线性矩阵方程的梯度法神经网络求解及其仿真验证*

张雨浓[1]，张禹珩[2]，陈 轲[1]，蔡炳煌[1]，马伟木[1]
（1. 中山大学电子与通信工程系，广东 广州 510275；
2. 广西大学物理科学与工程技术学院，广西 南宁 530004）

摘 要：介绍一种基于梯度法的Hopfield神经网络在线求解线性矩阵方程，并且探讨其MATLAB仿真技术以验证该神经网络在求解线性矩阵方程问题时的准确性和有效性。仿真过程中用以下几种重要技术手段：①Kronecker乘积，用来将描述该神经网络的矩阵微分方程（MDE）转化为向量微分方程（VDE），即标准的给定初始值常微分方程（ODE）；②MATLAB指令“ode45”，用来仿真上述转化后的给定初始值常微分方程；③各种激励函数的编码实现，用以检验该神经网络系统的收敛性和存在实现误差时的鲁棒性。仿真结果同理论分析的对应与一致，进一步证实基于梯度法的Hopfield神经网络在求解固定系数线性矩阵方程中具有很好的效验。

关键词：梯度法；递归神经网络；线性矩阵方程；Kronecker乘积；MATLAB仿真
中图分类号：TP183；O151.21 **文献标识码**：A **文章编号**：0529-6579（2008）03-0026-07

与矩阵运算相关的问题在科学与工程计算中应用得很广泛[1-2]，这类实时计算问题通常也是其它领域同样要解决的问题之一，例如：最优化计算，信号处理，机器人及其控制等[1-5]。自上世纪80年代中期以来，很多研究都把注意力放在矩阵运算的快速算法方面。一般来说，数值算法的最小计算复杂度通常与矩阵维数的立方成正比[4,6]。所以，当这种串行处理算法应用到维数较大的矩阵实时求解问题时就显得效率不高。有鉴于此，我们也曾提出复杂度与矩阵维数平方成正比的数值算法来解决这类矩阵问题[4,7]。然而，结果也同样未尽人意，如求解一个60 000维矩阵逆运算需要约一个小时的时间[7]。因此，许多学者（也包括我们）一直在探讨合适的并行计算方案[1-3,5,8-10]。

动力学系统方法是一种非常重要的并行计算方法，能够有效地解决固定矩阵运算问题[1,3,5,9]，随着对神经网络的深入研究，基于递归神经网络的动力学系统和模拟求解方法已经发展成熟[1-3,5,9-10]。神经动力学方法，因其并行分布计算的特性和简单的硬件/电路可实现性，被认为是在线解决这些固定矩阵运算问题的强有力的替换工具。

1 问题的提出

我们介绍一种梯度法神经网络（即Hopfield神经网络或称递归神经网络）求解线性矩阵方程 $\boldsymbol{AXB}=\boldsymbol{C}$，其中矩阵 $\boldsymbol{A}\in\mathbf{R}^{m\times n}$，$\boldsymbol{B}\in\mathbf{R}^{m\times n}$，$\boldsymbol{C}\in\mathbf{R}^{m\times n}$已知。

1.1 模型推导

下面依据负梯度设计方法推导该神经网络模型：

1）构造一个基于矩阵范数的标量误差函数：

$$\varepsilon(t)=\|\boldsymbol{AX}(t)\boldsymbol{B}-\boldsymbol{C}\|^2/2$$

2）为了使上述误差减小，可使用经典的负梯度方法，因此我们可以得到如下误差函数负梯度方向作为下降方向：

$$-\frac{\partial\varepsilon}{\partial\boldsymbol{X}}=-\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{AXB}-\boldsymbol{C})\boldsymbol{B}^{\mathrm{T}}$$

3）线性的基于负梯度的Hopfield神经网络模型如下：

$$\dot{\boldsymbol{X}}(t)=-\gamma\frac{\partial\varepsilon}{\partial\boldsymbol{X}}=-\gamma\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{AXB}-\boldsymbol{C})\boldsymbol{B}^{\mathrm{T}}\quad(1)$$

其中参数 $\gamma>0$ 决定网络的收敛速度，在电路实现中可由电容系数倒数或电感系数给出（如条件允许，越大越好）；

4）使用非线性激励函数的负梯度神经网络模型如下：

$$\dot{\boldsymbol{X}}(t)=-\gamma\boldsymbol{A}^{\mathrm{T}}\boldsymbol{F}(\boldsymbol{AXB}-\boldsymbol{C})\boldsymbol{B}^{\mathrm{T}}\quad(2)$$

