Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers and Chemical Engineering 30 (2006) 1038–1045

Computers & Chemical Engineering

www.elsevier.com/locate/compchemeng

Method for the selection of inputs and structure of feedforward neural networks

H. Saxén*, F. Pettersson

Faculty of Technology, Åbo Akademi University, Biskopsgatan 8, FIN-20500 Åbo, Finland

Received 22 April 2005; received in revised form 19 January 2006; accepted 23 January 2006
Available online 20 March 2006

Abstract

Feedforward neural networks of multi-layer perceptron type can be used as nonlinear black-box models in data-mining tasks. Common problems encountered are how to select relevant inputs from a large set of variables that potentially affect the outputs to be modeled, as well as high levels of noise in the data sets. In order to avoid over-fitting of the resulting model, the input dimension and/or the number of hidden nodes have to be restricted. This paper presents a systematic method that can guide the selection of both input variables and a sparse connectivity of the lower layer of connections in feedforward neural networks of multi-layer perceptron type with one layer of hidden nonlinear units and a single linear output node. The algorithm is illustrated on three benchmark problems.

Keywords: Neural networks; Detection of relevant inputs; Pruning algorithm

1. Introduction

Neural networks of multi-layer perceptron (MLP) type are often used as black-box models of systems where the underlying relations are poorly known or extremely complex. In many data-mining tasks, such as those encountered in the analysis of complex industrial processes, problems arise because the signals are noisy and there are numerous factors that may potentially influence the variables to be modeled. Yet another common problem is that the number of observations is limited; this is typically the case for data sets where some of the observations correspond to properties determined in time-consuming laboratory tests. A consequence of the above mentioned is, that the number of inputs and the number of hidden nodes in the networks have to be restricted to avoid over-fitting (Principe, Euliano, & Lefebvre, 1999). If knowledge of the system studied exists, it is possible to use this for the elimination of superfluous inputs or for preprocessing of the inputs (e.g., to dimensionless groups), thus reducing the input variable dimension. Another possibility is to use statistical data preprocessing or compression techniques (e.g., Aldrich, 2002), such as principal component analysis (Jolliffe, 2002) or partial least squares procedures (Wold, 1985). These techniques are, however, often based on an assumption of linearity of the problem at hand, while the use of a nonlinear modeling tool, such as neural networks, is mainly motivated by the presence of nonlinearities.

In cases where there is little prior knowledge of the system that could be used to restrict the input dimension, one has to resort to automatic methods for selection of relevant input variables. In nonlinear modeling, there are no fully general criteria for making such choices, even though many papers have been published on how to tackle the problem. For instance, Sridhar, Bartlett, and Seagrave (1998) developed a technique where a (growing) subset of relevant inputs is formed. Since the technique is based on the interdependence between inputs and outputs detected by methods from the field of information theory, it is generally applicable, but in their examples the authors developed standard MLP networks on the detected "optimal" sets of input variables. The method does not directly take a stand on whether the input–output relation can be captured by neural networks, or on the required network complexity. Bogler (2003) proposed a dimensionality reduction algorithm, where the variance of the inputs to the hidden layer was studied, and the relative contribution of a given input was used as a measure of its relevance. Similar, but simpler, techniques are discussed by

* Corresponding author. Tel.: +358 2 215 4442; fax: +358 2 215 4792.
E-mail addresses: henrik.saxen@abo.fi (H. Saxén), frank.pettersson@abo.fi (F. Pettersson).

0098-1354/$ – see front matter © 2006 Published by Elsevier Ltd.
doi:10.1016/j.compchemeng.2006.01.007

Nomenclature

Roman

a	parameter in Eq. (4)
b	parameter in Eq. (4)
F	objective function
$\mathcal{F}$	intermediate value of objective function in pruning step
k	iteration number
K	number of observations
m	number of hidden nodes (=dim($\mathbf{z}$))
n	number of lagged signals
N	number of inputs (=dim($\mathbf{x}$))
$\mathcal{N}$	normal distribution
t	(discrete) time
u	(dimensionless) oil valve position
$\mathbf{v}$	upper-layer weight vector, with elements v_i
$\mathbf{W}$	lower-layer weight matrix, with elements w_{ij}
$\mathbf{x}$	input vector, with elements x
$\mathbf{y}$	output vector, with elements y
$\mathbf{Z}$	matrix with outputs from hidden nodes, z, cf. Eq. (2)

Greek

ε	white noise
ψ	book-keeping matrix, with elements ψ_{ij}

Subscripts

u	input
y	output

Superscripts

(k)	iteration number
$\wedge$	estimate
$\sim$	minimum

Olden and Jackson (2002); an excellent review on pitfalls in measuring the importance of inputs is provided by Sarle (2000). A traditional way to attack the problem is to use constructive methods based on a growing neural network (Fahlman & Lebiere, 1990), but these techniques are usually not well suited for problems where the input variables have a joint effect. Furthermore, their ability to solve problems with a low signal-to-noise ratio is limited. Numerous procedures for the opposite approach, i.e., pruning of large trained networks, have also been proposed (Le Chun, Denker, & Solla, 1990; Thimm & Fiesler, 1995). These techniques are usually based on statistical or heuristic criteria expressing the importance or relevance of the connections, considering the weights, the state of the hidden neurons and possibly the variance of the inputs. However, in most cases a retraining of the networks is avoided after pruning "irrelevant" connections due to the high numerical costs involved. This is a potential weakness of the methods, considering the intricate interplay between the actions of the hidden nodes in (successfully) trained neural networks. Furthermore, the risk of getting stuck in local minima, which is always present in MLP training, may result in "wrong decisions" concerning the usefulness of certain network configurations. Therefore, in more recent papers the possibility to use genetic algorithms for a simultaneous optimization of network weights and structure has been actively explored (Gao, Li, Wang, Wang, & Yue, 1999; Hinnelä, Saxén, & Pettersson, 2003; Pettersson & Saxén, 2003).

