USC–SIPI REPORT #174

Computational Study of Structured Networks for Linear Algebra

by

Chung-Kuang Chu and Jerry M. Mendel

April 1991

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 400
Los Angeles, CA 90089-2564 U.S.A.

Computational Study of Structured Networks for Linear Algebra

Chung-Kuang Chu and Jerry M. Mendel

1 INTRODUCTION

Structured networks based on the error back-propagation method can solve many linear algebra problems [1, 2, 3]. In this report, the computational requirements for structured networks are analyzed in order to provide computation time information. With this data, we can roughly evaluate the performance of structured networks.

First, computation counts are estimated for serial implementations. These counts are based on a computation schedule which is provided for each linear algebra problem. We also give results for the theoretical processing time (which is estimated from the computation counts) and the actual processing time. Finally, we estimate the processing time for parallel implementations by introducing the basic computing units, which perform the inner product operations. Computation schedules are also provided for the parallel implementations.

2 LU DECOMPOSITION

The simulations for LU decomposition using structured networks follow the algorithm proposed by Wang and Mendel [3]. For an N×N matrix A there are N training patterns which

are denoted by $\{x_i, i = 1, \ldots, N\}$, and, which are linearly independent vectors. In order to reduce the computation time in "one cycle of computation" [3], during which we perform the training procedure once for one pattern, the training patterns are chosen to be $x_i = e_i, i = 1, ..., N$, where e_i is the i^{th} unit vector. The structured network trains on these patterns successively for i=1,2,...,N; training is continued by returning to i=1 and repeating for i=1,2,...,N. We define "one iteration of training" as the procedure of training on one cycle of N patterns. Its processing time T_s is the "basic computational requirement" defined in [3].

Our simulation program is modified from Wang and Mendel's original simulation program. They suggest that a normalization procedure should be included in a preprocessing step. We have included such a procedure, as described next. For an input matrix A whose largest-magnitude element is m, A is divided by m such that $A_1 = (m^{-1})A$. The largest magnitude element of A_1 is unity. This procedure adjusts the dynamic range of the magnitudes of elements of A so that the proper range for correction constant α (which appears in the back propagation training algorithm) will be easy to estimate. Furthermore, this procedure will not change the result of LU decomposition, because, if

$$A = LU, \tag{1}$$

then

$$(m^{-1})A = LU(m^{-1}) \tag{2}$$

i.e.,

$$A_1 = LU_1. \tag{3}$$

The diagonal elements of L are set equal to unity, i.e., they are fixed; therefore, the scaling

only affects U. After the converged U_1 is obtained, U can be recovered by unscaling U_1 by m, i.e.,

$$U = mU_1. \tag{4}$$

The second modification in our simulation is that each training pattern is trained once and then switched to the next pattern, instead of continuously training on one pattern until a convergence criterion is satisfied. The latter method is performed in Wang and Mendel's original algorithm. There are two advantages to this modified approach. First, the algorithm is easier to implement than the original one. We now only need one decision threshold ϵ for stopping instead of two (ϵ_1 and ϵ_2). Secondly, for some examples we have observed that the time needed for convergence is about 20 percent less than that of the original algorithm.

The third modification is that we used another stopping criterion instead of the original one proposed in [3], which was

$$J_1 = \sum_{i=1}^{N}\sum_{j=1}^{N} |A_{ij} - (LU)_{ij}|. \tag{5}$$

The error function adopted in our simulation is

$$J_2 = ||\underline{d}_i - \underline{y}_i||_1, \tag{6}$$

where $\underline{d}_i$=A$\underline{e}_i$ is the desired output vector from the i^{th} training pattern $\underline{e}_i$, and $\underline{y}_i$=LU$\underline{e}_i$ is the actual output vector for $\underline{e}_i$. The training procedure is terminated if N errors $||\underline{d}_i - \underline{y}_i||_1$, i=1,...,N, which are computed during one iteration of training are all less than a preset threshold. The reason we chose J_2 instead of J_1 is that J_2 needs less computation than J_1. In computing J_1, there are N^3 flops in calculating LU and N^2 flops in calculating $\sum_{i=1}^{N}\sum_{j=1}^{N} |A_{ij} - (LU)_{ij}|$. On the other hand, it only takes N flops to compute $||\underline{d}_i - \underline{y}_i||_1$ for

one pattern; hence, N^2 flops are needed for N patterns (one iteration of training).

The common way to evaluate the computational requirements for an algorithm in serial processing is to estimate the number of flops needed in executing the algorithm. According to the definition in [4] a flop is more or less the amount of computation associated with the statement $s := s + a_{ik}b_{kj}$. This measure is adopted in the following analyses.

The estimation of computation counts for the serial implementation of LU decomposition using the structured network depicted in Fig. 1 is summarized in Table 1. Simulation results are given in Table 2. The simulations were performed on a SUN 3/60 for matrices ranging from 3×3 to 12×12. These matrices are made up of random numbers, which were uniformly distributed between -10 and 10. We only estimated the computation requirement for one iteration. Here one randomly generated matrix of a specific dimension was used in the simulation. The computation time in one iteration is fixed because the operations are deterministic (independent of the elements of the matrix). The total computation times may be different for different matrices due to the different convergence rates.

The actual processing time for one iteration, T_s, was obtained by calling a specific system function. This function returned two times, namely, user elapsed time (for the user's program) and system elapsed time (for the system program). In our simulations the system elapsed time was ignored when compared to the user elapsed time because of our single programming environment, and the fact that there was no need to do swaps between main memory and disks for affordable memory. We obtained T_s by this system call. For example, to determine the elapsed time in a specific part of a program we called this function twice, first at the start of this specific part and then at the end of this specific part; then, we computed the difference of the elapsed times.

The ratios of processing times (T_s) in one iteration of training for successively increasing dimensions are also given in Table 2. Comparing the estimated processing time ratios with

the actual time ratios in Table 2, we see they are very close; therefore, our computation count analysis provides very acceptable accuracy. This also makes us confident about the computation analyses for parallel implementations given next, if the hardware architecture of the parallel realization is similar to that used in our software simulation.

For reference, the computation times for LU decomposition using MATLAB are given in Table 3. Here the matrices used are the same as those in the simulations of Table 2 for the structured network. Although MATLAB is faster, recall that our ultimate objective is to implement a structured network in parallel, and in hardware; it is not to compete with serial software such as MATLAB.

For a parallel implementation of the LU decomposition structured network we define two basic computation elements, namely, *forward computing unit* and *backward computing unit*, both of which are shown in Fig. 1. The forward computing unit is a multi-input weighted summer (i.e., an inner product unit). The backward computing unit is an arithmetic processing unit used to update one specific matrix element. There are 2N forward computing units, N of them in layer 1 and N in layer 2. Units in the same layer can be run concurrently. There are N(N-1)/2 backward computing units in layer 1 used to update matrix L and N(N+1)/2 units in layer 2 used to update matrix U.

A computation schedule for parallel implementation of LU decomposition is given in Table 4. The computations performed by the forward computing units are similar to those performed by the backward computing units. Basically they perform *inner product* type operations. In the forward computing unit the inner product operation is easily identified. On the other hand, the operations performed in the backward computing units are also made up of inner product type operations, with some minor modifications which are very clear in the defining equations of backward trainings.

There is a standard architecture to realize inner product operations in parallel [5]. The

inner product $\sum_{i=1}^{n} x_i y_i$ of two n-dimensional vectors can be computed in time $O(\log n)$ ($\log n \equiv \log_2 n$ in this report for simplicity) using n processors [5]. The processor used here is a two-input arithmetic operator, which performs the addition or multiplication. There are N processors in a forward computing unit, and one or N processors (depends on whether there is an inner product operation or not) in a backward computing unit.

