STRUCTURED TRAINABLE NETWORKS FOR MATRIX ALGEBRA

Lixin Wang and J.M.Mendel
Signal and Image Processing Institute
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA, 90089-0781

Abstract

In this paper, a novel approach to a large variety of matrix algebra problems is proposed. The basic idea of this approach is: first, represent a given problem by a structured network architecture; then, train the structured network to match some desired patterns; finally, obtain the solution to the problem from the weights of the resulting structured network. The basic unit used to construct the network is a simple linear multi-input single-output weighted-summer. The training algorithms for the problems are either standard error back-propagation or new modified error back-propagation. Three detailed structured networks and the corresponding training algorithms are presented in this paper for matrix *LU* decomposition, linear equations solving, and singular value decomposition, respectively. Extensions to other matrix algebra problems are straightforward. The advantages of these new approaches are: (1) parallel architectures and algorithms, suitable for VLSI realizations; (2) robust computations, no divisions are involved in all the calculations, so that they are free of the divide-by-zero problem; and, (3) very general, suitable for most matrix computation and matrix equation-solving problems.

1 INTRODUCTION

Matrix computations and matrix equation-solving are very important and basic problems in science and engineering. The existing technologies for these problems can be classified into: (1) traditional algorithm-based methods [1-3], epitomized by the very successful commercial software package MATLAB; and, (2) parallel processing approaches, based on VLSI-inspired systolic and wavefront array processors [4-6]. Although these methods are successful in solving a lot of practical problems, they do have some limitations and disadvantages.

The main disadvantages of the algorithm-based approaches are: (1) they are not parallel algorithms and cannot be realized directly by VLSI hardwares, so that the speed of processing is quite limited; and, (2) divisions are usually involved in the calculations, so that correct solutions cannot be obtained for ill-conditioned matrices.

For the parallel processing approaches, although the first disadvantage of the algorithm-based approaches is overcome, the second disadvantage still exists. This is beacuse the key procedure of the parallel processing approaches is to modify and decompose the traditional algorithms to make them suitable for VLSI architectures and realizations [4-6]. Consequently, the calculations performed by these two approaches are the same when viewed from a higher conceptual level, i.e., the difference is only in the specific ways to realize the calculations. For example, the parallel approach to solving linear equations described in [6] is still based on Gaussian elimination; hence, the parallel processing approaches still do not work for ill-conditioned matrices.

The parallel processing approaches have another weakpoint: they do not give a general framework for matrix algebra problems, i.e., they are quite problem dependent. This is because the first step of the parallel approaches is to obtain a set of parallel equations for the specific problem, and then to use parallel hardware to perform the calculations of these equations; hence, from a general conceptual point of view, the parallel processing approach is not an unified method for matrix algebra problems.

The new approaches in this paper overcome all the above disadvantages of the existing methods. The basic idea of the new approaches is: (1) represent a given problem by a structured network architecture; this is the "construction phase"; (2) train the structured network to match some desired patterns; this is the "training phase"; and, (3) obtain the solution to the problem from the weights of the resulting structured network; this is the "application phase". These three phases are shown in Figure 1.

The basic unit used to construct the structured network is a linear multi-input single-output weighted-summer, i.e. a unit whose output equals the weighted sum of it inputs. The weights are adjusted by a training procedure during the "training phase". The training algorithms are either standard error back-propagation [7] or new modified error back-propagation. From a general conceptual point of view, our approach is to view matrix algebra problems as special pattern recognition problems, i.e., we solve these problems by matching desired patterns by specifically-constructed networks.

Due to the space limitations, only three structured networks and their corresponding training algorithms are presented in this paper for: matrix LU decomposition, linear equations solving and matrix singular value decomposition. Extensions to other problems, like LDM^T decomposition, QR factorization, similarity decomposition, Schur decomposition, and Lyapunov equation solving, etc., will be reported in a future publication.

