A LOCAL MODEL NETWORK APPROACH TO NONLINEAR MODELLING

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER SCIENCE
OF THE UNIVERSITY OF STRATHCLYDE
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Roderick Murray-Smith
November, 1994

Abstract

This thesis describes practical learning systems able to model unknown nonlinear dynamic processes from their observed input-output behaviour. Local Model Networks use a number of simple, locally accurate models to represent a globally complex process, and provide a powerful, flexible framework for the integration of different model structures and learning algorithms.

A major difficulty with Local Model Nets is the optimisation of the model structure. A novel Multi-Resolution Constructive (MRC) structure identification algorithm for local model networks is developed. The algorithm gradually adds to the model structure by searching for 'complexity' at ever decreasing scales of 'locality'. Reliable error estimates are useful during development and use of models. New methods are described which use the local basis function structure to provide interpolated state-dependent estimates of model accuracy. Active learning methods which automatically construct a training set for a given Local Model structure are developed, letting the training set grow in step with the model structure – the learning system 'explores' its data set looking for useful information.

Local Learning methods developed in this work are explicitly linked to the local nature of the basis functions and provide a more computationally efficient method, more interpretable models and, due to the poor conditioning of the parameter estimation problem, often lead to an improvement in generalisation, compared to global optimisation methods. Important side-effects of normalisation of the basis functions are examined.

A new hierarchical extension of Local Model Nets is presented: the Learning Hierarchy of Models (LHM), where local models can be sub-networks, leading to a tree-like hierarchy of softly interpolated local models. Constructive model structure identification algorithms are described, and the advantages of hierarchical 'divide-and-conquer' methods for modelling, especially in high dimensional spaces are discussed.

The structures and algorithms are illustrated using several synthetic examples of nonlinear multivariable systems (dynamic and static), and applied to real world examples. Two nonlinear dynamic applications are described: predicting the strip thickness in an aluminium rolling mill from observed process data, and modelling robot actuator nonlinearities from measured data. The Local Model Nets reliably constructed models which provided the best results to date on the Rolling Mill application.

Acknowledgements

On the technical side, I would like to thank my employers Daimler-Benz AG for providing a stimulating research environment, and for letting me include research carried out for them in the thesis. I'd especially like to thank Frieder Lohnert for giving me the freedom to explore the areas I thought useful.

My other colleagues provided a great deal of advice and support, directly and indirectly: Jeff Donne for his insight into rolling mills (and cocktails) and for preparing the data for the mill modelling work; Henrik Gollee for his assistance in the vagaries of MATLAB, performing experiments and proof-reading the thesis; Ken Hunt for reading the thesis and trying to improve my mathematical precision; Dietmar Neumerkel for his organisational support in the mill work and for helping with the development of tools and ideas; Robert Shorten for co-operation with the normalisation work and for doing the dishes. Thanks to the other members of the group not directly involved in this work. Wise old Greeks, Indians and Persians also managed to put the ups and downs of research into perspective.

Cooperating in research has fewer geographical limitations than ever before, so I'd like to thank Tor Arne Johansen in Norway and Daniel Sbarbaro-Hofer in Chile for the many long distance E-mail discussions we had (thanks again to Dan for the help given at the start of the work, for reading the resulting thesis and for making insightful suggestions). I'd also like to thank Tom Kavli in Oslo for letting me use his robot actuator data. My supervisor Douglas McGregor was also, unusually for a supervisor, one of the long distance colleagues, but still managed to provide guidance in the right direction at the right time.

On the personal side, I thank my parents again for giving me the best start in life I could have hoped for.

I would like to thank history for making Berlin a fascinating place in the early 90's, and the Berliners (*Ossis*, *Wessis* and *Ausländer*) for being a lot of fun to be with.

Contents

List of Tables

List of Figures

Chapter 1

Learning Systems for Empirical Modelling

1.1 Learning

The ability to interact with the environment and to learn from the effects of these interactions is one of the defining features of intelligence. A system with the ability to learn from observation thus compensates for an initial lack of *a priori* knowledge about its given task. Such flexibility is becoming increasingly important in automatic systems, as the physical, technological and economical environments in which systems are operating are changing faster than ever before. Higher levels of autonomous action are desired from our robots, washing machines, cars and computers. Improved flexibility and adaptability is a major asset, whether the adaptability is in the product development process, or in the products themselves.

This thesis targets the task of modelling complex nonlinear, dynamic processes by allowing models to learn from the processes' observed behaviour. A goal of the work was to develop methods which not only had the required performance, but were also relatively interpretable, to support validation of models, and able to integrate models and methods from other paradigms. The introduction of explicit *a priori* knowledge about the target process is also an important element of applied learning systems.

Obtaining an accurate computer-based model of the physical process is the first step towards the creation of high performance diagnosis systems, supervisory systems, controllers and filters. In many practical systems, however, the process is still poorly understood, or new variations of the system are being constantly created, leading to a slightly different problem each time a controller is to be developed. The ability to use learning systems to cope with the uncertainties would allow developers to produce high performance systems faster and more cheaply than with conventional techniques.

The intuitive concept of 'learning' can be interpreted in a number of ways when trying to emulate it on a machine:

A typical dictionary definition of learning is:

- 1. to gain knowledge of something or acquire skill in some art or practice, 2. to commit to memory, 3. to gain by experience, example etc., 4. to become informed.

Other definitions from the researchers investigating machine learning include:

- Learning systems belong to the class of systems which show a gradual improvement of performance due to the improvement of the estimated unknown information (Fu, 1970).
- Learning is optimisation under conditions of insufficient *a priori* information (Tsypkin, 1971).
- Learning is the process by which one entity acquires knowledge (Rich, 1988).
- Learning can be regarded as synthesising an approximation of a multi-dimensional function, that is solving the problem of hypersurface reconstruction (Poggio and Girosi, 1990).
- ...modifying patterns of behaviour on the basis of past experience so as to achieve specific anti-entropic ends. In these higher forms of communicative organisms the environment, considered as the past experience of the individual, can modify the pattern of behaviour into one which in some sense or other will deal more effectively with the future environment (Wiener, 1948).
- Behaviour is primarily adaptation to the environment under sensory guidance. It takes the organism away from harmful events and toward favourable ones, or introduces changes in the immediate environment that make survival more likely (Hebb, 1949).

This work develops learning algorithms and model structures which gain knowledge about a process from observed input-output example by synthesising a suitable non-linear multi-dimensional function to fit the training data.

1.2 Learning & Engineering – Where is the Engineering?

Much research in recent years has been carried out within the artificial neural network paradigm, using simplified formal models of physiological systems. The neural network research in empirical modelling has been very experimentally oriented, so the learning algorithms and structures

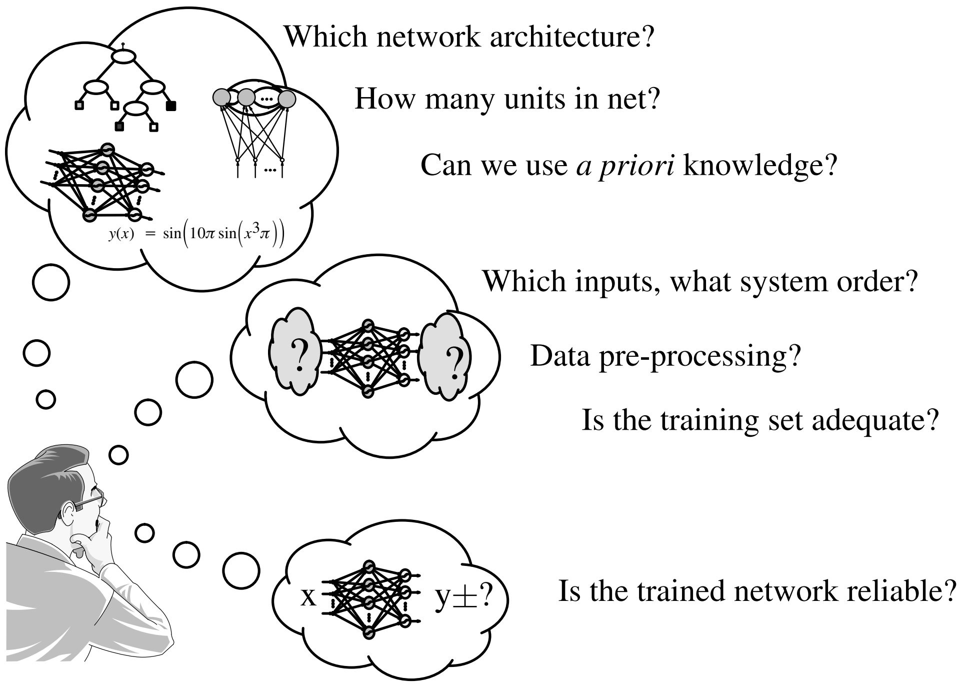

Which network architecture?
How many units in net?
Can we use *a priori* knowledge?
$y(x) = \sin\left(10\pi\sin\left(x^3\pi\right)\right)$
Which inputs, what system order?
Data pre-processing?
Is the training set adequate?
x
y±?
Is the trained network reliable?

Figure 1.1: The trouble with neural nets for engineering ... An engineer faces a number of difficult design decisions when using neural networks in practical projects.

which have become popular in this community have been successfully applied to a variety of challenging applications. However, few workers in the area have analysed the deeper theoretical issues, or exploited results and experience from closely related fields such as *function approximation*, *system identification* or *statistics*. This ignores decades of relevant work, and has made the scientific output less accessible to a broader community, leading to criticism of the neural network area by scientists and engineers from other fields.

The lack of a theoretical approach to the work has, unfortunately, also had the consequence that there is no clearly defined engineering process in which a model can reliably be created from measurements taken from a physical system, as shown in Figure 1.1. The impressive results quoted in the literature usually come after months of 'tweaking' the parameters of learning algorithms, implicitly using *a priori* knowledge by pre-processing the data, and in selective testing of the trained system. The length of time taken for training is often prohibitive, often without the guarantee of an optimal solution.

The difficulty in determining whether a good solution has been found stems from the fact that most networks need to have their structure initialised in advance using guesswork, with little guarantee against over-fitting the data, or under-specifying the model structure. It is vital for the successful application of the ideas, that robust (in terms of reliably finding a good model which generalises well to new inputs) learning algorithms be developed which can adaptively find a parsimonious model structure and optimise its parameters to fit a given problem and training set.

The problems for learning systems in industry are not, however, limited to poor learning algorithms. The creation of a training set for a nonlinear, multi-variable dynamic system can be an extremely complex task. The data acquisition process may also be very expensive, time-consuming, costly, and in some cases dangerous (if you don't already have a good controller) and time-consuming. It is also possible to unwittingly include undesired side-effects in the data specific to a particular run/day/setting which limit the usefulness of the final model.

A further problem with much of the current reported research is that the methods used to evaluate performance of the trained system are usually too simplistic. The reliability of the resulting model is also usually not clearly understood, partially because of the poor interpretability of the model architectures. In conventional neural networks it is often difficult to introduce *a priori* knowledge. This is highly important in practical applications, where there is usually a great deal of such knowledge available about the system in question. In most publications, the creation of the training set is implicitly affected by *a priori* knowledge about the system, and certain architectures may be more suited to certain system types. The assumption underlying the research here is that it is very important that prior knowledge can be explicitly built in to the system's architecture and optimisation algorithms. Methods to do this have been described, but are still not present in much of the current research. This work attempts to bring these various facets together within a single framework.

1.3 Thesis Contributions

The methods described in this thesis have been chosen for their suitability for integration with conventional engineering techniques, as well as their powerful representations and learning algorithms. The resulting interpretability and robustness with respect to sparse or noisy data was also an important aspect of the work.

- The Local Model Network, an existing generalisation of Basis Function[1] networks is analysed, and the theoretical and practical suitability of the architecture for practical modelling applications is demonstrated. The Local Model framework allows the integration of *a priori* knowledge, is more transparent than other architectures, and by pre-structuring the model structure, can better cope with high-dimensional, or high order processes.
- It was found that the commonly used global parameter optimisation in Local Model networks is computationally expensive, and in some cases poorly conditioned. A new Local Learning algorithm for the optimisation of the parameters in a local model net has been developed. This is significantly faster, produces more interpretable models and can

[1] Networks consisting of a single nonlinear layer, linearly weighted to the output, as described in Section 2.3.1.

have a regularisation effect on the optimisation process producing models with a better generalisation ability.

- Locally interpolated error estimates for Basis Function and Local Model networks are developed. These produce state-dependent error estimates for a trained local model network, which help validate the trained model, and can be used by constructive algorithms, or on-line when the model is in use.

- The effect of normalisation of the basis functions in Local model and Basis Function networks is analysed. Side-effects are described which reduce the interpretability, and have serious effects on the smoothness and robustness of the final model.

- The new Multi-Resolution Constructive (MRC) structure identification algorithm for local model nets is developed. This automatically fits the model structure (number, location and size of the basis functions, and the complexity of the local models) to the available training data. The algorithm uses a 'complexity heuristic' to gradually improve the model's structure.

- The local model network is extended to a hierarchy of local model networks – the Learning Hierarchy of Models (LHM) architecture. Parameter optimisation and structure identification algorithms are described which utilise the hierarchical nature of the structure to make learning more efficient.

- The algorithms and model structures are applied to model data from a real industrial process – an aluminium rolling mill. The algorithms performed better than competing learning systems such as MARS (Friedman, 1991), and Multi-Layer Perceptrons[2], and it is intended to implement the model in on-line tests. A further example, modelling robot actuator dynamics is also analysed.

1.4 Thesis Structure

- Chapter 2 is a review of the existing empirical modelling theory. The empirical modelling process is introduced and the importance of the various stages discussed: *Experiment design*, *construction of a suitable representation*, *optimisation of the parameters* and *validation of the trained models* are all vital stages, and the lack of support for these phases from conventional neural networks is criticised. The Local Model network architecture is reviewed and its suitability as a network for practical modelling applications discussed, as is the use of hierarchical learning structures. An overview of related work and theory is given.

[2]See Section 2.1.4.

- Chapter 3 investigates several aspects of local model networks. The problems with ill-conditioning of the parameter estimation problem in local model nets are analysed and a new local learning algorithm is described which is faster and in noisy or sparsely populated problems can be more robust than global methods. The trade-off between global and local training methods is discussed.

  A flexible, straightforward extension to local model nets is given which allows interpolation of local estimates of accuracy to provide a state-dependent error statistic for a trained network.

  The effect of normalisation of the basis functions on the network's representational properties is analysed. Several side-effects other than the desired partition of unity are described which can have serious consequences for the interpretability and robustness of basis function networks.

- In Chapter 4 the new MRC algorithm for constructive creation of Local Model Basis Function Nets is described. This development iteratively allocates units to the areas of greatest complexity in the system. This is repeated at ever increasing resolution of complexity, gradually creating an ever more accurate model, given the limitations of the available training data.

  Constructive techniques for active selection of the most important training data from a large training set are given. This allows the learning system to maintain the training set at the size and completeness needed for a given model structure. The strengths and weaknesses of the methods developed in this chapter are illustrated using several artificial static and dynamic modelling tasks.

- Chapter 5 introduces the Learning Hierarchy of Models architecture. This can be viewed as a hierarchical local model net structure. The advantage of this structure is that more efficient models can be produced, and that the hierarchical nature of the structure allows more efficient learning algorithms, especially for high-dimensional problems. Parameter optimisation algorithms and structure identification algorithms are defined. The confidence estimation and active learning components described in previous chapters for local model nets are extended to the LHM architecture.

- Chapter 6 demonstrates the practical application of the new methods with the modelling aspects of an aluminium rolling mill. The local model and LHM architectures were successfully applied to model the output thickness deviation of the strip, given the current state of the system.

  A further example, the modelling of robot actuator nonlinearities, is given. The results are compared with three other methods, MARS, ASMOD (Kavli, 1992) and LSA (Johansen and Foss, 1994b).

- Chapter 7 summarises and discusses the significance of the results and analysis in the thesis.

Chapter 2

Methods for training models

The scope of the use of learning techniques for empirical modelling in this thesis is outlined, and the classes of learning systems are described in general terms. The various research paradigms associated with machine learning are reviewed, with special attention paid to the practical problems with artificial neural networks in an engineering environment. To clarify the requirements of a learning framework for modelling, the modelling process is analysed, each phase of which is then associated with desirable features in a learning system. The Local Model Network, a generalisation of Basis Function nets, is proposed as an architecture suited to real applications. It provides the ability to introduce a priori knowledge and model complex systems robustly, while allowing estimates of accuracy and enhancing the training set. The relevant literature is reviewed and the close connections to conventional statistical methods, system identification and fuzzy logic are emphasised.

2.1 Using Learning Techniques for Empirical Modelling

The 'modelling problem' as discussed here is to try to robustly approximate the behaviour of a given complex system from observation data, where complex implies that the system can be non-linear, time-invariant, multi-variable and dynamic. This can be interpreted as a learning task, where the learning system has to learn a suitable representation for the process in question. The model should obviously be a good representation of the target system, for the purposes intended of it, but in an industrial or scientific environment it should ideally also be interpretable, so that the engineer gains improved understanding about the system. The learning systems used in this thesis can therefore be viewed as computationally intensive tools which are used to support the modelling process by attempting to induce a parsimonious representation of the process from the behaviour described by the observed input-output data. The basic assumption underlying the use of learning systems for modelling purposes is therefore that the behaviour of the process can be described in terms of its observed inputs (ψ) and outputs (y). The process can therefore be modelled as a function $f(\psi)$ of the inputs ψ, i.e. $\hat{y} = f(\psi)$, subject to a measurement error e, so that the true outputs are

$$y = f(\psi) + e. \tag{2.1}$$

Such methods are basically black-box modelling techniques, i.e. techniques which attempt to describe a system by finding relationships between the system's inputs, internal states and outputs using general model structures ($f(\psi)$ performs a nonlinear mapping from n dimensional inputs to m dimensional outputs, $\mathcal{R}^n \rightarrow \mathcal{R}^m$) to represent a fit to the given data, irrespective of physical meaning of the parameters.

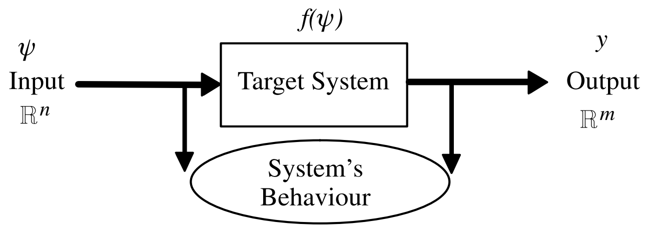


Figure 2.1: Systems can often be described by their input-output behaviour

In practice it will usually be impossible to find an absolutely correct representation of the underlying process because of the effects of unmeasured inputs and states, which can be generalised to be treated as noise on the data, leading to the need to use a stochastic framework for the modelling process. It will also be rare for the training data to be uniformly distributed around the input space, and there may be areas with insufficient data available for a good approximation. This uncertainty means that there is no general method which can be applied to all modelling problems. To reduce the effect of the uncertainty in the data, constraints on the form of the possible solutions must be used. Such constraints are basically any existing knowledge about the process, or its environment. The framework should therefore be able to include such knowledge wherever possible. If *a priori* knowledge of model structures or parameters is included in the estimation process, the model can be called a *grey-box* model. The task for the tools developed in this thesis is therefore to support the process

$$\text{Observed Data} + \textit{A priori}\ \text{Knowledge} \longrightarrow \text{Model}$$

as well as possible.

2.1.1 Empirical models of dynamic systems

Any real physical system's reactions to a given input are not likely to be instantaneous, and will depend on previous inputs. We are therefore dealing with dynamic systems, which are processes described by difference or differential equations, where the current output of the system depends not only on the current external stimuli, but also on previous stimuli and internal states. This brings a new dimension to the modelling process, as the optimal sampling rate of the unknown system must be estimated, the order of the system must be taken into account, and the model's dynamics must be validated. To represent the dynamic aspects of the system, it is necessary to have memory elements to store past inputs and model states which are passed to a static nonlinear model.

Most of the work in this thesis is based on the assumption that the process can be represented by a non-linear auto-regressive model with exogenous inputs (NARX) over the whole operation envelope. As the implementation of the learning system will be on a digital computer, we consider discrete-time non-linear systems having the general form

$$y(t) = f(y(t-1), \dots y(t-n_y), \boldsymbol{u}(t-k), \dots \boldsymbol{u}(t-k-n_u)) + e(t). \tag{2.2}$$

Here, $y(t)$ is the system output, and $\boldsymbol{u}(t)$ the input. The analysis is limited to multiple-input single-output (MISO) systems so that $y(t) \in Y \subset \mathcal{R}$ and $\boldsymbol{u}(t) \in U \subset \mathcal{R}^{n_{in}}$. Here k represents a time delay. The *information vector* passed to the nonlinear model is therefore defined as

$$\boldsymbol{\psi}(t-1) = [y(t-1), \dots y(t-n_y), \boldsymbol{u}(t-k), \dots \boldsymbol{u}(t-k-n_u)]^T \tag{2.3}$$

or, if a zero-mean disturbance term $e(t)$ is to be taken into account on-line, the widely studied NARMAX (Non-linear ARMAX) framework (Leontaritis and Billings, 1985) from system identification can be used,

$$\boldsymbol{\psi}(t-1) = [y(t-1), \dots y(t-n_y), \boldsymbol{u}(t-k), \dots \boldsymbol{u}(t-k-n_u), e(t-1), \dots e(t-n_e)]^T, \tag{2.4}$$

where $e(t) \in E \subset \mathcal{R}$.

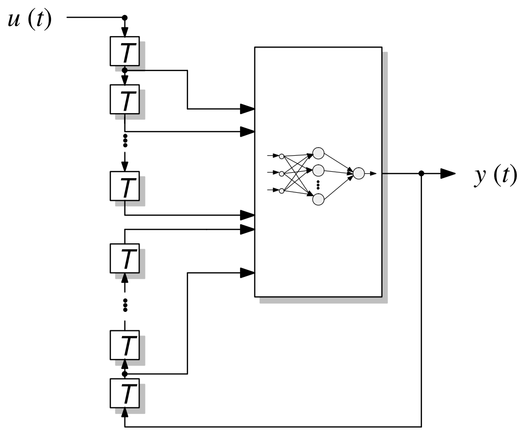


Figure 2.2: Using a tapped delay line (the T's represent delay elements) and static neural net to represent nonlinear dynamic systems.

The NARX, or tapped delay line approach (see the graphical representation of equation (2.2) in Figure 2.2) is fairly pragmatic, as it brings the neural network into the world of conventional system identification theory, where the static neural net can be viewed simply as a technique for function approximation. One disadvantage is that the input dimension becomes very large for even simple systems. Also, if the sampling rate has been correctly set, the data from a dynamic system will be highly correlated on the delayed inputs from the output state – i.e. it is impossible for the data to fill the input space, as shown in Figure 2.3, which shows a slice through the input space of the rolling mill data discussed in Chapter 6. This can

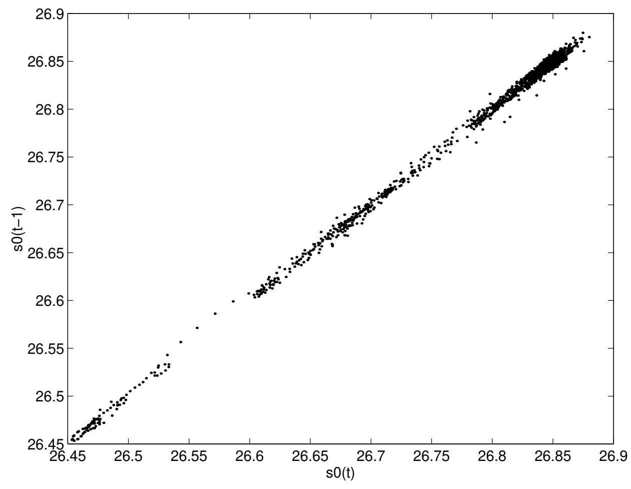


Figure 2.3: A slice through the input space of the rolling mill's training set, showing a variable plotted against the delayed version of itself.

have serious consequences for some learning algorithms, and model structures. If the model structure is such that the nonlinearity results from partitions which are orthogonal to the axes of the input space (an underlying assumption in learning systems such as MARS, ASMOD, ABBMOD, ID3), the model will be less suited to modelling such dynamic processes than an algorithm which can partition the input space more freely, e.g. by placing basis functions on data points from the training set, or by allowing axis-oblique partitions of the input space.

An alternative to an external tapped delay line is to have memory and feedback within the network itself, where the dynamics are learned by the network. Such recurrent networks, which contain local internal feedback connections have received a great deal of attention in the literature (see, for example (Hopfield, 1984, Żbikowski, 1994)). These networks are mathematically elegant structures which are, however, often more difficult to understand and train than static networks for practical problems. The local model network mixed order systems described in Section 2.5.4 are less susceptible to the dimensionality problems inherent to the NARX representation.

A further alternative to the NARX representation is to pass the derivatives of the variables to the model, which makes the data distribution more even and reduces the conditioning problems often caused by the highly correlated inputs found in the training data for NARX systems. Because of the sensitivity of such systems to noise, it is important to filter the data beforehand.

2.1.2 Classes of learning systems

As can be seen from the wide variety of definitions in Section 1.1, machine learning can be studied from a number of widely differing viewpoints. This is largely because the use of observed data to provide the information needed to better fulfil a given goal, whether on-line, or off-line, is a major aspect of almost all areas of science and engineering. This makes it important to try and describe the basic features of learning systems in as general a form as possible, so that the various branches of research can be more easily integrated. The systems shown in Figures 2.4-2.6 describe the range of learning ability for automatic model creation.

Systems which cannot learn

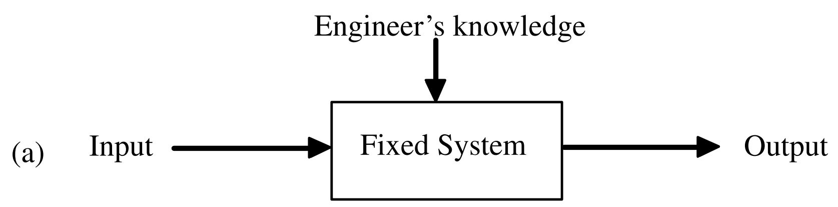


Figure 2.4: System without learning ability – behaviour is pre-programmed

System (a) is the traditional method of solving a problem. The problem is analysed, decomposed into subsystems, and an explicit solution is developed by human engineers using their knowledge of the problem, its constraints and its environment (note that much of this knowledge is however based on earlier empirical work). The resulting program/expert system/controller/classifier is then tested with experimental data and put into use. *No automatic learning is used* in the development, making the development process sensitive to changes in the environment, the system or the development goals.

Systems which can be trained

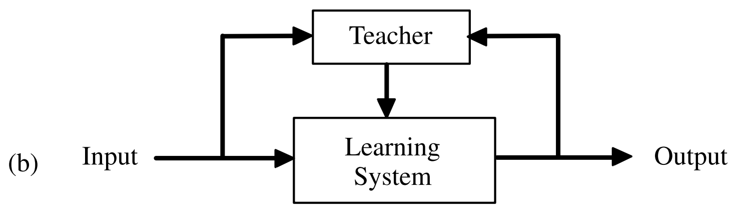


Figure 2.5: Supervised learning system

System (b) is a *supervised learning* system. The machine's task is therefore described by the 'teacher' using examples of what should be done, rather than instructions about how to do it, and criteria for the evaluation of the learning system's performance, compared to the output the 'teacher' expected. The learning system then learns the optimal mapping from input to output with the help of an external 'teacher'. The task of adjusting the learning system's parameters and structure to achieve this is not a trivial matter. For any learning structure the optimisation task usually becomes more difficult, the more flexible the representation used.[1]

Systems which can self-organise

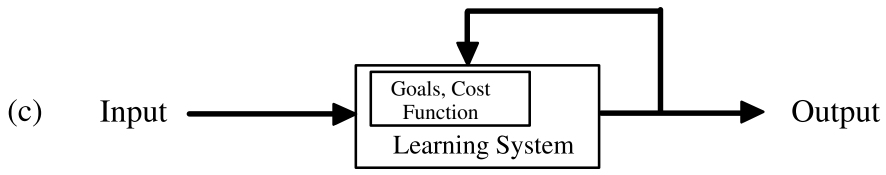


Figure 2.6: Self-organising system

System (c) is an *unsupervised learning* or *self-organising* system. There is no external teacher and the system adjusts its parameters and structure to optimise some predetermined cost function without instructive feedback from an external body. This very general description of an unsupervised system is extremely far-reaching, and should not be confused with the limited implementation of the current generation of unsupervised learning algorithms described in the literature, e.g. the *Self-Organising Maps* (Kohonen, 1990), which are basically clustering algorithms. The important feature of self-organising systems is that the goals and cost functions are built into the system, allowing them to autonomously adapt their behaviour to better achieve their given goals, rather than learning to imitate an existing solution.

The unsupervised learning system can provide a more general form of learning system, depending on the complexity of implementation. The teacher has, in effect, been 'hard-wired' into the learning algorithm in the form of goals and a quality functional which is dependent on the inputs and resulting outputs, taking system constraints into account. The methods described in this thesis are not directed towards the direct implementation of Self-organising learning systems, but will involve some of the self-organising principles to provide the structure for the solutions within the supervised learning schemes, so the desired output is described explicitly in the training set, although the structure of the learning system is identified from the training data in an unsupervised manner – i.e. the 'teacher' does not have to tell the learning system exactly what its structure should be and which training data it should use to train its parameters.

[1]A more abstract form of this style of learning can be seen in *reinforcement learning* systems, where the system is no longer told exactly what to do, but is given graded feedback about the system's performance.

2.1.3 Research paradigms for automatic empirical modelling

Cybernetics, control and adaptive systems

The idea of artificial systems capable of learning is not as new as some people imagine. Many of the basic ideas present in modern research have been in print since the 1940's, when Norbert Wiener initiated the field of *Cybernetics* (Wiener, 1948) – a field which, like neural networks, brought together control engineers, biologists, mathematicians, sociologists and computer scientists. The concepts of learning systems, especially with ideas from biological systems, were examined throughout the world, the popularity of the field shown in review papers such as (Sklansky, 1966) and (Fu, 1970). Steinbuch's work was often ahead of its time (note the rolling mill application of neural nets in (Steinbuch, 1963)!), while Tsypkin's work (Tsypkin, 1971, Tsypkin, 1973) still clarifies many of the basics involved and describes the application of learning and adaptation to a variety of technical systems.

Although cybernetics, like neural networks, is still seen as an elegant framework for promoting communication and interaction between various fields of research, it could not stand the strain brought by the explosion in progress and knowledge in its various sub-fields, which lead to the specialisation and more insular behaviour now common in the fields originating from cybernetics.

System identification (Ljung, 1987) is the area of modern control theory with the closest similarity to the learning systems philosophy described in this thesis, although the majority of the identification work has been based on the basic assumption of linear representations of the systems. Important contributions to the area of non-linear system identification can be found in the NARMAX modelling methods (Billings, 1980, Chen and Billings, 1989), based first on polynomial representations, then later on neural network implementations with multi-layer perceptrons and RBF (Radial Basis Function) Networks (Chen et al., 1990, Chen and Billings, 1992).

On-line adaptation of the model can be used to cope with slow variations in the parameters of a given system, or of a change in operating point. This adaptation allowed the use of linear models for the control of non-linear systems and environments, and was also a significant, if restricted, step towards learning systems. Adaptive signal processing is covered in (Haykin, 1991) and a review of the adaptive control field is given in (Åström, 1987), where he defines adaptive control as: *a special type of nonlinear feedback control, where the states of the process can be separated into two categories, which change at different rates. The slowly changing states are viewed as parameters.*

Most of the practical industrial applications of adaptive control systems have, however, been limited to self-tuning control, where linear controllers are adjusted automatically for a given system. In many cases, an accurate time-invariant nonlinear model of a system will remove the need for on-line adaptation.

Statistics

Many of the tasks routinely ascribed to learning systems (i.e. classification or modelling ability derived from data, as opposed to *a priori* knowledge about the problem) can be viewed as regression problems, and many of the tools for their solution have thus been a standard part of the statistician's toolkit for decades. (Barron and Barron, 1988) discusses the similarities between statistical methods and more recent developments. Fisher's simple linear discriminant algorithm is functionally similar to the Perceptron (Fisher, 1936), and has led to the more powerful quadratic and polynomial discriminant algorithms. Nearest-Neighbour algorithms are also powerful techniques, despite their simplicity (Dasarathy, 1990, Duda and Hart, 1973). Spline and Kernel based modelling methods have much in common with Basis Function Nets (to be described in Section 2.3.1) (Wahba, 1990, Wahba, 1992, Kavli, 1992) and local approximation ideas (to be described in Section 2.3.2) have also been used in statistics. The statisticians also developed tree based regression systems independently from the AI world (e.g. CART (Breiman et al., 1984) and MARS (Friedman, 1991)). A further important contribution from statistics is the wealth of evaluation techniques developed to validate the models and classifiers derived from observed data (e.g. cross-validation techniques, robust cost functions, sampling techniques, additive models etc.). *Projection pursuit* methods are also closely related to Multi-Layer Perceptron neural networks (Zhao, 1992).

Machine learning – the symbolic view

Non-neural machine learning has also a relatively long history, dating back to Samuel's checkers work (Samuel, 1959, Samuel, 1967) and although the neural network research faded away at the end of the 1960's, the Artificial Intelligence community kept working on learning systems. Most of this work was in the symbolic domain, as opposed to the numerical environment of neural networks, statistics and system identification. Despite this, some of the work on Inductive Learning is relevant to modelling nonlinear dynamic systems and, in general, the examination of different viewpoints often provide new insight into the learning mechanisms, improving understanding for both groups of researchers. Techniques such as decision trees have been used as models of dynamic systems (Isaksson et al., 1991). Omohundro describes a variety of standard algorithms which can be applied in many situations with more success than neural networks (Omohundro, 1987). *Case-Based Learning* also has applications in modelling (Kolodner, 1993), and we describe the overlap of Case-Based Learning with nearest neighbour methods and RBF neural networks in (Murray-Smith and Thakar, 1993), dealing with aspects of customer modelling from sales information, where the basis functions performed interpolation between cases. A general, easy to read overview of machine learning is given in (Weiss and Kulikowski, 1991), and (Shavlik and Ditterich, 1990) is a collection of the most significant papers in the field's history. (Michalski et al., 1983, Michalski et al., 1986) and (Kodratoff and Michalski, 1990) also provide collections of significant work.

2.1.4 Empirical modelling with neural networks

The basic philosophy of the *neural network* approach to machine learning is to use the style of information processing evident in the physiology of living beings as the inspiration for a computational paradigm. A large number of simple processing elements, based on highly simplified mathematical models of neurons, are densely interconnected by weighted connections which roughly represent the axons and synapses of the biological model. These structures are then optimised using a variety of algorithms, some aspects of which – although by no means the majority – are based on biologically plausible techniques.

The resulting networks can represent complex mappings from their input nodes to output nodes, making them interesting for engineering applications. The neural network architectures used in this thesis are examples of flexible black-box models which are created on the basis of input-output data pairs. There are several advantages of using neural networks to model systems, the most compelling of which is their ability to model continuous, non-linear, multivariable systems. As mentioned in the previous section, this ability is not unique to neural networks, but the progress made in recent years, due to the increase in available computing power, has shown that the new algorithms often perform well compared to previous methods of nonlinear system identification. More important, perhaps, is the multidisciplinary nature of the research, which has brought new ideas from artificial intelligence, mathematics and biology together with application-oriented engineering and computing practice.

A bit of history

The roots of the neural network approach to learning can be traced back to 1943, when McCulloch and Pitts (McCulloch and Pitts, 1943) made some proposals about how simple neural-like networks could compute. Hebb then suggested a biologically plausible learning rule for such networks (Hebb, 1949). The beginning of the field as it is known today can be found in Frank Rosenblatt's work on *Perceptrons* (Rosenblatt, 1962). He pioneered the use of formal mathematical analysis and the use of digital computers for simulation. He also made some over-enthusiastic claims about the power of perceptrons compared to normal computers which would spark off a debate which is still running to this day, but which then led to the field being laid dormant for twenty years. This was a combination of the limitations of the contemporary hardware, and the irritation caused to other scientists by Rosenblatt's claims. It was this irritation which lead Minsky and Papert to publish the famous *Perceptrons* book (Minsky and Papert, 1969) where they rigorously analysed the existing techniques and pointed out the limitations of the structures and their inability to scale up to larger problems.

(Widrow and Hoff, 1960), coming from a more engineering background also proposed a similar network called an *Adaline*, trained using the *Delta-rule*, which eventually found widespread use in telecommunications systems as an adaptive filter. Kurt Steinbuch was another engineer who did significant work in learning control systems, including his *Lernmatrix* learning system

(Steinbuch, 1961, Steinbuch, 1963). Grossberg's work in the 1970's introduced new ideas from biology and psychology to create the ART series of non-linear dynamic architectures. Albus made important contributions with the CMAC architecture in the field of learning in robotics, using many biologically motivated methods (Albus, 1972, Albus, 1975b, Albus, 1975a). Holland examined adaptation in (Holland, 1975), introducing *Genetic algorithms* and *classifier systems* for learning.

The main reason for the current renaissance in machine learning in general was the boom in 'artificial neural networks' in the eighties, precipitated by Hopfield's work (Hopfield, 1982, Hopfield, 1984) and the PDP Group's books (Rumelhart and McClelland, 1986). This was combined with a general frustration with the lack of progress in 'conventional' AI for tasks such as speech recognition, vision and pattern recognition, and the fact that the powerful computing hardware needed by these numerically intensive algorithms was now widely available.

The neural network 'label', as used in the current literature, seems to be applicable to almost any modelling or classification scheme, as networks can be used as convenient graphic representations for many mathematically described systems. (Barron and Barron, 1988) provides a clear description of the similarities between neural nets and statistical methods. General introductory books include (Haykin, 1994, Hertz et al., 1991, Wassermann, 1993). Historically important papers are reprinted in the *Neurocomputing* collections (Anderson and Rosenfeld, 1988, Anderson and Rosenfeld, 1990). Reviews of the neural network field for modelling and control include (Miller et al., 1990, Hunt et al., 1992, Warwick et al., 1992).

The Multi-Layer Perceptron

As discussed earlier, the first neural architecture to be implemented and studied in detail was Rosenblatt's Perceptron (Rosenblatt, 1958). This is a very simple system, which was suggested as a simple model of biological neurons present in the human visual system. The output is a function (usually non-linear, such as a hard limiter or sigmoidal function) of the sum of the weighted inputs.

$$\mathbf{y} = f(\mathbf{x}\mathbf{W} + bias), \tag{2.5}$$

where $f(\cdot)$ is the activation function, and $\mathbf{W}$ is the weight matrix connecting inputs $\mathbf{x}$ to outputs $\mathbf{y}$.

The extension of the perceptron to allow multiple layers – the *multi-layer perceptron* in equation (2.6) (see Figure 2.7 for a graphical representation), became widespread in the eighties after suitable learning algorithms, such as *back-propagation*, became widely known.

$$\mathbf{y} = f(\mathbf{W}_2 f(\mathbf{x}\mathbf{W}_1 + bias_1) + bias_2), \tag{2.6}$$

where $\mathbf{W}_1$ is the weight matrix connecting inputs $\mathbf{x}$ to the hidden layer neurons, and $\mathbf{W}_2$ connects the hidden layer neurons to the outputs $\mathbf{y}$. Back-propagation was developed independently by various workers (Werbos, 1974, Parker, 1985) with best known version being

(Rumelhart et al., 1986), bringing the neural networks field back to life. The Multi-Layer Perceptron soon became, for better or worse, the most commonly used (and abused) network architecture in the history of neurocomputing. The widespread use came about because of the

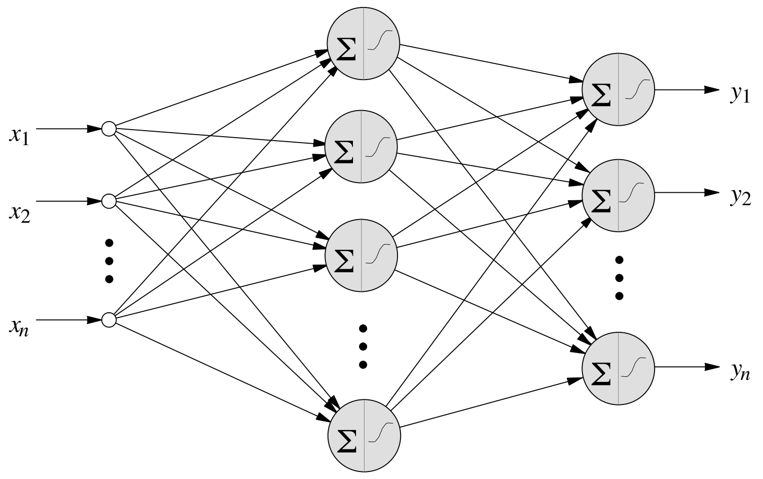


Figure 2.7: Multi-layer Perceptron

real flexibility of the structure in coping with complex high-dimensional problems, and because it managed to produce often excellent results compared to competing methods. The disadvantages of the structure are the slow training times associated with the back-propagation learning algorithm and poor transparency of the algorithm and of the trained networks. This led to an alchemy-like approach to training taken by the majority of researchers, producing a huge variety of heuristically motivated 'new improved' learning rules.

2.1.5 What is wrong with modelling with neural nets?

Empirical modelling as a methodology, even in its 'hard' form of System Identification, has its disadvantages. Often, the models produced in empirical modelling, even if they have achieved a useful representation of the process being modelled, give little physical insight into the process, since they are rarely based on detailed knowledge about the process structure. This is closely related to another disadvantage, which is that the product of a purely data based modelling process usually has limited validity in situations different to that in which the data was collected (e.g. at different working points, given different inputs, or a change in environment). The identification methods in existence are also still very much highly interactive art forms. They depend to a great extent on utilising as much knowledge about the process in question as possible, to simplify the modelling problem, and on an intuitive feel for the techniques used, as well as an understanding of the complex statistical tools available for the analysis of the data and estimation of the parameters.

While things have improved slightly in recent years, a characteristic of much work in the field of neural networks is the lack of rigour, compared to more established 'competing' areas such as statistics and system identification (see (Sjöberg et al., 1994) for a good discussion of the overlap between the fields, with the intention of 'removing the mystique' surrounding neural nets). This had certain advantages initially, as the fact that the area was linked to human intelligence often made it more appealing to young researchers, as well as to the various bodies which sponsor research and development, than the more highly mathematical established fields. Although the field has been rightly criticised for the wild claims made initially, it did bring in fresh ideas from other fields such as AI, psychology and physiology. These insights have enriched the scope and methods of the research into learning and adaptive systems, as well as a very application oriented style of research which got the method applied to a wide variety of problems.

The problem facing researchers at the start of the 1990's was that much of the experience gained in the existing fields had been ignored, and the field was too strongly linked to one architecture – the multi-layer perceptron. Initially much emphasis was laid on the fact that the multi-layer perceptron architecture is theoretically powerful enough to represent arbitrary nonlinear mappings, but such results did not help produce efficient training algorithms. A large proportion of the research was invested in a variety of problem specific 'fiddle factors' designed to speed up the optimisation process–which often took days of computing time–and to improve generalisation. The multi-layer perceptron can often be highly successful at modelling a given system, but despite this it is not ideal for many modelling tasks. The long training times are not suited to the iterative and interactive nature of the modelling process, especially as it was unclear when learning should be stopped, or how many units or layers a net should have for a particular problem. The poor interpretability limits the user's ability to validate a trained network, and limits the networks' applicability to safety critical situations. There is no straightforward way of introducing prior knowledge directly into the network structure, even though for many tasks the ability to simplify the problem using such knowledge is critical to reaching the desired level of accuracy.

Combining the different paradigms

While none of them is ideally suited to the task, the research paradigms for learning models described in this section can all contribute significantly to the goal of having a flexible, interactive learning system which robustly forms a model of a complex process, given a mixture of observed data and *a priori* knowledge. Systems theory and System Identification provide the user with a mathematically well founded base of theory and experience for modelling dynamic systems from observed data. Much of the work has been restricted to linear systems, but the basics remain relevant, and many of the algorithms can be directly applied to the hybrid local model architectures described in Section 2.3.2. Similarly with statistics, where the theory and experience developed by statisticians in areas such as experiment design, model optimisation, model validation and function approximation is an invaluable aid to a better understanding and evaluation of newer systems such as neural networks.

The models used in this thesis can benefit not only from numerically oriented techniques, but also from the integration of fuzzy logic aspects and symbolic machine learning systems which then allows the incorporation of linguistic or rule-based knowledge into the learning system. Section 2.3.2 presents an architecture suitable for the integration of the various paradigms.

2.2 The Modelling Process

The work in this thesis is aimed at improving techniques for the design of models of non-linear dynamic systems (the $f(\psi(t))$ in equation (2.1)), with the aid of computationally intensive data-driven techniques. The goal is to produce a mapping from the input space to the output space which best fits the observed data and meets the specified constraints. This involves integrating knowledge about the system with data observed from identification experiments. This allows the developer to produce better model structures and identify their parameters, as well as being able to validate the accuracy of the final model. Modelling from data and knowledge is often viewed as an art form, mixing 'expert' insight with the information in observed data, while using *ad hoc* simplifications to make the problem solvable. The typical

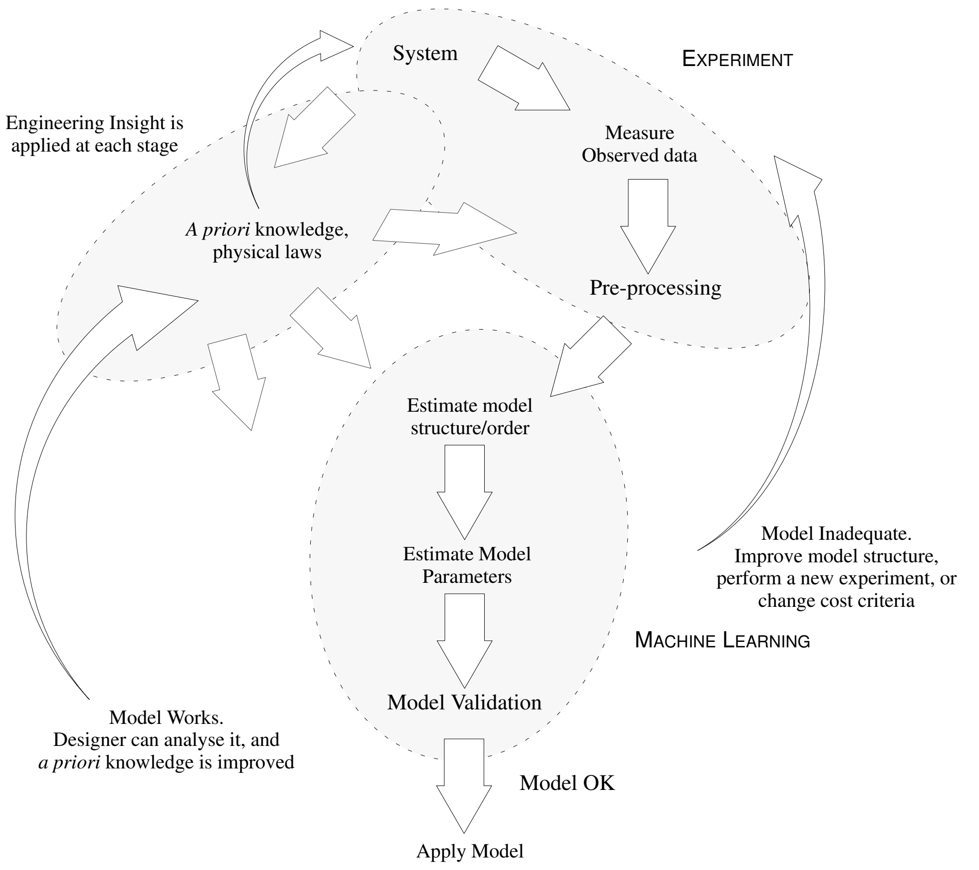

System
EXPERIMENT
Engineering Insight is applied at each stage
Measure Observed data
A priori knowledge, physical laws
Pre-processing
Estimate model structure/order
Estimate Model Parameters
Model Inadequate. Improve model structure, perform a new experiment, or change cost criteria
MACHINE LEARNING
Model Validation
Model Works. Designer can analyse it, and *a priori* knowledge is improved
Model OK
Apply Model

Figure 2.8: The Engineering Cycle for Training a Model

modelling cycle is shown in Figure 2.8, showing the interaction of *a priori engineering insight*, *experiment design*, *data acquisition* and *pre-processing*, followed by *machine modelling* and *validation*. Each of these phases will now be looked at in more detail.

2.2.1 Using *a priori* information

Modelling from observed data supported by learning systems is supposed to reduce the need to understand the detailed physical relationships within the system under investigation, but as can be seen in Figure 2.8, a major feature of the cycle is the important role of *a priori* knowledge at each stage of the modelling process. *A priori* information is initial knowledge about the system, or problem in question. This includes aspects such as the goals of the problem, the characteristics of the process, its parameters, the effect of the environment (expected noise, disturbances), and the robustness requirements for different situations. Learning or adaptive systems are usually used because of the insufficiency of the *a priori* information (few complex processes in reality can be described completely by *a priori* knowledge, due to the effects of noise, disturbances and unmeasured states). Such systems try to compensate for this by the continuous use of current information about the system:

> '*A priori* information is the basis for the formulation of an optimisation problem, but the current information provides the solution for the problem' (Tsypkin, 1971).

This describes the basis of Adaptive Control. Although the goal of learning systems is to reduce the need for *a priori* knowledge, its use almost invariably makes the learning problem more tractable. *A priori* knowledge can be used both *implicitly* and *explicitly*. It is used *implicitly* when framing the problem, in the act of creating a representative training set, deciding which learning algorithms and structures are best suited to the problem and which inputs and pre-processing algorithms are likely to make the learning task easiest.

The *explicit* use of *a priori* knowledge involves the direct integration of models or rule-bases into the learning system to reduce the learning effort. Knowledge about variable interaction can also be used to decouple the inputs, and reduce the dimensionality problems. Model structure, dynamic order and sampling rate are dependent on *a priori* knowledge. Knowledge about physical constraints can be used to limit the generalisation in areas with insufficient data. Existing models or controllers (e.g. human operators) can also be copied, where valid. Models of the environmental disturbances expected can also be included, as can knowledge about the relative importance of various areas of the input space.

Another important aspect of modelling from observed data is that the machine learning involved also usually results in the human engineer gaining a better understanding of the system, and the *a priori* knowledge about the system in question therefore being improved. The human designer is still a vital part of the process, and the goal should be to enhance the power of the interactive software by automating the learning process wherever possible, but by giving the engineer the freedom to intervene at each stage. As knowledge about the system being modelled increases, more possibilities of simplifying the machine learning should become clearer, allowing it to be used more successfully, closing the loop in the modelling cycle seen in Figure 2.8, where the *a priori* knowledge is improved by the availability of the validated model.

The final product, a working model, can therefore be seen as a contribution to the more general pool of engineering and scientific insight. The laws, rules or models we take as *a priori* today were also once poorly understood observed behaviour, which was then measured, analysed and turned into some simpler law or model. Kepler's laws of planetary motion were found only after painstaking acquisition of observed data, and the application of a variety of model structures to the data, estimation of the model parameters and validation of the models on new data!

2.2.2 Creating the training set – design & pre-processing

The *training set* $\mathcal{D}$ is the data set used for optimising the parameters and structure of the learning system. It is generally necessary to devote a great deal of attention to the process of extracting preprocessing and representing the data for training.

Experiment design

It is vitally important that the training set represents the task in hand correctly and adequately over the whole input space, but it is important to consider the relative importance of the various areas of the input space. In many situations a system spends most of its time in a particular operating region. It may make sense to weight this more heavily in the learning process. In others, a particular aspect of the model must be very accurate, for example the reaction to a step change in the inputs or disturbances is important for many processes. In others it is important to have very accurate models in relatively stable areas, as this is where the process spends most of its time. In many applications certain areas of the system are associated with danger – how should these be treated in the model? Constructing a representative training set is often far more difficult than learning the task from the training set. The general process of training set creation is shown in Figure 2.9[2]. A significant aspect of the diagram is the existence of important disturbances which can not be measured. The ability of the training procedure to cope robustly with such disturbances will often determine to a great extent the procedure's usefulness in real world applications. It is not only important

[2]In many cases the model is to be developed in order to produce a controller for the process in question. A valid point is to ask how this controller should be developed, without an adequate model to design it with? In critically unstable cases this will be a major problem, but in many cases the task is to improve on an existing controller which can be used for the purposes of data acquisition. There may, however, have to be a series of acquisition and modelling runs, as some aspects of the processes may not be apparent with less powerful controllers. In many cases, however, the learning system will have to make do with whatever data is available, because of the costs of extensive experimentation both in terms of money, as well as organisational effort.

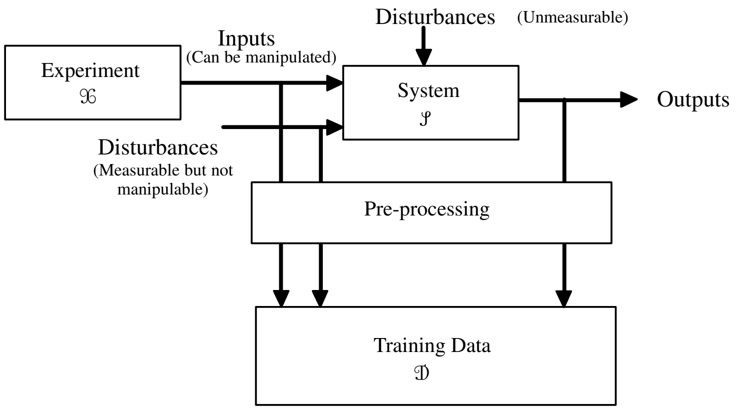

Disturbances (Unmeasurable)
Inputs (Can be manipulated)
Experiment 𝔛
System 𝔖
Outputs
Disturbances (Measurable but not manipulable)
Pre-processing
Training Data 𝔇

Figure 2.9: Acquiring and preparing the training data

to have training data covering the input space, but to have larger amounts of data where the decision surface is most complex, and to have some information about the relative significance of individual training data. This could be in the form of probability distribution functions, showing the relative frequency of particular situations, or the definition of areas which are particularly important. The science of creating an optimal sampling of the input space is called *Experiment Design* (Fedorov, 1972, Goodwin and Payne, 1977, Ljung, 1987), and has been picked up by workers in machine learning within the *active learning* framework.

Active learning

The neural network community traditionally threw every available training example at the network during the learning phase, irrespective of redundancy, noise levels, or local complexity in the process being modelled. The disadvantages of this procedure are being increasingly recognised, leading to the introduction of *active learning* methods which enhance the training set used during the learning process, as shown in Figure 2.10.

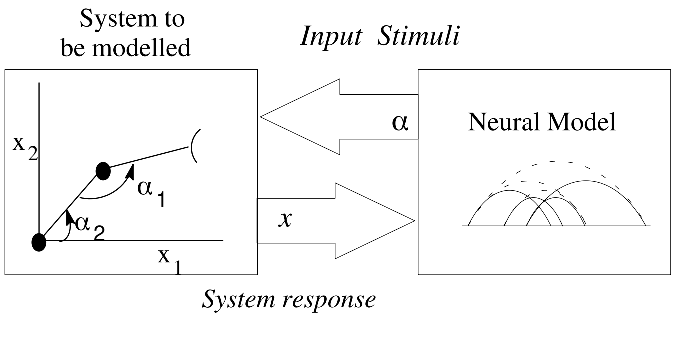


Figure 2.10: Active learning – exploring the input space. The learning system can interact with its environment to obtain new training data.

The research in *active learning* has gained momentum in recent years, see (Cohn et al., 1990, Cohn, 1994) and (Cohn et al., 1994). (Plutowski, 1994) is a recent thesis in the area. The use of local basis functions to guide active learning is described in (Murray-Smith, 1992). As described in (Thrun, 1992), active learning can be viewed as either *directed* guided search or *undirected*, random search. The undirected active learning methods described in the literature have the disadvantage that the effort needed to learn a given process increases exponentially with the dimension of the input space. Active learning can include *active sampling* and *active selection*,techniques:

- *Active sampling* is associated with the experiment design phase – in which regions of the input space should training data be acquired to best minimise the uncertainty in the model? The uncertainty estimate in the model is based on the information in the existing training set.[3]
- *Active selection*, which involves the most efficient use of a large existing training set uses closely related techniques to active sampling, without the connection to the environment.

The *active* label is due to the fact that the selection or sampling processes are dependent on the learning process and model structure, so that the learning process can be seen more broadly as actively *exploring* its environment or training data, then *exploiting* that data to optimise its performance.The use of complexity-based active learning algorithm in local model networks is described in Section 4.3, and an active sampling routine is used to improve the learning process in the rolling mill application in Chapter 6.

Pre-processing

A vital factor in all empirically driven model building approaches is that the training set should be pre-processed before learning. Any way in which the data can be transformed to make learning easier will lead to a significant improvement in performance. Pre-processing includes all action applied to the measured data, including which sensor information to use, how to sample it, how the amount of data can best be reduced, what transformations should be applied, and how can it be best encoded to make the learning task easier. The data may have to be pre-filtered to remove noise effects, anti-aliasing techniques will be necessary when continuous signals are sampled, outliers can be removed, or extra information can be used to reduce the distorting effect of measurable disturbances. In many cases non-linear characteristics in sensors or actuators are well known in advance, and the nonlinearity can be counteracted before learning commences. Somewhere between pre-processing and representation lies the aspect of minimising variable interaction. The data can be transformed to a lower

[3]The concept of giving a learning system a 'sense of curiosity', based on internal estimates of its own accuracy, and giving it the ability to search the most promising areas of the input space for new information is of major importance for the future of autonomously learning systems, and the techniques used are closely related to those needed for the recursive identification of time-varying processes, where it is important to be able to 'forget' the training examples in the right areas, while learning from those describing the new behaviour of the process.

dimension using principal components analysis to reduce the problems related to the '*curse of dimensionality*'. It may be known that particular variables interact in a given way, whether additively, multiplicatively or nonlinearly (Hastie and Tibshirani, 1990). This information can be used to limit the freedom of the learning system, forcing it to learn more efficiently and generalise more robustly (see Section 2.5.4 for more details on how this information can be used). In general, the pre-processing applied to the raw data is often a critical stage in the development of models from observed data, and its importance should not be underestimated for learning systems.

2.2.3 Learning algorithms and knowledge representation

Once the training data has been acquired and pre-processed, a *knowledge representation* ability is required for the model to be able to learn the data and a *learning algorithm* which can be used to go from the information presented to the learning system, in the form of inputs, outputs and feedback from the environment, to the desired representation. In simple cases this can be viewed as an optimisation process, where the optimal set of parameters for a given structure is found, or it can also involve the *construction* of the representation itself. Learning systems involve a wide variety of knowledge representation techniques, from simple numerical parameters or symbolic features to complex specialised structures. Each style of knowledge representation is biased towards a particular class of problems, a fact which can be seen in the classical fields of statistics and system identification, as well as the newer areas of neural networks and machine learning, each of which tend to use structures suited to the problems faced in that field

When considering learning systems in general, for any given data set, a variety of possible *model structures* usually compete for the best representation of the data. It is important that the most suitable style of representation is chosen for a given problem, as the right choice of model structure will be a major factor in producing a model which is able to 'generalise' correctly. *Generalisation* is the ability of a learning system to give a 'correct' output to inputs on which it has never been trained (see Figure 2.11). These could be new, noisy or incomplete inputs. Memorisation, or learning the training set to perfection is *not* the goal of learning – a random access memory (RAM) can do this adequately!

