Local Model Architectures for Nonlinear Modelling and Control

Roderick Murray-Smith and Kenneth Hunt

Daimler-Benz AG, Alt-Moabit 91 b, D-10559 Berlin, Germany.
E-mail: `murray` or `hunt` `@DBresearch-berlin.de`

Abstract. Local Model Networks are learning systems which are able to model and control unknown nonlinear dynamic processes from their observed input-output behaviour. Simple, locally accurate models are used to represent a globally complex process. The framework supports the modelling process in real applications better than most artificial neural network architectures. This paper shows how their structure also allows them to more easily integrate knowledge, methods and *a priori* models from other paradigms such as fuzzy logic, system identification and statistics. Algorithms for automatic parameter estimation and model structure identification are given.
Local Models intuitively lend themselves to the use of Local Controllers, where the global controller is composed of a combination of simple locally accurate control laws. A Local Controller Network (LCN) for controlling the lateral deviation of a car on a straight road is demonstrated.

1 Introduction

This paper presents methods for the automatic construction of local model networks. These are architectures capable of representing nonlinear dynamic systems by smoothly integrating a number of simpler locally accurate models. Local methods of controlling systems, such as gain scheduling, are very useful methods for nonlinear control, but determining the gain schedule becomes increasingly difficult as the process complexity increases. The *Local Controller Networks* presented here can be viewed as a general formulation of the gain scheduling idea, where the 'schedule' can be automatically determined by the *Local Model* structure constructed to represent the process. The effect of the choice of local or global parameter estimation methods on the resulting model structure is discussed.

1.1 Model Structures

We consider discrete-time nonlinear systems having the general form

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

Here, $y(t)$ is the system output, $u(t)$ the input and $e(t)$ is a zero-mean disturbance term. We restrict attention to single-input single-output systems so that $y(t) \in$

$Y \subset \mathcal{R}$, $u(t) \in U \subset \mathcal{R}$ and $e(t) \in E \subset \mathcal{R}$. k represents a time delay. This type of model is known as the NARMAX (Nonlinear ARMAX) model [1] and has been studied widely in nonlinear systems identification.

When we define the information vector as

$$\psi(t-1) = [y(t-1), \dots y(t-n_y), u(t-k), \dots u(t-k-n_u), e(t-1), \dots e(t-n_e)]^T \tag{2}$$

then the system (1) can be written as

$$y(t) = f(\psi(t-1)) + e(t). \tag{3}$$

The ^{T} in (2) denotes the transpose operator. The aim in empirical modelling is to find a parameterised structure which emulates the nonlinear function f.

1.2 Linear Models

Standard linear approaches to modelling consider a linearisation of (1) about a nominal operating point (e.g. by taking a first-order Taylor series expansion) which results in the linear ARMAX model

$$y(t) = \phi_{ar}^T(t-1)\theta + e(t) \tag{4}$$

where

$$\phi_{ar}(t-1) = [-y(t-1), \dots -y(t-n_y), u(t-k), \dots u(t-k-n_u), e(t-1), \dots e(t-n_e)]^T, \tag{5}$$

and

$$\theta = [a_1, \dots a_{n_y}, b_0, \dots b_{n_u}, c_1, \dots c_{n_e}]^T \tag{6}$$

is a constant parameter vector. Note that in (4)–(5) all signals now strictly represent deviations from the nominal operating point. With the definitions (5) and (6) the linear system (4) can be written in the familiar transfer-function form

$$y(t) = \frac{q^{-k}B(q^{-1})}{A(q^{-1})}u(t) + \frac{C(q^{-1})}{A(q^{-1})}e(t). \tag{7}$$

Here, q^{-1} is the unit delay operator and the polynomials A, B and C are

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + \dots + a_{n_y} q^{-n_y}, \\ B(q^{-1}) &= b_0 + b_1 q^{-1} + \dots + b_{n_u} q^{-n_u}, \\ C(q^{-1}) &= 1 + c_1 q^{-1} + \dots + c_{n_e} q^{-n_e}. \end{aligned} \tag{8}$$

1.3 Local Model Networks

Linear, or other simple models, can be used in a more general way, so that a more general class of systems can be represented. A number of such simple models, each weighted by a basis function associated with a different area of the input space, can be combined to create a globally complex model

$$\hat{y} = \hat{f}(\psi) = \sum_{i=1}^{n_{\mathcal{M}}} \hat{f}_i(\psi)\rho_i(\tilde{\phi}), \tag{9}$$

where $\tilde{\phi}$ defines the *operating point* of the system. The network form of equation (9) is shown in Fig. 1. The trained network structure can be viewed as a decomposition of the complex, nonlinear system into a set of $n_{\mathcal{M}}$ locally accurate sub-models, which are then smoothly integrated by their associated basis functions. To illustrate the workings of a local model network, a one dimensional

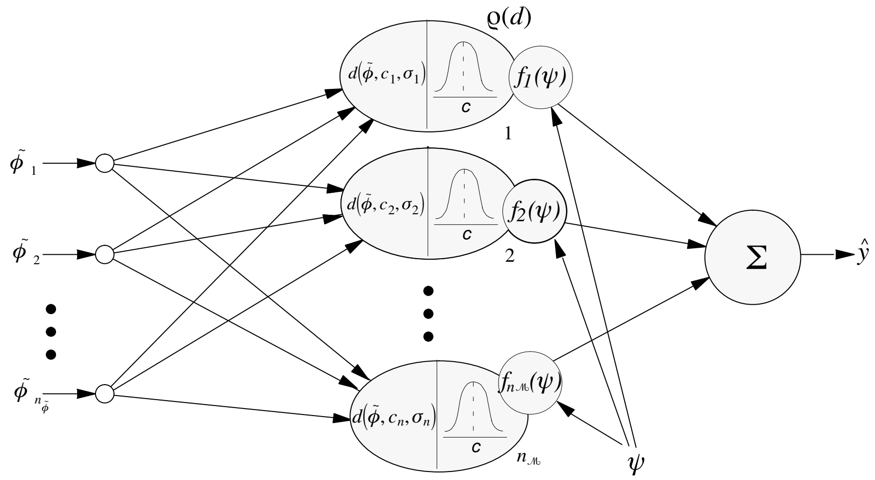


Fig. 1. Local Model Basis Function network

function is mapped using local models in Fig. 2.

