Activation Function Modulation in Generative Triangular Recurrent Neural Networks

Seshadri Sivakumar¹, Senior Member, IEEE, Shyamala Sivakumar², Senior Member, IEEE,
¹Pasumai Energytech LLC, Bradenton, FL, USA
²Computing Information Systems, Saint Mary’s University, Halifax, NS, Canada

Corresponding author: Seshadri Sivakumar (e-mail: seshadr.sivakumar@pasumaienergytech.com).

ABSTRACT Autonomously generating time series is challenging because the network must capture short-term features while tracking long-term time dependencies. This paper introduces the modulation of the activation function slopes of the upper-lower triangular neural networks (ULTRNNs) for dynamic variation of memory through a secondary recurrent network with its own independent states. A zigzag propagation algorithm for weight updates is proposed that accounts for the dynamic interaction of the states between the ULTRNN and the secondary network. A novel training method is proposed that distributes the eigenvalues of the closed-loop system around the unit circle in the complex z-plane to ensure that the network behaves as a nonlinear oscillator with an output that neither collapses nor saturates but continues to emulate the target. Examples encompassing the Lorenz series, Santa Fe laser data, Kolam patterns, electrocardiogram (ECG) signals, stock pricing data, and smart grid data are presented to demonstrate that the proposed approach is highly effective in the generative modeling of complex periodic, chaotic, and nonstationary time series. The qualitative and quantitative performance of the ULTRNN obtained with the proposed activation-function modulation technique is comparable to that of state-of-the-art techniques including feedforward networks and generative adversarial networks, but with far fewer trainable parameters and shorter computation times.

INDEX TERMS Triangular RNN, activation function slope modulation, generative networks, closed-loop training.

I. INTRODUCTION
This paper introduces a novel sparse recurrent neural network (RNN) architecture for the generative modeling of time series. Generative modeling is challenging because the network must capture short-term features while tracking long-term time-dependencies. RNN-based networks are uniquely suited to such problems owing to their autoregression ability. Typically, the activation function slope used in such networks is fixed and determines their ability to model long-term time-dependencies at the cost of short-term features or vice versa. We propose dynamically varying the activation function slope to control the memory retention and forgetting. In addition, unconstrained RNN weight placements may result in unstable networks whose outputs saturate or highly stable networks whose outputs collapse. Therefore, we propose a novel training method to constrain the eigenstructure of the closed-loop system for marginal stability to ensure a stable emulation of the target data with an output that neither collapses nor saturates. We apply activation function modulation, together with constraining the closed-loop eigenvalue distribution, to synthetically generate chaotic, complex periodic and quasi-periodic waveforms, and model non-stationary processes such as stock data.
Sparse RNNs have the advantage of reduced computation and storage compared with fully recurrent networks. In addition, special recurrent structures such as block-diagonal (BDRNN) [1] and upper-lower triangular (ULTRNN) [2] inherently facilitate monitoring stability and ensure a robust learning ability. The eigenvalues of the $2 \times 2$ diagonal blocks of the BDRNN recurrent weight matrix were constrained during the training to lie on the unit circle of the complex $z$-plane. Such placement inherently mitigates the vanishing and exploding gradient issues normally associated with conventional RNN [3]. Reported applications of BDRNNs include speech recognition [1], lung-sound processing [4], and telecom call volume prediction [5]. The ULTRNN [2] uses upper and lower triangular weight matrices, each with $2 \times 2$ diagonal blocks with the same constraints as the BDRNN. As a second constraint, the corresponding diagonal blocks in the two triangular matrices are set equal to reduce the chances of overfitting as the network learns to model the hidden oscillatory modes of the target trajectory. The ULTRNN is effective for modeling dynamic time series that exhibit chaotic, nonsymmetric, and long-term dependency behaviors [2].

Adaptation of the activation function slope has previously been considered a tool for improving learning performance. Reference [6] used this technique with a limited scope to improve the intrinsic stability of generic recurrent networks. Reference [7] used trainable activation functions in a deep feedforward neural network for an MNIST handwritten digit recognition task. In a previous study, the learning capability of a ULTRNN was enhanced by modulating the slope of the activation function associated with each state variable [8]. Modulation of the activation function incorporates variable memory with an objective similar to that of an LSTM network that employs multiplicative gates [9].

The slope of the activation function directly affects the time duration contribution of the corresponding state variable. A sharp activation function slope lengthens the time contribution and helps model and control long-term dependencies. Conversely, a slack slope shortens the time contribution and helps to model controlled “forgetting”. A fixed activation slope favors the equal contribution of all states over time and the network learns long-term dependencies at the expense of its short-term performance and vice versa [8]. Varying the slope of the activation function with time can be used to model the dynamics of the system time constants that produce the target waveform. Reference [8] dynamically computed the slope of the (tanh) squashing function for each state variable using a secondary network. For this secondary network, a feedforward structure that uses the main ULTRNN states was chosen to simplify the training with a direct extension of the conventional backpropagation algorithm. However, such sharing of states may result in suboptimal mapping of the activation function slope space. To address this shortcoming, in this study, we propose using a recurrent architecture for a secondary network with its own independent and dimensionally unconstrained state variables. We call this secondary network an independent state activation function network (ISAFN).

The interaction between the state variables of the ULTRNN and ISAFN imposes additional constraints on the gradient computation process during training. Considering this interaction, we developed a novel zigzag propagation (z-prop) algorithm that uses a combination of forward and backward steps for exact gradient computation. A truncation technique that increases the computational speed by trading off the desired gradient accuracy level is presented. In addition, the activation function slope is constrained to a range that ensures the marginal stability of the ULTRNN.

The activation function modulation of the ULTRNN can be applied to a wide range of time-series modeling and prediction tasks. This study focuses on the unique objective of autonomous generation or replication of time series with behavioral characteristics close to those of the parent system as envisioned in [10], [16]. From a practical perspective, such generative networks can be used to clone new data when collecting real data is difficult or when its availability is sparse. We consider one-step prediction as the primary mechanism for training, such that the trained network autonomously generates outputs by feeding back only its own output from the previous time step. Because the stable performance of the trained network depends on the eigenstructure of the closed-loop system, we introduce an eigenvalue-based heuristic cost function. Using this approach, the closed-loop network behaves as a nonlinear oscillator emulating the target with an output that neither collapses nor saturates. We demonstrate the performance of ISAFN-ULTRNN with the autonomous generation of synthetic examples of Lorenz limit cycles [11], Santa Fe laser data [12], one-stroke kolam patterns [13], ECG signals [14], [15], Google stock data [16], and smart grid data [17], [18].

The main contributions of this work include: (a) the controlled modulation of the activation function slopes of the ULTRNN which facilitates the dynamic variation of memory retention. A secondary recurrent network whose states are independent of the ULTRNN states is employed for the dynamic computation of the activation function slopes. (b) A zigzag propagation algorithm that uses a combination of
The remainder of this paper is organized as follows. In Section II, the structure of the ULTRNN with ISAFN is developed, and its key features are compared with the common-state AFN (CSAFN) presented in [8]. In Section III, we develop a learning algorithm for the ISAFN with a specific focus on the z-prop algorithm for gradient computation, which considers the interaction between the state variables of the ULTRNN and ISAFN. In this section, we also discuss the computational burden of the z-prop algorithm and suggest techniques for increasing computational speed through truncation. Section IV describes the development of an eigenvalue-based cost function that ensures the marginal stability of the linearized closed-loop system. Section V presents illustrative examples to demonstrate and contrast the generative performance of the ISAFN-ULTRNN with state-of-the-art networks. Section VI summarizes the key contributions of this study.

II. DYNAMIC ACTIVATION FUNCTION MODULATION

A. ACTIVATION FUNCTION MODULATED ULTRNN

The ULTRNN architecture uses two triangular state-feedback weight matrices [2] as shown in Figure 1. The system equations, with sampling instant \( k \), are given by:

\[
\begin{align*}
\mathbf{s}_x^\zeta(k+1) &= \mathbf{W}_x^\zeta \mathbf{x}^\zeta(k) + \mathbf{B}_x^\zeta \mathbf{u}(k) \\
\mathbf{x}^\zeta(k) &= \mathbf{f}_x(\mathbf{a}_x^\zeta(k), \mathbf{s}_x^\zeta(k)), \zeta = U, L \\
\mathbf{s}_y^\zeta(k) &= \mathbf{C}_x^U \mathbf{x}^U(k) + \mathbf{C}_x^L \mathbf{x}^L(k) \\
\mathbf{y}(k) &= \mathbf{h}_y(\mathbf{s}_y(k)), \zeta = U, L
\end{align*}
\]

(1)

where, with \( \zeta = U, L \) representing the upper and lower networks, respectively,

\( \mathbf{W}_x^\zeta = \{w_{xi,l,j}^\zeta, i, j = 1, 2, \ldots N_x\} \) are the triangular matrices,

\( \mathbf{B}_x^\zeta = \{b_{xi,l,j}^\zeta, i = 1, 2, \ldots N_x, j = 1, 2, \ldots N_i\} \) are the input matrices,

\( \mathbf{C}_x^\zeta = \{c_{xi,l,j}^\zeta, i = 1, 2, \ldots N_x, j = 1, 2, \ldots N_o\} \) are the output matrices.

