Computers Math. Applic. Vol. 26, No. 9, pp. 23–34, 1993
Printed in Great Britain. All rights reserved

0898-1221/93 $6.00 + 0.00
Copyright© 1993 Pergamon Press Ltd

Recurrent Neural Networks for Solving Linear Matrix Equations

J. WANG
Department of Industrial Technology, University of North Dakota
Grand Forks, ND 58202-7118, U.S.A.

(Received February 1993; accepted March 1993)

Abstract—Recurrent neural networks for solving linear matrix equations are proposed. The proposed recurrent neural networks consist of two bidirectionally connected layers and each layer consists of an array of neurons. The proposed recurrent neural networks are shown to be asymptotically stable in the large and capable of computing inverse matrices and solving Lyapunov matrix equations. The operating characteristics of the proposed recurrent neural networks are demonstrated via several illustrative examples.

1. INTRODUCTION

Matrix equations have been used for system modeling and design in a variety of application areas such as control, robotics, and signal processing. For many large-scale problems, the orders of the matrices are very large, and large-scale matrices often need to be computed in real time for monitoring and controlling dynamic systems. Sequential algorithms are usually not competent for solving large matrix equations in real time, and parallel methods are more desirable in these applications.

Since Hopfield and Tank's seminal work [1,2], neural networks have been developed for solving numerous optimization and constraint satisfaction problems. As parallel distributed processing models, neural networks are composed of many massively connected simple neurons that can operate concurrently in real time. It is the nature of parallel and distributed processing that offers the neural network approach computational advantages over the existing sequential algorithms in real-time applications.

Recently, neural networks have been proposed for solving a wide variety of matrix algebra problems. For example, structured feedforward neural networks have been developed for solving matrix algebra problems [3]. Nonlinear and linear recurrent neural networks [4–6] have been proposed for matrix inversion. Feedforward and recurrent neural networks have also been proposed for solving systems of linear algebraic equations [7–11] and eigenvalue problems [12]. The results of these investigations have demonstrated the feasibility and potential of neural networks for solving matrix equations.

In this paper, recurrent neural networks for solving linear matrix equations are presented. The proposed recurrent neural networks are proven to be asymptotically stable in the large and capable of solving a variety of linear matrix equations such as inverse matrix and Lyapunov equations.

The rest of this paper is organized in four sections. Section 2 discusses the configuration of a generic recurrent neural network for solving linear matrix equations. Section 3 describes a generic

This work was partially supported by a research grant from the Advancing Science Excellence in North Dakota Program.

recurrent neural network for matrix inversion. Section 4 describes two recurrent neural networks for solving continuous-time and discrete-time Lyapunov equations. Finally, Section 5 concludes this paper with final remarks.

2. GENERIC CONFIGURATION

A linear matrix equation can be generally described as $G(S) = N$ (i.e., $g_{ij}(S) = 0$ for $i = 1, 2, \ldots, p$; $j = 1, 2, \ldots, q$), where $S = [s_{ij}]$ is an $m \times n$ solution matrix, $G = [g_{ij}]$ is a $p \times q$ matrix of linear functions of S, and N is a $p \times q$ null matrix. Since $G(S) = N$ is essentially a set of equality constraints, the problem of solving a matrix equation can be considered as a constraint satisfaction problem. For example, a matrix inversion problem can be viewed as a constraint satisfaction problem such that $AS - I = N$ where A is a given coefficient matrix and I is the identity matrix.

In light of the above consideration, a matrix equation can be formulated as an unconstrained optimization problem as follows:

$$\min_V E[G(V)] = \sum_{i=1}^{p} \sum_{j=1}^{q} e_{ij}[g_{ij}(V)], \tag{1}$$

where $V = [v_{ij}]$ is an $m \times n$ matrix of variables corresponding to solution matrix S, e_{ij} is an objective function that measures the degree of constraint violation of element $g_{ij}(V)$.

The dynamical equation of a generic recurrent neural network for solving linear matrix equations can be described as follows:

$$\frac{dv_{ij}(t)}{dt} = -\mu \sum_{k=1}^{p} \sum_{l=1}^{q} \frac{\partial g_{kl}[V(t)]}{\partial v_{ij}} f_{kl}\{g_{kl}[V(t)]\}; \qquad i = 1, 2, \ldots, m; \quad j = 1, 2, \ldots, n; \tag{2}$$

where $\mu > 0$ is a scaling constant, $V(t) = [v_{ij}(t)]$ is an activation state matrix of the recurrent neural network, and $F[G] = [f_{ij}(g_{ij})]$ is an activation function matrix.

The dynamical equation of the generic recurrent neural network for solving linear matrix equations can be decomposed as follows. For $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$;

$$\frac{dv_{ij}(t)}{dt} = -\mu \left\{ \sum_{k=1}^{p} \sum_{l=1}^{q} \frac{\partial g_{kl}[V(t)]}{\partial v_{ij}} u_{kl}(t) \right\}; \tag{3}$$

$$u_{ij}(t) = f_{ij}\{g_{ij}[V(t)]\}. \tag{4}$$

Note that $e_{ij}(g_{ij})$ and $f_{ij}(g_{ij})$ are functions of g_{ij} only; namely,

$$E[G] = \begin{pmatrix} e_{11}(g_{11}) & e_{12}(g_{12}) & \cdots & e_{1q}(g_{1q}) \\ e_{21}(g_{21}) & e_{22}(g_{22}) & \cdots & e_{2q}(g_{2q}) \\ \vdots & \vdots & \ddots & \vdots \\ e_{p1}(g_{p1}) & e_{p2}(g_{p2}) & \cdots & e_{pq}(g_{pq}) \end{pmatrix},$$

$$F[G] = \begin{pmatrix} f_{11}(g_{11}) & f_{12}(g_{12}) & \cdots & f_{1q}(g_{1q}) \\ f_{21}(g_{21}) & f_{22}(g_{22}) & \cdots & f_{2q}(g_{2q}) \\ \vdots & \vdots & \ddots & \vdots \\ f_{p1}(g_{p1}) & f_{p2}(g_{p2}) & \cdots & f_{pq}(g_{pq}) \end{pmatrix}.$$

