*Journal of Structural Engineering and Geotechnics,*
*1 (1), 37-48, Spring 2011*

# Comparison Study on Neural Networks in Damage Detection of Steel Truss Bridge

Hassan Aghabarati[a,*],Mohsen Tehranizadeh[b]

[a] *Department of Civil and Architectural Engineering, Islamic Azad University, Qazvin Branch, Iran*

[b] *Department of Civil and Environmental Engineering, Amirkabir University of Technology, Tehran, Iran*

Received 15 Dec. 2010; accepted 10 Feb. 2011

**Abstract**

This paper presents the application of three main Artificial Neural Networks (ANNs) in damage detection of steel bridges. This method has the ability to indicate damage in structural elements due to a localized change of stiffness called damage zone. The changes in structural response is used to identify the states of structural damage. To circumvent the difficulty arising from the non-linear nature of the inverse problem, three neural networks, Multi-Layer Perceptron Neural Network (MLPNN), Radial Basis Function Neural Network (RBFNN) and General Regression Neural Network (GRNN) are employed to simulate damage states of steel bridges. It was observed that the performance of all three networks is well and they have good agreement with actual results performed with Finite Element analysis. The efficiency of GRNN in structural identification is so good, although RBFNN has results close to GRNN and MLPNN results are satisfactory. All networks have good results while there is a little damage in structural members. Generally, results would have more error when damages in structural members extend. The engineering importance of the whole exercise can be appreciated once we realize that the measured input at only a few locations in the structure is needed in the identification process using neural networks.

*Keywords*: Damage Detection; System Identification; Artificial Neural Networks; Finite Elements; Steel Bridges

## 1. Introduction

In civil engineering practice, existing structures are inspected by experienced engineers who determine the location of damage zone in the structure and the extent of the damage. Generally, it is believed that System Identification Technique (SIT) [1] can be extended to structures for systematic damage detection and evaluation. Structural identification is a process for constructing a mathematical description of a physical system when both the input and the corresponding output are known. When a structure undergoes various degrees of damage, certain characteristics have been found to undergo changes. In the order to identify those changes, during inspection, a sequence of tests may be conducted and the resulting data such as load, displacement, strains, acceleration etc., can be measured. From such data, mechanical properties, such as stiffness/strength, and dynamic characteristics, such as natural frequency and damping ratio, can be estimated. All these are dealt with by system identification techniques. Structural identification can be done both under static [2] and dynamic conditions. Parameter identification problems lack unique solution and are, thus, often formulated in an optimization framework in which the parameters of the assumed model are found within the predefined space of variables to minimize the difference between measured and computed responses in some norms. These techniques have been demonstrated in the past in structural damage detection using conventional computing techniques. The algorithm adopted is generally complex and is not appropriate for the situations where the measured data are imprecise or inadequate [3]. The recent emergence of artificial neural networks can be explored as an alternative tool for identification exercises in such situations. Many good reviews on the neural networks paradigms are available in the literature. Here, only a brief conceptualization of neural networks and their computational counterpart is given [4].

Ghaboussi et al. and Wu et al. demonstrated the use of back-propagation algorithm in structural application [4,5].

*corresponding Author Email: aghabarati@qiau.ac.ir

Hajela and Berke implemented neural networks paradigm in automated structural design [6]. They examined two distinct architectures, namely conventional layered architecture with input-output layers and hidden layers and modified flat network termed as function link nets. They adopted supervised learning for both types of networks with a “back-propagation” algorithm.

Wu et al. used back-propagation neural networks architecture with single hidden layer to simulate damage states in a three-story frame [4]. The structure was subjected to earthquake base acceleration and the transient response was computed in time domain. The Fourier spectra of the computed relative acceleration time histories of the top floor for various members were used in training the networks. The member damaged was defined as a reduced in member stiffness. In the course of training the network, they observed that if the hidden layer was too small, the network would not converge. On the other hand, if the network was too large, it would not converge either. Obviously, the performance of the network which depends on the hidden layer is a problem dependent and must be investigated further.

Szewczyk and Hajela considered the damage detection of structures as an inverse problem [7]. They modeled the damage as a reduction in the stiffness of structural elements which was associated with observed static displacement under prescribed loads. To generate this reverse mapping between stiffness of individual members of structure and the global static displacement, an improved counter propagation neural network was utilized. They performed simulation of frame structures with nine bending elements and degrees of freedom displacement and rotation at each node. The size of architecture was governed by control parameter, also called as resolution parameter. They observed, from the exercises on the frame structure, that the network performance was generally precise with gradual deterioration in presence of noisy and incomplete measurements. They also concluded that the resolution parameter plays an important role in designing the size of network and is highly problem dependent.

Elkordy et al. adopted back-propagation neural networks to model damage states of five-story steel frames. Three networks were used [8]. These networks trained with analytically generated states of damage, were used to diagnose damage states obtained experimentally from a series of shaking tests. Although the results were promising, they concluded that the relation between the number of damage patterns required for training the network to perform satisfactorily and the degree of simplification of the model should be investigated further.

Adeli and Yeh presented ANN model of machine learning in engineering design [9]. They demonstrated the application of ANN model in the detection of structural damage. The applications of NN models in structural engineering have also been done by researchers such as Hajela and Berke [6], Masri et al [10], Stephen and Vanluchene [11], Elkordy et al [12], etc. As to the application in model updating, Atalla and Inman used frequency domain data to train a ANN model; the trained ANN model can estimate the updated parameters quickly and yield a model representative of the measured data. They verified the proposed method on a frame structure with noisy, experimental data, and reported that the particular choice of input data can make the training more robust with respect to noise [13].

Levin and Lieven found that the ANN model is robust and can withstand the presence of noise in experimental data, and that the ANN approach can avoid the common problem of coordinate incompleteness [14]. The inputs and outputs of the ANN model can be selected with a certain flexibility, which provides the possibility of the direct updating of the structural parameters and boundary conditions by supplying only a limited number of modal parameters from the measurement. Instead of dealing with the sensitivity matrix that might be potentially ill-conditioned, the ANN model requires proper training using simulated or measured data. The training process could be very time-consuming and the accuracy of the predictions of this trained ANN model depends very much on the training data. However, once the model is properly trained, the ANN calculation is relatively fast regardless of the complexity of the structure to be updated. In addition, the ANN technique is well known for its ability to model nonlinear and complex relationship that is exactly the case between the structural parameters and the modal properties.

