Abstract—A new recurrent neural network (or say, net), i.e., Zhang Neural Network (ZNN), is recently proposed by Zhang et al for online time-varying matrix equations solving. Theoretical analysis, blocks modeling and verification results of Zhang neural network are investigated in this paper, in addition to the neural-solver design method and its comparable gradient neural network (GNN). Towards the final purpose of hardware realization, this paper highlights the model building and convergence illustration of ZNN model in comparison with GNN. The verification results substantiate the feasibility and efficacy of ZNN model for online time-varying linear matrix equations solving.

Index Terms—time-varying linear matrix equations, recurrent neural networks (RNN), gradient-based neural networks, blocks modeling, verification and comparison.

I. INTRODUCTION

The issue of linear matrix equations solving is considered to be one of the fundamental problems widely arising in science and engineering fields. It could be an essential part of many solutions and/or extensions, e.g., generalized inverse solving [1][2], preliminary steps for optimization [3], signal-processing [4], and robotics [5]. In mathematics, the linear matrix equation of our interest in this paper can be depicted as $AXB = C$, where $A \in R^{m \times m}$, $B \in R^{n \times n}$ and $C \in R^{m \times n}$ are coefficient matrices, while $X \in R^{m \times n}$ is the unknown matrix to be found. Evidently, matrix-inverse (or pseudoinverse) problem could be obtained as a sub-topic of such a matrix problem, when coefficient matrices $B = C = I \in R^{m \times m}$.

Recently, due to the in-depth research, neural-dynamic approaches have been widely considered as a powerful alternative to solving online linear matrix equations and related issues [4][6][9]. Compared to numerical algorithms, neural-dynamic methods mainly in the form of recurrent neural networks (or to say, Hopfield-type neural networks) appear to be more suitable for large-scale online application owing to their intrinsically-parallel-distributed processing nature and circuit-implementation convenience [5][10][11][12]. Recently, a special kind of recurrent neural network designed for online solution of time-varying problems has been proposed and established by Zhang et al [13][15]. The resultant Zhang neural networks could converge (globally exponentially) to theoretical solutions of time-varying problems exactly, while gradient-based neural networks may approximate only.

For the purpose of field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) realization, we investigate the blocks modeling and verification of ZNN model for time-varying linear matrix equations solving as well as GNN model for comparison. Simulink [16] is a graphical-based modeling tool which could exploit existing function blocks by using click-and-drag operations to construct mathematical and logical models as well as process flow instead of time-consuming coding. In addition to mathematical analysis, by our understanding, blocks modeling is an important (and key) step towards the final hardware implementation of artificial neural networks with interconnecting structures, which could be viewed as a virtual implementation of a real system satisfying a set of engineering requirements.

The rest of this paper is organized as follows. The mathematical ZNN and GNN models which solve online the time-varying linear matrix equation are presented and analyzed in Section II. Section III shows and synthesizes the blocks modeling techniques so as to develop the above two neural-dynamic solvers. Lastly, Section IV concludes this paper.

II. PROBLEM FORMULATION AND SOLVERS

Let us consider time-varying linear matrix equation below:

$$A(t)X(t)B(t) - C(t) = 0, \quad t \in [0, +\infty),$$

(1)

with smoothly-varying coefficient-matrices $A(t) \in R^{m \times m}$, $B(t) \in R^{n \times n}$ and $C(t) \in R^{m \times n}$. We are to find $X(t) \in R^{m \times n}$ for (1) in an error-free and real-time manner. Note that the solution-uniqueness condition for the above equation (1) is considered throughout the paper. That is, time-varying linear matrix equation (1) is uniquely solvable, if all eigenvalues of coefficient matrices $A(t) \in R^{m \times m}$ and $B(t) \in R^{n \times n}$ are nonzero at any time instant $t \in [0, +\infty)$ [17][18]. It is worth pointing out that the above mentioned problem might be reduced or extended to the generalized-inverse or pseudoinverse problem through the definition presented in Appendix A. For more general representations about matrix equations solving, please refer to [1][7][10][12][15][17][21] and references therein.
Figure 2. Overall model of gradient neural network (3) exploited for online time-varying equation $A(t)X(t)B(t) = C(t)$ solving.

