ELSEVIER

Advances in Engineering Software 35 (2004) 9–25

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

Learning improvement of neural networks used in structural optimization

Nikolaos D. Lagaros, Manolis Papadrakakis*

Institute of Structural Analysis and Seismic Research, National Technical University Athens, Zografou Campus, Athens 15780, Greece

Received 12 June 2002; accepted 22 July 2003

Abstract

The performance of feed-forward neural networks can be substantially impaired by the ill-conditioning of the corresponding Jacobian matrix. Ill-conditioning appearing in feed-forward learning process is related to the properties of the activation function used. It will be shown that the performance of the network training can be improved using an adaptive activation function with a properly updated gain parameter during the learning process. The efficiency of the proposed adaptive procedure is examined in structural optimization problems where a trained neural network is used to replace the structural analysis phase and capture the necessary data for the optimizer. The optimizer used in this study is an algorithm based on evolution strategies.

Keywords: Neural networks; Ill-conditioning; Structural optimization; Evolution strategies

1. Introduction

Over the last 10 years, artificial intelligence techniques have emerged as a powerful tool that could be used to replace time consuming procedures in many scientific or engineering applications. The use of artificial Neural Networks (NN) to predict finite element analysis outputs has been studied previously in the context of optimal design of structural systems [1–7] and also in some other areas of structural engineering applications, such as structural damage assessment, structural reliability analysis, finite element mesh generation or fracture mechanics [8–13]. NN have been recently applied to the solution of the equilibrium equations resulting from the application of the finite element method in connection to reanalysis type of problems, where a large number of finite element analyses are required. Reanalysis type of problems are encountered, among others, in the reliability analysis of structural systems using Monte Carlo simulation and in structural optimization using evolutionary algorithms such as Evolution Strategies (ES) and Genetic Algorithms (GA). In these problems, NN have proved to work very satisfactory [2,9].

The principal advantage of a properly trained NN is that it requires a trivial computational effort to produce an approximate solution. Such approximations, if acceptable, appear to be valuable in situations where the actual response computations are intensive in terms of computing time and a quick estimation is required. For each problem a NN is trained utilizing information generated from a number of properly selected analyses. The data from these analyses are processed in order to obtain the necessary input and output pairs, which are subsequently used to produce a trained NN. The training of a NN is an unconstrained minimization problem where the objective is to minimize the prediction error. In the case of structural optimization, the analysis corresponds to a finite element solution of the resulting equilibrium equations and the trained NN is then used to predict the response of the structure in terms of constraint function values due to different sets of design variables.

According to Saarinen et al. [14] the most widely used architecture, that of feed-forward NN, is likely to produce ill-conditioned Jacobian matrices due to the bad properties of the activation function used and that this type of ill-conditioning is encountered in many applications. This work is concerned with the implementation of a proper activation function that results to the improvement of the condition of the Jacobian matrices of the network. Theoretical analysis and experimental results presented in subsequent sections lead to the conclusion that the bad influence of ill-conditioning in the training phase of NN can be alleviated using an adaptive sigmoid activation function per layer.

* Corresponding author. Tel.: +30-1-7721694; fax: +30-1-7721693.
E-mail addresses: mpapadra@central.ntua.gr (M. Papadrakakis), nlagaros@central.ntua.gr (N.D. Lagaros).

0965-9978/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0965-9978(03)00112-1

2. Unconstrained optimization algorithms in NN training

Let us consider the following unconstrained optimization problem: find the vector/matrix $\mathbf{w}$ that minimizes the following real valued scalar function

$$\mathscr{E} = \mathscr{E}(\mathbf{w}), \tag{1}$$

which is called the cost, objective or energy function. Since the case of maximization of a function is the same as the minimization of its negative value there is no loss of generality in this consideration.

The NN attempts to create a desired relation for an input/output set of m learning patterns. This set which is called training set and consists of a finite number of m pairs $(\mathbf{inp}, \mathbf{tar}) \in R^k \times R^\ell$, where the first coordinate is a position in k-dimensional space corresponding to the input space and the second coordinate is a position in ℓ-dimensional space corresponding to the desired or target space. The algorithm that is usually used in order to form the relation $R^k \rightarrow R^\ell$ between those two spaces is the back propagation (BP) algorithm [15]. This algorithm tries to determine a set of parameters called weights, in order to achieve the right response for each input vector applied to the network. If the training is successful, application of a set of inputs to the network produces the desired set of outputs. Thus, in the case of NN training, $\mathbf{w}$ corresponds to the weight matrix defining the parameters to be determined, while the objective function can be defined as follows

$$\mathscr{E}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{E}(\mathbf{w})\|^2, \tag{2}$$

where the terms of the vector $\mathbf{E}(\mathbf{w}) = [E_1(\mathbf{w}), E_2(\mathbf{w}), \ldots, E_m(\mathbf{w})]^{\mathrm{T}}$ have to minimized. E_i denotes the residual between the value of the approximating function and the desired value, determined by the following expression

$$E_i(\mathbf{w}) = \sum_{j=1}^{\ell} [\mathbf{out}_j(\mathbf{inp}_i, \mathbf{w}) - \mathbf{tar}_{ij}], \tag{3}$$

where $\mathbf{inp}_i$ is a k-dimensional input vector, $\mathbf{tar}_i$ is the desired response corresponding to the ith input, tar_{ij} is the desire response of the jth node of the output vector for the ith input pattern, while $\mathbf{out}$ is the response of the network for the current values of the weight parameters.

There are two categories of methods proposed for the solution of the minimization problem of Eq. (2): deterministic and statistical. A deterministic training method follows a step-by-step procedure to adjust the network weights. On the other hand, statistical training methods make pseudo-random changes in the weight values retaining only those changes that result in improvement of the objective function. The latter training methods, however, appeared to be slow compared to the deterministic ones [16]. In this study, we will discuss only deterministic methods.

The numerical minimization algorithms used for the solution of the problem of Eq. (2) generate a sequence of weight matrices through an iterative procedure. To apply an algorithmic operator $\mathscr{A}$ we need a starting weight matrix $\mathbf{w}^{(0)}$, while the iteration formula can be written as follows:

$$\mathbf{w}^{(t+1)} = \mathscr{A}(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)}. \tag{4}$$

All numerical methods applied are based on the above formula. The changing part of the algorithm $\Delta\mathbf{w}^{(t)}$ is further decomposed into two parts as

$$\Delta\mathbf{w}^{(t)} = a_t\mathbf{d}^{(t)}, \tag{5}$$

where $\mathbf{d}^{(t)}$ is a desired search direction of the move and a_t the step size in that direction. Theoretically, we would like the sequence of weight matrices to converge to at least a local minimizer $\mathbf{w}^*$. The algorithm should generate the sequence of iterant matrices $\mathbf{w}^{(t)}$ so that away from $\mathbf{w}^*$ a steady progress toward $\mathbf{w}^*$ is achieved and once near $\mathbf{w}^*$ a rapid convergence to $\mathbf{w}^*$ itself occurs [17]. The convergence of an algorithm can be either global or local.

Global convergence refers to the ability of the algorithm to reach the neighborhood of $\mathbf{w}^*$ from an arbitrary initial weight matrix $\mathbf{w}^{(0)}$, which is not close to $\mathbf{w}^*$. The convergence of a globally convergent algorithm should not be affected by the choice of the initial point. Local convergence refers to the ability of the algorithm to approach $\mathbf{w}^*$ rapidly from a starting weight matrix (or iterant $\mathbf{w}^{(t)}$) in the neighborhood of $\mathbf{w}^*$.

The algorithms most frequently used in the NN training are the steepest descent, the conjugate gradient, the Newton and the Levenberg–Marquard methods with the following direction vectors

Steepest descent method

$$\mathbf{d}^{(t)} = -\nabla\mathscr{E}(\mathbf{w}^{(t)}).$$

Conjugate gradient method

$$\mathbf{d}^{(t)} = -\nabla\mathscr{E}(\mathbf{w}^{(t)}) + \beta_{t-1}\mathbf{d}^{(t-1)},$$

where β_t is defined as follows

$$\beta_{t-1} = \begin{cases} \nabla\mathscr{E}_t(\nabla\mathscr{E}_t - \nabla\mathscr{E}_{t-1})/\mathbf{d}^{(t-1)}(\nabla\mathscr{E}_{t-1} - \nabla\mathscr{E}_t) \\ \text{Hestenes–Stiefel} \\ \nabla\mathscr{E}_t\cdot(\nabla\mathscr{E}_t - \nabla\mathscr{E}_{t-1})/\nabla\mathscr{E}_{t-1}\cdot\nabla\mathscr{E}_{t-1} \\ \text{Polak–Ribiere} \\ \nabla\mathscr{E}_t\cdot\nabla\mathscr{E}_t/\nabla\mathscr{E}_{t-1}\cdot\nabla\mathscr{E}_{t-1} \\ \text{Fletcher–Reeves} \end{cases}.$$

Newton method

$$\mathbf{d}^{(t)} = -[\mathbf{H}(\mathbf{w}^{(t)})]^{-1}\nabla\mathscr{E}(\mathbf{w}^{(t)}).$$

Levenberg–Marquard method

$$\mathbf{d}^{(t)} = -[\mathbf{H}(\mathbf{w}^{(t)}) + \lambda_t\mathbf{I}]^{-1}\nabla\mathscr{E}(\mathbf{w}^{(t)}),$$

where λ_t is a positive constant and $\nabla\mathscr{E}(\mathbf{w}^{(t)})$ is the gradient of the function $\mathscr{E}$.

$$\nabla\mathscr{E}(\mathbf{w}) = \mathbf{J}(\mathbf{w})^{\mathrm{T}}\mathbf{E}(\mathbf{w}), \tag{6}$$

where $\mathbf{H}(\mathbf{w})$ is the Hessian matrix of the function $\mathscr{E}$

$$\nabla^2\mathscr{E}(\mathbf{w}) = \mathbf{H}(\mathbf{w}) = \mathbf{J}(\mathbf{w})^{\mathrm{T}}\mathbf{J}(\mathbf{w}) + \sum_{i=1}^{m} E_i(\mathbf{w})\mathbf{H}_i(\mathbf{w}), \tag{7}$$

where $\mathbf{J}(\mathbf{w})$ is the Jacobian matrix of vector function $\mathbf{E}(\mathbf{w})$ and $\mathbf{H}_i(\mathbf{w})$ is the Hessian matrix of the component function $E_i(\mathbf{w})$.

