A FRACTAL RADIAL BASIS FUNCTION NEURAL NET FOR MODELLING

Roderick Murray-Smith
Daimler-Benz Research, Alt-Moabit 91b, 1000 Berlin 21, Germany.

***Abstract* – A new recursively self-constructing fractal learning algorithm for an artificial neural network is presented. The network architecture is based on a tree-like structure of Radial Basis Function (RBF) networks. This paper examines some of the problems with RBF nets, and suggests a possible solution. Neurons in RBF nets have local receptive fields. The new learning algorithm 'grows' new sub-networks within the fields of existing units producing modelling errors greater than the tolerated limit. The network complexity is thus matched to the complexity of the system being modelled. The local nature of the neurons allows the modelling tolerance to be stored locally, enabling the network to give a varying confidence estimate for outputs from various areas of the input space. The learning algorithm continually undergoes a train–test–train procedure. This prevents overtraining and can be used to create representative training data automatically.**

1. INTRODUCTION

Artificial neural networks have proved to be highly successful in a wide range of application areas. The solutions arrived at, however, have usually been *ad hoc* ones, with only rough guidelines for the choice of network architecture, the learning algorithm, the algorithm parameters and the acquisition of training data. This paper gives a brief introduction to RBF neural nets, then describes a new network architecture and learning algorithm, in an attempt to make the modelling process for continuous systems more automatic and reliable.

This algorithm automatically constructs a neural network architecture with enough representational power to learn the target mapping. The algorithm can also be used on-line, to automatically generate stimuli for the system being modelled, creating a training set which is more densely populated in areas of input space which are mapped to areas where the system's response is more complex.

As the learning process inherently calculates the local modelling accuracy of the network, the final network can give not only the estimated answer, but also a measure of the expected accuracy of this response. The net 'knows what it doesn't know'.

1.1 Notation used

x – model input vector
y – model output vector
n – input dimension
w – weight
N – number of units in network
$\Phi()$ – basis function
$f()$ – local model
c – centre of basis function
δ – separation of basis functions
d – input to local model (distance from centre of local model)
p – number of children created when a model is split

1.2 Radial Basis Function networks

Radial Basis Function neural nets (RBF nets) represent mappings as linear combinations of functions with limited, usually spherical, receptive fields, whose centres are distributed throughout the input space (see Eqn. (1)). Such nets have previously been applied to continuous modelling problems with success, and offer theoretical advantages [10]. There is a close relationship between RBF nets and Albus' Cerebellar Model Articulation Controller (CMAC) [1][4]. These architectures have several advantages over other neural network architectures, due to the local nature of the receptive fields of the 'neurons'. Each neuron is 'locally tuned', so that it only reacts to a small area of the input space.

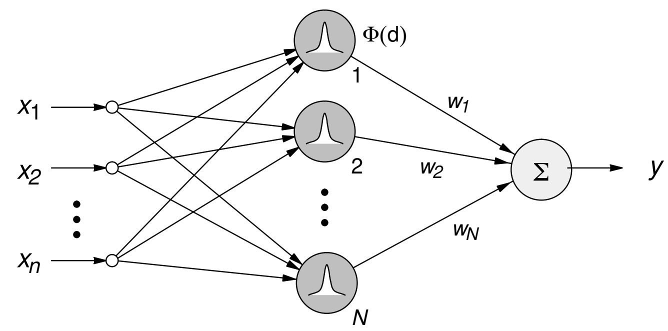

Figure 1: A Radial Basis Function network

$$y(x) = \sum_{i=0}^{N} \Phi_i \left(\| x - c_i \| \right) \cdot w_i \qquad (1)$$

These nets have also been suggested as suitable for incremental (on-line) learning. Training with data from one area of input space will not affect the net's output in a different area, *if the basis functions are local enough*. This means that rarely occurring inputs will still be learned correctly and that on-line adaptation is feasible.

1.2.1 Regular spacing of the basis functions

The basis functions of the network can be inititialised to cover the input space uniformly. The number of units, their radii and spacing must be suitably set beforehand. This makes the process of optimising the weights *(w)* straightforward, as the learning algorithm becomes least squares regression, which is linear and can be completed in one pass.

