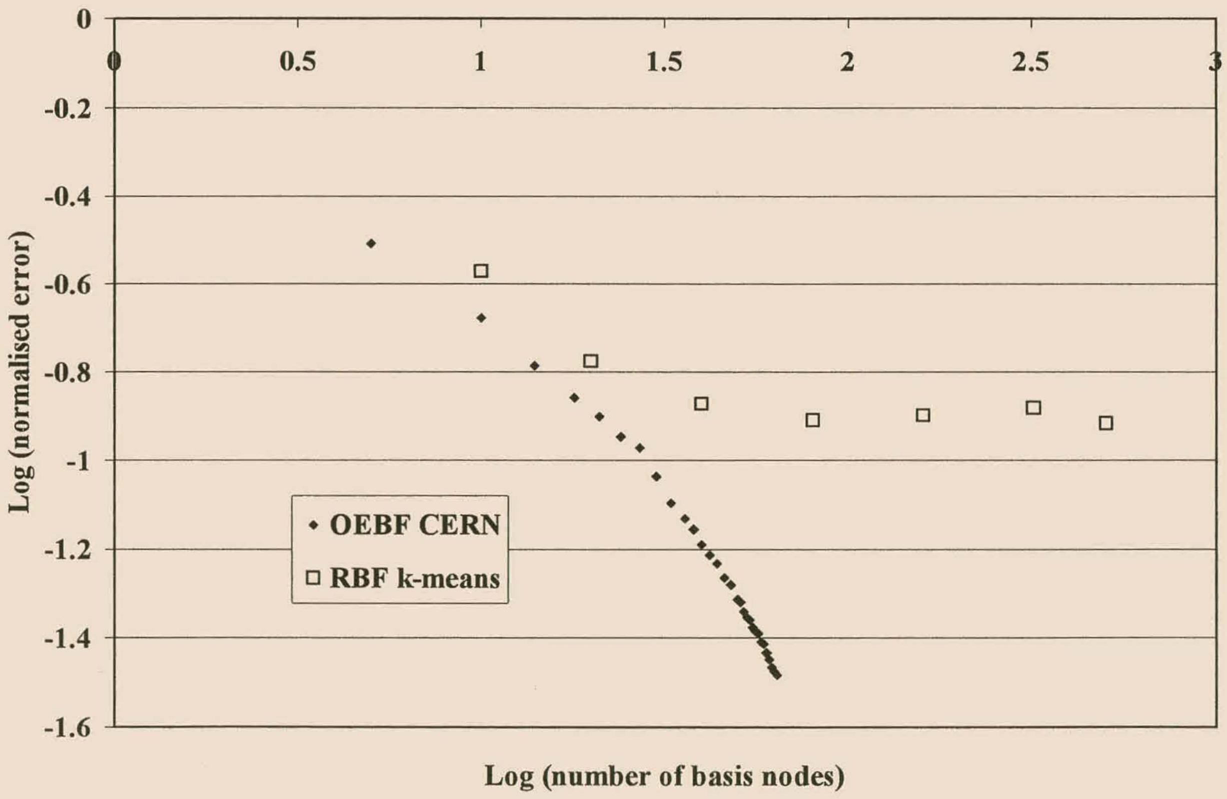

Figure 4.9 NRMS error of OEBF network and RBF network on Mackey-Glass time series vs. the number of local basis nodes.

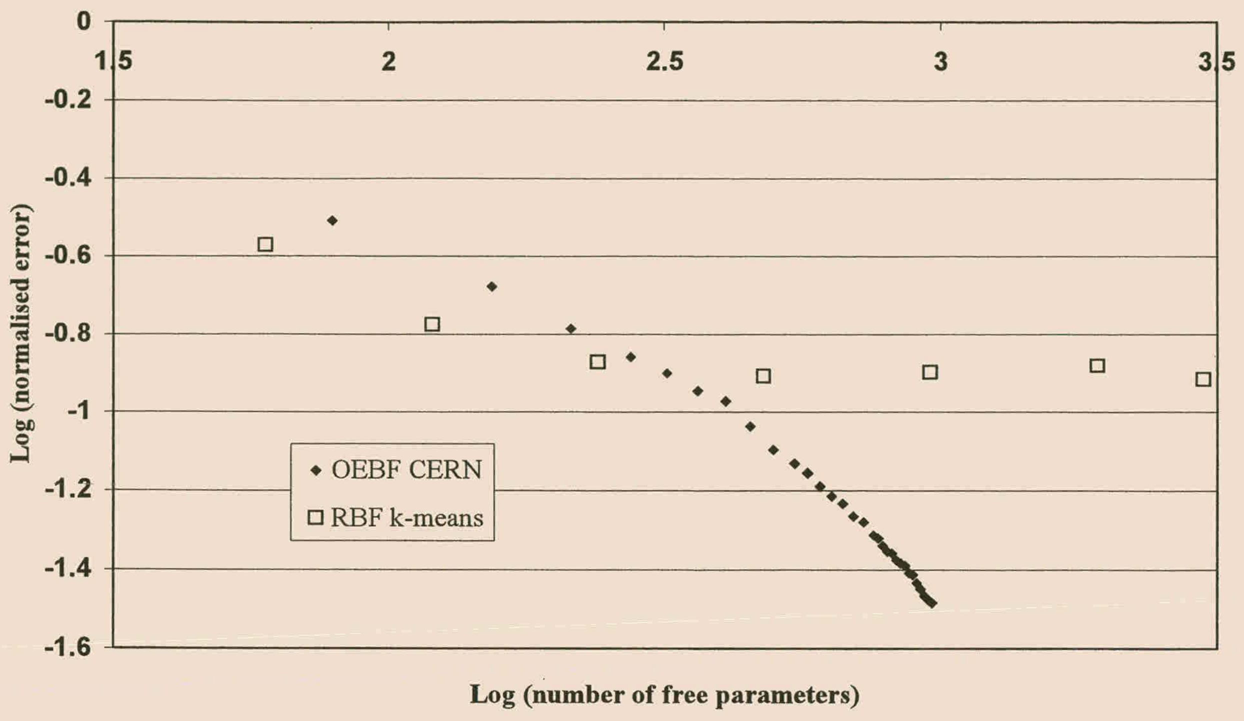

Figure 4.10 NRMS error of OEBF network and RBF network vs. number of free parameters for the Mackey-Glass time series

4.4.3 The Double-Spiral problem

CERN was used to evolve a MLP with hidden nodes employing only hyperbolic tangent transfer functions in the hidden layer, while the output node was restricted to a linear transfer function. Groups of eight nodes were sequentially added to the network, each to a new hidden layer. The population that CERN evolved contained 960 (120 × 8 nodes) individuals. At each phase of training the algorithm was allowed 120 iterations to evolve the best group of nodes to add to the network. CERN evolved five hidden layers of eight nodes each. Each hidden layer was connected to the previous layer as well as to the input layer. The network was able to classify all 194 records correctly. Figure 4.11 depicts the final decision surface obtained by the network.

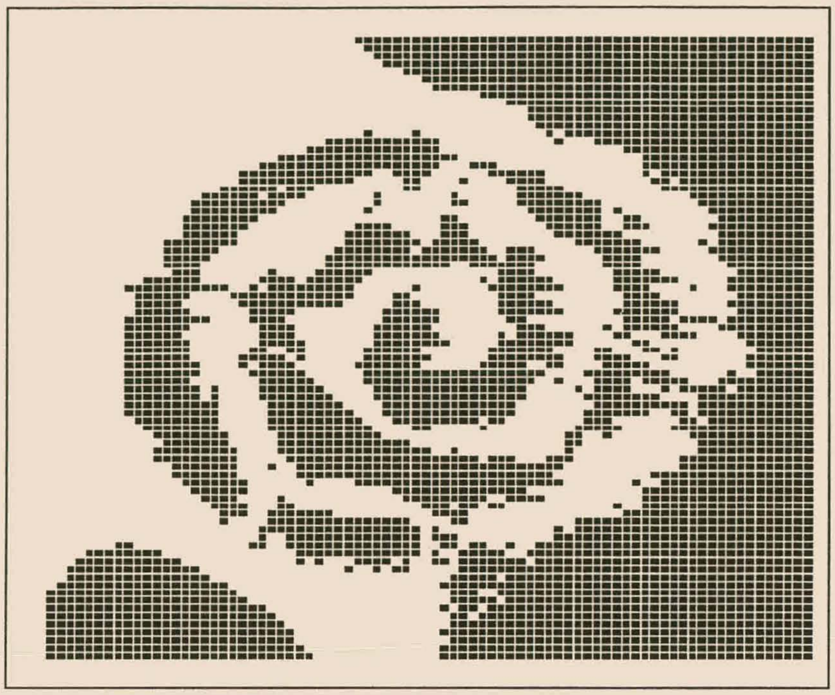

Figure 4.11 The decision region obtained by the MLP network evolved using CERN for the Double-Spiral problem. The dark shaded area indicates the region classified to belong to class 'o', see Figure A.1

An OEBF network was also evolved. As evolution continued the groups of nodes that were allocated to the network contained a decreasing number of nodes. In particular , the number of nodes decreased from 5 to 1 in the sequence g = 5, 4, 4, 3, 3, 2, 2, 1 and finally 1. Each population that CERN evolved contained 180 × *g* individuals. The number of iterations was set at 200 for each phase of evolution. The final network was able to classify all 194 records correctly after the addition of only these 25 basis nodes. The resulting decision region is given in figure 4.12.

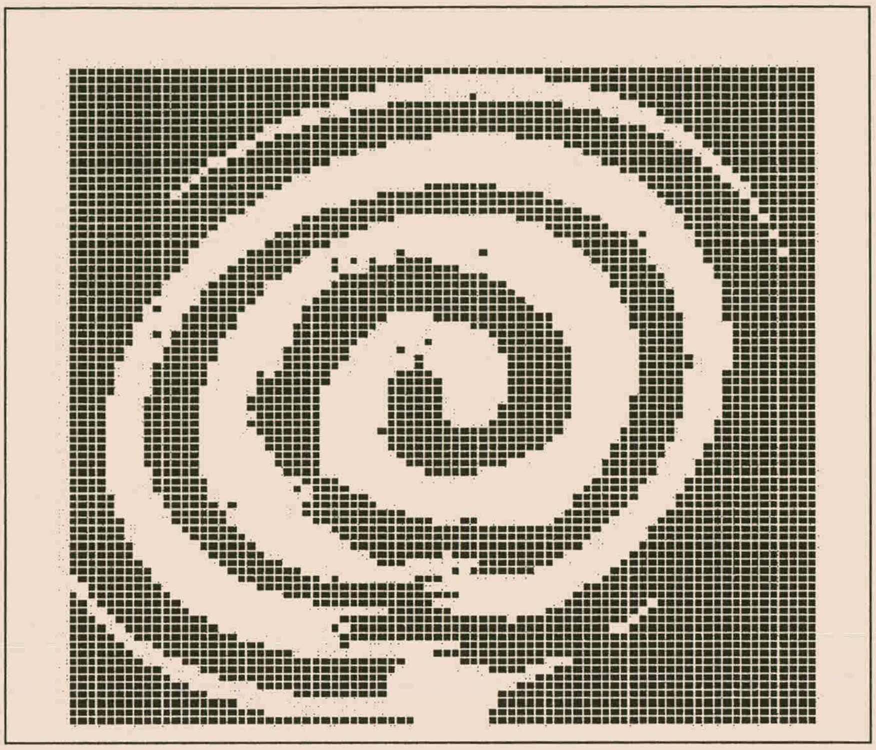

Figure 4.12 The decision region obtained by the OEBF network evolved using CERN for the Double-Spiral problem.

4.4.4 Abalone data set

For comparative purposes an MLP with a modified backpropagation, delta-bar-delta learning rule (Jacobs, 1988) as well as an RBF network with *k*-means clustering initialisation were trained on the data. The multilayer perceptron was trained for 10000 epochs.

The best architecture of each network, including the optimal number of hidden nodes, was found by performing a number of cross-validation runs on the training data. A single hidden layer containing 100 basis nodes for the RBF network and one containing 6 nodes in the case of the MLP network were found to work best. A slight decrease in performance during the cross-validation runs was observed for the MLP when the input layer was directly connected to the output node, and therefore these connections were not employed in the network architecture. Conversely, the RBF network showed improved generalisation when these connections were implemented and therefore they were retained in the final RBF network architecture.

Next, the optimal MLP and OEBF network sizes to be used by CERN were obtained by cross-validation on the training data. The number of nodes in the hidden layer as well as the optional connection of the hidden layer to the output layer were varied. The population used by CERN to evolve the best MLP network contained $150 \times g$ individuals. At each step the population was evolved for 900 iterations. The OEBF network was evolved using the same size population, but the number of iterations at each step was restricted to 300. This was done because the computation of the activations of orientated, ellipsoidal nodes takes significantly longer than the computation of the activations of the inner product nodes of a MLP network.

The MLP network that gave the best results with cross-validation was found to have six hidden nodes and no connection of the input layer to the output layer. This is the same architecture as the MLP network that was trained with backpropagation as described above. Consequently, the six nodes of the hidden layer of such a new MLP network were evolved with CERN (in one stage) using the entire training data set. Similarly, cross-validation on an OEBF network found that a group of five nodes was optimal, where the input nodes were not connected to the output nodes. Accordingly five oriented ellipsoidal nodes were placed in the hidden layer of a new OEBF network and evolved on the whole training data using CERN, again in a single stage.

4.5 DISCUSSION OF RESULTS

4.5.1 Mackey-Glass time series

Table 4.1 provides a summary of all the generalisation results of the various algorithms and network types considered here. The MLP network evolved by CERN for the Mackey-Glass time series achieved a normalised error of 0.072 using a single hidden layer of 98 nodes. This compares favourably with the result of 0.10 obtained by Littmann and Ritter (1993) who used a cascade neural network architecture with error training. However, their cascade network used a total of 240 nodes in 4 hidden layers to obtain this result. The lowest normalised error achieved by these authors using only a single layer was 0.15.

Unfortunately the Mackey-Glass benchmark has in the past been set up slightly differently by each of the various authors who have reported results of experiments involving this data set. Yao and Liu (1997) present results of experiments that used a value of 84 for T (the results reported here are based on a value of 85 for T). They have considered a number of different learning methods of MLP's. The EPNet algorithm, an algorithm that uses evolutionary programming with partial training, achieved a normalised error of 0.06 and a backpropagation network (see also Crowder (1990)) obtained a minimum normalised error of 0.05. These results are slightly lower than the one obtained by the MLP network evolved by CERN. The number of connections of their backpropagation network (540) is comparable to that of the CERN-optimised backpropagation network (593), but the EPNet required significantly fewer connections (103) to achieve the result reported by Yao and Liu. This could among other be attributed to the experimental setup used by Yao and Liu. Specifically, the network was trained to predict the output of the time series 6 time-steps ahead. These results were then used to iteratively compute the output after 12 time-steps, then at 18 time-steps, and so on until a final prediction at 84 time-steps ahead was made.

It is clear from the generalisation results of the Mackey-Glass time series (figure 4.9) that significantly fewer orientated, ellipsoidal nodes were used by the OEBF network in comparison with the spherical Gaussian nodes used by the RBF network which is initialised by *k*-means algorithm. Figure 4.10 shows that initially less free parameters were required by the spherical Gaussian RBF network to obtain the same normalised error as the OEBF network. This could be ascribed to the fact that both spherical Gaussians and orientated, ellipsoidal Gaussians were at first equally suitable as regression variables of the target function.

However, as more nodes were used, the CERN algorithm soon outperformed the RBF network, even if the results were normalised with respect to the number of free parameters used. As illustrated in figure 4.6 and figure 4.7, the final normalised error obtained after the addition of 64 ellipsoidal nodes, was very small and still decreasing. The same trend cannot be seen for the RBF network. Thus CERN was, at each stage of OEBF network evolution, able to place and rotate the ellipsoidal nodes in an efficient way. Owing to the fact that the normalised error continued to decrease if more nodes were added to the network, it was felt that that the normalised error could be made almost arbitrarily small. A final normalised error of 0.0329 was obtained on the test set after 64 nodes had been added to the network. This is such an accurate fit that plotting the predicted values on top of the actual values of the test series would show only a single curve. To the authors' knowledge it is the lowest error reported on this particular data set (500 training points, $T = 85$) using either radial basis type networks or other methods. Saha et al. (1991) achieved a normalised error of 0.065 after the addition of only 40 OEBF nodes. At the same stage (40 basis nodes) CERN obtained an error of 0.0646. Moody (1989) reported a normalised error of 0.05, achieved with a RBF network containing 516 multi-resolution Gaussians.

Method	**NRMS** *
OEBF –CERN (64 nodes)	**<0.033**
ONRBF (40 nodes)	0.065
Multi-Resolution-Gaussians	0.05
EPNet **	0.06
MLP with BP **	0.05
MLP with CERN	**0.07**
Cascade with Correlation**	0.32
Cascade with Error Training	0.10

Table 4.1 Generalisation results of various algorithms on the Mackey-Glass Time series prediction problem for T=85 time-steps ahead. * based on test data ** reported in Yao & Liu (1997) for (T = 84)

4.5.2 Double-spiral problem

Figure 4.11 depicts the decision regions found by the MLP network for the Double-Spiral problem. Two interwoven spirals can be distinguished, although the classification boundaries are not very smooth. A likely reason for this phenomenon is that a MLP that had a linear output transfer function that uses RMS error training is not ideally suited to this particular problem as it contains highly intertwined class regions. It can be noted that there are more free parameters in this network than there are data points. These parameters were however not all optimised simultaneously. It is also not unusual for models of this problem to have more free parameters than data points, see discussion below. It has been found previously (Fahlman & Lebiere, 1990), that a standard backpropagation network cannot solve this particular problem. Successful solutions, although also containing imperfections in the derived decision regions, were obtained by MLP networks that required either specialised network architectures such as cascade correlation or multiple hidden layers, where each of the hidden layers was connected to all previous layers, or modified backpropagation training schemes (Fahlman & Lebiere, 1990). In addition to this the networks had hyperbolic tangent functions in the output nodes that made them more suitable for classification problems.

The solution obtained by the OEBF network that was evolved by CERN contains only two small irregularities. These can be seen at the bottom of the decision surface shown in figure 4.12. Moreover, the spirals of the training pattern circle the origin 3 times whereas the decision surface produced by the OEBF network (figure 4.12) extends the pattern slightly outside the target pattern (figure A.1). This phenomenon is owing to the fact that the first group of optimised nodes will try to find a more global solution to the problem and place large nodes lying inside one another to best match the near circular spiral pattern. One of the best reported solutions to the two-spiral problem has been achieved by the GCS (growing cell structure) algorithm of Fritzke (1994), that solved the problem with almost no visible discrepancies between the final learned decision surface and the target spiral pattern. For classification problems the GCS algorithm has the advantage that the error term used to decide where to place new nodes has, in addition to an RMS component, a component whose value depends on whether a training point is correctly classified or not. The number of free parameters of the two-spiral model derived by GCS is significantly larger. A total of 150 spherical Gaussian nodes were used by the GCS algorithm. As each 2-dimensional Gaussian is defined by two centre co-ordinates, a width as well as an outgoing weight, the derived model had a total of $4 \times 150 = 600$ free parameters. This is much greater than the solution obtained using CERN that had only 151 free parameters. This number was computed as follows. Each two-dimensional OEBF had 6 free parameters, two centre co-ordinates, two width parameters, an orientation and an outgoing weight. The network also had a single bias. The total number of free parameters is therefore $25 \times 6 + 1 = 151$. The Offset algorithm (Martinez 1992) has been used after applying an orthographic projection of the inputs, and has only 23 free parameters. However using such preprocessing to the Spiral input data is 'unfair' as has been argued by Saffrey & Thornton (1991). For example, if the data are converted to polar co-ordinates the Double-Spiral problem could be solved with just one threshold gate. It seems that CERN with it's orientated ellipsoids is also well suited for this data set. Results not given here are that axis-parallel and also spherical Gaussians optimised by CERN also fitted the Double-Spiral problem very well with few parameters. Thus the good fit seems to be more as a result of the optimisation algorithm than the type of basis nodes selected.

4.5.3 Abalone experiment

CERN successfully and consistently evolved very accurate MLP networks (figure 4.13) in comparison to those MLP networks trained by the advanced gradient descent technique, delta-bar-delta. (Results of the generalised delta rule (Zurada, 1992) were very similar to those of the delta-bar-delta method.) It should be noted that the structure of the backpropagation network trained by this technique was identical to the MLP network evolved by CERN and should therefore in theory achieve identical performance results. In contrast to this, training the backpropagation network with the delta-bar-delta technique for a further 140000 epochs to give in total 150000 epochs, did not improve the networks performance. The difference arose because the backpropagation network more often got stuck in sub-optimal minima than CERN.

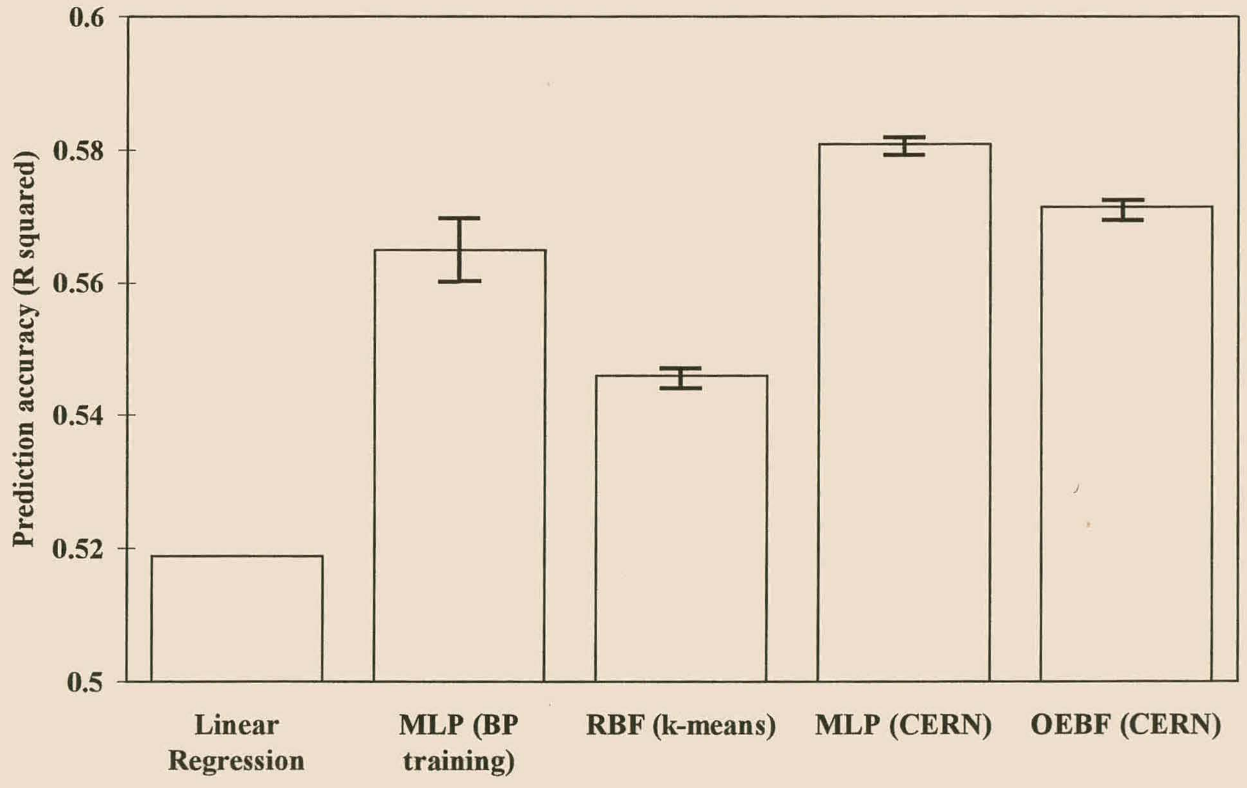

Figure 4.13 Generalisation performance on the Abalone data. The mean R^2 value on the test data over four independent experiments is shown. The error bars indicate the standard deviation of the R^2 values over the four runs.

While evolving the OEBF network, CERN was able to place orientated, ellipsoidal nodes more effectively than the *k*-means clustering algorithm was able to do for the RBF network.

Significantly fewer nodes were consequently required to obtain good results. The addition of five basis nodes to the OEBF network resulted in better classification than the classification accuracy achieved by the RBF network that contained 100 spherical basis nodes.

4.6 Conclusions and significance of Chapter 4

- It was demonstrated that CERN can evolve regression nodes with considerably higher efficiency than the conventional DE algorithm. This is owing to the fact that it employs a combinatorial search method to select the best new individuals. After each iteration, if a given node's extracted feature becomes more useful to the output node, it will be retained and the resulting group of nodes will have improved, even when all the other nodes of the group have deteriorated owing to crossover or mutation. In other words, if a given trial node contributes to the improvement of some combination of nodes, it will be preserved in the new population. This is not the case in the conventional DE algorithm. In this case the group of nodes is encoded as one individual. This individual (which represents a group of nodes) will be retained if and only if the particular individual, as a single entity, has better fitness than its predecessor.
- The extra floating point operations required for the combinatorial search do not contribute significantly to the total number of operations required. This is provided that the number of nodes to be optimised simultaneously is not much larger than 10 and the data set has more than 2000 data points and 5 dimensions.
- Based on benchmark problems it has been demonstrated that the novel CERN algorithm can be effectively utilised in the context of an algorithm that evolves a network by stepwise addition of groups of nodes. Moreover, analysis of a real-world problem indicated that CERN could also be used to successfully optimise an entire neural network. Besides these two functions CERN could also be used in the context of other algorithms. One such example is as an algorithm that finds the best group of *g* neural nodes, but then only adds the single best node to the existing neural network. In effect this means that with regard to network performance the algorithm will be continually looking $(g - 1)$ nodes ahead and then dynamically allocating the neural nodes to the existing network architecture based on this information. Such an algorithm will amongst other be developed in the next chapter.
- Of the three problems that were studied, the CERN algorithm proved as capable as advanced gradient descent techniques in the task of optimising neural nodes. The oriented, ellipsoidal

basis nodes used by CERN to evolve OEBF networks were found to be superior to the spherical basis nodes obtained by k-means clustering for the radial basis function networks, in terms of both accuracy and the number of nodes required to achieve good network performance. In particular CERN achieved the lowest reported error to date on the Mackey-Glass time series. Moreover, the conventional backpropagation technique is prone to get stuck in local optima and could solve the Double-Spiral problem only after significant adaptations to the network. Furthermore, Yao and Liu (1997) reported that backpropagation in particular has problems in dealing with compact networks. They found that it either trains very slowly or sometimes could not find a near optimal solution at all.

- CERN is a purely evolutionary approach and therefore has the advantage over other forms of optimisation techniques, such as gradient descent or evolutionary algorithms with partial training, that it can be applied directly to a network which uses any type of node. It is able to adapt parameters for which there exists no gradient information. It could for example be used to place and optimise hyperrectangular basis nodes with a threshold activation, similar to the fuzzy hyperboxes found by the fuzzy min-max neural network of Simpson (1992), except that hard decision boundaries may now be enforced.

- One of the limitations of the CERN algorithm is that, like other evolutionary algorithms, in each iteration it uses the complete training set. It is thus computationally expensive and is more suited to off-line learning.

- Another limitation of CERN is that it requires the network to have a linear output node and cannot use a hyperbolic tangent output node. In the context of optimising a multilayer perceptron it is consequently more capable of solving problems with a continuous output than classification problems.

Chapter 5 CERN with clustering and copying of individuals

5.1 Introduction

This chapter discusses various enhancements to the algorithm described in chapter 4. In the experimental section the effect of these on the CERN training speed and quality is examined. Comparisons were not made explicitly to other algorithms and thus the pragmatic reader can skip the experimental section of the chapter for later. Chapter 6 will make comparisons to standard algorithms on various data sets. The main novelty introduced in this chapter is the active support of niches in the CERN algorithm. This is achieved by forming sub-populations using clustering on the chromosome vectors. Using the chromosome vectors of individuals sharing the same cluster, new trial individuals are formed. The parent individual is also part of the cluster. This results in faster convergence of the algorithm. A further advantage is that this method allows for a sensible way to have two or more node types in the same population. Their chromosomes are strictly confined to different clusters and do not interact. However, nodes of different types can still compete in the selection process and be part of the same group. (Recall that a group consists of g nodes that are to be optimised simultaneously). It is therefore possible to optimise g nodes, where the type of each node, such as a radial basis node or a summation node is also a variable that is automatically selected. For example, when specifying that five nodes are to be optimised, four summation nodes and one ellipsoidal basis node could be found in the same layer.

A further enhancement made to the algorithm is that entire nodes are allowed to be copied into different groups. This occurs with a user-defined probability (XR) and also results in faster learning. The algorithm responsible for the addition of nodes to the network was also improved. In this chapter the CERN algorithm is used in the context of an algorithm that finds the best group of g neural nodes, but then only adds the single best node to the existing neural network. In effect this means that with regard to network performance the algorithm is continually looking $(g - 1)$ nodes ahead and then dynamically allocates the nodes to the existing neural network architecture based on this information.

The details of the changes are given in the following sections. Their effect on the learning speed is shown in several experiments on various real world and synthetic data sets. All the data sets used in this chapter are described in Appendix A. In these experiments the effect of the various parameters of the CERN algorithm was also examined thoroughly. A robust set of default parameters was found. The population of mixed nodes could automatically find the optimal node types, with respect to the training data, even if this implies allocating different node types in the same hidden layer.

5.2 Searching while looking ahead

The algorithm that adds nodes to the existing layer was adjusted to optimise a group of g nodes simultaneously. After the genetic algorithm meets a stopping criterion, such as a predefined maximum number of iterations, it will add the g nodes to the new layer. However, only the first added node is fixed and the other nodes can still be changed in the subsequent stages. In this way the algorithm effectively looks g-1 steps ahead until it is within reach of the final number of nodes to add. Instead of only decreasing the lookahead in the final stages, the program decreases the lookahead horizon at a constant rate. The user specifies the total number of nodes, how many nodes the algorithm must look ahead initially and at what rate the lookahead horizon should decrease. For example, add 20 nodes, $g = 8$, decrease g by one every 2 steps. This will result in the algorithm adding the nodes in 20 stages during which g will decrease in the following sequence: 8, 8, 7, 7, 6, 6, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 1, 1.

At each step the added node is the node of the group with the highest fitness. From this group the corresponding individual is then deleted. From all the other groups of the population the algorithm will delete a randomly-chosen individual. After the deletion the algorithm optimises only g-1 nodes. If needed a new node is created in every group. This implies that when g stays constant, a node is first deleted and then another node is added to the group again. It can be seen that this is necessary when there is only one individual in each group. The population might have converged too much. Thus, when the next node is to be optimised, the population is not sufficiently diverse and the chromosome vectors need to be reinitialised. For the case of $g = 1$, deleting and adding the nodes is equivalent of reinitialising the population. For $g > 1$, the deletion and addition process effectively reinitialises one node in each group.

5.3 Forming the sub-populations

As was discussed above, clustering can actively support niches in the population. Thus before each iteration, the chromosomes of the population $\{v\}$ are clustered. Any clustering algorithm could be used for this. In the implemented program the k-means clustering algorithm (Duda & Hart, 1973) was used for its ease of implementation and use. The number of clusters was chosen to be $l \times g$, where l is user-defined (g is the number of nodes to be optimised). For each parent individual in the current population the cluster membership is determined. The new trial vector is then formed with the chromosome vectors of generating individuals chosen randomly sharing the same cluster. Following this the parameters of the parent individual are copied over.

Sometimes a cluster does not have enough individuals from which to choose. Also mixing between different clusters or niches should be limited but not completely disallowed. Therefore the three generating individuals *r1, r2, r3* are chosen with a probability B to be drawn directly from the cluster of the parent. This parameter will be called the clustering strength. If the random test fails or if the cluster has less than three members, the individuals will be chosen randomly from the entire population. Of course, the members of the different clusters also interact in other stages of the algorithm. In each group of the population, members of different clusters compete in the selection process for their right to survive. CERN will thus sustain niches in two manners.

Firstly the sub-populations encourages the formation of new individuals that are likely to be part of the same niche as their parents. Secondly co-operation of different niches is enforced in the selection process. For example, in a basis function network, different (radial) basis nodes need to cover different regions of the input. After several iterations the basis nodes of each group are likely to consist of nodes having different centres and widths. The nodes from various groups with similar locations will likely be part of the same cluster. At each iteration similar nodes usually share their parameters or form new parameters that are combinations of existing ones. On the other hand, nodes with very different locations are not likely to interact as much. This is intended, since they are performing different tasks and will have little to 'learn' from each another.

5.4 Nodes of different types in the same population

Nodes not belonging to the same clusters have relatively little interaction of their parameters. If nodes of different types are strictly in separate clusters, they could be allowed to be in the same population without their unlike parameters disrupting one another. To facilitate this, nodes are separated into categories or *species*, one for each node type. Thus if basis nodes and summation nodes are to be in the same population, there would be one species for basis nodes and one for the summation nodes. In each species the clustering and formation of new trial individuals occurs completely independently. For each species there will be $l \times g$ clusters, instead of $l \times g$ clusters for the entire population. When the population is initialised, each node has an equal chance of becoming a member of either of the node types. The same applies when new nodes are created (after a node is inserted into the neural network).

The selection of the individuals in the groups proceeds in the same manner as before. Thus a group of size 5 can consist of 3 basis nodes and 2 summation nodes. Their offsprings will also be 3 basis nodes and 2 summation nodes. Of these the best set of 5 individuals are selected. These can again be in the same proportion as before or the composition of the group could be different (e.g. 4 basis nodes and 1 summation node).

Some preliminary experiments on several data sets were conducted. The algorithm successfully found the kind of nodes that worked well for the problem. Unfortunately, individuals of other node types tended to become very scarce or even become extinct. There are several reasons for this. It could be as a result of a certain node type not being as useful as the other node types for the specific data set. Another reason is that their parameters might take longer to be fine-tuned. In either case the extinct node type might nevertheless become useful in later stages of the search; therefore an effort should be made to keep some members of each species in the population. When the initial population contained more nodes of one type than of another, this type had a better chance of surviving than before. Nonetheless this created other problems. Which node type should be given the best chance and how big an advantage will be required? A better solution is to reserve parts of the population for each node type. This can be done by reserving some of the groups for only one type. The offsprings will always be the same type as the parent. The partitioning ensures that the population maintains some nodes of each type. It was decided that one half of the population would be mixed and that the other half would be divided into homogenous parts, one for each type. This is illustrated in figure 5.1

The chromosomes of the nodes of the various partitions can still interact through the formation of new individuals. A further way of strengthening the competition / co-operation of the nodes is given in the next section.

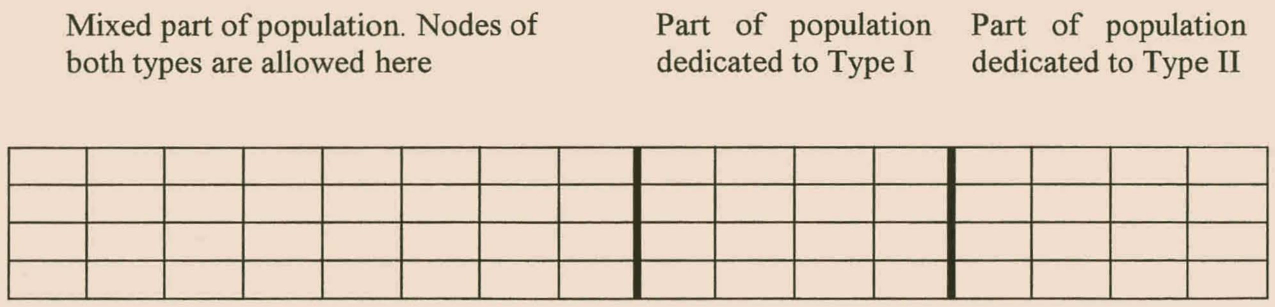
Mixed part of population. Nodes of both types are allowed here
Part of population dedicated to Type I
Part of population dedicated to Type II

Figure 5.1 The partitioning of a population, for two node types.

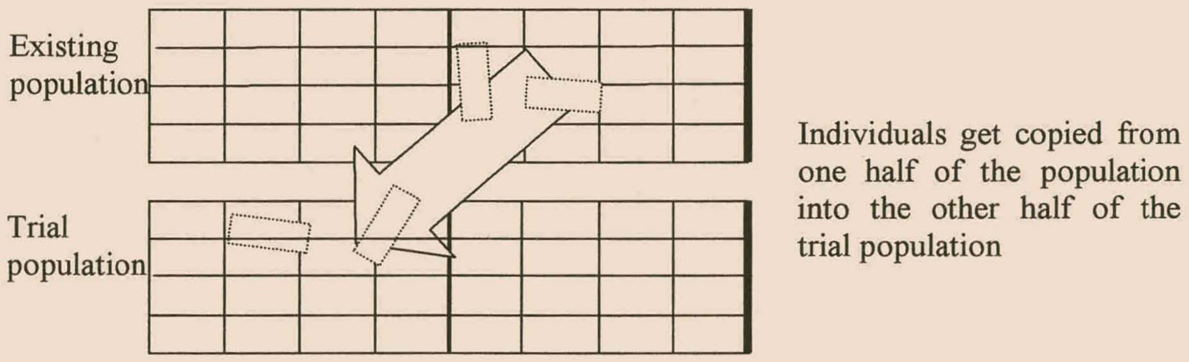
Existing population
Trial population
Individuals get copied from one half of the population into the other half of the trial population

Figure 5.2(a) The copying of complete individuals in the CERN algorithm for a homogeneous population.