The resulting quality of generalisation for a given problem will depend on the problem definition, encoding of the features, the quality of the training set, the power and suitability of the representational structure $\mathcal{M}$ and the learning algorithm used. To analyse the trade-off between learning the training set $\mathcal{D}$, and generalising to unseen inputs, it can be helpful to decompose the modelling error $J(\mathcal{M},\mathcal{D})$ into two aspects, the *bias* $J_B(\mathcal{M},\mathcal{D})$ and the *variance* $J_V(\mathcal{M},\mathcal{D})$ (Geman et al., 1992).

$$J(\mathcal{M},\mathcal{D}) = J_V(\mathcal{M},\mathcal{D}) + J_B(\mathcal{M},\mathcal{D}), \tag{2.7}$$

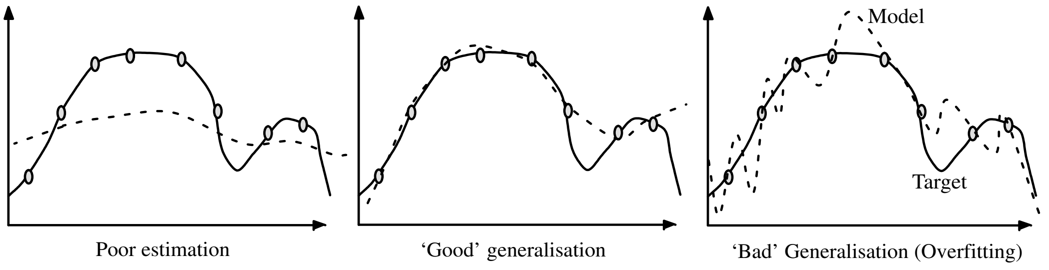


Figure 2.11: The generalisation/overfitting dilemma.

where the bias is

$$J_B(\mathcal{M},\mathcal{D}) = \left(E_{\mathcal{D}}[\hat{f}(\mathbf{x};\mathcal{M},\mathcal{D})] - E\left[y(\mathbf{x})|\mathbf{x}\right]\right)^2, \tag{2.8}$$

where $E_{\mathcal{D}}$ is the expectation over the training set $\mathcal{D}$, and the variance

$$J_V(\mathcal{M},\mathcal{D}) = E_{\mathcal{D}}\left[\left(\hat{f}(\mathbf{x};\mathcal{M},\mathcal{D}) - E_{\mathcal{D}}\left[\hat{f}(\mathbf{x},\mathcal{M},\mathcal{D})\right]\right)^2\right]. \tag{2.9}$$

The *bias* of a model is the average difference from the real system of models trained on a number of training sets, thus indicating what the system could not learn, even when given the information. The *variance* of a model is the average variation of the model estimates from all trained models to the 'average' model over all data sets, and can be used as an indicator of how robust the learning process was.

The flexibility required of the model to be able to model an arbitrary data set (reducing the bias part of the error) conflicts with the desire to reduce the variance of the resulting estimate. Use of a large flexible representation will reduce the bias, but without a correspondingly large data set is likely to lead to the *overfitting effect*, where the training set is learned adequately, but generalisation is poor – equivalent to high variance. The system has either learned the noise in the data, or has learned the data correctly, but interpolates between data points poorly. It is therefore important to use a suitably sized model structure for the given training data – the system must be over-determined (i.e. more training data than parameters, and – importantly for nonlinear systems – it must be locally over-determined in the complex areas of the input space). *One of the most important features of a learning algorithm is that it reliably finds a solution which generalises robustly to new data.* The regularisation methods described in Section 2.5.1 are methods which artificially increase the bias to improve the variance and therefore create more robust networks. In practice, algorithms which produce reliable models will be far better suited to engineering problems than algorithms which sometimes produce excellent results on one problem but then fail dismally on the next application.

2.2.4 Model validation

Once the model structure and parameters have been identified it is necessary to validate the accuracy of the final product. This is obviously a very important stage in a process which by its very nature has relatively little in the way of 'common sense' intuition about the behaviour of the target system. It is therefore important to combine data-driven validation – is it adequately accurate and robust for its purpose? – with more subjective validation, i.e. does the model behave in a way which seems physically plausible? Can the final machine learned model be interpreted to give the human engineer a better understanding of the system in question? Model validation is an issue which has often been neglected in the literature on learning systems, but one which is very important in industrial situations. There will usually be a trade-off between flexibility and interpretability, the outcome of which will depend on their relative importance for a given application.

Use of n-fold cross-validation

The most commonly applied method of predicting the accuracy of a neural network is that of measuring the quality of the system's response on the training and test sets, assuming that the data set was complete enough to have encountered all of the important areas of the input space. The general technique is called cross-validation. Generalisation ability is closely tied up with the concept of expected error prediction, so this is of fundamental importance to learning systems (Stone, 1974).

Resampling methods such as cross-validation aim to give unbiased estimates of the error rate of a learning system. They make minimal assumptions about the statistics of the training data. The simplest form of this is to use two sets of data (training and test sets), where the first is used to train the system and the test set is used to validate the results. A more general form of this is n-fold cross-validation, where the available data are partitioned into n subsamples. Each subsample is tested by training the net with the other $(n-1)$ subsamples and the error rate is then the average of these n sub-samples. This reduces the bias present in the error estimate, but is often a time consuming process and the error rates have a high variance. (Weiss and Kulikowski, 1991) *Leave-one-out validation* is a special case of cross-validation, where one example of the training set is left out to provide a test example for the model trained on all the other examples. This is then repeated until each member of the training set has been used as a test example. It is therefore very computationally expensive, and although it provides the most accurate prediction of error, is only suitable for problems with small training sets, or few parameters.

Cross-validation was initially not used with neural networks because of the computational expense of running the system several times, which for MLPs trained with back-propagation, can take several days, was fairly unrealistic. Less computationally expensive methods have been developed (e.g. Generalised Cross Validation GCV (Wahba, 1990), Aikaike's Final Prediction Error (FPE) have been used to optimise the size of multi-layer networks).

Cross-validation is therefore useful for automatically finding the optimal model structures or parameter estimates for a given data set, as well as supporting the human designer in validation and interpretation of models. The model can start small and increase its size until the cross-validation results start to show too much variance in the test errors.

The reliability of the accuracy estimates achieved by methods such as cross-validation depends strongly on the adequacy of the training data. If the training data is present in sufficient quantity throughout the input space (complex parts of the model will need a related number of data to train and test the parameters in that area) this provides a good estimate of the predictive power of the model. The amount of training data needed is also related to the noise on the training data. As the noise level increases, the amount of data points needed to train and validate a particular model increases.

2.2.5 Organisational aspects in empirical modelling projects

The procedure for the successful development of a model is often a major undertaking, involving a variety of experts from various fields. The data must be acquired somehow, usually from an experiment carried out by an experienced operator. The experiment design should ideally be an interactive process, involving several steps, including information from earlier modelling attempts. Initial data processing will involve signal processing engineers, the goals of the modelling are set by a mixture of business and engineering constraints, and the validation of how well the goals were achieved involves every link of the chain. For most realistic problems there will also have to be many iterations towards the goal of an accurate, useful, reliable model, so the representation used must be promote co-operation between specialists with very different backgrounds, and be able to integrate different types of *a priori* knowledge either directly as model structure, or in the optimisation process as cost-functions and constraints.

2.3 Local Methods in Modelling

The previous section described the aspects of the modelling process which should be supported by a learning architecture. This thesis discusses variations on one main class of network, the *Basis Function Network (BF Net)* because of its suitability for modelling continuous nonlinear systems. The BF nets used in this thesis are basically local structures, which inherently involve modularisation in representation and allow the easy integration of ideas and structures from other modelling paradigms.

2.3.1 Basis Function Networks for modelling

The basic Basis Function Network described in equation (2.10) is shown in Figure 2.12. The output[4] y is a weighted (by parameters $\boldsymbol{\theta}_i$) linear combination of the activations of the many

[4]The models discussed in this report are all Multi-Input/Single Output models. The extension to multi-output systems is mathematically straightforward. The optimisation of the units' weights is unchanged, other

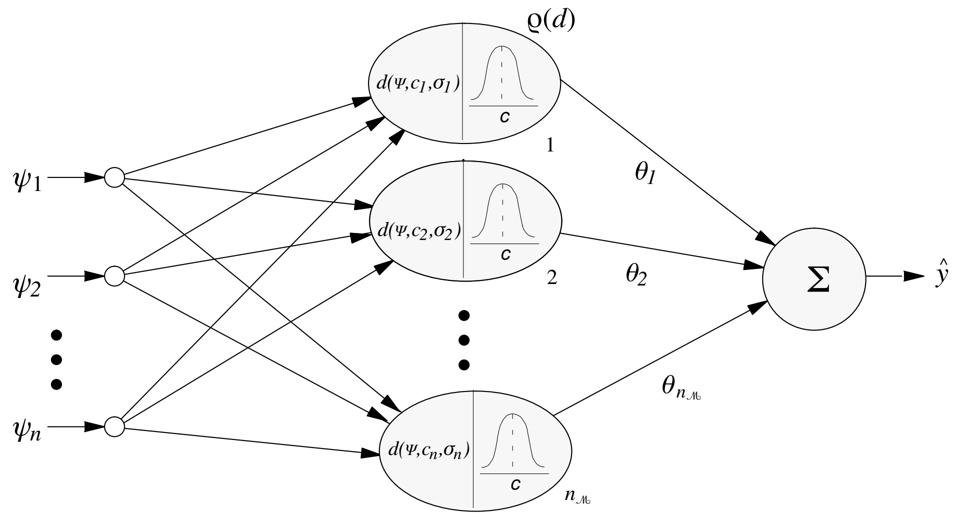


Figure 2.12: Radial Basis Function network

$(n_\mathcal{M})$ locally active non-linear *basis functions* $\rho_i(\cdot)$ which react to the input vector $\boldsymbol{\psi}$,

$$y = f(\boldsymbol{\psi}) = \sum_{i=1}^{n_\mathcal{M}} \theta_i \rho_i(\boldsymbol{\psi}). \tag{2.10}$$

The nonlinear basis functions basically map inputs into a higher dimensional space, where it is easier to learn the mapping to the outputs than from the original input space, so that a linear connection to the output suffices. The optimisation of the weights then becomes a linear process, meaning that the optimal solution can be found using the standard tools of linear optimisation theory, including a variety of powerful methods for coping with *ill-posed* problems (see Section 2.5.1), and methods for analysing the covariance of the parameter estimates (see Section 3.2.3), from which aspects such as experiment design criteria can be obtained.

Basis functions

Each unit's centre is a point in the input space, and the *receptive field* of the unit (the *support*, or volume of the input space to which it reacts) is defined by its distance metric $d(\boldsymbol{\psi}; \mathbf{c}, \sigma)$. The *basis* or *activation function* (similar to the *membership function* of a fuzzy set) of the unit is usually designed so that the activation monotonically decreases towards zero as the input point moves away from the unit's centre $(\mathbf{c}_i)$, e.g. B-Splines or Gaussian bells are common choices.

than that a matrix of output values is used instead of a vector. The optimisation of the model structure is obviously more difficult for multi-output problems, because the nonlinearity and complexity for the various output spaces will not always be in the same areas of the input space. The use of a single model structure with multiple outputs would mean that for the total model there are fewer parameters to optimise, which would suggest a lower variance than for a decomposed model. It may, however make more sense to decompose the problem into several single output problems, because each sub-problem will then have the required complexity and basis function locations for the complexity of the output in question, and will not produce an increase in variance in the other output variables, which happens in the multi-output case.

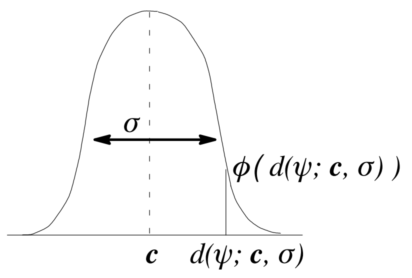


Figure 2.13: A typical locally active, smooth basis function

In Radial Basis Function (RBF) nets, the basis functions are composed of two elements. The distance metric $d(\psi; \mathbf{c}_i, \sigma_i)$ for basis function i, defined in equation (2.11), can scale and shape the spread of the basis function relative to its centre $\mathbf{c}_i$, depending on its width σ_i, and the basis function itself $\rho(\cdot)$, which takes the distance metric as its input

$$d(\psi; \mathbf{c}_i, \sigma_i) = \frac{(\psi - \mathbf{c}_i)^2}{\sigma_i^2}, \tag{2.11}$$

and can be generalised by using a matrix $\boldsymbol{\sigma}_i$ instead of a scalar, giving an ellipsoidal basis function (equation (2.12)). Training algorithms for such distance functions are given in Section 4.2.3.

$$d(\psi; \mathbf{c}_i, \sigma_i) = |\psi - \mathbf{c}_i|^T \boldsymbol{\sigma}_i{}^{-1} |\psi - \mathbf{c}_i| \tag{2.12}$$

There is a wide variety of possible basis functions, e.g. the Gaussian bells used in this thesis, as in equation (2.13), B-Splines, thin plate splines, linear functions etc.

$$\rho_i(\psi) = \rho(d(\psi; \mathbf{c}_i, \sigma_i)) = \exp(-d(\psi; \mathbf{c}_i, \sigma_i)), \tag{2.13}$$

(Carlin, 1992) compares the effectiveness of a variety of basis functions for modelling and control purposes. The basis functions used in this report will be assumed to have local properties[5], i.e. they are active in a limited area of the input space because of the improvement in transparency achieved.

If the units have localised receptive fields, *and a limited degree of overlap with their neighbours*, the unit's weights can be viewed as locally accurate piecewise constant models (in more complex networks, more general local models can be used, see Section 2.3.2), whose validity for a given input is indicated by their unit's own activation functions for a given input.

For modelling and control of continuously differentiable processes, the basis function should be smooth, and if the basis functions are to be local, they must decrease monotonically from a maximum at $\psi = \mathbf{c}$ (distance metric = 0) towards zero, according to the distance metric $d(\tilde{\phi}; \mathbf{c}, \sigma)$. This forces the influence of the local model associated with the basis function to decrease as the inputs move away from its centre (where the basis function's local model is the most accurate representation of the system).[6]

[5]Note that strictly speaking the Gaussian is not a local basis function, as it does not have compact support.
[6]Given relevant *a priori* knowledge, more complex basis functions can be used which do not adhere to the

Partition of unity

For modelling tasks the basis functions should form a *partition of unity* for the input space, i.e. at any point in the input space, the sum of all basis function activations should be 1. This is a necessary requirement for the network to be able to globally approximate systems as complex as the basis functions' local models, e.g. in the straightforward single weight case, so that constant areas of the input space can be modelled exactly. The partition of unity ensures that every point in the input space has an equal weighting, so that any variation in output over the input space is due only to the parameters $\boldsymbol{\theta}$ weighting the basis functions' activation. In many applications the network's basis functions are normalised to achieve the partition of unity, i.e.

$$\rho_k(\boldsymbol{\psi}) = \frac{\rho\left(d\left(\boldsymbol{\psi}, \mathbf{c}_k, \sigma_k\right)\right)}{\sum_{i=1}^{n_{\mathcal{M}}} \rho\left(d\left(\boldsymbol{\psi}, \mathbf{c}_i, \sigma_i\right)\right)} \tag{2.14}$$

where $\rho(\cdot)$ is the general *unnormalised* basis function, so that the *normalised* basis functions $\rho_k(\cdot)$ sum to unity,

$$\sum_{i=1}^{n_{\mathcal{M}}} \rho_i\left(d\left(\boldsymbol{\psi}, \mathbf{c}_i, \sigma_i\right)\right) = 1. \tag{2.15}$$

(Werntges, 1993) discusses the advantages of normalisation in RBF nets, promoting somewhat simplistically the advantages of a partition of unity produced by normalisation. Normalisation can be important for basis function nets, often making the model less sensitive to poor choice of basis functions, but it also has a number of side-effects which are discussed in detail in Section 3.3. These side-effects make the argument for or against the use of normalisation far more complicated than is often assumed.

Literature of Basis Function nets for modelling

Basis Function Networks and their equivalents have been used for function approximation and modelling in various forms for many years. The original Radial Basis Function Nets came from *Interpolation theory* and are described in (Powell, 1987), where a basis function is associated with each training point, as in (Specht, 1991). *Potential Functions* (Aizermann et al., 1964), *Kernels* (Wahba, 1992) and *Spline Models* (Wahba, 1990) are all similar structures. The literature of *local learning* methods in statistics is reviewed in (Atkeson, 1990). These methods store the training data and for a given input point form a locally weighted representation of the system from the related training points. *Smoothing methods* such as Gaussian Kernel methods, as described in (Hastie and Tibshirani, 1990) like other local averaging methods, suffer from the curse of dimensionality (Friedman, 1991), and are computationally expensive.

Aldus's CMAC ideas have a great deal of overlap with BF Nets with uniformly distributed local basis functions (Albus, 1975b, Lane et al., 1991, Brown and Harris, 1994). The close

features above, but which are specially relevant to a particular application, e.g. sinusoidal basis functions are a good choice if it is known that oscillatory components play an important role in the system being modelled.

relation of basis function nets to classes of *fuzzy logic systems* has also been discussed in (Jang and Sun, 1993), (Haas and Murray-Smith, 1993) and (Brown and Harris, 1994), where the similarity between membership functions and basis functions is pointed out. *Mixture Models* used in statistics are created by mixing a number of probability distributions, and have many similarities to RBF nets (Bishop, 1994) and (Xu et al., 1994).

Recently BF neural networks have received a growing amount of attention from the neural network community, starting with the early papers (Moody and Darken, 1989, Jones et al., 1989, Broomhead and Lowe, 1988). (Hlavácková and Neruda, 1993) gives a brief review of the use of RBF nets. (Poggio and Girosi, 1990) (Girosi et al., 1993) and (Mason and Parks, 1992) describe the networks within the mathematical framework of *Regularisation Theory* for function approximation. (Hutchinson, 1994) describes the use of RBF nets for financial time series modelling. (Park and Sandberg, 1991) and (Park and Sandberg, 1993) proved the universal approximation abilities of RBF nets.

Use of RBF nets for modelling and control purposes is described in (Barnes et al., 1991) (Sanner and Slotine, 1992) (Sbarbaro, 1992a) and (Pantaleón-Prieto et al., 1993). The methods were applied in (Röscheisen et al., 1992) to control a rolling mill. Early work related to this thesis can be found in (Murray-Smith et al., 1992) and (Neumerkel et al., 1993). RBF networks which use the local nature of the basis functions to give a local prediction of their own accuracy throughout the input space are examined in (Leonard et al., 1992) (this idea is extended to the local model nets in Section 3.2).

The *Gaussian Bar* nets (Hartman and Keeler, 1991, Kurcova, 1992) are also closely related, although less powerful. These are nets with 'semilocal' units formed by taking one-dimensional local basis functions and forming their tensor product to approximate a multi-variable function, as with tensor product spline models such as the ASMOD system (Kavli, 1992). The advantage of these units is that they can better cope with some classes of high dimensional problems, as they do not need to cover 'uninteresting' dimensions. A disadvantage of this style of network is that the representation does not cope well with processes where the nonlinearity depends on several variables, such as the mars1 benchmark used in Chapter 4.

Pao's *Functional Link Network* (Pao, 1992), and Billing's closely related *Extended Model Set* ideas (Billings and Chen, 1989), use links with a fixed non-linear function built in to expand the input vector, resulting in the production of extra 'higher-order' inputs. These are linearly weighted by the networks parameters. The well known *Polynomial methods* can also be viewed as Basis Function systems, as there is a single layer of nonlinear functions, and the parameter optimisation is a linear process. The problem here is to find the suitable model structure – i.e. which set of basis functions can approximate the system adequately. Some off-line structure identification algorithms are described in (Chen and Billings, 1994, Ivakhnenko, 1971). Holden describes a general framework for basis function nets, calling them *Phi-nets* in (Holden, 1994).

Are Basis Function nets too local?

An intrinsic feature of the Basis Function networks is the concept of 'locality'. In linear systems the data, optimisation and validation are all considered to be globally relevant, i.e. any results obtained are valid over the entire input space, whereas in nonlinear systems the process complexity varies throughout the input space. For nonlinear systems, however, (especially when multivariable) the problem can be simplified by partitioning the input space into multiple subspaces. This can involve a reduction of the problem's dimensionality by decomposing the problem, discarding irrelevant interactions, or of simply partitioning the input spaces into subspaces which are easier to handle – the traditional 'divide and conquer' strategy inherent to local modelling techniques.

The concept of 'locality' is obviously relative, depending on the complexity of the system, the availability of training data, the importance of the given area of the input space, and *a priori* knowledge of internal structures within the given system. The form of 'locality' utilised depends on the representation used; in decision trees, locality is introduced by partitioning the input space into hypercubes, in RBF nets locality is hyperspherical and in multi-layer perceptrons or Projection Pursuit nets locality is a projection of the input space. This locality can be used to make networks more transparent and computationally efficient, which can then make learning algorithms which utilise the locality more efficiently than alternative algorithms.

The problem with standard basis function networks is that the crudeness of the local approximation (weighted piecewise constant models), suffers like other local methods from the 'curse of dimensionality' (Bellman, 1961). This forces the system to use exponentially increasing numbers of basis function units to approximate a given system, as the input dimension increases. This leads to computational, transparency and robustness problems (the training data to train all of the units has to exist!). It is therefore important to be able to profit from the local nature of the basis functions while not having to have too many units[7]. This implies that the basis functions should be associated with more powerful representations than piecewise constant models, so that a smaller number of them could cover larger areas of the input space while achieving the desired modelling accuracy.

2.3.2 Local Model basis function nets

Linear models, although very restricted in their representational ability have proved to be very useful for a large range of problems. This is due to their simple representation, their easy interpretability, and their robustness to noisy or missing data. It makes sense therefore to include the ability to at least be able to form a linear model within the network, as in

$$\hat{y} = \hat{f}(\boldsymbol{\psi}) = \theta_1 + \sum_{i=2}^{n_\psi+1} \theta_i \psi_i + \sum_{i=n_\psi+2}^{n_{\mathcal{M}}+n_\psi+1} \theta_i \rho_i(\boldsymbol{\psi}), \tag{2.16}$$

[7]see (Lowe, 1994) for a discussion of further arguments against local basis functions.

where n_ψ is the dimension of the input vector $\boldsymbol{\psi}$. The basic architecture (Poggio and Girosi, 1990) is a simple improvement which is highly relevant for practical applications, e.g. used in (Hutchinson, 1994).

The ability to use linear (or other) models, can also be introduced in a more general way. Standard basis function networks can be generalised to allow not just a constant weight to be associated with each basis function, but a more general function of the inputs, so that the network can be described in the form

$$\hat{y} = \hat{f}(\boldsymbol{\psi}) = \sum_{i=1}^{n_{\mathcal{M}}} \hat{f}_i(\boldsymbol{\psi})\rho_i(\tilde{\boldsymbol{\phi}}), \tag{2.17}$$

where $\tilde{\boldsymbol{\phi}}$ defines the *operating point* of the system. This is a vector which can be defined on a lower dimensional subspace of the input space which is covered by the basis functions. These can be seen as *scheduling* or *gating* functions for the local models which are defined on the full input space.

The basis, or *model validity functions* used in this thesis are radial, i.e. they use a *distance metric* $d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i)$ which measures the distance of the current operating point $\boldsymbol{\psi}$ from the basis function's centre $\mathbf{c}_i$, relative to the width variable σ_i, as in equation (2.12). They are also *normalised*, so that they sum to unity, as in equation (2.14). See Figure 2.14 for a simple representation of operating regimes in a two dimensional operating space. The overlapping operating regimes allow the basis functions to smooth the transfer from one region of the model structure to the next.

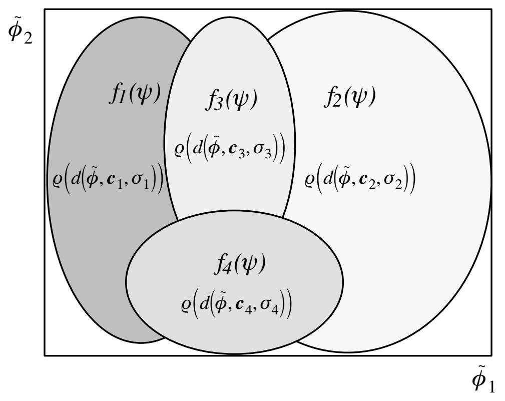

$\tilde{\phi}_2$
$f_1(\psi)$
$\varrho(d(\tilde{\phi}, c_1, \sigma_1))$
$f_3(\psi)$
$\varrho(d(\tilde{\phi}, c_3, \sigma_3))$
$f_2(\psi)$
$\varrho(d(\tilde{\phi}, c_2, \sigma_2))$
$f_4(\psi)$
$\varrho(d(\tilde{\phi}, c_4, \sigma_4))$
$\tilde{\phi}_1$

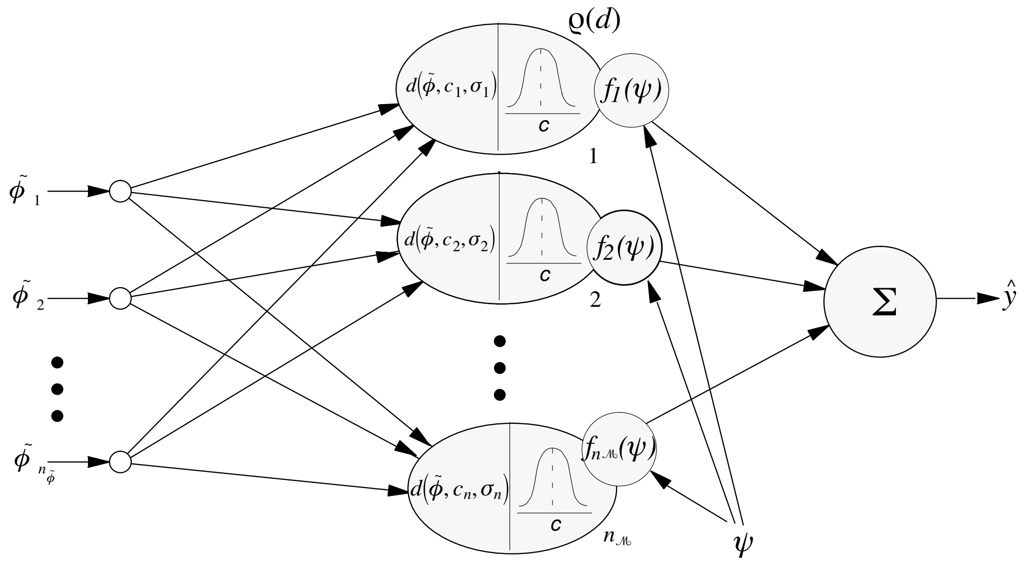
Figure 2.14: Local Model Operating Regimes. Each local model is associated with an operating regime. These regimes overlap, and the gradual decay of 'validity' provides interpolation between models.

This means that the structure has the advantages inherent to the local nature of the basis functions while, because of the more powerful local models associated with the basis functions, not requiring as many basis functions as before to achieve the desired accuracy. The improvement is more significant in higher dimensional problems. This generalisation of the

BF network to the *Local Model Network* described in equation (2.17), where the weights have been generalised to allow not just a constant weight to be associated with each parameter, but to have a function of the inputs weighted by the relevant basis function, has been applied by a number of authors. The $n_{\mathcal{M}}$ local models used are represented by general functions of the inputs $f_i(\boldsymbol{\psi})$, but in many cases simple linear models are chosen

$$\mathcal{M}_i(\boldsymbol{\theta}) = \hat{f}_i(\boldsymbol{\psi}) = \boldsymbol{\theta}_i^T\,[1\ \boldsymbol{\psi}]\,. \tag{2.18}$$

The network form of equation (2.17) is shown in Figure 2.15. The trained network structure can be viewed as a decomposition of the complex, nonlinear system into a set of locally active sub-models, which are then smoothly integrated by their associated basis functions.



$\varrho(d)$
$d(\tilde{\phi}, c_1, \sigma_1)$
$f_1(\psi)$
c
1
$d(\tilde{\phi}, c_2, \sigma_2)$
$f_2(\psi)$
c
2
$d(\tilde{\phi}, c_n, \sigma_n)$
$f_{n_{\mathcal{M}}}(\psi)$
c
$n_{\mathcal{M}}$
$\tilde{\phi}_1$
$\tilde{\phi}_2$
$\tilde{\phi}_{n_{\tilde{\phi}}}$
Σ
$\hat{y}$
ψ

Figure 2.15: Local Model Basis Function network

To illustrate the workings of a local model network, a one dimensional function is mapped using local models in Figure 2.16. The top plot shows the target function and the model's approximation, while the basis functions and associated local models are shown below.

Literature of local model methods in learning and modelling

The representational ability of the normal Basis Function (BF) net can be extended to a generalised form of BF network, where the basis functions are used to weight other functions of the inputs as opposed to straightforward weights. This was suggested in (Jones et al., 1989), followed up by (Stokbro et al., 1990) and (Barnes et al., 1991). The Adaptive Expert networks in (Jacobs et al., 1991) are essentially local model systems, where the local models are called *expert networks* and the integration of the various *experts* is made by *gating networks*. These were developed into hierarchical models in (Jordan and Jacobs, 1991, Jordan and Jacobs, 1993).

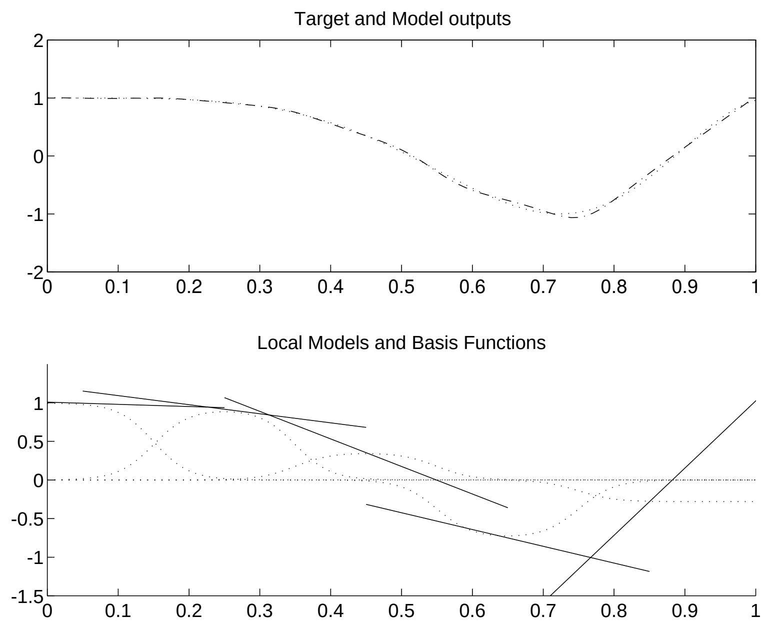


Figure 2.16: An example of local models representing a one dimensional function

The advantages of local representations are discussed in (Bottou and Vapnik, 1992), where they suggest that a proper compromise between local and global methods will usually prove most effective as varying levels of complexity are required throughout the input space, although they claim that the 'local capacity' should match the data density, which is not necessarily true, as this would not place units where the system is complex, but rather where there was most data. The more general goal of allocating local capacity is that the learning system should match the *'local complexity'* of the target system.

The idea of using locally accurate models is also described in the statistical literature in (Cleveland et al., 1988), where local linear or quadratic models are weighted by smoothing functions. (Priestley, 1988) describes *State Dependent Models* for non-linear time series which are basically linear models where the parameters depend on the operating vector $\tilde{\boldsymbol{\phi}}$ (corresponding to the 'state' in Priestley's terminology)

$$\hat{y}(t) = \boldsymbol{\psi}^T(t-1)\sum_{i=1}^{n_{\mathcal{M}}}\theta_i\rho_i(\tilde{\boldsymbol{\phi}}). \tag{2.19}$$

Local Model nets could be viewed as a finite parameterisation of the state-dependent model. Tong's *Smooth Threshold Autoregressive* (STAR) models (Tong, 1990) are also structurally equivalent to local model nets. In neither Priestly's or Tong's case, however, is much detail given about how to find the smooth weighting functions. (Billings and Voon, 1987) also uses a number of linear models to approximate a nonlinear system, but does not have smooth interpolation between basis functions.

Local model systems for diagnosis, modelling and control of dynamic systems have been applied extensively by Johansen and Foss in Trondheim, e.g. (Johansen and Foss, 1992c), (Johansen and Foss, 1992b) and (Johansen and Foss, 1993). A development of the ideas to a state-space implementation of local models is described in (Johansen and Foss, 1993).

Some *fuzzy logic* systems can also be viewed as Local Model networks, e.g. the methods used in (Takagi and Sugeno, 1985) are effectively overlapping piecewise linear models, with the interpolation between models provided by the membership functions. Similar applications are reported in (Sugeno and Kang, 1988, Foss and Johansen, 1993), (Wang, 1994) and (Harris et al., 1993). In (Haas and Murray-Smith, 1993) we discuss the similarity between fuzzy and basis function systems in more detail. (Back and Tsoi, 1991) describes the use of dynamic models as nodes in Multi-Layer Perceptrons – this could be seen as an MLP implementation of Local Model methods.

(Skeppstedt et al., 1992) describes the use of local dynamic models for modelling and control purposes, but with hard transfers from one model regime to the next. (Pottmann et al., 1993) describes a multi-model approach where although the local models overlap there is still a sharp transition from one model to the next, and to minimise model switching a heuristic criterion is introduced which only allows switching after three consecutive steps in the direction of the new model.

2.4 Hierarchical Approaches to Learning Models

We have discussed the use of locality in basis function networks for the limitation of complexity, but how can we decide on the suitable level of locality for any given area of the input space? How can we reduce the effect of high dimensionality by only concentrating on the areas of interest? Can we produce efficient training algorithms which take points outside a given local model into account, while not reducing the system to a global optimisation technique?

A common technique for the control of complexity, found in nature, society and technical systems, is *hierarchy* (Mesarovic et al., 1970). In the flat local model networks used so far, the structure identification phase tends to be computationally expensive, because any alteration to one unit affects many others. The effect of normalisation also alters the local properties of the basis functions, sometimes leading to unexpected results, as described in Section 3.3. Hierarchical methods offer, due to their structure, the ability to more effectively hide local complexity, in the traditional 'divide and conquer' manner, making the learning process more efficient and robust. The goal is truly hierarchical learning, where a network can grow to fit the data, and as the representation of the model improves decisions made earlier in the learning process can be reevaluated, leading to gradual changes to the higher levels. The local model framework is well suited for the creation of such hierarchies, as the local models in a given network can be further local model networks producing a hierarchy of local model nets. This is the Learning Hierarchy of Models (LHM) structure described in Chapter 5.

2.4.1 Hierarchical learning methods

The use of hierarchy in learning algorithms breaks down into two general camps. The decision tree methods started in the 1970's with *k-d trees* (Bently, 1975), *Classification And Regression Trees* (CART) (Breiman et al., 1984), and *ID3* and *C4.5* (Quinlan, 1993), and involve hierarchies of sharp partitions, dividing the input space into ever smaller areas. (Isaksson et al., 1991) and (Strömberg et al., 1991) describe the use of such trees with dynamic models in the leaf nodes to model nonlinear dynamic systems. Because of the sharp partitions, however, these methods are poorly suited to modelling continuous systems. An alternative approach is the

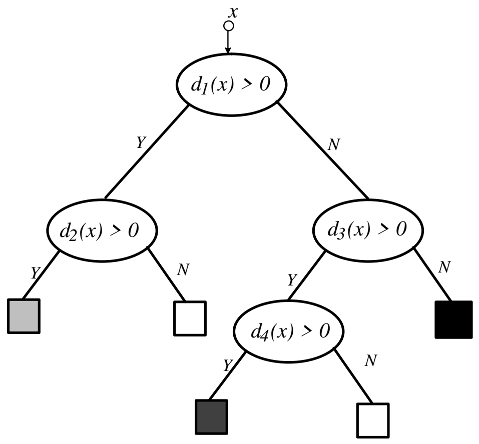


Figure 2.17: Decision tree structure

use of *soft splits*, where the input space is no longer sharply partitioned, but there is a gradual transfer from one local model space to the other, allowing smooth interpolation between the models. This also means that a point in the input space can activate models in several leaves of the tree, with a differing level of membership to each.

Examples of this type of structure include *Basis-Function Trees* (Sanger, 1991b) and the related paper (Sanger, 1991a). The *Hierarchical Mixtures of Experts* (HME) structure, is a hierarchical structure (Jordan and Jacobs, 1993) trained using Expectation Maximisation techniques (EM). Friedman used spline-based techniques to extend the CART ideas to soft splits for his *Multiple Adaptive Regression Splines* (MARS) algorithm (Friedman, 1991). Quinlan also started to work with continuous systems modelling using *Model Trees* (Quinlan, 1992). Links between wavelets and hierarchical networks (Bakshi and Stephanopoulos, 1993) have also been investigated. Banan describes a constructive hierarchical method, with local linear models, which trains many times, and partitions the input space *randomly* in the areas producing errors. The 'average' network produced by the random splits is then the result of training (Banan and Hjelmstad, 1992). (Omohundro, 1991) describes Bump- and *Balltrees* which have linear classifiers in the leaves of the tree. The first work from this thesis with hierarchical networks was the development of *Fractal Radial Basis Function Nets*, described in (Murray-Smith, 1992). The extension of this method is integrated into the local model paradigm in Chapter 5.

2.5 Learning in Local Model Basis Function Networks

The learning task for Local Model Networks is to try to adapt their structure and parameters to minimise a cost functional related to the deviation of the model from the target system. The optimisation of the parameters[8] is a linear process and relatively straightforward, while the optimisation of the number of basis functions and their position and determining a suitable level of 'locality' is a difficult non-convex problem, where *ad hoc* methods of reducing the complexity play an important role.

The description of the learning process is therefore split into the two highly interdependent stages of *structure identification* (Section 2.5.4 describes methods for introducing *a priori* knowledge into the model structure, Section 2.5.3 reviews the literature of structure identification methods) and *parameter estimation* (Section 2.5.1), which are repeated until the desired structure and parameters are found.

2.5.1 Parameter estimation in Local Model Nets

The assumption made during the parameter estimation stage is that the model structure (i.e. the basis functions and local model structures) already exists and remains fixed during the estimation phase. This thesis only deals with time-invariant processes, so the methods used are all *off-line* methods, where the entire training set is assumed to be available simultaneously for the estimation phase, as opposed to on-line or recursive methods, which assume a constant stream of new information[9]. The assumption made here is that the local models are linear in the parameters, as in equation (2.18).

The problem of parameter identification within such a framework is reasonably well understood, with a variety of efficient optimisation algorithms existing to solve the problem of optimising the parameters $\boldsymbol{\theta}$ of local models $\hat{f}_i(\cdot)$ in equation (2.17) to minimise the cost functional $J(\boldsymbol{\theta}, \mathcal{M}, \mathcal{D})$ for a given local model structure $\mathcal{M}$, where $\mathcal{M} = (\mathbf{c}, \boldsymbol{\sigma}, n_{\mathcal{M}}, \mathcal{M}_{1..n_{\mathcal{M}}})$ (i.e. the basis functions' centre locations and basis function sizes, as well as local model types) and training set $\mathcal{D} = (\psi(t-1), y(t)), t = 1..N$. Parameter optimisation for a given model structure finds the optimal cost J^*

$$J^*(\mathcal{M}, \mathcal{D}) = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathcal{M}, \mathcal{D}). \tag{2.20}$$

[8] In this work the term *parameters* refers to the parameters of the local models and does not include the basis function parameters, the centres and widths, which are deemed to define the *model structure*.

[9] Section 4.3 discusses methods for iteratively extending the training set used, but for any given estimation stage the optimisation is seen as a batch process.

Weighted Least Squares estimation

In many learning situations the relative importance of the training data varies throughout the input space, either because the system spends most of its time in one particular operating regime, or because a particular aspect of the system is more interesting than others, and it is also common to have varying measurement accuracy in different areas of the input space. It is therefore important to be able to weight points in the training set to have more or less significance. The global criterion for estimation of the parameters of the model in equation (2.17) is then the *weighted least squares* cost functional,

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \alpha(\boldsymbol{\psi}_i)(y_i - \hat{y}_i)^2, \tag{2.21}$$

and the vector containing all the estimated model parameters is that which minimises J. In equation (2.21) $\alpha(\boldsymbol{\psi}_i)$ are observation weights which can be attached to each measurement. For the case where the measurement noise estimate for each training point is available, this cost functional is the chi-squared function (where $\alpha(\boldsymbol{\psi}_i) = \frac{1}{\sigma(\boldsymbol{\psi}_i)}$, where $\sigma(\boldsymbol{\psi}_i)$ is the measurement error (standard deviation) of the ith training vector). The model obtained using this functional is known as a *Markov Estimator* or a *Best Linear Unbiased Estimator (BLUE)*.

Regularisation methods for complexity penalisation

The parameter optimisation for local model nets is an *ill-posed problem* as described in (Tikhonov and Arsenin, 1977). This is because there is insufficient data in the training set to reconstruct the input-output mapping uniquely, the data is usually corrupted by noise, meaning that a unique solution is impossible, and the continuity conditions are violated. To make the problem *well-posed* it is necessary to make assumptions about the smoothness of the underlying process being modelled, and the training set must have redundancy in an information-theoretic sense.

A variety of methods can be used to regularise the optimisation problem, to reduce the variance of the solution. Many neural network learning algorithms have implicitly (often unplanned!) had a regularisation effect, in that they have not found the 'optimal' (in the least squares sense) solution to the posed optimisation problem. Methods such as weight decay, stopping learning early (Sjöberg and Ljung, 1992), network pruning, learning with noise (Bishop, 1994) are all examples of *ad hoc* attempts to produce a regularisation effect. The classic regularisation method as defined in the *regularisation theory* proposed by (Tikhonov and Arsenin, 1977) is to extend the simple quadratic error cost functional to become a cost-complexity operator,

$$J(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \alpha(\boldsymbol{\psi}_i)\left((y_i - \hat{y}_i)^2 + \lambda R(\hat{f}(\boldsymbol{\psi}_i, \boldsymbol{\theta}))\right), \tag{2.22}$$

including penalising nonnegative functional $R(\hat{y})$ which includes *a priori* information such as smoothness constraints which makes the optimisation problem well-posed. This forces the optimisation to find a 'smoother' solution (λ is a small weighting, which defines the relative cost of the complexity compared to accuracy), which is likely to improve the generalisation of the network on new examples, at the cost of a slightly worse performance on the training data e.g. (Bishop, 1991) (Bishop, 1994) (Poggio and Girosi, 1990) (Girosi et al., 1993). These are, unless analytical solutions exist, too computationally expensive for problems of more than a few dimensions. More practical regularisation methods are described in Chapter 3. (Sjöberg et al., 1993) describes the use of regularisation methods in system identification, and the earlier paper (Sjöberg and Ljung, 1992) links the regularisation work to stopping training early, and discusses the number of important parameters in multi-layer perceptrons.

A priori knowledge of physical constraints on models can also be used to improve generalisation, as in (Kramer et al., 1992), and by (Röscheisen et al., 1992) who demonstrate the use of *a priori* models in training an RBF model of a rolling mill.

Using Singular Value Decomposition to estimate the local model parameters

The optimisation of the weights in RBF and Local Model Networks is theoretically a straightforward application of Linear Regression techniques, and as the optimisation problem is a linear one, the 'optimal' solution should always be found (assuming uncorrelated zero-mean noise). The regression problem can be viewed as finding the local model parameters $\boldsymbol{\theta}$ which satisfy the equation,

$$\mathbf{Y} = \boldsymbol{\Phi}\boldsymbol{\theta} \tag{2.23}$$

where $\boldsymbol{\Phi}$ is the design matrix, where the rows are defined by

$$\boldsymbol{\phi}_i = \left[\rho_1(\tilde{\boldsymbol{\phi}}_i)[1\ \boldsymbol{\psi}_{i_1}\ldots\boldsymbol{\psi}_{i_{n_\psi}}]\ldots\rho_{n_\mathcal{M}}(\tilde{\boldsymbol{\phi}}_i)[1\ \boldsymbol{\psi}_{i_1}\ldots\boldsymbol{\psi}_{i_{n_\psi}}]\right], \tag{2.24}$$

so that the design matrix $\boldsymbol{\Phi}$, and vector of output measurements $\mathbf{Y}$ are

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}_1 \\ \cdot \\ \cdot \\ \boldsymbol{\phi}_N \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \cdot \\ \cdot \\ y_N \end{pmatrix} \tag{2.25}$$

The task of matrix inversion is important for the optimisation process. In practical situations, however, the straightforward inversion of an information matrix is of little use, as the matrices are not square, and even if they are, the poor condition or singularity of the matrix in question makes inversion impossible. To avoid these problems, the Moore-Penrose pseudoinverse of $\boldsymbol{\Phi}$, $\boldsymbol{\Phi}^+$ is used to estimate the weights.

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\Phi}^+\mathbf{Y} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{Y} \tag{2.26}$$

so for weighted least squares as described in equation (2.21), with $\mathbf{Q}$ a diagonal matrix with $Q_{ii} = \alpha(x_i)$, the optimal weights are:

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{\Phi}^T\mathbf{Q}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{Q}\mathbf{Y} \tag{2.27}$$

The algorithm used in this work[10], as in many other papers, to calculate the pseudoinverse is *Singular Value Decomposition* (SVD) of a matrix of observed input data. The SVD algorithm decomposes any $N \times n_\phi$ matrix $\mathbf{\Phi}$ to matrices $\mathbf{U}$ ($N \times n_\phi$ column orthogonal), $\mathbf{\Sigma}$ ($n_\phi \times n_\phi$ diagonal) and $\mathbf{V}$ ($n_\phi \times n_\phi$ orthogonal), such that $\mathbf{\Phi} = \mathbf{USV}^T$. Due to the orthogonality of the matrices $\mathbf{U}$ and $\mathbf{V}$, $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = 1$. $\mathbf{V}$ is the matrix containing the eigenvectors of $\mathbf{\Phi}^T\mathbf{\Phi}$. $\mathbf{U}$ is made up of the eigenvectors of $\mathbf{\Phi}\mathbf{\Phi}^T$. The associated eigenvalues (σ_i) are the same in both cases, and are the squares of the singular values (s_i), i.e. $\sigma_i = {s_i}^2$ As $\mathbf{U}$ and $\mathbf{V}$ are orthogonal, their inverses are equal to their transposes. $\mathbf{\Sigma}$ is diagonal, so its inverse is the diagonal matrix containing the reciprocals of its diagonal elements (the singular values). The advantage of this decomposition[11] is that the inverse of $\mathbf{\Phi}$ is now trivial to compute, giving the pseudoinverse:

$$\mathbf{\Phi}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T \tag{2.28}$$

where

$$\mathbf{\Sigma} = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & s_{n_\phi} \end{pmatrix}. \tag{2.29}$$

The method is robust because it can, within limits, cope with singular or poorly conditioned matrices. The *condition number* of a matrix gives an indication of the rank of the matrix.[12] If this is infinite the matrix is singular, and if the reciprocal of the condition number approaches the machine precision, the matrix is said to be ill-conditioned. In such cases the singular values are so small that the result is corrupted by the round off effects caused by finite accuracy arithmetic. Their corresponding columns in $\mathbf{V}$ are linear combinations of $\mathbf{x}$'s which are insensitive to the data. The $\frac{1}{s}$ elements, where s is less than a preset tolerance are zeroed, reducing the number of free parameters in the fit. Once the singular values have been zeroed, the parameters $\boldsymbol{\theta}$ solving the regression problem in equation (2.23) can be calculated,

$$\hat{\boldsymbol{\theta}} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{Y}. \tag{2.30}$$

This method of optimising the parameters is not without its disadvantages, however, which are described in Section 3.1.1. For more details on SVD see the general treatment in books such as (Golub and van Loan, 1989), or (Press et al., 1988). The use of the method in system identification applications is described in (Söderström and Stoica, 1989). The review article (van der Veen et al., 1993), and papers on the general application of the method in (Deprettre, 1988) and (Vaccaro, 1991) give further background.

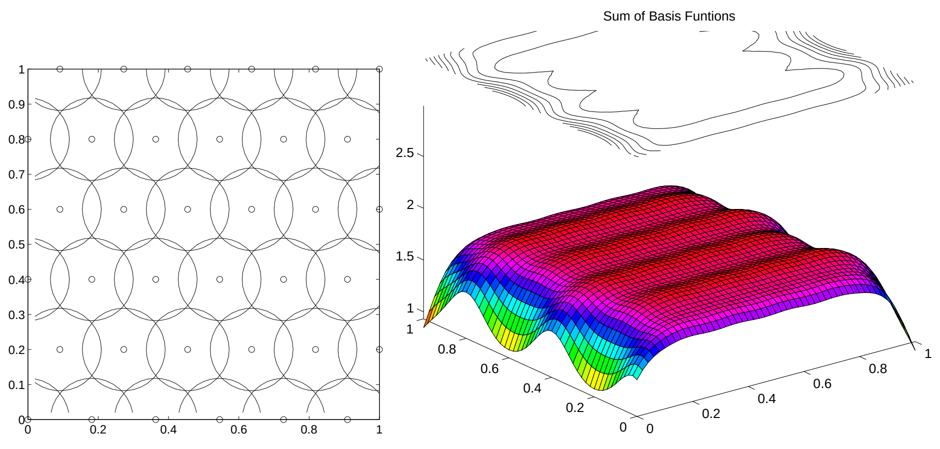
[10]The implementation of SVD used in this thesis is the MATLAB function svd().

[11]To better understand the behaviour of the original transformation $\mathbf{Y} = \mathbf{\Phi}\boldsymbol{\theta}$, the SVD of $\mathbf{\Phi}$ is a rotation of $\boldsymbol{\theta}$ in n_ϕ-space by $\mathbf{V}^T$, the components are then scaled by $\mathbf{S}$ and then rotated again by $\mathbf{U}$ to give $\mathbf{Y}$. If a vector $\mathbf{Y}$ lies in the range of $\mathbf{\Phi}$, there is a solution to $\boldsymbol{\theta}$. The solution is actually a set of solutions, as any vector in the nullspace can be added to $\boldsymbol{\theta}$ in any linear combination. SVD, however, finds the solution $\boldsymbol{\theta}$ with the smallest magnitude (see (Press et al., 1988) for details).

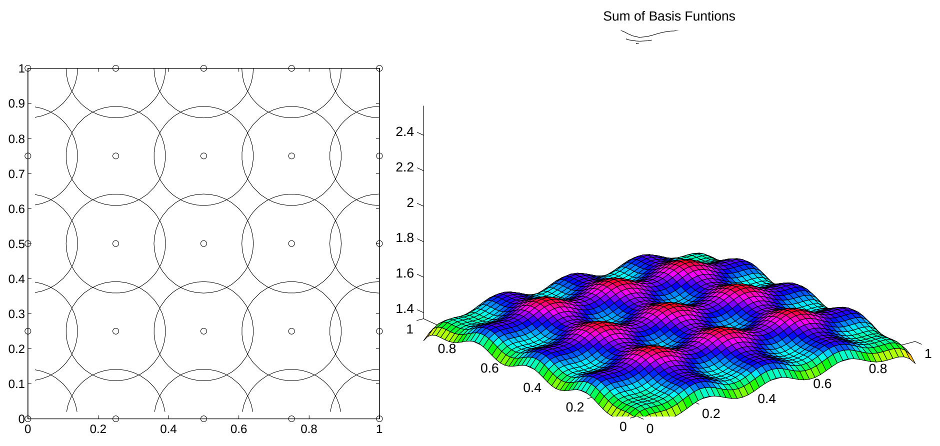
[12]See Section 3.1.1 for more details.

2.5.2 Uniformly distributed basis functions

The simplest method of determining the position and widths of the basis functions in a local model net is to fit a regular mesh or lattice of basis functions over the input space, as shown in Figure 2.18. This also allows the use of computationally simple methods to find active units, as



(a) Hexagonally distributed BF's (b) Sum of Hexagonally distributed BF's



(c) Grid-like distribution of BF's (d) Sum of grid-like distributed BF's

Figure 2.18: A lattice style distribution of basis functions in square and hexagonal forms

the net is effectively an interpolating memory. The total number of units required will rise – as the 'curse of dimensionality' would have us expect – exponentially with the input dimension. Many non-linear systems in the real world, however, have smooth nonlinearities which can be represented by relatively few units (this becomes even more relevant when the individual units are associated with more powerful local models). Also, as the system being modelled is often only active or of interest in a small region of the input space, a further saving of redundancy can be found by only placing basis functions in regions where the system operates. This demands more flexibility in the model structures than is possible with mesh-like networks, and methods for optimising the flexible structures.

2.5.3 Structure identification

The use of existing *a priori* knowledge, as described in Section 2.5.4, to define the model structure is important, but as many problems are not well enough understood for the model structure to be fully specified in advance, it will often be necessary to adapt the structure for a given problem, based on information in the training data. The optimisation of the network structure $\mathcal{M}$ is, however, a difficult non-convex optimisation problem, and is probably the most important area of research for basis function networks, if they are to be applied to demanding modelling problems where little is known about the model structure in advance.

The goal of the structure identification procedure is to provide a problem-adaptive learning scheme which automatically relates the density of basis functions and the size of their receptive fields to the local complexity and importance of the system being modelled. The desirable features of a structure identification algorithm are:

- *Consistency* – as the number of training points increases the algorithm should produce models which approximate the real process more accurately.
- *Parsimony* – the model structure produced by the algorithm should be the simplest possible which can represent the process to the required accuracy.
- *Robustness* – the model structures produced should be as robust as possible with regards to noisy data or missing data.
- *Interpretability* – the model structure produced should ideally be as interpretable as possible, given the available data, local models and basis functions.

The aim is therefore to find a model structure $\mathcal{M}$ which allows the network to best minimise the given cost function in a robust manner, taking the above points into consideration. Minimising $J^*(\mathcal{M}, \mathcal{D})$, from equation (2.20), over the possible model structures leads to the 'super-optimal' cost, using *a priori* knowledge about the process structure $\mathcal{K}_\mathcal{S}$,

$$J^{**}(\mathcal{D}, \mathcal{K}_\mathcal{S}) = \min_{\mathcal{M}} J^*(\mathcal{M}, \mathcal{D}). \tag{2.31}$$

The robustness is an important aspect, as constructive structure identification algorithms can obviously be very powerful, enabling the network to represent the training data very accurately by using a large number of parameters, but usually then leading to a high variance. The choice of model structure plays a major role in the *bias-variance trade-off* (see (Geman et al., 1992) for details about the trade-off), and this should be reflected in the cost functions $J$and J^* (from equation (2.20)) in the form of regularisation terms for J and terms which penalise over-parameterisation in the structure functional J^*.

Algorithms for structure identification from data should take into account the complexity of the target mapping, the representational ability of the local models associated with the basis functions, and the availability of data. This is a general non-convex optimisation problem, and in practice it is not possible to guarantee a general method which will provide an optimal solution to the problem for all possible learning tasks. The methods described here are optimisation techniques implicitly suited to the basis function optimisation problem.

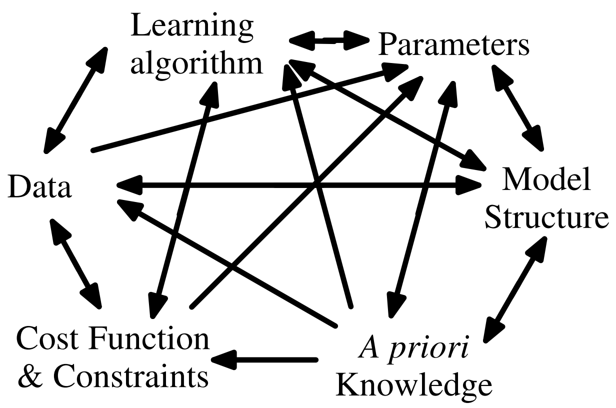


Figure 2.19: The structure learning process involves a number of complex interactions.

Structure through parameterised optimisation

A gradient descent optimisation technique for moving the centres and adapting the widths is described in (Poggio and Girosi, 1990). This is, however, reported to be a slow and unreliable technique. There is no guarantee of convergence to a global minimum and there is therefore likely to lead to variance in performance between runs on similar data. This method was applied in (Röscheisen et al., 1992) and also in (Hutchinson, 1994), where some practical guidelines for clustering are given.

Clustering basis functions

RBF researchers used methods where a fixed number of basis functions was assumed, and the structure identification task was seen as the optimisation of the centres and widths. In the papers (Moody and Darken, 1989, Sbarbaro, 1992b) clustering algorithms such as self-organising maps or k-means clustering were used to place the centres. A disadvantage of such algorithms is that they do not relate the location of the basis functions to the complexity of the function being mapped, only to the location of data in the input space. Pantaleón-Prieto *et al* (Pantaleón-Prieto et al., 1993) use a clustering routine where only the nearest unit to a given input is adapted, in order to reduce computation.

Placing centres on training data points

The disadvantage of the techniques described above is that the user must still define how many units the network should have before learning starts, but as the complexity of the system is usually not fully understood, the optimal number of units is also unknown. Some researchers developed methods which estimate the number of units from the position of the training data in the input space. Specht describes a method which has a basis function centred on every example in the training set (Specht, 1991). This is a simple technique, but one which scales up very poorly, and is not particularly robust when faced with noisy or sparse data. More promising methods use the redundancy in the training set to reduce the number of units needed to learn the desired training data. A hierarchical clustering technique based on a binary tree approach to recursively partition the input space is used in (Stokbro et al., 1990). The resulting network is a single layer net, only the partitioning process is hierarchical. The partitioning is not related to the local complexity of the system, but simply to the presence of data – a drawback of other similar algorithms. (Raipala and Koivo, 1992) describe a similar simple method for constructing a network, which is to insert a new unit whenever an input occurs which is not near the centre of any of the units' receptive fields. This is repeated in (Roberts and Tarassenko, 1994). The algorithms we used in (Murray-Smith et al., 1992) and (Neumerkel et al., 1993) can be seen as extending this type of technique by including the system complexity in the distance metric for the clustering process.

Iterative constructive techniques for gradual approximation

Another option is to start off with a simple model, to estimate its parameters, determine where the representation is still unsatisfactory and to dynamically add new models to the network. This leads to a sequence of model structures $\mathcal{M}_1 \rightarrow \mathcal{M}_2 \rightarrow \ldots \rightarrow \mathcal{M}_{n_\mathcal{M}}$, where $\mathcal{M}_i \rightarrow \mathcal{M}_{i+1}$ indicates an increase in the representational ability (more degrees of freedom) in the model structure followed by a parameter identification and confidence estimation stage. Constructive techniques which gradually enhance the model representation in this manner have a number of advantages. They automate the learning process by letting the network grow to fit the complexity of the target system, but they do this robustly, by forcing growth to be guided by the availability of data and the complexity of the local models. This automatically determines the size of the network needed to approximate the function adequately, while preventing overfitting. Two such constructive algorithms are described in Chapters 4 and 5, coming from the work published in (Murray-Smith and Gollee, 1994) and (Murray-Smith, 1992).

Chen *et al* (Chen et al., 1991) used orthogonal least squares for the clustering task. Their algorithm is a constructive one, which uses every training point as a candidate centre. Each time a unit is added to the network, it chooses the best candidate centre by attempting to minimise the variance in the model output due to the network parameters. The *Resource Allocation Net* described in (Platt, 1991) is a constructive algorithm, where when a pattern is presented which causes an error larger than a given threshold a new unit would be added at that point. Wynne-Jones suggests a constructive method where existing units in the network are split into two. The *Hierarchical Self-Organising Learning* (HSOL) algorithm, a hierarchical strategy for the construction of single layered basis function networks for classification is described in (Lee and Kil, 1991). Units are added to the network in a coarse to fine strategy. (Carlin, 1992) applied HSOL to modelling problems and similar coarse-to-fine ideas have been used for spline-based modelling applications (e.g. ASMOD in (Kavli, 1992)). (Fritzke, 1994) describes a constructive method based on a self-organising map framework. The LSA algorithm which splits the input space orthogonally to the axes of the input space is described in (Johansen and Foss, 1994b). The spline-based MARS algorithm (Friedman, 1991) mentioned earlier is also a gradual constructive method.[13]

2.5.4 Pre-structuring the local model net

The straightforward local model network, where the local models are simple linear models, can be viewed as a general structure which is well suited for use in modelling dynamic systems. A major advantage of the local model nets is, however, their ability to allow the introduction of *a priori* knowledge to define the model structure for a particular problem. This leads to more interpretable models which can be more reliably identified from a limited amount of observed data.

Incorporating local models based on *a priori* knowledge

The most general form of information is the expected order of the system, and the form of model to be identified (e.g. simple linear ARX models etc). If more knowledge is available, the local models could be physically oriented models, possibly with only a subset of their variables to be identified, thus allowing the engineer to easily create *grey-box* models. A generalised form would allow the designer to specify a pool of feasible local models, which could be locally tested for suitability in the various operating regimes defined by the basis functions.

In many cases, there will not be sufficient data to train the model throughout the input space. This is especially true in areas outside normal desired operation, where the model may have

[13]The *Model Merging algorithm* described in (Omohundro, 1991) attacks the problem in a different way, using a fine-to-coarse learning algorithm, where each training point is initially viewed as a model, and increasingly global models are created by merging the existing models. This has the disadvantage of being more computationally intensive than the top down methods, and will tend to overtrain where coarse-to-fine methods tend to over-generalise. Other workers have produced constructive algorithms for a variety of network types, e.g. *Cascade-correlation* (Fahlmann and Lebiere, 1990) and GAL (Alpaydin, 1991), but these structures lose the locality advantages of the basis function networks, and cannot easily introduce *a priori* knowledge, unlike local model nets.

to be very robust, and well understood. These situations can be covered by fixing *a priori* models in the given areas, and applying learning techniques only where the data is available and reliable.