The convergence properties of optimization algorithms for differentiable functions depend on properties of the first and/or second derivatives of the function to be optimized. For example, steepest descent and conjugate gradient methods require explicitly the first derivative to define their search direction, and implicitly relies on the second derivative whose properties govern the rate of convergence. Correspondingly, Newton and Levenberg–Marquard methods require explicitly the first derivative and the Hessian matrix to define their search direction. When optimization algorithms converge slowly for NN problems, this suggests that the corresponding derivative matrices are numerically ill-conditioned. It is proved that these algorithms converge slowly when rank-deficiencies appear in the Jacobian matrix of a NN, making the problem numerically ill-conditioned.

It has been reported in a benchmark test study [18] that, the learning algorithm 'Rprop' achieves training in fewer number of training cycles compared to other learning algorithms. It was found, however, that normalizing the training data makes the BP algorithm to perform equally well, if not better with respect to 'Rprop' [19,20]. In the present study, the Levenberg–Marquard method is used, since it was found that this method is much more efficient than the other methods particularly when the network contains less than a few hundred weights [21].

3. The back propagation learning algorithm

In the BP algorithm, learning is carried out when a set of input training patterns is propagated through a network consisting of an input layer, one or more hidden layers and an output layer. Each layer has its corresponding units (processing elements, neurons or nodes) and weight connections. A hidden or output layer node forms its output signal out_j, by first forming the weighted sum of its input **inp**

$$\mathrm{sum}_j = \sum_{i=1}^{n} w_{j,i}\mathbf{inp}_i + b_j, \tag{8}$$

where the $w_{j,i}$ is the connecting weight between the ith neuron in the source layer and the jth neuron in the target layer and b_j is a bias parameter which acts as a function shifting term.

In the biological system, a typical neuron may only produce an output signal if the incoming signal builds up to a certain level. This output is expressed in NN by

$$\mathrm{out}_j^{(k)} = \not\prec[\mathrm{sum}_j], \tag{9}$$

where $\not\prec$ is an activation function, which produces the output at the jth neuron, k denotes that this output corresponds to the kth training data point. The type of activation function that was used in the present study is the sigmoid function, given by the expression

$$\not\prec(\mathrm{sum}) = \frac{1}{1+\mathrm{e}^{-\mathrm{sum}}}. \tag{10}$$

The principal advantage of the sigmoid function is its ability to handle both large and small input signals. The determination of the proper weight coefficients and bias parameters is embodied in the network learning process. The nodes are initialized arbitrarily with random weight and bias parameters.

A network labeled as n_1–n_2–n_3 requires $n = n_1 \times n_2 + n_2 + n_2 \times n_3 + n_3$ total number of weight and bias parameters where n_1 is the number of input nodes, n_2 is the number of the hidden layer nodes and n_3 is the number of the output layer nodes. The output of the jth hidden layer node for the training data point k is computed as follows

$$\mathrm{outh}_j^{(k)} = \not\prec\left[\sum_{i=1}^{n_1} wh_{j,i}\cdot\mathrm{inp}_i^{(k)} + bh_j\right], \tag{11}$$

where $1 \le i \le n_1$, $1 \le j \le n_2$, $wh_{j,i}$ are the weights associated with the hidden layer nodes and bh_j are the corresponding biases. Similarly, for the output layer nodes, the output of the jth output layer node for training data point k is computed as follows

$$\mathrm{out}_j^{(k)} = \not\prec\left[\sum_{i=1}^{n_2} w_{j,i}\cdot\mathrm{outh}_i^{(k)} + b_j\right], \tag{12}$$

where $1 \le i \le n_2$, $1 \le j \le n_3$, $w_{j,i}$ are the weights associated with the output layer nodes and b_j are the biases associated with the output layer nodes.

The output of the sigmoid function used lies between 0 and 1. Thus, in order to produce meaningful results using Eq. (3), the output values of the training patterns should be normalized within this range. During the training phase, the weights can be adjusted to obtain very large values, which can force all or most of the neurons to operate with large output values in a region where the derivative of the activation function is very small. Since the correction of the weights depends on the derivative of the sigmoid function, the network in this case may become virtually standstill. Initializing the weights to small random values could help to avoid this situation, although a more appropriate one is to normalize the input patterns to lie between 0 and 1.

4. The adaptive sigmoid activation function

The Jacobian matrix of a NN is composed by rows corresponding to different input training patterns and columns corresponding to the weights and biases of the hidden or output layers of the network. Thus, the Jacobian

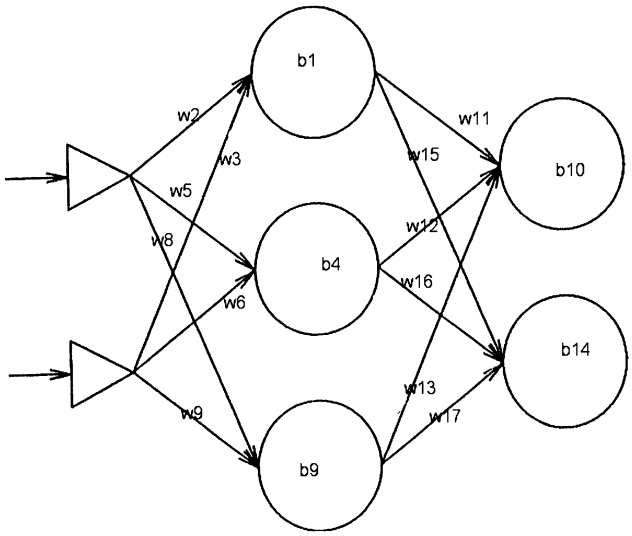

Fig. 1. The 2–3–2 network.

matrix for the 2–3–2 test case network, shown in Fig. 1, can be written as follows

$$\mathbf{J} = \begin{bmatrix} J_{1,1} & \cdots & J_{1,17} \\ \vdots & & \vdots \\ J_{k,1} & \cdots & J_{k,17} \\ \vdots & & \vdots \\ J_{m,1} & \cdots & J_{m,17} \end{bmatrix}, \tag{13}$$

where the kth row corresponds to the kth input pattern, with

$$J_{k,1} = (w_{1,1}\mathrm{out}'_1(k) + w_{2,1}\mathrm{out}'_2(k))\mathrm{outh}'_1(k),$$

$$J_{k,2} = (w_{1,1}\mathrm{out}'_1(k) + w_{2,1}\mathrm{out}'_2(k))\mathrm{outh}'_1(k)\mathrm{inp}_1^{(k)},$$

$$J_{k,3} = (w_{1,1}\mathrm{out}'_1(k) + w_{2,1}\mathrm{out}'_2(k))\mathrm{outh}'_1(k)\mathrm{inp}_2^{(k)},$$

$$J_{k,4} = (w_{1,2}\mathrm{out}'_1(k) + w_{2,2}\mathrm{out}'_2(k))\mathrm{outh}'_2(k),$$

$$J_{k,5} = (w_{1,2}\mathrm{out}'_1(k) + w_{2,2}\mathrm{out}'_2(k))\mathrm{outh}'_2(k)\mathrm{inp}_1^{(k)},$$

$$J_{k,6} = (w_{1,2}\mathrm{out}'_1(k) + w_{2,2}\mathrm{out}'_2(k))\mathrm{outh}'_2(k)\mathrm{inp}_2^{(k)},$$

$$J_{k,7} = (w_{1,3}\mathrm{out}'_1(k) + w_{2,3}\mathrm{out}'_2(k))\mathrm{outh}'_3(k),$$

$$J_{k,8} = (w_{1,3}\mathrm{out}'_1(k) + w_{2,3}\mathrm{out}'_2(k))\mathrm{outh}'_3(k)\mathrm{inp}_1^{(k)},$$

$$J_{k,9} = (w_{1,3}\mathrm{out}'_1(k) + w_{2,3}\mathrm{out}'_2(k))\mathrm{outh}'_3(k)\mathrm{inp}_2^{(k)},$$

$$J_{k,10} = \mathrm{out}'_1(k),$$

$$J_{k,11} = \mathrm{out}'_1(k)\mathrm{outh}_1^{(k)},$$

$$J_{k,12} = \mathrm{out}'_1(k)\mathrm{outh}_2^{(k)},$$

$$J_{k,13} = \mathrm{out}'_1(k)\mathrm{outh}_3^{(k)},$$

$$J_{k,14} = \mathrm{out}'_2(k),$$

$$J_{k,15} = \mathrm{out}'_2(k)\mathrm{outh}_1^{(k)},$$

$$J_{k,16} = \mathrm{out}'_2(k)\mathrm{outh}_2^{(k)},$$

$$J_{k,17} = \mathrm{out}'_2(k)\mathrm{outh}_3^{(k)}.$$

The terms of the columns of the Jacobian matrix of Eq. (13) are of the form $\mathcal{F}'(x)$, $\mathcal{F}'(x)\cdot\mathcal{F}(y)$ and $\mathcal{F}'(x)\cdot\mathcal{F}'(y)$ and in cases of networks with one output node of the form $\mathcal{F}(x)$, since $\mathrm{out}_j = \mathcal{F}(\mathrm{sum}_j)$ and $\mathrm{out}'_j = \mathcal{F}'(\mathrm{sum}_j)$. The derivative of the sigmoid function is given by

$$\mathcal{F}'(\mathrm{sum}) = \frac{\mathrm{e}^{-\mathrm{sum}}}{(1+\mathrm{e}^{-\mathrm{sum}})^2}. \tag{14}$$

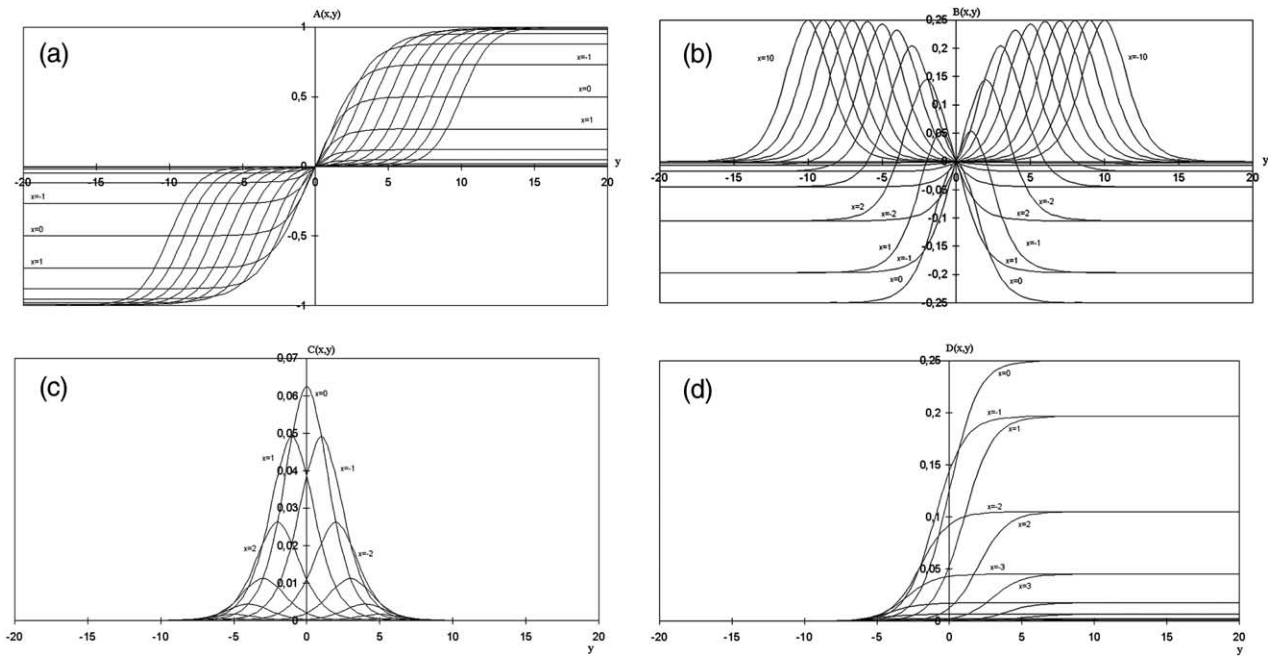

Fig. 2. Graphical representation of functions (a) $A(x, y)$; (b) $B(x, y)$; (c) $C(x, y)$; (d) $D(x, y)$.