5.5 Copying of complete individuals to other groups

In each group of the population various nodes co-operate in order to explain the variance of the target output. For such a group it is possible that one or more individuals from a different group could be very useful. This is allowed for when complete individuals are copied from one group to another. The individuals (nodes) get copied into the corresponding trial group. In the algorithm no new space is allocated and therefore each copied node will replace some other trial individual. The copied nodes take part in the combinatorial selection scheme just like the remaining newly formed trial nodes. This way the copied individuals, just like all the other trial individuals, have the opportunity to be selected into the current population. Copying the individuals into the trial population is better than copying the nodes directly over nodes in the existing population, as this might disrupt well-working groups. The copying process for a population with a single node type is illustrated in figure 5.2(a). It can be seen that the individuals get copied from one half of the population in the other. Thus one half of the population does not receive copied nodes. This prevents certain nodes being duplicated too often throughout the entire population, which could cause too early convergence of the genetic algorithm. The two halves of the population continue to interact in the usual way when forming new individuals.

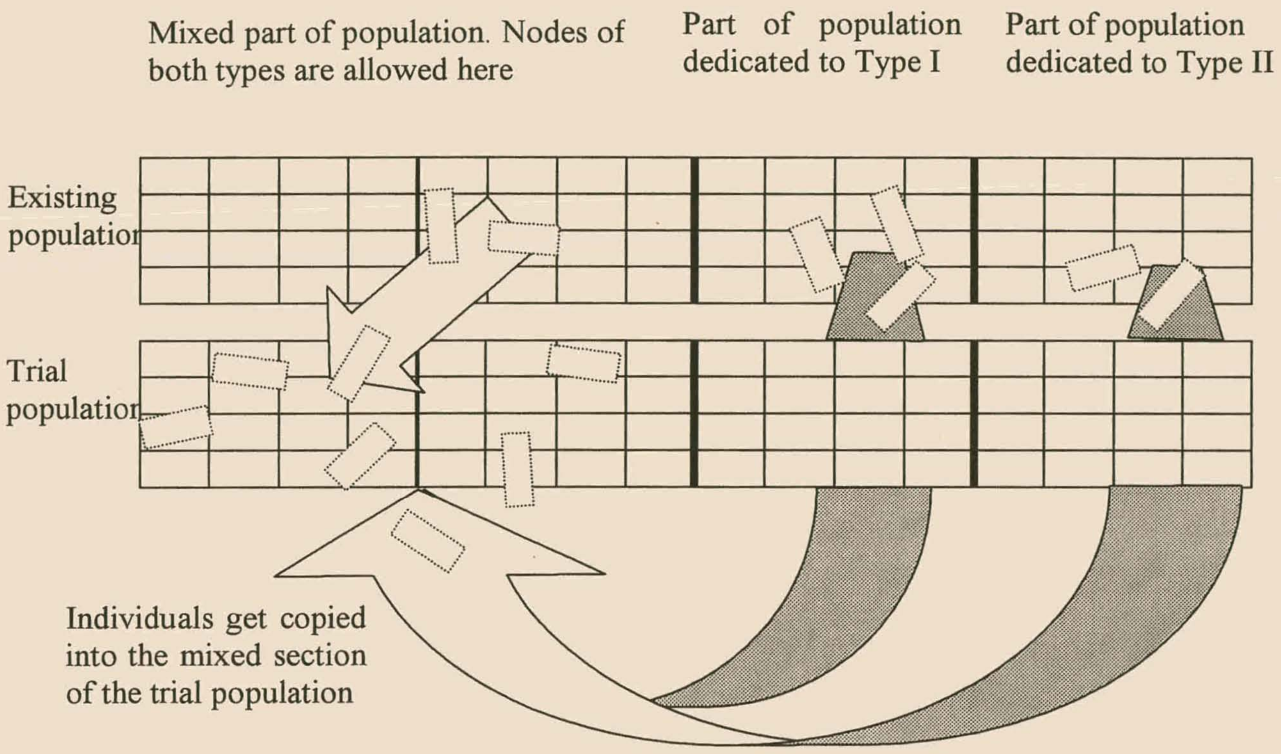
Mixed part of population. Nodes of both types are allowed here
Part of population dedicated to Type I
Part of population dedicated to Type II
Existing population
Trial population
Individuals get copied into the mixed section of the trial population

Figure 5.2(b) The copying of complete individuals in the CERN algorithm for mixed nodes.

Figure 5.2(b) illustrates the copying process for a population consisting of two node types. In addition to copying from one half of the mixed section into the other half, as for the homogeneous population of figure 5.2(a) another duplication process occurs. The nodes are copied from sections consisting of only nodes types I and II, to the trial population of the mixed section. Each individual that acts as a potential source has a user defined probability *XR* to get copied into another group. The influence of XR will be considered later. These copying schemes are based on heuristics and not necessarily the best routines of performing the copying. Different heuristics might also be successful, but were not conceptualised.

5.6 An overview of the algorithm

```
Initialize population()
    Allocate the $N_g$ groups .
    For each group allocate $g$ individuals.
    Divide the population into a mixed section, as well as section for each node type.
    Randomly chose a node type for each for each individual in the mixed section.
    Initialize the parameter vectors for each individual

For 1 to maximum nodes to add do:
{       For 1 to maximum iterations do:
        {       Generate a trial population()
                  Form clusters of the parameters of the entire population
                  For each individual, called parent individual generate a trial vector using
                  individuals from the same cluster with probability B each.
                  Copy over the parameters of the parent individual with probability CR each.
                  Copy over whole individuals with probability XR as shown in figure 5.2.

                  Select a new population ()
                  Translate the genotype of the individuals in the trial population (their vector
                  parameters) into their phenotype (the nodes).
                  Compute their activation to all the data points.
                  Update the covariance matrix.
                  In each group of the population use the combinatorial search to pick the best $g$
                  nodes. The corresponding individuals get placed in the new population.
        }
        Add a new node to current layer ()
        Find the single best group of nodes in terms of explained variance( $R^2$). Pick the best
        individual from this. Add the corresponding node to the current layer.

        Update the population ()
        For each group delete a random individual; except for the best group delete the individual
        corresponding to the node that was added to the layer.
        Groups now contain $g-1$ nodes. If necessary allocate and initialize a new node in each group.
}
```

Figure 5.3 The CERN algorithm with clustering and copying over of individuals.

Various steps were added to the algorithm in this chapter. The resulting CERN algorithm is no longer a straightforward genetic algorithm with a combinatorial selection scheme. The algorithm including the node addition algorithm is summarised in figure 5.3. The inside loop that is performed for each added node will from now on also be called an evolution cycle.

5.7 Specifications of Experiments

The effects of all the enhancements to the algorithm were examined in several computer experiments. The parameters F, CR and XR were also tuned in these experiments. For each run the user specified a base number of iterations. The evolutionary cycles during which more nodes were optimised usually required more iterations than those for which only 1 or 2 nodes were optimised. Therefore, for each cycle, the number of iterations during which the population was optimised was made dependent on the number of nodes that are being optimised (g) and the number of nodes that was already in the networks (p). However, it was decided that the number of iterations should be at least 20% of the base iterations for each stage in which a new node is added. Thus for an evolutionary cycle the algorithm ran for

$$\text{iterations} = \text{base iterations} \times \max\{g/(g+p), 0.2\} \qquad \text{Equation 5.1}$$

In all the experiments of this and also of subsequent chapters, the number of clusters was set to $3g$ viz. $l=3$. Thus, for each member in the group, there were three clusters available. This number was found to be efficient in some preliminary experiments. The effect of the clustering strength B was not investigated and was always set at 0.9. This depends very much on the efficiency of the clustering algorithm and can be seen to be part of it. In all experiments the k-means algorithm was run for a maximum of 30 iterations, even if it means that complete convergence was not guaranteed. In this way the clustering algorithm did not use a major share of the CPU time.

The effect of the population size and maximum number of iterations were also not examined systematically. Obviously the fitness increases with both. It is possible to configure them to bring about shorter training times. This was not attempted, because it is very much problem-dependent. The population size was changed for several additional experiments from 60 to 150 groups. This was to ensure that different optimal settings to those obtained in the described experiments are also applicable for a greater population size. The results of the experiments were in agreement with the results of the experiments with the smaller population size and are not discussed further.

5.7.1 Effect of clustering, strategy, F and CR settings on training

The aim of the first experiment was to observe the effect of the strategy (DE1 or DE2), as well as the effect of the clustering method, on the efficiency of the CERN algorithm. The investigation also shows the effect of various CR and F values.

For the Pine data set (see Appendix A), a mixed neural network and a network with summation nodes were evolved with the CERN algorithm. For the Mackey-Glass time series (Appendix A) a network with mixed nodes and one with axis parallel basis nodes were optimised. To all the networks a total of seven nodes were to be added to the hidden layer, looking four steps ahead. These networks were evolved with CERN using a variety of different settings: For each of the four combinations, clustering or no clustering, together with the greedy (DE2) or the more conservative mechanism (DE1) to generate the new vectors, various settings of F and CR were examined. The settings were given labels from 1 to 9. These values as well as other learning parameters are summarised in tables 5.1 (a) and (b).

Experiment	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
F	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
CR	0.25	0.25	0.25	0.5	0.5	0.5	0.75	0.75	0.75

Table 5.1(a) Values of F and CR for the nine experiments performed for each of the strategies.

XR	0.05
B (where applicable)	0.9
l (where applicable)	3
g	5
N_g	60
Base iterations	140
Number of nodes added	7

Table 5.1(b) The parameters of CERN used in the nine experimental setups for each of the four networks.

Each experiment was repeated with three different initial random number seeds, because the CERN algorithm is a stochastic search. This gave rise to $4 \times 9 \times 3 = 108$ runs for each of the four neural networks, 432 runs in total. For each experiment, the parameters B (if applicable) and XR were kept constant at 0.05 and 0.90 respectively. All the runs were stopped after three evolution cycles. In other words after three nodes were permanently added to the current network. All the fitness scores were recorded and the R^2 after two and three evolution cycles of the various runs were compared to one another. The population size was kept constant at $N_g = 60$ groups of 5 nodes. The base number of iteration was set to 140. Consequently as specified by equation 5.1 the first evolution cycle ran from 1 to 140 iterations, the second from 141 to 260 iterations and the third from 261 to 350 iterations.

5.7.2 Determining the influence of copying of individuals on training speed

After researching the effects of the strategy, CR and F, the effect of the XR parameter was examined. For this XR was ranged from 0 to 0.5 in 18 steps, for the four networks. However, in these experiments only the best found strategy (DE1 or DE2) as well as best CR and F settings were used. The experiments were also repeated with three different seeds for the random number generator. This resulted in another $3 \times 18 \times 3=162$ runs.

5.7.3 Effect of the extend of the lookahead on testing accuracy

One preliminary experiment was made, previous to any other experiments in the chapter and will be described first. The aim was to see whether looking ahead improved the performance in comparison to simply optimising one node at a time like the cascade correlation algorithm (Fahlman & Lebiere, 1991). For the Abalone data set six summation nodes were evolved simultaneously in the last chapter. On the same data set with the same specifications six summation nodes were added one at a time. For the latter run the algorithm was given more iterations and a bigger population to ensure a good node was found each time. The experiment was repeated three times. These six nodes added one at a time gave an average R^2 on the test set of 0.56 as opposed to 0.57, for optimising the nodes simultaneously. (The training R^2 was also higher for the simultaneously optimised network). Similarly, for the Mackey-Glass time series, 12 ellipsoidal nodes were optimised, once using a lookahead of 6 and once without a lookahead. The experiment was also repeated 5 times with three different initial populations. The population size was 300, with F = 0.4, CR = 0.4, running for 300 iterations. On the test set the network using optimisations with lookahead had on average only 0.67 times the sum-squared error, compared to the networks using no lookahead. Based on a simple ANOVA test, the result was significant with a 99% probability.

The following can therefore be concluded. For some data and a given network size, the approach using lookahead gives a significantly better fit on the testing and on the training data than can be obtained by the same neural network evolved one node at a time. This is not surprising and was already explained in the previous chapter. However it still remains to be seen how much lookahead is required. A greedy technique resulted in poorer networks for a fixed size. Nevertheless, the effect of allowing the greedy mechanism to add more nodes needs investigation. It is possible that a network, evolved one node at a time, might eventually be bigger, but have as good or better a fit as one that was evolved using lookahead. This is because the lookahead procedure fits more parameters simultaneously and thus also has a higher probability of overfitting the data. If this should happen consistently, it would mean that the error function that one minimises, is not the one that also minimises the generalisation error. One would have to use a different error measure, such as the minimum description length (Rissanen, 1978) or add other penalty terms to the solutions.

To examine the generalisation of the networks built with different lookahead schemes, an experiment was performed on five different data sets. For each data set three network types were built using various settings of initial g viz. $g = 1, 3, 5, 7$. The three network types consisted of a single hidden layer and nodes of either summation nodes, axis-parallel ellipsoidal nodes or a mixture of both types. For each of the networks the error on the test set is recorded every time a new node is added to it. All the settings like CR took on the default values as found in the previous sections. In all experiments g (the lookahead horizon -1) was decreased gradually to one during the allocation of the first half of the nodes. For example, for 80 nodes g was 1 after 40 nodes had been optimised and placed in the hidden layer of networks. The other details for each of the experiments, as well as the data sets used are given in table 5.2 below. For some of the experiments where $g = 1$, extra nodes had to be allocated, because the testing error had not reached the lowest point.

Data set (details in Appendix A)	*Network types*	*Number of nodes*	*Base iterations*	*Number of groups Ng*			
				g = 1	*g = 3*	*g = 5*	*g = 7*
Mackey-Glass	OEBF & mixed Summation	80 110	200 400	500	300	240	180
Kin-8nh	All types	15	400	500	300	240	180
Kin-8nm	All types	25	400	500	300	240	180
Sonar	All type	25	400	500	300	240	180
Furnace-2 selected	All types	15	400	500	300	240	180

Table 5.2 Learning parameters for the experiments to determine the effect of the node-types and of the initial lookahead horizon.

Although the population for the runs where only a few nodes are optimised simultaneously should converge more rapidly, the number of groups, N_g, was decreased as the group size g was increased. This was done so that the total number of individuals N_p ($g*N_g$) was more balanced throughout all the experiments.

5.7.4 Mixed node types

Networks that are made up of different node types can have an advantage over networks having nodes of only one specific type. The hyperbolic summation nodes could for example fit the global trends in the data, while the ellipsoidal basis nodes can fit trends of a more local nature. Whether this can be done successfully is to be examined on several data sets. The experiments described in the previous section also made it possible to examine the generalisation capabilities of the different node types, including the mixed node types.

5.7.5 PRESS statistic

Each time a new node was added to the networks the PRESS statistic of the output node was recorded. Given some independent variables, this statistic is equal to the leave-one-out cross-validation error computed explicitly (Myers, 1990). The PRESS statistic is typically used for linear regression, but it has also been employed in non-linear regression (Schaal & Atkeson, 1998). Experiments were performed to determine whether the PRESS statistic gives an indication of overfitting for the non-linear networks used by CERN.

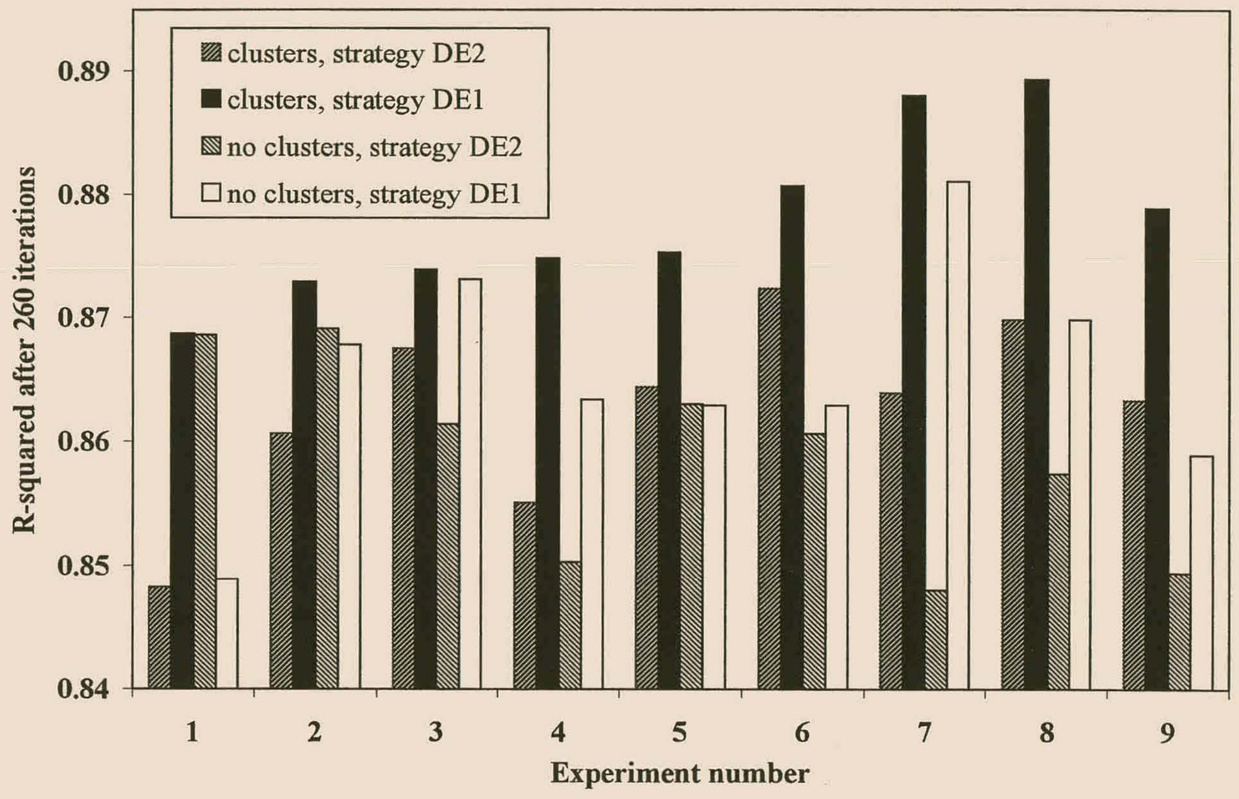

Figure 5.4(a) The R^2 fitness values on the Mackey-Glass time series training data during the optimisation of ellipsoidal basis nodes using different learning settings. Refer to table 5.1(a) for the settings of the 9 experiments.

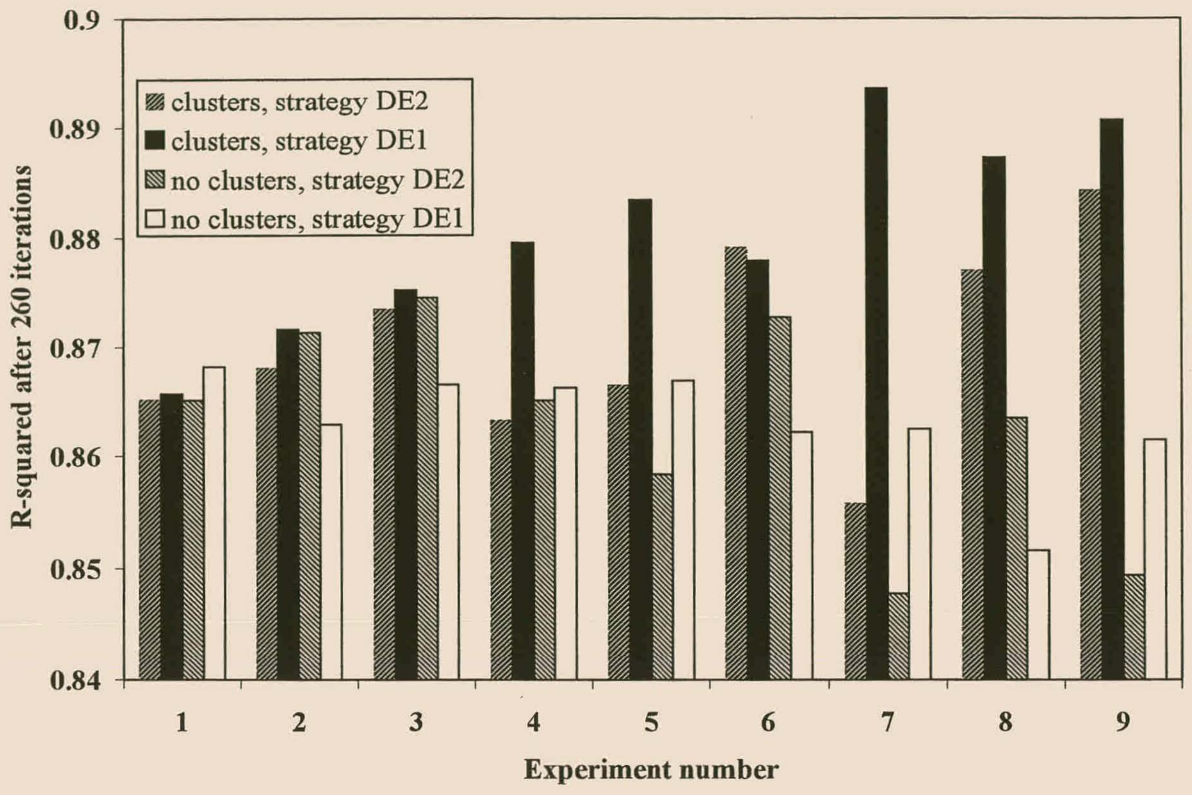

Figure 5.4(b) The R^2 fitness values on the Mackey-Glass time series training data during the optimisation of mixed nodes using different learning settings. The settings of the 9 experiments are given in table 5.1(a).

5.8 Results of experiments and discussion

5.8.1 Clustering and strategy

Figures 5.4(a)-(d) show the effect of clustering and the strategy for the various CR and F settings. The figures show the R^2 fitness values of the best groups after 260 iterations or after the second evolution cycle has finished. The plotted value for each experimental setup is the average of the runs with three different random seeds. Figure 5.4(a), summarising the neural networks with basis nodes for the Mackey-Glass time series, shows that for almost all settings clustering was superior to no clustering and that the conservative strategy DE1 was better than DE2. In figure 5.4(c),summation nodes for the Pine data, clustering is again superior to no clustering, while the strategies DE1 and DE2 are roughly equally efficient. In figures 5.4(b) and 5.4(d) the clustering approach is again superior to the no-clustering approach for almost all settings. The results for the last two networks are obvious. Evolving mixed nodes without clustering, has disadvantages, because chromosomes of different types can interact without taking into account that they describe different node types. The approach using clustering and strategy DE1 was clearly the most robust. Figure 5.5 plots the unexplained variance of summation nodes on the Pine training data against the iteration number for two setups. The first one is CERN using the DE1 strategy with clustering mentioned above for F = 0.5 and CR = 0.75. The other is DE1 without clustering with the best performing parameters of F = 0.5 and CR = 0.5. One can see that clustering indeed made a significant improvement to the CERN algorithm.

Clustering supports niches in the population. Consider for example some ellipsoidal basis nodes that cover a certain portion of the data. These nodes are likely to be in the same cluster. Thus they will mostly result in trial individuals similar to themselves, that also cover these data points. The cluster can thus specialise to explain the variance caused by these data points. In comparison, an algorithm that does not use clustering, trial individuals will generally have parents that cover completely different parts of the data. This will usually not result in useful individuals. Consequently learning will proceed slower. For summation nodes a similar argument will hold.

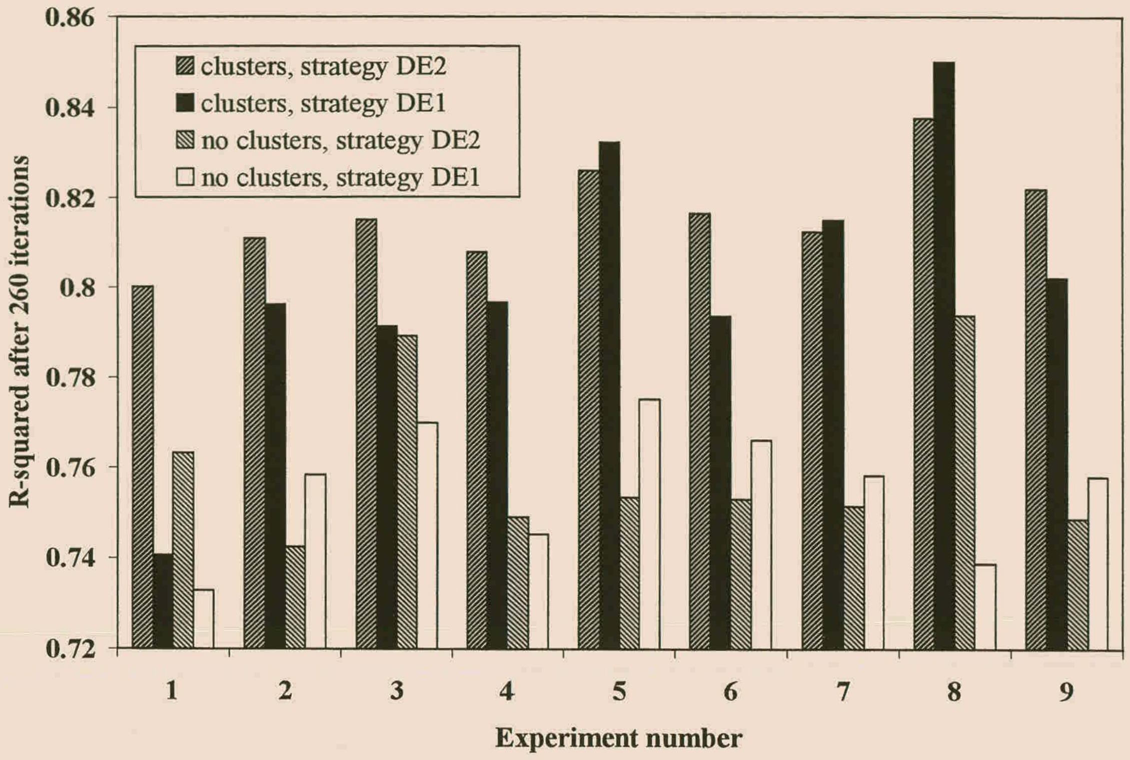

Figure 5.4(c) The R^2 fitness values on the Pine training data during the optimisation of summation nodes using different learning settings. The settings of the 9 experiments are summarised in table 5.1(a).

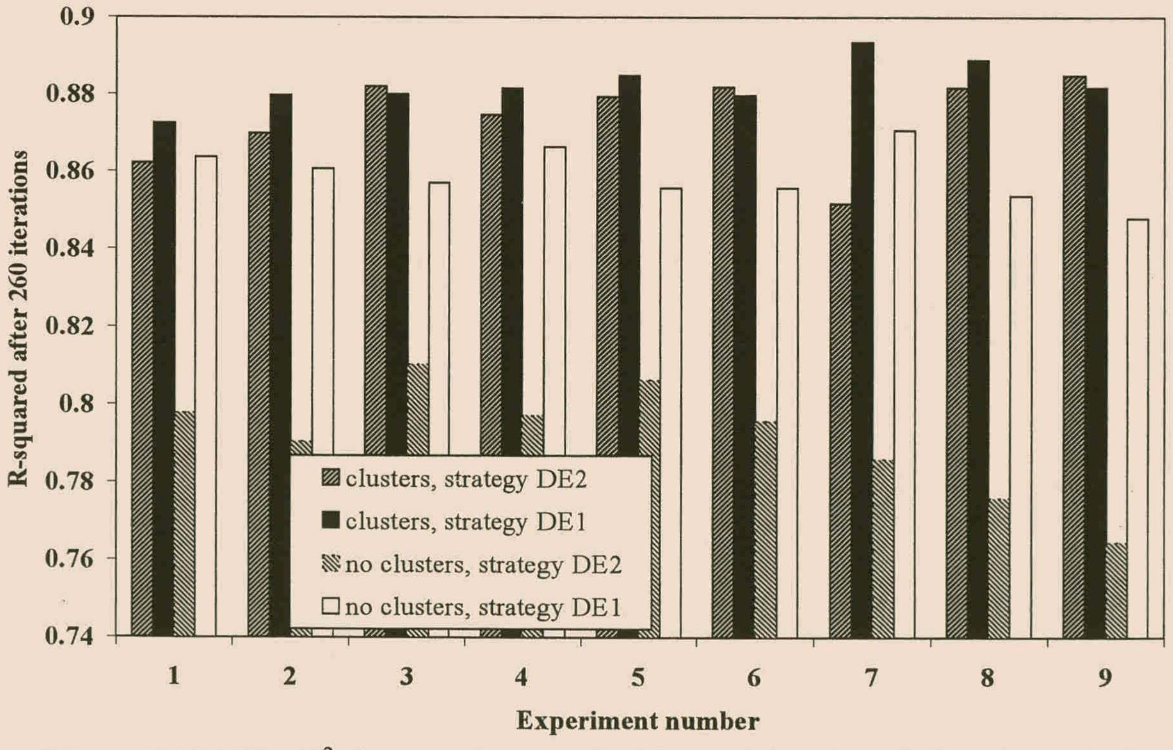

Figure 5.4(d) The R^2 fitness values on the Pine training data during the optimisation of mixed nodes using different learning settings. The settings of the 9 experiments are given in table 5.1(a).

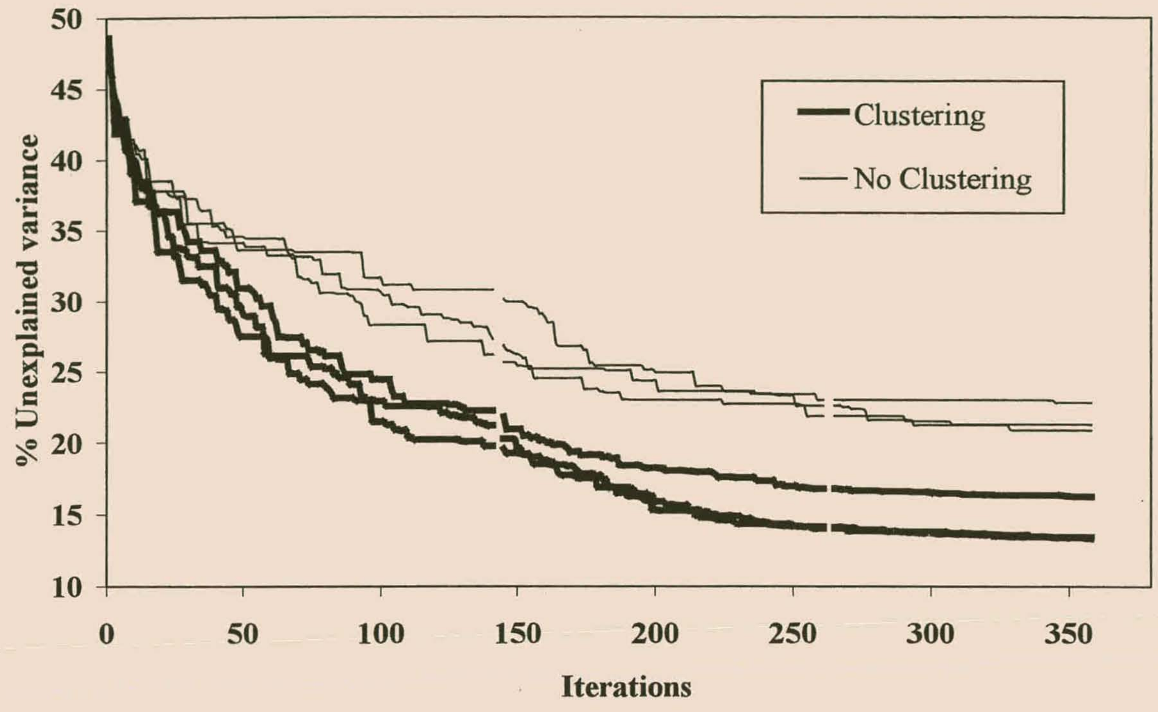

Figure 5.5 The training error on the Pine data set (summation nodes) for CERN with and without clustering.

5.8.2 Tuning of F and CR

Next a value of F and CR is needed that gave good training results for all of the four neural networks. The fitness scores after 260 iterations are shown in figures 5.4(a)–(d). For each of the four problems CERN with clustering and strategy DE1 reached the highest or a high fitness values for the experimental settings 8, that is for F = 0.5 and CR = 0.75. An inspection of the variances over the three different seeds was also undertaken. The variances showed that the DE1 strategy with clustering had in general the lowest variances. Thus this strategy together with F = 0.5 and CR = 0.75 was chosen as the default setting of CERN and was used in all succeeding experiments unless mentioned otherwise.

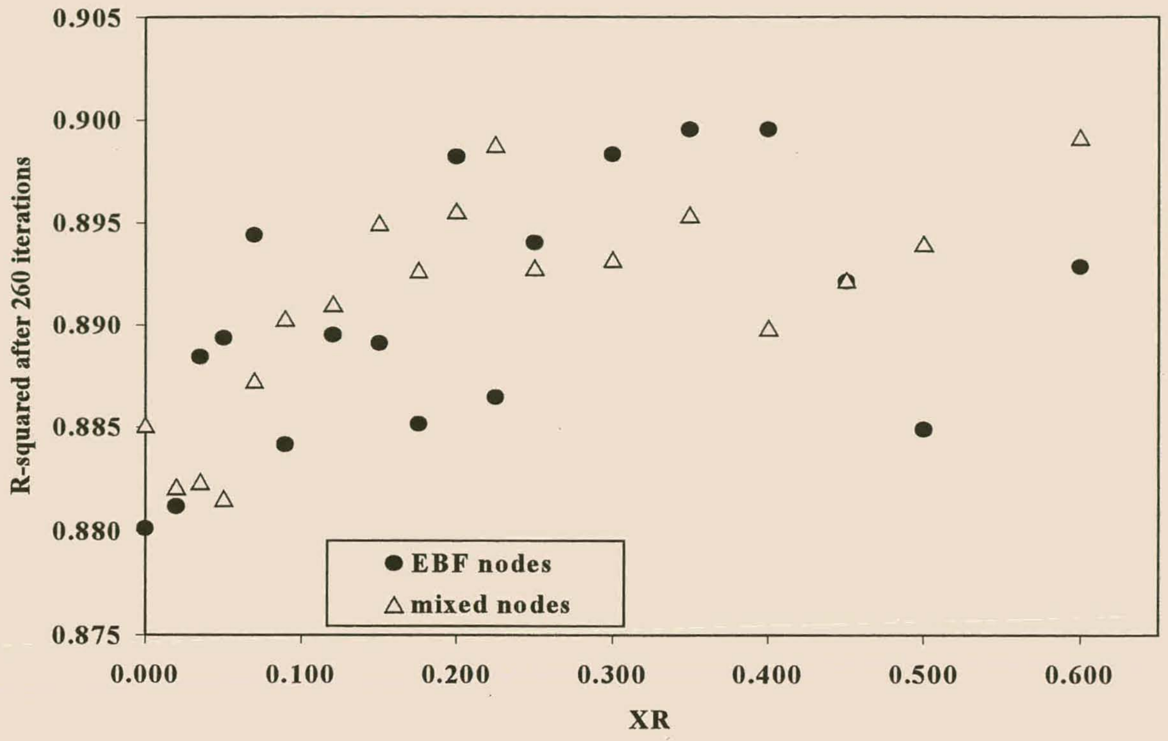

Figure 5.6(a) The R^2 fitness values on the Mackey-Glass time training data during the optimisation of mixed and ellipsoidal basis nodes vs. XR.