其中初始值 $\boldsymbol{X}(0)=\boldsymbol{X}_0\in\mathbf{R}^{m\times n}$，网络状态 $\boldsymbol{X}(t)\in\mathbf{R}^{m\times n}$ 对应于方程 $\boldsymbol{AXB}=\boldsymbol{C}$ 的理论解 $\boldsymbol{X}^*$。同线性神经网络模型（1）一样，网络参数 $\gamma>0$ 用来调节非线性神经网络模型（2）的收敛率。另外，（2）

* 收稿日期：2007-09-01
基金项目：国家自然科学基金资助项目（60643004、60775050）
作者简介：张雨浓（1973年生），男，教授，博士生导师；E-mail：zhynong@mail.sysu.edu.cn

式中的 $\boldsymbol{F}(\cdot):\mathbf{R}^{m\times n}$ 表示一组由 mn 个激励函数构成的矩阵变换阵列（或称运算阵列、处理阵列）[5,10]。

2　理论分析

通过非线性模型（2）可以看出，考虑不同的激励函数阵列可以得到不同的结论。一般来说，任何单调递增奇函数 $f(\cdot)$ 都可以作为 $\boldsymbol{F}$ 阵列元素从而构建该神经网络。

其中本文讨论四种基本类型的激励函数：

1）线性激活函数 $f(u)=u$；

2）双极 S 形激活函数

$f(u)=(1-\exp(-\xi u))/(1+\exp(-\xi u))$，$\xi\geqslant 2$；

3）幂激活函数，其中且为奇整数；

4）幂形 S 激活函数

$$f(u)=\begin{cases}u^{p} & |u|\geqslant 1,\\ \dfrac{1+\exp(-\xi)-\exp(-\xi u)}{1-\exp(-\xi)+\exp(-\xi u)} & \text{其它}\end{cases}\tag{3}$$

其中 $\xi\geqslant 2$，$\rho\geqslant 3$ 且为奇整数。

其它类型的激励函数也可以由上面四个基本类型函数组合而成。我们下面定性分析使用不同激励函数时的情况[5,10]。

命题 1　给定非奇异矩阵 $\boldsymbol{A}\in\mathbf{R}^{m\times n}$ 和 $\boldsymbol{B}\in\mathbf{R}^{m\times n}$，给定矩阵 $\boldsymbol{C}\in\mathbf{R}^{m\times n}$。任一严格单调递增的奇函数 $f(\cdot)$ 都可以用为 $\boldsymbol{F}$ 阵列元素从而构造神经网络系统（2）。

1）当使用线性激励函数时，神经网络（2）具有全局指数收敛特性，且收敛率与网络参数 γ 和矩阵 $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 的最小特征值得乘积成正比（此处 $\boldsymbol{H}:=\boldsymbol{B}^{\mathrm{T}}\otimes\boldsymbol{A}$）。

2）当使用双极 S 形激励函数时，相对于使用线性激活函数的情况，在误差区域 $[-\delta,\delta]\subset[-1,1]$，神经网络（2）具有更快的收敛特性。这是因为（2）中的误差信号 $e_{ij}=[\boldsymbol{ABC}-\boldsymbol{C}]_{ij}$ 在该区域内被双极形函数放大了。

3）当使用幂激活函数时，相对于使用线性激活函数的情况，在误差区域 $(-\infty,-1]$ 和 $[1,+\infty)$，神经网络（2）具有更快的收敛特性。这是因为（2）中的误差信号 $e_{ij}=[\boldsymbol{ABC}-\boldsymbol{C}]_{ij}$ 在此区域被幂激励函数放大了。

4）鉴于上述第 2 和 3 点，当使用幂形激励函数（3），相对于使用线性激活函数的情况，神经网络（2）在整个区间 $(-\infty,\infty)$ 都具有更快的收敛特性。

在模拟实现或仿真基于梯度神经网络的动力学方程（1）和（2）时，往往假设其处于理想条件下。然而，实际中常有误差存在，例如，线性激活函数不正确的实现结果看起来就像一个 S 形函数或是分段线性函数，这是因为运算放大器有限的增益和频率相关性。对这些可能出现的实现误差，得到下面的理论分析结果。

命题 2　考虑具有如下实现误差的梯度神经网络模型：

$$\dot{\boldsymbol{X}}(t)=-\gamma(\boldsymbol{A}+\Delta\boldsymbol{A})^{\mathrm{T}}\boldsymbol{F}((\boldsymbol{A}+\Delta\boldsymbol{A})\boldsymbol{X}(\boldsymbol{B}+\Delta\boldsymbol{B})-\boldsymbol{C})$$