The architecture of the recurrent neural network for solving linear matrix equations consists of two bidirectionally connected layers of neurons. Each layer is composed of an array of neurons. Figure 1 illustrates the architecture of the generic recurrent neural network for solving linear matrix equations. The activation state matrices $V(t) = [v_{ij}(t)]$ and $U(t) = [u_{ij}(t)]$ represent an

$m \times n$ array and a $p \times q$ array of the neurons in the output layer and the hidden layer, respectively. The activation state matrix $V(t)$ corresponds to the solution matrix $S = [s_{ij}]$. Although m, n, p, and q are equal in the majority of cases, they can be different in general. Since $g_{kl}[V(t)]$ is a linear function, $\frac{\partial g_{kl}}{\partial v_{ij}}$ is a constant. The connection weight from hidden neuron (k,l) to output neuron (i,j) is defined as $-\mu\frac{\partial g_{kl}}{\partial v_{ij}}$ and the connection weight from output neuron (k,l) to hidden neuron (i,j) is defined by the coefficient of $v_{kl}(t)$ in $g_{ij}[V(t)]$. There is no lateral connection among the neurons in either layer. There is a functional transformation for each neuron in the hidden layer and an integral transformation for each neuron in the output layer. One advantage of this generic recurrent neural network for solving linear matrix equations is that the implicit activation functions of output neurons are linear whereas the activation functions of hidden neurons can be nonlinear and bounded, so that there is no need for considering the ranges of variables.

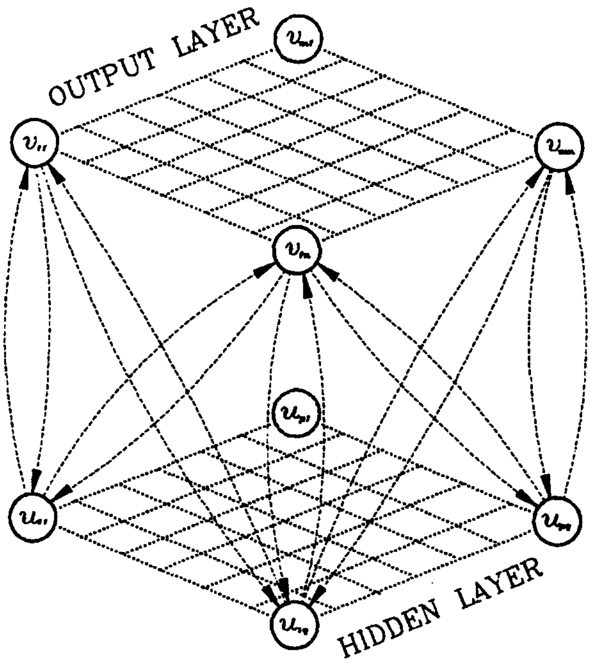


Figure 1. Architecture of the generic recurrent neural network for solving linear matrix equations.

THEOREM. *Assume that there exists $S \in \mathbf{R}^{m\times n}$ such that $G(S) = N$. If $f_{ij}(g_{ij}) = \frac{de_{ij}(g_{ij})}{dg_{ij}}$, $e_{ij}(g_{ij})$ is a convex function that is bounded below, $f_{ij}(0) = 0$ ($i = 1, 2, \ldots, p$; $j = 1, 2, \ldots, q$), and*

$$\operatorname{rank}\begin{pmatrix} \frac{\partial g_{11}}{\partial v_{11}} & \frac{\partial g_{21}}{\partial v_{11}} & \cdots & \frac{\partial g_{pq}}{\partial v_{11}} \\ \frac{\partial g_{11}}{\partial v_{21}} & \frac{\partial g_{21}}{\partial v_{21}} & \cdots & \frac{\partial g_{pq}}{\partial v_{21}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_{11}}{\partial v_{mn}} & \frac{\partial g_{21}}{\partial v_{mn}} & \cdots & \frac{\partial g_{pq}}{\partial v_{mn}} \end{pmatrix} = \min\{pq, mn\}, \tag{5}$$

then the recurrent neural network described in equations (3) and (4) is asymptotically stable in the large and its steady state matrix represents the solution matrix; i.e, $\lim_{t\to\infty} V(t) = S$.

PROOF. Since $E[G(V)] = \sum_{k=1}^{p}\sum_{l=1}^{q} e_{kl}[g_{kl}(V)]$ and $e_{ij}(g_{ij})$ is bounded from below, $E[G(V)]$ is also bounded from below. Since the sum of convex functions is still a convex function, $e_{ij}(g_{ij})$ is a convex function of g_{ij}, and $g_{ij}(V)$ is a linear matrix equation of V, $E[G(V)] = \sum_{i=1}^{p}\sum_{j=1}^{q} e_{ij}[g_{ij}(V)]$ is also a convex function of V. In view of the facts that $G(V)$ is a linear matrix equation and $E[G(V)]$ is convex and bounded below, $E[G(V)]$ is radially unbounded; i.e.,

$E[G(V)] \to \infty$ as $\|V\| \to \infty$. Furthermore, since $f_{kl}(g_{kl}) = \frac{de_{kl}(g_{kl})}{dg_{kl}}$ and equation (3),

