A simple method for selection of inputs and structure of feedforward neural networks

H. Saxén and F. Pettersson

Heat Engineering Laboratory, Åbo Akademi University, Finland
E-mail: hsaxen@abo.fi

Abstract

When feedforward neural networks of multi-layer perceptron (MLP) type are used as black-box models of complex processes, a common problem is how to select relevant inputs from a large set of potential variables that affect the outputs to be modeled. If, furthermore, the observations of the input-output tuples are scarce, the degrees of freedom may not allow for the use of a fully connected layer between the inputs and the hidden nodes. This paper presents a systematic method for selection of both input variables and a constrained connectivity of the lower-layer weights in MLPs. The method, which can also be used as a means to provide initial guesses for the weights prior to the final training phase of the MLPs, is illustrated on a class of test problems.

1 Introduction

When neural networks of multi-layer perceptron (MLP) type are used in black box modeling, e.g., for prediction of variables in a complex industrial process, a frequently occurring problem is that there are numerous factors that may potentially influence the variables to be modeled (predicted). Furthermore, it is also common that the number of observations is limited. The second problem has the consequence that the number of inputs and/or the number of hidden nodes has to be restricted to yield a meaningful parameter estimation problem (with, say, the number of network weights not exceeding a tenth of the number of output residuals [1]). This also means that all potential inputs cannot be included in the model. A remedy would be to limit the dimension of the input space by considering only meaningful variables. If knowledge of the system studied exists, it is possible to use such for the elimination of superfluous inputs or for preprocessing of the inputs, thus reducing the input variable dimension. In cases where there is no such knowledge, one has to resort to automatic methods for selection of relevant input variables. In nonlinear modeling, there are no general criteria for making such choices, even though papers have been published on how to tackle the problem [2,3]; an excellent review on pitfalls in measuring the importance of inputs is provided in [4]. A way to tackle the problem is to use constructive methods based on a growing neural network [5,6], but these techniques are often not well suited for problems where the input variables are correlated. Furthermore, their ability to solve problems with a low signal-to-noise ratio is also limited. On the other hand, pruning of large trained networks have also been proposed as a remedy [7,8], and in more recent papers the possibility to use genetic algorithms for a simultaneous optimization of network weights and structure has been explored [9-11]. However, most pruning methods are hampered by a laborious retraining process. 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 connectivities.

The present paper proposes a simple pruning approach, where a large single-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 method simple, efficient, rapid and robust. The method is described in the next subsection, followed by an illustration of it in Section 3. The final section presents some concluding remarks.

2 The Method

The method proposed in this paper is based on the following assumptions: We confine our study to feedforward neural networks of MLP type with a single layer of hidden nonlinear (typically sigmoidal) units and a single linear output node. The former limitation 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, while an extension to networks with multiple outputs is obvious.

The approach is based on the practical observation that for an arbitrary, but known, choice of weights in the lower layer of connections, **W**, (cf. Fig. 1) there are generally a corresponding set of weights, **w**, to the output node that will lead to a relatively good solution of the approximation problem at hand. After propagating the K input vectors through the first layer of connections and through the hidden nodes, the corresponding outputs of each of the n hidden nodes, can be determined. Collecting these together in a matrix, $\mathbf{Z}$, where a first column of ones (for the bias) is included, the upper-layer weights can be easily determined by solving the linear problem

$$\min_{\mathbf{w}}\left\{F = \sqrt{\tfrac{1}{K}\|\mathbf{f}-\hat{\mathbf{f}}\|^2}\right\} \qquad (1)$$

where

$$\hat{\mathbf{f}} = \begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_K \end{bmatrix} = \begin{bmatrix} 1 & z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ 1 & z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ 1 & z_{3,1} & z_{3,2} & \cdots & z_{3,n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & z_{K,1} & z_{K,2} & & z_{K,n} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \mathbf{Zw} \qquad (2)$$

by, e.g., Householder reflections using an orthogonal-triangular factorization [12].

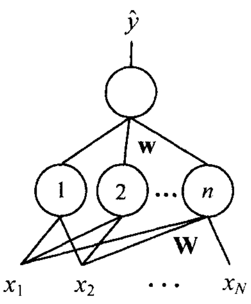

Figure 1. Schematic of the networks used in the study.