The application of the ANN methods in the area of model updating and damage identification appears to be still at its early stage. Most of the numerical examples presented so far are confined to simple structures such as cantilever beams and frames [13,14]. As the complexity of the structure and the number of structural parameters to be updated increase, problems associated with the ANN model will arise accordingly and this issue needs to be addressed. Also, as the number of structural parameters to be updated increases, the number of training samples required to ensure sample completeness increases exponentially. A large number of training samples would inevitably require a longer training time and a more efficient training algorithm.

Masri et al showed that two hidden layers could be sufficient in most of the structure-related problems [10]. The numbers of neurons in the hidden layers are determined normally by numerical experimentation (trial and error). The inputs considered would include modal parameters such as the natural frequencies and the mode shapes. The outputs on the other hand are the structural parameters to ensure sample completeness increases exponentially. A large number of training samples would inevitably require a longer training time and a more efficient training algorithm. The objective of the ANN model is to predict the structural parameters by inputting some measured modal parameters. A survey paper summarizing the application of neural networks to problems in computational mechanics was recently published by Yagawa and Okuda [15]. Soft computing encompasses a large class of often biologically inspired methods, including neural networks and genetic algorithms, which are frequently applied to inverse problems. These methods are model-free and robust to imprecision and uncertainty, making it possible to solve otherwise intractable problems [16].

From the literature survey, it is observed that the design of reliable ANN is as yet an unresolved issue. With the development of artificial intelligent techniques, the neural network methods have recently become widely accepted in the civil engineering area. This paper presents the application of three main neural networks, Multi-layer Perceptron Neural Network (MLPNN), Radial Basis Function Neural Network (RBFNN) and General Regression Neural Network (GRNN) in identification of damage in trussed bridge structures. A more detailed treatment on the aspect of ANNs is given in this paper. The efficiency of the neural networks is also looked at with special reference to bridge truss structure.

## 2. Artifical Neural Network Concept

Based on the current understanding of neurons, a computational model is developed. Artificial neural networks are non-linear mapping systems with a structure loosely based on principles observed in biological nervous systems. In greatly simplified terms, as a typical real neuron has a branching dendritic tree that collects signals from many other neurons in a limited area; a cell body that integrates collected signals and generates a response signal (as well as manages metabolic functions); and a long branching axon that distributes the response through contacts with dendritic trees of many other neurons. The response of each neuron is a relatively simple non-linear function of its inputs and is largely determined by the strengths of the connections from its inputs. In spite of the relative simplicity of the individual units, systems containing many neurons can generate complex and interesting behaviours.

An ANN shown in Fig.1 is very loosely based on these ideas. In the most general terms, an ANN consists of large number of simple processors linked by weighted connections. By analogy, the processing nodes may be called neurons. Each node output depends only on information that is locally available at the node, either stored internally or arriving via the weighted connections. Output depends only on information that is locally available at the node, either stored internally or arriving via the weighted connections.

Each unit receives inputs from many other nodes and transmits its output to yet other nodes. By itself, a single processing element is not very powerful; it generates a scalar output with a single numerical value, which is a simple non-linear function of its inputs. The power of the system emerges from the combination of many units in an appropriate way. Many good reviews on the neural network paradigms are available in the literature. Here, only a brief conceptualization of neural nets and its computational counterpart is given.

ANNs comprise a massively interconnected network of a large number of artificial neurons or computational units. Neural networks models are specified by the net topology, node (artificial neuron) characteristics and training or learning rules. The function of neural network is determined by these parameters. The architecture of the network determines the input of the node. The training or learning ruled determine how the network will react when an unknown input is presented to it.

The large connectivity degree of the neurons and massive parallelism as well as their nonlinear analogue response and learning capabilities are the basic factors which characterize the computational effectiveness of neural networks.

Computing devices based on neural network are a greater fault tolerance than a classical sequential computer due to the increased number of locally connected processing nodes. Thus, some neurons or links out of order do not diminish considerably the performance of the network.

Neural networks hold the promise of providing a fast, data-drive modeling system with desirable robustness properties since their strengths and weaknesses complement those of more traditional analytic technique.

A robust damage assessment methodology must be capable of recognizing pattern in the observed response of structures resulting from individual member damaged including the capacity of determining the extent of member damage. The area of damage assessment paradigm in the context of steel bridges has been presented by Pandey and Barai discussing in various issues [3]. The difficulty is encountered because the data available from site’s measurement are often imprecise and inadequate. This problem can be illustrated more effectively by using ANN as demonstrated in the recent papers. However, there is a gap in our understanding concerning the development of architecture of networks and simulation research should be carried out in order to design a reliable network. The present paper attempts to examine the suitability of three neural networks in damage detection of bridges [4].

### *2.1. Multilayer Perceptron Neural Network (MLPNN)*

Here, we have adopted the Multi-Layer Perceptron Neural Network (MLPNN) or Multi-Layer Feed-Forward (MLFF) for this inverse problem [16-20]. Neural networks are massively parallel computational models. Through training, neural networks learn and generalize complex relations and associations between input and output data. The trained neural networks are then capable of estimating output given new input according to the mapping or association resulting from the training procedure. A typical MLPNN consisting of thee layers of neurons is shown in Fig.1. The outer layers are input and output; the intermediate layers are hidden. Neurons or nodes in each layer are fully connected to all nodes in the adjoining layers. Each neuron receives signals through its incoming connections, performs some simple operations, adding the received signals to the bias to get an input value, calculating the output value by applying a transfer function to the input value, and sends signals through its outgoing connections. The strength of each connection depends on its weight. Learning is a procedure in which the connection weights and biases are updated using a back-propagation training algorithm such that the given input produces the known corresponding output. The resulting knowledge is stored in the connection weights and biases [21]. The application of neural networks to inverse problems consists of three stages:

(1) Determination of the neural network architecture,
(2) Selection of training patterns for ANN,
(3) Training the ANN.

The three stages are interrelated and interactive. For the MLPNN, error back-propagation is the most popular and efficient training algorithm [22]. Convergence of error back-propagation is strongly affected by the neural network architecture and the quality and quantity of selected training patterns. The quality and quantity of training patterns also have strong influence on the generalization capability of the neural network, and application of some kind of regularization of the input and output data before training has become standard practice [23]. With respect to network architecture, the number of nodes in the input and output layers corresponds to the number of elements of input and output, respectively. However, there is no rigorous method for selecting the appropriate numbers of hidden layers and neurons, although the automatic node generation of neural networks has been studied. 'Trial and error' is still the most widely used method in practical applications. The learning capability of a neural network depends on the number of hidden layers and the number of nodes in each. If the size of the neural network is too small compared to the complexity of the mapping between input and output data, the training procedure can be slow to converge, or the neural network can fall into a local minimum. On the other hand, if the neural network is too large, the training time increases dramatically, and the likelihood of over-fitting, in which case the neural network produces very accurate output upon input of training samples but gives large errors when subjected to a new input set,increases.

