Chaos

ChaosNet: A chaos based artificial neural network architecture for classification

Cite as: Chaos **29**, 113125 (2019); https://doi.org/10.1063/1.5120831
Submitted: 21 July 2019 . Accepted: 04 November 2019 . Published Online: 20 November 2019

Harikrishnan Nellippallil Balakrishnan, Aditi Kathpalia, Snehanshu Saha, and Nithin Nagaraj

ChaosNet: A chaos based artificial neural network architecture for classification

Cite as: Chaos **29**, 113125 (2019); doi: 10.1063/1.5120831
Submitted: 21 July 2019 · Accepted: 4 November 2019 ·
Published Online: 20 November 2019

Harikrishnan Nellippallil Balakrishnan,[1,a)] Aditi Kathpalia,[1,b)] Snehanshu Saha,[2,c)] and Nithin Nagaraj[1,d)]

AFFILIATIONS

[1]Consciousness Studies Programme, National Institute of Advanced Studies, Indian Institute of Science Campus, Bengaluru 560012, India
[2]Center for AstroInformatics, Modeling and Simulation (CAMS) and Department of Computer Science and Information Systems, BITS Pilani K.K. Birla Goa Campus, Zuarinagar, Goa 403726, India

Note: This paper is part of the Focus Issue, "When Machine Learning Meets Complex Systems: Networks, Chaos and Nonlinear Dynamics."
[a)]**Electronic mail:** harikrishnan.nb@nias.res.in
[b)]**Also at:** Manipal Academy of Higher Education, Manipal, Karnataka 576104, India. **Electronic mail:** kathpaliaaditi@nias.res.in
[c)]**Electronic mail:** snehanshusaha@pes.edu. **URL:** http://astrirg.org/projects.html
[d)]**Electronic mail:** nithin@nias.res.in

ABSTRACT

Inspired by chaotic firing of neurons in the brain, we propose `ChaosNet`—a novel chaos based artificial neural network architecture for classification tasks. `ChaosNet` is built using layers of neurons, each of which is a 1D chaotic map known as the Generalized Luröth Series (GLS) that has been shown in earlier works to possess very useful properties for compression, cryptography, and for computing XOR and other logical operations. In this work, we design a novel learning algorithm on `ChaosNet` that exploits the topological transitivity property of the chaotic GLS neurons. The proposed learning algorithm gives consistently good performance accuracy in a number of classification tasks on well known publicly available datasets with very limited training samples. Even with as low as seven (or fewer) training samples/class (which accounts for less than 0.05% of the total available data), `ChaosNet` yields performance accuracies in the range of 73.89% − 98.33%. We demonstrate the robustness of `ChaosNet` to additive parameter noise and also provide an example implementation of a two layer `ChaosNet` for enhancing classification accuracy. We envisage the development of several other novel learning algorithms on `ChaosNet` in the near future.

Chaos has been empirically found in the brain at several spatiotemporal scales.[1,2] In fact, individual neurons in the brain are known to exhibit chaotic bursting activity and several neuronal models such as the Hindmarsh-Rose neuron model exhibit complex chaotic dynamics.[3] Though Artificial Neural Networks (ANNs) such as Recurrent Neural Networks exhibit chaos, to our knowledge, there have been no successful attempts to build an ANN for classification tasks that is entirely comprised of neurons which are individually chaotic. Building on our earlier research, in this work, we propose `ChaosNet`—an ANN built out of neurons—each of which is a 1D chaotic map known as a Generalized Luröth Series (GLS). GLS has been shown to have salient properties such as ability to encode and decode information losslessly with Shannon optimality, computing logical operations (XOR, AND, etc.), universal approximation property, and ergodicity (mixing) for cryptography applications. In this work, `ChaosNet` exploits the topological transitivity property of chaotic GLS neurons for classification tasks with state-of-the art accuracies in the low training sample regime. This work, inspired by the chaotic nature of neurons in the brain, demonstrates the unreasonable effectiveness of chaos and its properties for machine learning. It also paves the way for designing and implementing other novel learning algorithms on the `ChaosNet` architecture.

I. INTRODUCTION

With the success of artificial intelligence, learning through algorithms such as Machine Learning (ML) and Deep Learning (DL) has become an area of intense activity and popularity

with applications reaching almost every field known to humanity. These include medical diagnosis,[4] computer vision, cyber security,[5] natural language processing, and speech processing,[6] just to name a few. These algorithms, though inspired by the biological brain, are remotely related to the biological process of learning and memory encoding. The learning procedures used in these artificial neural networks (ANNs) to modify weights and biases are based on optimization techniques and minimization of loss/error functions. The ANNs at present use an enormous number of hyperparameters which are fixed by an *ad hoc* procedure for improving prediction as more and more new data are input into the system. These synaptic changes employed are solely data-driven and have little or no rigorous theoretical backing.[7,8] Furthermore, for accurate prediction/classification, these methods require an enormous amount of training data that capture the distribution of the target classes.

Despite their tremendous success, ANNs are nowhere close to the human mind for accomplishing tasks such as natural language processing. To incorporate the excellent learning abilities of the human brain as well as to understand the brain better, researchers are now focusing on developing biologically-inspired algorithms and architectures. This is being done both in the context of learning[9] and memory encoding.[10–20]

One of the most interesting properties of the brain is its ability to exhibit "Chaos"[21]—the phenomenon of complex unpredictable and randomlike behavior arising from simple deterministic nonlinear systems.[22] The dynamics in electroencephalogram (EEG) signals is known to be chaotic.[2] The sensitivity to small shifts in internal functional parameters of a neuronal system helps to get the desired response to different influences. This attribute resembles the dynamical properties of chaotic systems.[23–25] Moreover, it is seen that the brain may not reach a state of equilibrium after a transient, but it is constantly alternating between different states. For this reason, it is suggested that with the change in functional parameters of the neurons, the brain is able to exhibit different behaviors—periodic orbits, weak chaos, and strong chaos for different purposes. For example, there has been evidence to suggest that weak chaos may be good for learning[26] and periodic activity in the brain being useful for attention related tasks.[27] Thus, chaotic regimes exhibiting a wide variety of behaviors help the brain in quick adaptation to changing conditions.

Chaotic behavior is exhibited not only by brain networks which are composed of billions of neurons, but the dynamics at the neuronal level (cellular and subcellular) are also chaotic.[2] Impulse trains produced by these neurons are actually responsible for the transmission and storage of information in the brain. These impulses or *action potentials* are generated when different ions cross the axonal membrane causing a change in the voltage across it. Hodgkin and Huxley were the first to propose a dynamical system's model for the interaction between the ion channels and axon membrane, that is capable of generating realistic action potentials.[28] Later, its simplified versions such as the Hindmarsh-Rose model[29] and the Fitzugh-Nagumo[30,31] model were proposed. All these models exhibit chaotic behavior.

Although there exist some artificial neural networks which display chaotic dynamics (an example is Recurrent Neural Networks[32]), to the best of our knowledge, none of the architectures proposed for classification tasks to date exhibit chaos at the level of individual neurons. However, for a theoretical explanation of memory encoding in the brain, many chaotic neuron models have been suggested. These include the Aihara model[10] which has been utilized for memory encoding in unstable periodic orbits of the network.[11] Freeman, Kozma, and group developed chaotic models inspired from the mammalian olfactory network to explain the process of memorizing of odors.[12–14] Chaotic neural networks have been studied also by Tsuda *et al.* for their functional roles as short term memory generators as well a dynamic link for long term memory.[15,16] Kaneko has explored the dynamical properties of globally coupled chaotic maps suggesting possible biological information processing capabilities of these networks.[17,18] Our group (two of the authors) has also proposed a biologically-inspired network architecture with chaotic neurons which is capable of memory encoding.[19]

In this work, we propose `ChaosNet`—an ANN built out of 1D chaotic map Generalized Luröth Series (GLS) as its individual neurons. This network can accomplish classification tasks by learning with limited training samples. `ChaosNet` is developed as an attempt to use some of the best properties of biological neural networks arising as a result of rich chaotic behavior of individual neurons and is shown to accomplish challenging classification tasks comparable to or better than conventional ANNs while requiring far less training samples.

The choice of 1D maps as neurons in `ChaosNet` helps to keep the processing simple and at the same time exploit the useful properties of chaos. Our group (two of the authors) has discussed the use of the property of topological transitivity of these GLS neurons for classification.[33] Building on this initial work, in the current manuscript, we propose a novel and improved version of topological transitivity scheme for classification. This improved novel scheme, proposed for the very first time in this paper, utilizes a "spike-count rate"-like property of the firing of chaotic neurons as a neural code for learning and is inspired from biological neurons. Moreover, the network is capable of displaying hierarchical architecture which can integrate information as it is passed on to higher levels (deeper layers in the network). The proposed classification scheme is rigorously tested on publicly available datasets—MNIST, KDDCup'99, Exoplanet, and Iris.

Current state-of-the-art algorithms in the AI community rely heavily on the availability of enormous amounts of training data. However, there are several practical scenarios where huge amounts of training data may not be available.[34] Learning from limited samples plays a key role especially in a field like cybersecurity where new malware attacks occur frequently. For example, detecting zero day malware attack demands the algorithms to learn from fewer data samples. Our proposed `ChaosNet` architecture addresses this issue. The paper is organized as follows. GLS-neuron and its properties are described in Sec. II. In Sec. III, we introduce one layer `ChaosNet` architecture and its application for topological transitivity symbolic sequence based classification algorithm. Experiments, results, and discussion including parameter noise analysis of the one layer TT-SS classification algorithm are dealt in Sec. IV. Multilayer `ChaosNet` architecture is introduced in Sec. V. We conclude with future research directions in Sec. VI.

II. GLS-NEURON AND ITS PROPERTIES

The neuron we propose is a piecewise linear 1D chaotic map known as Generalized Luröth Series or GLS.[35] The well known Tent

map, Binary map, and their skewed cousins are all examples of GLS. Mathematically, the types of GLS neurons we use in this work are described below.

A. GLS-neuron types: $T_{Skew-Tent}(x)$ and $T_{Skew-Binary}(x)$

A map $T_{Skew-Binary} : [0, 1) \rightarrow [0, 1)$ is defined as

$$T_{Skew-Binary}(x) = \begin{cases} \dfrac{x}{b}, & 0 \le x < b, \\ \dfrac{(x-b)}{(1-b)}, & b \le x < 1, \end{cases}$$

where $x \in [0, 1)$ and $0 < b < 1$. Refer to Fig. 1(a).

A map $T_{Skew-Tent} : [0, 1) \rightarrow [0, 1)$ is defined as

$$T_{Skew-Tent}(x) = \begin{cases} \dfrac{x}{b}, & 0 \le x < b, \\ \dfrac{(1-x)}{(1-b)}, & b \le x < 1, \end{cases}$$

where $x \in [0, 1)$ and $0 < b < 1$. Refer to Fig. 1(b).

The symbols **L (or 0)** and **R (or 1)** are associated with the intervals $[0, b)$ and $(b, 1)$, respectively, thereby defining the "symbolic sequence" [36] for every trajectory starting from an initial value on the map. We enumerate some salient properties of the GLS-neuron.

1. Each GLS-neuron has two parameters—an initial activity value x_0 and the skew value b. The parameter b defines the "generating Markov partition"[37] for the map and also acts as "internal discrimination threshold" of the neuron which will be used for feature extraction.
2. GLS-neuron can fire either chaotically or in a periodic fashion (of any finite period length) depending on the value of the initial activity value x_0. The degree of chaos is controlled by the skew parameter b. The Lyapunov exponent of the map[38] is given by $\lambda_b = -b\ln(b) - (1-b)\ln(1-b)$. If the base of the logarithm is chosen as 2, then $\lambda_b = H(S_{x_0})$ (bits/iteration), where S_{x_0} is the symbolic sequence of the trajectory obtained by iterating the initial neural activity x_0 and $H(\cdot)$ is Shannon Entropy in bits/symbol. For $0 < b < 1$, $\lambda_b > 0$.