The fact that an arbitrary (but well scaled) weight matrix $\mathbf{W}$ is sufficient is also in agreement with the seemingly odd observation, reported by numerous investigators, that a large network trains more rapidly than a small network, despite its larger weight space. The reason is that in a large network the likelihood increases of initially finding hidden nodes that operate in a proper region 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. This brings us into the basic pruning step of the method that in a nutshell can be condensed into the following algorithm:

1. Select a (sufficient) maximum number of hidden nodes, n, and generate a random weight matrix, $\mathbf{W}_0$, for the lower layer of connections. Introduce a set of indices to its weights, I_0 (excluding the biases), and set the iteration index to $i = 1$.
2. Reset, in turn, a weight j in the set I_{i-1}, and determine the corresponding optimal upper-layer weights $\mathbf{w}_j$ by Eqs. (1) and (2) and determine the value of the objective function, $F_{i,j}$.
3. Set $\mathbf{W}_i = \mathbf{W}_{i-1}$ and $\mathbf{I}_i = \mathbf{I}_{i-1}$. Remove the weight, j_i^* that corresponds to the minimum value of the objective function, i.e., $j_i^* = \arg\min_{\mathbf{w}}\{F_{i,j}\}$ from both $\mathbf{W}_i$ and $\mathbf{I}_i$.
4. Set $i = i+1$. If $i < n{\cdot}N$, go to 2. Else, end.

3 Illustration of the Method

In order to illustrate the method outlined above, the following function with $N = 3$ inputs, (x_1 , x_2 and x_3) and one output (y) is used.

$$y = a\left(x_1^2 + 0.5x_1x_2\right) + (1-a)\left(0.5x_2x_3 + x_3^2\right) + b\varepsilon \qquad (3)$$

Obviously, the function is designed to yield a varying and nonlinear dependence between 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 input variables.

A few runs of the algorithm next illustrate its possible use. 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., ε, $x_i = \mathcal{N}(0,1)$, 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 Reference case

The model was first run with $K_{\mathrm{tr}} = 100$ observations in the training set, using an additional $K_{\mathrm{te}} = 100$ observations in a test set for verification of the resulting models, using the parameter values $a = 0.5$ and $b = 0.2$. Figure 2 shows an example of the inputs and the output of a training set.

Figure 3 illustrates a typical evolution of the method on the training (solid lines) and test (dashed lines) sets, where the root-mean square errors F_{i,j^*} (cf. Eq. (1)) have been depicted as a function of the remaining weights (excluding biases) in the lower part of the networks. Note that the "iterations" progress from right to left in the figure. In summary, five to seven non-zero weights (to five active hidden nodes) are sufficient to produce an acceptable fit, and the corresponding networks generalize well on the test set. This is also seen in Fig. 4, which shows the resulting fit on the test set for the network with five remaining lower-layer weights.

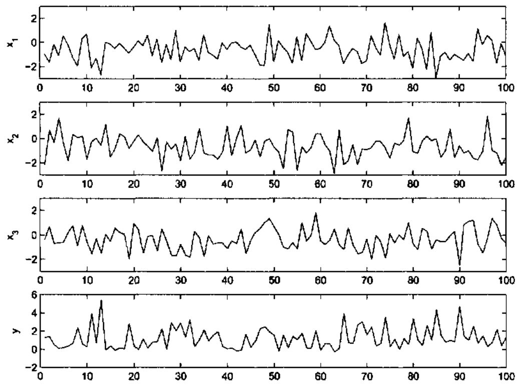

Figure 2. Inputs and output in a training set of the reference case.

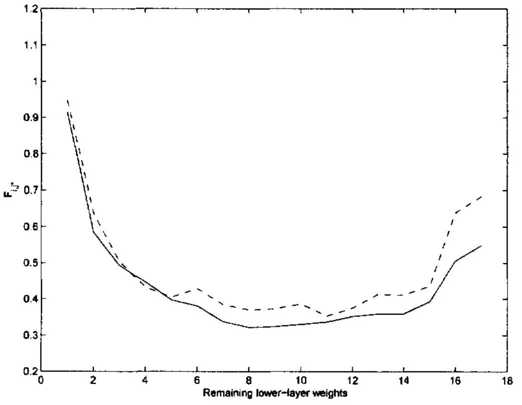

Figure 3. Evolution of the errors with the number of remaining weights (excluding biases) in the lower part of the network (—— training set, - - - test set).

Figure 5 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. 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. This is the result of the fact that the lower-layer weights are not retrained, but it still serves to illustrate the impact of the initial weights on the conditioning of the training problem. Therefore, the proposed technique can also be seen as a method for initializing a (sparse) network.

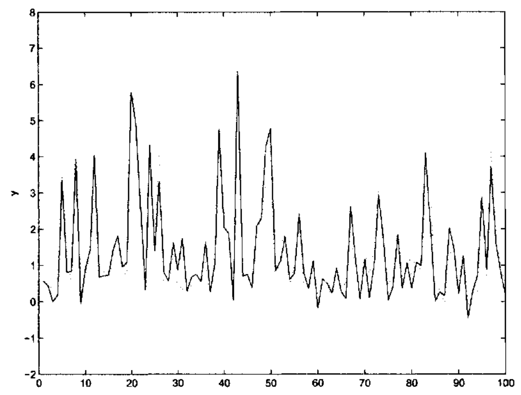

Figure 4 Example on the model fit on the test data using five non-zero weights in the lower part of the network (—— observations, ······ model).