With \( i = 1, 2, \ldots N_x \),

\( \mathbf{x}^\zeta(k) = \{x_i^\zeta(k)\}^T \) as the state vector,

\( \mathbf{s}_x^\zeta(k) = \{s_{xi}^\zeta(k)\}^T \) is the intermediate linear state vector,

\( \mathbf{a}_x^\zeta(k) = \{a_{xi}^\zeta(k)\}^T \) is the activation function slope vector, and

\( \mathbf{f}_x(\mathbf{a}, \mathbf{s}) = \{f_x(a_i, s_i)\}^T \) is the state activation function vector.

\( \mathbf{u}(k) = \{u_i(k), i = 1, 2, \ldots N_i\}^T \) is the network input vector. With \( i = 1, 2, \ldots N_o \),

\( \mathbf{s}_y^\zeta(k) = \{s_y^\zeta(k)\}^T \) is the intermediate linear output vector,

\( \mathbf{y}(k) = \{y_i(k)\}^T \) is the network output vector, and

\( \mathbf{h}_y(\mathbf{s}) = \{h_y(s_i)\}^T \) is the output activation function vector.

The state activation function is:

\[
f_x(a_i, s_i) = \frac{(1-e^{-a_i s_i})}{(1+e^{-a_i s_i})}, \quad a_i \geq 0
\]

(3)

Note \( a_i \) varies dynamically. With a fixed slope \( a > 0 \), the output activation function is:

\[
h_y(s_i) = f_x(a, s_i)
\]

(4)

Note that the \( i^{th} \) 2 \( \times \) 2 block diagonal submatrices of \( \mathbf{W}_x^\zeta \), \( \zeta = U, L \) are constrained to be equal and formulated using angular \( \theta_i \) variables as follows:

\[
\begin{bmatrix}
\cos(\theta_i) & \sin(\theta_i) \\
-\sin(\theta_i) & \cos(\theta_i)
\end{bmatrix}, \quad i = 1, 2, \ldots N_x / 2
\]

(5)

A key motivation for using (5) is that it is an effective mechanism for modeling the underlying oscillatory modes [1], [2]. A sufficient condition for network and training stability is [1],

\[
a_x^\zeta(k) \leq 2, \quad i = 1, 2, \ldots N_x.
\]

(6)

A diagram is shown in Figure 1. ULTRNN architecture with activation function slope modulation.

Eqn. (5) with (6) constrains each eigenvalue pair of \( \mathbf{W}_x^\zeta \) to lie on the unit circle of the complex \( z \)-plane. This ensures both network and learning stability without any need for online monitoring. Such eigenvalue placement eliminates the exploding and vanishing gradients issue associated with
RNN. Constraining the respective 2x2 block diagonals of $W_{x}^{L}$ and $W_{y}^{L}$ to be equal minimizes the possibility of overfitting while learning [2], [19].

Typically, in neural networks, the slope $\alpha$ of the activation function $f_x(a, s)$ is chosen a priori and is always fixed. In this study, we dynamically vary the activation function slopes to ensure that the contribution of some states to others varies with time. With each state variable of the main ULTRNN having a unique activation function, the variation in the activation function slopes over time enhances the dynamic performance of the ULTRNN.

We now describe two alternative architectures for a secondary network that output the variable slope activation functions required by the main network. The first (CSAFN), is a feedforward network that uses the state variables of the main network to compute the activation function slopes [8]. The second (ISAFN), is a novel architecture that employs a recurrent network with its own independent state variables to compute activation function slopes.

$B_{x}^{L} = \{b_{x}^{L,j}, i = 1,2,...N_x, j = 1 ... N_i\}$ are the input matrices, $s_{x}^{L}(k) = \{s_{x,h}^{L}(k), i = 1,2,...N_x\}^T$ are the intermediate linear state vectors, and $g_{x}(s) = g_{x}(s_{h}^{L}) = \{g_{x}(s_{h}^{L}), i = 1,2,...N_x\}^T$, where

$$g_{x}(s) = m_{0} + \frac{2m_{2}}{(1 + e^{-m_{3}(s_1+m_{1})})} \tag{8}$$

during training, the CSAFN simultaneously learns to output $a_{x}^{L}(k)$ as the ULTRNN learns to model dynamic processes. Note that in [8], $W_{x}^{L}$ is chosen as a feedforward, triangular matrix with block diagonals of the form (5). However, such a constraint is not necessary because the CSAFN does not use recurrence.

![FIGURE 2. ECG Common-state AFN architecture](image)

**B. COMMON STATES ACTIVATION FUNCTION NETWORK (CSAFN)**

The state variables are common to the ULTRNN and CSAFN as shown in Figure 2. The output of the CSAFN is the activation function slope $a_{x}^{L}(k)$, given by:

$$a_{x}^{L}(k) = g_{x}(s_{x}^{L}(k)), x = U, L \tag{7}$$

where,

$W_{x}^{L} = \{w_{x}^{L,i, j}, i,j = 1,2,...N_x\}$ are the state weights,

$B_{x}^{L} = \{b_{x}^{L,j}, i = 1,2,...N_x, j = 1 ... N_i\}$ are the input matrices,

$s_{x}^{L}(k) = \{s_{x,h}^{L}(k), i = 1,2,...N_x\}^T$ are the intermediate linear state vectors, and $g_{x}(s) = g_{x}(s_{h}^{L}) = \{g_{x}(s_{h}^{L}), i = 1,2,...N_x\}^T$, where

$$g_{x}(s) = m_{0} + \frac{2m_{2}}{(1 + e^{-m_{3}(s_1+m_{1})})} \tag{8}$$

**C. INDEPENDENT STATES ACTIVATION FUNCTION NETWORK (ISAFN)**

The CSAFN architecture may impact the learning performance by mapping the network state to a positive activation function slope. Such suboptimal mapping compromises network memory. This may impact the generalizability of the CSAFN, and thereby, the network’s robust learning ability. With the primary motivation of decoupling the network states of the AFN from the main network and improving the memory retention control, we propose a novel AFN architecture that employs recurrent states that are completely independent of the main ULTRNN states as shown in Figure 3. The outputs of the independent states AFN (ISAFN) are the activation function slopes $a_{x}^{L}(k)$

![FIGURE 3. Independent-state AFN architecture](image)
used by the main ULTRNN, as shown in Figure 1. The state equations of the ISAFN are given by:

\[
s_\gamma^z(k) = W_\gamma^z z(k) + B_\gamma^z u(k), \quad z(k) = h_\gamma s_\gamma^z(k), \quad \gamma = U, L
\]  

(9)

where,

\[
z(k) = [z_i^\gamma(k), i = 1, 2, \ldots N_z]^T \text{ is the state vector,}
\]

\[
W_\gamma^z = [w_{ij}^\gamma, i, j = 1, 2, \ldots N_z], \quad \text{is the state weight matrix,}
\]

\[
B_\gamma^z = [b_{ij}^\gamma, i = 1, 2, \ldots N_z, j = 1, 2, \ldots N_1] \quad \text{is the input matrix,}
\]

\[
s_\gamma^z(k) = [s_i^\gamma(k), i = 1, 2, \ldots N_z]^T \text{ is the intermediate linear state vector, and}
\]

\[
h_\gamma(s) = [h_s(s_i)]^T, \quad \text{s,} i = 1, 2, \ldots N_z \text{ is the activation function of the form (4).}
\]

Owing to the recurrence used in the ISAFN, the state matrices \(W_\gamma^z\) are constrained to triangular matrices with block diagonal submatrices of form (5). The output of the ISAFN is:

\[
s_\gamma^a(k) = C_\gamma^z z(k), \quad a_\gamma^z(k) = g_\gamma(s_\gamma^z(k)), \quad \gamma = U, L
\]  

(10)

where,

\[
C_\gamma^z = [c_{ij}^\gamma, i = 1, 2, \ldots N_z, j = 1, 2, \ldots N_s] \quad \text{is the output matrix, and}
\]

\[
g_\gamma(s) = [g_s(s_i), i = 1, 2, \ldots N_s]^T \quad \text{is the activation function of the same form as (8).}
\]

Note \(N_z\) does not need to be equal to \(N_s\) and can be selected as higher or lower depending on the task objectives.