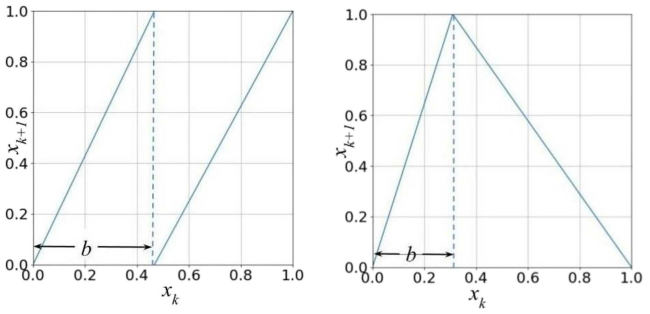

FIG. 1. Generalized Luröth series, GLS-Neuron, is fundamentally of two types: (a) Left: skew-binary map $T_{Skew-Binary}$. (b) Right: skew-tent map $T_{Skew-Tent}$.

3. Any finite length input stream of bits can be "losslessly" compressed as an initial activity value x_0 on the neuron by performing a backward iteration on the map. Further, such a lossless compression scheme has been previously shown to be "Shannon optimal."[39]
4. Error detection properties can be incorporated into GLS-neuron to counter the effect of noise.[40]
5. Owing to its excellent chaotic and ergodic (mixing) properties, GLS (and other related 1D maps) has been employed in cryptography.[38,39,41]
6. Recently, two of the authors of the present work have proposed a compression-based neural architecture for memory encoding and decoding using GLS.[19]
7. GLS-neuron can compute logical operations such as XOR, AND, NOT, OR, etc., by switching between appropriate maps as described in a previous work.[19,38]
8. GLS-neuron has the property of "topological transitivity"—defined in a later section, which we will employ to perform classification.
9. A single layer of a finite number of GLS-neurons satisfies a version of the "Universal Approximation Theorem" which we shall prove in a subsequent section of this paper.

The aforementioned properties make GLS-neurons an ideal choice for building our novel architecture for classification.

III. ChaosNet: THE PROPOSED ARCHITECTURE

In this section, we first introduce the novel `ChaosNet` architecture followed by a description of the single-layer Topological Transitivity-Symbolic Sequence classification algorithm. We discuss the key principles behind this algorithm, its parameters and hyperparameters, an illustrative example and a proof of the Universal Approximation Theorem.

A. Single layer ChaosNet topological transitivity–symbolic sequence (TT-SS) based classification algorithm

We propose for the first time, a single layer chaos inspired neuronal architecture for solving classification problems (Fig. 2). It consists of a single input and a single output layer. The input layer consists of n GLS neurons $C_1, C_2, \ldots, C_n$ and extracts patterns from each sample of the input data instance. The nodes $O_1, O_2, \ldots, O_s$ in the output layer store the representation vectors corresponding to each of the s target classes (s-class classification problem). The entire input data are represented as a matrix ($\boldsymbol{X}$) of dimensions $m \times n$, where m represents the number of data instances and n represents the number of samples for each data instance. When the input data consist of m images, each of dimensions $W \times Y$, we vectorize each image and the resulting matrix $\boldsymbol{X}$ will be of dimensions $m \times n$ with $n = WY$. Each row of this matrix will be an instance of the vectorized image. The number of neurons in the input layer of `ChaosNet` is set equal to the number of samples of data instance.

The GLS neurons $(C_1, C_2, \ldots, C_n)$ in the architecture in Fig. 2 have an initial neural activity of q units. This is also considered as the initial value of the chaotic map. The GLS neurons start firing chaotically when encountered by the stimulus. The stimulus is a real

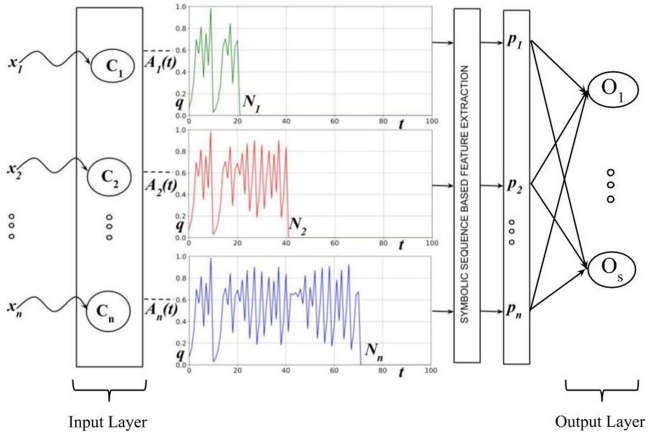

FIG. 2. ChaosNet: the proposed chaotic GLS neural network architecture for classification tasks. $C_1, C_2, \ldots, C_n$ are the 1D GLS chaotic neurons. The initial normalized neural activity of each neuron is q units. The input or the normalized set of stimuli to the network is represented as $\{x_i\}_{i=1}^{n}$. The chaotic firing of a GLS neuron C_i halts when its chaotic activity value $A_i(t)$ starting from initial neural activity (q) reaches the ε-neighborhood of stimulus. This neuron has a "firing time" of N_i *ms*. $A_i(t)$ contains topological transitivity symbolic sequence feature p_i. This feature is extracted from $A_i(t)$ of the C_ith GLS-neuron.

number which is normalized to lie between zero and one. The input vector of data samples (or data instance) represented as $x_1, x_2, \ldots, x_n$ in Fig. 2 are the stimuli corresponding to $C_1, C_2, \ldots, C_n$, respectively. The chaotic neural activity values at time t of the GLS neurons $C_1, C_2, \ldots, C_n$ are denoted by $A_1(t), A_2(t), \ldots, A_n(t)$, respectively, where

What is this T, T(Skew-Bin) or T(Skew-Tent)

$$A_i(t) = T(A_i(t-1)). \tag{1}$$

The chaotic firing of each GLS neuron stops when their corresponding activity value $A_1(t), A_2(t), \ldots, A_n(t)$ starting from the initial neural activity (q) reaches the epsilon neighborhood of $x_1, x_2, \ldots, x_n$. The time at which each neuron stops firing can thus be different. The halting of firing of each GLS neuron is guaranteed by "Topological Transitivity" (TT) property. The time taken (N_k ms) for the firing of kth GLS neuron to reach the epsilon neighborhood of incoming stimulus is termed as "firing time." The fraction of this firing time for which the activity of the GLS neuron is greater than the discrimination threshold (b) is defined as "Topological Transitivity—Symbolic Sequence (TT-SS) feature." The formal definition of "Topological Transitivity" is as follows.

Definition 1: "Topological Transitivity" property for a map $T : R \to R$ states that for all nonempty open set pairs D and E in R, there exists an element $d \in D$ and a non-negative finite integer n such that $T^n(d) \in E$.

For example, we consider a GLS-1D map (T) which is chaotic with $R : [0, 1)$. Let $D = (q - \varepsilon, q + \varepsilon)$ and $E = (x_k - \varepsilon, x_k + \varepsilon)$, where $\varepsilon > 0$. From the definition of Topological Transitivity, the existence of an integer $N_k \geq 0$ and a real number $d \in D$ such that $T^{N_k}(d) \in E$ is ensured. We consider $d = q$ (initial neural activity of the GLS-neuron) and x_k as the stimulus to the kth GLS neuron (after normalizing). Thus, $N_k \geq 0$ will always exist. It is important to highlight that for certain values of q there may be no such N_k, for, eg., initial values that lead to periodic orbits. However, we can always find a value for q for which N_k exists. This is because, for a chaotic map, there are an infinite number of initial values that lead to nonperiodic ergodic orbits.

Single layer `ChaosNet` TT-SS algorithm consists of mainly three steps:

- Feature extraction using TT-SS—TT-SS based feature extraction step is represented in the flow chart provided in Fig. 3. Let the ε-neighborhood of kth sample of ith data instance x_k^i be represented as $I_k^i = (x_k^i - \varepsilon, x_k^i + \varepsilon)$, where $\varepsilon > 0$. Let the normalized stimulus to the kth GLS-neuron be x_k^i and the corresponding neuronal firing time be N_k^i. The firing trajectory of the GLS neuron upon encountering a stimulus (x_k^i) is represented as $q \to T_k(q) \to T_k^2(q) \to T_k^3(q) \cdots \to T_k^{N_i}(q)$, where $T_k^{N_i}(q) \in I_k^i$. The trajectory is denoted as $\boldsymbol{A}_k = [q, T_k(q), \ldots, T_k^{N_i}(q)]$. The fraction of firing time for which the GLS-neuron's chaotic trajectory activity value ($A_k(t)$) is greater than the internal discrimination threshold value (b) is defined as the Topological Transitivity—Symbolic Sequence (TT-SS) feature and denoted by p_k^i,

$$p_k^i = \frac{h_k^i}{N_k^i}, \tag{2}$$

where h_k^i represents the duration of firing for which the chaotic trajectory is above the discrimination threshold (b) for the kth GLS neuron (see Fig. 4). The TT-SS feature can be looked at as a spike-count rate based neural code[42] for the GLS-neuron that is active (or firing) for a total of N_k^i time units, in which the spiking activity (greater than threshold activity) is limited to h_k^i time units.
- TT-SS Training—TT-SS based training step is represented in the flow chart provided in Fig. 5. Let us assume an s class classification problem where classes are represented by $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_s$ and the corresponding true-labels be denoted as $1, 2, \ldots, s$, respectively. Let the normalized distinct matrices be $U^1, U^2, \ldots, U^s$ of size $m \times n$. The matrices $U^1, U^2, \ldots, U^s$ denote the data belonging to $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_s$, respectively. Training involves extracting features from $U^1, U^2, \ldots, U^s$ using the TT-SS method so as to yield $V^1, V^2, \ldots, V^s$. Feature extraction using the TT-SS algorithm is applied on each stimulus, hence the size of $V^1, V^2, \ldots, V^s$ will be same as $U^1, U^2, \ldots, U^s$. Once the TT-SS based feature extraction is found for the data belonging to the s distinct classes, the average across row is computed next,

$$M^1 = \frac{1}{m}\left[\sum_{i=1}^{m} V_{i1}^1, \sum_{i=1}^{m} V_{i2}^1, \ldots, \sum_{i=1}^{m} V_{in}^1\right],$$
$$M^2 = \frac{1}{m}\left[\sum_{i=1}^{m} V_{i1}^2, \sum_{i=1}^{m} V_{i2}^2, \ldots, \sum_{i=1}^{m} V_{in}^2\right],$$
$$\vdots$$
$$M^s = \frac{1}{m}\left[\sum_{i=1}^{m} V_{i1}^s, \sum_{i=1}^{m} V_{i2}^s, \ldots, \sum_{i=1}^{m} V_{in}^s\right].$$

