Chapter 3

Aspects of Local Model Networks

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

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

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

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

3.1 Local Learning vs. Global Learning

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

3.1.1 Problems with global optimisation methods

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

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

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

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

Ill-conditioning in Local Model Networks

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

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

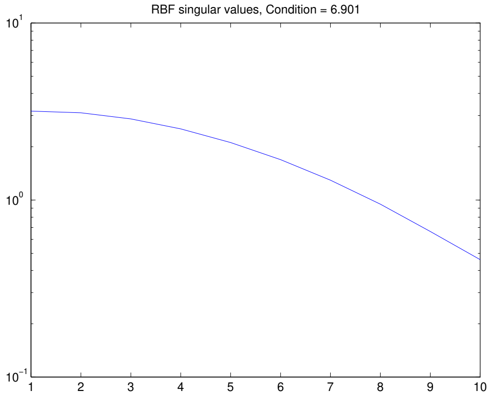
A one-dimensional example is used to illustrate the relationship between the properties of the networks and the associated singular values for a variety of local model nets. The arbitrarily chosen nonlinear function is

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

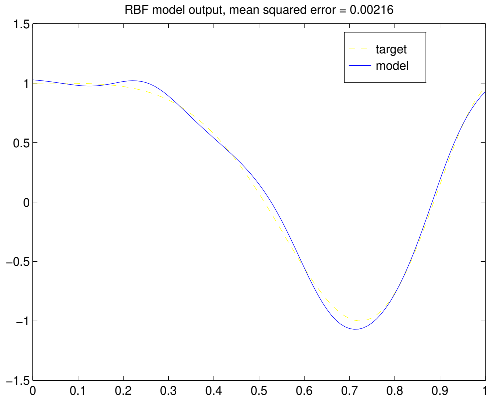
where the additive noise term $\varepsilon(x)$ is Gaussian with a varying standard deviation of $\sigma(x) = 0.4\exp\left(-\left|x - \frac{1}{2}\right| 4.6\right)$. The examples plotted in Figure 3.1 used 401 training examples (see Figure 3.5 on page 63 for a plot of the training data).

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

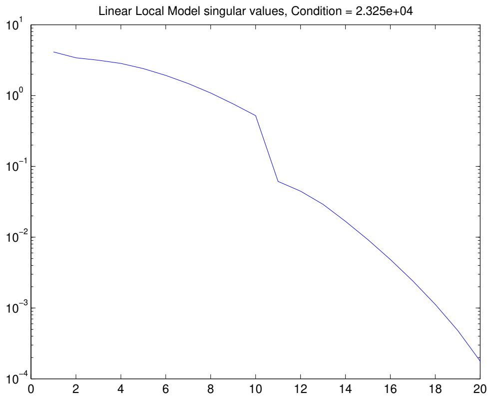
models $\psi_{dev} = \frac{\psi - c_i}{\sigma_i}$, so that the regression bases are in the same numerical range, but there is less correlation between bases.