Using the computational requirements of serial and parallel implementations, we can calculate a speedup factor. Here we adopt the definition of speedup factor given in [5]. Suppose our parallel algorithm uses p processors (p may depend on N), and requires $T_p(N)$ sec for one iteration of training. Let $T^*(N)$ be the optimal serial time to solve the same problem. The ratio

$$S_p(N) = \frac{T^*(N)}{(number\ of\ iterations) T_p(N)} \tag{7}$$

is called the *speedup* of the algorithm. In our case, the optimal serial time is unknown. We, therefore, use the alternative definition suggested in [5] for $T^*(N)$, namely $T_e(N)$, the processing time in one iteration by serial processing. Hence, from Tables 1 and 4, the *speedup* in the LU decomposition problem is

$$S_p(N) \approx \frac{T_e(N)}{T_p(N)} = \frac{(2.5N^3 + 3N^2 + 0.5N)\tau_s}{(4N log N + 6N)\tau_p} \tag{8}$$

where τ_s is the processing time for one flop in serial processing, τ_p the time for one elementary operation in parallel processing and $p = \frac{1}{2}N^3 + 3N^2 + \frac{1}{2}N$ (see Table 4). Note that for $N = 32$, we expect to achieve a speedup of $102\frac{\tau_s}{\tau_p}$. The ratio τ_s/τ_p is on the order of unity.

3 Singular Value Decomposition

An algorithm for singular value decomposition using a structured network was proposed in [3]. The problem is to decompose a matrix $A \in R^{m\times n}$ into $A = URV$, where $U \in R^{m\times m}$ and $V \in R^{n\times n}$ are orthogonal matrices, i.e., $U^TU = I$ and $VV^T = I$, and $R = diag(r_{11}, r_{22}, ..., r_{pp}) \in R^{m\times n}$, p=min(m,n) and $r_{ii} \geq 0, i = 1, \ldots, p$. There are three overlapped subnetworks in this structured network (see Figure 2), namely SN1, SN2 and SN3. Subnetwork, SN1, performs the operation of decomposing A into URV. Subnetworks SN2 and SN3 constrain V and U to be orthogonal matrices. It has been shown [3] that, for a given matrix $A \in R^{m\times n}$, if the following n patterns for SN1, $(x^{(1)}, Ax^{(1)})$, $(x^{(2)}, Ax^{(2)})$,..., $(x^{(n)}, Ax^{(n)})$, the following n patterns for SN2, $(x^{(1)}, x^{(1)})$, $(x^{(2)}, x^{(2)})$,..., $(x^{(n)}, x^{(n)})$, and the following m patterns for SN3, $(x^{(1)}, x^{(1)})$, $(x^{(2)}, x^{(2)})$,..., $(x^{(m)}, x^{(m)})$, are matched during the training of SN1, SN2 and SN3, the weights in SN1, SN2 and SN3 give the U, R and V matrices.

The computation counts for serial implementation of *one iteration of training* of the Fig. 2 structured network are listed in Table 5. Simulation results are given in Table 6. The simulations were performed on a SUN 3/60 for matrices ranging from 3×3 to 12×12. These matrices were generated in the same way as in the LU decomposition problem. The ratios of processing times (T_s) for one iteration of training and for successively increasing dimensions were calculated. Comparing the estimated processing time ratios with the actual time ratios in Table 6, we see they are also very close; therefore, our computation count analyses offer acceptable accuracy. For reference, the computation time for SVD using MATLAB is given in Table 7.

The computation counts for parallel implementation of one iteration of training of the SVD structured network are given in Table 8. The basic computing processors are illustrated

in Fig. 2. There are 5N forward computing units in this 5-layer network, N in each layer. There are N^2 backward computing units in layer 1, N^2 units in layer 2, N units in layer 3, N^2 units in layer 4 and N^2 units in layer 5. Units in the same layer are run concurrently. Furthermore, SN2 and SN3 are updated simultaneously. The computation requirements are equivalent in SN2 and SN3; therefore, no waiting time is needed for synchronizing the subsequent processing.

From Tables 5 and 8 the *speedup* for the SVD problem is

$$S_p(N) = \frac{(14N^3 + 13N^2)\tau_s}{(9NlogN + 17N)\tau_p} \tag{9}$$

where τ_s is the processing time for one flop in serial processing, τ_p the time for one elementary operation in parallel processing and $p = 3N^3+8N^2+4N$ (see Table 8). For N=32, we expect to achieve a speedup of $238\frac{\tau_s}{\tau_p}$.

4 Linear Equation Solving

The structured network for solving Ax=b, where A is an m×n matrix, is depicted in Figure 3 for m=4 and n=3. The computation schedules for solving Ax=b are given in Tables 9 and 10 for serial and parallel implementations. In the parallel scheme, there are (m+n) forward computing units, n in layer 1 and m in layer 2. Each unit uses one flop and units in the same layer can be run concurrently. There are mn backward computing units in layer 1 used to update B. No backward computing units are needed in layer 2 because the elements of A are fixed.

From Tables 9 and 10, the *speedup* for the linear equation solving problem is

$$S_p(m,n) = \frac{(3mn^2 + mn + n^2)\tau_s}{(2nlogn + nlogm + 3n)\tau_p} \tag{10}$$

where $p = 3mn + n$ (see Table 10). For a 32×32 matrix, we expect to achieve a speedup of $174\frac{\tau_s}{\tau_p}$.

5 Similarity Transformation

The structured network for similarity transformation [3] A=PBQ is shown in Figure 4, where A is an N×N real symmetric matrix, the columns of P are eigenvectors of A, B is a diagonal matrix whose elements are the eigenvalues of A, and PQ=I. The computation schedules for serial and parallel implementations of similarity transformation are summarized in Tables 11 and 12. In the parallel implementation, there are 3N forward computing units, N in layer 1, N in layer 2 and N in layer 3. Each unit uses one flop, and units in the same layer can be run concurrently. There are N^2 backward computing units in layer 1 used to update P, N units in layer 2 used to update B, and N^2 units in layer 3 used to update Q.

From Tables 11 and 12 the *speedup* for the similarity transformation problem is

$$S_p(N) = \frac{(10N^3 + 10N^2)\tau_s}{(9NlogN + 16N)\tau_p} \tag{11}$$

where $p = 2N^3 + 5N^2 + 3N$ (see Table 12). For N=32, we expect to achieve a speedup of $173\frac{\tau_s}{\tau_p}$.

6 Lyapunov Equation Solving

Consider the Lyapunov equation

$$AP + PA^T = -W \tag{12}$$

where A$\in R^{N\times N}$ and W$\in R^{N\times N}$, W^T=W are given, and solution P is a symmetric $N \times N$ matrix. The structured network for solving this Lyapunov equation is shown in Figure 5. The computation schedules for serial and parallel implementations are given in Tables 13 and 14, respectively. In the parallel scheme, there are 5N forward computing units, 2N in layer 1, 2N in layer 2 and N used to sum up the outputs of left and right parts (see Figure 5). Each unit uses one flop and units in the same layer can be run concurrently. There are N^2 backward computing units in layer 1 used to update $P^{(U)}$ and N^2 units in layer 2 for updating $P^{(L)}$.

From Tables 13 and 14, the *speedup* for solving the Lyapunov equation is

$$S_p(N) = \frac{(7N^3 + 6N^2)\tau_s}{(7NlogN + 8N)\tau_p} \tag{13}$$

where $p = N^3+6N^2+N$ (see Table 14). For N=32, we expect to achieve a speedup of $171\frac{\tau_s}{\tau_p}$.

7 Discussion

From the preceding analyses of computational requirements for linear algebra problems using structured networks, we see that the order of *speedup* for the parallel schemes is $O(N^2/logN)$. This is due to the fact that we have assumed that every matrix element can be updated concurrently. The speedup factors and number of basic processors used for the five linear algebra problems reported on herein are summarized in Table 15.

The major advantage of parallel structured networks is that the basic computing units are *inner product* units. This will make VLSI implementations of these networks easy because of a highly regular architecture.

References

[1] Lixin Wang and J.M.Mendel, "Structured Trainable Networks for Matrix Algebra," *Proceedings of Int. Joint Conf. on Neural Networks*, pp.II125-132, June 1990.