其中，附加项 $\Delta\boldsymbol{A}(t)$ 和 $\Delta\boldsymbol{B}(t)$ 分别为矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的电路实现误差，其存在满足。$\|\Delta\boldsymbol{A}(t)\|\leqslant\varepsilon_1$ 和 $\|\Delta\boldsymbol{B}(t)\|\leqslant\varepsilon_2$，$\exists\varepsilon_1,\varepsilon_2\geqslant 0,\forall t\geqslant 0$。若最终矩阵 $\hat{\boldsymbol{A}}=\boldsymbol{A}+\Delta\boldsymbol{A}$ 和 $\hat{\boldsymbol{B}}=\boldsymbol{B}+\Delta\boldsymbol{B}$ 仍然是非奇异的，则上述具有实现误差 $\Delta\boldsymbol{A}$ 和 $\Delta\boldsymbol{B}$ 的梯度神经网络模型的稳态误差 $\lim\limits_{t\to\infty}\|\boldsymbol{X}(t)-\boldsymbol{X}^{*}\|$ 总是一致有界的（即 $<+\infty$）。

我们也可以考虑如下受扰动的神经网络模型，其中 $\Delta D(t)$ 为整体模型非精确实现而产生的误差，满足 $\|\Delta D(t)\|\leqslant\varepsilon_3$，$\exists\varepsilon_3\geqslant 0$，$\forall t\geqslant 0$：

$$\dot{\boldsymbol{X}}(t)=-\gamma\boldsymbol{A}^{\mathrm{T}}\boldsymbol{F}(\boldsymbol{AXB}-\boldsymbol{C})\boldsymbol{B}^{\mathrm{T}}+\Delta D\tag{4}$$

命题 3　考虑具有实现误差 $\Delta D(t)$ 的梯度神经网络模型（4）。如果选取的参数 γ 足够大，那么模型（4）的稳态误差 $\lim\limits_{t\to\infty}\|X(t)-X^{*}\|$ 总是一致有界的。而且，当 $\gamma\to+\infty$ 时，该稳态误差 $\lim\limits_{t\to\infty}\|X(t)-X^{*}\|$ 趋于零。

3　MATLAB 仿真

在这一部分，我们介绍相应的仿真技术来检验这种基于梯度法的 Hopfield 神经网络求解线性矩阵方程时的特性。为了仿真（2）式，首先要在 MATLAB 中定义激励函数阵列。

3.1　激励函数阵列的编码

线性激励阵列 $\boldsymbol{F}(\boldsymbol{X})=\boldsymbol{X}\in\mathbf{R}^{m\times n}$ 可以如下简单实现。

function Y = AFMlinear ($\boldsymbol{X}$)

$Y=\boldsymbol{X}$;

缺省参数 $\xi=4$ 的双极 S 形激励函数阵列 $\boldsymbol{F}(\cdot)$ 可通过下面的 MATLAB 代码实现。

function Y = AFMsigmoid($\boldsymbol{X}$, xi)

if nargin = =1, xi =4; end;

$Y=(1-\exp(-xi*\boldsymbol{X}))./(1+\exp(-xi*\boldsymbol{X}))$;

缺省参数 $p=3$ 的幂激励函数阵列 $\boldsymbol{F}(\cdot)$ 可如下实现。

function Y = AFMpower(X,p)

if nargin = =1, p =3; end; Y = X.^p;

公式（3）中定义的缺省参数为 ξ =4 和 p =3 的幂 S 形激励函数阵列可以通过下面的 MATLAB 代码实现。

function Y = AFMpowersigmoid(X, xi, p)

if nargin = =1, xi =4; p =3; elseif nargin = = 2, p =3; end

Y = (1 + exp(−xi))/(1 − exp(−xi)) * (1 − exp(−xi * X)) ./(1 + exp(−xi * X));

i = find(abs(X) > =1);

Y(i) = X(i).^p;

在上面用户自定义的函数体内，MATLAB 会根据用户输入参量个数（nargin）的返回值调用相应函数。通过使用“nargin”，我们能够实现各种不同参数值的激励函数阵列。

3.2 Kronecker 乘积与向量化

由于神经网络模型（2）和（4）都是以矩阵微分方程形式表示的，这样我们就不能直接用 MATLAB 进行仿真。我们通常需要采用 Kronecker 矩阵乘积和向量化技术将该矩阵微分方程表述形式转化为标准的向量微分方程表述形式。

定义 1[1-2,10] 给出矩阵 $A = [a_{ii}] \in \mathbf{R}^{m\times n}$，$B = [b_{ii}] \in \mathbf{R}^{m\times n}$，则矩阵 A 与 B 的 Kronecker 乘积记为 $A\otimes B$，可表示为

$$A \otimes B := \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \cdots & \cdots & \cdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

其中，$A\otimes B \neq B\otimes A$。

定义 2[2, 10] 给出矩阵 $X = [x_{ij}] \in \mathbf{R}^{m\times n}$，我们可以将其拉伸成一列向量，即 vec（$X$）$\in \mathbf{R}^{m\times n}$，其形式为

$$\mathrm{vec}(X) := [x_{11},\cdots,x_{m1},x_{12},\cdots,x_{m2},\cdots,x_{1n},\cdots,x_{mn}]^{\mathrm{T}}$$