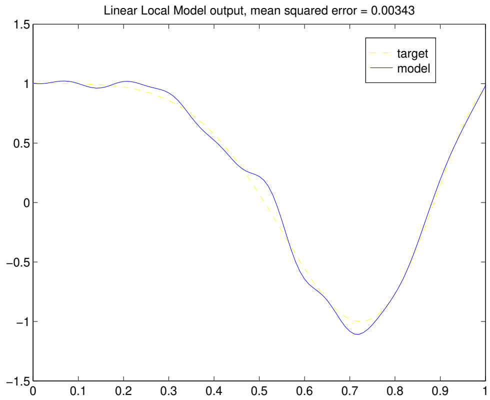
(a) RBF Singular Values



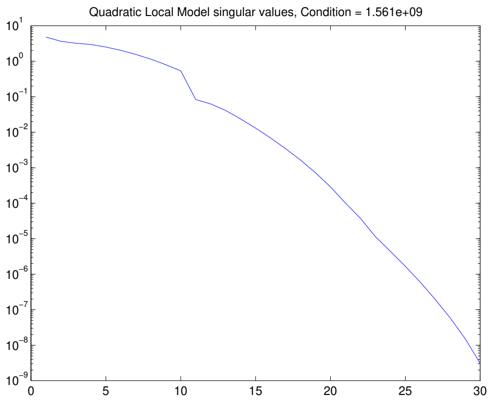
(b) RBF Approximation



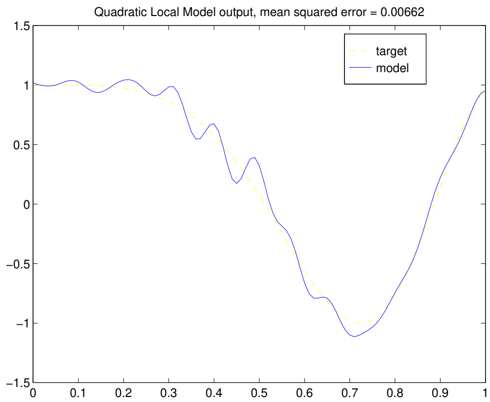
(c) Linear Local Model Singular Values



(d) Linear Local Model Approximation

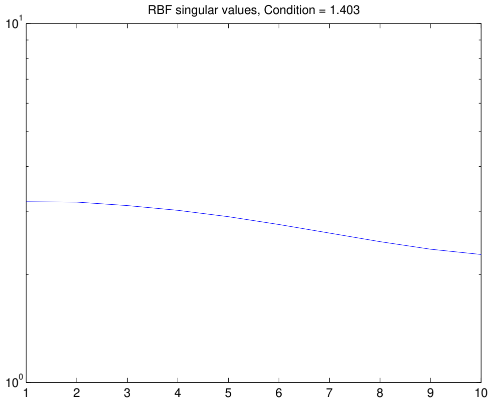


(e) Quadratic Local Model Singular Values

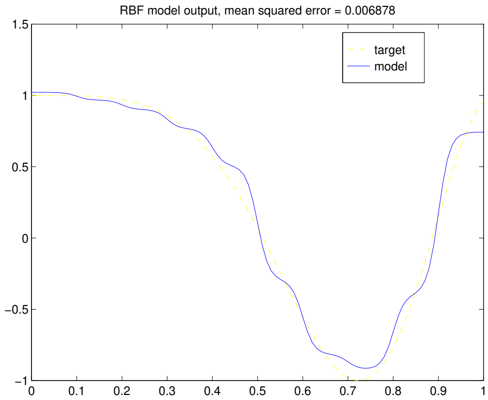


(f) Quadratic Local Model Approximation

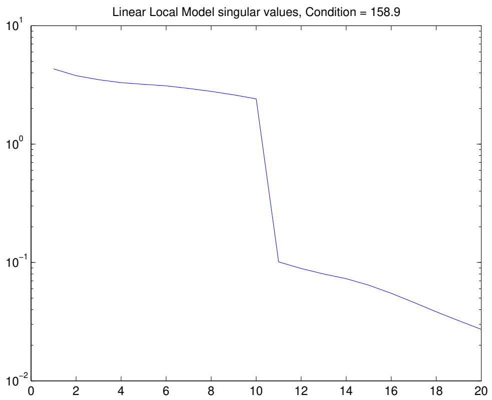
Figure 3.1: A set of 10 basis functions is fixed and used to represent the system using constant, linear and quadratic local models. The singular values for piecewise constant, linear and quadratic models are plotted. The drops in the singular values at 11 in the linear local model case, and at 11 and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