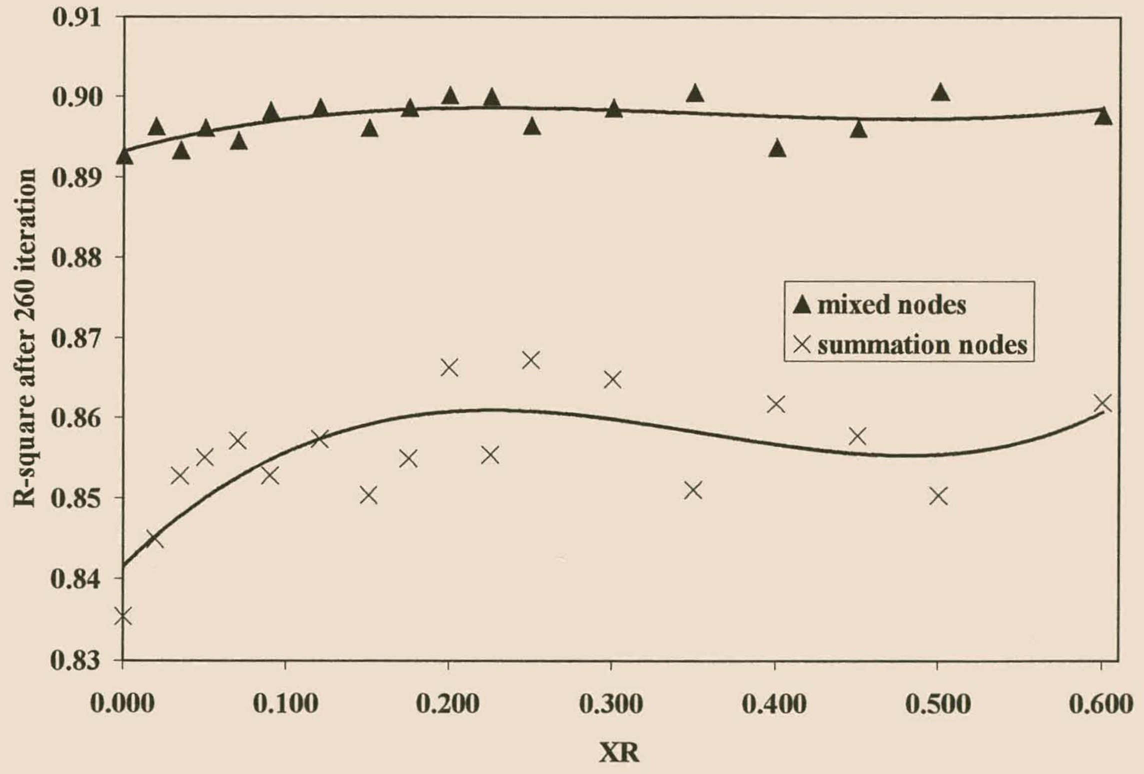

Figure 5.6(b) The R^2 fitness values on the Pine training data during the optimisation of mixed and summation nodes vs. XR.

5.8.3 Copying individuals and tuning of XR

Finally the effect of copying of individuals from one group into the other group as well as that of the setting XR were examined. The fitness score after 260 iterations is plotted against the XR in figure 5.6(a) and 5.6(b). A polynomial trendline was added to the second graph, making the trend more visible. The first thing one notices is that for all three networks the fitness with XR between 10% and 30% was much higher than that with no copying of individuals (i.e. XR = 0), and also higher than very little copying (XR < 10%). Also in this range the fitness was consistently high. It should be remembered that each point represents an average over three runs. It was thus decided to use an XR value of 20% as default. It should be kept in mind that these experiments were conducted for 5 nodes in each group. When there is for example only one node per group, copying over whole individuals into different groups, simply duplicates the same group. In this case no new combinations are created and the copying is probably not productive. It was therefore decided to set the copy over probability of individuals (*XR*) to a default value of $5 \times (g-1)/100$.

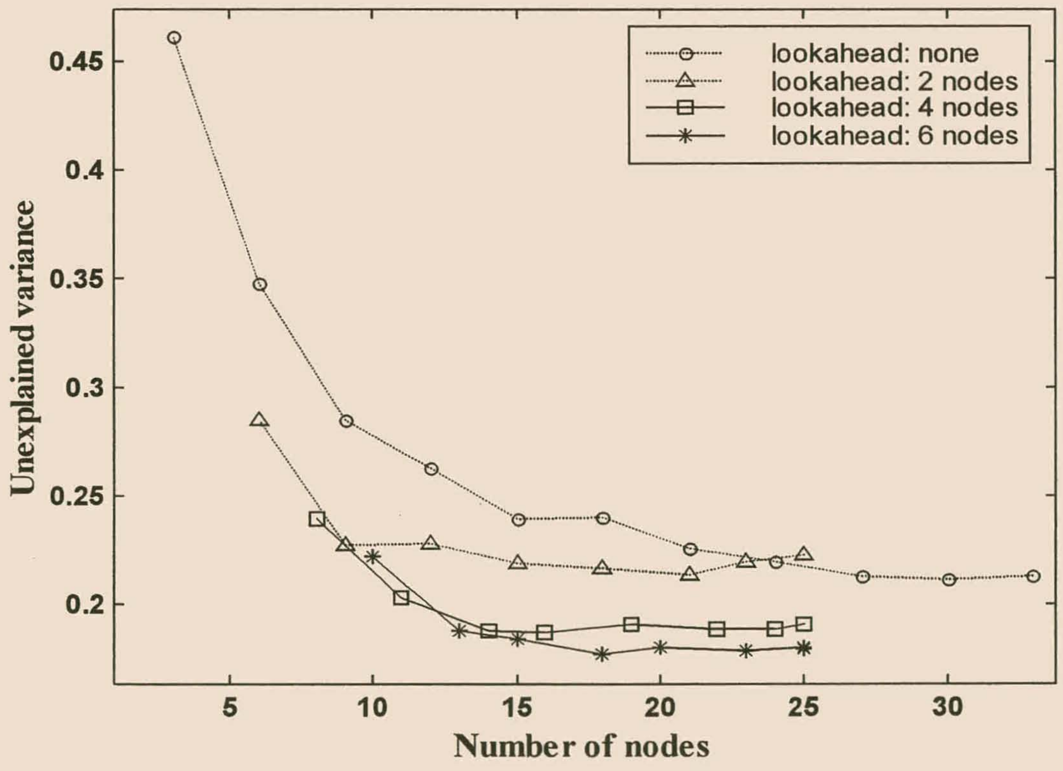

Figure 5.7(a) The unexplained variance versus the number of EBF nodes using various levels of lookahead (*g*-1) for CERN. The values are plotted for the Kine-8nm test data set.

5.8.4 Look ahead versus greedy and optimal lookahead

The discussion will first focus on the difference between lookahead ($g > 1$) versus no lookahead at all ($g = 1$), before moving onto the effects of various lookahead horizons. The performance of the various networks on the test data sets is plotted in figures 5.7 –5.11. These figures contain the raw results from the experiments and not summarising results. The horizontal axis represents the number of nodes and the vertical axis the error on the test set. The number of nodes includes the bias. Thus for an initial lookahead of for example 4 nodes and $g = 5$ the plot starts with 6 nodes (five hidden nodes plus the bias). Figures 5.7(a) –5.7(c) show the percentage of variance explained on the Kine-8nm data set. It can be seen that for all three node types the algorithms using lookahead had a consistently higher accuracy than those networks that were trained using the stepwise greedy method. Furthermore, the networks with lookahead also achieved a high accuracy with markedly fewer nodes, resulting in simpler models. The accuracies for the Kine-8nh are shown in figure 5.8 and show the same pattern. For the other synthetic data set, the Mackey-Glass time series, all the errors always decrease on the graphs of figure 5.9(a) – 5.9(c). However, the networks that were optimised one node at a time always had a much higher error. This difference was particularly manifested for the summation nodes. It should be kept in mind that the lookahead decreased gradually as was explained above.

Figures 5.10 (a) – 5.10(c) are plots of the unexplained variances for the Furnace-2 data, a real world data set. For the axis parallel ellipsoidal nodes the same conclusions can be drawn for the synthetic data sets. For the mixed and summation nodes, lookahead was not always beneficial. The reason is that a simple network of one hidden summation node with a hyperbolic transfer function and a bias already explained 72% of the variance. The best models contained between four and six hidden nodes (excluding the bias) and used a lookahead of 2 nodes. This could correctly explain 74.5% of the variance. The networks using a lookahead of 6 nodes could actually not find any of these models. The minimum number of nodes they could assign excluding the bias is seven. The classification errors for the Sonar data set are shown in figure 5.11(a) – 5.11(c). The best networks were found with ellipsoidal nodes using a lookahead of 4 and 6. Here one can again see clearly that lookahead was important. The error using summation nodes was much higher as is shown in figure 5.12(b). Also no clear trend can be seen in this figure. We see that just as for the Double-Spiral problem (chapter 4), that ellipsoidal nodes fair much better for classification problems. The reason could be that the output layer is linear instead of a hyperbolic squashing function, usually used for classification problems when using summation nodes in the hidden layer. The hidden layer has to deal with sudden jumps of the output from –1 to 1, but may not go beyond 1 for the next data point. A tanh activation function in the output layer would possibly control this better. For mixed nodes, the networks using a lookahead of 4 gave the best results, followed by the network using no lookahead. Although the performances of the various settings for the initial *g* are separated from one another, there is again no clear trend of the accuracy with regard to *g*.

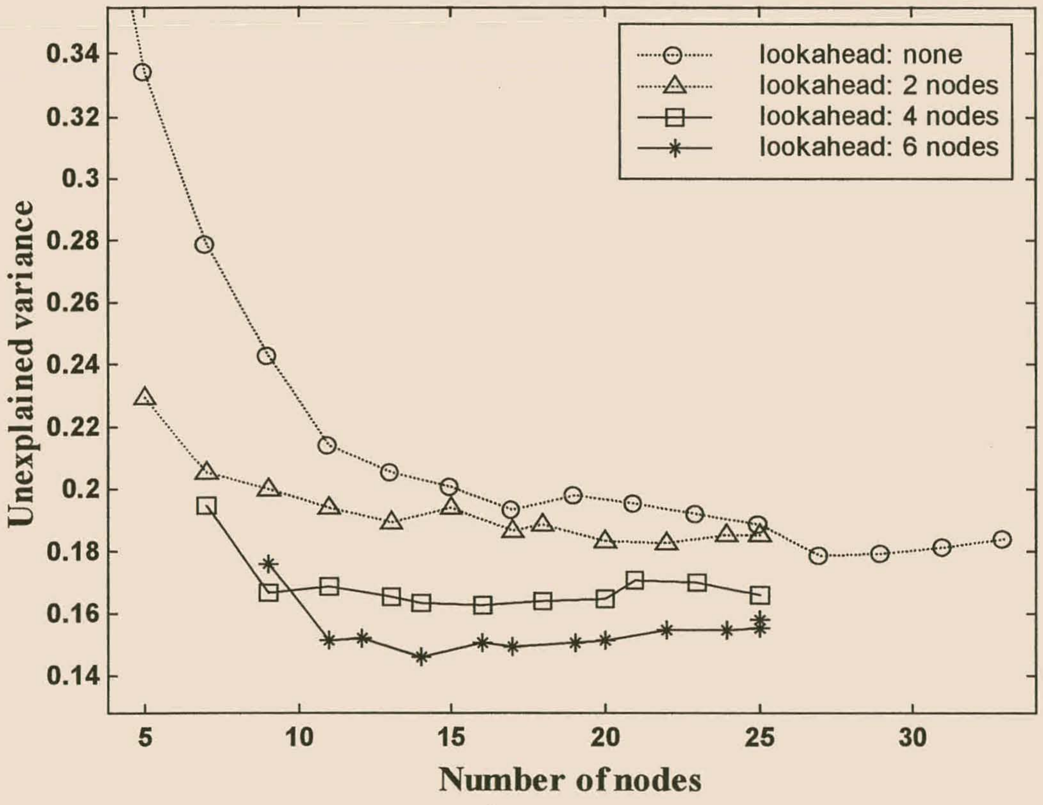

Figure 5.7(b) The test error of the Kine-8nm data of CERN using various levels of lookahead (*g*-1) for summation nodes.

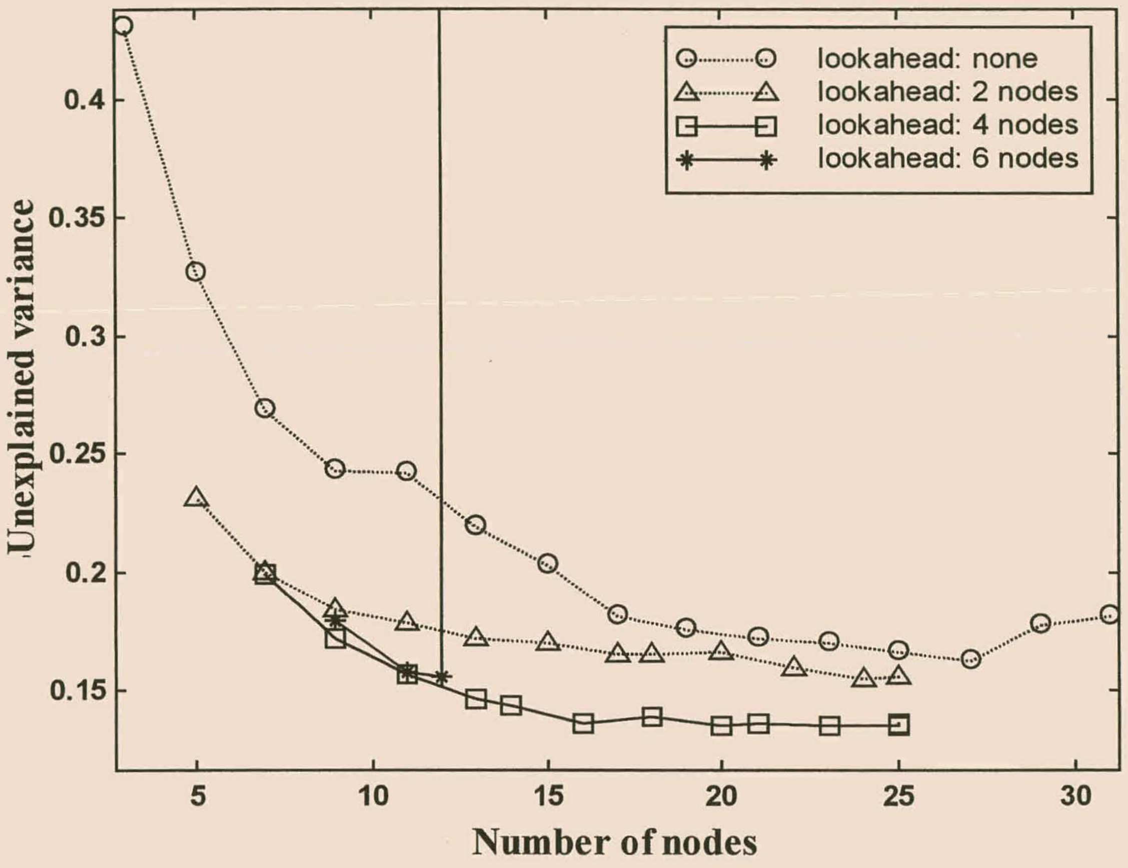

Figure 5.7(c) The test error on the Kine-8nm data versus the number of mixed nodes is plotted for various levels of lookahead (g-1) for mixed nodes. For a lookahead of 6 the error increased suddenly after 12 nodes, as described in the text at the end of 5.8.4.

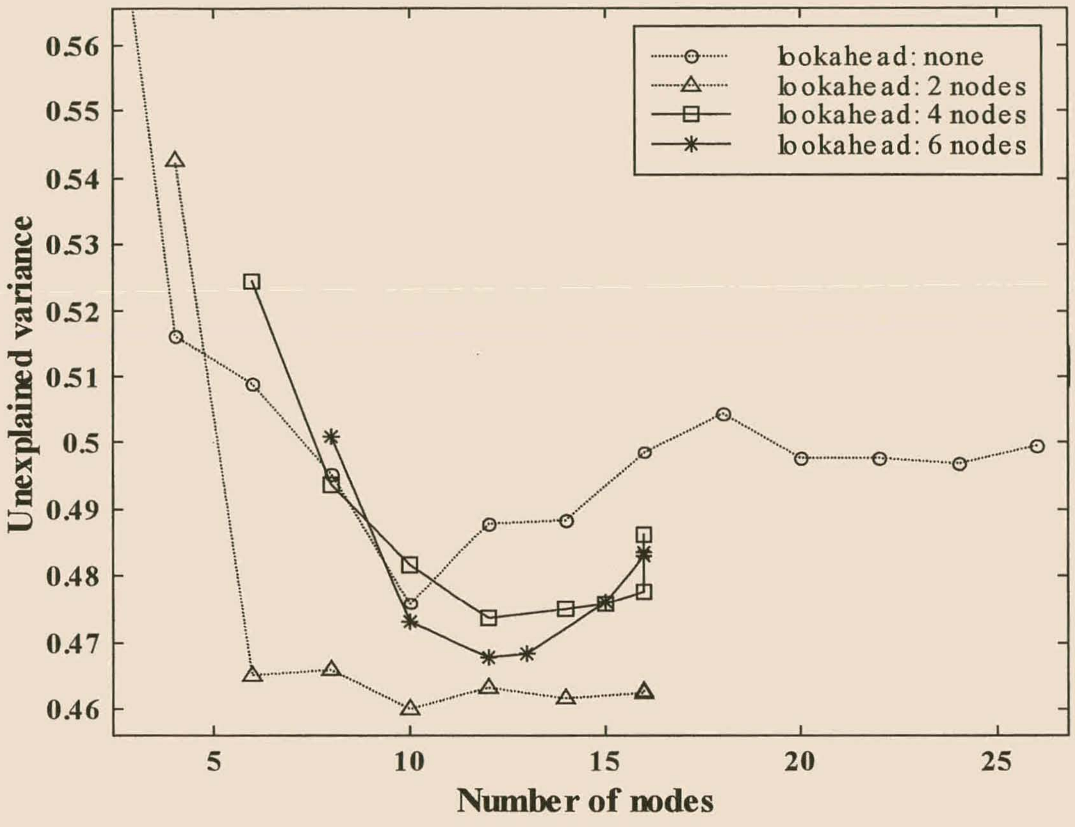

Figure 5.8(a) The test error of Kine-8nh data set of CERN using various levels of lookahead (*g*-1) for EBF nodes.

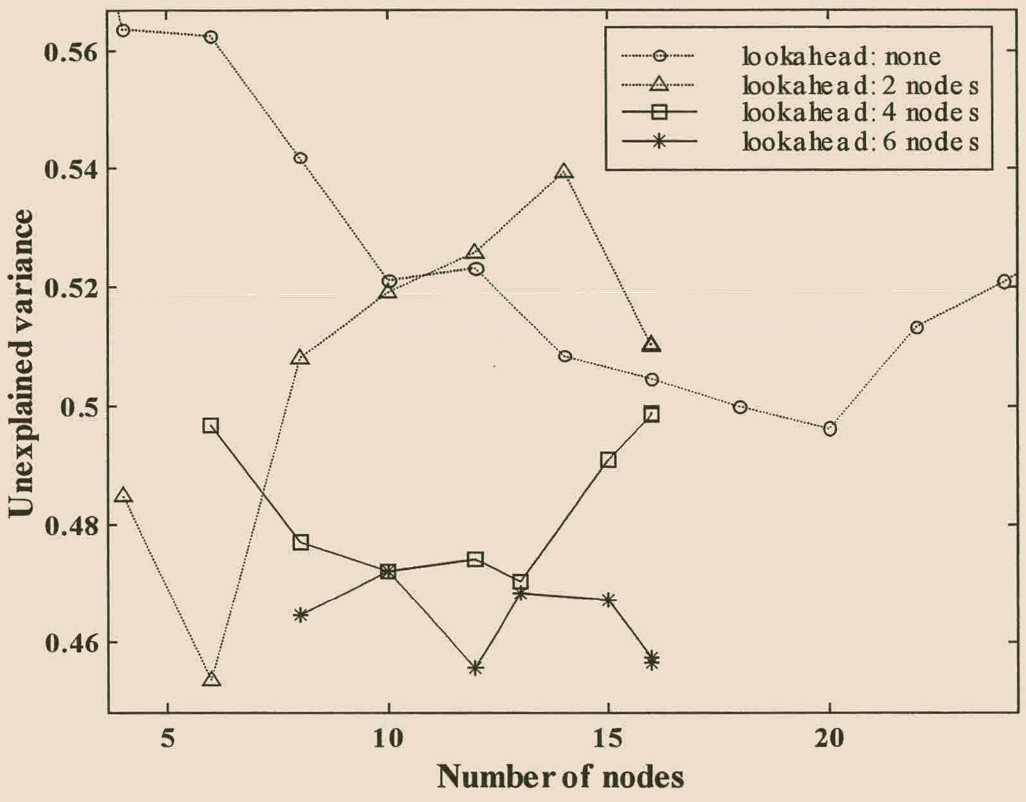

Figure 5.8 (b) The test error on the Kine-8nh data set of CERN using various levels of lookahead (*g*-1) for summation nodes.

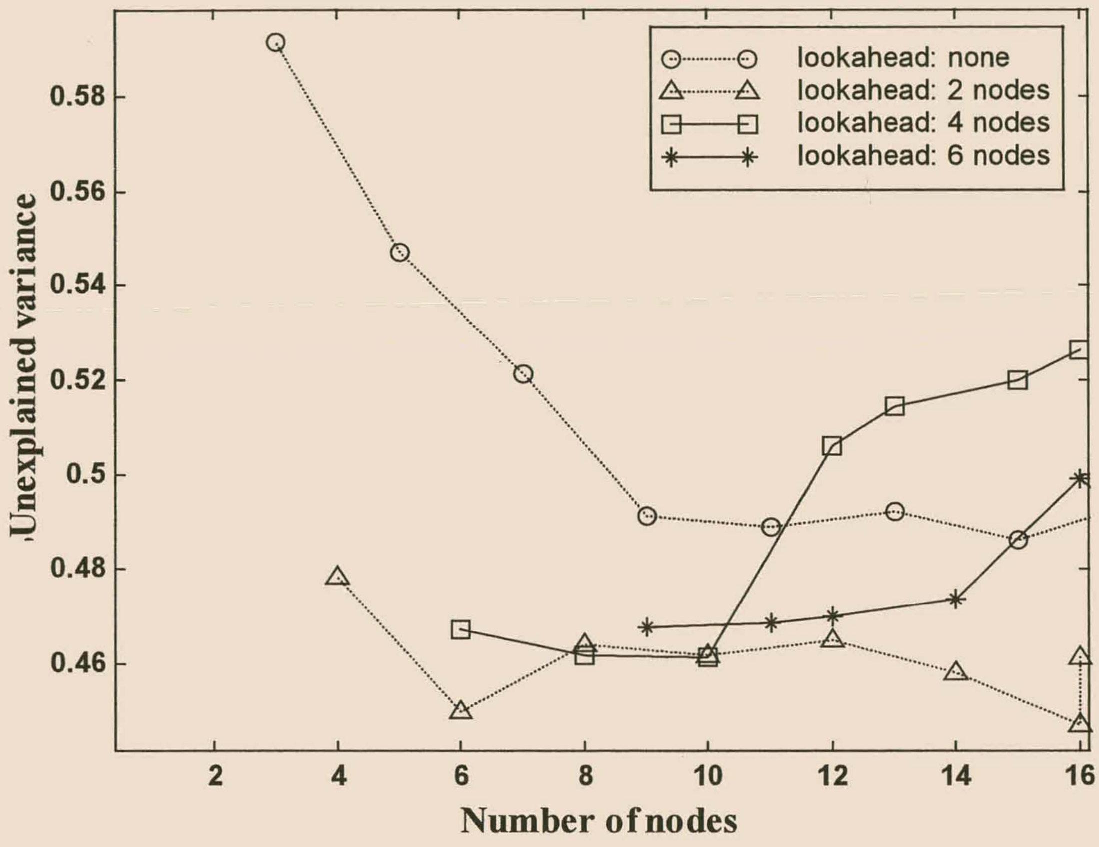

Figure 5.8(c) The unexplained variance on the test set of the Kine-8nh data set using various levels of lookahead (g-1) for mixed node types.

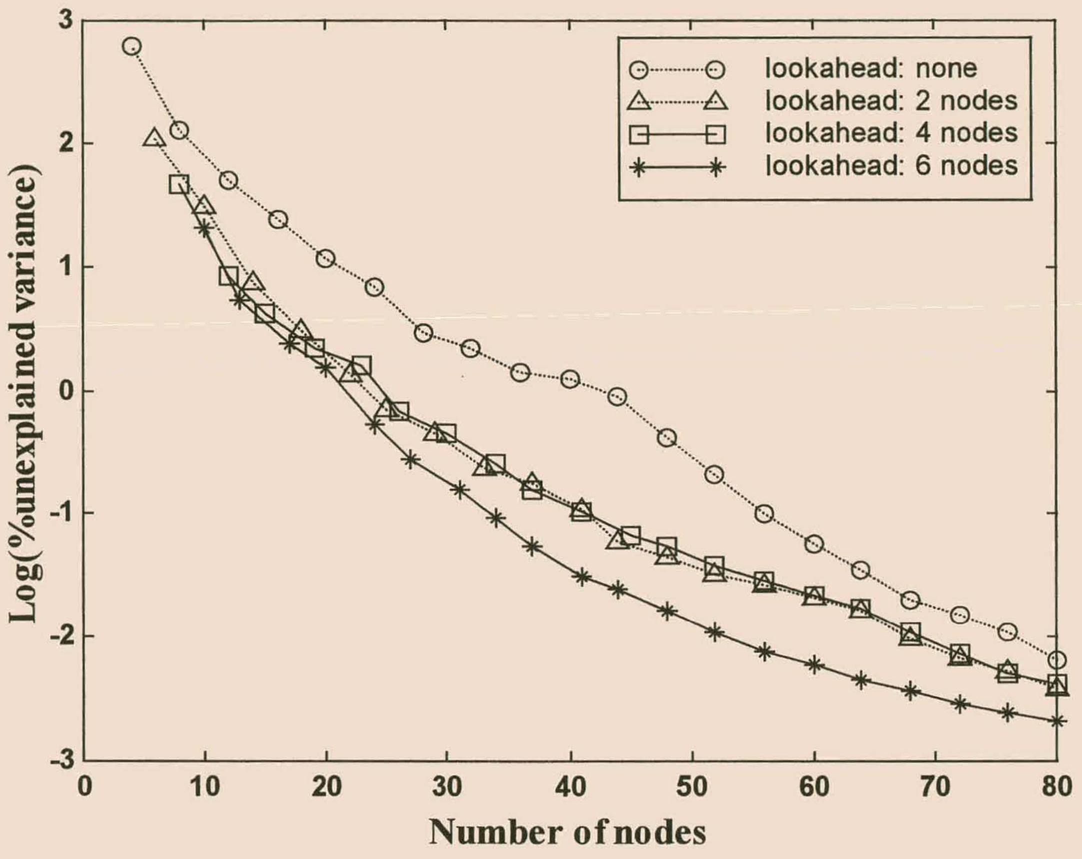

Figure 5.9(a) The test error on the Mackey-Glass time series of CERN using various levels of lookahead (g-1) for axis-parallel ellipsoidal basis nodes.

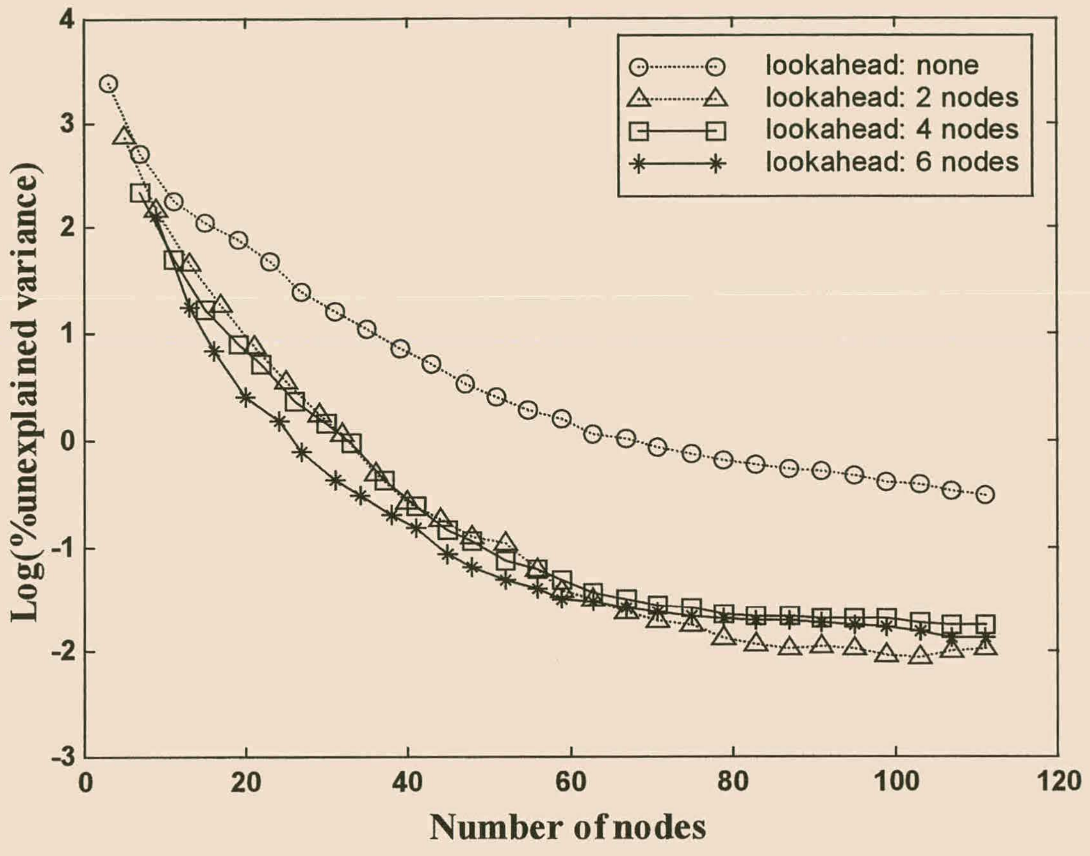

Figure 5.9(b) The testing error on the Mackey-Glass time series of CERN using various levels of lookahead (g-1) for summation nodes.

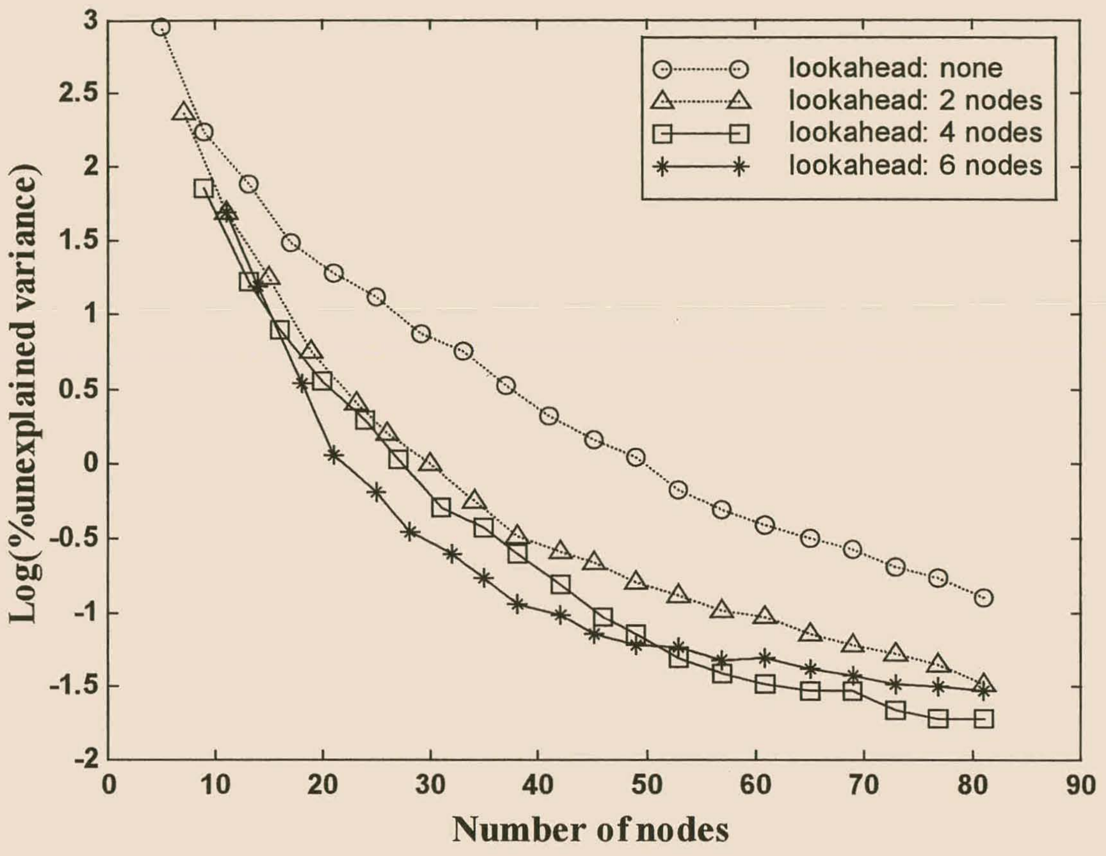

Figure 5.9(c) The testing error on the Mackey-Glass time series of CERN using various levels of lookahead (g-1) for mixed nodes.

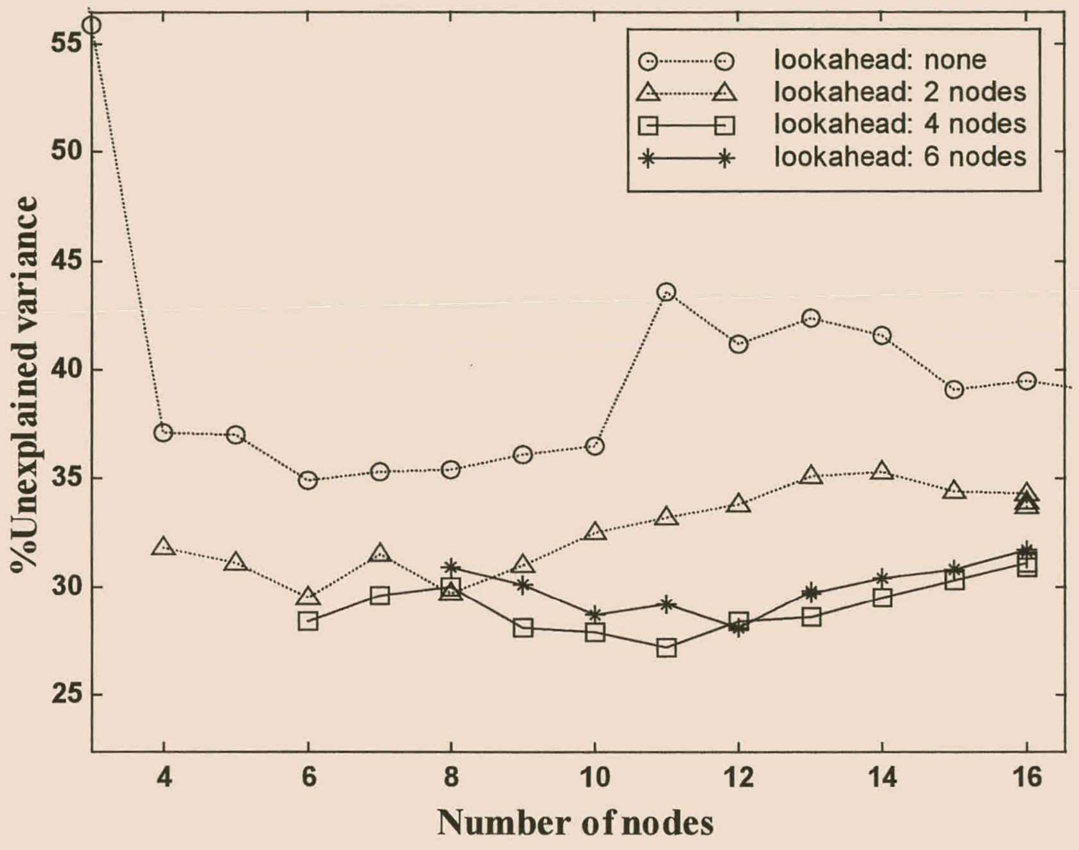

Figure 5.10(a) The testing error on the Furnace-2 data set of CERN using various levels of lookahead (*g*-1) for EBF nodes.

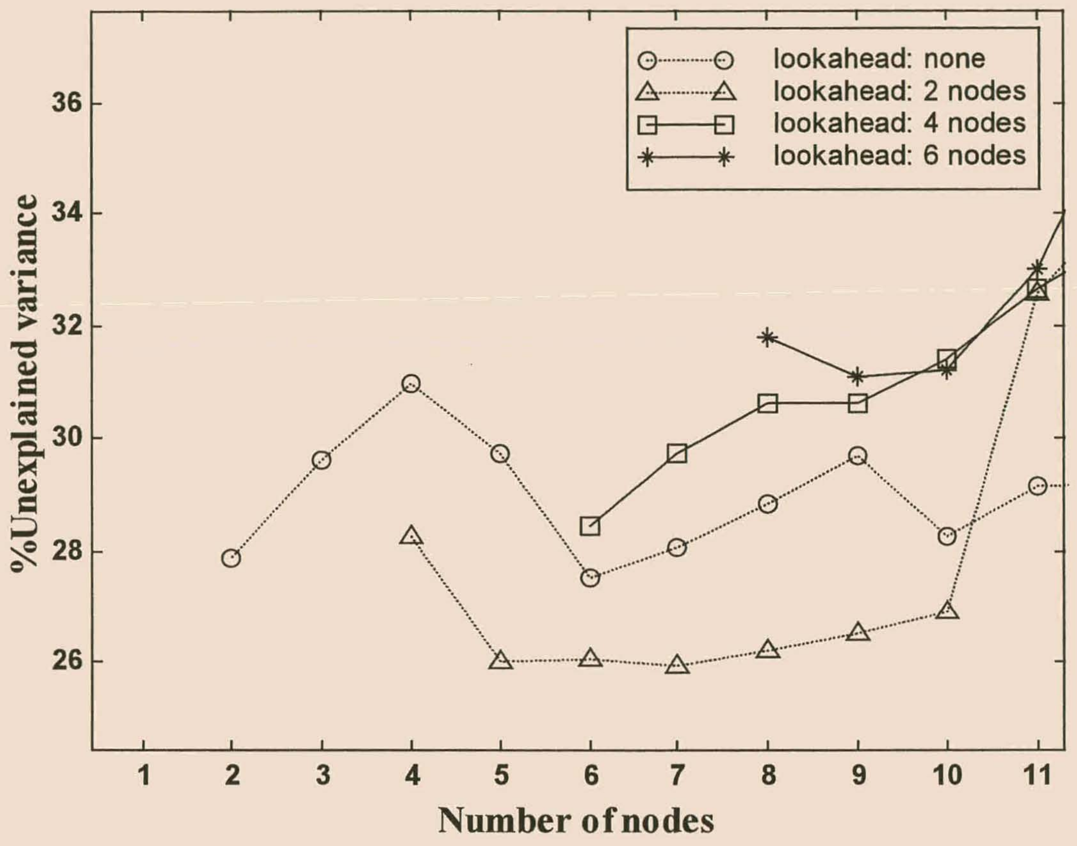

Figure 5.10(b) The testing error on the Furnace-2 data set of CERN using various levels of lookahead (g-1) for summation nodes.