另外，给定 A,B,C，若 X 是未知量，利用定义 1（Kronecker 乘积），则我们可得与矩阵方程等价的向量化方程为（$B^{\mathrm{T}}\otimes A$）vec（X）=vec（C）。

基于仿真的目的，使用上面定义的 Kronecker 矩阵乘积和向量化方法，神经网络（2）可以很容易地转化为如下向量微分方程，即给定初始值常微分方程（Initial-value ODE）。

定理 1 神经网络（2）可以描述成如下向量形式：

$$\mathrm{vec}(\dot{X}) = -\gamma(B\otimes A^{\mathrm{T}})F((B^{\mathrm{T}}\otimes A)\mathrm{vec}(X) - \mathrm{vec}(C))。 \quad (5)$$

证明 将神经网络（2）式重复如下：

$$\dot{X}(t) = -\gamma A^{\mathrm{T}}F(AXB - C)B^{\mathrm{T}}$$

利用 Kronecker 矩阵乘积和 vec（·）算子对式（2）进行处理，其中（2）式左端为 vec（$\dot{X}$）而神经网络（2）式的右端向量化为

我们可以注意到第 3.1 小节中激活函数阵列 F（·）的定义和编码是非常灵活的，既可以是从 $\mathbf{R}^{m\times n}$ 到 $\mathbf{R}^{m\times n}$ 的矩阵映射，也可以是拉伸了的从 $\mathbf{R}^{mn\times 1}$ 到 $\mathbf{R}^{mn\times 1}$ 的向量化映射。鉴于此，我们可以继续公式（2）而得到下面的公式：

$$\begin{aligned} &\mathrm{vec}((F(AXB - C)) \\ &= F(\mathrm{vec}(AXB - C)) \\ &= F(\mathrm{vec}(AXB) + \mathrm{vec}(-C)) \\ &= F((B^{\mathrm{T}}\otimes A)\mathrm{vec}(X) - \mathrm{vec}(C)) \end{aligned} \quad (7)$$

将式（6）与式（7）合并，即得到式（2）右端向量化的总形式：

$$\mathrm{vec}(-\gamma A^{\mathrm{T}}F(AXB - C)B^{\mathrm{T}}) = -\gamma(B\otimes A^{\mathrm{T}})F((B^{\mathrm{T}}\otimes A)\mathrm{vec}(X) - \mathrm{vec}(C)) \quad (8)$$

显然，神经网络式（2）两端的向量化形式应该相等，即

$$\mathrm{vec}(\dot{X}) = -\gamma(B\otimes A^{T})F((B\otimes A^{\mathrm{T}})\mathrm{vec}(X) - \mathrm{vec}(C))$$

定理因此得证。

3.3 模型（2）式右端代码实现

上述 Kronecker 乘积可以使用 MATLAB 的“kron”指令实现，例如：$A\otimes B$ 在 MATLAB 中表示为 kron（A,B）。

矩阵向量化可以使用 MATLAB 的“reshape”指令来实现。假设一个 n 行 m 列矩阵 X，使用 reshape（X,n * m,1）命令后产生一个列向量，即 $\mathrm{vec}(X) = [x_{11},\cdots,x_{m1},x_{12},\cdots,x_{m2},\cdots,x_{1n},\cdots,x_{mn}]^{\mathrm{T}}$。通过使用 MATLAB 指令“kron”和“vec”，写出用户自定义函数“GnnRightHandSide”，它返回的是神经网络（2）式［或向量表达式（5）］式右端。其代码实现如下：

function output = GnnRightHandSide(t, x, gamma)

if nargin = =2, gamma =1; end

A = Matrix$A$(); B = Matrix$B$(); C = Matrix$C$();

[m,n] = size(C);

M = kron(B.',A); N = kron(B,A.');

vec$C$ = reshape(C, m * n, 1); % generates the vectorization of C

output = −gamma * N * AFMpowersigmoid(M * x − vec$C$);

4 仿真实例

为了仿真、对比和验证上文给出的分析结论，

我们可以考虑与如下常系数矩阵相关的矩阵方程的神经网络求解：

$$\boldsymbol{A}=\begin{pmatrix}1&0&1\\1&1&0\\1&1&1\end{pmatrix},\boldsymbol{B}=\begin{pmatrix}0&1&1\\1&2&1\\1&1&1\end{pmatrix},\boldsymbol{C}=\begin{pmatrix}1&1&2\\1&1&1\\0&1&0\end{pmatrix},$$