$$\begin{aligned}
\frac{dE\{G[V(t)]\}}{dt} &= \sum_{k=1}^{p}\sum_{l=1}^{q} \frac{de_{kl}\{g_{kl}[V(t)]\}}{dt} = \sum_{k=1}^{p}\sum_{l=1}^{q}\sum_{i=1}^{m}\sum_{j=1}^{n} \frac{\partial e_{kl}}{\partial v_{ij}} \frac{dv_{ij}(t)}{dt} \\
&= \sum_{k=1}^{p}\sum_{l=1}^{q}\sum_{i=1}^{m}\sum_{j=1}^{n} \frac{de_{kl}\{g_{kl}[V(t)]\}}{dg_{kl}} \frac{\partial g_{kl}}{\partial v_{ij}} \frac{dv_{ij}(t)}{dt} \\
&= \sum_{k=1}^{p}\sum_{l=1}^{q}\sum_{i=1}^{m}\sum_{j=1}^{n} f_{kl}\{g_{kl}[V(t)]\} \frac{\partial g_{kl}}{\partial v_{ij}} \frac{dv_{ij}(t)}{dt} \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \left[\sum_{k=1}^{p}\sum_{l=1}^{q} \frac{\partial g_{kl}}{\partial v_{ij}} u_{kl}(t)\right] \frac{dv_{ij}(t)}{dt} \\
&= -\frac{1}{\mu}\sum_{i=1}^{m}\sum_{j=1}^{n} \left[\frac{dv_{ij}(t)}{dt}\right]^2 \begin{cases} < 0, & \text{if } \exists i,j,\ \frac{dv_{ij}(t)}{dt} \neq 0; \\ = 0, & \text{if } \forall i,j,\ \frac{dv_{ij}(t)}{dt} = 0. \end{cases}
\end{aligned}$$

Therefore, $E\{G[v(t)]\}$ is a Lyapunov function. According to the systems theory, the generic recurrent neural network for solving linear matrix equations is asymptotically stable in the large; i.e., $\forall V(0)$, $\lim_{t\to\infty} V(t) = \bar{V}$ where $\bar{V}$ is the steady state matrix of $V(t)$. The above derivation and (3) indicate that

$$\frac{dv_{ij}(t)}{dt} = -\mu \frac{\partial E\{G[V(t)]\}}{\partial v_{ij}}.$$

This means that the activation state matrix $V(t)$ of the recurrent neural network evolves in the direction of negative gradient of $E[G(V)]$ as time evolves. In other words, the steady activation state matrix $\bar{V}$ of the recurrent neural network minimizes $E[G(V)]$ in a gradient descent fashion. In view of (3) and (5), $\frac{dv_{ij}(t)}{dt} = 0$ iff $u_{kl}(t) = f_{kl}\{g_{kl}[V(t)]\} = 0$ for all i, j, k, l. Note that e_{ij} is a convex function, if and only if, $\frac{d^2 e_{ij}(g_{ij})}{dg_{ij}^2} = \frac{df_{ij}(g_{ij})}{dg_{ij}} \geq 0$ (i.e., $f_{ij}(g_{ij})$ is nondecreasing with respect to g_{ij}) for $i = 1, 2, \ldots, p$; $j = 1, 2, \ldots, q$. $f_{ij}(0) = 0$ implies that $f_{ij}(g_{ij}) = 0$ iff $g_{ij} = 0$. Therefore, the steady state matrix $G(\bar{V}) = N$ (i.e., $g_{ij}(\bar{V}) = 0$; $i = 1, 2, \ldots, p$; $j = 1, 2, \ldots, q$).

The convex objective function $E[G(V)]$ serves as a norm to evaluate the degree of constraint satisfaction of $G(V)$. It is worth noting that $\arg\min E(G) = 0$ (i.e., $\arg\min e_{ij}(g_{ij}) = 0$; $i = 1, 2, \ldots, p$; $j = 1, 2, \ldots, q$), if and only if, $\frac{de_{ij}(0)}{dg_{ij}} = f_{ij}(0) = 0$. There are many potential choices for an individual component e_{ij}. The selection of $e_{ij}(g_{ij})$, in turn, determines the selection of $f_{ij}(g_{ij})$. Figure 2 illustrates four typical examples of objective functions e_{ij} and their corresponding derivatives (activation functions) f_{ij}. Obviously, f_{ij} in Figures 2a–c are respectively signum function, linear function, and sigmoid function used widely in the literature as the activation functions of neural networks. The signum activation function, however, may cause oscillation in a numerical simulation due to the discontinuity of $f_{ij}(g_{ij})$ at $g_{ij} = 0$. A continuous piecewise linear function will be a better choice than the signum function.

3. MATRIX INVERSION

Matrix inversion deals with computing the inverse matrix A^{-1} of a given $n \times n$ nonsingular matrix A. As indicated in the preceding section, matrix inversion can be considered as a constraint satisfaction problem such that $G(V) = AV - I = N$ where V is the matrix of variables, and I is the identity matrix. Note that $m = n = p = q$ in this case.

Let $V(t)$ be an $n \times n$ activation state matrix of the recurrent neural network corresponding to the inverse matrix A^{-1}. Namely, the activation state $v_{ij}(t)$ of the neuron in the i^{th} row and j^{th} column of $V(t)$ represents the element of the same position in the inverse matrix A^{-1}. The