A. Neural-Solvers Description

With the above solution-uniqueness condition satisfied, the solution to time-varying linear matrix equation (1) can be given theoretically as $X^*(t) := A^{-1}(t)C(t)B^{-1}(t)$, which actually results in two matrix-inverse problems and their online solvers [12][15]. From [19], it follows that solving directly for $X^*(t)$ is generally more efficient and accurate than the way of solving firstly for the inverses $A^{-1}(t)$ and $B^{-1}(t)$ and then for $X^*(t)$. Thus the direct solution of time-varying matrix equation (1) is preferred here as well as in this kind of work of ours, with $X^*(t)$ given for comparative and illustrative purposes only.

As mentioned in [17][18], the ZNN model for online solution of time-varying equation (1) could be described as

$$A(t)\dot{X}(t)B(t) = -\dot{A}(t)X(t)B(t) - A(t)X(t)\dot{B}(t) - \gamma F(\frac{\gamma A^2F}{X(t)B(t) - C(t)}) B^T(t),$$  \hspace{1cm} (2) 

where neural-state $X(t) \in R^{m \times n}$, starting from an initial condition $X(0)$, is the activation state-matrix corresponding to the theoretical solution $X^*$ of (1), and design parameter $\gamma$ is used to adjust the convergence rate of the network. 

In addition, $F(\cdot) : R^{m \times n} \rightarrow R^{m \times n}$ denotes a matrix-to-matrix activation function (AF) array (or termed, mapping). The detailed ZNN design-procedure with specifications on such activation function arrays are depicted in Appendix B.

For comparative purpose, we could also develop and exploit the following GNN model for solving online equation (1) (see Appendix C and/or [12][17][18][20] for more details):

$$\dot{X}(t) = -\gamma A^T(t)F(\nabla A(t)X(t)B(t) - C(t)) B^T(t).$$  \hspace{1cm} (3) 

It is worth mentioning that traditional GNN models, e.g. (3), have been designed theoretically for solving the problems with only constant coefficients such as constant $A$, $B$ and $C$ here.
B. Theoretical Results

From [12][13][17][18], the following propositions on global exponential convergence of ZNN (2) could be summarized below for the online time-varying linear matrix equation solving.

**Proposition 1**: Given smoothly time-varying matrices \( A(t) \in \mathbb{R}^{m \times m} \), \( B(t) \in \mathbb{R}^{n \times n} \) and \( C(t) \in \mathbb{R}^{m \times n} \) of (1), if the unique-solution condition is satisfied and a monotonically-increasing odd activation function array \( \mathcal{F}_c(\cdot) \) is used, then the state matrix \( X(t) \) of ZNN model (2) starting from any initial state \( X(0) \in \mathbb{R}^{m \times n} \) always converge to the time-varying theoretical solution \( X^*(t) := A^{-1}(t)C(t)B^{-1}(t) \) of (1).

**Proposition 2**: In addition to Proposition 1, ZNN (2) possesses the following properties. If the linear activation function array is used, then global exponential convergence could be achieved for (2) with rate \( \gamma \) [in terms of error \( E(t) \)]. If power-sigmoid functions (as depicted in Appendix B) are used, then superior convergence could be achieved for (2) on the whole error range \((\infty, +\infty)\), as compared to the linear situation. □

III. BLOCKS MODELING AND PARAMETERS SETTING

While Section II presents the problem formulation, neural solvers and theoretical results of the ZNN model, in this section we investigate the modeling techniques for both ZNN (2) and GNN (3) to solve time-varying linear matrix equation (1).

