Proceedings of the 1998 IEEE
International Conference on Control Applications
Trieste, Italy 1-4 September 1998

A NEURAL NETWORK PRUNING ALGORITHM WITH EMBEDDED GRADIENT-CONJUGATE TRAINING FOR THE IDENTIFICATION OF LARGE FLEXIBLE SPACE STRUCTURES

Xiao-Hua Yu
Department of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92697, USA
E-mail: xhyu@ece.uci.edu

Abstract

The choice of network dimension is a fundamental issue in the design of artificial neural networks. A larger neural network is powerful for solving problems while a smaller neural network is always advantageous in real-time environment where speed is crucial. In this paper, a network pruning algorithm with embedded gradient-conjugate training is investigated and applied to the identification of a large flexible space structure. Computer simulation results show that this approach can dramatically reduce the size of neural network while maintaining compatible identification accuracy.

I. Introduction

System identification is a key issue of adaptive filtering, prediction and control. Unlike conventional approaches, the neural network (NN) approach does not require a priori knowledge on the system model and thus has been studied by more and more researchers. It has been proved that multi-layer feedforward artificial neural network is capable of representing any measurable function within any desired degree of accuracy with the correct values of weights and sufficient number of hidden units; however, how to train these weights efficiently and how to determine the number of nodes and weights are still open problems. Back-propagation is one of the most popular training algorithms; however, in some cases it converges very slowly. In general, if training is started with a neural network which is too small for a specific problem, this neural network may never be trained to solve the problem (for example, the famous XOR problem requires at least one hidden unit, otherwise learning may not happen). On the other hand, since the computational cost is proportional to the number of the arithmetic operations which are determined by the number of nodes and the connections between them, in the real-time implementation where speed is crucial, a smaller network is always expected. Generally speaking, there are two ways to determine the suitable size of a neural network. One possible way is to start with a small neural network, then add more hidden nodes and hidden layers in order to improve the convergency. The second approach, called net pruning, employs a neural network which is larger than the minimum size required for solving the problem at the beginning; then obtains a smaller neural network by reducing its size. In general, a larger NN may have the advantage of a faster rate of convergence even though it takes longer computation time. Therefore in practice, one would rather overestimate the network size than underestimate it.

In the following sections, a pruning algorithm with embedded gradient-conjugate training based on the second approach is developed and applied to the system identification problem.

II. The neural network pruning algorithm with embedded gradient-conjugate training for system identification

Fig. 1 illustrates a general block diagram of the pruning algorithm as applied to the identification of unknown plants. For the sake of simplicity, the plant is assumed to be

0-7803-4104-X/98/$10.00 ©1998 IEEE

stable with a single input and a single output. The input and output sequences of the plant to be identified are {u(k), y(k); k=1, ... , N} where N is the total number of samples over discrete time index k. A multi-layer feedforward neural network is employed as the identification model whose output is denoted by $\{\hat{y}(k); k=1, \ldots, N\}$. This neural network is comprised of three layers. The first layer has n fixed nodes where one node receives input from u(k), and the other (n-1) nodes receive inputs from (n-1) delayed values of the plant output y(k). The second, or the hidden layer, has a variable set of nodes, and the output layer has a single node. The "pruning" algorithm, which estimates the sensitivity of the total identification error to the inclusion/exclusion of each weight in the neural network, is employed to determine that from the identification error sequence $\{e(k); k=1, \ldots, N\}$, which weights of the network can be "pruned" (i.e., removed) without affecting the identification accuracy.

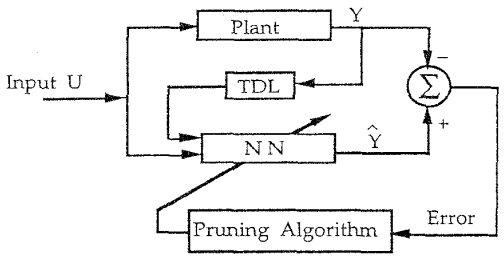

Fig. 1. Block diagram for the pruning algorithm

Mozer and Smolensky [1] defined S_{ij}, the sensitivity of a global error function with respect to w_{ij}, as the following:

$$S_{ij} = E(w_{ij} = 0) - E(w_{ij} = w_{ij}^f)$$

$$= E(\text{without } w_{ij}) - E(\text{with } w_{ij}) \qquad (1)$$

where w_{ij}^f is the final value of weight w_{ij} after training and $E(\cdot)$ is the identification error function:

$$E(W) \stackrel{\Delta}{=} \frac{1}{2} \sum_{k=1}^{N} [\hat{y}(k, W) - y(k)]^2 \qquad (2)$$

where W is a vector comprised of all the weights in the neural network.

