PERGAMON

Computers and Structures 79 (2001) 1541–1552

Computers & Structures

www.elsevier.com/locate/compstruc

Neural network design for engineering applications

M.Y. Rafiq [a,*], G. Bugmann [b], D.J. Easterbrook [a]

[a] School of Civil and Structural Engineering, University of Plymouth, Palace Court, Palace Street, Plymouth PL1 2DE, UK
[b] School of Computing, University of Plymouth, Drake Circus, Plymouth PL4 8AA, UK

Received 2 November 1999; accepted 10 April 2001

Abstract

Computers are an integral part of day to day activities in engineering design and engineers have utilised various applications to assist them improve their design. Although computers are used to model a variety of engineering activities, currently the main focus of computer applications are areas with well defined rules. Activities related to the conceptual stage of the design process are generally untouched by computers.

Artificial neural networks (ANN) have recently been widely used to model some of the human activities in many areas of science and engineering. Early applications of NN in civil engineering occurred the late eighties. One of the distinct characteristics of the ANN is its ability to learn from experience and examples and then to adapt with changing situations. Engineers often deal with incomplete and noisy data, which is one area where NN are most applicable. This is particularly the case at the conceptual stage of the design process.

This paper presents practical guidelines for designing ANN for engineering applications.

A brief introduction to NN is given; major aspects of three types of NN, multi-layer perceptron (MLP), radial basis network (RBF) and normalised RBF (NRBF) are discussed; new methods for selection and normalisation of training data are introduced and a practical example of a reinforced concrete slab design is presented.

Keywords: Neural networks; Multi-layer perceptron (MLP); Radial basis function networks (RBF); Normalised RBF networks (NRBF); Design

1. Introduction

Computers are an integral part of day to day activities in engineering design and engineers have utilised various applications to assist them improve their design. Since the 1970s artificial intelligence (AI) applications have been implemented by engineers to perform specialised design tasks.

Although computers are used to model a variety of engineering activities, at present the main focus of computer applications are areas with well defined rules, such as sophisticated analysis (e.g. finite element analysis), graphical and CAD applications. However where there are no defined rules or heuristics the use of the computer in the decision making process is very limited.

In the past knowledge based expert systems (KBESs) tried to model some of the activities which were related to the decision making process. Due to the restricted scope of these systems, their success was very limited.

Artificial neural networks (ANN) are another AI application which has recently been used widely to model some of the human interesting activities in many areas of science and engineering. Early applications of NN in engineering go back to the late eighties [1].

Engineers often deal with incomplete and noisy data which is one area where NNs are most applicable. NNs are AI tools which are able to learn and generalise from examples and experience to produce meaningful

* Corresponding author. Fax: +44-1752-233-658.
E-mail address: mrafiq@plymouth.ac.uk (M.Y. Rafiq).

0045-7949/01/$ - see front matter
PII: S0045-7949(01)00039-6

solutions to problems. This learning occurs even when the input data contains errors or is incomplete or fuzzy, which is often typical of the design process. These characteristics of ANNs make them a promising candidate for modelling some of the engineering problems.

The paper will endeavour to propose methodologies for designing better network architectures for engineering applications. Practical guidelines on optimum data selection for neural network training will be discussed and methodologies for duration of network training will be highlighted. Three popular types of NN will be compared for speed of training and ease of use. These are the multi-layer perceptron (MLP), radial basis function networks (RBF) and normalised RBF networks (NRBF).

Some practical examples of civil engineering applications of the NN in connection to network stability and network design will be given and appropriate conclusions will be drawn.

2. Introduction to ANN

ANNs are computing systems made up of a number of simple, highly interconnected processing elements, which processes information by their dynamic state response to external inputs [2].

Garrett [3] has given an interesting engineering definition of the ANN as: "A computational mechanism able to acquire, represent and compute mapping from one multivariate space of information to another, given a set of data representing that mapping".

One of the distinct characteristics of the ANN is its ability to learn and generalise from experience and examples and to adapt to changing situations. ANNs are able to map causal models (i.e. mapping from cause to effect for estimation and prediction) and inverse mapping (i.e. mapping from effect to possible cause) [3].

ANNs are models of real world problems which map from a set of given patterns (input patterns) to an associated set of known values (target output).

In simple terms a NN tries to imitate some of the learning activities of the human brain. ANNs are much simpler than the human brain. They comprise fewer components and operate in a manner that is greatly abstract.

The training process in the MLP network involves presenting a set of examples (input patterns) with known outputs (target output). The system adjusts the weights of the internal connections to minimise errors between the network output and target output.

After the NN is satisfactorily trained and tested, it is able to generalise rules and will be able to respond to unseen input data to predict required output, within the domain covered by the training examples.

ANNs are applied to perform many different tasks, such as filtering, identification, control, prediction, etc. Among the most common tasks of ANNs are:

Classification: Where the input to the network is an encoded description of an object to be recognised, and the output is generally a binary identifier of the class of the presented object. The ANN can learn linear or non-linear class boundaries from examples.

Function approximation: Where the ANN performs mapping from the input space to the output space. Vectors in both spaces generally use real number components. NNs are then function approximation tools that replace lengthy procedural algorithms.

2.1. Differences between ANNs and traditional computing

Traditional computing solutions are based on predefined rules or equations which, gives a clear definition of the problem. The program explicitly defines step-by-step tasks to be performed to achieve the required results. This may be an ideal way of modelling problems when the rules related to a problem are perfectly known. There are many practical cases for which the rules are either not known or they are extremely difficult to discover. Such problems cannot be modelled using traditional computing methods.

