COBEM 99
XV Congresso Brasileiro de Engenharia Mecânica
15th Brazilian Congress of Mechanical Engineering
22 - 26 de Novembro de 1999 / November 22 - 26, 1999 Águas de Lindóia, São Paulo,

SMART TRAINING IN NEUROCOMPUTING MODELING OF ENGINEERING SYSTEMS

Andrew J. Macdonald
Senior Staff, The Hughes Network Systems, Inc.
Z. Peter Szewczyk
Research Engineer, CALNET, Inc.

Abstract

In this paper, "smart" training is introduced to increase the accuracy of neurocomputing modeling and reduce the time required to develop neurocomputing applications. It is shown that both objectives are attained by (i) partitioning the training data into contiguous subsets whose response belongs to a similar class, and (ii) designating different neural networks to fitting partitioned data. Response-based partitioning is performed based on quantification of the response characteristic by an automated clustering routine. The routine uses a rule-based fuzzy logic inference engine to adapt data partitioning to the changing response characteristic and to make them suitable for training. The neurocomputing modeling with response-based clustering is applied to two distinctly different problems. The first problem addresses the need for continuous representation of data produced by a finite element analysis, and specifically, displacement fields in the upper skin of a wing box are modeled. In the second problem, simulated data representing power loss of transmitted signals to be used in design of a wireless communication system in Prague are modeled.

keywords: neurocomputing, smart training, wing deformation, power loss modeling

1 Introduction

Behavior of many engineering systems and processes can only be inferred from discrete data collected via their observation or simulation, and examples of such systems can be found in any engineering field. Such inference or generalization is indispensable in system design or modification if the information about the system behavior or its characteristics is not in the database. There are few systems whose the exact mathematical description or model exists, and in most cases approximate models within predefined ranges of validity can only be obtained. Since depending on a system, observations or simulations may also produce a large amount of data, there are two major reasons for modeling: (i) obtaining continuous models that are sufficiently accurate for the application, and (ii) compressing available data for storage.

There are many techniques for continuous modeling of the engineering systems behavior, but a neurocomputing paradigm has become an indispensable tool in modeling from discrete data. There are general proofs of neural networks being universal approximators (Funahashi, 1989; Moody and Darken, 1989), and the following advantages in their use: (i) a unified approach to handling modeling problems from any discipline suitable for automated implementation, (ii) compact models of discrete data yielding high data compression ratios to reduce storage requirements, (iii) fast function evaluation enabling interactive, rapid processing, and (iv) model-free data fit as the order of a fitting hyperplane is embedded in the network architecture has spawned many successful applications of the paradigm. An important issue that arises in the use of neurocomputing, however, is the efficiency of obtaining functional mappings and also the accuracy of approximations. It is shown (Szewczyk, 1998) that the efficiency and effectiveness of neurocomputing modeling can be increased by training networks of a different architecture on subsets of existing data, and that partial neural network models can be smoothly combined into a model of the whole domain. In a view of these facts, the immortal strategy of "divide and conquer" can also be applied to developing neurocomputing applications. However, to take full advantage of this strategy, a data partitioning technique must be available that produces: (i) contiguous subsets of data, and (ii) response in each subset belongs to a similar class of functions. Furthermore, such a technique must be fully automated as the amount of book-keeping may render it useless from a practical standpoint.

This paper presents a comprehensive approach to neurocomputing modeling that addresses aforementioned problems associated with the use of neural networks. In particular, (i) a response-based clustering technique is used which partitions the training data so that a response field in each contiguous subsets belongs to a similar class; (ii) networks with a different architecture are designated to map data in different subsets; and (iii) a distance-based classifier is used to obtain an estimate from the collection of neural networks. It is believed that since all decision-making is relegated to a programmable module and developing a neurocomputing application is fully automated, the term *"smart" neurocomputing modeling* is justified to designate this approach. To eliminate the need for human inspection when the response-based domain decomposition is performed, a set of rules describing a cluster of training data that is suitable for training is processed. A selection of a fuzzy set inference engine for processing is supported by explicit "reasoning", and the fact that the aggregated value of confidence in decision-making can also be computed. Another decision-making element embedded in the approach, the architecture selection, is a straightforward classification problem. Smart neurocomputing modeling is illustrated with modeling of two distinctively different data fields. The first example arises in finite element analysis, and a displacement field is considered. The second data field is produced by power loss data generated for a wireless communication system.

