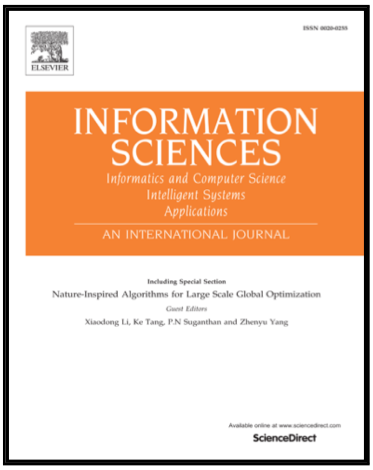
Accepted Manuscript



Parsimonious Random Vector Functional Link Network for Data Streams

Mahardhika Pratama, Plamen P. Angelov, Edwin Lughofer, Meng Joo Er

PII: S0020-0255(17)31112-X
DOI: 10.1016/j.ins.2017.11.050
Reference: INS 13278

To appear in: *Information Sciences*

Received date: 16 April 2017
Revised date: 22 November 2017
Accepted date: 24 November 2017

Please cite this article as: Mahardhika Pratama, Plamen P. Angelov, Edwin Lughofer, Meng Joo Er, Parsimonious Random Vector Functional Link Network for Data Streams, *Information Sciences* (2017), doi: 10.1016/j.ins.2017.11.050

Parsimonious Random Vector Functional Link Network for Data Streams

Mahardhika Pratama[a], Plamen P. Angelov[b], Edwin Lughofer[c], Meng Joo Er[d]

[a]School of Computer Science and Engineering, Nanyang Technological University, Singapore, 639798,Singapore
[b]School of Computing and Communication, Lancaster University, Lancaster, UK
[c]Department of Knowledge-based Mathematical Systems, Johannes Kepler University, Linz, Austria
[d]School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

Abstract

The majority of the existing work on random vector functional link networks (RVFLNs) is not scalable for data stream analytics because they work under a batch learning scenario and lack a self-organizing property. A novel RVLFN, namely the parsimonious random vector functional link network (pRVFLN), is proposed in this paper. pRVFLN adopts a fully flexible and adaptive working principle where its network structure can be configured from scratch and can be automatically generated, pruned and recalled from data streams. pRVFLN is capable of selecting and deselecting input attributes on the fly as well as capable of extracting important training samples for model updates. In addition, pRVFLN introduces a non-parametric type of hidden node which completely reflects the real data distribution and is not constrained by a specific shape of the cluster. All learning procedures of pRVFLN follow a strictly single-pass learning mode, which is applicable for online time-critical applications. The advantage of pRVFLN is verified through numerous simulations with real-world data streams. It was benchmarked against recently published algorithms where it demonstrated comparable and even higher predictive accuracies while imposing the lowest complexities.

Email addresses: mpratama@ntu.edu.sg (Mahardhika Pratama), p.angelov@lancaster.ac.uk (Plamen P. Angelov), edwin.lughofer@jku.at (Edwin Lughofer), emjer@ntu.edu.sg (Meng Joo Er)

Keywords: Random Vector Functional Link, Evolving Intelligent System, Online Learning, Online Identification, Randomized Neural Networks

1. Introduction

Significant growth of the problem space has led to a scalability issue for conventional machine learning approaches, which require iterating entire batches of data over multiple epochs. This phenomenon results in a strong demand for a simple, fast machine learning algorithm to be well-suited for deployment in numerous data-rich applications. This provides a strong case for research in the area of randomness in neural networks [5, 25], which was very popular in the late 80s and early 90s. This concept offers an algorithmic framework, which allows them to generate most of the network parameters randomly while still retaining reasonable performance [5]. One of the most prominent examples of randomness in neural networks is the random vector functional link network (RVFLN) which features solid universal approximation theory under strict conditions [7].

Due to its simple but sound working principle, randomness in neural networks has regained its popularity in the current literature [1, 26]. Nonetheless, the vast majority of works in the literature suffers from the issue of complexity which makes their computational complexity and memory burden prohibitive for data stream analytics since their complexities are manually determined and rely heavily on expert domain knowledge. The random selection of network parameters often causes the network complexity to go beyond what is necessary due to the existence of superfluous hidden nodes which contribute little to the generalization performance. Although the universal approximation capability of such an approach is assured only when sufficient complexity is selected, choosing a suitable complexity for a given problem entails expert-domain knowledge and is problem-dependent.