[2] Li-Xin Wang and Jerry M. Mendel, "Matrix Computations and Equation Solving Using Structured Networks and Training," *Proceedings of IEEE Conf. on Decision and Control*, Vol. 3, pp. 1747-1750, Dec. 1990.

[3] Lixin Wang and J.M.Mendel, "Structured Networks for Matrix Computations and Equation Solving," Submitted for publication, 1991.

[4] G.H.Golub and C.F.Van Loan, *Matrix Computations,* Johns Hopkins, Baltimore, MD, 1983.

[5] D.P.Bertsekas and J.N.Tsitsiklis, *Parallel and Distributed Computation,* Prentice Hall, Englewood-Cliffs, NJ, 1989.

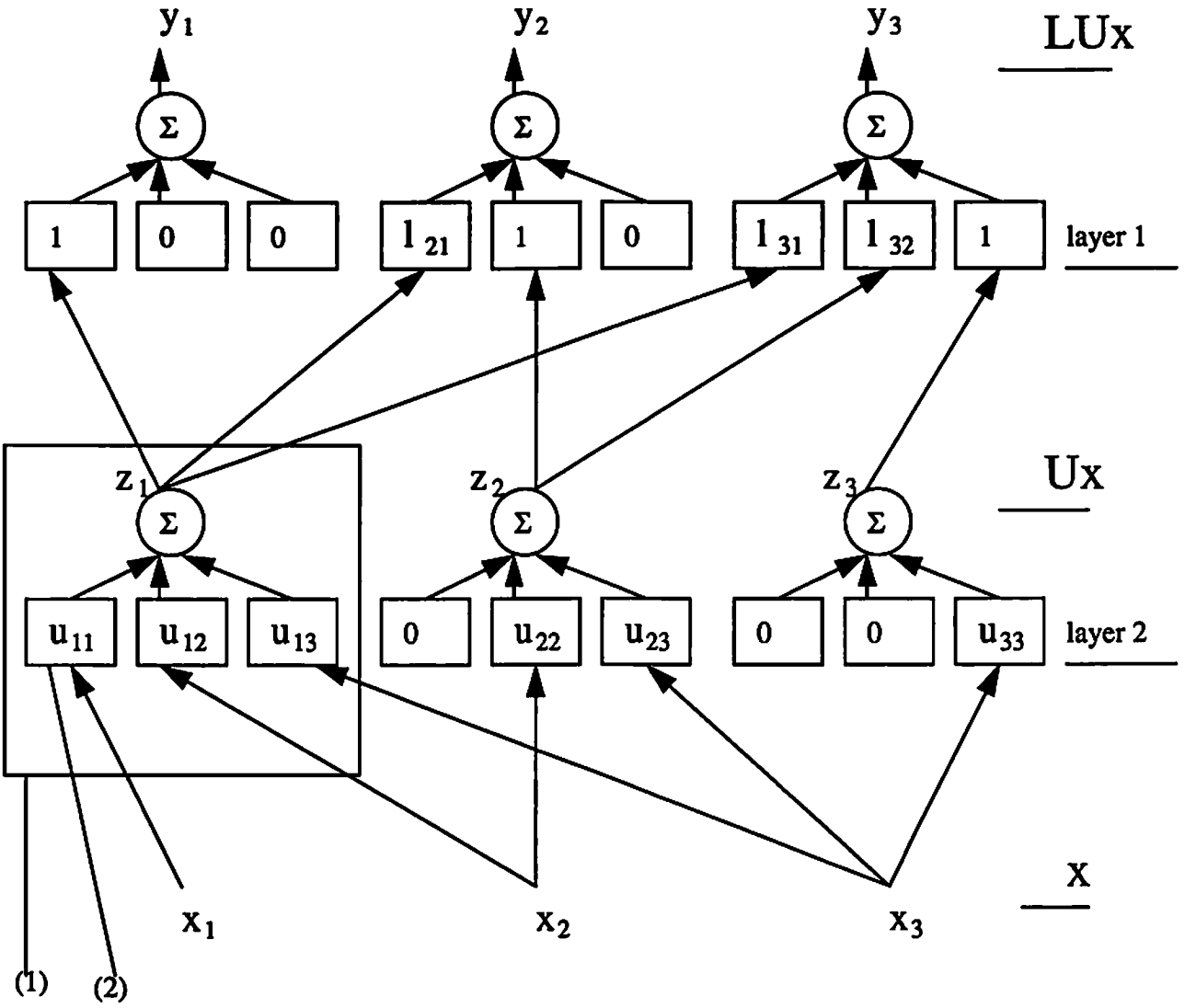

Figure 1: LU decomposition using structured network for N=3 case: (1) Forward computing unit; (2) Backward computing unit.

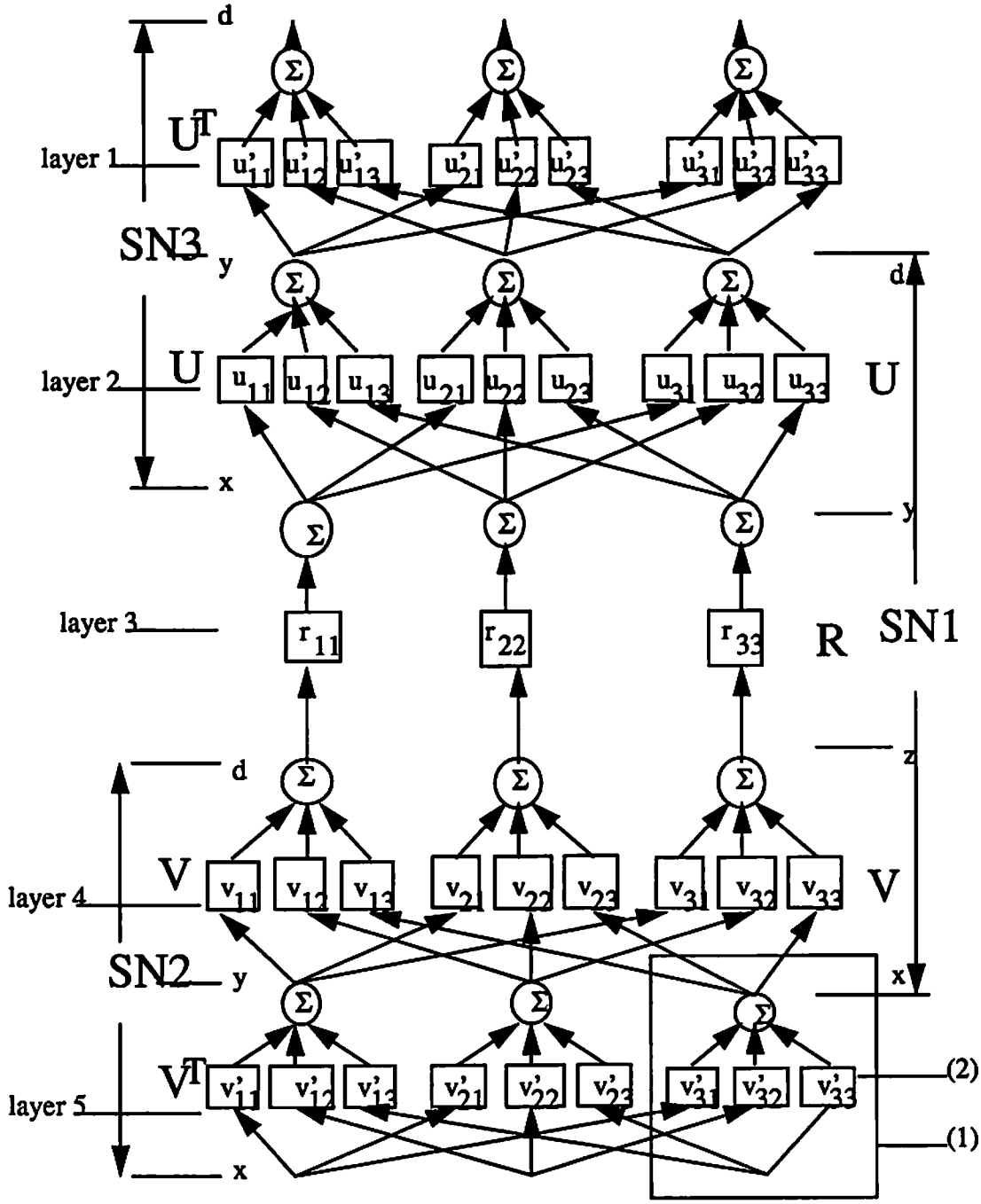

Figure 2: Structured network for SVD for m=n=N=3 case: (1) Forward computing unit; (2) Backward computing unit.