KBESs, which are an AI rule based systems, are not very different in their approach to problem solving from traditional computing techniques. They are based on problem specific rules, facts and inferences. KBESs are generally built from a set of predefined rules in which the outcome of every possible solution is defined in terms of rules such as if ... then etc. In KBESs the rules are defined in a higher level than the traditional computing approach. One of the advantages of the KBESs is that the rules defined within these systems are formally proved and tested. A major disadvantage of these systems is the extremely difficult and sometimes impossible task of extracting rules from the human experts. This has been a stumbling block for KBESs from the outset. Another disadvantage of these systems is the limitation in their scope since they are problem specific.

ANN are AI tools which are extremely useful in situations for which rules are either not known or they are very difficult to discover. Some of the major attributes of ANN can be listed as:

- NNs can learn and generalise from examples to produce meaningful solutions to problems.
- They can perfectly cope with situations where the input data to the network is fuzzy, discontinuous or is incomplete.
- NNs are able to adapt solutions over time and to compensate for changing circumstances.
- Data presented for training NNs can be theoretical data, experimental data, empirical data based on

good and reliable past experience or a combination of these. Training data can be evaluated, verified or modified by the human expert to inject human intelligence and intuition. These criteria make ANNs a promising tool for modelling activities related to design.

NNs might be seen as a good approximation to a perfect rule or function based approach. While they lack the exact precision and formality of the traditional and KBES approaches, they are powerful tools which can produce near perfect approximations to problems for which there is insufficient knowledge to allow formal programmed solutions. A NN is a generalisable model, based on the experience of a set of training data and consequently contains no explicit rules [4].

ANNs are very useful tools for application where the precision of traditional techniques is a hindrance; applications which require pattern recognition or simulation of a physical system too complex to model with rules [4].

One of the criticisms of ANNs is that, unlike KBESs, NNs are not able to provide explanations and justifications for their answers. One of the reasons for this is that NNs are not based on rules which backtrack the way the system has obtained its solution. Another criticism of ANNs is their lack of ability to extrapolate solutions for problems outside the network training domain. Research on both areas is in progress.

2.2. Building NNs

All three types of networks (MLP, RBF and NRBF) discussed in this paper use a feed-forward network architecture, as shown in Fig. 1, consisting of an input layer, one or more hidden layer(s) and an output layer. Layers are fully interconnected, as shown by arrows. Initially a random weight (usually in the range of -1 to $+1$) is assigned to each connection. These weights are then adjusted as learning progresses.

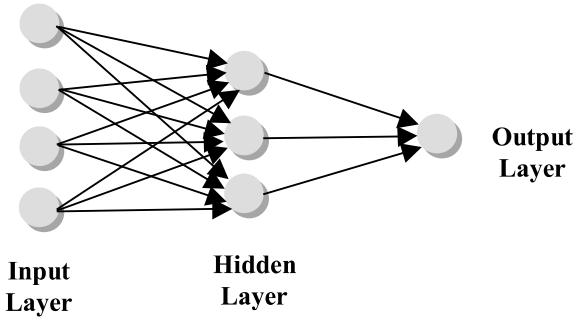

Fig. 1. Typical neural network model.

The function of an input layer is to receive input from the outside world. The function of the output layer is to output the result of the NN prediction to the outside world. The hidden layer links the input layer to the output layer. The function of the hidden layer is to extract and remember useful features and subfeatures from the input patterns to predict the outcome of the network (values of the output layer). The main difference between the network types lies in the type of activation function used by the hidden neurones. In MLPs, a common type of activation function used by the hidden neurones has a sigmoid transfer function $S_i(\vec{x})$:

$$S_i(\vec{x}) = \frac{1}{1+\exp(-z_i)}$$

where

$$z_i = \sum_{j=1}^{n} w_{ij} x_j$$

Here x_j is the activity of the jth input neurone and $S_i(\vec{x})$ is the activity of the ith hidden layer neurone. This sigmoid function divides a high-dimensional input space into two halves, with a high output in one half and a low output in the other, as illustrated in Fig. 2A.

In RBF networks, hidden neurones usually have a GAUSSIAN radial basis function $G_i(\vec{x})$:

$$G_i(\vec{x}) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - w_{ij})^2}{2\sigma^2}\right)$$

Here w_{ij} indicates the centre of the receptive field of the neurone and σ is its half-width. This function is selective to a small portion of the input space, as illustrated in Fig. 2B.

Normalised RBF networks use hidden neurones whose activity $Gn_i(\vec{x})$ is normalised by the total activity of all m hidden neurones:

$$Gn_i(\vec{x}) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - w_{ij})^2}{2\sigma^2}\right) \Big/ \sum_{k=1}^{m} Gn_k(\vec{x})$$

This function generates a Voronoi partition of the input space [14], producing a constant output in a neighbourhood limited by neighbouring neurones, as illustrated in one dimension in Fig. 3C. Its role is to detect inputs in a given extended areas of the input space.

Neurones in the output layer usually have linear transfer functions:

$$\mathrm{Out}_i = \sum_{k=1}^{m} w_{ik} x_k$$

where x_k is the activity of the kth hidden layer neurone and Out_i is the activity of the ith output neurone. For