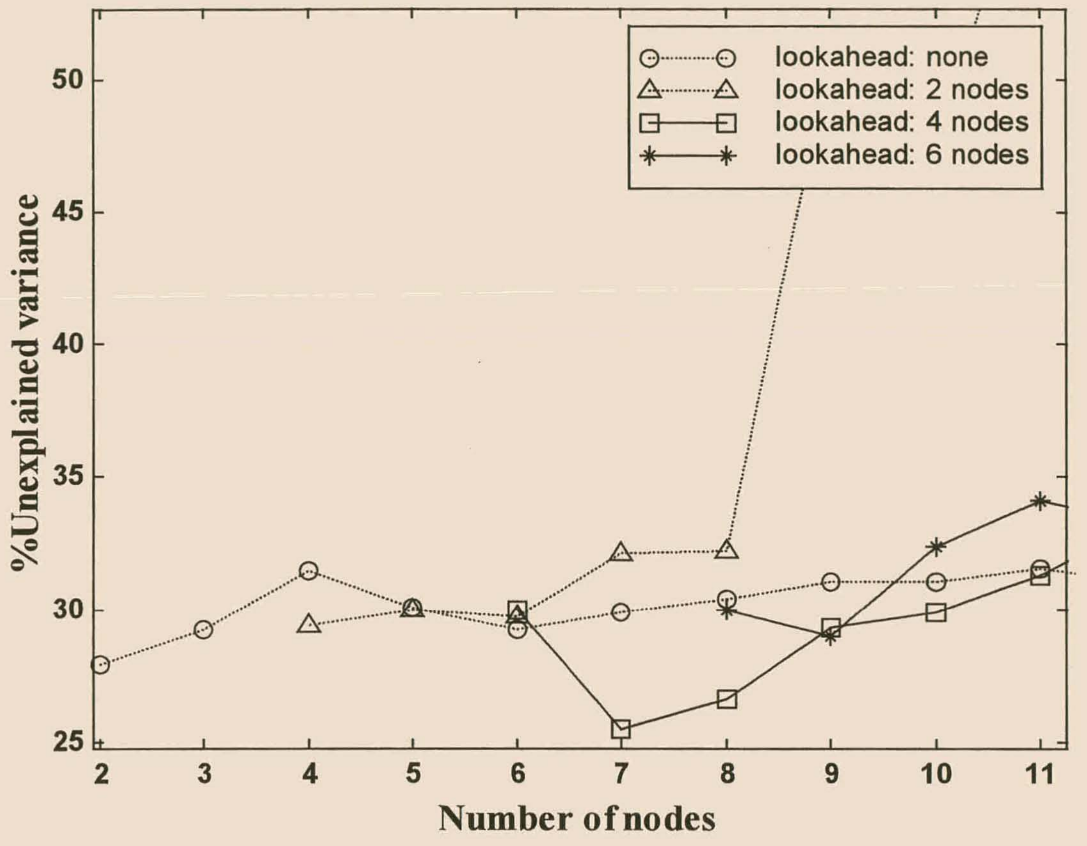

Figure 5.10(c) The testing error on the Furnace-2 data set is plotted against the number mixed nodes using various levels of lookahead (*g*-1). The error for a lookahead of 2 increased suddenly after 8 nodes were added, as is explained in the text at the end of section 5.8.4.

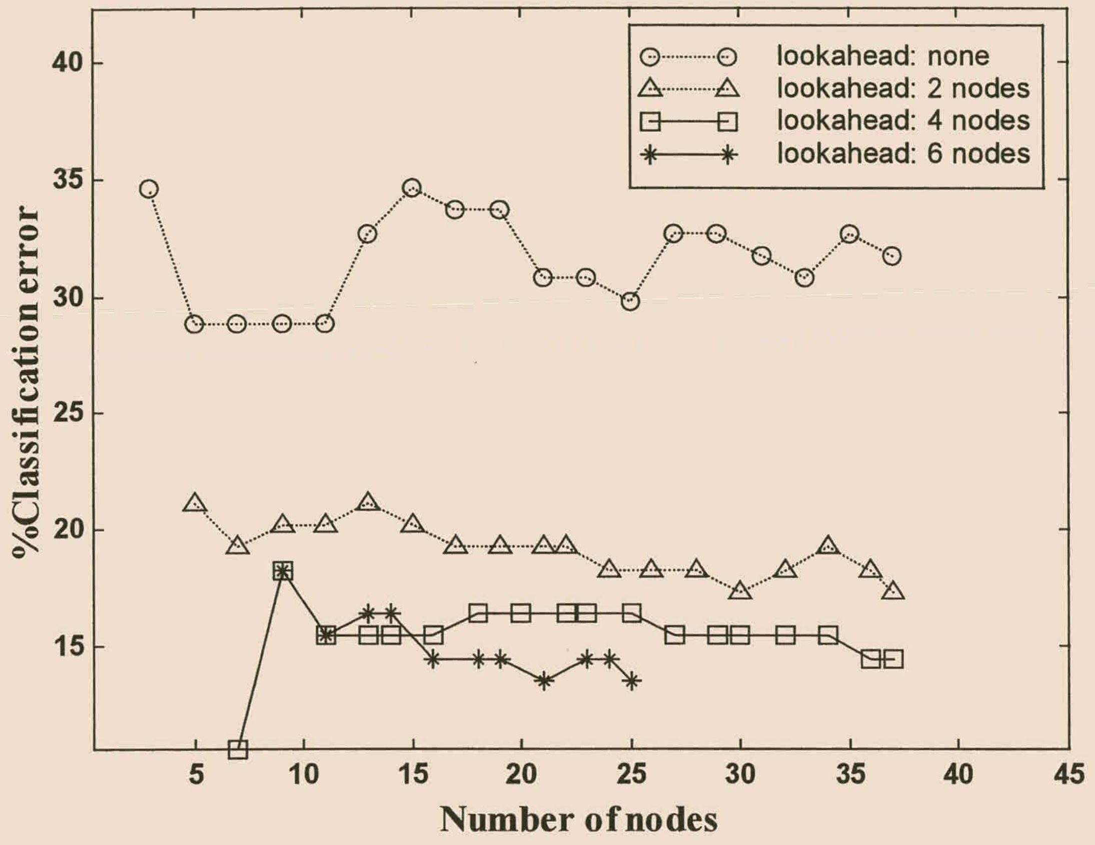

Figure 5.11(a) The testing error on the Sonar data set of CERN using various levels of lookahead (g-1) for EBF nodes.

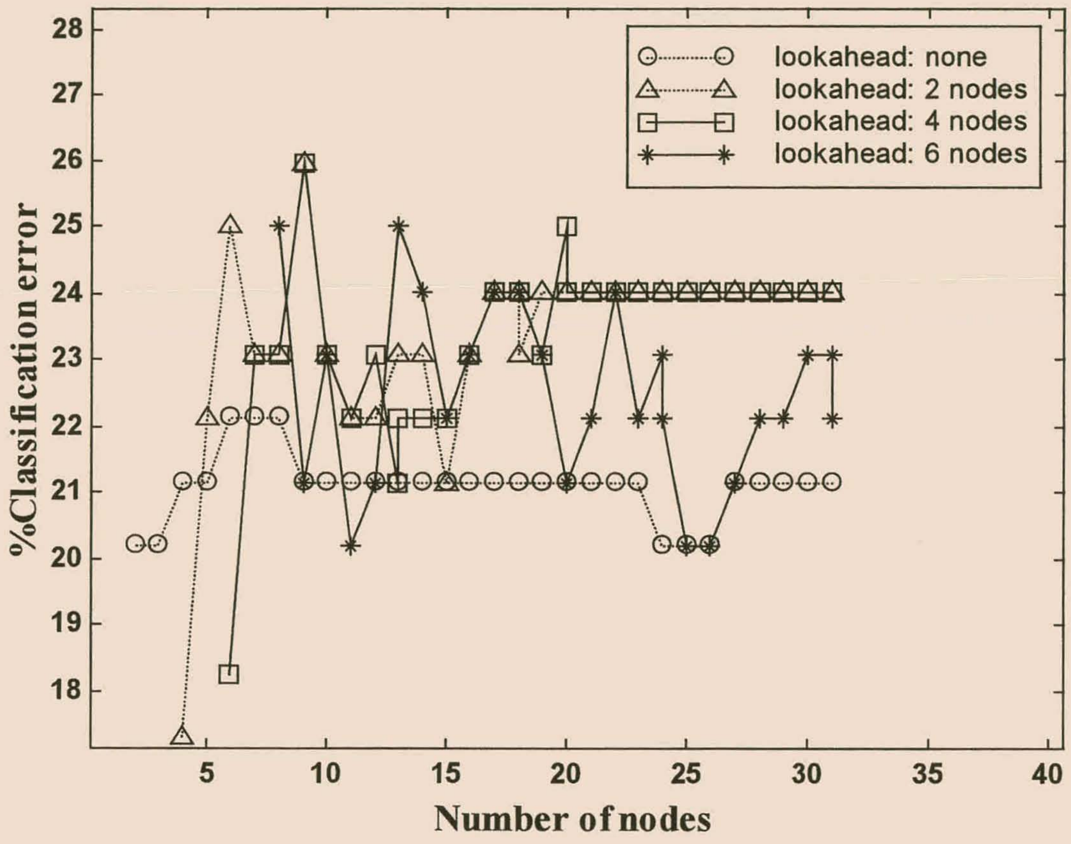

Figure 5.11(b) The testing error on the Sonar data set of CERN using various levels of lookahead (g-1) for summation nodes.

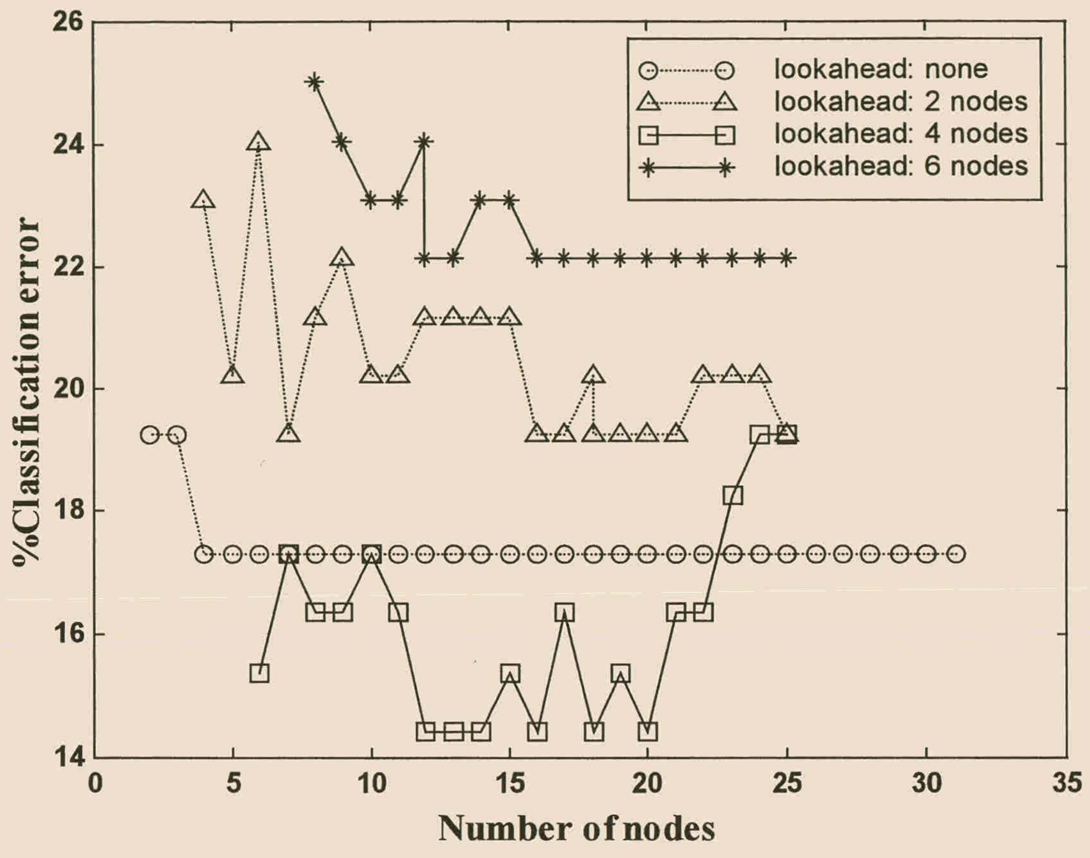

Figure 5.11(c) The testing error on the Sonar data set of CERN using various levels of lookahead (*g*-1) for mixed nodes.

Overall it can be said that lookahead can be of significant benefit, unless the nodes are not suited for the problem or the networks perform optimally using less than *g* nodes, which are required for an initial lookahead of *g*-1.

Next the effect of the extend of lookahead will be discussed. For certain data sets increasing the lookahead beyond 2 and 4 did not always increase the accuracy on the test set and in some cases made it worse. In light of an earlier discussion such behaviour could be expected to occur. Besides overfitting the data, it is also possible that the networks with an extensive lookahead were not trained sufficiently. This happened for example, on the Mackey-Glass time series for the summation nodes with initial g = 7 Here the networks trained with $g = 5$ and $g = 3$ actually had a lower training error. In other cases it was hard to determine whether sub-optimal performance on the test set resulted from overfitting or undertraining or a combination of both. Certainly for the Furnace-2 data set using summation nodes, the data was overfitted when using an initial $g = 7$. As was already discussed above, the best networks only have 1-6 nodes and cannot be found when using $g \geq 7$. For the type of problems studied in these experiments, an initial lookahead of four or five seems close to optimal, when considering the trade off between accuracy and speed. It should be mentioned that for an experiment, a lookahead of 6 using mixed nodes on kine-8nm data set, the network seems to have overtrained considerably when the unexplained variance jumps up sharply, as shown in figure 5.7(c). This was caused by large outgoing weights computed by the regression and can be easily spotted by examining the output weights. This can occur when a node's activation for the training data lies within a very small range. The regression will effectively rescale the range by giving the node a large output weight. The same phenomenon can be observed in figure 5.10(c). In practice, the best way to prevent this is to limit the ranges of all the nodes' outputs in the testing phase to the range occurring in the training phase.

5.8.5 The effect of various node types optimised simultaneously

For the kine8-nh data, and especially for the kine8-nm data, the mixed nodes gave the best results. The best network for the kine-8nm data set was found using mixed nodes. The network was constructed as follows: Nodes 1-10 were summation nodes, the rest was made up of roughly 2/3 basis nodes and 1/3 summation nodes. The summation nodes thus seem to fit the global trend initially and the ellipsoidal nodes the remaining local features. For the Furnace-2 data set, the best network was also a mixed network, but better performance was not consistent. Nevertheless, the mixed nodes generalised almost as well as the summation nodes and better than the ellipsoidal nodes.

The mixed nodes fitted the Mackey-Glass time series worse than only the ellipsoidal nodes or only the summation node. The only exception to this was that for no lookahead the mixed nodes learned to generalise better than the summation nodes. A possible explanation is that the data set contains no noise and has a low dimensionality. Consequently a network is not easily overtrained for this problem. See for example Littmann and Ritter (1993), who found that network with many layers had low testing error on this data set. The population of mixed nodes was divided into three sections, two of which are independent from one another and could thus take more iterations to converge. The testing error of the mixed nodes, which was higher than both the error of the summation nodes and of the ellipsoidal nodes can thus be caused by undertraining. Nevertheless for most data sets it seems advisable to also try networks containing summation nodes only, and networks containing ellipsoidal nodes only, and not just mixed nodes.

5.8.6 The PRESS statistic

For the neural networks the PRESS statistic followed the training accuracy closely and kept increasing with each added node, even when the testing accuracy decreased dramatically. The only times the R^2-PRESS statistic did decrease was for those networks where the regression resulted in the very large weights. This can be determined much more easily by a simple inspection of the weights as discussed in section 5.8.4. The PRESS statistic did not give an indication of overfitting for any of the other experiments. The following discussion outlines the reason for this. The regressors in a non-linear regression are typically formed before the PRESS statistic is computed. In this way, for each validation point all of the available data points were actually used and none were left out entirely. Thus in this case the formula used for the PRESS statistic is not a true leave-one-out crossvalidation error and should be used with caution for non-linear regression.

5.8.7 List of default parameters

From the experiments discussed in this chapter, the parameters contained in table 5.3 have been chosen as default parameters for the CERN algorithm. These parameters were used in subsequent chapters unless other values are specified.

CR	0.75
F	0.5
XR	$5*(g-1)/100$
B	0.9
l	3
Ridge k_r	0.005
g	6#
Rate of decrease of g	5#
N_g	200
Base iterations	400

Table 5.3 The default parameters of CERN. # indicates a value that is problem dependent and that was therefore allowed to change. When this occurs it will be mentioned.

5.9 Conclusions of Chapter 5

Several improvements have been made to the CERN algorithm described in chapter 4. In particular the following can be said:

- Using clusters on the chromosomes to actively support niches in the population, results in significantly faster learning.
- Copying complete individuals from one group to another, likewise improved the learning speed.
- Learning was most rapid with values of 0.75 and 0.5 respectively for the parameters CR and F.
- Clustering allows for different node types in the same population.
- In two case studies a mixture of summation nodes and axis-parallel ellipsoidal basis function nodes resulted in better networks than using either summation nodes only or ellipsoidal nodes only. In all other case studies either the ellipsoidal nodes or the summation nodes performed better.
- In comparison with an algorithm that only appends and optimises a single node at a time, networks optimised by using lookahead with respect to the nodes performed significantly better. The latter method also resulted in smaller networks.
- The PRESS statistic is not a good indication of overfitting for non-linear models such as those build by CERN.

Chapter 6 Comparison of CERN to backpropagation and other algorithms

6.1 Introduction

In this chapter CERN's performance is analysed on various data sets arising from chemical process. In order to allow other researchers to make comparisons to CERN, additional benchmark data sets were also included in the study. The use of these benchmark data allows CERN to be compared to state-of the art algorithms that cannot be easily obtained. When possible these benchmark data sets were chosen from the field of chemical engineering or engineering in general. It was already shown in chapter 4 that CERN is significantly faster than the DE algorithm for the training of neural networks. In addition CERN was also compared to other non-evolutionary algorithms, even when these could only optimise those parameters, which have a defined gradient. First CERN's training speed was first compared to standard techniques such as the generalised delta rule (GDR) and the Levenberg-Marquardt (LM) learning rule (Marquardt, 1963). After that its ability to generalise was compared to that of a variety of techniques, including multivariate adaptive regression splines (MARS), and multilayer-perceptrons trained with the generalised delta rule and the Levenberg-Marquardt algorithm. The training efficiency of CERN was subsequently evaluated on a data set arising from an industrial furnace and two benchmark data sets. The test accuracy was assessed on these data sets, as well as process data arising from the corrosion of metals, a synthesised autocatalytic reaction and a study of the sap flow rate in pine trees. A speaker independent vowel recognition benchmark task is also used for this purpose. See Appendix A for further information regarding all the data sets.

As was already mentioned in chapter 2, CERN can also be used to build certain types of fuzzy systems. For the current chapter such rule systems were constructed for data from induced aeration, the Miles/Gallon benchmark set and a realistic simulation of the dynamics of a PUMA robot. The resulting fuzzy systems were compared to those of adaptive neurofuzzy inference system (ANFIS) (Jang, 1993) and radial basis function (RBF) networks using k-means clustering.

6.2 Empirical analysis

6.2.1 Experiments to determine training speed and training quality of CERN

The evolutionary algorithm CERN was compared to the derivative based delta rule (DR), the generalised delta rule and the Levenberg-Marquardt algorithm. CERN was used with clustering and with copying over, as described in the previous chapter. The learning parameters took on the default values decided upon in the preceding chapter, as listed in table 5.3. The backpropagation algorithm was used with the standard delta rule as well as with the generalised delta rule having an additional momentum term (Zurada, 1992). Various learning rates were tried for the delta rule. A rate of 0.12 gave the best convergence on some preliminary runs and was therefore used in all experiments. The generalised delta rule was implemented with an adaptive learning rate. A momentum of 0.9 was used as the default value. The learning rule of the Levenberg-Marquardt algorithm was used with all the default settings. The update rule is $\Delta W=(J^TJ + \mu I)^{-1}J^T\mathbf{e}$, where J is the Jacobian for the network output of each weights for all the data points, $\mathbf{e} = \mathbf{t} - \mathbf{a}$ (target-network output) and I the identity matrix. The implementation decreases μ after every iteration when the update rule resulted in a greater error and increases it when the error diminished. See O' Brian (1998) for details. The default values were 0.001 for the initial value for μ, 10 for the multiplier increasing μ, 0.1 for the multiplier decreasing μ. The max value of μ was set at 10^{10} and the minimum at 10^{-6}.

These algorithms were used to train feedforward networks for three problems, viz. the double spiral problem, the Furnace-1 data set and the Abalone data (see Appendix A). The hidden layers of all the neural networks consisted of summation nodes only, because gradient descent based algorithms are not readily available for other node types. On the two real world data sets, validation runs were made with each of the algorithms to find the best network architecture. For the Abalone data set 6 nodes in the hidden layer and a linear output node gave the best results on the validation sets, as discussed in chapter 4. CERN, GDR and LM algorithms gave the best results on the validation set of the Furnace –1 data, by using 7 nodes in the hidden layer and one linear output node.

The algorithms were compared in terms of the number of floating point operations used. The number of function or network evaluations used served as a base line. For the generalised delta rule and the Levenberg-Marquardt algorithm this is equal to the number of epochs. For CERN the number of network evaluations is equal to the iterations multiplied by the number of groups in the population (N_g), which was set to 200. All the networks were trained for 100000 network evaluations. The Levenberg-Marquardt algorithm requires many more computations per iteration and also converged much earlier. It was thus only run for a maximum of 4000 network evaluations. In cases where an algorithm showed signs of significant improvement towards the end of the training, it was allowed to continue for some more iterations. All experiments were repeated with four different seeds for the pseudo-random number generator, since the algorithms are dependent on initial configurations. The testing error was also recorded for all three data sets used to evaluate the training efficiency. The next section describes some additional experiments that were conducted in order to evaluate the testing accuracy of CERN.

6.2.2 Assessing the generalisation of networks trained by CERN

Repeated independent runs were made on several of the data sets given in Appendix A. The test data of these data sets were not used previously in order to determine any of CERN's parameters, thereby making them proper test sets. For CERN and the other algorithms investigated, fractions of the training data were used as validation sets in order to fine tune parameters, especially the numbers of nodes to use*. The lookahead algorithm used with CERN causes the networks size to grow one node at a time. Therefore only a few runs need to be made in order to find a suitable network architecture for a specific problem. The neural network configuration used for the multilayer perceptrons trained with generalised delta rule and the Levenberg-Marquardt algorithm, as well as for CERN on each of the problems are given in tables 6.1(a) and 6.1(b) respectively. In cases where the LM algorithm was obviously stuck in non-optimal minima, based on the performance on the training set, the network was discarded and the experiment was repeated.

Some algorithms, such as MARS, automatically use cross-validation estimates on the training data in order to fine-tune the number of nodes to use. In such cases extra validation runs were not made, and default settings were used for other parameters. The data sets used were the same data

* In some papers the terminology differs. There the data set used try out different configurations are called test sets and the data set used to asses the final performance is called the validation set.

sets used in the previous section (Abalone, Furnace-1) as well as the Autocatalytic, Corrosion, Pine, Puma-32mn and Vowel Recognition data.

Data Set	*Number of hidden nodes*	*Output transfer function*	*Number of epochs*
Abalone	6 / 6	Linear / Linear	125000 / 100
Autocatalysis	23 / 20	Linear / Linear	125000 / 450
Corrosion	8 / 8	Tanh / Tanh	125000 / 30
Furnace –1	7 / 7	Linear/ Linear	150000 / 200
Pine	20 / 18	Linear/ Linear	125000 / 450
Puma-32mn	18 / 18	Linear/ Linear	125000 / 300
Vowel	88# / N.A.ϑ	Tanh# / N.A. ϑ	Not known#

Table 6.1(a) The best settings found for each of the validation sets when using the GDR / LM algorithm for training a multilayer-perceptron. # Results obtained from Robinson (1989) ϑ The LM software required too much memory for this particular problem.

Data Set	*Number of hidden nodes*	*Node Types*
Abalone	6	Summation
Autocatalysis	15	OEBF
Corrosion	8	Summation
Furnace –1	7	Summation
Pine	22	Mixed
Puma-32mn	14	Summation
Vowel	4	EBF

Table 6.1(b) The best settings found during the validation runs when using the CERN algorithm for training a regression network. For the other learning parameters the default values given in table 5.3 were used. *g* was always set to 6,

except when the number of hidden nodes to add was less than 10. In that case g was made equal to the number hidden nodes.

6.2.3 Fuzzy modelling using CERN

As was already mentioned in section 2.3.2, 0th order Sugeno fuzzy systems with weighted sum consequent determination are also *regression networks* and can therefore be built by CERN. In particular it was shown that a zero order Sugeno system using the weighted sum defuzzifier can be written as:

$$y(\boldsymbol{x}) = \Sigma_i a_i \mu_i(\boldsymbol{x}). \qquad \text{Equation 6.1}$$

Furthermore, if the product aggregation method is used the weight of the i^th rule, $\mu_i(\boldsymbol{x})$, becomes $\Pi_j \mu_{ij}(x_i)$. Such a Sugeno system is thus a function of the form:

$$y(\boldsymbol{x}) = \Sigma_i a_i \, \Pi_j \mu_{ij}(x_i). \qquad \text{Equation 6.2}$$

Although the underlying design philosophy differs, the functional form is exactly the same as that of radial basis function network if the basis function $\Phi(\boldsymbol{x},\boldsymbol{c})$ can be factorised into functions of its components:

$$\Phi(\boldsymbol{x},\boldsymbol{c}) = \Pi_j \, \Phi(x_j,c_j). \qquad \text{Equation 6.3}$$

This is the case for ellipsoidal Gaussians that are axis-parallel (Haykin, 1994). Therefore by restricting the hyperellipsoidal Gaussians of the network to lie axis-parallel, each hidden node of the ellipsoidal basis function (EBF) network can be converted directly into a fuzzy rule. It is therefore possible to translate the EBF networks optimised by CERN into rules of a fuzzy system: The membership functions are formed by factorising each of the ellipsoidal Gaussians into a product of one-dimensional Gaussians. Therefore for each ellipsoidal or spherical Gaussian there will be d one-dimensional Gaussian projections, one for each input feature. In the corresponding fuzzy system these Gaussian projections form the membership functions. Orientated ellipsoidal basis function (OEBF) networks can in principle also be translated into a system of so-called ellipsoidal fuzzy systems (Dickerson & Kosko, 1993). In this case the rules no longer have membership functions in the original co-ordinate system and are difficult to comprehend.

For the Aeration, the Miles/Gallon and the Puma-32mn data sets fuzzy systems were studied. These were formed using CERN, ANFIS and RBF networks trained by the k-means clustering to setup the hidden layer. The ANFIS algorithm that was available could only build 1st order

Sugeno systems and not 0th order Sugeno systems, as constructed by the other methods. Furthermore, the ANFIS algorithm initially required a grid-partitioning of the input space. Each partition is an antecedent of a rule and the number of rules is therefore u^d, where u is the number of membership function formed for each of the d inputs.

When considering fuzzy systems there is an accuracy-complexity trade-off. Thus a simpler fuzzy system may be preferable even when it generalises slightly worse. The choice of system is often a subjective matter. All three algorithms were therefore used to build several fuzzy systems, each having a different number of rules. The complexity is reported in terms of the number of rules and in terms of the number of parameters.

The number of parameters for a given number of rules (h) is computed as follows: Each RBF node requires d centre co-ordinates, a width and an outgoing weight. In addition the employed radial basis function algorithm used a bias and partial linear model, in other words the input nodes were connected to the output layer. Thus the RBF network has $\mathrm{h\,(d+2)+d+1}$ parameters. The ellipsoidal basis function network has d width factors instead. Thus the number of parameters are $h\,(2d+1)+1$. The orientated ellipsoids have an additional $\tfrac{1}{2}\mathrm{d(d-1)}$ rotations. Thus OEBF networks requires $h\,\{2\mathrm{d} + \tfrac{1}{2}\mathrm{d}\,(\mathrm{d}-1) + 1\} + 1 = h\,\{\tfrac{1}{2}d\,(d+3) + 1\}+1$ parameters for a full description. The fuzzy systems optimised by ANFIS have $\mathrm{u^d}$ rules, each of which has a linear consequent with d parameters and a constant. In addition each of the $\mathrm{u \times d}$ membership functions requires a centre and a width. This adds up to a total of $u^d\,(\mathrm{d}+1) + 2\mathrm{ud}$ parameters.

For most of the algorithms the number of rules (nodes) is the only important criterion that determines the accuracy on the training and on test set. However, different width factors, which scale the standard deviation of all the Gaussians, were also examined when using this unsupervised k-means clustering scheme of the hidden layer. From various experiments on validation sets using width values of 1.0; 1.25; 1.5; 1.75; 2.0; 2.25; 2.5; 3.0; 3.5; 4.0 it was found that the width factor was important. A value of 1.75 resulted in the best performance on the Puma-32mn data set, a factor of 2.0 for the Aeration problem and the best results were obtained with a value of 3.5 for the Miles/Gallon data. Interestingly, as the dimensionality of the problem increases (2,4,7) so does the optimal width factor (based on validation runs). The Miles/Gallon data set only has 400 data points and the Aeration set only contains 100 and 60 points, respectively for each of the impellers. Thus 5-fold crossvalidation was used on these two data sets in order to obtain more accurate estimations of the test error.

6.3 Results

6.3.1 Floating point operations of CERN, BP and LM

In chapter 4 it was established that for each of the network evaluations CERN requires $2ngd + ng$ floating point operations to compute the summation nodes, $3ng + 2ngp + 3ng^2$ to update the product matrix and some further operations are needed for the combinatorial search. The GDR algorithm has to work through the following calculations with each epoch, assuming that it optimises g nodes simultaneously and has already got p nodes (e.g. input nodes) connected to the output layer:

The forward sweep to the hidden layer requires the same number of iterations as CERN, which was $2ngd + ng$. An additional $2ng + 2np$ floating points operations are required for the output node. The backpropagation phase uses n operations to compute the error in the output layer. The sum of the squared errors takes another $2n$ operations. These errors needs to be propagated backwards to the g hidden nodes. This requires $4ng$ operations. The learning phase computes the changes that need to be made to the each of the weights. For the output node this amounts to $2ng + 2np + 3n$ operations and for the hidden layer to $2ngd$ plus $3gn$ floating point operations. The total number of operations for the GDR at each epoch is thus:

GDR: $2ngd + 3ng + 2np + 3n + 4ng + 2gn + 2np + 3n + 2ngd + 3gn$

$= 4ngd + 12ng + 6n + 4np$ Equation 6.4(a)

In order to highlight the differences to CERN this can be regrouped as below:

$(2ngd + 4ng) + 2ngd + 8ng + 6n + 4np$ Equation 6.4(b)

CERN on the other hand, ignoring the overhead and the combinatorial search uses:

$(2ngd + 4ng) + 2ngp + 3ng^2$ floating point operations. Equation 6.5

Besides the combinatorial search CERN has some additional overhead associated with clustering and the formation of new trial vectors. Empirically it was found that for most problems the overhead associated with this required less than 10% of the total time. Although the clustering takes relatively long, the chromosomes are clustered only once for every iteration and not for every single group of nodes. The extra overhead does not grow with n and is therefore not a problem for large data sets. It can consequently be ignored.

To optimise a function in v variables using the Levenberg-Marquardt algorithm for n data records requires $nv^2 + v^3 + nv$ floating point operations at each iteration, excluding the function evaluations (Powell, 1972). For a neural network with g summation nodes $v = g(d+1) + g + p + 1 = gd + 2g + 1$. Thus the number operations including the feedforward pass is:

$$n(gd + 2g + p + 1)^2 + (gd + 2g + p + 1)^3 + 4ngd \qquad \text{Equation 6.6}$$

The calculations of the delta rule without momentum are not analysed in this section. These are very similar to those of the GDR except that the momentum term does not have to be taken into account during the update phase. This reduced the number of floating point operations slightly.

It is difficult to compare the formulas of equation 6.4, equation 6.5 and equation 6.6 in general. A few remarks will be made nevertheless. First consider GDR and CERN. The extra term of CERN is squared in g. The combinatorial search becomes expensive when g becomes greater than 10. Thus CERN becomes infeasible for large amounts of lookahead g, whereas the generalised delta rule does not. CERN will also take longer when p increases, whereas GDR has an extra term proportional to the input dimensions d. From figures 4.6(a)-(c) we also see that as d becomes large, the time taken by the forward sweep becomes an increasing fraction of the total floating point operations. The orientated ellipsoidal nodes in particular take up a considerable amount of time. This trend is the same for GDR and CERN. However, the GDR has another phase where the computational cost grows with d, namely the update formula for each of the parameters of each node. Thus if d is large compared to p or g or if orientated ellipsoidal nodes are used, the GDR algorithm will take longer to complete one network evaluation than CERN. Thus in most cases a network evaluation of CERN will take roughly as long as the (G)DR, but in practice it may differ by a factor of two either way.

As n is usually greater than the number of network parameters, the cost for the Levenberg-Marquardt algorithm rises proportionally with the square of $d \times g$. It thus becomes increasingly expensive with the number of dimensions when compared to the (generalised) delta rule and CERN, the cost of which only increases linearly with d.

The number of floating point operations required by the update rules for the problems are calculated in Appendix B. The final numbers in terms of the number of data points (n) are given in table 6.2. Also shown are the equivalent network evaluations of CERN.

	CERN	*GDR*	*LM*
Double-Spiral	317*n*	146*n* (≈½CERN)	1024*n* (≈3CERN)
Abalone	230*n*	294*n* (≈CERN)	4796*n* (≈20CERN)
Furnace -1	304*n*	314*n* (≈CERN)	5540*n* (≈20CERN)

Table 6.2 The number of floating point operations required for each network iteration for the Double-Spiral, Abalone and Furnace-1 data sets. The number is given in multiples of the number of data points (*n*).

6.3.2 Comparison of CERN, backpropagation and LM training

Representative learning curves of each of the algorithms (CERN, (G)DR, LM) are plotted in figures 6.1 – 6.3. The graphs show the training error plotted against the number of network evaluations. The average training errors are given in table 6.3. For the Double-Spiral problem CERN's training error is always lower than that of (G)DR. Since the Levenberg-Marquardt algorithm is particularly effective for problems with few parameters (Sarle, 1999), it converges very quickly for the Double-Spiral problem. However, as with the (G)DR, the Levenberg-Marquardt update rule tends to gets stuck in sub-optimal solutions. CERN on the other hand continuously reduces the error to lower levels than the other algorithms. An analysis of variance test confirmed that the differences shown in figure 6.1 are significant.