2 LU DECOMPOSITION

Given a matrix $A \in R^{m\times n}$, the task is to obtain a lower-triangular matrix (with unity diagonal elements) $L \in R^{m\times m}$ and an upper-triangular matrix $U \in R^{m\times n}$ such that $LU = A$. Consider the two-layer structured network of Figure 2. We constrain the connections between the inputs and the lower-layer units so that the output of the lower-layer equal Ux, where x is a given input vector and U is an upper-triangular matrix. Similarly, we constrain the connections between the lower-layer and the upper-layer units so that the network output will be LUx, where L is a lower-triangular matrix with all diagonal elements equal to unity. Since the desired overall transformation $LU = A$ is known, the desired output Ax is also known when network input x is specified. (x, Ax) constitutes the training patterns.

There are two basic problems to implementing this structured network: (1) how to choose the input patterns (the x's) so that if their associated patterns are matched, the weight matrices give just what we want; and, (2) how to train this structured network.

For the first problem we have the following:

THEOREM 1. For a given matrix $A \in R^{m\times n}$ and the structured network of Figure 2, we choose the following n input patterns: $x^{(1)} = (1, 0, ..., 0)^T, x^{(2)} = (0, 1, 0, ..., 0)^T, ..., x^{(n)} = (0, ..., 0, 1)^T$, so we have n desired output patterns: $Ax^{(1)}, Ax^{(2)}, ..., Ax^{(n)}$. If the structured network matches all these n patterns, then the final weights of the lower-layer units give the desired U matrix, and the final weights of the upper-layer units give the desired L matrix, i.e. $LU = A$.

Due to space limitations, we omit the proofs of all Theorems.

The input vectors given in the above Theorem are the most straightforward choice. In fact, it is easy to show that any n linearly independent $n \times 1$ vectors are possible choices for the desired input patterns.

For the second problem, we use the well-known error back-propagation algorithm [7]. The main reason we use this algorithm is that it is parallel and local, so it is suitable for hardware realization. Specifically, to update the upper-layer unit weights l_{ij}, we use

$$l_{ij}(t+1) = l_{ij}(t) + \alpha(d_i - y_i)z_j \tag{1}$$

where y_i is the actual output of the i^{th} unit of the upper-layer, d_i is the corresponding desired output (which equals the i^{th} component of Ax), z_j is the output of the j^{th} unit of the lower-layer, and t denotes the steps of updatings ($t = 0, 1, 2, ...$). To update the lower-layer unit weights u_{ij}, we use

$$u_{ij}(t+1) = u_{ij}(t) + \alpha[\sum_{k=1}^{m}(d_k - y_k)l_{ki}(t+1)]x_j. \tag{2}$$

Since we constrain the connections, some weights are constants: $l_{ii} \equiv 1, l_{ij} \equiv 0$ for $i = 1, 2, ..., n, j = i+1, i+2, ..., n$; $u_{ij} \equiv 0$ for $i = 2, 3, ..., n, j = 1, 2, ..., i-1$. These weights are not updated.

From Eqs.(1) and (2) we note that there are no divisions. It is known from [1] that the standard Gaussian elimination algorithm for LU decomposition will fail if the matrix A has a singular leading principle submatrix. The main problem of Gaussian elimination, as well as many other classical methods [1], is that divisions are involved in the calculations. Since our training algorithm does not involve any divisions, our method will be robust to different matrices, i.e., our method will give correct LU decompositions for matrices which are ill-conditioned for Gaussian elimination. Simulation results have confirmed this, as will be shown below.

In order to show the convergence behavior of the training algorithm and the accuracy of our approaches to matrix LU decomposition, we give the simulation results for the following example. The matrix A in this example has a singular leading principle submatrix, so that it is ill-conditioned for Gaussian elimination.