For illustration, the following specific time-varying matrices \( A(t) \), \( B(t) \) and \( C(t) \) are employed as a typical example:

\[
A(t) = \begin{bmatrix}
2 & -\cos 3t & \sin 3t - 2 \\
\sin 3t + \cos 3t & 2 & 2 - \sin 3t \\
\sin 3t & \cos 3t & 2 - \sin 3t
\end{bmatrix},
\]

\[B(t) = \begin{bmatrix}
\sin 3t & -\cos 3t \\
\cos 3t & \sin 3t
\end{bmatrix},
\]

\[C(t) = [2 + \sin 3t \cos 3t - 2 \cos 3t, -\cos 3t + \sin^2 3t - 2 \sin 3t; \sin 3t + 3 \cos 3t - \sin 3t \cos 3t, 2 + 2 \sin 3t - \sin^2 3t; \sin 3t + 2 \cos 3t - \sin 3t \cos 3t, \cos 3t + 2 \sin 3t - \sin^2 3t].
\]

Besides, the following theoretical solution \( X^*(t) \) to such a matrix equation (1) with the above coefficients is shown here so as to verify the correctness of the neural-network solutions:

\[
X^*(t) = \begin{bmatrix}
\sin 3t & \cos 3t \\
-\cos 3t & \sin 3t
\end{bmatrix}.
\]

ZNN (2) and GNN (3) are then modeled based on Matlab Simulink platform [16], with the overall ZNN and GNN models depicted in Figures 1 and 2, respectively. The detailed considerations are given below in constructing the two models.

A. Generating Time-Varying Coefficients

For generating the three time-varying coefficient-matrices \( A(t) \), \( B(t) \) and \( C(t) \), generally speaking, the MATLAB Function block (with a Clock block as its input) can be exploited. However, as coefficient-matrices \( A(t) \), \( B(t) \) and \( C(t) \) consist of sine and cosine functions only, we could here use the Sine Wave block to generate them.

1) For matrix \( A(t) \) generation, we can add simply \( A_1(t) \) and \( A_2(t) \) as depicted in Figure 3. The parameters are set as

- “Amplitude”: 1
- “Bias”: 0
- “Frequency”: 3
- “Phase”: 0

2) To generate time-varying coefficient matrix \( B(t) \), the parameters could be set as (with no need for signals adding):

- “Amplitude”: 1
- “Bias”: 0
- “Frequency”: 3
- “Phase”: 0

3) Similar to time-varying matrix \( A(t) \), the generation of time-varying matrix \( C(t) \) could be achieved by adding \( C_1(t) \), \( C_2(t) \) and \( C_3(t) \) with the following parameters:

- “Amplitude”:
  - \( C_1(t) \): [0.1; 3; 0; 1 1]
  - \( C_2(t) \): [0.5; 0.5; 1; 2; 2 2]
  - \( C_3(t) \): [2 2; -0.5 0.5; -0.5 0.5]
- “Bias”:
  - \( C_1(t) \): [0; 0; 0; 0 0]
  - \( C_2(t) \): [0.5; 0.5; 0; 0]
  - \( C_3(t) \): [0; 0; 0; 0]
- “Frequency”:
  - \( C_1(t) \): [3 3; 3 3]
  - \( C_2(t) \): [6 6; 3 3; 3 3]
  - \( C_3(t) \): [3 3; 6 6; 6 6]
- “Phase”:
  - \( C_1(t) \): [0; 0; 0; 0 0]
  - \( C_2(t) \): [0; 0; 0; 0 0]
  - \( C_3(t) \): [-0.5 2/pi; 0 0; 0 0]

Besides, please note that the default option “Interpret vector parameters as 1-D” of all these blocks should be deselected.

