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.

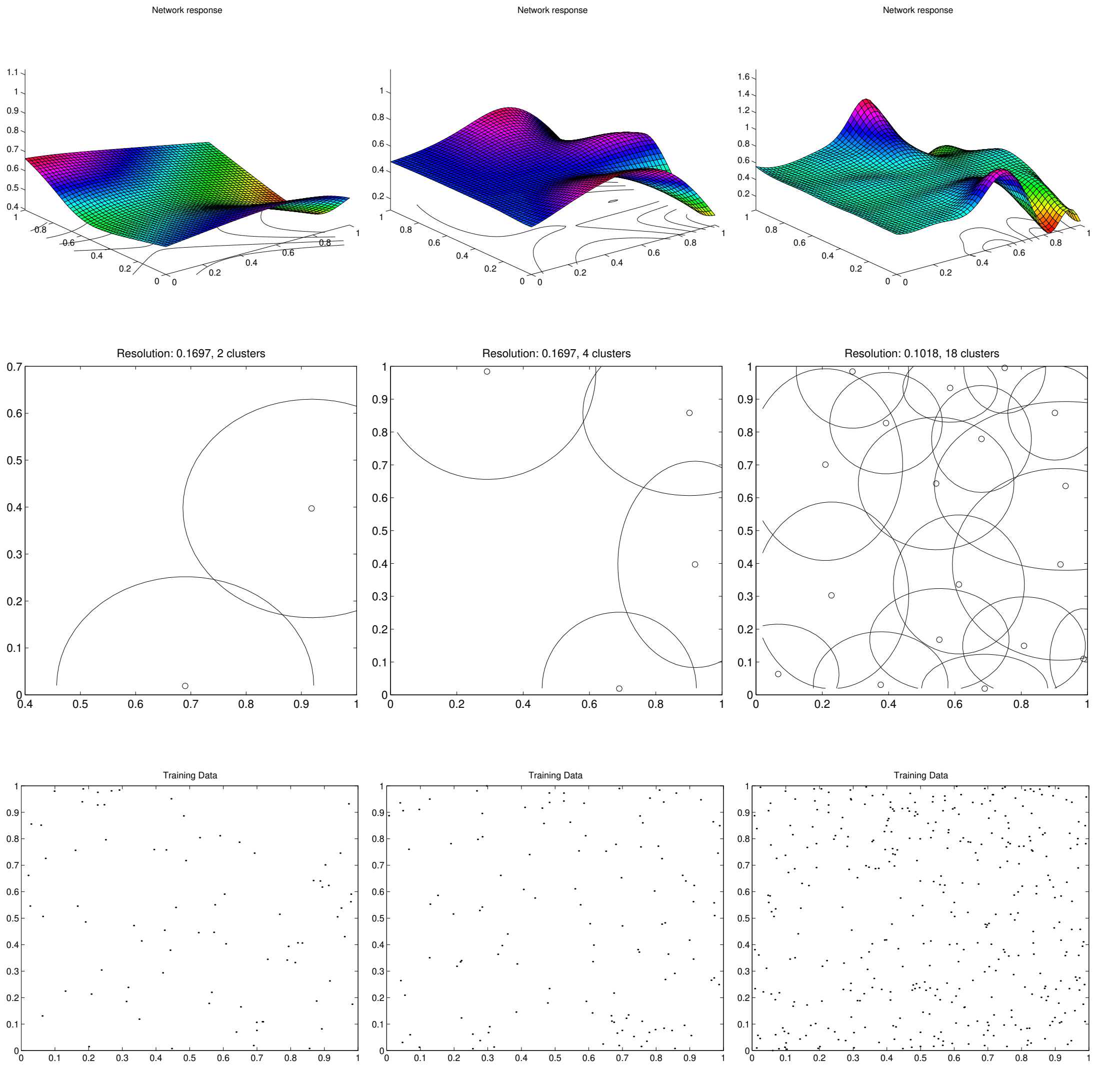


Figure 4.1: A gradual approach to constructing a model. The model responses are shown for a series of stages in model development, shown by the contour plots of the basis functions. The bottom row shows the training data used – note that the set is expanded with the model structure. The target function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

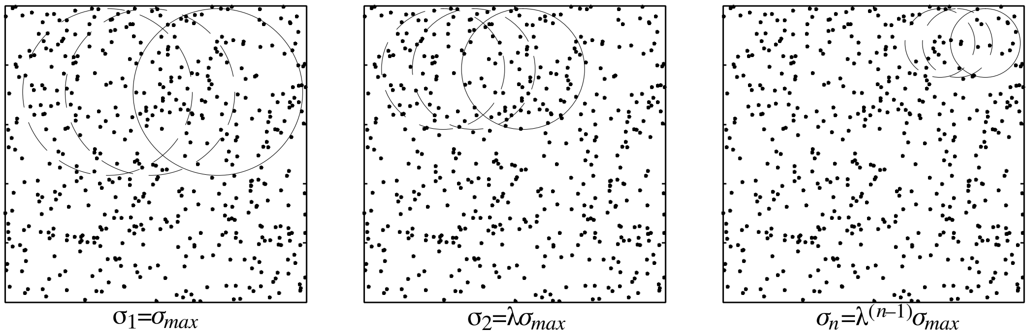


Figure 4.2: Multi-Resolution Windowing. The complexity measure is applied at different scales, and at each scale σ_i the 'window' is centred on points from the training set. The points corresponding to the greatest values of the complexity measure are those suggested as new centres for basis functions.

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 $\mathbf{c}_{new} \in \mathcal{R}^{n_\phi}$, 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.

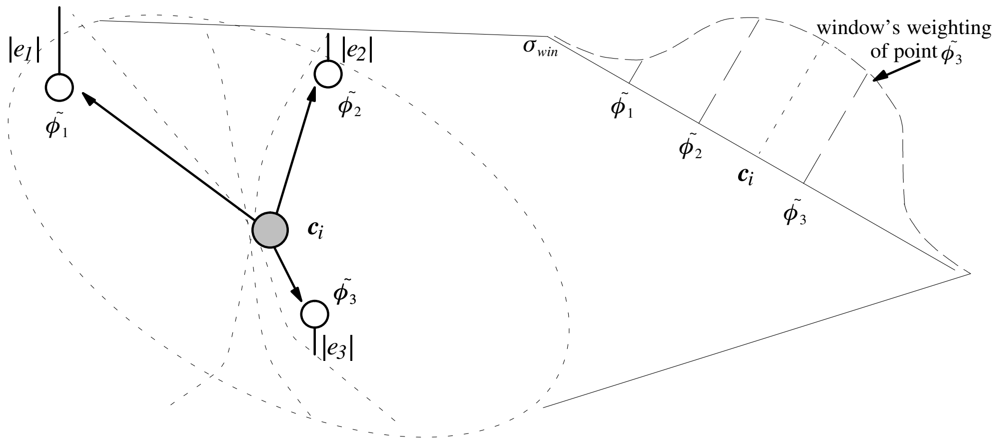


Figure 4.3: Windowed Complexity Estimate. The weighting function $\rho(\cdot)$ weights the error measures e_i at points $\tilde{\boldsymbol{\phi}}_i$ around the prototype centre. The window function is usually chosen such that points further from the centre have less effect on the outcome of the complexity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

