International Journal of Neural Systems, Vol. 8, Nos. 5 & 6 (October/December, 1997) 509–515

A CASCADING STRUCTURE AND TRAINING METHOD FOR MULTILAYER NEURAL NETWORKS

YAN LI and A. B. RAD
Department of Electrical Engineering,
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Received 10 November 1996
Revised 16 May 1997
Accepted 6 November 1997

A new structure and training method for multilayer neural networks is presented. The proposed method is based on cascade training of subnetworks and optimizing weights layer by layer. The training procedure is completed in two steps. First, a subnetwork, m inputs and n outputs as the style of training samples, is trained using the training samples. Secondly the outputs of the subnetwork is taken as the inputs and the outputs of the training sample as the desired outputs, another subnetwork with n inputs and n outputs is trained. Finally the two trained subnetworks are connected and a trained multilayer neural networks is created. The numerical simulation results based on both linear least squares back-propagation (LSB) and traditional back-propagation (BP) algorithm have demonstrated the efficiency of the proposed method.

1. Introduction

Since the 1986 publication of Rumelhart & McClelland,[5] many improved learning algorithms based on back-propagation (BP) learning rule have been reported in literature.[1–4,8,9] Some researchers use heuristic rules to find optimal learning parameters.[3] Others refine the gradient descent method to accelerate convergence.[4,10] Further approaches have employed different nonlinear optimization methods like conjugate gradients, Newton's method or other techniques.[1,2,8,9]

All these improvements have achieved better performance and are sufficient for many applications. However, for applications in real-time which require high precision output, the known algorithms are still slow and inefficient.

By looking at the structure of neural networks, Biegler-Konig and Barmann[1] have presented a new algorithm based on linear least-square problem recently. It is outstanding in both convergence and accuracy compared with the algorithms based on steepest descent and other optimization methods. The major drawback of the algorithm is its limited ability to reduce the error beyond the point reached after about three or four iterations. Although it has been proved that a three-layer network can approximate any continuous function to any precision provided the neurons in the hidden layer can be arranged without any limitation, the computation burden will increase if the number of the neurons in the hidden layer is too large. Thus, networks of more than three layers may be necessary in some cases.

To further improve the performance of training procedure, a multilayer neural network with new structure and a training method for it based on the least square method[1] is proposed. For the sake of completeness, the original algorithm is also reviewed. The main difference between the presented method and the known ones is that it trains neural networks in steps, while all the known algorithms train the multilayer neural networks in one step. The superior performance of the proposed method has been shown in the simulation results based on both LSB and BP algorithms.

The rest of this paper is organised as follows: The proposed method is described in detail in Sec. 2. In

Sec. 3, we present simulations and compare the results of the proposed method and traditional one. Finally, the paper is concluded in Sec. 4.

2. The Cascading Training Method

Building a neural networks involves two distinct tasks: Determination of the structure and modification of the weights. The selection of topology is discussed first followed by the training algorithm.

2.1. *Topology of neural networks*

In earlier articles, such as Ref. 6, it is demonstrated that a three-layer feedforward networks is sufficient to approximate any continuous function to any precision. The networks for our purpose consists of two subnetworks, and each has three layers as shown in Fig. 1. The number of inputs and outputs of the first subnetwork are m and n respectively. It is the same style as the training samples. There are p neurons in the hidden layer. The second subnetwork is a n-inputs and n-outputs network with q neurons in the hidden layer. When they are cascaded, a five-layer feedforward neural networks is formed because the output layer of the first subnetwork is the input layer of the second subnetwork.

The rationale behind constructing the network in this way is based on the following fact: In a supervised learning, the optimization criterion is defined as the error between the actual and desired outputs of a network. Learning seeks to reduce the error by using examples of the form $\langle x, y\rangle$, where y is the desired output for a typical input pattern x. If the units are output neurons, the difference between the desired output and actual output can be achieved directly. However, if the units are hidden, the desired output is not obvious, and the more hidden the layers are, the more obscure the desired output. Thus, in training the networks with more than three layers, some special measures should be taken. Here, we let the neurons in the middle hidden layer equal to the neurons in the output layer. Therefore, the desired outputs can be taken into use in both the output layer and the middle hidden layer.