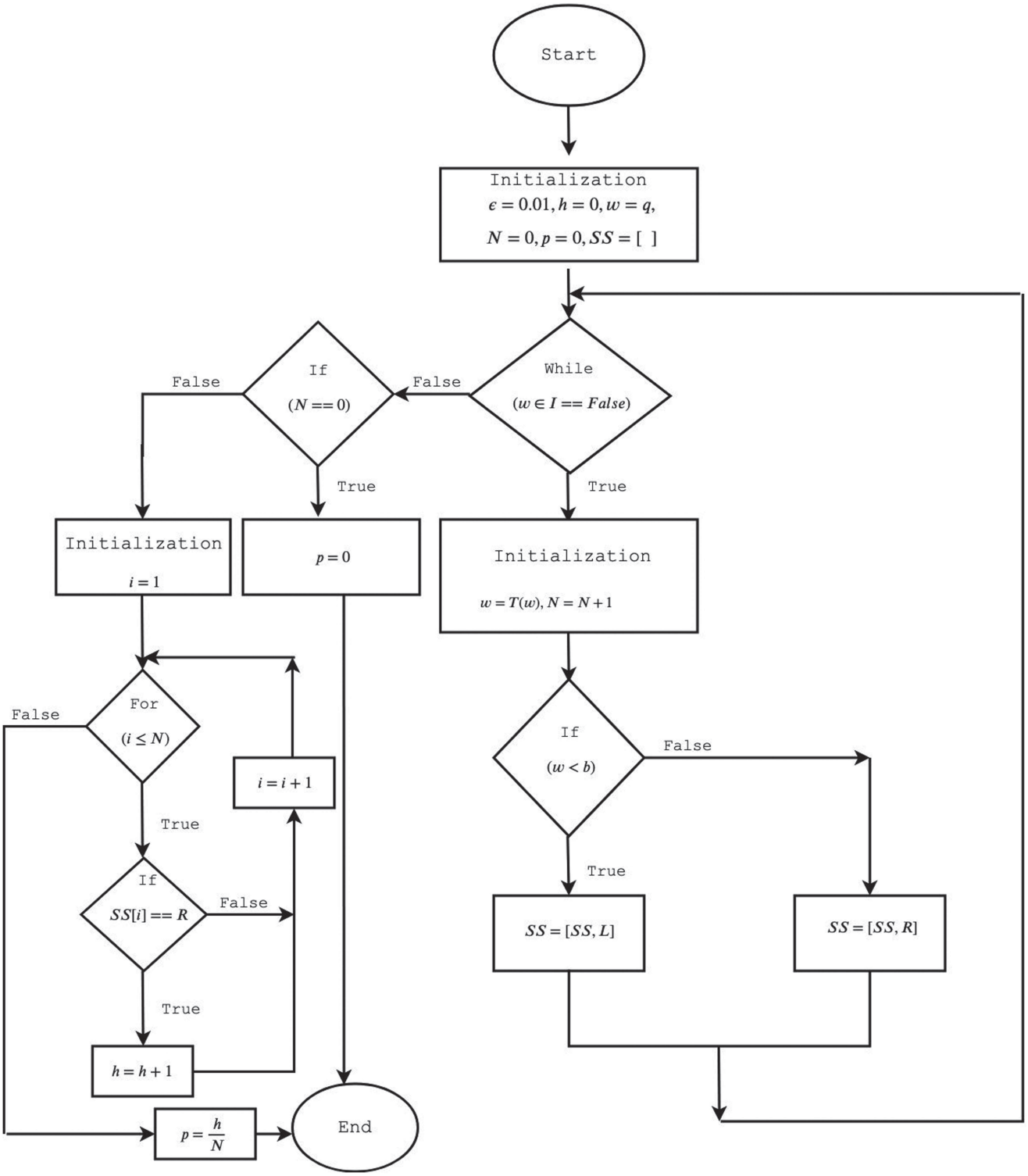

FIG. 3. Flow chart for the feature extraction step of the 1-layer ChaosNet TT-SS algorithm.

The s row vectors $M^1, M^2, \ldots, M^s$ are termed as "mean representation vectors" corresponding to the s classes. The "average internal representation" of all the stimuli corresponding to kth class is encoded in the vector M^k. As more and more input data are received, the mean representation vectors get updated.

- TT-SS Testing—the computational steps involved in testing are in the flow chart provided in Fig. 6. Let Z denote the normalized test data matrix of size $r \times n$. The ith row of Z represents ith test data instance denoted as $z^i = [z_1^i, z_2^i, z_3^i, \ldots, z_n^i]$. The TT-SS based feature extraction step is applied to each row [each test data instance (z^i) where $i = 1, 2, 3, \ldots, r$]. Let the feature extracted data of test samples using TT-SS algorithm be denoted as F, where $f^i = [f_1^i, f_2^i, \ldots, f_n^i]$ is the ith row of F. Feature extraction is followed by the computation of cosine similarity of f^i independently with each of the $M^1, M^2, \ldots, M^s$ ("mean representation" vectors), respectively,

$$\cos(\theta_1) = \frac{f^i \cdot M^1}{\|f^i\|_2 \, \|M^1\|_2},$$

$$\cos(\theta_2) = \frac{f^i \cdot M^2}{\|f^i\|_2 \, \|M^2\|_2},$$

$$\vdots$$

$$\cos(\theta_s) = \frac{f^i \cdot M^s}{\|f^i\|_2 \, \|M^s\|_2}.$$

The scalar product between vectors f^i and M^k is represented by $f^i \cdot M^k$, $\|v\|_2$ denotes the l_2 norm of row-vector v. The above will give s scalar values which are the cosine similarity values between $\{M^k\}_{k=1}^s$ and f^i. Out of these s scalar values, the index (l) corresponding to the maximum cosine similarity [$\cos(\theta_l)$] is considered as the label for f^i,

$$\theta_l = \arg\max_{\theta_i}(\cos(\theta_1), \cos(\theta_2), \ldots, \cos(\theta_s)).$$

If there is more than one index with maximum cosine similarity, we take the smallest such index. The above procedure is continued until a unique label is assigned to each test data instance.

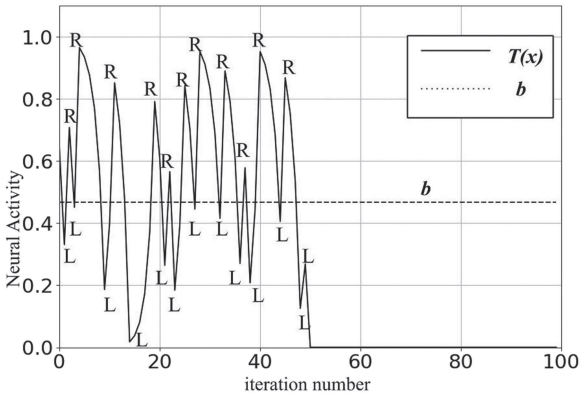

FIG. 4. Illustration of feature extraction using topological transitivity—symbolic sequence method (Algorithm 2). The GLS-Neurons has a default initial neural activity of q units. The ith GLS-Neuron fires for N_i iterations (chaotically) and stops by reaching the ε-neighborhood I_i of the ith input stimulus. For the duration of the GLS-Neuron being active (chaotic firing), the fraction of the time when the neural activity exceeds the internal discrimination threshold b of the neuron (marked as R above) is the extracted TT-SS feature.

B. Parameters vs hyperparameters

Distinguishing model parameters and model hyperparameters plays a crucial role in machine learning tasks. The model parameters and hyperparameters of the proposed method are as follows:

Model parameter: This is an internal parameter which is estimated from the data. These internal parameters are what the model learns while training. In the case of single layer `ChaosNet` TT-SS method, the mean representation vectors $M^1, M^2, \ldots, M^s$ are the model parameters which are learned while training. The model parameters for deep learning (DL) are the weights and biases learnt during training the neural network. In Support Vector Machines (SVMs), the support vectors are the parameters. In all these cases, the parameters are learned while training.

Model hyperparameters: These are configurations which are external to the model and typically not estimated from the data. The hyperparameters are tuned for a given classification or predictive modeling task in order to estimate the best model parameters. The hyperparameters of a model are often arrived by heuristics and hence are different for different tasks. In the case of the single layer `ChaosNet` TT-SS method, the hyperparameters are the "initial neural activity" (q), "internal discrimination threshold" (b), ε used in defining the neighborhood interval of x_k^i ($I_k^i = (x_k^i - \varepsilon, x_k^i + \varepsilon)$) and the "chaotic map" chosen. In DL, the hyperparameters are the "number of hidden layers," "number of neurons in the hidden layer," "learning rate," and the "activation function" chosen. In the case of K-Nearest Neighbors (KNN) classification algorithm, the "number of nearest neighbors" (k) is a hyperparameter. In SVM, the "choice of kernel" is a hyperparameter. In Decision Tree, "depth of the tree" and the "least number of samples required to split the internal node" are the hyperparameters. In all these cases, the hyperparameters need to be fixed by cross-validation.

C. Example to illustrate single layer ChaosNet TT-SS method

We consider a binary classification problem. The two classes are denoted as $\mathscr{C}_1$ and $\mathscr{C}_2$ with class labels one and two, respectively. The input dataset is a 4×4 matrix $\hat{X} = \begin{bmatrix} \hat{X}^1 \\ \hat{X}^2 \end{bmatrix}$. The first two rows of $\hat{X}$ are denoted as $\hat{X}^1$ and the remaining two rows of $\hat{X}$ are denoted as $\hat{X}^2$. $\hat{X}^1$ and $\hat{X}^2$ represent data instances belonging to $\mathscr{C}_1$ and $\mathscr{C}_2$, respectively. Since each row of $\hat{X}$ has four samples, the input layer of `ChaosNet` in this case has four GLS neurons $\{C_1, C_2, C_3, C_4\}$. We set the hyperparameters $q = 0.23$, $b = 0.56$, $\varepsilon = 0.01$, and chaotic map as $T_{Skew-Tent}$. The steps of the one layer `ChaosNet` TT-SS algorithm are as follows:

1. Step 1: Normalization of data: $\hat{X}$ is normalized[43] to X.
2. Step 2: Training—TT-SS based feature extraction: From the normalized matrix X, we sequentially pass one row of X

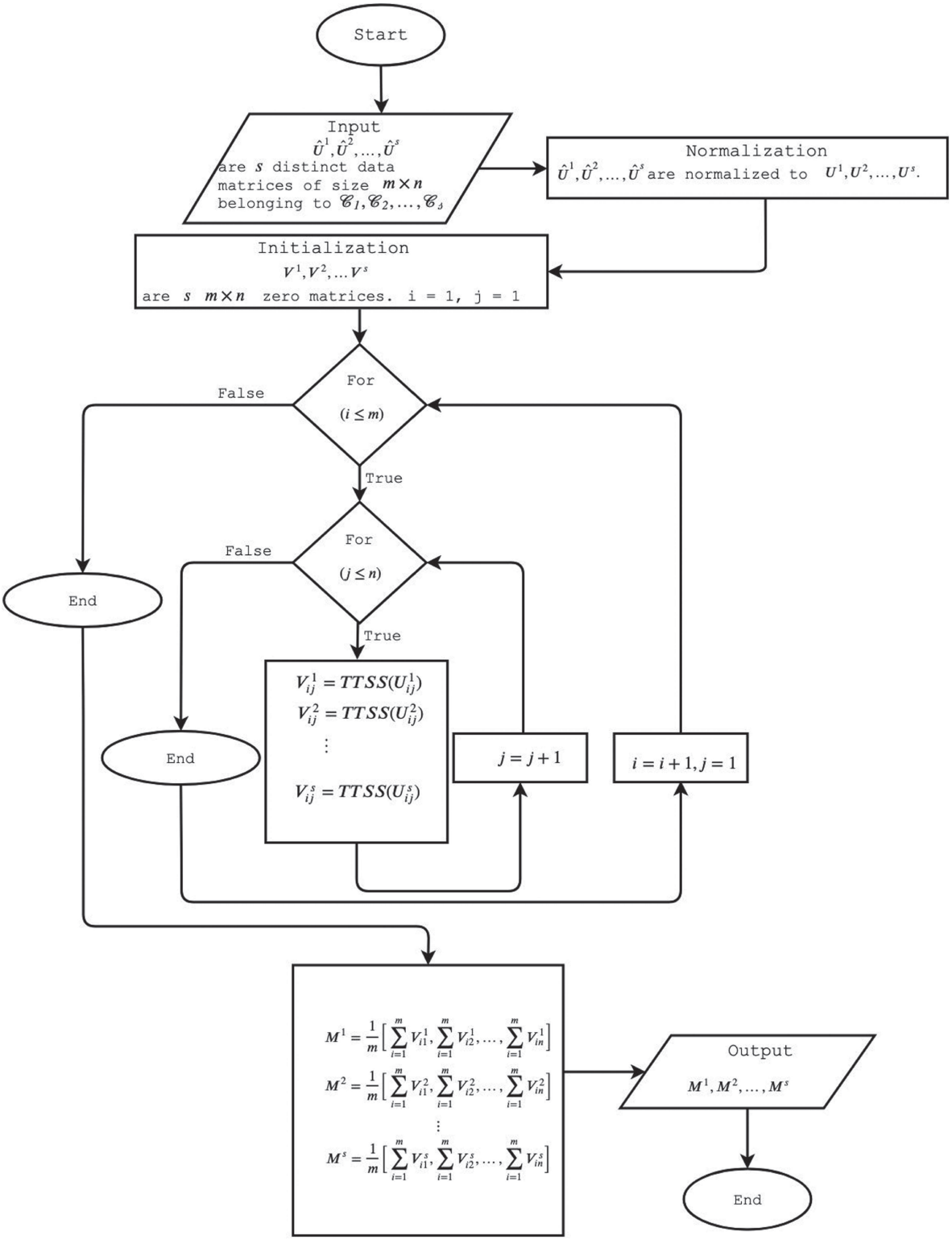

FIG. 5. Flow chart for training step of 1-layer ChaosNet TT-SS algorithm.