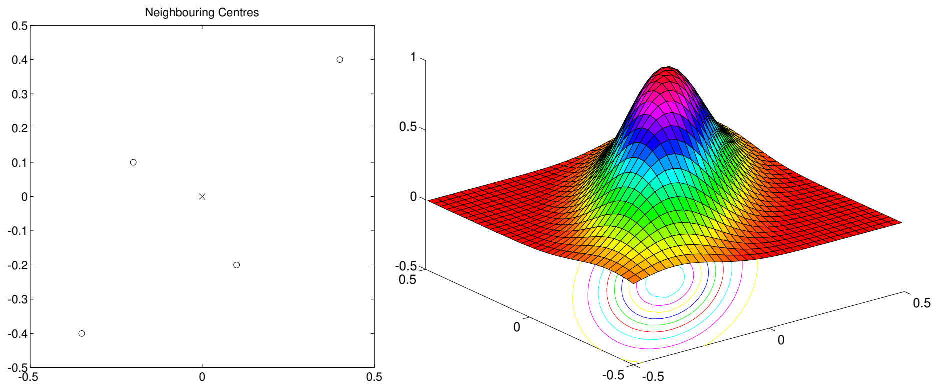
$$\boldsymbol{\sigma}_i \propto E_n\left[(\mathbf{c}_n - \mathbf{c}_i)(\mathbf{c}_n - \mathbf{c}_i)^T\right], \tag{4.6}$$

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.

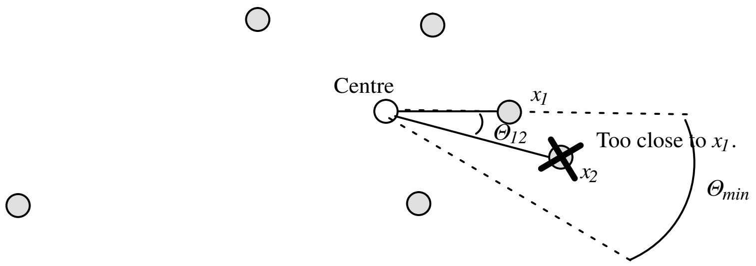


Figure 4.5: Eliminating centres with a common direction. The neighbouring points are used to define a 'covariance' matrix which is used to determine the overlap between neighbouring units. Neighbours in the same direction are ignored.

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

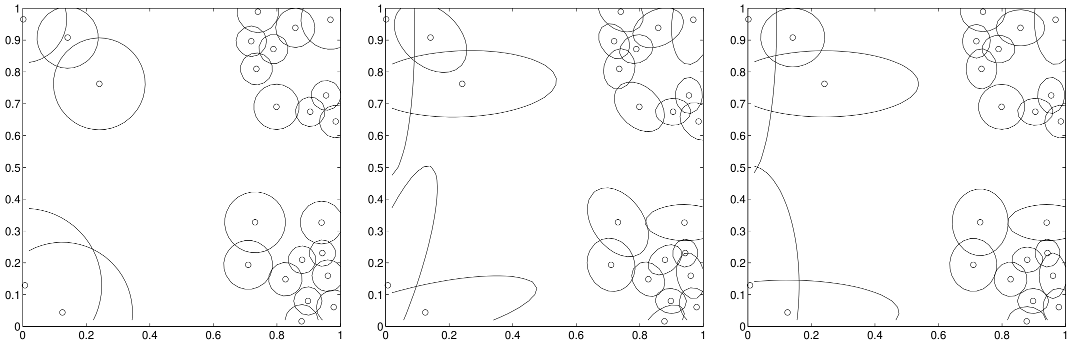


Figure 4.6: Radial, Ellipsoidal and Axis-orthogonal ellipsoidal basis functions. Contours are drawn at a level of 0.5.

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

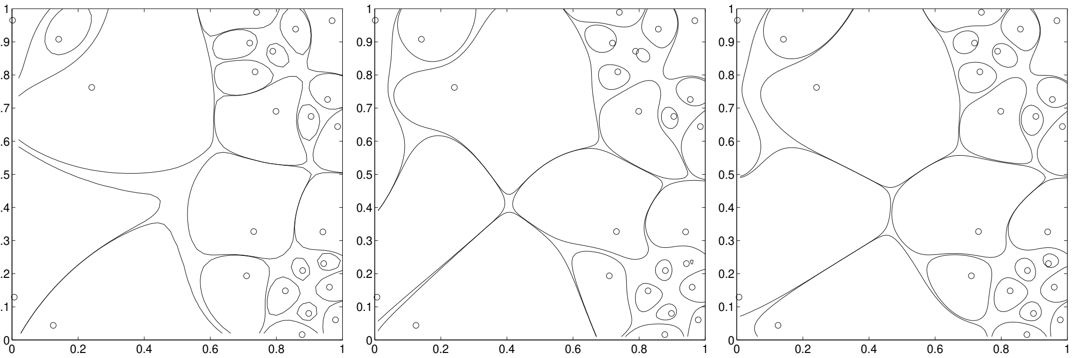


Figure 4.7: Normalised Radial, Ellipsoidal and Axis-orthogonal ellipsoidal basis functions. Contours are drawn at a level of 0.5.

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.