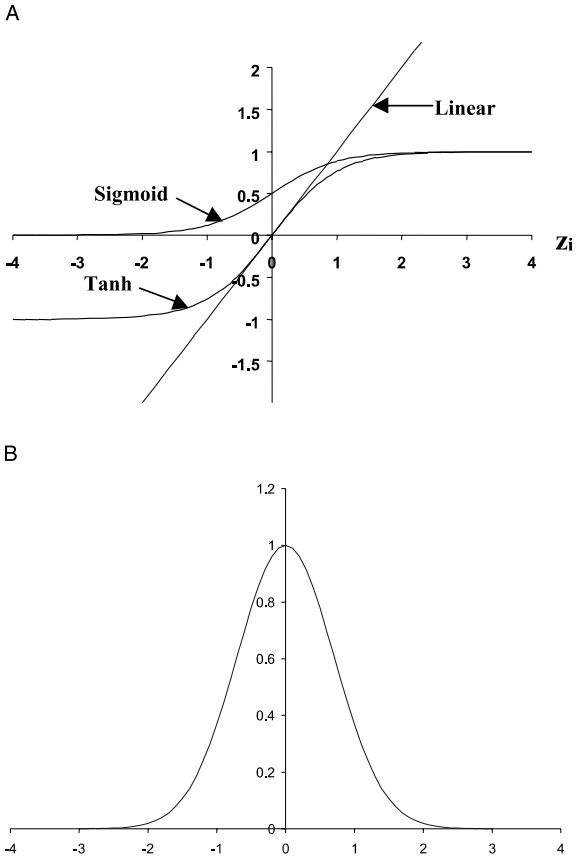

Fig. 2. Transfer functions of hidden neurones in MLP nets (A) and RBF nets (B). The transfer functions of hidden nodes in MLP nets are usually tanh or sigmoid. The output layer neurones are sometimes linear. Note: Single NRBF neurones produce a constant output unless there is competition from other hidden neurones.

classification tasks, sigmoid transfer functions may be used.

In a feed-forward operation, represented by Fig. 1, the flow of information is from left to right. The first step is to calculate the output of each hidden layer neurone using the appropriate transfer function. The next step is to calculate the activity of neurones in the output layer. During learning, an error is computed by comparing the actual output with the desired output, as given in the training data.

Learning in MLP networks (also called "back-propagation networks") involves the back-propagation of the network error and adjusting the interconnecting weights for each layer. This step is an essential part of the network learning process and is performed by the learning algorithm. The choice of the number of hidden neurones is left to the user, as described in Section 2.3.1.

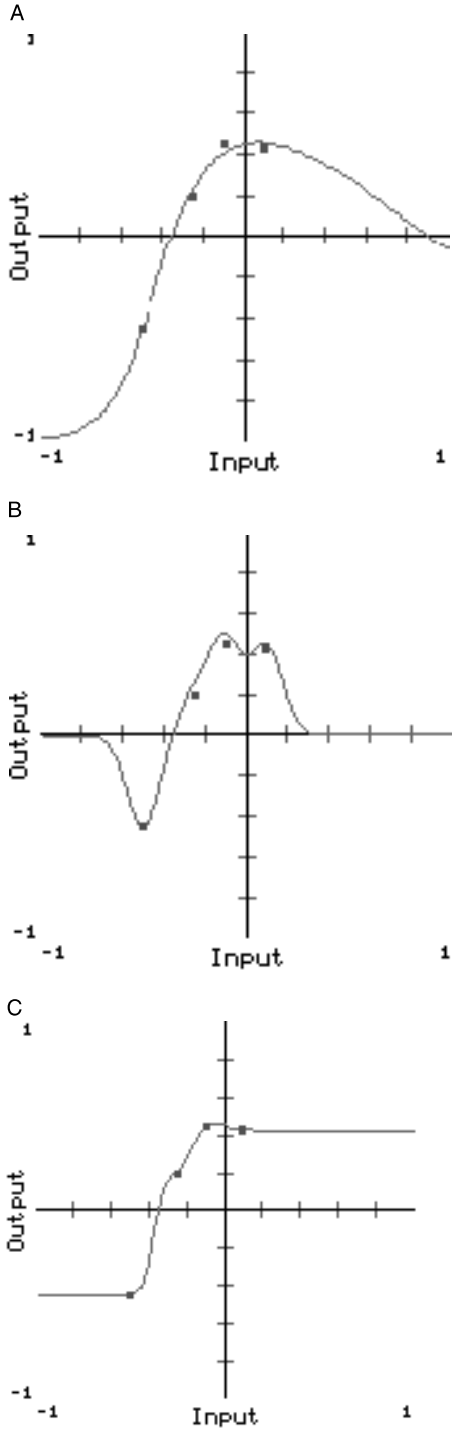

Fig. 3. One-dimensional example illustrating the functions resulting from training three types of neural networks to mapping the indicated four training data. (A) MLP, (B) RBF, (C) NRBF. The MLP has 10 hidden neurones. Both RBF and NRBF nets use four hidden layer neurones, one centred on each data point.

In RBF and NRBF nets, learning is generally a two stage process involving first the recruitment of the necessary number of hidden neurones, to reflect the distribution of training data in the input space. Then the connections between hidden and output layers are adjusted for optimal fit to the data. This is done in a supervised manner [15]. These two operations are automated in the learning algorithm. The size of the receptive fields of hidden neurones needs to be adjusted by the user for optimal generalisation, although best-guess heuristics exist [14].

2.3. Selecting a topology for a MLP network

Every stage of any NN project requires a little trial and error to establish a suitable and stable network for the project. Trial and error may be extended to building several networks, stopping and testing the network at different stages of learning and initialising the network with different random weights. Each network must be tested, analysed and the most appropriate network must be chosen for a particular project.

Before deciding on the topology of the network, it is important to select the required number of input and output parameters. In engineering problems the number of input and output parameters are generally determined by design requirements. The number of input parameters determines the spatial dimensions of the network and the number of output parameters determines the number of solution surfaces generated by the network [5,6]. Obviously too many input and output parameters will drastically slow down the learning process and too few sets of training data can provide insufficient information about localised features and cause the network to fail to generalise and the network response to unseen data will be poor. It is therefore essential to optimise the number of input and output parameters as much as possible.

For selecting input and output parameters Stein [7] proposed a statistical method. Using this method eliminates some of the less important parameters in the training process.