2 "Smart" neurocomputing modeling

The fact that neurocomputing data fits with the use of feed-forward neural networks are error minimization problems presents a formidable computational task, particularly for large input spaces with complex response and many data points. Several different approaches to improving the efficiency of developing a neurocomputing application have been developed, e.g. a specialized error minimization algorithm for training, or parallel computations. "Smart" neurocomputing modeling is yet another approach that can be used in concert with other techniques. It can be described as an implementation of the "divide and conquer" strategy in which all training data are preprocessed to extract information about response characteristics and designate different networks to model data in subdomains of similar response. The objective of smart neurocomputing modeling is to separate

existing data into subsets suitable for training without the need for human intervention.

2.1 Feed-forward neural networks

A feed-forward neural network can be viewed as a directed graph without any loops in the information flow through the network. A node of the graph is a simple processing unit that transforms a sum of all signals directed to it according some transfer function. Connections between nodes are subject to modifications so that the difference between target output values and their estimates is minimized. There are two major categories of feed-forward neural networks.

Backpropagation neural networks. Training of a BPN net is a minimization problem in a space of network parameters. It is a difficult optimization problem because the parameter space is high-dimensional and non-convex, and the fit may be required into a large number of data. Therefore, to find a set of parameters that satisfy an error tolerance, the data field may have to be simplified in addition to the use of an efficient search technique.

Radial basis function neural networks. Radial basis function neural (RBFN) network estimates are computed by blending values of basis functions, e.g. gaussian, that overlap for a given input. The training is completed when the prescribed tolerance is met by adding more basis functions, and in the case of gaussian functions the location of their centers and their standard deviations are optimized. The training of RBFN networks is simpler but the complexity of data may result in an unacceptably large network, therefore the data field simplification may also be necessary.

2.2 Response-based data partitioning

Data partitioning is the way to simplify the discrete data fields, and it is usually necessary when observations or simulations of a system produce a large amount of data. Clusters of data suitable for training should essentially be (i) *disjoint*, (ii) *contiguous*, and (iii) *dendriform-less,* as spread-out clusters with gaps that lead to inferior models and such clusters can be produced by a clustering routine that processes only input information. By restricting clustering to inputs, a domain decomposition can be attained, and such clustering is suitable for preprocessing training data when samples are non-uniformly distributed in the input space to identify regions or subsets with insufficient data for modeling, and when outputs belong to the same class of functions. If it is desirable that response in each subset belongs to a similar class of functions, clustering must also include response information, particularly when the response shows distinctly different characteristics in different regions. Clustering with response information may produce, however, even if some scaling and/or weighting factors are applied.

Clustering with response information can produce .

Is is clear that to make a clustering routine equally sensitive to a datum location and its value, some scaling and/or weighting factors may be required. Even with reasonable factors, clustering may produce islands of data far away from the cluster core and also gaps in the core. Unlike some small islands of data near the border between adjacent clusters that facilitate a smooth transition between neurocomputing models, .