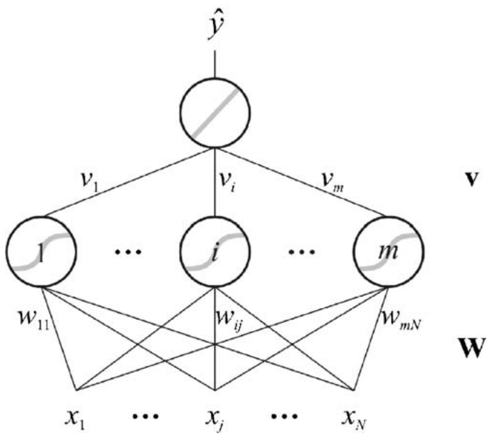

Fig. 1. Schematic of the networks used in the study.

The present paper proposes an intuitive pruning algorithm, where a sufficiently large single-hidden-layer sigmoid network with random initial weights in the lower layer of connections is used as a starting point. The complexity of this network part is gradually decreased by removing, on each iteration, the least significant connection. In comparing the networks with each other, the upper-layer weights are determined by linear least squares; this makes the algorithm efficient, rapid and robust. The procedure is described in the next subsection, followed by an illustration of it in Section 3. The paper ends with a section that presents some conclusions and proposes future prospects.

2. The algorithm

The algorithm proposed in this paper is based on feedforward neural networks of MLP type with a single layer of hidden nonlinear (typically sigmoid) units and a single linear output node (Fig. 1). The former choice is motivated by the fact that such networks have been shown to be able to approximate any continuous differentiable function to arbitrary accuracy if the number of hidden nodes is large enough (Cybenko, 1989), while an extension of the latter to networks with multiple outputs is obvious. The approach to be outlined is motivated by the practical observation that for an arbitrary, but known, choice of weights in the lower layer of connections, $\mathbf{W}$ (cf. Fig. 1) there is often a corresponding set of weights, $\mathbf{v}$, to the output node that will make the network a good approximator of the input–output relation studied.

The weights in $\mathbf{v}$ are determined as follows. After propagating the K input vectors through the first layer of connections and through the m hidden nodes, the outputs of the hidden nodes, z, are determined. Collecting these in a matrix, $\mathbf{Z}$, where a first column of ones (for the output bias) is included, the upper-layer weights can be determined by solving the linear problem:

$$\min_{\mathbf{v}}\{F = ||\mathbf{y} - \hat{\mathbf{y}}||_2\} \quad (1)$$

where $||\cdot||_2$ is the Euclidean norm, and

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_K \end{bmatrix} = \begin{bmatrix} 1 & z_{1,1} & z_{1,2} & \cdots & z_{1,m} \\ 1 & z_{2,1} & z_{2,2} & \cdots & z_{2,m} \\ 1 & z_{3,1} & z_{3,2} & \cdots & z_{3,m} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & z_{K,1} & z_{K,2} & & z_{K,m} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} = \mathbf{Zv} \quad (2)$$

The solution

$$\mathbf{v} = (\mathbf{Z}^{\mathrm{T}}\mathbf{Z})^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y} \quad (3)$$

can be determined by Householder reflections using orthogonal-triangular factorization (Golub, 1965).

The fact that an arbitrary – but well scaled – weight matrix **W** is sufficient is also in agreement with the seemingly odd practical observation reported by numerous investigators, that a large network trains more rapidly than a small network, despite its higher-dimensional weight space. The underlying reason is, obviously, that in a large network the likelihood increases of initially finding hidden nodes that operate in proper regions for solving the problem at hand. Obviously, such a large network is over-sized in a parametric sense, so it should be possible to remove superfluous connections in its lower part without major loss of accuracy of the fit.

The above reasoning is illustrated by a simple example. Consider the example function $y = 3\sin(2x) - x^2$; $x \in (-3,+3)$ depicted by a solid line in Fig. 2. From the appearance of the function and considering the resilience of neural networks, it can be deduced that a network with $m = 3 \vee 4$ hidden nodes should be able to approximate the function well. To increase the likelihood of getting hidden nodes that operate in appropriate points, networks with six hidden nodes were studied. For three such networks, the lower-layer weights were taken as random values in the interval (−3,+3), these weights were fixed and the upper-layer weights, **v**, were determined by Eq. (3). This resulted in the approximations illustrated by dotted, dashed and dash-dotted lines in Fig. 2, which are seen to be appropriate and would act as extremely good initial guesses for a further gradient-based fine-tuning of all network weights.