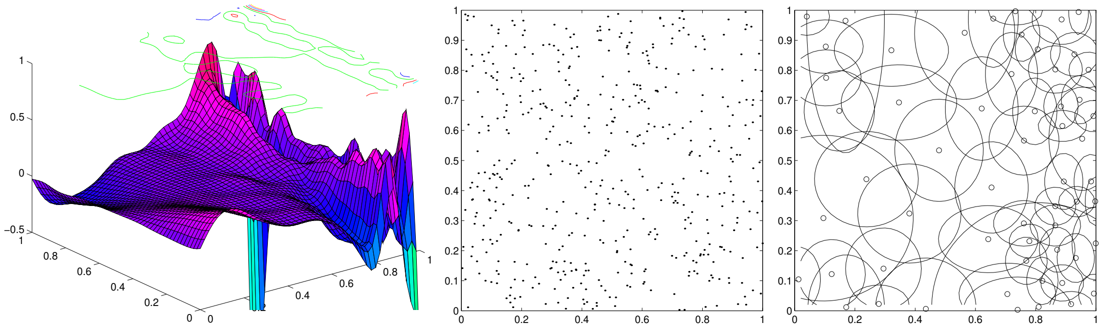
$$\boldsymbol{\psi} \in \mathcal{D}_i, \;\; \text{if} \;\; \rho_i(d(\tilde{\boldsymbol{\phi}}; \mathbf{c}_i, \sigma_i)) > \zeta. \tag{4.9}$$

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

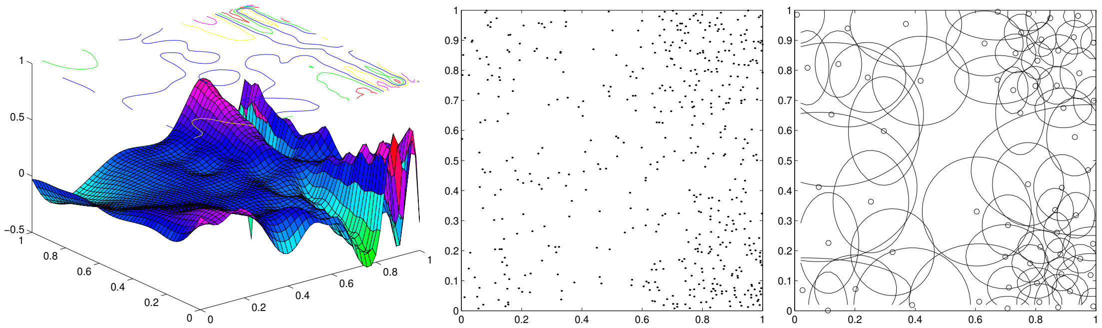
Unevenly distributed training data is a common feature of real industrial processes which are often relatively simple around the normal operating conditions, where most of the data is collected. The process often becomes increasingly complex as the operating point moves away from the given setpoint, 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



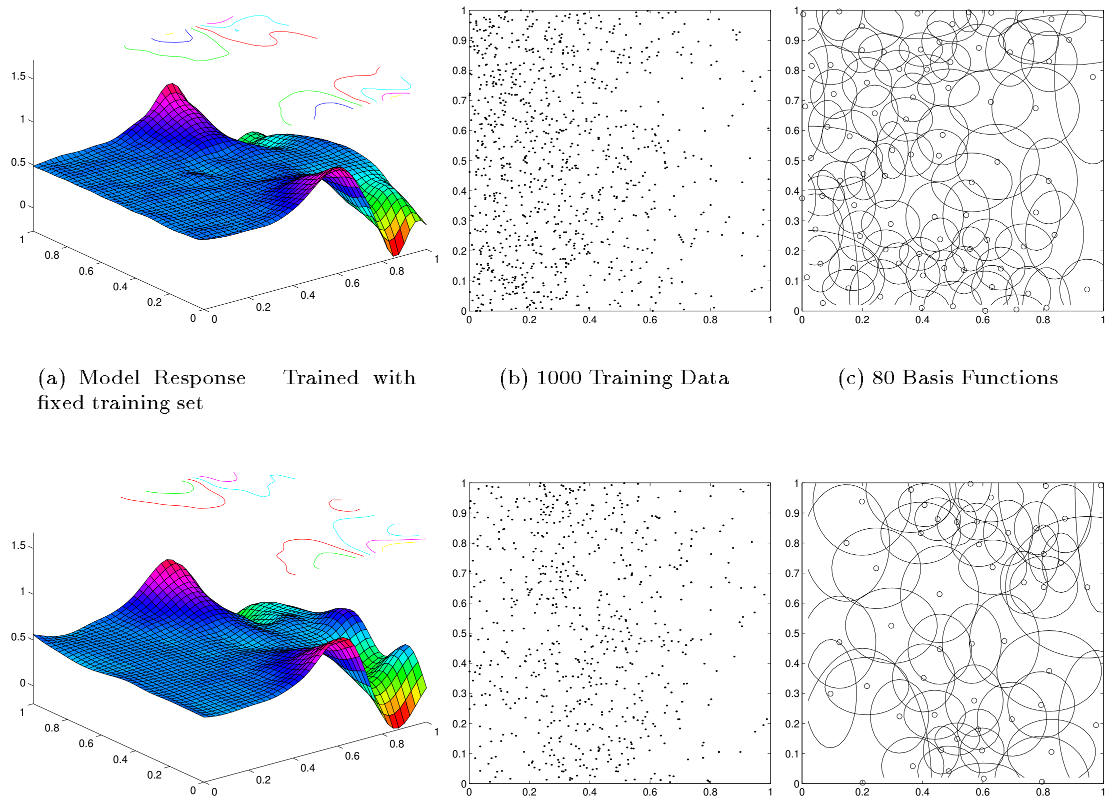
(d) Response - Trained using Active learning

(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

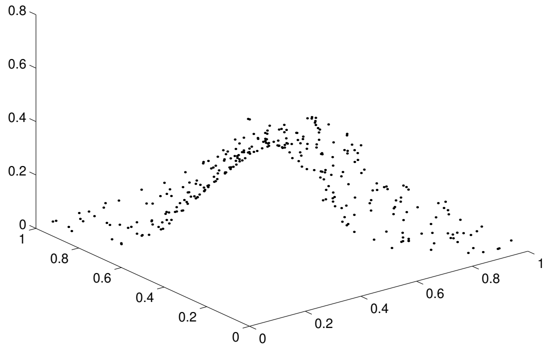
This is a benchmark from the nonparametric statistics literature, used in (Gu et al., 1990, Friedman, 1991).

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

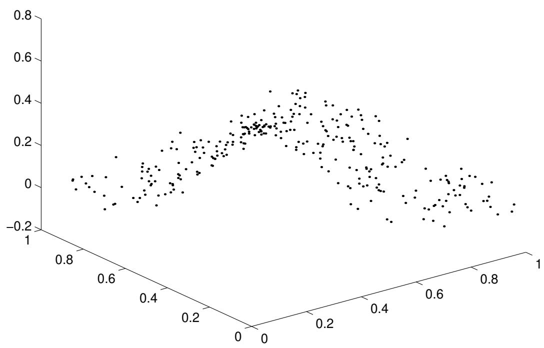
It is typical of a smooth nonlinear function, which should be relatively easy for a learning system to model. It is a collinear function (where the nonlinearity is dependent on linear combinations of the input dimensions), which makes it more difficult for methods which partition the input space orthogonal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

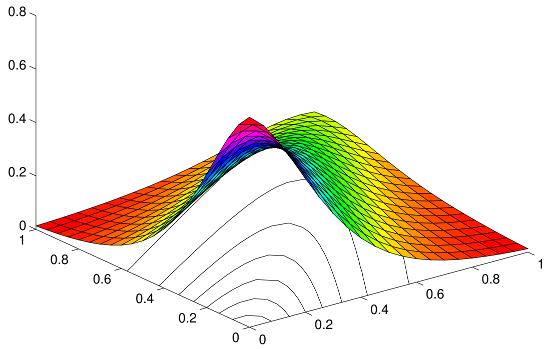
The response of Local Model Network trained on the Mars1 data is shown in Figure 4.11.