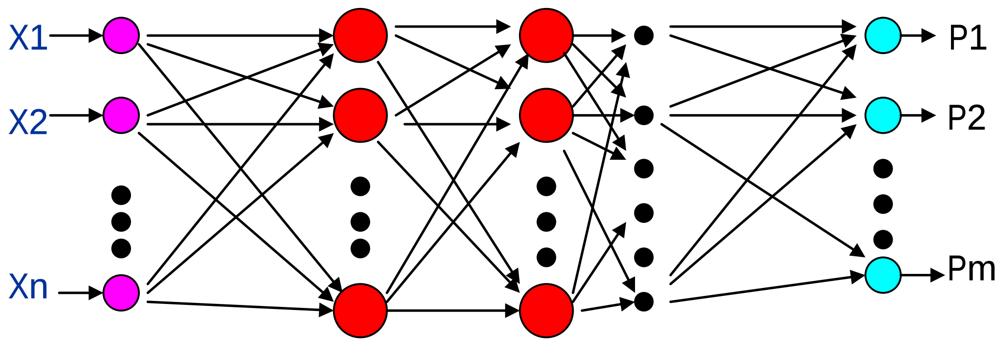


Fig. 1. Architecture of Multi-Layer Perceptron Neural Network (MLPNN)

A network's architecture is a specification of artificial neurons and their relationships. The multi-layer perceptron is very popular ANN architecture and had performed well in variety of applications in several domains including a few structural engineering applications so far. A multi-layer perceptron consist of an array of input neurons, known as an input layer, an array of output neurons, known as the output layer and number of hidden layers. Each neuron receives a weighted sum from each neuron in the preceding layer and provides an input to every neuron of next layer. The activation of each neuron is governed by a threshold function. In order to train the network, a popular training algorithm of multi-layer perceptron is the back-propagation method where the error calculated at the output of the network is propagated through the layers of neurons to update the weights. The learning algorithm is illustrated follow. Back-propagation algorithm (the generalized delta rule algorithm) is very effective for learning examples of the behavioral phenomena. The derivation of generalized delta rule is included in some follow steps. The schematic view of the network is shown in Fig.1 and steps involved for implementing the algorithm are as follows.

A network is specialized to implement different functions by varying the connection topology and the values of the connecting weights. Complex functions can be implemented by connecting units together with appropriate weights. In fact, it has been shown that a sufficiently large network with an appropriate structure and property chosen weights can approximate arbitrary accuracy any function satisfying certain broad constraints.

Usually, the processing units have responses like,

$$y = f\left(\sum_i u_i\right)$$

Where $u_i$ are the output signals of hidden layer to output layer, $f$ is a simple non-linear function such as the sigmoid, or logistic function. This unit computes a weighted linear combination of its inputs and passes this through the non-linearity to produce a scalar output. In general, it is a scalar output. In general, it is a bounded non-decreasing non-linear function; the logistic function is a common choice.

This model is, of course, a drastically simplified approximation of real nervous systems. The intent is to capture the major characteristics important in the information processing functions of real networks without varying too much about physical constraints imposed by biology.

Select a number of input nodes ( $n$ ), output nodes ( $p$ ) and hidden nodes ( $m$ ) and first training example ( $x_i$ )

$$\{x_i\} = \begin{Bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{Bmatrix} \tag{1}$$

Initialize the weights using random number generator in range of [-0.5-+0.5].

$$[W_{ji}] = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdot & \cdot & w_{1n} \\ w_{21} & w_{22} & w_{23} & \cdot & \cdot & w_{2n} \\ w_{31} & w_{32} & w_{33} & \cdot & \cdot & w_{3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ w_{m1} & w_{m2} & w_{m3} & \cdot & \cdot & w_{mn} \end{bmatrix} \tag{2}$$

Compute the value of { $net_j$ } for the hidden nodes.

$$\{net_j\} = \begin{Bmatrix} net_1 \\ net_2 \\ \cdot \\ \cdot \\ \cdot \\ net_m \end{Bmatrix} = [w_{ji}]\{X_i\} \tag{3}$$

Calculation the activation value { $out_j$ } for the hidden nodes. Here the Tangent Sigmoidal function has been used.

$$\{out_j\} = \begin{Bmatrix} out_1 \\ out_2 \\ \cdot \\ \cdot \\ \cdot \\ out_m \end{Bmatrix} = \begin{Bmatrix} f_1(net_1) \\ f_2(net_2) \\ \cdot \\ \cdot \\ \cdot \\ f_m(net_m) \end{Bmatrix} \tag{4}$$

Calculation the value of { $net_k$ } for the output nodes.

$$\{net_k\} = \begin{Bmatrix} net_1 \\ net_2 \\ \cdot \\ \cdot \\ \cdot \\ net_p \end{Bmatrix} = [w_{kj}]\{out_j\} \tag{5}$$

Calculation the activation value
{ $out_k$ } for output nodes.

$$\{out_k\} = \begin{Bmatrix} o_1 \\ o_2 \\ \cdot \\ \cdot \\ \cdot \\ o_p \end{Bmatrix} = \begin{Bmatrix} f_1(net_1) \\ f_2(net_2) \\ \cdot \\ \cdot \\ \cdot \\ f_p(net_p) \end{Bmatrix} \tag{6}$$

Calculation error [ $\Delta W_{kj}$ ].

$$[\Delta W_{kj}] = \eta\{t_k - o_k\}\{o_k\}\{out_j\}^T + \alpha[\Delta_p W_{kj}] \tag{7}$$

Where $\alpha$ and $\eta$ are momentum and learning parameter and usually selected from experience. Compute the new value of weights between the hidden and output layers.

$$[W_{kj}] = [W_{kj}] + [\Delta W_{kj}] \tag{8}$$

Calculate the [ $\Delta W_{ji}$ ] for input to hidden weights.

$$[\Delta W_{ji}] = \eta\{out_j\}\{t_k - o_k\}\{o_k\}[W_{kj}]^T\{X_i\}^T + \alpha[\Delta_p W_{ji}] \tag{9}$$

Calculate the new value of weights between input and hidden layer.

$$[W_{ji}] = [W_{ji}] + [\Delta W_{ji}] \tag{10}$$

The algorithm continues for all set until the Average System Error (ASE) between the target output and computed output is close to the tolerance specified.