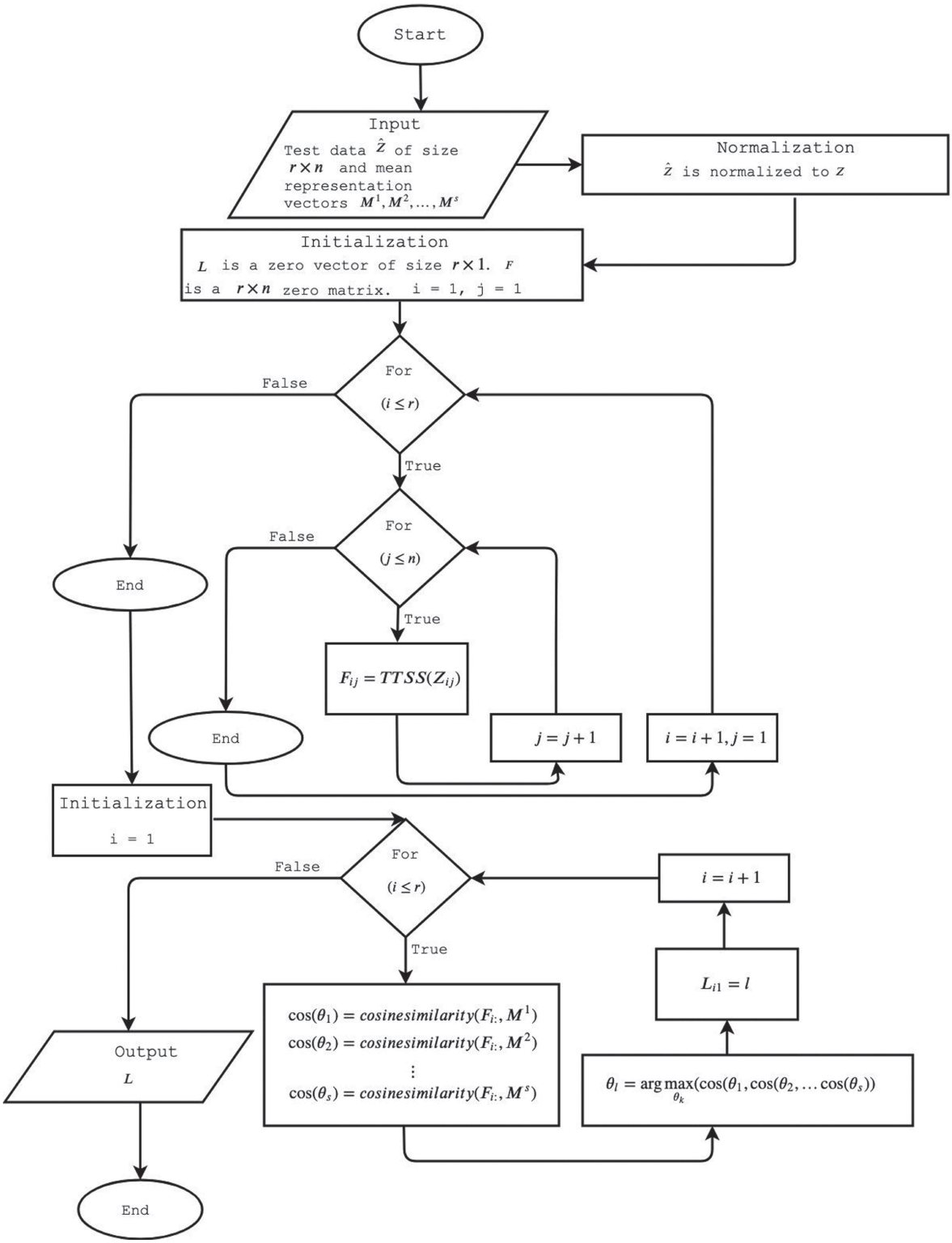

FIG. 6. Flow chart for the testing step of the 1-layer ChaosNet TT-SS algorithm.

at a time to the input layer of ChaosNet and extract the TT-SS feature. As an example, let $x^1 = [0.2, 0.5, 0.1, 0.9]$ and $x^2 = [0.23, 0.49, 0.15, 0.8]$ be the first two rows of X (pertaining to $\mathscr{C}_1$) which are passed to the input layer sequentially. Each GLS neuron in the input layer $\{C_1, C_2, C_3, C_4\}$ starts firing chaotically until it reaches the ε-neighborhood of x^1 which are $(0.2-\varepsilon, 0.2+\varepsilon)$, $(0.5-\varepsilon, 0.5+\varepsilon)$, $(0.1-\varepsilon, 0.1+\varepsilon)$, and $(0.9-\varepsilon, 0.9+\varepsilon)$. Let the neural firing of $\{C_1, C_2, C_3, C_4\}$ be denoted as $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$, $\boldsymbol{A}_4$. The fraction of firing time for which $A_i(t) > b$ (internal discrimination threshold) is determined for the four GLS neurons to yield the TT-SS feature vector $v^1 = [0.41, 0.35, 0.38, 0.5]$. Similarly, the TT-SS feature vector for x^2 is $v^2 = [0, 0.35, 0.35, 0.41]$. The mean representation vector of $\mathscr{C}_1$ is calculated as $M^1 = [\frac{(0.41+0)}{2}, \frac{(0.35+0.35)}{2}, \frac{(0.38+0.35)}{2}, \frac{(0.5+0.41)}{2}] = [0.205, 0.35, 0.365, 0.455]$. In a similar fashion, the mean representation vector of $\mathscr{C}_2$ is computed. Thus, at the end of training, the mean representation vectors M^1 and M^2 of $\mathscr{C}_1$ and $\mathscr{C}_2$ are stored in nodes O_1 and O_2 of the output layer, respectively.

3. Step 3: Testing—as an example, let $z = [z_1, z_2, z_3, z_4]$ be a test sample (after normalization) that needs to be classified. Similar to the training step, z is first fed to the input layer of ChaosNet having four GLS neurons and its activity is recorded. Subsequently, the TT-SS feature vector is extracted from the neural activity which is denoted as $f = [f_1, f_2, f_3, f_4]$. We compare f individually with the two internal representation vectors M^1 and M^2 by using cosine similarity metric. The test sample z is classified to that class for which the cosine similarity is maximum.

A single-layer ChaosNet with the finite number of GLS neurons has the ability to approximate any arbitrary real-valued, bounded discrete time function (with finite support) as we shall show in Sec. III D.

D. Universal approximation theorem (UAT)

Cybenko (in 1989) proved one of the earliest versions of the "Universal Approximation Theorem" (UAT). UAT states that continuous functions on compact subsets of $\mathbb{R}^n$ can be approximated by an ANN with one hidden layer having a finite number of neurons with sigmoidal activation functions.[44] Thus, simple neural networks with appropriately chosen parameters can approximate continuous functions of a wide variety. We have recently proven a version of this theorem for the GLS-neuron[19] which we reproduce below (with minor modifications).

Universal Approximation Theorem (UAT) for GLS-Neuron:

Let us consider a real-valued bounded discrete time function $g(n)$ having a support length *LEN*. UAT guarantees the existence of a finite set of GLS neurons denoted as W_1, W_2, ..., W_c, such that, for these c neurons, the GLS-encoded output values $x_1, p_1, x_2, p_2, \ldots, x_c, p_c$ can approximate $g(n)$. Here, x_is are the initial values and p_is are the skew parameters of the GLS maps. In other words, the UAT for GLS neurons guarantees the existence of the function $G(\cdot)$ satisfying the following:

$$|G(x_1, x_2, \ldots, x_c, p_1, p_2, \ldots, p_c) - g(n)| \leq \varepsilon, \tag{3}$$

where $\varepsilon > 0$ is arbitrarily small.

Proof: By construction: for a given $\varepsilon > 0$, the range of the function g is uniformly quantized in such a way that the quantized and original function differ by an error $\leq \varepsilon$. The boundedness of $g(n)$ ensures that the above is always true, because we can find integers a_1, a_2, ..., a_{LEN} corresponding to the time indices $1, 2, \ldots, LEN-1, LEN$ which satisfy the following inequality after proper global scaling of g:

$$\frac{1}{S_\varepsilon} \sum_{i=1}^{i=L} |(g^*(i) - a_i)| \leq \varepsilon, \tag{4}$$

where $\varepsilon > 0$, a_i-s are all integers and $g^*(\cdot) = S_\varepsilon g(\cdot)$, where S_ε (a finite real number) denotes the proper global scaling constant which in turn depends on ε. Let us consider only the significant number of bitplanes of a_i-s - denote it as c (the value of c is the least power of two which is just greater than the maximum of $\{a_i\}$).

Let the discrete-time quantized integer-valued signal that approximates $g(n)$ be called as $g_{quant}[n] = a_1, a_2, \ldots, a_{LEN}$. The c bitplanes of each value of the function $g_{quant}[n]$ are extracted next to yield c bitstreams $g_c[n], g_{c-1}[n], \ldots, g_1[n]$, where the Most Significant Bit (MSB) and Least Significant Bit (LSB) are denoted by c and one, respectively. The ith stream of bits is a *LEN* length binary list. The back-iteration on an appropriate GLS can encode the binary list *losslessly* to the initial value x_i (the probability of zero in the bitstream is represented by the parameter p_i which is also the skew parameter of the map). The above procedure is followed for the c bitstreams to yield $x_1, p_1, x_2, p_2, \ldots, x_c, p_c$ GLS encoded neurons. The perfect lossless compression property of GLS enables the recovery of original quantized bitstreams using GLS decoding.[39] Each step in our construction uses procedures that are deterministic (and which always halt). This means the compositions of several nonlinear maps (quantization, scaling, bitplane extraction, GLS encoding are all nonlinear) can be used to construct the desired function G. Thus, there exists a function $G(x_1, x_2, \ldots, x_c, p_1, p_2, \ldots, p_c)$ that satisfies inequality [Eq. (3)]. □

The function $G(x_1, x_2, \ldots, x_c, p_1, p_2, \ldots, p_c)$ is not unique since it depends on how the a_i-s are chosen and finding an explicit analytical expression is not easy (but we know it exists). The above argument can be extended for continuous-valued functions (by sampling) as well as functions in higher dimensions.

IV. EXPERIMENTS, RESULTS, AND DISCUSSION

Learning from insufficient training samples is very challenging for ANN/ML/DL algorithms since they are typically dependent on learning from vast amount of data. The efficacy of our proposed ChaosNet TT-SS method based classification is evaluated on MNIST, KDDCup'99, Iris and Exoplanet data in the low training sample regime. The description of the datasets used in the analysis of the proposed method is given here.