Consider vectors $\mathbf{s}_p = [\mathbf{x}_p, \mathbf{y}_p]$,where $p = 1, \ldots, P$, a juxtaposition of $\mathbf{x} \in \mathcal{R}^n$ and $\mathbf{y} \in \mathcal{R}^m$, input and output vectors. A set of these vectors, containing all information that is available about the mapping $\mathbf{x} \mapsto \mathbf{y}$ to be modeled, is to be partitioned into subsets that satisfy conditions (i) $\div$ (iii). Partitioning into disjoint subsets can be performed by any distance-based clustering routine. We use a clustering routine with a control parameter, called *resolution*, which allows to specify how far apart vectors need be to belong to a different cluster (Szewczyk, 1998.) Executing such a clustering

on vectors $\mathbf{s}_p$, yields number of clusters, κ, their exemplars, $\mathbf{c}_k$,(viewed as some average of all the data in the cluster) and partitioning, S_k, $k = 1, \ldots, \kappa$, of all the data. Thus,

$$\begin{array}{rrcl} \forall_p \, \exists_k & \mathbf{s}_p & \in & S_k, \\ \forall_{i,j} \quad S_i \cap S_j & = & \emptyset, \quad i \neq j, \quad i,j = 1, \ldots, \kappa, \\ S_k & \rightarrow & \mathbf{c}_k \end{array} \tag{1}$$

Subsets of data produced by such clustering are clearly disjoint, but they may violate the conditions (ii) and (iii), and also combining adjacent clusters into one may be justified if their response is sufficiently similar. By formalizing these intuitive requirements into knowledge-based rules, a fuzzy set based inference can be employed to assure that all conditions are satisfied.

2.3 Rule-based cluster processing

The knowledge about an engineering problem and experience can be expressed in the form of IF-THEN rules. Such rules attempt to capture relations between input and output quantities and may be imprecise if they employ linguistic terms taken from a natural language. It is a common practice to use a standardized *term-set*, $T(\Psi)$, to express rules in engineering applications, where $T(\Psi)$ is an ordinary union of such descriptors as *low, below average, average, above average,* and *high.* Once a set of rules has been formed, a fuzzy set approach may be taken to process a specific instance. In the fuzzy set approach, each term must be associated with a specific membership function. To process a new evidence, an inference engine checks whose rule antecedent is matched and to what degree. Consequents of these rules are combined, and a crisp answer is returned after defuzzification. It is also reasonable to express cut-off values in the decision-making as fuzzy numbers, $\widetilde{n}$, and use fuzzy arithmetic when combination of rules with fuzzy numbers must be processed. When the fuzzy set approach to automation of decision-making is taken, the aggregated measure of matching antecedents of all rules that fired rules can be computed and used as the measure of confidence in the process. It is worth noting that this conceptually simple inference may require a great expert's effort to extracting rules and associating $T(\Psi)$ with a membership function; particularly that rules and selection of membership functions are strongly problem dependent.

There is a small set of rules that captures the knowledge required to produce clusters that satisfy conditions (ii) and (iii). First of all, some inspection and the engineering feedback are needed to select the acceptable number of clusters, because if the value of resolution is reasonably chosen, then the domain is divided into initial clusters whose boundaries follow response characteristics quite well. We designate a fuzzy number $\widetilde{n}$ to describe what the acceptable number of clusters is. Next, the response similarity within initial clusters needs to be determined. If the response is within some $\widetilde{s}$, and the clusters are adjacent then they can be combined into one to reduce the number of neural networks in the modeling process. Note that adjacency of two clusters can be determined based on examining whether in the "lightbeam" emanating from the center of a given cluster, the cluster with the similar response is the closest. Finally, elimination of the dendriform and islands of data by swapping packets of data between adjacent clusters appears to be sufficient to satisfy the contiguity condition. Therefore, the last, but also the most complex rule attempts to formalize what a dendriform is.

The shape of a cluster depends only on its members location, therefore further processing can be done without any output information. Execute then the clustering algorithm with only input vectors as entries for a given resolution. The algorithm finds the exemplar for each subcluster, or some geometric center associated which some average of the output values. These average values may be somewhat different from the average value for the whole cluster, but the exemplars should

be on average the resolution apart from a few adjacent sub-clusters for a dendriform-less shape. If this is not the case then the dendriform can be eliminated by either moving spread-out data to an adjacent cluster, or acquiring more data from the adjacent cluster to fill up existing gaps in data.