## *2.2. Radial Basis Function Neural Network (RBFNN)*

The basic concept underlying the RBF is that of a fixed non-linear mapping of the input space to a higher dimensional space followed by a linear, adjustable output mapping. As it was shown in Fig.1, the structure of the RBF is a model of three-layer feed forward network like MLPNN. The hidden and output layer consists of a set of basis function units, each of which associated with a parametric vector known as its receptive field. These units compute the distance between the center of the field and the input vector. The output of the units is then a function of the distance measure. The RBF is expressed by [17-20]:

$$f(x)=\sum_{j=1}^{N} w_j y_j\left(x-c_j\right) \tag{11}$$

Where $x$ is a $n$-dimensional input vector; $N$, the number of hidden units and $c_j$, the receptive field. The basis function is such that $y_j$ has a significant result only in the neighborhood of $c_j$. There are several possibilities for the choice of basis functions. However, Gaussian type functions offer desirable properties making the hidden units responsive to locally tuned regions. The typical example of basis function has been explained. Gaussian type activation is employed in the proposed NN. The Gaussian activation function can be expressed as follows:

$$y_j(x)=\exp\left[-\frac{\left(x-\xi_j\right)^2}{\sigma_j^2}\right] \tag{12}$$

The chosen basis function influences both the learning and modeling abilities of the network, and will also influence the choice of learning rule used to train the network. The performance of the radial basis neural network depends critically on the placement of the centers of the receptive and the localization associated with each radial basis function is a vital factor for attaining faster training speeds. In this study, radial basis Gaussian function and back-propagation learning algorithm are employed to train the proposed NN. The learning algorithm topology, which was employed for the NN updating the weight can be described as follows; define the error function as:

$$J=\frac{1}{2}\sum_{i=1}^{n_0}\left(y_{di}(t)-y_i(t)\right)^2 \tag{13}$$

Where $y_{di}(t)$ are the $i$ th desired outputs and $y_i(t)$ are the $i$ th outputs of the network. This error function is to be minimized with respect to all the unknown parameters $\Theta$. In the steepest descent approach the parameter vector $\Theta=\left[\theta_1 \quad \theta_2, \quad . \quad . \quad . \quad \theta_n\right]^T$ is adjusted using the increment vector $\left[\Delta\theta_1 \quad \Delta\theta_2, \quad . \quad . \quad . \quad \Delta\theta_n\right]^T$ defined along the negative gradient direction of $J$,

$$\Delta\theta_i=-\eta\frac{\partial J}{\partial\theta_i} \tag{14}$$

Although the one-hidden-layer model is used in the present application, it is useful to derive the gradient of $J$ for the general case, and the result for the one-hidden-layer model can readily be obtained as a special case. Starting from the output layer $m$ of the network and setting $\theta_i=W_{ij}^m$, the application of the chain rule gives rise to:

$$\frac{\partial J}{\partial W_{ij}^m}=\frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial W_{ij}^m} \tag{15}$$

From Equation (4-3) we have:

$$\frac{\partial J}{\partial y_j}=-\left(y_{di}-y_i\right)=-\delta_i^m \tag{16}$$

Where $\delta_i^m$ is called the error signal of the $i$ th neuron in the $m$ th layer. From Equation (15):

$$\frac{\partial y_i}{\partial W_{ij}^m}=x_j^{m-1} \tag{17}$$

Thus,

$$\frac{\partial J}{\partial W_{ij}^m}=-\delta_i^m x_j^{m-1} \tag{18}$$

Next consider the $(m-1)$th layer. Using the chain rule yields:

$$\frac{\partial J}{\partial W_{ij}^{m-1}}=\sum_{k=1}^{n_0}\frac{\partial J}{\partial y_k}\frac{\partial y_k}{\partial x_i^{m-1}}\frac{\partial x_i^{m-1}}{\partial z_i^{m-1}}\frac{\partial z_i^{m-1}}{\partial W_{ij}^{m-1}} \tag{19}$$

Then,

$$\frac{\partial x_i^{m-1}}{\partial z_i^{m-1}}=g'\left(z_i^{m-1}\right) \tag{20}$$

And:

$$\frac{\partial z_i^{m-1}}{\partial W_{ij}^{m-1}}=x_j^{m-1} \tag{21}$$

$$g'(z)=\frac{\partial g(z)}{\partial z} \tag{22}$$

And $g(z_i)$ is the activation of neuron $i$. By defining the error signal for the $i$ th neuron of the $(m-1)$th layer as:

$$\delta_i^{m-1} = g'\left(z_i^{m-1}\right)\sum_{k=1}^{n_0} \delta_k^m W_{ki}^m \tag{23}$$

Equation (13) can be rewritten as:

$$\frac{\partial J}{\partial W_{ij}^{m-1}} = -\delta_i^{m-1} x_j^{m-2} \tag{24}$$

Similarly it can be shown that,

$$\frac{\partial J}{\partial b_i^{m-1}} = -\delta_i^{m-1} \tag{25}$$

Where $b_i^{m-1}$ is the bias input to neuron $i$ in layer $m-1$

By carrying on this procedure, Equations (23) to (25) can be used as a general algorithm for updating weights in other layers.

Equations (23) to (25) indicate how the error signals propagate backwards from the output layer of the network through the hidden layer to the input layer, hence the famous name back-propagation. The steepest-descent minimization of the error function defined in Equation (13) produces the following increments for updating $\Theta$ :

$$\Delta W_{ij}^m(t) = \eta_w \delta_i^m(t) x_j^{m-1}(t) \tag{26}$$

$$\Delta b_i^m(t) = \eta_b \delta_i^m(t) \tag{27}$$

Where in the output layer,

$$\delta_i^m(t) = y_{di}(t) - y_i(t) \tag{28}$$

And in other layers,

$$\delta_i^m(t) = g'\left(z_i^m(t)\right)\sum_j \delta_j^{m+1} W_{ji}^{m+1}(t-1) \tag{29}$$

The constants $\eta_w$ ($0 < \eta_w < 1$) and $\eta_b$ ($0 < \eta_b < 1$) represent the learning rates for the weights and biases, respectively. In practice, a large value of the learning rate would be preferable, because this would result in rapid learning. Unfortunately, a large value of the learning rate can also lead to oscillation or even divergence. To help speed up learning, but avoid undue oscillations, a momentum term is usually included so that Equations (26) and (27) become:

$$\Delta W_{ij}^m(t) = \eta_w \delta_i^m(t) x_j^{m-1}(t) + \mu_w \Delta W_{ij}^m(t-1) \tag{30}$$

$$\Delta b_i^m(t) = \eta_b \delta_i^m(t) + \mu_b \Delta b_i^m(t-1) \tag{31}$$

Where $\mu_w$ and $\mu_b$ are momentum constants, which determine the effect of past changes of $\Delta W_{ij}^m(t)$ and $\Delta b_i^m(t)$ on the current updating direction in the weight and the bias space respectively. This effectively filters out high frequency variations in the error surface. To summarize, the back-propagation algorithm updates the weights and thresholds of the networks according to:

$$W_{ij}^m(t) = W_{ij}^m(t-1) + \Delta W_{ij}^m(t) \tag{32}$$

And,

$$b_i^m(t) = b_i^m(t-1) + \Delta b_i^m(t) \tag{33}$$

Where the increments $\Delta W_{ij}^m(t)$ and $\Delta b_i^m(t)$ are given in Equations (30) and (31). This simulation study will carry out using radial basis Gaussian NN for analyzing steel bridge parameters in next portion. The back-propagation algorithm is used to update the network weights.