Gunaratnam [8] has proposed dimensional analysis (which leads to a reduction in the space dimensionality in which the domain relationship is defined) and dimensionless forms of parameter representation. In this approach it is possible to combine dimensionless products and reduce the number of parameters required to describe relationships.

The hidden layer allows a network to model complicated mapping and capture features and subfeatures within a training domain [9]. A single layer with an optimum number of neurones will be sufficient for modelling many practical problems. It has been suggested that for continuous functions a single hidden layer with a sufficient number of neurones will be suitable whilst a second layer may be needed for discontinuous functions [10]. The number of neurones in a hidden layer drastically affects the outcome of the network training. If too few neurones are included then the network may not be able to learn properly and the response of the network to unseen data will be poor. On the other hand if too many neurones are included, the solution surface produced by the network may develop false features at points between training patterns [5,6].

Generally, there is a trade-off between building a network which generalises well and is robust, and one which is more accurate but more brittle i.e. one in which the network learns the training data well, but its performance is poor when tested with unseen data. More complex, accurate, yet brittle networks require more data patterns for training. A more general model describes a smoother curve through the training data points, missing some, but resisting the effect of noise and peculiarities which might be present [4]. The number of neurones greatly affects the generalisation characteristics of a neural network.

There is no general rule for selecting the number of neurones in a hidden layer. The choice of hidden layer size is mainly problem specific and to some extent depends on the number and the quality of training patterns. The number of neurones in a neural network must be sufficient for the correct modelling of the problem, but it should be sufficiently low to ensure generalisation [4]. Too large a number of hidden neurones would encourage over-fitting, i.e. modelling the noise in the data or modelling the data with unnecessarily complex functions. In general, a smooth interpolation between training data is sought.

Some authors relate the number of neurones to the number of input and output variables and the number of training patterns [4,11–13], but these rules cannot be generalised. Some researchers have mentioned, that the upper bound for the required number of neurones in the hidden layer should be one greater than twice the number of input unites. This rule does not guarantee generalisation of the network.

To be able to design a stable MLP network, it would be more appropriate to carry out a parametric study by changing the number of neurones in the hidden layer in order to test the stability of the network. Fig. 4A shows the performance of the MLP network with various numbers of neurones in the hidden layer. In Fig. 4A the performance of MLP networks with various hidden neurones is measured by the RMS Errors, noted during the training of each network. RMS error is the vertical axis of this graph. Fig. 4A represents the architecture of a MLP network for optimum design of a continuous reinforced concrete flanged beam. In this network, as can be seen from Fig. 4A, 12 neurones in the hidden layer results in a stable and optimum network.