This style of function approximation is comparable with a look-up table using interpolation between entries. The disadvantage is that the complexity of a mapping varies over the input space, meaning that the most complex area dictates the density of the basis functions in the whole network, making this style of initialisation very inefficient for high-dimensional problems.

1.2.2 Irregular spacing of the basis functions

It is also possible to adapt the centres and radii of the basis functions to suit the local data complexity. The adaptation of all of these parameters is, however, a non-linear optimisation problem, and is therefore a more complex task, and convergence cannot be guaranteed.

Moody and Darken [8] describe a hybrid technique which used an unsupervised k-means clustering algorithm to place the centres (the number of centres is chosen explicitly), and then estimated the radii by using a nearest neighbour heuristic. They also attempted to optimise all of the parameters using a conjugate gradient algorithm, but the results were disappointing. Another technique is to place the centres directly on data points from the training set [10][11].

1.2.3 Problems with RBF Nets

The 'curse of dimensionality'; as described earlier, the number of units needed in a regular network grows exponentially with input dimension. As the receptive fields are local, the generalisation ability of the network is obviously reduced, implying requirement of large quantities training data, which is sometimes not feasible.

As described in 1.2.2, the placement of unit centres and the determination of the radii of the units is an area of active research, with several hybrid algorithms being used. There is still no systematic way of deciding on the number of units needed.

2. The Fractal RBF net

The network presented in this paper is described as 'fractal' because it recursively creates subnetworks with the same architecture and learning algorithm, applied at different scales over the input space.

Other authors have also proposed tree-like neural networks. Utgoff describes *Perceptron Trees,* which are decision trees with perceptrons as decision nodes, and which perform classification of integer problems [12]. Deffuent [3] describes a *Neural Units Recruitment* algorithm which uses perceptrons for Boolean classification problems with real inputs. Baram [2] describes a fractal neural network, based on the Hopfield neural network model, for image storage.

The use of RBF networks and the recursive construction algorithm described in this paper are similar to that used by Omohundro's *Bump-trees* [9], although the recall in the fractal RBF net is not equivalent to tree search. The automatic construction of a training set related to the modelled system's complexity is new. The validity estimation has previously been applied only to normal RBF nets [7].

In the following, a *unit* refers to a local model which should be a simple, optimisable system, such as a straightforward weight or a linear combiner. This is then modulated by a basis function which prescribes the region of the model's validity.

2.1 The network architecture

2.1.1 Local models

The outputs of particular units can have a direct physical meaning, as they can be thought of as 'local models' which provide an accurate model of the system within their limited receptive field.

The models used within the units can theoretically be anything from a single weight to a multi-layer perceptron or a conventional model, such as a polynomial representation. In some cases it may make sense to include a customised model structure, which was derived from *a priori* knowledge. The network equation, with local models $f_i()$, now becomes:

$$y(x) = \sum_{i=0}^{N} \Phi_i \left(\| x - c_i \| \right) \cdot f_i(x) \tag{2}$$

A simple model with linear learning algorithms, guaranteeing convergence, is desirable. In the example used in this paper, a simple hard limited linear combiner was used. This approach has also been used by Jones *et al.* [6].

2.1.2 Basis functions

The basis functions should be thought of as being a *model validity function* of the particular model. Their outputs are close to unity when the model is accurate and decrease towards zero as the model loses accuracy (i.e. as the input moves away from the model's normal operating point) [5]. The basis functions for the models should be chosen to have compact support, i.e. they have no influence outside a limited receptive field [4]. This means that training one area of the input data will not affect the output of an unrelated part of the net.

The centres and radii of the functions should be set so that only the directly neighbouring units overlap, by setting the radii of the units equal to δ, the distance between neighbouring centres. This ensures that between centres, $\sum_{i=0}^{N} \Phi_i(d) = 1$, and thus that the interpolation is consistent. Basis functions at the edge can be given maximum validity past their centres towards ∞, by normalising the basis functions (3). The basis functions thus perform a *partition of unity* [9]