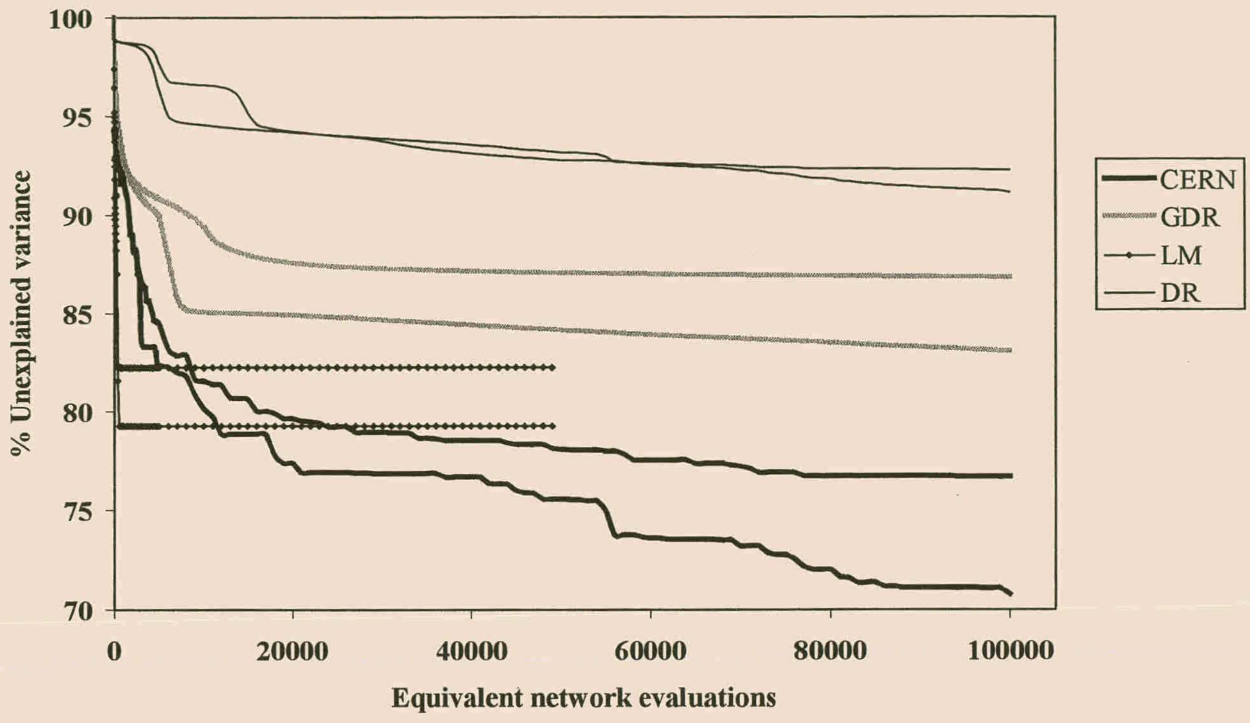

Figure 6.1 The learning curves of (G)DR, CERN and the Levenberg-Marquardt algorithm for seven summation nodes trained on the spiral data set.

The next figure, figure 6.2, shows the training error for the Abalone data set. After less than 10000 network evaluations CERN achieved an error, that the (generalised) delta rule only reached after over a 100 000 network evaluations. CERN also had a smaller final error for both the runs shown. The Levenberg-Marquardt algorithm converged an order of magnitude faster than the other algorithms, including CERN. It also achieved a much lower error. The average final R^2 values on the training data are indicated in table 6.3. An analysis of variance test showed that the smaller training error of the LM algorithm was significant with a level of 95%. The same test revealed that CERN was better than the (G)DR, also with a 95% significance. A linear model could give almost as good a model as the neural network model. A linear regression model had an unexplained variance of 45.7% on the output, compared with values of 37-42% for the multilayer-perceptrons. Thus the data set can be seen to be fairly linear. For a specific process the actual processing time spend on the Abalone problem will be given. The implementation of CERN programmed in this thesis in C++ took 72 minutes for 100 000 network iterations for the Abalone data set on an Intel Pentium II- 400 MHz processor, using the IBM Visual Age compiler without optimisation. This can be used to get a rough indication of the duration to optimise networks on such problems.

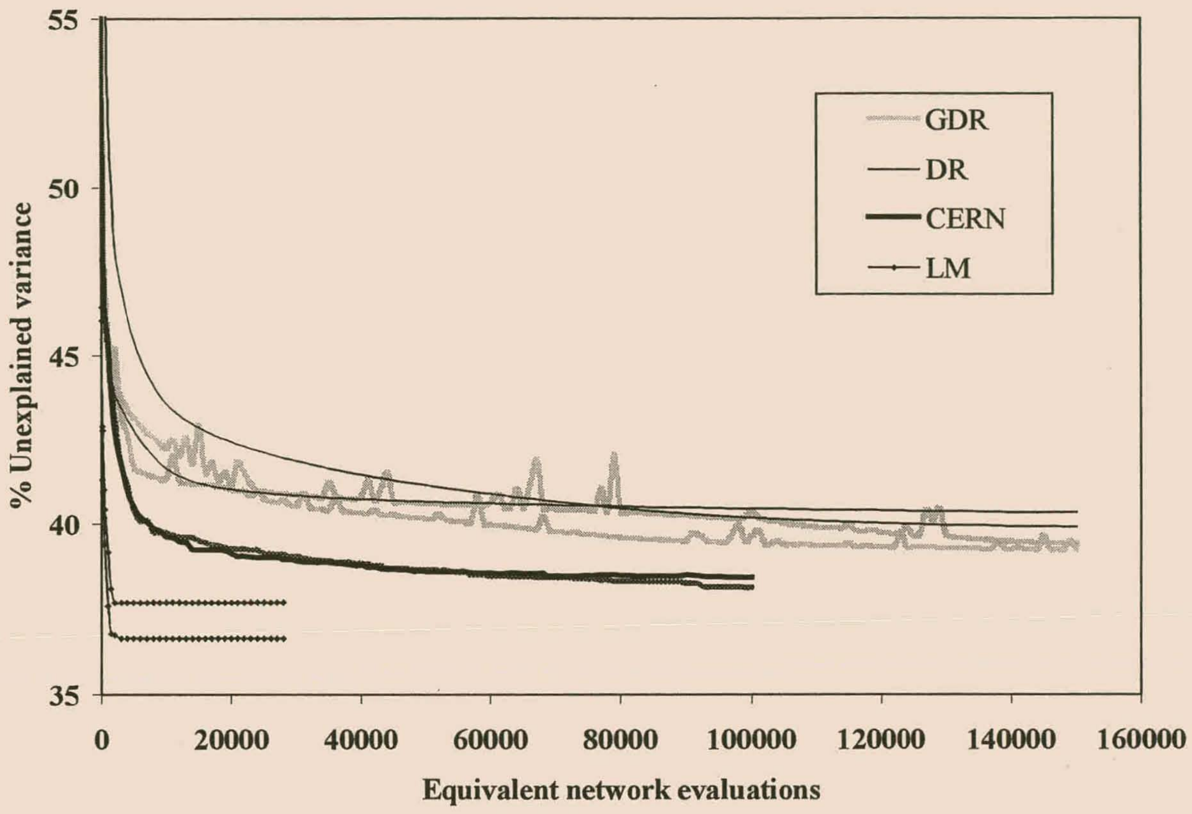

Figure 6.2 The learning curve of the algorithms for the Abalone data set.

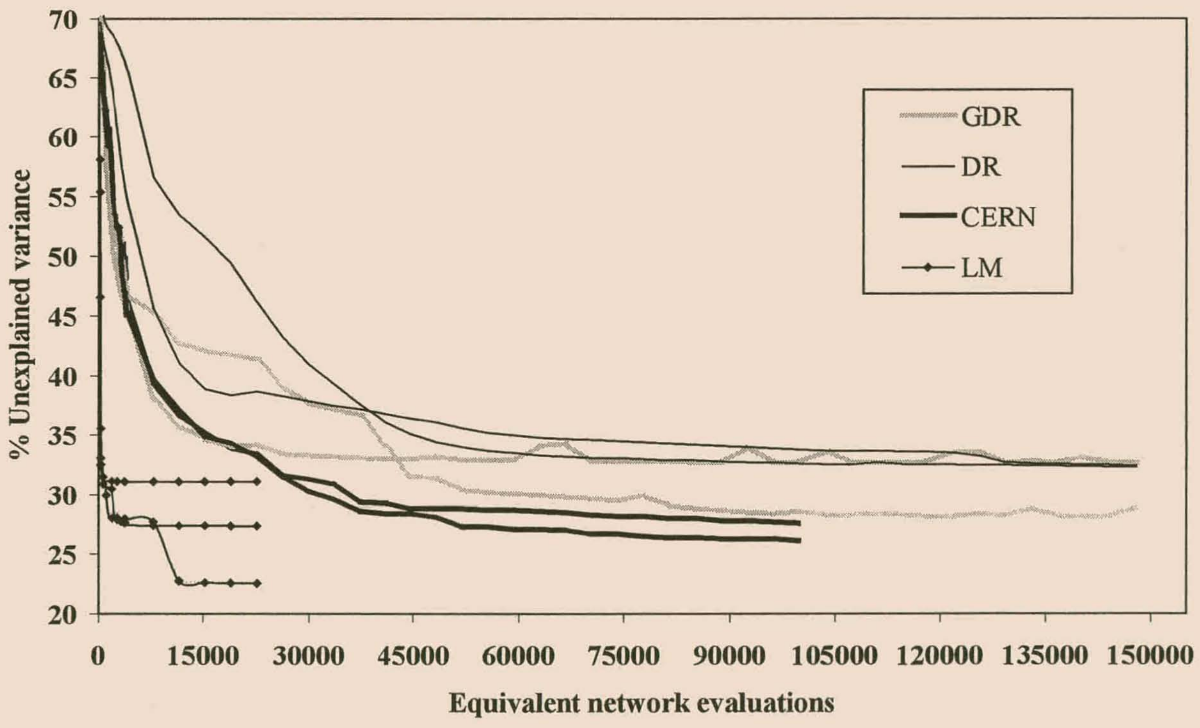

Figure 6.3 The learning curve of the algorithms for the Furnace-1 data set.

The networks of the next data set, Furnace-1, had a much stronger non-linearity. Linear regression on the variables resulted in a model that had an unexplained variance of 0.70, compared to 0.38 to 0.44 for the neural networks. Thus training of these networks was more difficult and all algorithms took longer to find a good solution. This can be seen in figure 6.3, which shows typical training curves of each algorithm. As before, CERN had a consistently lower error than the delta rule with and without momentum. However here the difference was greater and the DR flattened off at a much higher error level. The same also happened to one of the two neural networks trained by the GDR. Most probably the backpropagation algorithm got stuck in sub-optimal minima. Consequently CERN had a significantly lower training error than did (G)DR (95% significance level in an ANOVA test). The Levenberg-Marquardt algorithm converged approximately an order of magnitude faster than the other algorithms. The solutions it converged to sometimes had smaller and sometimes larger errors than the typical solutions found by CERN. That is why the training error of the LM had a higher variance on this data set than did CERN and was on average slightly worse, although the difference was not statistically significant.

With all three problems the CERN trained faster and also found better solutions than DR or GDR. The Levenberg-Marquardt algorithm on the other hand was an order of magnitude faster. For non-linear problems it sometimes got stuck on less optimal solutions. If the problem was highly ill-conditioned, such as the double-spiral data set, CERN attained much smaller training errors than the Levenverg-Marquardt algorithm. On the other hand for the two real world data sets the Levenberg-Marquardt algorithm could sometimes find better solutions more rapidly than the other methods. This is in agreement with findings of Searle (1999): *"In practice, Levenberg-Marquardt often finds better optima for a variety of problems than do the other usual methods."* However, the next section will show that the good performance on the training data of the LM method did not result in better generalisation.

Data set	*DR %*	*GDR %*	*LM %*	*CERN %*
Abalone	59.4 (0.5)	59.8 (1.1)	62.7 (0.37)	61.4 (0.1)
Furnace-1	67.6 (0.3)	69.7 (2.3)	72.6 (2.5)	73.1 (0.8)
Double-Spiral	8.3 (0.6)	15.7 (1.9)	19.5 (1.4)	25.4 (3.4)

Table 6.3 The average final percentage of explained variance (100 × R^2) on each of the three training data sets examined. The standard deviations are given in parentheses.

6.3.3 Generalisation error of CERN and other algorithms

For each of the non-linear models, figures 6.4 and 6.6 show the average explained variance ($100 \times R^2$) on the five data sets with continuous outputs. The classification scores of the Vowel and the Corrosion data set are given in figure 6.8. The vertical lines on these charts indicate the standard deviations. The Levenberg-Marquardt and the generalised delta rule were always used to set-up a multilayer-perceptron model. For simplicity these models are simply indicated by the abbreviations LM and GDR respectively. The delta rule without momentum always performed poorer than the delta rule with momentum and is therefore not shown.

On each of the benchmark data sets, those algorithms that gave the lowest error for the particular data are also included. These are the growing cell structure algorithm (GCS) (Fritzke, 1994), the pseudo-linear radial basis function network (PL-RBF) network trained with the algorithm of Small and Judd (1998) and Gaussian process models trained with a maximum-aposteriori approach (Williams & Rasmussen, 1996), indicated by GP-MAP. The generalisation of a linear model for each of the data sets are given as a further comparison in table 6.4. As can be seen from the table, the linear models had a significantly higher error than the non-linear techniques.

Data Set	*Variance explained or classification on test set obtained by a linear model*
Abalone	52%
Autocatalysis	87%
Corrosion	55%
Furnace –1	27%
Pine	57%
Puma-32mn	21%
Vowel	33%

Table 6.4 The generalisation of linear models on the data sets.

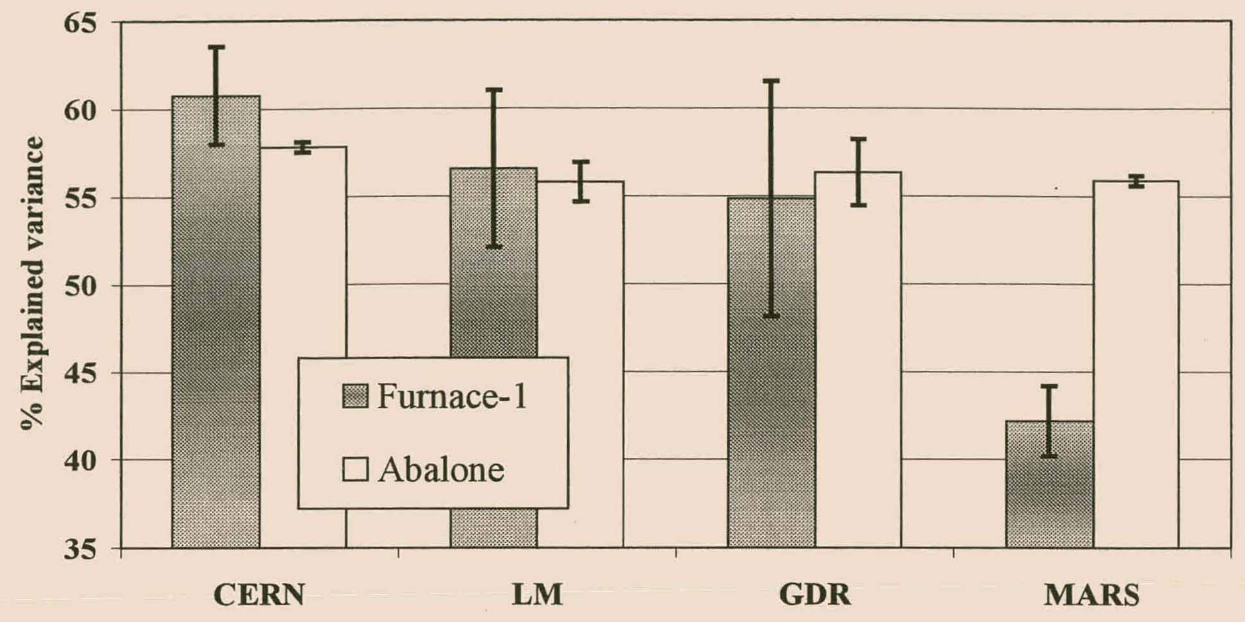

Figure 6.4 The average testing accuracy on the Furnace –1 and the Abalone data set is shown by the bar-chart. The error bars are an indication of the standard deviations of the algorithms.

Using an analysis of variance test, CERN was significantly better than MARS, LM and GDR on the Abalone, Corrosion, Furnace-1 and Pine data set. The significance level was always at least 95%. An exception was the Abalone data set, where the significance level was only 80% for the comparison to the GDR. Also the confidence level was only 90% when compared to the LM and GDR model on the Furnace –1 data set. For this data set a typical plot for the predicted outputs test outputs for both CERN and LM is given in figure 6.5. Compared to CERN the LM model has four predicted values, shown on the bottom right of the figure, which undermine the quality of the fit significantly. If these data points were removed the LM model would be more accurate than CERN.

On the Vowel data set, the LM and MARS results were not available*. The CERN results were however better than that of GDR. Although MARS, GDR and LM were all marginally better than CERN on the Puma-32mn data set these differences were marginal and statistically not significant. On this problem GP-MAP was significantly more accurate than all the other algorithms. Two related techniques that had a similar good performance on this specific data sets

* LM computation was not possible as a result of the high memory requirements that were used by the program and the available MARS version does not run on classification data sets with multiple outputs.

are trained by Bayesian methods using Monte-Carlo algorithms (Rasmussen, 1996). The models were Gaussian processes for regression (GP-MC) (Williams & Rasmussen, 1996), as well as multilayer perceptrons (MLP-MC) (Neal, 1996). A drawback is that the execution time of all these algorithms is proportional to the square or cube of the number of data points *(n)* and their storage requirement is proportional to the square of *n*. (Williams & Rasmussen, 1996). Although the Puma-32mn data set has only 1024 training cases, this can become a limiting factor for larger data sets.

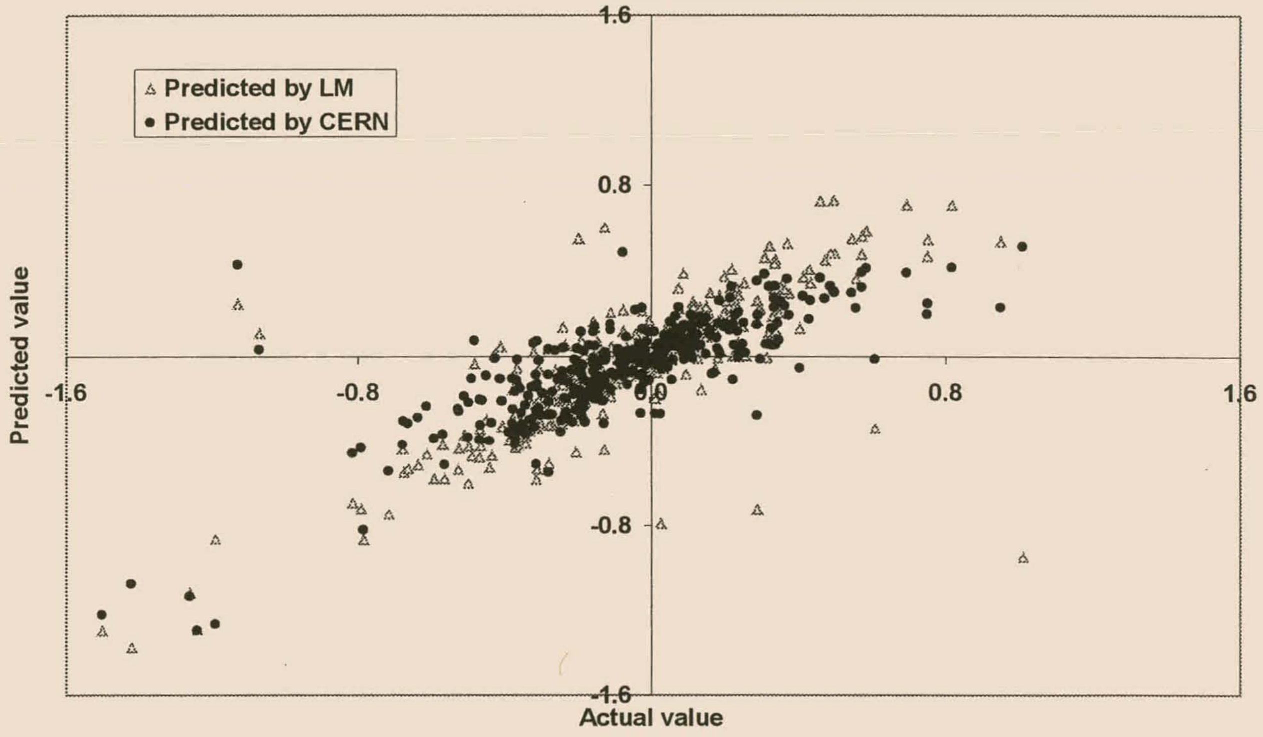

Figure 6.5 The rescaled predicted and actual values of the furnace-1 test data set for the multilayer perceptrons optimised by CERN and by LM.

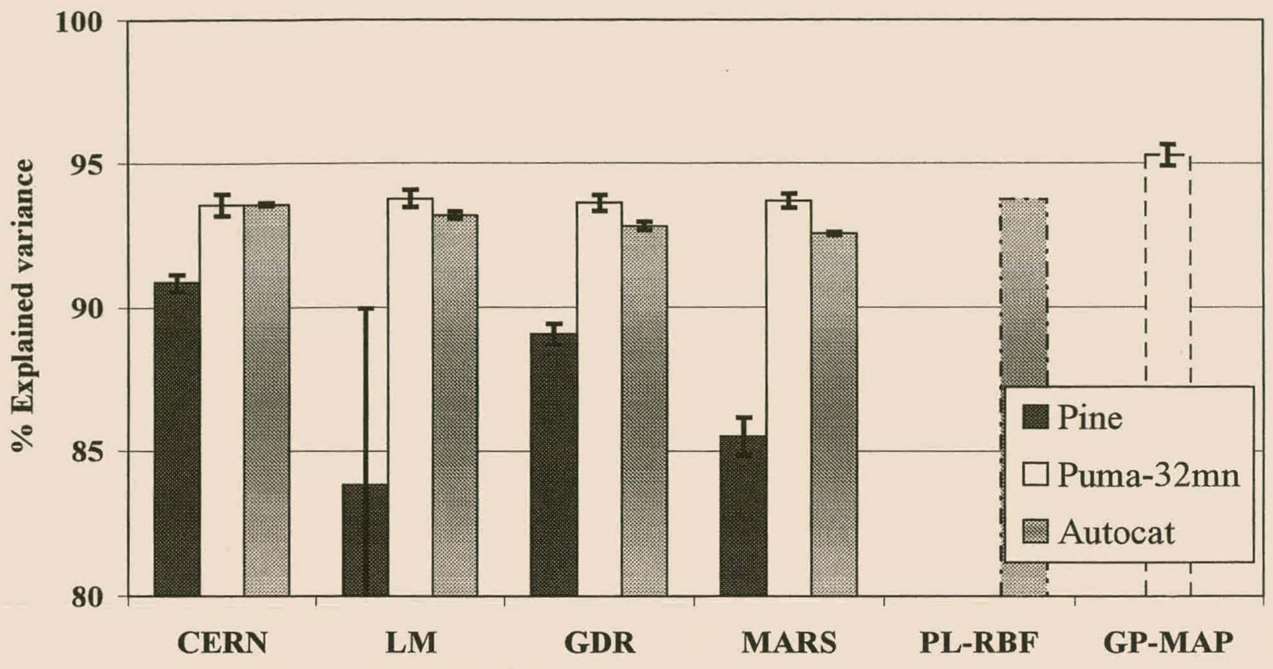

Figure 6.6 The average testing accuracy on the Pine, Puma-32mn and the Autocatalytic data set. The error bars are an indication of the standard deviations of the algorithms. The broken lines show results obtained from the literature.

The GCS had a significantly higher accuracy than the other algorithms on the Vowel data set. It should be noted that the GCS used a classification error during training and that these results were peak performances. A similar technique, the dynamic cell structure - growing cell structure (DCS-GCS) also obtained a peak performance of 65% classification (Bruske & Sommer, 1995). For such a data set with unordered classes, the CERN algorithm was not the most suitable algorithm. In particular, the k-nearest neighbour algorithm also achieved a slightly higher accuracy (classification score 56%). The PL-RBF network using minimum description length principles was slightly more accurate than CERN on the Autocatalytic data set by (0.2%). The significance of this could not be tested, as the standard deviation was not given with the result. However other statistics relevant to dynamic data sets were supplied (Barnard et al., 1999). The predicted values of two of the models are plotted in figure 6.7.

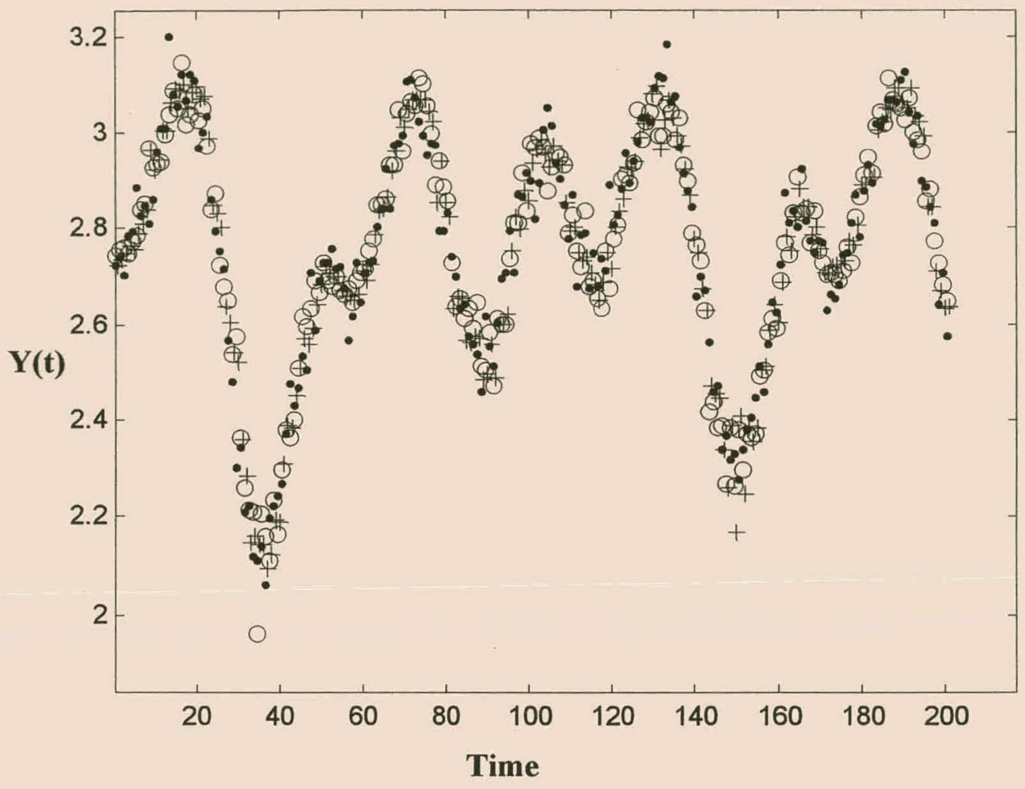

Figure 6.7 The outputs of two models for the last 200 test data points of the autocatalytic time series, predicted 9 time steps ahead. The actual time series is indicated by a •, the output of MARS by a o and the output of CERN by a +. A plot on the previous test points shows a similar picture.

When only regarding those data sets for which a multilayer-perceptron was evolved, (Furnace, Corosion, Abalones, Puma-32mn), CERN was on average always better than both LM and BP, with the exceptions of the Puma-32mn data. As was said above, LM and GDR were marginally better on this data set, but the difference was not statistically significant.

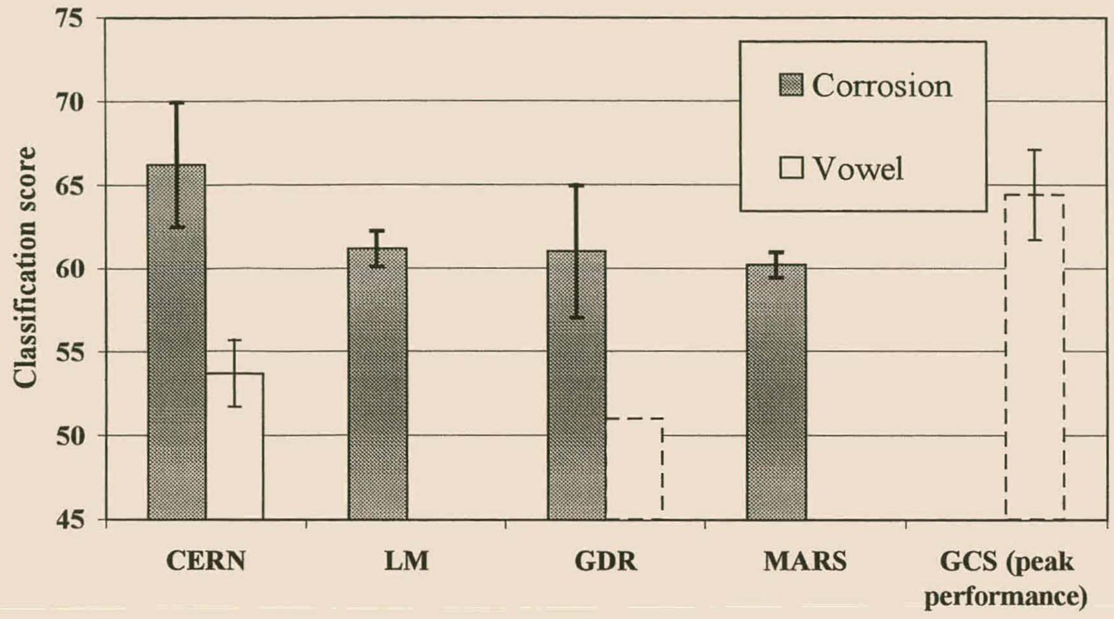

Figure 6.8 The average classification score on the Vowel and the Corrosion data. For some algorithms the results were only available on one of the data sets. The error bars are an indication of the standard deviations of the algorithms. The GCS result indicated by broken lines was obtained from the literature and shows the peak classification score of the algorithm.

6.3.4 Complexity and comprehensibility of fuzzy models generated by CERN

6.3.4.1 Fuzzy rules for the Puma-32mn data set

While using validation sets to estimate the angular acceleration of the robot arm it was found that it could be estimated just as well using only two of the three selected inputs. These were the torque on joint 4 and the angle between joint 5 and 6 and will simply be referred to as torque and theta. The percentage of unexplained variance for the Puma-32mn data set for the various fuzzy systems is given in table 6.5. One can see that most of the algorithms performed similarly in terms of the accuracy. The radial basis function networks trained by the unsupervised k-means clustering required many more parameters in order to achieve the same accuracy as the systems of CERN and ANFIS. Although ANFIS required four rules, as opposed to the 6 ellipsoidal rules of CERN, the rules are all of 1st order and each is thus a linear combination of the inputs and a bias. ANFIS did however have fewer continuous parameters (20) than the 0th order Sugeno system of CERN with 31 parameters. By merging the membership function of the EBF network, described in the next section, the number of parameters could be reduced slightly (to 26) with no loss of accuracy.

The rules before merging of the 0th order Sugeno system evolved by CERN with six axis parallel rules are shown in figures 6.9(a) - 6.9(f). The merged rules do not differ significantly. The model only uses two of the inputs and each of the rules' activation can be drawn in a 3-D surface plot against the torque and the angle theta. These rules then combine to form the final output surface that is shown in figure 6.10. It can be clearly seen how the six Gaussians multiplied by their consequents add up to form the surface. The bias was relatively small and had a value of -0.027.

Model and training method	Number of rules	Unexplained Variance (%)	Number of parameters
RBF with k-means	40	6.2	163
	30	6.1	123
	20	7.1	83
OEBF with CERN	10	6.1	61
	6	6.2	37
EBF with CERN	10	6.1	41
	6	6.1	31
	4	6.6	21
ANFIS	4	6.1	20
	9	6.2	39

Table 6.5 The percentage of unexplained variance for the test sets of the Puma-32mn problem.

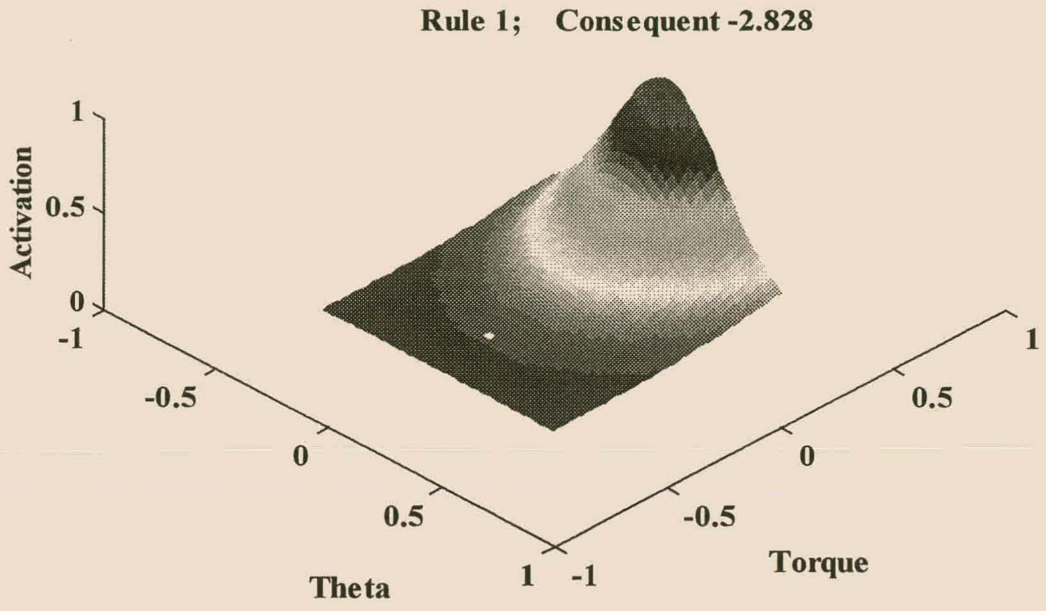

Figure 6.9(a) The activation of rule 1 in the input space of the Puma-32mn problem. The activation increases monotonically from 0 to 1 for this and all subsequent figures.

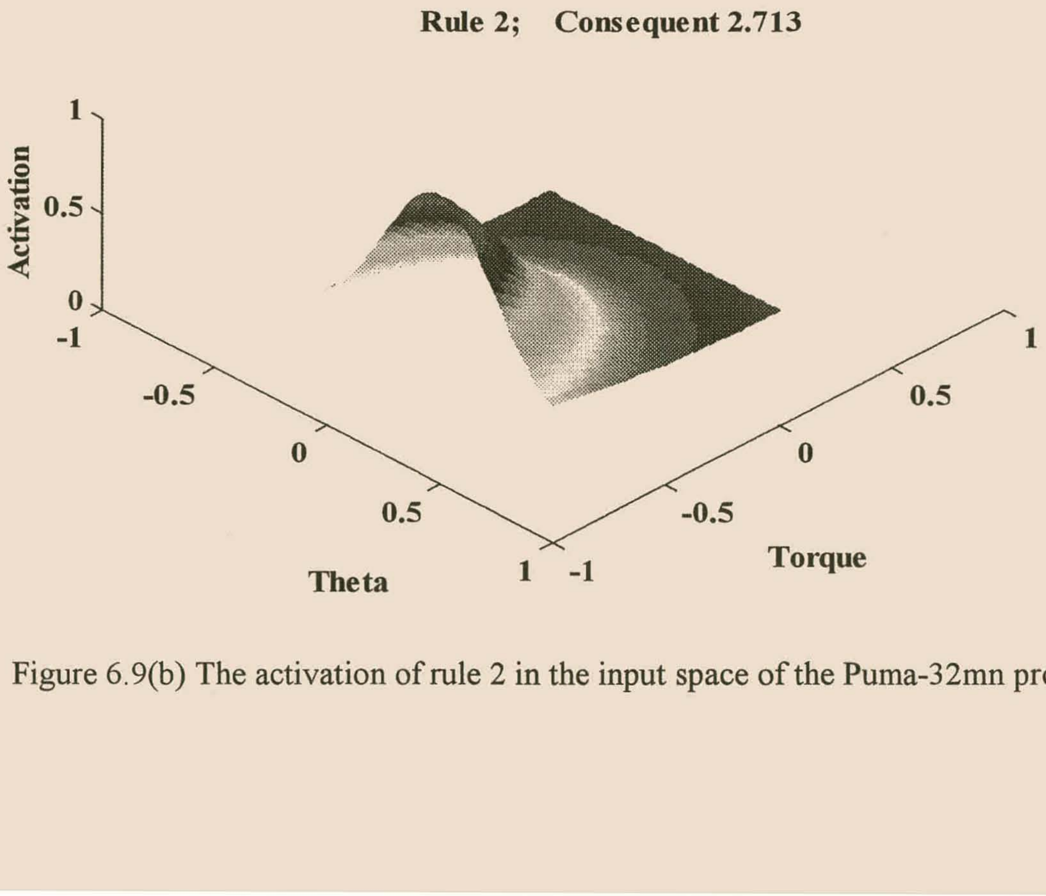

Figure 6.9(b) The activation of rule 2 in the input space of the Puma-32mn problem.