(a) Mars1 training data

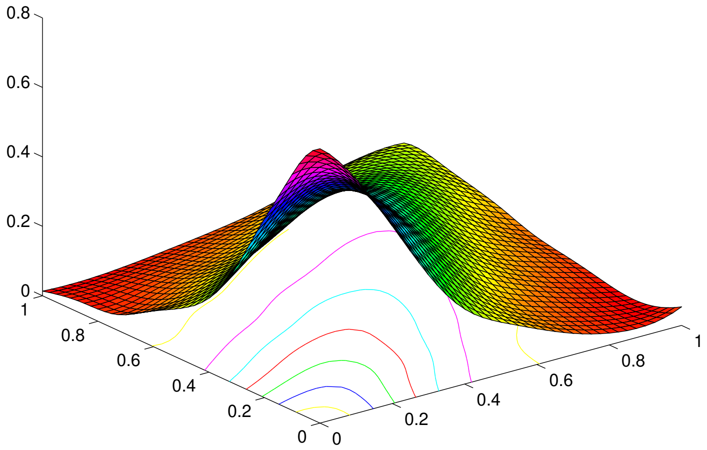


(b) Mars1 noisy training data

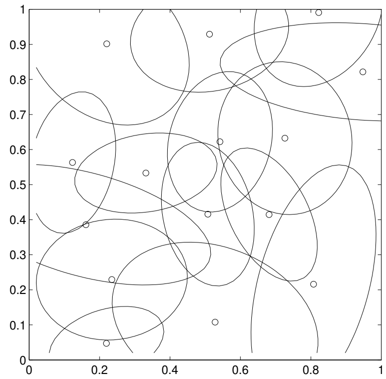


(c) Mars1 function

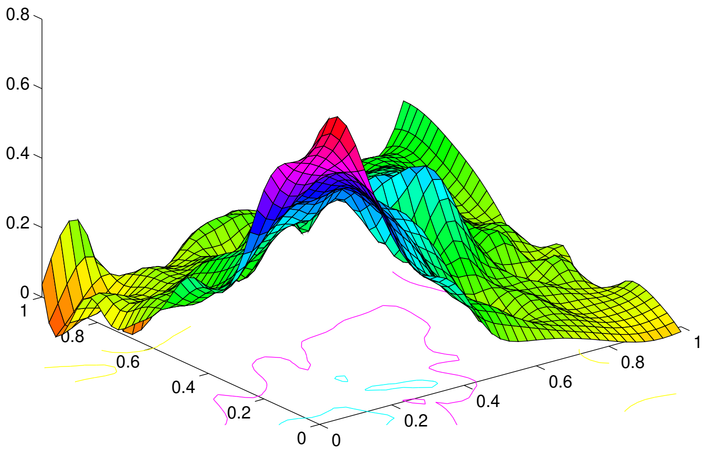
Figure 4.10: Mars1 test function and 300 training points. z axis is vertical. x and y are right and left respectively.



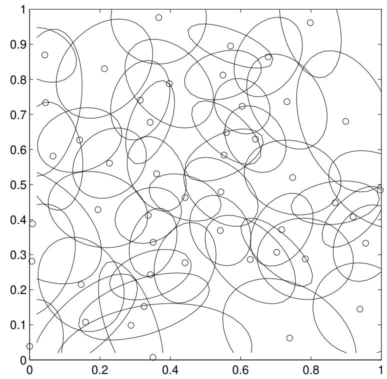
(a) LMN Response $n_{res} = 5, \lambda = 0.6, \eta = 0.005$



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

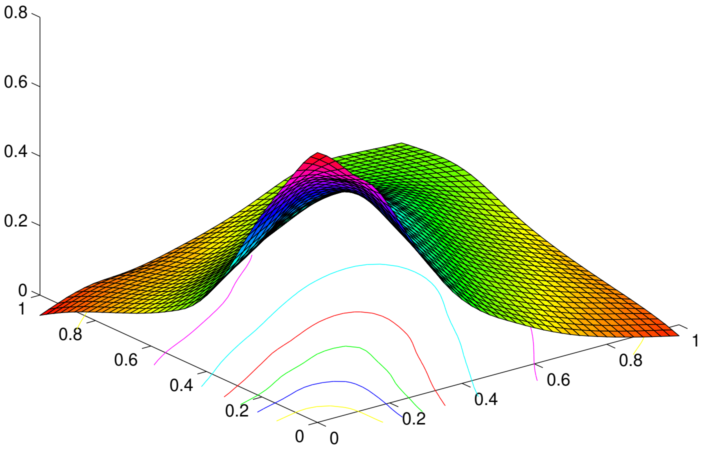


(c) LMN Response on noisy data $n_{res} = 5, \lambda = 0.6, \eta = 0.02$

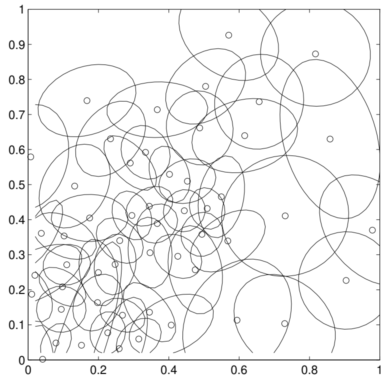


(d) LMN Basis Functions for noisy data. Contours are drawn at 0.5

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.