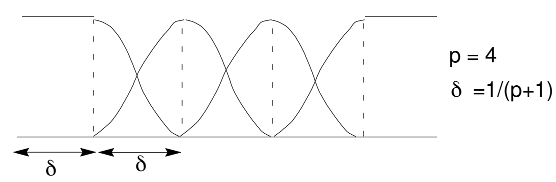

Figure 2: Normalised basis functions

$$y(x) = \sum_{i=0}^{N} \frac{\Phi_i \left(\| x - c_i \| \right)}{\sum_{k=0}^{N} \Phi_k \left(\| x - c_k \| \right)} \cdot f_i(x) \tag{3}$$

The Gaussian bell $\Phi(d) = e^{-\frac{d^2}{\delta^2}}$, although often used, is therefore not ideal, as it has neither compact support nor vertical symmetry. A better choice, for example, is the function $\Phi(d) = \cos^2(d)$, where $-\delta < d < \delta$ is scaled to the range $-\pi/2 < d < \pi/2$. This function has the advantage of the desired compactness, and correctly spaced units can create a level surface summing to 1. It was used in the example in this paper.

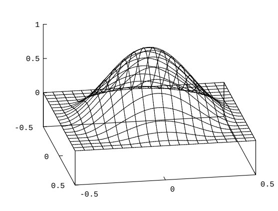

Figure 3: $\Phi(d) = \cos^2(d)$ as activation function

When dealing with a multi-dimensional input space, placing the centres becomes more complex. Figure 4a shows the standard placing of fields in a regularly spaced RBF net. This spacing leads to unequal distances between neighbours (δ & $\delta\sqrt{2}$).

In Figure 4b, the units are shown with a different layout. The use of an offset between rows of units allows the placement of receptive field centres so that they are always a unit distance (δ=radius of basis function) away from their neighbours. This was used to ensure that the input space is more evenly covered by the basis functions.

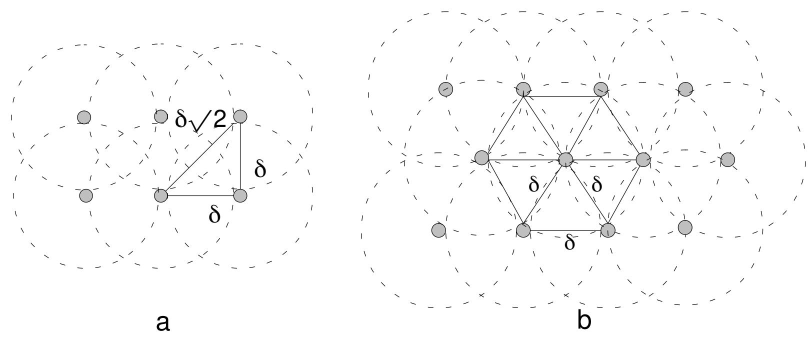

Figure 4: Two possible arrangements of the receptive fields

The use of regularly spaced basis functions with radii equal to the spacing also has the effect that any point in input space will be in at most 2^n receptive fields. As the net is regular, the output can be calculated very quickly, using a hashing algorithm, making the speed of recall dependent only on the number of input dimensions (*n*), not on the total number of units (*N*), which depends on the complexity of the function being modelled. This also potentially enables a simple implementation of RBF nets with standard RAM technology.

2.2 The learning algorithm

The flow diagram of the learning algorithm is shown in Figure 5. The form of the system using minimal *a priori* knowledge is a network initialised with only one unit, which has a uniformly active model validity function. Training proceeds until no more progress is made. At this point, the unit(s) producing errors greater than the permitted tolerance levels 'grow' another unit within their respective receptive fields (this unit is identical in structure to the main network, but applied at a smaller scale).

This process is then repeated recursively until the worst errors produced by the training set are satisfactory, at which point the net is tested using data from the test set. Beyond the aspects related to learning a given training set, the algorithm addresses two other important aspects; namely the adequacy of the training set and measurement of the confidence range of the output, throughout the input space.

2.2.1 How local should the model training be?

The training for each local model can be achieved globally, where inputs are presented to the whole system, allowing the network to use the error at the top level model output to train the relevant local models. An alternative is to train the local models individually, with no regard to the other models. The global approach will result in the most parsimonious model, but is more computationally expensive. The global solution was used in this work.

An alternative hybrid solution is a recursive training algorithm, where a net calls its children to optimise themselves locally, then optimises itself, and its children, 'globally' at its level. This results in the training process for all nodes starting locally and progressing to a network global optimisation.

