Marginally Stable Triangular Recurrent Neural Network Architecture for Time Series Prediction

Seshadri Sivakumar, *Senior Member, IEEE*, and Shyamala Sivakumar, *Member, IEEE*

***Abstract*—This paper introduces a discrete-time recurrent neural network architecture using triangular feedback weight matrices that allows a simplified approach to ensuring network and training stability. The triangular structure of the weight matrices is exploited to readily ensure that the eigenvalues of the feedback weight matrix represented by the block diagonal elements lie on the unit circle in the complex z-plane by updating these weights based on the differential of the angular error variable. Such placement of the eigenvalues together with the extended close interaction between state variables facilitated by the nondiagonal triangular elements, enhances the learning ability of the proposed architecture. Simulation results show that the proposed architecture is highly effective in time-series prediction tasks associated with nonlinear and chaotic dynamic systems with underlying oscillatory modes. This modular architecture with dual upper and lower triangular feedback weight matrices mimics fully recurrent network architectures, while maintaining learning stability with a simplified training process. While training, the block-diagonal weights (hence the eigenvalues) of the dual triangular matrices are constrained to the same values during weight updates aimed at minimizing the possibility of overfitting. The dual triangular architecture also exploits the benefit of parsing the input and selectively applying the parsed inputs to the two subnetworks to facilitate enhanced learning performance.**

***Index Terms*—Chaotic systems, marginally stable recurrent neural network architecture, nonlinear prediction, selective input parsing, upper-lower triangular recurrent neural networks.**

I. Introduction

THE FEEDBACK structure of discrete-time recurrent neural networks (DTRNNs) has proven effective in modeling dynamic characteristics of time varying signals [1]–[7]. A fully connected DTRNN architecture typically consists of a single layer of neurons fully interconnected with each other that allows interactions between all the state variables [1]–[4]. However, it has long been recognized that ensuring learning stability and network stability of fully connected DTRNN is a nontrivial problem that requires in most cases online computation of a stability metric such as the eigenvalues of the state transition matrix at each weight update instance. More importantly, the learning algorithm should include some direct or indirect means of making sure that learning stability is maintained. The computational complexity of monitoring and maintaining stability online is proportional to the square of the number of neurons in the network. This substantially limits the size of fully connected recurrent networks that can be applied for most practical applications.

Echo state networks (ESNs) have been successfully applied to a range of time-series prediction problems. They typically employ a large fixed recurrent weight matrix with full or sparse recurrent connections and update only the output weight matrix during learning [8]–[11] resulting in reduced computational requirement. The fixed feedback weight matrix mitigates the vanishing and exploding gradient problems by not using gradient descent learning [12]–[14]. However, any potential value of training the recurrent weight matrix is not exploited in ESNs.

DTRNNs with sparse and locally recurrent architectures employing gradient descent techniques for training have been shown to perform better and converge faster than fully recurrent networks [5]–[7]. DTRNNs with sparse feedback connections have reduced computational and storage requirements and are also advantageous in terms of monitoring and ensuring learning stability. Presented in [5] is a sparse DTRNN architecture, referred to as the block-diagonal recurrent neural network (BDRNN), in which the feedback connections are restricted to pairs of state variables. The BDRNN has two layers, a feedback layer with a block-diagonal state-transition matrix structure and an output layer that combines the state variables of the feedback layer to generate the network output. During learning, all weight matrices including the feedback weight matrix are updated. It was shown that the learning process in the BDRNN is inherently stable as the block-diagonal structure facilitates seamless maintenance of network stability at each weight update. Examples presented in previous works [5], [15]–[17] demonstrated that the BDRNN architecture can successfully model a weakly nonlinear dynamic system in which the system modes are "decomposable" into several lower order dynamics. These subdynamics can be characterized by the eigenvalues of the block diagonal state-transition weight matrix. The BDRNN structure has been successfully used for a range of applications including speech processing and recognition [5], lung-sound processing [6], and telecom call volume prediction [16].

However, the BDRNN architecture may be limited in its ability to model complex time series with high inherent nonlinearity because the interaction within the state transition

Manuscript received August 29, 2016; revised June 16, 2017 and August 20, 2017; accepted September 6, 2017. This paper was recommended by Associate Editor P. Tino. *(Corresponding author: Seshadri Sivakumar.)*

Seshadri Sivakumar is with Pasumai EnergyTech LLC, Richmond, CA 94804 USA (e-mail: seshadri.sivakumar@pasumaienergytech.com).

Shyamala Sivakumar is with Saint Mary's University, Halifax, NS B3H3C3, Canada (e-mail: shyamala.sivakumar@smu.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2017.2751005

matrix is limited to pairs of state variables. While this lack of interaction is partly compensated by the external interconnections between the state variables facilitated by the output matrix, the closely coupled interaction of the state variables within the state matrix gets muted by the sparseness of the BDRNN structure. On the other hand, a fully connected network can be more effective in modeling plant dynamics with high order nonlinearity than the sparse and locally recurrent counterparts [11].

A key objective addressed in this paper is to expand the BDRNN architecture with increased recurrent interconnections that mimics a fully connected network while retaining the robust stability characteristics of the BDRNN. A second objective is to develop an improved learning algorithm that mitigates the vanishing and exploding gradient problems commonly associated with the gradient-descent training process used in recurrent networks. A third objective is to employ a novel input parsing technique that is amenable with the architecture for enhancing the learning capacity and generalizability of the proposed recurrent architecture.

To enhance interactions between the state variables as possible with fully connected DTRNN while still retaining the advantages of network and learning stability, the block-diagonal matrix structure of the BDRNN is replaced with a triangular state-transition weight matrix structure to exploit the interactions of nondiagonal elements of the triangular matrix. This paper introduces a novel DTRNN architecture, termed upper-lower triangular recurrent neural network (ULTRNN), which combines two subnetworks, one with an upper triangular state-transition weight matrix, and the other with a lower triangular state-transition weight matrix. The upper and the lower triangular weight matrices contain 2×2 diagonal blocks with their eigenvalues constrained while training to lie on the unit circle in the complex z-plane. Constraining the eigenvalues is aimed at mitigating the vanishing and exploding gradient problems [13] associated with online gradient descent-based recurrent learning. In addition, while training each corresponding 2×2 diagonal block in both triangular matrix substructures are constrained to be the same such that the hidden oscillatory modes of the target trajectory are learnt by both subnetworks with a reduced possibility of overfitting. The twin triangular subnetworks in the ULTRNN facilitate the use of input parsing in which the inputs to the two subnetworks can be different from each other and independently derived from the network input which allows selective embedding of some prior information for improved learning performance.

This paper is organized as follows. The architecture of the ULTRNN is developed in detail in Section II-A. Section II-B presents input parsing techniques. Section II-C presents a motivational example. Section III discusses the stability constraints of the ULTRNN architecture, and presents a novel weight update technique that uses the differential of the angular error variable to ensure that the eigenvalues of the triangular weight matrices lie on the unit circle in the complex z-plane. Section IV discusses the learning algorithm for the ULTRNN that is local in space and presents a matrix manipulation means that also makes it local in time. Section V presents representative examples that illustrate the feasibility of the ULTRNN architecture in chaotic time series prediction and autonomous pattern generation problems. Section VI presents conclusions.

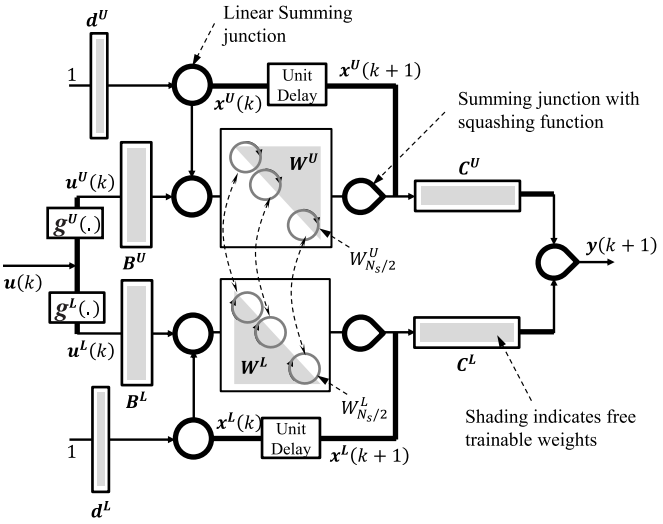

Fig. 1. ULTRNN architecture.

II. Triangular Recurrent Neural Network Architecture

A. Architecture

The proposed ULTRNN architecture that uses twin triangular state-feedback weight matrices is depicted in Fig. 1.

The system equations of the ULTRNN, with k as the sampling instant, is given by

$$\begin{aligned} \boldsymbol{x^U}(k+1) &= \boldsymbol{f}_a\Big(\boldsymbol{W^U x^U}(k) + \boldsymbol{B^U u^U}(k) + \boldsymbol{d^U}\Big) \\ \boldsymbol{x^L}(k+1) &= \boldsymbol{f}_a\Big(\boldsymbol{W^L}\ \boldsymbol{x^L}(k) + \boldsymbol{B^L u^L}(k) + \boldsymbol{d^L}\Big) \\ \boldsymbol{y}(k) &= \boldsymbol{f}_b\Big(\boldsymbol{C^U}\ \boldsymbol{x^U}(k) + \boldsymbol{C^L}\ \boldsymbol{x^L}(k)\Big) \end{aligned} \tag{1}$$

where

$\boldsymbol{W^U}$ the upper triangular state-feedback matrix, given by

$$\begin{bmatrix} w^U_{1,1} & w^U_{1,2} & w^U_{1,3} & w^U_{1,4} & & & w^U_{1,N_s-1} & w^U_{1,N_s} \\ -w^U_{1,2} & w^U_{1,1} & w^U_{2,3} & w^U_{2,4} & \cdots & & w^U_{2,N_s-1} & w^U_{2,N_s} \\ 0 & 0 & w^U_{3,3} & w^U_{3,4} & & & w^U_{3,N_s-1} & w^U_{3,N_s} \\ 0 & 0 & -w^U_{3,4} & w^U_{3,3} & & & w^U_{4,N_s-1} & w^U_{4,N_s} \\ & \vdots & & & \ddots & & \vdots & \\ & \mathbf{0} & & & \cdots & w^U_{N_s-1,N_s-1} & w^U_{N_s-1,N_s} & \\ & & & & & -w^U_{N_s-1,N_s} & w^U_{N_s-1,N_s-1} & \end{bmatrix}$$

$\boldsymbol{W^L}$ is the lower triangular state-feedback matrix, given by

$$\begin{bmatrix} w^L_{1,1} & w^L_{1,2} & 0 & 0 & & & \\ -w^L_{1,2} & w^L_{1,1} & 0 & 0 & \cdots & \mathbf{0} & \\ w^L_{3,1} & w^L_{3,2} & w^L_{3,3} & w^L_{3,4} & & & \\ w^L_{4,1} & w^L_{4,2} & -w^L_{3,4} & w^L_{3,3} & & & \\ & \vdots & & & \ddots & \vdots & \\ w^L_{N_s-1,1} & w^L_{N_s-1,2} & \cdots\cdots\cdots & & \cdots & w^L_{N_s-1,N_s-1} & w^L_{N_s-1,N_s} \\ -w^L_{N_s,1} & w^L_{N_s,2} & & & & -w^L_{N_s-1,N_s} & w^L_{N_s-1,N_s-1} \end{bmatrix}$$

$\boldsymbol{B^U} = \{b^U_{i,j}\}^T$, $\mathbf{B}^L = \{b^L_{i,j}\}^T$, $i = 1, \ldots, N_s$, $j = 1 \ldots N_i$ are the input matrices of the upper and lower triangular networks;

$\boldsymbol{C^U} = \{c_{j,i}^U\}^T$, $\boldsymbol{C^L} = \{c_{j,i}^L\}^T$, $i = 1 \cdots N_s$, $j = 1 \ldots N_o$ are the output matrices of the upper and lower triangular networks;