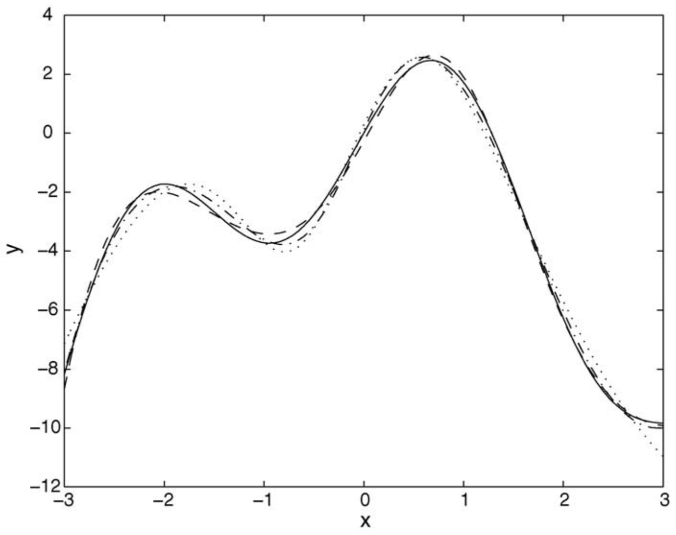

Fig. 2. Example function (—) and approximations (- - -, · · ·, - · -) by neural networks with six hidden nodes and alternative fixed random lower-layer weights and upper-layer weight vectors determined by linear least squares.

The above mentioned findings motivate the following algorithm for determining an appropriate network for a black-box modeling problem:

1. Select a set of N potential inputs, **x**, and the output, y, to be estimated for the K observations of the training set.
2. Choose a sufficient number of hidden nodes, m, and generate a random weight matrix, $\mathbf{W}^{(0)}$, for the lower layer of connections of the network. Set the iteration index to $k = 1$.
3. Equate to zero, in turn, each non-zero weight, $w_{ij}^{(k-1)}$, of $\mathbf{W}^{(k-1)}$, and determine the optimal upper-layer weights **v** by Eqs. (2) and (3). Save the corresponding value of the objective function, $F_{ij}^{(k)}$ (cf. Eq. (1)).
4. Find the minimum of the objective function values, $\mathcal{F}^{(k)} = \min_{ij}\{F_{ij}^{(k)}\}$. Set $\mathbf{W}^{(k)} = \mathbf{W}^{(k-1)}$ and equate to zero the weight corresponding to the minimum objective function value, i.e., $w_{\widetilde{ij}}^{(k)} = 0$ where $\widetilde{ij} = \arg\min_{ij}\{F_{ij}^{(k)}\}$.
5. Set $\psi_{\widetilde{ij}} = k$ and save this variable in a book-keeping matrix, $\Psi = \{\psi_{\widetilde{ij}}\}$ (of same dimension as **W**).
6. Set $k = k + 1$. If $k < mN$, go to 2. Else, end.

While the motivation for the other steps of the algorithm is obvious, the book-keeping matrix, Ψ, calls for further explanations. This matrix saves the iteration number at which each connection weight has been deleted, so by studying the elements in the columns (or rows) of the matrix it is possible to deduce when a certain input (or hidden node) has been eliminated. After passing through the steps of the algorithm, this information can be used for interpretation of the importance of the inputs (or the required complexity of the network). The algorithm bears slight resemblance with the technique proposed by Olden and Jackson (2002) in that randomized networks are used in the search. However, the latter authors only compared completely randomized networks and the pruning step was carried out afterwards on the best networks based on statistical arguments.

3. Illustration examples

The algorithm outlined in the previous section will next be illustrated by some numerical examples. First, a simple nonlinear test function with noise is used to evaluate whether the algorithm can produce sparsely connected networks with appropriate performance, and if irrelevant inputs can be eliminated. Next, a larger nonlinear (but deterministic) data set is used to demonstrate how the algorithm extracts the relevant inputs as well as lower-layer connections. In Section 3.3, the algorithm is tested on a benchmark problem with data from a hydraulic actuator of a crane arm. This example, with a single input and output, demonstrates how it can indicate useful time lags for predicting future values of the output.

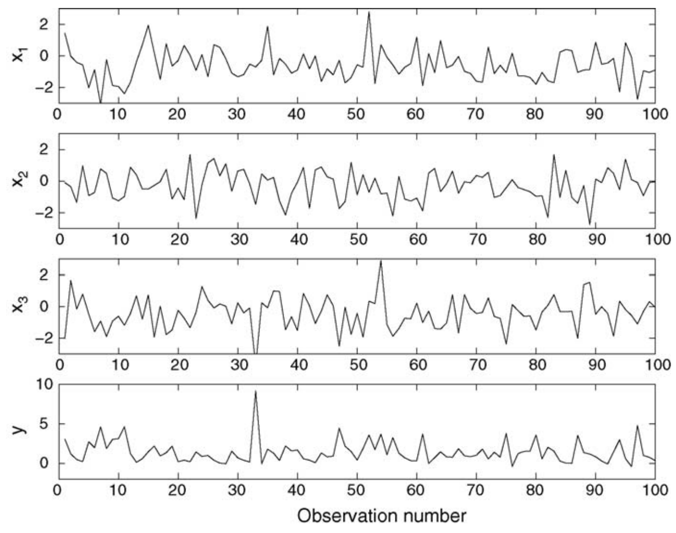

Fig. 3. Inputs and output in a training set of the reference case.