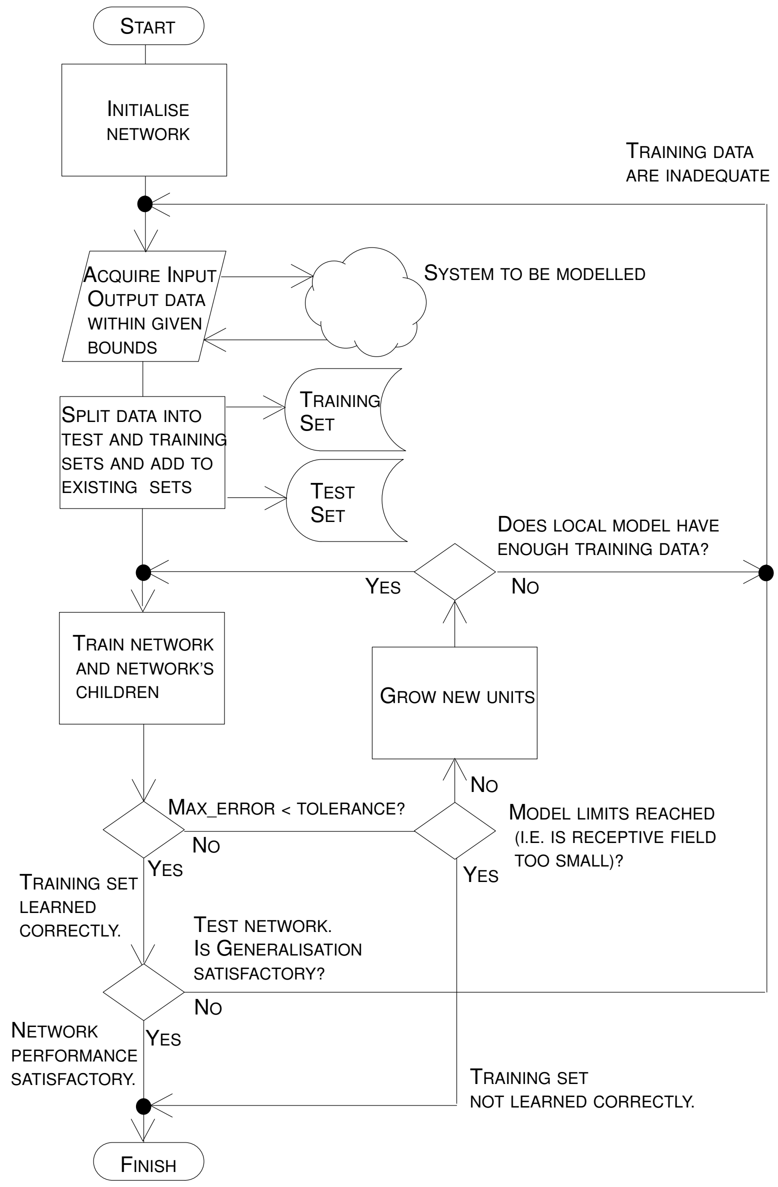

Figure 5: Flow chart of fractal net construction algorithm

2.2.2 Growing sub-networks

As already mentioned, an increase in representational power of the network can be achieved by growing sub-networks within the receptive fields of existing units, as shown in Figure 6.

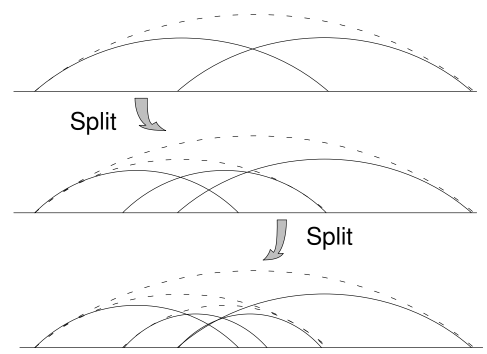

Figure 6: Growing new models by splitting the basis functions hierarchically

The units which are to be expanded are those with an error greater than the tolerance level within their receptive fields. The number of nodes to be split at each iteration depends on the training philosophy being used; local or global. When local training is used, each unit with *error > tolerance* will split, whereas the use of global training will result in a limited number of units being selected.