$\boldsymbol{d^U} = d_i^{\boldsymbol{U}}, i = 1 \ldots N_s$ and $\boldsymbol{d^L} = d_i^{\boldsymbol{L}}, i = 1 \ldots N_s$ are the bias weights;

$\boldsymbol{x^U}(k) = \{x_i^U(k)\}^T$, $\boldsymbol{x^L}(k) = \{x_i^L(k)\}^T$, $i = 1 \ldots N_s$, are the state vectors of the upper and lower triangular sub networks; $\boldsymbol{u}(k) = \{u_i(k)\}^T$, $i = 1 \ldots N_i$, is the network input vector; and $\boldsymbol{u^U}(k) = \boldsymbol{g^U}(\boldsymbol{u}(k)) = \{u_i^U(k)\}^T$, $\boldsymbol{u^L}(k) = \boldsymbol{g^L}(\boldsymbol{u}(k)) = \{u_i^L(k)\}^T$, $i = 1 \ldots N_i$, are the input vectors of the upper and lower triangular sub networks;

$\boldsymbol{y}(k) = \{y_j(k)\}^T$, $j = 1 \ldots N_o$, is the network output vector;

$\boldsymbol{g^U}(\boldsymbol{u}(k))$ and $\boldsymbol{g^L}(\boldsymbol{u}(k))$ are functions that process the network input vector $\boldsymbol{u}(k)$ to facilitate selective parsing and channeling of the input contents to the upper and the lower triangular subnetworks and further discussed in Section III; and $\boldsymbol{f}_\gamma(\boldsymbol{z})$, where $\gamma = a, b$ represents the function on vector $z = \{z_i\}^T$ of the form $\{f_\gamma(z_i)\}^T$, where

$$f_\gamma(x) = (1 - e^{\gamma x})/\left(1 + e^{\gamma x}\right), \ \gamma \geq 0. \tag{2}$$

Note that the nth 2×2 block diagonal submatrices of $\boldsymbol{W^L}$ and $\boldsymbol{W^U}$, denoted by W_n^U and W_n^L, respectively, are constrained to the scaled-orthogonal [5] form

$$W_n^U = \begin{bmatrix} w_{2n-1,2n-1}^U & w_{2n-1,2n}^U \\ -w_{2n-1,2n}^U & w_{2n-1,2n-1}^U \end{bmatrix} \text{for } n = 1, 2, \ldots N_s/2$$
$$W_n^L = \begin{bmatrix} w_{2n-1,2n-1}^L & w_{2n-1,2n}^L \\ -w_{2n-1,2n}^L & w_{2n-1,2n-1}^L \end{bmatrix} \text{for } n = 1, 2, \ldots N_s/2. \tag{3}$$

As discussed in [5] and [17], a key motivation for the use of the triangular architecture with the block diagonal form stems from linear time-invariant systems theory where it is known that a given complex system dynamic can be represented by a linear combination of several first and second order dynamics, each of which can be modeled by a 2×2 feedback weight matrix. Extending this concept to weakly nonlinear dynamic processes, it may be feasible to model such processes with a triangular network such as the ULTRNN with the elements of W_n^U and W_n^L representing the system eigenvalues in the complex z-plane from a linear-system perspective, hence directly facilitating an effective mechanism to model the underlying oscillatory modes.

While the block diagonal elements are constrained as discussed above, the off-block-diagonal elements of the upper and the lower triangular subsystems are free to assume any value in the training process. The off-block-diagonal elements of the two triangular subsystems allow the connections between the various oscillatory modes to interact more flexibly than achievable with only a block-diagonal, an upper triangular, or a lower triangular architecture.

Another key motivation relates to the fact that the complex 2×2 block diagonal substructure is conducive to devising a training algorithm that is local in space. The training algorithm previously developed for the BDRNN [5], [17] can be extended to the case of the ULTRNN with some modification that renders its training process local in time. Hence, just as in the case of the BDRNN, the use of a triangular feedback weight matrix eases the problem of monitoring and maintaining network stability at each weight update.

The block diagonal elements of $\boldsymbol{W^L}$ and $\boldsymbol{W^U}$ are further constrained as given by

$$W_n^U = W_n^L = \begin{bmatrix} \alpha_n & \beta_n \\ -\beta_n & \alpha_n \end{bmatrix}, \ n = 1 \ldots \frac{N_s}{2}$$
$$\sqrt{{\alpha_n}^2 + {\beta_n}^2} = \frac{2}{a}. \tag{4}$$

For the case of the slope of the sigmoidal function $a = 2$, without loss of generality, the constraint equation reduces to

$${\alpha_n}^2 + {\beta_n}^2 = 1. \tag{5}$$

The motivation behind the constraint of equality between W_n^U and W_n^L is to seek out the oscillatory modes present in the time-series through a minimal realization of the neural network by "synchronizing" the two triangular subnetworks during the training process. This constraint is aimed at limiting the proliferation of the number of system eigenvalues and hence to minimize overfitting. This approach has the same objective as presented in [18] where drop out techniques are used to minimize the number of states to avoid overfitting.

Equation (5) can alternatively be reformulated using a set of angular variables θ_n as

$$W_n^U = W_n^L = \begin{bmatrix} \cos(\theta_n) & \sin(\theta_n) \\ -\sin(\theta_n) & \cos(\theta_n) \end{bmatrix}, \ n = 1 \ldots N_s/2. \tag{6}$$

Constraining each complex conjugate pair to lie on the unit circle in the training process inherent through the formulation using the angular variable θ_n eases maintaining the stability of both the network and of learning, without specific need for other means of online monitoring of network stability, or the use of additional feedforward structures as proposed in [17].

B. Input Parsing for ULTRNN Structures

The twin triangular structures of the ULTRNN facilitate the application of inputs $\boldsymbol{u^U}(k)$ and $\boldsymbol{u^L}(k)$ that are derived from $\boldsymbol{u}(k)$ where it may be beneficial in improving the learning performance of the network. For example, the parsing and selective channeling of the input can be implemented using an appropriate selection of functions $\boldsymbol{g^U}(\cdot)$ and $\boldsymbol{g^L}(\cdot)$ to exploit some *a priori* knowledge of the system dynamics being modeled. For a class of problems involving transductive learning [19], it was shown that modeling such learning tasks on spectral graphs in which the labeled observations are positive or negative sources, has been used for encoding prior knowledge about the relationship between individual examples. In [20], for data that can be naturally partitioned into views, incorporating interactions among view models was shown to be advantageous in improving predictive performance and developing a model that provides insight into the underlying relationship among views.

Most real time-series signals are shaped by attributes such as trend, seasonality, periodicity, correlation, skewness, kurtosis, chaos, nonlinearity, and quasi self-similarity. The application of appropriate input signal parsing techniques that take advantage of critical attributes can allow the two triangular subnetworks to learn more effectively. Input parsing can be

implemented as an integral part of data preprocessing but with an extended objective of extracting key competing features and clustering the input in accordance with the extracted feature.

Some representative examples of input signal parsing for a scalar input case $u(k)$ are discussed below.

1) Threshold parsing (7) can be employed when the target waveform exhibits considerable dissymmetry about a threshold u_b. The threshold can either be a constant such as the mean or median, or a variable such as a trending function

$$\begin{aligned} \boldsymbol{g}^{\boldsymbol{U}}(u(k)) &= \begin{cases} u(k), & u(k) > u_b \\ 0, & u(k) \le u_b \end{cases} \\ \boldsymbol{g}^{\boldsymbol{L}}(u(k)) &= \begin{cases} u(k), & u(k) \le u_b \\ 0, & u(k) > u_b. \end{cases} \end{aligned} \tag{7}$$

2) Sum-difference parsing (8) combines adjacent input samples first as a sum and second as a difference such that any harmonic content inherent in the target waveform is highlighted by the difference, while any undesirable high-frequency content or noise is filtered out by the sum. The sum-difference parsing can also represent or be extended using multiple adjacent time series samples to extract the rolling mean and variance profile of the time series waveform. Also, it can help extract and separate the high and low frequency components in a target waveform

$$\begin{aligned} \boldsymbol{g}^{\boldsymbol{U}}(u(k)) &= u(k) + u(k-1) \\ \boldsymbol{g}^{\boldsymbol{L}}(u(k)) &= u(k) - u(k-1). \end{aligned} \tag{8}$$

3) Rectification parsing (9) can be considered in cases where the target waveform exhibits a high degree of skewness of frequency components about a threshold u_b

$$\boldsymbol{g}^{\boldsymbol{U}}(u(k)) = |u(k) - u_b|;\ \boldsymbol{g}^{\boldsymbol{L}}(u(k)) = u(k). \tag{9}$$

4) Envelop parsing (10) can be employed in cases where the target waveform is made up of low frequency envelope function $\varphi(\cdot)$ modulating a high frequency carrier-like waveform

$$\boldsymbol{g}^{\boldsymbol{U}}(u(k)) = \varphi(u(k))u(k);\ \boldsymbol{g}^{\boldsymbol{L}}(u(k)) = \varphi(u(k)). \tag{10}$$

The ULTRNN with its dual subnets provides a unique platform for effectively making use of the parsed inputs for improved learning for cases where the features of the target time series can be broadly decomposed into two competing subsets. In cases where the target time series contains multiple competing features, the learning performance of the ULTRNN can still be enhanced by sorting the features into two competing subsets and selectively channeling the corresponding processed inputs to the two subnets of the ULTRNN. As a scaling alternative, the ULTRNN can be recast as a multiple input system, with each of the multiple inputs fed to the two subnets appropriately parsed to represent a feature set. In such cases, the onus of identifying and modeling the inherent multiple features of the target waveform shifts to the ULTRNN's internal oscillatory modes and their interconnections, and hence, the network dimensions should be sufficiently large for enhancing the learning ability. It should be noted that recasting as a multiple input system to handle target waveforms with multiple features is applicable to any network architecture, and not just unique to the ULTRNN. However, the ULTRNN still has an advantage as its dual subnets provide an additional input parsing mechanism for enhanced learning.

Yet another scaling alternative is to expand the network with several distributed ULTRNNs as subnets with a combined output, with each individual ULTRNN assigned to selectively receive parsed inputs associated with a feature set of the target waveform. While the training process and stability aspects of the expanded network remains the same as that of a single ULTRNN, the training time is compounded by the increased dimensionality of the expanded network.

The input parsing techniques discussed in this paper mostly make use of some prior knowledge of the target time series. When there is no prior knowledge of the target waveform available or apparent, the type of parsing can be chosen by trial and error. A possible methodology can be to test several variants of input parsing on a validation data set and then select the parsing technique that yields the best performance. Feature extraction and time series clustering techniques [21] can conceivably be adapted for the automated assessment and selection of the best candidates, however, such techniques are not considered in this paper.

It should be noted while input parsing and selective channeling can be effective in enhancing the learning performance of the ULTRNN, it is not a necessary requirement for the effective application of the ULTRNN to modeling a wide range of time series prediction problems.

C. Motivational Example

The example in this section compares the performance of the ULTRNN with a fully connected recurrent neural network (FCRNN) and other sparse DTRNN architectures in replicating a simple deterministic nonlinear target waveform. The target waveform, shown in Fig. 2, is generated by a plant with a DTRNN structure with pulsed inputs. For positive input pulses, the plant generates a waveform with a dominant 11th harmonic component, and for negative input pulses, it generates a waveform with a dominant 3rd harmonic component.