且矩阵 $\boldsymbol{A}$，$\boldsymbol{B}$，$\boldsymbol{C}$ 可以由下面的 MATLAB 代码给出：

```
function A = MatrixA( )
A = [1 0 1;1 1 0;1 1 1];
function B = MatrixB( )
B = [0 1 1;1 2 1;1 1 1];
function C = MatrixC( )
C = [1 1 2;1 1 1;0 1 0];
```

神经网络式（2）因此变成下面的特定形式：

$$\begin{pmatrix}\dot{x}_{11}&\dot{x}_{12}&\dot{x}_{13}\\\dot{x}_{21}&\dot{x}_{22}&\dot{x}_{23}\\\dot{x}_{31}&\dot{x}_{32}&\dot{x}_{33}\end{pmatrix}=-\gamma\begin{pmatrix}1&1&1\\0&1&1\\1&0&1\end{pmatrix}F\left(\begin{pmatrix}1&0&1\\1&1&0\\1&1&1\end{pmatrix}\cdot\right.$$

$$\left.\left(\begin{pmatrix}x_{11}&x_{12}&x_{13}\\x_{21}&x_{22}&x_{23}\\x_{31}&x_{32}&x_{33}\end{pmatrix}\begin{pmatrix}1&0&1\\1&1&0\\1&1&1\end{pmatrix}-\begin{pmatrix}1&0&1\\1&1&0\\1&1&1\end{pmatrix}\right)\begin{pmatrix}1&1&1\\0&1&1\\1&0&1\end{pmatrix}\right)$$

4.1　收敛性仿真

利用第 3 节给出的方法仿真神经网络式（2），我们可以首先定义一个 MATLAB 函数“GnnConvergence”如下。

```
function GnnConvergence(gamma)
if nargin < 1, gamma = 1; end; tspan = [0 10];
m = size(MatrixA,1); n = size(MatrixB,1);
options = odeset( );
for i = 1:8
    x0 = 4 * (rand(m * n,1) - 0.5 * ones(m * n,1));
    [t,x] = ode45(@ GnnRightHandSide,tspan,x0,options,gamma);
    for j = 1:1:m * n
     k = mod(n * (j - 1) + 1,(m * n)) + floor((j - 1)/m);
     subplot(m,n,k);plot(t,x(:,j));hold on
    end
end
```

观察神经网络式（2）的收敛性（基于缺省参数值 $\xi=4$、$p=3$ 的幂 S 形激励函数和使用设计参数 $\gamma=1$），我们可通过 MATLAB 命令 GnnConvergence（1）产生图 1（a）；同理，由 GnnConvergence（10）产生图 1（b）。

为了展示梯度神经网络（2）在求解线性矩阵

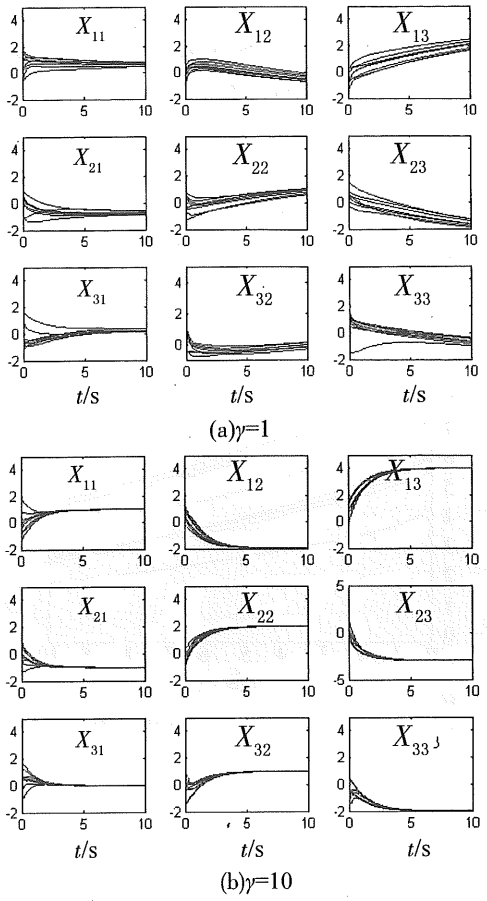

图 1　梯度神经网络（2）求解线性矩阵方程的状态收敛情况

Fig. 1　Convergence performance of gradient – based neural network（2）for solving online linear matrix equation

方程时的收敛性，我们也可以观察求解误差的范数 $\|X(t)-X^*\|$ 的收敛情况。定义函数“NormError”和“GnnNormError”，代码如下。

```
function NormError(x0, gamma)
tspan = [0 10]; options = odeset( );
[t, x] = ode45(@ GnnRightHandSide, tspan, x0, options, gamma);
A = MatrixA; B = MatrixB; C = MatrixC;
P = pinv(A) * C * pinv(B); [m,q] = size(P);
xstar = reshape(P, m * q, 1); x = x';
total = length(t);
for i = 1:total, nerr(i) = norm(x(:,i) - xstar);
end; plot(t,nerr); hold on
function GnnNormError(gamma)
if nargin < 1, gamma = 1; end; total = 8;
for i = 1:total,
  x0 = 4 * (rand(9, 1) - 0.5 * ones(9, 1));
  NormError(x0, gamma);
end
```