The basis, or *model validity functions* used in this work are radial, i.e. they use a *distance metric* $d(\tilde{\phi}; c_i, \sigma_i)$ which measures the distance of the current operating point ψ from the basis function's centre c_i, relative to the width variable σ_i. See Fig. 3 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. As there is a limited amount of overlap, however, there is a limited number of local models associated with any point in the input space, making the full model more interpretable.

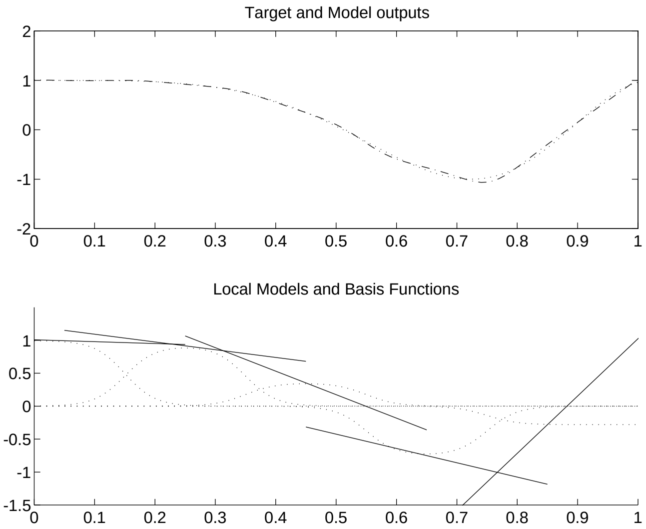


Fig. 2. An example of local models representing a one dimensional function. The top plot shows the target function and the model's approximation, while the basis functions and associated local models are shown below.

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. In many applications the network's basis functions are *normalised* to achieve the partition of unity, i.e.

$$\rho_k(\tilde{\phi}) = \frac{\rho\left(d\left(\tilde{\phi}, c_k, \sigma_k\right)\right)}{\sum_{i=1}^{n_{\mathcal{M}}} \rho\left(d\left(\tilde{\phi}, c_i, \sigma_i\right)\right)} \tag{10}$$

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

$$\sum_{i=1}^{n_{\mathcal{M}}} \rho_i\left(d\left(\tilde{\phi}, c_i, \sigma_i\right)\right) = 1. \tag{11}$$

[2] 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 we discussed in detail in [3]. These side-effects make the argument for or against the use of normalisation more complicated than is often assumed.

The use of local basis functions means that the structure has the advantages inherent to the local nature of the basis functions while, because of the more

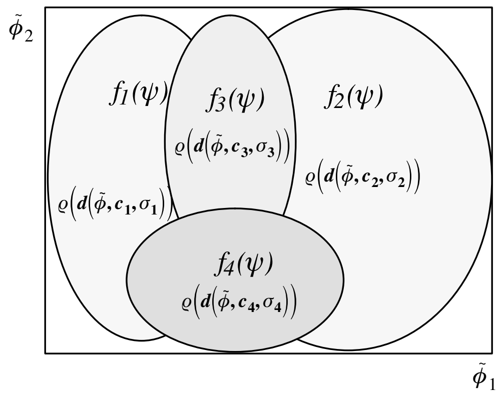


Fig. 3. Local Model Operating Regimes. Each local model is associated with an operating regime. These regimes overlap, and the gradual decay of 'validity' provides interpolation between models.

powerful local models associated with the basis functions, not requiring as many basis functions as a simple RBF net (e.g. as used in [4]) to achieve the desired accuracy. This improvement is more significant in higher dimensional and dynamic problems.

1.4 Pre-structuring Local Model Nets with *a priori* knowledge

A major advantage of local model nets is that they are not only useful architectures for general learning tasks, but that it is relatively easy to introduce *a priori* knowledge about 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 If the knowledge is available, local models could be physically motivated models, possibly requiring only a subset of their variables to be identified, thus allowing the engineer to easily create *grey-box* models. The most general form of information is the expected dynamic order of the process, and the form of model to be identified (e.g. simple linear ARX models, where θ_i is a local parameter vector corresponding to linearisation about a local nominal operating point). A generalised form outlined later allows 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 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 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.

For a fixed density of basis functions, the total number of local models required rises – as the 'curse of dimensionality' would have us expect – exponentially with the input dimension. This does not rule out the use of local methods, however, as many nonlinear systems in the real world have smooth nonlinearities which can be represented by relatively few local models (this becomes even more relevant 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. Constructive learning algorithms which automatically attempt this are described in section 2.3, but it is still important to use the available knowledge to limit the size of the search space faced by the learning algorithm.

The problems of dimensionality can 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 the necessary *a priori* knowledge, so that the system can be approximated over an additive combination of lower dimensional sub-models (e.g. the theory of additive modelling techniques [5, 6] was developed to support such decompositions).