### 2.3. General Regression Neural Network (GRNN)

The multi-layer perception and radial basis function are the best known feed-forward networks, which have been studied extensively and used widely in many fields. However, due to its learning process, RBF and specially MLP, have some shortcomings such as easy to converge to a local minimum, a slow convergence rate, sensitivity to initial conditions, low learning efficiency, etc. These disadvantages generally result from the use of the back-propagation learning algorithm. In the past few years, many advanced neural network architectures and learning algorithms have emerged and have been successfully applied to practical problems. Examples include Recurrent Neural Networks (RNN), Probabilistic Neural Network (PNN), General Regression Neural Network (GRNN), etc. As an alternative to the conventional heuristic methods, general regression neural network appears promising and can offer more formal and robust methods for the design of neural network. For example, GRNN can be used to aid in determining the best connections among network units, selecting the weights in a relatively fixed architecture, evolving appropriate network structures and learning parameters, developing the learning rule for ANN, finding an effective set of weights for a fixed set of connections, etc [24].

In this paper, the GRNN is also used to model the damage detection and to predict the states of damage at steel bridges. GRNN is known for its ability to train quickly with sparse data sets. Its architecture is of a three-layer network consisting of one hidden layer neuron. There are no training parameters such as learning rate and momentum as in back-propagation, but there is a smoothing factor that is applied after the network is trained. GRNN works by measuring how far a given sample pattern is from patterns in the training set in N dimensional space, where N is the number of neurons in the hidden layer. When a new pattern is presented to the network, the input pattern is compared in the N dimensional space to all of the patterns in the training set to determine how far in distance it is from those patterns. The output of the network is the proportional amount of all the outputs in the training set. The proportion is based upon how far the new pattern is from the given patterns in the training set [25]. A generalized regression neural network is also used for function approximation. As discussed below, it has a radial basis layer and a special linear layer. The architecture for the GRNN is similar to the radial basis network, but has a slightly different second layer. The first layer is just like that for RBF. It has as many neurons as there are input- target vectors in $P$ Specifically, the first layer weights are set approximately $P'$ and the bias is set to column vector of 0.8326/spread. The user chooses spread; the distance an input vector must be from a neuron's weight to be 0.5. Again, the first layer operates just like the radial basis layer described previously. Each neuron's weighted input is the distance between the input vector and its weight vector, calculated with a function of distance. Each neuron's net input is the product of its weighted input with its bias, calculated with a function. Each neuron's output is its net input passed through radial basis layer. If a neuron's weight vector is equal to the input vector (transposed), its weighted input will be 0, therefore its net input will be 0, and its output will be 1. If a neuron's weight vector is a distance of spread from the input vector, its weighted input will be spread, and its net input will be Sqrt (-log .5). The second layer also has as many neurons as input- target vectors.

Suppose we have an input vector $p$ close to $p_i$, one of the input vectors among the input vector/target pairs used in designing layer one weights. This input $p$ produces a layer 1 $a_i$ output close to 1. This leads to a layer 2 outputs close to $t_i$, one of the targets used forming layer 2 weights. A larger spread leads to a large area around the input vector where layer 1 neurons will respond $i$ th significant outputs. Therefore, if spread is small, the radial basis function is very steep so that the neuron with the weight vector closest to the input will have a much larger output than other neurons. The network will tend to respond with the target vector associated with the nearest design input vector. As spread gets larger, the radial basis function's slope gets smoother and several neuron's may respond to an input vector. The network then acts as if it is taking a weighted average between target vectors whose design input vectors are closest to the new input vector. As spread gets larger, more and more neurons contribute to the average with the result that the network function becomes smoother. In the present research, GRNN also was used to stimulate the structural response of bridge.

## 3. Simulation Results

An extensive neural networks investigation was carried out on a problem of 21-bar bridge truss as shown in Fig.2 with three distinct zones assumed as damage state in Fig.3. The bottom chord nodes, 7, 8, 9, 10 and 11 were considered for measuring the response of the structure under given loads as shown in Fig.2 The purpose of the exercise was to identify the members in the damaged zone and the reduction in their stiffness from the response data. Three networks employed to do this exercise. In this networks, the input nodes represent the measured parameters, Vertical displacement at the nodes, U7, U8, U9, U10 and U11 and the output nodes are the identified parameter, response data of a1, a2, … to a21 in the member representing the stiffness in assumed damage state for the neural networks. The details study was carried out with the following three networks,