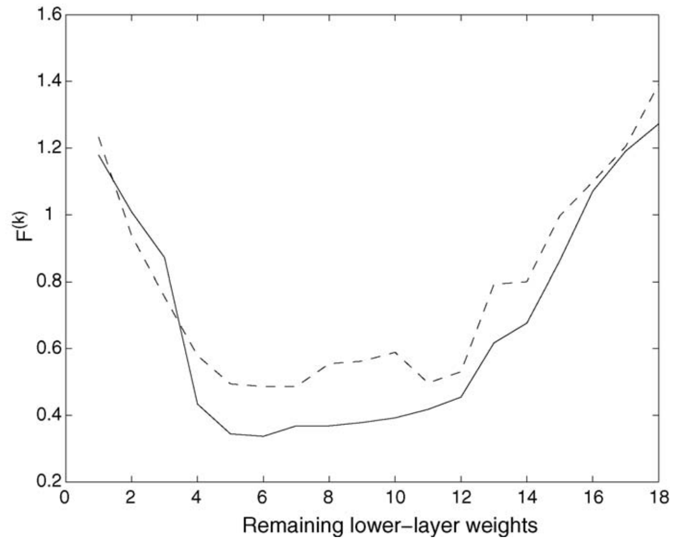

Fig. 4. Evolution of the errors with the number of remaining weights (excluding biases) in the lower part of the network (—— training set, - - - test set).

3.1. Simple test function

First, the following function with $N = 3$ inputs, (x_1, x_2 and x_3) and one output (y) is used

$$y = a(x_1^2 + 0.5x_1x_2) + (1 - a)(0.5x_2x_3 + x_3^2) + b\varepsilon. \tag{4}$$

This function is designed to yield a varying and nonlinear dependence between the inputs and the output for different values of the parameter $a \in (0,1)$. The output, y, is independent of x_3 for $a = 1$, for $a = 0$ it is independent of x_1, while for $0 < a < 1$ it depends on all three input variables. In the analysis the inputs as well as the noise term, ε, are taken to be normally distributed random variables with zero mean and unit variance, i.e., $\varepsilon, x_i = \mathcal{N}$ (0,1), while the non-negative parameter b is used to control the signal-to-noise ratio. The analysis is started from a network with six hidden nodes, which was considered sufficient for the task at hand.

3.1.1. Reference case

The algorithm was first run on 100 observations in the training set, using an additional 100 observations for verification of the resulting models, using the parameter values $a = 0.5$ and $b = 0.2$. Fig. 3 shows an example of the inputs and the output of a training set. Fig. 4 illustrates a typical result of the algorithm on the training (solid lines) and test (dashed lines) sets, where the root-mean square errors $\mathcal{F}^{(k)}$ have been depicted as a function of the remaining weights (excluding biases), $mN - k$, in the lower part of the network. Note that the steps of the algorithm progress from right to left in the figure. In summary, four to six non-zero weights are seen to be sufficient to produce an acceptable fit, and the corresponding networks generalize well on the test set. This is also seen in Fig. 5, which shows the resulting fit on the test set for the network with five remaining lower-layer weights.

Fig. 6 shows the errors on the training set for networks evolved from four random weight matrices. Even though the levels of the errors are seen to differ between the networks, the general features are very similar, especially as far as the minimum connectivity for achieving a good fit is concerned; the lower layer is seen to require 4–6 connections for good performance. A noteworthy fact is also that the fits for the largest networks are worse than those of the somewhat pruned networks, even on the training set. Thus, superfluous lower-layer connections can be directly detrimental for the model. Naturally, this is due to the fact that the lower-layer weights are “frozen”, but it still serves to illustrate the impact of the initial weights on the conditioning of the training problem.

In order to illustrate the robustness of the approach, the results are given for two additional runs, again using 100 + 100 observations and $a = 0.5$, but $b = 0.5$ or $b = 1.0$ in Eq. (4), i.e., with a lower signal-to-noise ratio. The solid lines in Fig. 7 illustrate how the training errors evolve, while the dashed lines show the corresponding errors on the test sets. The results are qualita-

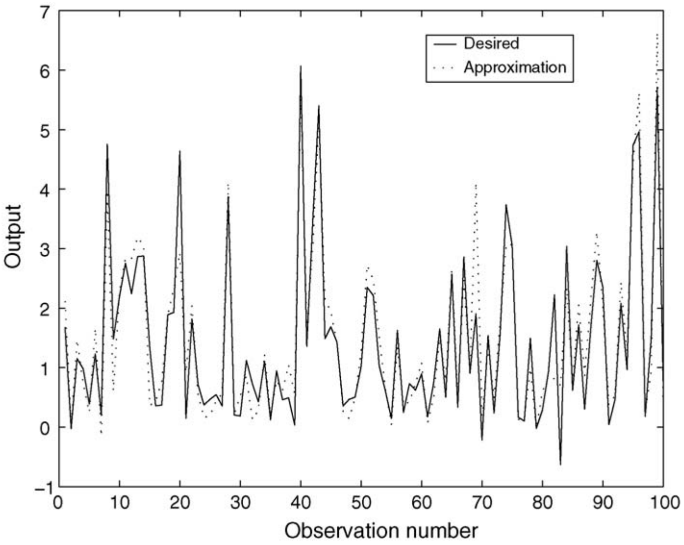

Fig. 5. Example on the model fit on the test data using five non-zero weights in the lower part of the network (—— observations, ········ model).