B. Other Blocks Construction and Setting

In this subsection, more attention is paid to other blocks appearing in Figures 1 and 2 with appropriate parameters.
Four types of activation-function arrays are investigated in this paper, where \( p = 3 \) and \( \zeta = 4 \) are the default parameter-values used in the AF subsystem. 1) For the situation of using linear activation functions, the *purelin* block of the “Neural Network Blockset” could be used. 2) For the situation of using power functions, we could use the *Math Function* block by choosing “pow” in its function list with an input parameter set as \( 3 \). 3) We could similarly construct the subsystems of bipolar-sigmoid and power-sigmoid function arrays, which were mentioned in [21] with more details. To test the convergence performance using different activation functions, by double-clicking the Manual Switch blocks in Figures 1 and 2, we can readily choose the function array and test it. 

To generate different initial states \( X(0) \), we could set the “Initial condition” parameter of the *Integrator* block to be “4*(rand(3,2)-0.5)+ones(3,2)”. The default option “Element-wise” of *Product* blocks has to be changed to “Matrix” so as to perform the standard matrix multiplication. Besides, the option “Save format” of *simout* blocks should be “Array”.

Before running the ZNN and GNN models, we had better pay attention to the following three important points by opening and using the “Configuration Parameters” dialog box: 1) max step size being 0.2; 2) relative tolerance being 1e-7 and absolute tolerance being 1e-7; 3) the check box in front of “States” as of the “Data Import/Export” option being selected. For the purpose of displaying the modeling results more clearly, we could also set the “StopFcn” as follows (which is of “Callbacks” in the “Model Properties” dialog box as started from the “File” pull-down menu).

C. Verification Results

As illustrated in Figures 4 and 5 about the convergence of ZNN (2), the solution generated by the ZNN model could always converge to theoretical solution \( X^*(t) \) exactly. In addition, Figure 4 shows that the convergence becomes much faster if design parameter \( \gamma \) increases. As seen from Figure 5, the residual error (i.e., \(|E(t)|/E|\) synthesized by using power-sigmoid functions vanishes nearly twice faster than that by linear functions under the same condition, e.g., \( \gamma = 1 \). Moreover, other modeling results (omitted here due to space limitation) substantiate as well that superior convergence can be achieved by using power-sigmoid functions under the same \( \gamma \) value, compared to three other types of activation functions. Meanwhile, in Figures 6 and 7, with the same value of \( \gamma \), GNN state-trajectories perform less favorably and the residual error is considerably large (about 1.5). In addition, together with the simulation results in [17][18], our results further show that when the variation frequency of coefficient matrices increases, the GNN model (which is intrinsically designed for stationary problem solving) performs much unfavorably in handling the time-varying problem. On the other hand, ZNN model (2) could still converge well to the time-varying theoretical solution \( X^*(t) \) exactly.

IV. Conclusion

A special kind of recurrent neural network, i.e., Zhang neural network, has been proposed in comparison with gradient-based neural network for online time-varying matrix (or to say, matrix stream) problems solving. The blocks modeling of ZNN and GNN is investigated in this paper and viewed as an important step towards their final circuit-implementation. Modeling results have substantiated the theoretical analysis. Besides, it is our (and possibly researchers’) first time to handle the blocks modeling of such an implicit neural-network system with two mass matrices which are before and after \( \dot{X}(t) \) on the equation’s left-hand side, i.e., \( \dot{A}(t)X(t)B(t) = \cdots \). Our future research directions may lie in the discrete-time algorithms development and/or hardware implementation of the models.

APPENDIX A

Definition 1: Given rectangular matrix \( A \in R^{m \times n} \), if \( X \in R^{n \times m} \) satisfies one or several of the following equations [1][2]:

\[
AXA = A, \quad XAX = X, \quad (AX)^T = AX, \quad (XA)^T = XA,
\]

where \( T \) denotes the transpose of a matrix, then \( X \in R^{n \times m} \) is termed the generalized inverse of \( A \). In addition, pseudoinverse (or to say, Moore-Penrose inverse) could be defined if all of the above four equations are satisfied, which is unique with the minimal Frobenius norm among all generalized inverses.

APPENDIX B

We can design a Zhang neural network solving the time-varying linear matrix equation (1) as follows.