The parameters of \(g_\gamma(\cdot)\) of the CSAFN and \(g_\gamma(\cdot)\) of the ISAFN, and how they impact the activation function \(f_\gamma(\cdot)(a, x)\) of the ULTRNN, are shown in Figure 4. The vertical-shift factor \(m_0\) determines the minimum limit of the activation function slope; the span factor \(m_1\) expands or contracts the span and determines the granularity of forgetting levels; and the slope factor \(m_2\) impacts both the range and sensitivity of forgetting levels; the horizontal-shift factor \(m_3\) impacts the desired value of the activation function slope for no external inputs. These parameters are empirically chosen such that (a) the marginal stability of the ULTRNN and its training are retained at all instants with or without external inputs, and (b) the slope assumes a range of values above a minimum threshold to represent various levels of forgetting.

III. TRAINING ALGORITHM

The combined CSAFN and ULTRNN are referred to as CULTN, and the combined ISAFN and ULTRNN are referred to as IULTN. The learning algorithms for the CULTN and IULTN are described in this section. The algorithms were derived from the small perturbation theory to compute the exact gradients. The cost function for minimizing the output error during training is:

\[
J_y = \sum_{k=1}^{N_q} \sum_{i=1}^{N_o} e_{yi}(k)^2
\]  

(11)

where \(e_{yi}(k)\) is the element of the error vector given by

\[
e_{yi}(k) = [e_y^p(k) = y^p(k) - y(k)]
\]  

(12)

where \(y^p(k) = [y^p_i(k), i = 1, 2, \ldots N_o]^T\) is the target vector and, \(N_q\) is the number of training samples in an epoch.

A. TRAINING CULTN

The feedforward architecture of the CSAFN allows training to be performed with a simple extension of the conventional backpropagation algorithm. The following steps are performed for \(k = N_q, \ldots, 1\), at each training cycle \(t = 1, \ldots, N_t\), with \(\gamma = U, L\):

\[
e_x = \begin{cases} e_y^T h_y(k) C_x^\gamma, k = N_q \\ [f_x^a(k) W_x^\gamma + f_x^\gamma(k) g_x^r(k) W_x^\gamma] e_x^\gamma(k) + e_y^T h_y(k) C_x^\gamma, k = N_q - 1, \ldots, 1 \end{cases}
\]  

(13)

where,

\(e_x^\gamma\) is an intermediate error function variable,

\(f_x^a(k) = diag \{f_x^a, i = 1, 2, \ldots N_1\}\), and

\(f_x^\gamma(k) = diag \{f_x^\gamma, i = 1, 2, \ldots N_s\}\)

\(f_x^a\) and \(f_x^\gamma\) are the partial derivatives of \(f_x(a_x^\gamma(k), s_x^\gamma(k))\) with respect to \(a_x^\gamma(\cdot)\) and \(s_x^\gamma(\cdot)\), respectively;

\(g_x^r(k) = diag\{g_x^r(k), i = 1, 2, \ldots N_x\}\),

\(g_x^r\) is the derivative of \(g_x(s_x^\gamma(k))\), and

\(h_y(k) = diag\{h_y, i = 1, 2, \ldots N_o\}\),

\(h_y\) is the derivative of \(h_y(s_y(k))\); and

\(e_x^\gamma(k)\) is updated per (16).

Weight differentials for the ULTRNN are accumulated at each \(k\) iteration step as
\[ \Delta W^c_x = -\varepsilon_x f^c_x(k) x^c(k-1) \]
\[ \Delta B^c_x = -\varepsilon_x f^c_x(k) u^c(k-1) \]
\[ \Delta C^c_x = -\varepsilon_x f^c_x(k) h^c_x(k)x^c(k) \] \hspace{1cm} (14)

Weight differentials for the CSAFN are accumulated at each k iteration step as
\[ \Delta W^c_x = -\varepsilon_x f^c_x(k) g^c_x(k)x^c(k-1) \]
\[ \Delta B^c_x = -\varepsilon_x f^c_x(k) g^c_x(k)u^c(k-1) \] \hspace{1cm} (15)

Note that the symbol \(\varepsilon\) in (14) and (15) represents the negative accumulation of the weight differentials at the kth iteration step to the accumulated values from the previous steps \(N_q\) through \(k + 1\). Before the next iteration step, the following update is performed
\[ \varepsilon_x^c(k) = \varepsilon_x^c. \] \hspace{1cm} (16)

After the accumulation of the differentials over all sample steps, the differential of the angular variable of each diagonal block of \(W^c_x\) is computed as [2]
\[ \Delta \theta^c_i = \cos \theta^c_i \left\{ \Delta w^u_{x,i+1,i} - \Delta w^u_{x,i,i+1} + \Delta w^l_{x,i,i+1} - \Delta w^l_{x,i+1,i} \right\} \]
\[ - \sin \theta^c_i \left\{ \Delta w^u_{x,i,i+1} + \Delta w^u_{x,i+1,i} + \Delta w^l_{x,i,i+1} + \Delta w^l_{x,i+1,i} \right\} \]
\[ i = 2,4, \ldots N_x. \] \hspace{1cm} (17)

(17) constrains the block diagonal elements of \(W^c_x\) to lie on the unit circle and helps maintain network stability and improve robust learning ability. The possibility of exploding and vanishing gradients is minimized while retaining the sensitivity required to model the oscillatory modes of the target.

B. TRAINING ULTRNN

The interaction of the recurrent states of the ISAFN with the recurrent states of the ULTRNN necessitates two parallel training processes, one for the ULTRNN and the other for the ISAFN.

1) BACKPROPAGATION ALGORITHM FOR THE ULTRNN

The following steps are performed for \(k = N_q, \ldots 1\) at each training cycle \(t = 1,2, \ldots N_t\) with \(\zeta = U, L\)
\[ \varepsilon_{zo}^c = \begin{cases} 
\varepsilon_x^c(k), & k = k \in \{1,2, \ldots 1\} \\
\varepsilon_{zo}^c(k), & k = k \in \{1,2, \ldots 1\} 
\end{cases} \]
\[ E_{zo}^c = \begin{cases} 
E_x^c(k), & k = k \in \{1,2, \ldots 1\} \\
E_{zo}^c(k), & k = k \in \{1,2, \ldots 1\} 
\end{cases} \] \hspace{1cm} (18)

where \(\varepsilon_{zo}^c\) is the intermediate error function variable. Weight differentials are accumulated at each \(k\)th iteration step as shown in (14) using \(\varepsilon_{zo}^c\) of (18). Before the next \(k\)th iteration step, the following update is performed:
\[ \varepsilon_x^c(k) = \varepsilon_x^c. \] \hspace{1cm} (19)

2) ZIGZAG PROPAGATION ALGORITHM FOR ISAFN

\(x^c(k)\) being a function of two variables is impacted by the ISAFN weights through two propagation paths. One is directly through \(a^c_x(k)\) of the ISAFN and the other is indirectly through \(\varepsilon^c(k)\) of the ULTRNN, which is in turn is impacted by \(a^c_x(k-1)\) and \(W^c_x\). Hence, the ISAFN gradient terms are computed through a combination of forward and backward propagation steps as summarized below.

The following steps are performed for \(k = 1,2, \ldots N_q\) at each training cycle \(t = 1,2, \ldots N_t\) with \(\zeta = U, L\)
\[ E_x^c(k) = f^c_x(k)g^c_x(k) \]
\[ E_{zo}^c(k) = e_y(k)^T h^c_y(k)C^c_x C^c_z(k) \] \hspace{1cm} (20)

where,
\[ E_x^c(k)\] and \(E_{zo}^c(k)\) are the intermediate error function variables,
\[ g^c_x(k) = diag \{g^c_{z_i}(k), i = 1,2, \ldots N_z\}, \text{ where } g^c_{z_i}(k) \text{ is the derivative of } g^c_{z_i}(s^c_x). \]
\[ h^c_y(k) = diag \{h^c_{z_i}(k), i = 1,2, \ldots N_z\}, \text{ where } h^c_{z_i}(k) \text{ is the derivative of } h^c_{z_i}(s^c_x). \]

Note that the interaction of the ISAFN and ULTRNN states is reflected in \(E_x^c(k)\).