A. Datasets

1. MNIST

MNIST[45] is a commonly used hand written digits (zero to nine) image dataset in the ML/DL community. These images have a dimension of 28 pixels × 28 pixels and are stored digitally as eight bit gray scale images. The training set of MNIST database consists of 60 000 images whereas the test set consists of 10 000 images. MNIST is a

multiclass classification (10 classes) problem and the goal is to classify the images into their correct respective classes. For our analysis, we have independently trained with randomly chosen $1, 2, 3, \ldots, 21$ data samples per class. For each such trial of training, the algorithm is tested with (10 000) unseen test images.

2. KDDCup'99

KDDCup'99[46] is a benchmark dataset used in the evaluation of intrusion detection systems (IDS). The creation of this dataset is based on the acquired data in IDS DARPA'98 evaluation program.[47] There are roughly 4 900 000 single connection vectors in the KDDCup'99 training data. The number of features in each connection vector is 41. Each data sample is labeled as either "normal" or a "specific attack type." We considered 10% of data samples from the entire KDDCup'99 data. In this 10% KDDCup'99 data that we considered, there are normal and 21 different attack categories. Out of these, we took the following nine classes for our analysis: "back," "ipsweep," "neptune," "normal," "portsweep," "satan," "smurf," "teardrop," and "warezmaster." Training was done independently with $1, 2, 3, \ldots, 7$ data samples per class. The data samples were chosen randomly for training from the existing training set. For each trial of training, the algorithm is tested with unseen data.

3. Iris[48]

Iris data[49] consist of attributes from three types of Iris plants. These plants are as follows: "Setosa," "Versicolour," and "Virginica." The numbers of attributes used in this dataset are four. The attributes are "sepal length," "sepal width," "petal length," and "petal width" (all measured in centimeters). This is a three class classification problem with 50-data samples per category. For our analysis, we have independently trained with randomly chosen $1, 2, 3, \ldots, 6, 7$ data samples per class. For each trial of training, the algorithm is tested with (120) unseen test data.

4. Exoplanet

PHL-EC dataset[50] (combined with stellar data from the Hipparcos catalog[51]) has 68 attributes (of which 55 are continuous valued and 13 are categorical) and more than 3800 confirmed exoplanets (at the time of writing this paper). Important attributes such as surface temperature, atmospheric type, radius, mass, flux, earth's similarity index, escape velocity, orbital velocity, etc., are included in the catalog (with both observed and estimated attributes). From an analysis point-of-view, this presents interesting challenges.[52,53] The dataset consists of six classes, of which three were used in our analysis as they are sufficiently large in size. The other three classes are dropped (while training) since they had very low number of samples. The classes considered are "mesoplanet," "psychroplanet," and "nonhabitable." Based on their thermal properties, these three classes or types of planets are described as follows:

1. Mesoplanets: Also known as M-planets, these have mean global surface temperature in the range of 0 °C to 50 °C, which is a necessary condition for the survival of complex terrestrial life. The planetary bodies with sizes smaller than Mercury and larger than Ceres fall in this category, and these are generally referred to as Earthlike planets.
2. Psychroplanets: Planets in this category have mean global surface temperature in the range of −50 °C to 0 °C. This is much colder than optimal for terrestrial life to sustain.
3. **Nonhabitable**: Those planets which do not belong to either of the above two categories fall in this category. These planets do not have necessary thermal properties for sustaining life.

The three remaining classes in the data are—thermoplanet, hyperthermoplanet, and hypopsychroplanet. However, owing to highly limited number of samples in each of these classes, we ignore these classes. While running the classification methods, we consider multiple attributes of the parent Sun of the exoplanets that include mass, radius, effective temperature, luminosity, and the limits of the habitable zone. The first step consists of preprocessing data from PHL-EC. An important challenge in the dataset is that a total of about 1% of the data is missing (with a majority being of the attribute P. Max Mass) and in order to overcome this, we used a simple method of removing instances with missing data after extracting the appropriate attributes for each experiment, as most of the missing data is from the nonhabitable class. We considered another dataset where a subset of attributes (restricted attributes) consisting of Planet Minimum Mass, Mass, Radius, SFlux Minimum, SFlux Mean, SFlux Maximum are used as input. This subset of attributes do not consider surface temperature and any attribute related to surface temperature at all, making the decision boundaries more complicated to decipher. Following this, the ML approaches were used on these preprocessed datasets. The online data source for the current work is available at http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database.

B. Results and discussion

We compare the proposed one layer `ChaosNet` TT-SS method with the following ML algorithms: Decision Tree,[54] K-Nearest Neighbor (KNN[55]), Support Vector Machine (SVM[56]), and Deep Learning algorithm (DL, two layers[57]). The parameters used in ML algorithms are provided in the supplementary material. We have used "Scikit-learn"[58] and "Keras"[59] package for the implementation of ML algorithms and two layer neural network, respectively.

Performance of `ChaosNet` *TT-SS Method on MNIST data*: Fig. 7 shows the comparative performance of the `ChaosNet` TT-SS method with SVM, Decision Tree, KNN, and DL (two layer) on the MNIST dataset. In the case of MNIST data `ChaosNet`, the TT-SS method outperforms classical ML techniques like SVM, Decision Tree and KNN. The `ChaosNet` TT-SS method gave slightly higher performance than DL upto training with eight samples per class. As the number of training samples increased (beyond eight), DL outperforms the `ChaosNet` TT-SS method.

Performance of `ChaosNet` *TT-SS Method on KDDCup'99 data*: Fig. 8 shows the comparative performance of the `ChaosNet` TT-SS method with SVM, Decision Tree, KNN and DL (two layer) on KDDCup'99 dataset. In the low training sample regime for KDDCup'99 data, the `ChaosNet` TT-SS method outperforms the classical ML and DL algorithms except for training with five samples/class, where Decision Tree outperforms the `ChaosNet` TT-SS method.

Performance of `ChaosNet` *TT-SS Method on Iris data*: Fig. 9 shows the comparative performance of the `ChaosNet` TT-SS method with SVM, Decision Tree, KNN, and DL (two layer) on Iris

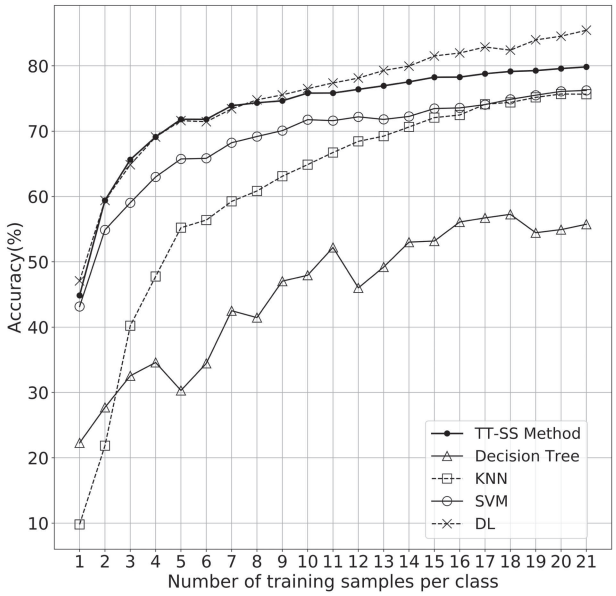

FIG. 7. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for MNIST dataset in the low training sample regime.

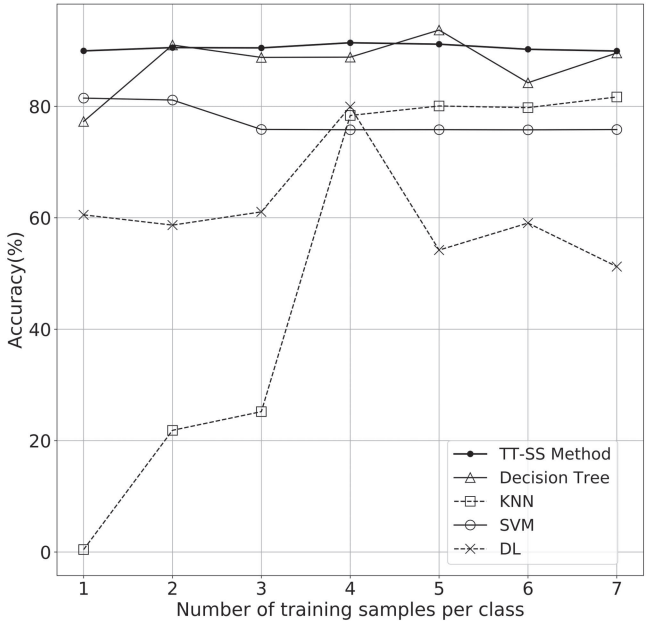

FIG. 8. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for KDDCup'99 dataset in the low training sample regime.

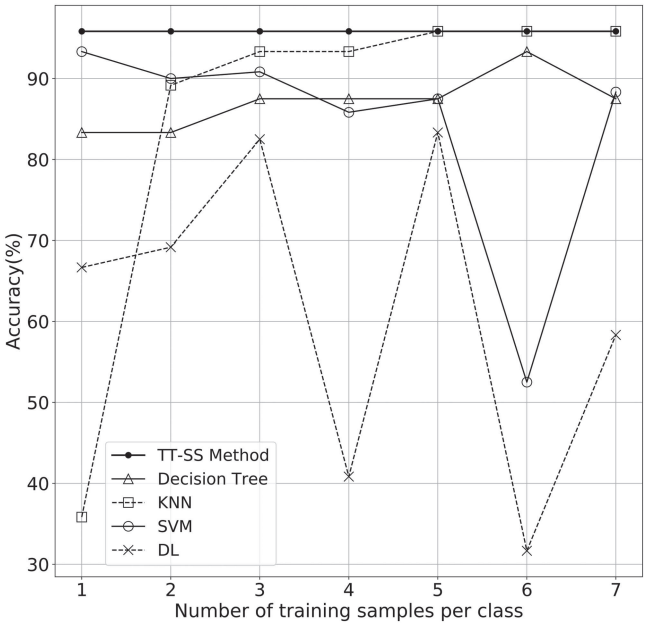

FIG. 9. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for Iris dataset in the low training sample regime.

dataset. In the case of Iris data, the `ChaosNet` TT-SS method gives consistently the best results when trained with $1, 2, \ldots, 7$ samples per class.

Performance of `ChaosNet` *TT-SS Method on Exoplanet data*: Fig. 10 shows the comparative performance of the `ChaosNet` TT-SS method with SVM, Decision Tree, KNN, and DL (two layer) on Exoplanet dataset. We consider three classes from the exoplanet data, namely, Nonhabitable, Mesoplanet, and Psychroplanet. There are a total of 45 attributes to explore the classification of three types of exoplanet. We have considered another scenario where surface temperature (Fig. 11) is removed from the set of attributes. This makes the classification problem harder as the decision boundaries between classes become fuzzy in the absence of surface temperature. Additionally, a restricted set of attributes (Fig. 12) is considered where the direct/indirect influence of surface temperature is mitigated by removing all related attributes from the original full set of attributes. This makes habitability classification an incredibly complex task. Even though the literature is repleted with possibilities of using both supervised and unsupervised learning methods, the soft margin between classes, namely, psychroplanet and mesoplanet makes the task of discrimination incredibly difficult. This has perhaps resulted in very few published work on automated habitability classification. A sequence of recent explorations by Saha *et al.* (2018)[53] expanding previous work by Bora *et al.*[60] on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data raises important questions.