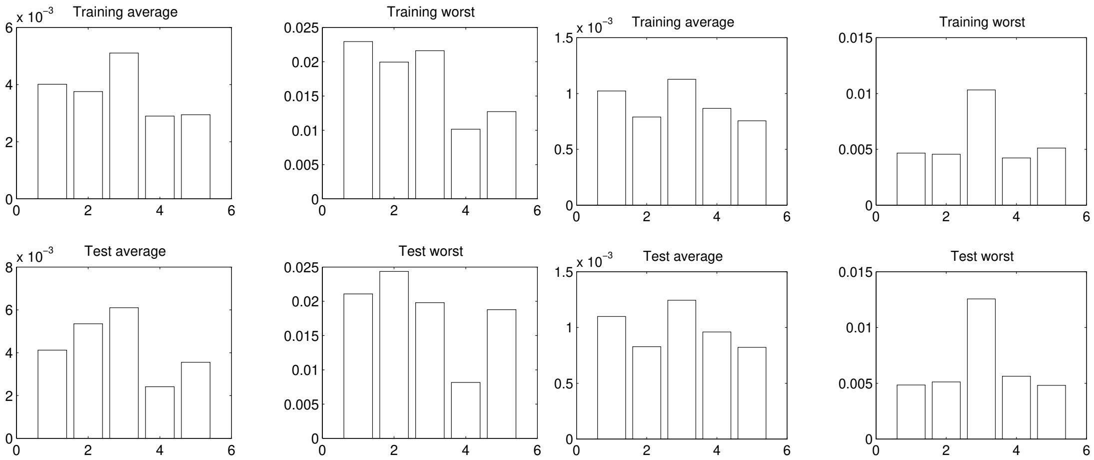


(a) LMN Response on noisy data $n_{res} = 5, \lambda = 0.6, \eta = 0.02$, using local learning.



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

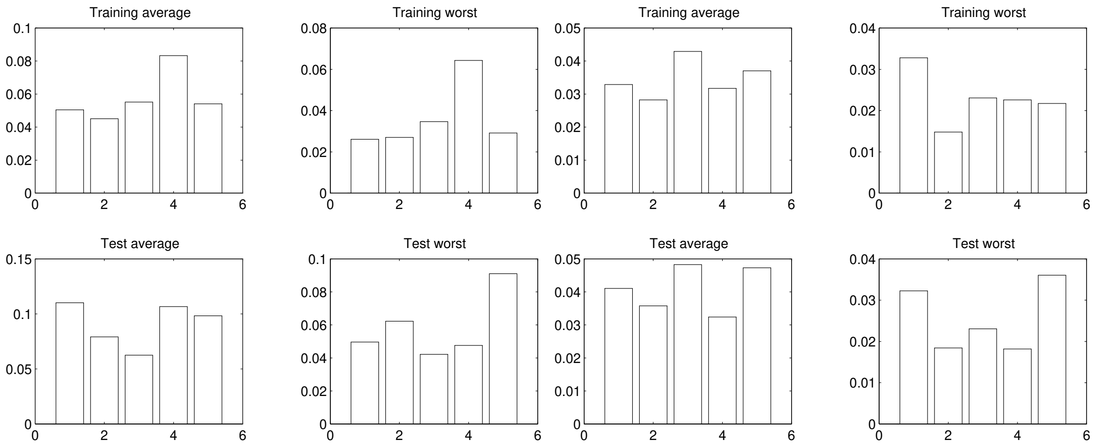
Figure 4.12: Resulting LMN responses for mars1 benchmark (300 training points) using local learning. Note the far smoother response than that of the globally trained network in Figure 4.11.



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

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

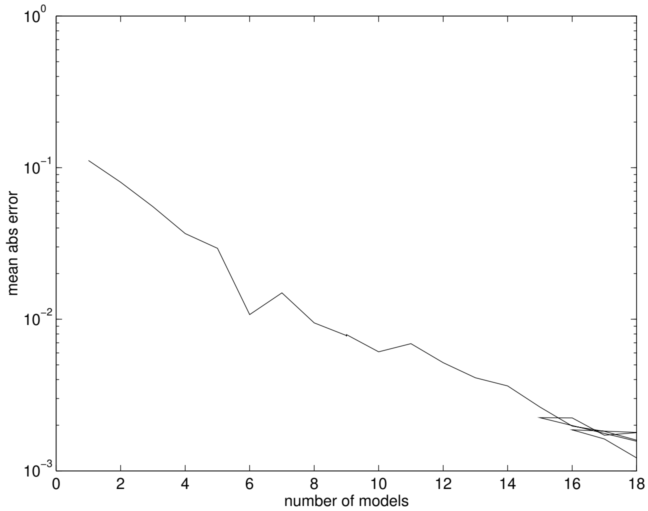
Figure 4.13: Cross-validation results for Local Model Net on noise-free Mars example. Error decreases with the increased amount of training data.



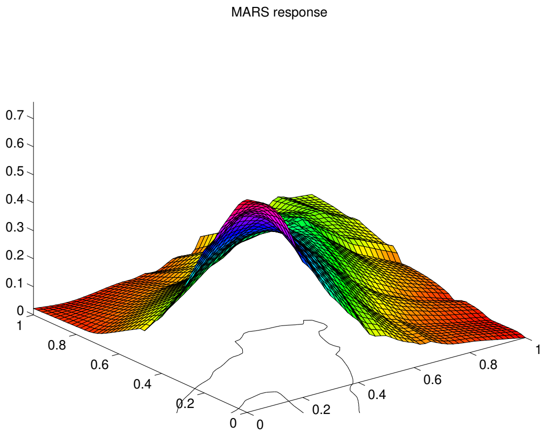
(a) Mars1 problem, 300 training points and MARS algorithm

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

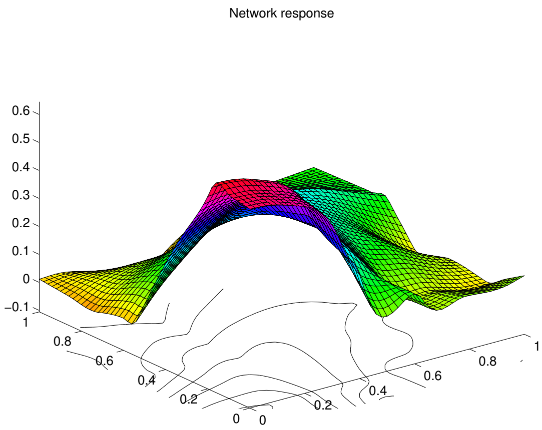
Figure 4.14: Cross-validation results for MARS algorithm on the noise-free Mars example. Note that MARS performs worse on training and test measures than the Local Model Net, and there is less similarity in the results for training and test runs than there is with the Local Model Nets.