For each \(k\)th iteration step above, the following steps are performed for \(k = k, k-1, \ldots 1\):
\[ \varepsilon_{zo}^c = \begin{cases} 
\varepsilon_x^c(k), & k = k \in \{1,2, \ldots 1\} \\
\varepsilon_{zo}^c(k), & k = k \in \{1,2, \ldots 1\} 
\end{cases} \] \hspace{1cm} (21)
\[ E_{zo}^c = \begin{cases} 
E_x^c(k), & k = k \in \{1,2, \ldots 1\} \\
E_{zo}^c(k), & k = k \in \{1,2, \ldots 1\} 
\end{cases} \] \hspace{1cm} (22)

Weight differentials are accumulated at each \(k\) iteration step as
\[ \Delta W^c_x = -\varepsilon_{zo}^c f^c_x(k) x^c(k-1) \]
\[ \Delta B^c_x = -\varepsilon_{zo}^c f^c_x(k) u^c(k-1) \]
\[ \Delta C^c_x = -\varepsilon_{zo}^c f^c_x(k) h^c_x(k)x^c(k) \] \hspace{1cm} (23)

Before the next \(k\)th step, respectively, the following updates are performed:
\[ \varepsilon_x^c(k) = \varepsilon_x^c. \] \hspace{1cm} (24)
\[ E_x^c(k) = E_{zo}^c. \]
box is obtained through post-multiplication of the term in the box above with the ISAFN backpropagation term $W^i_h k_x(\cdot)$. The process continues in the next column with the error in the top box obtained by pre-multiplying the error in the previous step $(k - 1)$ with the ULTRNN forward propagation term $f^i_x(\cdot)W^i_x$. This process is repeated recursively for all columns in the $k$th iteration step, and for all $k = 1, 2, \ldots N_q$.

3) WEIGHT UPDATES

The CULTN or IULTN weights are updated at each training cycle consisting of an epoch of $K_q$ time steps, using

$$\Phi(t + 1) = \Phi(t) - \frac{\eta_w}{K_q} \Delta \Phi(t), \mu > 0$$  \hspace{1cm} (25)

$\Phi(t)$ represents the weights in training cycle $t$. Note that updating $W^C_x$ and $W^I_x$ is subject to the form of (17).

4) COMPUTATIONAL REQUIREMENTS

The number of trainable weights for the CULTN is given by:

$$N^C_w = N_x(3N_x + 4N_i + 2N_o)$$  \hspace{1cm} (26)

and for the IULTN is given by

$$N^I_w = N_x(N_x + 2(N_i + N_o)) + N_2(N_2 + 2(N_i + N_o))$$  \hspace{1cm} (27)

For a given network size, the $z$-prop used by the ISAFN requires $O(N_q^2/2)$ computations whereas the CSAFN requires only $O(N_q)$ computations. Hence, the ISAFN can be computationally expensive for a large $N_q$. Truncating the number of $k$ recursions for a large $k$ to a fraction of $N_q$ trades off the accuracy of the gradients with a reduction in computational speed. For example, $k$ iterations can be limited to a threshold $k_{th}$, that can be selected empirically.

To empirically assess the impact of recursion truncation on the gradient accuracy and computation speed, a representative 2-output, 5/5-mode ($N_x = N_o = 10$) IULTN was considered. The inputs and outputs were chosen as random values in the range of $[-1:1]$. The gradients were computed first with the $z$-prop algorithm and then with the direct perturbation of the individual weight elements. Errors between the gradients were compiled for several truncation thresholds as a fraction of $N_q$ in the range $(0.1:1)$. The error comparison was repeated for several $N_q$ values in the range $(50:200)$. The normalized gradient accuracy based on the root-mean-squared (RMS) error versus the computational time profiles is summarized in Figure 6. The gradient error is close to zero with no truncation ($k_{th} = N_q$), which confirms the correctness of the $z$-prop equations (20) – (24). The gradient errors exhibited a hyperbolic profile with a $k_{th}$ knee point between $0.2N_q$ and $0.35N_q$. The lower the epoch size $N_q$, the higher the increase in error with increased truncation. The computational time exponentially increased as the truncation decreased to zero, with a worse impact for the higher epoch size cases. A threshold chosen close to the knee point provides a reasonable trade-off between the gradient accuracy and computation time.

![Zigzag propagation algorithm for ISAFN: Representative iteration steps](image)

**FIGURE 5.** Zigzag propagation algorithm for ISAFN: Representative iteration steps

![Impact of z-prop truncation depth on error and computation burden](image)

**FIGURE 6.** Impact of z-prop truncation depth on error and computation burden

III. CLOSED-LOOP GENERATIVE TRAINING

For autonomous generation using only the network’s predicted outputs, the question of how to ensure closed-loop
stability of the network needs to be considered. A tractable metric of the closed-loop system performance would be ideal to steer the training, such that the trained network behaves as a nonlinear oscillator with an output that neither collapses nor saturates but continues to emulate the target. However, owing to the nonlinearity of the activation functions and the variation in their slopes, such an ideal metric for guaranteed closed-loop performance is not computationally feasible. Instead, we seek a simple metric based on the linear control system theory that can potentially improve closed-loop performance. A heuristic metric based on the sufficiency condition for the local marginal stability of the linearized closed-loop system is derived. The closed-loop state transition equation of the ULTRNN with one-step prediction feedback for small perturbations is:

\[
\begin{bmatrix}
\Delta x^U(k + 1) \\
\Delta x^L(k + 1)
\end{bmatrix} =
\begin{bmatrix}
0 & I \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
W_{xCL}(k) \\
W_{xCL}^*(k)
\end{bmatrix}
\begin{bmatrix}
\Delta x^U(k) \\
\Delta x^L(k)
\end{bmatrix}
\]

(28)

where,

\[W_{xCL}^*(k) = \begin{bmatrix}
W^x_U + B^x_U \hat{h}_x^U(k) C^x_U & B^x_U \hat{h}_x^U(k) C^x_L \\
B^x_L \hat{h}_x^U(k) C^x_U & W^x_L + B^x_L \hat{h}_x^U(k) C^x_L
\end{bmatrix}
\]

(29)

In (4), with slope \(\alpha = 2\) for the output activation function \(\hat{h}_x^U(k)\), derivative \(\hat{h}_x^U(k)\) is a vector of positive values with a maximum of unity. In (29), we set \(\hat{h}_x^U(k)\) to unity to obtain a linearized closed-loop matrix (LCLM)

\[W_{xCL}^*(k) = \begin{bmatrix}
W^x_U + B^x_U C^x_U & B^x_U C^x_L \\
B^x_L C^x_U & W^x_L + B^x_L C^x_L
\end{bmatrix}
\]

(30)

and set \(\hat{h}_x^U(k)\) to zero to obtain the open-loop representation

\[W_{xOL}^*(k) = \begin{bmatrix}
W^x_U & 0 \\
0 & W^x_L
\end{bmatrix}
\]

(31)

Ensuring the marginal stability of the closed-loop system for small perturbations requires constraining the eigenvalues of \(W_{xCL}^*(k)\) within an annular ring to approximately unity in the complex z-plane. Because the elements of \(\hat{h}_x^U(k)\) vary from zero to unity, \(W_{xCL}^*(k)\) transitions within the ranges \(W_{xCL}^U\) and \(W_{xCL}^L\). Given that \(W_{xCL}^L\) is marginally stable due to constraint (5) imposed on its block diagonal elements, it is heuristically evident that imposing marginal stability on \(W_{xCL}^U\) in turn, ensures the marginal stability of \(W_{xCL}^*(k)\) for all values of \(\hat{h}_x^U(k)\). Note that in (28) as \(k\) varies, modulated by the AFN, \(f^U_x(\xi, k)\), \(\xi = U, L\) shapes the closed-loop system performance by dynamically adjusting the annular ring that constrains the eigenvalues.

The cost function (11) used for training is modified as

\[J_{\alpha} = J_y + J_{\lambda}
\]

(32)

to include the cost term \(J_{\lambda}\) which represents the effective distance of the eigenvalues of \(W_{xCL}^*(k)\) from unity.

\[J_{\lambda} = \frac{1}{2N_x} \sum_{i=1}^{2N_x} \| \lambda_i(W_{xCL}^*) \| - 1 \]

(33)

where, \(\lambda_i(W)\) is the eigenvalue of \(W\), and \(\| \cdot \|\) represents any scalar norm. The gradients of \(J_{\lambda}\) with reference to the individual elements of \(W_{xCL}^U\) are computed through direct perturbation at each weight update. Note that echo-state networks [20] employ eigenvalues in the training process to eliminate exploding gradients. The gradients of \(J_{\lambda}\) require \(O(N_x^4)\) computations and can be computationally expensive for large networks. The computational speed can be increased at the cost of accuracy using an alternative cost function based on a rough estimate of the spectral radius, that is, the largest eigenvalue, of \(W_{xCL}^U\) can be used [19]. Note that (29) applies only to equal input and output dimensions, and if other additional inputs or outputs are present, they should be separated from \(B^x_U, C^x_L, \xi = U, L\).