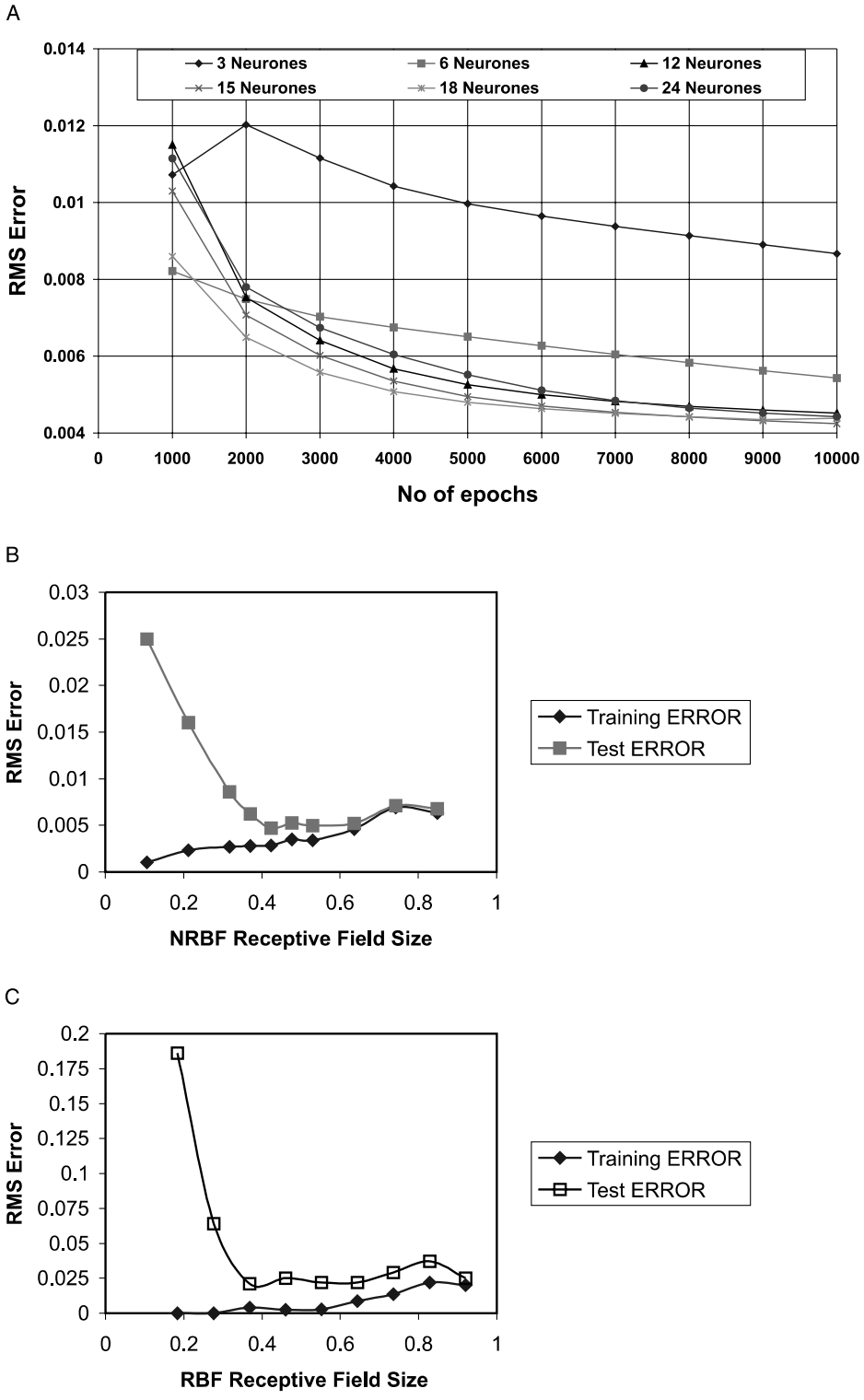

Fig. 4. (A) Learning curves for MLP networks with various number of neurones m in the hidden layer. For $m \geqslant 12$ the network is able to fit the data perfectly. (B) Training and test error of an NRBF net as a function of the receptive field size. Training data from Table 1. (C) Training and test error of an RBF net as a function of the receptive field size. Training data from Table 1.

2.3.1. Hidden layer in RBF and NRBF networks

For RBF and NRBF nets, the optimal number of hidden neurones and the optimal size of the receptive field need to be determined jointly. This is a more complicated task than for MLPs. As a rule of thumb, the size of receptive fields should be such as to be able to bridge the gaps between data in the input space, and only one hidden neurone should be used for clusters of data of the size of the receptive field. Hence, once the size of the receptive field has been chosen, the neurone recruitment process can be automated. Bugmann [14], found that the optimal sizes of the receptive field to be half of the mean distance between data. However, searching around this initial value is still necessary for optimal performance. In our experience, RBF nets are more difficult to optimise than NRBF nets. Fig. 4B and C shows a similar optimal receptive fields $\sigma = 0.4$ for both networks, but a larger error for RBF nets.

2.4. Pre-processing training data

NNs have been shown to be able to process data from a wide variety of sources. They are however, only able to process the data in a certain format. Furthermore the way the data is presented to the network affects the learning of the network. Therefore, a certain amount of data processing is required before presenting the training patterns to the network.

For better and smoother network generalisation, it may sometimes be necessary to remove some of the data outliers. Statistical analysis shows that 95% of a normally distributed data lies within two standard deviations and 99% within three standard deviations [4]. Removing data outside these ranges will greatly improve network training, provided that these data are extreme and uncharacteristic of the problem domain.

Dimensionality reduction is another area, which reduces the number of patterns required for network training and hence network complexity. Using statistical analysis [7] and dimensional analysis [8] and combining the number of variables to a smaller set of input variables are useful methods for optimising the number of input and output parameters. A method suggested by Swingler [4] is sometimes useful for dimensionality reduction. This approach suggests using a NN with too many inputs, and a small number of hidden neurones. If the weights in the network are initialised with small, random values, then there will be little change in the weights of the less important variables from their original values. These variables can then be removed if necessary.

Data scaling is another essential step for network training. One of the reasons for pre-processing the output data is that a sigmoidal transfer function is usually used within the network. Upper and lower limits of output from a sigmoid transfer function are generally 1 and 0 respectively. Scaling of the inputs to the range $[-1,+1]$ greatly improves the learning speed, as these values fall in the region of the sigmoid transfer function where the output is most sensitive to variations of the input values. It is therefore recommended to normalise the input and output data before presenting them to the network. Various normalisation or scaling strategies have been proposed [4,7]. Scaling data can be linear or non-linear, depending on the distribution of the data. Most common functions are linear and logarithmic functions [4].

A simple linear normalisation function within the values of zero to one is:

$$S = (V - V_{\min})/(V_{\max} - V_{\min})$$

where S is the normalised value of variable V, $V_{\min}$ and $V_{\max}$ are variable minimum and maximum values respectively.

Using the above normalisation process some data will have values equal or very close to 1 or 0. From experience the authors have found that a better fit will be achieved if the normalised data is kept away from the sigmoid extreme boundaries of 1 and 0.

A different strategy was adopted for normalising the output data by the authors, as represented by the following relationship:

$$A = (V/10^n)^{1/2} + C$$

C is a constant, usually between -0.25 and 0.25 to ensure that the values are in the range of 0.2 and 0.8. n is the number of digits in the integer part of the variable V (e.g. for $V = 234.5$, $n = 3$). The use of the square root function reduces the effect of outliers. Results obtained from this normalisation were satisfactory for a number of problems.

2.5. Data selection for the neural network training

In order to ensure that the network has properly mapped input training data to the target output, it is essential that the set of patterns presented to the network is appropriately selected to cover a good sample of the training domain. A well trained network is one which is able to respond to any unseen pattern within an appropriate domain. At present NNs are not good at extrapolating information outside the training domain.

The selection of an adequate number of training patterns is therefore, an extremely important issue. There are no acceptable generalised rules to determine the size of the training data for suitable training. Patterns chosen for training must cover upper and lower boundaries and a sufficient number of samples representing particular features over the entire training domain.

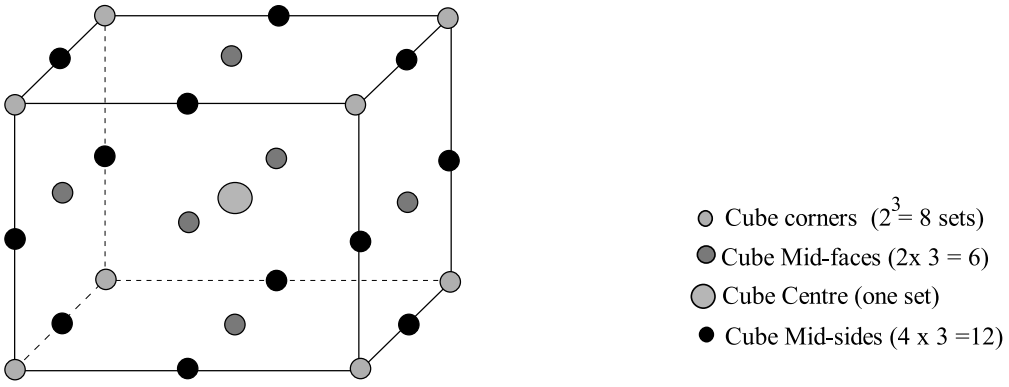

Fig. 5. Hypercube showing data points for the NN training.