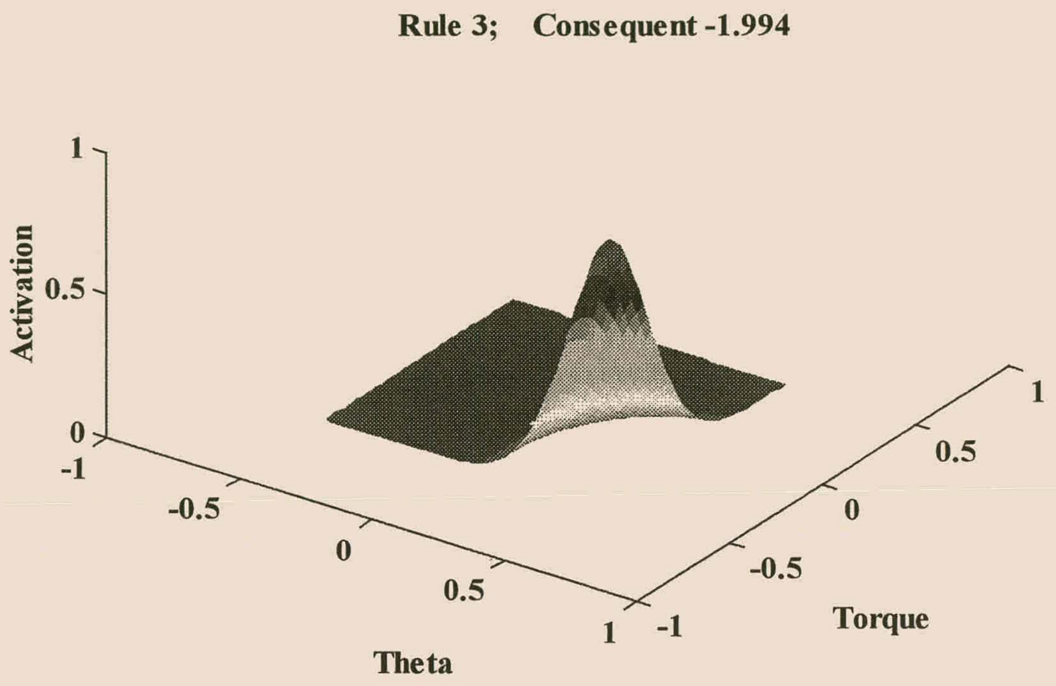

Figure 6.9(c) The activation of rule 3 in the input space of the Puma-32mn problem.

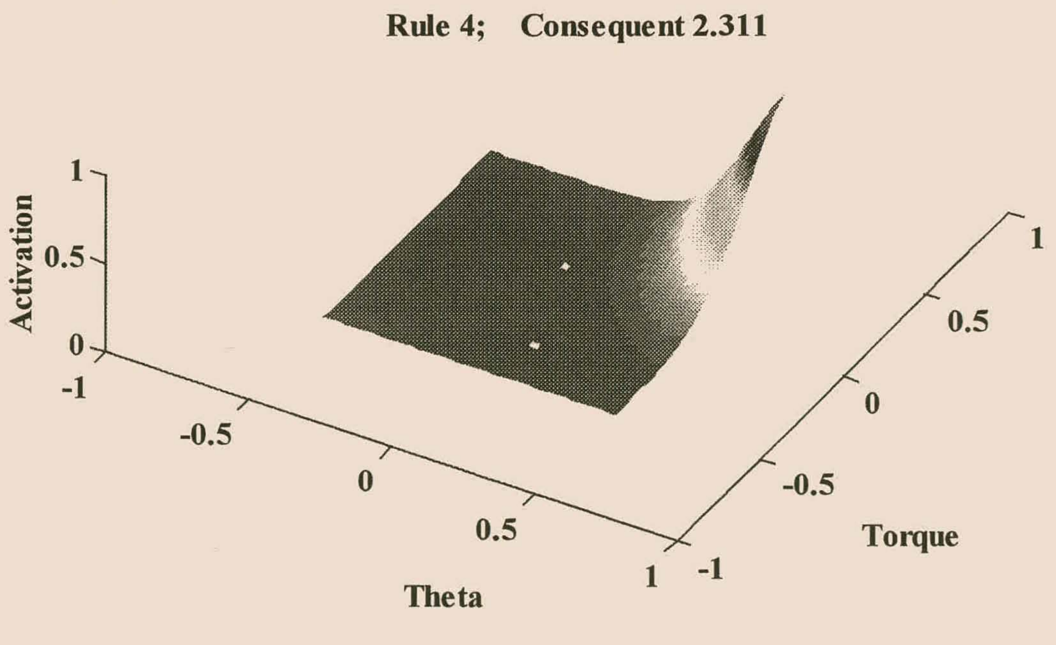

Figure 6.9(d) The activation of rule 4 in the input space of the Puma-32mn problem.

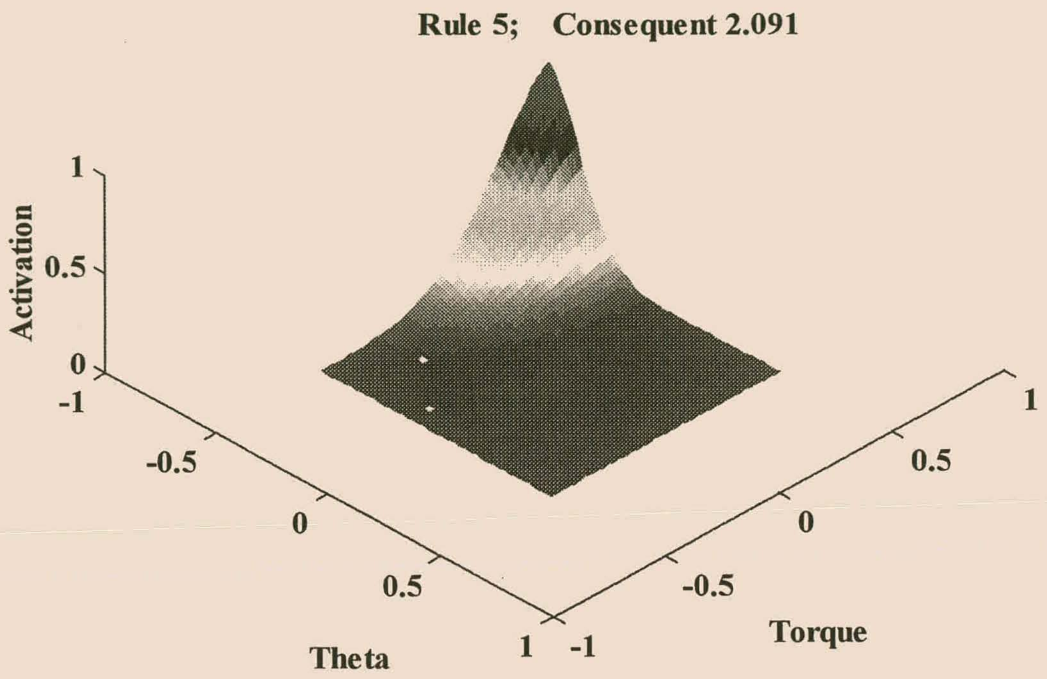

Figure 6.9(e) The activation of rule 5 in the input space of the Puma-32mn problem.

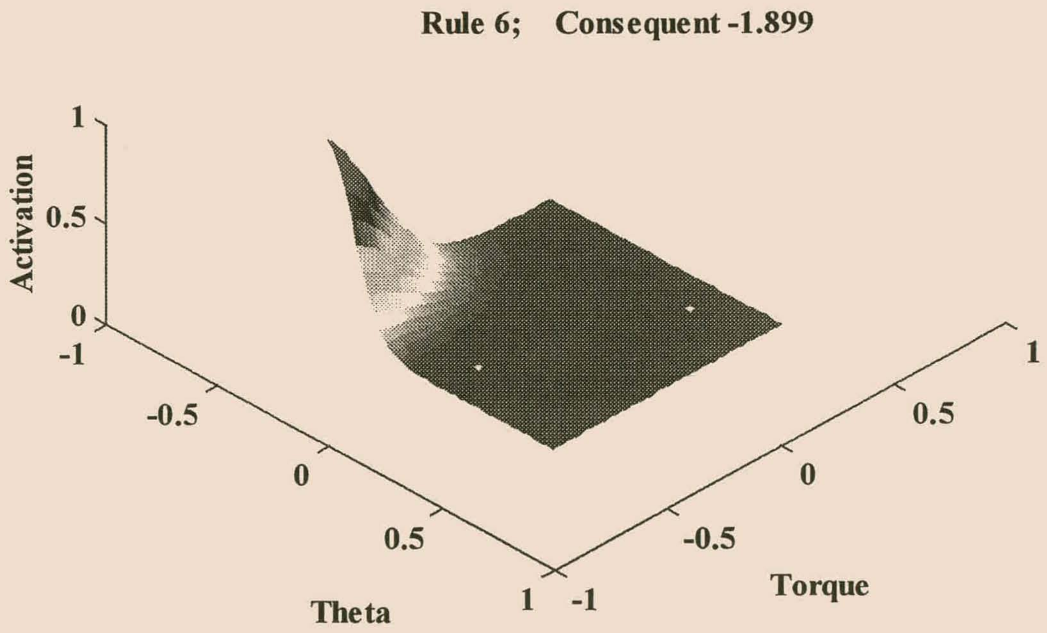

Figure 6.9(f) The activation of rule 6 in the input space of the Puma-32mn problem.

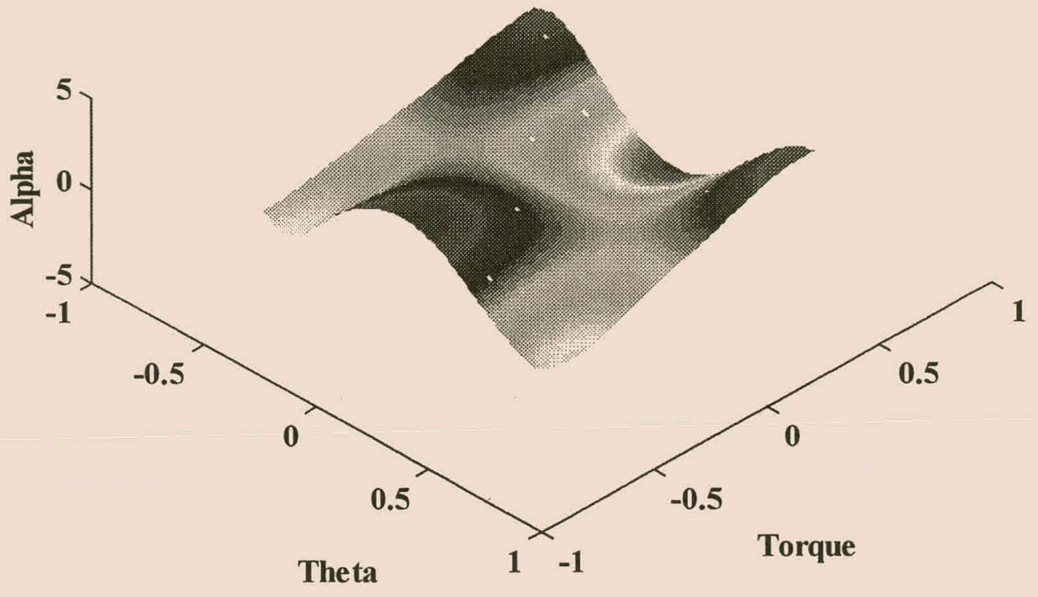

Figure 6.10 CERN's estimated angular acceleration of the sixth joint for the Puma32-mn data set, plotted against theta of the fifth angle and the torque of the fourth joint.

6.3.4.2 The rules for the induced Aeration

Table 6.6 gives the percentage of unexplained variance of the various algorithms on the five combined tests sets of the Aeration data. The table suggests that the networks of ellipsoidal basis nodes evolved by CERN are superior to the spherical basis nodes obtained by *k*-means clustering, both in terms of accuracy and the number of nodes required to achieve good network performances. ANFIS with $u = 2$ achieved much better accuracies then when the number of membership functions per attribute (*u)* was three. In the latter case the grid-partitioning resulted in too many free parameters and the model overtrained. The accuracy of ANFIS using 16 rules was the highest of all the fuzzy systems. However both the accuracy (% unexplained variance 0.9) and the complexity (93 parameters) of ANFIS can be compared to the best set of rules found by CERN (% unexplained variance 1.0 using 103 parameters). In analysis of variance test both CERN and ANFIS were found to be significantly more accurate than the RBF networks with a 95% confidence level.

Model and training method	Number of rules	Unexplained Variance (%)	Number of parameters
RBF with k-means	40	2.2	205
	30	1.9	155
	20	3.4	105
	7	9.0	41
OEBF with CERN	7	1.0	103
	4	1.6	61
EBF with CERN	12	1.1	109
	7	1.2	64
ANFIS	16	0.9	96
	81	8.0	429

Table 6.6 The percentage of unexplained variance for the Aeration test data sets.

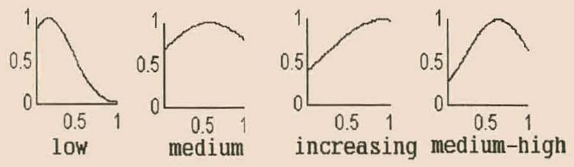

Figure 6.11(a) Membership functions of the Froude number.

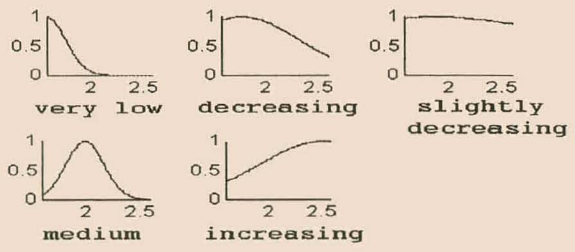

Figure 6.11(b) Membership functions of the density.

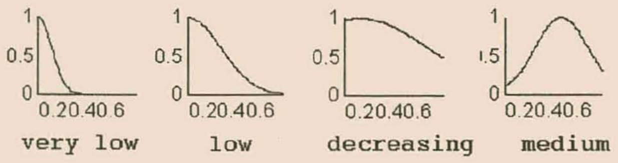

Figure 6.11(c): Membership functions of the viscosity.

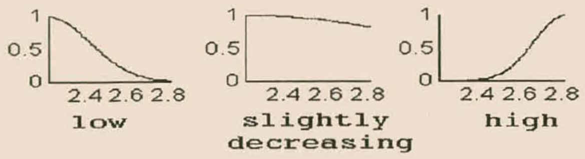

Figure 6.11(d): Membership functions of the draft tube diameter.

A very compact (64 parameters) yet accurate (error: 1.2%) set of rules was found by CERN when optimising seven EBF nodes. The corresponding rules for the six- blade impeller are given in figure 6.12. The membership functions shown in figure 6.11 resulted from post-processed by a simple algorithm (Schmitz & Aldrich, 1998). When evolving the membership functions it can happen that two membership functions of a certain input arising from two different ellipsoidal units are very similar. It is possible to replace the two membership functions by a single one in order to simplify the system. After this merging the consequents of the new rules are determined again by regression. The aim of the merging is not so much to prune the system in order to obtain better generalisation capabilities, but rather to form a more transparent system. One does not distinguish between what are essentially the same membership functions, which would unduly complicate the rule base. This also reduced the number of parameters of the model further. The accuracy was not affected much, changing from an error of 1.23 % to 1.26%.

Unlike the Puma-32mn data set, the rules overlap considerably and all the rules have to be considered together. This is as a result of the neurofuzzy design and the weighted sum consequent determination, which resulted in rules that are more distributive. This is not typical of a pure fuzzy logic design philosophy. The first two rules are particularly striking, as they extend over the whole data space. The first rule is in fact the bias. From figure 6.12 it can be seen that the other Gaussians have much smaller outgoing weights (numbers in brackets after each rule) and explain those trends in the data not explained by the first rule.

For all **Aeration rate** decreases (12.6)
For **FR-FR$_c$** increasing, **Draft-Tube Diameter** decreasing slightly, **Density** decreasing slightly and **Viscosity** decreasing **Aeration rate** increases (62.7).
For **FR-FR$_c$** medium, **Draft-Tube Diameter** low, **Density** increasing and **Viscosity** low **Aeration rate** decreases (9.4)
For **FR-FR$_c$** medium to high, **Density** very low and **Viscosity** low **Aeration rate** increases (7.9)
For **FR-FR$_c$** increasing, **Draft Tube Diameter** high, **Density** medium and **Viscosity** very low **Aeration rate** decreases (9.4)
For **FR-FR$_c$** medium, **Draft Tube Diameter** low, **Density** increasing and **Viscosity** medium **Aeration rate** decreases (16.5)
For **FR-FR$_c$** low and **Viscosity** decreasing **Aeration** increases (7.9)
For **Density** decreasing and **Viscosity** low **Aeration** decreases (9.3)

Figure 6.12 The seven post-processed rules for the six blade impeller from the ellipsoidal radial basis function neural network induced by the CERN algorithm.

6.3.4.3 The rules for the estimation of Miles/Gallon

For the Miles/Gallon data set, the ANFIS algorithm optimised a fuzzy system with 1052 free parameters. There are only 320 data points in each of the training sets and ANFIS therefore clearly overtrained. This can be seen from the summary of the results over the five validation runs, given in table 6.7. Both EBF-CERN and the radial basis function networks obtained similar accuracies for this problem. The four orientated ellipsoidal nodes optimised by CERN obtained slightly better results on the test set. When inspecting the number of parameters, it can be seen that the set of rules corresponding to the OEBF and the EBF networks optimised by CERN used roughly two thirds of the number of parameters for the same accuracy as the RBF networks trained by the unsupervised method.

The eight axis parallel EBF nodes were translated into eight corresponding fuzzy rules. Figure 6.13(a) shows the membership functions of the first rule. In this figure membership functions that have a constant value of 1.0, indicate rules that apply throughout the whole input range and these particular membership functions can thus be omitted. The 1st rule states that cars with small weight, a fairly small number of cylinders and volume of displacement and small or medium power have a high Miles/ Gallon. This is in agreement with our knowledge of fuel consumption. In contrast from figure 6.13(b) one can see that old cars with lots of power and a heavy weight have high fuel consumption. This is also plausible. The remaining six fuzzy rules for the Miles/Gallon data set are given in figure C.1-C.6 of Appendix C. The bias had a value of –0.42.

Again it can be seen that the rules overlap somewhat. Thus sometimes more than one rule has to be considered at once.

Model and training method	Number of rules	Unexplained Variance (%)	Number of parameters
RBF with k-means	50	14.7	508
	30	14.0	308
	20	14.0	208
	12	16.3	128
OEBF with CERN	6	15.3	217
	4	12.8	145
EBF with CERN	10	15.2	171
	8	13.8	137
	6	14.8	103
ANFIS	128	100	1052

Table 6.7 The unexplained variance for the Miles/Gallon data, based on the test sets.

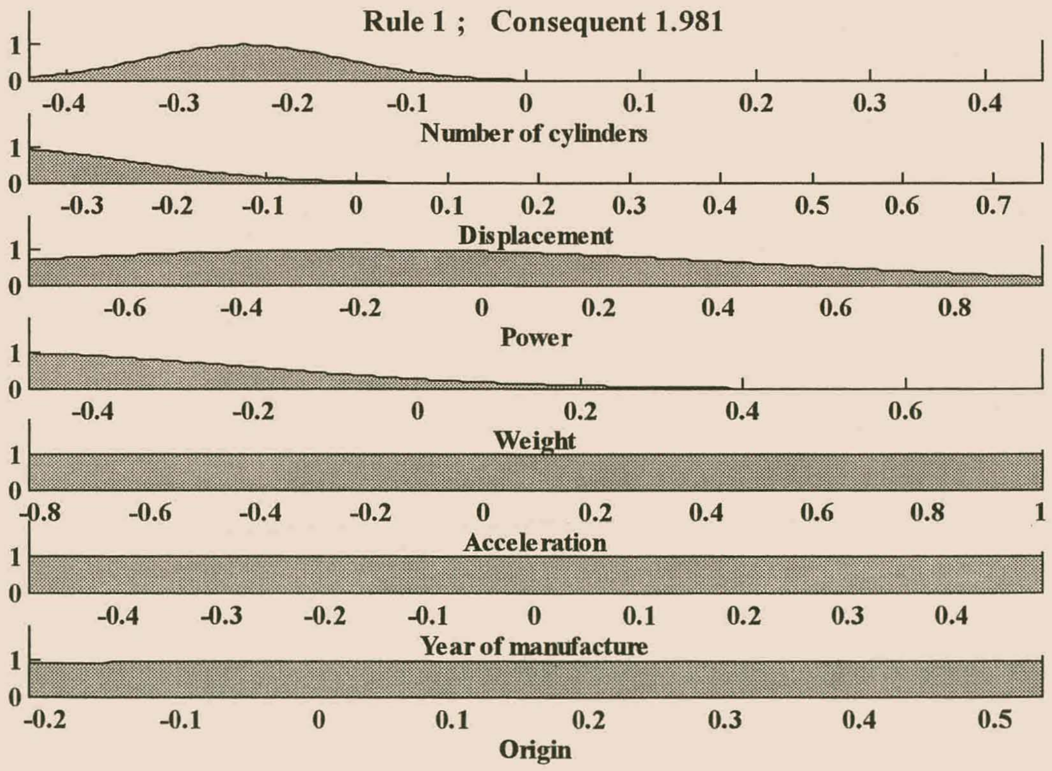

Figure 6.13(a) The membership functions of the first rule for the Miles/Gallon data. All the inputs and the output were rescaled.

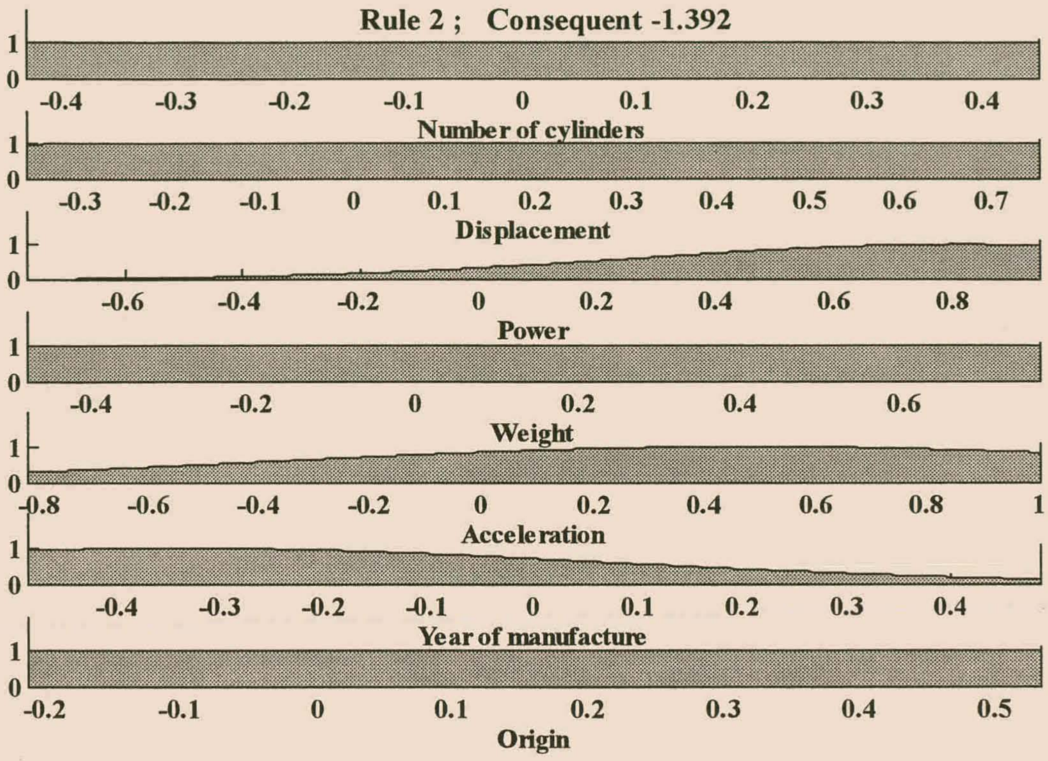

Figure 6.13(b) The second rule for the Miles/Gallon data.

6.4 Discussion

CERN requires roughly as long for one network evaluation and the accompanying updates as does the generalised delta rule. However, CERN becomes slower with an increasing number of nodes requiring simultaneously optimisation. On the other hand, relative to GDR it requires less computation with an increasing dimensionality of the data. In the analysis of the training process it was found that the CERN algorithm trained faster and also manoeuvred the network to better solutions than the (G)DR. The CERN algorithm learns without requiring a gradient and can therefore be used to optimise a wider range of parameters. In contrast, updating by means of the Levenberg-Marquardt rule converged an order of magnitude faster than CERN. Although the learning rule could sometimes obtain better solutions it also got stuck more easily in non-optimal solutions. This occurred especially with the ill-conditioned Double–Spiral data. Although the Levenberg-Marquardt algorithm could train better on the real-world data sets, the resulting networks often gave a poorer performance on the test set than CERN and also than GDR. This occurred despite the fact that no account was taken of the instance where the Levenberg-Marquardt was obviously stuck in very sub-optimal minima. The Levenberg-Marquardt algorithm therefore tends to overtrain very quickly and could possibly be improved by adding a penalty such as a weight-penalty term. However, this would make the solution space more ill-conditioned. A possible cure for this dilemma is given in Leung & Chow (1999). Another possibility is not to use the validation data during actual training and to use early stopping based on the performance of the network on the validation set (Tetko, 1995). When this method is employed not all the available data points can be used directly to train the network.

CERN, on the other hand, finds networks that generalise better than those using GDR and Levenberg-Marquardt. CERN is a global search technique. It will therefore tend to settle in areas that have a relatively small error, rather than run off into some narrow valley in the error surface. These areas with a relatively low error are more stable and often also result in networks with better generalisation. Nevertheless, on occasion, the Levenberg-Marquardt algorithm found neural networks that gave both a smaller training and a smaller testing error than CERN. It is thus possible that CERN failed to move to nearby good solutions and could possibly benefit from a local gradient search around the best solutions it found. This will have to be examined in the future.

It could be argued that the ridge of CERN, which was set very small at 0.005, in order to improve the conditioning of the matrix and make the inversion of it more stable, could have had a positive effect on the generalisation. Therefore all the results on the test data set were repeated with a ridge as small as 0.0005. The results obtained were at least as good as those with the previously used ridge. Using an even smaller ridge can have the disruptive effect that for some runs the matrix inversion becomes numerically unstable or notable rounding errors occur. As can be clearly seen from figures 6.4, 6.6 and 6.8, CERN was also significantly better than MARS. However, MARS can directly assign an importance to each of the features and may therefore be better suited for a sensitivity analysis, e.g. for selecting the most important variables.

Networks of ellipsoidal basis nodes evolved by CERN can be translated into a zero order Sugeno system. In the three cases studied, the resulting fuzzy system were always found to be at least as accurate or significantly more accurate than those of the radial basis function networks, partially trained by the k-means clustering algorithm. Furthermore, for the same accuracy the fuzzy systems were much more parsimonious than the radial basis function networks. For two of the cases studied, ANFIS obtained rules that were of comparable accuracy and complexity as CERN's rules. The best rules based on the selected two attributes of the Puma-32mn data set had 20 parameters as opposed to 31 parameters for CERN and were even more parsimonious, for the same accuracy. This is essentially a 2-dimensional problem and shows a lot of symmetries, therefore the grid-partitioning of ANFIS suits the problem well. However, the grid-partitioning of the input space results in an explosion of parameters for dimensions greater than six. This happened for example for the Miles/Gallon data set. CANFIS (Jang et al., 1997) uses CART to partition the inputs. In this way CANFIS depends on CART to partition the inputs space in a suitable fashion, but avoids the exponential increase of the number of rules with dimensionality. In these experiments only 1st order Sugeno systems were built by ANFIS. These are also slightly more difficult to comprehend than zero order systems. Unfortunately the software did not allow for the latter.

CERN uses the weighted sum consequent determination. In certain cases, such as the Aeration data, where the fuzzy rules overlapped considerably, it made the comprehensibility more difficult. This should be considered when using weighted summations in fuzzy systems. Nevertheless, the fuzzy rules for the Puma32-mn data set were very comprehensible. For the Miles/Gallon data set, it could be seen that the rules could be understood in a qualitative fashion. This can for example enable an expert of a given process to verify the rules in a qualitative fashion.

6.5 Conclusions of Chapter 6

- In the case studies analysed, the CERN algorithm with clustering and copying of individuals trained neural networks faster than could (G)DR.
- The Levenberg-Marquardt algorithm converged an order of magnitude faster than the CERN algorithm. It also found more accurate solutions on the training data sets, but the resulting networks are not necessarily better for generalisation onto unseen data points. In addition for very ill-conditioned data sets, the Levenberg-Marquardt algorithm tended to get stuck in local minima.
- It should be kept in mind that CERN does not require gradient information and it can thus be used to optimise a wider set of parameters than the GDR and the Levenberg-Marquardt algorithm.
- In the cases studied, generally CERN trained the networks to generalise better or as well as both the GDR algorithm and the Levenberg-Marquardt algorithm.

- The networks trained by CERN generalised better than the models of the MARS algorithm. Unlike CERN, MARS explicitly assigns an importance measure to the variables and for certain problems with many inputs it could be used to select the variables.
- The EBF and OEBF neural networks evolved with the CERN algorithm constituted either significantly more accurate or significantly more compact representations than those obtained by means of networks containing spherical Gaussians trained with the k-means clustering algorithm. The complexity and accuracy of the rules generated by ANFIS were comparable to those of CERN for the data sets with two and four dimensions. The grid-partitioning employed by ANFIS led to an extremely large number of parameters for the Miles/Gallon data set with seven input features and the resulting generalisation was poor.
- The axis parallel ellipsoidal basis nodes could moreover be translated to parsimonious fuzzy rules. These can be condensed somewhat more through appropriate post-processing.
- In certain cases the fuzzy rules generated by the EBF neural networks exhibited considerable overlap. The combination of high overlap and the weighted sum defuzzification process made the comprehension more difficult.

Chapter 7 Summary and conclusions

In the first chapter non-parametric modelling was compared to parametric modelling. Although parametric models can usually extrapolate better, they require explicit knowledge about the process, such as non-linear interactions. However, should the process deviate from the assumed structure parametric models can perform very poorly. Non-parametric or weakly parametric models are more flexible and do not require as much knowledge of the process. Chapter 2 discusses some non-parametric models, in particular neural networks, and so-called regression networks as well as their use in the field of chemical engineering. This thesis focuses on a novel algorithm that builds and optimises models in the form of a regression network.

Problems with existing optimisation algorithms for neural networks are discussed in Chapter 3. Local search algorithms, such as gradient descent techniques, can possibly get stuck in local minima. Furthermore, they usually require gradient information and this limits the type of network parameters that can be optimised. Global optimisation techniques suffer more than local techniques from the problem of permutational redundancy. For this and other reasons global search techniques, such as evolutionary algorithms, are often too slow. It is therefore necessary to reduce the size of the search space. One such method is to encode each network node as an individual. This requires solving the niche and credit assignment problem. A further way to reduce the search space is to optimise the nodes of the network one at a time. This has the disadvantage that optimising one node at a time will not necessarily lead to the global optimum.

In chapter 4, a novel algorithm is proposed viz. "**C**ombinatorial **E**volution of **R**egression **N**odes", CERN. This evolutionary based algorithm optimises groups of nodes simultaneously. Unlike optimising single nodes at a time, which is essentially a greedy approach, this enables a lookahead during the optimisation of the nodes. Each node is encoded as an individual in the population. The main novelty of CERN lies in the combinatorial selection scheme that it uses to select new groups of nodes from the trial and current population. As far as the author is aware, such a combinatorial search has not been applied before in non-linear regression. The problem of permutational redundancy could thereby be solved, and the selection scheme also guarantees that the best combination of nodes is selected from the competing individuals. The credit assignment problem is not solved. Strictly speaking the evolutionary algorithm does not require a fitness value for each individual, but only a selection criterion, which is supplied by the combinatorial search. In order to put to use the fast combinatorial search, regression by leaps and bounds, the output node needs to be linear.

CERN was compared to a DE algorithm without a combinatorial selection scheme. For the same task CERN minimised the error of the neural network an order of magnitude faster. It was shown that the superior performance of CERN was a result of the superior selection scheme used. On two synthetic and one real world data set, CERN achieved good results. In particular, orientated ellipsoidal basis nodes trained by CERN, achieved the lowest reported error on the Mackey-Glass time series, to date.

CERN's learning can be improved by clustering the individuals (representing nodes). This actively supports niches and allows for different nodes in the same population. Copying over also adds to the learning rate. The effects of various parameters on the training and on the predictive capabilities were examined and the best set of default parameters for CERN can be found in table 5.3.

Chapter 6 provides a comparison between CERN and other algorithms in terms of floating point operations, rate of learning and predictive capabilities of the resulting networks. CERN requires roughly as long for one network evaluation and the accompanying updates as does the generalised delta rule (GDR). CERN becomes more expensive as g (the number of simultaneous nodes to optimise) increases, especially when it increases beyond 10. On the other hand, it requires fewer floating operations for data sets with a high dimensionality d. The number of floating point operations of the Levenberg-Marquardt (LM) update rule are $O(d^2 \times g^2)$ and the updates therefore become increasingly expensive for networks with many nodes including the input nodes.

CERN trains faster than GDR and does not get stuck as often and is therefore more efficient. In the experiments that were conducted the LM learning rule was an order of magnitude faster than CERN. However, it also got stuck in sub-optimal minima, especially for ill-conditioned problems. CERN does not make use of gradient information and therefore does not get stuck as easily. As a result it can be used to optimise parameters for which no gradient exists. Tasks of this type include the pruning of weights, optimising nodes with hard-threshold activation functions, choosing the activation function type, using different node types, and optimising nodes such as those used in MARS that have a product in one or two or three variables, but not in all. Furthermore, CERN is also suitable for ill-conditioned error surfaces with many local minima, such as those found in the hidden layer of radial basis function networks (McLoone, 1996).

For five real world and three synthetic data sets, the accuracy of the predictions on the test data of CERN was compared to LM, GDR and MARS algorithms. In all the problems CERN generalised significantly better or as well as these techniques. Since the LM algorithm performed better or at least as well on the training data sets, but worse on the test sets, it was concluded that the LM tended to overtrain the neural networks. This happened despite the fact that the number of epochs and the number of hidden nodes were fine-tuned on a validation set. It therefore requires either an error function that gets penalised by large weights or the use of early stopping, determined by a validation set during training. The former method would make the error function more ill-conditioned, while the latter requires that some of the training data must be retained to determine the best point at which to stop training. It must also be mentioned that LM could sometimes find solutions with a smaller training and testing error than CERN. It is thus possible that for these instances CERN failed to discover good solutions nearby and could benefit from a local gradient search around the best solutions it found.

In a further comparison CERN was used for fuzzy model generation. The EBF and OEBF neural networks evolved with the CERN algorithm constituted either significantly more accurate or significantly more compact representations than those of radial basis function networks trained with the k-means clustering algorithm. The complexity and accuracy of the rules generated by the adaptive neuro fuzzy inference system, (ANFIS) algorithm were comparable to those of CERN. For one of the data sets the grid-partitioning employed by ANFIS resulted in a very large number of parameters and the resulting generalisation was consequently much poorer than that of the CERN models. The axis parallel ellipsoidal basis nodes of CERN can be translated to parsimonious fuzzy rules. These could be condensed even further through appropriate post-processing. In certain cases the fuzzy rules generated by the EBF neural networks exhibited considerable overlap and the fuzzy systems lost some of their comprehensibility.

In all the experiments of chapter 6 CERN was always used with default settings, a single hidden layer, a predefined transfer function. (hyperbolic tangent for summation nodes and a Gaussian for the local, radial or ellipsoidal, basis nodes) and no connection from the input to the output layer. The number of nodes placed in the hidden layer was obtained by a single run using the outputs on the validation set. Repeated experiments to estimate the optimal number of hidden nodes were only necessary where less than six nodes appeared optimal. The type of non-linear nodes to use also had to be selected, e.g. summation, basis or mixed. This fine-tuning was found to be of less effort than that needed for the LM and the GDR algorithm. Here the number of hidden nodes and the number of iterations had to be carefully tuned in many more experiments to prevent overfitting. However this was more a problem of optimising the entire network simultaneously (instead of incrementally growing the networks) and of the software than of the actual update rule.