IV. ILLUSTRATIVE EXAMPLES

Several illustrative examples are presented to demonstrate and compare the effectiveness of the IULTN architecture with state-of-the-art networks for generative tasks. The first two examples pertain to the autonomous generation of chaotic waveforms with characteristics close to those of Lorenz limit cycles [11], and Santa Fe laser data [12]. The third example deals with the autonomous generation of a complex periodic waveform based on the South Indian Kolam patterns [13]. The fourth example autonomously generates synthetic electrocardiogram (ECG) data with characteristics similar to real patient data [14], [15]. The fifth example models a nonstationary process using the Google stock data studied in [16]. The final example generates hourly solar generation and energy consumption data studied in [18] as part of smart grid data synthesis. In each example, the network was trained for one-step prediction using the cost function (32) and tested in a closed loop with the output from the current time step used to predict the next output. Techniques adopted to improve the learning robustness include [8]: (a) input noise injection to improve generalizability [21], [7], (b) use of additional pilot inputs to embed periodicity, (c) use of a composite input consisting of the target and the network outputs for guided training, and (d) combined use of n-step prediction outputs for improved noise rejection during generation.

For the chaotic time series considered in the first two examples, the Kullback-Leibler (KL) divergence [22] was used for performance comparison. The KL divergence is not a measure of the global closeness of the network output to that of the target. However, at the local level, it is useful to quantitatively compare the probability distribution of outputs from different networks. We use the following formulation to facilitate a performance comparison of CULTN and
IULTN: A covariance phase plot for each of the target and network outputs is created with coordinates \(\{y_i^p (k), y_i^p (k + N_y)\}\) and \(\{y_i (k), y_i (k + N_y)\}\), \(i = 1,2, ..., N_y\), \(k = 1,2, ..., N_d\), respectively, where \(N_y\) is the data length for an arbitrarily chosen covariance time step \(N_\tau\). Each point on the plot is quantized into \(\{y_m^p, y_m^s\}\), \(m, n = 1,2, ..., N_p\) using a quantization level \(N_p\). The probability distribution of the target outputs \(P_i^p (y_m^p, y_n^p)\) and the network outputs \(P_i (y_m^p, y_n^p)\), \(i = 1,2, ..., N_o\), \(m, n = 1,2, ..., N_r\), are computed as the joint probability of the occurrence of \(y_m^p\) and \(y_n^p\) over all time instances \(k = 1,2, ..., N_d\). The symmetric KL divergence is computed as

\[
D_I = 0.5 \left\{ \sum_{m=1}^{N_r} \sum_{n=1}^{N_p} P_i (y_m^p, y_n^p) \ln \left( \frac{P_i^p (y_m^p, y_n^p)}{P_i (y_m^p, y_n^p)} \right) + P_i^p (y_m^p, y_n^p) \ln \left( \frac{P_i (y_m^p, y_n^p)}{P_i^p (y_m^p, y_n^p)} \right) \right\}
\]  

(34)

The parameters for the comparative analysis were the covariance time step \(N_\tau\) and quantization range \(N_r\).

For all the examples, minimizing the cost function (32) was used to ensure closed-loop stability and robust learning. Typically, training is conducted in two phases to improve learning robustness. The first phase uses only the target waveform as the input, and the second combines the target waveform \(y_i^p (k)\) and predicted network output \(y_i (k)\):

\[
u_i (k + 1) = \gamma y_i^p (k) + (1 - \gamma) y_i (k), \quad i = 1,2, ..., N_o, 0 \leq \gamma \leq 1
\]  

(35)

where \(\gamma\) is chosen empirically. The training transition from the first to the second phase occurs when the eigenvalue metric \(J_\lambda\) decreases below a threshold. During the second training phase, the weight update parameter \(\eta_w\) was progressively reduced to ensure \(J_\lambda\) was maintained at or below the threshold.

To generate complex periodic waveforms, the waveform period was set to the epoch length \(N_p\). To reduce the initial transient errors, the initial values of the network state variables in each training cycle \(\tau\) are updated as

\[
x_i (1)_{\tau + 1} = \left( \beta x_i (1) + (1 - \beta) x_i (N_q) \right)_{\tau}, 0 \leq \beta \leq 1
\]  

(36)

with \(\beta\) chosen empirically. Note, \(x_i\) represents the network’s state variables.

For all networks used in the examples, the parameters \(m_0\), \(m_1 m_2\), and \(m_3\) of \(g_k(\cdot)\) used in the CSAFN and \(g_k(\cdot)\) used in the ISAFN per (8) are empirically chosen as 0.1, 2.1, 2.1, and 1.5, respectively. This selection limits the activation function slope to a maximum of 2.2 aimed at sustained long-term memory, and a minimum of 0.1 heuristically chosen to regulate the forgetting level. The initial angular variables \(\theta_i\) of the block diagonals of the ULTRNN and ISAFN were assigned random values that were uniformly distributed in the range \((0:2\pi)\). The elements of all other matrices were randomly distributed in the range \((-1:1)\). For input-output compatibility, before training, the inputs and targets were amplitude-normalized by scaling to span the range \((-0.85:0.85)\). The initial value of the weight update parameter \(\eta_w\) was set to 0.1.

Tables I and II summarize the network parameters for the CULTN and IULTN, respectively, and Table III lists the key training and testing parameters used in the examples.

### A. LORENZ LIMIT CYCLE

We consider the autonomous generation of time-series with characteristics close to the Lorenz limit cycles [11]. The target outputs are produced by solving the ordinary differential equation (ODE):

\[
\dot{x}_1 = \sigma (x_2 - x_1) \\
\dot{x}_2 = r x_1 - x_2 - x_1 x_3 \\
\dot{x}_3 = x_1 x_2 - b x_3
\]

where \(\sigma = 10, r_p = 28\), and \(\beta = 8/3\). The ODE is solved with initial values set at \(y_{10}^p = -10, y_{20}^p = -8, y_{30}^p = 30\),

\[
\frac{d}{dt}[y_1^p, y_2^p, y_3^p] = [\sigma_p (y_2^p - y_1^p),\ y_1^p y_3^p - y_2^p - \beta_p y_3^p,\ y_1^p y_2^p - \beta_p y_2^p]
\]  

(37)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170042, IEEE Access

---

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
using a fourth-order Runge Kutta with a step size of 0.001 to generate the targets $y_i^p$, $i = 1, 2, 3$. The trajectories after amplitude normalization are shown in Figure 7.

Figure 7. Lorenz limit cycles: target trajectories

With 3-inputs and 3-outputs, a 6-mode ($N_x=12$) CULTN, 4/4-mode ($N_x=N_z=8$) IULTN, and 6/6-mode ($N_x=N_z=12$) IULTN were considered. The networks were trained with 10000 points of the data generated by (37), with an epoch size $N_q = 100$. White noise with a variance of 0.016 was injected into the input. The network was first trained in an open loop and then in a closed loop with $\gamma$ in (35) reduced from 1 to 0.5 to 0.25 such that the network was forced to rely more on its prediction to improve the orbit switching accuracy. The generative performance of the network was assessed by replacing the targets $y_i^p(k)$ with the corresponding network outputs $y_i(k)$, $i = 1, 2, 3$. The performance of the 6-mode CULTN closely matched that of the 4/4-mode IULTN, but was inferior to that of the 6/6-mode IULTN. Hence, an 8-mode ($N_x=16$) CULTN was considered, and its performance was comparable to that of 6/6-mode IULTN. From Figure 8, it can be observed that the LCLM eigenvalues for the two networks are distributed around the unit circle.

Figure 8. Lorenz limit cycles: LCLM eigenvalue placement in the complex z-plane

Figure 9 shows the autonomously generated outputs of the 8-mode CULTN and the 6/6-mode IULTN. Qualitatively it can be seen that both networks can generate and sustain chaotic waveforms with attractor contours similar to that of the target. The two basins of attraction are distinguishable, and orbit switching between them is retained. Quantitative performance was assessed using symmetric KL divergences (33) for multiple ensembles of autonomous generation. The ensembles are computed for quantization levels $N_r$ in the range (4:44), by setting (a) the time instant for transfer from open-loop to closed-loop in the range (500:2500) and (b) the covariance parameters $N_c$ in the range (20:200). Figure 10 shows the ensemble average and standard deviation of the KL divergences for the 8-mode CULTN and 4/4 mode, and 6/6-mode IULTN. The average KL divergence of the 6/6-mode IULTN is consistently lower than that of the 8-mode CULTN with a lower standard deviation. Even the 4/4-mode IULTN exhibited marginally better performance than the 8-mode CULTN.