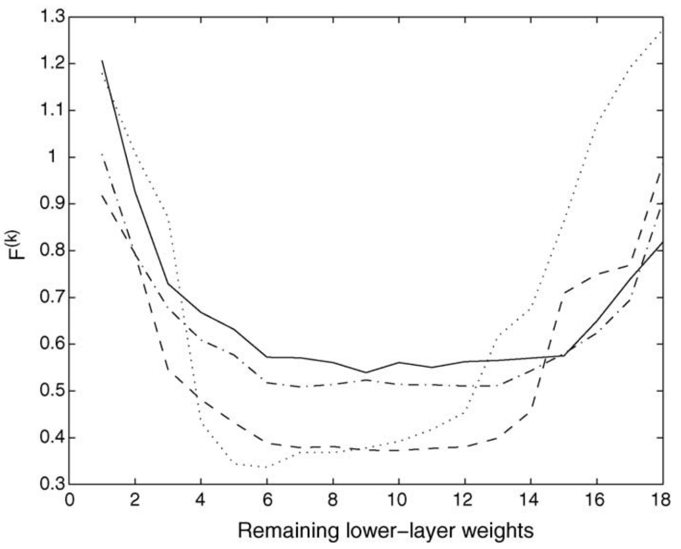

Fig. 6. Evolution of the training errors with the number of remaining weights (excluding biases) in the lower part of the networks from four random weight matrices.

tively similar to those presented above (cf. Figs. 4 and 6), which demonstrates the algorithm's insensitivity to noise.

3.1.2. Detection of relevant inputs

The capability of the algorithm to detect and remove irrelevant inputs is next evaluated by creating a data set with Eq. (4) using identical parameters to those of the first part of the previous subsection, except $a = 0.1$. This gives rise to a function y that only slightly depends on x_1, and the algorithm was used to detect the least significant input, i.e., the one that first loses all its connections to the hidden layer.

Starting from 20 random initial weight matrices, $\mathbf{W}_0$, the first input variable to be eliminated was always x_1, except in two cases where it was x_2. Table 1 shows the "frequency" of the number of remaining weight connections at the point where the final connection to x_1 was excluded, i.e., $mN - \max(\psi_{i1})$. Even though there is some scattering, it is interesting to note that the required lower-layer complexity at this point corresponds quite well to the one required for solving the task with $a = 0.5$. Another interesting observation is that the generalization ability of the networks at these very points turns out to be close to optimal for many of the runs. Fig. 8 illustrates this behavior for four of the runs, where the test-set errors have been depicted. The circles in the figure indicate the network complexity where x_1 was excluded from the model. Moving from left to right, these are seen to be points where further added complexity does not substantially improve the function approximation provided by the network.

Table 1
Frequency of the number of remaining lower-layer weights at the point where the first input (x_1) was eliminated by the pruning algorithm for the test function (4) with $a = 0.1$

Remaining weights	Frequency
3	1
4	2
5	4
6	6
7	3
8	1
9	1

3.2. Larger test problem

A further illustration of the potential of the algorithm is given by applying it to a data set with an input vector of dimension $N = 10$, with normally distributed random values with zero mean and unit variance, i.e., $x_1, \ldots, x_{10} = \mathcal{N}(0, 1)$. The dependent variable is given by

$$y = x_2 - 3x_4^2 + 2x_5x_7. \quad (5)$$

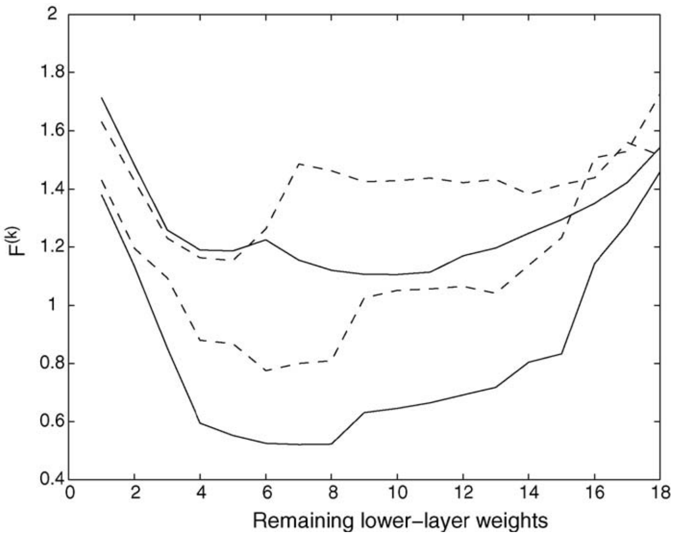

Fig. 7. Evolution of the errors (—— training set, - - - test set) with the number of remaining weights (excluding biases) in the lower part of the network for $a = 0.5$ and $b = 0.5$ or 1.0 in Eq. (4).

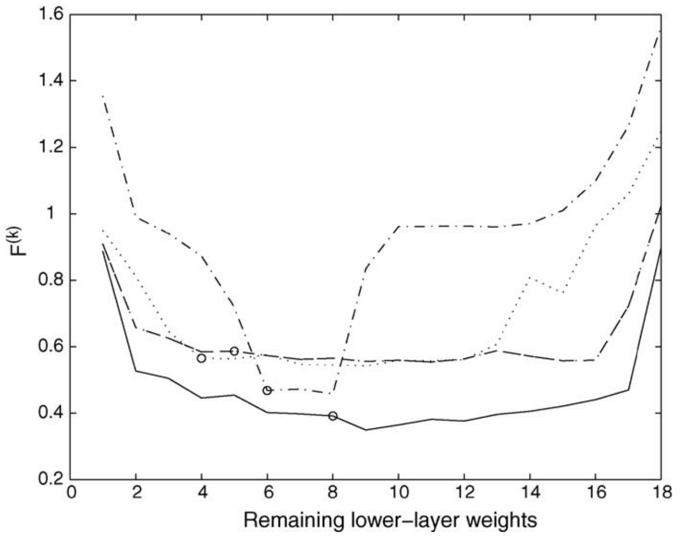

Fig. 8. Test-set errors of four networks trained on the test function (4) with $a = 0.1$. Circles denote the points where x_1 was excluded from the model.