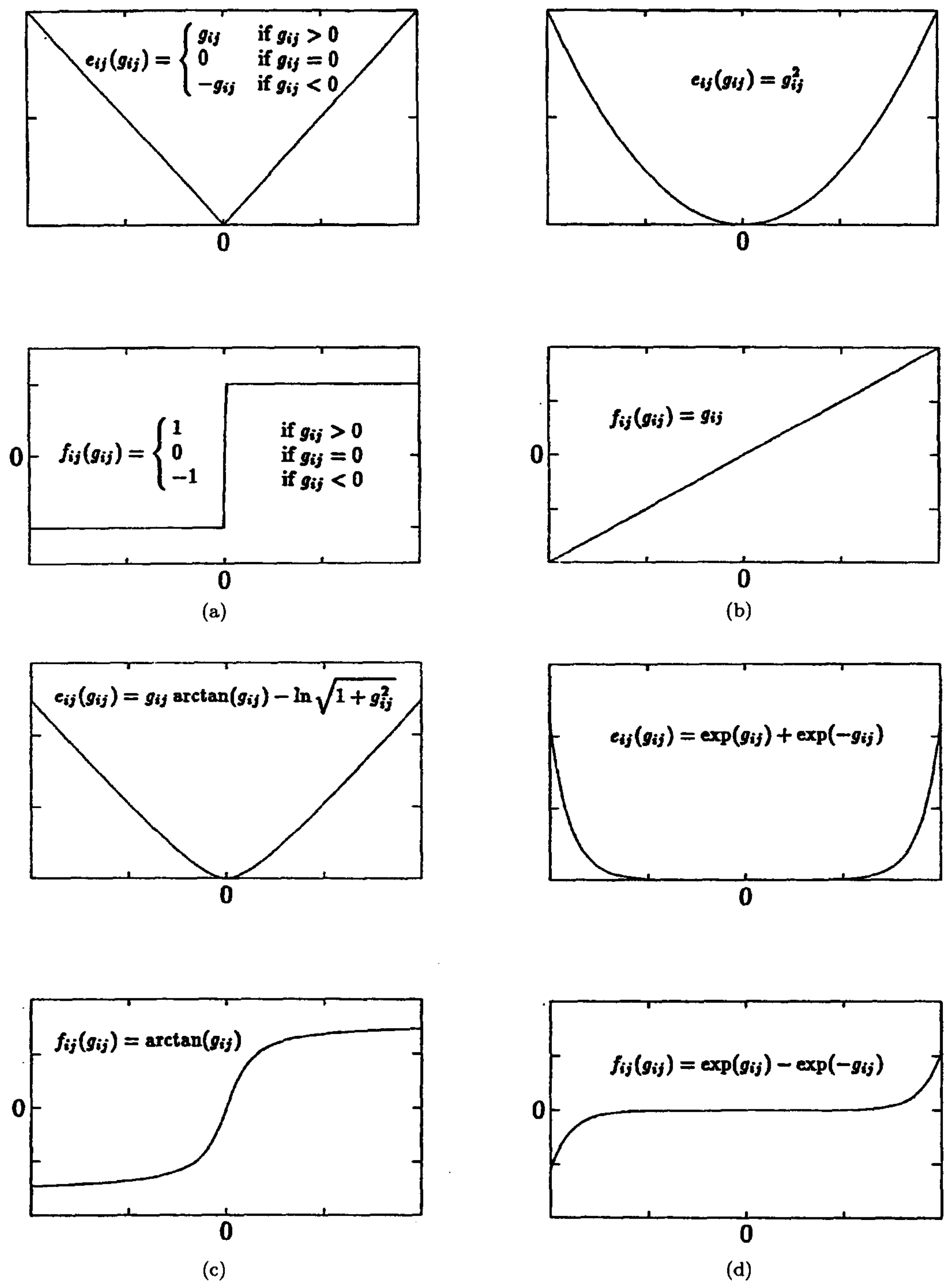


Figure 2. Typical examples of objective functions and corresponding activation functions.

dynamics of the recurrent neural network for matrix inversion can be described by the following matrix-valued differential equation.

$$\frac{dV(t)}{dt} = -\mu A^T F[AV(t) - I], \tag{6}$$

where A^T denotes the transpose of A, $\mu > 0$ is a scalar gain parameter (scaling constant).

Rewritten (6), the dynamical equation of the recurrent neural network can be expressed as follows:

$$\frac{dv_{ij}(t)}{dt} = -\mu \sum_{k=1}^{n} a_{ki} f_{kj} \left[\sum_{l=1}^{n} a_{kl} v_{lj}(t) - \delta_{kj} \right], \qquad i,j = 1,2,\dots,n; \tag{7}$$

where δ_{ij} is the Kronecker delta function defined as $\delta_{ij} = 1$ if $i = j$ or $\delta_{ij} = 0$ otherwise.

Let's define $U(t) = F[AV(t) - I]$ as an $n \times n$ matrix of instrumental variables; i.e., $u_{ij}(t) = f_{ij}\,[\sum_{l=1}^{n} a_{il} v_{lj}(t) - \delta_{ij}]$ for $i,\ j = 1,\ 2, \dots,\ n$. Thus, the dynamical equation of the recurrent neural network can be rewritten as follows:

$$\frac{dV(t)}{dt} = -\mu A^T U(t), \tag{8}$$

$$U(t) = F[AV(t) - I]; \tag{9}$$

or

$$\frac{dv_{ij}(t)}{dt} = -\mu \sum_{k=1}^{n} a_{ki} u_{kj}(t), \qquad i,j = 1,2,\dots,n; \tag{10}$$

$$u_{ij}(t) = f_{ij} \left[\sum_{l=1}^{n} a_{il} v_{lj}(t) - \delta_{ij} \right], \qquad i,j = 1,2,\dots,n. \tag{11}$$

Equations (10) and (11) show that $v_{ij}(t)$ is connected with $u_{1j}(t),\ u_{2j}(t), \dots,\ u_{nj}(t)$ only and $u_{ij}(t)$ is connected with $v_{1j}(t),\ v_{2j}(t), \dots,\ v_{nj}(t)$ only. Similar to the linear recurrent neural network for matrix inversion [4], this pattern of connectivity shows the proposed recurrent neural network can actually be decomposed into n independent subnetworks. Each subnetwork represents one column vector of $V(t)$. Equations (10) and (11) also indicate that the connection weight matrices are identical for each subnetwork. Because of the identical connection weight matrices for every subnetwork, the recurrent neural network can also be realized by a single subnetwork with time-sharing threshold vectors. In each time slot, the subnetwork biased by the corresponding threshold vector generates one column vector of the inverse matrix. Therefore, the spatial complexity of the neural network can be reduced by a factor of n.

The architecture of each subnetwork consists of two layers and each layer in a subnetwork consists of an array of n neurons. The connection weight matrix from the hidden layer to the output layer is defined as $-\mu A^T$ and the connection weight matrix from the output layer to the hidden layer is defined as A. There is no lateral connection among neurons in each layer. The biasing threshold (constant input) matrix in the hidden layer is defined as $-I$ and there are no biases for the neurons in the output layer.

Similar to its predecessor [4], the proposed recurrent neural network can be decomposed into n independent subnetworks and can be easily implemented. Different from its predecessor [4], the proposed recurrent neural network consists of two-layer architecture and its connection weights can be directly obtained from given matrix coefficients.