It is clear that for this decision-making process to yield satisfactory results, membership functions in the last rule and cut-off values of parameters must be properly tuned. Such tuning can only be done based on experiments with the process, and some human interference with it may be necessary. If firing of given rules is caused by relatively low values of antecedents, their aggregated value may be used as the prompt to inspect the decision-making process.

2.4 Training in subregions and the use of neurocomputing models

The process of obtaining the BPN network model of a mapping is of a statistical nature due to a randomized initial set of parameters, and it is uncertain whether a training tolerance can be met by the trained network. In a training routine used in this work, a simple Monte Carlo simulation is used to find a network with the smallest initial tolerance. Then, the MATLAB[1] implementation of the Marquardt's algorithm is executed on that network, and if the network that satisfies the training tolerance cannot be found in a prescribed number of training trials, the network architecture is increased.

Training of a RBFN network is simpler, as the architecture is automatically increased if the training tolerance cannot be attained for a given spread. Since network approximations depend on the value of this parameter, the search for its value must be included in the training process. A simple line search for the spread value is performed before the size of a network is increased.

As the result of completing training in subregions, there is a collection of networks available for approximation. Therefore, when a new estimate is required, a simple classification problem needs to be solved to determine to which model the input belongs. Since each model is valid in a specific subdomain, its scope can be determined by clustering each subdomain with respect to input quantities with the same value of the resolution. Such clustering yields a set of exemplars for each subdomain which can be used in the classification problem. Since each exemplar is uniquely associated with a model, finding the closest exemplar to the given input selects the valid model.

3 Numerical Studies

3.1 Neurocomputing modeling of a progressive deformation of a wing box

A geometrically nonlinear wing box with a hole shows a non-uniform response field when subject to a bending loading. There is stress and deformation concentrations around the cut-out, and a finite element model of the box contains 2631 nodes refined around the hole. In finite element analysis of nonlinear structures loading is gradually increased to avoid numerical instabilities, a progression in deformation and other response quantities is available. Such a progression can be visualized using smoothing capabilities of post-processing tools, but the information about a response remains only known at the nodes limiting its usefulness in a design process. Therefore, the objective of the first example is to obtaining a continuous model of the progressive deformation of the upper skin of the wing box for 15 load steps.

[1]MATLAB is the trademark of the Math Works, Inc.

Table 1: The efficiency of NN models of deformation at the load step 12

CID	#_data	#_par	max_e	DCR
A	377	18	0.08	21:1
B	105	17	0.06	6:1
C	410	21	0.04	19:1

3.1.1 Structural response classification

Since the quality of neurocomputing modeling depends on the complexity of a response within a cluster and the amount of information to be processed, a degree of nonlinearity of the response must be quantified and included in partitioning. Such a quantification can be done based on the *relative deformation* defined as the difference between the response in the vicinity of a given point and the response at that point, thus

$$\begin{aligned} \delta(x,y) &= |0.25[(f(x+h,y)+f(x-h,y) \\ &+ f(x,y+h)+f(x,y-h)] - f(x,y)| \end{aligned} \tag{2}$$

where h represents an arbitrary step. Note that this definition uses a continuous model derived from a neural network, therefore the neighborhood of a given point can be defined independently from a FE mesh. This quantified information about the degree of deformation is used, along with the location (x,y) to partition the data. It is worth noting that this derivative-type operator is very sensitive to the quality of function approximation, and it actually can be used as a benchmark for the accuracy of approximations.