Example 1: The matrix to be LU decomposed is

$$A = \begin{bmatrix} 0 & 0.8 & 0.7 & 0.5 \\ 0 & 1.5 & 0.3 & 0.1 \\ 0 & 0.5 & 1.7 & 0.9 \\ 0 & 0.5 & 0.6 & 1.4 \end{bmatrix} \tag{3}$$

The initial L matrix was chosen to be an identity matrix, and the initial U matrix was chosen to be a zero matrix. According to Theorem 1, there are four patterns to be learned. We trained the structured network for these four patterns, one by one, and then repeated the training, where the initial weights for a new pattern equal the final converged weights for the previous pattern. The pattern sequence to be trained is: $Ax^{(1)}, Ax^{(2)}, ..., Ax^{(4)}, Ax^{(1)}, ...$. If the sum of absolute errors for a pattern is less than a threshold (ϵ_1), the training is changed to the next pattern; and, if the sum of absolute errors for all the four patterns is less than another threshold (ϵ_2), all training stops. Figure 3 shows the convergence behavior of the training procedure for $\epsilon_1 = 0.02$ and $\epsilon_2 = 0.08$, where the x-axis represents the steps of the training, and the y-axis represents the sum of absolute errors for the pattern which is being learned. Specifically, the y-axis denotes $\sum_{i=1}^{4} |d_i - y_i|$, where d_i is the i^{th} element of the current desired output pattern, and y_i is the i^{th} element of the actual network output vector.

The final L, U and LU matrices are

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.037 & 1 & 0 & 0 \\ 0.371 & 0.136 & 1 & 0 \\ 0.265 & 0.185 & 0.262 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 0 & 0.796 & 0.698 & 0.506 \\ 0 & 1.529 & 0.321 & 0.123 \\ 0 & 0 & 1.389 & 0.696 \\ 0 & 0 & 0 & 1.055 \end{bmatrix} \quad LU = \begin{bmatrix} 0 & 0.796 & 0.698 & 0.506 \\ 0 & 1.500 & 0.296 & 0.104 \\ 0 & 0.503 & 1.692 & 0.901 \\ 0 & 0.494 & 0.609 & 1.395 \end{bmatrix} \tag{4}$$

3 LINEAR EQUATIONS SOLVING

Consider the problem of solving the linear equations

$$Ax = b \tag{5}$$

where $A \in R^{m \times n}$. If we can find a matrix $B \in R^{n \times m}$ such that

$$BA = I_{n \times n} \tag{6}$$

then we have

$$x = Bb. \tag{7}$$

In fact, the x obtained from Eq.(7) is the least-square solution of the linear equations Eq.(5), and the B which satisfies Eq.(6) is the pseudo-inverse of A.

Consider a two-layer structured network of Fig. 4, where the lower-layer performs the transformation A and the upper-layer performs the transformation B. According to Eq.(6), the desired overall transformation is $I_{n \times n}$. Since A is given, the lower-layer weights are set to A initially and do not change in the training procedure. We will use standard error back-propagation algorithm to update the upper-layer weights B.

After the desired B is obtained, we let the inputs to the upper-layer equal to the vector b; then, according to Eq.(7), the output of the network gives the solution $x = Bb$.

For the choice of desired patterns, we have the following:

THEOREM 2: For $A \in R^{m \times n}$ and the structured network of Fig.4, we choose the following n pairs of patterns: (e_i, e_i) where $n = 1, 2, ..., n$ and e_i is an $n \times 1$ vector with i^{th} component equal to one and all other components equal to zero. If the structured network matches all these n patterns, the upper-layer weights B will satisfy $BA = I_{n \times n}$.

To train this structured network, we use the following equation to update the weights of the upper-layer units,

$$b_{ij}(t+1) = b_{ij}(t) + \alpha[x_i - z_i(t)]y_j(t) \tag{8}$$

where x is the input vector to the network, z is the actual output vector, and y is the output vector of the lower-layer. The lower-layer weights are set equal to A initially and do not change.