There are several ways of selecting the units to be split. One possibility is that only the unit with the maximum worst error should be split. This method becomes problematic in conjunction with global training, when the function has very steep areas, or discontinuities. The network only grows around such areas, neglecting other more easily learnable parts of the function. Another method is to choose the units probabilistically, with the probability of a unit splitting being proportional to its worst error.

A minimum size of receptive field is set as a halting condition, as some functions, e.g. a step function, require an infinitely small radius, which can never be achieved. In some cases, units will not be able to split because the sub-models have reached this minimum. There may also be insufficient training data for some of the sub-models, prompting a request for more data in this area (see section 2.4).

Once the decision to grow from a unit has been made, how should it split the input space within its receptive field? The straightforward scheme for the one-dimensional case is shown in Figure 7. The area covered by one basis function now contains p basis functions, with radii $\delta = 1/(p+1)$, spaced at intervals of δ, from δ.

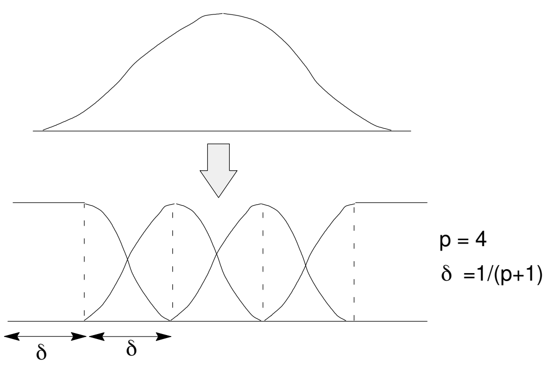

Figure 7: Splitting a basis function

This question of where and how to split basis functions is not trivial when dealing with multi-dimensional problems. The most straightforward scheme is simply to split each dimension in p places, introducing p^n new units. The network size, however, increases exponentially as n increases.

This means that, rather than splitting all dimensions, one of the dimensions must be chosen to be the discriminating one. The problem of choosing the dimension to split on is very similar to the problems faced in constructing decision tree algorithms. It should be possible to apply techniques from that field. The method used in this paper was to relate the probability of a split along a given dimension to the absolute weighting of that dimension (equivalent to the gradient of the hyperplane in that dimension) in the particular unit's local model.

2.3 Confidence of the network output

As the overall model is made by combining many local models, each of these models can also learn the tolerance of its output. This ranges between the worst positive and negative errors which occurred in the training and test phases within the receptive field of that local model. As the accuracy of any local model decreases as the operating point moves away from the local model's centre, the errors are weighted by the local model's basis function for the current input, before being compared to the worst errors.

This allows the output of the network to be accompanied by a confidence measure valid for the area of input space around the current input. The local model could also detect whether there was a significant amount of training data in its receptive field for it to be able to give a valid output. Leonard, Kramer and Ungar introduced a form of confidence estimation with their 'Validity Index' network [7], where statistical measures were used to evaluate the sufficiency of the available data, and to calculate the confidence limits for each unit.

2.4 Automatic construction and extension of the training set

To train any system capable of learning requires a representative training set of input-output pairs, yet surprisingly little attention has been paid to this aspect of machine learning. The system to be modelled must have been sufficiently stimulated to ensure that the training data cover the input space adequately. Satisfactory results have often been achieved with *ad hoc* solutions, but a general solution for the task becomes difficult for non-linear, multivariable systems. If the amount of training data is not to become impractically large, an amount of *a priori* knowledge must be used, to make a more intelligent acquisition of training data.

The fractal RBF network, like most other learning systems, can learn an arbitrary training set. The fractal construction algorithm, however, also offers the opportunity of creating a training set which is related to the complexity of the system being mapped. The network can acquire data in a more focused way, because of the local nature of the models, allowing the more complex areas of the model to have an appropriate amount of training data.