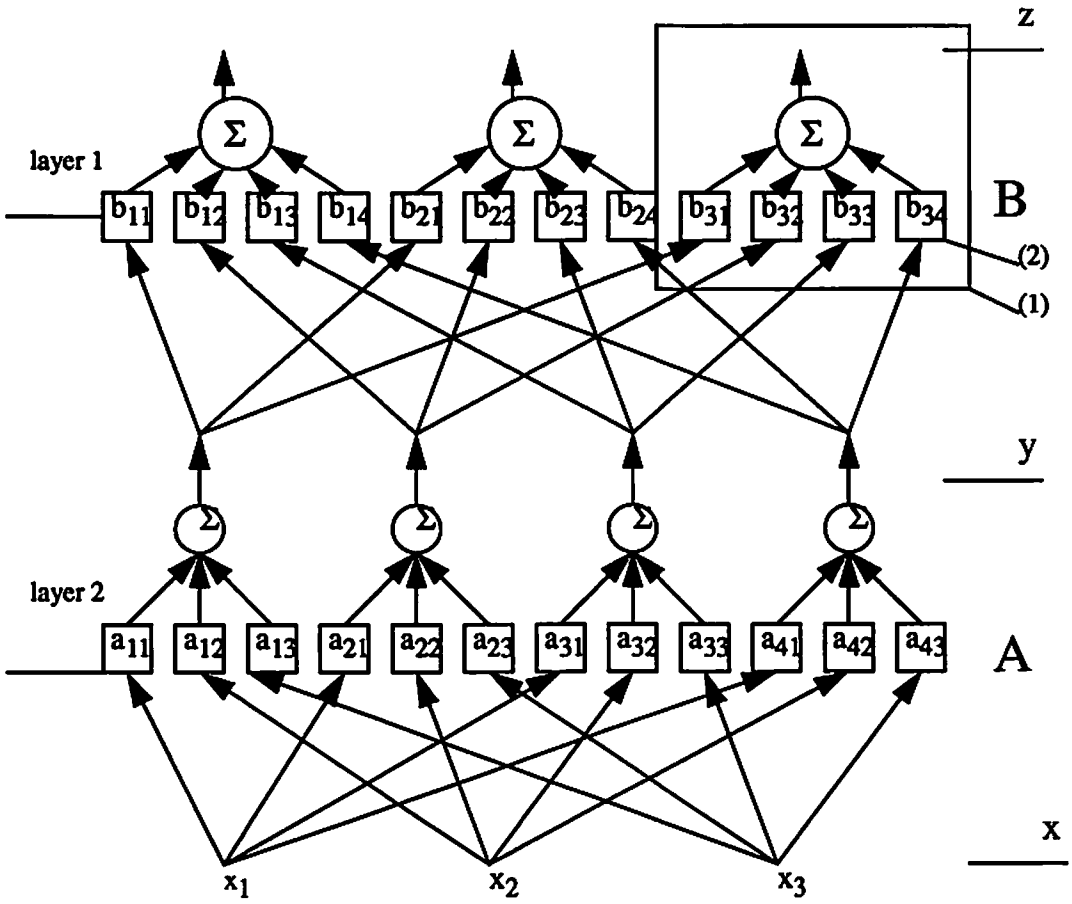

Figure 3: Solving linear equations Ax=b using structured network, where A is an $m \times n$ matrix, for m=4, n=3 case: (1) Forward computing unit; (2) Backward computing unit.

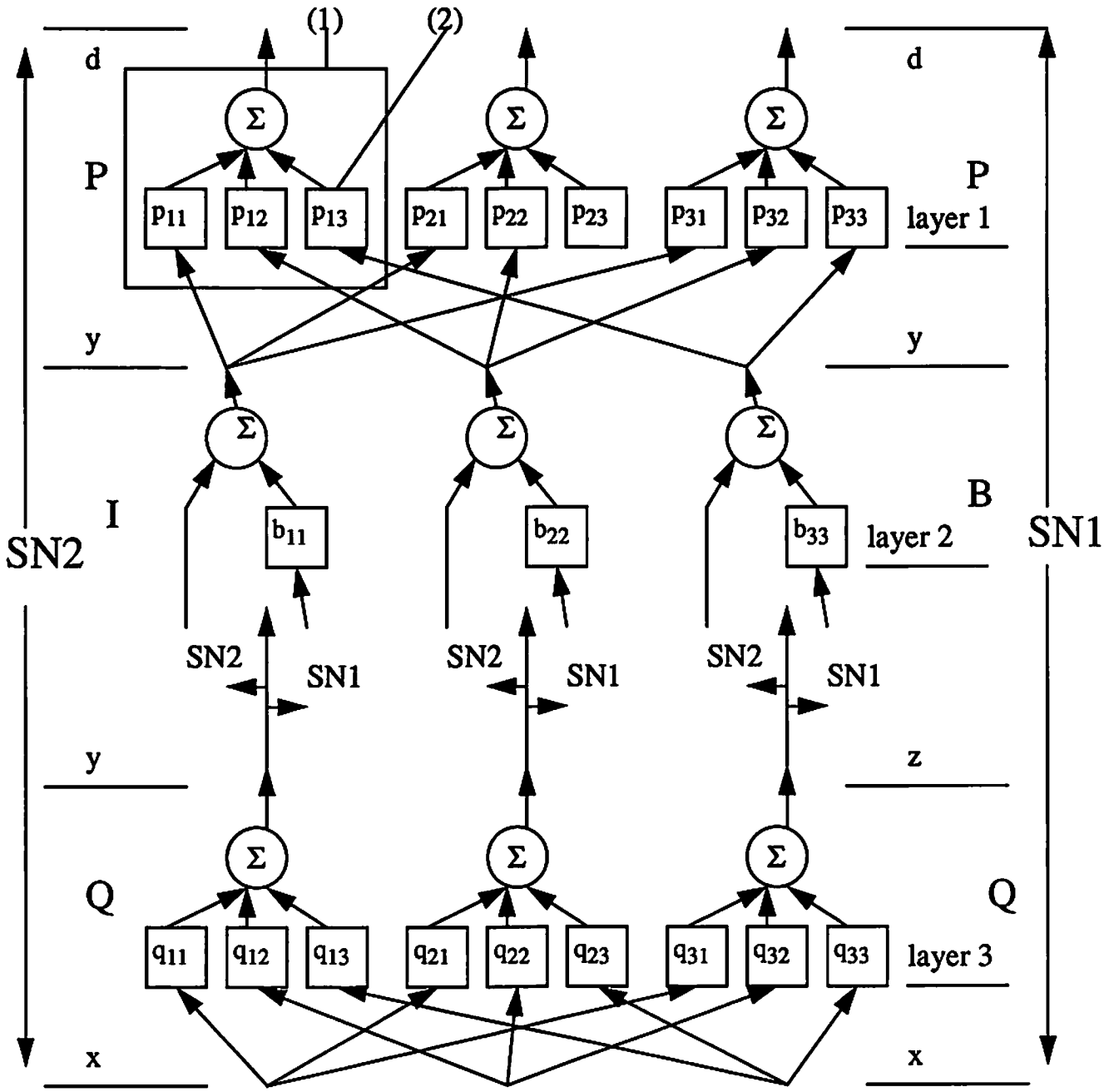

Figure 4: Similarity transformation using structured network for N=3 case, A=PBQ, PQ=I: (1) Forward computing unit; (2) Backward computing unit.