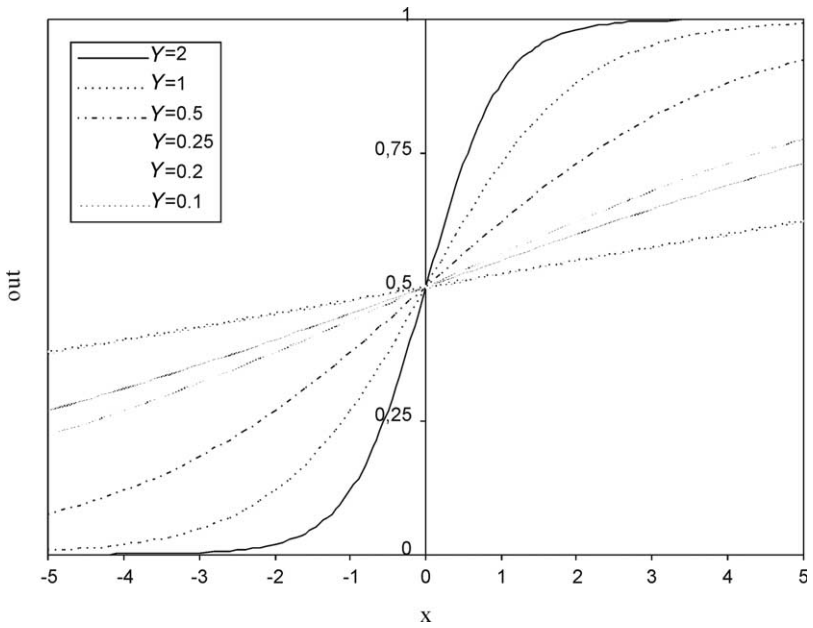

Fig. 3. The activation function for different values of the gain parameter.

To explain the near linear dependence between columns of the matrix **J** we look into the variation of the following quantities

$$A(x,y) = \not{f}(x+y) - \not{f}(x), \tag{15}$$

$$B(x,y) = \not{f}'(x+y) - \not{f}'(x),$$

$$C(x,y) = \not{f}'(x+y)\cdot\not{f}'(x),$$

$$D(x,y) = \not{f}(x+y)\cdot\not{f}'(x),$$

where $x \in [-p,p]$ and $y \in [-\delta,\delta]$. For the 2–3–2 test network, if $B(x,y)$ function do not vary for a large range of values of x and y, the columns 10 and 14 are linear dependent for this range of values of x and y. In the case of $D(x,y)$ function, the columns 11–13 and 15–17 are linear dependent when $D(x,y)$ does not vary for a large range of values of x and y.

Fig. 2 depict the graphs of functions A, B, C and D for the test case network and show the reason why the Jacobian matrix is rank deficient. $x+y$ and x are considered to be the weighted sums of the hidden and output nodes of the network. Fig. 2a gives the graph of function $A(x,y)$ for various values of $x \in [-10,10]$ and $y \in [-20,20]$. It can be seen that for large values of $|x| \geq 5$ there is no difference between the graphs. This coalescence of the curves leads to ill-conditioning. Similar observation can be made for the remaining graphs. Thus, the graphs of $B(x,y)$ for $x \in [-10,10]$ and $y \in [-20,20]$, are practically the same for $|x| \geq 5$ and $|y| \geq 14$ and $\forall y \in [-4,4]$. This means that the columns 10 and 14 of the Jacobian matrix of the test network are identical for the above range of values of x and y, since the difference between the two columns is equal to zero. In the graphical representation of $C(x,y)$ there is no difference between the graphs when $|y| \geq 7$. Similarly, in the case of Fig. 2d depicting the function $D(x,y)$ there is no difference between the graphs when $y \leq -7$ or $\forall y \in [-20,20]$ when $|x| \geq 7$.

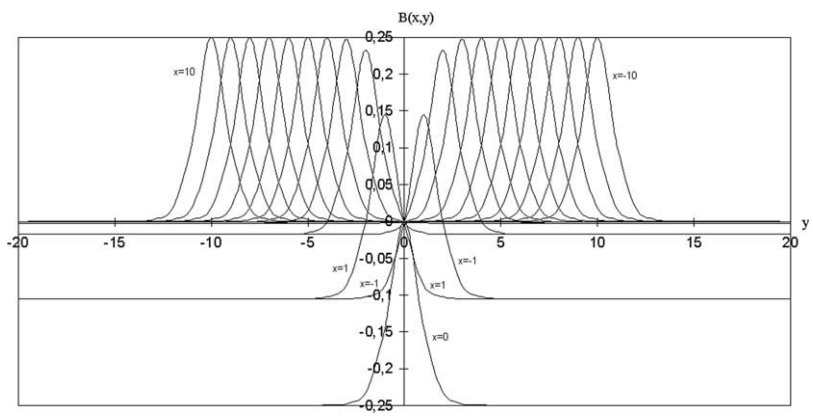

Fig. 4. Graphical representation of function $B(x,y)$ for $\gamma = 2$.

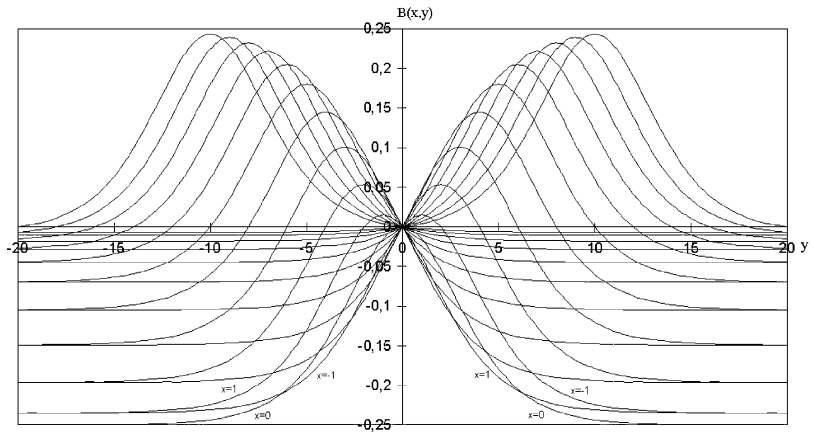

Fig. 5. Graphical representation of function $B(x, y)$ for $\gamma = 0.5$.

In order to overcome the co-linearity of the columns of **J** a modified sigmoid function is implemented in this study according to the following expression

$$f(\text{sum}) = \frac{1}{1 + e^{-\gamma \text{sum}}}, \tag{16}$$

which also has range $(0, 1)$. The parameter γ is called the gain parameter and defines the steepness (slope) of the activation function. The effect of changing the gain parameter of an activation function is shown in Fig. 3. The gain parameter moves the activation function in the direction of the horizontal axis. The modified graphs for $\gamma = 2$, 0.5, 0.25, 0.2 and 0.1 are depicted in Figs. 4–8, respectively.

In the case of $\gamma = 0.25$, for example, the graphical representations of the functions A, B, C and D, shown in Fig. 6, reveal that the graphs of Fig. 6a and b, depicting the functions $A(x, y)$ and $B(x, y)$, are quite distinct compared to the corresponding graphs of Fig. 2a and b. One can also observe from Fig. 6c and d, depicting the graphs of $C(x, y)$ and $D(x, y)$, that the curves remain distinct for a larger range of y values than the corresponding curves of Fig. 2c and d. Similar observation can be made for the other values of γ considered. The trend for the particular test case examined is that the curves for $\gamma > 1$ become more coincident compared to the corresponding graphs for $\gamma < 1$ and that when γ decreases ($\gamma < 0.25$) the range in which the curves remain distinct becomes smaller (Figs. 4, 5, 7 and 8).

As can be seen from these figures, the graphs corresponding to $\gamma = 0.25$ are more distinct in the range $x \in [-10, 10]$ and $y \in [-20, 20]$ than the graphs for the other values of γ. This value of the gain parameter can be

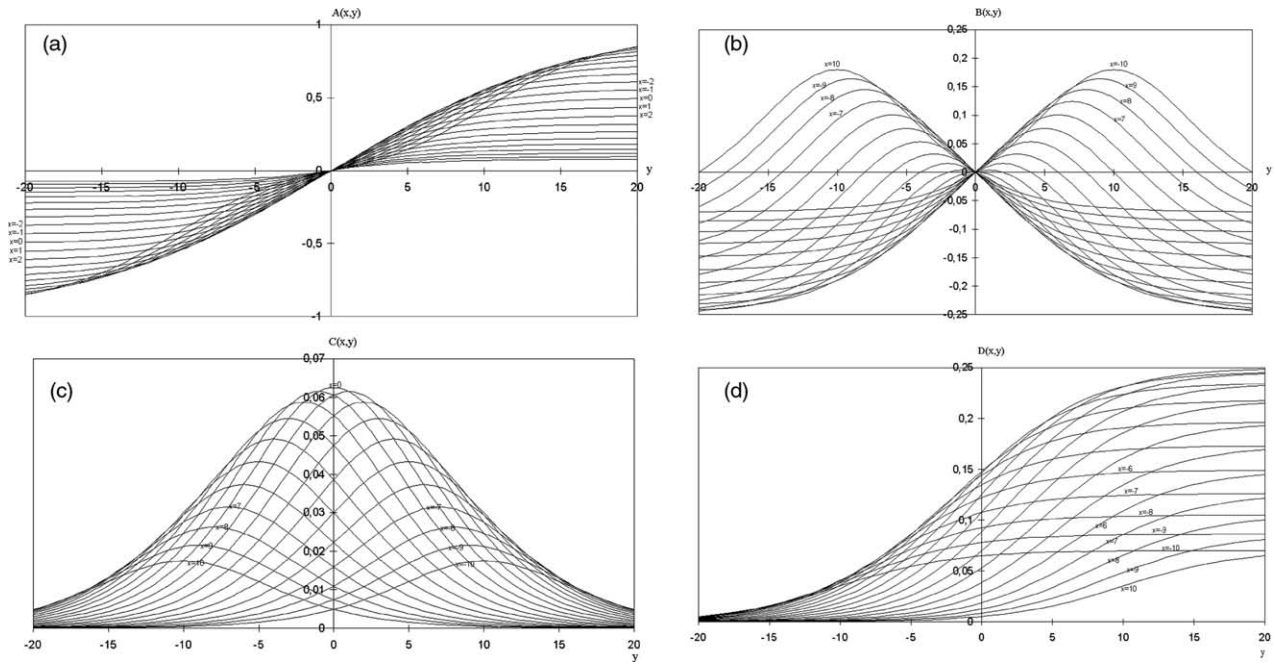

Fig. 6. Graphical representation ($\gamma = 0.25$) of functions (a) $A(x, y)$ (b) $B(x, y)$ (c) $C(x, y)$ (d) $D(x, y)$.