- Step 1. To monitor the solution process, instead of using scalar-valued error functions, we can use the following

\[
\text{figure(1)}
\]

\[
\text{for k=1:6}
\]

\[
\text{j=[1 3 5 2 4 6]; subplot(3,2,j(k)); axis([0 10 -2 2]); plot(tout, xout(:,k),’k’); hold on}
\]

\[
\text{subplot(3,2,1); axis([0 10 -2 2]); plot(tout, sin(3*tout),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X11’); subplot(3,2,2); axis([0 10 -2 2]); plot(tout,sin(3*tout),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X21’); subplot(3,2,3); axis([0 10 -2 2]); plot(tout,cos(3*tout),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X12’); subplot(3,2,4); axis([0 10 -2 2]); plot(tout,sin(3*tout),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X22’); subplot(3,2,5); axis([0 10 -2 2]); plot(tout,xout(:,k),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X31’); subplot(3,2,6); axis([0 10 -2 2]); plot(tout,xout(:,k),’k’); text(5.5,-1.5,’time t (s)’); text(4.5,1.4,’X32’); figure(2)
\]

\[
\text{axis([0 10 0 15]); plot(tout,xout,’k’)}
\]

\[
\text{\text{TABLE I}}
\]

<table>
<thead>
<tr>
<th>Activation Function (AF)</th>
<th>Superior Convergence Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power function</td>
<td>((-\infty,-1] \cup [1,\infty))</td>
</tr>
<tr>
<td>Bipolar sigmoid function</td>
<td>([-0.8,0.8] \cup -0.6 &lt; \gamma &lt; 0.6)</td>
</tr>
<tr>
<td>Power-sigmoid function</td>
<td>((-\infty,\infty))</td>
</tr>
</tbody>
</table>

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on July 3, 2009 at 02:58 from IEEE Xplore. Restrictions apply.
Step 2. The error-function’s time-derivative \( \dot{E}(t) \) should be chosen and forced so as to make every entry of the error function \( E(t) \) converge to zero, if and only if \( X(t) \) equals the theoretical solution \( X^*(t) \) of time-varying linear matrix equation (1).

- Step 2. The error-function’s time-derivative \( \dot{E}(t) \) should be chosen and forced so as to make every entry \( E_{ij}(t) \) of the error function \( E(t) \) converge to zero, \( \forall i \in \{1, 2, \ldots, m\} \) and \( \forall j \in \{1, 2, \ldots, n\} \). Simply put, the design formula of Zhang neural network could be depicted readily as the following:

\[
\frac{dE(t)}{dt} = -\gamma F(E(t)).
\]

where parameter \( \gamma \) and array \( F(\cdot) \) are defined the same as before. Generally speaking, any monotonically-increasing odd activation function (AF) \( f(\cdot) \), being the \( ij \)th processing-element of \( F(\cdot) \), can be used here for constructing the neural network. In this paper, we investigate four different types of activation functions, which are also related to Table I:

1) linear activation function \( f(E_{ij}) = E_{ij} \),

2) power activation function (with odd integer \( p \geq 3 \)) \( f(E_{ij}) = E_{ij}^{p} \),

3) bipolar-sigmoid activation function (with \( \zeta > 2 \))

\[
f(E_{ij}) = \frac{1}{1+\exp(-\zeta E_{ij})} - \frac{1}{1+\exp(+\zeta E_{ij})},
\]

4) power-sigmoid activation function

\[
f(E_{ij}) = \begin{cases} E_{ij}^{p}, & \text{if } |E_{ij}| \geq 1 \\ \frac{1+\exp(-\zeta E_{ij})}{1-\exp(-\zeta E_{ij})} & \text{if } |E_{ij}| < 1 \\ \frac{1-\exp(-\zeta E_{ij})}{1+\exp(-\zeta E_{ij})} & \text{if } |E_{ij}| < 1 \end{cases}
\]

with suitable design parameters \( \zeta > 2 \) and \( p \geq 3 \).