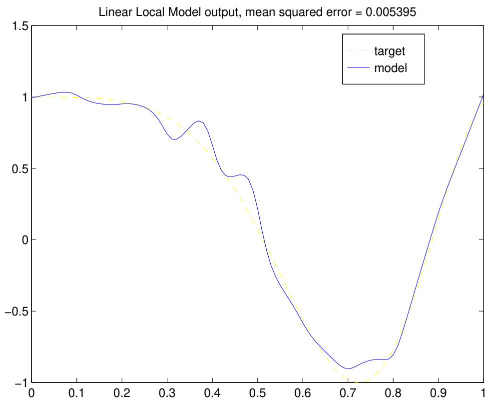
(a) RBF Singular Values



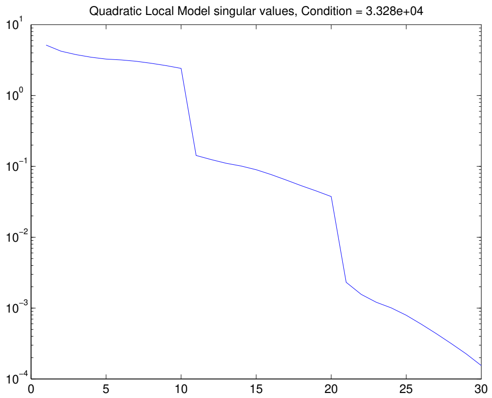
(b) RBF Approximation



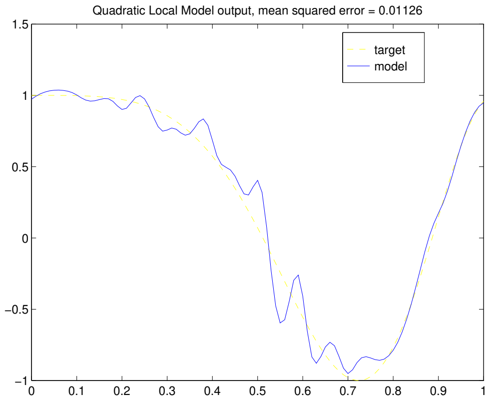
(c) Linear Local Model Singular Values



(d) Linear Local Model Approximation



(e) Quadratic Local Model Singular Values

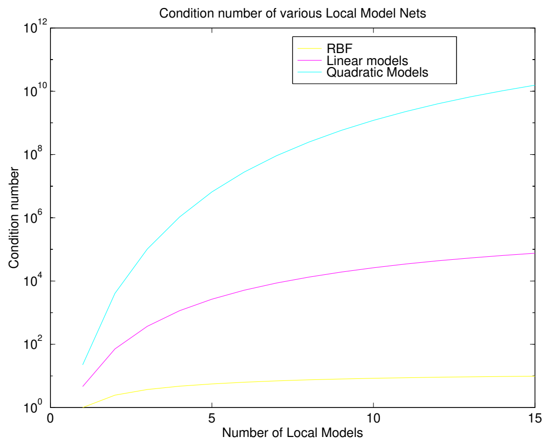


(f) Quadratic Local Model Approximation

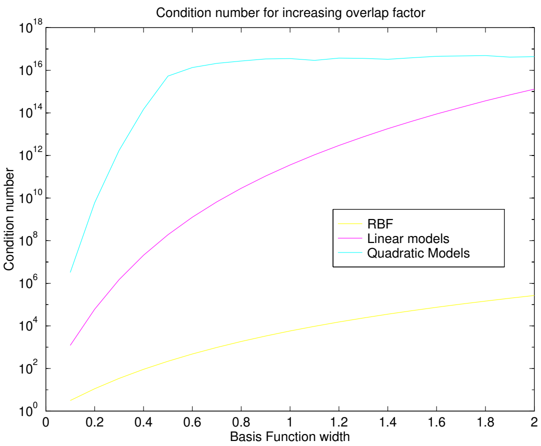
Figure 3.2: The same case as in Figure 3.1, but with basis functions half the size. Notice the sharp drops in the singular values at 11 in the linear local model case, and at 11 and 22 in the quadratic model case. These indicate that the basis functions associated with the higher order inputs are more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

Effect of basis function overlap factor on condition

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



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



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

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

3.1.2 Local learning

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

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

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

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

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

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

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

Local learning with weighted least squares

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

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

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

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

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

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

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

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

$$\mathbf{\Phi}_i = \begin{pmatrix} \boldsymbol{\phi}_{i\,1} \\ \cdot \\ \cdot \\ \boldsymbol{\phi}_{i\,N_i} \end{pmatrix}, \tag{3.7}$$

where the regression variables are:

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

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

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

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

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

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

so the local learning cost function is

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

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

The global least squares cost function

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

so

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1.3 Global vs. local SVD for computational effort

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

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

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

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

and

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

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

3.1.4 Local learning experiments

Approximation of a 1-D noisy function

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

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

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

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

Robustness of locally learned solutions

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

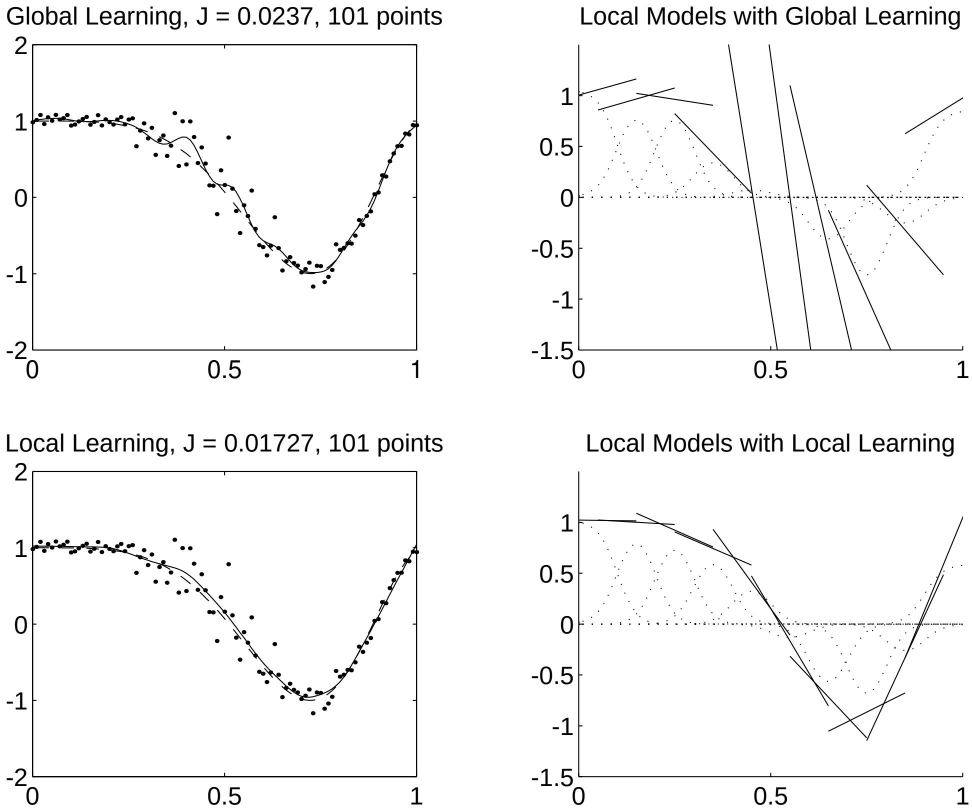


Figure 3.4: Experimental comparison of Global and Local Learning for 101 training points. The left hand side shows the target function, the noisy training data and the trained network's response. The cost functional J in the figure titles, is the same as equation (2.21) where the weighting function $\alpha(\mathbf{x})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