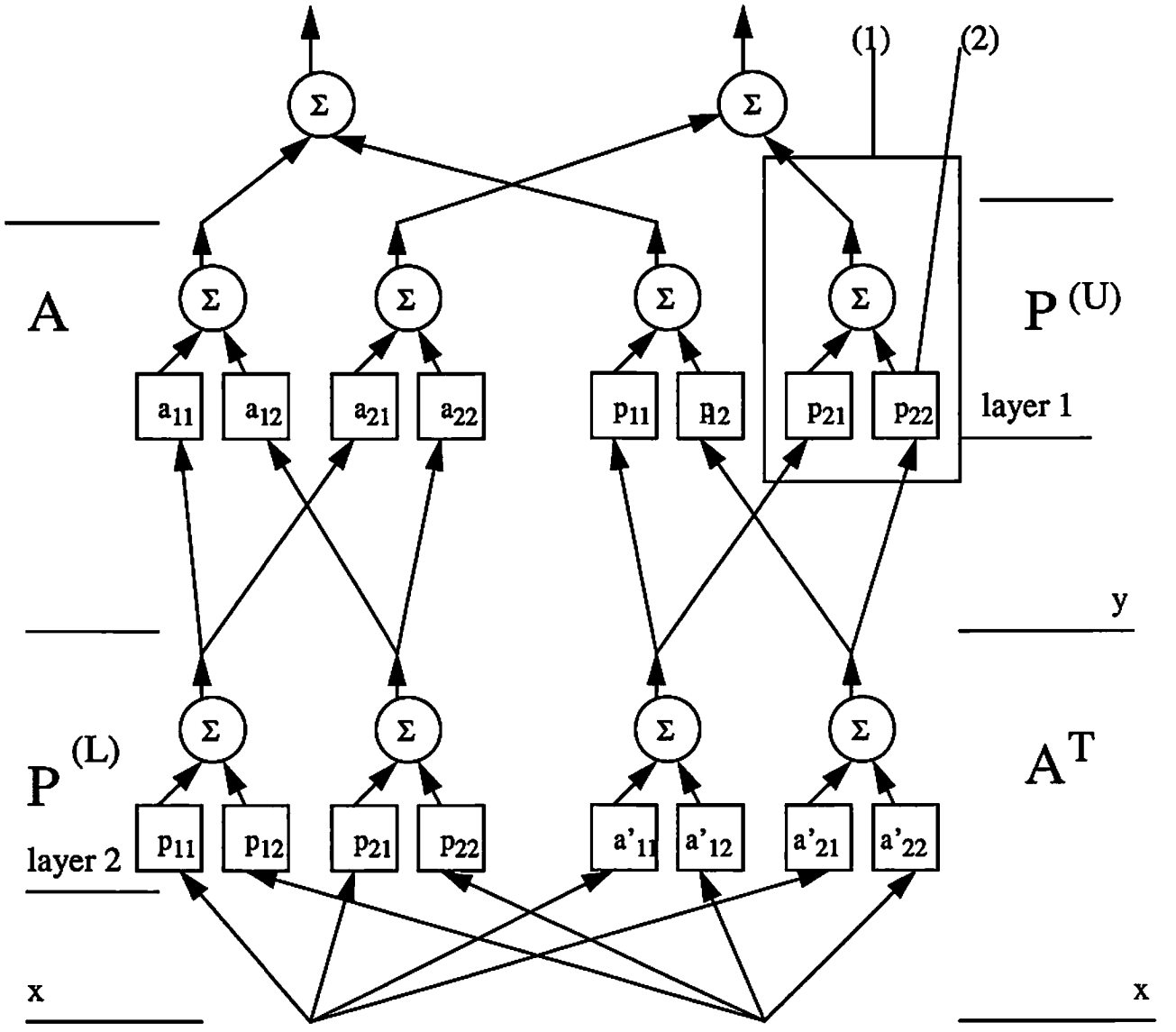

Figure 5: Lyapunov equation solving using structured network for N=2 case, $AP + PA^T = -W$: (1) Forward computing unit; (2) Backward computing unit.

Table 1: Computational requirements of LU decomposition in one iteration of training using structured network and serial implementation

Operations		computations (flops) for one pattern	computations for N patterns
compute desired output vector	$\underline{d}=A\underline{e}$	N	N^2
check stopping criterion	$\sum_{i=1}^{N} \lvert d_i-y_i \rvert$	N	N^2
forward computation $\underline{y}_i=(LU)\underline{e}_i$	$U\underline{e}_i$	N	N^2
	$L(U\underline{e}_i)$	N^2	N^3
backward training	1. $l_{ij}(t+1) = l_{ij}(t) + \alpha\,(d_i-y_i)\,z_j$, i=2,...,N, j=1,...,(i-1)		
	$l_{ij}(t) + \alpha\,(d_i - y_i)\,z_j$	2	
	total for N(N-1)/2 elements	N^2-N	N^3-N^2
	2. $u_{ij}(t+1) = u_{ij}(t) + \alpha\,[\sum_{k=1}^{N} (d_k - y_k)\,l_{ki}]\,e_j$, i=1,...,j; j=1,...,N.		
	$\sum_{k=1}^{N} (d_k - y_k)\,l_{ki}$	N	
	$u_{ij}(t) + \alpha\,[\sum_{k=1}^{N} (d_k - y_k)\,l_{ki}]\,e_j$	1	
	subtotal for one element of U	N+1	
	total of 1+2+$\cdots$+N elements to be updated	$(N^2+2N+1)/2$	$(N^3+2N^2+N)/2$
TOTAL OPERATIONS		$2.5N^2+3N+0.5$	$2.5N^3+3N^2+0.5N$

Table 2: Comparison of computation counts and actual processing time for one iteration of LU decomposition

N^a	3	4	5	6	7	8	9	10	11	12
$T_e(N)^b$	96	210	390	651	1008	1476	2070	2805	3696	4758
$\frac{T_e(N)}{T_e(N-1)}$		2.18	1.85	1.66	1.54	1.46	1.40	1.35	1.31	1.28
$T_s(N)^c$	4.56	9.15	16.6	27.0	42.0	60.5	86.3	114.5	151.8	195.7
$\frac{T_s(N)}{T_s(N-1)}$		2.00	1.81	1.63	1.55	1.44	1.43	1.32	1.32	1.29

[a]N: Dimension of matrix A.
[b]$T_e(N)$: Computation count approximated by $2.5N^3 + 3N^2 + 0.5N$, as given in Table 1.
[c]$T_s(N)$: The "basic computation requirement" processing time (msec) in our simulation.

Table 3: Computation time for LU decomposition using MATLAB

N[a]	3	4	5	6	7	8	9	10	11	12
T(N)[b]	4.9	7.2	8.1	9.9	12.7	16.2	20.1	24.8	30.2	36.3

[a]N: Dimension of matrix A.
[b]T(N): Computation time in msec.

Table 4: Computational requirements of LU decomposition in one iteration of training using structured network and parallel implementation

Operations		computations for one pattern, and [number of processors][a]	computations for N patterns, and [number of processors]
compute desired output vector[b]	$\underline{d}=A\underline{e}$	0	0
check stopping criterion	$\sum_{i=1}^{N} \lvert d_i-y_i \rvert$	logN, [N]	NlogN, [N]
forward computation $\underline{y}_i=(LU)\underline{e}_i$	$U\underline{e}_i$, (layer 2 in Fig. 1)	logN, [N^2]	NlogN, [N^2]
	$L(U\underline{e}_i)$, (layer 1 in Fig. 1)	logN, [N^2]	NlogN, [N^2]
backward training[c,d,e]	1. $l_{ij}(t+1) = l_{ij}(t) + \alpha\ (d_i-y_i)\ z_j$, i=2,...,N, j=1,...,(i-1). (layer 1 in Fig. 1)		
	$l_{ij}(t) + \alpha\ (d_i - y_i)\ z_j$	4	
	total for N(N-1)/2 elements	4, [N(N-1)/2]	4N, [N(N-1)/2]
	2. $u_{ij}(t+1) = u_{ij}(t) + \alpha\ [\sum_{k=1}^{N} (d_k - y_k)\ l_{ki}]\ e_j$, i=1,...,j. (layer 2 in Fig. 1)		
	$\sum_{k=1}^{N} (d_k - y_k)\ l_{ki}$	logN, [N]	NlogN, [N]
	$u_{ij}(t) + \alpha\ [\sum_{k=1}^{N} (d_k - y_k)\ l_{ki}]\ e_j$	2	
	subtotal for one element of U	logN+2, [N]	NlogN+2N, [N]
	total for N(N+1)/2 elements	logN+2, $[N^2(N+1)/2]$	NlogN+2N, $[N^2(N+1)/2]$
TOTAL OPERATIONS		4logN+6, $[\frac{1}{2}N^3+3N^2+\frac{1}{2}N]$	4NlogN+6N, $[\frac{1}{2}N^3+3N^2+\frac{1}{2}N]$

[a]The number of basic processors is illustrated in Fig. 1.