- Step 4. Substituting and expanding \( E(t) \) in the ZNN design formula (4) with the error function defined in Step 1, we could thus have the implicit dynamic equation of ZNN model (2).

**APPENDIX C**

By the following procedure, we can design a gradient neural network solving a stationary matrix equation and then apply it to solving online the corresponding time-varying equation (1).

- Step 1. Let us construct a scalar-valued norm-based energy function, \( \xi = \|AXB - C\|^2_F/2 \), with Frobenius norm \( \|C\|_F := \sqrt{\text{trace}(C^TC)} \). It is noted that the minimum
Besides, Y unong would like to encourage the student-coauthors
and the corresponding authorship of this paper, with his web-

Figure 6. State trajectories of GNN (3) using power-sigmoid functions with
\( \gamma = 1 \), where dotted curves correspond to theoretical solution \( X^*(t) \) of (1).

Figure 7. Residual error \( \|A(t)X(t)B(t) - C(t)\|_F \) as of GNN model (3).

point of the energy function could be achieved if and only if \( X \) is the theoretical solution of \( AXB - C = 0 \).

• Step 2. Design an algorithm to evolve along a descent
direction of this error function \( \xi \) until a minimum is
reached. The typical descent direction of \( \xi \) is the negative
gradient of it, e.g., using the following linear GNN model:

\[
\dot{X} = -\frac{\partial (\|AXB - C\|_F^2/2)}{\partial X} = -A^T(AXB - C)B^T.
\]

• Step 3. By extending the idea of using nonlinear activation
function arrays to the above linear GNN model, we
could then have the generalized nonlinear GNN model
as dynamic equation (3) shows.

ACKNOWLEDGMENT

This work is supported by Program for New Century Excel-
"References"

REFERENCES

the Moore-Penrose generalized inverse,” Proceedings of IEEE Energy
and Information Technologies in the Southeast, vol. 2, pp. 427-431, April
1989.

56, no. 9, pp. 4409-4418, 2008.

[3] W. E. Leithead and Y. Zhang, “\( O(N^2) \)-operation approximation of
covariance matrix inverse in Gaussian process regression based on quasi-
Newton BFGS methods,” Communications in Statistics – Simulation and

SVD and a neural network method for matrix inversion,” IEEE Trans-


converter, signal decision circuit, and a linear programming circuit,”
IEEE Transactions on Circuits and Systems, vol. 33, no. 5, pp. 533-
541, 1986.

inversion,” in Neural Information Processing Systems, D. Z. Anderson,


the inverse and pseudo-inverse of the complex matrix,” Applied Mathe-

for computation of the generalized inverse,” IEEE Transactions on

Wesley, 1989.

[12] Y. Zhang, “Revisit the analog computer and gradient-based neural system
for matrix inversion,” Proceedings of IEEE International Symposium on
Intelligent Control, pp. 1411-1416, June 2005.

Sylvester equation with time-varying coefficients,” IEEE Transactions on

for time-varying matrix inversion,” Proceedings of the 42nd IEEE

network model for time-varying matrix inversion,” IEEE Transactions on

available at http://www.mathworks.com/access/helpdesk/help/toolbox/si-

[17] Y. Zhang and K. Chen, “Comparison on Zhang neural network and gradi-
ent neural network for time-varying linear matrix equation \( AXB = C \)
solving,” Proceedings of IEEE International Conference on Industrial
Technology, pp. 1-6, April 2008.

[18] K. Chen, S. Yue, and Y. Zhang, “MATLAB simulation and comparison
of Zhang neural network and gradient neural network for online solution
of linear time-varying matrix equation \( AXB = C = 0 \),” Lecture Notes

regression based on Toeplitz computation of \( O(N^2) \) operations and
\( O(N) \)-level storage,” Proceedings of the 44th IEEE Conference on

of Wang neural network for solving online linear equations,” Electronics

neural network for online linear time-varying equations solving based
on Matlab Simulink,” Proceedings of the 7th International Conference