7.1 Future Work

- The lookahead is currently limited by the combinatorial search. As *g* (the number of nodes that get optimised simultaneously) becomes greater than 10, the combinatorial search will take up an ever increasing proportion of the CPU time. A possible method to increase the size of the groups (*g*) is to change the combinatorial search. The possible combinations can be restricted to look at only those combinations in which each node replaces its parent node.
- CERN selects the nodes based on the residual sum of squared errors. This could be adjusted by adding a penalty term based on the number of parameters the nodes use, as well as the size of the weights of the output node. The combinatorial search would need to be adjusted accordingly, as suggested above.
- CERN could be used to optimise the kind of models that the MARS algorithm finds. The basis nodes are axis parallel in most dimensions. That is their outputs are constant except in one, two or three variables. This is a question of choosing the right connections to the input nodes can therefore not be optimised in a straightforward manner by a gradient-based search. CERN could be used to optimise the connections to the input nodes for summation nodes or the nodes used in the multivariate regression splines. Unlike the MARS algorithm, which is 'greedy', this could then be done with a lookahead.
- The merits of CERN could be combined with the Levenberg-Marquardt algorithm into a single optimisation algorithm. CERN could be used to optimise those parameters that do not have gradients or which are extremely ill-conditioned, while the fast local search of the LM algorithm optimises the parameters that do have a gradient.
- Owing to the parallel nature of the CERN algorithm it could be implemented in a parallel processing architecture. Each sub-process can then perform the computations required to evaluate and update a group of the CERN population

NOMENCLATURE

A	Positive definite matrix
Ae	Aeration number
a_i	Output of the i th fuzzy rule
B	cluster strength, probablilty of generating individuals to be chosen from same cluster as parent individual
c_i	Centre of the i th basis function
c	Centre of a Gaussian unit
CR	Crossover (copy) over probability of parameters in the chromosome vector
d	Dimensions of the input space
D	Positive diagonal matrix
F	Scaling parameter
Fr	Froude Number
Fr_c	Critical Froude Number
g	Number of nodes in a group
h	Number of nodes in the hidden layer
k	Number of regressors considered in combinatorial search
k_r	Ridge factor used in the regression
l	$l*g$ are the number of clusters allocated for each node type
I	Identity matrix
m	Number of nodes connected to the output layer (often equal to h)
n	Number of training records
N_G	Number of groups in the population of CERN
N_P	Number of members in the population of CERN or DE. For CERN = $g* N_G$
p	Number of (previously found) fixed nodes connected to the output layer.
p_t	Pivot tolerance
Q	Orthogonal matrix
$\Re^d$	The d-dimensional real-valued vector space
r_i	i=1,2,3, the generating individuals in DE or CERN to form a new trail vector
$\|\mathbf{r}\|$	Euclidian norm of a vector **r**
$\|\|\mathbf{r}\|\|$	Norm of a vector **r** using a general metric
R^2	Coefficient of determination
s	Width of a Gaussian
T	Number of time steps in a time series that the predicted amplitude lies in the future
u	Number of membership functions for each of the inputs, used for the ANFIS algorithm
v	Number of variables that the Levenberg-Marquardt algorithm optimises
$\mathbf{v}$	The parameter vector of real values of an individual of the DE and CERN algorithm
V	The parameter matrix of real values of the population of the DE and CERN algorithm
w_i	Incoming weight of a neural node from the i th neural node in an previous layer
w_0	Incoming weight from the bias
$\boldsymbol{x}$	Input vector
XR	Copy over probability of a complete individual to another group
Δ	Number of time-steps between inputs in the time series, (lags)
λ_{max}	$(\sigma_{min})^{-2}$
δ	Draft Tube diameter in m
δ	Maximum distance, measured in the metric of the given basis node, that the centre of a basis function may move off the coordinates of a given training data point
Φ	Activation of the basis function
ρ	density kg.m^{-3}
σ_{min}	Minimum standard deviation of an ellipsoidal basis node in any direction
ϕ	Maximum angle of a Given's rotation which an oriented ellipsoidal basis node may rotate beyond $\pi / 4$
μ	Parameter of Levenberg-Marquardt learning rule

μ_f	Viscosity of the fluids in mPa.s
μ_i	Weight (value) of the ith fuzzy membership function

LIST OF FREQUENLTY USED ACRONYMS

ANFIS	adaptive neurofuzzy inference system
BP	backpropagation
CERN	combinatorial evolution of regression nodes
DR	delta rule
EBF	ellipsoidal basis function
GA	genetic algorithm
GCS	growing cell structure
GDR	generalized delta rule
GP-MA	Gaussian process model trained with a maximum-aposteriori approach
GP-MC	Gaussian process model optimized using a Monte-Carlo algorithm
MARS	multivariate adaptive regression splines
MLP	multilayer perceptron
MLP-MC	MLP trained with Bayesian methods using a Monte-Carlo algorithm
LM	Levenberg-Marquardt
LMS	least mean squares
OEBF	orientated ellipsoidal basis function
OLS	orthogonal least squared algorithm
ONRBF	orientated nonradial-basis function
PL-RBF	pseudo-linear radial basis function network
PRESS	Prediction sum of squares
RAN	resource allocation algorithm
RBF	radial basis function

References:

Aldrich C. & van Deventer J.S.J. (1995) Modelling of induced aeration in turbine aerators by use of radial basis function networks. *The Canadian Journal of Chemical Engineering*, **73,** pp. 808-816

Aldrich, C., Moolman, D.W., Gouws, F.S. & Schmitz ,G.P.J. *(1997)* Machine learning strategies for the control of flotation plants. *Control Engineering Practice* **5**(2), pp. 263-269

Angeline, P.J., Sauders, G.M. & Pollack, J.B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transaction on Neural Networks* **5**, pp. 54-65

Bandyopadnyay, J.K., Annamalai, S. Gauri, K.L. (1996) Application of artificial neural networks in modeling limestone- SO_2 reaction. *AIChE Journal* **42**(8), pp. 2295-2302

Barnard, J.P., Aldrich, C. & Gerber, M. (1999) Identification of non-linear chemical processes with surrogate data. submitted to AICHE Journal

Basta N. (1988 March) Expert systems. *Chemical Engineering* **26,** pp. 150-142

Battiti, R. & Tecchiolli, G. (1994) The reactive tabu search. *ORSA Journal on Computing* **6**(2), pp. 126-140

Battiti, R. (1992) First and second order methods for learning: Between steepest descent and newton's method. *Neural Computation* **4,** pp. 141-166

Belew, R.K., McInery, J. & Schraudolph, J.J (1990) Evolving networks: using the genetic algorithm with connectionist learning. San Diego (CA): UCAS (La Jolla), Technical Report CS90-174.

Bieu, V. (1996) Optimal implementation of neural networks: VLSI-friendly learning algorithms. *Neural Networks and Their Applications,* J.G. Taylor ed., John Wiley & Sons, Chichester, Chapter 18, pp. 255-276

Billings, S.A. & Zheng, G.L. (1995) Radial basis function network configuration using genetic algorithms. *Neural Networks* **8**(6), pp. 877-890

Bornhold, S. & Graudenz, D. (1992) General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks* **5**(1), pp. 327-334

Bowers, J.S. & Sammells, A.F. (1995) Structural Integrity in Aging Aircraft. *American Society of Mechanical Engineers*, Aerospace Division (Publication) AD **47**, ASME, New York, NY, pp. 227-235

Box, G.E.P. & Jenkins, G.M. (1970) *Time Series Analysis: Forecasting and Control.* Holden-Day, San Francisco, California.

Breiman, L. Friedman, J.H., Olsen R A. & Stone, C.J. (1984) *Classification and Regression Trees*, Chapman & Hall115 5th Avenue New York, NY 10003, USA.

Broomhead, D.S. & Lowe, D. (1988) Multivariable functional interpolation and adaptive networks. *Complex Systems* **2,** pp. 321-355

Bruske, J. & Sommer, G. (1995) Dynamic cell structure learns perfectly topology preserving map. *Neural Computation* **7,** pp. 845-865

Bulsari, A.B. (1995) *Neural Networks for Chemical Engineering*. Elsevier Science P.O. Box 211, 1000 AE Amsterdam, The Netherlands.

Calandranis J., Stephanopoulos, G. & Nunokawa, S. (1990) DiAD-Kit/Boiler: on-line performance monitoring and diagnosis. *Chemical Engineering Progress*, January 1990, pp. 60-75

Chen, S., Billings, S. A., Cowan, C. F. N. & Grant, P. M. (1990) Practical identification of narmax models using radial basis functions. *International Journal of Control* **52**(6), pp. 1327-1350

Chen, S., Cowan, C. F. N. & Grant, P. M. (March 1991) Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transaction on Neural Networks* **2**, pp. 302-203

Chovan, T. Catfolis, T.& Meert, K. (1996) Neural network architecture for process control based on the RTRL algorithm. *AIChE Journal* **42**(2), pp. 493-502

Cleveland, W.S. (1979) Robust locally weighted regression and smoothing scaterplots. *Journal of the American statistical association* **74**, pp. 829-836.

Cohen, A., Janssen, G., Brewster, S.D., Seeley, R., Boogert, A.A., Graham, A.A., Mardani, M.R., Clarke, N. & Kasabov, N.K. (1997) Application of computational intelligence for on-line control of a sequencing batch reactor (SBR) at Morrinsville sewage treatment plant. *Proceedings of the 1996 IAWQ International Conference on Advanced Wastewater Treatment: Nutrient Removal and Anaerobic Processes, AQUATECH'96*, Amsterdam, Neth. Source: *Water Science and Technology* **35**(10), pp. 63-71

Crowder, R. S. (1990) Predicting the Mackey-Glass time series with cascade correlation learning. *Proceedings 1990 Connectionist Models Summer School*, pp. 117-123

De Jong, K. A. (1975) *An analysis of the behaviour of a class of genetic adaptive algorithms.* Ph.D. dissertation, University of Michigan, Ann Arbor, Diss. Abstr. Int. **36**(10), 5140B, Univ. Microfilm #76-9381

De Veaux, R.D. & Ungar, L.H. (1994) Multicollinearity: a tale of two nonparametric regressions. *In Selecting Models from Data: AI and Statistics IV*; Lecture Notes in Statistics, 89, Springer-Verlag, New York, Cheeseman, P. and Oldford, R.W. (eds.), pp. 393-402

Deb, K. & Goldberg, D. E. (June 1989) An investigation of niche and species formation in genetic function optimisation. *Proceedings of the third International Conference on Genetic Algorithms*. Morgan Kaufmann, San Fransisco, pp. 42-50

Dickerson, J.A. & Kosko B. (1993) Fuzzy function learning with covariance ellipsoids. *Proceedings of the IEEE International Conference on Neural Networks* (IEEE ICNN-93), San Francisco, pp. 1162-1167

Duda, R. O. & Hart, E. (1973) *Pattern classification and scene analysis*. Wiley, New York, N.Y.

Fahlman, S. E. & Lebiere, C. (1990) The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems 2*, Touretzky, D.S. ed. San Mateo, CA: Morgan Kaufmann, pp. 524-532

Fahlman, S.E (1988) Fast learning variations on backpropagation: An empirical study. In *Proceedings 1988 Connectionist Models Summer School*, Touretsky, D.S , Hinton, G. & Sejnowski, T. eds., pp. 38-51

Farson, D., Fang, K. & Kern, J. (1992) Trainable fuzzy logic control of the laser welding process. *Proceedings of the 1992 Artificial Neural Networks in Engineering*, ANNIE'92, St.Louis, MO, USA, Source: *Intelligent Engineering Systems Through Artificial Neural Networks,* **2**, Fairfield, NJ, USA., pp. 809-814

Fayyad, U.M. & Irani, K.B. (1992) On the handling of continuous-valued attributes in decision tree generation. *Machine Learning* **8**, pp. 87-102

Fiesler E. (1994) Comparative bibliography of ontogenetic neural networks. *Artificial Neural Networks 4* (ICANN'94, Sorrento, Italy), Marinaro M. & Morasso P.G. eds. Springer-Verlag, pp. 793-795

Fisher, R.A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), pp.179-188

Fletcher, R. & Reeves C.M. (1964) Function minimization by conjugate gradients. *Computer Journal* 7, pp. 149- 154

Fogel, D. B. (1993) Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe. *Proceedings 1993 International Joint Conference on Neural Networks (IJCNN'93)* New York, NY: IEEE Press, pp. 875-880

Fogel, D. B., Fogel, L. J. & Porto, V. W. (1990). Evolving neural networks. *Biological Cybernetics* **64**, pp. 487-493

Friedman, J.H. & Stuetzle,W.(1981) Projection pursuit regression. *Journal of the American Statistical Association* **76,** pp. 817-823

Friedman, J.H. (1991) Multivariate adaptive regression splines. *The Annals of Statistics.* **1**(1), pp. 1-141

Fritzke B. (1995b) Incremental learning of local linear mappings. *ICANN'95 Proceedings of the international Conference on Artificial neural networks*, Paris, October, vol1, pp. 217-212

Fritzke, B. (1994) Growing Cell Structures: A self-organising network for unsupervised and supervised learning. *Neural Networks* 7 (9), pp. 1441-1460

Fritzke, B. (1995a). A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems*, Volume 7, Tesauro, G., Touretzky, D.S. and Leen, T.K. (eds.), MIT Press, Cambridge, Massachusetts, pp. 625-632

Furnival, G. M. & Wilson, W. W. Jr (Nov 1974) Regression by leaps and bounds. *Technometrics* **16**(4), pp. 499-511

Gasser , T. Müller, H.G. & Mammitzsch, V. (1985) Kernels for nonparametric curve estimation. *Journal of the Royal Statistical society*, Sr. B **47**, pp. 171-185

Gianetto, A. & P.L. Silverston (1986) *Multiphase Chemical Reactors-Theory Design and Scale-Up* Hemisphere Publishing Corp., Washington, D.C.

Girosi, F. Jones, M. & Poggio, T. (1993) Priors, stabilizers and basis functions, from regularization to radial, tensor and additive splines. *A.I. Memo No 1430*, Artificial Intelligence Laboratory, Massachusetts Institute of technology.

Girosi, F. Jones, M. & Poggio, T. (1995) Regularization theory and neural networks architectures. *Neural Computation* 7, pp. 219-269

Gorman, R. P. & Sejnowski, T. J. (1988) Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks* **1**, pp. 75-89

Hancock, P.J. (1992) Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specifications. *Combinations of genetic algorithms and Neural Networks* L.D. Whitley and Schaffer J.D. eds., IEEE Computer Society Press, Los Alamitos, CA, pp. 108-122

Hancock, P.J.B. (1990) GANNET: Design of a neural network for face recognition by a genetic algorithm. Stirling U.K., University of Stirling. *Technical Report.*

Harp, S.A. Samad T., Guha A. (1989) Towards the genetic synthesis of neural networks. In *Proceedings of the third international conference on Genetic Algorithms*. San Mateo, CA, Morgan Kaufmann, pp. 360-369

Hastie, T.J. & Tibshirani, R.J. (1990) *Generalized additive models.* Chapman & Hall, 11 New Fetter Lane London EC4P 4EE.

Hastie, T.J. & Tibshirani, R.J. (1994) Nonparametric regression and classification: Part I: Nonparametric regression. *From statistics to neural networks: Theory and pattern recognition applications*. ASI Proceedings, subseries F, Computer and Subsystem Science, Springer Verlag, Berlin.

Haykin, S. (1994) *Neural Networks: A comprehensive foundation.* Macmillan College Publishing Company, 866 3rd Avenue, NY 10022.

Haykin, S. (1999) *Neural Networks: A comprehensive foundation.* 2nd Edition Upper Saddle River, N.J. Prentice Hall.

Hertz, J., Krogh, A. & Palmer, R. G. (1991). *Introduction to the theory of Neural Computation.* Addison-Wesley Publishing Company, 350 Bridge Parkway, Redwood City, CA 94065.

Hinton, G.E. (1987) Learning translation invariant recognition in a massively parallel network. *Proceedings Conf. Parallel Architectures and Languages Europe*, Eindhoven.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University Michigan Press.

Hornik, K., Stinchcombe, M. & White, H., (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, pp. 359-366

Hrycej, T. (1992) *Modular learning in neural networks: a modularised approach to neural network classification.* Wiley, New York, N.Y.

Ingber, L. (1993) Simulated annealing: practice versus Theory, Journal of *Mathl. Comput. Modelling* **18**(11), pp. 29-57

Ishida M., & Zhan, J. (1995) Neural model-predictive control of distributed parameter crystal growth process. *AIChE Journal*, **41**(10), pp.2333-2336

Jacobs, R. A. (1988) Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**, pp. 295-307

Jang, J.-S.R. (1993) ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics* **23**(3), pp. 665-685

Jang, J.-S.R., Sun, C.-T. & Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing*, Prentice-Hall, New Jersey.

Johannsen, G. & Alty, J.L. (1991) Knowledge engineering for industrial expert systems. *Automatica* **27**(1),pp. 97-114

Joubert, H.W., Theron, P.L. and Lange, T.(1996 September) The use of neural networks for gas loop process optimisation. *SA Instrumentation and Control*, pp. 44-51

Juang C.F. & Lin C.T. (1998) An on-line sel-constructing neural fuzzy inference network and its applications. *IEEE Transactions on Fuzzy Systems*, **6**(1), pp.12-22

Kattan, M. (1994) Inductive expert systems vs. human experts. *AI Expert*, April 1994,pp. 32-38

Kavuri, S.N. & Venkatasubramanian, V. (1993) Representing bounded fualt classes using neural networks with ellipsoidal activation functions. *Computers in Chemical Engineering*, **17**(2), pp. 139-163

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983 May). Optimization by simulated annealing. *Science* **220**(4598), pp. 671-680

Kitano, H. (1990) Designing neural networks using genetic algorithms with graph generation systems. *Complex Systems* **4**, pp. 461-476

Klir, G.J. & Yuan, B. (1995) *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, New Jersey.

Kolodner, J. (1993) *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California.

Koza J.R. (1992) *Genetic Programming: On the Programming of Computers By Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts.

Krogh, A.& Hertz, J.A. (1992) A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems 4*, Moody, J.E., Hanson S.J. & Lippmann, R.P. eds., Morgan Kaufmann, San Mateo, CA, pp. 950-957

Lange, J.M., Voigt, H.-M. & Wolf D. (1994) Task Decomposition and correlations in growing artificial neural networks. *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*. Mariano, M. & Morasso, P.G. eds., Springer Verlag, London, pp. 735-738

Lee, M. & Park, S. A (1992) New scheme combining neural feedforward control with model-predictive control, *AIChE Journal* **38**(2), pp. 193-200

Leonardis A. & Bischof H. (1998) An efficient MDL-based construction of RBF networks. *Neural Networks* **11,** pp. 963-973

Leung, C.-T. & Chow, T.W.S. (1999) Adaptive regularization parameter selection method for enhancing Generalization Capabilities of Neural Networks. *Artificial Intelligence* **107**, pp. 347-356

Linden A., & Kindermann, J. (1989) The inversion of multilayer nets . International Joint Conference on Neural Networks, Washington, pp.425-432

Littmann, E. & Ritter, H. (1992) Cascade network architectures. *Proceedings International Joint Conference on Neural Network.* Baltimore, Section II, pp. 398-404

Littmann, E. & Ritter, H. (1993) Generalisation abilities of cascade network architectures. *Advances in Neural Information processing Systems.* Hanson, J. S., Cowan, J.D. & Giles, C.L. eds. Morgan Kaufmann, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, pp. 188-195

Liu, G.P. , Kardirkamanthan V., Billings, S.A. (1998) On-line identification of nonlinear systems using Volterra polynomial basis function networks. *Neural Networks* **11**, pp. 1645-1657

Lynch, D.T. (1991) Chaotic behavior of reaction systems: parallel cubic autocatalators. *Chemical Engineering Science* 1991, pp. 357-381

MacKay, D.J.C. (1991) *Bayesian modelling and Neural Networks.* Ph.D. thesis, Computation and Neural systems, California Institute of Technology, Pasadena, CA.

Mackey, M. C. & Glass, L. (1977) Oscillations and chaos in physiological control systems. *Science* **197**, pp. 287-289

Mandani, E.H. & Assilian, S. (1975) An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-machine Studies* 7(1), pp. 1-13

Mandenius, C.-F., Eklov, T. & Lundstrom, I. (1997) Sensor fusion with on-line gas emission multisensor arrays and standard process measuring devices in baker's yeast manufacturing process. *Biotechnology and Bioengineering* **55,** pp. 427-438.

Maniezzo, V. (1994) Genetic evolution of the topology and weight distribution of neural networks, *IEEE Transactions on Neural Network* **5**(1), pp. 39-53

Marquardt, D. (1963) An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal of Applied Mathematics*, **11**, pp. 431-441.

Martinez D. (1992) Dealing with continuous inputs for the Offset Algorithm. *Proceedings of the International Conference on Signal Processing Appl. & Tech.* SPAT'92, Boston, pp. 1167-1175

Martinez, T. & Schulten, K. (1994) Topology representing networks, *Neural Networks* 7, pp. 507-522

McDonnel, J.R. & Waagen, D. (1993) Neural network structure designed by evolutionary programming. *Proceedings of the Second Annual Conference on Evolutionary Programming*. Fogel, D.B. & Atmar, W., eds., Evolutionary Programming Society, La Jolla CA, pp. 79-89

McLoon, S. (1996) *Neural Network training for Modeling and Control*. Ph.D. dissertation, Advanced Control Engineering Research Group, Queen's University of Belfast, U.K.

McLoon, S., Brown M.D. & Irwin G. (1998) A hybrid linear/nonlinear training algorithm for feedforward neural networks, *IEEE Transactions on Neural Networks* **9**(4), pp. 669-683

Mézard, M. & Nadal, J.-P. (1989) Learning feedforward layered networks: The Tiling algorithm. *Journal of Physics A* **22**, pp. 2191-2204

Michie, D. Spiegelhalter, D.J. & Taylor CC. (eds.) (1994) *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, England.

Miller, A. J. (1990) *Subset selection in regression*. Chapman and Hall, London.

Moody, J. & Darken, C.J. (1989) Fast learning in networks of locally-tuned processing units. *Neural Computation* **1**(4), pp. 281-294

Moody, J. (1989) Fast learning in multi-resolution hierarchies. *Advances in Neural Information Processing Systems 1*, Touretzky, D.S. ed., Morgan Kaufmann, San Mateo, CA, pp. 29-39

Moolman, D.W., Aldrich, C., Schmitz, G.P.J. & van Deventer, J.S.J. (1996) The interrelationship between surface froth characteristics and industrial flotation performance. *Minerals Engineering* **9**(8) pp. 837-854

Muggleton, S. (1990) *Inductive Acquisition of Expert Knowledge*. Turing Institute Press, Glasgow in association with Addison-Wesley Publishing Company, New York

Myers, R.H. (1989) *Classical and Modern Regression with applications*. 2nd edition. PWS-KENT Publishing Company, 20 Park Plaza, Boston, MA 02116.

Narula, S.C. & Wellington, J.F. (1977) An algorithm for the minimum sum of weighted absolute errors regression. Commun. In Statist. **B6**, pp. 341-352

Narula, S.C. & Wellington, J.F. (1979) Selection of variables in linear regression using the sum of weighted absolute errors criterion. *Technometrics* **21**, pp. 299-306

Nash ,W.J., Sellers, T. L., Talbot S. R, Cawthorn A. J. & Ford W. B. (1994) The population biology of abalone (Haliotis species) in Tasmania. I. blacklip abalone (H. rubra) from the North Coast and Islands of Bass Strait, Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288)

Neal, R.M. (1996) *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics No. 118, Springer-Verlag, New York, NY.

O' Brian, Paul (1998) *Linear Prediction Filters and Neural Networks*, Chapter 9 http://www-ssc.igpp.ucla.edu/personnel/russell/ESS265/Ch9/linear_predict/

Odri, S.V., Petrovacki, D. P. & Krstonosic, G. A. (1993) Evolutional development of a multilevel neural network, *Neural Networks* **6**(4), pp. 583-595

Otten, R.H.J.M. & van Ginneken, L.P.P.P (1989) *The annealing algorithm* Kluver Academic.

Park, J. & Sandberg, I.W. (1991) Universal approximation using radial basis function networks. *Neural Computation* **3**, pp. 246-257

Picciolo G. (1989) Availability assessment of complex distributed control system of a petro chemical plant. *Reliability Data Collection and Use in Risk and Availability Assessment. Proceedings of the 6th EuReDatA Conference*. Colombari, V., ed., Springer-Verlag, Berlin, pp. 524-30

Pietruschka, U. & Kinder, M. (1995) Ellipsoidal basis functions for higher-dimensional approximation problems. *ICANN '95. International Conference on Artificial Neural Networks. Neuronimes '95 Scientific Conference*. Fogelman-Soulie, F. & Gallinari, P. eds. Paris, France, EC2 & Cie, vol. 2, pp. 81-85

Platt, J. (1991) A resource allocating network for function interpolation. *Neural Computation* **3**, pp. 213-225

Pottmann, M. & Henson, M.A. (1997) Compactly supported radial basis functions for adaptive process control. *Journal of Process Control*, 7(5), pp. 345-356

Powell, M. J. D. (1972) Problems related to unconstrained optimisation. *Numerical Methods for Unconstrained Optimisation*. Murray, W. ed., 1972, Academic Press London and New York, pp. 29-55

Powell, M. J. D. (1987) Radial basis functions for multivariable interpolation: A review. *IMA Conference on Algorithms for the Approximation of Functions and Data.* Mason, J. C. & Cox, M. G., eds. Royal Military College of Science, Oxford University Press, U.K., pp. 143-167

Prechelt, L. (1996) A quantitative study of experimental evaluations of neural network algorithms: current research and practice. *Neural Networks* **9**, pp. 457-462

Press, W.H. , Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (1992) *Numerical Recipes in C*, Cambridge University Press.

Price, K. & Storn, R. (1997) Differential evolution: numerical optimization made easy, *Dr. Dobb's Journal*, April 97, pp. 18-24

Radcliff, N.J. (1990) *Genetic Neural Networks on MIMD Computers.* Dissertation, Edinburgh (UK): University of Edinburgh.

Radcliff, N.J. (1991) Equivalent class analysis of genetic algorithms. *Complex Systems* **5**(2), pp. 183-205

Radcliff, N.J. (1993) Genetic set recombination and its application to neural network topology , Optimisation *Neural Computing & Applications* **1**(1), pp. 67-90

Rasmussen, C.E.(1996) *A Practical Monte Carlo Implementation of Bayesian Learning. Advances in Neural Information Processing Systems 8*, Touretzky, D.S., Mozer, M.C. & Hasselmo, M.E. eds., MIT Press, Cambridge MA, pp. 598-604

Ratkowsky D.A. (1990) *Handbook of Nonlinear regression models*. Marcel Dekker, Inc., 270 Madison Avenue New York, NY 10016.

Rissanen, J. (1978) Modelling by shortest data description. *Automatica* **14**, pp. 465-471

Rissanen, J. (1989) *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore.

Robinson , A.J. (1989) *Dynamic error propagation networks*. Cambridge University, Ph.D. thesis.

Roger Jang , J.S. & Sun, C.T. (1993) Functional equivalence between radial basis function networks and fuzzy interference systems. *IEEE Transactions on Neural Networks* **4**(1), pp. 156-159

Rumelhart, D.E. & McClelland ,J.L. eds. (1986). *Parallel Distribution Processing: Exploration in the Microstructure of Cognition* **1**, MIT Press, Cambridge, MA.

Russo, M. (1998) FuGeNeSys – a fuzzy genetic neural system for fuzzy modeling. *IEEE Transactions on Fuzzy Systems*, **6**(3), pp. 373-388

Saffrey J. & Thornton C. (1991) Using stereographic projection as a preprocessing technique for upstart. *Proceedings 1991 International Joint Conference on Neural Networks (IJCNN'91)* Seatle, IEEE Press, **2**, pp. 441-446

Saha, A. & Keeler, J.D. (1990) Algorithms for better representation and faster learning in radial basis function networks. *Advances in Neural Information Processing Systems 2,* Touretzky, D. S. eds., Morgan Kaufmann, San Mateo, CA, pp. 482-489

Saha, A., Christian, J. Tang, D. S. & Wu, C.-L. (1991) Orientated non-radial basis functions for image coding and analysis. *Advances in Neural Information Processing Systems 3,* Lippmann, R. P., Moody, J. E., Touretzky, D. S. eds., Morgan Kaufmann, San Mateo, CA, pp. 728-734

Saraiva, P.M. & Stephanopoulos, G. (1992) Continuous process improvement through inductive and analogical learning. *AIChE Journal* **38**(2), pp. 161-183

Sarle, W. S. (1999) *Neural Networks: Frequently asked Questions* http://www.cis.ohio-state.edu/hypertext/faq/usenet/ai-faq/neural-nets/part2/faq-doc-3.html

Schaal S. & Atkeson C.G. (1998) Constructive incremental learning from only local information. *Neural Computation* **10,** pp. 2047-2084

Schaffer, J.D., Whitley, D.& Eshelman, L. J. (1992 June) Combinations of genetic algorithms and neural networks: A survey of the state of the art. *Combinations of Genetic algorithms and Neural Networks*, Whitley, D. & Schaffer, J. D. eds., IEEE Computer Soc. Press, Los Alamitos, CA, pp. 1-37

Schetzen, M. (1980) *The Volterra and Wiener Theories of Nonlinear systems*. John Wiley & Sons, New York.

Schiffmann, W., Joost, M. & Werner, R. (1991) Performance evaluation of evolutionary created neural network topologies. *Lecture Notes in Computer Science* 496: Parallel Problem Solving From Nature, Goos, G. and Hartmanis, J., eds., Springer Verlag, Berlin, pp. 274-283

Schmitz, G.P.J & Aldrich, C. (1998) Neurofuzzy modeling of chemical process systems with ellipsoidal radial basis function networks and genetic algorithms. *Computers and Chemical Engineering,* **22**, Suppl. pp. S1001-S1004

Schmitz, G.P.J & Aldrich, C. (1999) Combinatorial evolution of regression nodes in feedforward neural networks. *Neural Networks* **12**(1), pp. 175-189

Schultz, M.H. (1969.) *Theory and applications of spline functions.* Edited by T.N.E. Greville. - New York, N.Y. Academic Press, (Publication of the Mathematics Research Center, United States Army, the Univ. of Wisconsin, no. 22). - Proceedings of an advanced seminar conducted by the Mathematics Research Center.

Seber, G.A.F.(1977) *Linear regression analysis* - Wiley series in probability and mathematical statistics, Wiley, New York, N.Y.

Seber, G.A.F. & Wild,C.J. (1989) *Nonlinear Regression.* John Wiley & Sons, New York.

Shang, Yi. & Wah, B.W. (1996) Global optimization for neural network training. *Computer (IEEE)* **29**(3), pp. 45-54

Sherstinsky , A. & Picard, R. W. (1996) On the efficiency of the orthogonal least squares training method for radial basis function networks. *IEEE Transactions on Neural Networks* 7(1), pp. 195-201

Simpson, P. K. (September, 1992) Fuzzy min-max neural networks: - part 1: classification. *IEEE Transactions on Neural Networks* **3**(5), pp. 776-786.

Small, M. & Judd, K. (1998) Comparisons of new nonlinear modeling techniques with applications to Infant Respiration. *Physica D* **117**, pp. 283-298

Smalz, R. & Conrad, M. (1994) Combining evolution with credit apportionment: A new learning algorithm for neural nets. *Neural Networks* 7(2), pp. 341-351

Storn, R. & Price, K. (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* **11**, pp. 341-359

Storn, R. & Price, K. (1995) Differential evolution - a simple and efficient adaptive scheme for global optimisation over continuous spaces. *Technical Report TR-95-012*, ICSI, Berkeley, U.S.A.

Sugeno, M. & Kang, G.T.(1988) Structure identification of a fuzzy model. *Fuzzy Sets and Systems* **28,** pp. 15-33

Sun, C.-T. (1994) Rule-base structure identification in an adaptive-network-based fuzzy inference system. *IEEE Transactions on Fuzzy Systems* **2**(1), pp. 64-73

Tanaka, K. & Sugeno, M. (1992) Stability analysis and design of fuzzy control systems. *Fuzzy Sets and Systems* **45**, pp. 135-156

Taylor, W. (1989) *What every engineer should know about artificial intelligence*, MIT Press

Terry, P.A. & Himmeblau, D.M.(1993) Data rectification and gross error detection in a steady-state process via artificial neural networks. *Industrial & Engineering Chemistry Research*, **32**(12), pp. 3020-3028

Tetko, I.V., Livingstone, D.J. & Luik, A.I. (1995) Neural network studies, 1. comparison of overfitting and overtraining. *Journal of Chem. Info. Comp. Science* **35**, pp. 826-833

Van der Smagt, P. (1994) Minimisation methods for training feedforward neural networks. *Neural Networks 7(1)*, pp. 1-11

Van der Walt, T.J. (1992) *The development of connectionist networks for the modelling of chemical engineering systems* . Dissertation (Ph. D.)-University of Stellenbosch, R.S.A.

Van der Walt, T.J., Barnard, E. & van Deventer, J. (1995) Process modeling with the regression network. *IEEE Transactions on Neural Network* **6**(1), pp. 78-93

Vijayakumar,S. & Schaal, S. (1997) Local dimensionality reduction for locally weighted learning. *IEEE International symposium on computational intelligence in Robotics and Automation,* Montery, CA, July10-11, pp. 220-225

Wang, L-.X. and Mendel, J.M. (1992) Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics* **22**(6), pp. 1414-1427

Wang, Li-Xin (1997) *A course in Fuzzy Systems and Control.* Prentice Hall PTR, Uppers Saddle River, NJ 07458.

Wang, X.Z, Chen, B.H., Yang, S.H., McGreavy,C. & Lu, M.L. (1997) Fuzzy rule generation from data for process operational decision support. *Computers in Chemical Engineering,* **21**,suppl. pp. S661-S666

Watkins, D.S. (1991) *Fundamentals of matrix computations.* Wiley, New York, N.Y.

Weiss, S.M. & Kulikowski, C.A. (1991) *Computer Systems that learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning, and Expert Systems.* Morgan Kaufmann, San Mateo, CA

Werbos, P.J. (1974) Beyond regression: *New Tools for prediction and analysis in the behavioural science.* Ph.D. Thesis Harvard University, Cambridge, MA.

Wettschereck, D. & Dietterich, T. (1992) Improving the performance of radial basis function networks by learning center locations. *Advances in Neural Information Processing Systems 4*, Moody, J.E., Hanson S.J. & Lippmann, R.P. eds., Morgan Kaufmann, San Mateo, CA, pp. 1130-1140

Whitehead, B.A. & Choate, T.D. (1994, Jan.) Evolving space filling curves to distribute radial basis function networks over an input space. *IEEE Transactions on Neural Networks* **5**(1), pp. 15-23.

Whitehead, B.A. & Choate, T.D. (1996, July) Co-operative-competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks* 7(4), pp. 869-880.

Whitehead, B.A. (1996) Genetic evolution of radial basis function coverage using orthogonal niches. *IEEE Transaction on Neural Networks* 7(6), pp. 1525-1528

Whitley, D. Starkweather T. & Bogart, C. (1990) Genetic algorithms and neural networks: optimising connections and connectivity. *Parallel Computing* **14**, pp. 347-361

Williams, C.K.I. & Rasmussen, C.E. (1996) Gaussian processes for regression. *Advances in Neural Information Processing Systems 8*, Touretzky, D.S., Mozer, M.C. & Hasselmo, M.E. eds., MIT Press, Cambridge MA, pp. 514-520

Yao, X. & Liu, Y. (1997 March) A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks* **8**(3), pp. 694-713

Yao, X. (1993) A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems* **8**(4), pp. 539-567

Yingwei, Lu, Sundararajan, N. & Saratchandran, P. (1997) A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation* **9**(2), pp. 461-478

Zadeh, L.A. (1965) Fuzzy Sets. *Information and Control* **8**, pp. 338-353

Zadeh, L.A. (1971) Towards the theory of fuzzy systems. *Aspects of Network and Systems Theory*, Kalman R.E., DeClaris, N. eds.

Zurada, J.M. (1992) *Introduction to artificial neural systems.* West Publishers, St. Paul, Minnesota.

Appendix A: Data sets used

According to chapter 1 the data sets are divided into four categories:

1. Tune parameters to improve CERN's learning speed;
2. compare learning speed of CERN to other techniques on the same task;
3. study the effects of groups size *g*, node types and number of nodes on the generalisation capabilities of CERN; and
4. assess CERN's generalisation relative to other non-linear models.

Table A1 gives a brief summary of each data set, as well as the category and the chapters in which the data was used. Further information regarding the origins of the data sets and any pre-processing steps made can be obtained from the description of the individual data sets that follow. The input scaling is given for each of the problems, because some algorithms such as RBF networks are sensitive to it. The output scaling is generally not given. It is not relevant to most of the algorithms and the LM and (G)DR software that trained the multilayer perceptrons automatically scaled all the outputs to lie between –0.9 and 0.9.

	Real or Synthetic	*Dimensions Input (categorical) + Output*	*Training set size (n)*	*Test or validations set size*	*Used in cate-gories*	*Chapters*
Abalone	R	9(2) +1	3133	1044	2,4	3 & 6
Aeration	R	4 +1	180	5x36	4	6
Autocatalytic reaction	S	5+1	8000	2000	4	6
Corrosion	R	43(41)+1	757	442	4	6
Double-Spiral	S	2+1	194	N/A	2,3	4,6
Kine 8nm	S	8+1	1024	1024	3	5
Kine 8nh	S	8+1	1024	1024	3	5
Furnace data	R	32+2	1344	348	4	5
Furnace output 1, selected inputs	R	7+1	1344	348	2,4	6
Furnace output 2, selected inputs	R	10+1	1344	348	3	5
Mackey-Glass Time series	S	4+1	500	500	1,3	4,5
Miles/Gallon	R	7+1	498	5x100	4	6
Pine data set	R	9+1	4408	2204	1,4	5,6
Puma-32mn	S	32+1	4x1024	4x1024	4	6
Puma-32mn, selected inputs	S	3 +1	4x1024	4x1024	4	6
Sonar	R	60 +1	104	104	3	5
Vowel	R	10+11	528	462	4	6

Table A.1 A summary of the different data sets used in the dissertation. A kx in the number of data points indicates that either k-fold crossvalidation was performed on the real world data set or for synthetic data sets k data sets were generated.