The Local Model net can thus be generalised to cater for possible redundancy in the operating point for certain operating regimes; some operating regimes may be specified by only a sub-vector of the operating vector. To allow this the argument to each $\rho_i(\cdot)$, which we will now call ϕ_i^{op}, is defined as belonging to a set Φ_i^{op} defined on a subspace of $\mathcal{R}^{n_{\tilde{\phi}}}$, i.e. $\phi_i^{\mathrm{op}} \in \Phi_i^{\mathrm{op}} \subset \mathcal{R}^{n_{\phi_i^{\mathrm{op}}}}$, with $n_{\phi_i^{\mathrm{op}}} \leq n_{\tilde{\phi}}$. The *generalised local model network*, or GLMN, is defined as

$$\hat{y}(t) = \hat{f}(\psi) = \sum_{i=1}^{n_{\mathcal{M}}} \hat{f}_i(\psi)\rho_i(\phi_i^{\mathrm{op}}). \tag{12}$$

Using the above framework, the strong links between Fuzzy membership functions and Basis Functions (see [7, 8]) become even more apparent. *A priori* knowledge of how best to decompose the problem could therefore be expressed as fuzzy rules with accompanying basis/membership functions.

In summary, for many high-dimensional nonlinear dynamic processes, although the system may be globally strongly nonlinearly dependent on the inputs, it may now be possible to use the most important subset of the inputs to partition the input space into subspaces which are more manageable for the automated learning algorithm.

2 Learning in Local Model Nets

2.1 Parameter estimation in Local Model Nets

The problem of parameter estimation for systems which are linear in the parameters is reasonably well understood, with a variety of efficient optimisation algorithms existing to optimise the parameters θ of local models $\hat{f}_i(\cdot)$ in equation (9) to minimise the cost functional $J(\theta, \mathcal{M}, \mathcal{D})$ for a given local model structure $\mathcal{M}$, where $\mathcal{M} = (c, \sigma, n_{\mathcal{M}}, \mathcal{M}_{1..n_{\mathcal{M}}})$ (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^*

$$J^*(\mathcal{M}, \mathcal{D}) = \min_{\theta} J(\theta, \mathcal{M}, \mathcal{D}). \tag{13}$$

The methods described in the literature are usually *global* optimisation methods based on the assumption that all of the parameters θ can be optimised simultaneously with a single linear regression operation,

$$\mathbf{Y} = \Phi\theta \tag{14}$$

where Φ is the design matrix, where the rows are defined by

$$\boldsymbol{\phi}_i = \left[\rho_1(\tilde{\phi}_i)[1\ \psi_{i_1} \ldots \psi_{i_{n_\psi}}] \ldots \rho_{n_{\mathcal{M}}}(\tilde{\phi}_i)[1\ \psi_{i_1} \ldots \psi_{i_{n_\psi}}]\right], \tag{15}$$

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

$$\Phi = \begin{pmatrix} \boldsymbol{\phi}_1 \\ \cdot \\ \cdot \\ \boldsymbol{\phi}_N \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} y_1 \\ \cdot \\ \cdot \\ y_N \end{pmatrix} \tag{16}$$

The Moore-Penrose pseudoinverse of Φ, Φ^+ is then used to estimate the weights,

$$\hat{\theta} = \Phi^+\mathbf{Y} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{Y} \tag{17}$$

but this is not always computationally feasible for larger problems. 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.

The lack of independence in globally trained local models is significant for later use in *Local Controller Networks* (described in section 3.1).

Local learning An alternative to global learning is to locally estimate the parameters of each of the local models independently[1] using the basis functions as local weighting criteria. This results in a set of local estimation criteria for the i-th local model (where $i = 1..n_{\mathcal{M}}$) of

$$J_i(\theta_i) = \frac{1}{N_i} \sum_{k=1}^{N_i} \rho_i(\tilde{\phi}_{i_k})(y_{i_k} - \hat{y}_{i_k})^2, \tag{18}$$

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 θ_i is given by $\hat{\theta}_i = \arg\min J_i(\theta_i)$. In matrix terms the operation is now

$$\hat{\theta}_i = (\Phi_i^T Q_i \Phi_i)^{-1} \Phi_i^T Q_i \mathbf{Y}, \tag{19}$$

where Φ_i is an $N_i \times (n_\psi + 1)$ vector,

$$\Phi_i = \begin{pmatrix} \boldsymbol{\phi}_{i1} \\ \cdot \\ \cdot \\ \boldsymbol{\phi}_{iN_i} \end{pmatrix}, \boldsymbol{\phi}_{ik} = [1 \ \psi_k] \tag{20}$$

where the k refers to the kth example in local training set $\mathcal{D}_i$. 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 $\mathcal{D}_i$,

$$Q_i = \begin{pmatrix} \rho_i(\tilde{\phi}_{i_1}) & 0 & 0 & 0 \\ 0 & \rho_i(\tilde{\phi}_{i_2}) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \rho_i(\tilde{\phi}_{i_{N_i}}) \end{pmatrix}. \tag{21}$$

The local learning method involves the computation of $n_{\mathcal{M}}$ 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.

2.2 Structure Identification in Local Model Nets

The use of existing *a priori* knowledge, as described in Section 1.4, 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 an important but difficult non-convex optimisation problem. For a review of structure identification algorithms see [9] or the paper by Johansen and Foss in this volume.

[1] This assumes that the basis functions achieve a partition of unity.