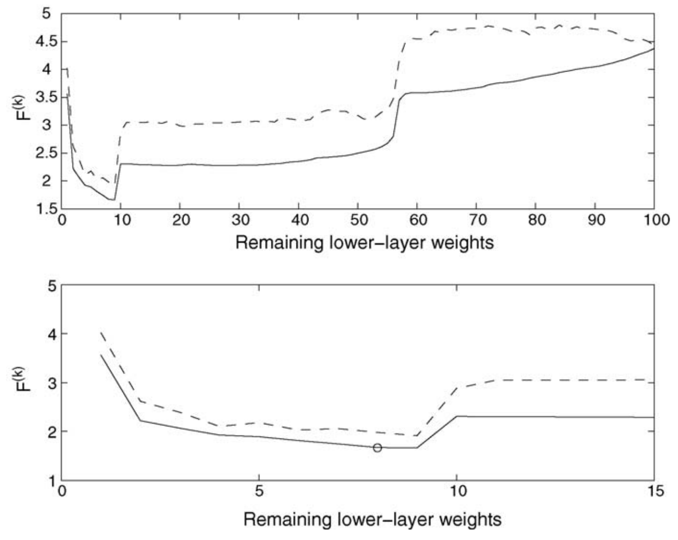

Fig. 9. Evolution of the errors (—— training set, - - - test set) with the number of remaining weights (excluding biases) in the lower part of the network for the 10-input variable test problem, Eq. (5).

Thus, six of the inputs are useless variables for capturing the input–output relation at hand. Because of the larger dimension of the problem, both the training set and the test set were given 250 observations each, using a network with $m = 10$ hidden nodes. The upper panel of Fig. 9 illustrates how the errors on the training set (solid line) and test set (dashed line) evolve along with the pruning process. The errors are seen to decrease in a step-wise manner, and, in the specific case studied, particularly strongly as the algorithm enters low network complexities (<10 lower-layer weights); the lower panel of the figure shows this region in better focus.

It is now interesting to study the arising models in more detail. If, for instance, the model with a lower-layer complexity of eight connections (indicated by a circle in the lower panel of Fig. 9) is studied, it is found to correspond to a network with seven hidden nodes and only four inputs, x_2, x_4, x_5 and x_7, i.e., only the relevant ones. A detailed view of the network – here with hidden nodes rearranged to make the interpretation easier – is given in Fig. 10, where the weights for the purpose of clarity have been rounded to three significant digits. Clearly, this model has a sparse and an intuitively appealing connectivity, where three hidden nodes are spent on approximating the square of x_4 while a joint hidden node is used in the approximation of the product between x_5 and x_7. Such a sparse network lends itself perfectly to a deeper theoretical analysis of how it constructs its approximation and of the nonlinearity of the transformations (Hinnelä et al., 2003), which may be contrasted to the use of contribution plots (see, e.g., Olden & Jackson, 2002) for more complex network architectures. Furthermore, it is obvious that the resulting sparse connections and the corresponding weights together form a good starting point for a fine-tuning of the model, e.g., by gradient-based training.

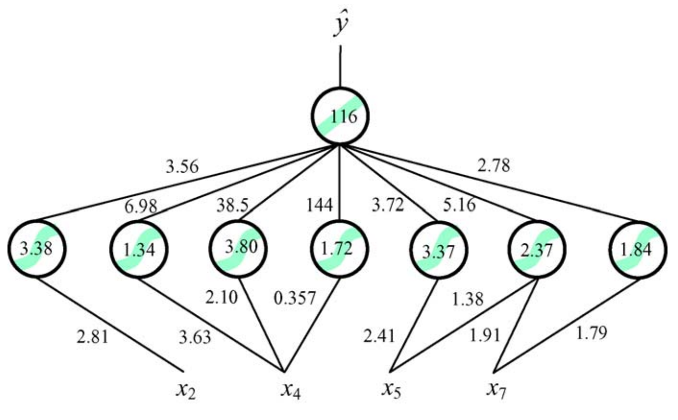

Fig. 10. Schematic of the network with a lower-layer complexity of 8 (cf. Fig. 9). Weights are reported at the connections and biases in the corresponding nodes.

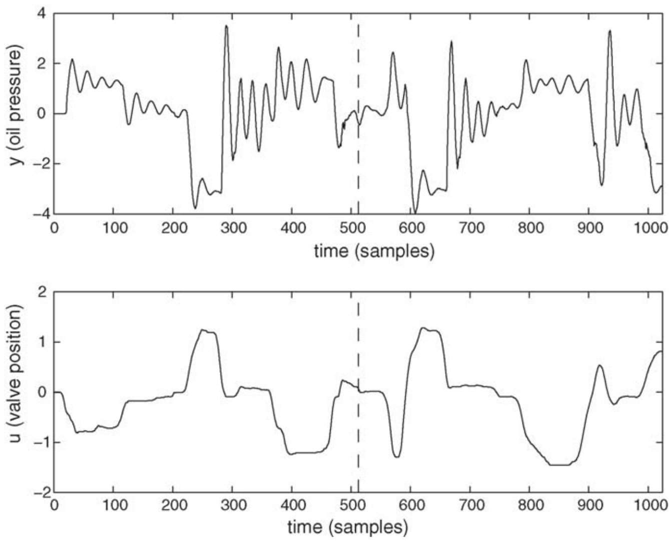

Fig. 11. Input and outputs of the crane actuator benchmark problem, with a division of the data into a training (left) and test (right) set of 512 observations each.