分别取不同的γ值，调用“GnnNormError”函数三次，结果见图2。不难发现，在区间上随机选取初始值，求解误差的范数$\|X(t)-X^*\|$都将全局收敛到零值。也即，从任意初始值出发，神经网络式（2）的状态矩阵都将收敛到原线性矩阵方程的理论解X^*。与此同时，设计参数γ越大，神经网络式（2）就收敛得越快。

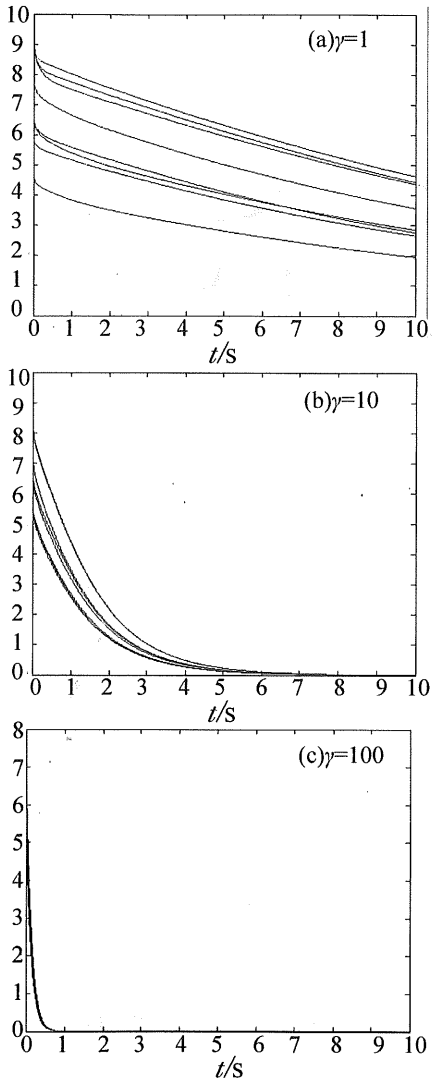

图2 梯度神经网络（2）的求解误差$\|X(t)-X^*\|$的收敛情况

Fig. 2 Computational error $\|X(t)-X^*\|$ of gradient-based neural network (2) with different values of design parameter γ

4.2 鲁棒性仿真

类似于将神经网络式（2）向量化为式（5），带有扰动的梯度神经网络式（4）可以如下向量化：

$$\mathrm{vec}(\dot{X}) = -\gamma(B\otimes A^{\mathrm{T}})F((B^{\mathrm{T}}\otimes A)\mathrm{vec}(X)-\mathrm{vec}(C))+\mathrm{vec}(\Delta D) \quad (8)$$

为了检验这种基于梯度法神经网络的鲁棒性，我们可以考虑如下正弦波形式的模型实现误差：

$$\Delta D = \varepsilon\begin{pmatrix}\cos(3*t) & -\sin(3*t) & 0\\ 0 & 0 & \sin(3*t)\\ 0 & \sin(2*t) & 0\end{pmatrix},$$

其中取$\varepsilon=0.5$。

下面定义的函数“GnnRightHandSideImprecise”可以返回带有扰动变量vec（ΔD）的向量表达式（8）的右端项。

```
function output = GnnRightHandSideImprecise(t, x, gamma)
if nargin = =2, gamma =1; end; e2 =0.5;
deltaD = e2 * [cos(3 * t) -sin(3 * t) 0; 0 0 sin(3 * t); 0 sin(2 * t) 0];
[m,n] = size(deltaD); vecD = reshape(deltaD, m * n, 1);
A = MatrixA; B = MatrixB; C = MatrixC; [m, n] = size(C);
M = kron(B.', A); N = kron(B, A.'); vecC = reshape(C,m * n,1);
output = -gamma * N * AFMpowersigmoid(M * x -vecC) +vecD;
```

基于“GnnRightHandSideImprecise”和下面自定义的MATLAB函数“GnnRobust”，我们调用GnnRobust（gamma）命令，即可生成图3。

```
function GnnRobust(gamma)
tspan = [0 40]; options = odeset();
[m, q] = size(MatrixC);
for i = 1:8
x0 =4 * (rand(m * q, 1) -0.5 * ones(m * q, 1));
[t, x] = ode45(@ GnnRightHandSideImprecise, tspan,x0,options,gamma);
for j = 1:m * q
k = mod(q * (j -1) +1,m * q) +floor((j -1)/m);
subplot(m, q, k); plot(t, x(:,j)); hold on
end
end
```