(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



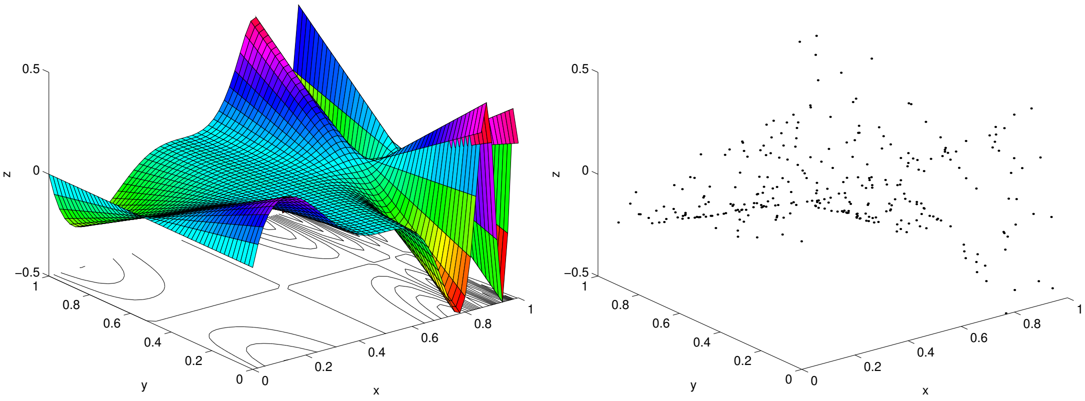
(c) MARS Response on noisy data

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

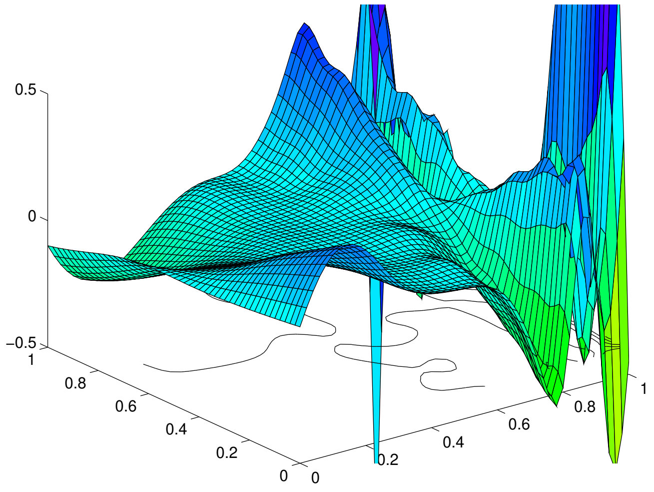
Figure 4.16: Squiggle test function and 300 training data points

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.

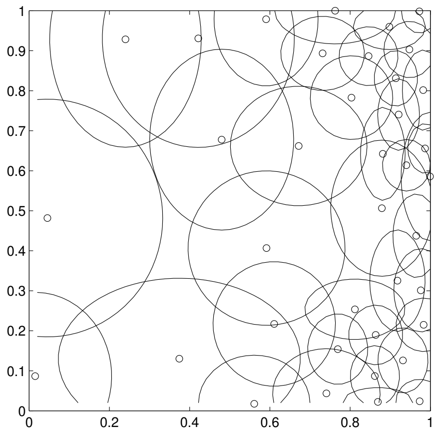
The MARS algorithm performs very well on the Squiggle function, because the nonlinearity is axis-orthogonal, and the function is only highly nonlinear in x, and MARS strength lies in its ability to separate important inputs from unimportant inputs when developing the model structure. Other axis orthogonal 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

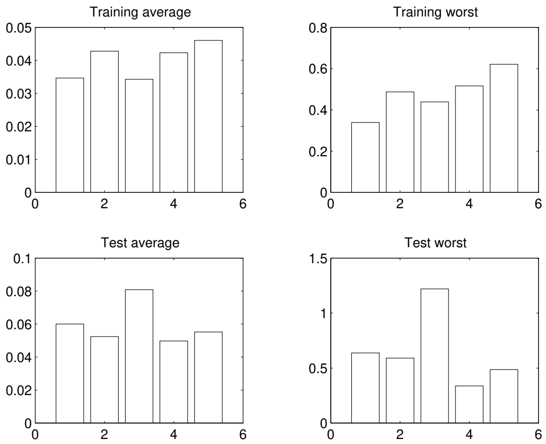
To show the disadvantage of axis-orthogonal representations, however, all we have to do is rotate the function. In this case it has been rotated by $\frac{\pi}{4}$ radians. The rotated squiggle function and various responses are shown in Figure 4.19. The mean absolute error for MARS, trained on 1000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



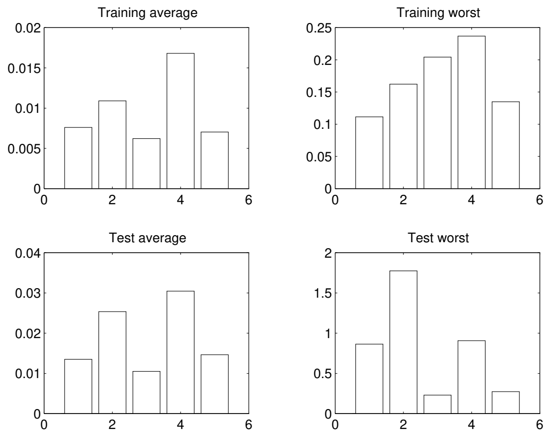
(a) LMN Response



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

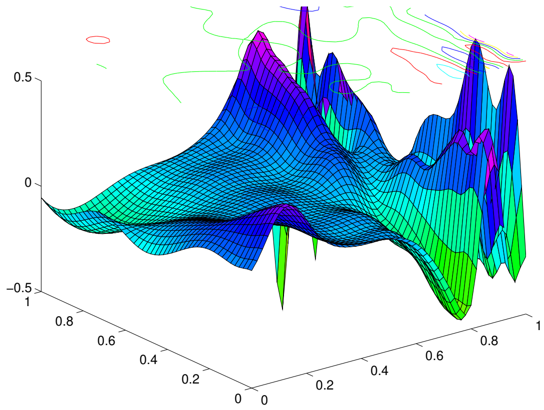


(c) Squiggle problem, 300 training points and Local Model Net

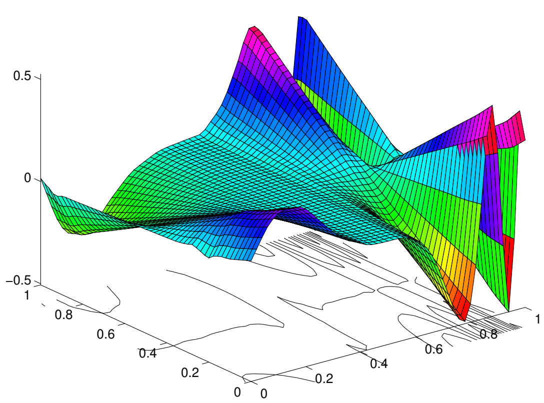


(d) Squiggle problem, 1000 training points and Local Model Net

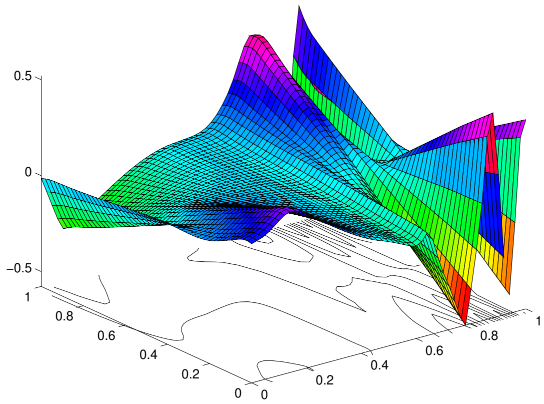
Figure 4.17: Resulting LMN for Squiggle benchmark (1000 training points), with 38 axis-orthogonal ellipsoidal basis functions and global learning. Compare with MARS response in Figure 4.18 which is very close to the original function. The LMN for the noisy data had $N_{min} = 30$, instead of 15, and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