Incorporating *a priori* knowledge of non-linearity into the basis functions

Locality of representation provides advantages for learning efficiency, generalisation and transparency. It is, however, very difficult to automatically find the 'correct' level of locality for a given subspace of an arbitrary problem. The problems of dimensionality can also be reduced in many systems with a large number of inputs, as there are often combinations of input dimensions which are of no interest, or which are additively or linearly related. The problem can then be decomposed, if the user already has *a priori* knowledge about the system being modelled, thus allowing the user to treat the system as an additive combination of lower dimensional sub-models. (The use of such physically based knowledge makes on-line adaptation of the system's parameters much more feasible). The statisticians have developed the theory of additive modelling techniques, e.g. (Hastie and Tibshirani, 1990, Friedman, 1991) to support such decompositions. (Hrycej, 1992) also describes similar methods for the modularisation of neural networks. Also, because of the strong links between Fuzzy membership functions and Basis Functions (see our review in (Haas and Murray-Smith, 1993), or the books (Harris et al., 1993, Brown and Harris, 1994)), the *a priori* knowledge of how best to decompose the problem could be expressed as linguistic rules with accompanying basis functions. (Bridgett et al., 1994) analyses the functionality of the MARS and ASMOD algorithms and relates them to fuzzy systems.

High dimensional Local Models, low dimensional Basis Functions

The decomposition of the input space is especially interesting for nonlinear, high order dynamic systems, as the input space is very large (and in practice such high dimensional input spaces are often impossible to fill with data), but the nonlinearity may only be dependent on a small number of the inputs. Local model nets are well suited for modelling such systems because although the system may be globally strongly nonlinearly dependent on the inputs,it may be possible to use the most important subset of the inputs to partition the input space. The system can then be locally approximated sufficiently accurately by simple (possibly linear) models which use the entire input vector, so

$$\tilde{\boldsymbol{\phi}} \subset \boldsymbol{\psi}, \dim \tilde{\boldsymbol{\phi}} < \dim \boldsymbol{\psi}. \tag{2.32}$$

In the dynamic systems' case, a dramatic reduction in the input space used for the nonlinear partition could be achieved by including only a subset of the delayed values of the inputs and state in $\tilde{\boldsymbol{\phi}}$, while all are present in $\boldsymbol{\psi}$ (i.e. the dimension of $\tilde{\boldsymbol{\phi}}$ is usually smaller than that of $\boldsymbol{\psi}$).

The well-known consequences of the 'curse of dimensionality' can be greatly reduced by defining a lower dimensional projection of the input space for the location of the basis functions and the evaluation of the distance metric (shown in Figure 2.20). This greatly simplifies the scale of task facing the structure identification algorithm.

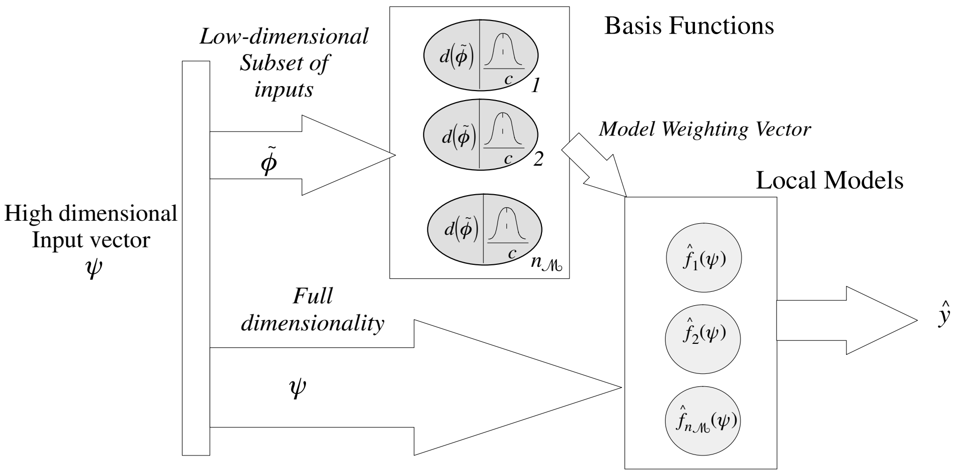


Figure 2.20: A mixed order hybrid Local Model Net system, where the operating point $\tilde{\boldsymbol{\phi}}$ has a lower dimension than the model inputs $\boldsymbol{\psi}$.

2.6 Conclusions

2.6.1 Engineering deficits of neural net solutions

The idea of creating a model of a given system by examining its behaviour, as opposed to gaining an understanding of the physical processes within the system, was not an innovation of the neural network field. The related fields of Statistics, System Identification, Cybernetics and Machine Learning all offer much support in both theoretical and practical aspects of the modelling task. Although the last decade has seen the publication of thousands of papers on neural networks, the deficiencies of many artificial neural networks for reliable use in practical applications are now becoming obvious to many working in the field. While networks like the multi-layer perceptron have been applied with success in a number of real applications, the lack of clear methods for training, analysis and validation lessen their applicability to difficult, or safety-critical projects. The 'curse of dimensionality' is a basic fact of life when producing models from data, making it necessary to introduce *a priori* knowledge–a procedure which is not well supported in multi-layer perceptrons. The lack of an engineering methodology also makes project administration in any such work more difficult, due to the unpredictability of success or failure, the variation in time needed to achieve a solution and the uncertainty about the quality of the final trained model.

2.6.2 Local Model Basis Function nets for practical problems

Local Model nets can be seen as a more general implementation of the more widely used basis function net. Local Model Nets have fewer of the engineering problems described above. The two main advantages of the architecture are:

- given local basis functions, i.e. limited overlap between models, a significantly higher level of transparency is achieved. This allows the easy local introduction of tools from other modelling paradigms (system identification, statistics etc.), and makes it easier to build *a priori* knowledge into the architecture in the form of partially or fully parameterised physical models.

- the partition also simplifies the evaluation of local confidence limits, and therefore leads to more efficient parameter and structure identification algorithms. The localised confidence estimates and constructive structure identification methods have a further advantage, as it becomes possible to automatically determine local sparsity in the training data so that more data can be demanded in active learning systems, if necessary.

- an enhanced representation, making it more suitable for modelling high dimensional and dynamic systems. Although the basis functions are still local, the more powerful local models associated with them allow the representation to be significantly more global, without the problems seen in more powerful fully global representations such as polynomial approximations, or the curse of dimensionality in fully local methods such as RBF nets.

The local model nets seem therefore to be a promising framework for the improvement of the 'learning engineering' problems which this thesis set out to attack. The local nature of the basis functions makes it easier to develop constructive structure identification algorithms. The Local Model Nets are more interpretable than other neural network architectures, and thus allow existing modelling techniques to be more easily integrated. This makes the framework potentially very powerful, as it can benefit from the wealth of theory and experience in domains such as statistics and system identification.

Chapter 3

Aspects of Local Model Networks

The methods used for the global optimisation of the parameters in local model networks are analysed, revealing frequent problems with ill-conditioning. Global optimisation methods are computationally expensive, and produce poorly interpretable final models when the underlying structure is not physically meaningful.

Local learning methods utilise the locality in the network structure to provide a more computationally efficient, more flexible and often more robust alternative. Local learning also has a regularisation effect on the optimisation, and often produces more interpretable results than global learning. The simultaneous use of a variety of local optimisation routines in heterogeneous local model nets is outlined.

Methods which use the local nature of the network structure to provide state-dependent estimates of model accuracy are described. These use the basis functions to interpolate general local error statistics. Methods for estimating the covariance of local model parameters are given, and the detection of extrapolation is discussed.

An investigation of normalisation of the basis functions reveals that normalisation can fundamentally alter properties of the basis functions in a manner not appreciated by many researchers: the shape is no longer uniform, maxima of basis functions can be shifted from their centres, and the basis functions are no longer guaranteed to decrease monotonically as distance from their centre increases. In many cases basis functions can re-appear far from the basis function centre. The consequences for model interpretation and learning algorithm development are outlined.

3.1 Local Learning vs. Global Learning

The optimisation process described in Section 2.5.1 is based on the assumption that all of the parameters $\boldsymbol{\theta}$ would be optimised simultaneously with a single regression operation. This is not always computationally feasible if a large number of training patterns or local models are needed for a particular problem (see Section 3.1.3). A further problem is that the global nature of the observation can lead to the trained network being less transparent, as the parameters of the local models cannot be interpreted independently of neighbouring nodes. Also, even with robust identification algorithms, ill-conditioning in the design matrix can lead to the 'optimal' network parameters consisting of delicately balancing large positive and negative weights which minimise the output error on the training set, but which are not robust when confronted with new examples – i.e. the model generalises poorly.

3.1.1 Problems with global optimisation methods

The *condition* of the design matrix is very important for the robustness of the optimisation process. The *condition number* of a square matrix $\mathbf{A}$ is defined to be

$$c(\mathbf{A}) = ||\mathbf{A}|| \left\|\mathbf{A}^{-1}\right\|. \tag{3.1}$$

The larger the condition number, the larger the effect of slight changes in the matrix $\mathbf{A}$ on the solution of the pseudoinverse $\mathbf{A}^{+}$. As the weights are dependent on $\mathbf{A}^{+}$, a slight change in data would lead to different weights, so generalisation is likely to be poor. SVD is used to avoid robustness problems, but it is still important to try to improve the condition of the design matrix. The condition number for the pseudoinverse of $\mathbf{A}$ can be easily calculated from the singular values produced by the SVD (see Appendix A.4 in (Söderström and Stoica, 1989)), as the norm of a matrix $||\mathbf{A}|| = s_1$ (the largest singular value) and the norm of the pseudoinverse is $||\mathbf{A}^{+}|| = \frac{1}{s_{n_s}}$, where there are n_s nonzero singular values, so the condition number is

$$c(\mathbf{A}) = \frac{s_1}{s_{n_s}}. \tag{3.2}$$

Ill-conditioning in Local Model Networks

RBF nets with widely varying basis function sizes can have condition problems, because the smaller basis functions will have relatively few data points in their receptive fields, compared to the larger ones, leading to them being treated as singularities in some cases. Local model networks, with global optimisation, tend to be more prone to ill-conditioning than simple RBF nets, because of the increased level of correlation between basis elements in the regression problem – the same inputs appear in each local model, the only difference being differing weightings provided by their basis functions.[1]

[1]The condition can be improved slightly by norming the inputs to the individual local models by using deviation from the centre (or operating point in conventional linearisation theory) as an input to the local

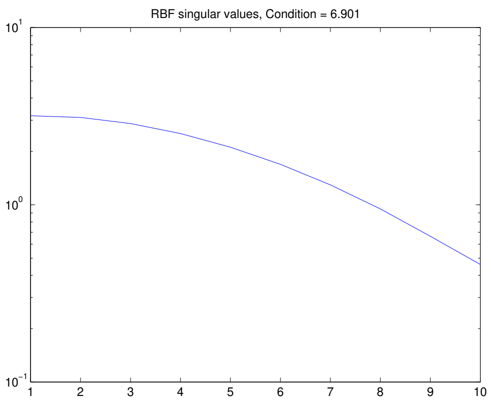
A one-dimensional example is used to illustrate the relationship between the properties of the networks and the associated singular values for a variety of local model nets. The arbitrarily chosen nonlinear function is

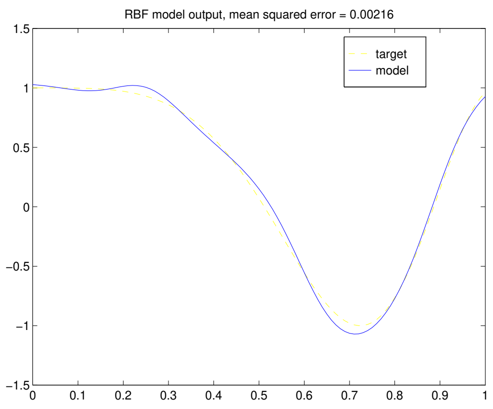
$$y(x) = \cos(6x^2) + \varepsilon(x), \tag{3.3}$$

where the additive noise term $\varepsilon(x)$ is Gaussian with a varying standard deviation of $\sigma(x) = 0.4 \exp\left(- \left|x - \frac{1}{2}\right| 4.6\right)$. The examples plotted in Figure 3.1 used 401 training examples (see Figure 3.5 on page 63 for a plot of the training data).

Even using deviation inputs to the local models, the condition of the design matrix still deteriorates rapidly with increasing numbers of local models, as can be seen from the example in Figure 3.3(b).

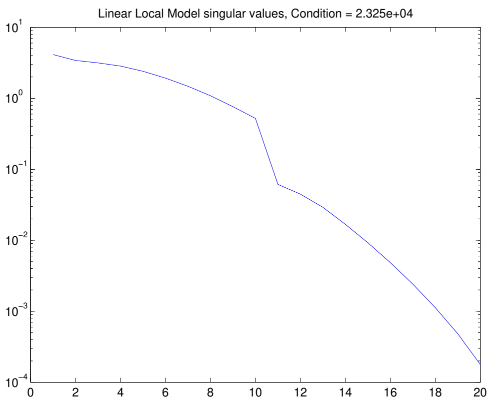
models $\boldsymbol{\psi}_{dev} = \frac{\boldsymbol{\psi} - \mathbf{c}_i}{\sigma_i}$, so that the regression bases are in the same numerical range, but there is less correlation between bases.

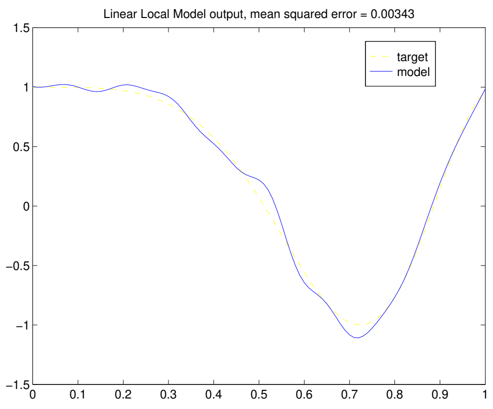




(a) RBF Singular Values

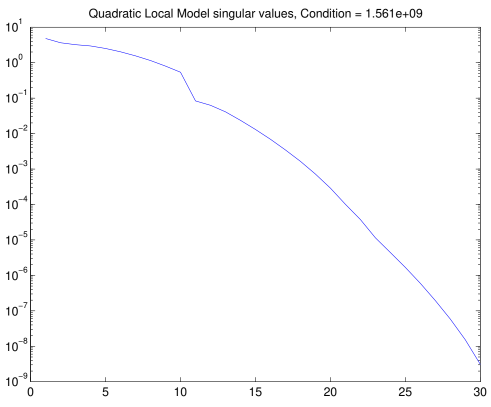
(b) RBF Approximation

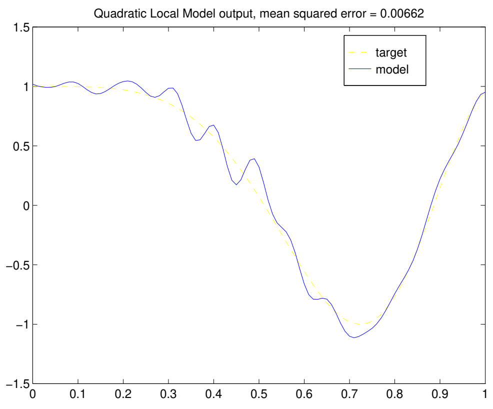




(c) Linear Local Model Singular Values

(d) Linear Local Model Approximation

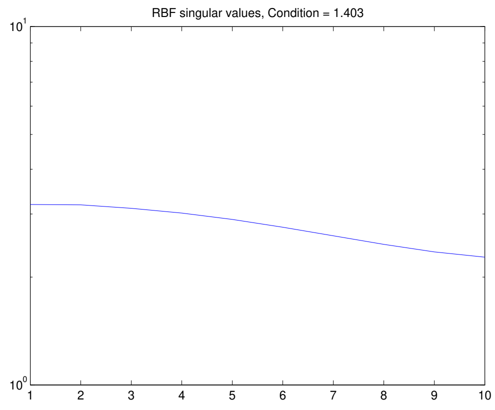




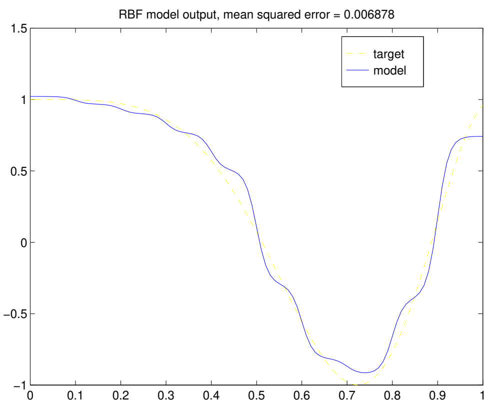
(e) Quadratic Local Model Singular Values

(f) Quadratic Local Model Approximation

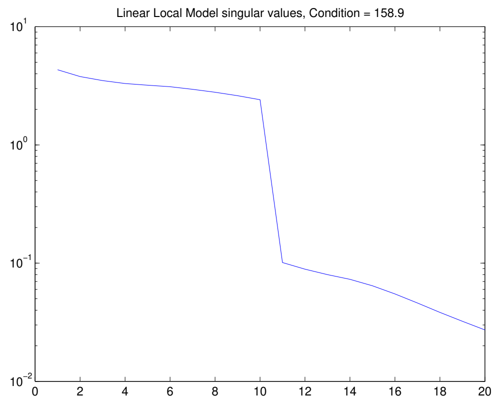
Figure 3.1: A set of 10 basis functions is fixed and used to represent the system using constant, linear and quadratic local models. The singular values for piecewise constant, linear and quadratic models are plotted. The drops in the singular values at 11 in the linear local model case, and at 11 and 22 in the quadratic model case are less pronounced than in Figure 3.2.



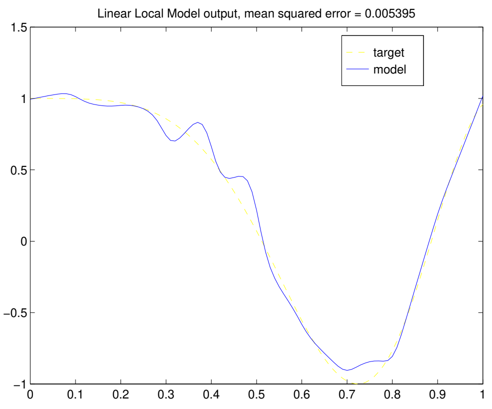
(a) RBF Singular Values



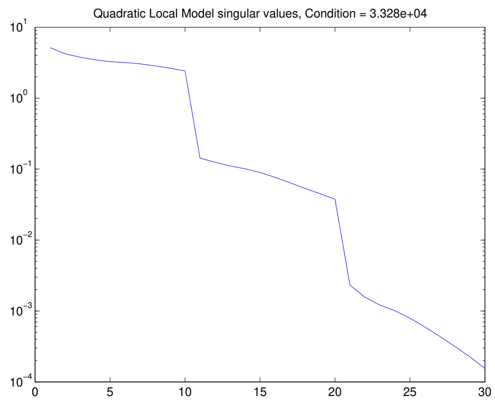
(b) RBF Approximation



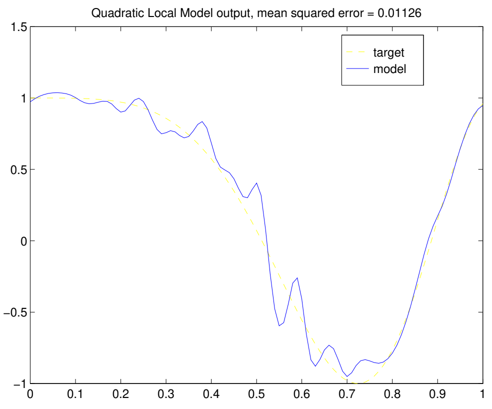
(c) Linear Local Model Singular Values



(d) Linear Local Model Approximation



(e) Quadratic Local Model Singular Values

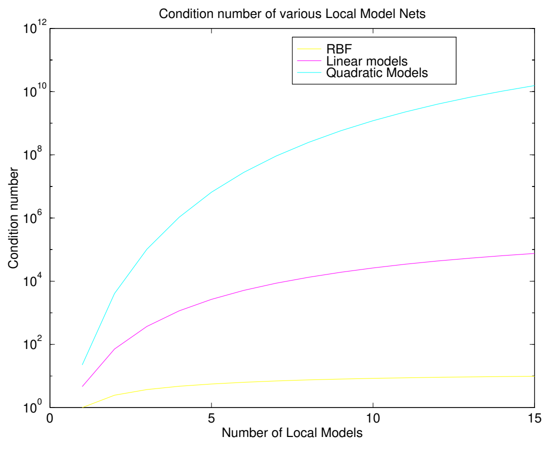


(f) Quadratic Local Model Approximation

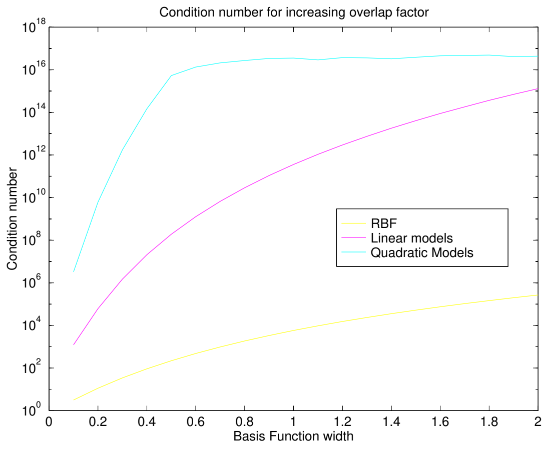
Figure 3.2: The same case as in Figure 3.1, but with basis functions half the size. Notice the sharp drops in the singular values at 11 in the linear local model case, and at 11 and 22 in the quadratic model case. These indicate that the basis functions associated with the higher order inputs are more likely to contribute to poor conditioning in learning. Note also the effect of narrow basis functions on the smoothness of the RBF model output – the normalised basis functions are more step-like, reducing the smoothness of the model output.

Effect of basis function overlap factor on condition

Figure 3.3(a) shows the increase in condition number with increasing numbers of local models, when the relative overlap remains identical. To better understand the role of overlap, Figure 3.3(b) shows the increase in condition number in a local model net with a fixed number of units (seven) when the overlap is increased. The interpretation of this for learning systems is that an increase in the number of models, or the level of overlap of the basis functions can lead to a poorly conditioned optimisation problem, and models which do not generalise well. Too few models, however, will obviously lead to poor approximation, and too little overlap leads to non-smooth approximations.



(a) Constant relative Overlap for net with uniform basis functions evenly space, with $\sigma = |c_1 - c_2|$.



(b) Varying Overlap. The x-axis shows the scaling factor for the basis function width. 1 represents a width related to the distance between centres, i.e. $\sigma = |c_1 - c_2|$.

Figure 3.3: Condition number increasing with number of local models or with overlap. Basis functions were normalised.

3.1.2 Local learning

An alternative to global learning is to locally estimate the parameters of each of the local models (as defined in equation (2.18)) independently[2]. As described in (Murray-Smith, 1994), potential advantages of *local learning* include:

- The local optimisation will be more computationally efficient (see Section 3.1.3).

[2]this assumes that the basis functions achieve a partition of unity.

- In the global case the design matrix ($\mathbf{\Phi}$) has many elements which are close to zero since only a small number of validity functions are significantly non-zero at any point in the input space. Also, given a large degree of overlap, many local model bases will appear very similar. Both features can lead to poor conditioning of the optimisation equations and numerical problems.

  The restriction involved in local learning means that the final locally trained network will often be more robust when facing new data than a globally trained model.

- Heterogeneous local model networks can be defined which use a variety of optimisation algorithms (possibly also nonlinear), each suited to the individual local model type.

- If global training is used, the parameters of a 'local' model can often have no local meaning, as they depend on interaction with their neighbours to produce the correct model behaviour, whereas locally trained local models can be interpreted independently of neighbouring local models. This is extremely important, as an oft-cited advantage of local model networks is that the trained local models are easier to interpret than other representations as they are already in a locally interpretable form[3].

Local learning with weighted least squares

For the global criterion it is possible to set $\alpha(\boldsymbol{\psi}_i) = 1$ for all i, or to select a function $\alpha(\boldsymbol{\psi}_i)$ to weight the importance of areas of the input space to the optimisation process (similarly to equation (2.21). To achieve local learning it is necessary to define a set of local criteria. A given local model's basis function can be used to define that model's relevance for any given input. For the local criteria, on the other hand, the weights must be chosen to take direct account of the interactions of the validity functions. Our confidence in a given observation regarding its relevance for the i-th local model is directly reflected in the i-th validity function. The local weighting functions should therefore be set as

$$\alpha_i(\boldsymbol{\psi}) = \rho_i(\tilde{\boldsymbol{\phi}}), \tag{3.4}$$

where $\tilde{\boldsymbol{\phi}}$ is the subset of $\boldsymbol{\psi}$ related to the basis functions, which results in a set of local estimation criteria for the i-th local model (where $i = 1..n_{\mathcal{M}}$) of

$$J_i(\boldsymbol{\theta}_i) = \frac{1}{N_i}\sum_{k=1}^{N_i} \rho_i(\tilde{\boldsymbol{\phi}}_{i_k})(y_{i_k} - \hat{y}_{i_k})^2. \tag{3.5}$$

where N_i is the number of examples in the local training set $\mathcal{D}_i$ limited to the receptive field of local model i, and $\hat{y}_{i_k}$ is the output from local model i, for data point k from $\mathcal{D}_i$. In this case the estimate of the local model parameter vector $\boldsymbol{\theta}_i$ is given by $\hat{\boldsymbol{\theta}}_i = \arg\min J_i(\boldsymbol{\theta}_i)$. In matrix terms the operation is now

$$\hat{\boldsymbol{\theta}}_i = (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{Y}, \tag{3.6}$$

[3]The reduction in interference will also possibly make it more suitable for use with on-line adaptation algorithms.

where $\mathbf{\Phi}_i$ is an $N_i \times (n_\psi + 1)$ vector,

$$\mathbf{\Phi}_i = \begin{pmatrix} \boldsymbol{\phi}_{i1} \\ \cdot \\ \cdot \\ \boldsymbol{\phi}_{iN_i} \end{pmatrix}, \tag{3.7}$$

where the regression variables are:

$$\boldsymbol{\phi}_{ik} = [1 \; \boldsymbol{\psi}_k], \tag{3.8}$$

where the k refers to the kth example in local training set $\mathcal{D}_i$. $\mathbf{Q}_i$ is an $N_i \times N_i$ diagonal matrix, where the diagonal elements are the activations of the basis function of the ith model over the training set $\mathcal{D}_i$,

$$\mathbf{Q}_i = \begin{pmatrix} \rho_i(\tilde{\boldsymbol{\phi}}_{i_1}) & 0 & 0 & 0 \\ 0 & \rho_i(\tilde{\boldsymbol{\phi}}_{i_2}) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \rho_i(\tilde{\boldsymbol{\phi}}_{i_{N_i}}) \end{pmatrix}, \tag{3.9}$$

The local learning method is therefore to compute $n_\mathcal{M}$ locally weighted least squares regressions, one for each local model, using only the training data $\mathcal{D}_i$ within the model's receptive field, and with only the bases related to the given local model's parameters. The analogous method for straightforward RBF networks would be to set the weight of a unit to the weighted average of the data points in its receptive field, as in (Pantaleón-Prieto et al., 1993). (Johansen and Foss, 1992a) applied local learning to local model networks, but did not use the basis functions as a weighting function.

To gain a better understanding of the difference in the cost function associated with the local learning technique the global and local cost functions can be expanded. First the local one:

$$J_i(\boldsymbol{\theta}_i) = \frac{1}{N_i}\left(\mathbf{Y} - \hat{\mathbf{Y}}_i\right)^T \mathbf{Q}_i \left(\mathbf{Y} - \hat{\mathbf{Y}}_i\right), \tag{3.10}$$

so the local learning cost function is

$$J_i(\boldsymbol{\theta}_i) = \frac{1}{N_i}\left(\mathbf{Y}^T\mathbf{Q}_i\mathbf{Y} - 2\hat{\mathbf{Y}}_i^T\mathbf{Q}_i\mathbf{Y} + \hat{\mathbf{Y}}_i^T\mathbf{Q}_i\hat{\mathbf{Y}}_i\right), \tag{3.11}$$

where $\hat{\mathbf{Y}}_i = \mathbf{\Phi}_i\boldsymbol{\theta}_i$.

The global least squares cost function

$$J(\boldsymbol{\theta}) = \frac{1}{N}\left(\mathbf{Y} - \hat{\mathbf{Y}}\right)^T \left(\mathbf{Y} - \hat{\mathbf{Y}}\right), \tag{3.12}$$

so

$$J(\boldsymbol{\theta}) = \frac{1}{N}\left(\mathbf{Y}^T\mathbf{Y} - 2\hat{\mathbf{Y}}^T\mathbf{Y} + \hat{\mathbf{Y}}^T\hat{\mathbf{Y}}\right), \tag{3.13}$$

where $\hat{\mathbf{Y}} = \mathbf{\Phi}\boldsymbol{\theta}$, $\mathbf{\Phi}$ being as defined in Section 2.5.1,

$$J(\boldsymbol{\theta}) = \frac{1}{N}\left(\mathbf{Y}^T\mathbf{Y} - 2\boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{Y} + \boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{\Phi}\boldsymbol{\theta}\right). \tag{3.14}$$

Consider only the subset of the parameters $\boldsymbol{\theta}_i$ associated with local model i resulting from the global optimisation of $\boldsymbol{\theta}$. Decomposing the design matrix $\mathbf{\Phi}$ into $\mathbf{\Lambda}_i$ and $\mathbf{\Gamma}_i$ gives more insight into the interplay of the neighbouring basis functions with the parameters for model i,

$$\mathbf{\Phi} = \mathbf{\Lambda}_i + \mathbf{\Gamma}_i, \tag{3.15}$$

where the parameters $\boldsymbol{\theta}_i$ are supported by $\mathbf{\Gamma}_i$, the submatrix of $\mathbf{\Phi}$ for inputs covered by basis function $\rho_i(\cdot)$

$$\mathbf{\Gamma}_i = \begin{pmatrix} 0 & \dots & \rho_i(\tilde{\boldsymbol{\phi}}_{i1})\boldsymbol{\phi}_{i1} & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \dots & \rho_i(\tilde{\boldsymbol{\phi}}_{iN_i})\boldsymbol{\phi}_{iN_i} & \dots & 0 \end{pmatrix}. \tag{3.16}$$

The parameters $\boldsymbol{\theta}_i$ are also affected by the matrix $\mathbf{\Lambda}_i$ containing the inputs supported by neighbouring basis functions other than i.

$$\mathbf{\Lambda}_i = \begin{pmatrix} \rho_1(\tilde{\boldsymbol{\phi}}_{i1})\boldsymbol{\phi}_{i1} & \rho_k(\tilde{\boldsymbol{\phi}}_{i1})\boldsymbol{\phi}_{i1} & \dots & \rho_i(\cdot)0 & \dots & \rho_{n_{\mathcal{M}}}(\tilde{\boldsymbol{\phi}}_{i1})\boldsymbol{\phi}_{i1} \\ & \vdots & & \vdots & & \\ \rho_1(\tilde{\boldsymbol{\phi}}_{iN_i})\boldsymbol{\phi}_{iN_i} & \rho_k(\tilde{\boldsymbol{\phi}}_{iN_i})\boldsymbol{\phi}_{iN_i} & \dots & \rho_i(\cdot)0 & \dots & \rho_{n_{\mathcal{M}}}(\tilde{\boldsymbol{\phi}}_{iN_i})\boldsymbol{\phi}_{iN_i} \end{pmatrix}, \tag{3.17}$$

Decomposing the cost functional to highlight the effect of the other basis functions on model i's parameters, gives

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{N}\left(\mathbf{Y}^T\mathbf{Y} - 2\boldsymbol{\theta}^T\mathbf{\Gamma}_i\mathbf{Y} - 2\boldsymbol{\theta}^T\mathbf{\Lambda}_i\mathbf{Y} + \boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{\Phi}\boldsymbol{\theta}\right) && (3.18) \\ &= \frac{1}{N}\left(\mathbf{Y}^T\mathbf{Y} - 2\boldsymbol{\theta}_i^T\mathbf{\Phi}_i{}^T\mathbf{Q}_i^T\mathbf{Y} - 2\boldsymbol{\theta}^T\mathbf{\Lambda}_i\mathbf{Y} + \boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{\Phi}\boldsymbol{\theta}\right), && (3.19) \end{aligned}$$

To better understand the difference in the cost functions we examine $\boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{\Phi}\boldsymbol{\theta}$, where

$$\mathbf{\Phi}^T\mathbf{\Phi} = \mathbf{\Gamma}_i^T\mathbf{\Lambda}_i + \mathbf{\Gamma}^T\mathbf{\Gamma} + \mathbf{\Lambda}_i^T\mathbf{\Gamma}_i + \mathbf{\Lambda}_i^T\mathbf{\Lambda}_i, \tag{3.20}$$

which clearly shows the interaction between the overlapping local models (the $\mathbf{\Lambda}_i$ terms) and local model i (the $\mathbf{\Gamma}_i$ term). By multiplying by the weights we can bring the equation into a form more comparable with the local cost function in equation (3.11).

$$\begin{aligned} \boldsymbol{\theta}^T\mathbf{\Phi}^T\mathbf{\Phi}\boldsymbol{\theta} &= \boldsymbol{\theta}_i^T\mathbf{\Gamma}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} + \boldsymbol{\theta}_i^T\mathbf{\Gamma}^T\mathbf{\Gamma}\boldsymbol{\theta}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{\Gamma}_i\boldsymbol{\theta}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} && (3.21) \\ &= \boldsymbol{\theta}_i^T\mathbf{\Phi}_i{}^T\mathbf{Q}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} + \boldsymbol{\theta}_i^T\mathbf{\Phi}_i{}^T\mathbf{Q}_i^T\mathbf{Q}_i\mathbf{\Phi}_i\boldsymbol{\theta}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{Q}_i\mathbf{\Phi}_i\boldsymbol{\theta}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} && (3.22) \\ &= \hat{\mathbf{Y}}_i^T\mathbf{Q}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} + \hat{\mathbf{Y}}_i^T\mathbf{Q}_i^T\mathbf{Q}_i\hat{\mathbf{Y}}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{Q}_i\hat{\mathbf{Y}}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta}, && (3.23) \end{aligned}$$

so that the cost function for $\boldsymbol{\theta}_i$ is

$$\begin{aligned} J'(\boldsymbol{\theta}) &= \frac{1}{N}(\mathbf{Y}^T\mathbf{Y} - 2\hat{\mathbf{Y}}_i^T\mathbf{Q}_i^T\mathbf{Y} + \hat{\mathbf{Y}}_i^T\mathbf{Q}_i^T\mathbf{Q}_i\hat{\mathbf{Y}}_i - \\ &\quad 2\boldsymbol{\theta}^T\mathbf{\Lambda}_i\mathbf{Y} + \hat{\mathbf{Y}}_i\mathbf{Q}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta} + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{Q}_i\hat{\mathbf{Y}}_i + \boldsymbol{\theta}^T\mathbf{\Lambda}_i^T\mathbf{\Lambda}_i\boldsymbol{\theta}). && (3.24) \end{aligned}$$

The local form's $\hat{\mathbf{Y}}_i^T \mathbf{Q}_i \hat{\mathbf{Y}}_i$ from equation (3.11) corresponds to a $\hat{\mathbf{Y}}_i^T \mathbf{Q}_i^T \mathbf{Q}_i \hat{\mathbf{Y}}_i$. The global $\hat{\mathbf{Y}}^T \hat{\mathbf{Y}}$ corresponds to the weighted $\hat{\mathbf{Y}}^T \mathbf{Q}_i \hat{\mathbf{Y}}$ in the local functional.

The major effect of local learning on the cost functional is to remove the $\mathbf{\Lambda}_i$ terms. $\mathbf{\Lambda}_i$ consists of model i's neighbouring basis functions which overlap with i, so the greater the level of overlap, the more significant the difference between cost functions. High levels of overlap lead to $\mathbf{\Lambda}_i$ being more significant and leading to a higher correlation with the off-diagonal term, which becomes the major contributing factor to the poor conditioning of global optimisation problems in local model nets.

3.1.3 Global vs. local SVD for computational effort

The effort needed to find the pseudoinverse using SVD for a $(p \times q)$ matrix is roughly (Noble and Daniel, 1988),

$$O\left(p^2 q + p q^2 + \min(p, q)^3\right). \tag{3.25}$$

In terms of the design matrix for a basis function network, p relates to the number of training points and q is the number of basis terms. The cubic term shows the importance of the smallest dimension of the matrix on the complexity of the calculation. As the set of linear equations should be over-determined, the smaller number is q, representing the number of basis elements, and this implies that the product of the number of local models and the number of their parameters is the crucial factor with regard to computational effort. To compare global and local learning, calculate O_{global} for a homogeneous local model net, with linear local models, where $p = N$ and $q = n_{\mathcal{M}} n_{\psi}$, where n_{ψ} represents the full dimension of the model's input space, and O_{local} where $p = N_i$ (number of training points in local model i's receptive field, $q = n_{\psi}$ and which is repeated $n_{\mathcal{M}}$ times. It is difficult to compare the methods exactly, as the reduction in the number of training points in a particular area is dependent on the problem in question and will vary for each local model, but even using the conservative estimate, which expects N to be the same in both cases,

$$O_{global} = O\left(N^2 (n_{\mathcal{M}} n_{\psi}) + N (n_{\mathcal{M}} n_{\psi})^2 + \min(N, (n_{\mathcal{M}} n_{\psi}))^3\right) \tag{3.26}$$

and

$$O_{local} = O\left(n_{\mathcal{M}} \left(N_i^2 n_{\psi} + N_i n_{\psi}^2 + \min(N_i, n_{\psi})^3\right)\right). \tag{3.27}$$

Even ignoring the speed-up gained by the reduced number of points (as $N_i \leq N$), the local variant will be faster for all $n_{\mathcal{M}}$ greater than 1. The effort for local learning also increases linearly in $n_{\mathcal{M}}$ as opposed to the cube of $(n_{\mathcal{M}} n_{\psi})$, which makes local learning far more suitable for larger problems.

3.1.4 Local learning experiments

Approximation of a 1-D noisy function

The system described in equation 3.3 is used to give an impression of the regularisation effect of local learning on the final solution of a learning problem, compared to global learning. The function is shown below in Figure 3.4. The global learning was carried out using the SVD algorithm, zeroing singular values smaller than 10^{-5}. In the figures shown here, the inputs to the local models were not deviations from the centre, but the absolute value. Experiments where the local models used deviations from their centres resulted in the same order of model mismatch and ill-conditioning. The cost measures were taken from the model's deviation from 1000 noise free points randomly distributed throughout the input space.

For smaller training sets (101 patterns), the robustness of the local learning is immediately obvious, both in the smoother response, and in the lower error on the training data. Figure 3.5 shows the same problem with 401 patterns, but even though the amount of training data has increased, the global measure is still worse than that from local learning.

Even when there are 1001 training points, although the global approximation is now better than the local one, the parameters of the global model have little physical significance, in the sense of being a local approximation to the real system.

The robustness of local learning is demonstrated experimentally with artificial test examples in Chapters 4 and with real applications in Chapter 6. A hierarchical form of local learning is introduced in Chapter 5.

Robustness of locally learned solutions

Ideally, a learning algorithm should robustly deliver a solution which has as high a level of accuracy as possible, and which responds robustly to new data, i.e. it is likely to generalise well. The smoothness of the resulting model is also important for many applications. In many cases, a smooth model which has a poorer least squares cost is better than a 'more' exact but 'wrinkled' model. For example, in model based predictive control, the optimal control setting is often found using gradient search methods, which, if the model is not smooth, would then be subject to many local minima and would lead to unreliable control. The examples here, the two-dimensional example shown in Figure 4.12 on page 103, and the practical results in Chapter 6 indicate that local learning can often have a smoothing effect on the trained model's response. It cannot be stated generally, as the weaker approximation abilites of local learning may force the structure identification algorithm to use a far larger number of basis functions, leading to a less robust network, but in a number of examples in this thesis an improvement in smoothness was noted.

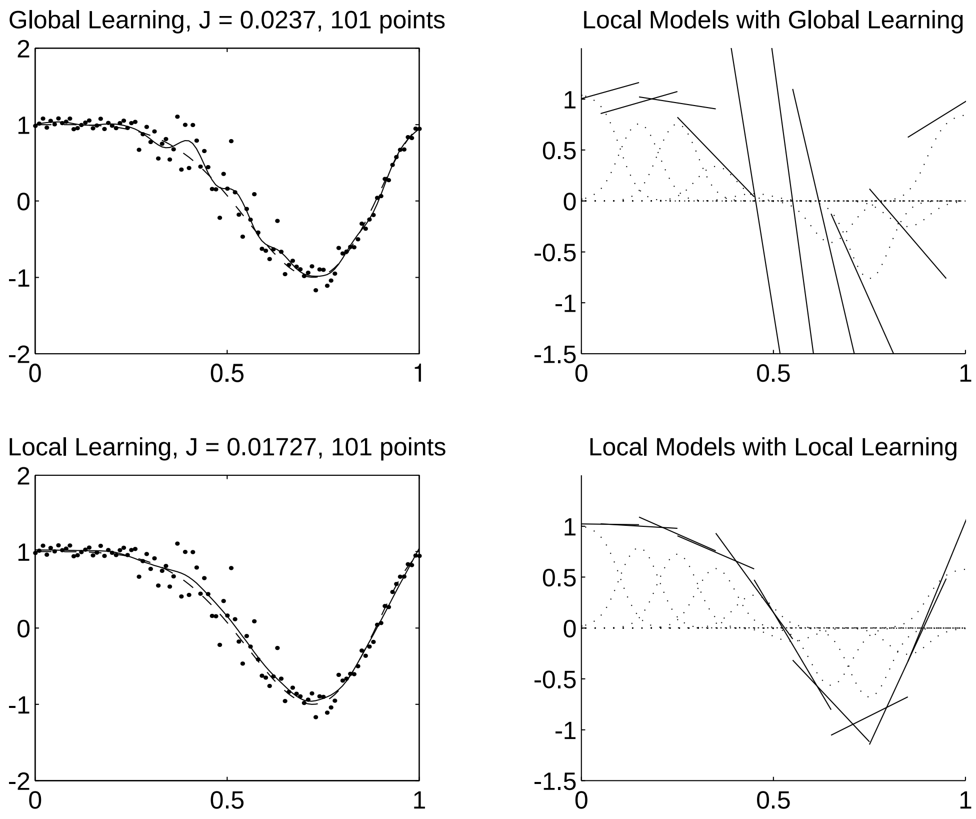


Figure 3.4: Experimental comparison of Global and Local Learning for 101 training points. The left hand side shows the target function, the noisy training data and the trained network's response. The cost functional J in the figure titles, is the same as equation (2.21) where the weighting function $\alpha(\mathbf{x}) = \frac{1}{\sigma(x)}$. J is evaluated on the model's deviation from the noise-free outputs. The right hand side of each figure shows the normalised basis functions and the associated local linear models.

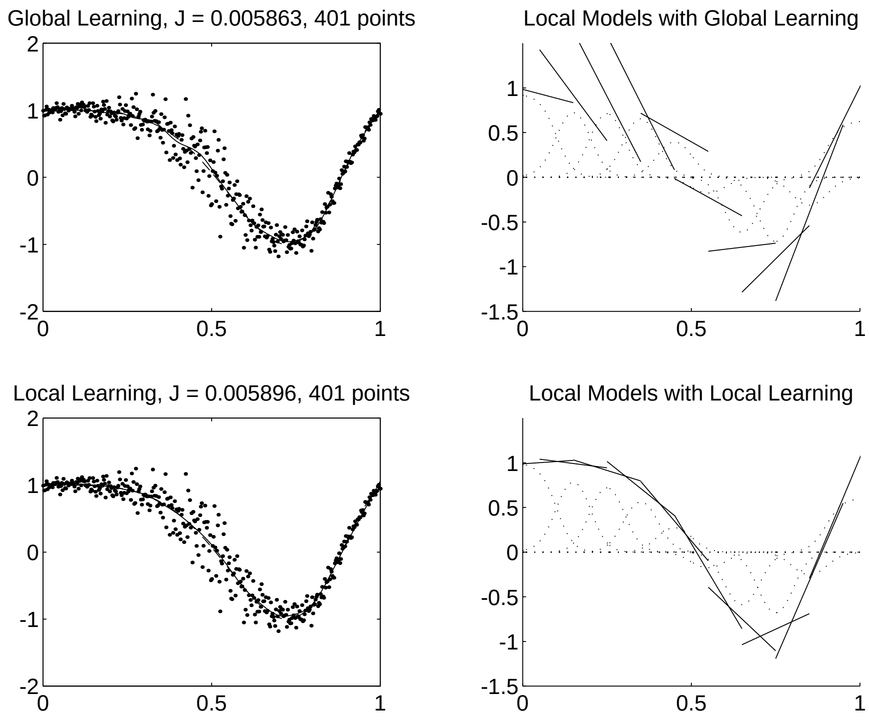


Figure 3.5: Local vs. Global estimation – continuation of Figure 3.4 for 401 training points.

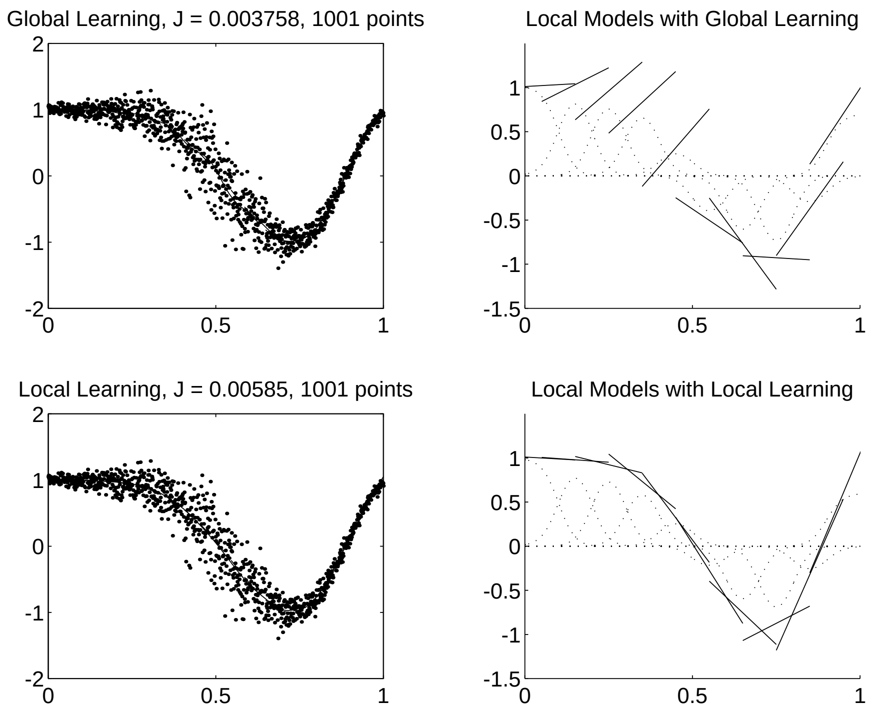


Figure 3.6: Local vs. Global estimation – continuation of Figure 3.4 for 1001 training points.

3.1.5 Training heterogeneous local model nets

The local learning method can also be used as a method for optimising the parameters of local model nets which are not linear in the parameters, so that each local model locally applies its own optimisation algorithm. For example, in Figure 3.7 the three local models could have three different optimisation algorithms. $f_1(\psi)$ is an ARX model, and could use an linear optimisation routine, $f_2(\psi)$ is a step model, which could be adapted by a local line search, and $f_3(\psi)$ is a neural network, e.g. a multi-layer perceptron which could be optimised by an algorithm such as back-propagation. The basis functions could also be pre-structured using fuzzy rules, or some *a priori* choice of basis function.

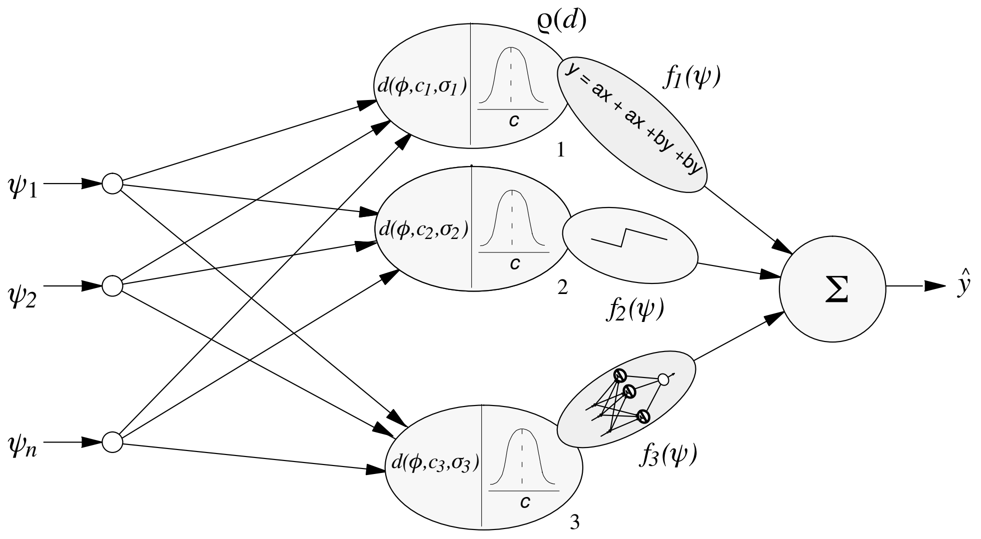


Figure 3.7: Heterogeneous local model network with multi-algorithm optimisation

3.2 Estimating the Confidence in a Trained Network

When using learning systems like neural networks for tasks such as classification, prediction, modelling or control it is essential to have an estimate of the system's accuracy for any given operating region. There is little sense in reacting to the output provided by such a system with no clear estimate of the accuracy of the information. Such confidence limits often provide necessary information for further processing – which algorithms are most suitable, which can be ignored, what is the likely cost of a mistake? The relevance of good estimates of model accuracy for model based controllers and model based fault diagnosis systems is obvious, and this has long been an area of research in conventional statistics, where a variety of methods for the analysis of samples, the validation of models, etc., has been developed. The methods developed in standard statistics or identification tend to produce global measures of accuracy for the system. The methods developed for the simpler techniques are, however, not suitable for methods which have varying representational complexity in different areas of the input space.

An example of the inadequacy of global measures of error is the maximum absolute error of a model. In some cases this can *increase* during training, e.g. learning to model a step function, as shown in Figure 3.8:

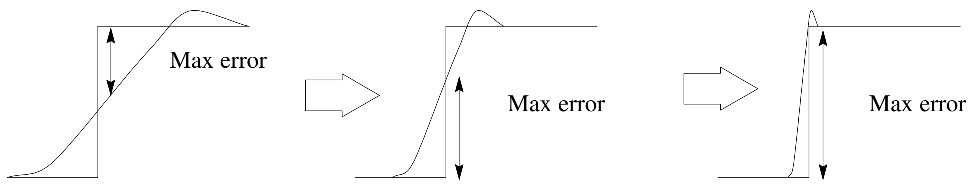


Figure 3.8: As the model improves its average performance, the worst error can increase!

Despite the problems described above, the general problem of how to estimate the confidence in the system for given situations is even more relevant for the case of complex non-linear methods such as neural networks than for linear systems.

The confidence in the output of the trained empirical model at run-time is related to several factors:

- The accuracy achieved at modelling the input-output behaviour, which is related to
  - the accuracy achieved on the given data
  - the accuracy of the information in the training set.
  - the ratio of data available against model complexity (parsimonious models are preferred[4]).
- The measurement noise on the inputs at run-time.

[4]Pruning methods for model reduction are described in Section 4.2.4. These can be seen as simple methods for the automatic detection and reduction of non-parsimonious model structures.

- The performance of the model as a whole, used as it is intended to be used.
- The sense of the fitted model in the light of *a priori* knowledge.

The last point is important for an improved understanding of the model's limits, and in order to try to learn more about the process being modelled. In local model networks where the local model structures are based on physical insight, this becomes even more important. The local interpretation will depend, however, on the parameters being identified locally, as described in Section 3.1.2. Well-known statistical techniques can be applied to estimate the variance of the estimated parameters, as described in Section 3.2.3.

One important aspect, however, is the suitability of the model for the purpose intended of it. In many applications important aspects of the real process may have been ignored during the modelling process, or cost functions which seemed suitable, may not produce the desired result when coupled with other system components – this is especially true in dynamic systems. It is usually too expensive, dangerous or time-consuming to test all proposed models in the real system, so other methods must be available to eliminate faulty models in advance.

Cross-validation methods are described in the next section which use the local nature of the basis functions to produce local confidence limits, show how standard statistical methods can be used to determine the covariance of the model parameters, and discuss methods for the detection of model extrapolation in areas in which it had insufficient training data.

3.2.1 Local confidence measures

The bulk of model validation techniques used in day-to-day system identification are based on measuring the difference between the model outputs and the observed outputs on validation data sets not used to develop the model. The majority of these are global methods which do not produce state-dependent results, even though the errors found in the system may vary dramatically throughout the input space. The local model network structure is well suited to the production of local estimates of accuracy because of the partitioning of the input space inherent to the model structure. The well-known model validation methods which can be applied to conventional systems can now be applied at a local level, and the results interpolated by the basis functions associated with the local models.

The statistics resulting from the local test are weighted by the basis function activations for a given input to give a local estimate of the given statistic. If a general error statistic ϵ_i is locally acquired for each local model $i = 1..n_{\mathcal{M}}$, the global estimate $\epsilon(\tilde{\boldsymbol{\phi}})$ for a given operating point $\tilde{\boldsymbol{\phi}}$ is

$$\hat{\epsilon}(\tilde{\boldsymbol{\phi}}) = \sum_{i=1}^{n_{\mathcal{M}}} \hat{\epsilon}_i \rho_i(\tilde{\boldsymbol{\phi}}). \tag{3.28}$$

Examples of error statistics

As an example of the above technique, cross-validation methods can be applied locally to give local estimates of worst errors, variance and bias. This is a very simple method, but can still be very useful. The most straightforward error statistics are the local mean and worst errors on a single validation set. The mean absolute error can be derived as,

$$\hat{\epsilon}_i = \frac{1}{N_i}\sum_{k=1}^{N_i} \rho_i(\tilde{\boldsymbol{\phi}}_k)\,|y_k - \hat{y}_k|\,. \tag{3.29}$$

using the basis function activation to weight the errors, where N_i is the number of points from the validation set in ρ_i. The mean squared error can be defined similarly.

$$\hat{\epsilon}_i = \frac{1}{N_i}\sum_{k=1}^{N_i} \rho_i(\tilde{\boldsymbol{\phi}}_k)\,(y_k - \hat{y}_k)^2\,. \tag{3.30}$$

The use of the basis function to weight the error becomes more complex for local worst error estimates. The weighted form is

$$\hat{\epsilon}_i = \max\left(\rho_i(\tilde{\boldsymbol{\phi}}_k)\,|y_k - \hat{y}_k|\right), k = 1..N_i, \tag{3.31}$$

which tends to underestimate the worst error, whereas using a cutoff point ζ for the basis function activation, to determine whether an error will be defined as being associated with a given local model,

$$\hat{\epsilon}_i = \max\left(|y_k - \hat{y}_k|\right), k = 1..N_i \;\;\&\;\; \rho_i(\tilde{\boldsymbol{\phi}}_k) > \zeta \tag{3.32}$$

is too conservative.

Illustrative example

The results of the implementation for the function $z = (y - 0.5)^2 \sin(2\pi x \sin(\pi(x - 0.1)^3))$ are shown in Figure 3.10, with the basis functions organised in a grid. The target function is shown in Figure 3.9. Data points were noise-free and uniformly randomly distributed over the entire input space. The weighted worst absolute error statistic from equation (3.31) was used. The results seem intuitively compatible with the model's approximation of the function. The confidence bands for the solution are larger where the target function is more complex and the model therefore a poorer fit. Further experimental results with the various error statistics are given with the rolling mill application in Chapter 6.

3.2.2 Detecting extrapolation

Extrapolation is the act of estimating the response of a system, assuming certain restrictions such as smoothness constraints, beyond the bounds of knowledge about that system, and can often lead to unpredictable results. This happens with learning systems when the trained

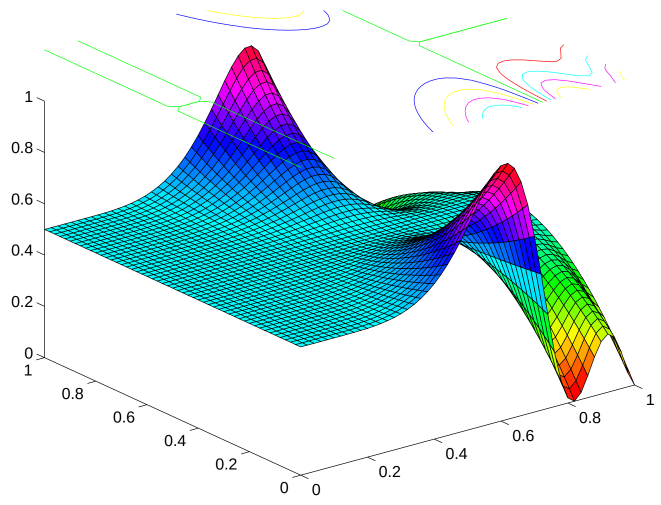


Figure 3.9: Test function. z is vertical axis. x and y are right and left axes respectively

model is presented with data outwith the area covered by the training data. Leonard et al. (Leonard et al., 1992) describe a 'Validity Index' net which warns the user when a value is being extrapolated. Another method by Kramer is the use of 'Rho-nets' (Kramer, 1993), which uses basis function networks to create a probability distribution function of the data existing in the training set, and then notifies the user if the inputs stray from this area at run-time.[5] This method has also been used in constructive structure identification algorithms for basis function nets (e.g. (Raipala and Koivo, 1992, Roberts and Tarassenko, 1994))

As described in Section 3.3.1, the use of normalised basis functions leads to the basis functions covering larger areas of the input space, and means that the methods described above cannot be as easily used to detect extrapolation in normalised BF or Local Model nets.