Example 2: The linear equations to be solved are (5) with

$$A = \begin{bmatrix} 1.2 & 0.8 & 0.7 & 0.5 \\ 0.4 & 1.5 & 0.3 & 0.1 \\ 0.1 & 0.5 & 1.7 & 0.9 \\ 0.1 & 0.5 & 0.6 & 1.4 \end{bmatrix} \tag{9}$$

$$b = [1, 2, 3, 4]^T. \tag{10}$$

The initial B was chosen to be an identity matrix. The convergence behavior is shown in Fig.5, where the meanings of the x-axis and y-axis are the same as those of Fig.3, and the stopping rule is also the same as that for Example 1. The final B gave

$$x = Bb = [-1.218, 1.464, 0.179, 2.347]^T \tag{11}$$

with

$$Ax = [1.008, 2.000, 3.026, 4.003]^T. \tag{12}$$

Comparing Ax in Eq.(12) with b in Eq.(10), we see that we obtained a reasonablely accurate solution. Accuracy can be improved by lowering the thresholds ϵ_1 and ϵ_2.

4 SINGULAR VALUE DECOMPOSITION

Given a matrix $A \in R^{m \times n}$, the task here is to decompose it into $A = URV$, where $U \in R^{m \times m}$ and $V \in R^{n \times n}$ are orthogonal matrices, i.e., $U^T U = I, VV^T = I$, and $R = diag[r_{11}, r_{22}, ..., r_{pp}] \in R^{m \times n}$ with $p = min[m, n]$ and $r_{ii} \geq 0$ $(i = 1, 2, ..., p)$. If we use the idea similar to that used in LU decomposition for singular value decomposition, i.e., construct a three-layer structured network according to $A = URV$, with the lowest-layer performing transformation V, the middle-layer performing transformation R, and the top-layer performing transformation U, then we must solve two additional problems: (1) how to constrain U and V to be orthogonal transformations; and, (2) how to guarantee that the singular values $r_{ii} \geq 0$ $(i = 1, 2, ..., p)$.

To constrain U and V to be orthogonal transformations, we add two additional layers to the basic three-layer URV network: one, which performs transformation V^T, is added to the input layer; the other, which performs transformation U^T , is added to the output layer. The final structured network for singular value decomposition is shown in Fig.6 for the $m = n = 3$ case. We view this structured network as a combination of three overlapped and independently trained sub-networks, as shown in Fig.6 by SN1, SN2, and SN3, respectively. Here by "overlapped" we mean that some layers of different sub-networks perform exactly the same transformation (U for both SN1 and SN3, and V for both SN1 and SN2), and by "independently trained" we mean that the training of the sub-networks run, synchronously or asynchronously, in such a way that any sub-network just updates its own weights as if no other sub-network existed. As a result, when a sub-network updates its overlapped weights, the previous weights it uses may or may not be the weights obtained at its own previous updating step.

Observing SN2 and SN3, we see that we have a new problem: how to guarantee that the upper-layer of SN2 (SN3) performs transformation V (U^T) and at the same time the lower-layer of SN2 (SN3) performs transformation V^T (U). If we use the standard error back-propagation, as in LU decomposition, we cannot meet these constraints. We shall propose a new modified error back-propagation algorithm to train this structured network.

In summary, there are now three problems to implementing this overlapped structured network: (1) how to choose the desired patterns for SN1, SN2 and SN3, such that if all the corresponding desired patterns are matched by the corresponding sub-network, the weight matrices of SN1 give the required V, R, and U; (2) how to train SN1 with constraint $r_{ii} \geq 0$; and, (3) how to train SN2 and SN3 such that the upper-layer of SN2 (SN3) performs transformation V (U^T) and at the same time the lower-layer of SN2 (SN3) performs transformation V^T (U).