In our study, independent trials of training is done with very less samples: $1, 2, \ldots, 7$ randomly chosen data samples per class. The algorithm is then tested on unseen data for each of the independent

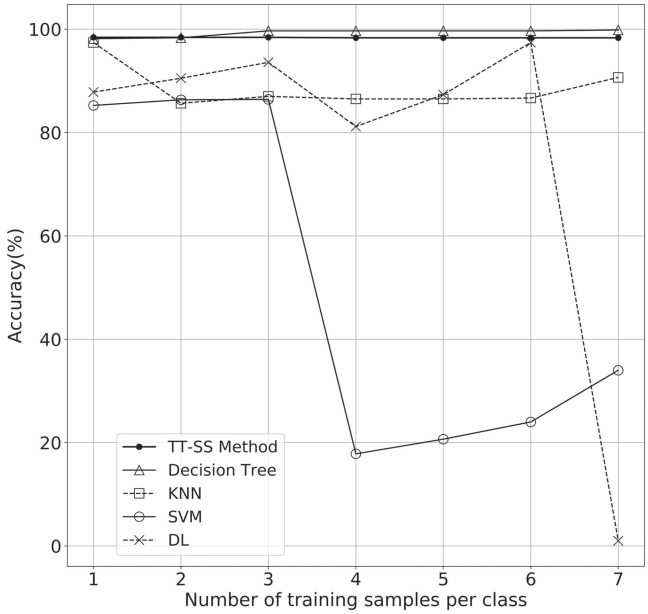

FIG. 10. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for Exoplanet dataset in the low training sample regime.

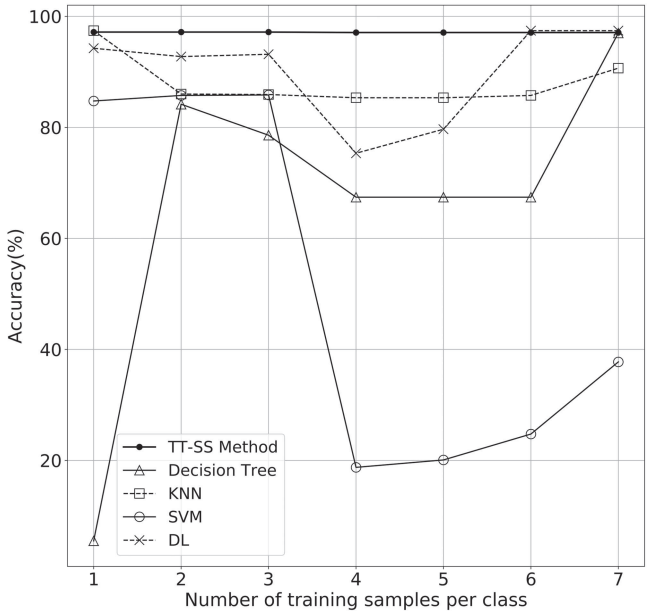

FIG. 11. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for Exoplanet dataset with no surface temperature in the low training sample regime.

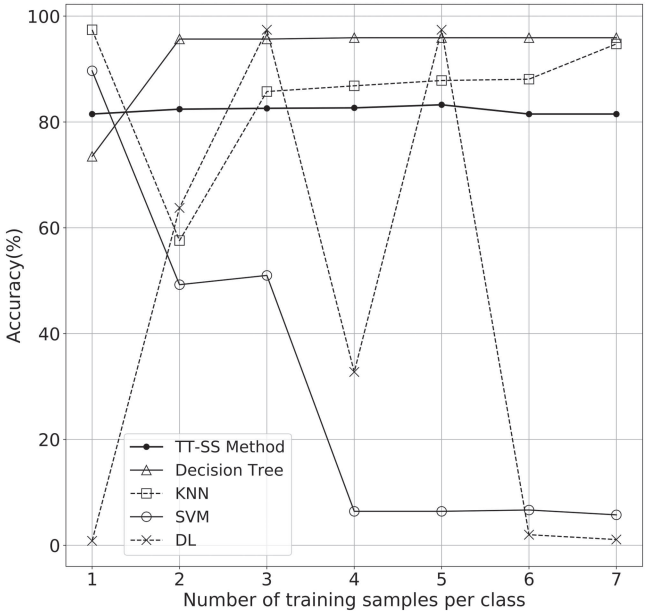

FIG. 12. Performance comparison of the ChaosNet TT-SS algorithm with DL (two layers), Decision Tree, SVM, and KNN for Exoplanet dataset with six restricted attributes in the low training sample regime.

trials. This accounts for the efficacy of the algorithm in detecting variances in new data. A consistent performance of the ChaosNet TT-SS method is observed in the low training sample regime. The ChaosNet TT-SS method gives the second highest performance in terms of accuracy when compared to SVM, KNN, and DL. The highest performance is given by Decision Tree. Despite the sample bias due to the nonhabitable class, we were able to achieve remarkable accuracies with the proposed algorithms without having to resort to under sampling and synthetic data augmentation. Additionally, the performance of ChaosNet TT-SS is consistent compared to other methods used in the analysis.

C. Single layer ChaosNet TT-SS algorithm in the presence of noise

One of the key ways to identify the robustness of a ML algorithm is by testing its efficiency in classification or prediction in the presence of noise. Robustness needs to be tested under the following scenarios: noisy test data, training data attributes affected by noise, inaccurate training data labels due to the influence of noise and noise affected model parameters.[61] Among these, "noise affected model parameters" is the most challenging since it can significantly impact performance of the algorithm. We consider a scenario where the parameters learned by the model while training are passed through a noisy channel. As an example, we compare the performance of the single layer ChaosNet TT-SS algorithm and two layer neural network (DL architecture) for Iris data. The parameters for the both algorithms are modified by Additive White Gaussian Noise (AWGN) with zero mean and increasing variance. The Iris dataset consists of

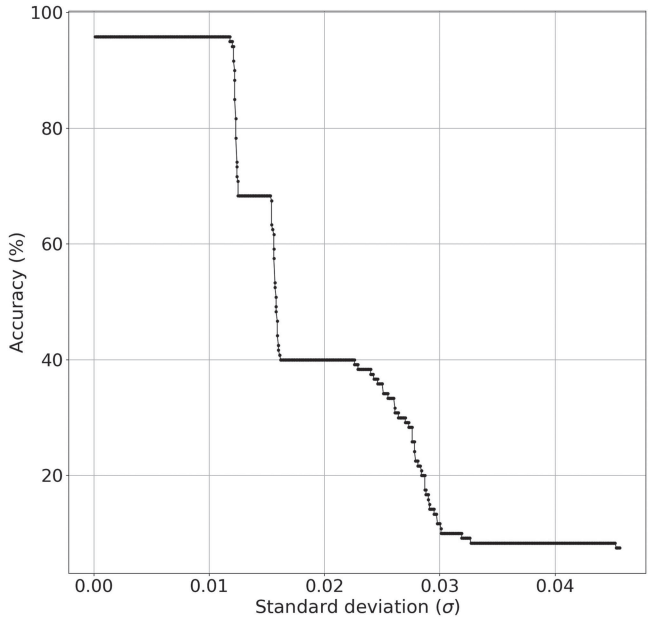

FIG. 13. Parameter noise analysis: accuracy of single-layer TT-SS algorithm on ChaosNet for Iris data in the presence of AWGN noise with zero mean and increasing standard deviation (σ).

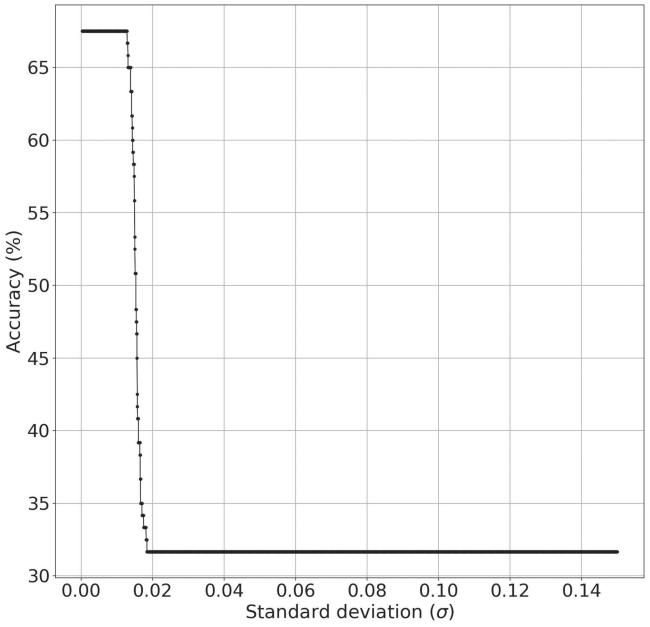

FIG. 14. Parameter noise analysis: Accuracy of the two layer neural network for Iris data in the presence of AWGN noise with zero mean and increasing standard deviation (σ).

three classes with four attributes per data instance. We considered the specific case of training with only seven samples per class.

Parameters settings for the single layer TT-SS algorithm implemented on `ChaosNet` *for Iris data:* Corresponding to the three classes for the Iris data, the output layer of `ChaosNet` consists of three nodes O_1, O_2, and O_3 which store the mean representation vectors. Since each representation vector contains four components (corresponding to the four input attributes), the total number of learnable parameters is 12. These parameters are passed through a channel corrupted by AWGN with zero mean and increasing standard deviation (from 0.0001 to 0.0456). This results in a variation of the Signal-to-Noise Ratio (SNR) from −6.98 dB to 45.36 dB.

Parameter settings for the two layer neural network for Iris data: The two layer neural network has four nodes in the input layer, four neurons in the hidden layer, and three neurons in the output layer. Thus, the total number of learnable parameters (weights and biases) for this architecture are: $(4 \times 4) + 4 + (3 \times 4) + 3 = 35$. These parameters are passed through a channel corrupted by AWGN with zero mean and increasing standard deviation (from 0.0003 to 0.15). This results in a variation of the Signal-to-Noise Ratio (SNR) from −8.78 dB to 45.48 dB.

Parameter Noise Analysis: The results corresponding to additive gaussian parameter noise for the `ChaosNet` TT-SS algorithm and two layer neural network are provided in Figs. 13 and 14, respectively. Figures 15 and 16 depict the variation of SNR (dB) with σ of the AWGN for the two algorithms. First, we can observe that the performance of the `ChaosNet` TT-SS algorithm degrades gracefully with increasing σ, whereas for the two layer neural network there is a sudden and drastic fall in performance (as $\sigma > 0.02$). A closer observation of the variation of accuracy for different SNRs is provided in Table I. In the case of the one layer `ChaosNet` TT-SS method, for the SNR in the range 4.79 dB to 45.36 dB, the accuracy remains unchanged at 95.83%. Whereas for the same SNR,

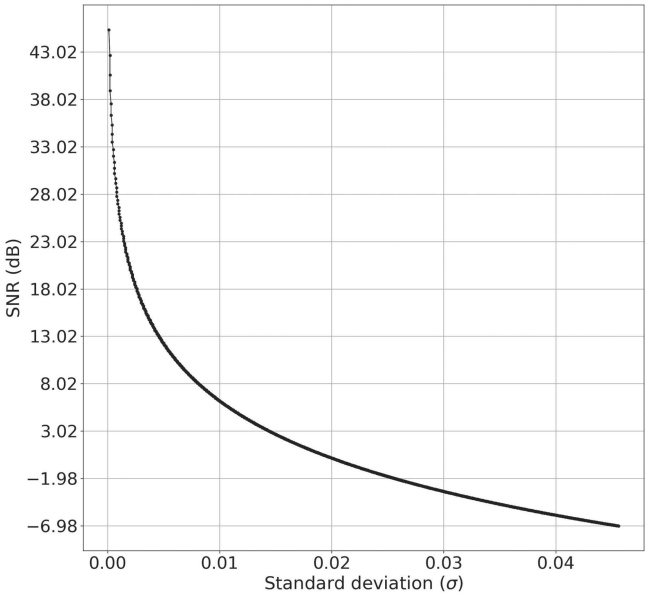

FIG. 15. SNR vs standard deviation (σ) of AWGN corresponding to Fig. 13.