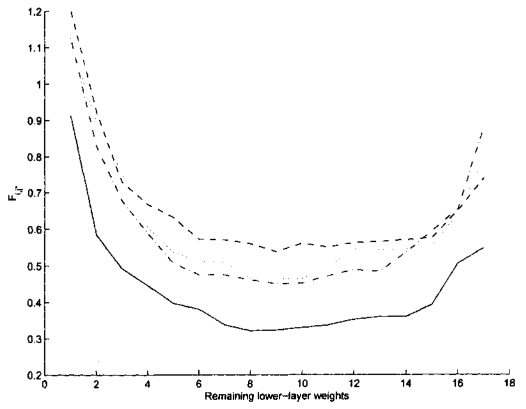

Figure 5. Evolution of the training errors with the number of remaining weights (excluding biases) in the lower part of the networks from four random weight matrices.

3.2 Detection of relevant inputs

The capability of the model to detect and remove irrelevant inputs is next evaluated by creating a data set with identical parameters to those of the previous subsection, except $a = 0.1$. This gives rise to a function y that only slightly depends on x_1. The algorithm was used to detect the one of the inputs that would be first eliminated, i.e., would lose 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. Table 1 shows the "frequency" of the number of remaining weight connections at the point where the final connection to x_1 was excluded. 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. Figure 6 illustrates this behavior for four of the runs, where the test set errors have been depicted. The arrows in the figure indicate the network complexity where the first input variable (x_1) was excluded from the model. Moving from left to right, these are the points where a further added complexity does not improve the function approximation provided by the network.

Table 1. Frequency of remaining lower-layer weights at the point where the first input (x_1) was eliminated by the pruning method for the function (3) with $a = 0.1$.

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

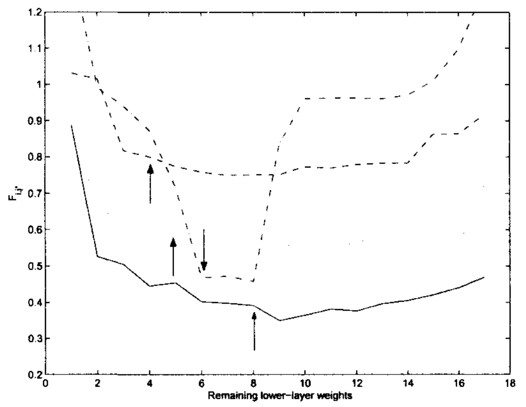

Figure 6. Test set errors of four networks trained on y generated by Eq. (3) with $a = 0.1$. Arrows denote the points where x_1 was excluded from the model.

4 Conclusions

The paper has described a systematic method for selection of both input variables and a constrained connectivity of the lower-layer weights in MLPs. It also provides initial guesses for a gradient based training. The method has been illustrated on a class of test problems, where it has shown promising performance.

REFERENCES

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

[2] Sridhar, D.V., E. B. Bartlett and R. C. Seagrave, (1998) "Information theoretic subset selection for neural networks", *Comput. Chem. Engng.* **22**, 613-626.

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

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

[5] Frean, M., (1989) "The Upstart Algorithm. A method for Constructing and Training Feed-forward Neural Networks", Edinburgh Physics Department, Preprint 89/469, Scotland.

[6] Fahlman, S.E. and C. Lebiere, (1990) "The Cascade-Correlation Learning Architecture", in *Advances in Neural Information Processing Systems* **II**, (Ed. D.S. Touretzky), pp. 524-532.

[7] Le Chun, Y., J. S. Denker and S. A. Solla, (1990) "Optimal Brain Damage", in *Advances in Neural Information Processing Systems* **2**, ed. D.S. Touretzky, pp. 598–605, (Morgan

[8] Thimm, G. and E. Fiesler, (1995) "Evaluating pruning methods", Proc. of the *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, Hsinchu, Taiwan, ROC.

[9] Maniezzo, V., (1994) "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", *IEEE Transactions on Neural Networks* **5**, 39–53.

[10] Gao, F., M. Li, F. Wang, B. Wang and P. Yue, (1999) "Genetic Algorithms and Evolutionary Programming Hybrid Strategy for Structure and Weight Learning for Multilayer Feedforward Neural Networks", *Ind. Eng. Chem. Res.* **38**, 4330-4336.

[11] Pettersson, F. and H. Saxén, (2003) "A hybrid algorithm for weight and connectivity optimization in feedforward neural networks", in *Artficial Neural Nets and Genetic Algorithms* (eds. Pearson, D. et al.), pp. 47-52, Springer-Verlag.

[12] Golub, G., (1965) "Numerical methods for solving linear least squares problems", *Numer. Math.* **7**, 206–216.