The sparse networks considered for comparison include a BDRNN, an UTRNN—a DTRNN with only one triangular feedback weight matrix, a dual BDRNN—a DTRNN with the two feedback matrices made up of only block-diagonal elements the eigenvalues of which are not constrained to be the same. Listed in Table I are the various networks along with their input configurations that were considered for comparison. Cases 1–4 pertain to networks with a single feedback weight matrix, cases 5 and 6 pertain to networks with dual feedback matrices fed with common inputs, and cases 7–9 pertain to networks with dual feedback matrices fed with parsed inputs. With the focus on the ULTRNNs (cases 6 and 8), the dimensions of the BDRNN, the UTRNN, the FCRNN, and the dual BDRNN (cases 1–3, 5, and 7) were chosen so that they have roughly the same number of free trainable weights

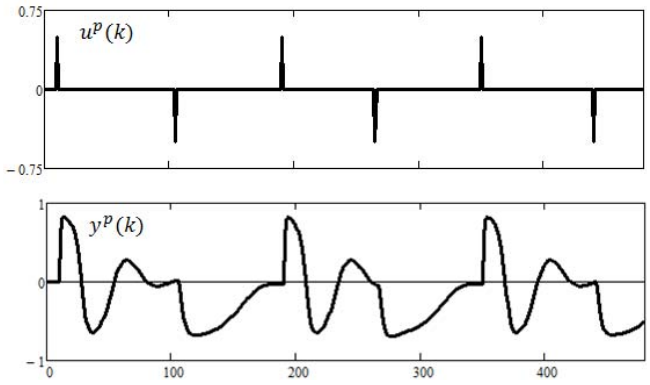

Fig. 2. Nonsymmetric target waveform—(top) input and (bottom) output.

TABLE I
NONSYMMETRIC WAVEFORM MODELING PERFORMANCE COMPARISON OF DTRNNS

Case	Network architecture	Oscillatory modes	Free trainable weights	Test NRMSE
NETWORKS WITH SINGLE FEEDBACK MATRIX				
1	BDRNN	5	40	0.44
2	UTRNN	3	36	0.33
3	FCRNN	3	54	0.397
4	FCRNN	4	88	0.224
NETWORKS WITH DUAL SUBNETS– COMMON INPUT				
5	Dual BDRNN	2 x 3	42	0.484
6	ULTRNN	2 x 2	32	0.256
NETWORKS WITH DUAL SUBNETS – PARSED INPUTS				
7	Dual BDRNN	2 x 2	48	0.305
8	ULTRNN	2 x 2	32	0.167
9	ULTRNN	2 x 3	64	0.163

as that of the ULTRNNs for a balanced performance comparison. Cases 4 and 9 pertain to networks with higher dimensions for extended comparison.

The inputs to the networks of cases 1–4 are the positive and negative pulses applied to the plant. For the common input cases 5 and 6 both the upper and the lower subnets receive both positive and negative input pulses. For the parsed input cases 7–9 the upper subnet receives only positive input pulses, while the lower subnet receives only negative input pulses.

All the networks were trained with 500 points of the target waveform with the inputs derived from the strobe of pulses spaced at random time steps as shown in Fig. 2. The training used the modified backpropagation through time (BPTT) technique described later in Section IV. For all networks other than the FCRNNs, the initial block-diagonal weights were chosen such that the eigenvalues of the feedback weight matrix lie randomly on the unit circle. For the FCRNNs, to ensure the stability of its training process, while the weight elements were chosen randomly, the feedback weight matrix was constrained to have its maximum eigenvalue less than 1 at each weight update step. Each network was trained until convergence and their performance tested for input pulses at arbitrary time steps.

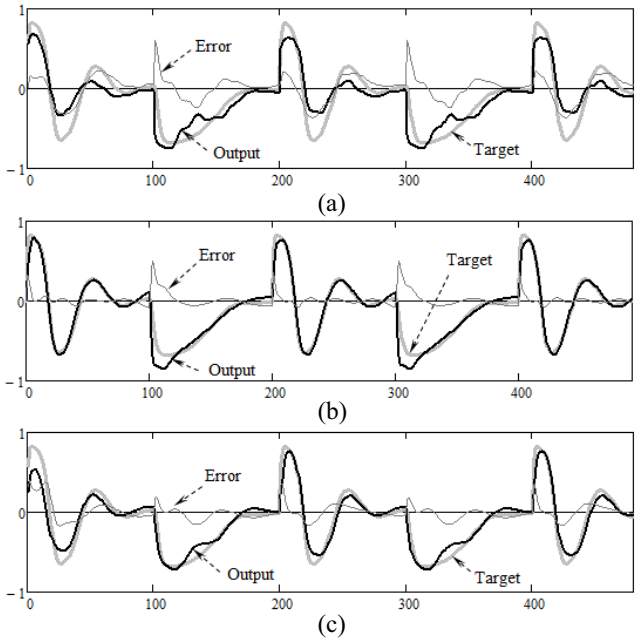

Fig. 3. Nonsymmetric waveform modeling with common input. (a) 5-mode BDRNN (case 1). (b) 4-mode FCRNN (case 3). (c) 2-mode ULTRNN (case 6).

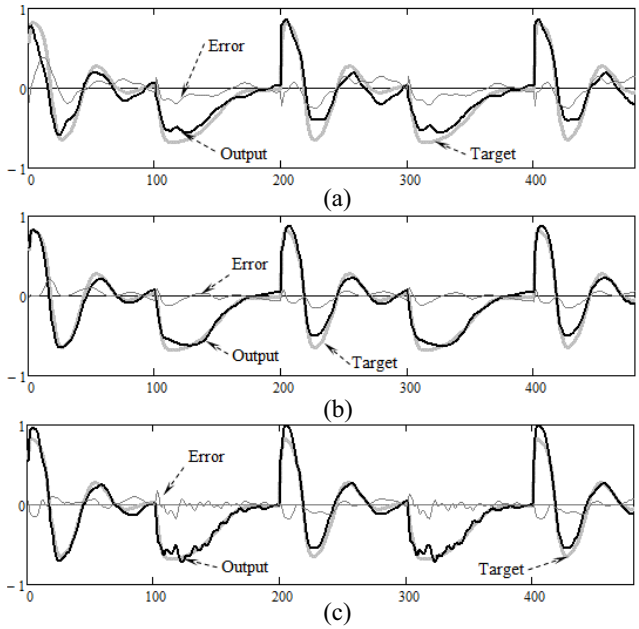

Fig. 4. Nonsymmetric waveform modeling with parsed inputs. (a) 3-mode dual BDRNN (case 7). (b) 3-mode dual ULTRNN (case 8). (c) 3-mode ULTRNN (case 9).

Table I summarizes the test data NRMSE for the various networks considered. Figs. 3 and 4 compare the key network output waveforms with the target waveform for common input and parsed inputs, respectively. From the results in Table I and the plots in Figs. 3 and 4, it is seen as follows.

1) The 2-mode ULTRNN with parsed inputs (case 8), with the fewest trainable parameters, outperforms all other networks including both FCRNNs (cases 3 and 4). From Fig. 4(b), it is seen that the ULTRNN is also better able to differentiate between the varied harmonic content.

2) The higher order (3-mode) ULTRNN (case 9) marginally improves tracking performance, but the onset of overfitting is evident from the plot of Fig. 4(c).
3) The 2-mode ULTRNN with common input [case 6, Fig. 3(c)] outperforms the BDRNN [case 1, Fig. 3(a)], the UTRNN (case 2), and the 3-mode FCRNN [case 3, Fig. 3(b)]. However, the 4-mode FCRNN (case 4) has a lower NRMSE than the ULTRNN with common input, but has more than twice the trainable weights.
4) The 3-mode UTRNN (case 2) performs marginally better than the BDRNN (case 1) due to the additional interconnections between feedback states.

It is also noted that the higher NRMSE of the 3-mode FCRNN (case 3) may be due to the maximum eigenvalue constraint imposed during training; the performance of the BDRNN [case 1, Fig. 3(a)] and the dual BDRNN with common input (case 5) is impaired probably due to reduced interconnections between feedback states; the dual BDRNN with input parsing [case 7 and Fig. 4(a)] has a substantially lower NRMSE.

III. Stability Considerations

The global asymptotic stability conditions for a nonlinear dynamic system can be found by applying the contraction mapping theorem. It is known for a fully connected DTRNN with a feedback weight matrix $\boldsymbol{W^f}$ that the global behavior of its output trajectories for any initial state variable of the system is determined by the square root of each of the eigenvalues of $(\boldsymbol{W^f})^{\boldsymbol{T}}\boldsymbol{W^f}$. The stability of the learning dynamic on the other hand depends on the eigenvalues of $\boldsymbol{W^f}(\boldsymbol{W^f})^{\boldsymbol{T}}$. Hence, the *global* stability of the network dynamic is also a sufficient condition for the learning dynamic as well.

A sufficient condition for the *local* stability of a DTRNN with feedback matrix $\boldsymbol{W^f}$ [1], [22] is

$$\left|\lambda_i(\boldsymbol{W^f})\right| \leq \frac{2}{a} \text{ for } i = 1 \ldots N \tag{11}$$

where $\lambda_i(\boldsymbol{W^f})$ is the ith eigenvalue of $\boldsymbol{W^f}$. A sufficient condition for the *global* stability of the DTRNN using contraction mapping theorem was shown to be [23]

$$\left[\lambda_{\max}\left\{(\boldsymbol{W^f})^{\boldsymbol{T}}\boldsymbol{W^f}\right\}\right]^{1/2} \leq 2/a \tag{12}$$

where $\lambda_{max}\{\cdot\}$ represents the maximum eigenvalue. From (11) and (12), it is clear that the local or global stability of the DTRNN and its training depends on the eigenvalue with the largest magnitude of $\boldsymbol{W^f}$ or $(\boldsymbol{W^f})^{\boldsymbol{T}}\boldsymbol{W^f}$, respectively, at each weight update. Satisfying these conditions at each weight update in a fully connected DTRNN makes this monitoring difficult and computationally expensive. In the case of a sparse network with diagonal (as in the BDRNN) or triangular (as in the ULTRNN) state-transition matrices, as the eigenvalues are determined by the main block diagonal elements, satisfying global and learning stability while training can be achieved relatively easily by continuously monitoring, and imposing appropriate constraints on the main diagonal elements. The global *and* local stability condition for the ULTRNN is given by

$$\left[\lambda_{max}\left((W_n^{\chi})^T W_n^{\chi}\right)\right]^{1/2} \leq \frac{2}{a},\ n = 1 \ldots N_s/2,\ \chi = U, L. \tag{13}$$

With the chosen structure of the block diagonal submatrices W_n^{χ}, this reduces to

$$\left|\lambda(W_n^{\chi})\right|_{max} \leq \frac{2}{a} \leq 1,\ n = 1 \ldots N_s/2,\ \chi = U, L. \tag{14}$$

With the equality constraint per (4) on the block diagonal elements of W_n^U and W_n^L, the sufficient condition for the network and learning stability reduces to

$$\sqrt{{\alpha_n}^2 + {\beta_n}^2} \leq \frac{2}{a} \leq 1,\ n = 1 \ldots \frac{N_s}{2}. \tag{15}$$

Hence, the constraint (5) and the angular formulation (6) inherently facilitate a simple and direct means of ensuring that both the network and the learning are stable.

IV. Learning algorithm

Trajectory learning in the proposed ULTRNN architecture is accomplished using the BPTT algorithm [24] modified as in [5]. The traditional BPTT algorithm computes the exact error gradient in the backward pass using spatially local computations. This requires that the input, state, and error vectors be stored for each time instant in the forward pass, and hence, the storage requirement increases with the length of the training pattern, which makes the BPTT algorithm nonlocal in time. In the modified version reported in [5], the gradient computation can be made local in time by recursively recomputing the state vector in the backward pass, thereby eliminating the storage requirement. However, the modified algorithm requires that the state weight matrix be invertible after each weight update. In addition, the recalculation of the state vector in the backward pass can occasionally result in numerical instability especially for long training sequences. In the ULTRNN, the use of triangular state-transition matrices whose diagonal weights are nonzero at every weight update ensures their invertibility. Additionally, as discussed earlier, constraining the block diagonal elements to lie on the unit circle in the complex plane ensures marginal stability at each weight update. As shown later in Section IV-F, this formulation leads to the easy invertibility of the state-transition matrices with low-computational burden. Additionally, requiring the block diagonal weights to lie on the unit circle also reduces the possibility of numerical instability. Thus, the ULTRNN architecture is conducive to devising a trajectory learning algorithm in which the gradient computation is local in both space and time, provided that the numerical stability is maintained through suitable means. Numerical stability is achieved extending the back-stepping method proposed for the BDRNN in [5], and this is discussed later in Section IV-F.