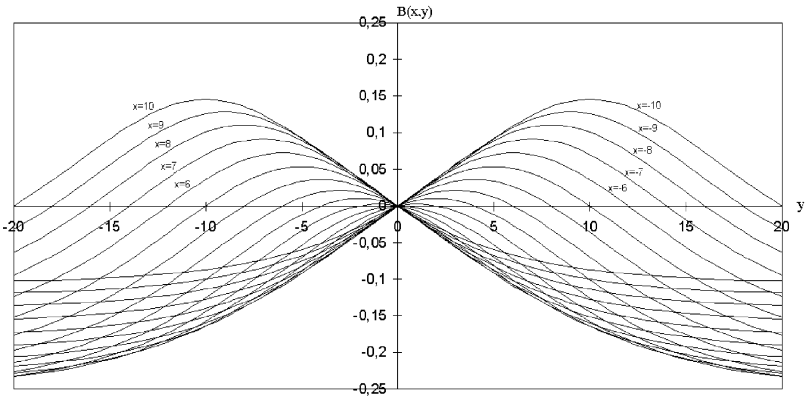

Fig. 7. Graphical representation of function $B(x, y)$ for $\gamma = 0.2$.

considered the optimum one for the particular test case in order to improve the conditioning of the network with the above characteristics. This value will be the basis for the proper selection of the gain parameter for the test examples that will be considered later. This improved performance of the modified sigmoid function motivated us to study in detail the characteristics of NN in structural optimization problems by considering either a common gain parameter for all layers of the network or by using different sigmoid functions per layer.

4.1. The basic idea of adaptation

The idea of an adaptively updated value for the gain parameter γ in Eq. (16) is motivated by the observation that the Epoch steps needed for training are affected by the values of the gain parameter γ. Epoch is called a full step of the training process comprising a forward and a backward propagation. The optimum value of the parameter γ, however, is not known a priori. Only by a trial and error procedure it is possible to find a value of the parameter γ that would perform properly. In order to overcome any trial and error procedure, that will increase substantially the cost of training, the following adaptive procedure is proposed for selecting the value of the parameter γ.

First, the size of the input signals during the learning process are monitored so as to have similar performance to the generic one shown in Fig. 6. In order to achieve better performance of the adaptive activation function, it was found that the weight parameters, which may vary considerably during the training phase, should be bound between two prespecified values. In the examples considered, the lower and upper bounds of the weight parameters were set to -10 and $+10$, respectively. The suggested adaptive scheme of the gain parameter is based on the minimum and maximum input signals for the layer nodes examined. During the training phase for each Epoch it is possible to calculate, for each layer of the network, the maximum and the minimum weighted sums. In order to maintain the condition of the Jacobian matrices similar to the generic test case we use the following procedure by projecting the generic test case to the current one:

- Calculate the maximum and the minimum weighted sums of the layer examined for the kth input pattern.
- Calculate the gain parameter for the weighted sum using the equation

$$\gamma = \frac{0.25\text{sum}}{60\dfrac{\text{sum} - \text{min}}{\text{dif}} - 30}, \tag{17}$$

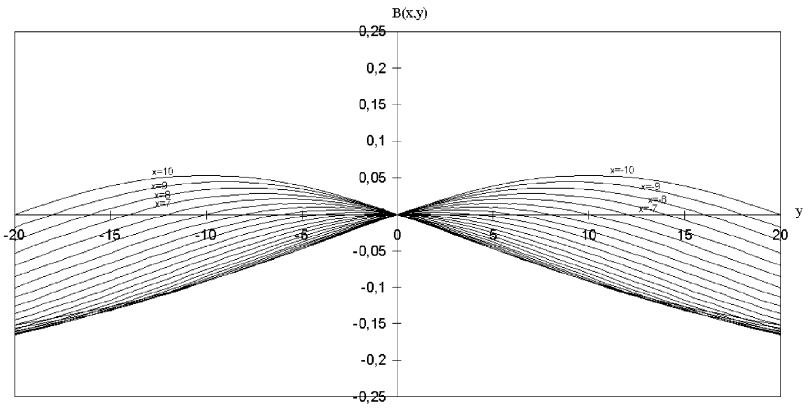

Fig. 8. Graphical representation of function $B(x, y)$ for $\gamma = 0.1$.

since we want

$$\frac{0.25}{\text{sum}_p} = \frac{\gamma}{\text{sum}}, \tag{18}$$

where sum is the current weighted sum at each node of a layer; max, min are the maximum and minimum weighted sums, respectively, among the nodes of a layer for the kth input pattern; dif = max − min and sum_p is the projected value of the sum in the range $[-30, 30]$. This range is obtained from the values of $x \in [-10, 10]$ and $y \in [-20, 20]$ giving the range of the weighted sum $x + y = [-30, 30]$.

$$\text{sum}_p = 60\frac{\text{sum} - \min}{\text{dif}} - 30. \tag{19}$$

Eq. (17) is used to project the best gain parameter found for the test case examined ($\gamma = 0.25$) to the current one.

5. Rank-deficiency

In this section, the test case 2–3–2 network is further examined in order to demonstrate the improvement achieved by the implementation of the adaptive activation function in some cases of ill-conditioned or rank-deficient Jacobian matrices. These considerations can be extended to networks with more hidden layers.

Let as consider the output vector

$$\mathbf{out}_i = [\text{out}_i^{(1)}, \text{out}_i^{(2)}, \ldots, \text{out}_i^{(m)}]^T, \tag{20}$$

which corresponds to the ith output node of $k = 1, \ldots, m$ input training patterns. Similarly, we define the output vectors $\mathbf{out}_i'$, $\mathbf{outh}_i$ and $\mathbf{outh}_i'$, corresponding to the derivatives of the ith output node, the ith hidden node and the derivative of the ith hidden node, respectively.

Case 1. If for some i, $\mathbf{outh}_i$ is a multiple of the vector $\mathbf{I} = [1, 1, \ldots, 1]^T$, then any pair of columns in the Jacobian matrix corresponding to a weight and a bias, which have $\mathbf{outh}_i$ as input, will produce on the output layer two identical columns of $\mathbf{J}$. This can be seen in Fig. 9, where for the Epoch step No. 3, $\mathbf{outh}_2$ is an exact pattern of the vector $\mathbf{I}$ while $\mathbf{outh}_3$ and $\mathbf{outh}_1$ differ in two (4, 8) and three (1, 4, 8)

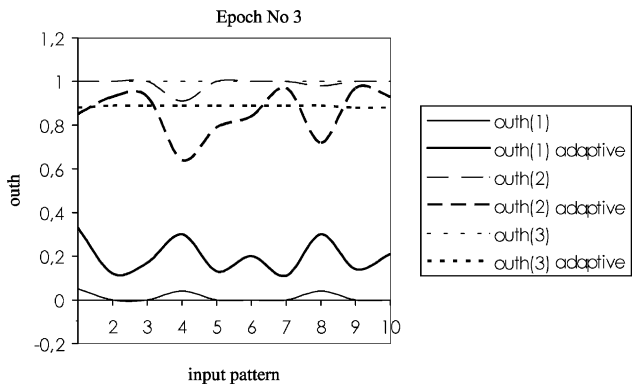

Fig. 9. Train the 2–3–2 network: outh.

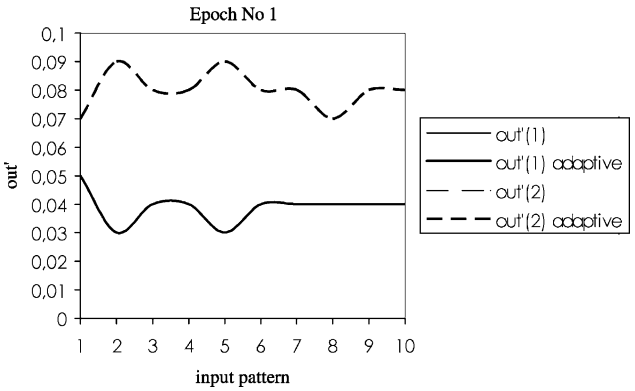

Fig. 10. Train the 2–3–2 network: out′.

input patterns, respectively. It can be seen in this figure the variation achieved, in components of the corresponding vectors, with the use of the adaptive activation function.

Case 2. If $\mathbf{out}_{i_1}'$ and $\mathbf{out}_{i_2}'$ are multiples of each other then the block of columns in $\mathbf{J}$ corresponding to the parameters of the nodes i_1 and i_2 of the output layer are identical, producing a rank deficiency. For example, for $i_1 = 1$ and $i_2 = 2$, the columns 11, 12 and 13 are linear dependent with columns 15, 16 and 17, respectively. This can be seen in Fig. 10, where for the Epoch step No. 1, $\mathbf{out}_1'$ and $\mathbf{out}_2'$ are both equal to 0. Fig. 10 depicts the variation in the components of the corresponding vectors with the use of the adaptive activation function.

Case 3. If $\mathbf{outh}_{i_1}'$ and $\mathbf{outh}_{i_2}'$ are multiples of each other then the block of columns corresponding to the first layer node i_1 of $\mathbf{J}$ (i.e. the weight and bias parameters) is a multiple of the block of columns corresponding to the first layer node i_2. For the case $i_1 = 1$ and $i_2 = 2$, the columns 1, 2 and 3 are linear dependent with columns 4, 5 and 6, respectively. This can be seen in Fig. 11, where for the Epoch step No. 1, $\mathbf{outh}_2'$ and $\mathbf{outh}_3'$ are multiples of each other. Fig. 11 depicts the variation in the components of the corresponding vectors with the use of the adaptive activation function.