图3再次证明，设计参数γ的取值对网络收敛速度有很大影响；而且，即使存在扰动和实现误差，我们也可选取一个较大的γ值，一方面使网络快速收敛到理论解，另一方面有效消除实现误差的干扰。这显示该神经网络良好的鲁棒性。

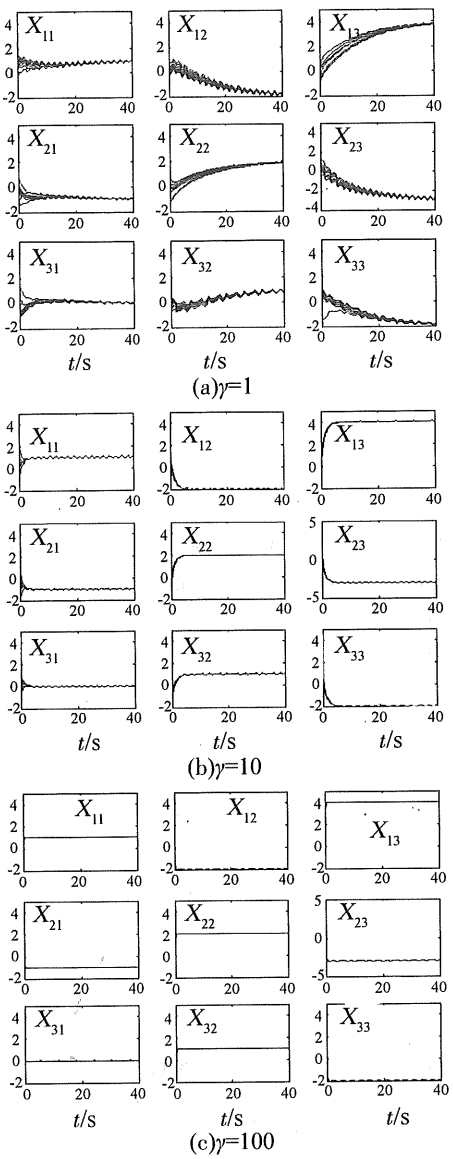

图 3　带扰动的基于梯度法神经网络的线性矩阵方程求解仿真

Fig. 3　Convergence performance of perturbed gradient-based neural network (4) for solving online linear matrix equation

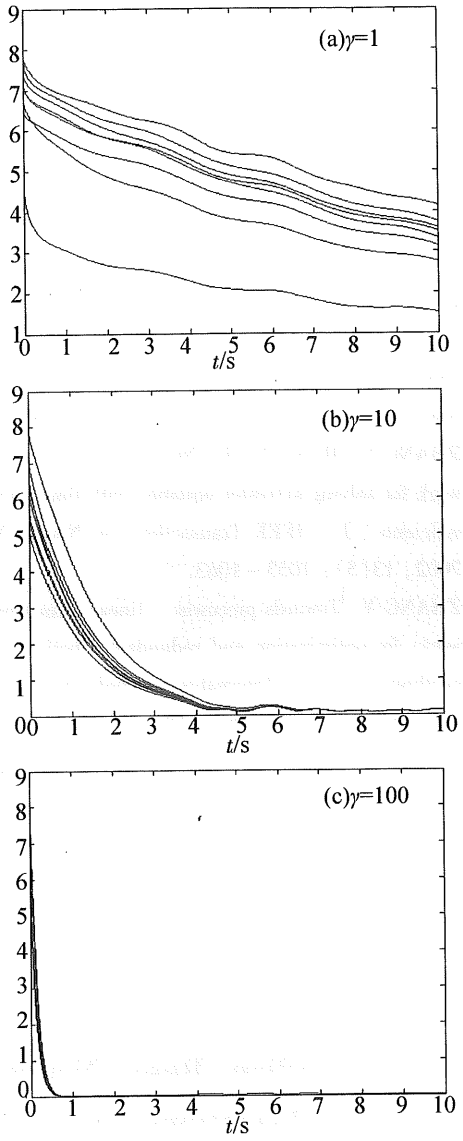

图 4　带扰动梯度神经网络（4）的 $\| X(t) - X^* \|$ 的收敛情况

Fig. 4　Computational error $\| X(t) - X^* \|$ of perturbed gradient - based neural network (4) with different values of design parameter γ

为了展示该带有较大实现误差的梯度神经网络（4）的鲁棒性，我们也可以观察其误差范数 $\| X(t) - X^* \|$ 的收敛情况。这时候，只需要对前面使用 MATLAB 定义的函数“NormError”进行一下修改，将其中的“GnnRightHandSide”改写成如下形式“GnnRightHandSideImprecise”即可。