The goal of the structure identification procedure is to provide a problem-adaptive learning scheme which automatically relates the complexity of the local models, and the density and overlap of basis functions 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 regard 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 (13), over the range of possible model structures leads to the 'super-optimal' cost, using *a priori* knowledge about the process structure $\mathcal{K}_{\mathcal{S}}$,

$$J^{**}(\mathcal{D}, \mathcal{K}_{\mathcal{S}}) = \min_{\mathcal{M}} J^*(\mathcal{M}, \mathcal{D}). \tag{22}$$

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 [10] for details about the trade-off), and this should be reflected in the cost functions $J$and J^* (from equation (13)) in the form of regularisation terms for J and terms which penalise over-parameterisation in the structure functional J^*.

2.3 The constructive approach

The constructive method 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 [11] and [12].

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.

The Multi-Resolution Constructive Algorithm 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 there will be areas of the input space which *cannot* be filled with data.

The first point implies that we should consider the local complexity of the process output when altering 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 third 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.

The MRC algorithm differs from other 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 where the current model differs most from the underlying process, i.e. where the greatest complexity is. The model structure is then developed in these areas. 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 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.[2]
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 constructive procedure is illustrated in Fig. 4 where the model complexity is increased gradually for a two-dimensional function approximation problem.

Complexity detection – where are extra units needed? The 'complexity detection' heuristic used to place new local models was inspired by the *Vector Field Approach to Cluster Analysis* [13]. 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 c_{new} for the new basis function. The centre $c_{new} \in \mathcal{R}^{n_{\tilde{\phi}}}$, and the 'complexity' of the mapping in a windowed area, where $\rho(\cdot)$ is the windowing function[3] around this point is measured using

$$F_{total}(c, \sigma_{win}) = \frac{1}{N} \sum_{i=1}^{N} \rho(d(\tilde{\phi}_i, c, \sigma_{win}))e_i \,, \tag{23}$$

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

$$e_i = |y_i - \hat{y}_i| \,. \tag{24}$$

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

[3] In this work a Gaussian bell was used.

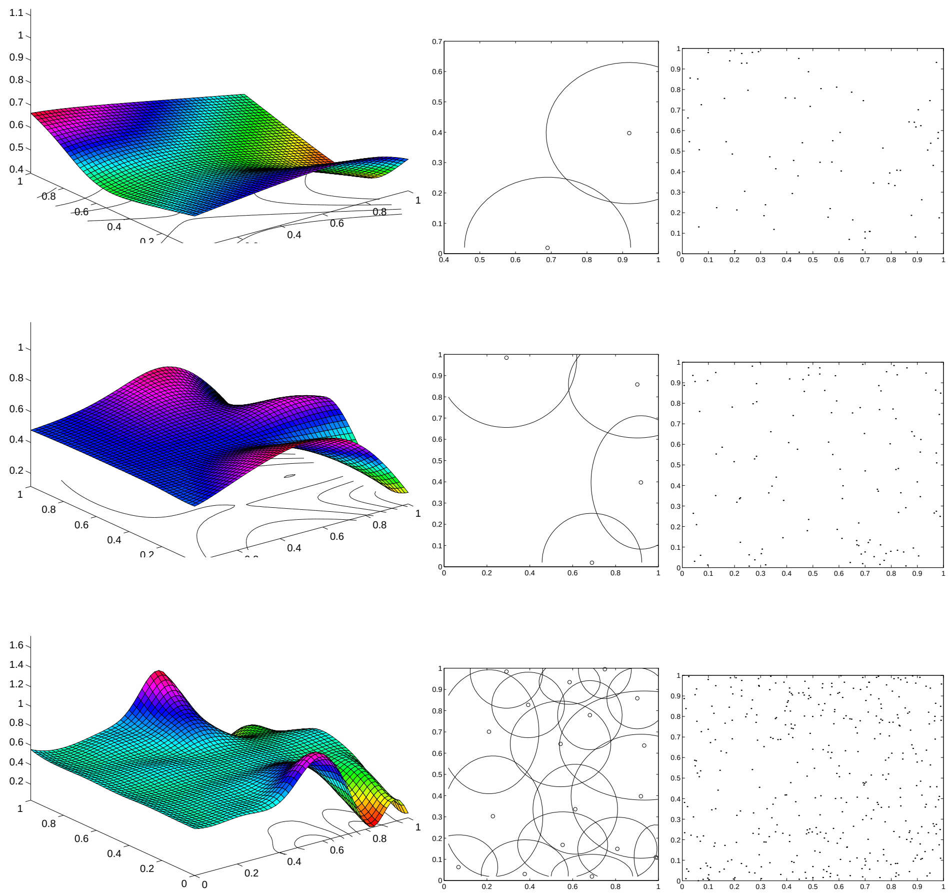


Fig. 4. A gradual approach to constructing a model. The model responses are shown for a series of stages in model development, shown by the contour plots of the basis functions. The right hand column shows the training data used – note that the set is expanded with the model structure.

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. The windowing function operates at different scales σ_{win}, starting off by searching for coarse complexity and refining the search as learning progresses.

As new models must be a certain minimum distance away from previous

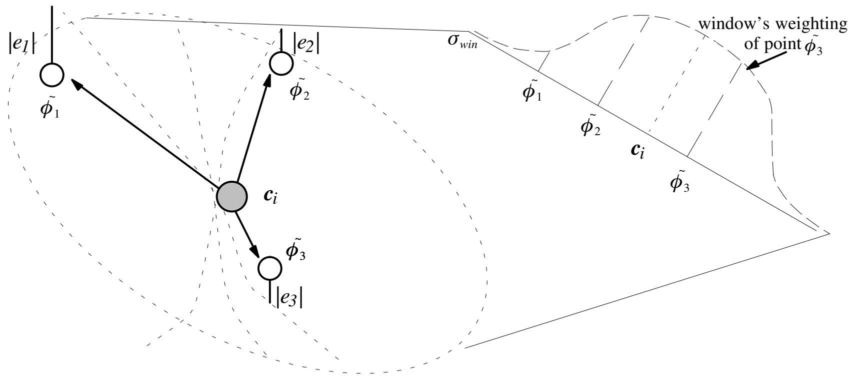


Fig. 5. Windowed Complexity Estimate. The weighting function $\rho(\cdot)$ weights the error measures e_i at points $\tilde{\phi}_i$ around the prototype centre. The window function is usually chosen such that points further from the centre have less effect on the outcome of the complexity estimate.