Case 4. If $\mathbf{outh}_i$ and $\mathbf{outh}_j$ are multiples of each other, but are not multiples of $\mathbf{I}$ (so that it does not belong to Case 1), then the columns corresponding to the weights of the nodes i

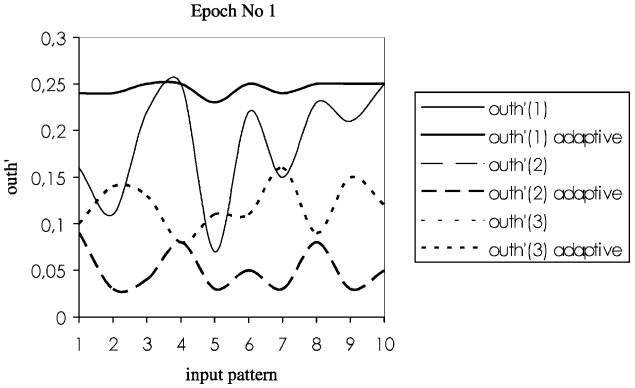

Fig. 11. Train the 2–3–2 network: outh′.

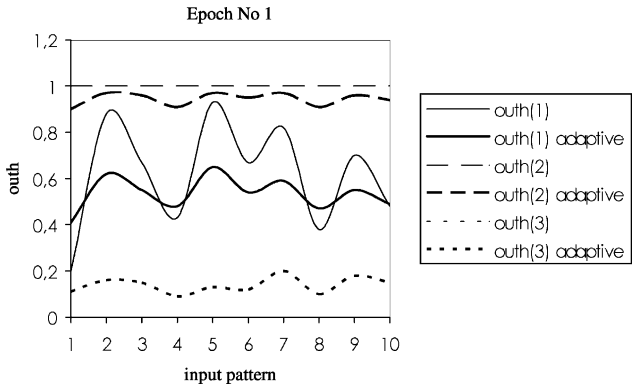

Fig. 12. Train the 2–3–2 network: outh.

and j are multiples of each other. Thus, for $i = 1$ and $j = 2$ the columns 11 and 15 are linear dependent with columns 12 and 16, respectively. This can be seen in Fig. 12, where for the Epoch step No. 1, $\mathbf{outh}_2$ and $\mathbf{outh}_3$ are multiples of each other. Fig. 12 depicts the variation in the components of the corresponding vectors with the use of the adaptive activation function.

Figs. 13–15 depict the graphs of functions B, C and D for some randomly chosen Epochs of the training process of the 2–3–2 test case network. The training set used to train the NN contains 10 randomly chosen input patterns. In these figures both simple and adaptive activation functions are present, giving a visual representation of the rank deficiency of the Jacobian matrix Fig. 13 shows the graphs of $B(x, y)$ for three randomly chosen Epochs. As can be seen for the case of the simple activation function the graphs are practically the same since the difference between the 10th and the 14th column of the Jacobian matrix is equal to zero. Form Fig. 14 it can be seen that for the case of the simple activation function there is no difference between the graphs, so the differences between the values of some of the columns 1–9

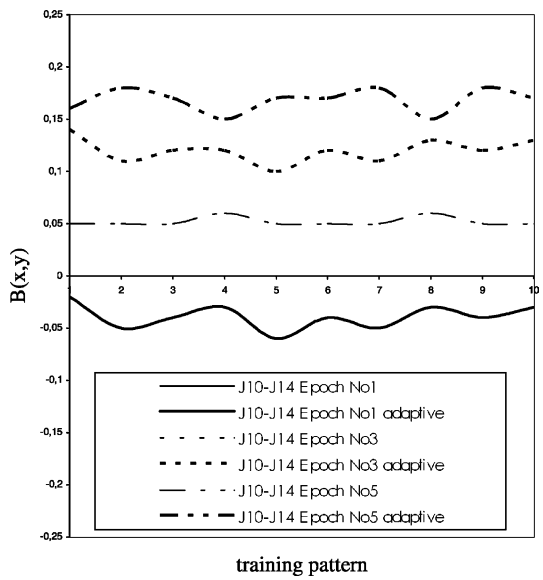

Fig. 13. Train the 2–3–2 network: $B(x, y)$.

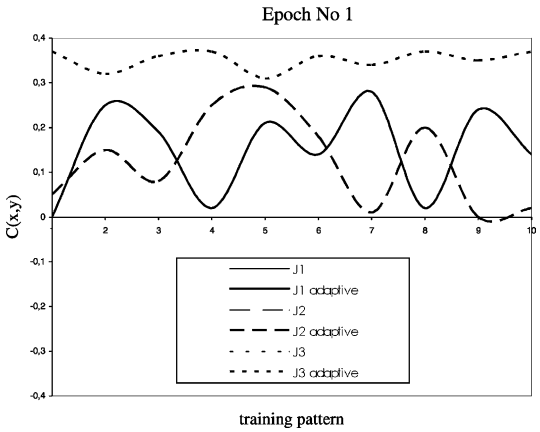

Fig. 14. Train the 2–3–2 network: $C(x, y)$.

of the matrix **J** is equal to zero, i.e. the columns are identical. Similar observation can be made from Fig. 15 where the difference of some of the columns 11–13 and 15–17 is examined.

6. Hybrid ES–NN methodology

There are two types of algorithms belonging to the class of evolutionary computation that imitate nature by using biological methodologies in order to find the optimum solution of a problem: (i) genetic algorithms (GA) and (ii) evolution strategies (ES). Both algorithms have a common characteristic when applied to structural optimization problems that of a repeated solution of a system of linear equations in order to check the suitability of the chosen design vectors. A complete survey of these methods can be found in Refs. [22,23].

In the present study, the objective is to investigate the ability of the NN to predict accurate structural analysis outputs that are necessary during the optimization process. This is achieved with a proper training of the NN. The NN training comprises the following tasks: (i) select the proper training set and (ii) find suitable network architecture. An

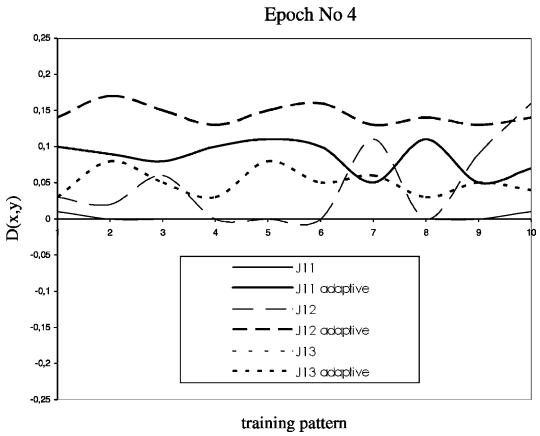

Fig. 15. Train the 2–3–2 network: $D(x, y)$.

important factor governing the success of the learning procedure of a NN is the selection of the training set. A sufficient number of input data properly distributed in the design space together with the output data resulting from complete structural analyses are needed for the BP algorithm in order to provide satisfactory results. A few tens of structural analyses have been found sufficient for the examples considered to produce a satisfactory training of the NN [2,24]. Ninety percent of those runs are used for training and the rest is used to test the results of the NN. In an effort to increase the robustness as well as the computational efficiency of the NN procedure the following training set selection scheme is adopted: the training set is chosen automatically based on a Gaussian distribution of the design variables around the midpoints of the design space. In the test examples considered in this study, we have examined sets composed by 100, 200 and 400 training patterns in order to examine the influence of the adaptive transfer function for different size of the training set. All three training sets have been produced by the Gaussian distribution selection scheme.

After the selection of the suitable NN architecture, the training procedure is performed using a number (M) of data sets, in order to obtain the I/O pairs needed for the NN training. Since the NN based structural analysis can only provide approximate results it is suggested that a correction on the output values should be performed in order to alleviate any inaccuracies entailed, especially, when the constraint value is near the limit which divides the feasible with the infeasible region. Thus, a relaxation of this limit was introduced in this study before entering the optimization procedure during the NN testing phase. Therefore, a 'correction' of the allowable constraint values was performed analogous to the maximum testing error of the NN configuration. The maximum testing error is the biggest average error of the output values among testing patterns. When the predicted values were smaller than the accurate ones derived from the normal structural analysis then the allowable values of the constraints were decreased according to the maximum testing error of the NN configuration and vice versa [2,24].

The hybrid ES–NN optimization procedure is performed in two phases. The first phase includes the training set selection, the structural analyses required to obtain the necessary I/O data for the NN training, and finally the selection, training and testing of a suitable NN configuration. The second phase is the ES optimization stage where instead of the standard structural analyses the trained NN is used to predict the response of the structure in terms of objective and constraints function values due to different sets of design variables.

The hybrid methodology ES–NN can be described with the following algorithm:

- NN training phase:
  1. *Training set selection step*: select M training patterns.
  2. *Constraints check step*: perform the check for each input pattern vector.
  3. *Training step*: selection and training of a suitable NN architecture.
  4. *Testing step*: test NN and 'correct' the allowable constraint values.
- ES optimization phase:
  1. *Parents' initialization*.
  2. *NN constraints check*: all parent vectors become feasible.
  3. *Offspring generation*.
  4. *NN Constraints check*: if satisfied continue, else and go to *step 3*.
  5. *Parents' selection step*.
  6. *Convergence check*.

7. Numerical tests

In sizing optimization problems, the aim is usually to minimize the weight of the structure under certain behavioural constraints on stress and displacements. The design variables are most frequently chosen to be dimensions of the cross-sectional areas of the members of the structure. Due to engineering practice demands, the members are divided into groups having the same design variables [25]. In structural shape optimization problems, the aim is to improve a given topology by minimizing an objective function subjected to certain constraints. All functions are related to the design variables, which are some of the coordinates of the key points in the boundary of the structure [26].

The use of NN was motivated by the time-consuming repeated structural analyses required for ES during the optimization process. The quality of NN predictions is investigated in three structural design problems optimized with ES, where the computational advantages of the proposed approach for improving the conditioning of the NN are demonstrated. In the tables containing the results of the test examples the following abbreviations are used: ES refers to the standard evolution strategies optimization procedure, in which structural analyses are performed following a numerical solution of the system of linear equations resulting from the application of the finite element method. The equation solver is based on the Cholesky factorization of the stiffness matrix, which is stored in skyline form. ES–NN refers to the combination of NN with ES optimization procedure, where the structural analysis response is predicted by a trained NN and the activation function used is the simple sigmoid function. ES–NN(a) refers to the combination of NN with ES as above but the activation function used in this case is the adaptive sigmoid function.