仿真结果如图 4 所示，我们可以看到即使存在较大的实现误差，神经网络（4）仍然运行良好，并且其求解误差的范数 $\| X(t) - X^* \|$ 总是一致有界的。与此同时，当 γ 从 1 增加到 100 的过程中，收敛时间变快并且稳态时的求解误差下降至一个非常小的数值。在此值得一提的是，同线性激励函数阵列和幂激励函数阵列相比，神经网络（4）使用幂 S 形和 S 形激励函数阵列会有更好的表现：能够在稳态时获得更小的残差。仿真结果再次证实了前面所作的理论分析是正确的。

5　结　论

基于梯度法的 Hopfield 神经网络（1）和（2）提供了一个有效的实时求解线性矩阵方程的并行计算方法。通过采用不同的激励函数阵列和考虑多种实现误差，本文对这种递归神经网络进行了理论研究和 MATLAB 仿真。其中我们采用了一些重要的仿真技术，如：灵活的激励函数阵列的 MATLAB

编码，矩阵的 Kronecker 乘积和向量化技术，及 MATLAB 指令“ode45”的应用。仿真的结果证明了该神经网络求解线性矩阵方程（也即 AXB = C）问题的有效性和高效率特点（即，求解时间与矩阵的维数无关）。

参考文献：

[1] ZHANG Y, WANG J. Global exponential stability of recurrent neural networks for synthesizing linear feedback control systems via pole assignment [J]. IEEE Transactions on Neural Networks, 2002, 13(3): 633 - 644.

[2] ZHANG Y, JIANG D, WANG J. A recurrent neural network for solving sylvester equation with time - varying coefficients [J]. IEEE Transactions on Neural Networks, 2002, 13(5): 1053 - 1063.

[3] ZHANG Y. Towards piecewise - linear primal neural networks for optimization and redundant robotics [C]. Proceedings of IEEE International Conference on Networking, Sensing and Control 2006: 374 - 379.

[4] LEITHEAD W E, ZHANG Y. O(N^2) - operation approximation of covariance matrix inverse in Gaussian Process regression based on quasi - Newton BFGS methods[J]. Communications in Statistics - Simulation and Computation, 2007, 36(2): 367 - 380.

[5] ZHANG Y. Revisit the analog computer and gradient - based neural system for matrix inversion [C]. Proceedings of IEEE International symposium on Intelligent Control, 2005: 1411 - 1416.

[6] NEAGOE V E. Inversion of the Van der Monde matrix [J]. IEEE Signal Processing Letters, 1996, 3(4): 119 - 120.

[7] ZHANG Y, LEITHEAD W E, LEITH D J. Time - series Gaussian process regression based on Toeplitz computation of O(N2) operations and O(N) - level storage [C]. Proceedings of the 44th IEEE Conference on Decision and Control, 2005: 3711 - 3716.

[8] 李晓东,杨国华. 反馈神经网络对非线性时变离散系统的近似 [J]. 中山大学学报:自然科学版, 2002, 41(5): 28 - 30.
LI X, YANG G. Approximation of recurrent neural networks to nonlinear time - variant discrete systems [J]. Acta Scientiarum Naturalium Universitatis Sunyatseni, 2002, 41(5): 28 - 30.

[9] WANG J. A recurrent neural network for real - time matrix inversion [J]. Applied Mathematics and Computation, 1993, 55: 89 - 100.

[10] ZHANG Y, GE S S. Design and analysis of a general recurrent neural network model for time - varying matrix inversion [J]. IEEE Transactions on Neural Networks, 2005, 16(6): 1477 - 1490.

Gradient-Based Neural Network for Solving Linear Matrix Equations and its MATLAB Simulative Verification

ZHANG Yu-nong[1], *ZHANG Yu-heng*[2], *CHEN Ke*[1], *CAI Bing-huang*[1], *MA Wei-mu*[1]

(1. Department of Electronics and Communication Engineering, Sun Yat-sen University, Guangzhou 510275, China;
2. College of Physical Science and Technology, Guangxi University, Nanning 530004, China)

Abstract: A gradient-based Hopfield-type neural network was investigated for the online solution of linear matrix equation. In addition to theoretical analysis of such a neural-network model, several important MATLAB simulation techniques are employed . Kronecker product of matrices is introduced to transform a matrix-form differential equation (MDE) to a vector-form differential equation (VDE); then, a standard ordinary-differential-equation (ODE) is obtained. MATLAB routine “ode45” is introduced to solve the transformed initial-value ODE problem. In addition to various implementation errors, different kinds of activation-function array are coded to show the characteristics of such a gradient-based neural network. Simulation results substantiate the theoretical analysis and efficacy of the proposed gradient-based neural network for solving online the linear matrix equation (LME) problem.

Key words: gradient method; Hopfield neural network; linear matrix equation; Kronecker product; MATLAB simulation