For the first problem, we have the following:

THEOREM 3: For a given matrix $A \in R^{m \times n}$ and the structured network of Fig.6, we choose the following n patterns for SN1: $(e_1, Ae_1), (e_2, Ae_2), ..., (e_n, Ae_n)$; the following n patterns for SN2: $(e_1, e_1), (e_2, e_2)$, ..., (e_n, e_n); and, the following m patterns for SN3: $(e_1, e_1), (e_2, e_2), ..., (e_m, e_m)$. If SN1, SN2 and SN3 match all the corresponding patterns respectively, then the final weights of the lowest-layer, middle-layer, and top-layer of SN1 give the desired U, R, and V matrices respectively, i.e., $URV = A$ with $U^TU = I$ and $VV^T = I$. (The positive requirement for the elements of R is guaranteed by the training algorithm which will be given next.)

For the second problem, we use the following modified error back-propagation algorithm to train SN1. To update the top-layer weights of SN1, we use

$$u_{ij}(t+1) = u_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j \tag{13}$$

where $\hat{d}_i$ is the actual output of the i^{th} unit of the top-layer of SN1, d_i is the corresponding desired output (which equals the i^{th} component of Ax, where x is the input vector), and y_j is the output of the j^{th} unit of the middle-layer. To update the middle-layer weights of SN1, we use

$$r_{ii}(t+1) = r_{ii}(t) + \frac{\alpha}{2}[|\sum_{k=1}^{m}(d_k - \hat{d}_k)u_{ki}(t+1)z_i| + \sum_{k=1}^{m}(d_k - \hat{d}_k)u_{ki}(t+1)z_i] \tag{14}$$

where z_i is the output of the i^{th} unit of the lowest-layer of SN1. We see from Eq.(14) that if the correction term $\sum_{k=1}^{m}(d_k - \hat{d}_k)u_{ki}(t+1)z_i > 0$, Eq.(14) reduces to the standard error back-propagation; if $\sum_{k=1}^{m}(d_k - \hat{d}_k)u_{ki}(t+1)z_i \leq 0$, r_{ii} remains the same. So if we choose the initial $r_{ii}(0)'s$ equal to zero, Eq.(14) guarantees that $r_{ii}(t) \geq 0$ for any t. Finally, to update the lowest-layer weights of SN1, we use

$$v_{ij}(t+1) = v_{ij}(t) + \alpha[\sum_{k=1}^{m}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)]x_j. \tag{15}$$

For the third problem, we propose the following modified error back-propagation algorithm to train SN2 and SN3. To update the upper-layer weights of SN2, v_{ij}, we use

$$v_{ij}(t+1) = v'_{ji}(t) + \alpha[d_i - \hat{d}_i]y_j \tag{16}$$

where v'_{ji} is the $(j,i)^{th}$ weight (i^{th} weight of the j^{th} unit) of the lower-layer of SN2, $\hat{d}_i$ is the actual output of the i^{th} unit of the upper-layer, d_i is the corresponding desired output, and y_j is the output of the j^{th} unit of the lower-layer. To update the lower-layer weights of SN2, v'_{ji}, we use

$$v'_{ji}(t+1) = v_{ij}(t+1) + \alpha[\sum_{k=1}^{n}(d_k - \hat{d}_k)v_{kj}(t+1)]x_i. \tag{17}$$

From Eqs.(16) and (17) we see that the training for SN2 is occurring in the following way: first, update the upper-layer weight v_{ij} by correcting the lower-layer weight v'_{ji}; then, update the lower-layer weight v'_{ji} by correcting the just updated upper-layer weight v_{ij}. So if we choose the initial $v_{ij}(0) = v'_{ji}(0)$, then after convergence we have $v_{ij} = v'_{ji}$, the result we desire.

Similarly, we update the upper-layer weights of SN3, u'_{ji}, by

$$u'_{ji}(t+1) = u_{ij}(t) + \alpha[d_j - \hat{d}_j]y_i \tag{18}$$

and update the lower-layer weights of SN3, u_{ij}, by

$$u_{ij}(t+1) = u'_{ji}(t+1) + \alpha[\sum_{k=1}^{m}(d_k - \hat{d}_k)u'_{ki}(t+1)]x_j. \tag{19}$$

The meanings of the variables in Eqs.(18) and (19) and the explanation of these equations are similar to those for Eqs.(16) and (17).