7.1. Sizing optimization test examples

Two benchmark test examples of space frames with 6 and 20 storeys, have been considered to illustrate the efficiency of the proposed methodology in sizing optimization problems with discrete design variables. The objective in this type of problems is to select appropriate cross-sections for the members of the structure that lead to the least possible weight and satisfy the behavioural constraints of the structure. In both examples, the modulus of elasticity is 200 GPa and the yield stress is $\sigma_y = 250$ MPa. The cross-section of each member is assumed to be an I-shape, while for each member two design variables are considered as shown in Fig. 16. The values of b and h are selected from an integer design space, while t and w are fixed ($f = 0.06h + 0.10(b - 10)$, $w = 0.625t$). Those two expressions make sure that the web thickness is less than b, the opposite of which would have not been acceptable.

The objective function of the problem is the weight of the structure. For rigid frames with I-shapes, the stress constraints, under allowable stress design requirements specified by Eurocode 3 [27], are expressed by the non-dimensional ratio q of the following formulas

$$q = \frac{f_a}{F_a} + \frac{f_b^y}{F_b^y} + \frac{f_b^z}{F_b^z} \le 1.0 \qquad \text{if } \frac{f_a}{F_a} \le 0.15 \tag{21}$$

and

$$q = \frac{f_a}{0.60\sigma_y} + \frac{f_b^y}{F_b^y} + \frac{f_b^z}{F_b^z} \le 1.0 \quad \text{if } \frac{f_a}{F_a} > 0.15, \tag{22}$$

where f_a is the computed compressive axial stress, f_b^y, f_b^z are the computed bending stresses for y- and z-axis, respectively. F_a is the allowable compressive axial stress, F_b^y, F_b^z are the allowable bending stresses for y- and z-axis, respectively, and σ_y is the yield stress of the steel. The allowable inter-storey drift is limited to 1.5% of the height of each storey. The constraints are imposed on the inter-storey drifts and the maximum non-dimensional ratio q of Eqs. (21) and (22) in each element group which combines axial force and bending moment. The values of allowable axial and bending stresses are 150 and 165 MPa, respectively, whereas the maximum allowable inter-storey drift is limited to 5.5 cm, which corresponds to 1.5% of the height of each storey.

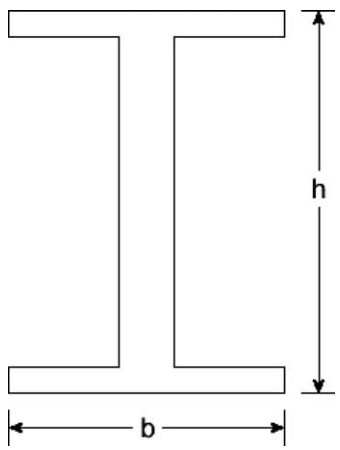

Fig. 16. I-shape cross-section design variables.

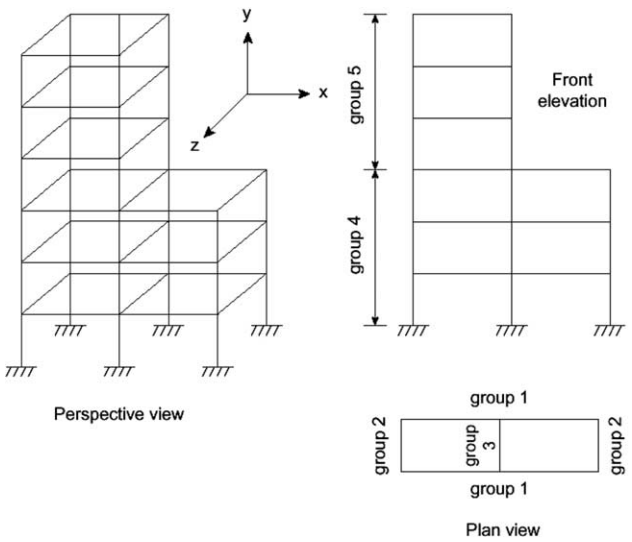

Fig. 17. Six-storey space frame.

7.1.1. Six-storey space frame

The first example is a six-storey space frame, first analyzed by Orbinson et al. [28], with 63 elements and 180 nodal degrees of freedom (dof). The length of the beams is $L_1 = 7.32$ m and the length of the columns $L_2 = 3.66$ m. The loads consist of 17 kPa gravity load on all floor levels and a lateral load of 100 kN applied at each node in the front elevation in the z direction. The element members are divided into five groups shown in Fig. 17 and the total number of design variables is 10. The constraints are imposed on the maximum allowable inter-storey drift and the non-dimensional ratio q at each element group. For this test case, the $(\mu + \lambda)$-ES approach is used with $\mu = \lambda = 5$.

Tables 1–4 depict the Epochs needed to train the network for different values of training samples and for the various transfer functions used in this study. Table 1 contains the Epochs and the time in seconds needed to train a network of size 10–20–6, for different number of training samples, when the standard sigmoid function of Eq. (10) is used. The input units are equal to the number of design variables where the six output units correspond to the maximum value of the non-dimensional ratio q of

Table 1
Six-storey space frame: performance of the standard transfer function ($\gamma = 1$)

Number of training samples	Epochs	Time (s)
100	66	953
200	127	4074
400	251	14,817

Table 2
Six-storey space frame: performance of uniform modified transfer function in the two layers of the network

γ	Number of training samples	Epochs	Time (s)
0.1	100	56	809
0.25	100	73	1054
0.5	100	86	1242
0.1	200	99	3177
0.25	200	130	4171
0.5	200	180	5774
0.1	400	280	16,528
0.25	400	220	12,987
0.5	400	238	14,049

the five element groups plus one for the value of the maximum inter-storey drift.

In Table 2, we examine the performance of the modified sigmoid function of Eq. (16) where the gain parameter γ is the same in both hidden and output layers. It can be seen that the optimum value for γ is not fixed for different numbers of training samples with a trend to move from 0.1 to 0.25 for larger number of training samples. In Table 3, we examine the use of a modified sigmoid function where the gain parameter γ is different in the hidden and the output layers. The size of the network used is the same as in the previous tests. The results indicate that there is not any specific trend in combining the values of γ in the two layers for achieving better results. Table 4 shows the performance of the adaptive sigmoid function where the gain parameter γ is not fixed but is updated during the learning process according to Eq. (17). In Table 5 the rms (Eq. (2)) and the maximum testing errors are reported, for the standard ($\gamma = 1$) and adaptive transfer functions. As it can be observed the testing error is reduced when using a largest training set.

Tables 6 and 7 depict, the performance of the proposed ES–NN methodology for various numbers of NN training patterns. These results clearly demonstrate the improvement achieved on the total computational time required for solving the optimization problem when the adaptive

Table 3
Six-storey space frame: performance of different modified transfer functions in the two layers of the network

$\gamma_1-\gamma_2$	Number of training samples	Epochs	Time (s)
0.5–0.25	100	59	852
0.25–0.1	100	60	867
0.1–0.05	100	72	1040
0.5–0.25	200	103	3304
0.25–0.1	200	95	3047
0.1–0.05	200	207	6641
0.5–0.25	400	188	10,872
0.25–0.1	400	268	15,820
0.1–0.05	400	335	19,775

Table 4
Six-storey space frame: performance of the adaptive transfer function

Number of training samples	Epochs	Time (s)
100	55	793
200	95	3049
400	184	10,761

sigmoid function is used. It should be noted that the standard ES optimization procedure without NN appears to be more efficient for this case due to the small size of the test problem considered. This was expected because of the small size of the structure (only 180 dof). The CPU improvement on the training phase and on the hybrid optimization time of the ES–NN procedure is presented in Table 7. Figs. 18 and 19 depict the training history for the simple and the adaptive sigmoid transfer functions, respectively. It can be seen that the CPU time improvement achieved in training affects the computing time required to perform the whole optimization procedure.

7.1.2. Twenty-storey space frame

The second example is the 20-storey space frame, first analyzed by Papadrakakis and Papadopoulos [29], shown in Fig. 20 with 1020 members and 2400 dof. The loads considered here are uniform vertical forces applied at joints and are equivalent to uniform load of 4.8 kPa and horizontal forces equivalent to uniform forces of 1.0 kPa on the largest surface. The element members are divided into 11 groups shown in Fig. 20 and the total number of design variables is 22. The constraints are imposed on the maximum allowable inter-storey drift and the maximum non-dimensional ratio q at each element group, as in the previous example. For this test case, the $(\mu + \lambda)$-ES approach is used with $\mu = \lambda = 10$.

Tables 8–11 depict the Epochs needed to train the network for different values of training samples and for various transfer functions. Table 7 contains the Epochs and the time in seconds needed to train a network of size 22–30–12, for different number of training samples with the standard sigmoid function. The input units are equal to the number of design variables where the 12 output units

Table 5
Six-storey space frame: NN accuracy for different number of training patters

Number of training samples	Standard transfer function ($\gamma = 1$)		Adaptive transfer function	
	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)
100	0.02	9.1	0.02	9.3
200	0.02	5.4	0.02	5.2
400	0.02	3.1	0.02	3.7

Table 6
Six-storey space frame: performance of different optimization schemes

Analysis type	Number of FE analyses/training patterns	Number of NN analyses	Computing time (s)				Optimum weight (kN)
			Analysis	Training	ES–NN	Total	
ES	281/–	–	116	–	–	116	867
ES–NN	–/100	255	40	953	3	996	883
ES–NN(a)	–/100	255	40	793	3	836	883
ES–NN	–/200	261	80	4074	3	4157	875
ES–NN(a)	–/200	261	80	3049	3	3132	875
ES–NN	–/400	275	160	14,817	3	14,980	873
ES–NN(a)	–/400	275	160	10,761	3	10,924	873

correspond to the maximum value of the non-dimensional ratio q for the 11 element groups plus one for the value of the maximum inter-storey drift.

In Tables 9 and 10 we examine the performance of the modified sigmoid functions with the same and different values for γ in the hidden and output layers, respectively. It can be seen that similar trends with the previous example apply to this test case as well. The optimum value for γ, in the first case, is 0.1 for all training samples, while marginal improvement is observed for some combinations of $\gamma_1-\gamma_2$ in the second case. Table 10 demonstrates the performance of the adaptive sigmoid function where the gain parameter γ is not fixed but is automatically updated during the learning process. The results demonstrate the favorable effect of the adaptive sigmoid function, used for the improvement on the condition of the Jacobian matrix during the training phase of NN. In Table 12 the rms (Eq. (2)) and the maximum testing errors are reported, for the standard ($\gamma = 1$) and adaptive transfer functions. The reduction of the training error follows a similar trend with respect to the size of the training set, as in the previous example.