In summary, IULTN with fewer weights ($N_w=792$) outperformed CULTN ($N_w=1056$) both qualitatively and quantitatively in the generative modeling of the Lorenz limit cycles. A similar task considered in [23] used a reservoir computing network with an estimated 1800 trainable weights.
and then in a closed loop with $y$ in (35) reduced from 1 to 0.5 such that the network relied more on its prediction. Figure 12 shows that the LCLM eigenvalues of the two networks were distributed within an annular ring around the unit circle.

The generative performance of the trained networks was assessed by replacing the target $y^p(k)$ with the mean of the predicted network outputs $y(k)$ and $y(k-4)$. The performance of the 4-mode CULTN is significantly inferior to that of the 4/4-mode IULTN. Hence, an 8-mode CULTN was considered to have the performance comparable to that of a 4/4 mode IULTN. Figure 13 shows the CULTN and IULTN output time plots for 2000 sampling instants. The first 200 points show the network output with the target input, and the remaining autonomously generated. It is qualitatively evident that both networks can sustain high-frequency oscillations and chaotic low-frequency variations similar to the original laser data. However, the CULTN waveform has 'softer' valleys with higher values at which the envelope collapses compared to that of the IULTN. The IULTN output was more similar to the target (Figure 11) with a sharper rise and lower valleys of the envelope.

**FIGURE 10.** Generating Lorenz limit cycles: Ensemble average and standard deviation of KL divergences vs. quantization level (top: first output, middle: second output, bottom: third output); gray line: 4/4 mode IULTN, dash line: 6/6-mode IULTN, dark thin line: 8-mode CULTN

**FIGURE 11.** Santa Fe laser data: target waveform

**B. SANTA FE LASER DATA**

We considered autonomously generating synthetic data similar to the Santa Fe Laser dataset [12]. The original dataset consists of periodic to chaotic intensity pulsations of a far-infrared laser. The laser output exhibited significant dissymmetry around the mean value and mimicked a chaotic low-frequency waveform modulating a high-frequency carrier as shown in Figure 11.

First, a 4-mode ($N_x=8$) CULTN and a 4/4-mode ($N_x=N_y=8$) IULTN were considered. With the laser data as the target input, a 1-input 2-output network was used with the outputs being the 1-step and 4-step prediction values. To improve the learning robustness, a second output is included to enhance the network's ability to capture the structure of the laser data better, for example, its rise and collapse. In addition, white noise with a variance of 0.01 is added to the input. The networks were trained with the first 1500 data points of the original data series "A.Cont," with an epoch size of $N_q = 100$. The networks were first trained in an open loop

**FIGURE 12.** Santa Fe laser data: LCLM eigenvalue placement in the complex $z$-plane

Figure 14 compares the representative covariance phase plots of the autonomously generated outputs with those of the target. The phase plots describe $y(k+50)$ vs. $y(k)$ where the time step 50 is arbitrarily chosen to represent the fractional period of the dominant low-frequency envelope of the target. It can be seen that the contour edges of the phase plots of the 4-mode CULTN are fuzzier than those of the target. The contour edges of the 8-mode CULTN phase plots are sharper and are similar to the target. The phase plots of the 4-mode IULTN are the closest to the target, including a distinct eye-like feature.

Quantitative performance was assessed using symmetric KL divergences (33) for multiple ensembles of autonomous generation. The ensembles were computed for quantization levels $N_r$ in the range (4:44) and setting the covariance
parameters $N_c$ in the range (20:200). Figure 15 shows the ensemble average and standard deviation of the KL divergences as a function of the quantization level $N_q$. The average KL divergence of the 4/4-mode IULTN was consistently lower with a smaller standard deviation than the 4-mode CULTN and closer to that of the 8-mode CULTN.

In summary, IULTN with far fewer weights ($N_w^I = 320$) outperformed CULTN ($N_w^C = 896$) in the generative modeling of Santa Fe Laser data. A similar task considered in [10] with a multilayer perceptron network used an estimated 1300 trainable weights.

C. KOLAM PATTERN

Kolam is a floor art of complex and intricate patterns that is commonly practiced in South India. A Kolam is drawn as a continuous line looping or joining straight and curved lines typically around symmetric dot patterns [13]. The mathematical properties implied in Kolam design have been studied in [13], [24] and [25]. A single-stroke Kolam variant can be represented as a trajectory plot with x and y coordinates. The modeling complexity of a single-stroke Kolam lies in the rich harmonic content of the x- and y-variables, and their spatiotemporal synchronization.

In summary, IULTN with far fewer weights ($N_w^I = 320$) outperformed CULTN ($N_w^C = 896$) in the generative modeling of Santa Fe Laser data. A similar task considered in [10] with a multilayer perceptron network used an estimated 1300 trainable weights.

Case 1 compares the performance of CULTN and IULTN in the autonomous generation of a single-stroke Kolam with 4-4-4-4 dot placements, as shown in Figure 16. A 2-input, 2-output, 6-mode ($N_x=12$) CULTN, and 5/5-mode ($N_w^C=576$) IULTN were trained to perform 1-step prediction with the target being the Kolam’s x and y coordinates. The kolam pattern was periodically normalized with $N_q = 100$. Given that the kolam pattern is periodic, updates to the initial values of the network states for each training epoch used $\beta = 0.95$ in (36). The network was first trained in an open loop and then in a closed loop with $\gamma$ in (35) reduced from 1 to 0.5, to improve the replication accuracy. The Kolam plots autonomously generated by CULTN and IULTN are shown in Figure 16. Both CULTN and IULTN can stably and continuously generate a Kolam pattern. However, the 5/5-mode IULTN, with fewer weights ($N_w^I = 520$), outperforms the 6-mode CULTN ($N_w^C = 576$) in terms of accuracy and symmetry around the dot pattern, as confirmed by the average and standard deviation of the errors. Errors were obtained by comparing the distances between the generated pattern and target to the nearest dot.

Case 2 shown in Figure 17 is the autonomously generated plot by a set of nine 6/6-mode ($N_x=N_y=12$) IULTNs, each trained to reproduce one of the symmetric one-stroke kolams

FIGURE 13. Santa Fe laser data: autonomously generated network outputs (top: 8-mode CULTN, bottom: 4/4 mode IULTN)

FIGURE 14. Santa Fe laser data: covariance phase plots of original laser data (top left) and autonomously generated outputs (top right: 4-mode CULTN, bottom left: 8-mode CULTN, bottom right: 4/4-mode IULTN)

FIGURE 15. Santa Fe laser data: KL divergences of autonomously generated outputs

FIGURE 16. Santa Fe laser data: 1-step prediction for 1 single-stroke Kolam with 4-4-4-4 dot placements
drawn around 1-3-5-3-1 dot placements studied in [13]. The low average errors and low standard deviations confirm the quantitative accuracy achieved by IULTN.

The autonomously generated outputs, and LCLM eigenvalue distribution of a 4-mode CULTN ($N_z=8$) are shown in Figure 19. Note that the pilot sine wave was also generated by the networks; hence the process was fully autonomous. The CULTN continuously reproduces the subperiod, shape, and peak values of the P, R, and T waves, and heartbeat with reasonable accuracy. However, it cannot not accurately reproduce flat segments.

**FIGURE 16.** Kolam with 4-4-4-4 dot pattern: target waveform (left), autonomously generated output with 6-mode CULTN (middle), 5/5-mode IULTN (right). ($\mu$: error average; $\sigma$: error standard deviation).

**FIGURE 17.** Kolam with 1-3-5-3-1 dot pattern: autonomously generated outputs with 6/6-mode IULTNs Fe laser data: KL divergences of autonomously generated outputs. ($\mu$: error average; $\sigma$: error standard deviation).