A. Weight Initialization

The block diagonal elements of $\boldsymbol{W^U}$ and $\boldsymbol{W^L}$ are initialized using random angular variable θ_n, $n = 1, \ldots, N_s/2$,

uniformly distributed in the range $(0{:}2\pi)$ and the corresponding weights computed according to (6). The nondiagonal elements of $\boldsymbol{W^U}$ and $\boldsymbol{W^L}$, the elements of $\boldsymbol{B^U}$, $\boldsymbol{B^L}$, $\boldsymbol{C^U}$, $\boldsymbol{C^L}$, $\boldsymbol{d^U}$, $\boldsymbol{d^L}$, and the state vectors $\boldsymbol{x^U}(0)$ and $\boldsymbol{x^L}(0)$ are initialized with random values uniformly distributed in the range $(-1{:}1)$.

B. Forward Pass

In the forward pass, for any training cycle t, the state vectors $\{x_i^U(k)\}$ for the upper triangular subnetwork and $\{x_i^L(k)\}$ for the lower triangular subnetwork are computed for all time steps k for the qth training sequence of length K_q as

$$\begin{aligned} s_i^\chi(k+1) &= \sum_{j=1}^{N_s} w_{i,j}^\chi\, x_j^\chi(k) + \sum_{j=1}^{N_i} b_{i,j}^\chi\, u_j^\chi(k) + d_i^\chi \\ x_i^\chi(k+1) &= f_a\big(s_i^\chi(k+1)\big),\ \ i = 1\ldots N_s,\ \ \chi = U, L. \end{aligned} \tag{16}$$

The network output $\{y_h(k)\}$ is given by

$$y_h(k) = f_b\left(\sum_{j=1}^{N_s} c_{h,j}^U\, x_j^U(k) + \sum_{j=1}^{N_s} c_{h,j}^L\, x_j^L(k)\right),\ \ h = 1\ldots N_o. \tag{17}$$

The error vector $\{e_h(k)\}$ is computed as

$$e_h(k) = \ y_h^p(k) - y_h(k),\ \ h = 1\ldots N_o \tag{18}$$

where $y_h^p(k)$ is the desired output for the hth output unit at instant k. The total squared error over all the N_o output units for the qth training sequence is given by

$$J_q(k) = \frac{1}{2}\sum_{h=1}^{N_o}\ \big(y_h^p(k) - y_h(k)\big)^2\Big|_q. \tag{19}$$

The computed state and error vectors are stored for use in the backward pass computations.

C. Backward Pass

At any training cycle t, let $\Delta w_{i,j}^U(t)$ and $\Delta w_{i,j}^L(t)$ represent the accumulated error gradients of the nonzero elements of the upper and the lower triangular subnetworks over K_q steps of the qth training sequence. They are computed using the state vectors $\{x_i^U(k)\}$ and $\{x_i^L(k)\}$ and the error vector $\{e_h(k)\}$ (computed and stored in the forward pass) with spatially local computations. Using the chain rule as detailed in [1] and [17]

$$\begin{aligned} \Delta w_{i,j}^\chi(t) &= \frac{\partial J_q(t)}{\partial w_{i,j}^\chi(t)} = \sum_{k=1}^{K_q}\left(\frac{\partial J_q(k,t)}{\partial w_{i,j}^\chi(t)}\right) \\ &= \sum_{k=1}^{K_q}\left(\frac{\partial J_q}{\partial x_i^\chi(k)}\right)\left(\frac{\partial x_i^\chi(k)}{\partial w_{i,j}^\chi}\right),\ \ \chi = U, L. \end{aligned} \tag{20}$$

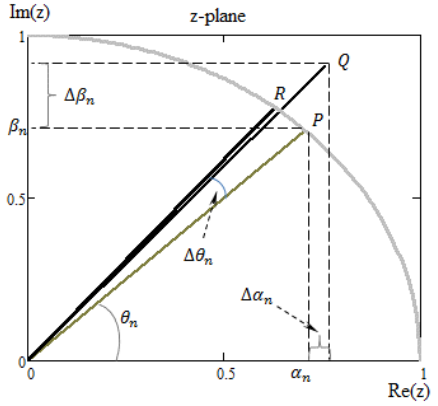

Fig. 5. Computation of the differential of the angular error variable $\Delta\theta_n$.

Denoting $(\partial J_q/\partial x_i^\chi(k))$ as $\varepsilon_i^\chi(k)$, it is computed recursively in the backward pass from the value of $\varepsilon_i^\chi(k+1)$ as

$$\begin{aligned} \varepsilon_i^\chi(k) = &-\sum_{h=1}^{N_o} e_h(k) f_b'\left(\sum_{j=1}^{N}\left(c_{h,j}^U x_j^U(k)\right) + \left(c_{h,j}^L x_j^L(k)\right)\right) c_{h,i}^\chi \\ &+ \sum_{j=1}^{N} w_{j,i}^\chi f_a'\Big(s_j^\chi(k+1)\Big)\varepsilon_j^\chi(k+1),\ \ \chi = U, L \end{aligned} \tag{21}$$

$$\frac{\partial x_i^\chi(k)}{\partial w_{i,j}^\chi} = f_a'\big(s_i^\chi(k)\big)x_j^\chi(k-1),\ \ \chi = U, L \tag{22}$$

where $f_a'(\cdot)$ is the derivative function of $f_a(\cdot)$ and is given by $f_a'(\cdot) = 1 - [f_a(\cdot)]^2$. With the initial value of $\varepsilon_i^\chi(K_q)$

$$\varepsilon_i^\chi\big(K_q\big) = -\sum_{h=1}^{N_o} e_h\big(K_q\big)c_{h,i}^\chi,\ \ \chi = U, L. \tag{23}$$

D. Updating of Block Diagonal Weights

To constrain the block diagonal elements of both the upper and the lower triangular state-transition matrices to stay on the unit circle in the complex z-plane, the differential of the angular variable of each block diagonal submatrix is computed as a combination of the differential of the block diagonal weight elements, as shown in Fig. 5. In this figure, P represents the complex z-plane location of the eigenvalue corresponding to a block diagonal element pair prior to a weight update. Q represents the new weight-update location based only on the differentials computed per (20) with no constraint imposed. R represents the actual updated location after the unit circle constraint is imposed, as per the computational steps outlined below.

From (4) and (6)

$$\tan(\theta_n) = \frac{\beta_n}{\alpha_n} = \frac{w_{2n-1,2n}^\chi}{w_{2n-1,2n-1}^\chi},\ \ \chi = U, L. \tag{24}$$

On each weight update, the differential of the angular variable $\Delta\theta_n$ is related to the differential of the block diagonal weight elements $\Delta\alpha_n$ and $\Delta\beta_n$ as given by

$$\tan(\theta_n + \Delta\theta_n) = \frac{\beta_n + \Delta\beta_n}{\alpha_n + \Delta\alpha_n}. \tag{25}$$

For small values of $\Delta\theta_n$, this reduces to

$$\frac{\sin(\theta_n) \ + \cos(\theta_n) \ \Delta\theta_n}{\cos(\theta_n) \ - \sin(\theta_n)\Delta\theta_n} = \frac{\beta_n + \Delta\beta_n}{\alpha_n + \Delta\alpha_n}. \tag{26}$$

Rearranging the terms, and simplifying

$$\Delta\theta_n = -\Delta\alpha_n \sin(\theta_n) \ + \Delta\beta_n \cos(\theta_n). \tag{27}$$

In terms of the individual differentials of the block diagonal weights of the upper and the lower triangular subnetworks, the differential of the angular variable can be written as

$$\begin{aligned}\Delta\theta_n = &-\sin\theta_n \left\{ \frac{1}{4}\big(\Delta w^U_{2n,2n} + \Delta w^U_{2n+1,2n+1} \right. \\ &\left. + \ \Delta w^L_{2n,2n} + \Delta w^L_{2n+1,2n+1}\big) \right\} \\ &+ \cos\theta_n \left\{ \frac{1}{4}\big(\Delta w^U_{2n,2n+1} - \Delta w^U_{2n+1,2n} \right. \\ &\left. + \ \Delta w^L_{2n,2n+1} - \Delta w^L_{2n+1,2n}\big) \right\}. \end{aligned} \tag{28}$$

Note that the two terms within { } in (28) represent the average differentials of pertinent block diagonal weights. Unlike in the conventional DTRNN where weight updates may very often result in the updated weight matrix becoming unstable, or move the eigenvalues to a location within the unit circle in the z-plane, using the differential of the angular variable for updating the weights (in place of the direct differentials of the block diagonal weight elements) forces the eigenvalues to be retained on the unit circle in the z-plane. This helps to retain network and learning stability, while ensuring that the sensitivity of the ULTRNN is sustained through the training process to help capture the plant's oscillatory modes. At training cycle $t+1$, θ_n is updated as

$$\theta_n(t+1) = \theta_n(t) - \frac{\mu}{K_q}\Delta\theta_n(t) \tag{29}$$

where μ is the learning rate.

E. Updating Nonblock Diagonal, Input, Output, Bias Weights

1) Updating Nonblock Diagonal Weights: The nonblock diagonal nonzero weights of the ULTRNN are updated as

$$w^\chi_{i,j}(t+1) = \ w^\chi_{i,j}(t) - \frac{\mu}{K_q} \ \Delta w^\chi_{i,j}(t), \ \ \chi = U, L. \tag{30}$$

2) Updating Output Weights: The output weight matrices $\boldsymbol{C^U}$ and $\boldsymbol{C^L}$ of the ULTRNN are updated as

$$c^\chi_{i,j}(t+1) = \ c^\chi_{i,j}(t) - \frac{\mu}{K_q} \ \Delta c^\chi_{i,j}(t), \ \ \chi = U, L \tag{31}$$

where, $\Delta c^\chi_{i,j}(t)$ is the accumulation of the instantaneous error gradient given by

$$\begin{aligned}\Delta c^\chi_{h,j}(t) &= \frac{\partial J_q(t)}{\partial c^\chi_{h,j}(t)} = \sum_{k=1}^{K_q}\left(\frac{\partial J_q(k,t)}{\partial c^\chi_{h,j}(t)}\right) \\ &= \sum_{k=1}^{K_q}\left(\frac{\partial J_q}{\partial y_h(k)}\right)\left(\frac{\partial y_h(k)}{\partial c^\chi_{h,j}}\right) \\ &\qquad h = 1\ldots N_o, \ j = 1\ldots N_s \ , \ \ \chi = U, L \end{aligned} \tag{32}$$

where

$$\frac{\partial y_h(k)}{\partial c^\chi_{h,j}} = f_b^{'}\left(\sum_{j=1}^{N}\left(c^U_{h,j}x^U_j(k)\right) + \left(c^L_{h,j}x^L_j(k)\right)\right)c^\chi_{h,i}, \ \ \chi = U, L \tag{33}$$

$$\frac{\partial J_q}{\partial y_h(k)} = -e_h(k) \ , \ \ h = 1\ldots N_o. \tag{34}$$

3) Updating Input Weights: The input weight matrices $\boldsymbol{B^U}$ and $\boldsymbol{B^L}$ are updated as

$$b^\chi_{i,j}(t+1) = \ b^\chi_{i,j}(t) - \frac{\mu}{K_q} \ \Delta b^\chi_{i,j}(t), \ \ \chi = U, L \tag{35}$$

where, $\Delta b^\chi_{i,j}(t)$ is the accumulation of the instantaneous error gradient given by

$$\begin{aligned}\Delta b^\chi_{i,j}(t) = \frac{\partial J_q(t)}{\partial b^\chi_{i,j}(t)} = \sum_{k=1}^{K_q}\left(\frac{\partial J_q(k,t)}{\partial b^\chi_{i,j}(t)}\right) = \ \sum_{k=1}^{K_q}\varepsilon^\chi_i(k)\left(\frac{\partial x^\chi_i(k)}{\partial b^\chi_{i,j}}\right) \\ i = 1\ldots N_s, \ j = 1\ldots N_i \ , \ \ \chi = U, L \end{aligned} \tag{36}$$

where, $\varepsilon^\chi_i(k)$ is given by (21), and

$$\frac{\partial x^\chi_i(k)}{\partial b^\chi_{i,j}} = \ f_a^{'}\big(s^\chi_i(k)\big)u^\chi_j(k-1), \ \ \chi = U, L. \tag{37}$$