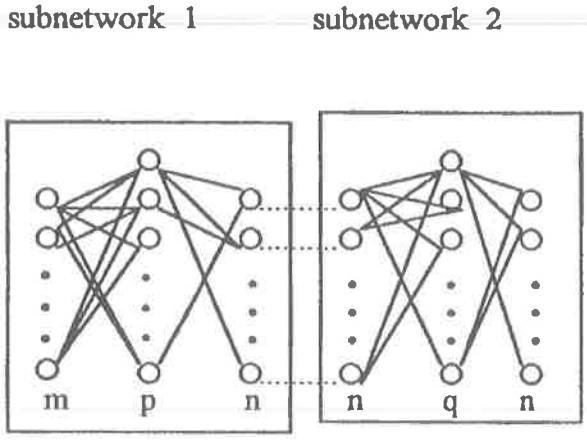

Fig. 1. The cascading architecture of networks.

2.2. *Cascading training*

Let us consider the above network. In general, the training will begin using certain training algorithms after the topology of the networks is determined. Here, we will complete the training procedure in two steps. The former subnetworks will be considered first.

2.2.1. *First subnetwork*

The structure of the first subnetwork is m-p-n and is fully connected between layers. To obtain a good convergence, the first neurons in each layer is a bias node with a constant output of 0.5.

The weights on the connections between input layer and hidden layer form a matrix W^1 of real numbers with elements w_{ij}^1 $(i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, p)$. Element w_{ij}^1 connects neuron i of input layer with neuron j of hidden layer. Row number 1 of W^1 contains the weights of the connections initiating at the bias node of input layer. Similarly, W^2 is a pxn weights matrix between hidden layer and the output layer.

The neurons used are of the standard type: They first build a scalar product of the incoming signals with their respective weights. Afterwards an activity function is applied, producing the output of the neuron which is sent to all neurons of the following layer respectively, or, to the outside world. The standard sigmoid activity function with values between 0 and 1 is used:

$$Q(x) = 1/(1 + e^{-x})\,. \tag{1}$$

For given N sets of input signals (each set contains m real numbers between 0 and 1), there will be N corresponding sets of output (each contains n real numbers between 0 and 1). We can summarize

all given input signals in a matrix A with N rows and m columns. Each row contains the signals of one input set. All elements of the first column are constant 0.5 (the outputs of the bias node of input layer). Similarly, summarize all output signals we want to teach the network in a matrix T with N rows and n columns (no bias column).

Propagating the given examples through the network, we get (by multiplying the input matrix with the weights matrix between input layer and hidden layer, applying the activation function to all matrix elements, and adding a bias constant column of 0.5).

$$A^1 = [0.5|\delta(A * W^1)] \tag{2}$$

$$A^2 = [0.5|\delta(A^1 * W^2)] \tag{3}$$

where A^1 and A^2 are the outputs of the hidden layer and output layer.

Teaching the network means trying to adjust the weights such that A^2 is equal to (or as close as possible to) $[0.5|T]$. By introducing

$$S = Q^{-1}(T)\,. \tag{4}$$

The task of learning is reformulated: Adjust the weights of the network such that S is as close as possible to $A^1 * W^2$. Dropping the indices 1 and 2 the problem of determining W optimally can be formulated as a linear least squares problem:

$$\text{minimize}\|A * W - S\|_2\,. \tag{5}$$

Note that, since Q is a nonlinear function, the minimum of Eq. (5) is not necessarily identical with the minimum of $\|A^2 - [0.5|T]\|_2$.

A linear least-squares problem can be solved either directly by employing Housseholderm transformations and a QR decomposition. An exact solution to the problem ($A * W = S$) can be achieved if the space spanned by the columns of A includes the space spanned by the columns of S. In the case of more than one solution, all algorithms mentioned above compute the minimal-norm solution, i.e. the solution of $A * W = S$, which minimizes $\|W\|_2$. Of course, if such a solution does exist, the neural network can learn all given examples exactly.