**FIGURE 18.** ECG: target (dark) and pilot (grey) waveforms

**FIGURE 19.** ECG: autonomously generated outputs (left) by 4-mode CULTN (ECG output (dark) and pilot (grey); LCLM eigenvalue placement in the complex z-plane (right))

The autonomously generated outputs, and the associated LCLM eigenvalue distributions, of a 2/2-mode ($N_z=N_x=4$), 3/3 mode ($N_z=N_x=6$), 3/4 mode ($N_z=6$, $N_x=8$) IULTNs are shown in Figure 20. It can be observed that the 2/2 IULTN, with fewer weights ($N_{w}^{I}=112$), outperformed the 4/4 mode CULTN ($N_{w}^{C}=288$), with P and T waves better reproduced. The reproduction of the flat segment between the T and P waves improved with the 3/3 mode IULTN. The ECG waveform reproduction was significantly improved with the 3/4 mode IULTN, and with approximately the same number of weights ($N_{w}^{I}=276$), it significantly outperforms the 4-mode CULTN ($N_{w}^{C}=288$). Note that the additional dimensional freedom of the 3/4 mode IULTN, with the 3-mode ULTRNN ($N_z=6$) and 4-mode ISAFN ($N_z=8$), results in a better reproduction of the valleys in the Q and S waves. Relaxing the dimensional constraint may have resulted in a more optimal activation function slope dynamic, thus improving the network memory for modeling.

Case 2 considered the generation of clones with characteristics close to the ECG of a general population using a single network. Similar tasks using generative adversarial networks (GANs) were studied in [26], [27]. The synthesized generation of clone waveforms is useful for data augmentation in biosignal classification, clinical training,
and data analysis tasks where real field data are scarce and difficult to collect. In addition, synthetic data are useful for anonymizing and alleviating patient privacy concerns [26]. The dataset [15], it was noted that ECGs with elevated T waves are normal variants and form a small fraction (<5%) of over 4000 patients with no heart ailments. Such cases may not be well represented in data-modeling tasks, and their synthetic generation may be useful for increasing the data diversity. We consider a 3-input, 2-output, 12/12-mode ($N_x=N_y=24$) IULTN for the autonomous generation of ECG clones with elevated T waves. We used the 50 samples drawn from [15] as the training set. Each training sample was period normalized with $N_t = 100$. While the first and second inputs are the target ECG and pilot sine wave, the third input is a random seed drawn from a Gaussian noise generator and is associated with each target ECG. The outputs were one-step predicted values of the target and the pilot sine wave. The network was first trained in the open loop, and then in the closed loop with $γ$ gradually reduced to 0.5. During closed-loop training, only the first two inputs use output feedback, and the noise seed is retained as the external third input.

![Figure 20](image1.png)

**FIGURE 20.** ECG: autonomously generated IULTN outputs (left) by 2/2-mode (top), 3/3-mode (middle), ¾-mode (bottom) - ECG output (dark) and pilot (grey), and LCLM eigenvalue placement in the complex z-plane (right)

Figure 21 shows the representative target ECGs, along with the corresponding autonomously generated outputs when fed with the training seeds. In addition, the test outputs were autonomously generated using new Gaussian noise inputs that were different from the training seeds. The generative capability of IULTN is evident from cloned outputs that have structural similarity to the training set, while exhibiting natural diversity. This example demonstrates the ability of the IULTN to autonomously generate complex biosignals with characteristics similar to those of a parent population. Reference [26] considered an LSTM-based deep GAN with 200 hidden units and three layers for an estimated 800000 trainable weights for each of the generator and discriminator networks. Comparable results were obtained with IULTN, which uses only 2880 trainable weights, a minuscule fraction (0.2%) of LSTM-GAN [26].

![Figure 21](image2.png)

**FIGURE 21.** ECG with elevated-T segments: target waveforms (left), autonomously generated outputs by 12/12-mode IULTN with training seed (middle) and with random seed (right); x-axis: time steps; y-axis: target (left) and network outputs (middle and right)

### E. STOCK DATA MODELING

We consider synthetically generating nonstationary time-series data with a probability distribution that best approximates that of the parent. We specifically modeled the distribution of Google’s stock data previously studied in [16] which used TimeGAN. TimeGAN uses an embedding subnetwork and a recovery subnetwork to learn the underlying temporal dynamics, in addition to the discriminator and generator subnetworks common to a generic GAN. The training process has three stages: the first for the embedding subnetwork, the second for the full network, but with only supervised loss, and the third for a joint process considering all losses. TimeGAN exhibits the best performance among competing techniques for a range of time-series generative tasks [16].

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
The stock dataset consists of six variables: open, low, high, closing, and adjusted closing stock prices, and the stock trading volume for each trading day for the years 2004-2019. The objective of this exercise is to synthetically generate stock data over a 48-day period that mimics the Google stock data distribution. Stock data exhibit a high feature correlation among the five pricing variables. A 48-day period is considered here, as the temporal correlation for the 24-day periods studied in [16] is high, but significantly reduced for 48-day periods. We trained a 7-input 6-output 10/10 mode (N_T=N_v=10) IULTN to perform generation through a one-step prediction. The first six inputs are the stock data variables. To improve learning robustness, the seventh is a Gaussian noise vector that better maps the temporal variability in any period. The network was first trained in the open loop and then in the closed loop, with γ in (35) progressively reduced to 0.25. The network was tested for autonomous generation of stock data using Gaussian noise as the external input. TimeGANs with four, six, and eight gated recurrent units (GRUs) for each of the six stock variables, similar to the one studied in [16] were trained, and the one with the eight GRUs that had the best performance was chosen for comparison with the IULTN. The key comparative items of the network architecture, its dimensions, and the computational burden are listed in Table IV. As shown in the table, the number of trainable parameters for IULTN (2400) is a small fraction (~1.4%) of that of TimeGAN (174727). This is reflected in the substantially low computation time per epoch iteration for IULTN training (~15.3% of TimeGAN). The number of generative parameters required for the trained IULTN remains at 2400 and is still a small fraction (~6.2%) of the trained TimeGAN (38400).

### TABLE IV

| Category               | TimeGAN | IULTN |
|------------------------|---------|-------|
| Training parameters    | 174727  | 2400  |
| Generating parameters  | 38400   | 2400  |
| The relative computational time for training | 6.55 | 1.0 (Normalization base) |

Figure 22 shows a qualitative comparison of the IULTN-generated data with that of TimeGAN. The visualization plots compare the principal component analysis (PCA) [28] and t-distributed stochastic neighbor embedding (t-SNE) [29] maps of the synthetically generated data with those of the original data. It is apparent that the distribution of the IULTN-generated data closely follows that of the original data, and is comparable to that of the TimeGAN-generated data.

A quantitative comparison was performed using the train on the synthetic test on real (TSTR) technique using generic assessment networks based on GRU [16]. The first assessment network was trained to predict the closing stock price using training data extracted from the synthetic generation, with the trading volume data discarded. Next, the second assessment network with the same structure and dimensions as the first was trained using the training data extracted from the original data. The prediction performance of both networks was tested using fresh test data extracted from the original data. The mean absolute errors (MAE) of the predictions obtained using the two assessment networks collected over 200 train and test runs were used as comparison metric. The prediction MAE densities obtained for the TimeGAN- and IULTN-generated data are shown in Figure 23. It is clear from the figure that the prediction MAE with IULTN-generated data is closer to that obtained with the original data than that of TimeGAN. Specifically, the peak MAE of the IULTN case (0.0135) occurred in close proximity to that corresponding to the original data (0.0125). However, an accurate objective comparison with TimeGAN is not possible because the network dimensions and training are not fully optimized in either case.
In summary, the generative performance of IULTN compares well with that of TimeGAN both qualitatively and quantitatively with far fewer weights, a simpler training process, and lower computational times.

F. SMART GRID DATA MODELING

The objective of this exercise, similar to the task addressed in [18], is to synthetically generate energy consumption and solar generation data with the same probabilistic distribution as that of the Pecan Street Dataset [17]. We observed that the solar generation data for multiple users with installed PV panels exhibited repetitive patterns with variations owing to the onset of clouds and rain. Energy consumption data also exhibit similar trends but are accompanied by irregular spikes because of random consumption. The training data comprised hourly data of nine users in Austin collected over a year. We trained a 23-input 22-output 15/15 mode ($N_x=N_z=15$) IULTN to perform autonomous generation through one-step prediction. The first 18 inputs and outputs correspond to the solar generation and energy consumption data of nine users. The four inputs/outputs were the pilot sine waves to represent the periodicity of the daily, weekly, and annual variations. A Gaussian noise seed is used as the final input. The autonomous generation capability of the network was tested using pilot waves and a Gaussian noise seed as the only external inputs. Figure 24 shows the IULTN-generated solar generation and energy consumption data over a one-year period for a representative user, compared with the corresponding original data.

Figure 25 shows a qualitative assessment of the IULTN-generated data using visualization plots based on PCA [28] and t-SNE [29] maps. It is apparent that the distribution of IULTN-generated data closely follows that of the original data.