4) Updating Bias Weights: The bias weight vectors $\boldsymbol{d^U}$ and $\boldsymbol{d^L}$ are updated as

$$d^\chi_i(t+1) = \ d^\chi_i(t) - \frac{\mu}{K_q} \ \Delta d^\chi_i(t), \ \ \chi = U, L \tag{38}$$

where, $\Delta d^\chi_i(t)$ is the accumulation of the instantaneous error gradient given by

$$\begin{aligned}\Delta d^\chi_i(t) = \frac{\partial J_q(t)}{\partial d^\chi_i(t)} = \sum_{k=1}^{K_q}\left(\frac{\partial J_q(k,t)}{\partial d^\chi_i(t)}\right) = \ \sum_{k=1}^{K_q}\varepsilon^\chi_i(k)\left(\frac{\partial x^\chi_i(k)}{\partial d^\chi_i}\right) \\ i = 1\ldots N_s, \ \ \chi = U, L \end{aligned} \tag{39}$$

where, $\varepsilon^\chi_i(k)$ is given by (21), and

$$\frac{\partial x^\chi_i(k)}{\partial d^\chi_i} = f_a^{'}\big(s^\chi_i(k)\big), \ \ \chi = U, L. \tag{40}$$

F. Learning Algorithm for Reduced Storage

With the conventional BPTT applied to the ULTRNN, the memory required for the storage of the state, the input and the error vectors is of $O\{K_q(N_s + N_i + N_o)\}$. With the modified BPTT algorithm reported in [5] the storage requirement can substantially be reduced at the cost of computational time, by recomputing the state variables in each step of the backward pass from the previously recomputed values of the state variables.

The recomputation of the state vectors is performed recursively according to

$$\begin{aligned}x^{\chi'}_i(k) = \sum_{j=1}^{N_s} w^{\chi\,*}_{i,j} \left\{ f_a^{-1}\Big(x^{\chi'}_i(k+1)\Big) - \sum_{j=1}^{N_i} b^\chi_{i,j} \ u^\chi_j(k) \right. \\ \left. - \ d^\chi_i \right\}, \ \ i = 1\ldots N_s, \ \ \chi = U, L \end{aligned} \tag{41}$$

where $x_i^{\chi'}(k)$ is the recomputed value of the ith state variable at each backward pass step k, and $w_{i,j}^{\chi\,*}$ is the i,jth element of the inverse of the state weight matrix $\boldsymbol{W}^{\boldsymbol{\chi}}$.

From (41), it is seen that the recomputation of the state variables requires the inversion of the sigmoidal function $f_a(\cdot)$ and the inverses of $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$. The triangular structure of $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$ together with the requirement that their block diagonal elements remain on the unit circle facilitates the computation of their inverses by simple iterative operations on the elements of $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$, using the following outline steps [25].

Step 1: Compute the inverse of each of the block diagonal submatrices as its transpose, that is

$$\left(W_n^{\chi}\right)^{-1} = \left(W_n^{\chi}\right)^T,\ n = 1 \ldots \frac{N_s}{2},\ \chi = U, L. \tag{42}$$

Step 2: Partition $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$ along their diagonal into square submatrices with each submatrix composed of an adjacent diagonal block pair the inverses of which have been computed in the previous iterative step along with the associated off-diagonal blocks. The partitioned ith submatrix V_i^U of $\boldsymbol{W}^{\boldsymbol{U}}$ is of the generic form

$$V_i^U = \begin{bmatrix} \Psi_{11}^U & \Psi_{12}^U \\ 0 & \Psi_{22}^U \end{bmatrix}_i. \tag{43}$$

The partitioned ith submatrix V_i^L of $\boldsymbol{W}^{\boldsymbol{L}}$ is of the form

$$V_i^L = \begin{bmatrix} \Psi_{11}^L & 0 \\ \Psi_{21}^L & \Psi_{22}^L \end{bmatrix}_i. \tag{44}$$

The inverses of V_i^U and V_i^L are computed as

$$\left(V_i^U\right)^{-1} = \begin{bmatrix} \left(\Psi_{11}^U\right)^{-1} & -\left(\Psi_{11}^U\right)^{-1}\Psi_{12}^U\left(\Psi_{22}^U\right)^{-1} \\ 0 & \left(\Psi_{22}^U\right)^{-1} \end{bmatrix}_i \tag{45}$$

$$\left(V_i^L\right)^{-1} = \begin{bmatrix} \left(\Psi_{11}^L\right)^{-1} & 0 \\ -\left(\Psi_{22}^L\right)^{-1}\Psi_{21}^L\left(\Psi_{11}^L\right)^{-1} & \left(\Psi_{22}^L\right)^{-1} \end{bmatrix}_i. \tag{46}$$

Step 3: Step 2 is repeated recursively until the inverse of the entire triangular matrix has been computed.

It was shown in [5] and [17] that the state recomputation technique helps to significantly reduce the storage requirement but may occasionally result in numerical instability, especially for long training sequences due to the recursive nature of the recomputation. A key reason for the numerical instability is the amplifying effect of the inverse of the state-transition matrix on a small error introduced in any step of the iteration. The dynamics of the numerical stability error was shown to be the same as the dynamics of the recomputation of the state variable, hence it is divergent. In the case of the ULTRNN, the error increase due to the recursive nature of the amplifying effect is eliminated, as the eigenvalues of $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$, and hence, that of their inverses, are on the unit circle. However, numerical instability may still occur due to the amplifying effect of ${f_a}^{-1}(\cdot)$ on the recomputed state variables. For practical situations, despite this effect, it is still possible to perform several recursive computations without significant loss of accuracy. If numerical stability is degraded during the recursive computation, recovery is possible through a back-stepping method using the selective intermediate state vectors captured and stored during the forward pass computation as has been previously demonstrated for the BDRNN [5]. The steps for the recomputation of the state vectors with numerical stability are as follows.

Step 1 (Recursive State Vector Recomputation): In the forward pass, intermediate values of the state vectors are stored at evenly spaced intervals over the length of the training pattern. In the backward pass, these stored values are used as initial values for the recomputations of the state vectors performed over sublengths of the training pattern.

Step 2 (Monitoring Numerical Stability): Any signs of numerical instability in the backward pass are monitored using a scalar shadow error $e^s(k)$, which is obtained by comparing a scalar shadow output $y_h^f(k)$ computed in the forward pass with a scalar shadow output $y_h^r(k)$ computed in the backward pass using the recomputed values of the state vectors. The shadow error is thus a measure of the numerical stability of the state recomputations.

Step 3 (Recovery): When the shadow error $e^s(k)$ exceeds a threshold value, the state vectors $\{x_i^U(k)\}$ and $\{x_i^L(k)\}$ are recomputed from the nearest intermediate stored state vectors per step 1 by iteratively using the forward pass computations.

V. Illustrative Examples

Several illustrative examples are presented in this section to demonstrate the effectiveness of the proposed ULTRNN architecture. The examples encompass: 1) 1-step prediction of single-variable chaotic time series; 2) autonomous generation of loop-like waveforms; 3) output response reproduction of a multiple-input multiple-output (MIMO) plant driven by noisy input waveforms; 4) 5-step prediction of multivariable chaotic time series; and 6) n-step prediction ($n = 1, 2$, and 100) of a low-frequency waveform with chaotic pulsations. Most of the examples employ input signal parsing discussed in Section II-B to encode prior knowledge when the target dynamics exhibit nonlinear features such as dissymmetry, kurtosis, skewness, varied harmonic content, and multiple time constants.

In the following examples, the ULTRNNs were trained with the initial angular variable θ_n randomly assigned values uniformly distributed in the range (0:2π), and the corresponding block diagonal weights of $\boldsymbol{W}^{\boldsymbol{U}}$ and $\boldsymbol{W}^{\boldsymbol{L}}$ computed according to (6). The nondiagonal elements of the ULTRNN, the elements of the corresponding input, the output and the bias weight matrices are initialized with random values uniformly distributed in the range (−1:1).

A. Example 1: Chaotic Henon Attractor Prediction

This example considers the one step ahead prediction of the Henon attractor [26] given by

$$y^p(k) = 1.5z^p(k) \tag{47}$$

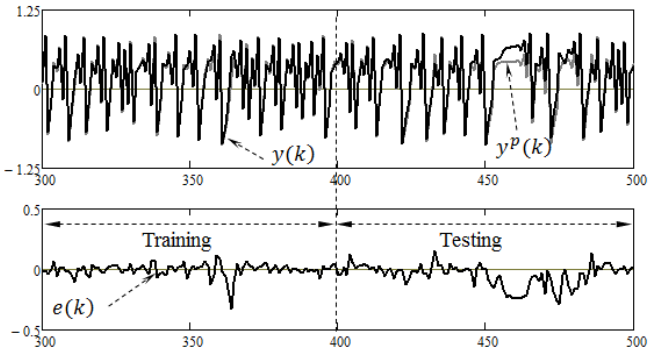

Fig. 6. Case 2. Henon attractor—rectified input parsing. Plant and ULTRNN outputs (top) and prediction error (bottom).

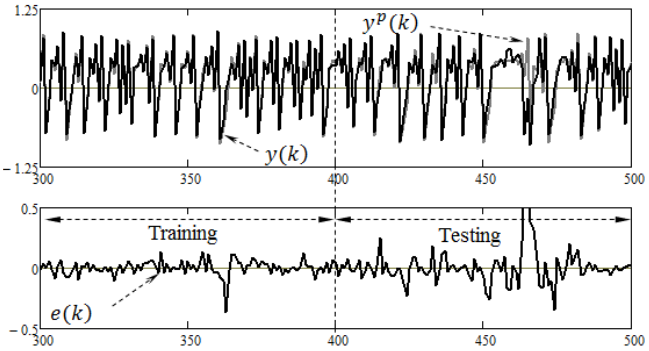

Fig. 7. Case 3. Henon attractor—sum-difference parsing. Plant and ULTRNN outputs (top) and prediction error (bottom).

where $z^p(k)$ is generated by state equations

$$x^p(k+1) = 1 - 1.4x^p(k)^2 + z^p(k);\ \ z^p(k+1) = 0.3x^p(k). \tag{48}$$

The Henon attractor exhibits chaotic behavior as shown in Figs. 6 and 7.

Several variants of the input parsing are formulated and an ULTRNN with 16 feedback variables representing eight oscillatory modes was trained to perform 1-step output prediction, with $y^p(k)$ given as the inputs. Input parsing is employed as the chaotic Henon attractor exhibits dissymmetry around its mean value, and high frequency variations are largely confined to the positive half of each cycle. For each input configuration, the network was trained over a cycle of 400 sampling instants with the weights updated at the end of each cycle. Convergence was achieved after about 50 000 training iterations with a nominal learning rate μ = 2E-4. Comparative results of the ULTRNN performance for three selective input configurations for an independent test data set consisting of 400 sampling instants are summarized in Table II.

As noted from Table II, all three cases of the 8-mode ULTRNNs exhibit acceptable prediction performance. The ULTRNNs with sum-difference parsing (case 3) and the rectification parsing (case 2) outperform the ULTRNN with the unprocessed inputs (case 1). However, the ULTRNN with sum-different parsing is perhaps handicapped by overfitting as evident from the increased NRMSE on test data. When the rectified values ($|y^p(k)|$) are used as the inputs (case 2),

TABLE II
ULTRNN PERFORMANCE WITH VARIANTS OF INPUT PARSING

Case	Network Inputs		NRMSE	
	Upper subnet	Lower subnet	Training	Test
1	$y^p(k)$	$y^p(k)$	0.163	0.253
2	$\|y^p(k)\|$	$-\|y^p(k)\|$	0.108	0.133
3	$y^p(k)+y^p(k-1)$	$y^p(k)-y^p(k-1)$	0.108	0.201

the ULTRNN outperforms others in terms of robust learning as evident from the lowest NRMSE for test data, and this can be attributed to the fact that the rectified wave eases the task of modeling the plant's high frequency oscillatory modes leading to improved generalization. The performance of the networks of cases 2 and 3 for a set of training and test samples are depicted in Figs. 6 and 7. A similar prediction task with several ESN variants was considered in [11]. While an objective one-to-one comparison is difficult, it is seen that the test performance of the ULTRNN with rectified parsing (NRMSE: 0.133) is comparable to the ESN delay line reservoir variant with a reservoir size of 50 (NRMSE: 0.108) [11].