Example 3: The matrix to be singular value decomposed is the following arbitarily chosen 3-by-3 matrix

$$A = \begin{bmatrix} 1 & 0.5 & 0.3 \\ 0.5 & 2 & 0.2 \\ 1.3 & 0.6 & 2.5 \end{bmatrix} \tag{20}$$

The initial U and V were chosen to be identity matrices, and the initial R was chosen to be a zero matrix. The convergence behaviors of SN1, SN2 and SN3 are shown in Figs.7, 8 and 9, respectively. The meanings of these figures are similar to that for Example 1 (Fig.3). The final U, R and V gave

$$URV = \begin{bmatrix} 0.999 & 0.495 & 0.300 \\ 0.496 & 1.990 & 0.198 \\ 1.297 & 0.587 & 2.500 \end{bmatrix} \tag{21}$$

$$U^TU = \begin{bmatrix} 0.987 & -0.002 & -0.001 \\ -0.002 & 0.995 & 0.004 \\ -0.001 & 0.004 & 1.009 \end{bmatrix} \tag{22}$$

$$VV^T = \begin{bmatrix} 0.992 & -0.003 & -0.001 \\ -0.003 & 0.994 & 0.004 \\ -0.001 & 0.004 & 1.014 \end{bmatrix} \tag{23}$$

The singular values were

$$R = diag[0.6459, 1.8156, 3.1542], \tag{24}$$

while the singualr values using MATLAB were

$$R_{MATLAB} = diag[0.6385, 1.8068, 3.2028]. \tag{25}$$

5 CONCLUSIONS

In this paper, three structured networks and their corresponding training algorithms were proposed for matrix LU decomposition, linear equations solving, and matrix singular value decomposition, respectively. The basic idea of these approaches can be easily extended to a lot of other matrix algebra problems. This basic idea is: first, represent a given problem by a structured network; then, train this structured network to match some desired patterns; finally, obtain the solution to the problem from the weights of the resulting structured network. Simulation results show that our new approaches work quite well.

As compared with existing methods for matrix algebra, our new approaches have some very important advantages. First, our new approaches use parallel architectures (i.e. structured networks) to represent the problems, and use parallel and local algorithms to train the structured networks, so that they are suitable for VLSI realizations. Second, no divisions are involved in any of the calculations, so that our new approaches are free of the divide-by-zero problem and can give good solutions to ill-conditioned matrices. Third, our new approaches are so simple and straightforward that no complicated matrix algebra knowledge is required in order to understand and to use these new methods. Fourth, the basic idea of these new approaches is quite general and can be used for most matrix algebra problems, i.e., our new approaches present a general framework for solving a large variety of matrix algebra problems.

6 ACKNOWLEDGEMENTS

The work reported here was performed at the University of Southern California and was supported by USC Faculty Research and Innovation Grant 22-1503-9336, 1989-1990.

7 REFERENCES

1 G.H.Golub and C.F.Van Loan, *Matrix Computations*, Johns Hopkins, 1983.

2 J.Dongarra, F.R.Bunch, C.B.Moler and G.W.Stewart, *LINPACK Users Guide*, SIAM Publications, Philadelphia, 1978.

3 The MathWorks Inc., *PC-MATLAB User's Guide*, 1989.

4 E.E.Swartzlander,Jr., ed., *Systolic Signal Processing Systems*, Marcel Dekker, INC., 1987.

5 S.Y.Kung, K.S.Arun, R.J.Gal-Ezer and D.V.Bhaskar Rao, "Wavefront Array Processor: Language, Architecture, and Applications," *IEEE Trans. on Computers*, Vol.C-31, Nov., pp.1054-1066, 1982.

6 S.Y.Kung and D.V.Bhaskar Rao, "Highly Parallel Architectures to Solving Linear Equations," *Proc. ICASSP*, Atlanta, pp.39-42, 1981.