After having computed an optimal set W of weights, a required output matrix R of hidden layer which is as close as possible to A would be determined and fulfils

$$\|R * W - S\|_2 = \text{minimal}\,. \tag{6}$$

Again, this is for given matrix W and S, a linear least-square problem. Equivalently, one can find the minimal norm solution of ΔR in

$$\|\Delta R * W - (S - A * W)\|_2 = \text{minimal} \tag{7}$$

and determine R as $R = A^+\Delta R$.

Since the output of the bias node is constant 0.5, we require the first column of ΔR to vanish. Hence, we have to reformulate Eq. (7) to take care of this constraint. Let T and ΔT be the matrixes containing all columns of R and ΔR except the first and V be the matrix containing all rows of W except the first. We then have to solve the linear least squares problem.

$$\|\Delta T * V - (S - A * W)\|_2 = \text{minimal} \tag{8}$$

$$R = [0.5|T] = A + [0.0|\Delta T]\,. \tag{9}$$

Now, we try to adjust the weights between the input layer and the hidden layer to reach R as output in the hidden layer. But since the activation function (1) can only produce values between 0 and 1 this is only feasible if all elements of R are in this range. In order to transform R to a matrix with elements between 0 and 1, we construct a matrix C with p rows and columns such that

$$R_{\text{trans}} = R * C^{-1} \quad \text{and} \quad W_{\text{trans}} = C * W\,. \tag{10}$$

The minimal condition (8) is then still valid for the new matrices. Using the following notation:

$$\alpha_k = \frac{\sum_{j=1,\ldots,N} \gamma_{j,k}}{N}\,, \quad k = 1, 2, \ldots, p \tag{11}$$

$$\beta_k = \max_{j=1,\ldots,N} \gamma_{j,k}\,, \quad k = 1, \ldots, p \tag{12}$$

$$\gamma_k = \min_{j=1,\ldots,N} \gamma_{j,k}\,, \quad k = 1, \ldots, p \tag{13}$$

we can define the elements of C:

$$c_{0,0} = 1\,,$$

$$c_{0,k} = 2(\alpha_k - \max(0.5, \beta_k - \alpha_k, \alpha_k - \gamma_k))\,, \quad k = 1, \ldots, p \tag{14}$$

$$c_{k,k} = 2\max(0.5, \beta_k - \alpha_k, \alpha_k - \gamma_k)\,, \quad k = 1, \ldots, p. \tag{15}$$

All other elements of C are zeros. The inverse matrix of C has the same structure as C with elements

$$c_{0,k}^{(-1)} = -c_{0,k}/c_{k,k}\,, \quad k = 1, \ldots, p \tag{16}$$

$$c_{k,k}^{(-1)} = 1/c_{k,k}\,, \quad k = 0, \ldots, p\,. \tag{17}$$

With this choice of C, all elements of R_{trans} are between 0 and 1, and the average output of each node is 0.5. We can now use the same formalism as before to teach the hidden layers of the networks to produce

$$[0.5|T] = R_{\text{trans}} \tag{18}$$

as output layer.

Hence, we get a back propagation algorithm which, at the beginning uses the given training output, then computes for each layer a required output and adjusts the weights of the layers below accordingly. This process can be iterated until the desired accuracy is reached.

2.2.2. *Second subnetwork*

For the second subnetwork, the training procedure is same as above in general. However, in this case, the input of the training samples is the output of the first subnetwork and the desired output is the same as before. So the second has the structure of n-q-n.

After the first subnetwork and the second network are trained, they are cascaded. Thus, a trained five-layer neural networks with the structure m-p-n-q-n is formed.

2.2.3. *Complete network*

The whole training procedure can be stated as follows:

- Initialization: Choose initial values for the first subnetworks
- Optimization of the output layer weights:
  - — Get the weighted sums of the output neurons by inverse activation function (4)
  - — Compute the optimal weights of output layer according to (5)
  - — Determine the required output of the hidden layer using (6) and (7)
  - — Normalize desired output of the hidden layer through (11, ..., 17)
- Optimization of the hidden layer weights:
  - — Determine the weighted sum of the hidden neurons according to the desired output of the hidden layer computed from above step using (4)
  - — Compute the optimal weights of the hidden layer by (5)
- Repeat above steps until certain conditions are satisfied.
- Do the same thing to the second subnetworks
- Cascade the two trained subnetworks

A similar approach based on BP can be derived too.