B. Example 2: Autonomous Loop Generation

This example considers the autonomous generation of a 7-petal trajectory shown in Fig 8(a). The 7-petals trajectory is generated by the autonomous plant given by

$$\begin{bmatrix} y_1^p(k) \\ y_2^p(k) \end{bmatrix} = \frac{7}{10}\begin{bmatrix} \sin\left(\frac{k}{2}+1.173\right) & \sin\left(\frac{7}{2}k\right) \\ \cos\left(\frac{k}{2}+1.173\right) & \sin\left(\frac{7}{2}k\right) \end{bmatrix}. \tag{49}$$

A plot of $y_2^p(k)$ versus $y_1^p(k)$ yields the target trajectory as shown in Fig. 8(a), with the length of the trajectory K_q = 120.

An autonomous ULTRNN with eight state variables (four oscillatory modes), with two output units was trained without any input to reproduce the 7-petals flower. With no inputs, the ULTRNN must be marginally unstable if its outputs are to move through the 7-petal limit cycles. The training was started with randomly allotted weights, and after each iteration τ of the 7-petal trajectory the initial state variable conditions were set according to

$$x_i^n(0)|_{\tau+1} = \rho x_i^n(0)|_\tau + (1-\rho)x_i^n\left(K_q\right)|_\tau \tag{50}$$

where the filter time-constant ρ was set at 0.95. This technique for resetting the initial conditions is to ensure stable learning as the network's performance is highly dependent on the initial states with no inputs to guide the learning process.

The ULTRNN was trained with 120 points of the 7-petal trajectory (49), with the weights updated at the end of a cycle of the trajectory. The training was performed until convergence achieved after 100 000 iterations with a relatively low learning rate μ = 5E-6. After training, for testing purposes, the ULTRNN's "freewheeling" autonomous outputs were collected for six additional cycles of the 7-petal trajectory, and compared with the corresponding target outputs. A plot of $y_2(k)$ versus $y_1(k)$ shown in Fig. 8(b) depicts the ULTRNN output trajectory for these six cycles. Fig. 9 shows the ULTRNN test outputs and the corresponding error traces for

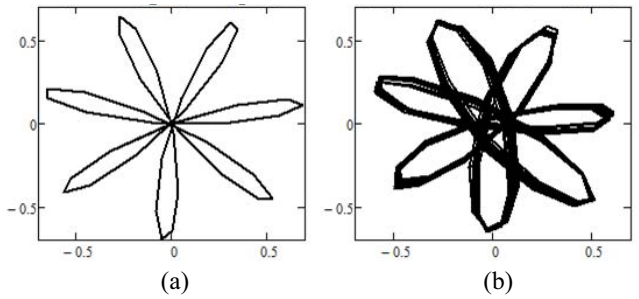

Fig. 8. Autonomous loop generation. (a) Plant: $y_2^p(k)$ versus $y_1^p(k)$. (b) ULTRNN: $y_2(k)$ versus $y_1(k)$.

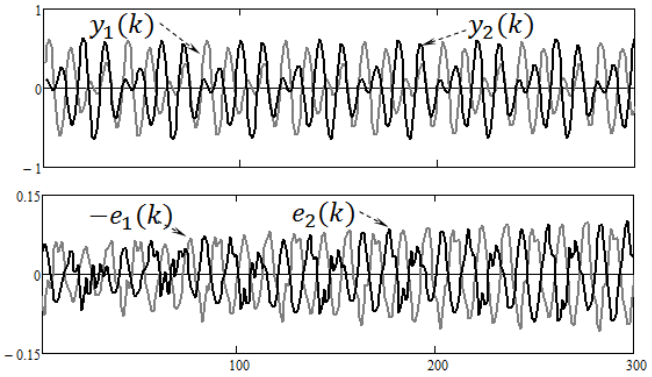

Fig. 9. Autonomous loop generation—(top) ULTRNN test outputs and (bottom) tracking errors.

300 sampling instants. It was observed that the ULTRNN's performance degrades uniformly with NRMSE increasing from 0.08 to 0.19 over six cycles as the small errors of a cycle accumulate and impact the performance in the next cycle.

Although not addressed in this example, it should be noted that for long length trajectories (large K_q) the ULTRNN can still be effectively trained by partitioning the data set into several segments (e.g., one segment for each petal) and the weights and the initial conditions updated at each segment transition. As the training progresses, additional segments (e.g., two, four, and all seven petals can be presented before updating the weights and the initial conditions.

C. Example 3: MIMO Plant Modeling

This example considers the modeling of a MIMO plant studied in [27], given by

$$\begin{bmatrix} x_1^p(k+1) \\ x_2^p(k+1) \end{bmatrix} = \frac{1}{1+\left[x_2^p(k)\right]^2}\begin{bmatrix} x_1^p(k) \\ x_1^p(k)x_2^p(k) \end{bmatrix} + \begin{bmatrix} u_1^p(k) \\ u_2^p(k) \end{bmatrix} \quad (51)$$

$$\begin{bmatrix} u_1^p(k) \\ u_2^p(k) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\left\{\sin\frac{2\pi k}{25} + \sin\frac{2\pi k}{10}\right\} + \eta_1(k) \\ \frac{1}{2}\left\{\cos\frac{2\pi k}{25}\right\} + \eta_2(k) \end{bmatrix}. \quad (52)$$

The plant output is given by $y_1^p(k) = 0.2x_1^p(k)$, and $y_2^p(k) = 0.2x_2^p(k)$, and $\eta_1(k)$ and $\eta_2(k)$ represent noise terms.

A two-input two-output 12-state ULTRNN (six oscillation modes) employing parsed inputs was trained to model this plant. Observing that one input of the plant is a sinusoidal at a single frequency, while the other input has additional harmonic content, the sum-difference parsing per (8) was employed to combine the inputs so to distribute the input harmonic content to both subnets. As shown in Fig. 10, the two inputs to the upper subnet are the plant inputs $u_1^p(k)$, and $u_2^p(k)$ while the two inputs to the lower subnet are the sum of the plant inputs $u_1^p(k) + u_2^p(k)$, and the difference of the plant inputs $u_1^p(k) - u_2^p(k)$. The target plant outputs are shown at the bottom of Fig. 10. Training was performed over an epoch of 100 time steps with the variance of noise terms $\eta_1(k)$ and $\eta_2(k)$ set at 0.1, and the network weights were updated at the end of each epoch. Convergence was achieved after about 70 000 training iterations with a learning rate μ = 1E-5. The trained network was tested over 1000 data points with the noise variance in the range of 0.05 to 0.2. The average training and test NRMSEs at different noise variances are presented in Table III. Fig. 11 shows the ULTRNN output traces compared to the plant outputs and the corresponding errors for test data.

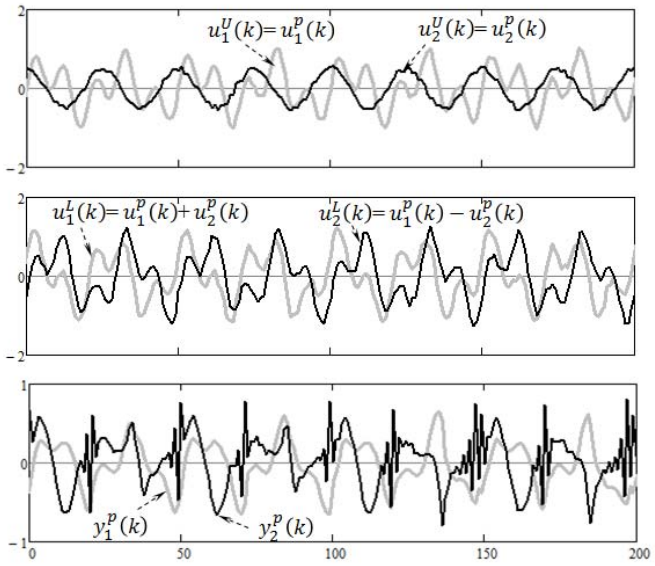

Fig. 10. MIMO plant modeling—(top and middle) ULTRNN inputs and (bottom) target outputs.

TABLE III
MIMO PLANT MODELING ULTRNN PERFORMANCE VERSUS NOISE VARIANCE

Noise Variance	Training NRMSE		Test NRMSE	
	Output 1	Output 2	Output 1	Output 2
0.05	0.225	0.566	0.237	0.55
0.1	0.28	0.633	0.262	0.609
0.15	0.32	0.64	0.26	0.66
0.2	0.3	0.68	0.366	0.74

It is noted that while the ULTRNN is robustly tracking the output trajectories for varied noise variance cases, the high NRMSE in modeling the second output can be attributed to the unpredictability of the amplifying effect of the input noise on the output trajectory due to the multiplicative term in the target state transition matrix (51). This was confirmed by training and testing the ULTRNN with no input noise which yielded an NRMSE of 0.137 for output 1 and 0.117 for output 2.

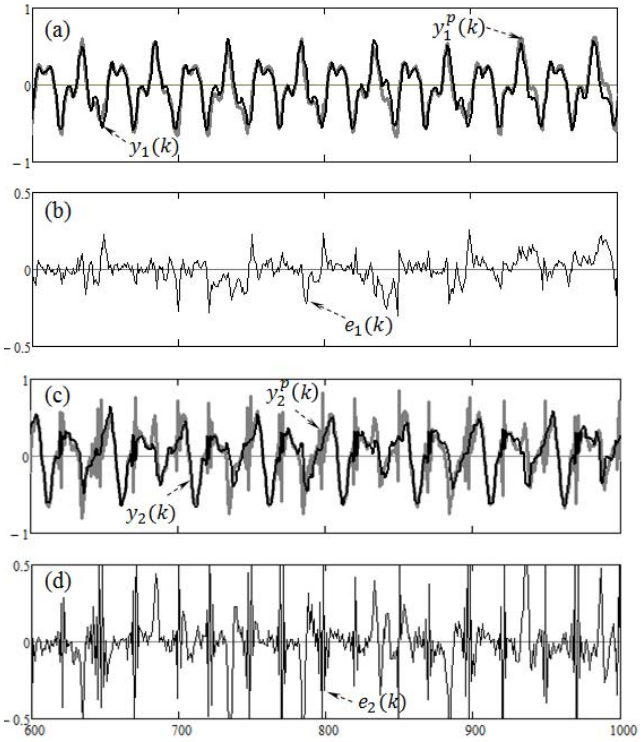

Fig. 11. MIMO plant modeling—ULTRNN test outputs [(a) and (c)] and errors [(b) and (d)] for inputs with noise (var. 0.1).

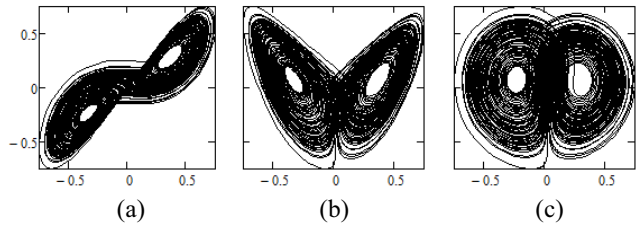

Fig. 12. Lorentz limit cycles. (a) $y_2^p(k)$ versus $y_1^p(k)$. (b) $y_3^p(k)$ versus $y_1^p(k)$. (c) $y_3^p(k)$ versus $y_2^p(k)$.

Addition of feedforward structures as in [17] may be beneficial in improving modeling accuracy when input noise is present.