In matrix inversion, (5) is satisfied if and only if A is nonsingular. According to the theorem above, if $f_{ij}(g_{ij}) = \frac{de_{ij}(g_{ij})}{dg_{ij}}$, $e_{ij}(g_{ij})$ is a convex function, and $\arg\min e_{ij}(g_{ij}) = 0$ $(i,\ j = 1,\ 2, \dots,\ n)$, then the recurrent neural network for matrix inversion is asymptotically stable in the large, and its equilibrium state matrix represents the inverse of a given nonsingular matrix.

EXAMPLE 1. Consider the following nonsingular matrix and its inverse matrix:

$$A = \begin{pmatrix} -2 & -2 & 0 & -1 \\ 0 & 2 & 2 & 1 \\ 1 & -2 & -3 & -2 \\ 0 & 1 & 2 & 1 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} -0.25 & -0.25 & 0.5 & 1 \\ 0 & 1 & 0 & -1 \\ 0.25 & 0.25 & 0.5 & 1 \\ -0.5 & -1.5 & -1 & 0 \end{pmatrix}.$$

Without loss of generality, let $v_{ij}(0) = 0$ for $i, j = 1, 2, 3, 4$. Let $f_{ij}(g_{ij}) = \exp(g_{ij}) - \exp(-g_{ij})$ (Figure 2d). Simulation was performed using a simulator based on Runge-Kutta method. The parameters were set as $\mu = 10^4$ and $\Delta t = 10^{-6}$. The steady-state matrix of the simulated neural network is shown as follows:

$$\bar{V} = \begin{pmatrix} -0.250000 & -0.249997 & 0.500002 & 0.999998 \\ -0.000002 & 0.999993 & -0.000004 & -0.999996 \\ 0.250000 & 0.249999 & 0.500000 & 0.999999 \\ -0.499998 & -1.499989 & -0.999995 & 0.000004 \end{pmatrix}.$$

Figure 3 illustrates the transient states of the simulated recurrent neural network for the illustrative example, where each subplot shows one column of states in the activation state matrix (the circle, square, cross, and star markers indicate the states in the first, second, third, and fourth row, respectively).

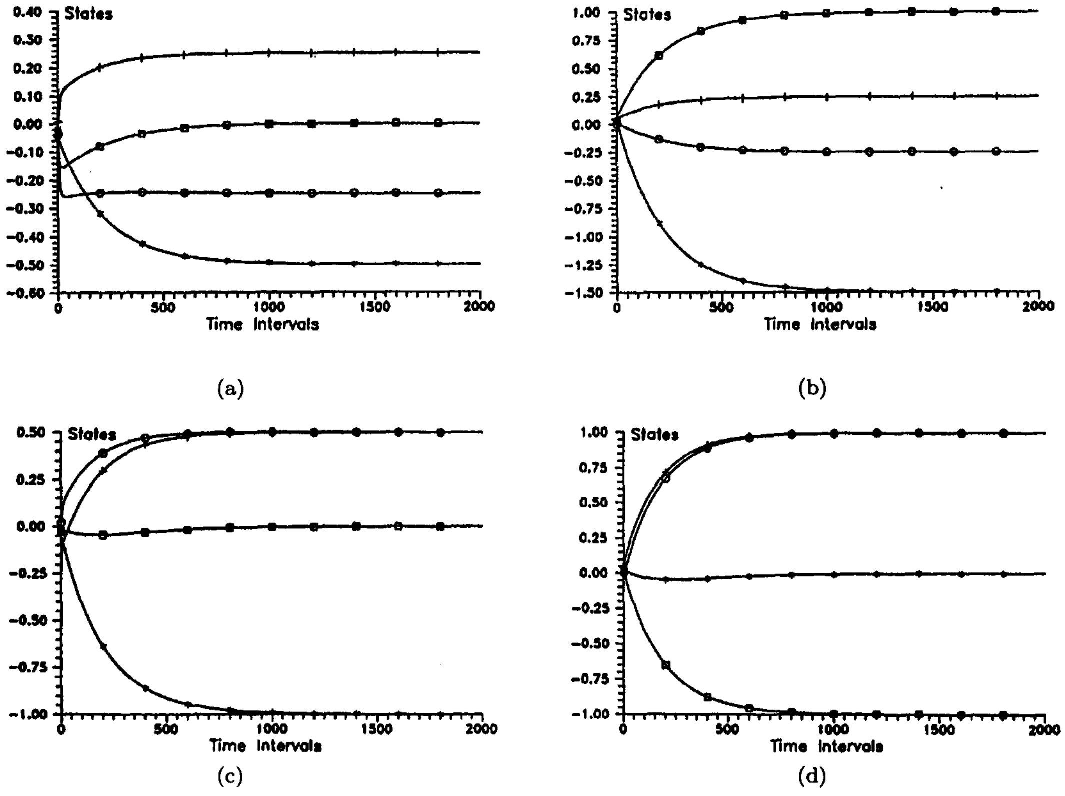


Figure 3. Transient states of the simulated recurrent neural network in Example 1.

4. SOLVING LYAPUNOV EQUATIONS

In dynamical system analysis and design, the Lyapunov's second method is widely used for stability and performance analysis [13–16]. In analyzing a linear dynamic system, a Lyapunov equation is usually needed to be solved. The general Lyapunov matrix equation can be described as:

$$AS + SB + Q = N, \tag{12}$$

where S is an $n \times n$ solution matrix, A, B, and Q are $n \times n$ matrices. The general Lyapunov equation is sometimes called Sylsvester equation. Again, $m = n = p = q$ in this case.