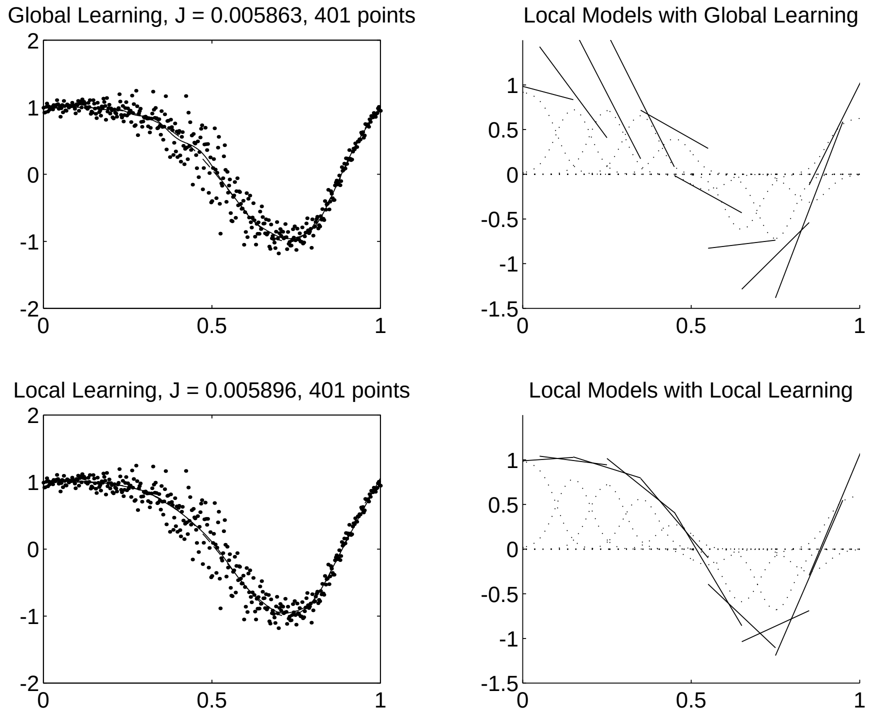


Figure 3.5: Local vs. Global estimation – continuation of Figure 3.4 for 401 training points.

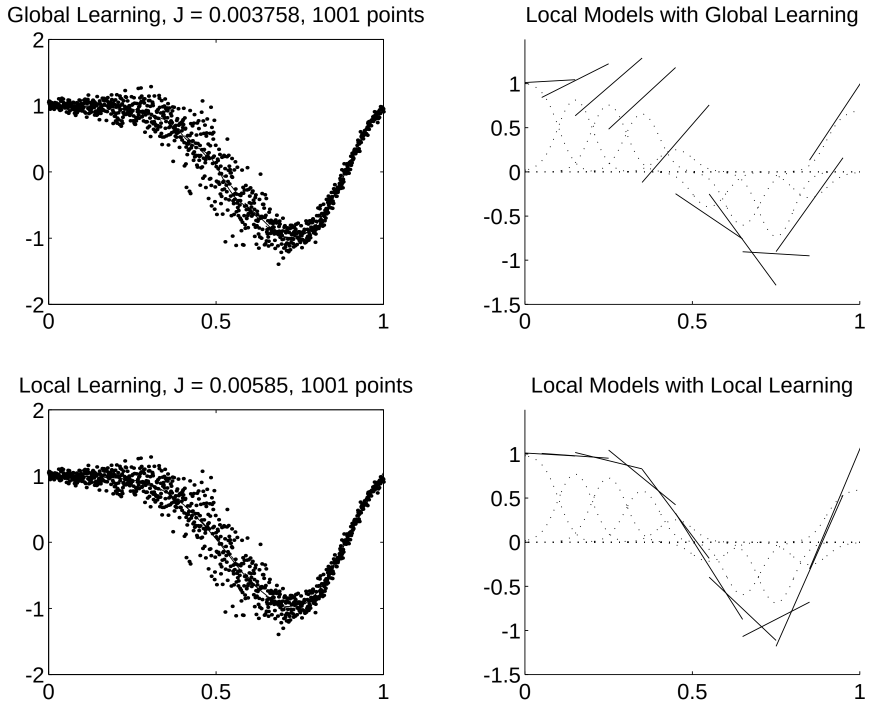


Figure 3.6: Local vs. Global estimation – continuation of Figure 3.4 for 1001 training points.