3. Numerical Results

In this section, we present numerical results based on the above mentioned LSB and BP algorithms aimed as demonstrating the training method. The efficiency of a training procedure depends mainly on the training error and convergence speed. The performance of the presented training method is compared with those of traditional multilayer neural networks architectures.

Example 1

Consider a nonlinear mapping of eight input quantities x_i into three output quantities y_i, defined by

$$\begin{aligned} y_1 &= (x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8)/4 \\ y_2 &= (x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8)/8 \\ y_3 &= (1 - y_1)^2\,. \end{aligned} \tag{19}$$

All three functions are defined for values between 0 and 1 and produce values in this range. For the training set, N sets of input signals x_i were generated with a random number generator, the corresponding y_i were computed using Eq. (19). The error reported is the average absolute error on this learning set example and output node:

$$\Delta = \left(\sum_{i=1,\ldots,N} \sum_{j=1,\ldots,n_L} |A_{i,j}^L - T_{i,j}^L| \right) \Big/ (Nn_L)\,. \tag{20}$$

The starting values for the weights of the networks have also been determined between 0 and 1 using a random number generator.

The following table contains the learning errors using the proposed method, LSB and traditional BP algorithms for nine networks with the structure of 8-8-3-8-3, 8-8-3-12-3, 8-8-3-24-3, 8-12-3-8-3, 8-12-3-12-3, 8-12-3-24-3, 8-24-3-8-3, 8-24-3-12-3 and 8-24-3-24-3 and three networks with a single hidden layer containing 8, 12 and 24 nodes, respectively. All the errors in the table are the training errors after 10 training iterations. Here, the number of training sets is 50. The BP algorithm is the standard BP algorithm of [Rumelhart & McClelland, 1986], with learning rate, 0.5 and momentum factor, 0.9. The values of the learning rate and momentum factor are selected dependent on the best convergence speed of the BP algorithm. Changing the values of the learning rate and momentum factor would affect the convergence speed, but not very significantly in the proposed cases. One iteration (or epoch) indicates that the whole training process that all sets of input signals X_i are sent to a certain network, a set of corresponding output of the network are obtained and all the weights are adjusted one time. In fact, the LSB method needs about twice the computer time of backpropagation per step, independent of the network's structure.[1] But for BP algorithm to reach the accuracy as that of LSB, it will require thousands iterations or more.[1,2]

Comparing the results in Table 1, the performance of the five-layer networks using the LSB method is worse than that of three-layer one, the former training errors are 0.0195, 0.0196 and 0.0166, the latter training errors are 0.0127, 0.0109 and 0.0096. By using the proposed training method, the training error are 0.0084, 0.0082 and 0.0037, best in all the cases. In the same case as the former, 55–80% of the training error is improved. Obviously, the convergence speed of the LSB method is much higher than that of BP algorithm.

The following curves in Fig. 2 has shown the training processes using 1000 training sets. The solid line, dashed line, dot-dot line and dashed-dot line are the proposed method, five-layer LSB method, three-layer BP and five-layer BP algorithm, respectively.

MATLAB is used to implement these simulations in an IBM586. Every case has been repeated for many times. The final error is different every time.

Table 1. Training errors.

	The Proposed Method	LSB Method	BP Algorithm
8-8-3-8-3	0.0084	0.0195	0.0804
8-8-3-12-3	0.0082	0.0196	0.0841
8-8-3-24-3	0.0037	0.0166	0.4084
8-12-3-8-3	0.0094	0.0186	0.0812
8-12-3-12-3	0.0083	0.0178	0.0835
8-12-3-24-3	0.0031	0.0163	0.2478
8-24-3-8-3	0.0083	0.0104	0.0816
8-24-3-12-3	0.0045	0.0096	0.0838
8-24-3-24-3	0.0032	0.0109	0.3270
8-8-3	—	0.0127	0.0854
8-12-3	—	0.0109	0.0885
8-24-3	—	0.0096	0.4525