Specifically, there are two types of Lyapunov matrix equations: the continuous-time Lyapunov matrix equation for a continuous-time linear homogeneous system $\dot{x}(t) = Ax(t)$ and the discrete-time Lyapunov equation for a discrete-time linear homogeneous system $x(k+1) = Ax(k)$, where

$x(t)$ and $x(k)$ are n-vectors of state variables for a continuous-time and a discrete-time system, respectively. The continuous-time and discrete-time Lyapunov equations are defined, respectively, as:

$$A^*S + SA + Q = N, \tag{13}$$

$$A^*SA - S + Q = N, \tag{14}$$

where the superscript $*$ denotes the conjugate transpose operator. The solutions of continuous-time and discrete-time Lyapunov equations can be expressed, respectively, as [13]

$$S = \int_0^\infty \exp(A^*t)Q \exp(At)\,dt, \tag{15}$$

$$S = \sum_{k=0}^{\infty} (A^*)^k Q A^k. \tag{16}$$

The necessary and sufficient condition (Lyapunov Theorem) for a linear time-invariant system to be asymptotically stable at the origin of the state space is that the unique solution matrix S for (13) or (14) is Hermitian (symmetric) and positive-definite for any Hermitian (symmetric) positive-definite Q [13,14]. If S is Hermitian (symmetric) positive definite, then $x^T Sx$ is a Lyapunov function. Since Q is Hermitian (symmetric), S is necessarily Hermitian (symmetric) for any A.

There have been continued efforts in analyzing the solutions and developing solution procedures for the Lyapunov matrix equations. Recent research has been focused on deriving the bounds on the determinants, mean size, traces, and eigenvalues of the solutions to the Lyapunov matrix equations (e.g., [17–24]).

In the following, two recurrent neural networks for solving both continuous-time and discrete-time Lyapunov equations are discussed. For simplicity, let's consider real matrices A and Q in this study. The results can be extended to the cases involving complex matrices in a similar way as that discussed in [10].

4.1. Solving Continuous-Time Lyapunov Matrix Equation

The dynamical equation of the recurrent neural network for solving a continuous-time Lyapunov equation can be written as follows:

$$\frac{dV(t)}{dt} = -\mu[AU(t) + U(t)A^T], \tag{17}$$

$$U(t) = F[A^T V(t) + V(t)A + Q], \tag{18}$$

viz., for $i, j = 1, 2, \ldots, n$;

$$\frac{dv_{ij}(t)}{dt} = -\mu \sum_{k=1}^{n} [a_{ik}u_{kj}(t) + a_{jk}u_{ik}(t)]; \tag{19}$$

$$u_{ij}(t) = f_{ij}\left\{\sum_{k=1}^{n} [a_{ki}v_{kj}(t) + a_{kj}v_{ik}(t)] + q_{ij}\right\}. \tag{20}$$

The architecture of the recurrent neural network for solving a continuous-time Lyapunov matrix equation also consists of two bidirectionally connected layers and each layer consists of an $n \times n$ array of neurons. The neuron (i, j) in the output (hidden) layer is connected with the neurons of the i^{th} row and j^{th} column in the hidden (output) layer only. The connection weight from the hidden neurons (i, k) and (k, j) to the output neuron (i, j) are defined as $-\mu a_{jk}$ and $-\mu a_{ik}$, respectively. The connection weight from the output neurons (i, k) and (k, j) to the hidden neuron (i, j) are defined as a_{kj} and a_{ki}, respectively. The biasing threshold (bias) matrix for the neurons in the hidden layers is defined as Q. There are no biases for the neurons in the output layer.

EXAMPLE 2. Consider a continuous-time Lyapunov equation with the following coefficient matrices:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -4 & -6 & -4 \end{pmatrix}, \quad Q = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix}.$$

Without loss of generality, let $v_{ij}(0) = u_{ij}(0) = 0$ for $i, j = 1, 2, 3, 4$. Let $f_{ij}(g_{ij}) = \arctan(g_{ij})$, $\mu = 10^4$, $\Delta t = 10^{-6}$. The steady-state matrix of the simulated neural network and the resultant matrix function $G(\bar{V})$ are shown as follows:

$$\bar{V} = \begin{pmatrix} 6.245950 & 7.994691 & 3.747075 & 0.999330 \\ 7.994691 & 16.740654 & 10.993946 & 2.248642 \\ 3.747075 & 10.993946 & 10.245577 & 1.998981 \\ 0.999330 & 2.248642 & 1.998981 & 0.749721 \end{pmatrix},$$

$$G(\bar{V}) = \begin{pmatrix} 0.000134 & -0.000012 & -0.000270 & 0.000034 \\ -0.000012 & 0.000246 & -0.000047 & -0.000176 \\ -0.000270 & -0.000047 & 0.000120 & -0.000031 \\ 0.000034 & -0.000176 & -0.000031 & 0.000194 \end{pmatrix}.$$

It can be verified that $\bar{V}$ is a positive definite matrix. Figure 4 illustrates the transient states of the simulated recurrent neural network for Example 2, where each subplot shows one column of states in the activation state matrix (the circle, square, cross, and star markers indicate the states in the first, second, third, and fourth row, respectively). Figure 4 shows that the activation state matrix is always symmetric.