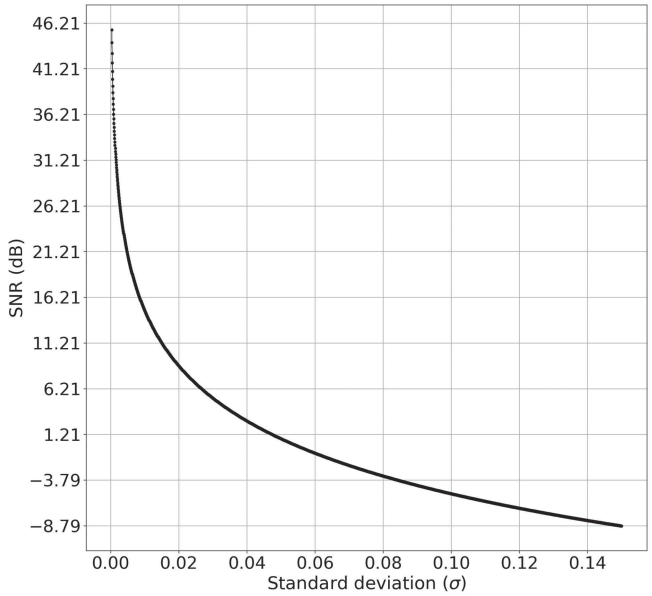

FIG. 16. SNR vs standard deviation (σ) of AWGN corresponding to Fig. 14.

the accuracy for the two layer neural network reduced from 67.5% to 31.66%. This preliminary analysis on parameter noise indicates the better performance of our method when compared with two layer neural network. However, more extensive analysis will be performed for other datasets and other noise scenarios in the near future.

V. MULTILAYER ChaosNet TT-SS ALGORITHM

So far, we have discussed the TT-SS algorithm implemented on the single-layer ChaosNet architecture. In this section, we investigate the addition of hidden layers to the ChaosNet architecture. A three layer ChaosNet with two hidden layers and one output layer is depicted in Fig. 17. It consists of an input layer with n-GLS neurons $\{C_1, C_2, \ldots, C_n\}$, and two hidden layers with GLS neurons $\{H_1^{[1]}, H_2^{[1]}, \ldots, H_r^{[1]}\}$ and $\{H_1^{[2]}, H_2^{[2]}, \ldots, H_k^{[2]}\}$, respectively. Let the neural activity of the input layer GLS neurons be represented by the chaotic trajectories $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n$ with firing times $N_1, N_2, \ldots, N_n$, respectively, and $N_{max} = \max\{N_i\}_{i=1}^n$ denotes the maximum firing time. The jth GLS neuron in the hidden layer has its own intrinsic dynamics starting with an initial neural activity of $q_j^{[1]}$ (represented by a self-connection with a coupling constant γ) and potentially connected to every GLS neuron in the input layer (represented by coupling constant η).

The coupling coefficient connecting the ith GLS neuron in the input layer to the jth GLS neuron in the first hidden layer of the multilayer ChaosNet architecture is $\eta_{i,j}^{[1]}$. The output of the jth GLS neuron of the first hidden layer ($H_j^{[1]}(t), j = 1, 2, \ldots, r$) is as follows:

$$H_j^{[1]}(t) = \begin{cases} \sum_{i=1}^n \eta_{i,j}^{[1]} A_i(t) + \gamma_{j,j}^{[1]} q_j^{[1]}, & t = 0, \\ \sum_{i=1}^n \eta_{i,j}^{[1]} A_i(t) + \gamma_{j,j}^{[1]} T(H_j^{[1]}(t-1)), & 0 < t \le N_{max}, \end{cases} \tag{5}$$

where $\sum_{i=1}^n \eta_{i,j}^{[1]} + \gamma_{j,j}^{[1]} = 1$, $q_j^{[1]} \in (0,1)$ and $\forall\ t > N_i$, $A_i(t) = 0$. The output of the jth neuron of the second hidden layer ($H_j^{[2]}(t)$, $j = 1, 2, \ldots, k$) is as follows:

$$H_j^{[2]}(t) = \begin{cases} \sum_{i=1}^r \eta_{i,j}^{[2]} H_i^{[1]}(t) + \gamma_{j,j}^{[2]} q_j^{[2]}, & t = 0, \\ \sum_{i=1}^r \eta_{i,j}^{[2]} H_i^{[1]}(t) + \gamma_{j,j}^{[2]} T(H_j^{[2]}(t-1)), & 0 < t \le N_{max}, \end{cases} \tag{6}$$

where $\sum_{i=1}^r \eta_{i,j}^{[2]} + \gamma_{j,j}^{[2]} = 1, q_j^{[2]} \in (0,1)$. In the above equations, $T(\cdot)$ represents the 1D chaotic GLS map. From the output of the second hidden layer, the TT-SS features are extracted from the k GLS neurons which are subsequently passed to the output layer for the computation and storage of the mean representation vectors corresponding to the s classes.

In the multilayer ChaosNet TT-SS method, ηs and γs are the additional hyperparameters. The above algorithm can be extended in a straightforward manner for more than two hidden layers.

A. Multilayer ChaosNet TT-SS method for exoplanet dataset

We have implemented the multilayer ChaosNet TT-SS method for the Exoplanet dataset with two layers (one hidden layer and one output layer). The number of GLS neurons in the input and first hidden layer are $n = 45$ and $r = 23$, respectively. The output layer consists of $s = 3$ nodes as it is a three class classification problem (Mesoplanets, Psychroplanets, and Nonhabitable). Every neuron in the first hidden layer is connected to only two neurons in the input layer (except the last neuron which is connected to only one neuron in the input layer). The output of the GLS neurons in the hidden layer

TABLE I. Parameter noise analysis for AWGN noise: comparison of accuracies for one layer ChaosNet TT-SS method and two layer neural network for various SNR ranges.

SNR (dB) (one layer TT-SS)	4.79–45.36	4.61–4.76	4.41–4.59	4.27–4.39
Accuracy (%) (one layer TT-SS)	95.83	95.00	81.66–94.16	70.83–78.33
SNR (dB) (two layer NN)	12.56–45.48	11.94–12.36	11.57–11.91	10.65–11.54
Accuracy (%) (two layer NN)	67.50	65.00	60.00–63.33	40.83–59.16

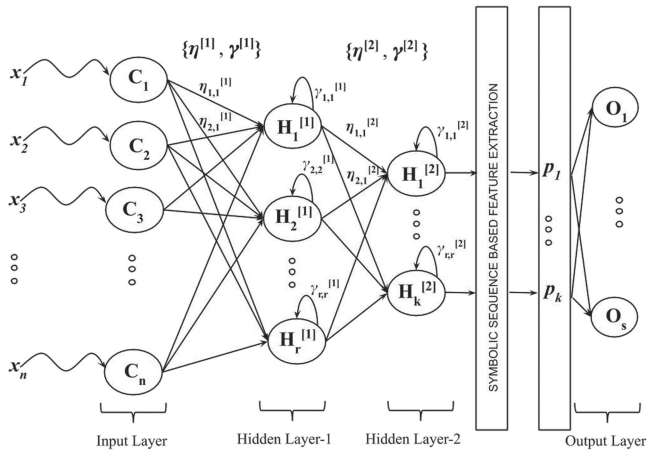

FIG. 17. Multilayer ChaosNet architecture with two hidden layers and one output layer. Here, the ηs and γs are the additional hyperparameters. There is a dimensionality reduction in the representation vectors (if $k < n$).

is given by

$$H_j^{[1]}(t) = \begin{cases} \eta_{2j-1,j}^{[1]} A_{2j-1}(t) + \eta_{2j,j}^{[1]} A_{2j}(t) \\ \quad +\gamma_{j,j}^{[1]} q_j^{[1]}, & t = 0, \\ \eta_{2j-1,j}^{[1]} A_{2j-1}(t) + \eta_{2j,j}^{[1]} A_{2j}(t) \\ \quad +\gamma_{j,j}^{[1]} T(H_j^{[1]}(t-1)), & 0 < t \le N_{max}, \end{cases} \tag{7}$$

for $j = 1, 2, \ldots, 22$. To the last neuron of first hidden layer, the input value is passed as such. The hyperparameters used in the classification task are $\eta = 0.4995$ and $\gamma = 0.001$, $q_j^{[1]} = 0.56$; initial neural activity q, internal discrimination threshold (b), type of GLS neuron used and ε chosen for Exoplanet classification task are the same as in Table II (fourth row).

From Fig. 18, the two layer ChaosNet TT-SS method has slightly improved the accuracy of Exoplanet classification task over

TABLE II. Hyperparameter settings—initial neural activity, discrimination threshold value, type of GLS-Neuron, and value of ε used in the study.

Dataset	Initial neural activity (q)	Discrimination threshold (b)	GLS-Neuron type	ε
MNIST	0.3360	0.331 000 0	$T_{Skew-Binary}$	0.01
KDDCup'99	0.6000	0.335 000 0	$T_{Skew-Tent}$	0.01
Iris	0.6000	0.986 755 6	$T_{Skew-Binary}$	0.01
Exoplanet	0.2624[a]	0.149 000 0	$T_{Skew-Tent}$	0.01
Exoplanet[b]	0.2624[c]	0.149 000 0	$T_{Skew-Tent}$	0.01
Exoplanet[d]	0.9500[e]	0.476 000 0	$T_{Skew-Tent}$	0.001

[a]Actual value was 0.262 424 242 424 242 45.
[b]Exoplanet with no surface temperature.
[c]Actual value was 0.262 424 242 424 242 45.
[d]Exoplanet with restricted attributes.
[e]Actual value was 0.950 000 000 000 000 6.

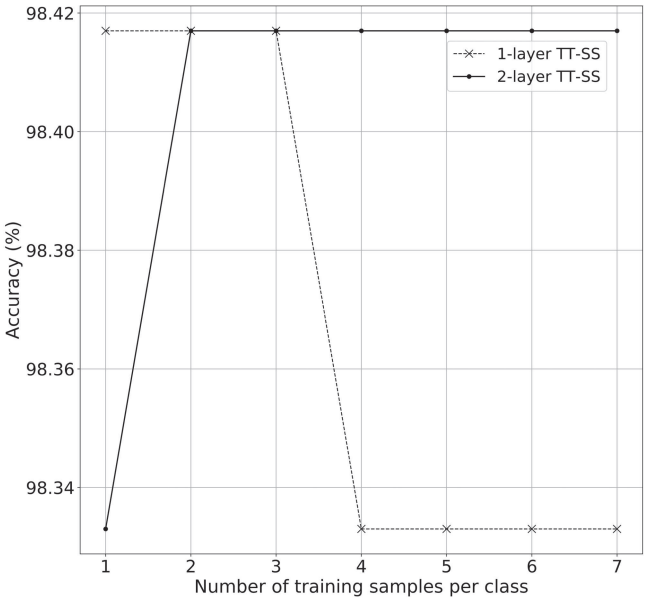

FIG. 18. Multilayer ChaosNet TT-SS method for Exoplanet dataset. Classification accuracy vs number of training samples per class for one layer TT-SS and two layer TT-SS methods.

that of the one layer ChaosNet TT-SS method for four and higher number of training samples per class. There is a 50% reduction in the dimensionality of the representation vectors (at the cost of increase in the number of hyperparameters). While these preliminary results are encouraging, more extensive testing of the multilayer ChaosNet TT-SS method with fully connected layers (and more than one hidden layer) need to be performed in the future.

VI. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

State-of-the-art performance on classification tasks reported by algorithms in the literature is typically for 80%–20% or 90%–10% training-testing split of the datasets. The performance of these algorithms will dip considerably as the number of training samples reduces. ChaosNet demonstrates (one layer TT-SS method) consistently good (and reasonable) performance accuracy in the low training sample regime. Even with as low as seven (or fewer) training samples/class (which accounts for less than 0.05% of the total available data), ChaosNet yields performance accuracies in the range of 73.89%–98.33%.

Future work includes determining optimal hyperparameter settings to further improve accuracy, testing on more datasets and classification tasks, extension to predictive modeling and incorporating robustness into GLS neurons to external noise. The multilayer ChaosNet architecture presents a number of avenues for further research such as determining optimal number of layers, type of coupling (unidirectional and bidirectional) between layers, homogeneous and heterogeneous layers (successive layers can have neurons with different 1D chaotic maps), coupled map lattices, use of 2D

and higher dimensional chaotic maps and even flows in the architecture and exploring properties of chaotic synchronization in such networks.