3.1.5 Training heterogeneous local model nets

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

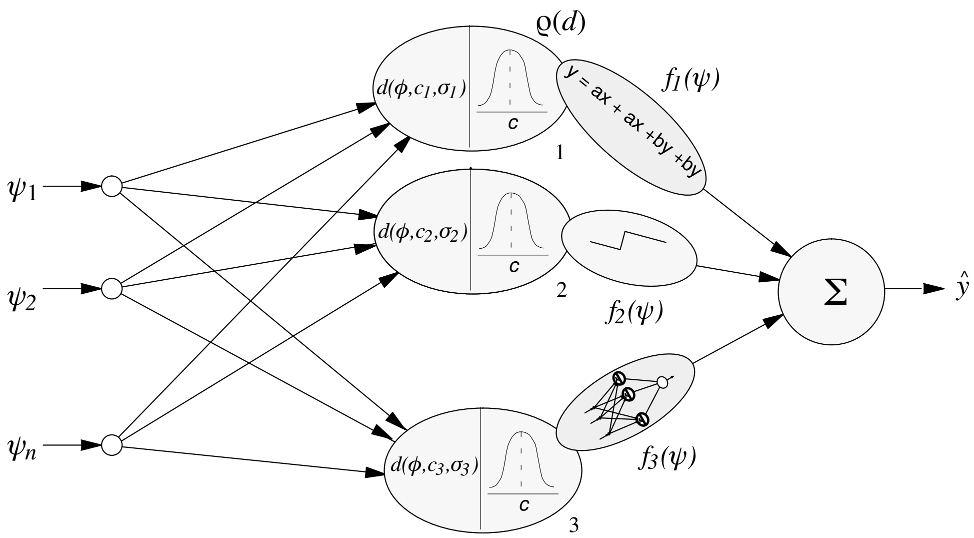


Figure 3.7: Heterogeneous local model network with multi-algorithm optimisation

3.2 Estimating the Confidence in a Trained Network

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

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

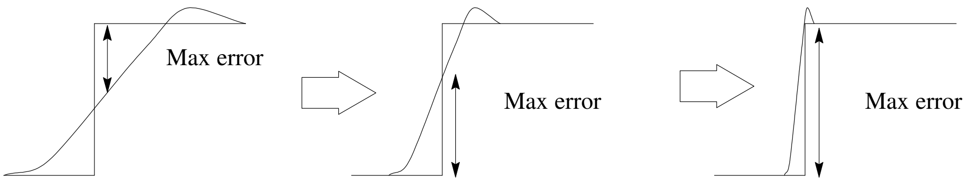


Figure 3.8: As the model improves its average performance, the worst error can increase!

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

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

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

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

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

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

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

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

3.2.1 Local confidence measures

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

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

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

Examples of error statistics

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

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

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

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

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

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

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

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

is too conservative.

Illustrative example

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

3.2.2 Detecting extrapolation

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

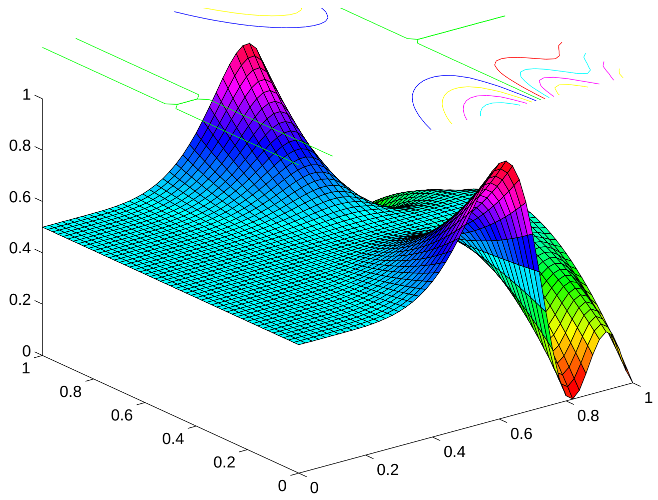


Figure 3.9: Test function. z is vertical axis. x and y are right and left axes respectively

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

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

3.2.3 Estimating covariance of weight estimates from the residuals

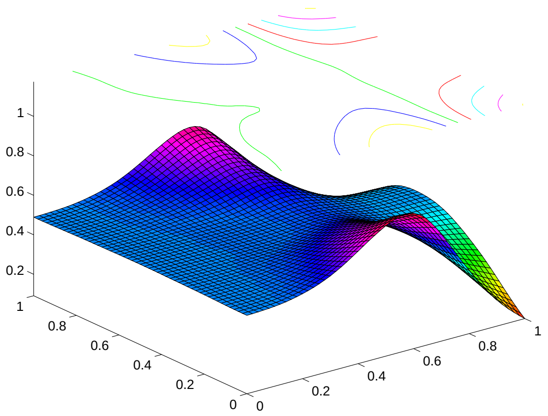
If the local models being used are based on underlying physical structures, it is often interesting to examine the parameters after training in order to interpret the physical relevance of the model. It is therefore useful to know the covariance of a given parameter estimate. The covariance estimation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

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

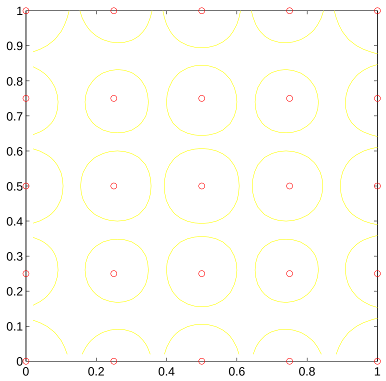
such that the real output vector $\mathbf{Y}$ will be defined by the model's parameters ($\boldsymbol{\theta}_i$) and its residuals ($\mathbf{D}$):