A quantitative comparison was performed using the TSTR technique with assessment networks based on GRU [16], similar to that one considered in Section E for stock data modeling. The two assessment networks were trained for one-step prediction, the first with the original data and the second with the IULTN-generated data. The MAEs of the solar and energy data prediction for the nine users collected over 200 train and test runs are shown in Figure 26. It is clear from Figure 26 that the MAE density of the predictions with the IULTN-generated data closely matched that of the original data. An objective comparison with the results of [18] is not possible because of the sparsity of network information.

In summary, the autonomously generated IULTN data compares well with the original solar generation and energy consumption data both qualitatively and quantitatively.

V. CONCLUSION

The main contributions of this study are as follows: (1) Activation function slope modulation with a recurrent ISAFN architecture where the states and dimensions are decoupled from the ULTRNN for better control of memory retention. (2) Zigzag propagation algorithm for the weight updates to account for the state interactions in the IULTN. A truncation technique is presented that significantly reduces the speed of gradient computation without significantly
affecting accuracy. (3) Closed-loop training with an eigenvalue metric of the linearized closed-loop system to ensure sustained output generation emulating the target. The effectiveness of the proposed approach was demonstrated with applications to synthetic data generation encompassing chaotic systems, complex periodic patterns, biosignals, and nonstationary processes.

We developed a novel method to dynamically vary the activation function slopes of a ULTRNN by using a secondary network. Varying memory retention over time allows the network to capture short-term features while continuing to remember long-term dependencies, which is a key requirement for the generative modeling of time series. We compared two architectures for the secondary network, CSAFN and ISAFN. The CSAFN is a feedforward architecture that uses the main ULTRNN states to compute activation function slopes. This simplifies training by directly extending the backpropagation algorithm. The ISAFN uses a recurrent architecture with independent states to compute activation function slopes. The interaction between the ISAFN and ULTRNN states makes the z-prop training process complex. The sharing of states in the CULTN architecture results in suboptimal performance. The independence of states in the IULTN architecture allows for a better optimization of the activation function slopes.

The marginal stability of the triangular state-feedback submatrices of the ULTRNN ensures stable and robust training of the open-loop system, because the gradients are not allowed to either explode or vanish. In generative tasks, with the predicted output fed back as the input to generate the next prediction, it is essential to ensure marginal stability of the closed-loop system. An eigenvalue metric of the linearized closed-loop system was employed to ensure that the network behaved as a nonlinear oscillator whose output was neither saturated nor collapsed. Maintaining the marginal stability of both open- and closed-loop systems inherently allows for the robust and stable learning of the target features. Robust learning was enhanced by noise injection, n-step prediction, and pilot waves.

Examples demonstrate that the autonomous waveforms generated by the IULTN have qualitative and quantitative features closely similar to the target, and that the IULTN outperforms state-of-the-art neural network architectures with a smaller number of trainable weights and lower training times, as shown in Table V. Specifically, the IULTN uses only a very small fraction of the weights when compared with deep GAN networks.

**REFERENCES**

[1] S. C. Sivakumar, W. J. Robertson, and W. J. Phillips, “Online stabilization of block-diagonal recurrent neural networks,” *IEEE Trans. on Neural Networks*, 10(1), pp. 167-175, Jan. 1999.

[2] S. Sivakumar, and S. Sivakumar, “Marginally stable triangular recurrent neural network architecture for time series prediction,” *IEEE Trans. on Cybernetics*, 48(10), pp. 2836-2850, Oct. 2018.

[3] Y. Bengio, A. Courville, and P. Vincen, (2013, August) “Representation learning: A review and new perspectives,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 35(8), pp. 1798-1828, Aug. 2013.

[4] 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. on Sys., Man, and Cyb.*, 36(2), pp. 242-254, Feb. 2006.

[5] P. A. Mastorocostas, C. H. Hilas, D. Varsamis, and S. Dova, “A Recurrent Neural Network-based Forecasting System for Telecommunications Call Volume,” *Applied Mathematics and Information Sciences*, 7(5), pp. 1643-1650, May 2013.

[6] P. Campolucci, “A circuit theory approach to recurrent neural network architecture and learning methods”, Research Doctorate in Electrotechnical Engineering, University of Bologna, 1998.

[7] O. Adigun, and B. Kosko, “Using noise to speed up video classification with recurrent backpropagation,” *Intl. Joint Conf. on Neural Networks*, pp. 108-115, 2017.

[8] S. Sivakumar, and S. Sivakumar, Modulation of Activation Function in Triangular Recurrent Neural Networks for Time Series Modeling, *Intl. Joint Conf. on Neural Networks*, Budapest, July 2019.

[9] J. Schmidhuber, “Deep learning in neural networks: an overview,” *Neural Networks*, 61, pp. 85-117, Jan. 2015.

[10] R. Bakker, J. C. Schouten, C. L. Giles, F. Takens, and C.M. van den Bleek, “Learning chaotic attractors by neural networks,” *Neural Computation*, 12, 2000, pp. 2355-2383.

[11] E. N. Lorenz, “Deterministic nonperiodic flow,” *Journal of the Atmospheric Sciences*, 20(2), pp. 130-141, 1963.

[12] A. S. Weigend, and N. A. Gershenfeld, “Time Series Prediction,” *Addison-Wesley*, 1994.

[13] K. Yanagisawa, and S. Nagata, “Fundamental Study on Design System of Kolam Pattern”, *Forma*, 22, pp. 31-46, 2007.

[14] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith, “A Dynamical Model for Generating Synthetic Electrocardiogram Signals,” *IEEE Trans on Biomedical Eng*, vol. 50, No. 3, pp. 289-294, Mar. 2003.
[15] R. Bousseljot, and D. A. Kreiseler, “Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet,” Biomedizinische Technik, Band 40, Ergänzungsband 1 S 317, Schnabel - ECG Heartbeat Categorization Dataset, Segmented and Preprocessed ECG Signals for Heartbeat Classification, 1995.

[16] J. Yoon, D. Jarrett, and M. van der Schaar, “Time-series Generative Adversarial Networks,” 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada, 2019.

[17] “Pecan Street Dataset”, http://www.pecanstreet.org/category/dataport/, 2018.

[18] C. Zhang, S. R. Kuppannagiri, R. Kannan, and V. K. Prasanna, “Generative Adversarial Network for Synthetic Time Series Data Generation in Smart Grids”, IEEE SmartgridComm, Oct. 2018.

[19] V. Kozyakin, “On accuracy of approximation of the spectral radius by the Gelfand formula,” Linear Algebra and its Applications, vol. 431, Iss. 11, pp. 2134-2141, Nov. 2009.

[20] H. Jaeger, “The ‘echo state’ approach to analysing and training recurrent neural networks – with an Erratum note,” Fraunhofer Institute for Autonomous Intelligent Systems, 2010.

[21] K. C. Jim, C. L. Giles, and B. G. Horne, “An analysis of noise in recurrent neural networks: Convergence and Generalization,” IEEE Trans. on Neural Networks, 7(6), pp. 1424-1438, 1996.

[22] D. J. C. MacKay, “Information Theory, Inference, and Learning Algorithms,” Cambridge Univ. Press., ISBN 9780521642989, p.34, 2003.

[23] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, “Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data,” Chaos, vol. 27 (12), 2017.

[24] G. Siromoney, R. Siromoney, and K. Krithivasan, “Array Grammars and Kolam,” Computer Graphics and Image Processing, Elsevier, vol. 3, Iss. 1, pp. 63-82, Mar. 1974.

[25] M. Asher, “The Kolam Tradition: A tradition of figure-drawing in southern India expresses mathematical ideas and has attracted the attention of computer science,” American Scientist, vol. 90, No. 1, pp. 56-63, Jan./Feb. 2002.

[26] S. Harada, H. Hayashi, and S. Uchida, (2018) “Biosignal Data Augmentation Based on Generative Adversarial Networks,” Conf. Proc IEEE Eng Med Biol Soc., pp. 368-371, 2018.

[27] T. G. Technion, and K. Radinsky, “PGANs: Personalized Generative Adversarial Networks for ECG Synthesis to Improve Patient-Specific Deep ECG Classification,” Proc. Of the AAAI Conf. on Artificial Intelligence, vol. 33 No 01: 2019.

[28] F. B. Bryant, and P. R. Yarnold, “Principal-components analysis and exploratory and confirmatory factor analysis,” Reading and understanding multivariate statistics, pp. 99–136, American Psychological Association, 1995.

[29] L. van der Maaten, and G. Hinton, “Visualizing data using t-SNE,” Journal of Machine Learning Research, vol. 9, pp. 2579-2606, Nov. 2008.