Jenkins [12,13] has used a so-called 'hypercube' concept for selecting training patterns for his grillage analysis model. He found that this method gives satisfactory results for small scale problems. Jenkins argues that data representing the corner of the cube, mid-sides of the cube faces and a number of random data from within the cube will be sufficient for training the network. Jenkins has shown that data representing only the mid-faces, upper bounds and mid-points of the cube will be sufficient for suitable training of the network. Jenkins has used NNs generally for elastic structural analysis problems with clear linear elastic relationships.

The authors have modified Jenkins' method by adding data for the mid-sides of the cube into the training patterns (as indicated by black dots in Fig. 5) and used it for non-linear problems which has proved to be satisfactory for modelling reinforced concrete slabs and flanged beams to predict optimum design parameters using NNs. Fig. 5 shows details of the proposed hypercube data points for NN training.

2.6. Duration of a NN training

2.6.1. MLP networks

Presenting the entire set of training patterns to the network is called an epoch. The number of 'epochs' (number of times that the whole set of patterns is presented to the network) affects the performance of the network. This number depends on many factors, of which the following are more important:

- number of training data,
- number of hidden layers,
- number of neurones in hidden layers,
- number of dependent output parameters.

If the network is trained for too long, it begins to overfit, e.g. model noise in the training data, or develop an unnecessarily complex function to fit the data. As a result the network will not perform well to unseen patterns. If the training is not carried out long enough, the network will not learn all features and subfeatures and hence the network performance will be unsatisfactory. Hence, in MLPs, a limitation of the training time has a similar role as the limitation of the number of hidden neurones.

The question is how long to continue the network training and what is the optimum limit? This problem is network dependent.

There are a number of reasons for not being able to define a general rule about the number of epochs which the training patterns should be shown to the network [4].

1. The back-propagation algorithm does not guarantee a convergence for MLP NNs. The network may fail to reach a suitable weight configuration which minimises the network error.
2. The lowest possible error for the noisy data is not zero.
3. MLP networks are designed to minimise training error. They may not necessarily minimise the generalisation error.

For the above reasons, it is practically necessary to carry out a parametric study to determine the optimum number of epochs necessary for network training.

Once a suitable network architecture is found and the training patterns are selected and normalised, the network training can be started. The result of the training can be saved at various intervals for network testing purposes. Testing of trained networks at regular intervals during training aims to avoid the important problem of overfitting and hence produce a good generalisation network. The test data set consists of a set of patterns unseen by the ANN during training.

Using the results of testing, the following criteria determine if the network has been sufficiently trained:

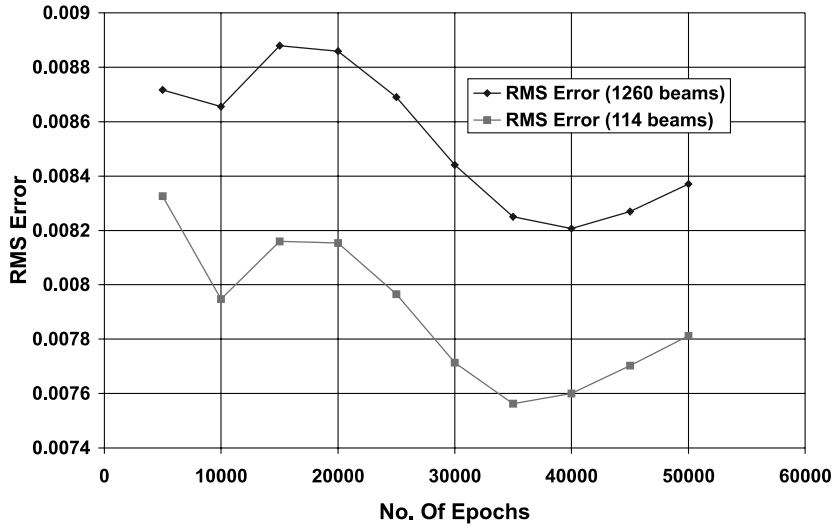

Fig. 6. Root mean sum squared error (RMS) vs no. of epochs (MLP).

- The average training error level has reached a predetermined target value.
- The average error on testing data does not fall, or it falls very slowly.
- The average error on testing data starts to rise (see Fig. 6).

Fig. 6 shows result of a parametric study carried out by the authors on a MLP network. From Fig. 6 it is clear that, for the example used, the optimum number of epochs for both sets of tests lies between 30 000 and 40 000 epochs.

Using this method of investigation, the performance of the network and its optimum training length can easily be determined during the training process.

2.6.2. RBF/NRBF networks

RBF/NRBF nets learn very rapidly, usually in less than 50 epochs. The problem of over-fitting is addressed solely by selecting an appropriate size of receptive field. This provides a good control of the complexity of the fitting surface. Using test data enables, by cross validation, the determination of the optimal receptive field size. Fig. 4B and C exemplifies such procedures.