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

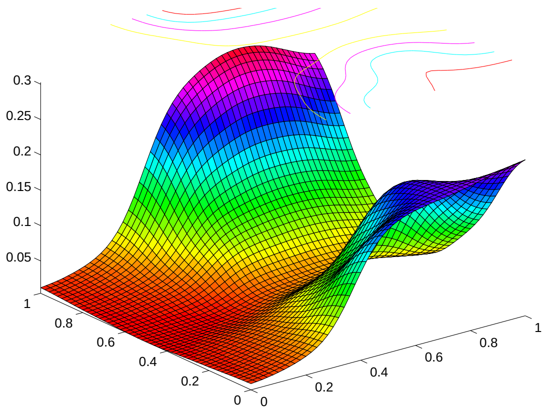
[5] This can be very important for model-based predictive control, where the optimisation routines can produce very 'unnatural' inputs, lying outside the sampled data range, so that the model behaves poorly for the purpose intended of the model



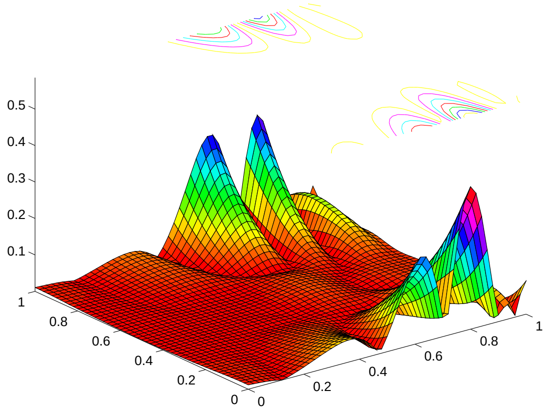
(a) Model Output



(b) Basis Functions



(c) Locally Estimated Maximum Error



(d) Measured Absolute Error

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

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

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

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

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

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

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

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

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

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

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

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

3.3 The Effect of Normalisation of the Basis Functions

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

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

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

3.3.1 Side-effects of normalisation for the basis functions

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

Change of shape of basis function

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

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

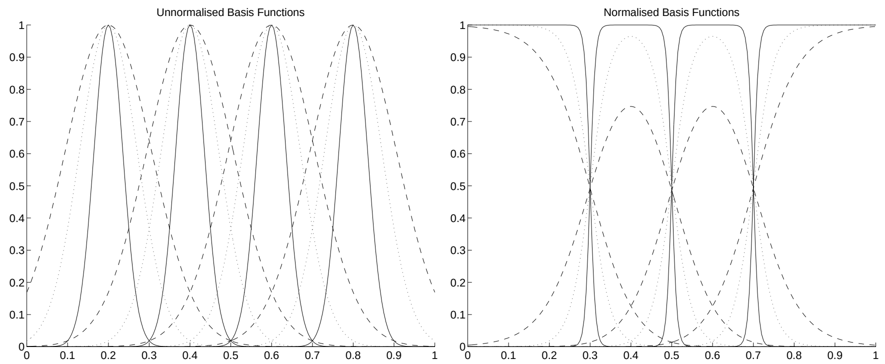
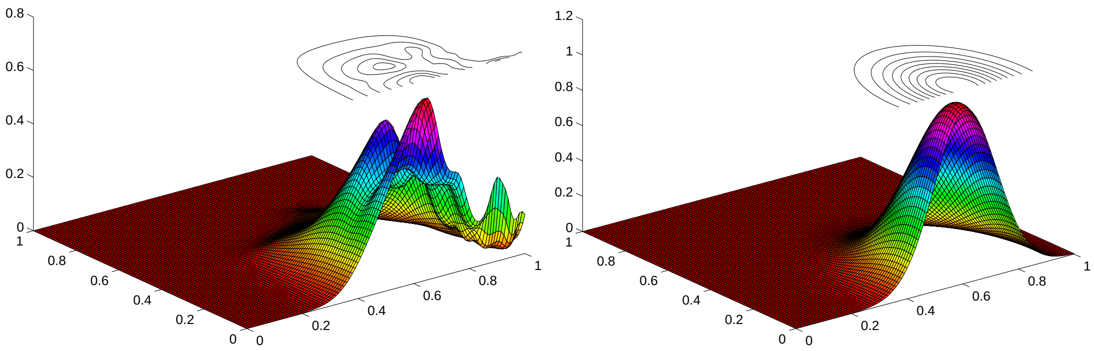


Figure 3.11: Change in shape due to normalisation. As the original functions become wider the normalised basis functions become less square and their maxima are reduced. Edge basis functions tend to unity as they move to the limits of their support.