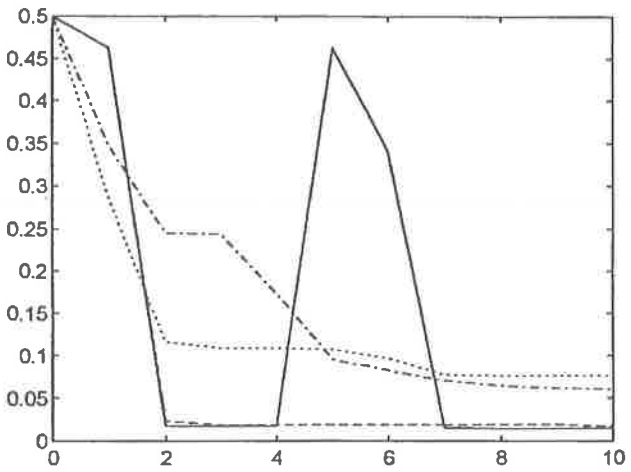

Fig. 2. Errors of training process using the proposed method, LSB method and BP algorithm. —— the proposed method; ---- five-layer LSB; ..... three-layer BP; —·—· five-layer BP.

A set of typical data from the simulation results is presented to show the efficiency of this method.

From the simulation results, it is observed that the performance of the proposed method has a second peak as shown in Fig. 2 (the solid line). The reason is that the first subnetwork is in training at the first four epochs. The error is the difference between the desired output and the output of the first subnetwork. Then, the output of the first subnetwork is put on as the input of the second subnetwork. So, the second subnetwork is in training. All initial weights of two subnetworks are set randomly

between 0 and 1. The error is high at the first several training iterations. However, it converges very fast and becomes flat. A smaller error can be achieved by cascading training two subnetworks. For traditional three-layer and five-layer architectures for BP and LSB, no substantial progress could be achieved by continuing the iterations after the first three or five epochs. However, for the proposed method, not only it can further reduce the error but also it converge within 10 iterations. For classical BP (with learning rate and momentum factor) method to reach this accuracy, it will take 2459 iterations or more for the same training sets, in this case.

Figure 3 shows the simulation results of cascading structure and traditional one for multilayer neural networks using BP algorithm. To reach the same level of accuracy and convergence, more epochs are needed as compared with LSB. It is observed that the performance of the proposed method (8-12-3-3-12-3) is better than that of five layer neural networks (8-12-3-12-3). The former training error is 0.0094, the later training error is 0.0159, 40.88% of the training error is improved.

Example 2

There is another nonlinear mapping of five input quantities x_i into four output y_i, defined by

$$\begin{aligned} y_1 &= (x_1 x_2 (x_1 + 2.5))/(1 + x_1^2 + x_2^2) \\ y_2 &= (x_1 x_2 x_3 x_5 (x_3 + 1) + x_4)/(1 + x_2^2 + x_3^2) \\ y_3 &= y_1 + y_2 \\ y_4 &= y_1 + (1/2) y_2 \,. \end{aligned} \tag{21}$$

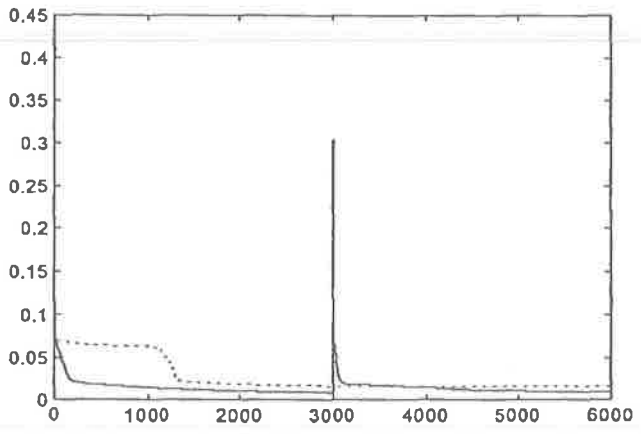

Fig. 3. Errors of training processes using BP. —— the proposed method; ..... five-layer BP.