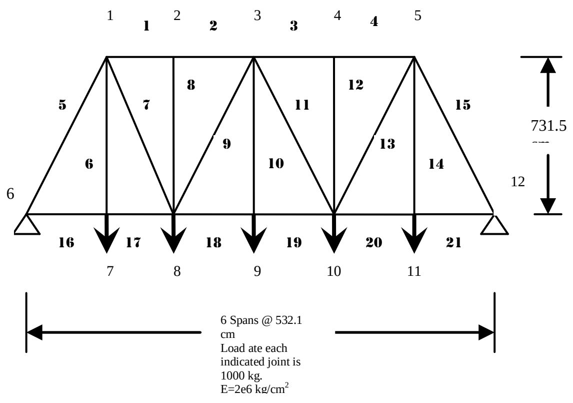


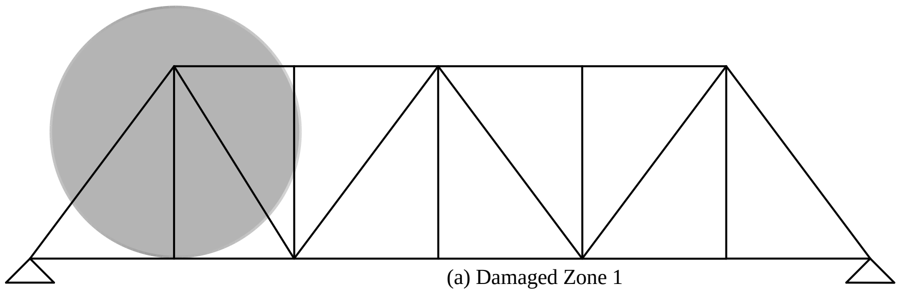
Fig. 2. Steel Bridge Configuration

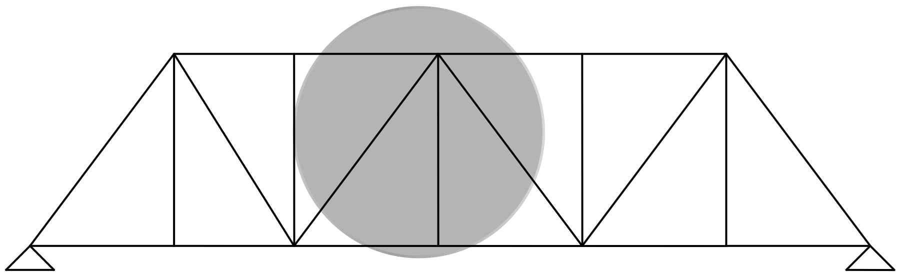
(1). For MLPNN, it's need to find the best architecture that has minimum ASE between the target output and computed output from all set of architecture that would be chosen and it's close to the tolerance specified. In Fig.4 the performance of MLPNN is depited. As showed in Fig.4 when the number of neurons in hidden layer had chosen 19 the MSE error is minimum (9.02E-13), then the best architecture for design MLPNN is 5-19-21; this network has 5 neurons in input layer, 19 neurons in hidden layer and 21 neurons in output layer. Learning parameter $\eta$ and momentum parameter $\alpha$ are 0.05 and 0.9, respectively that are quite appropriate in such application. It should be noted that this architecture is best for this investigation. The best architecture should be found via trial and error in practice.

(2). The RBFNN with architecture 5-19-21 (consists of three layers, input layer, one hidden layer and output layer) that have Gaussian function with non-linear neurons in hidden and output layer. Training parameters of RBF network are selected as $\eta = 0.001$ (learning parameter) and $\mu = 0.01$ (momentum parameter) via trial and error. The input layer has 5 neurons; hidden layer and output layer have 19 and 21 Gaussian function with RBF non-linear neurons, respectively.

(3). The GRNN with the architecture 5-19-21, as illustrated in previous section it has a non-linear RBF layer and special linear layer. For this study, spread parameter has chosen 1.0 that it helps more neurons to contribute to the average, and the result that the network's function becomes smoother.

Again all networks consist of three layers, five input neurons, one hidden layer with nineteen neurons and twenty one neurons in output layer. The steel bridge structure has been identified having three distinct damage zones where members are assumed to be damaged in turn. Considering each damaged zone independently and separately, a total of 48 training patterns, 16 patterns for each damage zone were generated with the help of finite element method stored in files and additional 12 testing patterns were generated and stored separately for verification of the trained networks after proper normalization. The normalization is essential while using the Tangent-Sigmidal or Gaussian function, in order to have the input and training patterns values between 0 and 1. These normalized training patterns presented to the network help faster convergence. For normalization of the input – output pair, an interface program was developed. In this study, the back-propagation algorithm was implemented for RBFNN and MLPNN. The networks were trained for the three distinct damaged zones and then tested for 12 testing patterns. For each zone, all the testing patterns were presented here for the three zones. Form sample.1 to sample.4, the damage in every zone will be extending. The results obtained for computational experimentation were quite promising. Some of the typical results for the three zones are given in Tables 1-3. The observations of this study are discussed in the following section.


(a) Damaged Zone 1


(b) Damaged Zone 2

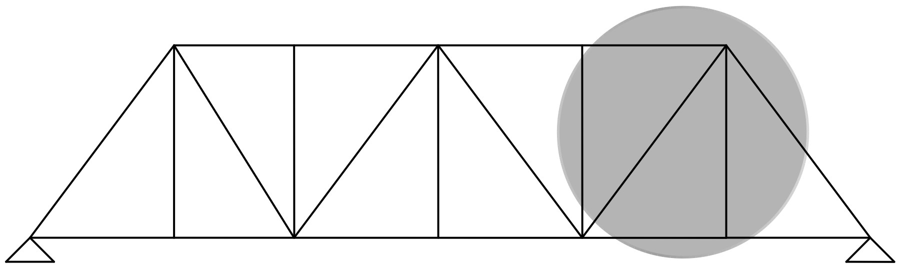

(c) Damaged Zone 3

Fig.3: States of Damage in Steel Bridge and Identification of Damaged Zones

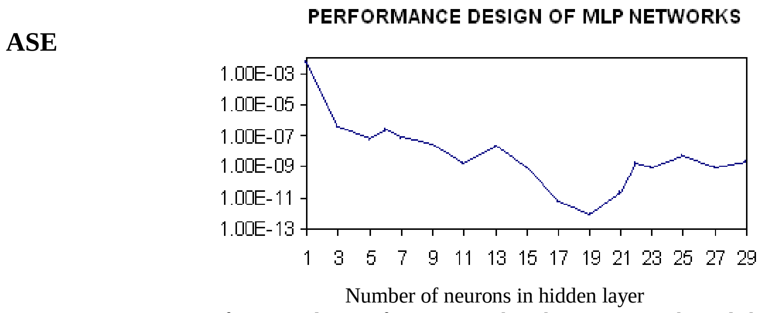


Fig. 4. Performance design of MLP networks, The MLPNN with single hidden layer architecture 5-19-21 was the best architecture from all the examined MLPNNs