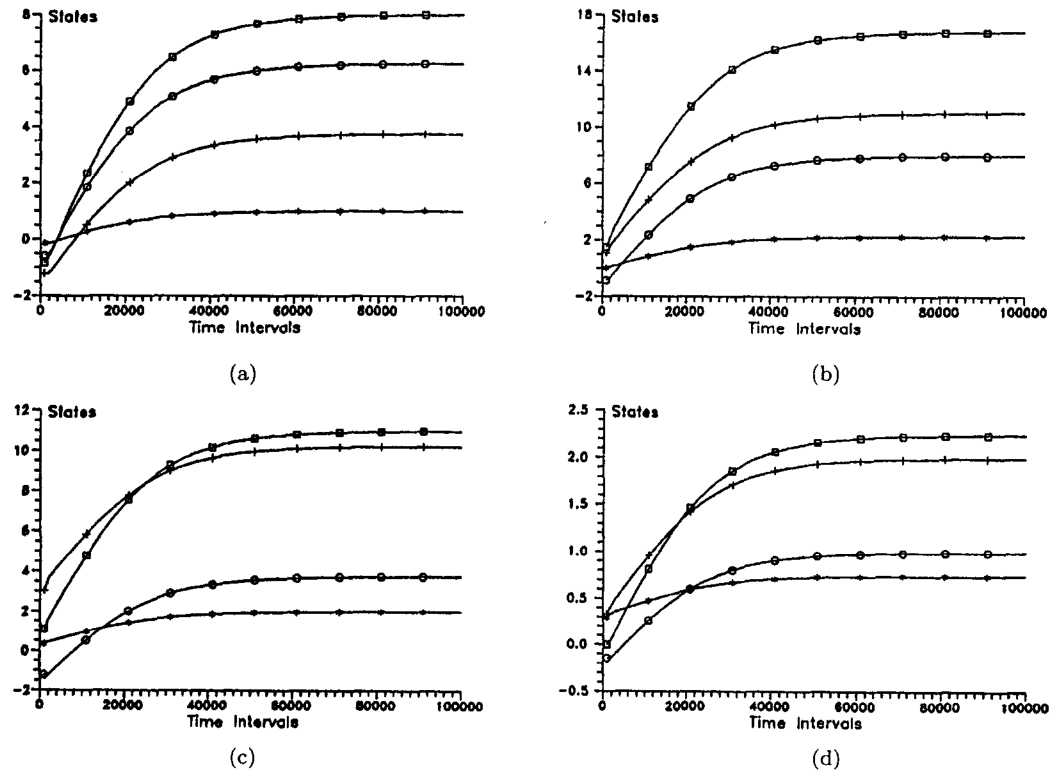


Figure 4. Transient states of the simulated recurrent neural network in Example 2.

4.2. Solving Discrete-Time Lyapunov Matrix Equation

Similar to the recurrent neural network for solving continuous-time Lyapunov equations, the dynamical equation of the recurrent neural network for solving discrete-time Lyapunov matrix equations can be written as follows:

$$\frac{dV(t)}{dt} = -\mu[AU(t)A^T - U(t)], \tag{21}$$

$$U(t) = F[A^T V(t)A - V(t) + Q], \tag{22}$$

viz., for $i, j = 1, 2, \ldots, n$;

$$\frac{dv_{ij}(t)}{dt} = -\mu \left[\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ik}a_{jl}u_{kl}(t) - u_{ij}(t)\right]; \tag{23}$$

$$u_{ij}(t) = f_{ij}\left[\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ki}a_{lj}v_{kl}(t) - v_{ij}(t) + q_{ij}\right]. \tag{24}$$

The architecture of the recurrent neural network for solving discrete-time Lyapunov matrix equations also consists of two layers and each layer consists of an $n \times n$ array of neurons. The neuron (i,j) in the output (hidden) layer is connected with all the neurons in the hidden (output) layer. As indicated in (23) and (24), the connection weight from the hidden neuron (k,l) to the output neuron (i,j) is defined as $-\mu(a_{ik}a_{jl} - \delta_{ik}\delta_{jl})$ and the connection weight from the output neuron (k,l) to the hidden neuron (i,j) is defined as $a_{ki}a_{lj} - \delta_{ik}\delta_{jl}$. Similar to the recurrent neural network for solving continuous-time Lyapunov equations, the biasing threshold (bias) matrix for the neurons in the hidden layer is defined as Q and there are no biases for the neurons in the output layer.

EXAMPLE 3. Consider a discrete-time Lyapunov equation with the following coefficient matrices:

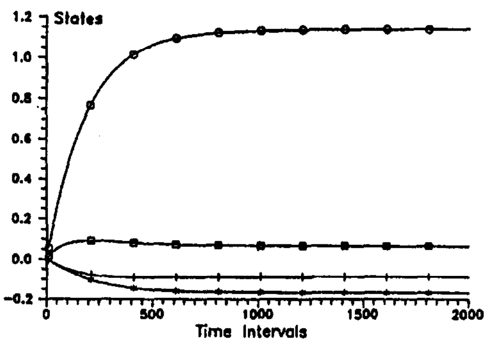
$$A = \begin{pmatrix} 0.1 & 0.2 & 0.3 & -0.4 \\ -0.1 & -0.3 & 0.2 & 0.4 \\ -0.2 & 0.4 & 0.2 & 0.1 \\ 0.2 & 0.3 & -0.1 & -0.1 \end{pmatrix}, \quad Q = I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The eigenvalues of A are $\{0.089347+0.3573i,\ 0.089347-0.3573i,\ -0.5923747,\ 0.3136811\}$. Without loss of generality, let $v_{ij}(0) = u_{ij}(0) = 0$ for $i, j = 1, 2, 3, 4$. Let $f_{ij}(g_{ij}) = \arctan(g_{ij})$, $\mu = 10^4$, $\Delta t = 10^{-6}$. The steady-state matrix of the simulated neural network and the resultant matrix function $G(\bar{V})$ are shown as follows:

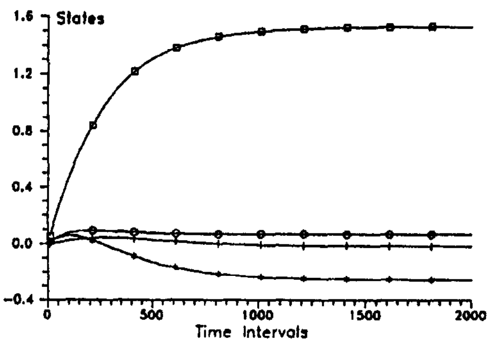
$$\bar{V} = \begin{pmatrix} 1.135817 & 0.065025 & -0.089873 & -0.168179 \\ 0.065025 & 1.531902 & -0.020241 & -0.261497 \\ -0.089873 & -0.020241 & 1.241761 & 0.043311 \\ -0.168179 & -0.261497 & 0.043311 & 1.444992 \end{pmatrix},$$

$$G(\bar{V}) = \begin{pmatrix} 0.000083 & -0.000021 & 0.000013 & 0.000004 \\ -0.000021 & 0.000091 & -0.000010 & -0.000033 \\ 0.000013 & -0.000010 & 0.000076 & 0.000008 \\ 0.000004 & -0.000033 & 0.000008 & 0.000072 \end{pmatrix}.$$

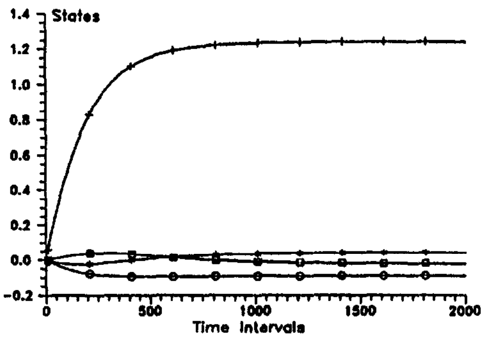
It also can be verified that $\bar{V}$ is a positive definite matrix. Figure 5 illustrates the transient states of the simulated recurrent neural network for Example 3, where each subplot shows one column of states in the activation state matrix (the circle, square, cross, and star markers indicate the states in the first, second, third, and fourth row, respectively). Again, the activation state matrix is always symmetric.