A.1 Abalone Data

This data set was obtained from a study that attempted to predict the age of abalone using the physical characteristics of the specimens as input (Nash, et al., 1994). The eight inputs are the sex of the specimen (male, female or infant), the length, the diameter, the height, the total weight, the shucked weight, the weight of the viscera and finally the shell weight of the specimen. The age in years of a specimen was determined by cutting its shell through the cone, staining the shell and counting the number of shell rings by microscope. This is a time-consuming task and it would therefore be useful to obtain the age from the more easily obtainable physical characteristics. The data were obtained from the UCI Repository of Machine Learning Databases at Irvine. Records with missing data had already been removed. 3133 records were used for training and the remaining 1044 records for network evaluation. The sex input feature was encoded as {1; 0} for type male, {0; 1} for type female and {0; 0} for the infant type.

The data were scaled so that all inputs had a mean of 0 and a standard deviation of 0.17. This operation gave most of the inputs an approximate range of 1. The output was scaled to lie between -0.9 and 0.9.

A.2 Aeration data

Induced aeration is a convenient process to contact gases and liquids in for example fermentation processes, bioleaching of minerals and the treatment of waste water, and has been the subject of various investigations. (Gianetto & Silverstone 1986; Aldrich & van Deventer 1995). The rate of induced aeration, expressed in terms of a dimensionless aeration number (*Ae)* is a function of the modified Froude Number ($Fr\text{-}Fr_c$) the density (ρ) and viscosity (μ) of the liquid, as well as the geometry of the vessel. In this study the vessel geometry was fixed, except for the diameter (δ) of the draft tube.

There were 180 data exemplars of the form $\{Fr\text{-}Fr_c, \rho, \mu, \delta | Ae\}$ arising from experiments using two different impeller types with six blades (T6) and twelve blades (T12). Since the viscosity spanned several orders of magnitude (≈1-1000 cP), which can have a detrimental effect on the training of the neural network model, it was scaled by taking the square root of the viscosity. For convenience, the rate of induced aeration was multiplied by a factor of 1000. The data were rescaled by simple transformations so that every input variable had a standard deviation between 0.25 and 0.3. In particular ρ was multiplied by 2, δ by 30 and the viscosity by 10.

A.3 Simulated data from an autocatalytic reaction

In a study by Lynch (1991) an autocatalytic reaction in a continuous stirred tank reactor was simulated. The system is capable of producing self-sustained oscillations based on cubic autocatalysis with catalyst decay and is governed by the following equations:

$$\begin{aligned} \mathrm{A} + 2\mathrm{B} \rightarrow 3\mathrm{B}, \quad -r_\mathrm{A} &= k_1 c_\mathrm{A} c_\mathrm{B}^2 \\ \mathrm{B} \rightarrow \mathrm{C}, \quad r_\mathrm{C} &= k_2 c_\mathrm{B} \\ \mathrm{D} + 2\mathrm{B} \rightarrow 3\mathrm{B}, \quad -r_\mathrm{D} &= k_3 c_\mathrm{D} c_\mathrm{B}^2 \end{aligned}$$

where A, B, C and D are the participating chemical species and k_1, k_2, k_3 the rate constants for the chemical reactions. This process is represented by the following set of ordinary differential equations.

$$\begin{aligned} \frac{dX}{dt} &= 1 - X - aXZ^2 \\ \frac{dY}{dt} &= 1 - Y - bYZ^2 \\ \frac{dZ}{dt} &= 1 - (1+c)Z + daXZ^2 + ebYZ^2 \end{aligned}$$

where X, Y, and Z denote the dimensionless concentrations of species A, B and D, while a, b, c denote the Damköhler number for A, B and D respectively. The ratio of feed concentration of A to B is denoted by d and the similar ratio of D to B by e. For the settings: $a = 18000$; $b = 400$; $c = 80$; $d = 1.5$; $e = 4.2$, and initial conditions $[0,0,0]^\mathrm{T}$, the set of equations was solved by using a 5th order Runge Kutta numerical method over 100 simulated seconds. This gave 10 000 observations, which were resampled with a constant sampling period of 0.01 s. The data generated by this method was also studied by (Barnard et al., 1999).

Similar to the procedure for the Mackey-Glass time series the Y values were embedded so that each data record consists of the several lagged points of the time series $Y(t - \mathrm{k}\Delta)$ as well as the corresponding output $Y(t + T)$. The data had Gaussian noise added to them and were to be predicted three time steps into the future. A fixed embedding dimensions of 5 (i.e. k = 0 …4) and a lag $\Delta = 9$ was used (Barnard et al., 1999). However, one method, the PL-RBF network, adaptively decides on an embedding dimension. The first 8000 data points of the times series were used for training and the following 2000 points for testing. The data set was also rescaled for CERN to give the time series a standard deviation of 0.25.

A.4 Corrosion data

These data consist of 35 metals and 8 chemicals that reacted with one another under varying temperature and concentrations of the acid. The extend of damage of the metal was classified to be "strongly corroded" (+1), "moderately corroded" (0) or "no visible corrosion" (-1). The numbers in parentheses indicate the output encoding that was used for the different models. There were 757 records in the train set and 424 records in the test set. Of the data points in the tests set half the records consisted of combinations of chemical and metals that did not occur in the train set. These records are particularly difficult to predict. Each network was trained using an outcome for each data record as either –1, 0 or 1. As the outputs could just be one of these three possibilities the accuracies were given as a classification score. The classification score was 100% - % of cases classified false - % of cases classified as strongly corroded that in fact showed no corrosion and vice versa. So if a record was classified completely wrongly (difference = 2) it was weighted twice as much as a record that was just classified wrongly by a single class difference of 1.

A.5 Double-Spiral problem

Although CERN minimises RMS in its search for an optimal solution, has a linear output node and is primarily aimed at problems with continuous outputs, its behaviour was also examined on the so-called Double-Spiral problem, which is actually a classification benchmark.

The Double-Spiral problem has two continuous-valued inputs and a single discrete output. The data consist of 194 records describing two interlocked spirals in two dimensions. The x and y co-ordinates of the data points were scaled to lie between –0.75 and 0.75 for the experiments described here. The two spirals are each defined by a separate class. The purpose of the benchmark is to distinguish between the two spirals, or classes, while producing decision regions which best match the pattern created by the interlocked spirals. This task is especially difficult to for an MLP as the intertwined classes are difficult to separate using only a combination of linear discriminant functions.

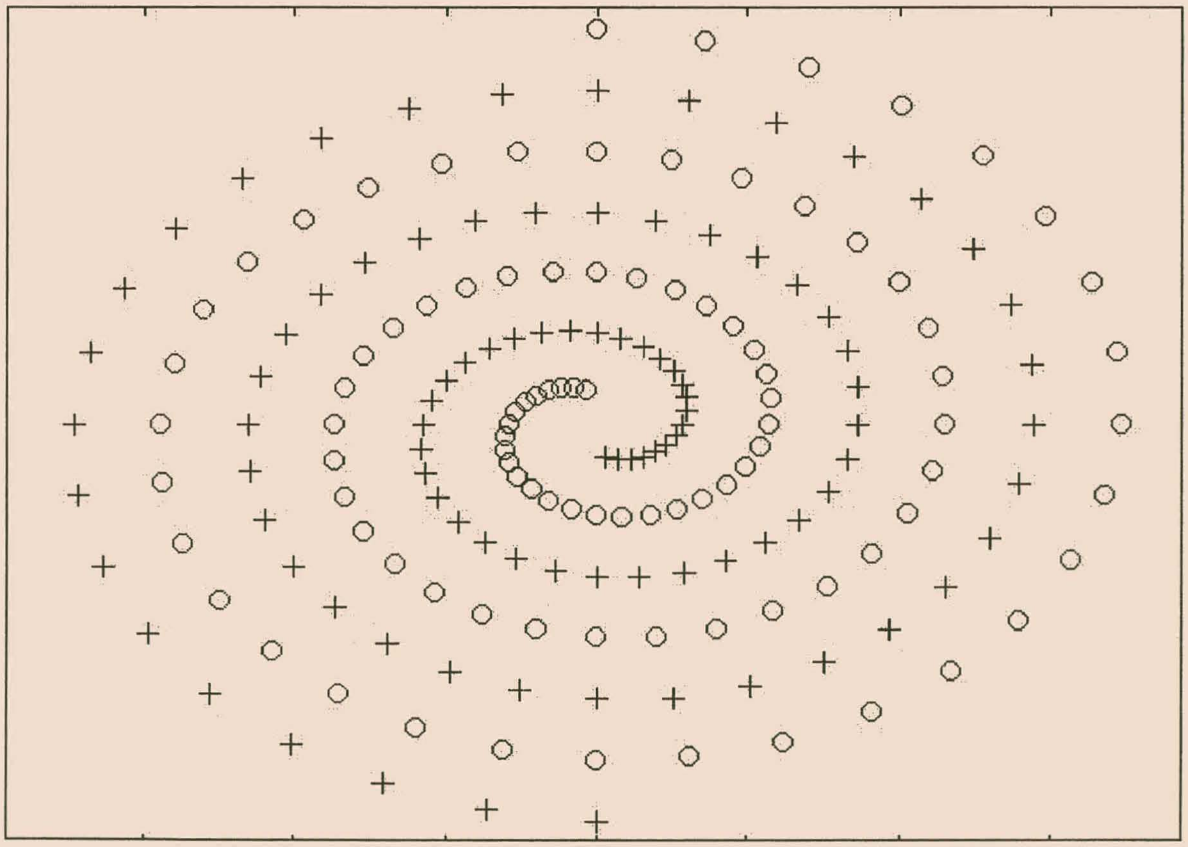
Figure A.1 The data points of the Double-Spiral classification problems.

A.6 The Furnace data set

34 process variables of an industrial furnace were recorded for over 4½ years. The data consists of 1750 consecutive data points, one for each day. Those data points that were taken while the plant was shut off and those points that were obvious outliers were removed. Obvious outliers were single points, where one or more input or output was outside 20 times the standard deviation for that input or output. When two or more points were outside this range, the points were not regarded as outliers. After removal of the outliers 1692 complete records remained. The data was divided into three sets: a training set, a validation and a final test set. Since the furnace may have some memory, shuffling of the data points may not be a save form of dividing the data points. Consecutive points were thus put in the training data set for 3 × four running weeks, the next four weeks of data were put in the validation set and the data collected during another four consecutive weeks were put in the test set. This process was repeated several times and the remainder was put in the test set. After the completion of this procedure there were in total 1008 training points, 336 validation records and 348 test points in total. This way the test set always consists of 4 consecutive weeks of data, but spread over the entire 4½ years. This division is safer than having all the test points taken from the final year. If the latter division were used, there would be less variation in the inputs and consequently a smaller portion of the feature space would be tested.

It was decided to scale the inputs so that each one has a standard deviation of 0.35. Two of the outputs were examined. Preliminary runs indicated that, if all the 32 inputs were used by standard feedforward neural networks, the predictions on the validation set were poor. Hence an attempt was made to identify the relevant attributes, first using linear models and then using the MARS algorithm. A linear model was build using the subset search method of Furnival and Wilson (1974). The number of regressors was determined by maximising the R^2-PRESS statistic (Myers 1990). MARS could not find a better model than the linear model for the first output. When using the principle components instead of the inputs as suggested by (De Veaux & Ungar, 1994) MARS was also not more successful. Hence the seven variables found by the linear model were selected. On the other output MARS could find a model that generalises better on the validation set ($R^2 = 0.72$) than the linear model ($R^2 = 0.67$). Therefore the variables that MARS indicated to be valuable were used. On the validation set neural networks trained with these selected variables resulted in much better performance than those trained using all the variables.

The Kine8-nm data set

This data set was obtained from the DELVE benchmark page from the University of Toronto (http://www.cs.toronto.edu/~delve/). It arises from a realistic simulation of the forward kinematics of an 8 link all-revolute robot arm. The 8 inputs are the angles of the 8 joints. The output is the Cartesian distance of the endpoint to a certain fixed point. The data set was normalised so that the inputs had a standard deviation of 0.3 and the output a standard deviation of 1. The benchmark site rates the data set as highly non-linear with a medium amount of noise added to it.

A.8 The Kine8-nh data set

This data set was also obtained from the DELVE benchmark page of the University of Toronto. It is similar to the Kine-8nm data set except that its noise is higher than that of the kine-8nm data set.

A.9 The Mackey-Glass time series

The Mackey-Glass time series problem (Mackey & Glass, 1977) can be found through integration of the Mackey-Glass time delay differential equation (Yingwei, et al. 1997).

$$\frac{d\left(x(t)\right)}{dt} = -bx(t) + \frac{ax\left(t-\tau\right)}{1 + x^{10}\left(t-\tau\right)}$$

Benchmark tests for comparing neural networks, in particular radial basis function training schemes, take a, b and τ as 0.2, 0.1 and 17, respectively (e.g. Moody & Darken 1989; Platt 1991; Whitehead & Choate, 1996; Yingwei, et al., 1997). To make comparisons possible with the algorithms examined in the above-mentioned papers the series is predicted 85 time-steps ahead (T) using four past samples which are 6 time-steps (Δ) apart. Each data record consequently consists of the inputs $x(t)$, $x(t - \Delta)$, $x(t - 2\Delta)$ and $x(t - 3\Delta)$ as well as the corresponding output $x(t + T)$. The series is shown for 500 time-steps in Figure A2. As was done by Yingwei, et al. (1997) and Platt (1991), the first 4000 data records were discarded to allow the initialisation transients in the series to decay. The next 500 records of the time series were used for training and the test set comprised the 500 records that followed thereafter. All the inputs to the neural networks already had a standard deviation around 0.23 and were not rescaled.

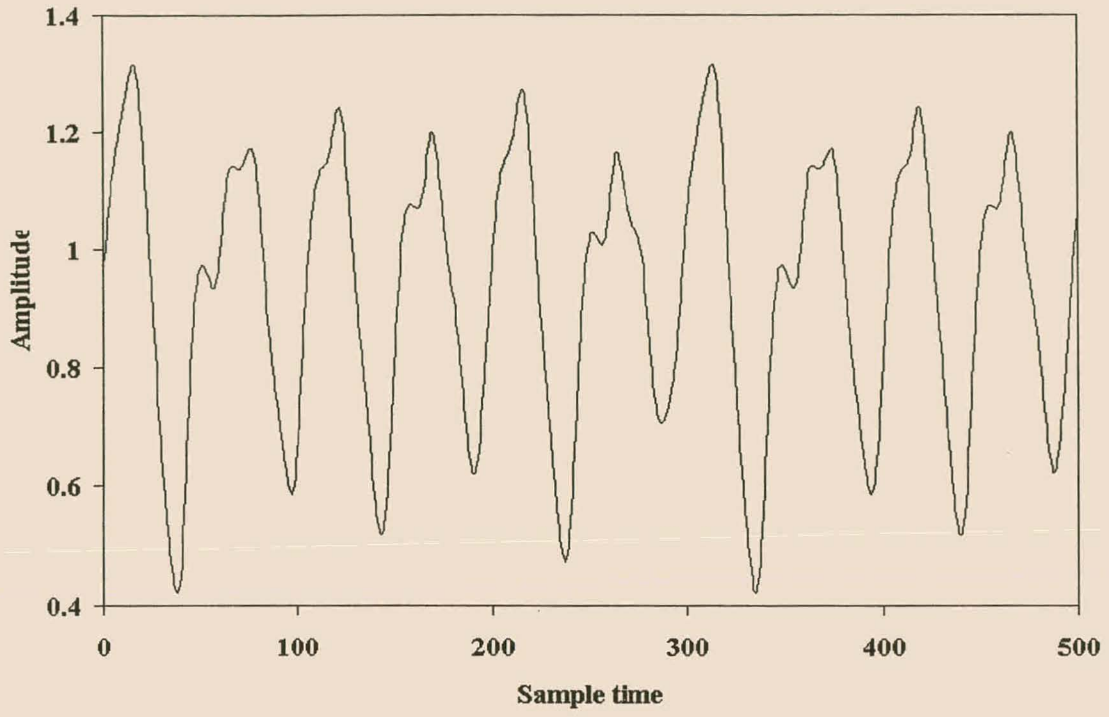

Figure A.2 Sample of the Mackey-Glass time series.

A.10 Miles per Gallon data set

The aim of this problem is to predict the fuel consumption of a variety of different automobiles. The attributes are the number of cylinders, displacement and power of the engine, the weight and acceleration of the automobile, the origin of the car, the year in which the car model was first manufactured and finally the miles that the car can travel per gallon. The data set was obtained from UCI Repository of machine learning databases and has as an eighth attribute the name of the car. This attribute was ignored in this study. The power attribute has 6 missing values. These were replaced by the mean of the power attribute. The attributes were than scaled to have a standard deviation of 0.3 each.

A.11 Pine data set

Measurements were taken from pine trees and of their surrounding atmosphere for several seasons. The inputs of the pine data set consists of 9 continuous measurements, namely temperature, relative humidity, differential vapour pressure, photoactive radiation, leaf mass, height of the tree, diameter at breast height, xylem pressure potential and the season. The sap flow rate is a continuous output. The data consisted of 6612 data records of which two-thirds were used for training and the rest for testing the various algorithms. It was found using validation sets that scaling all the input attribute to have a standard deviation of 0.5 performed slightly better than a scaling where each attribute's standard deviation was 0.3. Consequently the former scaling was used.

A.12 The Puma-32mn data set

The data, obtained from the DELVE home page, was generated by a realistic simulation of the dynamics of a Unimation Puma 560 robot arm. The inputs are six angles of joints, their angular velocities, the torque on the joints, five proportional changes in masses of the links, proportional changes in the length of the links and five proportional changes in viscous friction of the links. The output is the angular acceleration of joint 6. The data set was normalised in the same way as the Kine-8 data sets. The data set is highly nonlinear and has a medium amount of noise. Runs on validation sets revealed that the MARS algorithm could find much better solutions than straightforward neural networks, hence the attributes that MARS found to be important were selected these were attributes number 5, 16 and 20. These are the angle 5, torque 4 and the proportional change of link 3. On validation sets, the neural networks performed much better using only these attributes instead of all 32 inputs.

A.13 Sonar data

The Sonar classification problem was taken from the UCI Repository of Machine Learning Databases at Irvine. The aim of the model is to distinguish between the sonar signals reflected off a roughly cylindrical rock from those bounced off a cylindrical metal cylinder (Gorman & Sejnowski 1988). The signal consists of 60 features used as inputs and one enumerated output. The data has 102 training and 102 testing points that were divided by Gorman and Sejenowski so that each set contained cases from each aspect angle in appropriate proportions. All the inputs were scaled to have a standard deviation of 0.25.

A.14 Vowel recognition data

The next data set analysed is the speaker independent Vowel recognition data set. It comprises a training set of 528 samples and a test set of 46 examples. The input vector is 10-dimensional continuous and there are in total 11 output classes. More information about this data set can be obtained from a thesis by Robinson (1989). In order to not change the relative scaling of the variables to that of the original problem all variables were divided by the same amount of 3.0. In this way each input had a standard deviation lying in the range of 0.16 to 0.4.

Appendix B: Floating point operations required by CERN, BP and LM

B.1 Spiral data set

Equations 6.4, 6.5 and 6.6 will be use to compute the floating point operations of the leading terms required for one network update of the spiral data, when optimising 7 summation nodes. Here $d = 2$, $g = 7$, $n = 194$

BP : $(2n \times 14 + 4n \times 7) + 2n \times 14 + 8n \times 7 + 6n = 28n + 28n + 28n + 56n + 6n = 146n$

LM : $n(14 + 14 + 1)^2 + (14+14+1)^3 + 4n \times 14 = 841n + (29)^3 + 56n \approx 841n + 127n + 56n = 1024n$

CERN : $(2n \times 14 + 4n \times 7) + 3n \times 49 = 28n + 28n + 147n = 203n.$

There are some additional operations required in the combinatorial search and in the clustering. The combinatorial search will roughly require 22000 floating point operations. This was obtained from the empirical result of Furnival and Wilson (see section 4.3.1). It is equivalent to $\approx 114n$ operations. Thus CERN takes $317n$ operations.

B.2 Abalone

The number of floating point operations will now be assessed for the abalone data set. Here $d = 9$, $g = 6$ and $n = 3314$.

BP $(2 \times n \times 54 + 6n) + 2 \times n \times 54 + 66n + 6n = 108n + 108n + 66n + 12n = 294n$

CERN $(2 \times n \times 54 + 6n) + 3 \times n \times 36 = 108n + 6n + 108n = 222n$

LM $n(54 + 12 + 1)^2 + (54 + 12 + 1)^3 + 4 \times n \times 54 = 4489n + 216n + 67^3 \approx 4796\ n$

It can be computed easily, using section 4.31, that the combinatorial search takes less than 8n operations. Thus the cost of one network evaluation of CERN is slightly less than that of backpropagation and they are comparable. The LM algorithm takes roughly 20 times more floating point operations for the computations associated with one network evaluation than does CERN.

B.3 Furnace-1

The furnace-1 training data set has $d = 8$, $g = 7$ and $n = 1300$. The number of floating point operations for each network evaluation and corresponding updates is:

BP = $(2n \times 56 + 4n \times 7) + 2n \times 56 + 8n \times 7 + 6n = (112 + 28 + 112 + 56 + 6)n = 314n$

CERN = $(2n \times 56 + 4n \times 7) + 3n \times 49 = (112 + 28 + 147)n = 287n$

LM = $n(56 + 14 + 1)^2 + (56 + 14 + 1)^3 + 4n \times 56 = 5041n + 71^3 + 224n \approx 5540n$

Additionally CERN requires roughly *17n* floating point operations for the combinatorial search. As for the Double-Spiral data set this can be computed using the empirical results of section 4.3.1.

Appendix C: Fuzzy rules of CERN for the Miles/Gallon data set

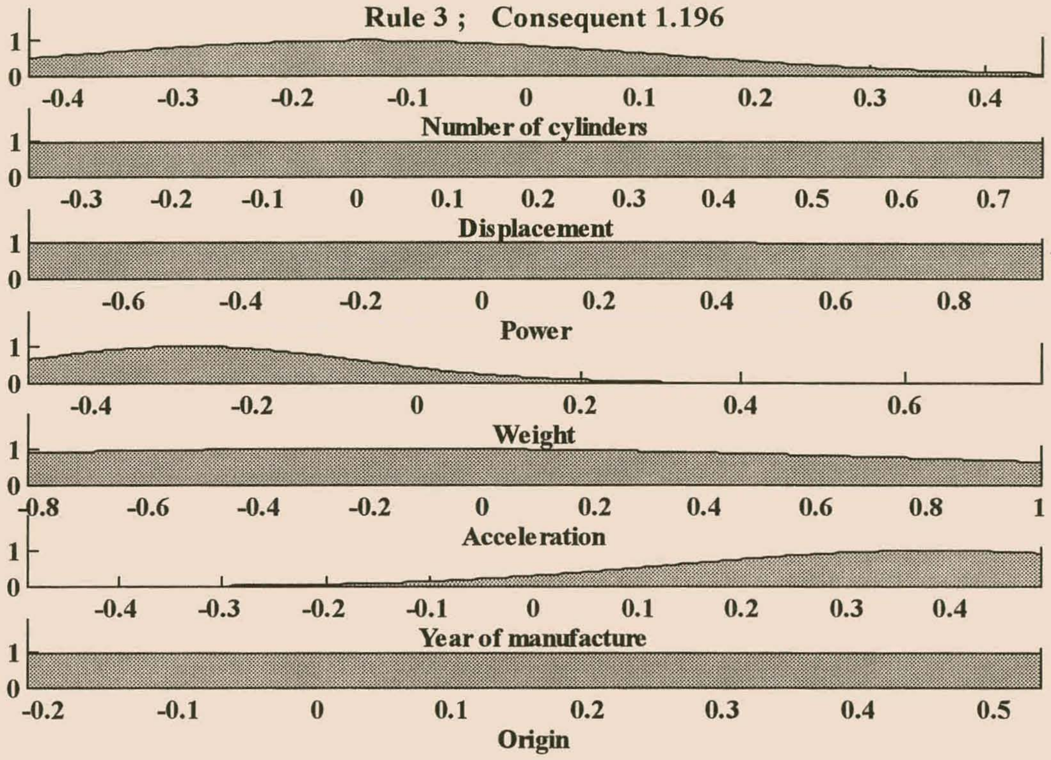

Figure C.1 The third rule for the Miles/Gallon data. All the inputs and the outputs were rescaled.

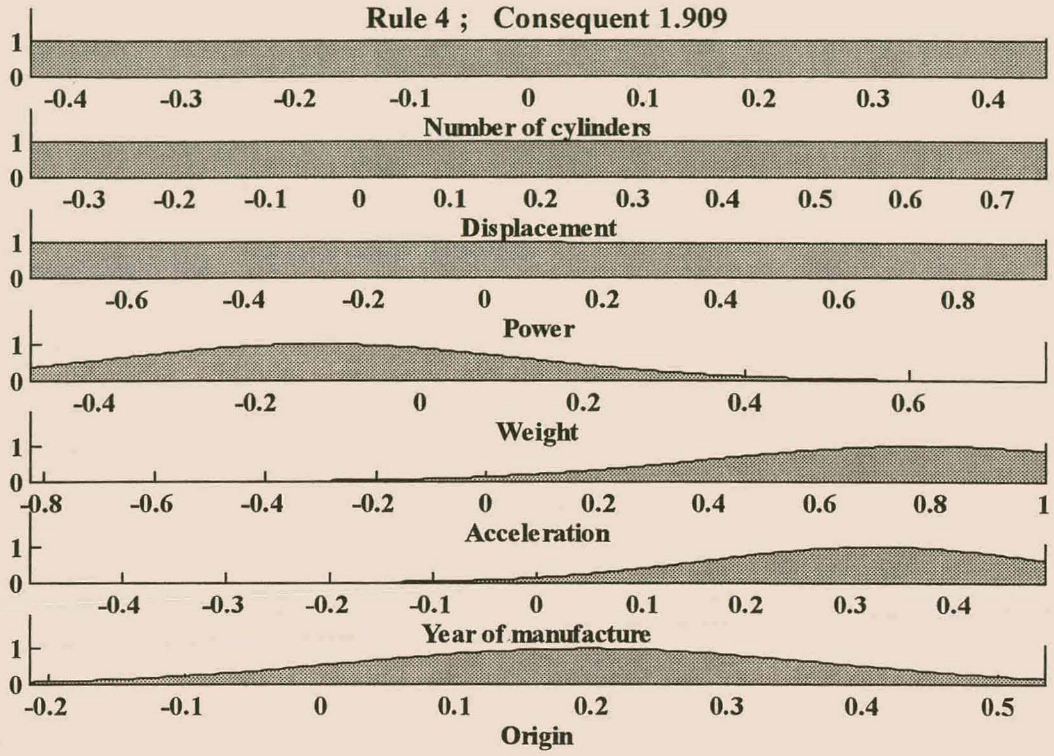

Figure C.2 The fourth rule for the Miles/Gallon data.

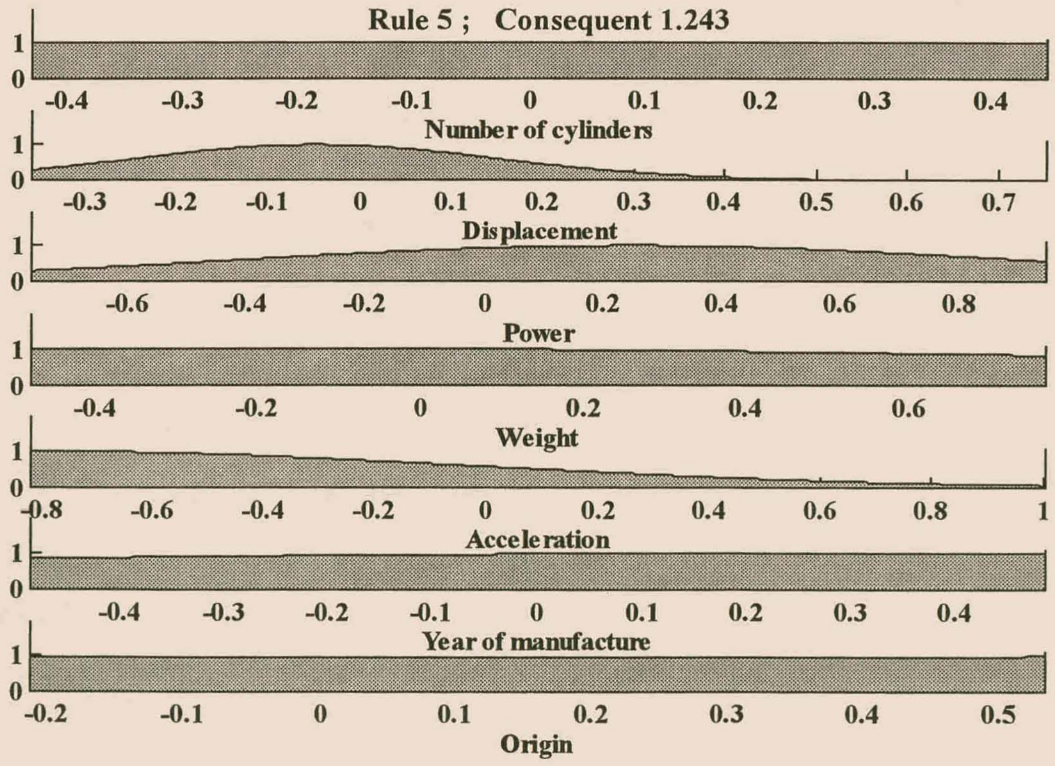

Figure C.3 The fifth rule for the Miles/Gallon data.

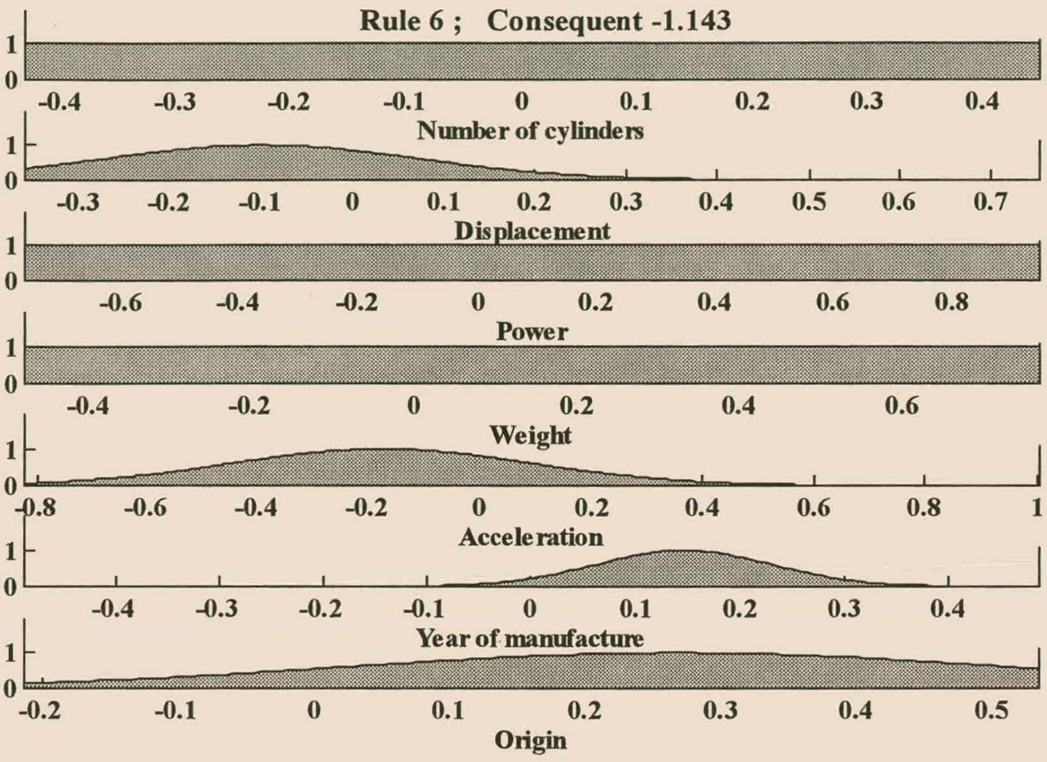

Figure C.4 The sixth rule for the Miles/Gallon data.

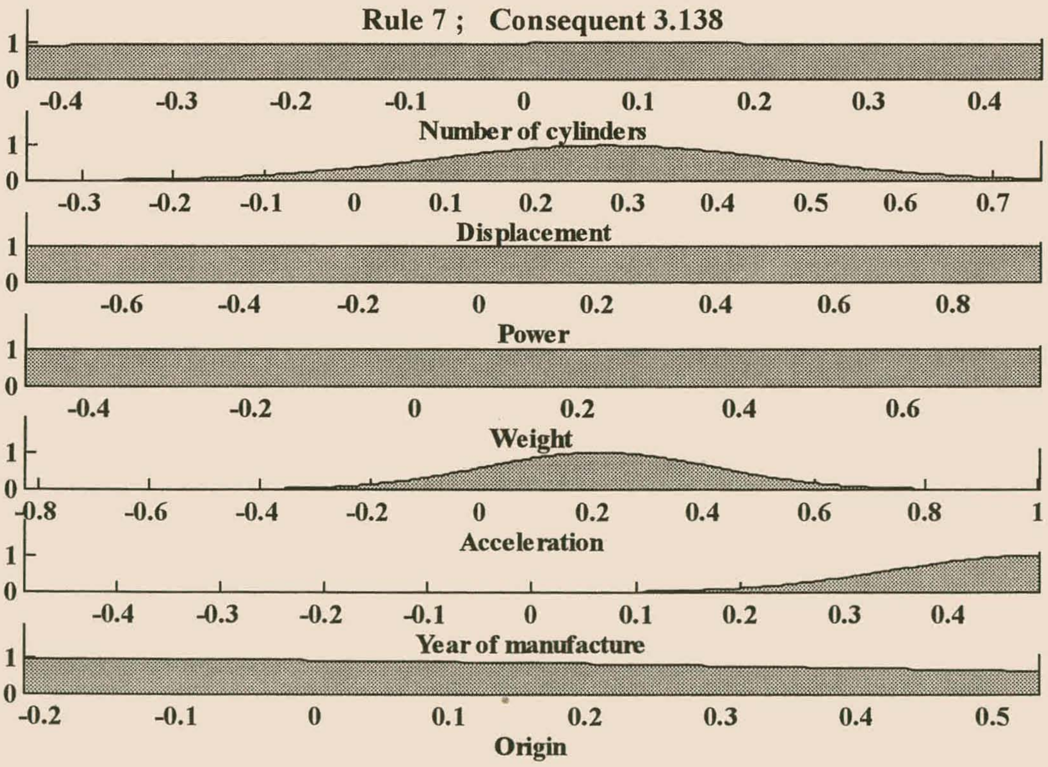

Figure C.5 The seventh rule for the Miles/Gallon data.

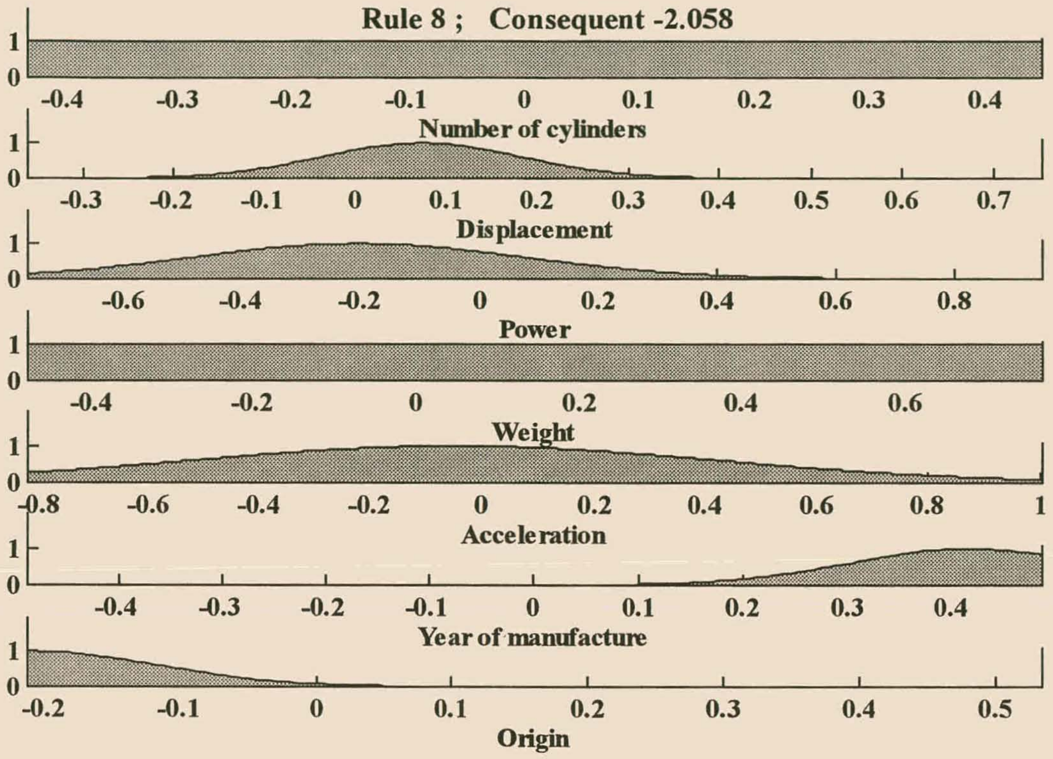

Figure C.6 The eighth rule for the Miles/Gallon data.