3.2.3 Estimating covariance of weight estimates from the residuals

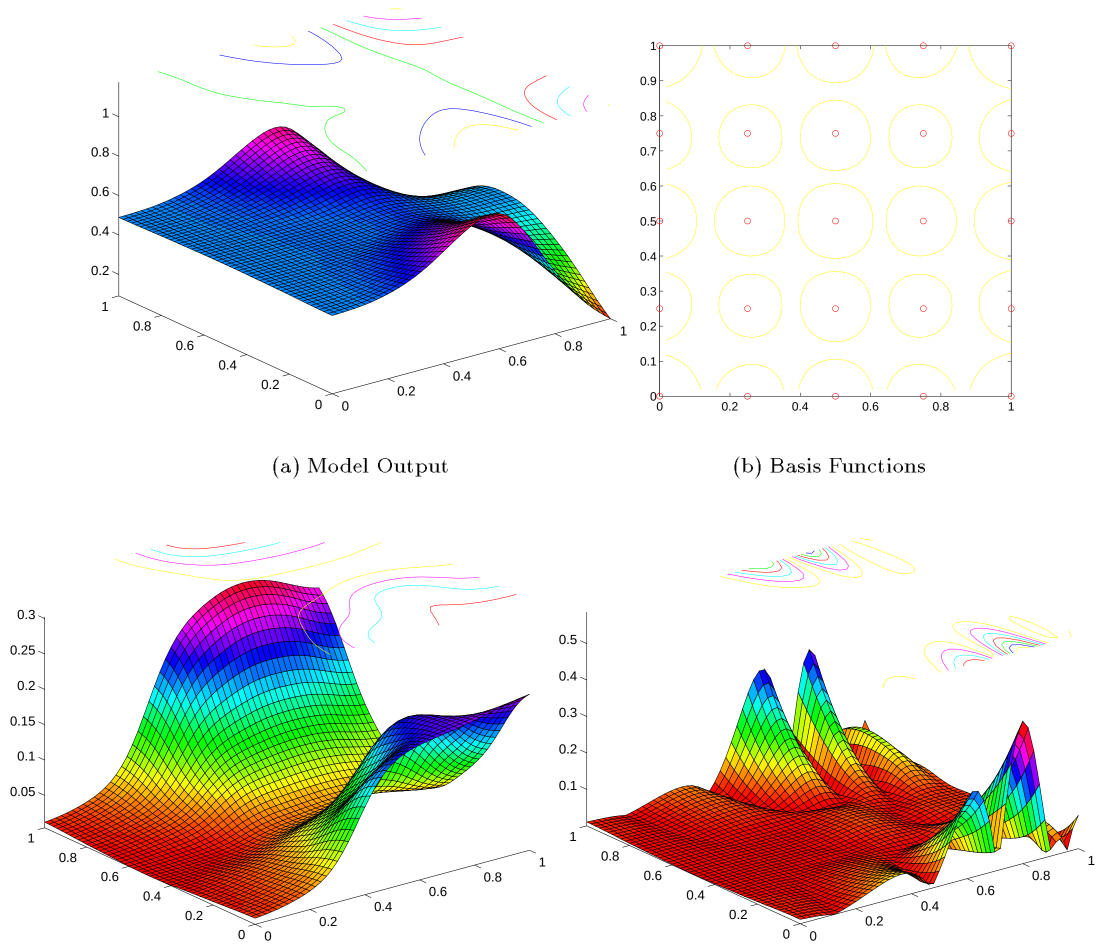
If the local models being used are based on underlying physical structures, it is often interesting to examine the parameters after training in order to interpret the physical relevance of the model. It is therefore useful to know the covariance of a given parameter estimate. The covariance estimation problem for *locally trained* local model nets is equivalent to that of an optimal weighted linear local model,

$$\hat{\mathbf{Y}}_i = \mathbf{\Phi}_i \boldsymbol{\theta}_i, \tag{3.33}$$

such that the real output vector $\mathbf{Y}$ will be defined by the model's parameters ($\boldsymbol{\theta}_i$) and its residuals ($\mathbf{D}$):

$$\mathbf{Y} = \mathbf{\Phi}_i \boldsymbol{\theta}_i + \mathbf{D}, \tag{3.34}$$

[5] This can be very important for model-based predictive control, where the optimisation routines can produce very 'unnatural' inputs, lying outside the sampled data range, so that the model behaves poorly for the purpose intended of the model



(a) Model Output

(b) Basis Functions

(c) Locally Estimated Maximum Error

(d) Measured Absolute Error

Figure 3.10: Forming local confidence limits from 'worst error' cross-validation results. The model is trained on the test function shown in Figure 3.9 using local learning.

where $\mathbf{\Phi}_i$ is as defined in equation (2.25). The estimates of the parameters for a weighted least squares system are:

$$\hat{\boldsymbol{\theta}}_i = (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{Y} \tag{3.35}$$

$$\hat{\boldsymbol{\theta}}_i = (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \mathbf{\Phi}_i^T \mathbf{Q}_i (\mathbf{\Phi}_i \boldsymbol{\theta}_i + \mathbf{D}) = \boldsymbol{\theta}_0 + \tilde{\boldsymbol{\theta}}_i \tag{3.36}$$

where $\boldsymbol{\theta}_0$ indicate the 'true' parameters for the given structure, and $\tilde{\boldsymbol{\theta}}_i$ represent the differences. If $\mathbf{Q}$ is the unweighted form, then the differences $\tilde{\boldsymbol{\theta}}_i$ of the parameters can be found by taking the pseudoinverse of the design matrix $\mathbf{\Phi}_i$, otherwise the general form is:

$$\tilde{\boldsymbol{\theta}}_i = \hat{\boldsymbol{\theta}}_i - E\hat{\boldsymbol{\theta}}_i = (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{D} \tag{3.37}$$

where $\mathbf{R}$ is the covariance matrix of the residuals ($\mathbf{R} = E\mathbf{D}\mathbf{D}^T$),

$$\text{cov}\hat{\boldsymbol{\theta}}_i = E\tilde{\boldsymbol{\theta}}_i \tilde{\boldsymbol{\theta}}_i^T = (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{R} \mathbf{Q}_i \mathbf{\Phi}_i (\mathbf{\Phi}_i^T \mathbf{Q}_i \mathbf{\Phi}_i)^{-1} \tag{3.38}$$

in the unweighted case ($\mathbf{Q}_i = \frac{1}{N_i}\mathbf{I}$)

$$\text{cov}\hat{\boldsymbol{\theta}}_i = \sigma_0 [\mathbf{\Phi}_i^T \mathbf{\Phi}_i]^+ \tag{3.39}$$

where the covariance of the output signal (i.e. the measured errors at the output) is σ_0.

When using these methods to examine the local model's parameters it is important to remember that the covariance statistics here assume that the model structure is capable of modelling the underlying system, which will not be true in many cases.

3.3 The Effect of Normalisation of the Basis Functions

As described in Section 2.3.1, normalisation of the basis functions is sometimes desired because it results in every point in the input space being covered by the basis functions to the same degree, i.e. the basis functions form a *partition of unity* across the input space. In many cases the use of *normalised basis functions* has resulted in an improvement in performance. While the approximation capabilities of normalised networks have been demonstrated (Benaim, 1994), in (Shorten and Murray-Smith, 1994) we observed that the side-effects of normalisation had not been considered in detail by most authors. Normalisation is used by many authors, e.g. for RBF nets in (Moody and Darken, 1989) and (Jones et al., 1989), for local model nets in (Johansen and Foss, 1992a) and for fuzzy systems in (Takagi and Sugeno, 1985) and (Brown and Harris, 1994). Use of normalisation is most relevant for RBF nets, as other networks which partition the input space in an axis-orthogonal manner (e.g. B-Spline nets), can be designed to achieve a partition of unity without normalisation.

The output of a normalised basis function network (BFN) is described by taking the standard basis function network, described in equation (2.10), where the basis functions $\rho_k(\tilde{\phi})$ are normalised forms of a basis function $\rho(\tilde{\phi})$,

$$\rho_k(\tilde{\phi}) = \frac{\rho(d(\tilde{\phi}; \mathbf{c}_k, \sigma_k))}{\sum_{i=1}^{n_{\mathcal{M}}} \rho(d(\tilde{\phi}; \mathbf{c}_i, \sigma_i))}. \tag{3.40}$$

3.3.1 Side-effects of normalisation for the basis functions

Normalisation leads to a number of side effects other than the intended partition of unity described in Section 2.3.1, which can have important consequences for the resulting network.

Change of shape of basis function

Unnormalised networks usually use homogeneous basis functions, sometimes with differing widths. In normalised nets this is not the case – the shape of the basis functions is usually quite different from the un-normalised basis function, and the shape is influenced not only by the basis function's width, but also by the proximity of the other functions in the network. Note the decrease in basis function maxima in the normalised case shown in Figure 3.11. As the width of the basis function decreases the normalised network becomes less smooth, and tends towards a crisp nearest-neighbour classifier.

The basis function centred at $(0.6097, 0.0361)$ from the network in example 2 (see normalised contour plot shown in Figure 3.17) is shown in Figure 3.12 for both cases. Note the multiple peaks, reduced maximum and convoluted surface of the normalised version.

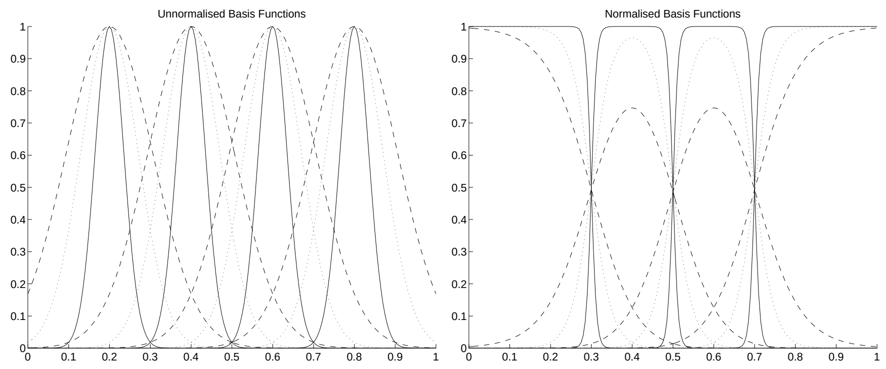
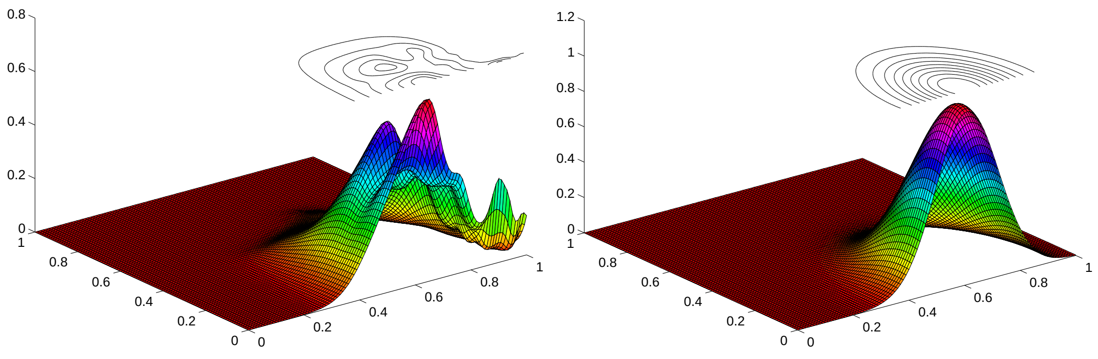


Figure 3.11: Change in shape due to normalisation. As the original functions become wider the normalised basis functions become less square and their maxima are reduced. Edge basis functions tend to unity as they move to the limits of their support.



(a) Normalised Basis Function (b) Unnormalised Basis Function

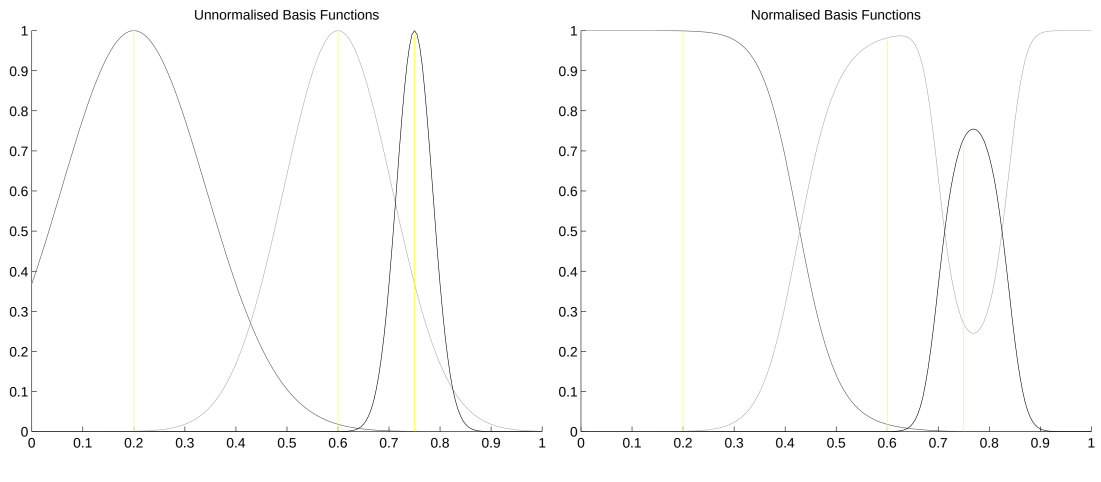
Figure 3.12: Effect of Normalisation on Basis Function Shape. The normalised basis function has a far more complex surface than before, with many local maxima.

Covering of the input-space

In the case where the basis function used is non-compact in nature, for example when Gaussians are used, then normalisation results in the *whole* of the input space being covered and not just the region of the input space defined by the training data. It can be seen from Figures 3.11 and 3.13 that in the normalised case the activation tends toward unity at the edges of the space. This can lead to unpredictable and often unstable behaviour in dynamic models if the operating point drifts outside the region of the input space that has been learned during training. It also reduces the ability of a basis function net to detect extrapolation using the methods described in Section 3.2.2.

Irregular networks: reactivation and shift in maxima

A further difficulty with normalised basis functions involves two further phenomena. If centres are not uniformly spaced, or if basis functions of differing widths are used, the maximum of the basis function may no longer be at its centre. A further effect of varying basis widths is that the basis function can become multi-modal, meaning that it can now also increase as the distance function increases, instead of continuously decreasing – the unit 'reactivates'. These effects are shown in Figure 3.13.



(a) Unnormalised Basis Functions (b) Normalised Basis Functions

Figure 3.13: Shift in maxima and reactivation. Note the reactivation of the centre basis function, the reduced maximum of the right hand basis function, and the shift in maximum for all three functions. The vertical lines show the positions of the basis functions centres to emphasise the centre-shift effect

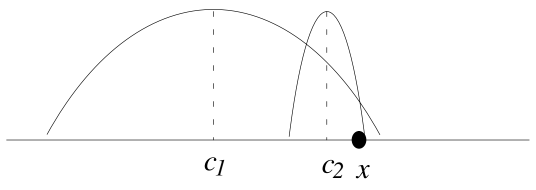


Figure 3.14: Reactivation example. The point in the input space x where the basis function reactivates can be determined from the units' centres (c_1 and c_2, where c_1 is furthest from the input x) and their widths (σ_1 and σ_2).

The reactivation occurs when neighbouring basis functions have differing widths. A one-dimensional example shown in Figure 3.14 using two basis functions illustrates how the phenomenon occurs. The reactivation point x, assuming monotonically decreasing basis functions, is the point at which the distance metric d_1 is no longer smaller than d_2, so it can be determined from the functions' centres and widths.

$$d_1 < d_2, \tag{3.41}$$

For a Euclidean distance metric,

$$\left(\frac{x-c_1}{\sigma_1}\right)^2 < \left(\frac{x-c_2}{\sigma_2}\right)^2, \tag{3.42}$$

$$\frac{\sigma_2}{\sigma_1} < \frac{\mid x-c_2 \mid}{\mid x-c_1 \mid}. \tag{3.43}$$

Equation (3.43) shows that reactivation only occurs when the ratio between σ_1 and σ_2 is less than the ratio of the unweighted distances from the centres. This implies that in networks with uniformly wide basis functions, reactivation cannot occur. The shift in the position of the activation function's maximum occurs when neighbouring basis functions are either unevenly spaced or have differing widths.

This behaviour can cause problems if the network is being used to estimate an underlying probability distribution as is the case when local linear models are being used to approximate the function (Johansen and Foss, 1992c) (Johansen and Foss, 1992b). Within this framework, reactivation can lead to models becoming significantly active in regions in which they were never intended to operate. The examples in this section demonstrates the effect of this on a regression problem.

Effects of normalisation on multi-dimensional problems

The effects of normalisation can become more pronounced as the input dimension increases. Due to the increased number of neighbouring basis functions in higher dimensions, the cumulative activation in a given region tends to increase with dimension, leading to normalised basis functions often having dramatically reduced maxima. Note also that the difference between the normalised radial and ellipsoidal basis functions is less extreme than the difference between the original functions – in many cases normalisation makes the use of more complex distance metrics less significant. See Figures 4.6 and 4.7 on page 93.

3.3.2 Effect of normalisation on optimal network parameters

The robustness of a trained network is closely related to the magnitude of the basis function parameters. For example, with noisy data, large weights can cause potentially large errors or even instability. This section examines what happens to the *least squares* weight solution for the estimated weights $\hat{\boldsymbol{\theta}}$, when a given basis function network as defined by the network structure parameters $\mathcal{M} = (n_{\mathcal{M}}, \rho_{1\ldots n_{\mathcal{M}}}(\cdot))$, is normalised.

We consider the system described by equation (2.10) where the exact form of the basis function is defined by the activation function used and whether it is normalised or not. We also assume that the output observations are all positive[6]. Equation (2.23) can be decomposed to include a matrix $\mathbf{C}$

$$\hat{\mathbf{Y}} = \mathbf{C}\boldsymbol{\Phi}\boldsymbol{\theta}, \tag{3.44}$$

where N is the number of observations, $\boldsymbol{\Phi}$ is the Nxn_ϕ design matrix of basis function activations from the training set, $\boldsymbol{\theta}$ is the n_ϕx1 vector of weights and $\mathbf{C}$ is a NxN positive definite diagonal matrix (this assumes basis functions which are positive for all of their support). In the unnormalised case $\mathbf{C}$ is simply the identity matrix, while in the normalised case $\mathbf{C}$'s entries are given by

$$c_{kk} = \left(\sum_{i=1}^{n_{\mathcal{M}}} \rho_i(\boldsymbol{\psi}_k)\right)^{-1} \qquad \forall \;\; 1 \le k \le N. \tag{3.45}$$

Then the normalised output $\bar{\mathbf{Y}}$ is,

$$\bar{\mathbf{Y}} = \mathbf{C}^{-1}\hat{\mathbf{Y}} = \boldsymbol{\Phi}\boldsymbol{\theta}, \tag{3.46}$$

since $\mathbf{C}$ is invertible. Therefore the solution to equation (3.46) can be written,

$$\boldsymbol{\theta} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{C}^{-1}\hat{\mathbf{Y}} \tag{3.47}$$

which can be written

$$\boldsymbol{\theta} \;\; = \;\; \mathbf{J}\mathbf{C}^{-1}\bar{\mathbf{Y}}, \tag{3.48}$$

where $\mathbf{J}$ is an n_ϕxN matrix and assuming that the inverse $(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}$ exists. Expanding equation (3.48) yields,

$$\boldsymbol{\theta}_i = j_{i1}c_{11}^{-1}y(t_1) + \cdots + j_{in_{\mathcal{M}}}c_{n_{\mathcal{M}}n_{\mathcal{M}}}^{-1}y(t_{n_{\mathcal{M}}}), \tag{3.49}$$

[6]In practice this is not a restriction since the output can be normalised to lie in the interval (0,1) during the pre-processing stage. At the output the inverse operation can be carried out.

where $\boldsymbol{\theta}_i$ denotes the ith normalised weight, j_{mn} is the mn entry of $\mathbf{J}$. After some manipulation the following inequality can be obtained from equation (3.49),

$$c_{min}^{-1}\bar{\boldsymbol{\theta}}_i \leq \boldsymbol{\theta}_i \leq c_{max}^{-1}\bar{\boldsymbol{\theta}}_i, \tag{3.50}$$

where c_{max}^{-1} and c_{min}^{-1} are the maximum and minimum entries of the $\mathbf{C}^{-1}$ matrix and $\bar{\boldsymbol{\theta}}_i$ denotes the ith weight under the constraint that the $\mathbf{C}$ matrix is the identity matrix, i.e. the network is not normalised.

Equation (3.50) indicates that the magnitude of optimal weights may be increased or decreased after normalisation of the basis functions. An increase in weights typically occurs when the widths are large (as $\sum_{i=1}^{n_{\mathcal{M}}} \rho(d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i))$ is then also large), whereas a decrease in weight magnitude tends to be associated with small widths. In multidimensional cases, the effect of large basis functions becomes even more dramatic, for the reasons described in Section 3.3.1. It is therefore important not to normalise blindly, but to compensate for the normalisation by altering the design criteria for the structure (centre positions and width magnitudes) identification procedure.

Example 1: Modelling a pulse function

A simple one-dimensional example is used to illustrate the effects of normalisation – a pulse function defined between 0 and 1, which is 1 for $0.2 < x < 0.7$ and zero elsewhere. The training data consists of 100 points spread uniformly throughout the input space. Five basis functions (centred at [0.15 0.25 0.45 0.65 0.75]) are used to model the function. The model's approximation of the target function is shown, with the associated basis functions.

The example in Figure 3.15 shows the reactivation of the large basis function in the normalised case. This forces the model to 'sag' in the middle, as the edges of the basis function are far removed from the main part, and the approximation becomes an average of far removed areas of the input space. The approximation is locally flatter than the non-normalised case, because of the ability of the normalised model to approximate constant surfaces.

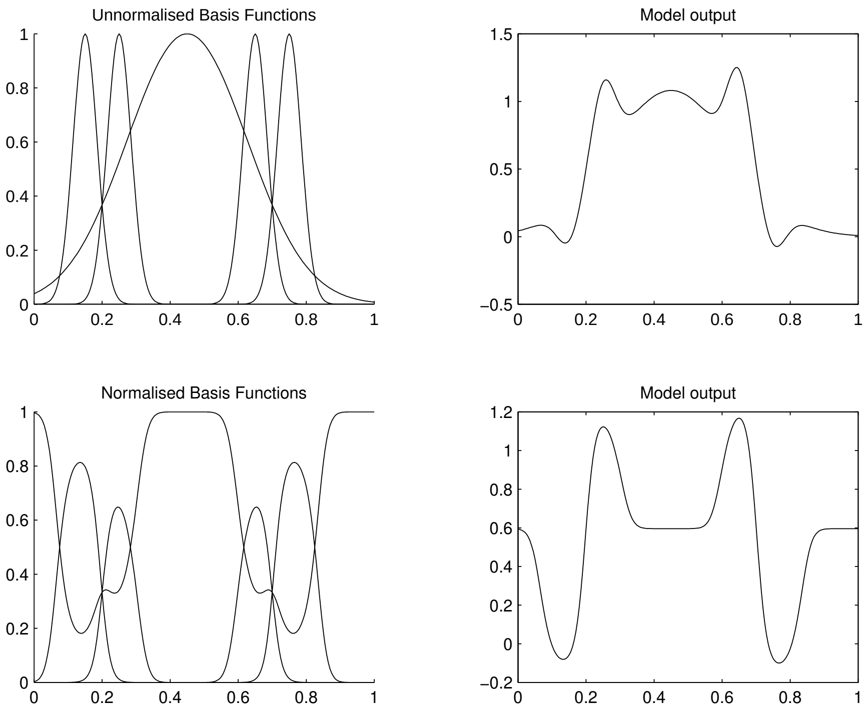
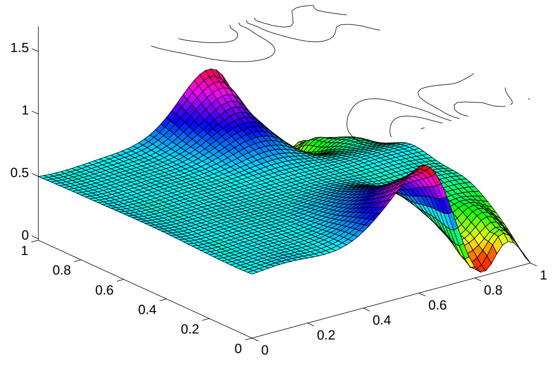


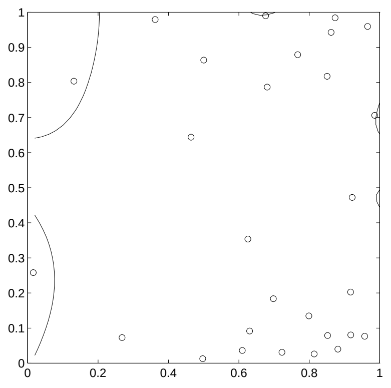
Figure 3.15: Normalisation example. The pulse function is modelled with and without normalisation. In the normalised case, the reactivation of the centre basis function is clearly visible, leading to the sag in the middle of the model output plot. Note the unnormalised model's inability to model the flat area of the pulse, due to the lack of a partition of unity.

Example 2: Modelling a two dimensional Function

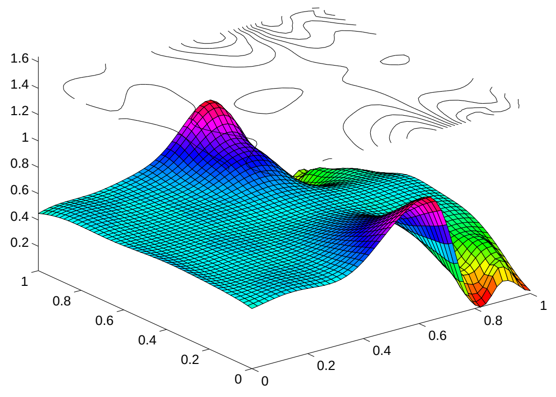
Figures 3.16 and 3.17 show the effect of normalisation on the representation of a 2-D function with basis functions which have little overlap, and those with greater overlap (the σs are twice as large in Figure 3.17). The response plots were created using the same basis functions, but the weights were trained individually for the normalised and unnormalised cases. Note the relative robustness of the normalised network to the change in width parameters, whereas the unnormalised network provides a very poor representation with the smaller basis functions.



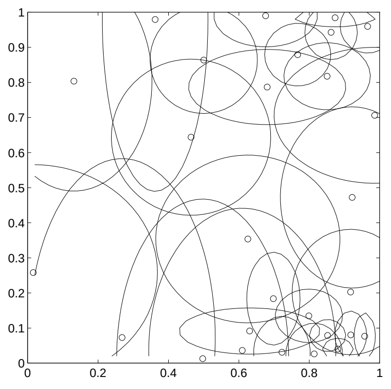
(a) Normalised Model Output



(b) Normalised Basis Functions

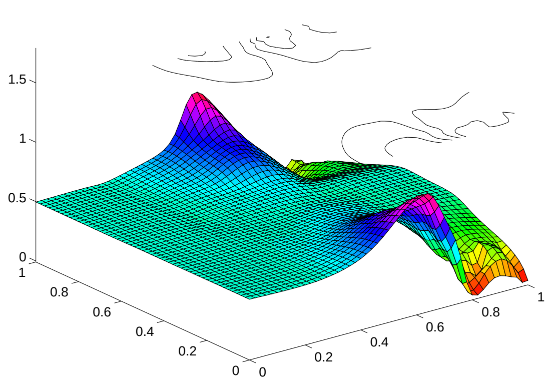


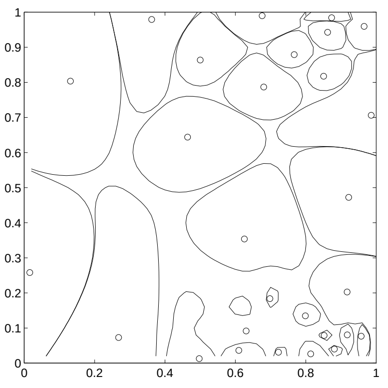
(c) Unnormalised Model Output

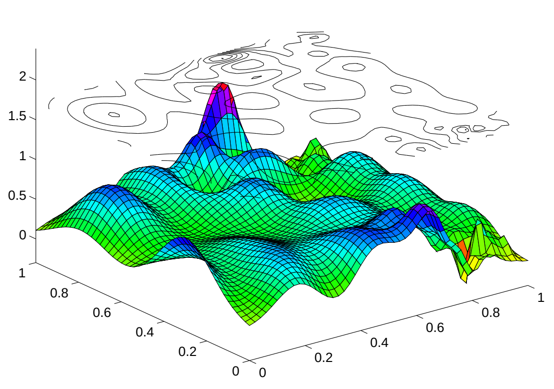


(d) Unnormalised Basis Functions

Figure 3.16: Effect of Change of Shape on Model Representation – Wide BFs. Both models produce a good approximation of the target function. Compare this to Figure 3.17, where the narrower basis functions lead to a large difference for the unnormalised model.


(a) Normalised Model Output


(b) Normalised Basis Functions


(c) Unnormalised Model Output

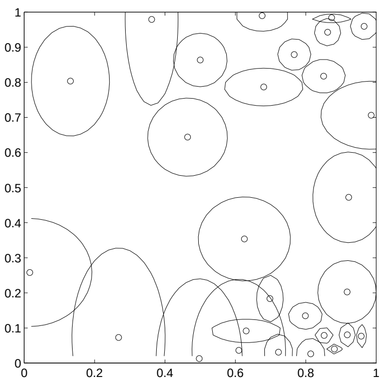

(d) Unnormalised Basis Functions

Figure 3.17: Effect of Change of Shape on Model Representation – Narrow BFs. The Unnormalised model is far less robust to changes is the basis function size, as shown by its inability to model the function. The normalised model performs only slightly worse than in Figure 3.16.

3.4 Conclusions

3.4.1 Local learning as a robust optimisation algorithm

Two methods (global and local learning) for optimising the local model parameters using singular value decomposition were compared. The problems involved in a global optimisation were examined. These are: the drastic increase of effort with increasing numbers and complexity of local models, and the ill-conditioning common to such regression problems. This leads to non-robust, and poorly interpretable models.

The advantages of the new local learning methods are:

- The analysis of the computational complexity shows that local learning is faster than global learning (linear increase with number of local models, as opposed to cubic in the number of parameters). This speeds up the learning process significantly.
- Local learning can be seen as a simple form of regularisation, meaning that the local methods often produce models with higher accuracy, and greater robustness than globally optimised methods (especially in noisy, poorly populated or high dimensional problems), without having to resort to expensive cost functionals.
- A further point is that the parameters found for locally trained networks can be more interpretable as local approximations to the real system than those for globally trained ones. Globally trained local models cannot be meaningfully examined without taking account of neighbouring local models.
- The structure also allows more flexibility in the use of optimisation algorithms, which will be especially useful with heterogeneous local model networks which require a variety of nonlinear optimisation algorithms for the different local models.

In general, the experience gained during this work indicates that local learning will tend to be better than global learning when there is insufficient training data, or noisy training data. Global learning will tend to do well when faced with smooth underlying nonlinearities in low dimensional spaces well populated by training data.

Future work

There are a number of interesting aspects which are still to be fully investigated in local learning. The relationship between local learning and other regularisation methods (e.g. the smoothing splines work in (Hastie and Tibshirani, 1990)) may provide insight, and could lead to the application of the ideas in other structures. The application of local methods in adaptive control is obviously interesting for on-line parameter adaptation, with minimal interference with other areas of the model.

3.4.2 Local confidence limits

The local confidence interpolation methods described in this section are simple applications of the local modelling philosophy to the model validation stage, which produce more significant measures of model accuracy than global methods, and allow the local use of many conventional statistical methods. The dangers of extrapolation were pointed out, with the observation that the normalisation of basis functions makes many of the extrapolation detection techniques from the literature inapplicable to normalised nets. The use of well known methods for the estimation of variance in the parameters of local model nets was described.

Apart from being important for the interpretation and validation of trained networks, local confidence estimates can be used to improve the learning process itself, by directing the search for a better model structure to those areas with the poorest confidence limits. The model structure identification algorithms in Chapters 4 and 5 are based on such ideas.

Future work

The methods described here basically interpolate local error estimates to produce state-dependent global ones. There is obviously plenty of room for the integration of a great variety of statistical tests existing for linear systems into this framework. More direct use of the confidence limits for the detection of model structure mismatch is obviously interesting for use in structure identification algorithms, as are better methods for validating the dynamic aspects of the local models.

The methods used in this work for the prediction of model error were fairly straightforward applications of the local model philosophy, but they do not consider the effect of noise on the inputs at run-time. This can vary during operation, or in different areas of the input space. It is also possible that some sensors could fail totally. It is obviously desirable for a model to continue to operate, while having poorer confidence in its results.

3.4.3 Effects of basis function normalisation

Phenomena which occur in basis function networks when a partition of unity is achieved by normalisation of the basis functions were discovered and analysed. These effects can be summarised as follows:

1. Normalisation leads to a change in shape of basis function. This can lead to a loss in smoothness of representation if the widths of units are too narrow (i.e. little overlap). If there is a great deal of overlap between units, the maxima of the model validity functions are drastically reduced.

2. If non-compact basis functions (e.g. Gaussians) are used normalisation leads to the whole of the input space being covered. This is important,as it makes some extrapolation detection techniques unusable, and it seriously affects the stability of dynamic basis function models.

3. For irregular networks the maxima of the units shift away from the centres, and the units can reactivate in other parts of the input space. Reactivation, and the resulting non-localised behaviour of individual basis functions means that the very motivation behind much of the work carried on local RBF nets, i.e. localised behaviour, is no longer guaranteed.

4. Normalisation also affects the magnitude of the 'optimal' parameters $\boldsymbol{\theta}$ for a given network. This can subsequently affect the robustness and stability (for dynamic systems) of the network, depending on the level of overlap before normalisation.

5. The above effects become even more pronounced as the input dimension increases, due to the larger number of neighbouring basis functions.

While partitioning unity is highly desirable for many modelling applications, these phenomena, or side-effects, can lead to unpredictable network behaviour. It is therefore important that researchers and users of local model nets, BF nets or fuzzy systems should consider these effects when designing both networks and training algorithms, and when interpreting and validating trained networks.

Future work

It is important that the effect of normalisation be better understood, especially with respect to the stability of dynamic models built with basis function networks. Many aspects of basis function nets and local model nets rely on the locality of the basis functions. The results presented in this chapter may lead to the development of new training algorithms which try to minimise the side-effects. The significance of the results here should also be noted in the related fields of fuzzy logic and in the recent models based on mixtures of probability density functions.

Chapter 4

Structure Identification in Local Model Networks

This chapter describes a novel constructive structure identification algorithm for local model networks which gradually adds to the model structure by placing extra representation in 'complex' areas of the input space. The search for 'complexity' is repeated at ever decreasing scales, as far as the training data will allow. A new method for estimating ellipsoidal distance metrics for the basis functions is described. To prevent structural overfitting, stopping and pruning criteria are developed. Local model structure selection methods are also suggested

Active selection methods which construct a training set automatically are introduced. The algorithms described allow the training set to be constructed in step with the model structure, selecting the most suitable training data for the given problem, and available training data.

The algorithms are demonstrated on several static and dynamic test systems.

4.1 Constructive Structure Identification

The structure identification problem was introduced in Section 2.5.3, and the previous methods were reviewed. As stated, the goal of the structure identification procedure for local model nets is to provide a problem-adaptive learning scheme which automatically relates the density of basis functions, the associated local model structures and the size of their receptive fields to the local complexity and importance of the system being modelled. The aim is to find a model structure $\mathcal{M}$ which allows the network to best minimise the given cost function in a robust manner. Minimising $J^*(\mathcal{M}, \mathcal{D})$, from equation (2.20), over the possible model structures leads to the 'super-optimal' cost, using *a priori* knowledge about the process structure $\mathcal{K}_{\mathcal{S}}$,

$$J^{**}(\mathcal{D}, \mathcal{K}_{\mathcal{S}}) = \min_{\mathcal{M}} J^{*}(\mathcal{M}, \mathcal{D}). \tag{4.1}$$

This is a general statement of the problem, and much of the work in the area of structure identification formulates the task as an explicit search or optimisation problem where the task is to find the structure associated with the minimum of the given cost function. The construction becomes a multi-step process: At each step various options are constructed, the model parameters optimised, and the structure with the best cost-complexity value chosen. The procedure is then repeated at the next stage of construction. Unfortunately the search space is usually very large and such algorithms therefore suffer from the 'curse of dimensionality' and scale up badly to larger problems. This becomes even more acute when multiple search paths are followed, where not only the best option is chosen at each step, but several of the best model structures are chosen and used for future steps.

To produce efficient, practical algorithms the following observations about the modelling problem should be noted:

1. Although highly desirable, the distribution of training data will probably not be directly related to the complexity of the observed process.
2. The process will probably have varying levels of complexity throughout different areas of the input space.
3. The training data will not be uniformly distributed, and because of the physical constraints inherent to the process being modelled, there will be areas of the input space which *cannot* be filled with data.

These points have a number of implications for constructive algorithms for local model nets. The first point implies that we should consider the local complexity of the process output in the improvement of model structure, as opposed to unsupervised learning techniques, which only consider the density of the input data, regardless of the output response. The second point implies the need for a multi-resolution technique which will find model structure representing varying volumes of the input space (varying levels of 'locality'). The last point lets us reduce the volume of the input space we consider for new local models to only points covered by the available training data.

4.1.1 The constructive approach

The constructive approach starts off with a simple model. The parameters are then estimated, and the model validated. The validation helps determine the areas of the input space where the representation is still unsatisfactory so that new degrees of freedom can be added to the model in these areas. This generally leads to a sequence of model structures of increased representational ability (i.e. increasing degrees of freedom in the model structure, although sometimes the model structure may be reduced). The basic algorithm for a constructive structure identification algorithm is therefore:

1. Initialise model structure using *a priori* knowledge.
2. Estimate model parameters from training data.
3. Validation (Determine model quality, and where model structure most needs improvement)
4. Improve model structure, if necessary and feasible, given the available data. (Can involve an increase or decrease in degrees of freedom)
5. Goto Step 2 if validation unsatisfactory.

Constructive techniques which gradually enhance the model representation in this manner have a number of advantages.

- The network first allocates representation where most needed, according to the complexity heuristic. The main features of a process are captured first, then the details. This is an implicit style of regularisation, as the model construction process can now be seen as a gradual increase in variance and decrease in bias. Learning continues until the desired level of bias-variance trade-off is achieved.
- Modelling accuracy and generalisation ability tend to be improved, as the model structure is extended as a far as possible to fit the data, while the overfitting protection inherent to the constructive algorithm limits overtraining.
- *A priori* knowledge can be introduced in the form of a pool of local model structures, so that the local model structure best suited to a local area of the input space is chosen. This automatically creates a heterogeneous model structure.
- The proportion of the available training data used can also be selectively extended during learning to have a density matching the density of the basis functions, improving the quality of the parameter estimates, and speeding up the learning process. If there is insufficient data in certain areas of the input space, the constructive algorithm can be linked to an *active learning* procedure which interacts with its environment to obtain more information.

The constructive procedure is illustrated in Figure 4.1 where the model complexity is increased gradually for a two-dimensional function approximation problem. The algorithm used to construct this network is the Multi-Resolution Constructive (MRC) algorithm for local model nets, described in the next section.

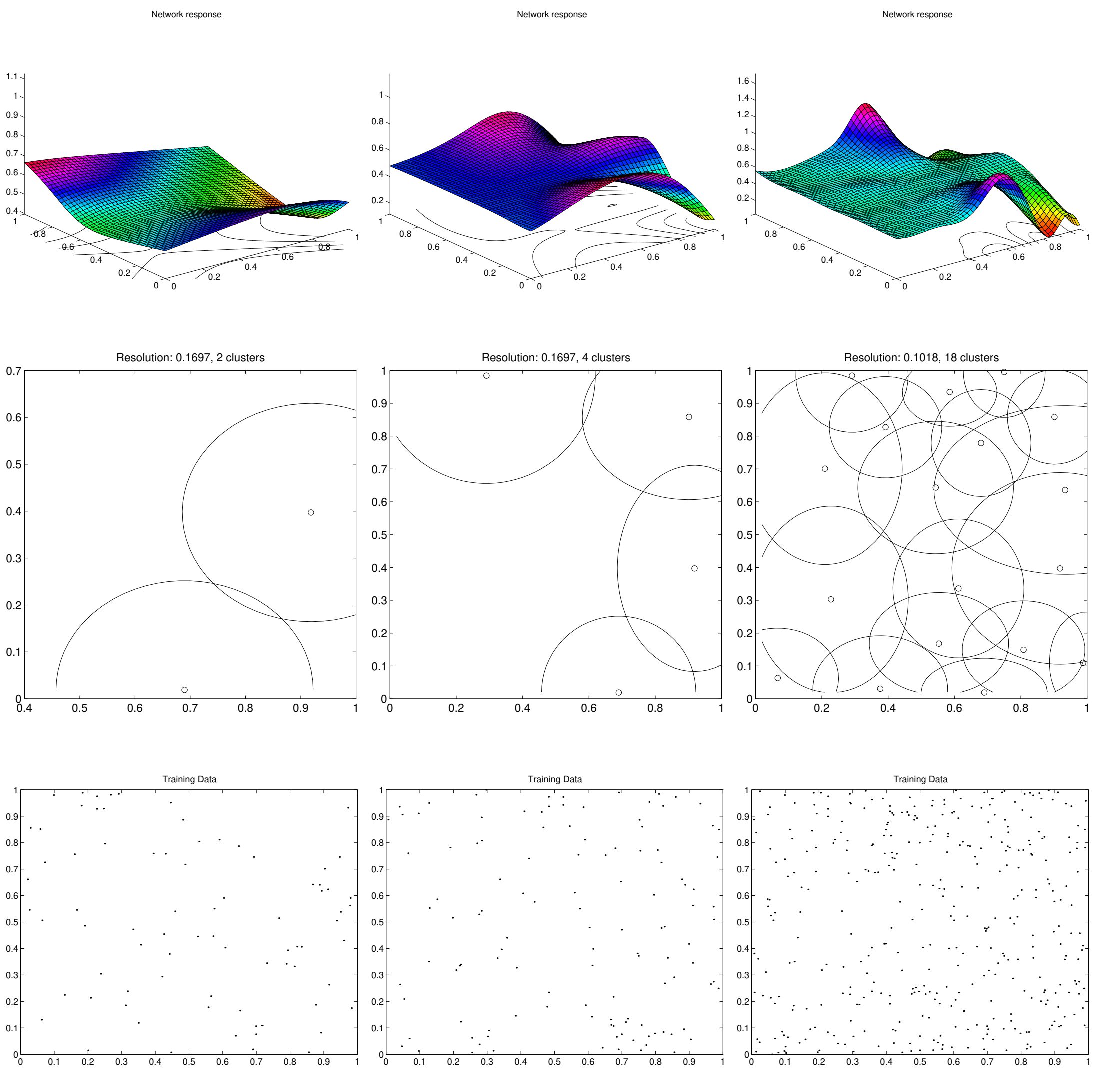


Figure 4.1: A gradual approach to constructing a model. The model responses are shown for a series of stages in model development, shown by the contour plots of the basis functions. The bottom row shows the training data used – note that the set is expanded with the model structure. The target function is that shown in Figure 3.9 on page 68.

4.2 The Multi-Resolution Constructive Algorithm

This section describes the Multi-Resolution Constructive Algorithm, which differs from the existing constructive algorithms in a number of ways. The conventional methods tend to basically apply search techniques to find a model structure which minimises the cost-complexity functional. The method used here is to use a 'model mismatch' or 'complexity heuristic' to indicate the areas of the input space with the worst errors, or greatest complexity and to develop the model structure in this area. The options for model structure extension are also drastically reduced by restricting the possible positions of basis function centres to be on input points in the training set. This leads to a relatively simple method for extending the model structure which expends its effort in predicting where new structure would be useful, rather than trying out the options and selecting the best of them. The process is described below:

1. The model starts off with a minimal representation (perhaps only one linear model, depending on the state of the *a priori* knowledge) and searches for 'coarse' complexity. It refines the model structure at ever increasing levels of resolution until the desired accuracy has been achieved, or the training data has been exhausted.
2. To determine where to add the extra representation the 'complexity' heuristic is needed. This decides where new models should be placed, based on a weighted local statistic of the training data, or from measured model residuals. To enforce the gradual nature of the approximation the new centres must be a minimum distance d_{min} from existing centres.[1]
3. Given the suggested location of the new model centre the desired overlap with neighbouring regions is determined, thus completing the basis function optimisation for this stage of the model construction.
4. If a pool of local model structures has been defined, the best fitting local model for each basis function can be chosen by estimating the local model parameters for the new model structure and running cross-validation runs. If the receptive field of any given basis function has too few units to reliably estimate the associated local model parameters it can be removed (and Step 3 is repeated), or the local model structure simplified.
5. If the model is still not accurate enough, the search for the next most 'complex' area of the input space is then restarted. This is repeated until either no further local models can be added, or the added models do not bring any improvement, whereupon the scale of the 'complexity window' is reduced, and the search is restarted at the finer resolution.

The details about each of these stages are described in the following sections.

[1] d_{min} is related to the current resolution of search σ_{win}.

4.2.1 Scheduling the multi-scale search for complexity

The search principle is shown in Figure 4.2, where the radial complexity window described in Section 4.2.2 is applied at different scales, starting off by searching for coarse complexity and refining the search as learning progresses. The resolution of the window σ_{win} is reduced by a factor of λ each iteration, starting at an initial size of σ_{max}.

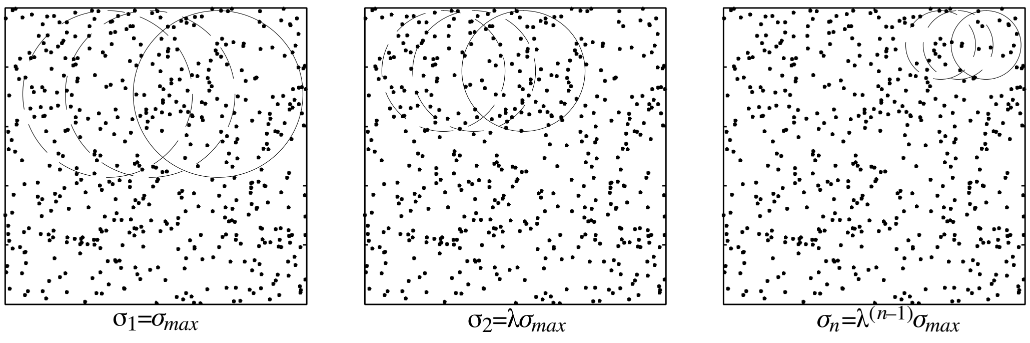

$\sigma_1=\sigma_{max}$
$\sigma_2=\lambda\sigma_{max}$
$\sigma_n=\lambda^{(n-1)}\sigma_{max}$

Figure 4.2: Multi-Resolution Windowing. The complexity measure is applied at different scales, and at each scale σ_i the 'window' is centred on points from the training set. The points corresponding to the greatest values of the complexity measure are those suggested as new centres for basis functions.

As new models must be a certain minimum distance away from previous models, the input space will gradually be filled with basis functions, the density being determined by the current resolution. Once no more basis functions can be inserted at this resolution the algorithm moves on to the next, finer search stage. To prevent non-complex areas of the input space being unnecessarily filled with local models, the search at a given resolution is abandoned if over a window of n_{cutoff} successive insertions no improvement in mean cross-validation error is made ($n_{cutoff} = 4$ was used for the experiments in this work).

Choice of algorithm parameters

In general, the algorithm worked well on a range of problems without the need to fiddle with the parameters. Once the initial model was built, the user could alter parameters to try and improve the approximation, but the initial results were usually satisfactory.

The choice of initial search resolution σ_{max}, reduction rate λ and number of window resolutions n_{res} involves the usual trade-offs in parameterisation. A large number of iterations will make the process more robust, but also more expensive. Having λ too small, or starting with a small σ_{max} will lose the multi-resolution nature of the algorithm, as the resolution jumps from a coarse level to a relatively fine window. Setting these parameters to be too large will increase the number of iterations and the computational cost.

The scaling factor γ is used to determine the minimum distance a new centre has to have from existing centres $d_{min} = \gamma\sigma_{win}$. This has an effect on the manner in which complexity is increased during learning. The σ_{win} defines the size of the complexity window being used, but if γ is too small, the gradual nature of the approximation will also be lost, as the algorithm will place too many units in the complex areas of the input space (where 'complex' is defined for the current level of search), at the cost of other areas.

The variance in conventional search-style algorithms, such as decision trees, which recursively partition the input space comes from the effect of making a non-optimal decision early in the construction process which has a serious effect on following stages of learning, as the amount of available data has been drastically reduced. This is less of a problem in the multi-resolution constructive algorithm because the multi-stage nature of the algorithm brings added robustness. From experimental experience, the learning algorithm does not seem to be overly sensitive to early decisions, whereas in decision trees this is not so. The variance in network size tends to be due to the decision to stop construction at a given level.

4.2.2 Complexity detection – where are extra units needed?

The multi-resolution cluster algorithm uses a 'complexity detection' heuristic to place new local models. The heuristic was inspired by the *Vector Field Approach to Cluster Analysis* (Andrews, 1983). Observation 3 above indicates that we should simplify the search task by assuming that the training data covers the significant areas of the input space adequately for the initial search (an assumption which must be true for any degree of learning to take place). Initially, all training points which are a minimum distance $d_{min} = \gamma\sigma_{win}$ from existing centres are viewed as possible centres $\mathbf{c}_{new}$ for the new basis function. The centre $\mathbf{c}_{new} \in \mathcal{R}^{n_{\tilde{\phi}}}$, and the 'complexity' of the mapping in a windowed area, where $\rho(\cdot)$ is the windowing function[2], around this point is measured using

$$F_{total}(\mathbf{c}, \sigma_{win}) = \frac{1}{N}\sum_{i=1}^{N} \rho(d(\tilde{\boldsymbol{\phi}}_i, \mathbf{c}, \sigma_{win}))e_i, \tag{4.2}$$

where N is the number of neighbouring data points used, e_i can be a general error statistic[3], but which is often simply

$$e_i = |y_i - \hat{y}_i|\,. \tag{4.3}$$

The function $d(\cdot)$ is a distance measure. The complexity is estimated by an analogy to the concepts of forces acting on a mass in physics. The weighting of the forces depends on their associated error statistic (e_i). The windowing function focusses the heuristic's attention on the level of locality currently being examined. The larger the level of F_{total}, the larger the estimated complexity.

[2]In this work a Gaussian bell was used, as defined in equation (2.13).

[3]If pruning is not used it can make sense to use $e_i = |\Delta y_i||e_i|$, where $\Delta y_i = y_i - y_{centre}$, as this prevents the system allocating nodes in relatively simple areas of the input space, purely because the initial windowing size is too coarse to capture the finer model structure.

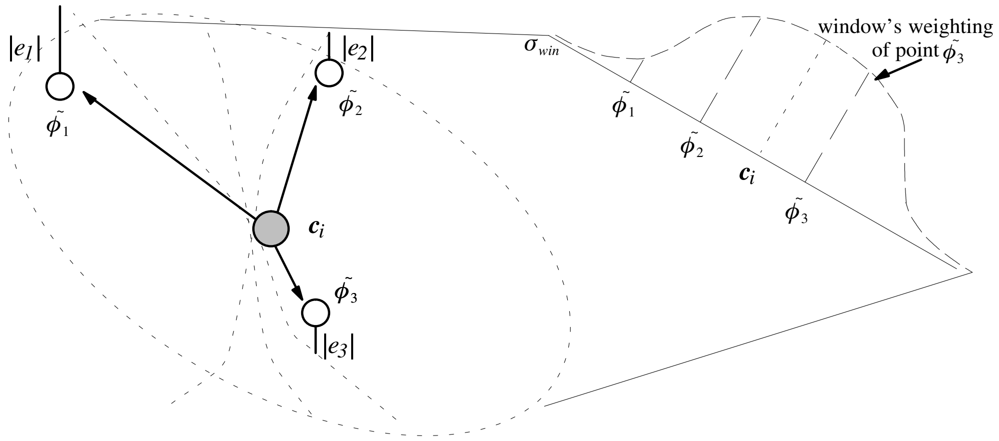


Figure 4.3: Windowed Complexity Estimate. The weighting function $\rho(\cdot)$ weights the error measures e_i at points $\tilde{\boldsymbol{\phi}}_i$ around the prototype centre. The window function is usually chosen such that points further from the centre have less effect on the outcome of the complexity estimate.

The major disadvantage of the complexity heuristic is its computational load. It can require a maximum of N^2 calculations of the weighting function and the associated offset for each prototype centre. However, as extra basis functions are added for any given resolution, the number of potential centres sinks, due to the distance constraint d_{min} covering a higher percentage of the training points. This means that the search for complexity gets faster as the model grows. The computational effort can also be reduced by limiting the search by using only a subset of the training data as potential centres by applying active selection. Hierarchical decomposition of the input space also limits the number of data points under consideration, as described in Chapter 5.

4.2.3 Overlap determination

Given a set of basis function centres we need to define the basis function widths $\boldsymbol{\sigma}$, which determine the level of overlap between neighbouring local models. When basis functions are used where centres can be distributed unevenly throughout the input space finding the 'correct' degree of overlap is a difficult problem. The conventional method is to set the radius σ_j proportional to the average distance of the k nearest neighbours from the centre $\mathbf{c}_j$,

$$\sigma_j \propto \frac{1}{k}\sum_{i=1}^{k} ||\mathbf{c}_j - \mathbf{c}_i||. \tag{4.4}$$

In many cases this will be unsatisfactory, as the immediately neighbouring units could be widely varying distances apart in different directions, meaning that the resulting level of overlap with the neighbouring basis functions would vary greatly. Too much overlap leads to problems with poor estimation and singularities in the regression process. With too little overlap and normalisation the mapping loses smooth interpolation between local models, and the behaviour further away from the centre becomes unpredictable, because of the interaction with other basis functions. In Section 3.1.1 it was shown that the condition of the design matrix is highly dependent on the level of overlap between basis functions. More powerful distance metrics such as the ellipsoidal distance metric in equation (4.5), where $\boldsymbol{\sigma}_i$ is a positive definite square matrix, as in (Poggio and Girosi, 1990, Röscheisen et al., 1992) can be used to provide more flexible basis functions

$$d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i) = \left|\tilde{\boldsymbol{\phi}} - \mathbf{c}_i\right|^T \boldsymbol{\sigma}_i{}^{-1} \left|\tilde{\boldsymbol{\phi}} - \mathbf{c}_i\right|, \tag{4.5}$$

which potentially allow a more even level of overlap with the neighbouring units. The overlap optimisation problem is therefore to robustly determine the matrix $\boldsymbol{\sigma}$ for each basis function from the distance of the neighbouring units.

Use of a 'covariance' heuristic

To make a fast estimate of $\boldsymbol{\sigma}_i$, a heuristic is used which calculates the 'covariance' of the set of centres $\mathbf{c}_n$ surrounding the chosen centre $\mathbf{c}_i$ (which serves as the 'mean' in the calculation). The inverse of the 'covariance' estimate can then be used to define the distance function, as used in equation (4.5):

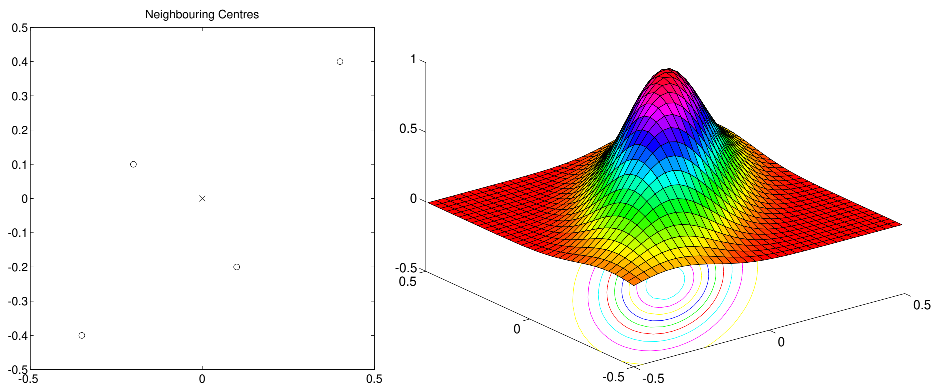
$$\boldsymbol{\sigma}_i \propto E_n\left[(\mathbf{c}_n - \mathbf{c}_i)(\mathbf{c}_n - \mathbf{c}_i)^T\right], \tag{4.6}$$

where E_n is the expected value over the set of chosen neighbouring centres $\mathbf{c}_n$. The results of this technique are shown in Figure 4.4.

The problem with this technique is that we are interested in estimating the distance to the nearest neighbours in *all* directions around the centre. It is not sufficient to take the k nearest, as they could all be in the same area. To avoid this, the algorithm demanded a minimum angle between neighbouring centres before they were included in $\mathbf{c}_n$ to be used for the covariance estimate, as shown in Figure 4.5.

Spherical, ellipsoidal or axis-orthogonal ellipses?

The ellipsoidal basis functions demand an extra $\frac{n(n+1)}{2}$ free parameters, and the variance inherent in this increase in parameterisation is not trivial. To reduce the variance we used singular value decomposition (SVD) to determine the inverse of the 'covariance' matrix $\boldsymbol{\sigma}$. This allows us to apply simple regularisation ideas to the distance metric by altering the singular values, and thus reducing the degrees of freedom in the final distance metric. In our experiments, however, we found that the minimal improvement in performance compared to the simple RBF net was usually not enough to justify the use of fully parameterised ellipsoidal distance metrics. More restricted axis-orthogonal elliptical basis functions, where $\boldsymbol{\sigma}$ is a diagonal matrix, can be easily derived by only considering the diagonal terms of the covariance



(a) Neighbouring centres (b) Basis Function

Figure 4.4: Using the 'covariance' measure to determine the basis functions' size and orientations. Neighbouring points are found and the desired overlap is estimated. 'x' is the centre of the unit being adjusted, and the 'o' points are the centres of neighbouring units. The surface plot on the right shows the shape of the resulting basis function.

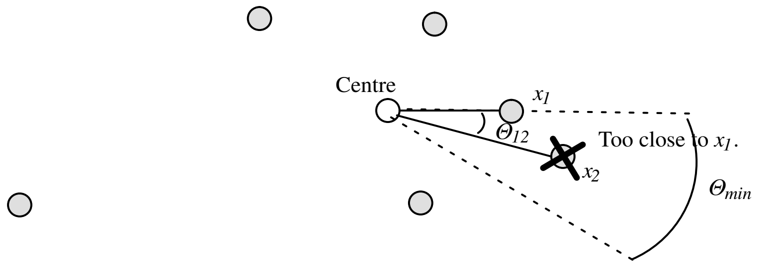


Figure 4.5: Eliminating centres with a common direction. The neighbouring points are used to define a 'covariance' matrix which is used to determine the overlap between neighbouring units. Neighbours in the same direction are ignored.

matrix. The use of these simplified distance metrics tended to lead to networks with higher accuracy and better generalisation. Figures 4.6 and 4.7 show the contours of the basis functions found for a two-dimensional problem with the given centres, using the three styles of distance metric with and without normalisation. Note from Figure 4.7 that when the basis

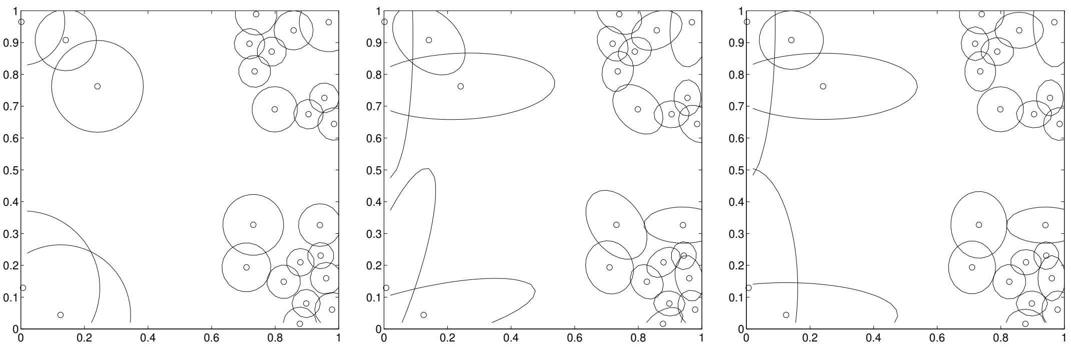


Figure 4.6: Radial, Ellipsoidal and Axis-orthogonal ellipsoidal basis functions. Contours are drawn at a level of 0.5.

functions are normalised, there is often little difference between the various options.

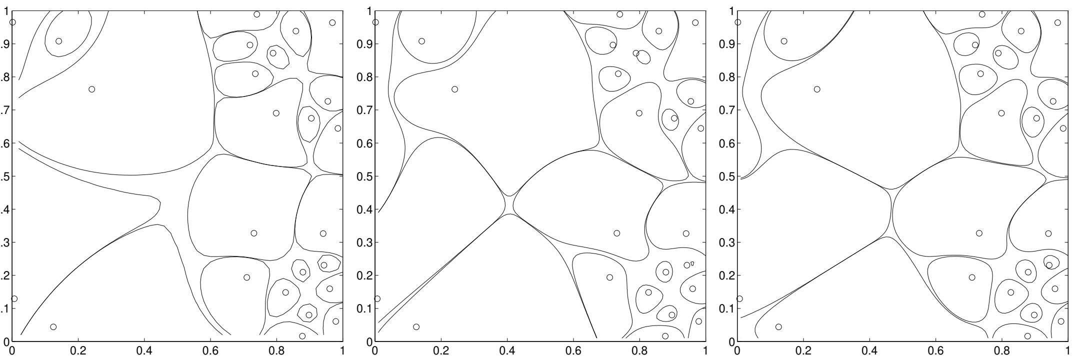


Figure 4.7: Normalised Radial, Ellipsoidal and Axis-orthogonal ellipsoidal basis functions. Contours are drawn at a level of 0.5.

The heuristic method described in this section could be seen as a fast initialisation routine, which could be further optimised by iterative gradient techniques, as in (Poggio and Girosi, 1990).

4.2.4 Preventing overfitting

The problem of differentiating between errors due to noise on the training data and errors due to bias caused by an inadequate model structure is often a difficult one. Overfitting is the result of extending the structure so far that noise is learned, instead of system structure. In Section 2.5.1 we discussed ways of reducing the variance in the parameter estimation phase. It is also possible to reduce variance by limiting the model structure, which can be done by stopping growth, pruning structure, or careful selection of local model structures.

Stopping model growth

A simple constraint on the structure identification algorithm is to require a minimum number of data points within the receptive field of a given local model for it to be considered viable, i.e. only if $\rho_{total_l} > N_{min}$, where N_{min} is the minimum number of training points needed and

$$\rho_{total_l} = \sum_{i=1}^{N} \rho_l(\tilde{\boldsymbol{\phi}}_i) \tag{4.7}$$

is a heuristic 'count' of the local data points for smoothly overlapping basis functions. N_{min} is dependent on the level of noise on the data and the complexity of the local model. The local model structure selection criteria described in Section 4.2.5 uses this style of data count to encourage the algorithm to choose simpler models when data is sparse, and stopping growth can be viewed as the extension of this local structure selection to the case of preferring no increase in model structure to other model structures.

Pruning techniques

Further pruning techniques, which merge neighbouring local models if their parameters are similar enough, have also been successfully applied to simplify the networks. These rely on the use of local learning techniques, as described in Section 3.1.2, to ensure that the parameters have a strictly local interpretation.