One of the main drawbacks in calculating the sensitivity according to Eq. (1) is that one has to compute it for every weight which is a candidate for elimination through out all training phases, and hence for large networks this can be extremely time consuming. Karnin [2] suggested that the sensitivity in Eq. (1) can be approximated by:

$$S_{ij} = -\frac{E(w_{ij}^f) - E(0)}{w_{ij}^f - 0} w_{ij}^f \qquad (3)$$

assuming all other weights are fixed (at their final states, upon completion of training).

To further simplify Eq. (3), let's consider the example of a network with only two weights, denoted by u and w. For this case the numerator in Eq. (3) becomes

$$E(u^f, w^f) - E(u^f, 0) \qquad (4)$$

where only the contribution due to changes in w is taken into account. In Fig. 2 , the error E(u, w) is illustrated by constant value contours. The initial point in the weight space is designated by I in Fig. 2; and the learning path is the solid line from I to F, the final point. The numerator of Eq. (3) can then be evaluated as:

$$E(w=w^f) - E(w=0) = \int_A^F \frac{\partial E(u^f, w)}{\partial w} dw \qquad (5)$$

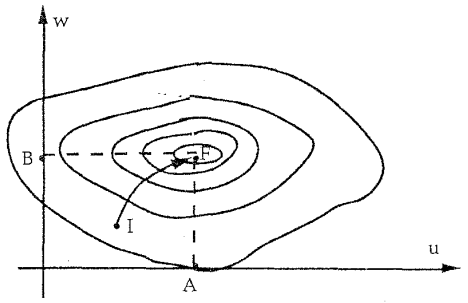

Fig. 2. Learning as motion along the error surface

This integral is along the line from point A which corresponds to w=0 (while all other weights are in their final states), to the final weight state F. However, a typical training process starts with some small initial value (e.g., point I) rather than 0. Therefore,

$$E(w=w^f)-E(w=0) \cong \int_I^F \frac{\partial E(u,w)}{\partial w} dw \qquad (6)$$

This expression can be further approximated by replacing the integral by summation, taken over the discrete steps that the network passes while learning. Thus the estimated sensitivity to the removal of connection w_{ij} becomes

$$\hat{S}_{ij} = -\sum_0^{M-1} \frac{\partial E}{\partial w_{ij}}(n)[\Delta w_{ij}(n)]\frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i} \qquad (7)$$

where M is the total number of iterations needed to minimize Eq. (2).

Eq. (7) indicates that the values of the sensitivities are also related with the neural network adaptation law. For different training methods, the way for calculating the gradient is the same, but the way for evaluating ΔW is varied depending on each algorithm. That means, even though we start with the same size of neural network, the same initial value for every weight, the sensitivity matrix may still be distinct if the neural network is trained by different training rules. An appropriate training algorithm may improve the identification accuracy and reduce the size of neural network even further.

Back-propagation (steepest descent) is generally a very efficient training algorithm for multi-layer feedforward neural network, however, its rate of convergence may be very slow in some cases. This may be due to the fact that the direction of the gradient vector may not directly point toward the minimum point of the error surface. To accelerate the rate of convergence, second order back-propagation which involves the inverse matrix of the second order derivative of the error function (Hessian matrix) has been investigated by several researchers. However, the arithmetic operation for calculating the second order derivative and the inverse matrix adds the burden of both computational time and memory requirement. In addition, the inverse Hessian matrix may not even exist.

The Fletcher and Reeves conjugate gradient algorithm combines the advantage of both first and second order methods while attempting to eliminate their disadvantages. Based on the Gram-Schmidt orthogonization principle, this algorithm searches the new gradient in each iteration along such direction which is constructed to be conjugate to all the previous directions traversed. In other words, the basic idea is to set up a new conjugate direction p_k after each one dimensional minimization and thus to achieve faster rate of convergence:

$$p_k = \frac{\partial E}{\partial W}(k); \qquad (k=1) \quad (8)$$

$$p_k = \frac{\partial E}{\partial W}(k) + \frac{\left\|\frac{\partial E}{\partial W}(k)\right\|^2}{\left\|\frac{\partial E}{\partial W}(k-1)\right\|} \cdot p_{k-1}; \; (k \geq 2) \quad (9)$$

where k is the number of iteration. The weights of the neural network, W, are then updated by:

$$W(k+1) = W(k) - \alpha_k p_k \qquad (10)$$

where α_k is a scalar.