D. Example 4: Lorentz Limit Cycles 5-Step Prediction

This example considers the 5-step prediction of the the Lorenz limit cycles [28] produced by solving for the ordinary differential equation (ODE) given by

$$\frac{d}{dt}\begin{bmatrix} y_1^p \\ y_2^p \\ y_3^p \end{bmatrix} = \begin{bmatrix} \sigma\left(y_2^p - y_1^p\right) \\ \rho y_1^p - y_1^p y_3^p - y_2^p \\ y_1^p y_2^p - \beta y_3^p \end{bmatrix} \tag{53}$$

where $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$. The ODE is solved with initial values set at $y_1^p = -10$, $y_2^p = -8$, and $y_3^p = 30$, using a fourth order Runge Kutta with a step size of 0.001 to provide values of y_i^p, $i = 1, 2, 3$ at discrete intervals. The limit cycle trajectories are plotted in Fig. 12.

A three-input, three-output ULTRNN with 12 state variables (representing six oscillation modes) was trained to predict the plant output. In order to improve the learning performance of the ULTRNN given the waveform nonsymmetry as evident

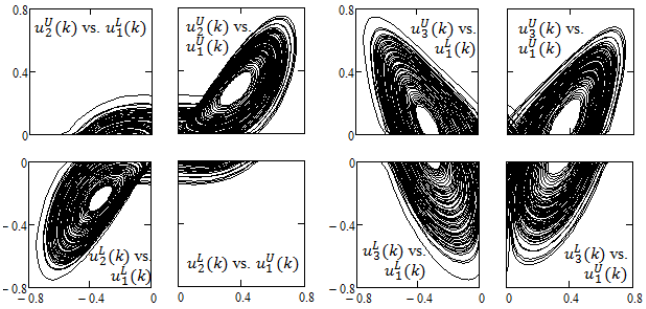

Fig. 13. Lorentz limit cycles—threshold input parsing.

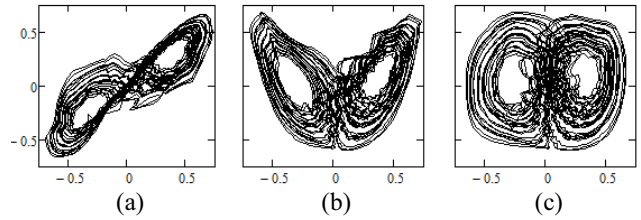

Fig. 14. Lorentz limit cycles—ULTRNN test outputs. (a) $y_2(k)$ versus $y_1(k)$. (b) $y_3(k)$ versus $y_1(k)$. (c) $y_3(k)$ versus $y_2(k)$.

from Fig. 12, the inputs of the ULTRNN are parsed such that the positive signal values of $y_1^p(k)$, $y_2^p(k)$, and $y_3^p(k)$ are fed as inputs $u_1^U(k)$, $u_2^U(k)$, and $u_3^U(k)$, respectively, to the upper triangular subnetwork, and their negative signal values are fed as inputs $u_1^L(k)$, $u_2^L(k)$, and $u_3^L(k)$, respectively, to the lower triangular subnetwork. As examples, the graphical depiction of threshold parsing for the first and second signals are given in Fig. 13.

Training was performed with 10 000 points of the Lorenz data series described by the solution to (53), with the weights updated after each training run consisting of 100 points. The network output is the 5-step predicted values. Convergence was achieved after 50 000 training iterations with a nominal learning rate $\mu = 1\text{E-}4$. The performance of the ULTRNN for each output was assessed by testing with an additional 4500 points. Fig. 14 shows the time-series trajectories as predicted by the ULTRNN for the test data series. Fig. 15 shows the ULTRNN output traces compared with the corresponding Lorentz plant output traces for the test data series, and the corresponding error traces. The NRMSE was less than 0.067 for all three test data series. It is observed that the ULTRNNs performance does not degrade significantly over the 4500 points of the Lorentz limit cycles. A 1-step prediction task was considered in [29] that uses a DTRNN with 89 free trainable parameters. While one-to-one objective comparison is not possible, it is evident that the 5-step prediction of the ULTRNN with threshold parsing outperforms the 1-step prediction results in [29].

E. Example 5: Santa Fe Laser Data n-Step Prediction

This example considers the Santa Fe Laser generated data set [30] that consists of periodic to chaotic intensity pulsations of a far-infrared-laser. The objective is to use ULTRNNs for three cases of n-step prediction of the laser data series ($n = 1, 2,$ and 100). It is observed that the laser output

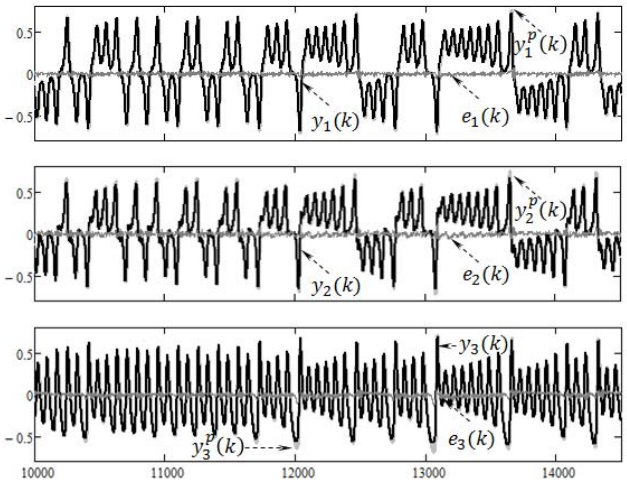

Fig. 15. Lorentz limit cycles—ULTRNN outputs versus target wavforms and errors for test data.

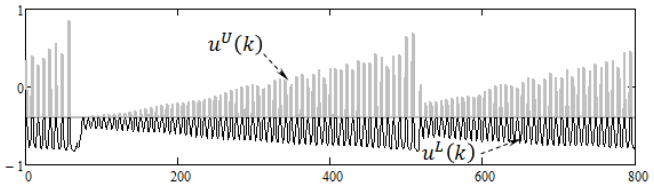

Fig. 16. Sante Fe Laser data—mean-threshold input parsing.

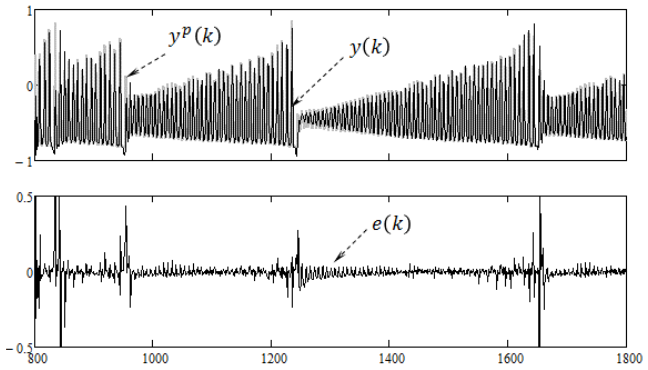

Fig. 17. Santa Fe Laser data 1-step prediction—ULTRNN test output versus (top) target wavefrom and (bottom) error.

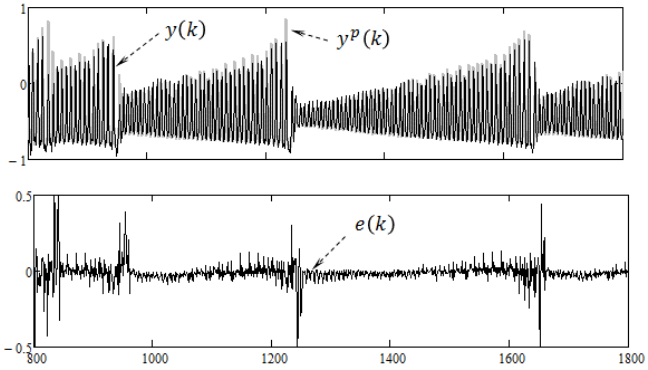

Fig. 18. Santa Fe Laser data 2-step prediction—ULTRNN test output versus (top) target wavefrom and (bottom) error.

$y^p(k)$ exhibits significant dissymmetry around the mean value, and mimics a chaotic low frequency waveform modulating a high frequency carrier. For the short term (1-step and 2-step) prediction cases threshold parsing per (7) is used, while for the long term (100-step) prediction case envelop parsing per (10) is used.

The input parsing for the 1-step and 2-step prediction cases is depicted in Fig. 16. The input $u^U(k)$ to the upper subnetwork of the ULTRNN consists only of the values of $y^p(k)$ greater than or equal to the mean $\overline{y^p(k)}$. The input $u^L(k)$ to the lower subnetwork consists of values of $y^p(k)$ below the mean $\overline{y^p(k)}$.

The 1-step prediction task uses an 8-state ULTRNN (four oscillatory modes). The ULTRNN was trained with the first 800 points of the Santa Fe laser data series *A.cont* as depicted in Fig. 16, with the weights updated after every 100 points. Convergence was achieved after 50 000 training iterations with a learning rate $\mu = 0.001$. The trained ULTRNN was tested with the following 8000 points of the same data series. The NRMSE obtained for the training dataset is 0.102, and that for the test dataset is 0.180. Fig. 17 shows the ULTRNN output trajectory and the corresponding error trajectory for the first 1000 test data points. A similar task was considered in [11] where the performances of a fully connected ESN and several sparse ESN variants with reservoir sizes ranging from 50 to 200 were assessed. While an objective one-to-one performance comparison is difficult, it is evident that the ULTRNN with only four oscillatory modes (with a total of 48 free trainable parameters) performs comparatively well with the fully connected ESN with a reservoir size of 50 (NRMSE: 0.136). It is to be noted that, while the number of free trainable parameters in the ULTRNN and the ESN are about the same, the fully connected ESN version employs a large feedback matrix (50×50) with randomly selected fixed weights, while the ULTRNN uses much smaller triangular feedback matrices (8×8) with all trainable weights.

The 2-step prediction task also uses an 8-state ULTRNN trained with 800 points of the laser data series *A.cont* (with the weights updated after every 100 points) and tested with the following 8000 points of the same series. Training convergence was achieved after 80 000 iteration with a learning rate $\mu = 0.001$. The NRMSE for the training dataset is 0.176 and that for the test dataset is 0.210. Fig. 18 shows the ULTRNN ouput trajectory and the corresponding error trace for the first 1000 test data points. The small difference in the NRMSE between the training and the test data sets for both 1-step and 2-step prediction cases indicate good generalization capability.

In order to probe the ULTRNN's performance stability and robustness when the uncertainty in the input increases, the ULTRNN's input was modified to be a combination of the plant output and the predicted ULTRNN output as per

$$u_o(k+2) = \left\{(1-\xi)x^p(k+2) + \xi y^n(k)\right\}. \tag{54}$$

Fig. 19 shows the ULTRNN's NRMSE averaged over 8 sets of 1000 data points from the test dataset *A.cont*, as ξ is varied from 0 to 1. From Fig. 19 it is seen that the NRMSE increases only marginally as ξ increaces from 0 to 0.5 implying that

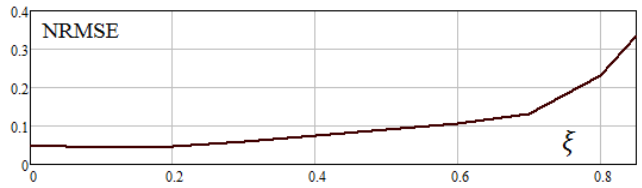

Fig. 19. Santa Fe Laser data 2 step prediction—ULTRNN prediction performance robustness.

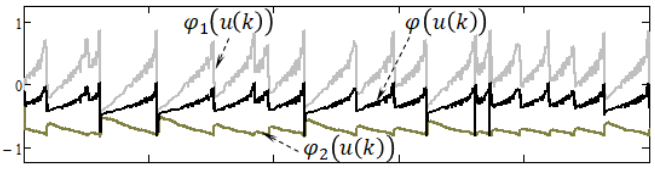

Fig. 20. Santa Fe Laser data—envelop computation for parsing.