7 D.E.Rumelhart, G.E.Hinton and R.J.Williams, *Parallel Distributed Processing*, Vol.I, Cambridge, MA:MIT Press, 1986.

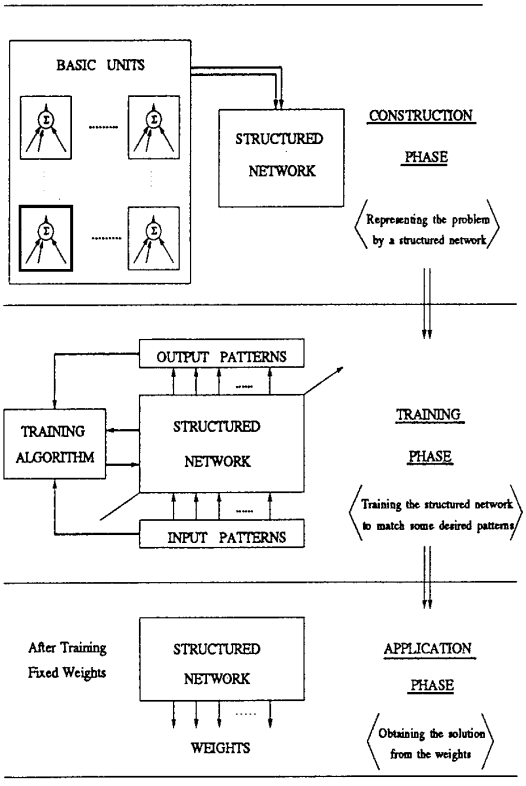

Figure 1: Three phases of our new approaches.

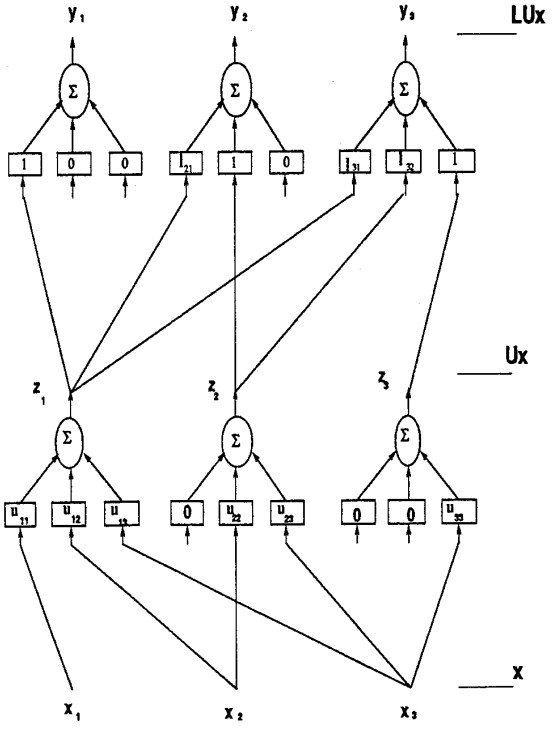

Figure 2: Structured network for matrix LU decomposition for the $m = n = 3$ case.

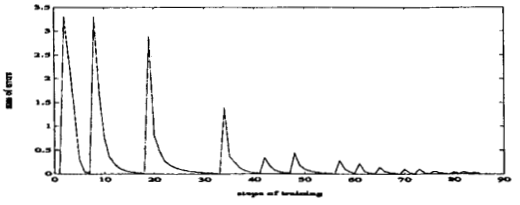

Figure 3: Convergence behavior of the training procedure for Example 1.