Once the neural network identification model is trained to achieve the input/output mapping with the desired accuracy, the pruning algorithm can be activated to remove the unnecessary weights. Since the partial derivatives $\partial E/\partial w_{ij}$ which are the components of the gradient are always available during training, the only extra computational demand for implementing the pruning procedure is the summation in (7). This (negligible) overhead merely calls for maintaining a "shadow array" (of the same size as the number of connections in the network) that keeps track of the accumulated terms that build up S_{ij} in (7).

The number of weights eliminated also depends on when the pruning process starts. Longer training time before pruning results smaller neural network and complete training leads to the minimum neural network size. In addition, if training is continued after pruning, the size of the neural network can even be further reduced.

III. Application to Large Flexible Space Structures

In this section, the above neural network pruning algorithm is applied to the identification of a finite dimensional subsystem

model of the Jet Propulsion Laboratory/AFAL flexible spacecraft simulator. This facility is described as a typical 3-D antenna-like large space structure with twelve ribs (Fig. 3). A detail explanation of the experimental facility and the finite dimensional model can be found in [3]. The complete structure is typically modeled using 30 modes (i.e., 60 state variables). One of the finite-dimensional subsystem models for this facility which incorporates 3 flexible modes of the structure and a single rib, is considered here. This 3-mode subsystem is modeled by a 6^{th} order dynamic equation:

$$\dot{\underline{x}} = \underline{A}\,\underline{x} + \underline{B}\,u\,; \qquad y = \underline{C}\,\underline{x} \qquad (11)$$

where the three pairs of state variables $\{x_1, x_4\}$, $\{x_2, x_5\}$, and $\{x_3, x_6\}$ represent the modal displacement and velocity for the modal frequencies ω_1=0.61, ω_2=1.68, and ω_3=4.85 (both in radians/sec), respectively.

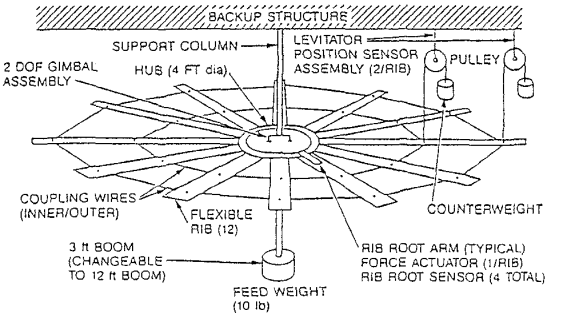

Fig: 3. The large flexible space structure

$$\underline{A} = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\ -14.93 & 0.00 & 0.00 & -0.07 & 0.00 & 0.00 \\ 0.00 & -111.98 & 0.00 & 0.00 & -0.21 & 0.00 \\ 0.00 & 0.00 & -931.54 & 0.00 & 0.00 & -0.61 \end{bmatrix}$$

$\underline{B} = [\,0, 0, 0, 0.009, 0.0279, -0.0702\,]^T$; and $\underline{C} = [\,1, 1, 1, 0, 0, 0\,]^T$.

The continuous time system of Eq. (11) is sampled every Δt=0.05 (sec.). Then the transfer function of this discrete system can be obtained:

$$\frac{Y(z)}{U(z)} = 10^{-3} \cdot \frac{-0.0254z^{-1}+0.1463z^{-2}-0.1152z^{-3}-0.1080z^{-4}+0.1431z^{-5}-0.0254z^{-6}}{1.0-3.7647z^{-1}+6.6446z^{-2}-7.6875z^{-3}+6.4972z^{-4}-3.6267z^{-5}+0.9561z^{-6}} \qquad (12)$$

A multi-layer feedforward neural network is employed to identify the system given by Eq. (12). The neural network used here is arranged into three layers, namely, an input, a hidden and an output layer. There are 6 nodes in the input layer, corresponding to a set of delayed outputs y(k-1), y(k-2), y(k-3), y(k-4), y(k-5), and the input u(k). Only one neuron is needed for the output layer because the system to be identified is a single output system. The hidden layer is chosen to have 20 nodes. Thus for the original network, the total number of weights is 140.