2.7. Training modes

Training a NN involves gradual reduction of the error between NN output and the target output. Generally, there are two different modes of training NN: batch mode and pattern mode. In a batch mode, when an epoch is completed (i.e. when the entire set of training data is presented to the network) a single average error is calculated and the weights in the network are adjusted according to that error. In a pattern mode, the error is calculated after each pattern is presented to the network, and network weights are adjusted. Choosing between the two modes is generally, problem specific. Swingler [4] has indicated that the following points should be considered when choosing a training mode:

- Batch mode requires less weight update and hence it is faster to train.
- Batch mode provides a more accurate measurement of the required weight changes.
- Batch mode is more likely, than pattern mode, to become tapped in local optima.

Our experience has also shown that batch modes are much sensitive to initial weights.

It would be advisable to train the network using batch mode to start with and test and analyse the network output. If the level of error after testing the network with unseen data was not satisfactory then a pattern mode should be used.

2.8. Speeding up the training process in MLPs

If the NN program is written by the user or the user has access to the NN code, then it would be advantageous to implement techniques to speed up the training process of the network.

A practical way of achieving this goal is to associate the learning rate with the changes in the network weights (adjusting the learning rate during the training process).

The following observations should help speeding up the training process [4]:

- If the change in network weights is consistently in the same direction, then the learning rate can be increased.
- If the network weights change in alternating directions, then the learning rate should be decreased.
- The magnitude of change in the learning rate is dependent of the changes in the network weights.

2.9. Checking network performance

After the training is completed, usually, the network error is minimised and the network output shows reasonable similarities with the target output. This is not unusual. To make sure that the network training has been satisfactorily completed and the network is capable of generalisation, a set of unseen patterns must be selected and the network should be tested using these patterns.

With the pattern selection methodologies presented in this paper, it is relatively easy to prepare a reasonable size data at the outset. Using the hypercube data selection method, only a small portion of data will be selected for training purposes. The rest of the data can be used for network validation and testing.

3. Case studies

In order to demonstrate the usefulness of the methodologies presented in this paper, a reinforced concrete slab example is given below. The aim of this example is to use NNs to predict optimum design parameters for the slab. In this paper only one design parameter (the optimum depth for the slab) is presented. The slab depth is the most important parameter which influences many other design parameters and the final cost of the overall structure.

For comparison purposes, MLP, RBF and NRBS networks were trained and tested in this study, using the same training and testing patterns. The results generated by the NRBF were very similar to those of the MLP, and the results of the RBF were relatively poorer than that of the MLP and NRBF. In this section only result generated using a MLP network is presented.

3.1. Reinforced concrete slab example

For preparation of training patterns, the following design parameters were considered:

- slab short span (4–12 m) resulting in nine different spans,
- slab aspect ratio (1–2) resulting in six different long spans for each short span,
- applied design load (imposed load of 2.5–7.5 kN/m^2) resulting in 11 different loads.

The above combination results in 594 different designs.

An initial investigation was also conducted to evaluate which parameters affect the practical design of reinforced concrete slab. Parameter ranges were selected carefully to cover the majority of the practical range of applications. A linear optimisation program was used to obtain the optimum depth and the required reinforcement steel weights for each design.

For the calculation of the optimum slab depth the following three criteria were considered:

- minimum cost,
- Serviceability requirements (span/effective depth limits specified in the British Concrete Code BS8110),
- maximum steel limit (4% of the cross-section as specified in BS 8110) for buildibility and ease of construction purposes.

3.2. Data selection

Once a decision was made about input and output parameters and appropriate set designs have been generated, the next stage was to decide upon the size of hidden layer to set up the architecture of the NN. It was decided to use a single hidden layer. A parametric study, similar to that shown in Fig. 3 was conducted. The result of this study is presented in Fig. 4A (MLP). One hidden layer with 15 neurones was then selected for this study.

After the architecture of the network was completed, it was time to select the training patterns from the pool of design data generated for training and testing purposes. A 'hypercube' approach, discussed in the previous section was used to select patterns for the NN training. From the set of 594 design data, the following set of data was selected for network training:

- hypercube corners (this includes upper and lower limits) $2^3 = 8$ sets,
- hypercube mid-faces $2 \times 3 = 6$ sets,
- hypercube centre = 1 set,
- hypercube mid-point of all sides $4 \times 3 = 12$ sets.

Total number of data needed for the network was $(8 + 6 + 1 + 12 = 27)$ sets. This data is presented in Table 1.

Results of the network training are presented in Fig. 7. Fig. 7 shows a perfect match between target output and the NN output. The maximum error was less than 2% for any point.

Table 1
Training data

Position	Short span (m)	Long span (m)	Design load (kN/m^2)
Corners	4	4	10.636
	4	8	10.636
	12	12	19.54
	12	24	19.54
	4	4	19.14
	4	8	19.14
	12	12	28.548
	2	24	28.548
Mid faces	8	12	15.004
	8	8	19.34
	8	12	23.676
	8	16	19.34
	4	6	14.972
	12	18	24.044
Centre	8	12	19.34
Mid sides	4	6	10.636
	4	4	14.972
	4	6	19.14
	4	8	14.972
	12	18	19.54
	12	12	24.044
	12	18	28.548
	12	24	24.044
	8	8	15.004
	8	8	23.676
	8	16	23.676
	8	16	15.004

Fig. 8 shows the result of the MLP network test on unseen data. Fig. 8 shows the network response to 83 sets of unseen data. The maximum error in this case was about 3%, which is extremely good. A test was carried out over the entire range 594 data. The maximum error for 594 data was a little over 4%. It is, therefore, concluded that the network is capable of generalisation and the training is satisfactory.

4. Conclusions

ANNs are capable of learning and generalising from examples and experience to produce meaningful solutions to problems even when input data contain errors and are incomplete. This makes ANNs a powerful tool for solving some of the complicated engineering problems.

The paper has highlighted methodologies for designing better network architecture for engineering applications.

Three types of NNs were compared. The MLP and the NRBF performed equally well. The RBF showed a poorer performance. The NRBF has the advantage of a faster training.