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 [4].

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 with *active selection* of the training data.

Overlap determination The conventional method for overlap determination is to set the radius σ_j proportional to the average distance of the k nearest neighbours from the centre c_j. In many cases this will be unsatisfactory, as the resulting level of overlap with the neighbouring basis functions would vary greatly[5]. One alternative method is to observe that a covariance matrix $\boldsymbol{\sigma}_i$, can

[4] 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).

[5] 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 [9]

be estimated using a heuristic which calculates the 'covariance' of a selected set of centres c_n surrounding[6] the chosen centre 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:

$$\boldsymbol{\sigma}_i \propto E_n \left[(c_n - c_i)(c_n - c_i)^T \right], \tag{25}$$

where E_n is the expected value over the set of chosen neighbouring centres $\mathbf{c}_n$.

Local model structure selection & 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. Earlier 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.*

Local Model Structures 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[7]. Such methods 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. The most basic 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 $\rho_{total_l} = \sum_{i=1}^{N} \rho_l(\tilde{\phi}_i)$ 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.

A further method is to use *pruning techniques*, which merge neighbouring local models if their parameters are similar enough[8], have also been successfully

it was shown that the condition of the design matrix is highly dependent on the level of overlap between basis functions.

[6] It is important to choose the neighbours in as wide a range of directions as possible, see [9] for details.

[7] The resulting local model net will be a heterogeneous structure, 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, a variety of local learning methods can be used.

[8] The distance between the local model parameter vectors is then calculated $\delta_{ik} = |\theta_i - \theta_k|$, 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.

applied to simplify the networks. These rely on the use of local learning techniques, as described earlier, to ensure that the parameters have a strictly local interpretation.

3 Control Based on Local and Time-varying Models

We turn our attention now to control methods based upon the local model network. In Section 3.1 a direct analogue of the local model network, the *local controller network*, is considered. In Section 3.2 on-line adaptive control is considered. This is based upon a linear time-varying interpretation of the local model network, coupled with control redesign and perhaps on-line parameter update. Further details of the control designs discussed here can be found in Żbikowski *et al* [14].

3.1 Local Controller Networks

We postulate a general nonlinear controller for the system (1):

$$u(t) = C(u(t-1), \ldots u(t-n_u^C), y(t), \ldots y(t-n_y^C), r(t), \ldots r(t-n_r)). \quad (26)$$

The signal $r(t)$ is a reference value while C is the nonlinear function characterising the controller. The controller can be written more compactly as

$$u(t) = C(\psi^C(t)), \quad (27)$$

where

$$\psi^C(t) = [u(t-1), \ldots u(t-n_u^C), y(t), \ldots y(t-n_y^C), r(t), \ldots r(t-n_r)]^T. \quad (28)$$

Note that when the controller is designed on the basis of a model which is valid around some operating point the signals in the preceding equations represent deviations from the operating point.

By directly exploiting the structure of the local model network a *local controller network* (LCN) can be defined. For each operating regime Φ_i the system is characterised by the local model $\hat{f}_i$, to a degree given by the validity function ρ_i. It is natural to exploit the given partition of the operating set and for each operating regime to design a local controller C_i whose validity is also given by the same ρ_i. The overall controller is then defined analogously to the local model network by using the validity functions as smooth interpolators. The local controller network is described by

$$u(t) = C(\psi^C(t)) = \sum_{i=1}^{n_{\mathcal{M}}} C_i(\psi^C(t-1))\rho_i(\phi_i^{\text{op}}) \quad (29)$$

and this should be compared to equation (12). For linear local models of the form $\hat{f}_i(\psi(t-1)) = \psi^T(t-1)\theta_i$, linear local controllers are natural;

$$C_i(\psi^C(t)) = {\psi^C}^T \theta_i^C. \quad (30)$$

The θ_i^C are the local controller parameters which can be directly obtained from some linear control design algorithm mapping the local model parameters;

$$\theta_i^C = \Xi(\theta_i, J_i^C). \tag{31}$$

Here, Ξ denotes the mapping of the linear control design algorithm, while J_i^C is a set of local control design specifications.

The LCN can be rearranged by combining equations (29) and (30) to give

$$u(t) = C(\psi^C(t)) = {\psi^C}^T \sum_{i=1}^{n_{\mathcal{M}}} \theta_i^C \rho_i(\phi_i^{\mathrm{op}}). \tag{32}$$

This can be interpreted as a linear controller with operating-point-dependent parameters:

$$u(t) = {\psi^C}^T \tilde{\theta}^C(\tilde{\phi}(t)), \tag{33}$$

with $\tilde{\theta}^C(\tilde{\phi}(t)) = \sum_{i=1}^{n_{\mathcal{M}}} \theta_i^C \rho_i(\phi_i^{\mathrm{op}})$. Here the controller parameters $\tilde{\theta}^C$ are obtained by interpolating the local controller parameters θ_i^C using the validity functions $\rho_i(\phi_i^{\mathrm{op}})$. This is clearly very similar to gain scheduling control [15, 16], whereby the LCN structure provides a more general framework and a systematic way to do the actual scheduling. Further details of the link to gain scheduling are discussed in [14]. Other special cases of the LCN structure include the modular control architecture of Jacobs and Jordan [17], and the heterogeneous control approach of Kuipers and Åström [18]. As mentioned earlier, there is also a direct link with fuzzy control and this is discussed in depth in [19, 20, 21, 22].