(a) Normalised Basis Function

(b) Unnormalised Basis Function

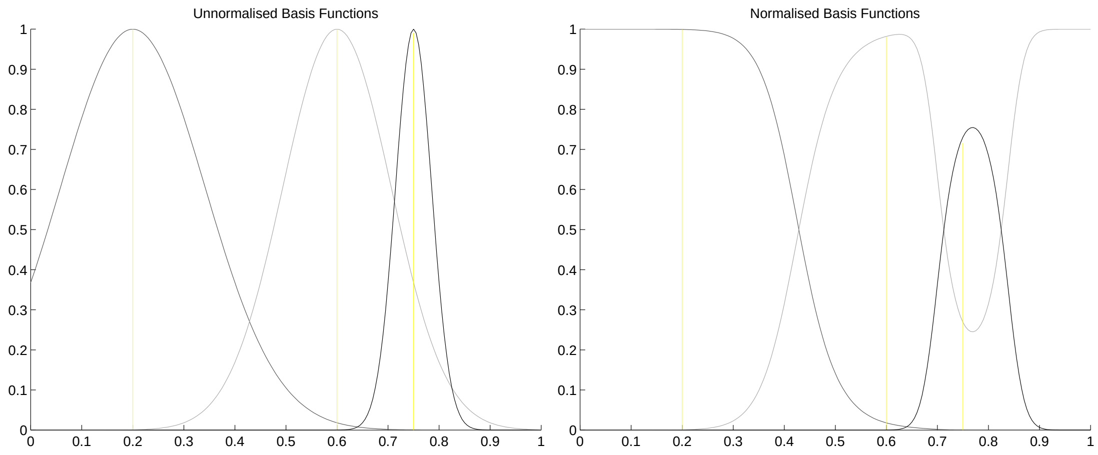
Figure 3.12: Effect of Normalisation on Basis Function Shape. The normalised basis function has a far more complex surface than before, with many local maxima.

Covering of the input-space

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

Irregular networks: reactivation and shift in maxima

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



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

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

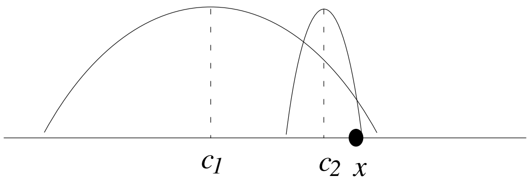


Figure 3.14: Reactivation example. The point in the input space x where the basis function reactivates can be determined from the units' centres (c_1 and c_2, where c_1 is furthest from the input x) and their widths (σ_1 and σ_2).

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

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

For a Euclidean distance metric,

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

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

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

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

Effects of normalisation on multi-dimensional problems

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

3.3.2 Effect of normalisation on optimal network parameters

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

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

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

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

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

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

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

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

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

which can be written

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

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

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

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

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

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

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

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

Example 1: Modelling a pulse function

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

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

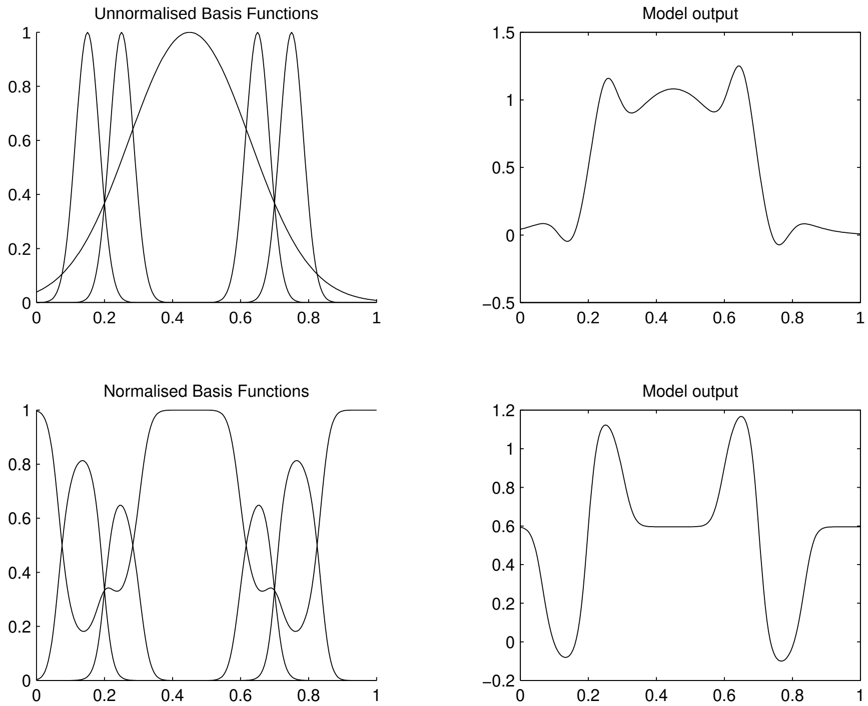
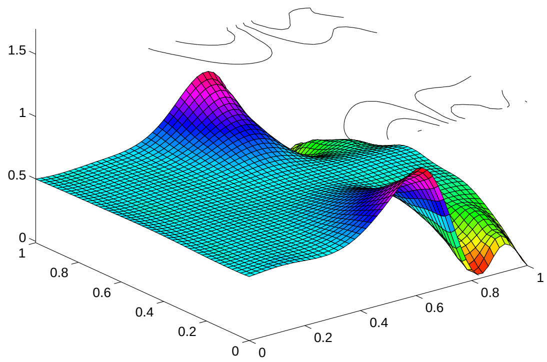