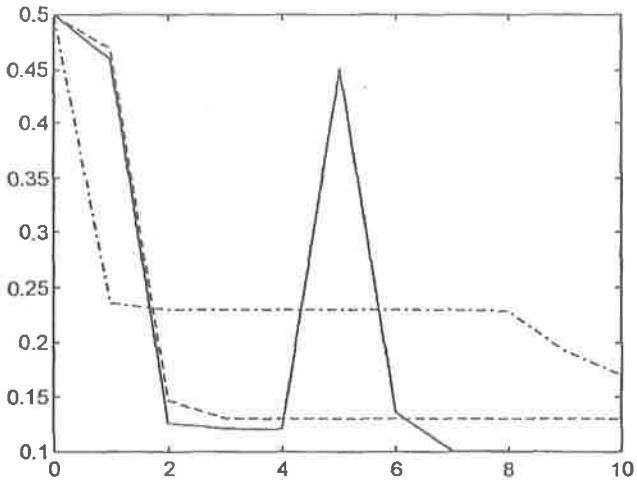

Fig. 4. Training processes of Example 2. —— the proposed method; ----- five-layer LSB —·— five-layer BP.

Figure 4 shows the training process using 2000 training sets of above nonlinear mapping function. The training sets and initial weights are generated as in Example 1. The error reported is also defined in Eq. (20). Figure 4 demonstrates the efficiency of the proposed method. The second subnetwork gives more significant improvement in this problem. A much smaller error can be achieved by cascading training the second subnetwork after the first subnetwork has been trained. The larger the training set, The better the performance.

4. Conclusion

A new training method for multilayer neural networks based on the cascading approach using the linear least square technique was presented. The simulation results have shown that the training error has been dramatically reduced and convergence has been improved (within 10 iterations). The proposed method is based on cascading training of the subnetworks and optimization weights layer by layer. The training procedure is completed in two steps. The first subnetwork is in training at the first several epochs. Then, the output of the first subnetwork is put on as the input of the second subnetwork. A smaller error can be achieved by cascading training the second subnetwork. Not only can it further reduce the error, but it can also converge within 10 iterations. For traditional three-layer and five-layer architectures for either LSB or BP, no substantial

progress could be achieved quickly by continuing the iterations after the first several epochs. The numerical simulation results demonstrate the promising potential of this algorithm. Work is in progress to apply this training method to some real control problem such as the work reported in Ref. 11.

References

1. F. Biegler-Konig and F. Barmann 1993, "A learning algorithm for multilayer neural networks based on linear least square problems," *Neural Networks* **6**, 127–131.
2. S. Ergezinger and E. Thomsen 1995, "An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer," *IEEE Trans. Neural Networks* **6**(1), 31–42.
3. B. K. Humpert 1994, "Improving back propagation with a new error function," *Neural Networks* **7**, 1191–1192.
4. R. Battiti 1989, "Accelerating back propagation learning, two optimization methods," *Complex System* **3**, 331–342.
5. D. L. Rumelhart and J. L. McClelland (eds.) 1986, "Parallel distributed processing: Exploration in the microstructure of cognition," (MIT Press, Cambridge, MA).
6. Z. Wang, M. Tham and A. J. Morris 1992, "Multilayer feedforward neural networks: A canonical form approximation of nonlinearity," *Int. Journal of Control* **56**, 655–672.
7. T. P. Vogel, J. K. Mangis, A. K. Rigler, W. T. Zink and D. L. Alker 1988, "Accelerating the convergence of the back propagation method," *Cybernetics* **59**, 259–263.
8. W. Kinnebrock 1994, "Accelerating the standard back propagation method using a genetic approach," *Neurocomputing* **6**, 583–588.
9. B. Zhang, L. Zhang and F. Wu 1995, "Programmimg based learning algorithms for neural networks with self-feedback connection," *IEEE Trans. Neural Networks* **6**(3), 771–775.
10. R. A. Jacobs 1988, "Increased rates of convergence through learning rate adaption," *Neural Networks* **1**, 295–307.
11. Y. Li, A. B. Rad, Y. K. Wong and H. S. Chan 1996, "Model based control using artifical neural networks," *Proc. 1996 IEEE Int. Symp. Intelligent Control*, Dearborn, MI, September 15–18, 1996, pp. 283–288.