In the case of linear local models it is natural to propose a network of linear local controllers. A local two-degrees-of-freedom control structure for each local model is defined by

$$u(t) = \frac{1}{H(q^{-1})}(S(q^{-1})r(t) - G(q^{-1})y(t)), \tag{34}$$

where the polynomials G, H and S have the form

$$S(q^{-1}) = s_0 + s_1 q^{-1} + \ldots + s_{n_r} q^{-n_r}, \tag{35}$$

$$G(q^{-1}) = g_0 + g_1 q^{-1} + \ldots + g_{n_y^C} q^{-n_y^C}, \tag{36}$$

$$H(q^{-1}) = 1 + h_1 q^{-1} + \ldots + h_{n_u^C} q^{-n_u^C}. \tag{37}$$

The controller (34) can be expressed as

$$u(t) = -H'(q^{-1})u(t) - G(q^{-1})y(t) + S(q^{-1})r(t) \tag{38}$$

where $H'(q^{-1}) = H(q^{-1}) - 1$. This structure can furthermore be expressed in the vector form of equation (30) as

$$u(t) = {\psi^C}^T \theta^C \tag{39}$$

where the controller parameter vector θ^C has the form

$$\theta^C = [-h_1, \ldots - h_{n_u^C}, -g_0, \ldots - g_{n_y^C}, s_0, \ldots s_{n_r}]^T. \tag{40}$$

When we have a local model network representation of the nonlinear system this controller structure is repeated $n_{\mathcal{M}}$ times and there exist $n_{\mathcal{M}}$ local controller parameter vectors θ_i^C. Each local controller is designed for the local linear model. The model parameter vectors θ_i can be decoded to give a local polynomial representation A_i, B_i, C_i of the form (7). The above control structure then defines a notional characteristic equation for each model/controller pair. This has the form

$$A_i(q^{-1})H_i(q^{-1}) + q^{-k}B_i(q^{-1})G_i(q^{-1}) = T_i(q^{-1}). \tag{41}$$

The local controller polynomials G_i, H_i whose coefficients form part of the θ_i^C are solutions to this linear equation. The desired characteristic polynomial $T_i(q^{-1})$ can be set in a number of ways, e.g. by directly choosing closed-loop poles or by minimising a cost function (T_i are then given by spectral factorisation). The remaining controller polynomials S_i (which act on the reference signal r) can be designed to achieve desirable command tracking properties.

3.2 Adaptive Control Interpretation

An alternative approach is to interpret a nonlinear local model network as a linear time-varying system [23, 24]. Standard linear control design, of the form outlined above, can then be repeatedly applied on-line to the time varying linear model. This can also be done in conjunction with on-line parameter estimation for update of the LMN parameters. This scheme then becomes strongly reminiscent of standard self-tuning control. To see how this works consider the LMN based upon linear local models:

$$y(t) = \psi^T(t-1)\sum_{i=1}^{n_{\mathcal{M}}} \theta_i \rho_i(\phi_i^{\mathrm{op}}) + e(t) \tag{42}$$

with

$$\psi(t-1) = [y(t-1), \ldots y(t-n_y), u(t-k), \ldots u(t-k-n_u)]^T. \tag{43}$$

The system can be written in a linear form with time-varying parameters,

$$y(t) = \psi^T(t-1)\tilde{\theta}(\tilde{\phi}(t)) + e(t). \tag{44}$$

The time-varying parameter vector $\tilde{\theta}$ is defined as

$$\tilde{\theta} = [a_1(\tilde{\phi}(t)), \ldots a_{n_y}(\tilde{\phi}(t)), b_0(\tilde{\phi}(t)), \ldots b_{n_u}(\tilde{\phi}(t))]^T. \tag{45}$$

The individual coefficients have the specific parameterisation

$$a_1(\tilde{\phi}(t)) = \sum_{i=1}^{n_{\mathcal{M}}} \theta_{i,1}\rho_i(\phi_i^{\mathrm{op}}), \tag{46}$$

$$\vdots$$

$$a_{n_y}(\tilde{\phi}(t)) = \sum_{i=1}^{n_{\mathcal{M}}} \theta_{i,n_y}\rho_i(\phi_i^{\mathrm{op}}), \tag{47}$$

$$b_0(\tilde{\phi}(t)) = \sum_{i=1}^{n_{\mathcal{M}}} \theta_{i,n_y+1}\rho_i(\phi_i^{\mathrm{op}}), \tag{48}$$

$$\vdots$$

$$b_{n_u}(\tilde{\phi}(t)) = \sum_{i=1}^{n_{\mathcal{M}}} \theta_{i,n_y+n_u+1}\rho_i(\phi_i^{\mathrm{op}}). \tag{49}$$

When the time-delay is $k = 1$ the predictive form of the system equation can be expressed as

$$\begin{aligned} y(t) = {} & a_1(\tilde{\phi}(t))y(t-1) + \ldots + a_{n_y}(\tilde{\phi}(t))y(t-n_y) + \\ & b_0(\tilde{\phi}(t))u(t-1) + \ldots + b_{n_u}(\tilde{\phi}(t))u(t-1-n_u) + e(t). \end{aligned} \tag{50}$$

Note that the procedure can be easily generalised to arbitrary k, but for ease of notation we restrict attention to the case $k = 1$. From this model the time-varying A and B polynomials of a linear ARX-type model can be identified:

$$\begin{aligned} A(q^{-1}) &= 1 - a_1(\tilde{\phi}(t))q^{-1} - \ldots - a_{n_y}(\tilde{\phi}(t))q^{-n_y}, \\ B(q^{-1}) &= b_0(\tilde{\phi}(t)) + \ldots + b_{n_u}(\tilde{\phi}(t))q^{-n_u}. \end{aligned} \tag{51}$$

These A and B polynomials then form the basis for redesign.