The neural network is trained by the Fletcher and Reeves conjugate gradient algorithm using a sinusoid control input signal u(k) = sin(2πk/100). Training is performed by minimizing the following performance index:

$$E(W) \stackrel{\Delta}{=} \frac{1}{N} \sum_{k=1}^{N} [\hat{y}(k) - y(k)]^2 \qquad (13)$$

where the total number of samples N=200. All the weights of the neural network are initialized at random with uniform distribution on the interval [-1, 1]. Minimization of Eq. (13) is repeated until the error function is reduced to an acceptable level. The identification results are shown in Fig. 4, where the plant and neural network outputs are represented by the dotted and dashed lines, respectively.

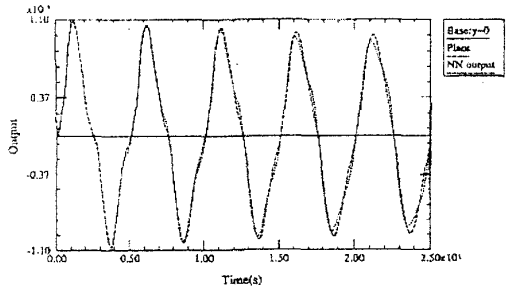

Fig. 4. Simulation results (before pruning)

The block diagram for pruning is shown in Fig. 5. Before pruning, the total "system error" (i.e., the difference between the plant output and the neural network output) is the identification error only. After pruning, it becomes a function of both the identification error and the pruning error. Reducing the network size usually accompanies an increasing in the total error. In order to assure the performance of the neural network with the reduced size, a suitable threshold must be set. In general, it is desirable that the root mean square error is less than 10% of the maximum magnitude of the plant output. Fig. 6 illustrates the simulated output when 94 weights are removed from the original neural network. Comparing with Fig. 4, the percentage of the root mean square error versus the maximum magnitude of the system output is increased from 5.38% to 7.89%; however, the network dimension is reduced dramatically by 67.14%. In other words, similar identification accuracy is achieved even though the neural network dimension is reduced to only 32.86% of its original size.

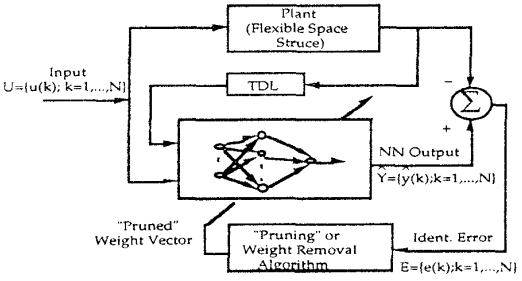

Fig. 5. Block diagram on the application of pruning algorithm

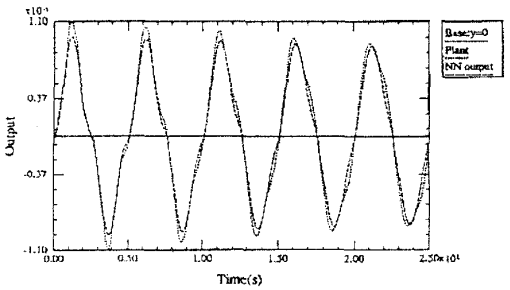

Fig. 6. Simulation results (after pruning)

IV. Conclusions

In this paper, a pruning algorithm with embedded gradient-conjugate training algorithm is presented and applied to the identification of a large flexible space structure. Computer simulation results show that approximately 67% of the original network weights can be removed without affecting the identification accuracy.

References

[1] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," *Advances in Neural Information Processing I*, D. S. Touretzky, Ed., Morgan Kaufmann, 1989

[2] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," IEEE Trans. Neural Networks, vol. 1, no. 2, June 1990

[3] H. C. Vivian, "Flexible Structure Control Laboratory Development and Technology Demonstration", Final Report for USAF-AFAL/NASA-OAST under Task Order 69006, Jet Propulsion Laboratory

[4] X.-H. Yu, "On the analysis of training algorithms and network dimension for neural network system identification models", Thesis, Temple University, 1992