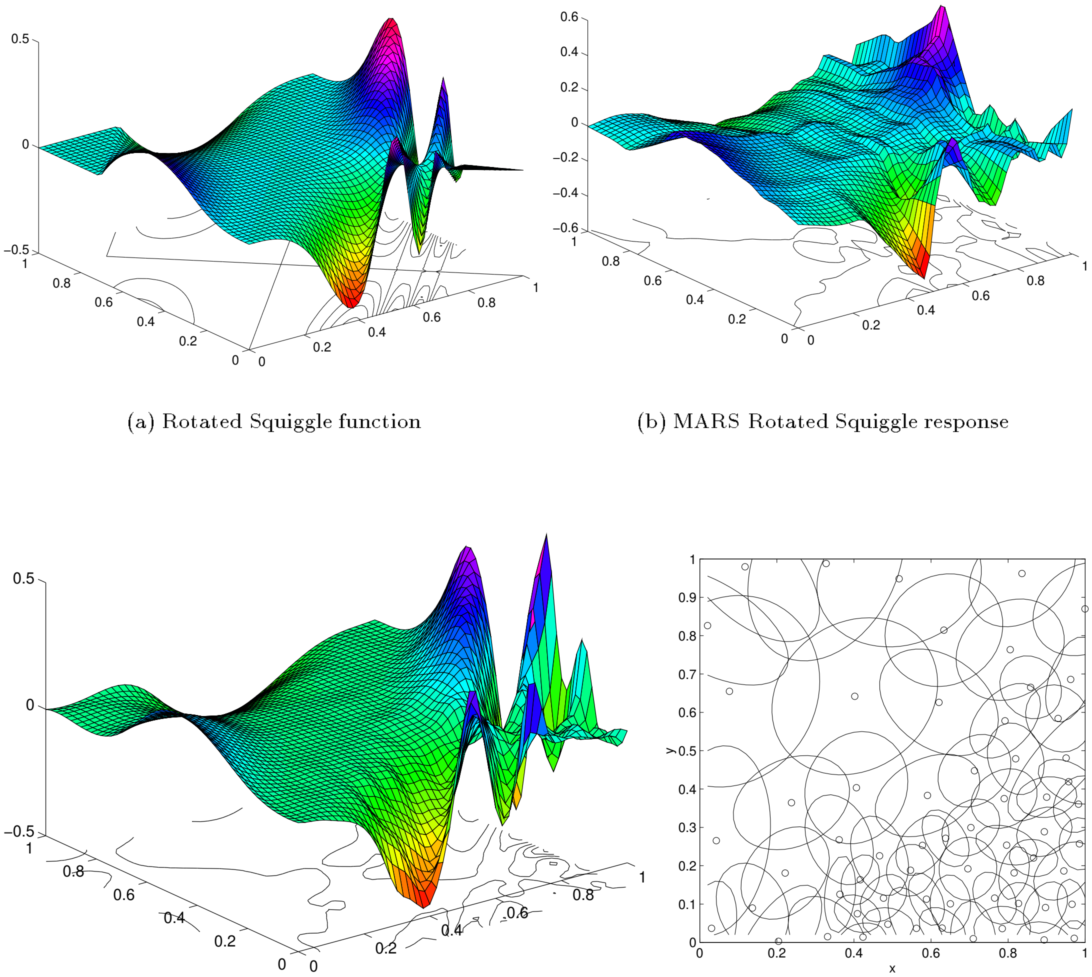


(b) MARS Response on clean data



(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:

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

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

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

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

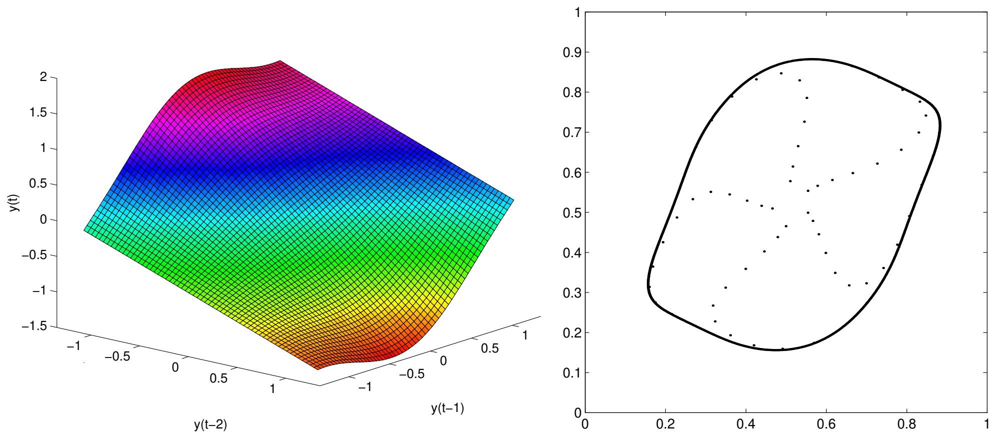
Nonlinear time-series example

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

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

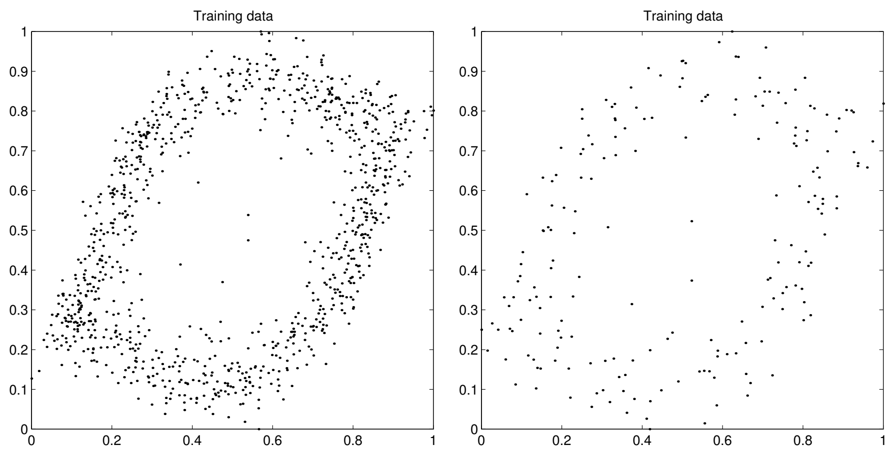
$$+ \left[0.3 + 0.9\exp(-y^2(t-1))\right] y(t-2) + e(t), \tag{4.20}$$