3.3. Crane arm actuator

The analysis of this section is based on a benchmark problem frequently used for testing nonlinear black-box models in literature, where the signals from a hydraulic actuator of a crane arm are studied. The data (downloaded from Nørgaard, Ravn, Poulsen, & Hansen, 2001) originates from the Division of Automatic Control in Linköping, Sweden (Gunnarsson & Krus, 1997), where the actuator in question is used for controlling the position of a crane by controlling a boom at the lower joint of the crane arm. The control (input) variable, u, is the opening position of the valve through which oil flows into the actuator, while the output signal, y, is the oil pressure. Fig. 11 presents the 1024 observations of the valve position (lower panel) and oil pressure (upper panel).

The goal is to carry out a successful one-step-ahead prediction of the output, $y(t)$, using the first 512 data points for training and the remaining 512 points for testing the model quality. Most approaches presented in the literature (see e.g., Sjöberg et al., 1995) have used present and previous control signals (valve opening) as well as lagged pressure signals as

Table 2
Survival number of different inputs in the crane actuator benchmark problem

Input variable	Survival number
$y(t-1)$	9.00
$y(t-4)$	5.56
$y(t-2)$	4.86
$u(t-2)$	4.76
$u(\mathrm{t}-1)$	4.42
$u(\mathrm{t}-3)$	4.38
$u(t-4)$	4.24
$y(t-3)$	3.96
$u(t)$	3.82

inputs, i.e., $\mathbf{x}(t)=[u(t),\ldots,u(t-n_u),y(t-1),\ldots,y(t-n_y)]$ with $n_u=2\vee 3$ and $n_y=2\vee 3$.

In the test of the algorithm proposed in the present paper, the input vector was extended by increasing the maximum time lags to $n_u=n_y=4$, yielding a nine-dimensional input. Starting from 50 random initial weight matrices, the algorithm was applied to determine how long different inputs would survive the connection elimination procedure. The "survival number" is obtained by sorting the maximum values of the columns of the book-keeping matrix Ψ. Table 2 reports the average survival number of the inputs, where a value of 1 would imply that the input in question was always the first one to be eliminated, while a 9 would mean that the input always survived all other inputs. The latter is seen to hold true for $y(t-1)$, which was, by far, the most relevant input. This is quite natural for a model that makes a one-step-ahead prediction of a variable that exhibits strong autocorrelation. The other top-ranked inputs are, however, more interesting. It turns out that the pressure with lags of four and two time steps as well as the valve position lagged by two time steps would be potential candidates. Naturally, it is possible that $y(t-4)$ and $y(t-2)$ are interchanged in the models evolved (since their survival numbers are about 5), so both inputs need not necessarily be present simultaneously. As for the valve position, the results indicate a clear time lag in the system between u and y, since $u(t-2)$ is the most relevant and $u(t)$ is the least relevant one.

In order to demonstrate the relevance of the detected inputs, and that the findings do not merely indicate that the suggested lower-layer connectivity is useful only for an incompletely trained network, the following approach was made. All weights and biases in three alternative fully connected small neural networks were trained (by a software based on the Levenberg-Marquardt method (Saxén & Saxén, 1995)) using 2–5 nodes in the hidden layer. The first alternative used the inputs $y(t-1)$, $y(t-4)$, $u(t-2)$ suggested by the pruning algorithm, while the other two the more "intuitive" choices $y(t-1)$, $y(t-2)$, $u(t)$, and $y(t-1)$, $u(t)$, $u(t-1)$ respectively. Table 3 reports the lowest training errors achieved starting from 20 initial weight vectors in each case as well as the corresponding test-set errors. The results show that the first input vector, detected by the pruning algorithm, gives a clearly better fit on the training set, and also lower test-set errors, while the last, rather intuitive, input vector yields poor model performance. This demonstrates how the algorithm has been successfully used for detecting the relevant inputs from a set of several potential ones.

4. Conclusions and future prospects

The paper has outlined an algorithm that can be used in black-box modeling tasks to guide the selection of relevant inputs and connectivity of single-hidden layer feedforward neural networks with a linear output node. Random initial weights in the lower layer of connections are used as a starting point, and the complexity of this part of the network is gradually decreased. On each step of the algorithm, the connection which is least significant is eliminated by a procedure, where the lower-layer weights are zeroed, in turn, and the network corresponding to the minimum approximation error is selected. In carrying out the comparison between the networks, the upper-layer connection weights are determined by linear least squares. This is a key feature of the algorithm that makes it efficient and robust. The results of the algorithm are saved in a book-keeping matrix that can be interpreted in retrospect to suggest a suitable set of inputs and connectivity of the lower part of the network.