[b]Matrix A is stored in memory and, for simplicity, the access time is ignored.

[c]All the elements of matrix L can be updated concurrently.

[d]The quantity $(d_k - y_k)$ is available from "check stopping criterion."

[e]All the elements of matrix U can be updated concurrently.

Table 5: Computational requirements of SVD in one iteration of training using structured network and serial implementation

Operations[a]			Computations (flops) for one pattern	Computations for N patterns
SN1	compute desired output vector	$\underline{\hat{d}} = A\underline{e}$	N	N^2
	forward computation	$\underline{z}$=V$\underline{e}$	N	N^2
		$\underline{y}$=R$\underline{z}$	N	N^2
		$\underline{d}$=U$\underline{y}$	N^2	N^3
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
	backward training	1. $u_{ij}(t+1) = u_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$u_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	2	
		total for N^2 elements	$2N^2$	$2N^3$
		2. $r_{ii}(t+1) = r_{ii}(t) + \alpha(\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1))z_i$, i=1,...,N		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)$	N	
		$r_{ii}(t)+\alpha(\sum_{k=1}^{N}(d_k-\hat{d}_k)u_{ki}(t+1))z_i$	2	
		total for N elements	N^2+2N	N^3+$2N^2$
		3. $v_{ij}(t+1) = v_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)]x_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)$	2N	
		$v_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)]x_j$	1	
		total for N elements	$2N^2$+N	$2N^3$+N^2
SN1 TOTALS			$6N^2$+7N	$6N^3$+$7N^2$

[a]Refer to Fig. 2.

Table 5: (continued)

Operations[a]			Computations (flops) for one pattern	Computations for N patterns
SN2	compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=V$^t\underline{e}$	N	N^2
		$\underline{d}$=V$\underline{y}$	N^2	N^3
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
	backward training	1.$v_{ij}(t+1) = v'_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$v'_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	2	
		total for N^2 elements	$2N^2$	$2N^3$
		2. $v'_{ji}(t+1) = v_{ij}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)]x_i$, j=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)$	N	
		$v'_{ji}(t+1) = v_{ij}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)]x_i$	1	
		total for N elements	N^2+N	N^3+N^2
SN2 TOTALS			$4N^2$+3N	$4N^3$+$3N^2$

Table 5: (continued)

Operations[a]			Computations (flops) for one pattern	Computations for N patterns
SN3	compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=U$\underline{e}$	N	N^2
		$\underline{d}$=U$^t\underline{y}$	N^2	N^3
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
	backward training	1. $u'_{ji}(t+1) = u_{ij}(t) + \alpha[d_j - \hat{d}_j]y_i$, i=1,...,N,j=1,...,N.		
		$u_{ij}(t) + \alpha[d_j - \hat{d}_j]y_i$	2	
		total for N^2 elements	$2N^2$	$2N^3$
		2. $u_{ij}(t+1) = u'_{ji}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)]x_j$,i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)$	N	
		$u'_{ji}(t+1)+\alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)]x_j$	1	
		total for N elements	N^2+N	N^3+N^2
SN3 TOTALS			$4N^2$+3N	$4N^3$+$3N^2$
OVERALL TOTALS			$14N^2$+13N	$14N^3$+$13N^2$

Table 6: Comparison of computation counts and actual processing time for one iteration of SVD decomposition

N^a	3	4	5	6	7	8	9	10	11	12
$T_e(N)^b$	486	1088	2050	3456	5390	7936	11178	15200	20086	25920
$\frac{T_e(N)}{T_e(N-1)}$		2.24	1.88	1.68	1.60	1.47	1.41	1.36	1.32	1.29
$T_s(N)^c$	23	39	74	158	246	360	515	700	920	1170
$\frac{T_s(N)}{T_s(N-1)}$		1.64	1.90	2.13	1.56	1.46	1.42	1.35	1.31	1.29

[a] N: Dimension of matrix A.
[b] $T_e(N)$: Computation count approximated by $14N^3 + 13N^2$, as given in Table 5.
[c] $T_s(N)$: The "basic computation requirement" processing time (msec) in our simulation.

Table 7: Computation time for SVD using MATLAB

N^a	3	4	5	6	7	8	9	10	11	12	13	14
$T(N)^b$	18.5	29.5	53.4	74.6	106	148	221	259	315	408	543	616

[a] N: Dimension of matrix A.
[b] $T(N)$: Computation time in msec.

Table 8: Computational requirements of SVD in one iteration of training using structured network and parallel implementation

Operations[a]			Computations for one pattern, and [number of processors]	Computations for N patterns, [number of processors]
SN1	compute desired output vector[b]	$\underline{\hat{d}} = A\underline{e}$	0	0
	forward computation	$\underline{z}$=V$\underline{e}$, (layer 4)	logN, [N^2]	NlogN, [N^2]
		$\underline{y}$=R$\underline{z}$, (layer 3)	1, [N]	N, [N]
		$\underline{d}$=U$\underline{y}$, (layer 2)	logN, [N^2]	NlogN, [N^2]
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	logN, [N]	NlogN, [N]
	backward training[c,d,e,f]	1. $u_{ij}(t+1) = u_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j, i = 1,\ldots,N, j = 1,\ldots,N$		
		$u_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	4	
		total for N^2 elements	4, [N^2]	4N, [N^2]
		2. $r_{ii}(t+1) = r_{ii}(t) + \alpha(\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1))z_i$, i=1,...,N		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)$	logN, [N]	NlogN, [N]
		$r_{ii}(t)+\alpha(\sum_{k=1}^{N}(d_k-\hat{d}_k)u_{ki}(t+1))z_i$	3	
		total for N elements	logN+3, [N^2]	NlogN+3N, [N^2]
		3. $v_{ij}(t+1) = v_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)]x_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)$	logN, [N]	NlogN, [N]
		$v_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u_{ki}(t+1)r_{ii}(t+1)]x_j$	3	
		total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
	SN1 TOTALS		5logN+11, [N^3+4N^2+2N]	5NlogN+11N, [N^3+4N^2+2N]

[a]The operations and number of basic processors are illustrated in Fig. 2.
[b]Matrix A is stored in memory and, for simplicity, access time is ignored.
[c]All the elements of matrix U can be updated concurrently.
[d]The quantity $(d_k - \hat{d}_k)$ is available from "check stopping criterion."
[e]All the elements of matrix R can be updated concurrently.
[f]All the elements of matrix V can be updated concurrently.