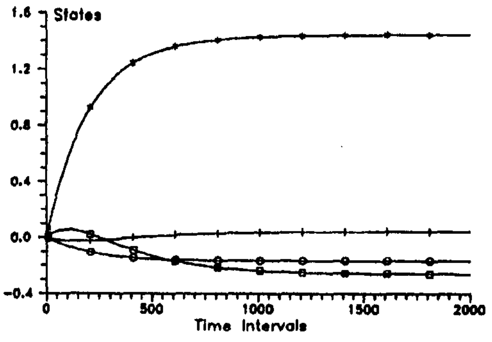
(a)



(b)



(c)



(d)

Figure 5. Transient states of the simulated recurrent neural network in Example 3.

5. CONCLUSIONS

The proposed recurrent neural networks have been proven to be asymptotically stable in the large and capable of solving linear matrix equations. The relation between the Lyapunov function and activation function has also been revealed. Compared with the supervised learning approach, the proposed recurrent neural networks are advantageous since prior training is not necessary. Compared with the other recurrent neural networks, the proposed recurrent neural networks are more general and flexible. Because of the inherently parallel distributed nature in neural computation, the proposed recurrent neural networks can be used for solving large-scale problems in real-time applications.

Further investigations based on the proposed neural networks may be aimed at extension to computing other matrix algebra problems, applications of the proposed recurrent neural networks to specific problems of interest, and design and implementation of the recurrent neural networks in VLSI circuits.

REFERENCES

1. J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics* **52**, 141–152 (1985).
2. D.W. Tank and J.J. Hopfield, Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit, *IEEE Transactions on Circuits and Systems* **33** (5), 533–541 (1986).
3. L.X. Wang and J.M. Mendel, Parallel structured networks for solving a wide variety of matrix algebra problems, *Journal of Parallel and Distributed Computing* **14**, 236–247 (1992).
4. J. Jang, S. Lee and S. Shin, An optimization network for matrix inversion, In *Neural Information Processing Systems*, pp. 397–401, American Institute of Physics, New York, (1988).
5. F.L. Luo and B. Zheng, Neural network approach to computing matrix inversion, *Applied Mathematics and Computation* **47**, 109–120 (1992).
6. J. Wang, A recurrent neural network for real-time matrix inversion, *Applied Mathematics and Computation* (in press, 1993).

7. L.X. Wang and J.M. Mendel, Three-dimensional structured network for matrix equation solving, *IEEE Transactions on Computers* **40**, 1337–1345 (1991).
8. A. Cichocki and R. Unbehauen, Neural networks for solving systems of linear equations and related problems, *IEEE Transactions on Circuits and Systems* **39**, 124–138 (1992).
9. J. Wang, Electronic realization of a recurrent neural network for solving simultaneous linear equations, *Electronics Letters* **28**, 493–495 (1992).
10. J. Wang, Recurrent neural networks for solving systems of complex-valued linear equations, *Electronics Letters* **28**, 1751–1753 (1992).
11. J. Wang and H. Li, Solving simultaneous linear equations using recurrent neural networks, *Information Sciences* (in press).
12. N. Samardzija and R.L. Waterland, A neural network for computing eigenvectors and eigenvalues, *Biological Cybernetics* **65**, 211–214 (1991).
13. C.-T. Chen, *Linear System Theory and Design*, Holt, Rinehart, Winston, New York, (1984).
14. T. Kailath, *Linear Systems*, Prentice-Hall, Englewood Cliffs, NJ, (1980).
15. D.S. Bernstein, Robust stability and performance analysis using Lyapunov equation: LQG controller, *IEEE Transactions on Automatic Control* **34**, 751–758 (1989).
16. Y.K. Foo, Robust stability analysis of family of perturbed matrices using Lyapunov equation approach, *IEEE Transactions on Automatic Control* **35**, 1257–1259 (1990).
17. M. Takehiro, Bound on determinant of Lyapunov matrix equation, *IEEE Transactions on Automatic Control* **26**, 941–942 (1981).
18. M. Takehiro, Bounds for mean size of solution to discrete Lyapunov matrix equation, *IEEE Transactions on Automatic Control* **27**, 462–464 (1982).
19. M. Takehiro, Bounds on trace and determinant of solution of algebraic Riccati and Lyapunov matrix equation, *IEEE Transactions on Automatic Control* **30**, 162–164 (1985).
20. V.R. Karanam, Eigenvalue bounds for algebraic Riccati and Lyapunov equation, *IEEE Transactions on Automatic Control* **27**, 461–463 (1982).
21. S.D. Wang, Trace bounds on solution of algebraic Riccati and Lyapunov equation, *IEEE Transactions on Automatic Control* **31**, 654–656 (1986).
22. M. Takehiro, Eigenvalue bounds for discrete Lyapunov matrix equation, *IEEE Transactions on Automatic Control* **30**, 925–926 (1985).
23. I. Troch, Improved bounds for eigenvalues of Lyapunov equations, *IEEE Transactions on Automatic Control* **32**, 744–747 (1987).
24. N. Komaroff, Upper bound for eigenvalue of solution of Lyapunov matrix equation, *IEEE Transactions on Automatic Control* **35**, 737–739 (1990).