3.1.2 Response-based partitioning and training

A neurocomputing model of the skin deflection can easily be obtained for the first few load steps, as the deformation is very small and almost linear. Such a model allows quantification of the degree of nonlinearity using Eq.2 as it is continuous. Assuming that $\widetilde{n} = 4$ and $\widetilde{s} = 0.1$, all clusters are combined into one until the load step 7 when a cluster around the whole is isolated as having the different degree of nonlinearity. When the load increases, more clusters are identified, and Figure 1 shows two snapshots of how the clustering routine works. At the load step 7, the data partitioning is done for the first time, and two clusters of data are identified. The training is done in each cluster independently. Boundaries of clusters keep changing slightly as the load increases. At the load step 12, the training data is separated into three clusters, and the details of their training is shown it Table 1. The BPN nets are used in Cluster B (2-4-1) and C (2-5-1), and the data in Cluster C are modeled using the RBFN net with only 5 neurons; the total number of net parameters is denoted "#_par." The data compression ratio (DCR) shown in the Table is computed based on the uniform mesh that is used to show the deflection (the total number is 892) not the number of nodes that is used in the FE analysis and training the networks. If all the nodes are taken into account, the DCR would be three times greater than shown. The training error, denoted max_e, is computed as the maximum discrepancy between a target value and its approximation expressed in inches. It is estimated that the time required to obtain neurocomputing models is shortened fourfold in this particular application. It is worth pointing out that the decision how to separate the data into clusters is sensitive to the shape of membership functions. It requires a lot of adjustments to make automated decisions intuitively correct. The time required to design the FL inference engine is not taken into account when efficiency of the clustering routine is estimated.

3.2 Neurocomputing modeling of power loss data

In the second numerical example, the modeling of the power loss in the deployed wireless communication system in Prague is considered. To It demonstrates the feasibility of neurocomputing modeling with smart training in the case when the complexity of the response and the amount of data to be modeled makes the modeling task insurmountable otherwise.

3.2.1 Propagation models

Power loss characteristics are an integral part of design of any wireless system as the signal-to-noise (SNR) ratio depends on them. An exponential relation between the power loss and the distance from an antenna with the exponent reflecting the type of environment, can be used to build a theoretical model of wave propagation. But in-field measurements have shown that the actual power loss can differ significantly from the values predicted by a distance separation due to shadowing and fading effects. To account for these effects, it is a common modeling practice to assume a log-normal distribution for shadowing and the Raleigh fading. Therefore, a random distribution with the mean value, μ, that follows the exponential law, and with some assumed standard deviation, $\sigma > 0$, simulates power loss more realistically. Further improvements can be achieved with the use of drive-test data produced by listening to all A antennas in a system from a given point. The average signal strength, μ_i, and its deviation, σ_i, where $i = 1, \ldots, A$ can be estimated based on these measurements to increase the model fidelity. In our work, power loss data are generated based on theoretical models but limited drive-test data are also used to adjust the model.

3.2.2 Power loss for a selected antenna in the system

It is assumed that the mean value of the power loss is to be modeled, and there are about 40,000 data available for a deployed antenna in the system. Each datum can be represented as a vector, $\mathbf{s}_p = [x_p, y_p, \mathrm{L}_p], p = 1, \ldots, P$, and the coordinates and power loss are normalized onto the $[0.1, 0.9]$ interval. The power loss data are shown in Figure 2a. Figure 2b shows the same data, but already divided up into response-based clusters assuming that $\widetilde{n} = 15$ is the acceptable number of clusters. When $\widetilde{s} = 0.1$, three adjacent clusters are combined into one.

For the illustrative purposes, the training of a BPN network on a selected cluster of 1041 data is shown in Figure 2c. The network with 21 neurons in each of the two hidden layers (2-21-21-1) is the smallest that attained a prescribed training tolerance, and it yields approximations with the maximum error of 4.47dB, and the average of 0.66dB. As an addition error measure, it is shown that for 36 grid points the error is greater than 1.9dB.