In the test examples used to illustrate it, the algorithm has been found to be a valuable tool for the user in extracting relevant inputs from a set of potential ones. As illustrated in Section 3.2, guidelines on the required connections in the lower part of the network are simultaneously provided. The algorithm has for the purpose of illustration been applied on rather simple test problems in this paper, but it is presently being evaluated on complex data-mining problems in the metals industry, where the relations between raw material properties and process performance indices are studied (Helle & Saxén, 2005; Laitinen & Saxén, 2006). Forthcoming work will make a deeper analysis of the sparse connectivity of the lower layer of the networks for understanding the interaction between the inputs in the resulting models, and also be focused on an efficient implementation

Table 3
Training and test errors on the crane actuator data set for fully-connected neural networks with $m=2$–5 hidden nodes and three different input vectors

m	Inputs: $y(t-1)$, $y(t-4)$, $u(t-2)$		Inputs: $y(t-1)$, $y(t-2)$, $u(t)$		Inputs: $y(t-1)$, $u(t)$, $u(t-1)$	
	Training error	Test error	Training error	Test error	Training error	Test error
2	0.0902	0.1063	0.0931	0.1084	0.2019	0.2114
3	0.0827	0.1044	0.0911	0.1082	0.1916	0.2067
4	0.0811	0.0985	0.0877	0.1133	0.1824	0.2277
5	0.0763	0.1061	0.0815	0.1120	0.1669	0.3074

of the algorithm. The latter is especially important for cases where the algorithm has to be applied repetitively, such as in a multi-objective-based evolution of neural networks (Pettersson, Chakraborti, & Saxén, 2006).

References

Aldrich, C. (2002). *Exploratory analysis of metallurgical process data with neural network and related methods*. Elsevier.

Bogler, Z. (2003). Selection of quasi-optimal inputs in chemometrics modeling by artificial neural network analysis. *Analytical Chimica Acta*, *490*, 31–40.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematical Control Signals System*, *2*, 303–314.

Engelbrecht, A. P. (2001). A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, *12*, 1386–1399.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems: Vol. 2* (pp. 524–532).

Fogel, D. B. (1991). An information criterion for optimal neural network selection. *IEEE Transactions on Neural Networks*, *2*, 490–497.

Frean, M. (1989). The upstart algorithm. A method for constructing and training feed-forward neural networks. Scotland: Edinburgh Physics Department, Preprint 89/469.

Gao, F., Li, M., Wang, F., Wang, B., & Yue, P. (1999). Genetic algorithms and evolutionary programming hybrid strategy for structure and weight learning for multilayer feedforward neural networks. *Industrial and Engineering Chemistry Research*, *38*, 4330–4336.

Golub, G. (1965). Numerical methods for solving linear least squares problems. *Numerical Mathematics*, *7*, 206–216.

Gunnarsson, S., & Krus, P. (1997). Fluid power control of a flexible mechanical structure. Technical Report LiTH-ISY-R-1961. Sweden: Linköping University.

Haykin, S. (1994). *Neural networks—A comprehensive foundation*. New York: Macmillan Publishing Co.

Helle, M., & Saxén, H. (2005). A method for detecting cause-effects in data from complex processes. In B. Ribeiro, et al. (Eds.), *Adaptive and natural computing algorithms* (pp. 104–107). Wien: SpringerComputerScience.

Hinnelä, J., Saxén, H., & Pettersson, F. (2003). Modeling of the blast furnace burden distribution by evolving neural networks. *Industrial and Engineering Chemistry Research*, *42*, 2314–2323.

Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer Verlag.

Laitinen, P., & Saxén, H. (2006). Data-driven modeling of quality and performance indices in sintermaking. *Steel Research*, *77*, 152–157.

Le Chun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In D. S. Touretzky (Ed.), *Advances in neural information processing systems: Vol. 2* (pp. 598–605).

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, *5*, 39–53.

Nørgaard, M., Ravn, O., Poulsen, K., & Hansen, L.K. (2001). Neural networks for modelling and control of dynamic systems. Springer-Verlag (Data downloaded from www.iau.dtu.dk/nnbook/systems.html).

Olden, J. D., & Jackson, D. D. (2002). Illuminating the "black box": A randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling*, *154*, 135–150.

Pettersson, F., Chakraborti, N., & Saxén, H. (2006). A genetic algorithms based multiobjective neural net applied to noisy blast furnace data. *Applied Soft Computing*, doi:10.1016/j.asoc.2005.09.001.

Pettersson, F., & Saxén, H. (2003). A hybrid algorithm for weight and connectivity optimization in feedforward neural networks. In D. Pearson, et al. (Eds.), *Artificial neural nets and genetic algorithms* (pp. 47–52). Springer-Verlag.

Principe, J. C., Euliano, N. R., & Lefebvre, W. C. (1999). *Neural and adaptive systems: Fundamentals through simulations*. New York: John Wiley & Sons.

Sarle, W.S. (2000). How to measure importance of inputs, ftp://ftp.sas.com/pub/neural/importance.html.

Saxén, B., & Saxén, H. (1995). NNDT—A neural network development tool. In D. Pearson, et al. (Eds.), *International conference on artificial neural nets and genetic algorithms* (pp. 325–328). Wien: Springer-Verlag.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., et al. (1995). Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, *31*, 1691–1724.

Sridhar, D. V., Bartlett, E. B., & Seagrave, R. C. (1998). Information theoretic subset selection for neural networks. *Computers and Chemical Engineering*, *22*, 613–626.

Thimm, G., & Fiesler, E. (1995). "Evaluating pruning methods", 1995. International symposium on artificial neural networks (ISANN'95) (pp. A2 20–25). Hsinchu, Taiwan, ROC.

Wold, H. (1985). Partial least squares. In S. Kotz & N. L. Johnson (Eds.), *Encyclopedia of statistical sciences: Vol. 6* (pp. 581–591). New York: Wiley.