The distance between the local model parameter vectors is then calculated $\delta_{ik} = |\boldsymbol{\theta}_i - \boldsymbol{\theta}_k|$, and the most similar $\min \delta_{ij}, i, j = 1..n_{\mathcal{M}}$ local models were merged into one, and the new basis function centred between the old ones.

The pruning algorithm can be applied during the constructive process, not just at the end of it. When the algorithm moves to a finer level of resolution the pruning stage is started, redundant models are removed, and only then are new models added.

4.2.5 Local model structure selection

The structure optimisation extends also to the model structures of the individual local models. These need not be homogeneous, the user can define a pool of possible local models which can then be inserted into a given operating regime, with the 'best' one being chosen. This would allow a more robust fitting of models to operating regimes, taking the amount of data and the local process complexity into account. Such a cost-complexity functional is

$$J(\mathcal{M}_i) = \frac{\hat{\epsilon}_i}{\left(1 - \frac{\beta P_i}{\rho_{total_i}}\right)} \tag{4.8}$$

where $\hat{\epsilon}_i$ is a statistic such as mean squared error, as defined in equation (3.30) and the $\left(1 - \frac{\beta P_i}{\rho_{total_i}}\right)$ part is a GCV penalty term, where P_i represents the number of parameters in local model structure $\mathcal{M}_i$. $\beta, (0 \leq \beta < 1)$, is a factor which can be related to the level of noise in the training data.

The resulting local model net will be a heterogeneous structure, as shown in Figure 3.7 where each local model could be different. If all local models are linear in the parameters, the standard global optimisation techniques remain valid. This will be true in the most straightforward example of heterogeneous LMN's, where the local models are linear, but with varying dynamic order. If some local models require nonlinear optimisation techniques, the heterogeneous local learning methods described in Section 3.1.2 can be used.

4.3 Active Learning with Local Model nets

As mentioned in Section 2.2.2, interest in *active learning* has grown in recent years. Local model networks are well suited to the application of these techniques for a number of reasons: In local model nets where the local models are linear in the parameters, a number of existing statistical methods for linear systems can be applied at local level, reducing the computational load in a similar manner to the local learning techniques described in Section 3.1.2. The basis functions give a clear indication of the data points related to any given local model, so that areas with insufficient data can be easily identified. The methods described in Section 3.2 for estimating the local confidence in the network for any given unit can be used to guide the search for new training data, collecting more in areas where the model has poor confidence in its accuracy.

Confidence estimates can be linked with Experiment Design techniques to produce the 'optimal' sampling of the input space for a given regression problem (Fedorov, 1972). This is done by adding samples where the uncertainty in the model is greatest. The experiment design framework has recently been applied to networks with a structure identical to local model networks in (Cohn et al., 1994). If local basis functions are used, this is obviously an improvement to the general experiment design philosophy, as the locality can be used to better determine regions of low confidence, and to select new areas for sampling.

4.3.1 Active selection of training data in Local Model Nets

In many learning problems, there is a large number of data points present in the training set $\mathcal{D}$, but often most are in 'uninteresting' areas of the input space. Such areas are often straightforward to model because they are usually the setpoints of the system, where control is most accurate, and it makes little sense to have such a large proportion of the training data representing such simple areas of the input space.

It is important therefore to reduce the training set $\mathcal{D}$ to a $\mathcal{D}_{opt}$, where the 'optimal' training set depends on the current model structure and a cost-complexity functional to weight the relative importance of residuals, costs of expanding the data set and computational effort. This should lead to a more computationally tractable optimisation process with a more information-rich training set.

Selecting from Local Training Sets

An active selection algorithm for use in conjunction with the iterative constructive algorithm was developed. The method selects a given number of training points from the full data set randomly from the receptive field of each local model. The points are deemed to be within the local model's training set $\mathcal{D}_i$, if the activation of the model's basis function is above a specified minimum activation level ζ, i.e.

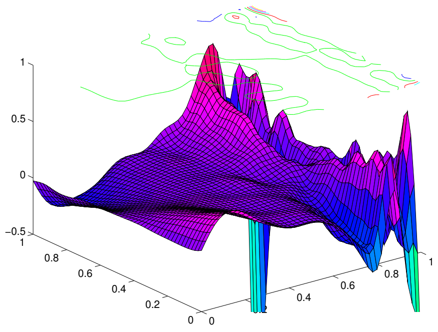
$$\boldsymbol{\psi} \in \mathcal{D}_i, \ \text{ if } \ \rho_i(d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i)) > \zeta. \tag{4.9}$$

The global training set $\mathcal{D}_{opt}$ is then created by reducing the subsets $\mathcal{D}_i, i = 1..n_{\mathcal{M}}$ and combining the reduced subsets (the $\mathcal{D}_{i_{opt}}$). There is a variety of methods which can be used to find the optimal local training sets. The simplest is to randomly select training data from the local set $\mathcal{D}_i$. A slightly more sophisticated method is to weight the probability of a point's inclusion by the associated value of the basis function $\rho_i(d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i))$. The size of the local data sets depends on the expected noise level in the data and the complexity of the associated local models.

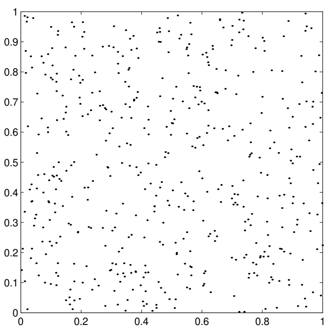
The process is illustrated in Figure 4.8. Two models were allowed a maximum number ($N =$ 500) of training points to learn from. One had only the 500 points, and used all of them. The other used an active selection system to choose its points from a training set of 5000 (for the experiment $N_{min} = 30$), but was allowed no more than 500 at any one moment. The system trained using active selection performed better on a uniformly distributed independent test set of 1000 patterns.

Unevenly distributed training data

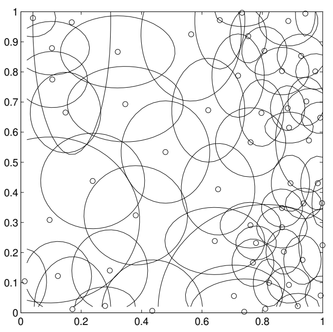
Unevenly distributed training data is a common feature of real industrial processes which are often relatively simple around the normal operating conditions, where most of the data is collected. The process often becomes increasingly complex as the operating point moves away from the given setpoint, and because such areas are harder to control, the process spends less time in them, and there is less data available for training. The importance of not just



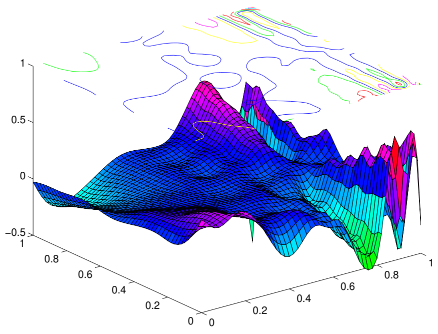
(a) Response- Trained with fixed training set



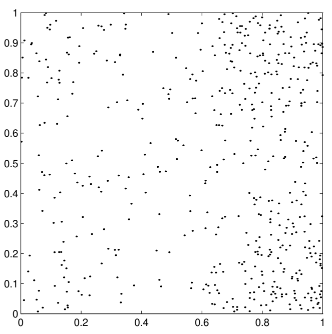
(b) 500 Training Data



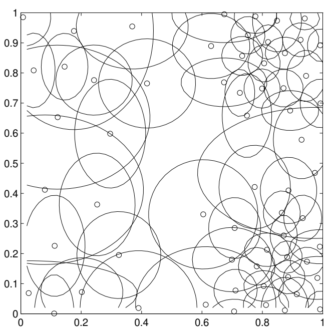
(c) 57 Basis Functions



(d) Response - Trained using Active learning



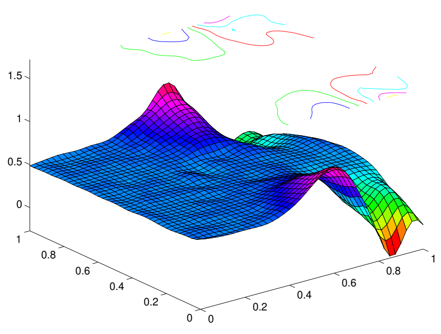
(e) 500 Training Data

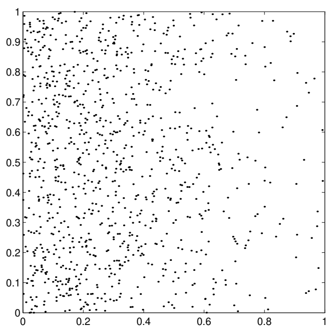


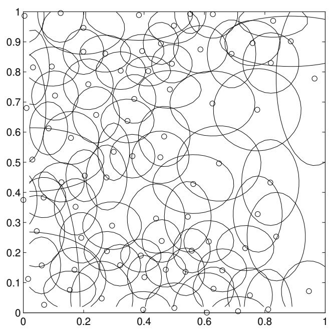
(f) 62 Basis Functions

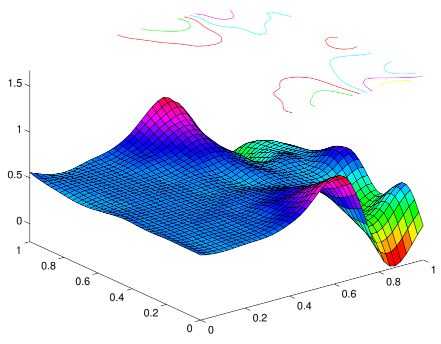
Figure 4.8: Active Selection and Random Selection of the same number of training data. Target function is shown in Figure 4.16 on page 106. Both algorithms could use only 500 points at any given stage of learning. The active selection algorithm, could however choose its 500 from a total set of 5000. Active learning had mean error 1.99%, worst of 59.6%. Fixed data set had a mean error of 2.27% and a worst error of 96.5% (results from a randomly distributed test set of 1000 points.)

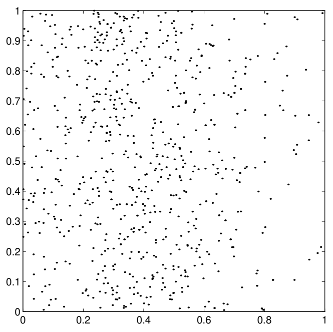
using the given distribution of data to model the target process is shown by the example in Figure 4.9, where the data is distributed most heavily in a relatively simple area of the target system. The data had additive normally distributed noise with a variance of 0.02.


(a) Model Response – Trained with fixed training set


(b) 1000 Training Data


(c) 80 Basis Functions


(d) Model Response – Trained using Active learning


(e) 742 Actively Selected Training Data

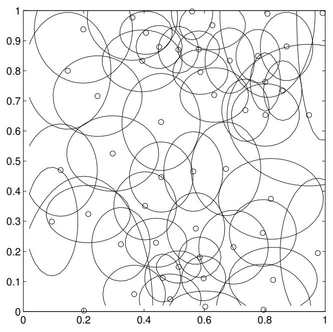

(f) 49 Basis Functions

Figure 4.9: Use of active learning to cope with training set distributions which are not related to the process complexity. The active learning system produces a model structure more suited to the local complexity of the system than the model produced using the larger fixed training set, which has concentrated its structure on the area of the input space with greatest data density. Target function is that shown in Figure 3.9 on page 68.

Adding randomly selected patterns

One danger of using constructive algorithms with active learning is that the trained model can become very 'narrow-minded', and dependent on the initial conditions. If the initial training data or model structure is biased towards one area of the input space, the constructive algorithm will start to develop there. Due to the more complex model structure in that area, the system will demand more local data, allowing it to develop even further, eventually becoming 'locked-in' to that one aspect of the process, ignoring other equally complex areas of the input space.

To make the active selection process more robust, a set of N_{rand} data points $\mathcal{D}_r$ was selected randomly from the whole training set $\mathcal{D}$, regardless of process complexity or model structure.

$$\mathcal{D}_{opt} = \bigcup_{i=1}^{n_{\mathcal{M}}} \mathcal{D}_{i_{opt}} \cup \mathcal{D}_r \tag{4.10}$$

The use of such active selection schemes speeds up the training process and the new distribution of the training points can improve the robustness of the training process. The fact that the training set is changed after each iteration also tends to make the structure identification algorithm more robust, as poor structure which by chance was well suited to one set of data will be discovered at the next iteration when the data set changes.

4.4 Illustrative Examples

In order to evaluate the use of the new constructive techniques described in the previous chapters we use a variety of test systems to illustrate the algorithm's strengths and weaknesses and we compare the algorithms with competing methods.

The *robustness* of the constructive algorithms and the parameter estimation routines can be verified using cross-validation techniques. For the cross-validation experiments in this thesis a 5-fold cross-validation is used to measure robustness of learning algorithms. This means that the model is trained 5 times with 80% of the data randomly chosen as the training set and tested each time on the remaining 20%. The quality index used to determine the model performance trained on the training set $\mathcal{D}_i$ was the average absolute error from each pattern p in the associated test set $\mathcal{T}_i$,

$$E_{abs_i} = \frac{1}{N}\sum_{p=1}^{N} |y_p - \hat{y}_p| . \tag{4.11}$$

Also important is the worst error found in the data set,

$$E_{max_i} = \max |y_p - \hat{y}_p| . \tag{4.12}$$

The cross-validation results are then found by taking the mean and variance of the results from the set of training sets.

$$E_{mean} = \frac{1}{f}\sum_{i=1}^{f} E_i, \tag{4.13}$$

and

$$E_{var} = \frac{1}{f}\sum_{i=1}^{f} (E_i - E_{mean})^2 , \tag{4.14}$$

where f is the number of 'folds' in the cross-validation process. If there is a high variance in the error produced on the training set over the various folds, it implies that there are some data patterns which map to a complex surface which the model is unable to represent. This could be due to disturbances, or simply to an inadequate sampling of the input space. If, however, the errors produced on the test sets vary greatly, and the errors on the training set do not, it implies that the architecture chosen is over parameterised (too powerful) for the given training set. The cross-validation results therefore give us an estimate of the robustness of the learning algorithms' models in the experiments.

The *interpretability* of trained networks is discussed in Section 6.3.4. It is difficult to determine whether a model is *parsimonious* or not, if little *a priori* knowledge about the system is available. One method is to apply pruning techniques to the model, as described in this chapter, and to observe the effect on the model accuracy. The *consistency* of an algorithm is easier to validate, as the same problem can be attacked with increasing amounts of data, and the resulting models compared.

4.4.1 Static examples

Unless otherwise stated, the MRC algorithm was used to create local model nets. Training was global, the initial window size was $\sigma_{max} = 0.2\sqrt{2}$, $\lambda = 0.6$, $n_{res} = 5$, $\gamma = \frac{\sqrt{2}}{2}$ and $N_{min} = 15n_{\psi}$, $\eta = 0.02$, $N_{des} = 1.5N_{min}$, $N_{rand} = N_{des}$. Basis functions were normalised.

Mars

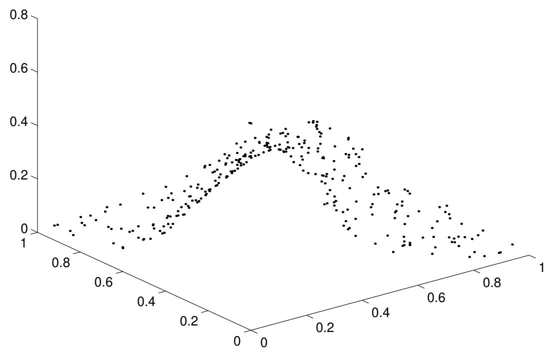
This is a benchmark from the nonparametric statistics literature, used in (Gu et al., 1990, Friedman, 1991).

$$z = \frac{2\exp\left(8\left[(x-0.5)^2+(y-0.5)^2\right]\right)}{\exp\left(8\left[(x-0.2)^2+(y-0.7)^2\right]\right)+\exp\left(8\left[(x-0.7)^2+(y-0.2)^2\right]\right)} \tag{4.15}$$

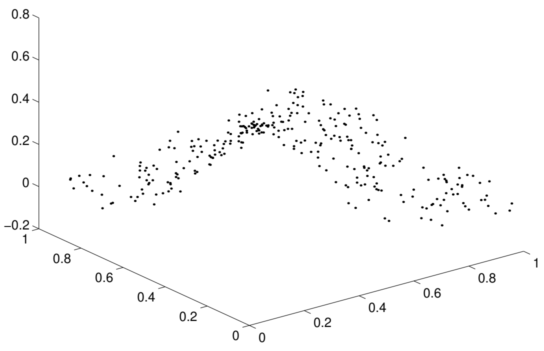
It is typical of a smooth nonlinear function, which should be relatively easy for a learning system to model. It is a collinear function (where the nonlinearity is dependent on linear combinations of the input dimensions), which makes it more difficult for methods which partition the input space orthogonal to the axes of the input space.

To give an impression of the effect of the noise on the training set, the 300 pattern training sets are plotted for the 2-dimensional benchmarks, with and without noise (normally distributed, variance = 0.02). The target tolerance η was set to 0.005 for noise free examples, and 0.02 for noisy data.

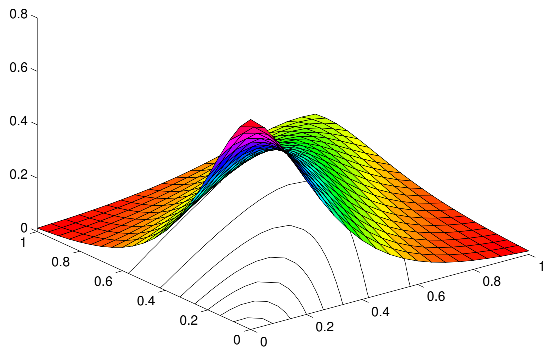
The response of Local Model Network trained on the Mars1 data is shown in Figure 4.11.



(a) Mars1 training data

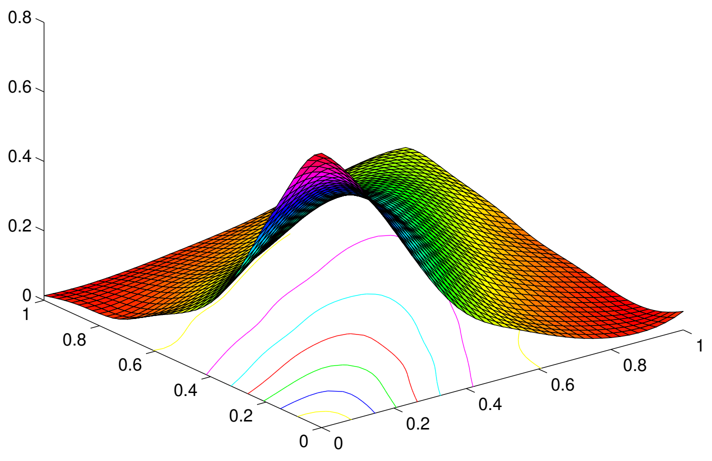


(b) Mars1 noisy training data

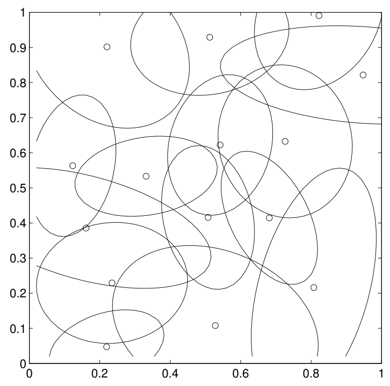


(c) Mars1 function

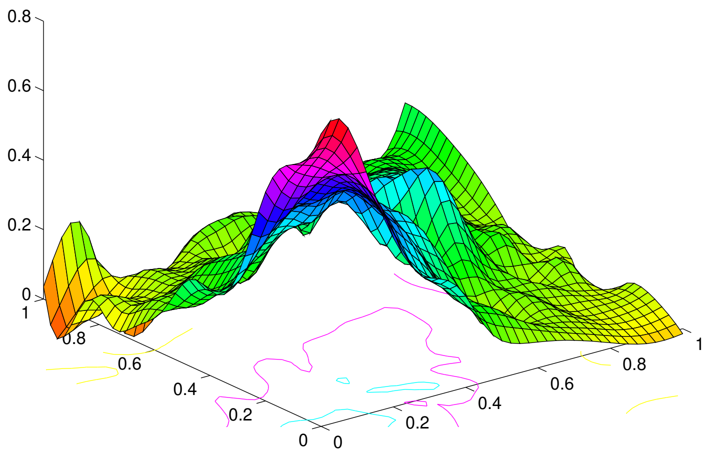
Figure 4.10: Mars1 test function and 300 training points. z axis is vertical. x and y are right and left respectively.



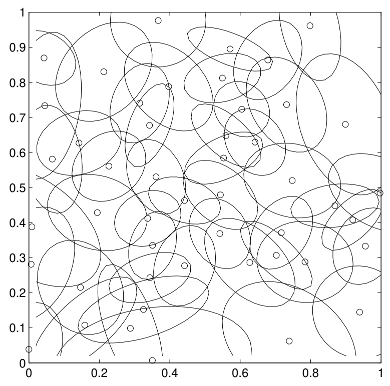
(a) LMN Response $n_{res} = 5, \lambda = 0.6, \eta = 0.005$



(b) LMN with 11 ellipsoidal Basis Functions. Contours are drawn at 0.5

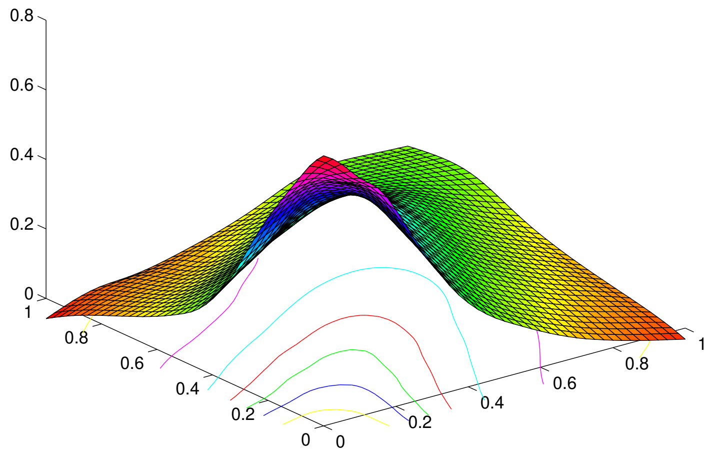


(c) LMN Response on noisy data $n_{res} = 5, \lambda = 0.6, \eta = 0.02$

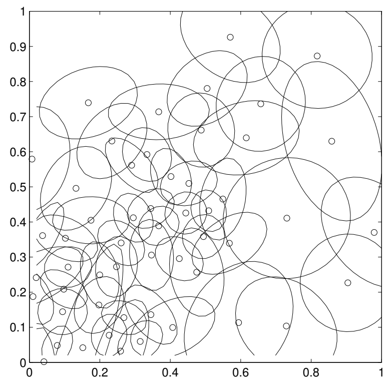


(d) LMN Basis Functions for noisy data. Contours are drawn at 0.5

Figure 4.11: Resulting LMN responses for mars1 benchmark (300 training points) using global learning. Note the effect the noise has on the smoothness of the approximation. This is due to the model structure extending too far, in an attempt to model the noise. Compare to the network response for the locally trained network in Figure 4.12.

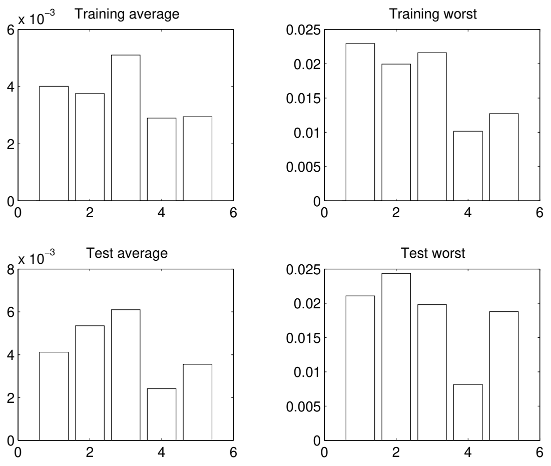


(a) LMN Response on noisy data $n_{res} = 5, \lambda = 0.6, \eta = 0.02$, using local learning.

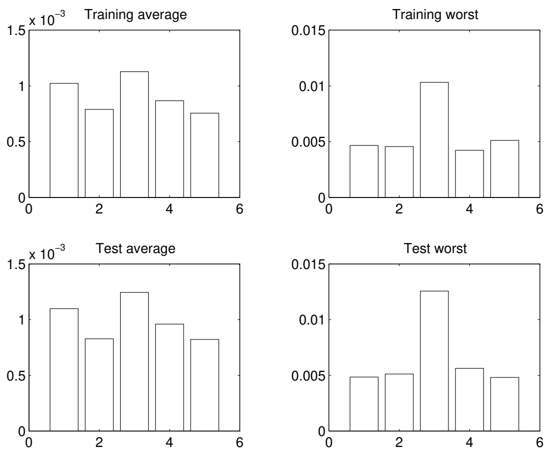


(b) LMN Basis Functions for noisy data. Contours are drawn at 0.5

Figure 4.12: Resulting LMN responses for mars1 benchmark (300 training points) using local learning. Note the far smoother response than that of the globally trained network in Figure 4.11.

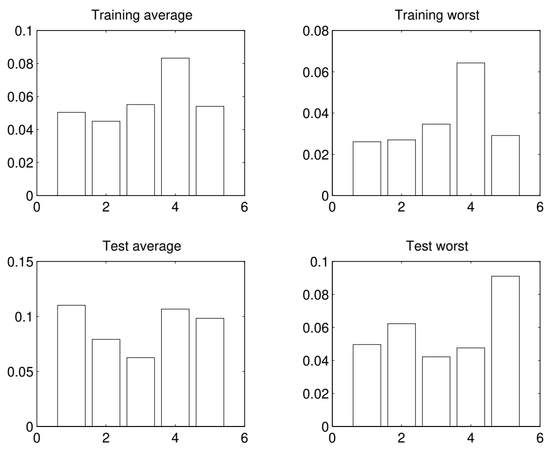


(a) Mars1 problem, 300 training points and local models

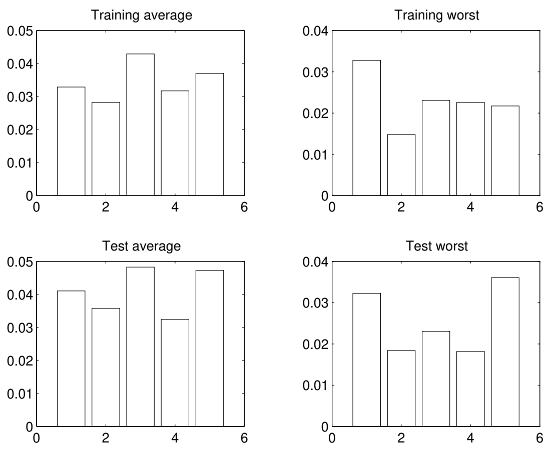


(b) Mars1 problem, 1000 training points and local models

Figure 4.13: Cross-validation results for Local Model Net on noise-free Mars example. Error decreases with the increased amount of training data.

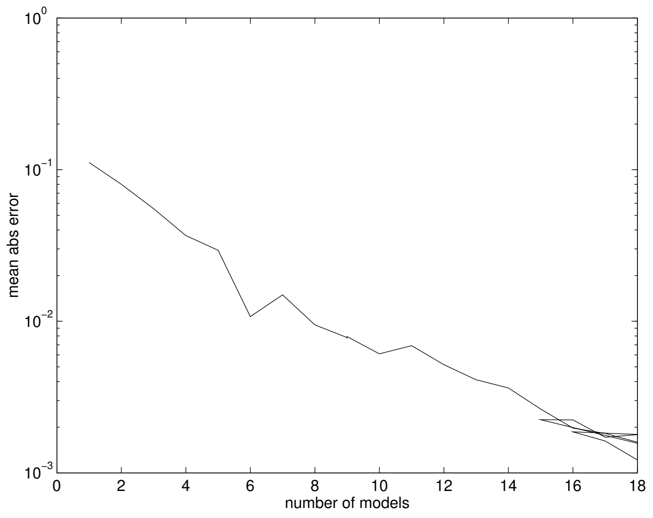


(a) Mars1 problem, 300 training points and MARS algorithm

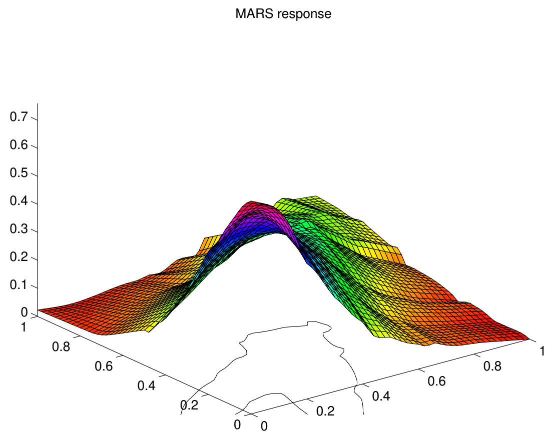


(b) Mars1 problem, 1000 training points and MARS algorithm

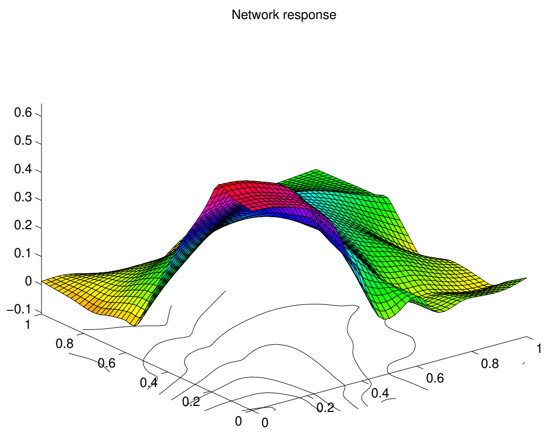
Figure 4.14: Cross-validation results for MARS algorithm on the noise-free Mars example. Note that MARS performs worse on training and test measures than the Local Model Net, and there is less similarity in the results for training and test runs than there is with the Local Model Nets.



(a) LMN Learning curve. Note the glitches towards the end of learning, where the network prunes units with insufficient training data.



(b) MARS Response on clean data



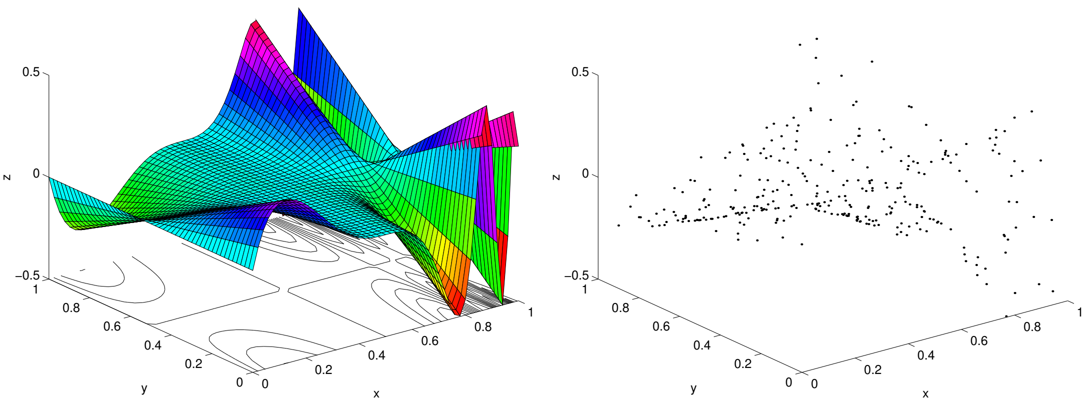
(c) MARS Response on noisy data

Figure 4.15: MARS response to mars1 function, trained with 300 data points

Squiggle

This benchmark has several aspects: the influence of y is as a linear multiplication, whereas x has a highly nonlinear behaviour. The nonlinearity is also of varying complexity, having very high frequency components at large values of x, and changing very slowly in the middle.

$$z = (y - 0.5)\sin\left(10\pi x \sin(x - 0.5)^3\right) \tag{4.16}$$



(a) Squiggle function (b) Squiggle training data

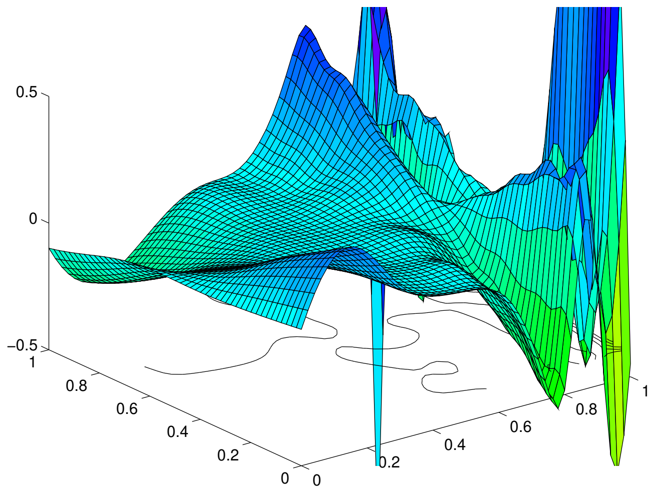
Figure 4.16: Squiggle test function and 300 training data points

This is an extremely difficult benchmark for most learning systems. Because of the extreme variation in complexity it shows the necessity of intelligent construction of the training set for such systems, using either experiment design techniques, or active learning.

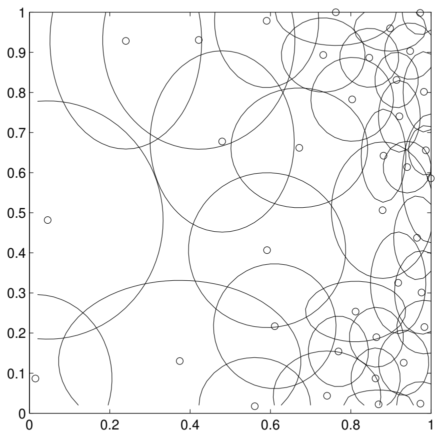
The MARS algorithm performs very well on the Squiggle function, because the nonlinearity is axis-orthogonal, and the function is only highly nonlinear in x, and MARS strength lies in its ability to separate important inputs from unimportant inputs when developing the model structure. Other axis orthogonal schemes such as LSA (Johansen and Foss, 1994b) produce similar results, and it would be expected that ASMOD would also work well on this example.

Rotated Squiggle

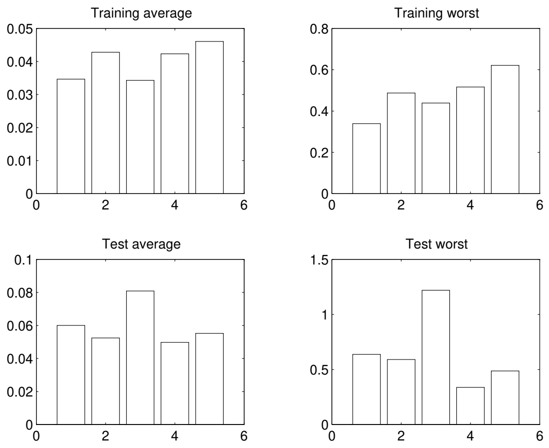
To show the disadvantage of axis-orthogonal representations, however, all we have to do is rotate the function. In this case it has been rotated by $\frac{\pi}{4}$ radians. The rotated squiggle function and various responses are shown in Figure 4.19. The mean absolute error for MARS, trained on 1000 points was 0.0237 (used 64 basis functions). LMN achieved a mean error of 0.0043 with 62 local models (with ellipsoidal distance metrics).



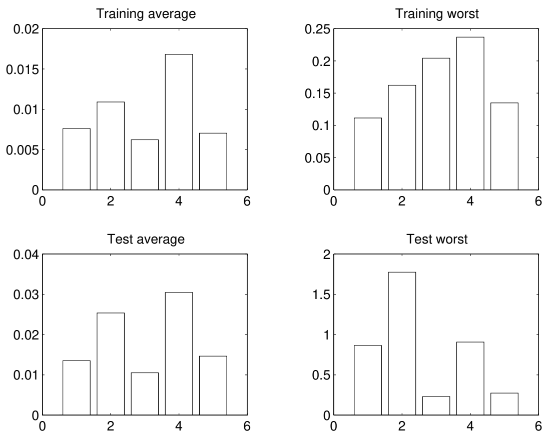
(a) LMN Response



(b) LMN Basis Functions. Contours are drawn at 0.5

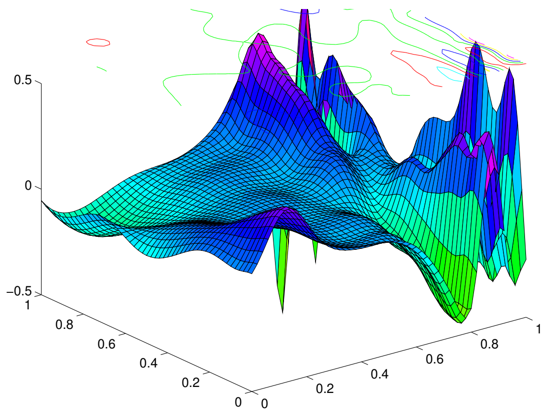


(c) Squiggle problem, 300 training points and Local Model Net

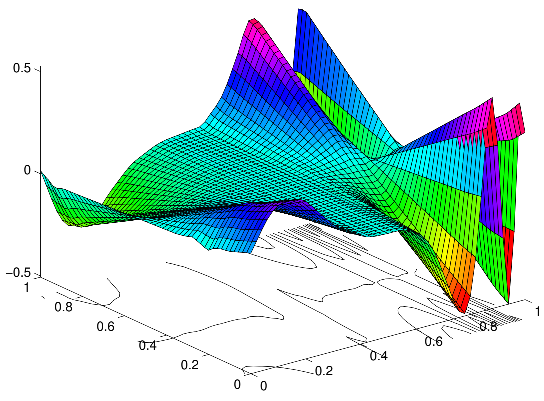


(d) Squiggle problem, 1000 training points and Local Model Net

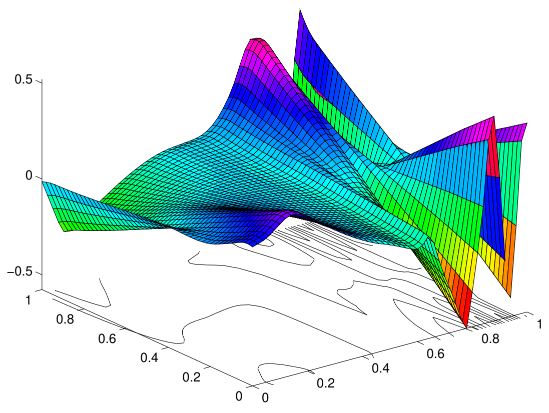
Figure 4.17: Resulting LMN for Squiggle benchmark (1000 training points), with 38 axis-orthogonal ellipsoidal basis functions and global learning. Compare with MARS response in Figure 4.18 which is very close to the original function. The LMN for the noisy data had $N_{min} = 30$, instead of 15, and $\eta = 0.02$ instead of 0.005 for the noise free case, which lead it to produce a good model with 30 local models.



(a) LMN Response on noisy data

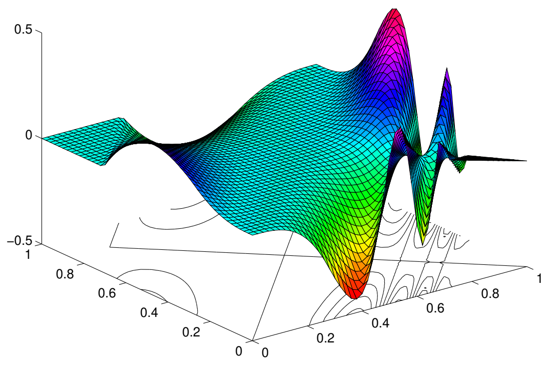


(b) MARS Response on clean data

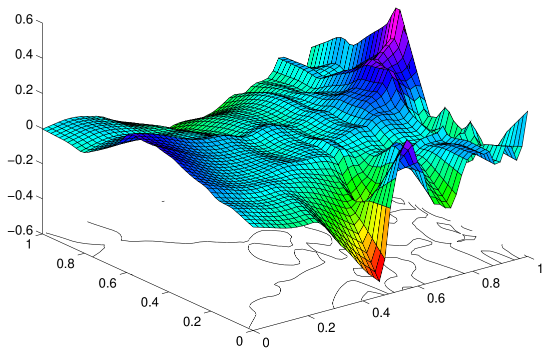


(c) MARS Response on noisy data

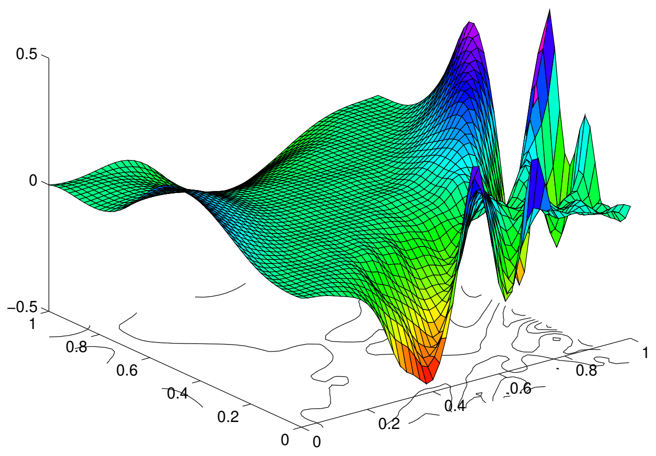
Figure 4.18: MARS responses and LMN response on noisy data.



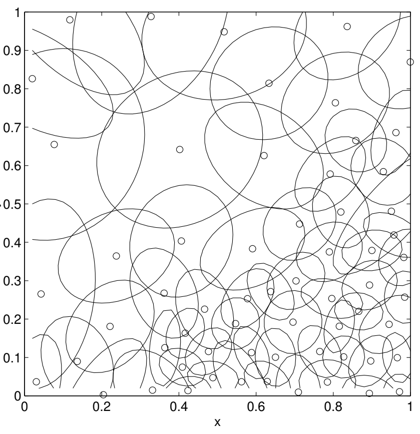
(a) Rotated Squiggle function



(b) MARS Rotated Squiggle response



(c) LMN Rotated Squiggle response



(d) LMN Rotated Squiggle response

Figure 4.19: Rotated Squiggle responses. Note the difficulty MARS has in comparison with the original Squiggle benchmark, due to the non-axis orthogonal nonlinearity, whereas the LMN copes well with the change.

4.4.2 Dynamic systems

The static examples above are interesting test cases for constructive algorithms, because they can be easily visualised, and as they are synthetic, data can be easily created with varying degrees of noise. They are, however, not high dimensional and the uniform data distribution is also not typical of modelling applications for dynamic systems. One major difference between static and dynamic systems is that in the static case the input dimension of the problem is known, whereas in dynamic systems this is not necessarily the case. In the modelling examples in this thesis the learning algorithms do not identify the order of the system, the user defines it before learning starts.

Measuring model quality for dynamic systems

A straightforward measure of network accuracy is the *one-step-ahead prediction* measure, where previous system inputs and outputs are fed to the model, which should then predict the system output at the next time step:

$$E_{osh} = \frac{1}{N-1} \sum_{m=1}^{N-1} \sum_{k=m}^{m+1} \left(y_k - \hat{y}_k\right)^2, \tag{4.17}$$

where $\hat{y} = f(\psi)$, and $\psi = [y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots u_{k-n}]$. It is obviously important to compare the errors produced at each stage with the expected change in the output between samples. If the sampling rate is too high, the learning system can have difficulty in learning the system's dynamics. In practice, unless interpreted carefully, the one-step ahead method can give misleadingly good results for a model which in reality is a poor representation of the process.

A more realistic test of the system's modelling ability is to leave the model free-running over a horizon of h steps, with its own output states being fed back and to see whether the model reacts in the same way as the target system – the *Multiple-step-ahead prediction* test.

$$E_{msh} = \frac{1}{N-h} \sum_{m=1}^{N-h} \sum_{k=m}^{m+h} \left(y_k - \hat{y}_k\right)^2, \tag{4.18}$$

where $\hat{y} = f(\psi)$, and $\psi = [\hat{y}_{k-1}, \ldots, \hat{y}_{k-n}, u_{k-1}, \ldots u_{k-n}]$

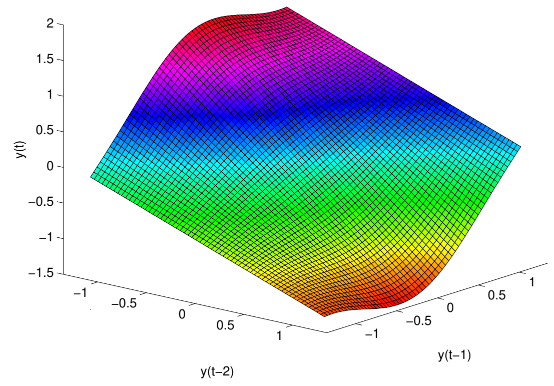
Nonlinear time-series example

A two input one output nonlinear system is described by the following equation:

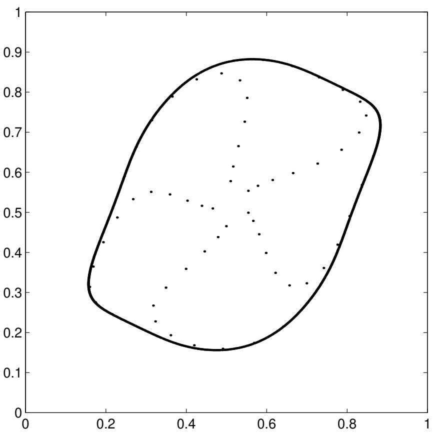
$$y(t) = \left[0.8 - 0.5\exp(-y^2(t-1))\right] y(t-1) - 0.1\sin(\pi y(t-1)) \tag{4.19}$$

$$+ \left[0.3 + 0.9\exp(-y^2(t-1))\right] y(t-2) + e(t), \tag{4.20}$$

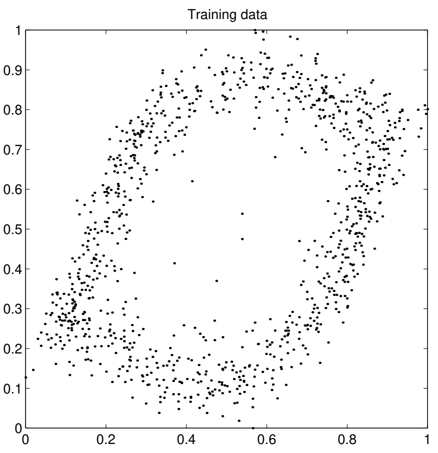
where $e(t)$ is zero mean Gaussian noise, with variance 0.01. The inputs $x(t) = [y(t-1)y(t-2)]$, and the output is $y(t)$. The response surface for the function without noise is shown in



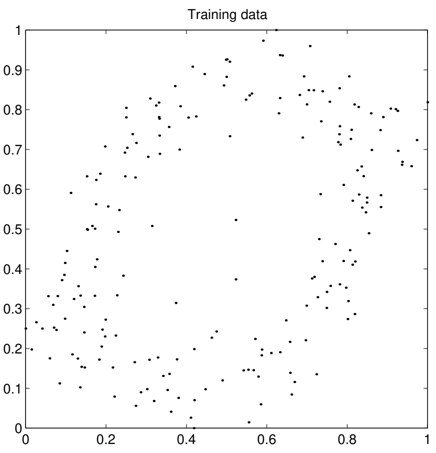
(a) Target function



(b) Target Phase portrait (normalised to same range as experiments)



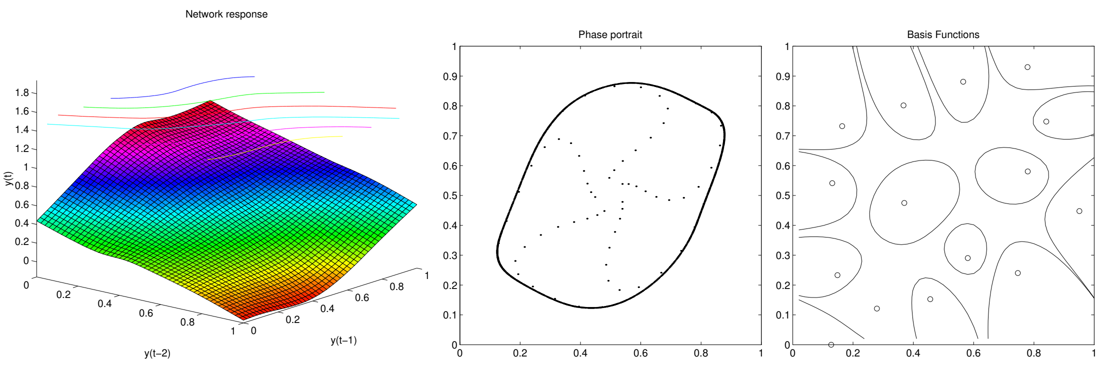
(c) Training Data - 1000 examples



(d) Training Data - 200 examples

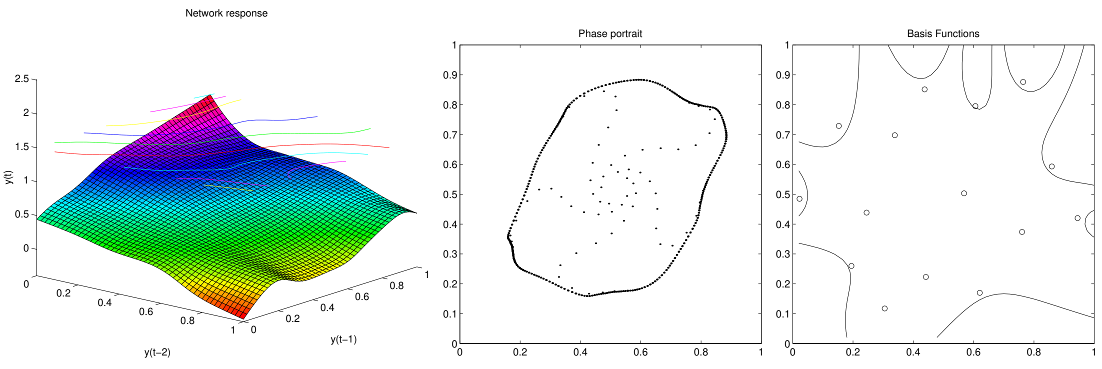
Figure 4.20: 4.20(a) Time series function response. Figure 4.20(b) Phase portrait. Figure 4.20(c) and Figure 4.20(d) are noisy data sets.

Figure 4.20. The time series represents an unstable equilibrium, enclosed by a stable attracting limit cycle. This process has been used in (Chen and Billings, 1992) and (Harris et al., 1993). Two sets of experiments were performed with the given data, one with 1000 points, as used in (Harris et al., 1993), and another more difficult case, where only 200 training points were used. To test the results a free-running simulation was used, where the model was given the initial value [0.1, 0.1]. The results of the 1000 set experiments are shown in Figures 4.21 and 4.22. Both locally and globally trained models capture the essential dynamics of the process. The



(a) Model Response (b) Test run (c) Basis Functions

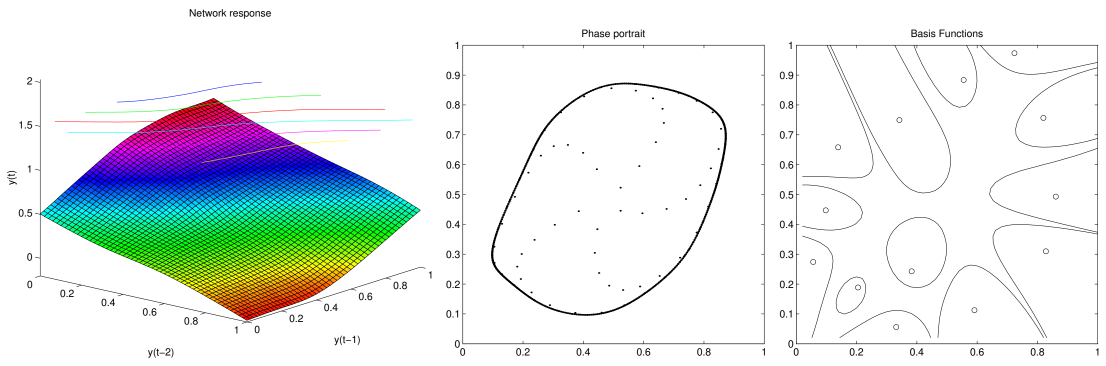
Figure 4.21: Time series model. Local model net with local learning, trained on 1000 examples.



(a) Model Response (b) Test run (c) Basis Functions

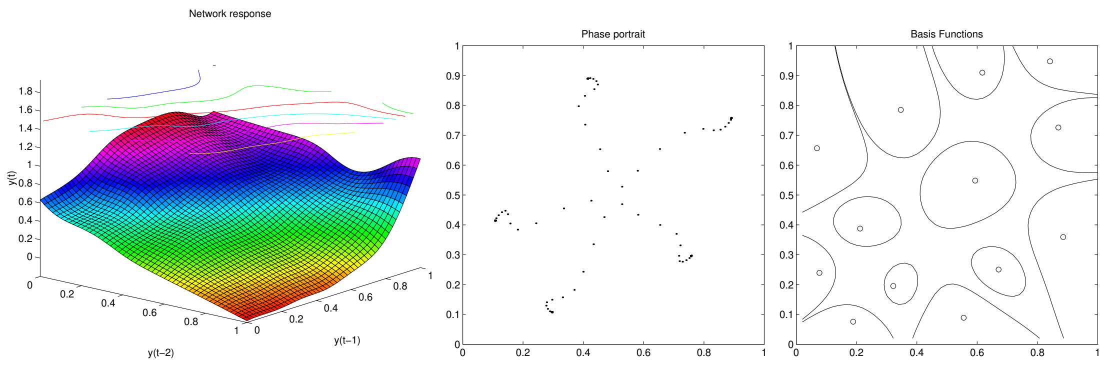
Figure 4.22: Time series model. Local model net with global learning, trained on 1000 examples. Note the smoother approximation of the locally trained model.

process is modelled with a small number of basis functions, compared to the work reported in (Chen and Billings, 1992), which used over 30 basis functions to approximate the same problem. Note, however, the difference in the phase portraits between the locally and globally trained systems. The local method produces a far smoother model. The results from the more



(a) Model Response (b) Test run (c) Basis Functions

Figure 4.23: Time series model. Local model net with local learning, trained on 200 examples.



(a) Model Response (b) Test run (c) Basis Functions

Figure 4.24: Time series model. Local model net with global learning, trained on 200 examples. The locally trained model still approximates the process well, while the global method does not provide a stable solution.

difficult 200 point problem are shown in Figures 4.23 and 4.24. The globally trained model in Figure 4.24 proved to be unable to produce a robust free-running simulation, because of the poor generalisation. This is also visible in the form of the response plot, which is highly variable outside the area covered by the training data. This problem with robustness in a free-running simulation is one which is of great importance for the practical use of dynamic models, but which has been often disregarded in the literature, as the accuracy of models is often evaluated only on one-step-ahead prediction tests.

4.5 Conclusions

4.5.1 Structure identification in Local Model Networks

The new Multi-Resolution Constructive (MRC) structure identification algorithm proposed in this chapter has a number of advantages. The procedure is relatively fast, and automatically determines its own structure for a given modelling problem. The complexity detection heuristic performs well at allocating resources to the more complex areas of the model's state-space, although it is likely that more efficient ways can be found of implementing the basic idea. The algorithm also limits overfitting by the simple but very effective method of gradual construction, stopping structural growth in areas which are not sufficiently populated by training examples. Pruning methods which compare neighbouring local models' parameters to detect overparameterisation were also successfully used.

A new algorithm for determining the size and shape of the basis functions was developed. It is based on a 'covariance' measure of the surrounding centres. Radial, axis-orthogonal ellipsoidal and fully ellipsoidal basis functions were used. The axis-orthogonal ellipses tended to produce the most robust models. From the experience in this work, the algorithm works well for relatively low dimensional problems, but becomes less robust with increasing input dimension.

It is interesting that the algorithm, despite its relatively simple search technique can model some extremely nonlinear systems better than other more sophisticated algorithms, such as MARS. This reflects the representation of the nonlinearity inherent to the competing model structures. Axis-orthogonal structures such as LSA, MARS or ASMOD produce better and more interpretable models for some systems, whereas MRC produces better models when the nonlinearity is at an angle through the input space.

The dependence on the training data of the MRC algorithm, is both a strength and a weakness of the algorithm. From experience, the MRC algorithm tends to perform poorly at ignoring unimportant input dimensions, but the computational effort does not increase exponentially with input dimension, meaning that the problem can still be used usefully on higher dimensional problems. The increase of computational effort with the amount of training data can be controlled by using the active selection techniques described in this chapter.

The disadvantages of the limited amount of feedback[4] in the MRC algorithm become more obvious when noisy data is used, especially when the data distribution is skewed, as the algorithm tends to fit model structure in the wrong areas of the input space. Local learning proved to be a remedy for some of these problems, performing well with the noisy systems.

[4]Here the feedback discussed is the effect of the new model on the cost-complexity measure.

Future work

The structure identification problem is far from solved, and this should remain a major research area in machine learning in the near future. The problems with the methods described in this chapter are that they tend not to cope well with a large number of relatively unimportant input dimensions.

The criterion for stopping construction is not particularly robust, especially when local learning is used, and it seems to be the main source of variance in the algorithm's performance. The effect of changing the reduction schedule for the multi-resolution complexity 'window' is still not fully understood, leaving too many 'fiddle factors' (the values used in this work, however, proved to be reasonably general, working well on a variety of problems). It also indicates the need for *hierarchical structure identification*, which as learning progresses, revises earlier structure decisions, producing a more parsimonious, robust and interpretable model.

Local Controller Networks are the obvious generalisation of Local Model networks, where the local functions can be viewed as controllers, instead of models. The framework is clearly similar to *gain scheduling* in conventional control (Shamma and Athans, 1991), and has recently been described as the use of *Heterogeneous Control Laws* in (Kuipers and Åström, 1994). The methodology could, however, benefit from the use of structure identification algorithms such as MRC to automatically place the controllers. In the linear case, the transfer from local model to local controller is easiest to imagine: The local model network for a given system could be created, and the corresponding local controllers then created using conventional linear design techniques. A global nonlinear controller is then the result of the interpolated linear controllers. It would seem that locally trained local models are better suited for the creation of local controllers, due to their local interpretability.

Initial experiments have shown the structure to be well-suited for classification purposes. The work will be published in a future paper.

4.5.2 Active learning

Active learning was easily introduced to the local model networks. The algorithm described in this chapter for active selection of important data from the training set, in conjunction with the constructive structure identification algorithm was found to make the learning process more efficient and in some cases led to more accurate models. Active selection also lets the constructive algorithm cope better with non-uniformly distributed training data. The local nature of the basis functions makes the task of selecting the new inputs easier than with other architectures.

The improvement gained by using active learning, in speed and accuracy is demonstrated again in Chapter 6.

Future work

The extension of the automatic modelling methodology to the unsupervised automated creation of training sets is an important goal, but for industrial applications there are still many practical and safety aspects which have to be dealt with. The next step following the work in this thesis would be to apply the theory of experiment design, as in (Cohn et al., 1994).

The use of active learning in its *active sampling* form in dynamic systems requires an existing controller to perform the experiment, and in the case of large expensive industrial plant it is not always going to be possible to allow the learning system to stimulate the process as it pleases! This should not, however, be seen as an indication that active learning has no role in modelling real dynamic systems, as the online version can be implemented with a human in the loop – the active learning system could be seen as an *experiment assistant* who describes where the system lacked data, so that the engineer could design an experiment which would provide the missing information.

The active learning area, be it active sampling or active selection, is likely to be of benefit to all areas of learning systems, but the application of active learning in nonlinear dynamic systems is very much in its infancy. The development of methods which automatically explore the input space of a dynamic system within defined constraints will be an important extension of the research which should be significant both for engineers developing models of given processes, and for future intelligent, autonomous machines which have to cope with a continually changing environment.

Chapter 5

Hierarchies of Local Models

The Learning Hierarchy of Models (LHM) Network is described. This is effectively a hierarchy of Local Model Nets where the local models can consist of further sub-networks. Parameter estimation methods are developed which apply conventional regression techniques by first unravelling the hierarchy. Local methods for estimating the parameters of sub-trees within the model are introduced. A constructive algorithm for the structure is described. The constructive algorithms are compared with MARS and flat Local Model Nets on a number of test systems.

5.1 The LHM Architecture

The Local Model Net framework can be extended by replaced by allowing the local models to also be sub-networks instead of simple linear models. A hierarchy of models can then be iteratively constructed to model the target system. Simpler local models are replaced by sub-networks, leading to the new hierarchical model structure having an increasing representational ability.