Inspection of the Figure 2c indicates that response-based clustering may produce data clusters that contain gaps and/or isolated points, as the result of including response information into processing. Neurocomputing models obtained from clusters of data that are disjoint and violate the contiguity condition may be less accurate as there are regions within clusters that simply do not contain enough information to train a network properly. Therefore, it is desirable to modify the shape of a cluster for more amiable for training. In Figure 3 the results of such a modification are shown. The cluster still contains the same amount of data, but the distribution is changed. When a network is trained on this new cluster, not only the accuracy of approximations increases, but also a smaller network with 17 neurons is sufficient to meet training tolerance. Since the number of parameters in a network is $n_h^2 + 5n_h + 1$,where n_h is a number of neurons in each of the equal hidden layer, the size of the network decreases almost 50%, thereby improving the data compression ratio (DCR). The DCR for the shown data cluster is 5:1, but it is important to underscore that

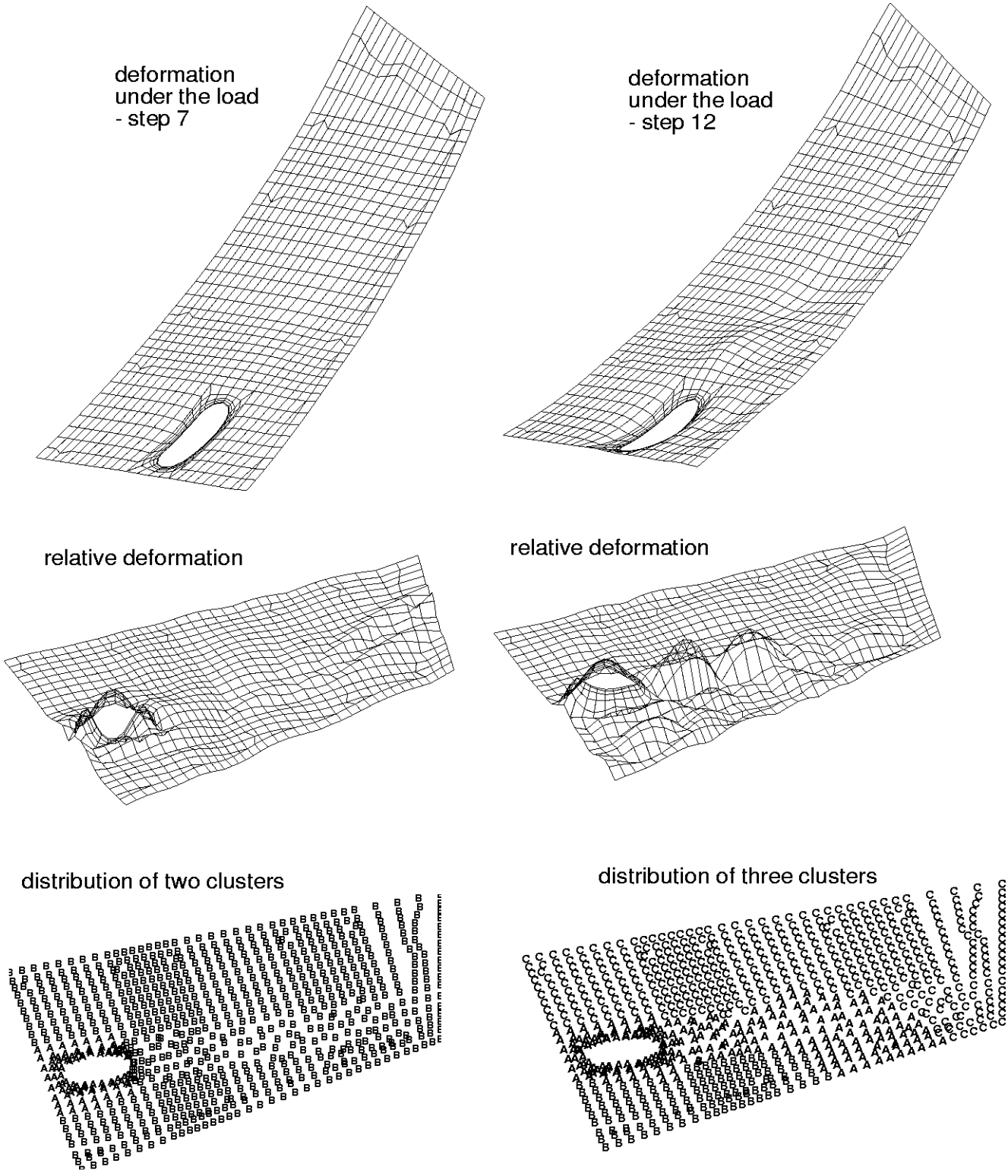


Figure 1. Progressive deformation of a wing skin under a bending loading. When the displacement fields under the load steps 7 and 12 are processed, regions of a different degree of nonlinearity are identified and partitioned by the response-based clustering. Note that the number of clusters increases from two to three as the field is getting more complex. For the step load 12, data in Cluster A are modeled with a RBFN network, and BPN nets are used for the Cluster B and C of data. The average data compression ratio is 15:1.

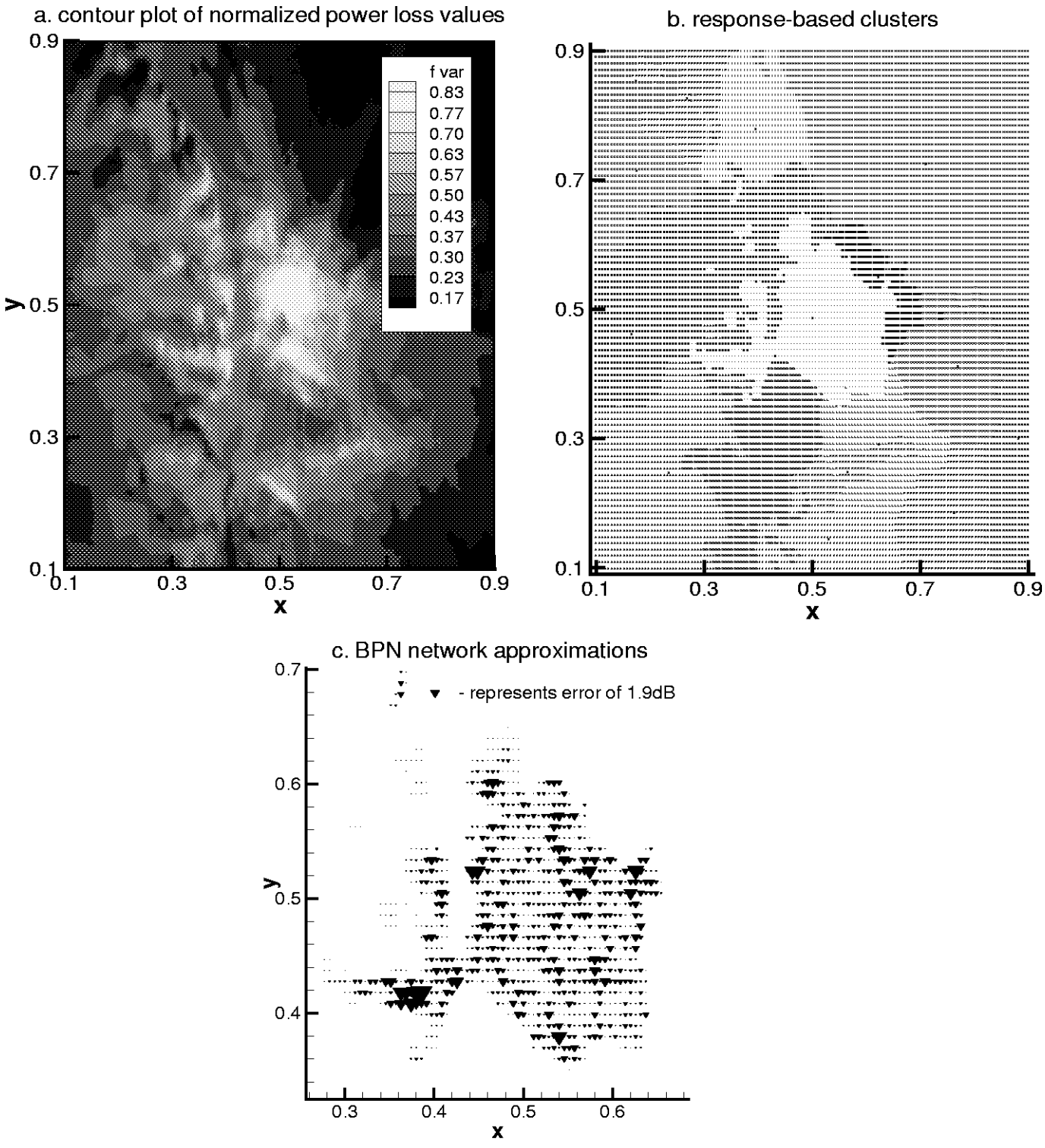
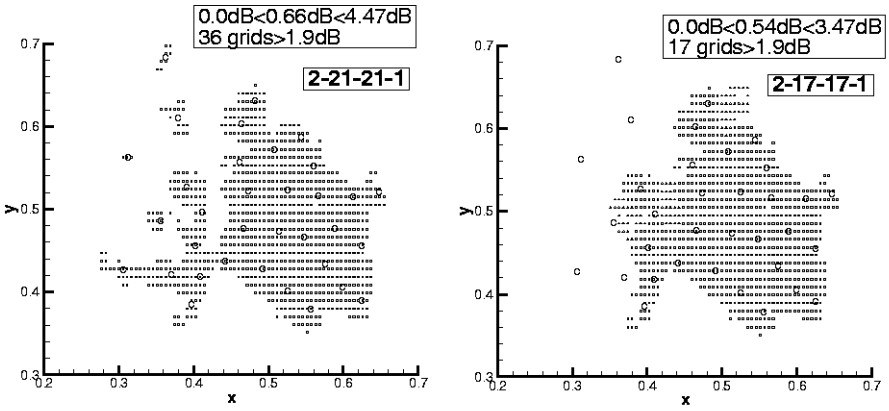