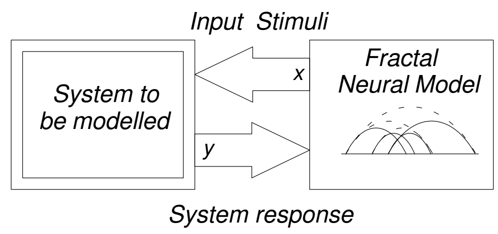

Figure 8: Network stimulates the system to enhance its training data

If, after training, certain areas of the net produce unacceptable errors on the test set, or if there is too little training data within the receptive field of a given unit for it to grow children, the network will create new input stimuli for the system. These stimuli are created by taking random points from within the receptive field of the unit which requires more data. The data measured from the system's response are immediately split into test and training patterns, ensuring that the test set is also representative.

The previously inadequate training and test sets, can thus be steadily improved, from zero contents if necessary, to a set of input–output pairs, forming a representative sample of the system responses.

3. The Fractal Net algorithm in practice

As an example of the network's learning ability, the function:

$$y(x) = \sin\left(10\pi \sin\left(x^3\pi\right) \cdot \left(x + \frac{1}{2}\right)\right); \qquad -\frac{1}{2} < x < \frac{1}{2}$$

was chosen as a test system for the learning algorithm to model, as this is an example of a non-linear function containing areas of differing complexity.

The learning progress of the fractal network is shown in Figures 9 to 11. The upper part of each figure shows the validity functions (dark) of, and model outputs (grey) from the local models. The lower part shows the target function, the network's approximation and the tolerance area of the network's response (shaded area). The position of the worst error is shown by a vertical line.

For this implementation, the network was initialised with only one unit, the local models were simple hard limited linear combiners. Each unit could grow two sub-units. A global approach to training, and error estimation was taken. To ensure a more even training progress, each non-leaf node could send a split command to all children with an *error > tolerance*, but only the worst leaf node of any given model could be grown from.

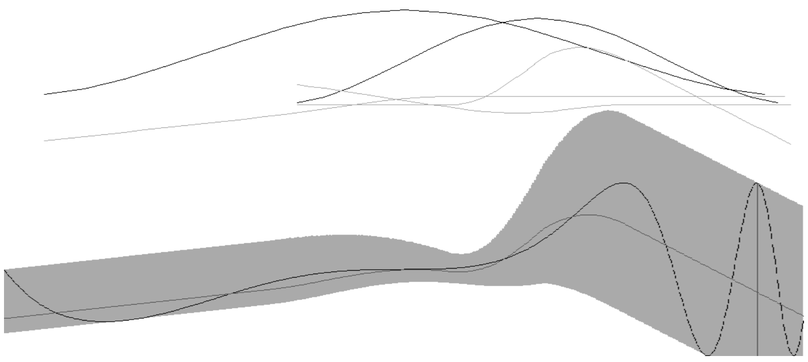

Figure 9: Early stage of learning (2 iterations), only very rough approximation.

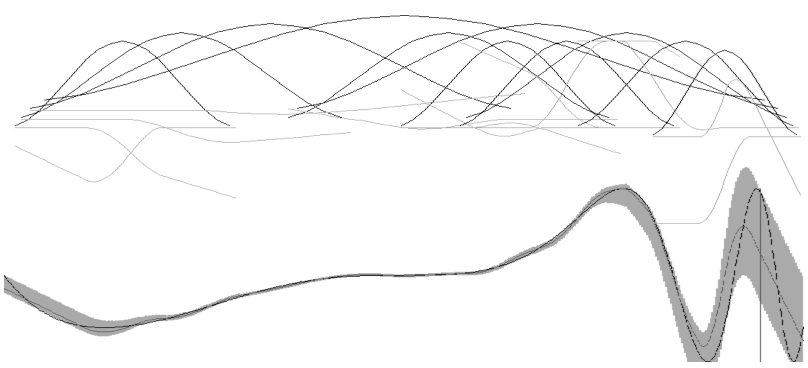

Figure 10: After 4 iterations. Some areas already well learned, others less so.

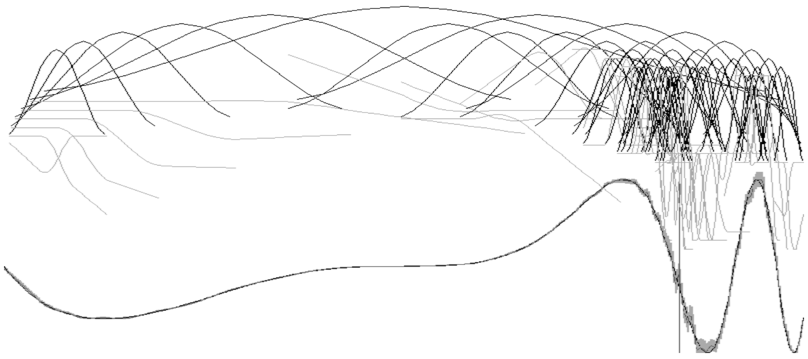

Figure 11: Final state. Note the different densities of the basis functions.