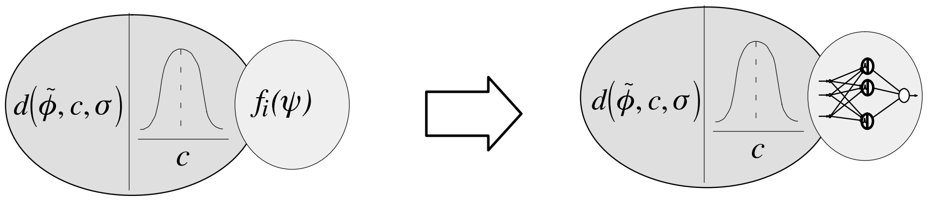


Figure 5.1: Local Models can be replaced by sub-networks to improve representational ability

The top level of an LHM structure can be described as a local model net,

$$\hat{y}(t) = \hat{f}(\psi(t-1); \mathcal{M}) = \sum_{i=1}^{n_{\mathcal{M}_0}} \hat{f}_i(\psi(t-1); \mathcal{M}_i)\rho_i(\tilde{\phi}), \tag{5.1}$$

where the level in the hierarchy of the basis functions and local models is indicated by the number of subscripts, and $n_{\mathcal{M}_{i_j}}$ indicates the number of local models at level 2 in model j, the child of model i. The model structures $\mathcal{M}_i$ are the local model structures (e.g. linear ARX model, *a priori* models), or are again general LHM structures defining the form of the hierarchy of sub-models below the current level in the tree.

We assume, for simplicity, that the operating point is $\tilde{\phi}$ at each level. A local model at the next level in the tree, $\hat{f}_i(\psi(t-1);\mathcal{M}_i)$, can be defined as

$$\hat{f}_i(\psi_i(t-1);\mathcal{M}_i) = \sum_{j=1}^{n_{\mathcal{M}_i}} \hat{f}_{i_j}(\psi(t-1);\mathcal{M}_{i_j})\rho_{i_j}(\tilde{\phi}). \tag{5.2}$$

5.1.1 Soft-splits

Here, a binary 'soft-split' is assumed at each stage ($n_{\mathcal{M}} = 2$), giving each parent model two children. The subspace is therefore partitioned by a soft-split oriented orthogonal to a hyperplane (unless the basis functions were of unequal size, which would lead to a curved partition). In the simplest form of partition, both local models' basis functions can be thought

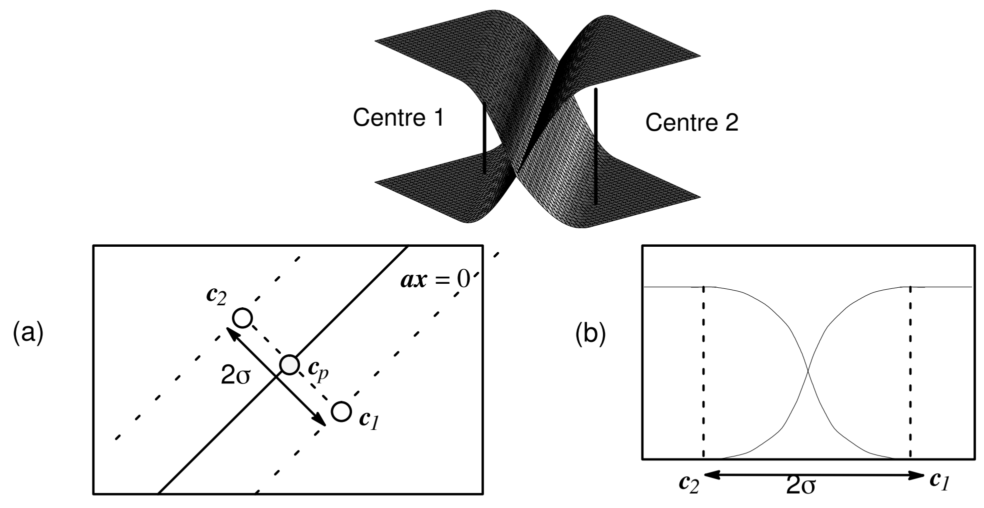


Figure 5.2: (a) The soft split from above and (b) The split from the side

of as two interlocked identical ridge functions, with opposite signs. The hyperplane produced by the intersection of the two functions is defined by the vector of angle weights $\mathbf{a}, \mathbf{a} \in \mathcal{R}^{n_{\tilde{\phi}}}$ in the equation $\mathbf{a}\tilde{\phi} = 0$. The split can be created from two normalised radial basis functions with equal widths. The centres are placed orthogonal to the plane defined by vector $\mathbf{a}$, and are both σ away from the centre $\mathbf{c}_p$ of the parent local model (through which the plane defined by $\mathbf{a}$ passes). The exact form of the non-linear split function is only restricted by the fact that it must sum to unity, and that smoothness is a desirable feature. The splits are optimised to partition the input space at each stage in a way which will reduce the modelling error as far as possible. The adaptation of the orientation of the splits is discussed in Section 5.4.3. Splitting the input space is usually better for eliminating 'uninteresting' dimensions than the methods described in the previous chapter, which were limited to the data points in the training set. This makes the structure more suitable for high-dimensional modelling problems.

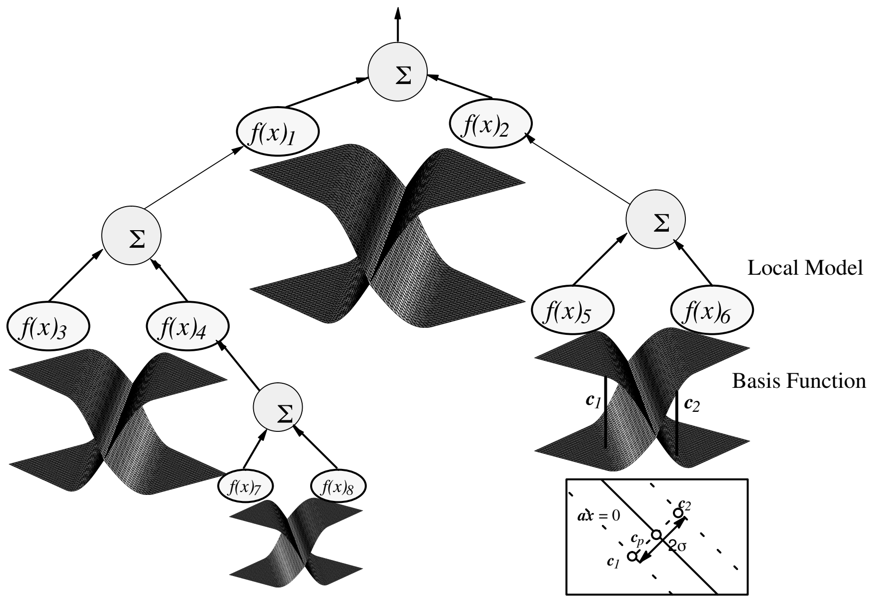


Figure 5.3: The Learning Hierarchy of Models architecture. A tree of local models is produced, where the leaves of the tree consist of the local models, the results of which are interpolated with the results from other leaf models by the parent basis functions of the higher levels of the structure.

5.2 Optimising the Local Model Parameters

Once the hierarchy and the basis functions in the LHM architecture have been fixed, the structure can be treated as a linear (in the parameters) system, and the parameters of the local models can be optimised using standard least squares algorithms such as SVD, exactly as with BF and local model nets. This is done by unravelling the tree to become a standard local model network, as shown below for a two level tree:

$$\hat{y}(t) = \sum_{i=1}^{n_{\mathcal{M}_0}} \left(\sum_{j=1}^{n_{\mathcal{M}_i}} \hat{f}_{i_j}(\psi(t-1); \mathcal{M}_{i_j}) \rho_{i_j}(\tilde{\phi}) \right) \rho_i(\tilde{\phi}). \tag{5.3}$$

If the local models are linear, and the tree binary,

$$\hat{y}(t) = \sum_{i=1}^{2} \left(\sum_{j=1}^{2} \left(\psi^T(t-1)\theta_{i_j}\right) \rho_{i_j}(\tilde{\phi}) \right) \rho_i(\tilde{\phi}), \tag{5.4}$$

which leads to each local model effectively being weighted by a cumulative basis function $\rho_{cum}(d(\boldsymbol{\psi}))$ which is the product of parent models' basis functions back up the hierarchy to starting level s:

$$\rho_{cum_i}(d(\psi)) = \prod_{l=s}^{h(i)} \left(\rho_{p(s)\cdot_{p(l-1)_{p(l)}}} \right), \tag{5.5}$$

where the $p(l)$ at each level l from starting level s to final level h indicates the relevant node in the tree on the path to the i-th leaf. The cumulative basis function design matrix $\mathbf{\Phi}_{cum}$ can then be used in the same manner as for the flat local model net. $\mathbf{\Phi}_{cum}$ is composed of the inputs to the local models on the leaves of the tree, weighted by the cumulative basis functions.

The hierarchical network can now be represented in the non-hierarchical form:

$$\hat{y} = \sum_{i=1}^{n_{\mathcal{M}}} \left(\psi_i^T(t-1)\theta_i \right) \mathbf{\Phi}_{cum_i}(\tilde{\boldsymbol{\phi}}), \tag{5.6}$$

where $n_{\mathcal{M}}$ is the number of leaf nodes. This format allows the parameters θ to be estimated using the standard estimation methods described earlier. The method is, however, similarly plagued by ill-conditioned design matrices, leading again to the need to apply solutions such as local learning.

5.2.1 Sub-tree optimisation using weighted least squares

The local learning ideas described in Section 3.1 can also be applied to the hierarchy of models. The weight optimisation procedure can be carried out at any position in the tree, for the sub-tree below the selected node. The optimisation process for subtrees can be made a weighted least squares procedure, by using the cumulative effect of a node's parents basis functions as the weighting function. This weighting function $\alpha(\boldsymbol{\psi})$ (assuming that the start level s is not the top level) is then

$$\alpha(\psi) = \prod_{l=0}^{s} \left(\rho_{p(0)\cdot_{p(s-1)_{p(s)}}} \right). \tag{5.7}$$

A sub-tree with M leaves and n_ψ input dimensions and local linear models would become a general linear least squares problem with Mn_ψ parameters. The solution to the problem is then the same as that described above. Locally optimising a sub-tree is especially relevant when adding new units to the tree, as the effect of various possibilities can be evaluated without optimising the whole tree. As with the flat networks, the effect of the more localised optimisation will be that the optimisation stage will be faster, as well as sometimes being more robust and interpretable. The representational ability of the network to learn a given training set will, however, often be reduced. The hierarchical nature of the architecture gives the designer more control over the level of locality used in learning than with the flat Local Model Net.

5.3 Confidence Limits with LHM

The methods described in Section 3.2.1 for the interpolation of local estimates of accuracy can be easily extended to the hierarchical case. The local error statistic from a leaf node can be weighted by the leaf node's parents' basis functions. The two level net is again used to illustrate the concept,

$$\hat{\epsilon}(\tilde{\phi}) = \sum_{i=1}^{n_{\mathcal{M}_0}} \left(\sum_{j=1}^{n_{\mathcal{M}_i}} \hat{\epsilon}_{i_j} \rho_{i_j}(\tilde{\phi}) \right) \rho_i(\tilde{\phi}). \tag{5.8}$$

The error estimation can obviously also be unravelled to produce

$$\hat{\epsilon}(\tilde{\phi}) = \sum_{i=1}^{n_{\rho_{cum}}} \hat{\epsilon}_i \rho_{cum_i}(\tilde{\phi}), \tag{5.9}$$

where the $\hat{\epsilon}$'s at the leaves have been reordered into a vector form. As with the sub-tree optimisation, the hierarchical structure of the model can be used to give the developer greater control of the scope of the error statistics. Higher level models can derive their statistics from lower level ones, as in equation (5.8), or the error statistics can be calculated from the 'global' model below that point in the hierarchy.

5.3.1 Using local error statistics to indicate poor model structure

A structure identification algorithm usually uses a cost-complexity term for optimisation. The *cost* is generally the degree to which the model differs from the test data. The method used in this work was to take the local mean squared error statistic, i.e. the network's squared errors at all N_i points in local training set $\mathcal{D}_i$ within local model $\hat{f}_i(\psi)$'s receptive field, then weighted by the local model's cumulative basis function $\rho_{cum_i}(\tilde{\phi})$, as described in Section 3.2.1:

$$J(\mathcal{M}_i) = \hat{\epsilon}_i = \frac{1}{N_i} \sum_{t=1}^{N_i} \rho_{cum_i}(\tilde{\phi})(y(t) - \hat{y}(t))^2. \tag{5.10}$$

Complexity terms

The complexity aspect of the optimisation functional is a term which penalises over-complex networks. It results in simpler networks, which have slightly poorer performance on the training data, but which are expected to show more robustness in the face of new data, and which are less likely to have spurious 'folds' in their mappings[1]. A simple way of penalising complexity in the resulting model is to use a weighted product of the parameters used in the local model and the breadth of the model's basis function. The *GCV* (Generalised Cross Validation)

[1]important if the model is to be used in a model-based controller, where an optimisation procedure makes use of the model's derivatives

term below is based on (Craven and Wahba, 1979, Wahba, 1990) which will penalise complex models (i.e. local models with a poor data-to-parameters ratio),

$$GCV(\mathcal{M}_i) = \frac{1}{1 - \frac{\beta P_i}{N_i}}. \tag{5.11}$$

The *range* term below is a heuristic which penalises models with smaller basis functions more heavily. If used in a constructive structure identification algorithm this will bias the algorithm to split models covering more of the input space first, thus having a smoothing effect on the resulting mapping,

$$Range(\mathcal{M}_i) = 1 + \gamma\sigma_i. \tag{5.12}$$

The cost complexity functional can then be created by taking the product of the components. P_i is a measure of the complexity of the local model i (usually the number of parameters). The parameters γ and β can be adjusted to fit the type of problem. For example, a model to be used in a control system should usually be as smooth as possible, so would possibly have a higher γ than a model which was to be used in simulation. The β factor $(0 \leq \beta < 1)$ can be related to the level of noise in the training data. The overall cost-complexity measure is then

$$J(\mathcal{M}_i) = \hat{\epsilon}_i \frac{(1 + \gamma\sigma_i)}{\left(1 - \frac{\beta P_i}{N_i}\right)}, \tag{5.13}$$

where $\hat{\epsilon}_i$ is as defined in equation (5.10). N_i can be derived as in equation (4.7).

5.4 The Constructive Algorithm

The weight optimisation technique described above should find the best parameters $\boldsymbol{\theta}$ to fit the data, given the fixed basis functions. The difficult part, however, as for conventional Basis Function nets, is the adaptation of the number, position and size of the basis functions. As with conventional LMN's, the problem space is initially partitioned using the *a priori* knowledge available. Local models can be either simple linear models, nonlinear models, or partially or fully parameterised models. This acts as the first approximation to the system being modelled.

The method used for extending the hierarchical method was, once the current architecture has been optimised and tested, and had not yet met the tolerance requirement, to iteratively refine an existing local model by splitting its input space and replacing it with a sub-network containing two local models.

As in conventional decision tree theory, there are a variety of methods used for construction of the hierarchy, e.g. multiple-step look ahead or blind search using a fixed number of nodes. We use a simple method which at each construction stage adds children to the most promising (i.e. where the extra representation is most needed) leaf in the tree.

Which local model should be split?

The choice of where to extend the model by splitting a sub-model is a crucial one, to which there is no generally correct answer. The choice must be made efficiently to minimise the training error, while still leading to good generalisation. The method used is to define a cost-complexity function which evaluates the potential benefit brought by a particular split, allowing the system to choose the split which minimises the function most. The cost-complexity function is designed to provide a compromise between model complexity and accuracy. The cost term tries to decrease the network's error bias for the given training data, while the complexity penalty decreases the network's error variance, so that it will be able to generalise properly.

The work in decision trees produced a variety of such functions. These systems usually made axis-orthogonal splits, meaning that not only should a leaf be chosen for splitting, but the dimension to be split should be chosen as well. In many cases, each possibility was tried out and the resulting cost-complexity measure recorded, the largest one then becoming the new node. The technique used in this work is different – a node is selected for splitting where it seems most needed (where the cost-complexity function, as described in Section 5.3.1, at the leaves is greatest). The angle of the split through the input space is then optimised locally, thus reducing the computational effort dramatically.

When not to split?

The cost-complexity function encourages the system to grow nodes where they are most needed, and where there is enough data to train the new models. In some cases, though, the chosen node may turn out not to be suitable for splitting for one of the following reasons:

1. There is insufficient data in this area of the input space to robustly train the new local models, given the level of noise and the number of parameters to be optimised.
2. The cross-validation process after training shows that the new models are worse than the old model, indicating that the system has started to overtrain, probably due to insufficient local training data. (This method is expensive, but general)
3. The centres of the new unit are closer than a predefined minimum (can be related to the noise on the input variables, or expected complexity).

In this case, the new models are removed, and the old model is marked as being unsuitable for further splitting.

5.4.1 One-dimensional example

To illustrate the constructive process, a one dimensional version of the 'Squiggle' test function

$$y = \sin\left(10\pi x \sin(x-0.5)^3\right) \tag{5.14}$$

is modelled. Figure 5.4 shows the gradual approximation of the target function.

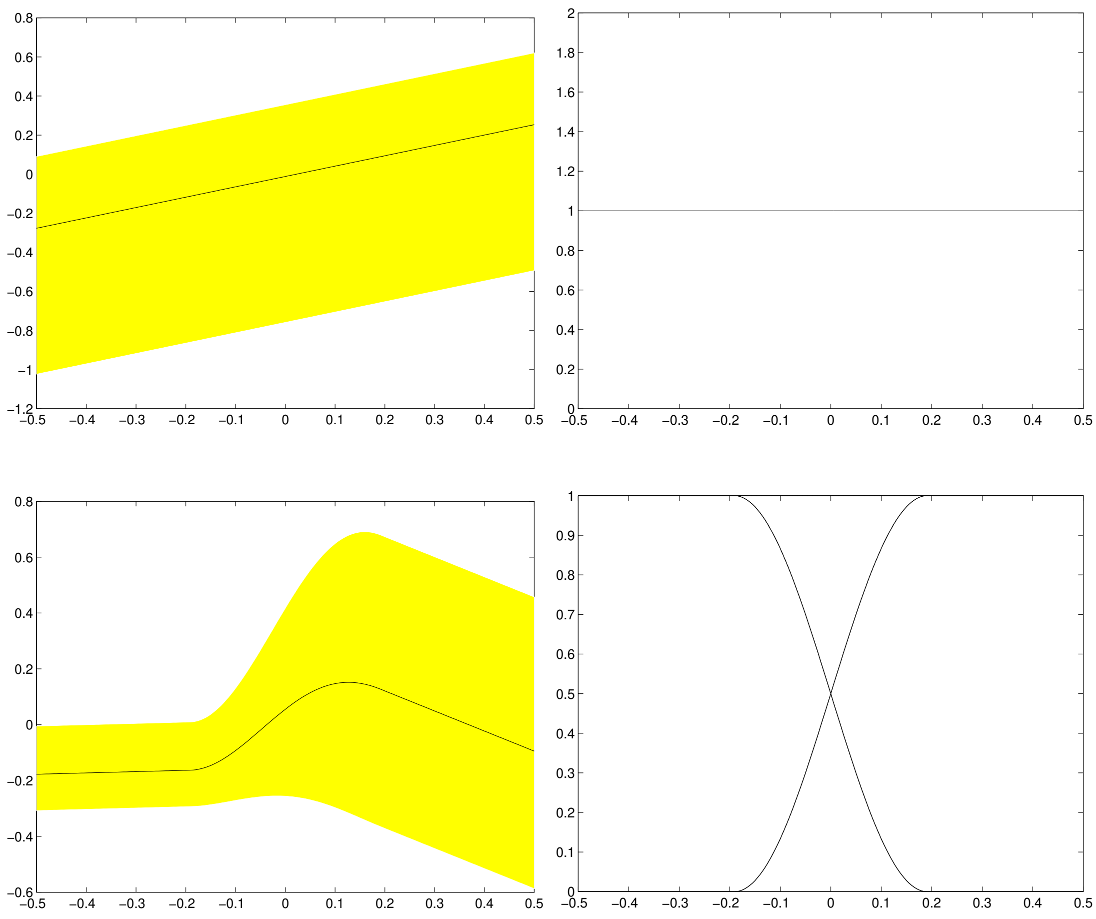


Figure 5.4: Construction of a one-dimensional LHM Model Structure. The Model's response and worst error estimates are shown on the left, and the model structure is shown on the right. The shaded area indicates the model's own weighted worst confidence limits around the estimate.

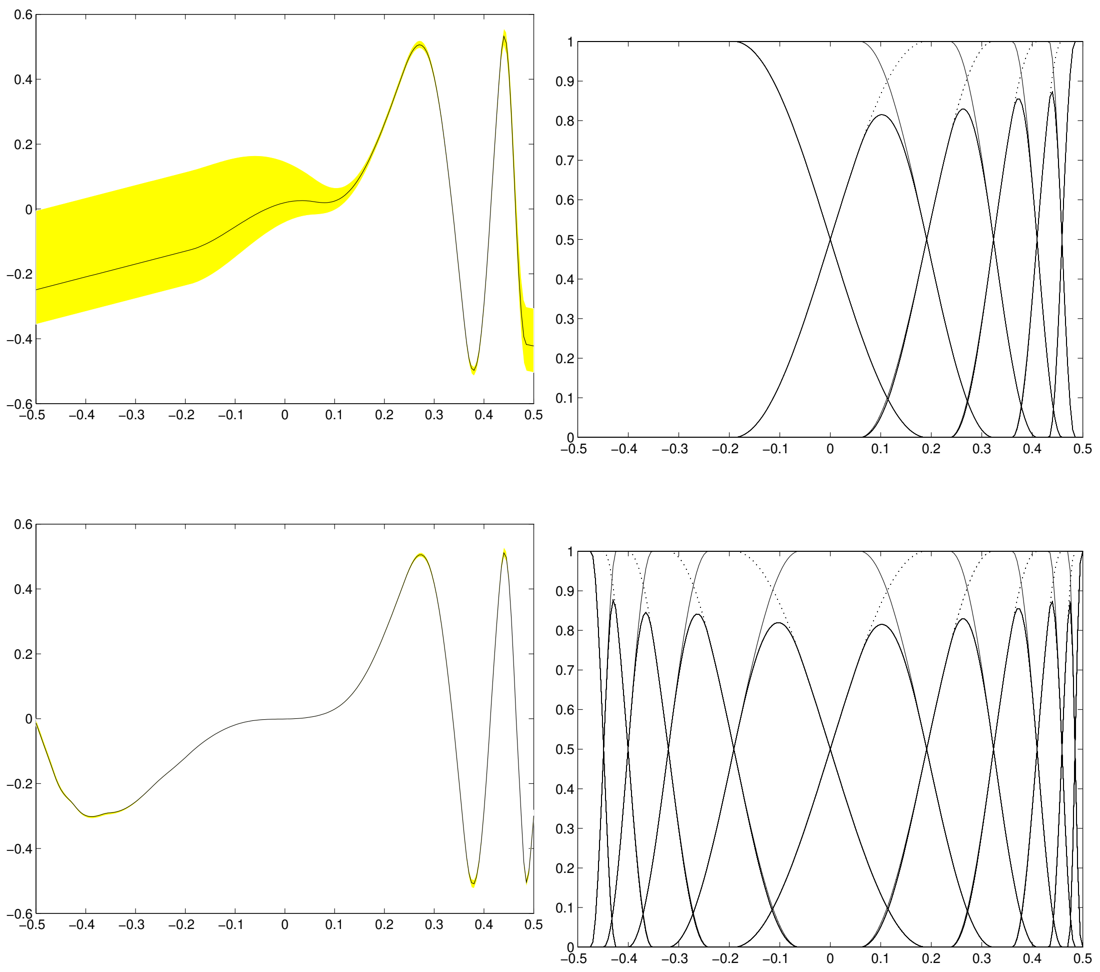


Figure 5.5: Continuation of Figure 5.4.

5.4.2 Axis-orthogonal partitions

Finding the optimal partition for the basis functions is a vital aspect of the learning algorithm. The straightforward method, as suggested in (Johansen and Foss, 1994b), is to try all possible axis-orthogonal splits, optimise the parameters and use the split which improves the cost-complexity function most. A more sophisticated method is at each step to have a limited search horizon, where future splits are attempted, and the split which minimises the multi-step optimisation problem is chosen. This is the same basic technique as the optimisation used in (Breiman et al., 1984) or (Friedman, 1991). The smoothness of the split could also be adapted. In the implementations described in this thesis the smoothness, dependent on σ is not adapted. For axis-orthogonal splits σ is chosen so that the space is divided using the same proportions at each step, forcing a continuous increase in the absolute 'sharpness' of the splits in any given dimension as submodels are added.

The restriction of splits to hyperplanes orthogonal to the axes makes the method simple to implement, but it scales up poorly to higher dimensions and more complex models, especially when look-ahead search is used. It is also difficult to have truly hierarchical learning in such a system, as this requires a constant cycle of pruning and construction of nodes[2], with no opportunity for gradual optimisation of parent nodes' splits, as learning progresses.

5.4.3 Axis-oblique partitions

Splits may be axis-oblique, i.e. the split depends on several variables, which also brings a number of problems. For axis-orthogonal splits, the width of the soft split could be easily determined by the previous splits. For oblique splits, there is no easy analytical way of determining the width of the split. The method used here estimates the separation between the centres using the available training data in the local model's receptive field, and its distance from the separating hyperplane. The covariance $D(\tilde{\boldsymbol{\phi}})$ of the points from the separating hyperplane $\mathbf{a}\tilde{\boldsymbol{\phi}} = 0$ can then be measured and the separation of the children set proportional to the standard deviation, i.e.

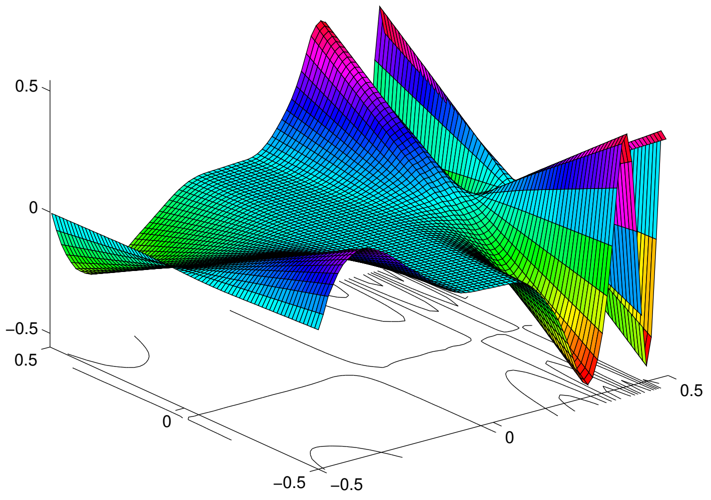
$$\sigma \propto \left| D(\tilde{\boldsymbol{\phi}})\mathbf{a} \right|, \tag{5.15}$$

and the centres can then be set in a similar manner,

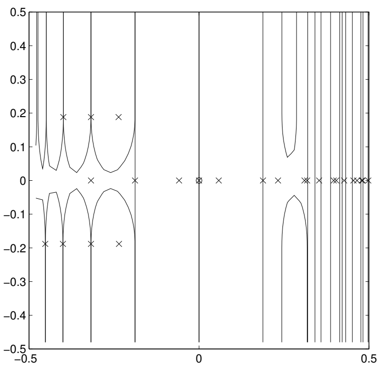
$$\mathbf{c}_l = \mathbf{c}_p \pm \mathbf{a}\sigma, \tag{5.16}$$

where $\mathbf{c}_p$ is the parent's centre, and $\mathbf{c}_l$ is the new centre. A more general approach is to allow full flexibility in the partitioning of the input space.

[2]pruning is necessary, otherwise early 'mistakes'–non-optimal splits–lower the quality of the final results drastically, as they dramatically reduce the amount of training data available to their children



(a) LHM Response



(b) LHM Basis Functions. Contours are drawn at 0.5

Figure 5.6: Squiggle results for axis-orthogonal LHM. The function is easily approximated due to the limited nature of the model structure being well-suited to the target function. The contours (0.5) of the leaf models' basis functions are shown, to give an impression of the model structure found.

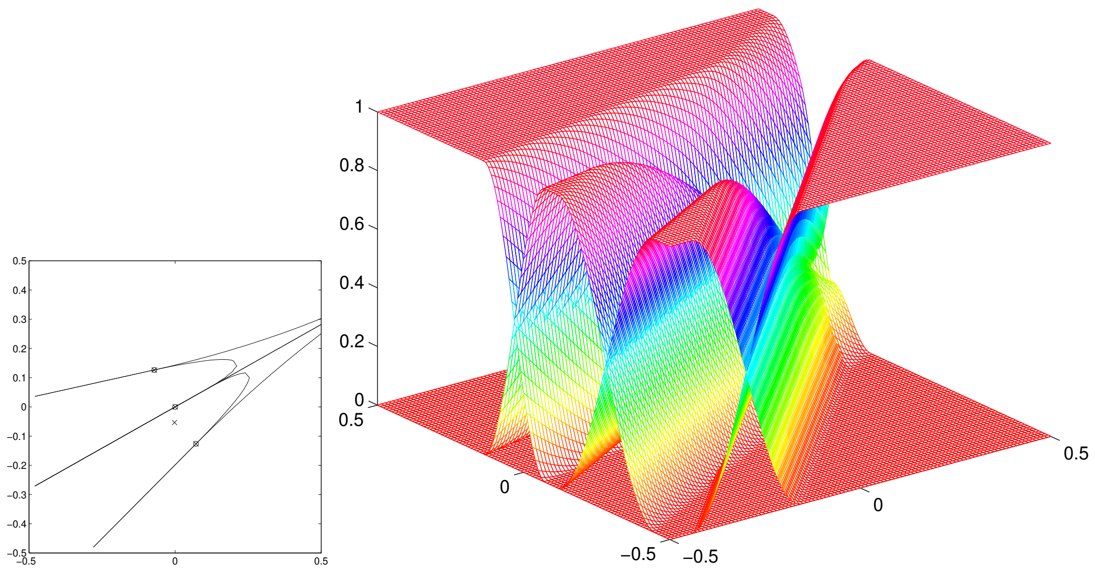
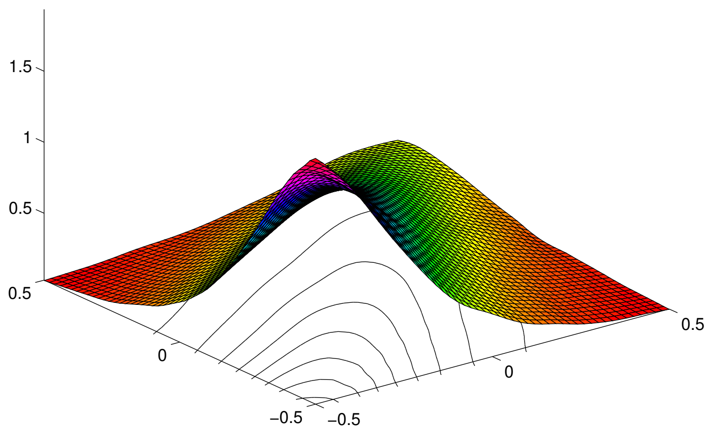
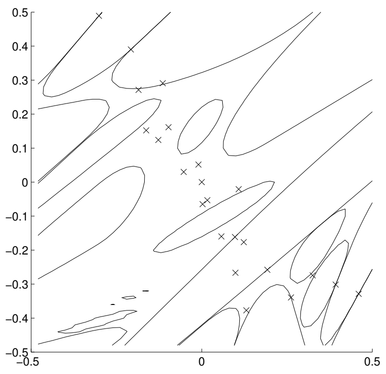


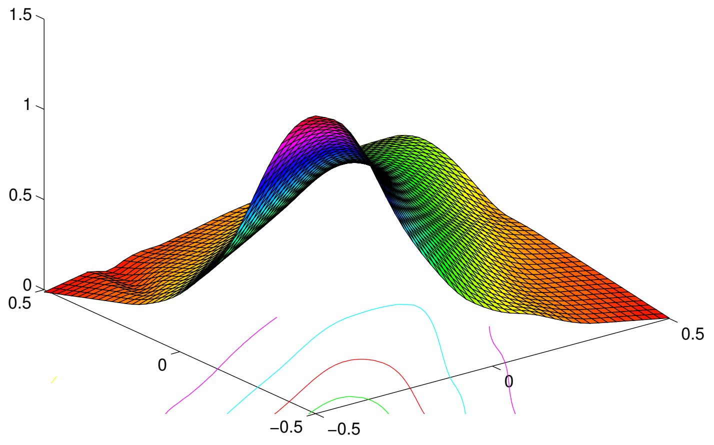
Figure 5.7: Contour and 3d representations of leaves of model tree for axis-oblique partition.



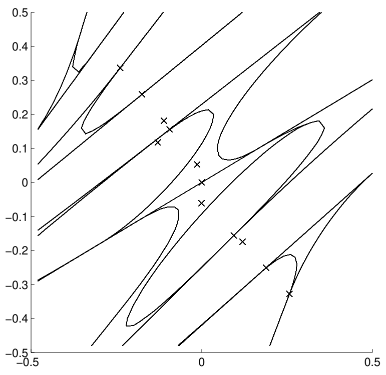
(a) LHM Response.



(b) LHM Basis Functions. Contours are drawn at 0.5.



(c) LHM Response for noisy data.



(d) LHM Basis Functions. Contours are drawn at 0.5

Figure 5.8: Mars1 results with LHM and axis-oblique partitions.

Optimisation of the new split

Partitioning the input space orthogonal to the axes is a useful restriction of the search space in many applications, but it is restricted in the types of system it can model well, and there is no possibility for gradual changes in hierarchical manner. Increasing the freedom of the partition should give the model more flexibility to better model systems where the nonlinearity depends on a combination of variables.

The use of smooth splits allows a variety of gradient-based optimisation algorithms to be applied to find the optimal partition, which is not possible in the classical decision tree methodology due to their crisp partitions. For simplicity, however, the method used in this thesis is *simulated annealing* (Kirkpatrick et al., 1983) to optimise the split parameters. The vector **a** is initialised to have the same value as the parent split (making the useful assumption that the direction of greatest nonlinearity is probably not going to change dramatically). The split angle is then randomly altered,

$$\mathbf{a(t+1)} = \frac{\mathbf{a(t)} + \boldsymbol{\delta}(t)}{||\mathbf{a(t)} + \boldsymbol{\delta}(t)||} \tag{5.17}$$

the width determined, as in equation (5.15) from the local data, and the local models are locally optimised using SVD. The potential split is evaluated on test data by treating the

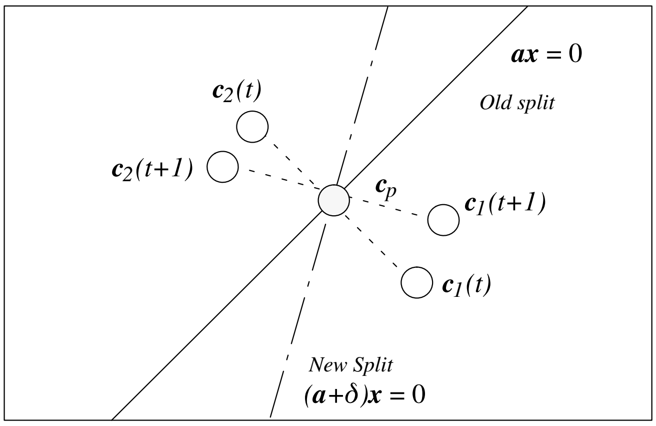


Figure 5.9: Adjusting the split angle. The split is rotated around the centre by the random variable $\boldsymbol{\delta}$. The local models are trained, and if the new split provides a better model, this is chosen as the starting point for future splits. As time passes the changes become smaller, until the split converges.

two local models as a representation of the system locally (the cumulative basis function of the parent model is used to weight the validation results). The best split is then used as the starting point for further random alterations. The random $\boldsymbol{\delta}$ vector responsible for the alterations initially allows large changes – equivalent to high temperature in the annealing analogy – but the 'temperature' T is gradually reduced, until the split has converged to a steady position. The process can be visualised as randomly rotating a hyperplane around the parent's centre. The annealing schedule used in this work was to have the temperature related to an exponential decay, which was related to the number of iterations desired before adaptation was completed,

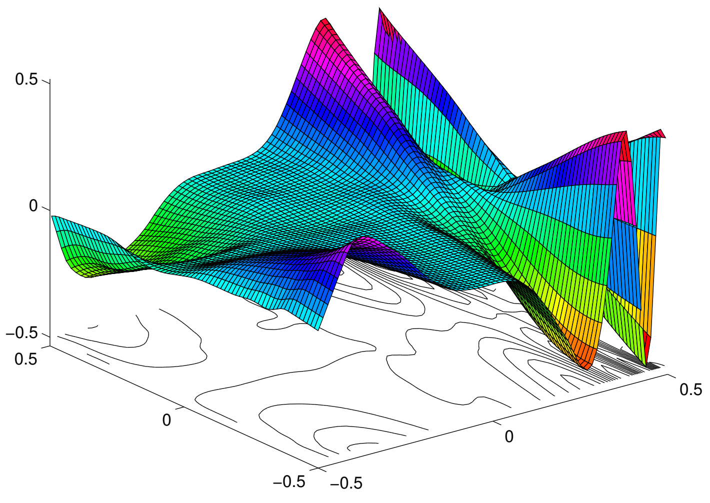
$$T(t) = \exp\left(-4\frac{t}{t_{max}}\right) \tag{5.18}$$

i.e. each element of $\boldsymbol{\delta}$ is a normally distributed random variable with variance $T(t)$ proportional to the exponentially decaying annealing schedule

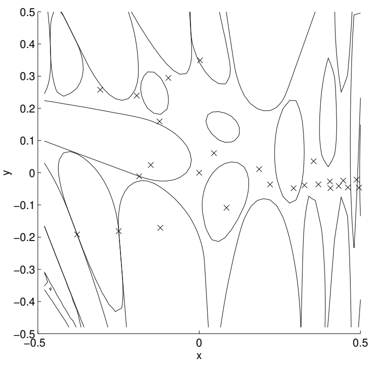
$$\boldsymbol{\delta}(t) = N(0, T(t)), \tag{5.19}$$

where t indicates the iteration, from 1 to t_{max}. The use of linear regression methods, and only the local data points means that the learning process can be quite fast, despite the simplicity of the optimisation technique.

The results for the Squiggle benchmark are shown in Figure 5.10. Note that despite the increased degree of freedom in the model structure, the learning algorithm copes well and produces a good model. Compare the results here with those for the local model net in Figure 4.11(a), which required 62 local models to achieve a lower accuracy. The real benefit comes with non-axis-orthogonal nonlinearities such as those shown in Figures 5.8 and 5.11.

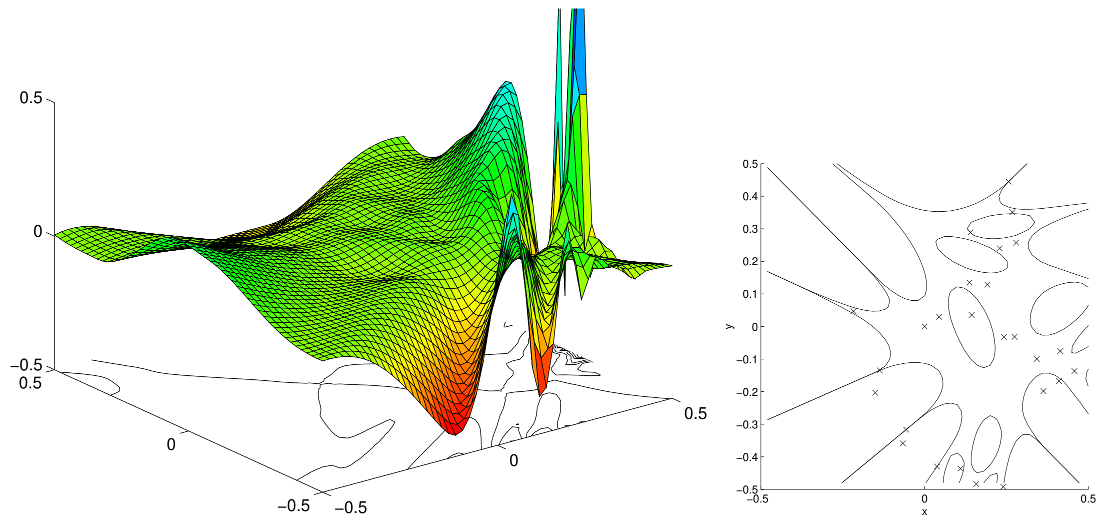


(a) LHM Response



(b) LHM Basis Functions. Contours are drawn at 0.5

Figure 5.10: Squiggle results for axis-oblique LHM. Despite the extra degrees of freedom the learning algorithm does well.



(a) LHM Response

(b) LHM Basis Functions. Contours are drawn at 0.5

Figure 5.11: Rotated Squiggle results for axis-oblique LHM. Compare with Figure 4.19 where LMN and MARS responses are given.

5.5 Conclusions

The Learning Hierarchies of Models structure is a new architecture which is well suited to constructive learning algorithms. There are many overlaps with decision trees in symbolic machine learning, but the advantage of the LHM architecture lies in the soft-splits of the input space, and the local models in the leaves, making the architecture more suitable for the representation of continuous dynamic systems.

Due to the hierarchical nature of the model structure, where local models can hide entire sub-trees of other models, it is well suited to constructive structure identification algorithms. Unlike in flat local model nets, where the addition of a new model means that the neighbouring units must adjust their basis functions and parameters, the replacement of a given leaf model in an LHM does not affect the other models in the structure. Also, because of the partition of unity inherent to the soft splits, normalisation is not necessary, so the problems found in Section 3.3.1 with normalisation are no longer relevant.

The axis-oblique splits used to partition the input space are an important feature of the LHM architecture, as they differentiate the methods more from conventional methods which partition the space using only one variable at a time. Simulated annealing proved to be useful, despite its simplicity. Other more sophisticated optimisation algorithm may produce better results.

As the nonlinearity is achieved by splitting the input space, the hierarchical extension of the local model framework has the potential to produce more efficient methods for representing and identifying unknown model structures. It copes better with high-dimensional spaces than non-hierarchical methods.

The local confidence estimates and the local learning methods can be easily integrated into the hierarchical structure. This also provides interesting potential for controlling the level of locality of the local learning, and active learning through the hierarchy.

Future work

The most straightforward extensions to the theory described here would be to the structure adaptation algorithm so that more flexible structures are created. e.g. optimisation of the separation of child nodes, use of smarter optimisation algorithms for the partitioning angles.

A major advantage of axis-oblique partitions with the hierarchical architecture is the potential for *hierarchical structure adaptation.* This would mean that once a model had grown children, and had improved its representation of the system, the partitioning decisions made earlier could be adjusted to take account of the better understanding of the underlying system. If this is done gradually during the learning process a more efficient, parsimonious and interpretable approximation should be possible. Such hierarchical structure identification and adaptation is obviously also of great interest in adaptive control situations.

The binary tree LHM structure with homogeneous linear local models developed in this thesis is the most basic form of the structure but the framework can be viewed in a more general manner, if the interface between the child and parent is made more powerful. The task of the sub-model can be viewed as being exactly the same as that of the original problem, just at a smaller spatial scale, with a reduced portion of the data set. This means that all of the considerations used initially in framing the modelling problem are also relevant at this level (i.e. which inputs are important? can the problem be decomposed into an additive model of several input spaces? if the system is a dynamic one, what model order should be used, what sampling speed should be used?). The concept of hierarchy in the time-domain could be used to produce hierarchies of models working at different sampling speeds, allowing the system to cope with stiff systems with a wide range of time constants. Different types of hierarchy from the simple binary tree could also be considered. If this were linked with the more powerful child/parent interface and heterogeneous local models (which could be dynamic models, expert systems, etc.), the LHM has the potential to become a much more general representation than simply a method for representing nonlinear systems.

Chapter 6

Rolling Mill & Robot Actuator Modelling Examples

To show the methods in use for practical applications, the local model networks and learning hierarchies of models networks are applied to real problems, with data measured from physical systems.

They are used to produce a predictor which can model the roll gap in a rolling mill based on data sampled from a real mill. The Local Model net produced the best results in terms of accuracy, and generalisation ability, compared to multi-layer perceptrons, MARS, linear models and RBF networks. The interpretability of the trained network is discussed and visualised.

The methods are applied to a second example, a robot actuator modelling problem.

6.1 Rolling Mill Problem Description

To demonstrate the practical applicability of the modelling structures, methodology and algorithms developed in this thesis, the methods were applied to the task of modelling a rolling mill from real data. Rolling mills, are examples of complex nonlinear processes, where a wide variety of physical effects play a significant role. Despite the obvious usefulness of rolling mills, and long practical experience with their operation, the physical models developed for them tend to be highly complex (too complex for on-line use), and because of the uncertainties in industrial processes the models tend to be of little practical use. In practice, the control algorithms used in real mills tend to be simple enough to be adapted by the operators, and are often developed in a very heuristic manner. This suggested that the use of learning methods to better model the poorly understood characteristics of the rolling process would be a productive path to follow.

The mill investigated is shown in Figure 6.1, and is a four-high single stand aluminium rolling mill. A coil is placed on the left and the material is threaded into the roll bite, where the material is reduced and exits to be collected again in a coil on the other side.

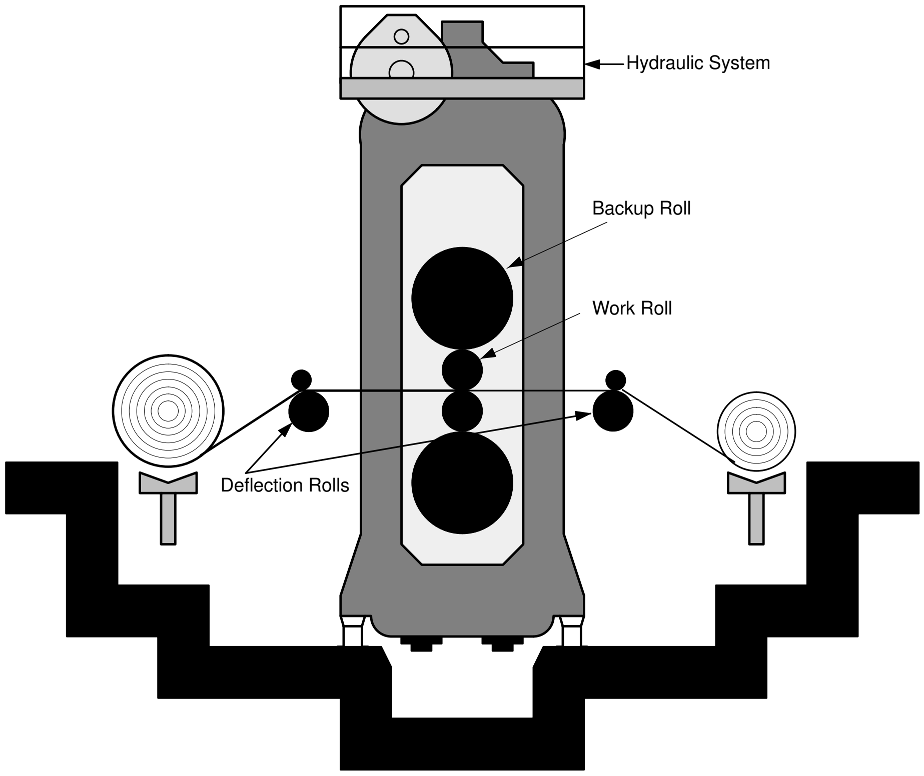


Figure 6.1: Single stand of a rolling mill.

The strip moves with a velocity v_1 and enters the roll gap with a thickness h_1. It exits with a thickness h_2 and a velocity v_2, preserving the mass-flow relationship,

$$h_1 v_1 = h_2 v_2. \tag{6.1}$$

The strip is deformed by two cylindrical rolls which apply a force f_w to the material to produce a flat strip of a pre-determined thickness, as shown in Figure 6.2.

The roll gap is varied by exerting force on the backup roll bearing. This force is transferred to the centre of the backuproll, to the centre of the work roll and then to the strip. Flat strip is produced when the force on both sides is equivalent. Roll gap position is measured on both sides and is averaged to give the *roll gap* variable s_0. The setting for the roll gap is a function of strip input thickness and the elastic deformation of the surrounding mill housing and roll bearings (shown in Figure 6.2 as springs).

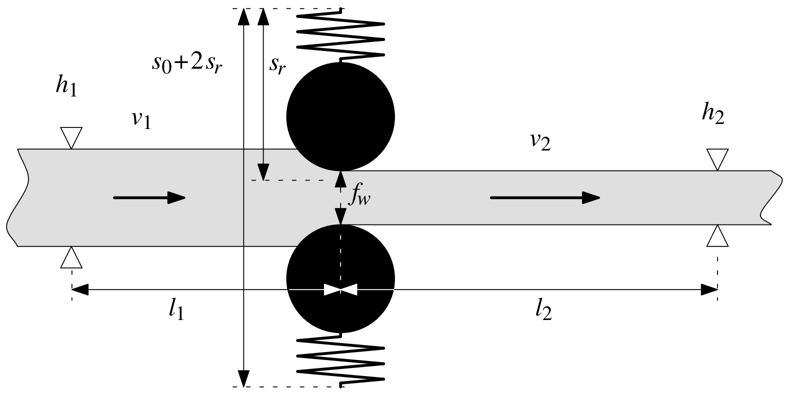


Figure 6.2: Roll bite

6.1.1 Nonlinearities in the rolling process

The rolling process is nonlinear in a number of ways. The deformation characteristics of the rolled material, the work rolls and the surrounding mill housing are nonlinear. The material undergoes elastic deformation, then plastic deformation, followed by elastic relaxation. This, combined with the elastic deformation of the work rolls and mill housing, is a process which is very hard to describe with simple differential equations. Further complexity is added by the effect of variations in temperature on the process (materials change their properties, the rolls swell). Lubrication effects further confuse the situation, and are very difficult to model from first principles.

6.1.2 Measurement noise and disturbances

Many of the disturbances in the modelling process are obviously closely related to the complex, unmeasured and poorly understood processes described above. As can be seen in Figure 6.2, the sensors for the measurement of h_2 are a distance l_2 away from the roll bite, leading to a dead time between actuation and sensing which varies with operating speed v_1. The sensors are usually X-ray gauges measuring the thicknesses h_1 and h_2, and these measurements tend to be very noisy. The rolling force can be measured either directly (the more accurate way)by load cells, or indirectly by using pressure transducers from the hydraulic cylinders and converting this quantity to force. The methods used in this application are the *indirect* methods. The strip velocities are calculated from the deflection rolls, and the work roll velocity is calculated using the tacheometer of the drive system. In each case the measured quantity is the angular velocity, which when combined with the roll diameter can give the actual roll speed. Due to the complex nature of the disturbances and the poorly understood interactions, it did not seem promising to build explicit disturbance models.

6.1.3 Modelling goals

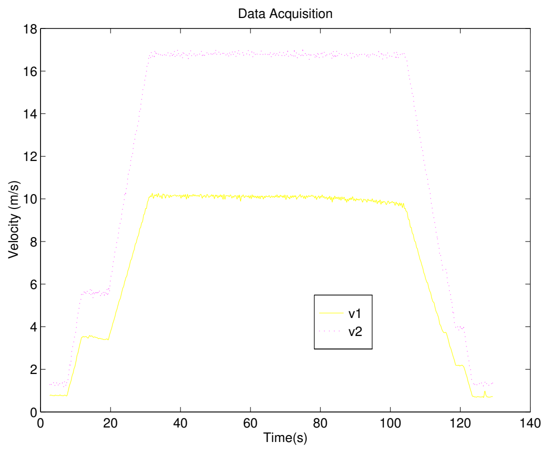
The goals of the learning task were therefore to try to produce a model of the system which reliably represented the data, and which could be used in a model-based control algorithm. The variable of interest is dh_2, a measure of the deviation from the reference output thickness. At present, the acceleration and deceleration phases are poorly modelled and controlled, so an improvement in model accuracy in those phases would be a major contribution to making the rolling process more economic by avoiding wastage caused by roll material which does not meet the accuracy specifications. In the general case it is obviously desirable to have a system which can cope with different types of metal and different reduction schedules. One possibility is to hand segment the training data into different training sets for different types of metal, then to bring them together committee-style using a classification network to decide which model matches the current strip best. Another possibility is to amalgamate a variety of independently trained models, to try to produce an 'average' general model. The method used in this thesis, however, concentrates on producing a single model from a training set composed of a variety of training runs, from a single type of strip.

Important areas of the input space

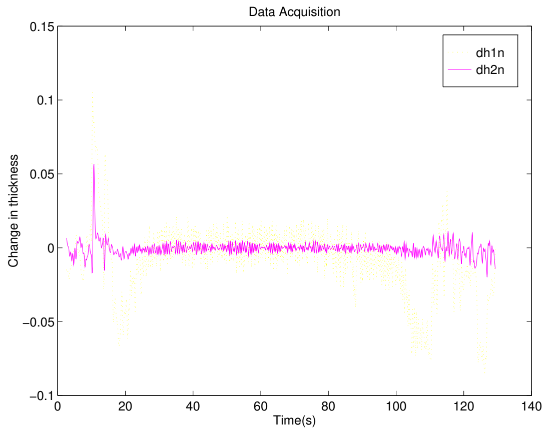
In many modelling applications there are given areas of the input space which are especially important to model accurately, because the model-based system needs particular accuracy in a given situation, bandwidth or region. In the rolling process, the long periods of relative stability, where the velocity does not change, are obviously important, as despite the fact that the modelling here is easier, this phase is where the mill produces most of its product. The acceleration and deceleration phases are, however, important because it is much more difficult to achieve an accurate model, so more data is needed to produce robust accurate representations of the physical process. The trade-off between the two areas is one for the engineer to decide on. (A further possibility is of course, to treat the three areas separately, by segmenting the training data, and associating a local model with each phase of the operation)

6.2 Rolling Mill Training and Test Data

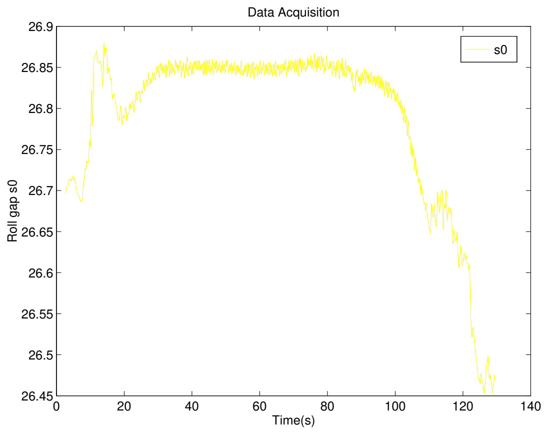
We have described the complex nature of the problem, which makes it clear that there is little hope of a clean, uniform, conflict free training set. Each run of a rolling mill is going to produce slightly different results, many of the reasons for which cannot be directly measured (e.g. temperature differences, different roll materials, possibly different machine settings). To give an impression of the distribution of the data in the input state-space, the correlation between various inputs is shown in Figure 6.4. To improve the numerical robustness of the learning process, the data was normalised so that each variable was distributed between 0 and 1. The normalised data are shown in Figure 6.5.



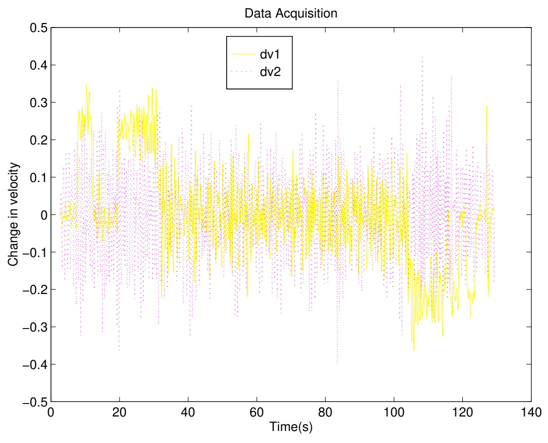
(a) Mill Velocity (Input/Output)



(b) Strip Thickness (Input/Output)

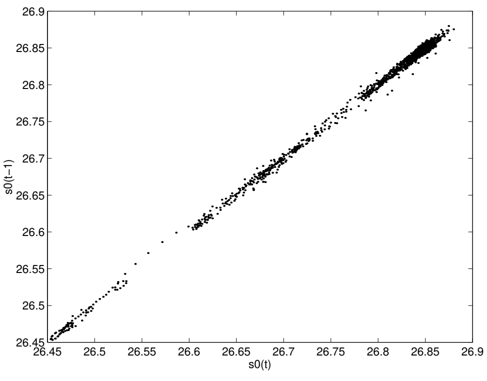


(c) Roll Gap

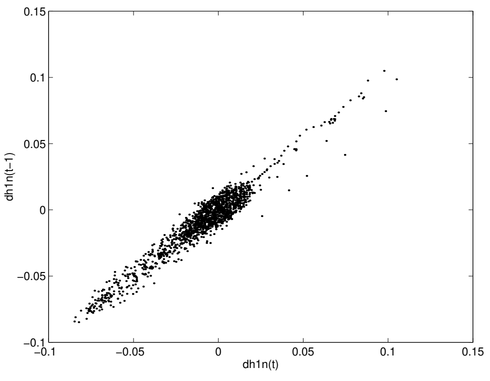


(d) Velocity derivatives

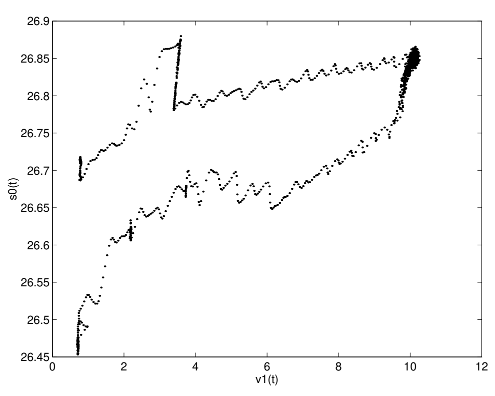
Figure 6.3: Observed data from mill



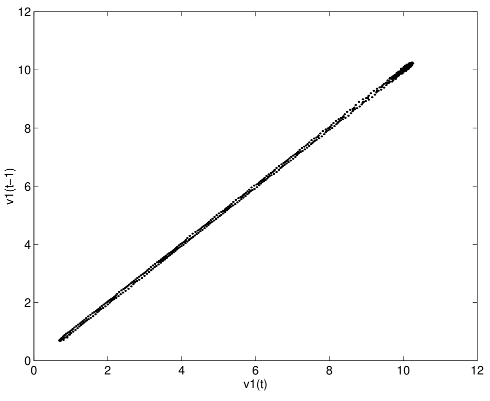
(a) Roll gap (s_0) correlation



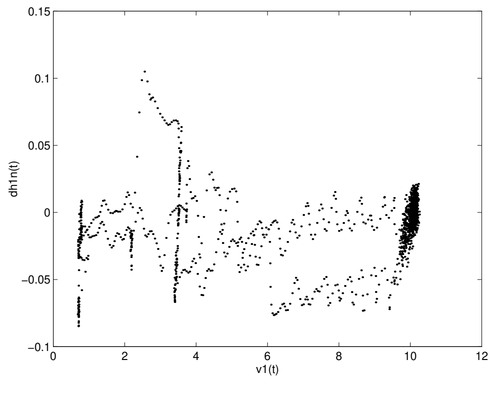
(b) Input thickness (dh_1n) correlation



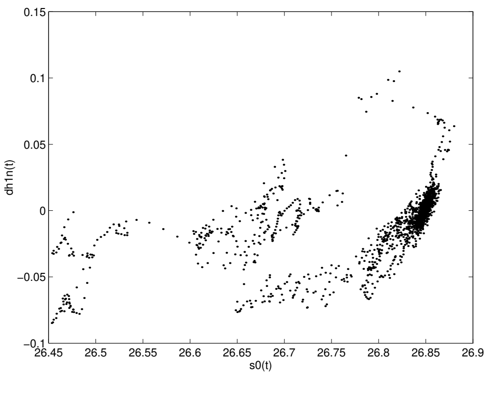
(c) Speed against Roll gap correlation



(d) Input velocity (v_1) correlation



(e) Input thickness (dh_1n) against input velocity (v_1) correlation



(f) Roll gap against Input thickness correlation

Figure 6.4: Observed data from mill – state portraits for a single run

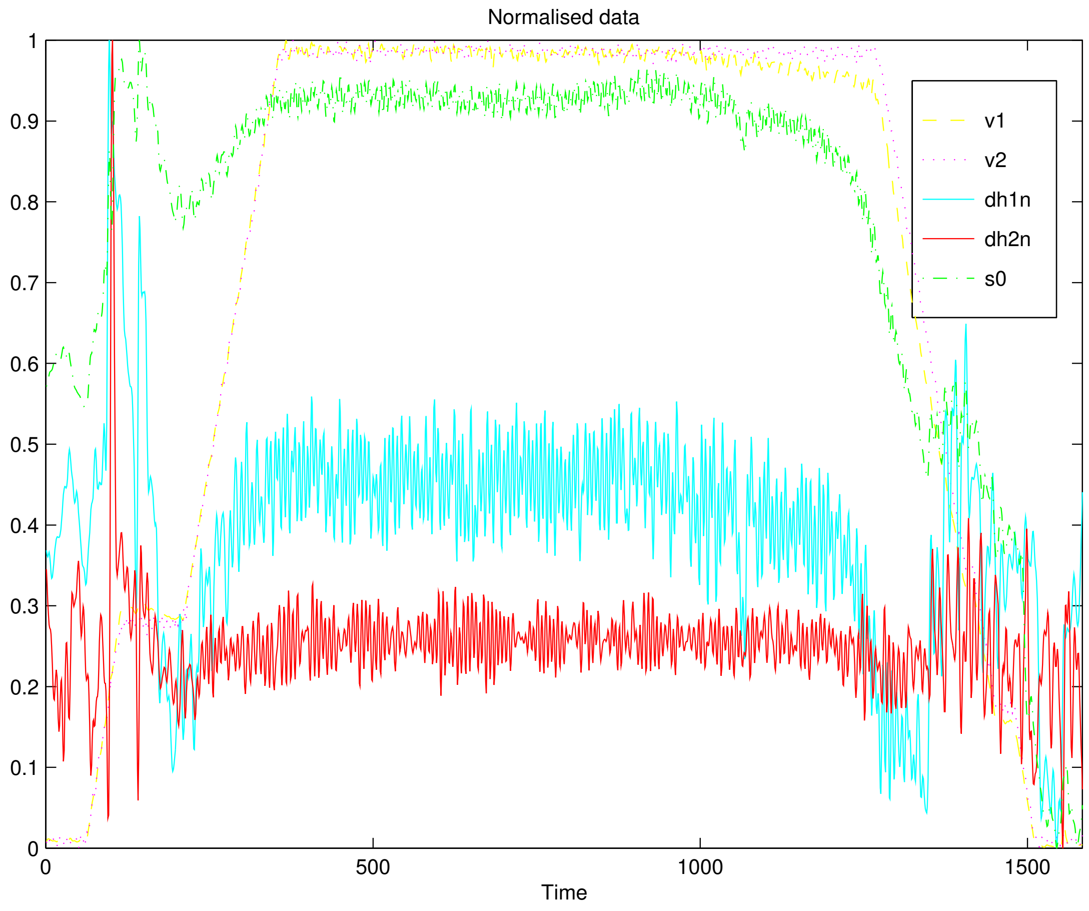


Figure 6.5: Normalised training data

Notation

dh_1	input thickness deviation
dh_2	output thickness deviation
dh_1n	input thickness deviation (time aligned)
dh_2n	output thickness deviation (time aligned)
v_1	strip input velocity
v_2	strip output velocity
s_0	roll gap
f_w	roll force

6.2.1 Pre-processing used

Filtering and time alignment

The data used for modelling is low-pass filtered and down-sampled to a more appropriate frequency for the modelling task. The filtering was done offline using a 32-order FIR filter and the MATLAB *filtfilt( )* routine, which results in zero-phase distortion after passing through the data set in both forward and backward directions.