Furthermore, a comparison of the performance of various NN training patterns and the conventional ES optimization scheme is depicted in Table 13. These results demonstrate the improvement achieved on the total optimization time. The percentage of the improvement in terms of training and the hybrid optimization techniques ES–NN is presented in Table 14.

Table 7
Six-storey space frame: CPU improvement with the adaptive scheme

Number of training samples	CPU improvement (%) in training	CPU improvement (%) in the total optimization procedure
100	17	16
200	25	25
400	28	27

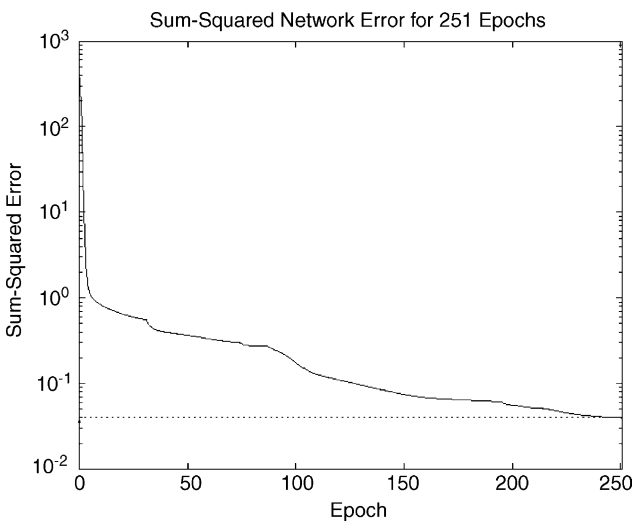

Fig. 18. Six-storey space frame: Epochs needed for the training (simple sigmoid).

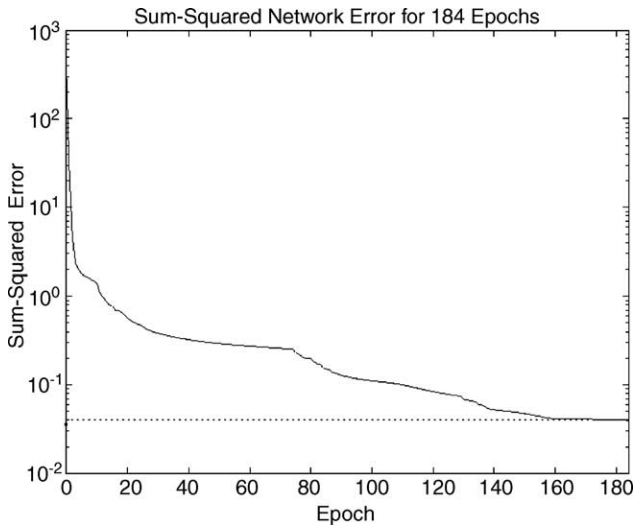

Fig. 19. Six-storey space frame: Epochs needed for the training (adaptive sigmoid).

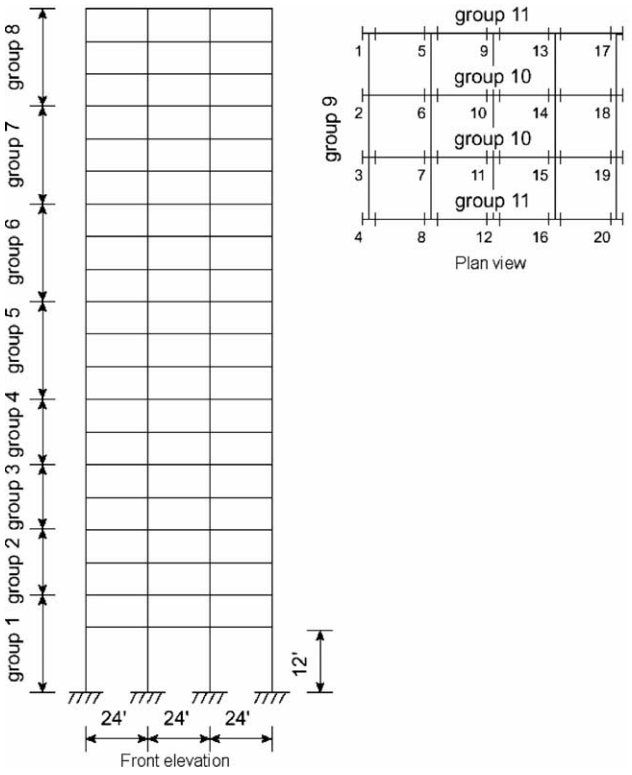

Fig. 20. Twenty-storey space frame.

7.2. Shape optimization test example

One benchmark test example [26] have been considered to illustrate the efficiency of the proposed methodology in shape optimization problems with continuous design variables. In this example, plane stress condition and isotropic material properties are assumed (elastic modulus, $E = 210,000\ \text{N/mm}^2$ and Poisson's ratio, $\nu = 0.3$).

Table 8
Twenty-storey space frame: performance of the standard transfer function ($\gamma = 1$)

Number of training samples	Epochs	Time (s)
100	47	958
200	111	4025
400	216	15,666

Table 9
Twenty-storey space frame: performance of uniform modified transfer function in the two layers of the network

γ	Number of training samples	Epochs	Time (s)
0.05	100	49	999
0.1	100	41	836
0.25	100	53	1080
0.5	100	82	1672
0.05	200	127	4606
0.1	200	97	3518
0.25	200	110	3989
0.5	200	135	4896
0.05	400	201	14,578
0.1	400	209	15,158
0.25	400	331	24,007
0.5	400	294	21,323

Table 10
Twenty-storey space frame: performance of different modified transfer functions in the two layers of the network

$\gamma_1-\gamma_2$	Number of training samples	Epochs	Time (s)
0.5–0.25	100	83	1692
0.25–0.1	100	56	1142
0.1–0.05	100	39	795
0.05–0.025	100	42	856
0.5–0.25	200	121	4388
0.25–0.1	200	93	3373
0.1–0.05	200	115	4170
0.05–0.025	200	114	4134
0.5–0.25	400	240	17,407
0.25–0.1	400	239	17,334
0.1–0.05	400	210	15,231
0.05–0.025	400	188	13,635

7.2.1. Connecting rod

The problem definition is given in Fig. 21a whereas the optimized shape is depicted in Fig. 21b. The linearly varying line load between key points 4 and 6 has a maximum value of $p = 500\ \text{N/mm}$. The objective is to minimize the volume of the structure subject to a limit on the equivalent maximum stress $\sigma_{\max} = 1200\ \text{N/mm}^2$ allowed to be developed within the structure. The design model, which makes use of symmetry, consists of 12 key points, 4 primary design variables (7, 10, 11, 12) and 6 secondary design variables (7, 8, 9, 10, 11, 12). The stress constraints are imposed as a global constraint for all Gauss points and as key point constraints are considered the points 2–6 and 12. The movement directions of the design variables are indicated by the dashed arrows.

Table 11
Twenty-storey space frame: performance of the adaptive transfer function

Number of training samples	Epochs	Time (s)
100	37	755
200	90	3262
400	185	13,419

Table 12
Twenty-storey space frame: NN accuracy for different number of training patters

Number of training samples	Standard transfer function ($\gamma = 1$)		Adaptive transfer function	
	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)
100	0.02	15.7	0.02	15.8
200	0.02	14.2	0.02	14.0
400	0.02	12.8	0.02	12.5

Table 13
Twenty-storey space frame: performance of different optimization schemes

Analysis type	Number of FE analyses/ training patterns	Number of NN analyses	Computing time (s)				Optimum weight (kN)
			Analysis	Training	ES–NN	Total	
ES	1566/–	–	24,930	–	–	24,930	5430
ES–NN	–/100	1507	1590	958	12	2560	5449
ES–NN(a)	–/100	1507	1590	755	12	2357	5449
ES–NN	–/200	1592	3180	4025	13	7218	5439
ES–NN(a)	–/200	1592	3180	3262	13	6455	5439
ES–NN	–/400	1571	6360	15,666	13	22,039	5434
ES–NN(a)	–/400	1571	6360	13,419	13	19,792	5434

Key points 8 and 9 are linked to point 7 so that the shape of the arc is preserved throughout the optimization. For this test case, the $(\mu + \lambda)$-ES approach is used with $\mu = \lambda = 10$.

Tables 15–18 depict the Epochs needed to train the network for different values of training samples and for various transfer functions. Table 13 contains the Epochs and the time in seconds needed to train a network of size 9–10–8, for different number of training samples with the standard sigmoid function.

In Tables 16 and 17 we examine the performance of the modified sigmoid functions with the same and different values for γ in the hidden and output layers, respectively. The optimum value for γ, in the first case is 0.1 for all training samples, while marginal improvements are observed for some combinations of $\gamma_1 - \gamma_2$ in the second case. Table 16 depicts is performance of the adaptive sigmoid function where the gain parameter γ is not fixed but is automatically updated during the learning process. In Table 19 the rms (Eq. (2)) and the maximum testing errors are reported, for the standard ($\gamma = 1$) and adaptive transfer functions. The reduction of the training error follows a similar trend with respect to the size of the training set, as in the previous examples.

These results demonstrate the favorable effect of the adaptive sigmoid function, used for the improvement on the condition of the Jacobian matrix during the training phase of NN. Furthermore, a comparison of various NN training patterns is depicted in Table 20. The depicted results demonstrate again the improvement achieved on the total optimization time. The percentage of the improvement in terms of training and the hybrid

Table 14
Twenty-storey space frame: CPU improvement with the adaptive scheme

Number of training samples	CPU improvement (%) in training	CPU improvement (%) in the total optimization procedure	CPU improvement (%) compared to the ES
100	21	8	91
200	19	11	74
400	14	10	21

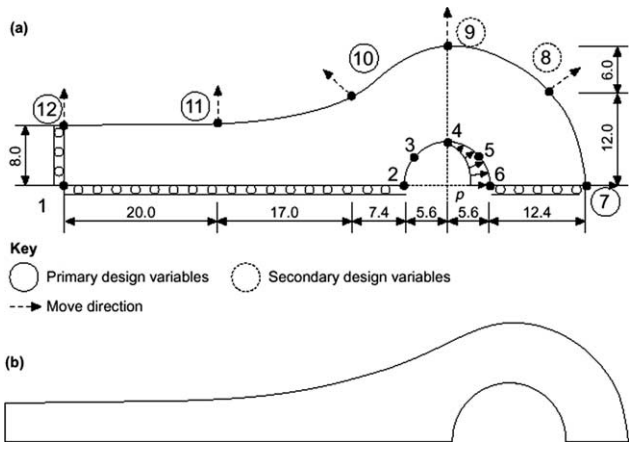

Fig. 21. Connecting rod: (a) initial shape; (b) final shape.