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)

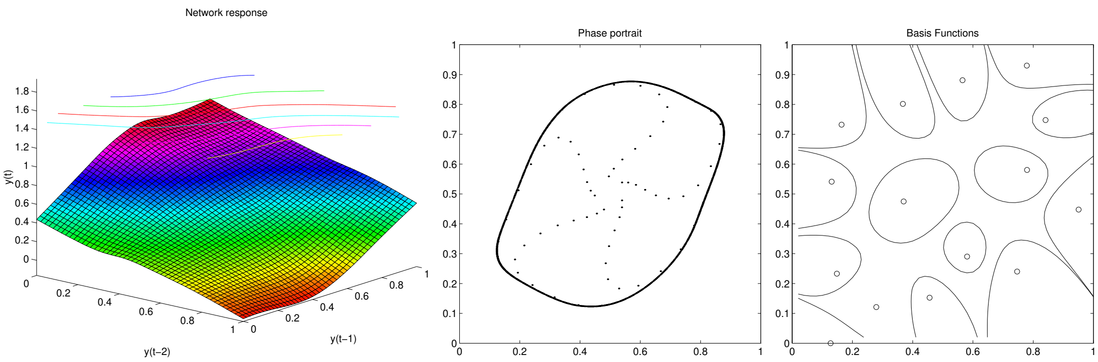


(c) Training Data - 1000 examples

(d) Training Data - 200 examples

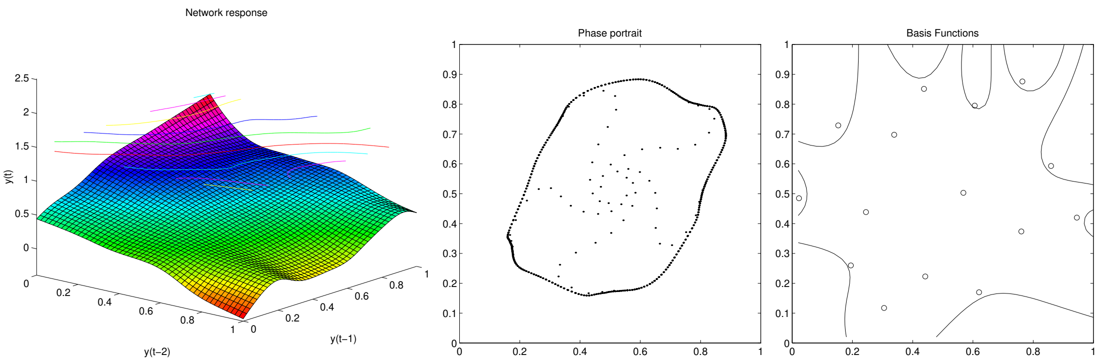
Figure 4.20: 4.20(a) Time series function response. Figure 4.20(b) Phase portrait. Figure 4.20(c) and Figure 4.20(d) are noisy data sets.

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

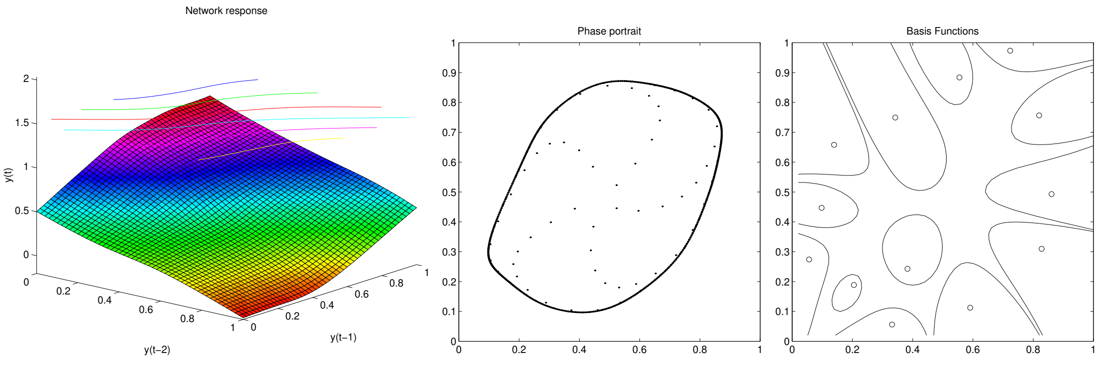
Figure 4.21: Time series model. Local model net with local learning, trained on 1000 examples.



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

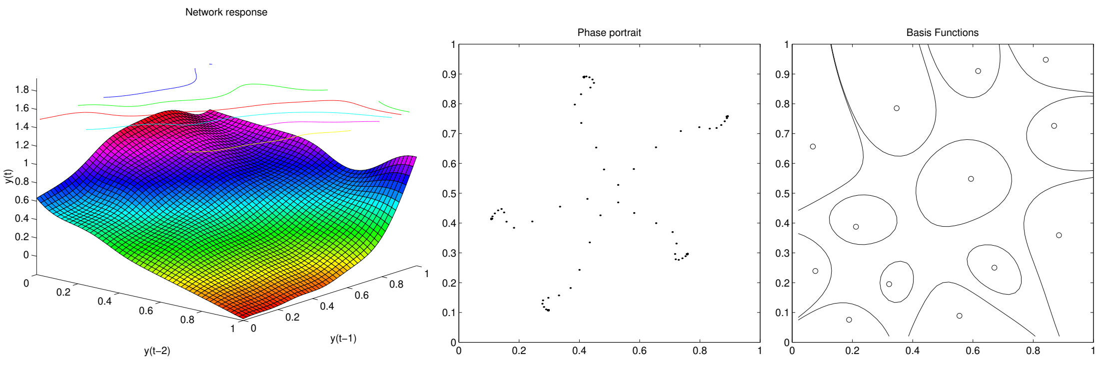
Figure 4.22: Time series model. Local model net with global learning, trained on 1000 examples. Note the smoother approximation of the locally trained model.

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.