One of the main difficulties in modelling aspects of rolling mills is that delays in sensor measurements are inevitable, due to the distance of the thickness gauges from the roll bite. These delays are also velocity dependent, so the delay for the forward sensors is

$$t_{d1} = \frac{v_1}{l_1}, \tag{6.2}$$

where v_1 is the input velocity, and l_1 is the distance from the roll bite, as shown in Figure 6.2. To make the modelling task more straightforward, the data was pre-processed so that it would be referenced to the roll bite, such that the new thickness dh_1n is given by

$$dh_1n(t) = dh_1(t - t_{d1}), \tag{6.3}$$

where at the sampling rates used, t_{d1} varies from eight sample delays at low speed, to 2 sample delays at top speed. The output thickness is analogously,

$$dh_2n(t) = dh_2(t + t_{d1}), \tag{6.4}$$

where $t_{d2} = \frac{v_2}{l_2}$. This time shifting was validated by checking correlations between the thickness variation and the roll force signal, confirming that the time-shift was accurate to within one sample.

Sampling rate

The mill has a variety of subsystems with different bandwidths–the hydraulic system has a bandwidth of 15-20Hz, the roll eccentricity effects are 12-15 Hz, but the most important effects are dominated by the slowest subsystems. From experience, the sampling time needed to cope with the change in response from the moment of sensing a disturbance, actuating the hydraulics and moving the rolls is between 1-2Hz. This is what was used in the design of the filtering and sampling algorithms.

The original data was sampled at 100Hz. Using the old rule of thumb that the sampling rate should be around 8-10 times the closed loop bandwidth (Ljung, 1987), the data was downsampled to a frequency of $f_s = 12.5$ Hz.

6.2.2 Planning the experiment

The problems of acquiring the data in many industrial processes are non-trivial, involving a great deal of time and money. In this case the data existed already, having been taken from a real plant under normal operating conditions. We had no influence over the experiments performed, and new experiments were deemed too costly to be worthwhile. Four different runs, each similar to that shown earlier were used for training, and the data selected from the combined data sets. The sets used to produce the training data are shown in Figure 6.6. The data are normalised to be within the limits defined by the maxima found in training and test sets. The test data used to validate the trained model were taken from five different runs with the same type of strip. The sets used for the validation are shown in Figure 6.7.

Open questions

The methods used to prepare the training data all seem to be practical measures aimed at a concrete engineering solution, but there are still many open questions. How many data do we require to be confident that our model is general enough? How will the model react to different materials, or reduction schedules? Will hot mills, where the input thickness is more variable need more runs?

6.3 Rolling Mill Modelling Results

6.3.1 Modelling specifications

Model structure and order

The local model structures used were linear models based on delayed inputs (equivalent to an FIR filter structure).

$$f_i(\boldsymbol{\psi}(t)) = \boldsymbol{\psi}(t)\boldsymbol{\theta}_i, \tag{6.5}$$

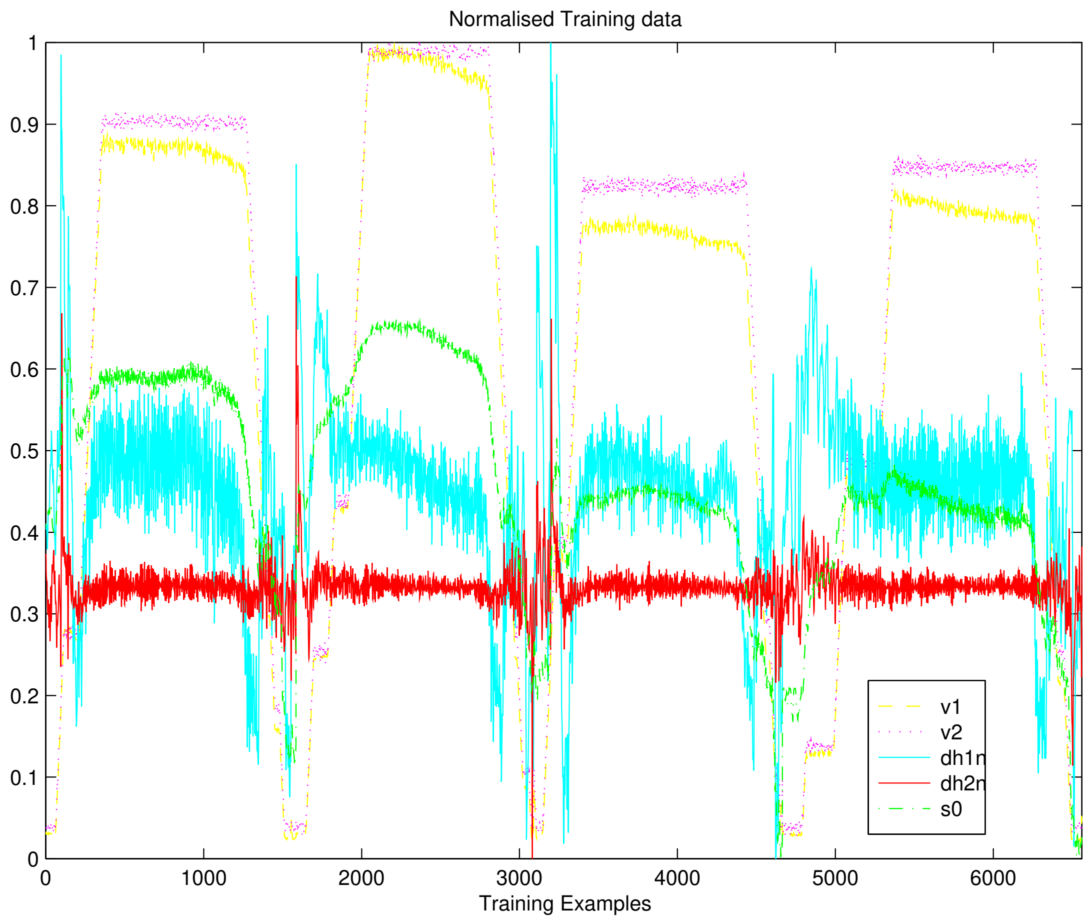


Figure 6.6: Combined training sets

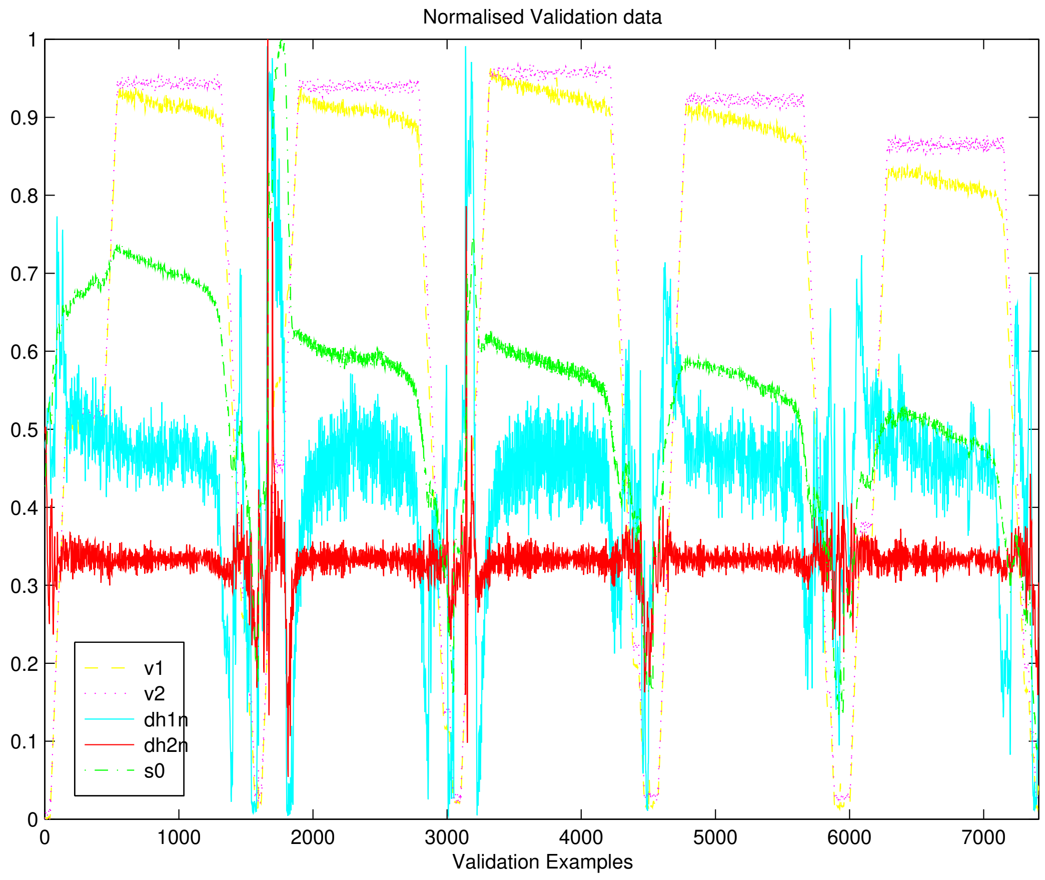


Figure 6.7: Combined Validation sets from 5 different strips of the same material. x-axis indicates time, y-axis indicates the normalised value of the various variables.

where

$$\boldsymbol{\psi}(t) = [v_1(t-1), s_o(t-1), dh_1n(t-1), \ldots, v_1(t-n_u), s_o(t-n_u), dh_1n(t-n_u), 1] \qquad (6.6)$$

The previous measured output thickness could not be used for control purposes, because of the dead time in the output measurement process. The order n_u of the processes involved was not clear in advance, so models with a variety of dynamic orders were used. In general, models with an order of around $n_u = 7$ performed best. It is not clear whether this is due to high order dynamics or low order dynamics with the effect of unknown dead times.

Selection of the cost function for optimisation

The cost function used will depend on the noise on the training data and the relative importance of the area of the input space. In this case we simply applied the quadratic cost function, as the active learning and constructive algorithm would automatically devote more resources to the more complex acceleration and deceleration phases.

Parameter selection for learning algorithm

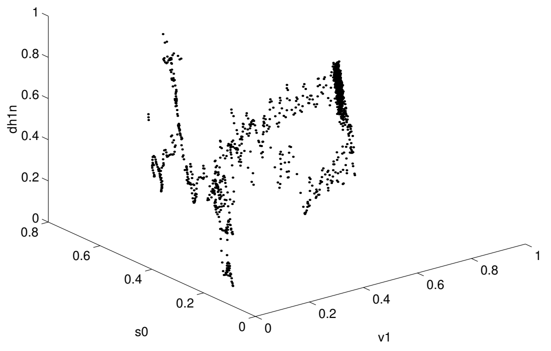
Ideally, the need to fiddle with the parameters of the learning algorithm should be kept to a minimum. In practice, this can prove to be highly important for the modelling process. The results in this chapter with the local model nets were obtained using the following parameter settings: $\sigma_{max} = 0.2\sqrt{n_{\tilde{\phi}}}, \lambda = 0.6, n_{res} = 3, \gamma = 0.5\sqrt{n_{\tilde{\phi}}}, N_{max} = 1000, N_{min} = 10n_{\psi}, n_{cutoff} = 4$. The distance metrics used for the basis functions were axis-orthogonal ellipses. The basis functions in all nets were normalised.

Reducing the operating space

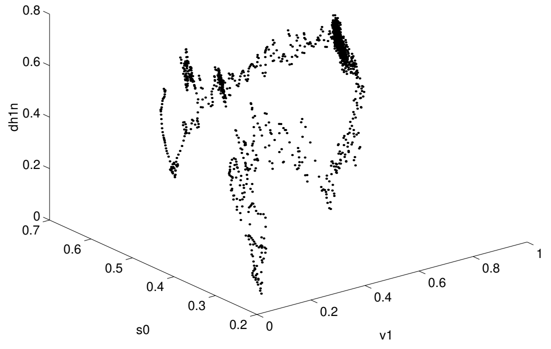
In order to allow a more straightforward and robust model structure, the dimension of the space in which the operating regimes are placed was reduced. The operating point $\tilde{\boldsymbol{\phi}}$ is defined by the inputs s_0, v_1 and dh_1n, so these are the variables used to place the basis function centres. The local models inputs $\boldsymbol{\psi}$ are composed of s_0, v_1 and dh_1n, delayed n_u times, where n_u is the order of the tapped delay line on the inputs. Other variables were not found to contribute significantly to a reduction in the model error.

As mentioned in Section 2.5.4, in some applications it makes sense to form a rough partition of the input space initially, as seems logical to the development engineer. The results in this chapter did not use this technique, although it may prove useful in future work (e.g. to form an explicit partition of the acceleration and deceleration phases from the normal operating speed phase).

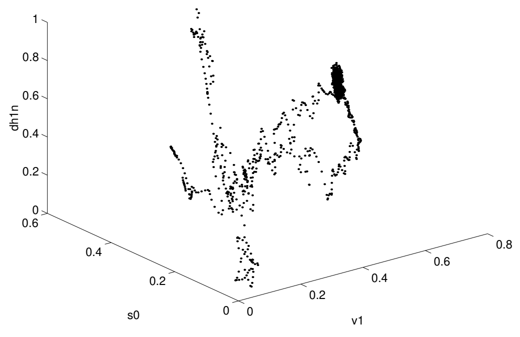
The distribution of the data points in the basis function space (this can also be viewed as the state-space in which the current operating point $\tilde{\boldsymbol{\phi}}$ is defined) is shown in Figure 6.8. The four runs which make up the training set are shown here.



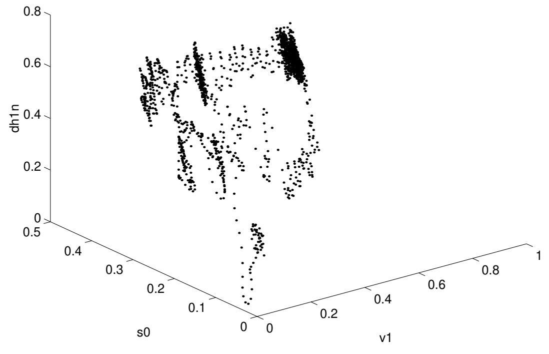
(a) Run 1



(b) Run 2



(c) Run 3



(d) Run 4

Figure 6.8: Training data distribution in operating point space over the four runs. See Figure 6.4 for 2-dimensional slices fo the space.

Use of cross-validation to estimate robustness

The results given in this chapter are all one-step-ahead prediction results. As noted earlier, models of dynamic systems should normally be validated by applying the exogenous inputs and letting the system run, feeding back the model states as opposed to the measured states. For the rolling mill, however, the model is not autoregressive as the dead time involved in the sensor feedback would mean that the data would not be available for use in a control algorithm. The one-step-ahead prediction is therefore the ultimate off-line test in this case. The potential problems involved in on-line application of the model for control purposes are described later.

6.3.2 Benchmark algorithms

Linear modelling results

To provide a straightforward benchmark for the local model nets, we trained a simple non-autoregressive linear model (as in equation (6.5)) using the same data as the LMN. The results were surprisingly good for such a simple structure, as shown in Figure 6.9.

Figure 6.10 shows a cross-validation plot of the linear modelling results, showing that they are consistent over the different folds, as would be expected for such a simple model.

MLP modelling results

To provide a benchmark with the more traditional neural networks methods, a multi-layer perceptron was applied to the problem. A network with a single hidden layer of 30 neurons with sigmoidal activation functions was used. The output unit was a linear one. The training was carried out over 8000 runs through the training set of 6497 patterns, presented in a random order. The straightforward back-propagation algorithm was used with a learning rate of 0.1. (Other settings were tried, but had little effect on the outcome.) The modelling accuracy was found to be not as good as the MRC algorithm's, and training duration was several days, as opposed to minutes for the other methods. An interesting comparison is with the simple linear model, which proved to be more accurate than the potentially powerful MLP. The reasons for the MLP's poor performance were not investigated in detail, but the problem proably lies in the inefficient learning algorithm, back-propagation.

MARS results

As a benchmark with the methods developed in the statistical community the MARS structure identification algorithm was used. The results again, were not as good as the linear, or Local Model or LHM methods, as shown in Figure 6.12. This is probably due to the axis-orthogonal nature of the partitioning of the input space, combined with the lack of local linear models in the standard setup.

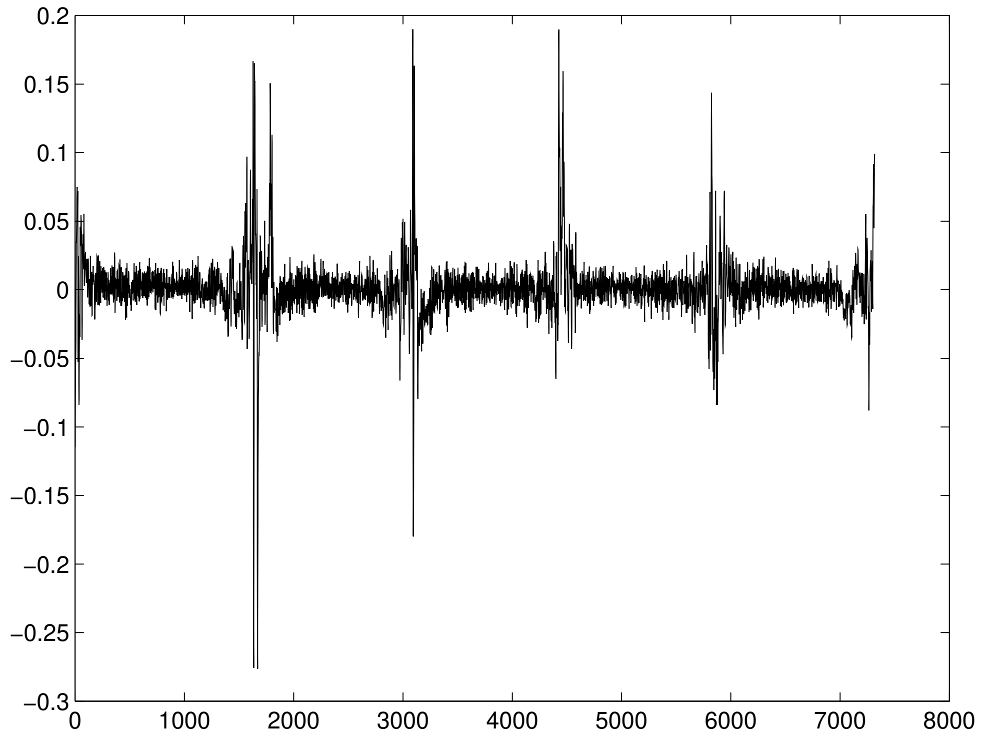


Figure 6.9: Mill model residuals for linear model on the validation data. Mean squared error = 0.00049. The plot shows the error residuals on the combined thickness modelling results from 5 strips used in the validation runs shown in Figure 6.7. The x-axis is time. y-axis indicates magnitude of the residual. Note the large errors in the acceleration and deceleration phase.

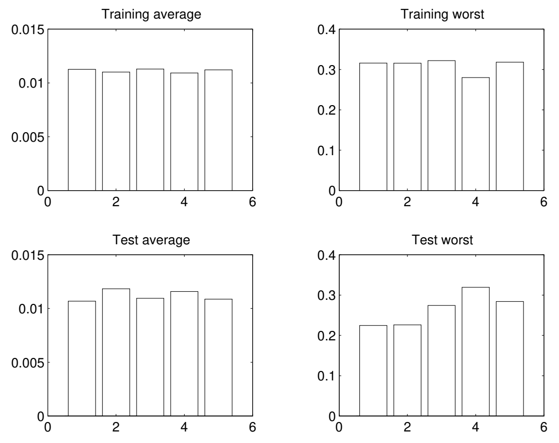


Figure 6.10: Cross-validation results for linear model

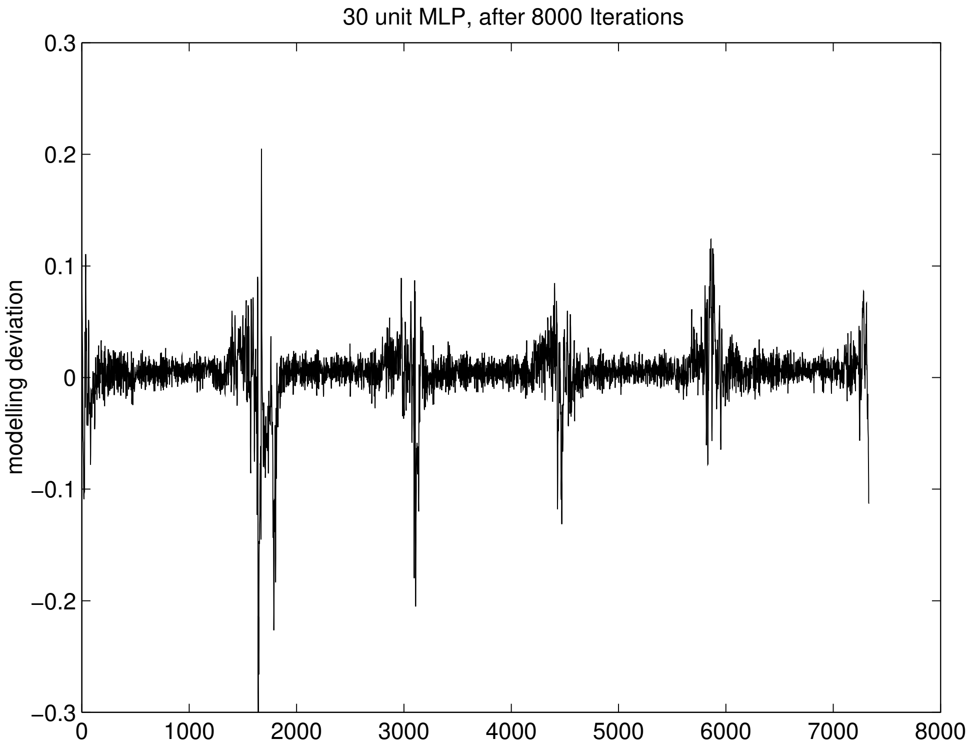


Figure 6.11: Mill model residuals for MLP with 30 units on the validation data. Mean squared error = 0.00065.

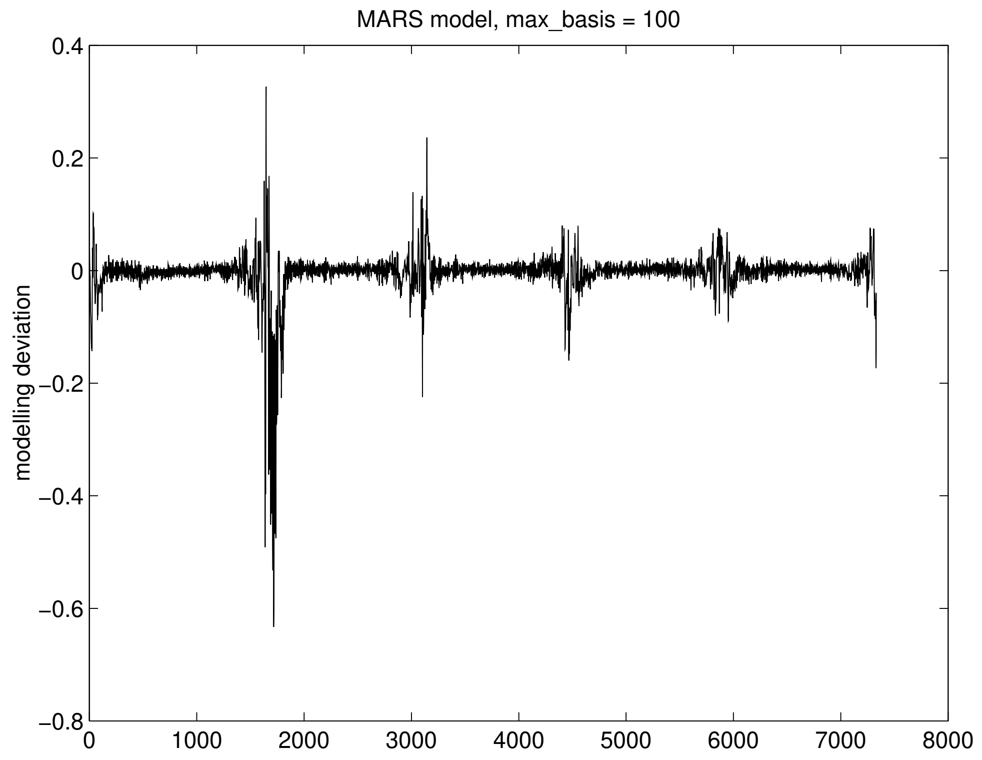
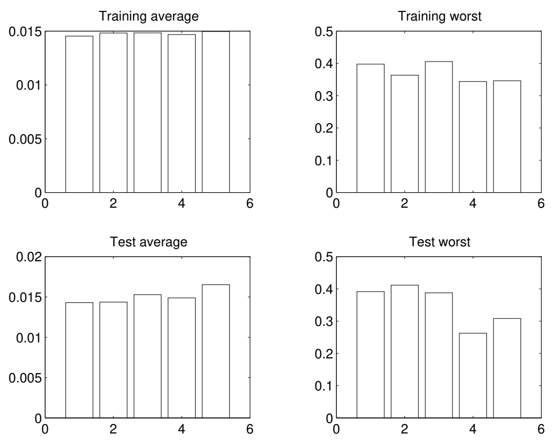
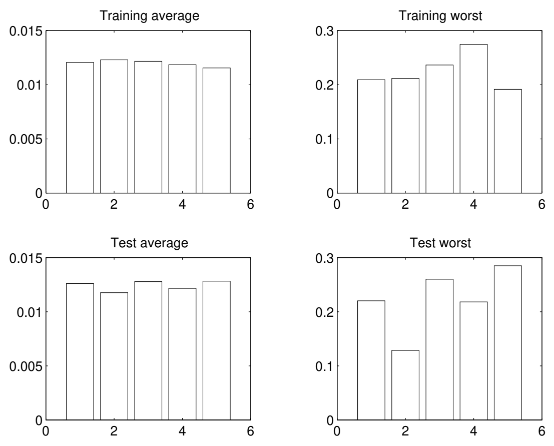


Figure 6.12: Mill model residuals for MARS on the validation data. Mean squared error = 0.0017. Large errors especially in the acceleration and deceleration phase. Model even worse than a linear model.



(a) Cross-validation results for MARS with maximum of 20 basis functions and $n_u = 7$



(b) Cross-validation results for MARS with maximum of 100 basis functions and $n_u = 7$

Figure 6.13: Cross-validation results for MARS on mill data. Reasonably robust with a cross-validation on the training data, but performed badly when face with data from a totally different run.

6.3.3 Local model net results

Local learning

The error results for a LMN with 21 models, trained locally are shown in Figure 6.14

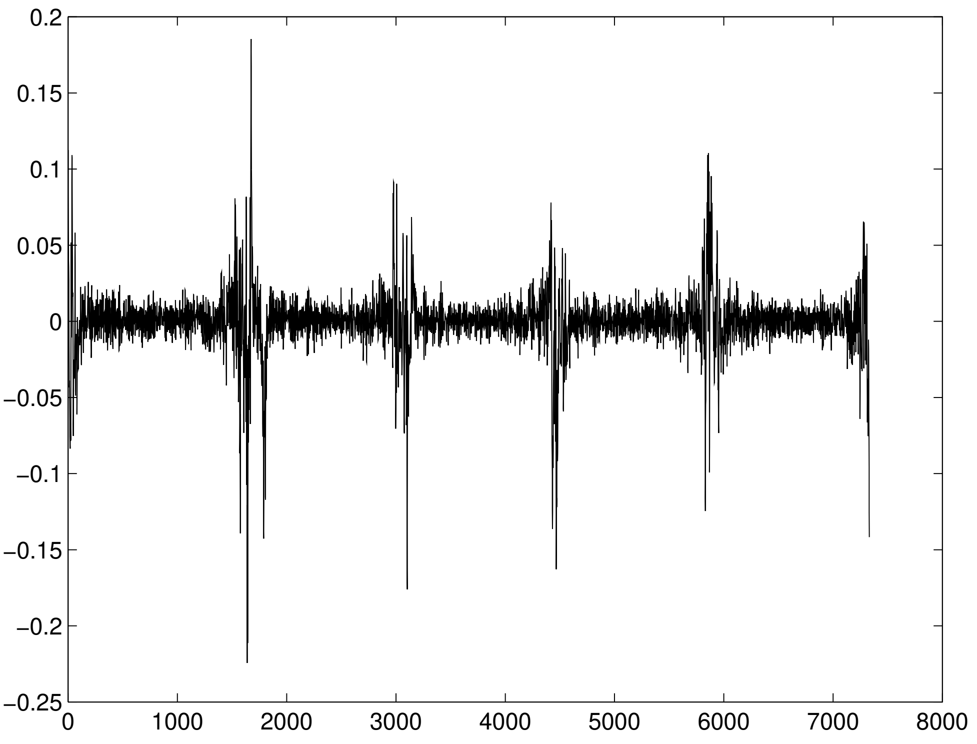


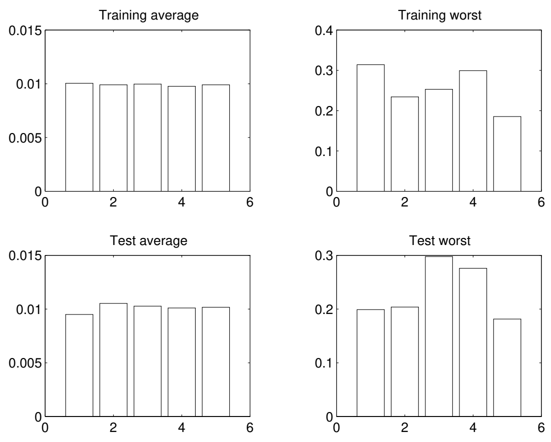
Figure 6.14: Mill model residuals for LMN (local training) on the validation data. Mean squared error = 0.00040.

The mean absolute errors on the training set at each stage of the local learning model construction are shown in Figure 6.15(b). The cross-validation results are given in Figure 6.15(a), and indicate that the approximation process is relatively robust.

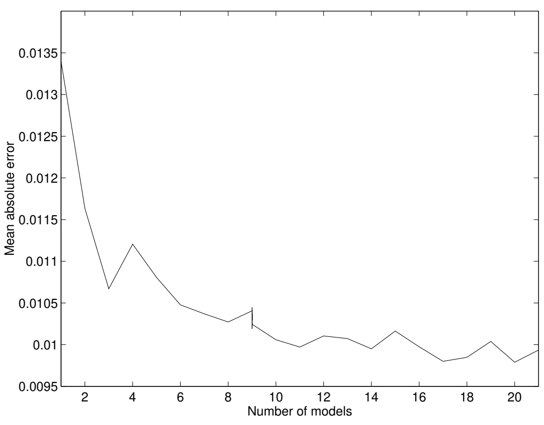
Other parameter settings for learning were used, for example, letting the training continue for a further resolution level, which lead to the smallest training residuals, with an average absolute error of 0.85

Global learning

The error results for a LMN with 23 models, trained globally are shown in Figure 6.14. The global method used was SVD, where singular values which were a factor of 10^4 smaller than the largest singular value were zeroed. Note the poorer performance in the acceleration and deceleration phases, compared to the locally trained model.



(a) Cross-validation results for Local Model Net construction. The results are relatively consistent.



(b) Local Learning Error Curve. The glitch in the graph indicates that a model was removed at this stage, due to a lack of training data.

Figure 6.15: Cross-validation results for net construction, and training error development during a single construction run.

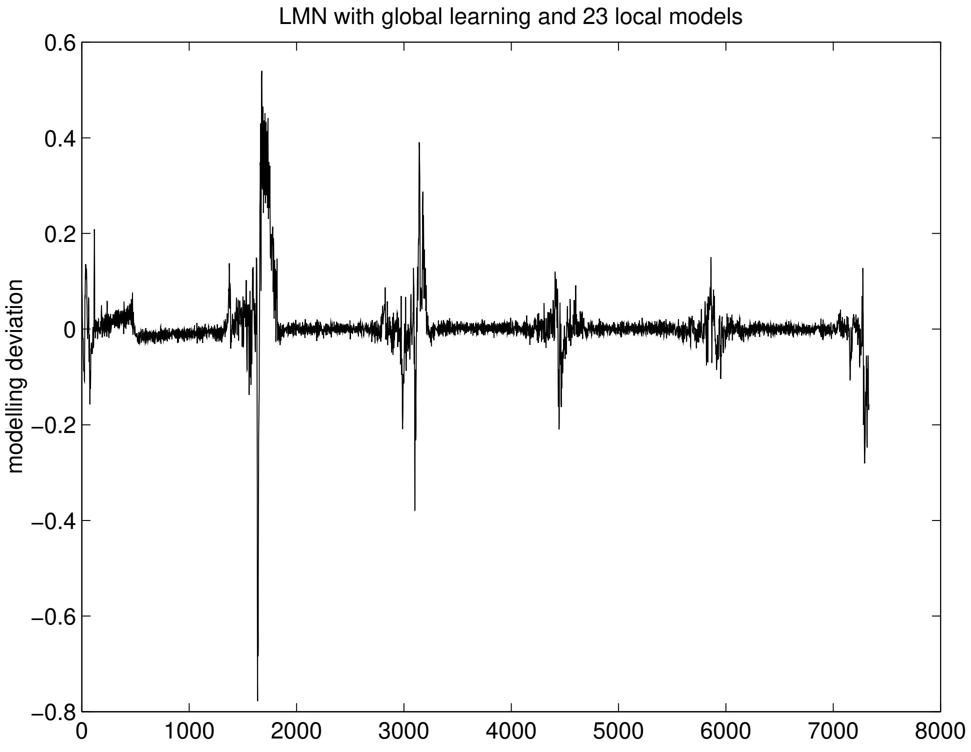


Figure 6.16: Mill model residuals for LMN (global training) on the validation data. Mean squared error = 0.0036.

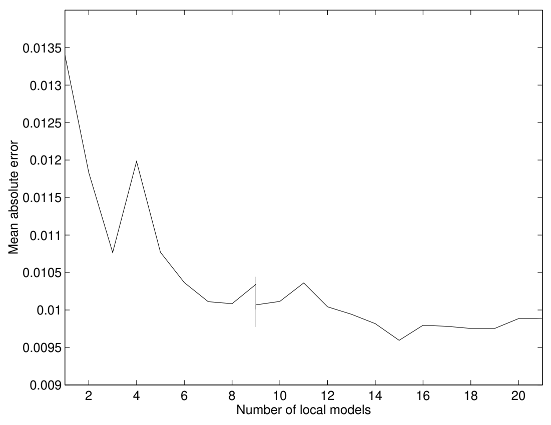
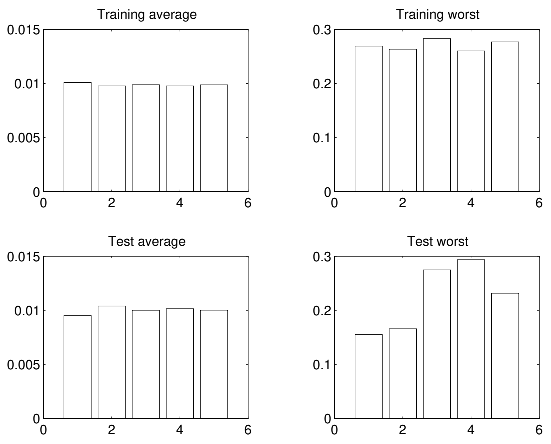


Figure 6.17: Global Learning Error Curve. Global optimisation performs well on training set, but poorly on validation set. Glitch in curve indicates pruning of models.

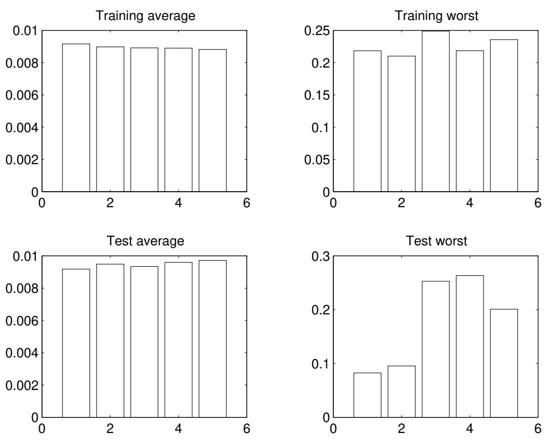
To compare the robustness of the local and global parameter estimation, independently of the structure identification, the structure which produced the residuals in Figure 6.14 was fixed, and a cross-validation experiment was carried out, where only the local model parameters were adapted, both locally and globally. The errors are, in general, slightly better in the global case. How does this fit in with the residual plots in Figures 6.14 and 6.16, where the local results are clearly better? The difference is that the validation results from the cross-validation done here use data points extracted from the rolling runs used for training, whereas the residuals are plotted on validation data from completely fresh runs, which provide a better test of the modelling framework.

Summary of results on validation data

Table 6.1 summarises the modelling results tested on the validation data. The Local Model nets consistently provided the highest level of accuracy, producing both the smallest average errors and smallest maximum errors. Surprisingly, the simple Linear model performed better than MARS and Multi-Layer Perceptrons. This is not to say that an MLP could not be trained to produce better results, but it does indicate the lack of robustness in the widely used learning algorithms, such as back-propagation, which have trouble identifying a linear system.



(a) Cross-validation results for a fixed structure Local Model Net with local learning of parameters.



(b) Cross-validation results for a fixed structure Local Model Net with global learning of parameters.

Figure 6.18: Cross-validation Comparison between local and global learning. In both cases the parameter estimation process seems to produce robust results. However, the residuals on completely new validation data (shown in Figures 6.14 and 6.16 show that local learning produced better generalisation ability).

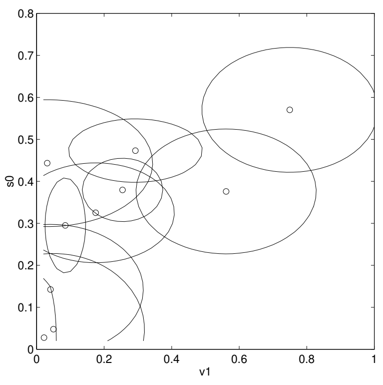
Model Type	Mean abs error	Mean squared error	Max abs error
Linear model	0.0116	0.00049	0.28
MARS (19 BF's)	0.0168	0.0014	0.45
MARS (85 BF's)	0.0162	0.0017	0.63
RBF (39 Basis Functions)	0.0185	0.0017	0.72
MLP (30 Units)	0.0144	0.00065	0.29
LMN (gl, axis-orth)	0.0235	0.0036	0.78
LMN (ll,RBF, n_u =2)	0.0151	0.00068	0.32
LMN (ll, no active)	0.0118	0.00059	0.32
LMN (ll,RBF, n_u =5)	0.0115	0.00045	0.26
LMN (ll, ellipse)	0.0116	0.00044	0.227
LMN (ll,RBF)	0.0111	0.00046	0.264
LMN (local learning, axis-orth)	0.0110	0.00040	0.22

Table 6.1: Summary of mill modelling results on validation run. gl – global learning, ll – local learning, ellipse – full ellipsoidal distance metric, axis-orth – axis-orthogonal ellipsoidal distance metric.

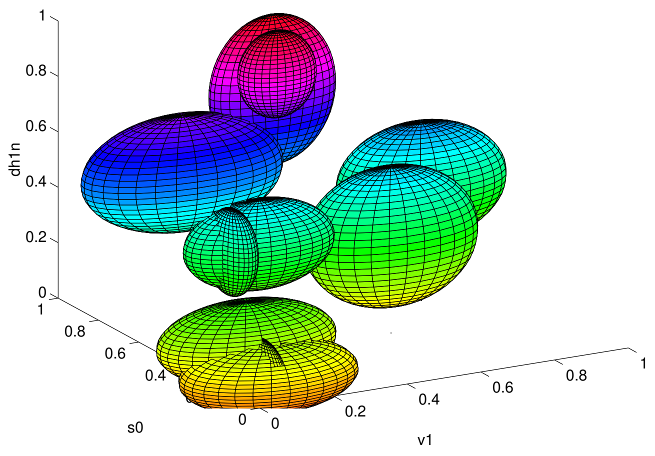
6.3.4 Interpreting the trained models

Can the final machine learned model be interpreted to give the human engineer a better understanding of the system in question? This is a question which has often been ignored in the literature on learning systems, but one which is very important in industrial situations. There will usually be a trade-off between flexibility and interpretability, which will depend on their relative importance for a given application.

The positions of the basis functions can be visualised in a three dimensional plot, as shown in Figure 6.19. This gives us some insight into the location of complexity in the problem's state space. The hyper-ellipsoids in the figure correspond to the scales of the distance metrics of the basis functions. The constructive algorithm seemed to develop models with the complexity in the 'intuitively correct' areas of the input space. The acceleration and deceleration phases are deemed the most complex, and these were covered to a greater extent than the constant velocity area, which in general was covered by only one or two models. The resulting models were also reasonably small, the overfitting protection limiting the detrimental effects of sparse data in the complex areas of the input space – the model structures found tended to have less than 20 local models, despite training sets of over 6000 examples.



(a) BF Contour plot



(b) 3-D plot of Basis functions for roll mill model. The basis functions are shown here in the unnormalised form, basically as ellipsoids representing the volume equivalent to that of the contour plot at 0.5.

Figure 6.19: Visualisation of the mill model operating regimes.

A further aid to understanding the model is to go through the validation runs examining the model output and actual output, while viewing the current position $\tilde{\phi}$ in the operating space.

The top plot in Figure 6.20 shows the model and actual output, while the plot below shows the run for the entire training set in the operating space, with the portion being examined above is marked as the darker area.

The use of such visualisation tools lets the developer examine the areas of the operating space responsible for large model deviations, to try and determine possible inadequacies in the model structure (e.g. too few local models in a particular region, ill-suited local model structures, etc.) with the aid of visualisation tools such as shown in Figure 6.19. These tools are obviously limited to any three given dimensions, but can still provide useful insight in many cases. Further development of such tools would allow the user to select certain areas of the inputs space and find the nearest local models, so that their structures and parameters can be investigated, or gain more insight about the model workings. In general, as noted in (Johansen and Foss, 1994a), there should always be room for the human user to intervene in the modelling process. To give a more detailed feeling for the accuracy of the model, three

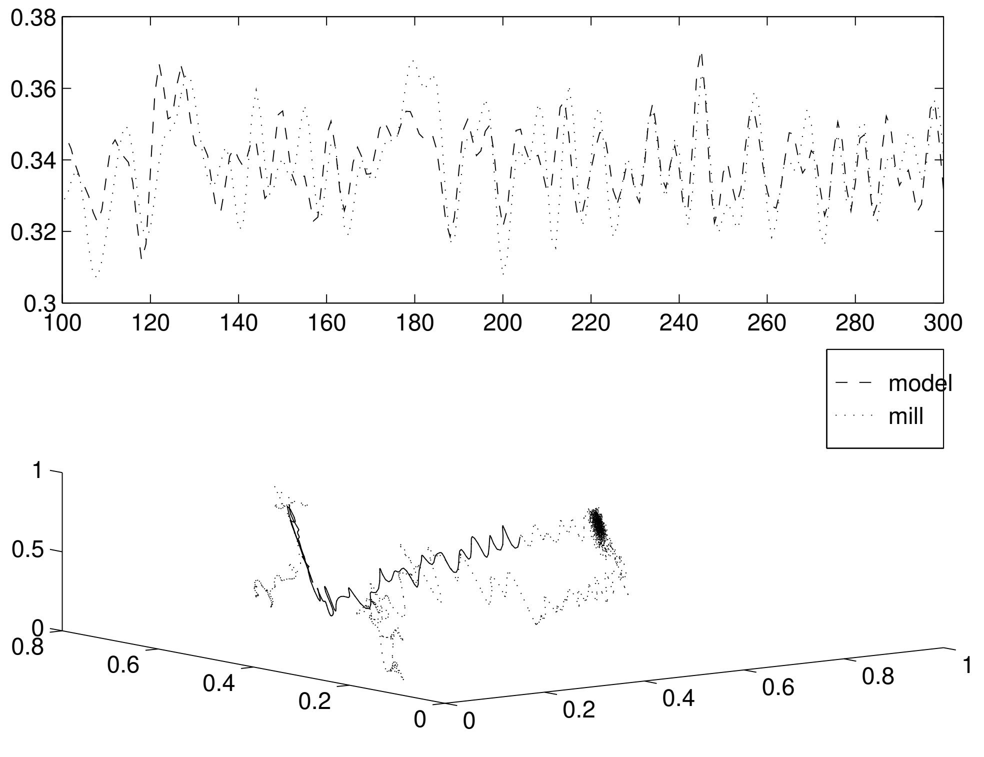


Figure 6.20: Model and Real Mill output, with the related area of the input space. The visualisation tool allows you to examine the model residuals, while viewing the current position in the operating space below (the subset of the data corresponding to the model plot above is highlighted). More powerful tools would allow the use such plots in conjunction with images such as Figure 6.19, where the local model closest to the data shown could be 'clicked' on and the structure and parameters viewed.

areas of the test data set are plotted in Figures 6.21 to 6.23.

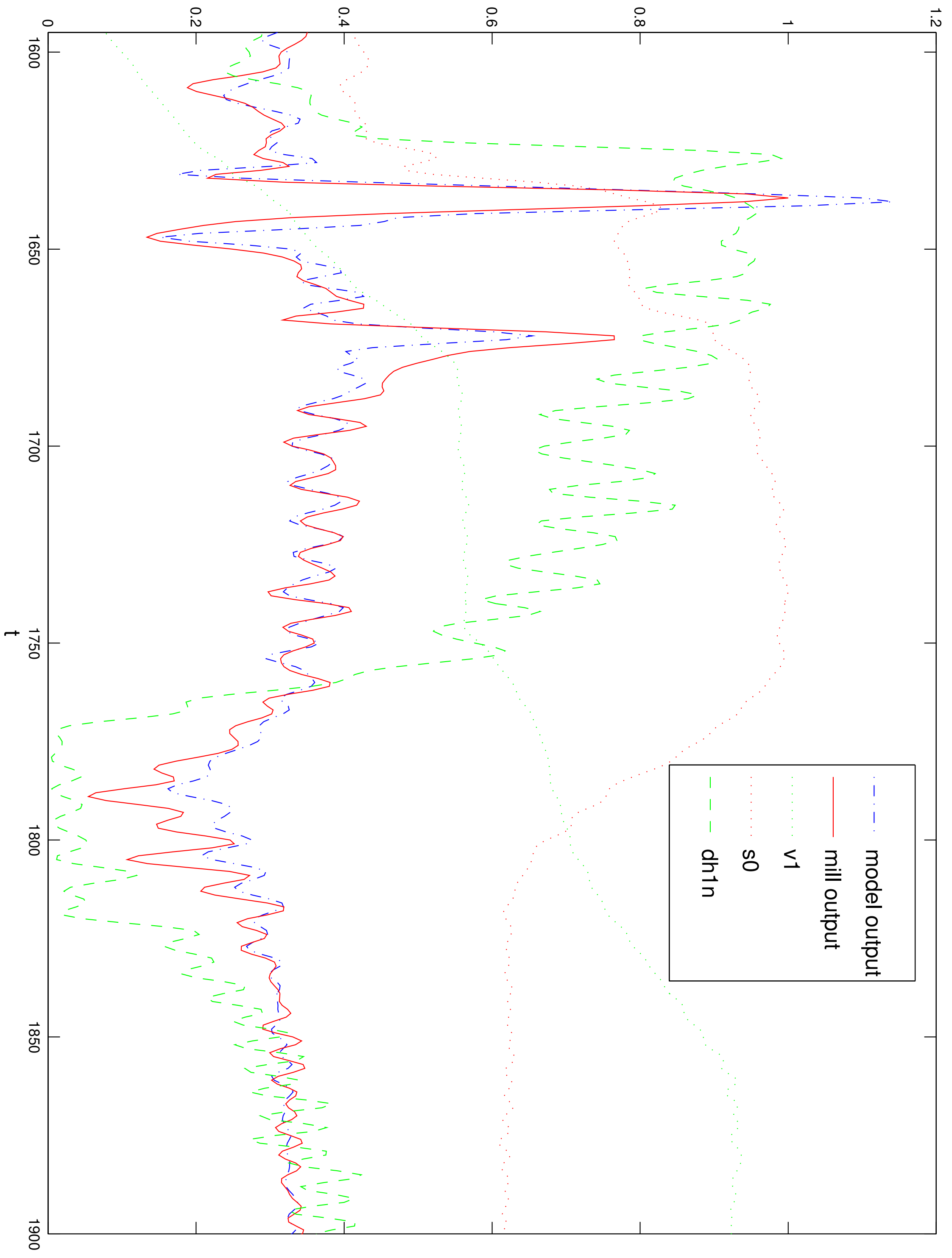


Figure 6.21: Detailed validation modelling runs with LMN and local learning. LMN model in acceleration phase. Despite the changes in the inputs, the model performs well, although it goes off track between 1760 and 1820.

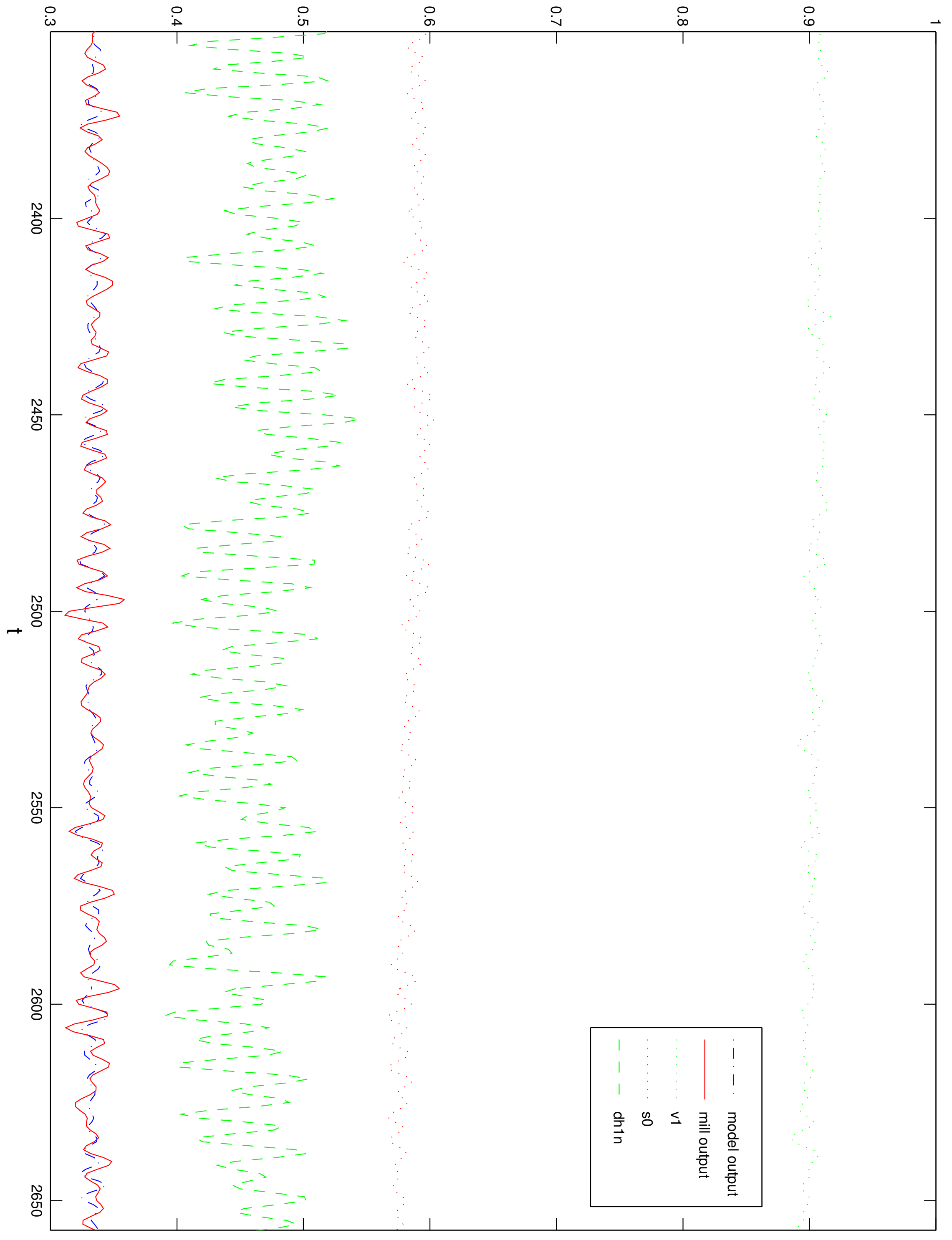


Figure 6.22: Detailed validation runs with LMN and local learning, in constant velocity phase.

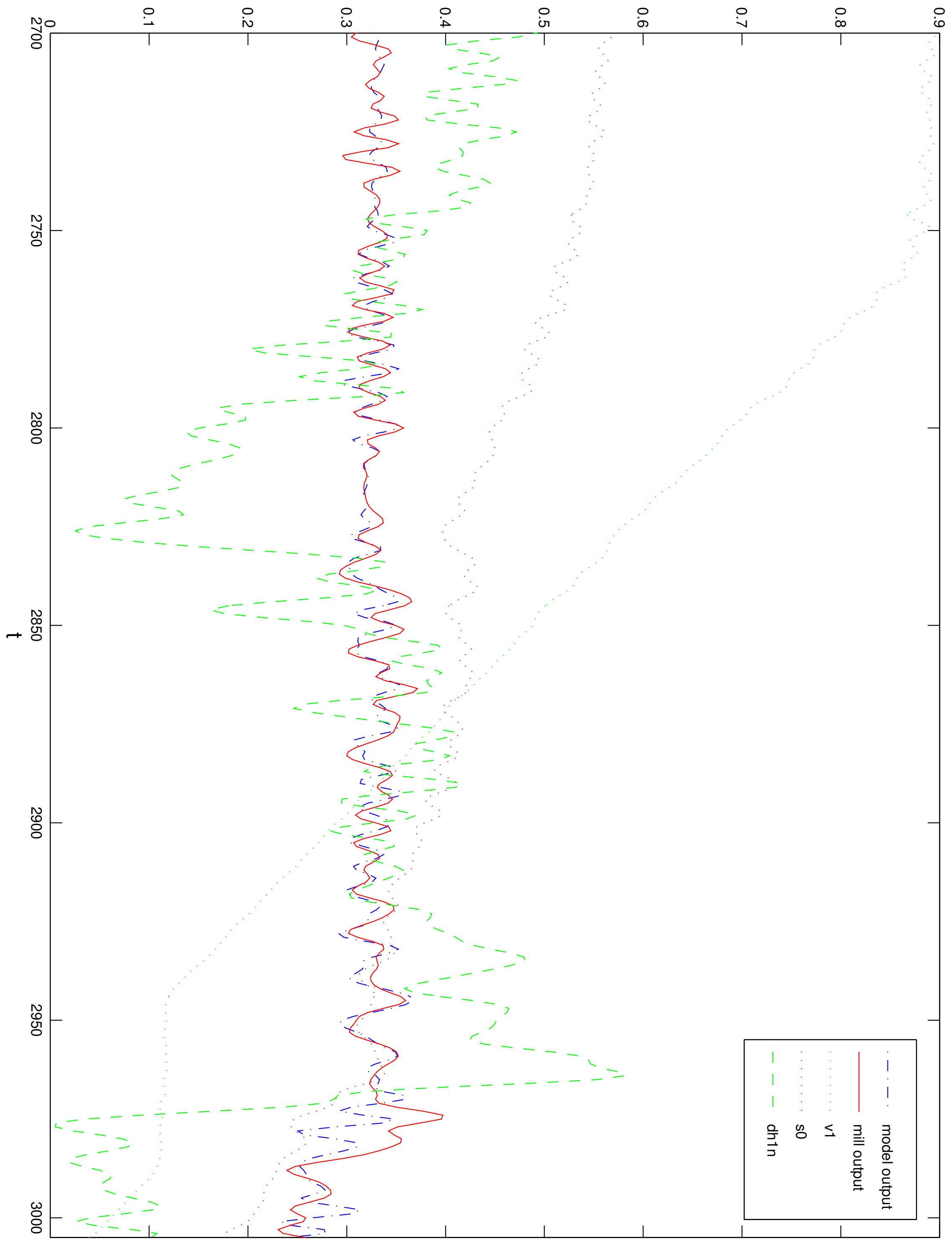


Figure 6.23: Detailed validation modelling runs with LMN and local learning. LMN model in deceleration phase. Again model does well despite the changing state of the physical system.