Highly desirable features such as Shannon optimal lossless compression, computation of logical operations (XOR, AND, etc.), universal approximation property and topological transitivity—all thanks to the chaotic nature of GLS neurons—makes `ChaosNet` a potentially attractive ANN architecture for diverse applications (from memory encoding for storage and retrieval purposes to classification tasks). We expect design and implementation of novel learning algorithms on the `ChaosNet` architecture in the near future that can efficiently exploit these wonderful properties of chaotic GLS neurons.

The code for the proposed `ChaosNet` architecture (TT-SS method) is available at https://github.com/HarikrishnanNB/ChaosNet.

SUPPLEMENTARY MATERIAL

See the supplementary material for (1) algorithms for the one layer TT-SS Method implemented on `ChaosNet` and (2) hyperparameters that we have used for the machine learning algorithms for generating the results.

ACKNOWLEDGMENTS

H.N.B. thanks "The University of Trans-Disciplinary Health Sciences and Technology (TDU)" for permitting this research as part of the Ph.D. program. Aditi Kathpalia is grateful to the Manipal Academy of Higher Education for permitting this research as a part of the Ph.D. program. The authors gratefully acknowledge the financial support of Tata Trusts. N. N. dedicates this work to late Professor Prabhakar G. Vaidya who initiated him to the fascinating field of Chaos Theory. This research has also made use of the PHL's Exoplanet Catalog, maintained by the Planetary Habitability Laboratory at the University of Puerto Rico at Arecibo, and NASA Astrophysics Data System Abstract Service. Snehanshu Saha would like to thank the Science and Engineering Research Board (SERB), Department of Science and Technology, Government of India, for supporting our research by providing resources to conduct experiments (Project Reference No. EMR/2016/005687).

REFERENCES

[1] P. Faure and H. Korn, "Is there chaos in the brain? I. Concepts of nonlinear dynamics and methods of investigation," C. R. de l'Académie des Sci. Ser. III Sci. de la Vie **324**, 773–793 (2001).
[2] H. Korn and P. Faure, "Is there chaos in the brain? II. Experimental evidence and related models," C. R. Biol. **326**, 787–840 (2003).
[3] Y. Fan and A. V. Holden, "Bifurcations, burstings, chaos and crises in the Rose-Hindmarsh model for neuronal activity," Chaos, Solitons Fractals **3**, 439–449 (1993).
[4] Y. Ding, J. H. Sohn, M. G. Kawczynski, H. Trivedi, R. Harnish, N. W. Jenkins, D. Lituiev, T. P. Copeland, M. S. Aboian, C. Mari Aparici *et al.*, "A deep learning model to predict a diagnosis of Alzheimer disease by using 18f-FDG PET of the brain," Radiology **290**, 456–464 (2018).
[5] N. Harikrishnan, R. Vinayakumar, and K. Soman, "A machine learning approach towards phishing email detection," in *Proceedings of the Anti-Phishing Pilot at ACM International Workshop on Security and Privacy Analytics (IWSPA AP)* (2018), Vol. 2013, pp. 455–468.
[6] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2013), pp. 6645–6649.
[7] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox, "On the information bottleneck theory of deep learning," in *International Conference on Learning Representations* (2018).
[8] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 IEEE Information Theory Workshop (ITW)* (IEEE, 2015), pp. 1–5.
[9] C. B. Delahunt and J. N. Kutz, "Putting a bug in ML: The moth olfactory network learns to read MNIST," preprint arXiv:1802.05405 (2018).
[10] K. Aihara, T. Takabe, and M. Toyoda, "Chaotic neural networks," Phys. Lett. A **144**, 333–340 (1990).
[11] N. Crook and T. O. Scheper, "A novel chaotic neural network architecture," in *ESANN'2001 Proceedings—European Symposium on Artificial Neural Networks Bruges (Belgium), 25–27 April 2001* (D-Facto, 2001), pp. 295–300.
[12] W. J. Freeman *et al.*, *Mass Action in the Nervous System* (Academic Press, 1975), Vol. 2004.
[13] H.-J. Chang and W. J. Freeman, "Parameter optimization in models of the olfactory neural system," Neural Netw. **9**, 1–14 (1996).
[14] R. Kozma and W. J. Freeman, "A possible mechanism for intermittent oscillations in the kiii model of dynamic memories-the case study of olfaction," in *IJCNN'99 International Joint Conference on Neural Networks (Cat. No. 99CH36339)* (IEEE, 1999), Vol. 1, pp. 52–57.
[15] I. Tsuda, "Dynamic link of memory–chaotic memory map in nonequilibrium neural networks," Neural Netw. **5**, 313–326 (1992).
[16] J. S. Nicolis and I. Tsuda, "Chaotic dynamics of information processing: The "magic number seven plus-minus two" revisited," Bull. Math. Biol. **47**, 343–365 (1985).
[17] K. Kaneko, "Lyapunov analysis and information flow in coupled map lattices," Phys. D Nonlinear Phenom. **23**, 436–447 (1986).
[18] K. Kaneko, "Clustering, coding, switching, hierarchical ordering, and control in a network of chaotic elements," Phys. D Nonlinear Phenom. **41**, 137–172 (1990).
[19] A. Kathpalia and N. Nagaraj, "A novel compression based neuronal architecture for memory encoding," in *Proceedings of the 20th International Conference on Distributed Computing and Networking* (ACM, 2019), pp. 365–370.
[20] Z. Aram, S. Jafari, J. Ma, J. C. Sprott, S. Zendehrouh, and V.-T. Pham, "Using chaotic artificial neural networks to model memory in the brain," Commun. Nonlinear Sci. Numerical Simul. **44**, 449–459 (2017).
[21] K. T. Alligood, T. D. Sauer, and J. A. Yorke, *Chaos* (Springer, 1996).
[22] Deterministic chaos is characterized by the " Butterfly Effect"—sensitive dependence of behavior to minute changes in initial conditions.
[23] A. Babloyantz and C. Lourenço, "Brain chaos and computation," Int. J. Neural Syst. **7**, 461–471 (1996).
[24] C. Barras, "Mind maths: Brainquakes on the edge of chaos," New Scientist **217**, 36 (2013).
[25] T. Elbert, B. Rockstroh, Z. J. Kowalik, M. Hoke, M. Molnar, J. E. Skinner, and N. Birbaumer, "Chaotic brain activity," Electroencephalogr. Clin. Neurophysiol./Suppl. **44**, 441–449 (1995).
[26] J. Sprott, "Is chaos good for learning?," Nonlinear Dyn. Psychol. Life Sci. **17**, 223–232 (2013).
[27] G. Baghdadi, S. Jafari, J. Sprott, F. Towhidkhah, and M. H. Golpayegani, "A chaotic model of sustaining attention problem in attention deficit disorder," Commun. Nonlinear Sci. Numer. Simul. **20**, 174–185 (2015).
[28] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," J. Phys. **117**, 500–544 (1952).
[29] J. L. Hindmarsh and R. Rose, "A model of neuronal bursting using three coupled first order differential equations," Proc. R. Soc. Lond. B Biol. Sci. **221**, 87–102 (1984).
[30] R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," Biophys. J. **1**, 445–466 (1961).
[31] J. Nagumo, S. Arimoto, and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," Proc. IRE **50**, 2061–2070 (1962).
[32] A. Zerroug, L. Terrissa, and A. Faure, "Chaotic dynamical behavior of recurrent neural network," Annu. Rev. Chaos Theory Bifurc. Dyn. Syst. **4**, 55–66 (2013).

[33]N. B. Harikrishnan and N. Nagaraj, "A novel chaos theory inspired neuronal architecture," preprint arXiv:1905.12601 (2019).
[34]R. C. Staudemeyer and C. W. Omlin, "Extracting salient features for network intrusion detection using machine learning methods," S. Afr. Comput. J. **52**, 82–96 (2014).
[35]K. Dajani and C. Kraaikamp, *Ergodic Theory of Numbers* (Cambridge University Press, 2002), Vol. 29.
[36]Let $X = \{x_0, x_1, x_2, \ldots\}$ be the trajectory of a chaotic map with initial condition x_0, where $x_i \in [U, V)$. The interval $[U, V)$ is partitioned into k sub intervals denoted as $I_0, I_1, \ldots, I_{k-1}$. If $x_i \in I_j$, then we denote x_i by the symbol $j \in \{0, 1, \ldots, k-1\}$ The new sequence of symbol $\{j_0, j_1, \ldots, j_{k-1}\}$ is the symbolic sequence of the trajectory of X.
[37]Generating Markov Partition or GMP is based on splitting the state space into a complete set of disjoint regions, namely, it covers all state space and enables associating a one-to-one correspondence between trajectories and itinerary sequences of symbols (L and R) without losing any information?.
[38]N. Nagaraj, "Novel applications of chaos theory to coding and cryptography," Ph.D. thesis (NIAS, 2008).
[39]N. Nagaraj, P. G. Vaidya, and K. G. Bhat, "Arithmetic coding as a nonlinear dynamical system," Commun. Nonlinear Sci. Numer. Simul. **14**, 1013–1020 (2009).
[40]N. Nagaraj, "Using cantor sets for error detection," PeerJ Comput. Sci. **5**, e171 (2019).
[41]K.-W. Wong, Q. Lin, and J. Chen, "Simultaneous arithmetic coding and encryption using chaotic maps," IEEE Trans. Circuits Syst. II Express Briefs **57**, 146–150 (2010).
[42]W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition* (Cambridge University Press, 2014).
[43]For a nonconstant matrix X, normalization is achieved by performing $\frac{X-\min(X)}{\max(X)-\min(X)}$. A constant matrix X is normalized to all ones.
[44]G. Cybenko, "Approximation by superpositions of a sigmoidal function," Math. Control Signals Syst. **2**, 303–314 (1989).
[45]Y. LeCun and C. Cortes, See http://yann.lecun.com/exdb/mnist/ for "MNIST Handwritten Digit Database" (2010).
[46]K. Cup, See http://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data for "Data (1999)" (1999).
[47]R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00* (IEEE, 2000), Vol. 2, pp. 12–26.
[48]See http://archive.ics.uci.edu/ml/datasets/iris for database for Iris data set.
[49]C. L. Blake and C. J. Merz, See http://www.ics.uci.edu/~mlearn/MLRepository.html for "UCI Repository of Machine Learning Databases" (1998).
[50]See http://phl.upr.edu/hec for "The habitable Exoplanet Catalog."
[51]A. M'endez, "The night sky of exo-planets," Hipparcos catalog (2011).
[52]S. Saha, N. Nagaraj, A. Mathur, and R. Yedida, "Evolution of novel activation functions in neural network training with applications to classification of exoplanets," preprint arXiv:1906.01975 (2019).
[53]S. Saha, S. Basak, M. Safonova, K. Bora, S. Agrawal, P. Sarkar, and J. Murthy, "Theoretical validation of potential habitability via analytical and boosted tree methods: An optimistic study on recently discovered exoplanets," Astron. Comput. **23**, 141–150 (2018).
[54]J. R. Quinlan, "Induction of decision trees," Mach. Learn. **1**, 81–106 (1986).
[55]T. M. Cover, P. Hart *et al.*, "Nearest neighbor pattern classification," IEEE Trans. Inf. Theory **13**, 21–27 (1967).
[56]M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," IEEE Intell. Syst. Appl. **13**, 18–28 (1998).
[57]Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature **521**, 436 (2015).
[58]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," J. Mach. Learn. Res. **12**, 2825–2830 (2011).
[59]F. Chollet *et al.*, See https://keras.io for "Keras" (2015).
[60]K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh, and A. Narasimhamurthy, "Cd-hpf: New habitability score via data analytic modeling," Astron. Computing **17**, 129–143 (2016).
[61]Hyperparameters are rarely subjected to noise and hence we ignore this scenario. It is always possible to protect the hyperparameters by using strong error correction codes.