Table1. Typical samples of zone 1

| | Actual Stiffness (FEM) | MLPNN model (%) | RBFNN model (%) | GRNN model (%) |
|---|---|---|---|---|
| ***Sample 1.*** | | | | |
| member 1. | 103304.03 | 0.0030 | 0.0008 | 0.0007 |
| member 5. | 38693.273 | 0.0037 | 0.0009 | 0.0008 |
| member 6. | 61517.430 | 0.0039 | 0.0005 | 0.0005 |
| member 7. | 38693.273 | 0.0044 | 0.0009 | 0.0008 |
| member 8. | 61517.430 | 0.0030 | 0.0005 | 0.0005 |
| member 16. | 103304.03 | 0.0031 | 0.0008 | 0.0007 |
| member 17. | 103304.03 | 0.0035 | 0.0008 | 0.0007 |
| ***Sample 2.*** | | | | |
| member 1. | 93967.299 | 0.0025 | 0.0006 | 0.0006 |
| member 5. | 33165.662 | 0.0036 | 0.0013 | 0.0013 |
| member 6. | 54682.159 | 0.0033 | 0.0005 | 0.0005 |
| member 7. | 33165.662 | 0.0040 | 0.0013 | 0.0013 |
| member 8. | 54682.159 | 0.0028 | 0.0005 | 0.0005 |
| member 16. | 93967.299 | 0.0026 | 0.0006 | 0.0006 |
| member 17. | 93967.299 | 0.0028 | 0.0006 | 0.0006 |
| ***Sample 3.*** | | | | |
| member 1. | 84570.569 | 0.0014 | 0.0004 | 0.0003 |
| member 5. | 27638.052 | 0.0092 | 0.0024 | 0.0023 |
| member 6. | 47846.889 | 0.0016 | 0.0002 | 0.0002 |
| member 7. | 27638.052 | 0.0051 | 0.0024 | 0.0023 |
| member 8. | 47846.889 | 0.0065 | 0.0002 | 0.0002 |
| member 16. | 84570.569 | 0.0019 | 0.0004 | 0.0003 |
| member 17. | 84570.569 | 0.0008 | 0.0004 | 0.0003 |
| ***Sample 4.*** | | | | |
| member 1. | 75173.839 | 0.1608 | 0.0137 | 0.0137 |
| member 5. | 22110.441 | 0.2125 | 0.0364 | 0.0364 |
| member 6. | 41011.619 | 0.1842 | 0.0404 | 0.0403 |
| member 7. | 22110.441 | 0.2235 | 0.0364 | 0.0364 |
| member 8. | 41011.619 | 0.1270 | 0.0404 | 0.0403 |
| member 16. | 75173.839 | 0.1251 | 0.0137 | 0.0137 |
| member 17. | 75173.839 | 0.1267 | 0.0137 | 0.0137 |

The following observations are made based on all the twelve tested patterns:

1- Training networks were able to identify the damaged zones for all the testing patterns.
2- Training networks had provided reasonable value of stiffness of all members for the testing patterns. They have good agreement with the actual stiffness that is present by solution of Finite Element Method.
3- The performance of the GRNN was the best, although RBFNN has results so closed to GRNN. RBFNN and GRNN have better results than MLPNN in all case of damage and for all samples. The percentage error in MLPNN, RBFNN and GRNN is increase when the damage of the structure will be extended.
4- As we expected form the results, the error in the members that have same stiffness should be equal in ANN with liner mapping or non-linear mapping.
5- Time consumed for training GRNN and RBFNN networks is more than MLPNN, but they have less error than MLPNN,
6- In training MLPNN, when the number of neurons in hidden layer increase, it will take further time in the CPU time usage to train the MLPNN

.

Table2. Typical samples of zone 2

| | Actual Stiffness (FEM) | MLPNN model (%) | RBFNN model (%) | GRNN model (%) |
|---|---|---|---|---|
| ***Sample 1.*** | | | | |
| member 2. | 103304.030 | 0.0007 | 0.0004 | 0.0003 |
| member 3. | 103304.030 | 0.0001 | 0.0004 | 0.0003 |
| member 9. | 38693.273 | 0.0022 | 0.0004 | 0.0004 |
| member 10. | 61517.430 | 0.0005 | 0.0006 | 0.0005 |
| member 11. | 38693.273 | 0.0009 | 0.0004 | 0.0004 |
| member 18. | 103304.030 | 0.0012 | 0.0004 | 0.0003 |
| member 19. | 103304.030 | 0.0001 | 0.0004 | 0.0003 |
| ***Sample 2.*** | | | | |
| member 2. | 93967.299 | 0.0008 | 0.0016 | 0.0016 |
| member 3. | 93967.299 | 0.0004 | 0.0016 | 0.0016 |
| member 9. | 33165.662 | 0.0020 | 0.0027 | 0.0027 |
| member 10. | 54682.159 | 0.0008 | 0.0020 | 0.0019 |
| member 11. | 33165.662 | 0.0012 | 0.0027 | 0.0027 |
| member 18. | 93967.299 | 0.0009 | 0.0016 | 0.0016 |
| member 19. | 93967.299 | 0.0004 | 0.0016 | 0.0016 |
| ***Sample 3.*** | | | | |
| member 2. | 84570.569 | 0.0043 | 0.0035 | 0.0035 |
| member 3. | 84570.569 | 0.0042 | 0.0035 | 0.0035 |
| member 9. | 27638.052 | 0.0075 | 0.0057 | 0.0056 |
| member 10. | 47846.889 | 0.0055 | 0.0043 | 0.0042 |
| member 11. | 27638.052 | 0.0075 | 0.0057 | 0.0056 |
| member 18. | 84570.569 | 0.0040 | 0.0035 | 0.0034 |
| member 19. | 84570.569 | 0.0043 | 0.0035 | 0.0034 |
| ***Sample 4.*** | | | | |
| member 2. | 75173.839 | 0.0420 | 0.0031 | 0.0030 |
| member 3. | 75173.839 | 0.0419 | 0.0031 | 0.0030 |
| member 9. | 22110.441 | 0.0778 | 0.0648 | 0.0648 |
| member 10. | 41011.619 | 0.0554 | 0.0436 | 0.0436 |
| member 11. | 22110.441 | 0.0820 | 0.0648 | 0.0648 |
| member 18. | 75173.839 | 0.0372 | 0.0310 | 0.0309 |
| member 19. | 75173.839 | 0.0431 | 0.0310 | 0.0309 |