Figure 2. Normalized power loss values, 0.1<L(x,y)<0.9, are shown in (a).
Response-based clusters that take into account the location of a data point and its L value are shown in (b).
When a neural network is fit into a selected cluster comprising 1041 data points (c), the average approximation error is 0.66dB with the maximum of 4.47dB. There are also 36 grid points for which the error exceeds 1.9dB. The network has 21 neurons in each of the two hidden layers.

Figure 1: A smaller network with 17 neurons in each hidden layer yielding better approximations is needed to model data whose distribution is more suitable for training.



the usefulness of neurocomputing models goes beyond a data storage problem. Such models can be used to interpolate power loss values for locations for which such data are not available.

4 Conclusions

Smart neurocomputing modeling appears to be a remedy to problems associated with development of neural network applications. The study shows that complicated fields can be broken up into parts using a response-based clustering technique and then smoothly combined into one piece after a neurocomputing model is obtained for each part separately. Despite partitioning of the whole field, a very smooth transition between adjacent clusters can be achieved with the increased accuracy of approximations and also reduced computational effort to find a suitable neural network for data fit. The clustering technique used in the present study minimizes the number of neural nets needed for modeling, and the decision making process needed for data partitioning is relegated to a programmable unit. However, the use of fuzzy rules encounters problems associated with the adjustment of membership functions, and a unified approach to designing such units seems to be needed to make automation based on a FL inference engine a practical alternative.

References

[1] K. Funahashi. On the approximate realization of continuous mappings using neural networks. *Neural Networks*, 2:183–192, 1989.

[2] J. Moody and C. Darken. Fast-learning in networks of locally-tuned processing units. *Neural Computing*, 1(2):281–294, 1989.

[3] Z. Peter Szewczyk. A knowledge-based adaptive clustering technique for neurocomputing modeling of structures. In *The 39th SDM Conference, AIAA-98-1831*, pages 1153–1161, 1998.