The confidence limits were simply the worst errors occurring during the training and test phases. These were then modulated by the basis functions and summed together. The desired tolerance was set to 5%, and the limit on the size of receptive field was set to $(2\delta)^9 = 0.026$.

Although this example shows that the ideas developed in this paper are feasible, achieving an average absolute error of 1.2% in this example, the various open questions regarding the training algorithm and network architecture need further work. Which other structures are suitable as local models? How can the global/local trade off in training be best solved? How should the units be split?

For complex, multi-variable function approximation, the computational load may become excessive for conventional workstations. Due to the local nature of the activation functions, standard RBF nets lend themselves to parallel implementation. Each local model can train relatively independently, with its own local training and test sets.

4. Conclusions

The Fractal RBF net algorithm described in this paper represents a method of creating a neural network to model a given function or system. The algorithm does this by recursively constructing a tree-like structure of structurally identical sub-networks.

The algorithm can also automatically produce the necessary training and test data, if the system to be modelled is available for on-line testing. Testing, tolerance and validity measurements are all intrinsic parts of the learning process, ensuring good generalisation.

The automatic creation of training and test data dealt only with multivariable static systems. The obvious next step to the automatic production of training sets for non-linear dynamic systems is a difficult, but significant one.

With the advantages listed above, this network architecture has the potential to make the process of modelling non-linear, multivariable functions faster and more reliable than with other neural networks.

5. References

[1] J. S. Albus, *Data Storage in the Cerebellar Model Articulation Controller*, Trans. ASME, Journal of Dynamic Systems, Measurement & Control, 97, p220–227, 1975

[2] Yoram Baram, *Associative memory in Fractal Neural Networks*, IEEE Trans. Syst., Man and Cybernetics, Vol. 19,No. 5, p1133-1141.

[3] Guillaume Deffuant, *Neural units recruitment algorithm for generation of decision trees*, IJCNN 90, San Diego, Vol. I, p637-642.

[4] Stephen H. Lane, David A. Handelman, Jack J. Gelfand, *Higher-Order CMAC Neural Networks – Theory and Practice*, Proc. American Control Conf. 1991, Vol 2., p1579-1585.

[5] Tor A. Johansen, Bjarne A. Foss, *Representing and Learning Unmodeled Dynamics with Neural Network Memories*, American Control Conference, Chicago, June 1992.

[6] R. D. Jones, Y. C. Lee C. W. Barnes G. W. Flake, K. Lee, P. S. Lewis, S. Qian, *Function Approximation and Time Series Prediction with Neural Networks,* IJCNN 90, San Diego, Vol 1, p649-665, June 1990.

[7] J. A. Leonard, Mark A. Kramer, L. H. Ungar,
A neural network architecture that computes its own reliability, Submitted to Computers and Chemical Engineering.

[8] J. Moody, C. Darken, *Fast Learning in Networks of Locally Tuned Processing Units*, Neural Computation 1, p281–294, 1989.

[9] Stephen M. Omohundro, *Bumptrees for Efficient Function, Constraint, and Classification Learning*, Tech. Report 91–009, January 1991, Internat. Computer Science Institute, Berkeley, California.

[10] Thomas Poggio, Federico Girosi, *Networks for Approximation and Learning,* Proc. IEEE, Vol 78, No. 9, p1481-1496, Sept. 1990.

[11] Donald F. Specht, *A General Regression Neural Network* IEEE Trans. Neural Networks, Vol. 2, No. 6, p568-576, Nov. 1991.

[12] Paul E. Utgoff, *Perceptron Trees: A Case Study in Hybrid Concept Representations*, Proc. of the 7th Nat. Conf. on AI, St. Paul, Minn., AAAI Press, Menlo Park. Calif., 1988, p601-606.