Table 8: (continued)

Operations			Computations (flops) for one pattern, and [number of processors]	Computations for N patterns, [number of processors]
SN2	compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=V$^t\underline{e}$, (layer 5)	logN, [N^2]	NlogN, [N^2]
		$\underline{d}$=V$\underline{y}$, (layer 4)	logN, [N^2]	NlogN, [N^2]
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	logN, [N]	NlogN, [N]
	backward training[g,h]	1. $v_{ij}(t+1) = v'_{ji}(t) + \alpha[d_i - \hat{d}_i]y_j, i = 1, \ldots, N, j = 1, \ldots, N.$ (layer 4)		
		$v'_{ji}(t) + \alpha[d_i - \hat{d}_i]y_j$	3	
		total for N^2 elements	3, [N^2]	3N, [N^2]
		2. $v'_{ji}(t+1) = v_{ij}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)]x_i, j = 1, \ldots, N.$ (layer 5)		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)$	logN, [N]	NlogN, [N]
		$v'_{ji}(t+1) = v_{ij}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)v_{kj}(t+1)]x_i$	3	
		total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
	SN2 TOTALS		4logN+6, [N^3+3N^2+N]	4NlogN+6N, [N^3+3N^2+N]

[g]All the elements of matrix V can be updated concurrently.
[h]All the elements of matrix V^t can be updated concurrently.

Table 8: (continued)

Operations			Computations (flops) for one pattern, and [number of processors]	Computations for N patterns, [number of processors]
SN3	compute desired output vector	$\hat{\underline{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=U$\underline{e}$, (layer 2)	logN, [N^2]	NlogN, [N^2]
		$\underline{d}$=$U^t\underline{y}$, (layer 1)	logN, [N^2]	NlogN, [N^2]
	check stopping criterion	$\|\|\underline{d} - \hat{\underline{d}}\|\|_1$	logN, [N]	NlogN, [N]
	backward training[i,j]	1. $u'_{ji}(t+1) = u_{ij}(t) + \alpha[d_j - \hat{d}_j]y_i$, i=1,...,N,j=1,...,N. (layer 1)		
		$u_{ij}(t) + \alpha[d_j - \hat{d}_j]y_i$	3	
		total for N^2 elements	3, [N^2]	3N, [N^2]
		2. $u_{ij}(t+1) = u'_{ji}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)]x_j$,i=1,...,N. (layer 2)		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)$	logN, [N]	NlogN, [N]
		$u'_{ji}(t+1) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)u'_{ki}(t+1)]x_j$	3	
		total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
	SN3 TOTALS		4logN+6, [N^3+3N^2+N]	4NlogN+6N, [N^3+3N^2+N]
OVERALL TOTALS[k]			9logN+17, [3N^3+8N^2+4N]	9NlogN+17N, [3N^3+8N^2+4N]

[i]All the elements of matrix U^t can be updated concurrently.
[j]All the elements of matrix U can be updated concurrently.
[k]SN2 and SN3 are run concurrently.

Table 9: Computational requirements of linear equation solving in one iteration of training using structured network and serial implementation

Operations[a]		computations (flops) for one pattern	computations for n patterns
compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
check stopping criterion	$\|\underline{z} - \underline{\hat{d}}\|_1$	n	n^2
forward computation $\underline{z}$=(BA)$\underline{e}$	$\underline{y}$=A$\underline{e}$, (layer 2)	m	mn
	$\underline{z}$=B(A$\underline{e}$), (layer 1)	mn	mn^2
backward training	$b_{ij}(t+1) = b_{ij}(t) + \alpha(e_i - z_i)y_j$, i=1,...,n, j=1,...,m. (layer 1)		
	$b_{ij}(t) + \alpha(e_i - z_i)y_j$	2	
	total for (mn) elements in B	2mn	$2mn^2$
TOTAL OPERATIONS			$3mn^2+mn+n^2$

[a]Refer to Fig. 3.

Table 10: Computational requirements of linear equation solving in one iteration of training using structured network and parallel implementation

Operations		computations for one pattern and [number of processors][a]	computations for n patterns and [number of processors]
compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
check stopping criterion	$\|\|\underline{z} - \underline{\hat{d}}\|\|_1$	logn, [n]	nlogn, [n]
forward computation $\underline{z}$=(BA)$\underline{e}$	$\underline{y}$=A$\underline{e}$, (layer 2)	logn, [mn]	nlogn, [mn]
	$\underline{z}$=B(A$\underline{e}$), (layer 1)	logm, [mn]	nlogm, [mn]
backward training[b]	$b_{ij}(t+1) = b_{ij}(t) + \alpha(e_i - z_i)y_j$, i=1,...,n, j=1,...,m. (layer 1)		
	$b_{ij}(t) + \alpha(e_i - z_i)y_j$	3	
	total for (mn) elements in B	2, [mn]	3n, [mn]
TOTAL OPERATIONS		2logn+logm+3, [3mn+n]	2nlogn+nlogm+3n, [3mn+n]

[a]The operations and number of basic processors are illustrated in Fig. 3.
[b]All the elements of matrix B can be updated concurrently.

Table 11: Computational requirements of similarity transformation in one iteration of training using structured network and serial implementation

Operations[a]			Computations (flops) for one pattern	Computations for N patterns
SN1	compute desired output vector	$\underline{\hat{d}} = A\underline{e}$	N	N^2
	forward computation	$\underline{z}$=Q$\underline{e}$	N	N^2
		$\underline{y}$=B$\underline{z}$	N	N^2
		$\underline{\hat{d}}$=P$\underline{y}$	N^2	N^3
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
	backward training	1. $p_{ij}(t+1) = p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	2	
		total for N^2 elements	$2N^2$	$2N^3$
		2. $b_{ii}(t+1) = b_{ii}(t) + \alpha(\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1))z_i$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)$	N	
		$b_{ii}(t)+\alpha(\sum_{k=1}^{N}(d_k-\hat{d}_k)p_{ki}(t+1))z_i$	2	
		total for N elements	N^2+2N	N^3+$2N^2$
		3. $q_{ij}(t+1) = q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)]e_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)$	2N	
		$q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)]e_j$	1	
		total for N elements	$2N^2$+N	$2N^3$+N^2
SN1 TOTALS			$6N^2$+7N	$6N^3$+$7N^2$

[a]Refer to Fig. 4.

Table 11: (continued)

Operations			Computations (flops) for one pattern	Computations for N patterns
SN2	compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=Q$\underline{e}$	N	N^2
		$\underline{d}$=P$\underline{y}$	N^2	N^3
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
	backward training	1. $p_{ij}(t+1) = p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	2	
		total for N^2 elements	$2N^2$	$2N^3$
		2. $q_{ij}(t+1) = q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}]e_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}$	N	
		$q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}]e_j$	1	
		total for N elements	N^2+N	N^3+N^2
SN2 TOTALS			$4N^2$+3N	$4N^3$+$3N^2$
OVERALL TOTALS			$10N^2$+10N	$10N^3$+$10N^2$

Table 12: Computational requirements of similarity transformation in one iteration of training using structured network and parallel implementation.

Operations[a]			Computations for one pattern, and [number of processors]	Computations for N patterns, [number of processors]
SN1	compute desired output vector[b]	$\underline{\hat{d}} = A\underline{e}$	0	0
	forward computation	$\underline{z}$=Q$\underline{e}$, (layer 3)	logN, [N^2]	NlogN, [N^2]
		$\underline{y}$=B$\underline{z}$, (layer 2)	1, [N]	N, [N]
		$\underline{d}$=P$\underline{y}$, (layer 1)	logN, [N^2]	NlogN, [N^2]
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	logN, [N]	NlogN, [N]
	backward training[c,d,e,f]	1. $p_{ij}(t+1) = p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	3	
		total for N^2 elements	3, [N^2]	3N, [N^2]
		2. $b_{ii}(t+1) = b_{ii}(t) + \alpha(\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1))z_i$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)$	logN, [N]	NlogN, [N]
		$b_{ii}(t)+\alpha(\sum_{k=1}^{N}(d_k-\hat{d}_k)p_{ki}(t+1))z_i$	3	
		total for N elements	logN+3, [N^2]	NlogN+3N, [N^2]
		3. $q_{ij}(t+1) = q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)]e_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)$	logN, [N]	NlogN, [N]
		$q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}(t+1)b_{ii}(t+1)]e_j$	3	
		total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
SN1 TOTALS			5logN+10, [N^3+4N^2+2N]	5NlogN+10N, [N^3+4N^2+2N]

[a]The operations and number of basic processors are illustrated in Fig. 4.
[b]Matrix A is stored in memory and, for simplicity, access time is ignored.
[c]All the elements of matrix P can be updated concurrently.
[d]The quantity $(d_k - \hat{d}_k)$ is available from "check stopping criterion."
[e]All the elements of matrix B can be updated concurrently.
[f]All the elements of matrix Q can be updated concurrently.