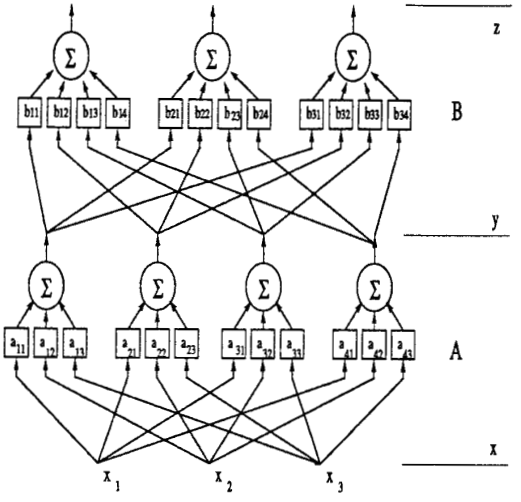

Figure 4: Structured network for linear equations solving for the $m = 4, n = 3$ case.

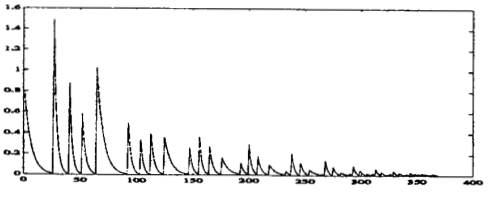

Figure 5: Convergence behavior of the training procedure for Example 2.

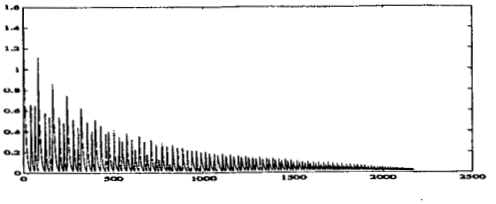

Figure 9: Convergence behavior of NN3 for Example 3.

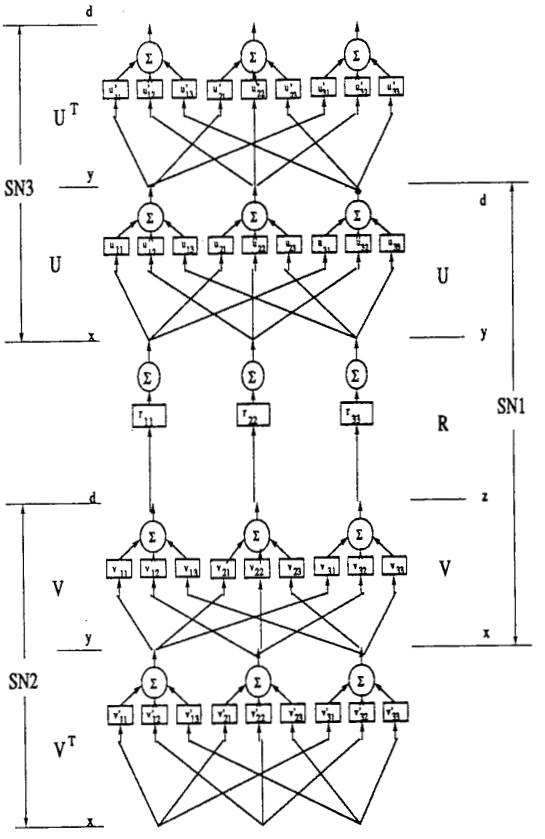

Figure 6: Structured network for matrix singular value decomposition for the $m = n = 3$ case.

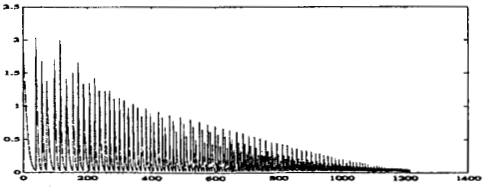

Figure 7: Convergence behavior of NN1 for Example 3.

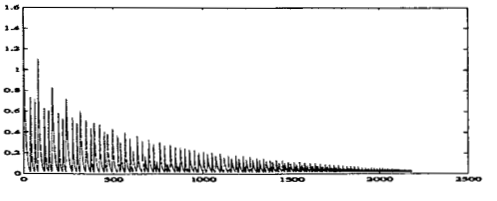

Figure 8: Convergence behavior of NN2 for Example 3.