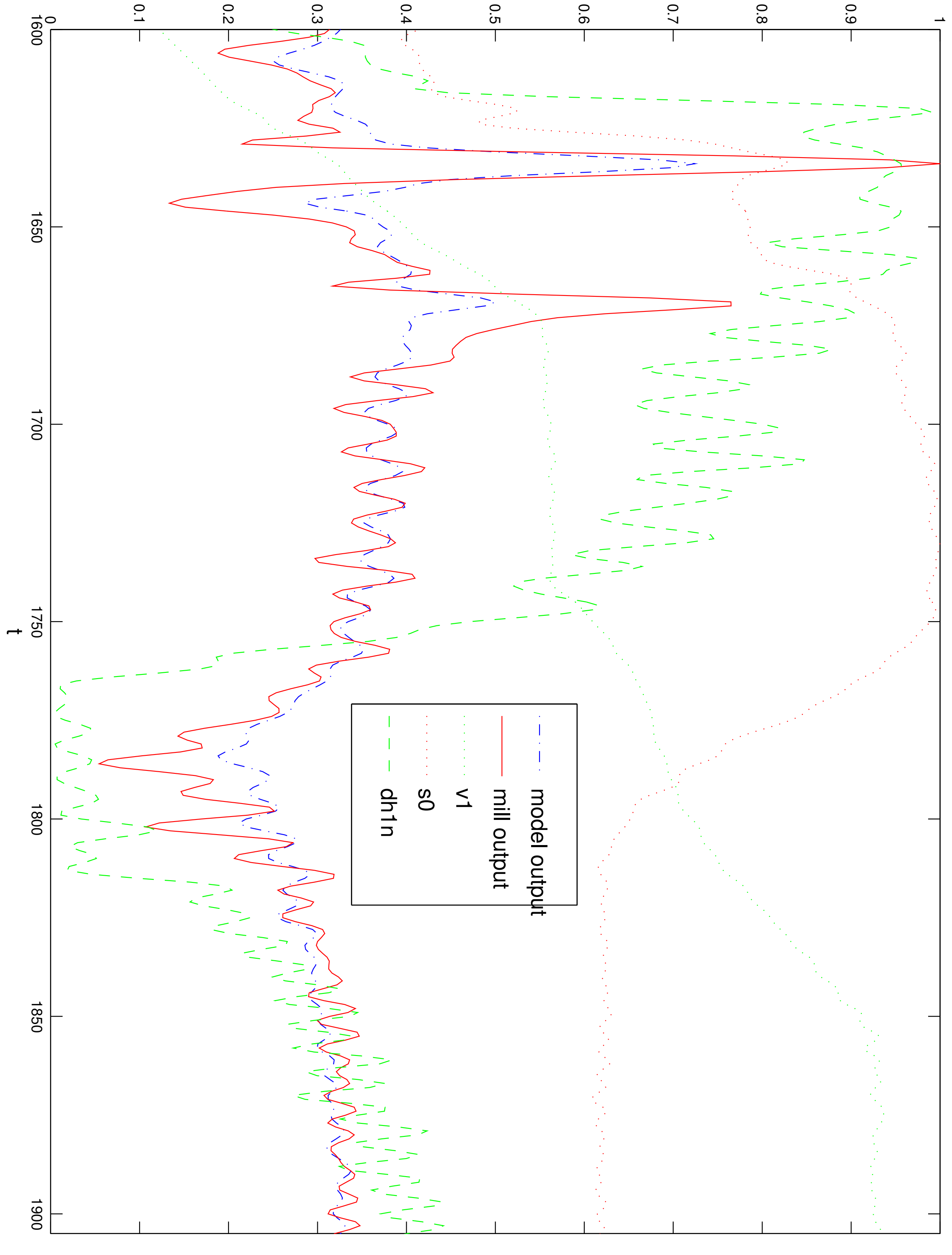
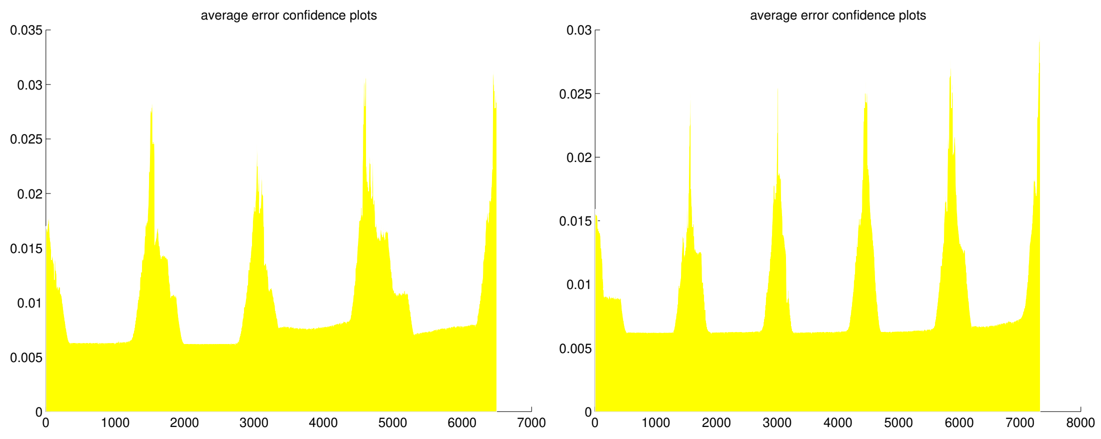


Figure 6.24: Detailed validation modelling runs with linear model. Compare with the LMN Model in Figure 6.21. The linear model does not manage to come as close to the actual output in the area of large output changes between 1600 and 1700. Nevertheless, the linear model could still prove to be useful in a real application because of its simplicity.

6.3.5 Analysing the local confidence limits

By examining the local error statistics (as defined in Section 3.2.1) for the local models the location of the worst errors can be deduced, possibly helping the engineer learn where the complexity in the real process is, or where insufficient experimental data exists. The test run is plotted below, where the error is shown, with the network's estimation of its own accuracy.



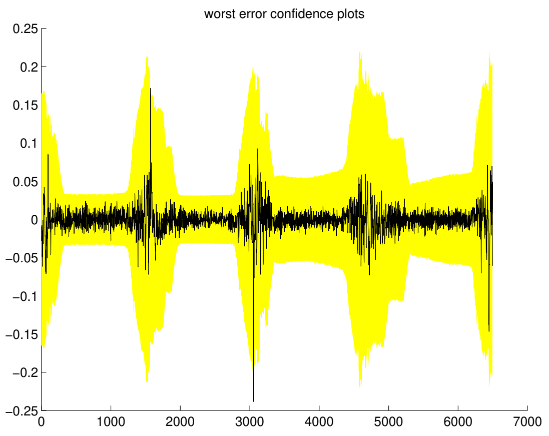
(a) Error statistics and LMN's weighted average error prediction on the training set used to derive the estimates.

(b) Error statistics and LMN's weighted average error prediction on the validation set.

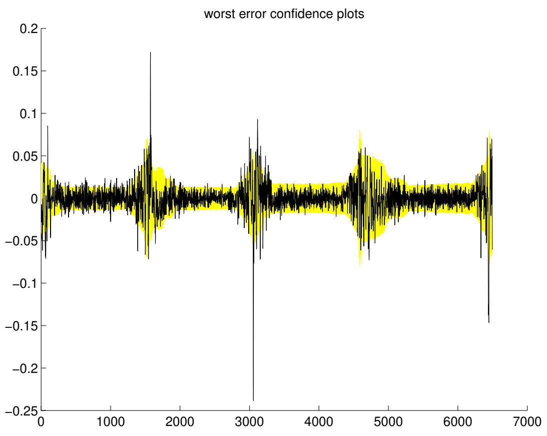
Figure 6.25: State-dependent average error statistics on the training and validation data

What happens when we close the loop?

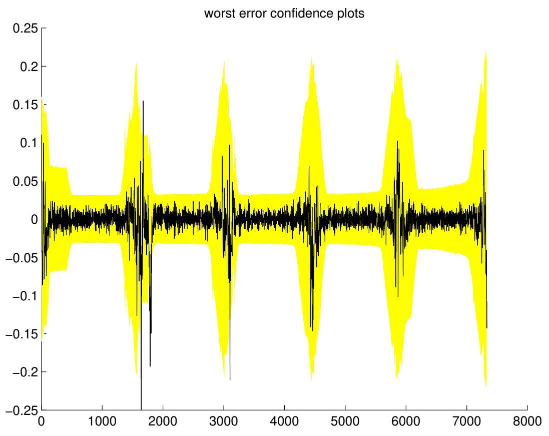
Despite the validation process, the real test of the model will only happen when it is used as it was intended in the real system. In this case, any false assumptions about the system, or disturbances on the training data will become immediately apparent. Once a controller is built which uses the model from the learning phase, it may also make the old model invalid. This is because the new, improved controller will impose a different type of stimulation on the process, which will cover different areas of the input space, and which may produce quite different responses from the process.



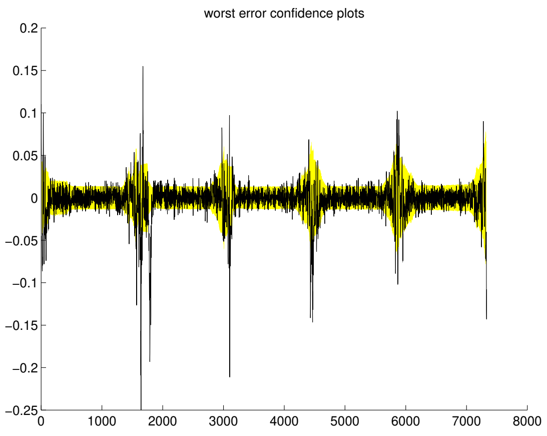
(a) Model residuals and LMN's local worst error prediction on the training set used to derive the estimates. The worst error from the local model's receptive field is taken (where basis function > 0.1), so the algorithm tends to overestimate the worst error.



(b) Model residuals and LMN's weighted worst error prediction on the training set used to derive the estimates. Because the worst error is weighted the algorithm tends to underestimate the maximum error



(c) Model residuals and LMN's local worst error prediction on the validation set.



(d) Model residuals and LMN's weighted worst error prediction on the validation set.

Figure 6.26: State-dependent worst error statistics on the training and validation data.

6.4 Robot Actuator Modelling

The robot application described in this section is based on data sampled from a physical system, supplied by Tom Kavli, SINTEF, Oslo. The application and the data sets have been described for modelling work in (Kavli, 1992) and in (Johansen and Foss, 1994b). A brief overview is given here:

Many industrial robot applications now demand high dynamic accuracy. Model based control schemes have the potential to improve performance, but the use of hydraulic manipulators has suffered due to the lack of good nonlinear models for the hydraulic components. Model based control schemes have been more successful on electric direct drive arms with low friction and linear actuators. The goal of the learning task is to form a model of the servo valve/actuator system of a hydraulic robot. The robot is an ABB Trallfa TR4000 Robot, specially designed for spray painting, where tracking accuracy over a desired trajectory is extremely important. The control signal u is described as a function of the joint position q, velocity $\dot{q}$ and acceleration $\ddot{q}$:

$$u = f\left(q, \dot{q}, \ddot{q}\right). \tag{6.7}$$

The nonlinearities are due to:

- the changing moment arm of the cylinder over the operating range,
- the nonlinear damping coefficient due to the quadratic flow/pressure relation for turbulent flow and,
- the changing pressure gain characteristics for the servo valve at different flow rates.

The data was sampled by logging the data at 100Hz, while the manipulator moved along a randomly generated path. The velocity and acceleration signals were calculated by low pass filtering the data and differentiating the joint positions. The linear effects in the system were subtracted from the data to emphasise the nonlinearity of the system. The training data consisted of 8000 training points and the test set had 1000 points.

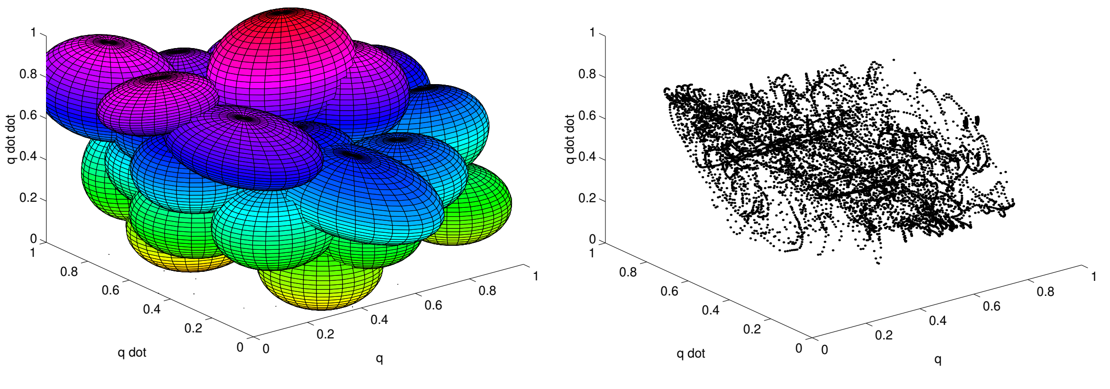
6.4.1 Experimental results

The ASMOD results * are taken from (Kavli, 1992). LSA (Local Search Algorithm) results + are from (Johansen and Foss, 1994b). The LMN results were obtained by using the algorithms described in this thesis, as well as the active selection algorithm to reduce the number of points in the training set (the training set was reduced to a maximum of 1200 points).

The MARS results are shown in Figure 6.30(a). The cross-validation results for the local model net with local learning are shown in Figure 6.29(a) is compared with the global learning case in Figure 6.29(b). Note the poor performance of local learning in the cross-validation results.

Model Type	Test Error (NRMS)
LSA+	17
MLP (3-20-1)	23
ASMOD*(Quadrat.)	15
ASMOD*(Linear)	17
RBF*	23
MARS	17
LMN (local learning)	18
LMN (global learning)	17
LHM (axis oblique)	18
LMN (local learning)	18
LMN (global learning)	17

Table 6.2: Robot modelling results on the test set. The only method to provide better results than the LMN was the ASMOD model with quadratic splines (and a large number of parameters at 512 basis functions). Local Modelling proved to lead to slightly worse results, and required a larger number of parameters, but produced 'safer' extrapolation to areas with little training data.



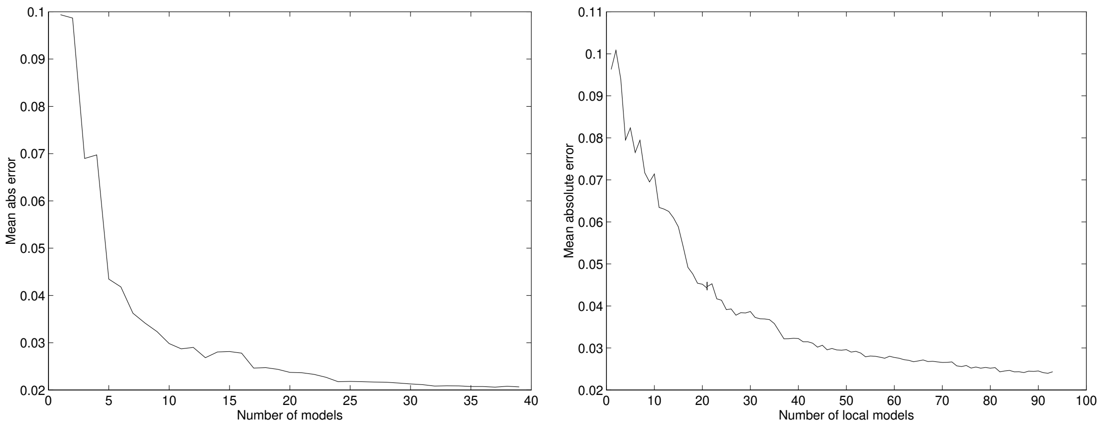
(a) LMN Basis Functions for the robot model

(b) Robot training data

Figure 6.27: Distribution of robot training data and local models' basis functions.

Only one of the runs produces a mean error similar to that of the global optimisation. This is, however, due to a weakness in the structure identification algorithm, which stops growing when the error stops decreasing. In global learning, the greater degree of freedom for the optimisation means that the additional local models have a more pronounced effect, and construction continues. By altering n_{res} to 5 for local learning produced the results shown in Figure 6.28(b), where a larger number of local models were built, but which achieved an improvement in accuracy to that close to the globally trained model.

The problems with ill-conditioning for global learning are less prevalent in this application, due to the low dimensionality, accompanied by large amounts of training data, and the apparent smoothness of the underlying process nonlinearity.



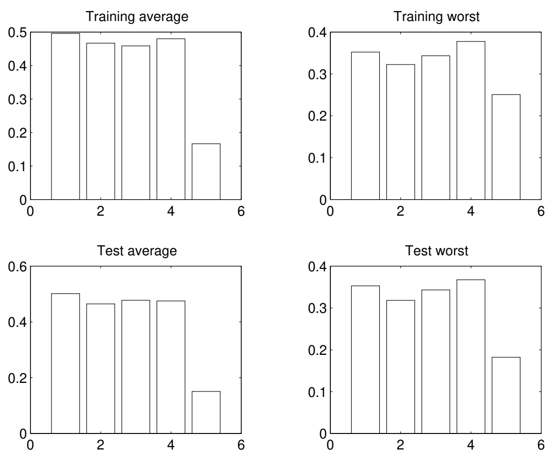
(a) LMN error curve with global learning

(b) LMN error curve with local learning

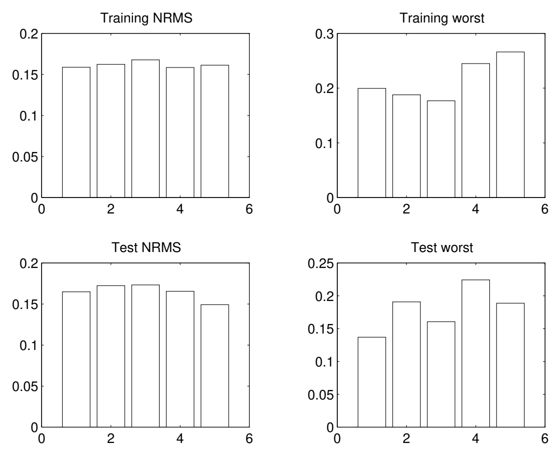
Figure 6.28: The progress of the average absolute error as new models are added to the robot modelling example. Note that local learning finishes earlier than global learning.

Visualisation of the system

The following surfaces are representations of the input space as seen through different slices through the space. The corners of the plots are areas where the system had no training data, and are therefore unreliable.

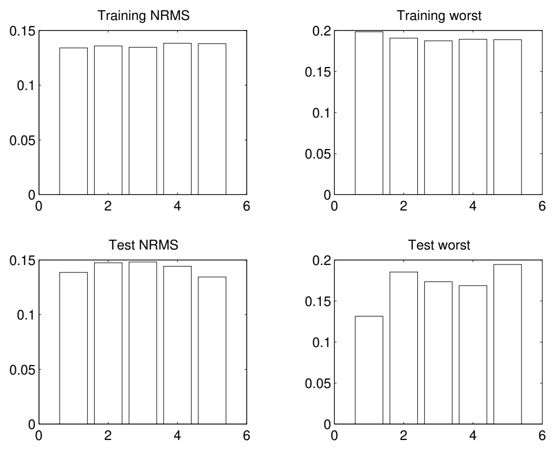


(a) LMN results with local learning



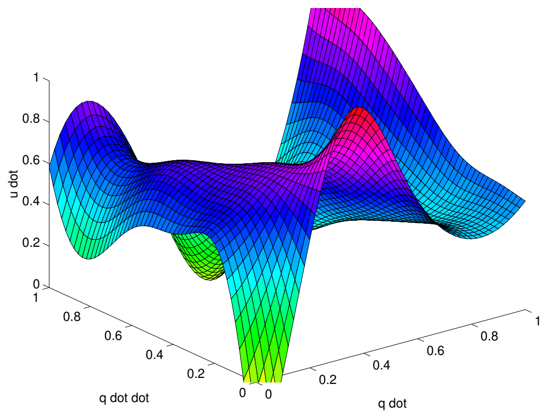
(b) LMN results with global learning

Figure 6.29: Cross-validation results for the Trallfa robot with Local Model Nets. Note the poor performance of local learning in all but one example. By increasing n_{res} to 5, the local learning could perform better, as shown in Figure 6.28(b)

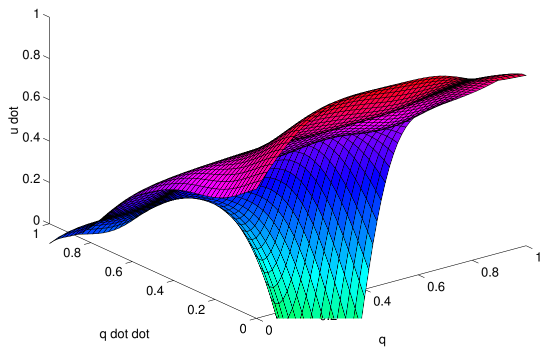


(a) MARS results with maximum number of basis functions set to 200

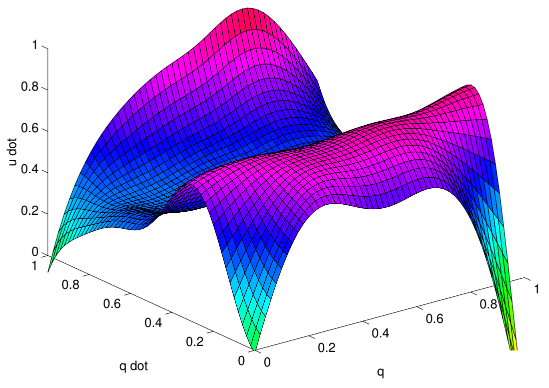
Figure 6.30: Modelling results for the Trallfa robot with MARS.



(a) Speed and acceleration, with position at 0.5.

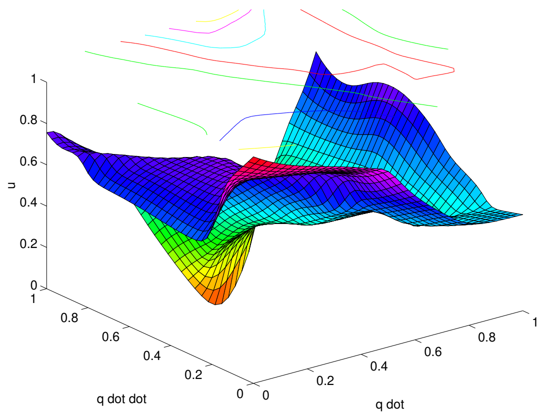


(b) Position and acceleration with speed at 0.5.

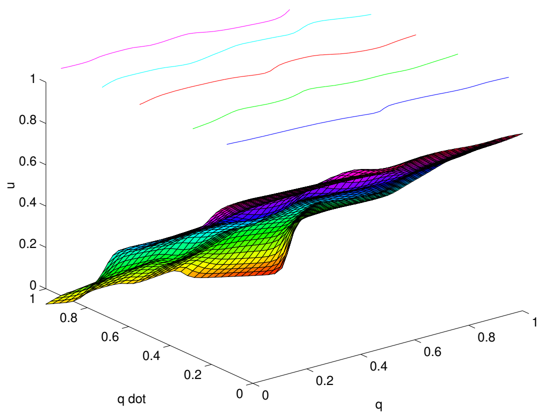


(c) Position and speed with acceleration at 0.5.

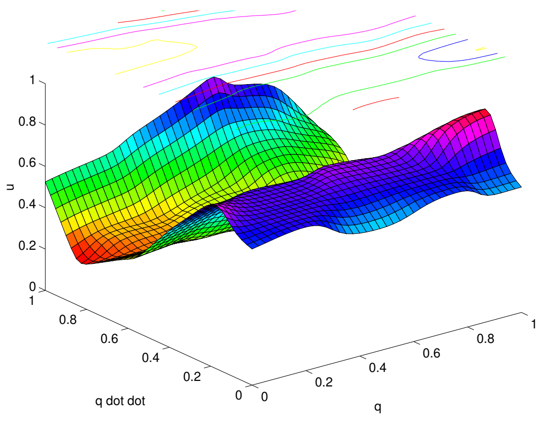
Figure 6.31: Output responses for slices through the robot actuator model. Note the way the model response surface flies off in the corners where data does not exist. This is one disadvantage of normalised basis functions, where the basis functions are supported through the whole input space, combined with global learning, which produces less robust models. Compare the plots here to those achieved with local learning in Figure 6.32.



(a) Speed and acceleration, with position at 0.5.



(b) Position and acceleration with speed at 0.5.



(c) Position and speed with acceleration at 0.5.

Figure 6.32: Output responses for slices through the robot actuator model using local learning. Compare the plots here to those achieved with local learning in Figure 6.31. The locally trained models provide a less dramatic form of extrapolation outside the populated area of the input space.

6.5 Conclusions

6.5.1 Rolling mill results

The rolling mill application provided an interesting environment for testing the local model network methodology and the constructive algorithms proposed in this thesis. To solve the problem meant dealing with the practical problems of data acquisition, pre-processing, and variable selection. In some ways though it is also not an ideal choice to show off the structure, due to the poor available understanding of the system, and the difficulty in performing experiments to acquire data, or to validate the usefulness of models produced. Despite the relative deficit of formalised *a priori* knowledge about the system, the local model network methods proved to be the most accurate, and most interpretable representations of the rolling mill process. The MARS algorithm was relatively fast, but did not have the desired accuracy, and the conventional MLP neural network was very slow during training, and produced low-accuracy models.

The validation phase used cross-validation techniques to test the robustness of the learning algorithms, as well as using validation sets from completely different runs. This showed up some interesting features in the experimental setup. The local learning techniques led to significantly more accurate models for new validation data than globally trained ones, even though they were found to be slightly less accurate on 5-fold cross-validation runs based on partitions of the training set. This seems to indicate that the local learning produced a more robust model of the mill, at the expense of accuracy on the training data.

The state-dependent error estimates proved, despite their simplicity, that they could capture the areas of the input space where the model is least accurate – the acceleration and deceleration phases. Other tools were used which helped interpret the model, including the visualisation of model residuals, accompanied by the position of the data in the operating space of the local model net's basis functions. This can also be combined with slices through the basis functions, in the form of either contour plots or ellipsoidal plots of basis functions in three dimensions.

6.5.2 Robot actuator results

The results obtained by the Local Model Net are comparable with the results quoted in the original work (Kavli, 1992) and also with those in (Johansen and Foss, 1994b). The problem is low dimensional, the nonlinearity seems to be fairly smooth, and there is an abundance of training data. This means that other model structures can also cope with the problem, and that while the local model nets perform well at the modelling task, the potential advantages of local model nets are not as evident as in other applications. A further interesting result from the experiment pointed out weaknesses in the structure identification algorithm working with local learning, where it often stops construction too early. Using a larger n_{res} lead to better results, but also needed a larger number of parameters.

Chapter 7

Conclusions

7.1 Local Model nets

The work undertaken has shown that Local Model Nets have a great deal of potential as a general model structure, suitable for a wide variety of empirical modelling and learning tasks. An important aspect of the architecture is that it forms a link between the world of learning systems such as neural networks and the more conventional world of systems theory and statistical modelling. The local nature of the models means that the trained networks are more transparent, and can easily integrate *a priori* models from the other paradigms. Knowledge about the process being modelled can be used in hybrid local model nets to better cope with high-dimensional systems.

7.1.1 Local Model Net extensions

This thesis extends the understanding of local model networks in several respects:

- The Local Learning methods developed in this work have a regularisation effect which in many cases improve the generalisation of the trained net, as the methods overcome some of the conditioning problems discovered in the parameter estimation phase of training. The nature of the conditioning problems had previously gone unnoticed, and this work demonstrated the dependence of the condition on the level of overlap between models.
- Methods have been developed which use the local nature of the basis functions to allow local error estimates to be interpolated to produce a state-dependent error statistic for the whole network. Such statistics are useful for the validation and interpretation phases of modelling and are useful for on-line application in model-based control and diagnosis systems. The techniques were applied to the rolling mill problem and produced error estimates which corresponded well with the measured residuals on a validation run.

- The effect of the widely used normalisation technique for basis function nets has been analysed, and hitherto undocumented results were discovered. These have a serious effect on the smoothness of the representation and can compromise the local nature of the basis functions. Normalisation still has advantages in that it produces a partition of unity, and makes the model less sensitive to a poor choice of basis functions, but the unexpected side-effects described in Section 3.3.1 should be taken into account when interpreting trained basis function nets, and when developing new learning algorithms.

The thesis also provides descriptions of a number of practical techniques for use in Local Model Networks:

- New Local Learning methods were developed for the parameter estimation phase in Local model nets. These methods are far less computationally expensive than the global ones, and can often result in more interpretable local models. In general, with a reasonable level of basis function overlap, the local optimisation tends to cope better than global learning in high-dimensional, noisy or sparsely populated learning problems. Section 4.4 gives examples of how the local learning methods performed more robustly on noisy test functions, and Chapter 3 demonstrates the better generalisation ability of locally trained local models on the rolling mill application.

  Local learning is also well-suited to heterogeneous local model nets, where a variety of types of local model can be used, each with its own optimisation algorithm. This makes the local model framework much more general than the basic homogeneous linear local model structure, allowing it to integrate a variety of styles of model and methods of knowledge representation.

- The new Multi-Resolution Constructive (MRC) structure identification algorithm for Local Model networks has been developed. This allows the network to fit the basis functions to the data set in a gradual, problem-adaptive manner. The constructive nature of the algorithm speeds up the modelling process by reducing the amount of 'fiddling' needed to produce a good model. It also tends to find more accurate models, as the complexity is increased as needed for the given problem, while overfitting is limited by taking the local density of the training data into account. The fact that trained models are now being trained in minutes as opposed to days with previous learning techniques obviously better supports the interactive nature of the modelling process.

- Active data selection techniques have been developed for local model nets. These speed up the learning process by automatically selecting a training set for the current model structure, where the number of data points needed is reduced, but the most informative data is chosen from the complex areas of the input space. Training tends also to be more robust with respect to the distribution of the training data, as demonstrated in Section 4.3 with synthetic examples and in Chapter 6 on the real applications.

7.1.2 Learning Hierarchies of Models

The novel Learning Hierarchy of Models architecture is a hierarchical extension of the local model networks, which due to the split-like nature of its partitioning mechanism decomposes high dimensional spaces more efficiently than flat Local Model nets.

A new constructive algorithm which automatically grows tree-like structures to fit the target function is described. The soft-splits can be axis-oblique, making the structure more powerful than other more restricted hierarchical structures which are limited to partitioning the input space one variable at a time. Local learning, error estimation and active learning can all be applied in a hierarchical manner.

Use of soft, axis-oblique splits provides the potential for hierarchical structure adaptation, where gradual structure learning occurs simultaneously at several levels of the structure at differing timescales. This is likely to lead to more efficient learning algorithms, which produce more parsimonious models. It is also interesting for on-line use for structure adaptation in time-varying systems. The architecture can also be seen as a fuzzy decision tree, and could apply methods from both decision tree theory and fuzzy systems.

7.1.3 Experimental work

To demonstrate the applicability of the methods to real industrial processes data was taken from an aluminium rolling mill and a robot actuator:

- Data has been taken from an aluminium rolling mill to train a predictive model of the strip thickness using local model nets. This was a real application of the modelling techniques to aspects of a nonlinear, dynamic process which were poorly understood. The Local Model Net techniques described in this thesis have produced the most accurate modelling results known to date on this problem, and further testing with an on-line implementation is planned.
- The Trallfa robot actuator nonlinearities were also learned successfully. This problem has been used as a test example for a number of other structure identification algorithms, and it is characterised by its low dimensionality, smooth nonlinearities and large training set. Despite its relatively simple nature, the problem demonstrated that the methods could still be competitive on more straightforward, low-dimensional problems.

7.2 Outlook

The training algorithm used to identify the underlying structure of the basis functions is a practical but still relatively *ad hoc* routine, and there is certainly a great deal of progress to be made in the area of structure construction algorithms. User friendly methods for the easy integration of *a priori* knowledge into the network will become more and more important as the demands for accuracy, robustness and transparency increase. There are many opportunities for the development of easy-to-use tools which allow the developer to creatively build engineering knowledge directly into the modelling process, leaving the learning algorithms to cope with the uncertainty in the process, and to warn the user where more information is needed. Tools based on the ideas described in this thesis are likely to benefit the developer in the computer intensive, data driven areas of modelling. Despite the improvements in tools, it is perhaps relevant to quote one of the leading system identification researchers, Lennart Ljung:

> 'Thinking, intuition and insight cannot be made obsolete by automated model construction' (Ljung, 1987)

An immediately practical view of the methods described in this thesis is to see them as computationally- and data-intensive ways of supporting more traditional modelling methods, allowing the engineer to better understand the system by reproducing the behaviour with a learned model, understanding the behaviour and then creating a 'hand built' simplified model which exhibits the essential behaviour of the structurally more complex learned model. This is then more easily understood and validated, and is therefore more likely to be used in practical applications.

Appendix A

Notation

A.1 Notation used

$\mathbf{A}^T$	the transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	the inverse of matrix $\mathbf{A}$
$\mathbf{A}^+$	the pseudoinverse of matrix $\mathbf{A}$
B_j, p	Basis Spline j of order p
$Cov(\mathbf{x})$	the covariance matrix of vector $\mathbf{x}$
t	knot of a basis spline
u	system inputs
$\boldsymbol{\psi}$	information vector inputs
$\mathbf{c}_i$	position of basis function i's centre in the input space
$\tilde{\boldsymbol{\phi}}$	operating point state
y	outputs
$\hat{y}$	estimate of outputs
$\mathcal{D}$	the training set of input-output pairs
$\mathcal{D}_i$	the training set of input-output pairs in receptive field of basis function i
ϕ	basis vector for regression
$\mathbf{\Phi}$	the design matrix for the regression problem
$\mathbf{\Phi}_i$	the design matrix for the local regression problem for local model i
J	cost function
J_V	cost due to model variance
J_B	cost due to model bias
J^*	optimal cost for parameter optimisation
J^{**}	optimal cost for structure optimisation
J_i	cost functional for optimisation of parameters of local model i.
$\mathcal{M}$	the whole model structure

$\mathcal{M}(\boldsymbol{\theta})$	the model structure parameterised by $\boldsymbol{\theta}$
$\mathcal{M}_i$	the model structure of local model i
n_ψ	number of input dimensions to local models
$n_\mathcal{M}$	number of basis functions
n_ϕ	dimension of basis vector for regression
n_{cutoff}	number of new units added for 'stopping window
n_{res}	number of resolution stages in multi-resolution clustering
$n_{\tilde{\phi}}$	dimension of basis function space
n_u	dynamic order of local models in rolling mill model
N	number of training patterns
N_i	number of training patterns for local model i
N_{des}	number of data points desired for a given local model for active selection
N_{min}	minimum number of training patterns in the receptive field of a local model
N_{rand}	number of data points randomly chosen from entire training set during active selection
P_l	the number of parameters in local model l
$\rho(\cdot)$	the basis activation function
$\rho_i(\cdot)$	the basis activation function for basis function i (can be after normalisation)
ρ_{total}	the cumulative activation of a basis function over the whole training set
$\mathcal{R}^n$	Euclidean n-dimensional space
$\mathcal{S}$	the process being modelled
σ	scaling matrix (or factor) for distance function of the basis function
σ_{win}	current window size for the complexity window of Chapter 4
σ_{max}	coarsest (initial) window size for the complexity window of Chapter 4
$\boldsymbol{\theta}$	weights
$\alpha(\cdot)$	error weighting function for weighted least squares problems
$\mathbf{a}$	weighting matrix for distance metric
γ	scaling factor for minimum distance between units
$\mathbf{D}(\boldsymbol{\psi})$	scaling factor for the distance from centre for calculation of σ
λ	reduction factor for window size in iterative cluster algorithm
Q_{min}	minimum angle between two neighbouring centres for covariance calculation
$*\lambda$	weighting of cost function in cost-complexity optimisation
$\mathcal{X}$	the experiment used to gather the training data $\mathcal{D}$
$\mathbf{Y}$	the vector of outputs in training set $\mathcal{D}$
ζ	threshold for basis function activation
$\lvert \cdot \rvert$	Euclidean norm
$\lvert\lvert \cdot \rvert\rvert$	Matrix norm

A.2 Abbreviations

ANN	Artificial Neural Network
ANOVA	ANalysis Of VAriance
ARMAX	Auto-Regressive Moving Average model with eXternal variable
ARX	Auto-Regressive model with eXternal variable
ART	Adaptive Resonance Theory
ASMOD	Adaptive Splines MODelling
BF	Basis Function
BP	Back-Propagation
CART	Classification And Regression Trees
CMAC	Cerebellar Model Articulation Controller
EBF	Ellipsoidal Basis Function
FIR	Finite Impulse Response
GCV	Generalised Cross-Validation
GSS	Generalised Smoothing Splines
HME	Hierarchical Mixtures of Experts
HSOL	Hierarchical Self-Organising Learning
ID3	Iterative Dichotomiser 3
k-N-N	k-Nearest-Neighbour
LHM	Learning Hierarchy of Models
LMN	Local Model Network
LMS	Least Mean Squared
LSA	Local Search Algorithm
LTU	Linear Threshold Unit
LVQ	Learning Vector Quantisation
MARS	Multiple Adaptive Regression Splines
MLP	Multi-Layer Perceptron
NARMAX	Non-linear Auto-Regressive Moving Average model with eXternal variable
PCA	Principle Component Analysis
PDP	Parallel Distributed Processing
PDF	Probability Distribution Function
PID	Proportional Integral Derivative
PLS	Partial Least Squares
PNN	Probabilistic Neural Network
PVM	Predictive Value Maximisation
RBF	Radial Basis Function
RCE	Restricted Coulomb Energy
RK	Reproducing Kernels
SOM	Self-Organising Map

SVD	Singular Value Decomposition
TDNN	Time-delay neural network
VI-Net	Validity Index Net

References

Åström, K. J. (1987). Adaptive feedback control. Proc. IEEE **75**, 185–217.

Aizermann, M., Braverman, E., and Ronzonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automatika i Telemekhanika **25**, 147–169.

Albus, J. S. (1972). *Theoretical and experimental aspects of a cerebellar model.* PhD thesis, University of Maryland.

Albus, J. S. (1975a). Data storage in the cerebellar model articulation controller (CMAC). Trans. ASME. Jnl. Dyn. Sys. Meas. and Control **63**, 228–233.

Albus, J. S. (1975b). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). Trans. ASME. Jnl. Dyn. Sys. Meas. and Control **63**, 220–227.

Alpaydin, E. (1991). GAL: Networks that grow when they learn and shrink when they forget. Technical Report 91-032, Int. Comp. Sci. Inst., Berkeley.

Anderson, J. A. and Rosenfeld, E., editors (1988). *Neurocomputing: Foundations of Research.* MIT Press, Cambridge.

Anderson, J. A. and Rosenfeld, E., editors (1990). *Neurocomputing 2: Directions for Research.* MIT Press, Cambridge.

Andrews, H. C. (1983). *Introduction to Mathematical Techniques in Pattern Recognition.* Robert E. Krieger.

Atkeson, C. G. (1990). Memory-based approaches to approximating continuous functions. In *Workshop on Nonlinear Modelling and Forecasting, Santa Fe Institute.* Addison-Wesley.

Back, A. D. and Tsoi, A. C. (1991). FIR and IIR synapses, a new neural network architecture for time series modeling. Neural Computation **3**, 375–385.

Bakshi, B. R. and Stephanopoulos, G. (1993). Wave-Net: a Multiresolution, Hierarchical Neural Neural Network with Localized Learning. AIChE Journal **39**.

Banan, M. R. and Hjelmstad, K. D. (1992). Self-organization of architecture by simulated hierarchical adaptive random partitioning. In *IJCNN, Baltimore*, volume III, pages 823–828.

Barnes, C. et al. (1991). Applications of neural networks to process control and modelling. In *Artificial Neural Networks, Proceedings of 1991 Internat. Conf. Artif. Neur. Nets*, volume 1, pages p321–326.

Barron, A. R. and Barron, R. L. (1988). Statistical learning networks: a unifying view. In *Computer Science and Statistics: Proc. of 21st Interface*, pages 192–203.

Bellman, R. E. (1961). *Adaptive Control Processes.* Princeton University Press, Princeton, NJ.

Benaim, M. (1994). On Functional Approximation with Normalised Gaussian Units. Neural Computation **6**, 319–333.

Bently, J. L. (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM **18**, 509–517.

Billings, S. A. (1980). Identification of nonlinear systems–a survey. IEE Proc.,Pt D **127**, 272–285.

Billings, S. A. and Chen, S. (1989). Extended model set, global data and threshold model identification of severly non-linear systems. Int. J. Control **50**, 1897–1923.

Billings, S. A. and Voon, W. S. G. (1987). Piecewise linear identification of non-linear systems. Int. J. Control **46**, 215–235.

Bishop, C. (1991). Improving the generalization properties of Radial Basis Function neural networks. Neural Computation **3**, 579–588.

Bishop, C. M. (1994). Training with noise is equivalent to Tikhonov Regularization. Submitted for publication.

Bottou, L. and Vapnik, V. (1992). Local learning algorithms. Neural Computation **4**, 888–900.

Breiman, L. et al. (1984). *Classification and Regression Trees.* Wadsworths & Brooks, Monterey, Ca.

Bridgett, N. et al. (1994). Associative memory network construction algorithms.

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

Brown, M. and Harris, C. (1994). *Neurofuzzy Adaptive Modelling and Control.* Prentice Hall, Hemel-Hempstead, UK.

Carlin, M. (1992). Radial Basis Function networks and nonlinear data modeling. In *Neuro–Nimes 92*, pages 623–633.

Chen, S. and Billings, S. A. (1989). Representation of non-linear systems: The NARMAX model. Int. J. Control **49**, 1013–1032.

Chen, S. and Billings, S. A. (1992). Neural networks for non-linear dynamic system modelling and identification. Int. J. Control **56**, 319–346.

Chen, S. and Billings, S. A. (1994). Neural networks for nonlinear dynamic system modelling and identification. In J., H. C., editor, *Advances in Intelligent Control*, pages 85–112, London. Taylor and Francis.

Chen, S., Billings, S. A., and Grant, P. M. (1990). Non-linear system identification using neural networks. Int. J. Control **51**, 1191–1214.

Chen, S., Cowan, C. F. N., and Grant, P. M. (1991). Orthogonal Least Squares learning algorithm for Radial Basis Function networks. Trans. IEEE on Neural Networks **2**.

Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988). Regression by local fitting. Journal of Econometrics **37**, 87–114.

Cohn, D. (1994). Neural network exploration using optimal experiment design. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, San Franciso, CA. Morgan Kaufmann Publishers.

Cohn, D., Atlas, L., and Ladner, R. (1990). Training connectionist networks with queries and selective sampling. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann Publishers, San Mateo, CA.

Cohn, D., Ghahramani, Z., and Jordan, M. I. (1994). Active learning with statistical models. Submitted to NIPS'94.

Craven, P. and Wahba, G. (1979). Smoothing noisy data with spline functions. Estimating the correct degree of smoothing by the method of generalized cross-validation. Numerical Math. **31**, 317–403.

Dasarathy, B. (1990). *Nearest Neighbour Pattern Classification Techniques.* IEEE Computer Society Press.

Deprettre, E. F., editor (1988). *SVD and Signal Processing: Algorithms, Analysis and Applications.* North Holland.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons.

Fahlmann, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann Publishers, San Mateo, CA.

Fedorov, V. V. (1972). *Theory of Optimal Experiments.* Academic Press, New York.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. Annals of Eugenics , 179–188.

Foss, B. A. and Johansen, T. A. (1993). On local and fuzzy modelling. In *3rd Int. Conf. on Industrial Fuzzy Control and Intelligent Systems*, Houston, Texas.

Friedman, J. H. (1991). Multivariate Adaptive Regression Splines. Annals of Statistics **19**, 1–141.

Fritzke, B. (1994). Supervised learning with growing cell structures. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 255–262. Morgan Kaufmann Publishers, San Franciso, CA.

Fu, K. S. (1970). Learning control systems – review and outlook. Trans. IEEE on Automatic Control **16**, 210–221.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the Bias/Variance Dilemma. Neural Computation **4**, 1–58.

Girosi, F., Jones, M., and Poggio, T. (1993). Priors, Stabilizers and Basis Functions: from regularization to radial, tensor and additive splines. MIT AI Memo 1430, MIT.

Golub, G. H. and van Loan, C. F. (1989). *Matrix Computations.* Johns Hopkins University Press.

Goodwin, G. C. and Payne, R. L. (1977). *Dynamic System Identification: Experiment Design and Data Analysis.* Academic Press, New York.

Gu, C. et al. (1990). The computation of GCV function through Householder tridiagonalization and application to the fitting of interaction spline models. SIAM J. Matrix Analysis , 457–480.

Haas, R. and Murray-Smith, R. (1993). Fuzzy/Neuro-Kombinationen – Konzepte und Perspektiven. Technischer Bericht, Daimler-Benz Research, Berlin. E-mail:haas@Dbresearch-berlin.de, murray@DBresearch-berlin.de.

Harris, C., Moore, C. G., and Brown, M. (1993). *Intelligent Control: Aspects of Fuzzy Logic and Neural Nets.* World Scientific.

Hartman, E. and Keeler, J. D. (1991). Predicting the future: Advantages of semilocal units. Neural Computation **3**, 566–578.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models.* Monographs on Statistics and Applied Probability 43. Chapman and Hall, London.

Haykin, S. (1991). *Adaptive Filter Theory, 2nd ed.* Prentice-Hall.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation.* Macmillan.

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, New York. Partially reprinted in (Anderson and Rosenfeld, 1988).

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City.

Hlaváckova, K. and Neruda, R. (1993). Radial Basis Function networks. Neural Network World **1**, 93–101.

Holden, S. B. (1994). *On the Theory of Generalization and Self-Structuring in Linearly Weighted Connectionist Networks.* PhD thesis, Cambridge University.

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

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, USA **79**, 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. Proceedings of the National Academy of Sciences, USA **81**, 3088–3092.

Hrycej, T. (1992). *Modular Learning in Neural Networks: A Modularized Approach to Neural Network Classification.* John Wiley and Sons.

Hunt, K. J. et al. (1992). Neural networks for control systems: a survey. Automatica **28**, 1083–1112.

Hutchinson, J. M. (1994). *Radial Basis Function Approach to Financial Time Series Analysis.* PhD thesis, Massachusetts Institute of Technology, Dept. EECS.

Isaksson, A. J., Ljung, L., and Strömberg, J.-E. (1991). On recursive construction of trees as models of dynamical systems. In *30th Conf. on Decision & Control, Brighton*, pages 1686–1687.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. IEEE Transactions on Systems, Man, and Cybernetics **1**, 364–378.

Jacobs, R. A. et al. (1991). Adaptive mixtures of local experts. Neural Computation **3**, 79–87.

Jang, J. S. R. and Sun, C. T. (1993). Functional Equivalence between Radial Basis Function networks and Fuzzy Inference Systems. Trans. IEEE on Neural Networks **4**, 156–158.

Johansen, T. A. and Foss, B. A. (1992a). A NARMAX model representation for adaptive control based on local models. Modelling, Identification and Control **13**, 25–39.

Johansen, T. A. and Foss, B. A. (1992b). Nonlinear local model representation for adaptive systems. In *Proceeding of the Singapore Int. Conf. on Intelligent Control and Instrumentation*, volume 2, pages 677–682.

Johansen, T. A. and Foss, B. A. (1992c). Representing and learning unmodeled dynamics with neural network memories. In *Proceedings of the American Control Conference, Chicago, Il.*, pages 3037–3043.

Johansen, T. A. and Foss, B. A. (1993). State-space modeling using operating regime decomposition and local models. In *Preprints 12th IFAC World Congress, Sydney, Australia, 19-23 July. Extended paper in Technical Report 93-40-W, Department of Engineering Cybernetics, Norwegain Institute of Technology, Trondheim.*

Johansen, T. A. and Foss, B. A. (1994a). A dynamic modeling framework based on local models and interpolation – combining empirical and mechanistiv knowledge and data. Submitted to Computers and Chemical Engineering.

Johansen, T. A. and Foss, B. A. (1994b). Identification of non-linear system structure and parameters using regime decomposition. To be presented at the IFAC Symposium on System Identification, Copenhagen.

Jones, R. D. et al. (1989). Function approximation and time series prediction with neural networks. Technical Report 90-21, Los Alamos National Lab., New Mexico.

Jordan, M. I. and Jacobs, R. A. (1991). Hierarchies of adaptive experts. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Publishers, San Mateo, CA.

Jordan, M. I. and Jacobs, R. A. (1993). Hierarchical Mixtures of Experts and the EM algorithm. Technical Report 9301, MIT, Computational Cognitive Science.

Kavli, T. (1992). *Learning Principles in Dynamic Control.* PhD thesis, University of Oslo.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science **220**, 671–680.

Kodratoff, Y. and Michalski, R. S., editors (1990). *Machine Learning: an artificial intelligence approach (Vol. 3).* Morgan Kaufmann, Los Altos.

Kohonen, T. (1990). The self-organizing map. Proceeding of the IEEE **78**, 1464–1480.

Kolodner, J. (1993). *Case Based Reasoning.* Morgan Kaufmann.

Kramer, M. A. (1993). Diagnosing dynamic faults using modular neural nets. IEEE Expert .

Kramer, M. A., Thompson, M. L., and Phagat, P. M. (1992). Embedding theoretical models in neural networks. In *Proc. American Control Conference, Chicago, Il.*, pages 475–479.

Kuipers, B. and Åström, K. (1994). The composition and validation of heterogeneous control laws. Automatica **30**, 233–249.

Kurcova, V. (1992). Universal approximation using feedforward neural networks with Gaussian Bar units. In *Proc. ECAI'92, Vienna*, pages 193–197.

Lane, S. H., Handelman, D. A., and Gelfand, J. J. (1991). Higher order CMAC neural networks - theory and practice. In *Proc. American Control Conference, Boston, USA*, pages 1579–1585.

Lee, S. and Kil, M. R. (1991). A gaussian potential function network with hierarchically self-organizing learning. Neural Networks **4**, 207–224.

Leonard, J. A., Kramer, M. A., and Ungar, L. H. (1992). A neural network architecture that computes its own reliability. MIT Industrial Liason Report 3-7-92, MIT, Dept. of Chemical Engineering.

Leontaritis, I. J. and Billings, S. A. (1985). Input-output parametric models for non-linear systems. Int. J. Control **41**, 303–344.

Ljung, L. (1987). *System Identification — Theory for the User.* Prentice-Hall, Englewood cliffs, New Jersey, USA.

Lowe, D. (1994). Non local Radial Basis Functions for forecasting and density estimation. In *IEEE Intern. Conf. on Neural Networks*, volume II, pages 1197–1198, Florida.

Mason, J. C. and Parks, P. C. (1992). Selection of neural network architectures: Some approximation theory guidelines. In *In: K. Warwick, G. W. Irwin, K. J. Hunt (Eds), Neural networks for control and systems*, pages 151–180. Peter Peregrinus.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics **5**, 115–133.

Mesarovic, M. D., Macko, D., and Takahara, Y. (1970). *Theory of Hierarchical, Multilevel Systems.* Academic Press.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1983). *Machine Learning: an artificial intelligence approach.* Morgan Kaufmann, Los Altos.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1986). *Machine Learning: an artificial intelligence approach (Vol. 2).* Morgan Kaufmann, Los Altos.

Miller, W. T., Sutton, R. S., and Werbos, P. J., editors (1990). *Neural Networks for Control.* MIT Press, Cambridge, MA.

Minsky, M. L. and Papert, S. A. (1969). *Perceptrons.* The MIT press, Cambridge, Mass.

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

Murray-Smith, R. (1992). A Fractal Radial Basis Function network for modelling. In *Inter. Conf. on Automation, Robotics and Computer Vision, Singapore*, volume 1, pages NW–2.6.1–NW–2.6.5. E-mail:murray@DBresearch-berlin.de.

Murray-Smith, R. (1994). Local Model Networks and Local Learning. In *Fuzzy Duisburg, '94*, pages p404–409. E-mail:murray@DBresearch-berlin.de.

Murray-Smith, R. and Gollee, H. (1994). A constructive learning algorithm for local model networks.

In *Proc. IEEE Workshop on Computer-intensive methods in control and signal processing, Prague, Czech Republic*, pages 21–29. E-mail:murray@DBresearch-berlin.de.

Murray-Smith, R., Neumerkel, D., and Sbarbaro-Hofer, D. (1992). Neural Networks for Modelling and Control of a Non-linear Dynamic System. In *IEEE Symposium on Intelligent Control, Glasgow*, pages p404–409. E-mail:murray,neumerk@DBresearch-berlin.de.

Murray-Smith, R. and Thakar, S. (1993). Combining Case Based Reasoning with neural networks. In *AAAI'93 Workshop on AI in Service and Support, Washington DC.* E-mail:murray,thakar@DBresearch-berlin.de.

Neumerkel, D., Murray-Smith, R., and Gollee, H. (1993). Modelling dynamic processes with clustered time-delay neurons. In *Proc. International Joint Conference on Neural Networks, Nagoya, Japan.* E-mail:neumerk,murray,gollee@DBresearch-berlin.de.

Noble, B. and Daniel, J. W. (1988). *Applied linear algebra.* Prentice–Hall Int., 3rd edition.

Omohundro, S. M. (1987). Efficient algorithms with neural network behavior. J. Complex Systems **1**, 273–347.

Omohundro, S. M. (1991). Bumptrees for efficient function, constraint and classification learning. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 693–699. Morgan Kaufmann Publishers, San Franciso, CA.

Pantaleón-Prieto, C. J., de María, F. D., and Figueiras-Vidal, A. (1993). On training RBF networks. In *Neural Networks and their Industrial & Cognitive Applications, Nimes, France*, pages 279–288.

Pao, Y.-H. (1992). Functional link net computing. Computer , 76–79.

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

Park, J. and Sandberg, I. W. (1993). Approximation and Radial-Basis-Function networks. Neural Computation **5**, 305–316.

Parker, D. (1985). Learning-logic. Tech. Report. TR-47, Center for Computational Research in Economics and Management Science, MIT, MA.

Platt, J. (1991). A Resource-Allocating Network for function interpolation. Neural Computation **3**, 213–225.

Plutowski, M. (1994). *Selecting Training Exemplars for Neural Network Learning.* Ph.D. Thesis, University of California, San Diego, USA.

Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. In *Proceedings of the IEEE*, volume 78, pages 1481–1497.

Pottmann, M., Unbehauen, H., and Seborg, D. E. (1993). Application of a general multi-model approach for identification of highly nonlinear processes - a case study. Int. J. Control **57**, 97–120.

Powell, M. J. D. (1987). Radial Basis Functions for multivariable interpolation: A review. In *Algorithms for Approximation*, pages 143–167, Oxford. Clarendon Press.

Press, W. H. et al. (1988). *Numerical Recipes (C): The Art of Scientific Computing.* Cambridge Press, UK.

Priestley, M. B. (1988). *Non-linear and Non-stationary Time Series Analysis.* Academic Press.

Quinlan, J. (1992). Learning with continuous classes. In *Australian AI Conf*, pages 343–348.

Quinlan, J. (1993). *C4.5 Programs for Machine Learning.* Morgan Kaufmann.

Raipala, J. and Koivo, H. N. (1992). Self-generating radial base network in fault diagnosis. In *Inter.*

Conf. on Automation, Robotics and Computer Vision, Singapore, volume 1, pages NW–3.2.1–NW–3.2.5.

Rich, E. (1988). *Artificial Intelligence.* McGraw-Hill.

Roberts, S. and Tarassenko, L. (1994). A probabilistic resource allocating network for novelty detection. Neural Computation **6**, 270–284.

Röscheisen, M., Hofmann, R., and Tresp, V. (1992). Neural control for rolling mills: Incorporating domain theories to overcome data deficiency. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 659–666, San Mateo, CA. Morgan Kaufmann.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review **65**, 386–408.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing.* MIT Press, Cambridge, Mass.

Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructures of Cognition,* Vol. 1: *Foundations.* MIT Press, Cambridge, Mass.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3**, 210–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. part II–recent progress. IBM Journal of Research and Development **11**, 601–617.

Sanger, T. D. (1991a). A tree-structured adaptive network for function approximation in high-dimensional spaces. IEEE Trans. on Neural Networks **2**, 285–293.

Sanger, T. D. (1991b). A tree-structured algorithm for reducing computation in networks with separable basis functions. Neural Computation **3**, 67–78.

Sanner, R. M. and Slotine, J.-J. (1992). Gaussian networks for direct adaptive control. IEEE Trans. Neural Networks **3**, 837–863.

Sbarbaro, D. (1992a). *Connectionist Feedforward Networks for Control of Nonlinear Systems.* Ph.D. Thesis, Department of Mechanical Engineering, Glasgow University, Glasgow, Scotland.

Sbarbaro, D. G. (1992b). A comparative study of different learning algorithms for gaussian networks. In *IFAC symposium on Intelligent Components and Intruments for Control Applications. Malaga, Spain*, pages 301–305.

Shamma, J. S. and Athans, M. (1991). Gain scheduling: Potential hazards and possible remedies. In *Proceedings American Control Conference, Boston, Ma.*, pages 516–251.

Shavlik, J. W. and Ditterich, T. G. (1990). *Readings in Machine Learning.* Morgan Kaufmann, San Mateo.

Shorten, R. and Murray-Smith, R. (1994). On Normalising Basis Function networks. In *4th Irish Neural Networks Conf., Univ. College Dublin.*

Sjöberg, J., Hjalmarsson, H., and Ljung, L. (1994). Neural networks in system identification. Technical Report LiTH-ISY-I-1622, Dept. of Electrical Engineering, Linköping University, Sweden. Ftp address:130.236.24.1 cd pub/Reports/1994.

Sjöberg, J. and Ljung, L. (1992). Overtraining, regularization, and searching for minimum in neural networks. In *Proc. IFAC Symposium on Adaptivve Systems in Control and Signal Processing,* *Grenoble, France.*, pages 669–674.

Sjöberg, J., T.McKevey, and Ljung, L. (1993). On the use of regularization in system identification. In *Proc. 12th World Congress IFAC, Sydney Australia.*, volume 7, pages 381–386.

Skeppstedt, A., Ljung, L., and Millnert, M. (1992). Construction of composite models from observed data. Int. J. Control **55**, 141–152.

Sklansky, J. (1966). Learning systems for automatic control. IEEE Trans. on Automatic Control **11**, 6–19.

Söderström, T. and Stoica, P. (1989). *System Identification.* Prentice Hall, Englewood Cliffs, NJ.

Specht, D. F. (1991). A general regression neural network. Trans. IEEE on Neural Networks **2**.

Steinbuch, K. (1961). Die lernmatrix. Kybernetik **1**, 36–45.

Steinbuch, K. (1963). *Automat und Mensch.* Springer Verlag.

Stokbro, K., Umberger, D. K., and Hertz, J. A. (1990). Exploiting neurons with localized receptive fields to learn chaos. Complex Systems **4**, 603–622.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. J. Royal Statistical Soc. B **36**, 111–133.

Strömberg, J.-E., Gustafsson, F., and Ljung, L. (1991). Trees as black-box model structures for dynamical systems. In *European Control Conference, Grenoble*, pages 1175–1180.

Sugeno, M. and Kang, G. T. (1988). Structure identification of fuzzy model. Fuzzy Sets and Systems **26**, 15–33.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications for modeling and control. IEEE Trans. on Systems, Man and Cybernetics **15**, 116–132.

Thrun, S. B. (1992). The role of Exploration in Learning Control. In *Handbook of Intelligent Control: Neural Fuzzy and Adaptive Approaches.* Van Nostrand Reinhold.

Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of Ill-posed problems.* Winston, Washington DC.

Tong, H. (1990). *Non-linear Time Series: A Dynamical System Approach.* Oxford University Press. Oxford Statistical Science Series 6.

Tsypkin, Y. Z. (1971). *Adaptation and Learning in Automatic Systems.* Academic Press, New York.

Tsypkin, Y. Z. (1973). *Foundations of the Theory of Learning Systems.* Academic Press, New York.

Vaccaro, R. J., editor (1991). *SVD and Signal Processing II: Algorithms, Analysis and Applications.* Elsevier.

van der Veen, A.-J., Deprettre, E. F., and Swindlehurst, A. L. (1993). Subspace-based signal analysis using singular value decomposition. IEEE Proceedings **81**, 1277–1308.

Wahba, G. (1990). Spline models for observation data. In *Regional Conference Series in Applied Mathematics*, Philadelphia, PA. SIAM.

Wahba, G. (1992). Multivariate function and operator estimation, based on smoothing splines and reproducing kernels. In Casdagli, M. and Eubank, S., editors, *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity*, volume XII. Addison-Wesley.

Wang, L.-X. (1994). *Adaptive Fuzzy Systems and Control: Design and Stability Analysis.* Prentice Hall.

Warwick, K., Irwin, G. W., and (Eds), K. J. H., editors (1992). *Neural networks for control and systems.* Peter Peregrinus.

Wassermann, P. D. (1993). *Advanced methods in neural computing.* Van Nostrand Reinhold, New

York.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems that Learn.* Morgan Kaufmann, San Mateo, California.

Werbos, P. J. (1974). *Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Masters Thesis, Harvard University.

Werntges, H. W. (1993). Partitions of unity improve neural function approximation. In *Proc. IEEE Int. Conf. Neural Networks*, pages 914–918, San Francisco, CA. Vol. 2.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, volume 4, pages 96–104. IRE, New York.

Wiener, N. (1948). *Cybernetics: or Control and Communication in the Animal and the Machine.* MIT Press.

Xu, L., Jordan, M. I., and Hinton, G. (1994). An alternative model for mixtures of experts. Submitted to NIPS'94.

Żbikowski, R. (1994). *Recurrent Neural Networks: Some Control Problems.* Ph.D. Thesis, Department of Mechanical Engineering, Glasgow University, Glasgow, Scotland.

Zhao, Y. (1992). *On Projection Pursuit Learning.* PhD thesis, MIT, Dept. of Mathematics and the AI Lab.

Index