Practical guidelines on data selection NN training, using a modified and extended version of the Jenkins 'hypercube', was discussed and tested. It was shown that using this method can drastically reduce the size of the training data. Because of the good sampling nature of the data selected by this method, the network generalisation capability is enhanced and the training time is considerably reduced.

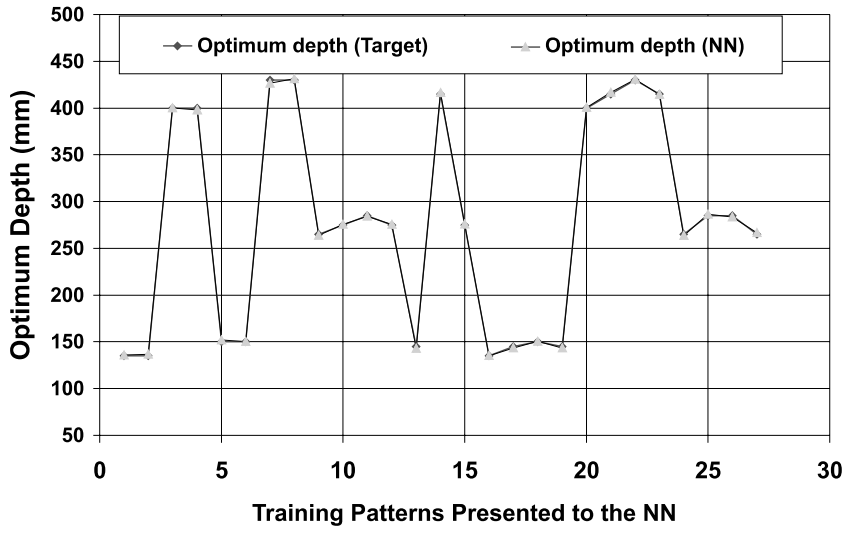

Fig. 7. Neural network training (MLP).

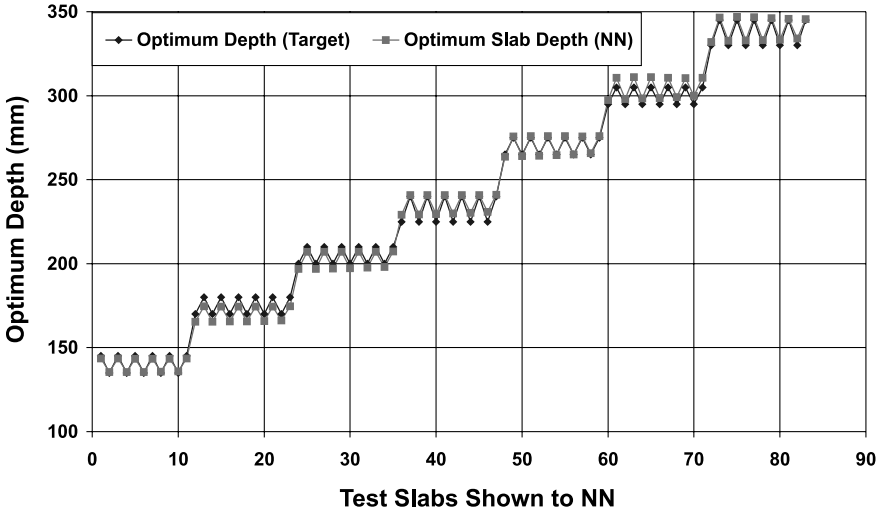

Fig. 8. NN result on 83 test slabs.

The duration of network training and methodologies for speeding up the training process were highlighted and an example of the application of the MLP in a civil engineering problem was presented.

References

[1] Flood I. A neural network approach to the sequencing of construction tasks. Proceedings of the Sixth International Symposium on Automation and Robotics in Construction, Construction Industry Institute, Austin, TX, 1989. p. 204–11.

[2] Caudill M. Neural networks primer, Part I. AI Expert, December 1987. p. 46–52.

[3] Garrett JH. Where and why artificial neural networks are applicable in civil engineering. ASCE, J Comp Civ Engng [special issue] 1994;8(2):129–30.

[4] Swingler K. Applying neural networks a practical guide. New York: Academic Press; 1996.

[5] Flood I, Kartam N. Neural network in civil engineering I: Principles and understandings. ASCE, J Comput Civ Engng 1994;8(2).

[6] Flood I, Kartam N. Neural network in civil engineering II: Systems and applications. ASCE, J Comput Civ Engng 1994;8(2).

[7] Stein R. Selecting data for neural networks. AI Expert, No. 2, 1994. p. 42–7.

[8] Gunaratnam DJ, Gero JS. Effect of representation on the performance of neural networks in structural engineering applications. Microcomput Civ Engng 1994;9:97–108.

[9] Garrett JH, Gunaratnam DJ, Ivezic N. In: Kartam N, Flood I, editors. Artificial neural networks for civil engineers: fundamentals and applications. ASCE, 1997. p. 1–17.

[10] Goh AT. Some civil engineering applications of neural networks. Proc Instn Civ Engrs Structs Buildings 1994;104:463–9.

[11] Rogers JL. Simulating structural analysis with neural networks. ASCE, J Comput Civ Engng 1994;8(2):252–65.

[12] Jenkins WM. An introduction to neural computing for the structural engineer. The Struct Engng 1997;75(3):38–41.

[13] Jenkins WM. Approximate analysis of structural grillages using a neural network. Proc Instn Civil Engrs Structs Buildings 1997;122:355–63.

[14] Bugmann G. Normalized radial basis function networks. Neurocomputing [special issue on Radial Basis Function Networks] 1998;20:97–110.

[15] Moody J, Darken CJ. Fast learning in networks of locally-tuned processing units. Neur Computat 1989;1:281–94.