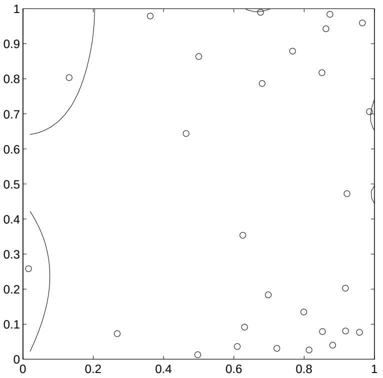
Figure 3.15: Normalisation example. The pulse function is modelled with and without normalisation. In the normalised case, the reactivation of the centre basis function is clearly visible, leading to the sag in the middle of the model output plot. Note the unnormalised model's inability to model the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

Example 2: Modelling a two dimensional Function

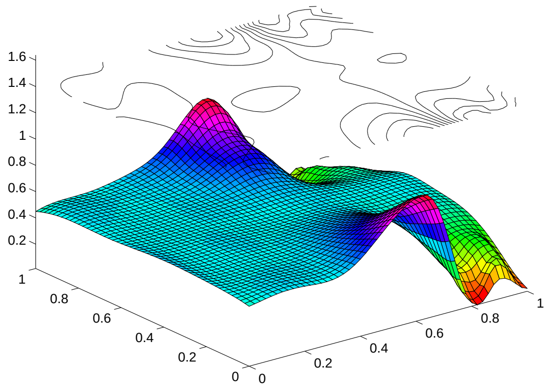
Figures 3.16 and 3.17 show the effect of normalisation on the representation of a 2-D function with basis functions which have little overlap, and those with greater overlap (the σs are twice as large in Figure 3.17). The response plots were created using the same basis functions, but the weights were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



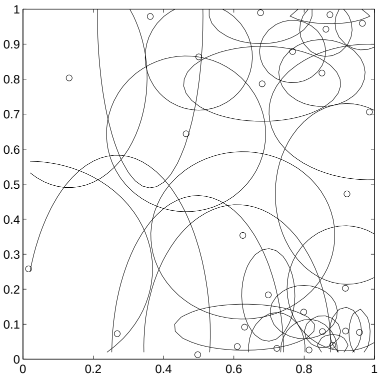
(a) Normalised Model Output



(b) Normalised Basis Functions

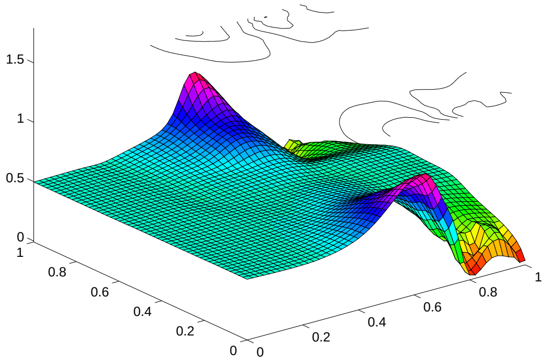


(c) Unnormalised Model Output

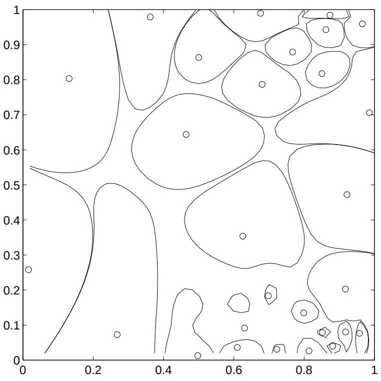


(d) Unnormalised Basis Functions

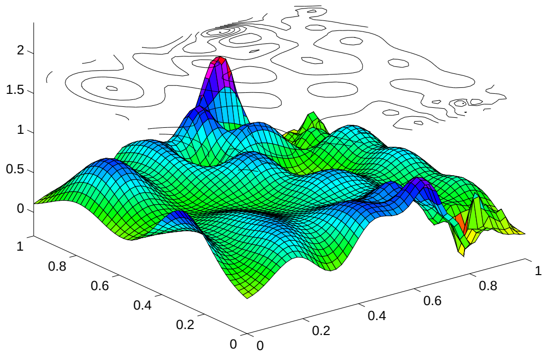
Figure 3.16: Effect of Change of Shape on Model Representation – Wide BFs. Both models produce a good approximation of the target function. Compare this to Figure 3.17, where the narrower basis functions lead to a large difference for the unnormalised model.



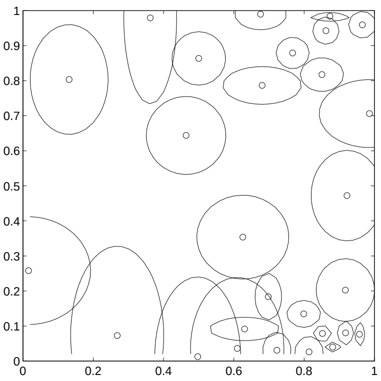
(a) Normalised Model Output



(b) Normalised Basis Functions



(c) Unnormalised Model Output



(d) Unnormalised Basis Functions

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

3.4 Conclusions

3.4.1 Local learning as a robust optimisation algorithm

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

The advantages of the new local learning methods are:

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

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

Future work

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

3.4.2 Local confidence limits

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

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

Future work

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

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

3.4.3 Effects of basis function normalisation

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

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

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

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

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

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

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

Future work

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