Table 15
Connecting rod: performance of the standard transfer function ($\gamma = 1$)

Number of training samples	Epochs	Time (s)
100	31	430
200	53	1461
400	190	9886

Table 16
Connecting rod: performance of uniform modified transfer function in the two layers of the network

γ	Number of training samples	Epochs	Time (s)
0.1	100	30	416
0.25	100	37	513
0.5	100	29	402
0.1	200	48	1323
0.25	200	56	1544
0.5	200	46	1268
0.1	400	110	5723
0.25	400	67	3486
0.5	400	95	4943

Table 17
Connecting rod: performance of different modified transfer functions in the two layers of the network

$\gamma_1-\gamma_2$	Number of training samples	Epochs	Time (s)
0.5–0.25	100	20	278
0.25–0.1	100	26	361
0.1–0.05	100	35	486
0.5–0.25	200	31	855
0.25–0.1	200	40	1103
0.1–0.05	200	37	1020
0.5–0.25	400	79	4116
0.25–0.1	400	94	4891
0.1–0.05	400	83	4318

Table 18
Connecting rod: performance of the adaptive transfer function

Number of training samples	Epochs	Time (s)
100	20	278
200	31	855
400	72	3746

Table 19
Twenty-storey space frame: NN accuracy for different number of training patters

Number of Training samples	Standard transfer function ($\gamma = 1$)		Adaptive transfer function	
	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)	$\mathscr{E}(\mathbf{w})$ error	Max testing error (%)
100	0.02	24.6	0.02	24.7
200	0.02	20.1	0.02	19.9
400	0.02	14.7	0.02	14.7

Table 20
Connecting rod: performance of different optimization schemes

Analysis type	Number of FE analyses/ training patterns	Number of NN analyses	Computing time (s)				Optimum volume (mm^3)
			Analysis	Training	ES–NN	Total	
ES	133/–	–	2617	–	–	2617	305
ES–NN	–/100	139	1967	430	2	2399	308
ES–NN(a)	–/100	139	1967	278	2	2247	308
ES–NN	–/200	137	3934	1461	2	5397	305
ES–NN(a)	–/200	137	3934	855	2	4791	305
ES–NN	–/400	137	7868	9886	2	17,756	305
ES–NN(a)	–/400	137	7868	3746	2	11,616	305

Table 21
Connecting rod: CPU improvement with the adaptive scheme

Number of training samples	CPU improvement (%) in training	CPU improvement (%) in the total optimization procedure	CPU improvement (%) compared to the ES
100	35	6	14
200	42	11	–
400	63	35	–

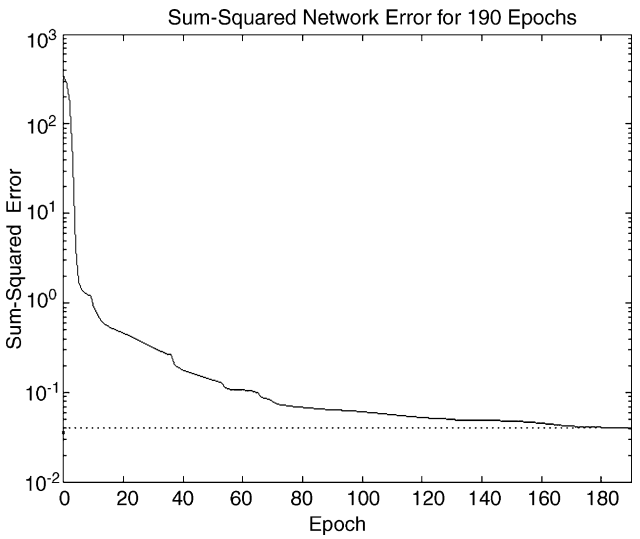

Fig. 22. Connecting rod: Epochs needed for the training (simple sigmoid).

optimization techniques ES–NN is presented in Table 21. Finally, Figs. 22 and 23 present the training history for the simple and the adaptive sigmoid transfer functions, respectively.

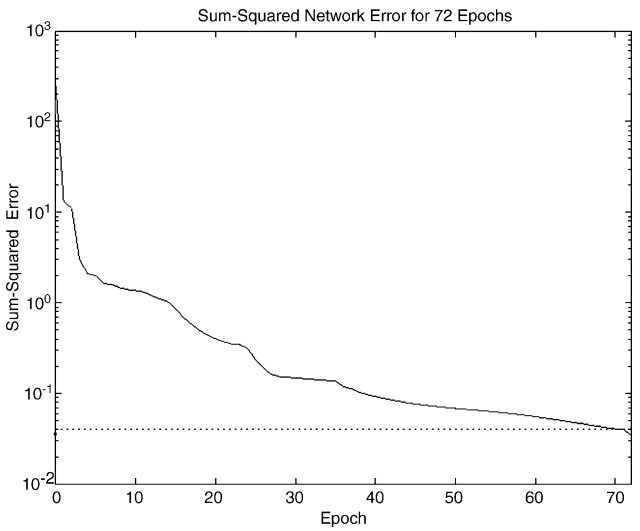

Fig. 23. Connecting rod: Epochs needed for the training (adaptive sigmoid).

8. Conclusions

The implementation of a hybrid optimization procedure, based on the combination of ES and NNs, in shape and sizing structural optimization problems was found to be very effective particularly for large-scale optimization problems. This deduction can be drawn mainly from the first test example, which is comparatively a small one (180 dof), where the NNs are not efficient compared to the conventional FE analysis in terms of the required CPU time. On the other hand, in the third example and particularly in the second example NNs significantly outperformed the conventional FE analysis. It is expected that this trend will further enhanced when largest test examples are examined.

The time-consuming requirements of repeated structural analyses associated with the optimization procedure using ES motivated the use of properly trained NNs to predict the structural response for different combinations of the design variables. The computational efficiency of the procedure is increased by using the adaptive sigmoid transfer function leading to better conditioned Jacobian matrices of the network. This has a direct influence on the training phase of the NN by decreasing the training time as well as on the total optimization time required by the ES. The computational effort involved in the optimization procedure using ES becomes excessive in large-scale problems and the use of NNs to 'predict' the necessary optimization data for ES can practically eliminate any limitation on the size of the problem. The methodology presented in this paper is an efficient, robust and generally applicable optimization procedure capable of finding the global optimum design of complicated structural optimization problems.

Acknowledgements

The authors wish to thank Matthew J. Simoneau developer of the Mathematics Group of the MathWorks, Inc for his helping tips.

References

[1] Hajela P, Berke L. Neurobiological computational models in structural analysis and design. Comput Struct 1991;41:657–67.
[2] Papadrakakis M, Lagaros ND, Tsompanakis Y. Structural optimization using evolution strategies and neural networks. Comput Meth Appl Mech Engng 1998;156:309–33.
[3] Berke L, Patnaik SN, Murthy PLN. Optimum design of aerospace structural components using neural networks. Comput Struct 1993;48:1001–10.
[4] Arslan MA, Hajela P. Counterpropagation neural networks in decomposition based optimal design. Comput Struct 1997;65(5):641–50.
[5] Shieh RC. Massively parallel structural design using stochastic optimization and mixed neural net/finite element analysis methods. Comput Syst Engng 1994;5(4–6):455–67.
[6] Adeli H, Hyo Seon P. Neural dynamics model for structural optimisation: theory. Comput Struct 1995;57(3):383–99.
[7] Adeli H, Hyo Seon P. Optimization of space structures by neural dynamics. Neural Networks 1995;8(5):769–81.
[8] Stephens JE, VanLuchene D. Integrated assessment of seismic damage in structures. Microcomput Civil Engng 1994;9(2):119–28.
[9] Papadrakakis M, Papadopoulos V, Lagaros ND. Structural reliability analysis of elastic–plastic structures using neural networks and Monte Carlo simulation. Comput Meth Appl Mech Engng 1996;136:145–63.
[10] Topping BHV, Bahreininejad A. Neural computing for structural mechanics. UK: Saxe Coburg; 1997.
[11] Khan AI, Topping BHV, Bahreininejad A. Parallel training of neural networks for finite element mesh generation. In: Topping BHV, Khan AI, editors. Neural networks and combinatorial optimisation in civil and structural engineering. New York: Civil-Comp Press; 1993. p. 81–94.
[12] Theocharis PS, Panagiotopoulos PD. Neural networks for computing in fracture mechanics: methods and prospects of applications. Comput Meth Appl Mech Engng 1993;106:213–28.
[13] Gunaratnam DJ, Gero JS. Effect of representation on the performance of neural networks in structural engineering applications. Microcomput Civil Engng 1994;9:97–108.
[14] Saarinen S, Bramley R, Cybenko G. Ill-conditioning in neural network training problems. SIAM J Sci Comput 1993;14(3):693–714.
[15] Rummelhart DE, McClelland JL. Parallel distributed processing. vol. 1, Foundations. Cambridge: MIT Press; 1986.
[16] Wasserman PD. Neural computing, theory and practice, ANZA research. New York: Van Nostrand Reinhold; 1989.
[17] Papalabros PY, Wilde WJ. Principles of optimal design: modelling and computation. New York: Cambridge University Press; 1988.
[18] Hagan MT, Menhaj MB. Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Networks 1994;5(6):989–93.
[19] Schiffmann W, Joost M, Werner R. Comparison of optimized backpropagation algorithms. Proc ESANN93, Brussels; 1993.
[20] Rognvaldsson T. On Langevin updating in multilayer perceptrons. Neural Comput 1994;6(5):916–26.
[21] Jordan MI, Bishop CM. Neural networks. A.I. Memo No. 1562, C.BC.L. Memo No. 131, MIT AI Lab; 1996.
[22] Papadrakakis M, Lagaros ND, Tsompanakis Y. Optimization of large-scale 3D trusses using evolution strategies and neural networks. Spec Issue Int J Space Struct 1999;14(3):211–23.
[23] Schwefel HP. Numerical optimization for computer models. Chichester: Wiley; 1981.
[24] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[25] Papadrakakis M, Lagaros ND, Thierauf G, Cai J. Advanced solution methods in structural optimization based on evolution strategies. Engng Comput 1998;15(1):12–34.
[26] Hinton E, Sienz J. Studies with a robust and reliable structural shape optimization tool. In: Topping BHV, editor. Developments in computational techniques for structural engineering. Edinburgh: Civil-Comp Press; 1995. p. 343–58.
[27] Eurocode 3. Design of steel structures. Part 1.1. General rules for buildings. CEN, ENV 1993-1-1/1992.
[28] Orbinson JG, McGuire W, Abel JF. Yield surface applications in nonlinear steel frames analysis. Comput Meth Appl Mech Engng 1982;33:557–73.
[29] Papadrakakis M, Papadopoulos V. A computationally efficient method for the limit elasto plastic analysis of space frames. Comput Mech J 1995;16(2):132–41.