Table3. Typical samples of zone 3

| | Actual Stiffness (FEM) | MLPNN model (%) | RBFNN model (%) | GRNN model (%) |
|---|---|---|---|---|
| ***Sample 1.*** | | | | |
| member 4. | 103304.03 | 0.0025 | 0.0010 | 0.0009 |
| member 12. | 61517.430 | 0.0027 | 0.0008 | 0.0007 |
| member 13. | 38693.273 | 0.0035 | 0.0012 | 0.0012 |
| member 14. | 61517.430 | 0.0012 | 0.0008 | 0.0007 |
| member 15. | 38693.273 | 0.0034 | 0.0012 | 0.0015 |
| member 20. | 103304.03 | 0.0028 | 0.0010 | 0.0009 |
| member 21. | 103304.03 | 0.0036 | 0.0010 | 0.0009 |
| ***Sample 2.*** | | | | |
| member 4. | 93967.299 | 0.0020 | 0.0008 | 0.0008 |
| member 12. | 54682.159 | 0.0043 | 0.0007 | 0.0007 |
| member 13. | 33165.662 | 0.0052 | 0.0016 | 0.0015 |
| member 14. | 54682.159 | 0.0070 | 0.0007 | 0.0007 |
| member 15. | 33165.662 | 0.0048 | 0.0016 | 0.0015 |
| member 20. | 93967.299 | 0.0022 | 0.0008 | 0.0008 |
| member 21. | 93967.299 | 0.0005 | 0.0008 | 0.0008 |
| ***Sample 3.*** | | | | |
| member 4. | 84570.569 | 0.0188 | 0.0004 | 0.0004 |
| member 12. | 47846.889 | 0.0202 | 0.0014 | 0.0137 |
| member 13. | 27638.052 | 0.0304 | 0.0037 | 0.0037 |
| member 14. | 47846.889 | 0.0175 | 0.0014 | 0.0013 |
| member 15. | 27638.052 | 0.0301 | 0.0037 | 0.0037 |
| member 20. | 84570.569 | 0.0200 | 0.0004 | 0.0004 |
| member 21. | 84570.569 | 0.0220 | 0.0004 | 0.0004 |
| ***Sample 4.*** | | | | |
| member 4. | 75173.839 | 0.1707 | 0.0023 | 0.0023 |
| member 12. | 41011.619 | 0.1854 | 0.0287 | 0.0286 |
| member 13. | 22110.441 | 0.3163 | 0.0119 | 0.0118 |
| member 14. | 41011.619 | 0.3244 | 0.0287 | 0.0286 |
| member 15. | 22110.441 | 0.3312 | 0.0119 | 0.0118 |
| member 20. | 75173.839 | 0.1294 | 0.0023 | 0.0023 |
| member 21. | 75173.839 | 0.0725 | 0.0023 | 0.0023 |

## 4. Conclusion

Two neural networks with back-propagation learning algorithm (MLPNN and RBFNN) and GRNN have been adapted to model typical bridge truss simulated damaged states. The training patterns were generated using general structural analysis program finite element method with assumed zones of damage in the structure. The working of the networks was demonstrated by computing the output with the algorithmically generated performance parameter not considered in the training. The issues related to the performance of networks were examined. From observations, it is concluded that the GRNN is quite appropriate for the structural damage identification and RBFNN has results closer to GRNN. For the case illustrated, the performance of architecture and the MSE was discussed for MLPNN.

All three networks have good agreement with analytical solution and they represent good results when damage in members is little but when damage extend, MLPNN has more error than GRNN and RBFNN. The engineering significance of this investigation lies in the fact that the measured data at only a few locations in the structure is needed to train the network for identification exercise.

## References

[1] P. Hajela and F. J. Soeiro, Recent development in damage detection based on system identification method Structural Optimization. 1-10, 1990.

[2] M. Sanyayei and O. Oinpede, Damage assessment of structures using static test data. AIAA Jnl 29,1174- 1179, 1991.

[3] P. C. Pandey and S. V. Barai, Damage assessment of steel bridges. J. Struct. Engng 20, 9-21, 1993.

[4] X. Wu, J. Ghaboussi and J. H. Garrett, Use of neural networks in detection of structural damage. Computer&Structures. 42, 649-659, 1992.

[5] J. Ghaboussi, J. H. Garrett and X. Wu, Knowledge based modeling of material behaviour with neuralnetworks. J. Engn Mech, ASCE..117, 132-153, 1991.

[6] P. Hajela and Berke, Neurobiological computational model in structural analysis and design. Computer&Structures. 41, 657-667, 1991.

[7] Z.P.Szewczyk and P.Hajela, Neural networks based damage detection in structures. Technical Report, RPI, Troy 1992.

[8] M. F. Elkordy, K. C. Chang and G. C. Lee, Neural networks trained by analytically simulated damage states.J. Computing in Civil Engng, ASCE 7, 130-145, 1993.

[9] H. Adeli and C. Yeh, Perceptron learning in engineering design. Microcomput. Civil Engng 4, 247–56, 1989.

[10] S. F. Masri, A. G. Chassiakos and T. K. Caughey, Identification of nonlinear dynamics system using neural networks. J. Appl. Mech. 60, 123–33, 1993.

[11] J. E. Stephens and R. D. Vanluchene, Integrated assessment of seismic damage in structures. Microcomput. Civil Engng 9, 119–28, 1994.

[12] M. F. Elkordy, C. K. Chang and G. C. Lee, A structural damage neural network monitoring system.Microcomput. Civil Engng 9, 83–96, 1994.

[13] M. J. Atalla and D. J. Inman, On model updating using neural networks. Mech. Syst. Signal Process. 12, 135–61, 1998.

[14] R. I. Levin and N. A. J. Lieven, Dynamic finite element model updating using neural networks. J. Sound Vib. 210, 593–607, 1998.

[15] G. Yagawa, H. Okuda, Neural networks in computational mechanics. Archives of Computational Methods in Engineering. 3,4, 435–512, 1996.

[16] J. Ghaboussi, Biologically inspired soft computing methods in structural mechanics and engineering.Structural Engineering and Mechanics 11, 485–502, 2001.

[17] L. Fausett, Fundamentals of Neural Network, Prentice Hall International, Inc. New Jersey, 1994.

[18] J. E. Dayhoff, Neural Networks Architecture: An Introduction, Van Nostrand Reinhold, New York, 1990.

[19] J. A. Freeman, D.M. Skapura, Neural Networks: Algorithms, Applications, and Programming Techniques, Addison-Wesley, Reading, MA, 1991.

[20] L. Tarassenko, A Guide to Neural Computing Applications, Wiley, New York, 1998.

[21] P.C. Pandey, S.V. Barai, Multilayer perception in damage detection of bridge structures. Computers & Structures, 54, 597–608, 1995.

[22] S.V. Barai, P.C. Pandey, Performance of the generalized delta rule in structural damage detection.Engineering Application of Artificial Intelligence. 8, 211–221, 1995.

[23] S. Yoshimura, A. Matsuda, G. Yagawa, New regularization by transformation for neural network basedinverse analyses and its application to structural identification. International Journal for Numerical Methods in Engineering. 39, 3953–3968, 1996.

[24] Q.S.Li, D.K.Liu, J.Q.Fang, A.P.Jeary, C.K.Wong, Damping in buildings: its neural network model and AR model. Engineering Structures. 22, 1216-1223, 2000.

[25] D. Howard, M. Beale, Neural Network Toolbox, for Use with MATLAB, User’s Guide, Version 4, The MathWorks, Inc., Natick, MA, 2000.