Table 12: (continued)

Operations			Computations for one pattern, and [number of processors]	Computations for N patterns and [number of processors]
SN2	compute desired output vector	$\underline{\hat{d}} = \underline{e}$	0	0
	forward computation	$\underline{y}$=Q$\underline{e}$, (layer 3)	logN, [N^2]	NlogN, [N^2]
		$\underline{d}$=P$\underline{y}$, (layer 1)	logN, [N^2]	NlogN, [N^2]
	check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	logN, [N]	NlogN, [N]
	backward training[g,h]	1. $p_{ij}(t+1) = p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
		$p_{ij}(t) + \alpha[d_i - \hat{d}_i]y_j$	3	
		total for N^2 elements	3, [N^2]	3N, [N^2]
		2. $q_{ij}(t+1) = q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}]e_j$, i=1,...,N.		
		$\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}$	logN, [N]	NlogN, [N]
		$q_{ij}(t) + \alpha[\sum_{k=1}^{N}(d_k - \hat{d}_k)p_{ki}]e_j$	3	
		total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
SN2 TOTALS			4logN+6, [N^3+3N^2+N]	4NlogN+6N, [N^3+3N^2+N]
OVERALL TOTALS[i]			9logN+16, [2N^3+5N^2+3N]	9NlogN+16N, [2N^3+5N^2+3N]

[g] All the elements of matrix P can be updated concurrently.
[h] All the elements of matrix Q can be updated concurrently.
[i] SN1 and SN2 can not be processed simultaneously.

Table 13: Computational requirements of Lyapunov equation solving in one iteration of training using structured network and serial implementation

Operations[a]		computations (flops) for one pattern	computations for N patterns
compute desired output vector	$\underline{\hat{d}} = -W\underline{e}$	N	N^2
check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	N	N^2
forward computation	$\underline{d}=(P^{(U)}A^T + AP^{(L)})\ \underline{e}$		
	$P^{(U)}A^T\underline{e}$	N^2+N	N^3+N^2
	$AP^{(L)}\underline{e}$	N^2+N	N^3+N^2
	$P^{(U)}A^T\underline{e}$+$AP^{(L)}\underline{e}$	N	N^2
backward training	1. $p_{ij}^U(t+1) = p_{ij}^L(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
	$p_{ij}^L(t) + \alpha[d_i - \hat{d}_i]y_j$	2	
	total for N^2 elements	$2N^2$	$2N^3$
	2. $p_{ij}^U(t+1) = \frac{1}{2}[p_{ij}^U(t+1) + p_{ji}^U(t+1)]$, i=1,...,N, j=1,...,N	N^2	N^3
	3. $p_{ij}^L(t+1) = p_{ij}^U(t+1) + \alpha[\sum_{k=1}^N (d_k - \hat{d}_k)a_{ki}]e_j$, i=1,...,N.		
	$\sum_{k=1}^N (d_k - \hat{d}_k)a_{ki}$	N	
	$p_{ij}^U(t+1)+\alpha[\sum_{k=1}^N (d_k-\hat{d}_k)a_{ki}]e_j$	1	
	total for N elements	N^2+N	N^3+N^2
	4. $P_{ij}^L(t+1) = \frac{1}{2}[P_{ij}^L(t+1) + p_{ji}^L(t+1)]$, i=1,..,N, j=1,..,N	N^2	N^3
TOTAL OPERATIONS		$7N^2$+6N	$7N^3$+$6N^2$

[a]Refer to Fig. 5.

Table 14: Computational requirements of Lyapunov equation solving in one iteration of training using structured network and parallel implementation

Operations[a]		computations for one pattern and [number of processors]	computations for N patterns and [number of processors]
compute desired output vector[b]	$\underline{\hat{d}} = -W\underline{e}$	0	0
check stopping criterion	$\|\|\underline{d} - \underline{\hat{d}}\|\|_1$	logN, [N]	NlogN, [N]
forward computation[c]	$\underline{d}=(P^{(U)}A^T + AP^{(L)})\underline{e}$		
	$P^{(U)}A^T\underline{e}$	2logN, [$2N^2$]	2NlogN, [$2N^2$]
	$AP^{(L)}\underline{e}$	2logN, [$2N^2$]	2NlogN, [$2N^2$]
	$P^{(U)}A^T\underline{e}+AP^{(L)}\underline{e}$	logN, [N]	NlogN, [N]
backward training[d,e]	1. $p_{ij}^U(t+1) = p_{ij}^L(t) + \alpha[d_i - \hat{d}_i]y_j$, i=1,...,N, j=1,...,N.		
	$p_{ij}^L(t) + \alpha[d_i - \hat{d}_i]y_j$	3	
	total for N^2 elements	3, [N^2]	3N, [N^2]
	2. $p_{ij}^U(t+1) = \frac{1}{2}[p_{ij}^U(t+1) + p_{ji}^U(t+1)]$, i=1,...,N, j=1,...,N.	1, [N(N-1)/2]	N, [N(N-1)/2]
	3. $p_{ij}^L(t+1) = p_{ij}^U(t+1) + \alpha[\sum_{k=1}^N (d_k - \hat{d}_k)a_{ki}]e_j$, i=1,...,N.		
	$\sum_{k=1}^N (d_k - \hat{d}_k)a_{ki}$	logN, [N]	NlogN, [N]
	$p_{ij}^U(t+1)+\alpha[\sum_{k=1}^N (d_k-\hat{d}_k)a_{ki}]e_j$	3	
	total for N^2 elements	logN+3, [N^3]	NlogN+3N, [N^3]
	4. $P_{ij}^L(t+1) = \frac{1}{2}[P_{ij}^L(t+1) + p_{ji}^L(t+1)]$, i=1,...,N, j=1,...,N.	1, [N(N-1)/2]	N, [N(N-1)/2]
TOTAL OPERATIONS		7logN+8, [N^3+6N^2+N]	7NlogN+8N, [N^3+6N^2+N]

[a]The operations and number of basic processors are illustrated in Fig. 5.
[b]Matrix W is stored in memory and, for simplicity, the access time is ignored.
[c]$P^{(U)}A^T\underline{e}$ and $AP^{(L)}\underline{e}$ can be computed concurrently.
[d]All the elements in $P^{(U)}$ can be updated in parallel.
[e]All the elements in $P^{(L)}$ can be updated in parallel.

Table 15: Summary of speedup factors and number of processors

problem	speedup[a]	number of processors
LU decomposition: A=LU, $A \in R^{N \times N}$.	$\frac{(2.5N^3+3N^2+0.5N)\tau_s}{(4N\log N+6N)\tau_p}$	$0.5N^3+3N^2+0.5N$
SVD: A=URV, $A \in R^{N \times N}$.	$\frac{(14N^3+13N^2)\tau_s}{(9N\log N+17N)\tau_p}$	$3N^3+8N^2+4N$
Linear equations: Ax=b, $A \in R^{m \times n}$.	$\frac{(3mn^2+mn+n^2)\tau_s}{(2n\log n+n\log m+3n)\tau_p}$	$3mn+n$
Similarity transformation: A=PBQ, $A \in R^{N \times N}$.	$\frac{(10N^3+10N^2)\tau_s}{(9N\log N+16N)\tau_p}$	$2N^3+5N^2+3N$
Lyapunov equation: $AP+PA^T=-W$, $A \in R^{N \times N}$, $W \in R^{N \times N}$.	$\frac{(7N^3+6N^2)\tau_s}{(7N\log N+8N)\tau_p}$	N^3+6N^2+N

[a] τ_s is the processing time for one flop in serial processing, and τ_p is the time for one elementary operation in parallel processing.