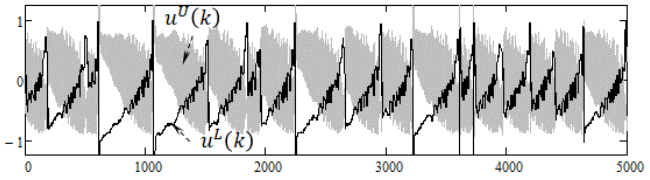

Fig. 21. Santa Fe Laser data 100-step prediction—envelop parsing.

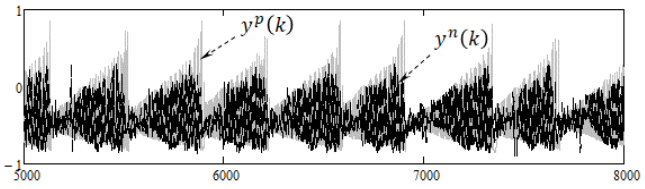

Fig. 22. Santa Fe Laser data 100-step prediction—ULTRNN test output versus target waveform.

the ULTRNN performance is stable and sufficiently robust for a moderate range of input uncertainties.

The 100-step prediction task uses a 24-state ULTRNN (twelve oscillatory modes). Envelope parsing per (10) is employed surmising that it will improve the ULTRNN's ability to learn the laser data's chaotic low frequency envelop while simultaneously keeping track of its high frequency carrier like pulsations. Shown in Fig. 20 is the envelop wave $\varphi(u(k))$ used in parsing computed per (55) by averaging the positive and the negative envelopes, $\varphi_1(u(k))$ and $\varphi_2(u(k))$, respectively

$$\varphi(u(k)) = \frac{1}{2}\{\varphi_1(u(k)) + \varphi_2(u(k))\}. \tag{55}$$

As shown in Fig. 21, the input $\boldsymbol{u}^{\boldsymbol{L}}(k)$ to the lower subnetwork is the average envelop $\varphi(u(k))$, and the input $\boldsymbol{u}^{\boldsymbol{U}}(k)$ to the upper subnetwork is the product of the average envelop and the laser data

$$\boldsymbol{u}^{\boldsymbol{U}}(k) = \varphi(u(k))u(k);\ \ \boldsymbol{u}^{\boldsymbol{L}}(k) = \varphi(u(k)). \tag{56}$$

The ULTRNN was trained with the first 5000 points from the data series *A.cont*, with the weights updated every 100 data points. Training convergence was achieved after 150 000 iterations with a learning rate $\mu = 1\text{E-}5$. The NRMSE obtained for the training set is 0.729. The trained ULTRNN was tested on the next 4500 data points. The NRMSE obtained for the test data set is 0.676. Shown in Fig. 22 are the first 3000 test data points and the corresponding ULTRNN output. As suggested in [31] where a prediction task on the laser data was considered using a large multilayer perceptron-based network, the large NRMSE is attributable to the fact that the laser dynamic is highly unpredictable at signal collapses with several possible continuations. As a result, the ULTRNN output exhibits a "phase lag" with respect to the laser signal's high frequency pulsations at transitions following signal collapses. The result is that the ULTRNN's short-term performance is worse for 100-step prediction than it is for 1-step and 2-step predictions considered earlier. None the less, it is clearly evident from Fig. 22, that the ULTRNN with envelop parsing has indeed learnt to model the long-term dependency.

VI. Conclusion

This paper has dealt with an effective analytical formulation and development of the ULTRNN architecture using twin triangular feedback weight matrices, and has extended the learnings from the previously studied BDRNN structure on aspects including network and learning stability, simplicity of gradient descent learning process, storage requirement minimization, and improved robustness and sensitivity. The ULTRNN architecture and the training algorithm presented are shown to facilitate enhanced and highly effective solution to a wide range of time series prediction problems. Specific contributions of this paper include the following.

1) The triangular architecture of the ULTRNN is aptly suited for modeling and prediction of dynamic trajectories with inherent and varied oscillatory modes.
2) The ULTRNN is inherently stable with the eigenvalues of the feedback weight matrices constrained to lie on the unit circle in the complex z-plane.
3) The nonzero off-diagonal elements of the ULTRNN feedback weight matrices facilitate close interaction between the feedback state variables of the network resulting in improved learning when compared with other sparse networks.
4) The novel weight update technique of the block-diagonal submatrices is formulated based on the differential of the angular error variable $\Delta\theta_n$ that constrains the eigenvalues of the feedback weight matrices to lie on the unit circle. This technique increases the network's sensitivity in detecting the underlying oscillatory modes of the time series being modeled while maintaining learning stability. It also eliminates the need for monitoring on-line learning stability through specific ad-hoc means.
5) The ULTRNN aims to mimic the performance of a fully connected recurrent network, with the eigenvalues of twin-triangular subnetworks constrained to be the same to facilitate enhanced interaction between the feedback state variables, resulting in better trajectory learning while mitigating overfitting.

6) The twin triangular subnetworks of the ULTRNN allow the parsing and channeling of the input signals, thereby improving the learning capability, specifically for dynamic processes that exhibit chaotic, nonsymmetric, and/or long-term dependency behavior.

REFERENCES

[1] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[2] L. B. Almeida, "Backpropagation in perceptrons with feedback," in *Neural Computers* (NATO ASI Series), vol. 41, R. Eckmiller and C. V. D. Malsburg, Eds. Berlin, Germany: Springer-Verlag, 1988, pp. 199–208.

[3] F. J. Pineda, "Recurrent backpropagation and the dynamical approach to adaptive neural computation," *Neural Comput.*, vol. 1, no. 2, pp. 161–172, 1989.

[4] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1212–1228, Sep. 1995.

[5] S. C. Sivakumar, W. Robertson, and W. J. Phillips, "Online stabilization of block-diagonal recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 10, no. 1, pp. 167–175, Jan. 1999.

[6] P. A. Mastorocostas and C. H. Hilas, "A stable learning algorithm for block-diagonal recurrent neural networks: Application to the analysis of lung sounds," *IEEE Trans. Syst., Man, Cybern. B, Cybern*., vol. 36, no. 2, pp. 242–254, Apr. 2006.

[7] A. C. Tsoi and A. D. Black, "Locally recurrent globally feedforward networks: A critical review of architectures," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 229–239, Mar. 1994.

[8] M. C. Ozturk, D. Xu, and J. C. Príncipe, "Analysis and design of echo state networks," *Neural Comput*., vol. 19, no. 1, pp. 111–138, Jan. 2007.

[9] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Proc. 15th Int. Conf. Neural Inf. Process. Syst.*, 2002, pp. 609–616.

[10] M. Lukoševièius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade* (LNCS 7700). Berlin, Germany: Springer-Verlag, 2012, pp. 659–686.

[11] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, Jan. 2011.

[12] S. Basterrech, "An empirical study of the L2-boost technique with echo state networks," in *Proc. 13th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Bangi, Malaysia, Dec. 2013, p. 8.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[14] D. Prokhorov, "Echo state networks: Appeal and challenges," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 3. Montreal, QC, Canada, 2005, pp. 1463–1466.

[15] T. D. Batzel and K. Y. Lee, "An approach to sensorless operation of the permanent-magnet synchronous motor using diagonally recurrent neural networks," *IEEE Trans. Energy Convers.*, vol. 18, no. 1, pp. 100–106, Mar. 2003.

[16] P. Mastorocostas, C. Hilas, D. Varsamis, and S. Dova, "A recurrent neural network–based forecasting system for telecommunications call volume," *Appl. Math. Inf. Sci.*, vol. 7, no. 5, pp. 1643–1650, 2013.

[17] S. C. Sivakumar, "Architectures and algorithms for stable and constructive learning in discrete time recurrent neural networks," Ph.D. dissertation, Dept. Elect. Eng., Dalhousie Univ., Halifax, NS, Canada, 1997.

[18] N. Srivastav, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] T. Joachims, "Transductive learning via spectral graph partitioning," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, Washington, DC, USA, 2003, pp. 290–297.

[20] M. Culp, G. Michailidis, and K. Johnson, "On multi-view learning with additive models," *Ann. Appl. Stat.*, vol. 3, no. 1, pp. 292–318, 2009.

[21] A. Sardá-Espinoza. *Comparing Time-Series Clustering Algorithms in R Using the dtwclust Package*. Accessed: May 31, 2017. [Online]. Available: https://cran.r-project.org/web/packages/dtwclust/vignettes/dtwclust.pdf

[22] S. Y. Kung, *Digital Neural Networks: Deterministic Temporal Neural Networks.* Englewood Cliffs, NJ, USA: Prentice-Hall, 1993, ch. 6, pp. 219–224.

[23] L. Jin and M. M. Gupta, "Globally asymptotical stability of discrete-time analog neural networks," *IEEE Trans. Neural Netw.*, vol. 7, no. 4, pp. 1024–1031, Jul. 1996.

[24] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[25] W. Nasri and Z. Mahjoub, "Optimal parallelization of a recursive algorithm for triangular matrix inversion on MIMD computers," *Parallel Comput.*, vol. 27, no. 13, pp. 1767–1782, Dec. 2001.

[26] M. Hénon, "A two-dimensional mapping with a strange attractor," *Commun. Math. Phys.*, vol. 50, no. 1, pp. 69–77, 1976.

[27] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[28] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmos. Sci.*, vol. 20, no. 2, pp. 130–141, 1963.

[29] S. V. Dudul, "Prediction of a Lorenz chaotic attractor using two-layer perceptron neural network," *Appl. Soft Comput.*, vol. 5, no. 4, pp. 333–355, 2005.

[30] A. S. Weigend and N. A. Gershenfeld. *The Santa Fe Time Series Competition Data: Data Set A: Laser Generated Data*. Accessed: Aug. 18, 2016. [Online]. Available: http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html

[31] R. Bakker, J. C. Schouten, C. L. Giles, F. Takens, and C. M. van den Bleek, "Learning chaotic attractors by neural networks," *Neural Comput.*, vol. 12, no. 10, pp. 2355–2383, 2000.

Seshadri Sivakumar (S'81–M'86–SM'14) received the B.E. degree in electrical technology and electronics from the Indian Institute of Science, Bengaluru, India, in 1977, and the M.Sc.E. and Ph.D. degrees in electrical engineering from the University of New Brunswick, Fredericton, NB, Canada, in 1983 and 1987.

He is currently the Chief Consultant with Pasumai Energytech LLC, Richmond, CA, USA, and is on assignment with Healy Wave Energy LLC, Hollis, NH, USA, developing smart control and power electronic systems for a wave energy converter. He has led engineering and research and development assignments on power electronic products and systems for Bharat Heavy Electricals Ltd., Bengaluru, India, from 1977 to 1981, Pivotal Power Inc., Bedford, NS, Canada, from 1987 to 2006, MKS Instruments Ltd., Rochester, NY, USA, from 2007 to 2009, United Technologies Research Center, East Hartford, CT, USA, from 2009 to 2010, and SunPower Corp., Richmond, CA, USA, from 2010 to 2016. He was also an Adjunct Faculty Member of electrical engineering with Dalhousie University, Halifax, NS, Canada, from 2004 to 2012. His current research interests include machine learning algorithms and power electronic topologies and control systems for alternative and distributed energy applications.

Shyamala Sivakumar (M'00) received the B.E. degree in electrical engineering from Bangalore University, India, in 1984, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the Technical University of Nova Scotia (currently Dalhousie University), Halifax, NS, Canada, in 1992 and 1997.

She was a Post-Doctoral Research Fellow with the Internetworking Program, Dalhousie University from 1997 to 1999. She started her career as an Aeronautical Engineer with Hindustan Aeronautics Limited, Bengaluru, India, from 1985 to 1989. Her academic career began as an Assistant Professor from 1999 to 2000 with the Internetworking Program, Dalhousie University. She moved to Saint Mary's University, Halifax, in 2000, where she is currently a Professor of computer and information systems with the Sobey School of Business. She has also been an Adjunct Faculty Member of engineering mathematics and internetworking with Dalhousie University since 2000. Her current research interests include architectures and algorithms for recurrent neural networks and quality of service and energy conservation algorithms for wireless sensor and body area networks.