3.3 Example

We now illustrate the LCN of Section 3.1 with a simple example. The problem of controlling the lateral deviation of a car on a straight road is considered [25, 26]. The model is given by:

$$\begin{aligned} \dot{\beta} &= \left(-2\,\frac{k}{mv}\right)\,\beta \;+\; \left(\frac{v}{a} - \frac{k}{mv}\right)\,\lambda \\ \dot{y} &= v(\psi \;+\; \beta) \\ \dot{\psi} &= \left(\frac{v}{a}\right)\,\lambda \\ \dot{\lambda} &= u \end{aligned}$$

The variables in this model are:

- y – lane centre offset (to be controlled),
- ψ – vehicle yaw angle and road course angle difference,
- β – sideslip angle,
- λ – steering angle (control input),

– v – vehicle speed,

and the constants are: $m = 5800\ kg$ - vehicle mass, $a = 4.25\ m$ - wheel base and $k = 150\ kN/rad$ - lateral friction.

The task is to regulate the offset to zero by appropriate manipulation of the steering angle. It can be seen from the above equations that the model (the transfer-function between steering angle and offset) is a linear time-varying system; the model parameters depend on the vehicle speed v. This is therefore a prime candidate for control with the LCN structure using speed as the scheduling variable. Note that this system is unstable (3 poles at $s = 0$).

First, we illustrate the difficulty of attempting to control this system with a fixed linear controller. A linear controller was designed for an operating speed of 20m/s. A simple pole-assignment regulator based on a pair of dominant closed-loop poles giving a rise-time of 5s was designed. The response of this controller at a speed of 20m/s to initial offsets of 1m and 0.4m is shown in Fig. 6. With this

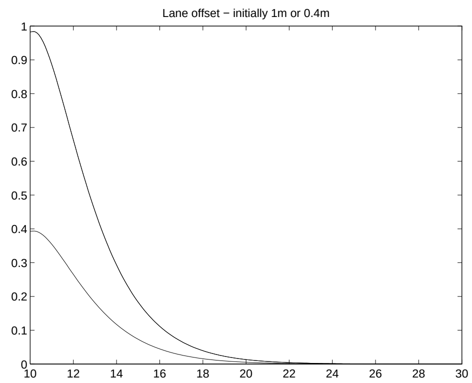


Fig. 6. Linear controller, $v = 20$m/s

same controller the car was controlled at a speed of 15m/s. Although reasonable offset correction was still achieved, the control input was unacceptably active. This is illustrated in Fig. 7 where the steering angle for an initial offset of 1m is plotted. Note that for speeds lower than 15m/s or higher than 24m/s the controller became unstable.

A local controller network was then designed based upon linear models for a number of operating speeds. For each operating point the control specification was the same, i.e. the system should have a rise-time of 5s in reponse to step-like offsets. The controller response for this LCN system was consistent over all speeds. An example is illustrated in Fig. 8(a) for $v = 10$m/s (the single linear controller designed at 20m/s was unstable for this condition). The system responds to the initial offset with the desired speed, and the response is visually indistinguishable from that shown in Fig. 6 even though the vehicle speed is very much lower.

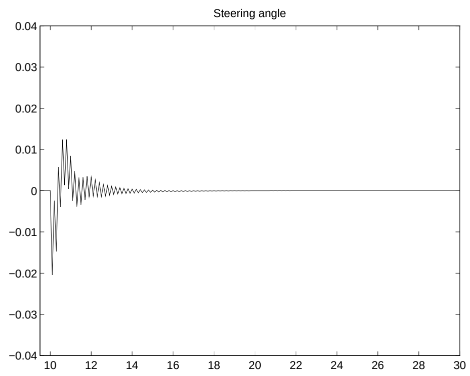
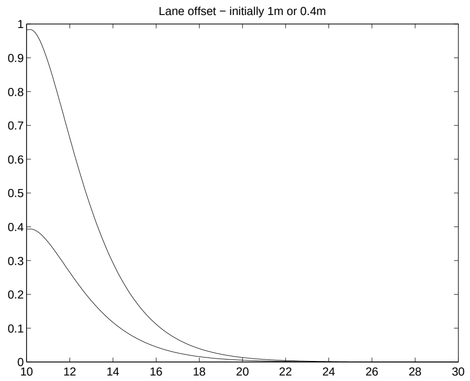
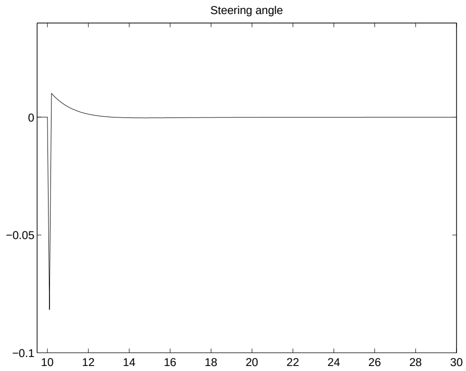


Fig. 7. Linear controller, $v = 15\mathrm{m/s}$



(a) Lane offset



(b) Steering angle

Fig. 8. LCN controller, $v = 10\mathrm{m/s}$

4 Conclusions

Local Model Nets Local Model networks are practical learning architectures for use in real problems. The structure is general enough to be applicable to a wide range of processes, while being relatively simple to understand, and lending itself well to the interpretation of results and introduction of *a priori* knowledge to simplify training. The use of constructive learning algorithms, as described in this paper, provides the user a way of automatically creating a model structure capable, given the training data, of representing the target process.

Constructive Learning Algorithms 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 further important point is that *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.

Local Controller Networks Once the *Local Model Net (LMN)* structure has been estimated during learning it is relatively straightforward to produce a matching *Local Controller Net (LCN)* using standard linear control design methods. The method requires, however, that the linear local model be a local approximation of the system in question, so a local learning method should be used during construction of the Local Model Net. This forces the algorithm to extend the model structure so that the local models can become independent local approximations of the target system, rather than the smaller model structures produced by global learning which cannot be interpreted individually.

The results were applied to a simple simulation of a car steering problem to illustrate the easy applicability of the ideas, but it should be remembered that the problems associated with conventional gain scheduling [15] are also relevant to LCNs.

References

1. I. J. Leontaritis and S. A. Billings, "Input-output parametric models for non-linear systems," *Int. J. Control*, vol. 41, no. 2, pp. 303–344, 1985.
2. H. W. Werntges, "Partitions of unity improve neural function approximation," in *Proc. IEEE Int. Conf. Neural Networks*, (San Francisco, CA), pp. 914–918, 1993. Vol. 2.
3. R. Shorten and R. Murray-Smith, "On Normalising Basis Function networks," in *4th Irish Neural Networks Conf., Univ. College Dublin*, Sept. 1994.
4. J. Moody and C. Darken, "Fast-learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1989.
5. T. J. Hastie and R. J. Tibshirani, *Generalized Additive Models*. Monographs on Statistics and Applied Probability 43, London: Chapman and Hall, 1990.
6. J. H. Friedman, "Multivariate Adaptive Regression Splines," *Annals of Statistics*, vol. 19, pp. 1–141, 1991.
7. C. Harris, C. G. Moore, and M. Brown, *Intelligent Control: Aspects of Fuzzy Logic and Neural Nets*. World Scientific, 1993.
8. M. Brown and C. Harris, *Neurofuzzy Adaptive Modelling and Control*. Hemel-Hempstead, UK: Prentice Hall, 1994.

9. R. Murray-Smith, *A Local Model Network Approach to Nonlinear Modelling.* Ph.D. Thesis, Department of Computer Science, University of Strathclyde, Glasgow, Scotland, Nov. 1994. E-mail:murray@DBresearch-berlin.de.
10. S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the Bias/Variance Dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.
11. R. Murray-Smith and H. Gollee, "A constructive learning algorithm for local model networks," in *Proc. IEEE Workshop on Computer-intensive methods in control and signal processing, Prague, Czech Republic*, pp. 21–29, 1994. E-mail:murray@DBresearch-berlin.de.
12. R. Murray-Smith, "A Fractal Radial Basis Function network for modelling," in *Inter. Conf. on Automation, Robotics and Computer Vision, Singapore*, vol. 1, pp. NW–2.6.1–NW–2.6.5, 1992. E-mail:murray@DBresearch-berlin.de.
13. H. C. Andrews, *Introduction to Mathematical Techniques in Pattern Recognition.* Robert E. Krieger, 1983.
14. R. Żbikowski, K. J. Hunt, A. Dzieliński, R. Murray-Smith, and P. J. Gawthrop, "A review of advances in neural adaptive control systems," Technical Report of the ESPRIT NACT Project TP-1, Glasgow University and Daimler-Benz Research, 1994.
15. J. S. Shamma and M. Athans, "Gain scheduling: Potential hazards and possible remedies," *IEEE Control Systems Magazine*, pp. 101–107, June 1992.
16. W. J. Rugh, "Analytical framework for gain scheduling," *IEEE Control Systems Magazine*, vol. 11, pp. 79–84, 1991.
17. R. A. Jacobs and M. I. Jordan, "Learning piecewise control strategies in a modular neural network," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 337–345, 1993.
18. B. Kuipers and K. Åström, "The composition and validation of heterogeneous control laws," *Automatica*, vol. 30, no. 2, pp. 233–249, 1994.
19. K. J. Hunt, R. Haas, and M. Brown, "On the functional equivalence of fuzzy inference systems and spline-based networks," *International Journal of Neural Systems*, 1995. To appear - March issue.
20. M. Brown and C. Harris, "A nonlinear adaptive controller: A comparison between fuzzy logic control and neurocontrol," *IMA J. Math. Control and Info.*, vol. 8, no. 3, pp. 239–265, 1991.
21. M. Brown and C. Harris, *Neurofuzzy Adaptive Modelling and Control.* Hemel-Hempstead, UK: Prentice Hall, 1994.
22. T. A. Johansen, "Fuzzy model based control: stability, robustness, and performance issues," *IEEE Trans. on Fuzzy Systems*, vol. 2, no. 3, pp. 221–233, 1994.
23. T. A. Johansen, "Adaptive control of MIMO non-linear systems using local ARX models and interpolation," in *IFAC ADCHEM 94*, (Kyoto, Japan), May 1994.
24. H. Wang, M. Brown, and C. Harris, "Modelling and control of nonlinear, operating point dependent systems via associative memory networks," *J. of Dynamics and Control*, accepted for publication 1994.
25. J. Ackermann, *Robuste Regelung.* Berlin: Springer-Verlag, 1993.
26. K. Mecklenburg, T. Hrycej, U. Franke, and H. Fritz, "Neural control of autonomous vehicles," in *Proc. IEEE Vehicular Technology Conference, Denver, USA*, 1992.