A novel RVFLN, namely the parsimonious random vector functional link network (pRVFLN), is proposed. pRVFLN combines the simple and fast working principles of RFVLN where all network parameters but the output weights are randomly generated with no tuning mechanism for hidden nodes. Since it characterises the online and adaptive nature of evolving intelligent systems, pRVFLN is capable of tracking any variations of data streams no matter how slow, rapid, gradual, sudden or temporal the drifts in data streams. It can initiate its learning structure from scratch with no initial structure and its structure is self-evolved from data streams in the one-pass learning mode by automatically adding, pruning and recalling its hidden nodes [24]. Furthermore, it is compatible for online real-time deployment because data streams are handled without revisiting previously seen samples. pRVFLN is equipped with a hidden node pruning mechanism which guarantees a low structural burden and the rule recall mechanism which aims to address cyclic concept drift. pRVFLN incorporates a dynamic input selection scenario which makes possible the activation and deactivation of input attributes on the fly and an online active learning scenario which rules out inconsequential samples from the training process. pRVFLN is a plug-and-play learner where a single training process encompasses all learning scenarios in a sample-wise manner without pre-and/or post-processing steps.

pRVFLN offers at least four novelties: 1) it introduces the interval-valued data cloud paradigm which is an extension of the data cloud in [4]. This modification aims to induce robustness in dealing with data uncertainty caused by noisy measurement, noisy data, etc. Unlike conventional hidden nodes, the interval-valued data cloud is parameter-free and requires no parametrization. It evolves naturally following the real data distribution; 2) an online active learning scenario based on the sequential entropy method (SEM) is proposed. The SEM is derived from the concept of neighbourhood probability [35] but here the concept of the data cloud is integrated. The data cloud concept simplifies the sample selection process because the neighbourhood probability is inferred with ease from the activation degree of the data cloud; 3) pRVFLN is capable of automatically generating its hidden nodes on the fly with the help of a type-2 self-constructing clustering (T2SCC) mechanism [36]. This rule growing process differs from existing approaches because the hidden nodes are created from the rule growing condition, which considers the locations of the data samples in the input space; 4) pRVFLN is capable of carrying out an online feature selection process, borrowing several concepts of online feature selection (OFS) [30]. The original version [30] is generalized here since it is originally devised for linear regression and calls for some modification to be a perfect fit for pRVLFN. The prominent trait of this method lies in a flexible online feature selection scenario, which makes it possible to select or deselect input attributes on demand by assigning crisp weights (0 or 1) to input features.

The effectiveness of pRVFLN was thoroughly evaluated using numerous real-world data streams and was benchmarked against recently published algorithms in the literature, with pRVFLN demonstrating a highly scalable approach for data stream analytics while retaining acceptable generalization performance. An analysis of the robustness of random intervals was performed. It is concluded that random regions should be carefully selected and should be chosen close to the true operating regions of a system being modelled. Moreover, we also present a sensitivity analysis of the predefined threshold and study the effect of learning components. Key mathematical notations are listed in Table 1.

The rest of this paper is structured as follows: related work is reviewed in Section 2; Section 3 elaborates basic concepts of pRVFLN, encompassing the principle of RVFLN and data cloud; network architecture of pRVFLN is discussed in Section 4; Section 5 explains the learning policy of pRVFLN; Numerical examples are presented in Section 6; conclusions are drawn in the last section of this paper.

2. Related Work

The concept of randomness in neural networks was initiated by Broomhead and Lowe in their work on radial basis function networks (RBFNs) [5]. A closed pseudo-inverse solution can be formulated to obtain the output weights of the RBFN and the centres of RBF units can be randomly sampled from data samples. This work later was generalized in [14], where the centres of the RBF neurons can be sampled from an independent distribution of the training data. The randomness in neural networks was substantiated by the findings of White [26], who developed a statistical test on hidden nodes. It was found that some nonlinear structures in the mapping function can be neglected without substantial loss of accuracy. In [26], the input weights of the hidden layers are randomly chosen. It is shown that the input weights are not sensitive to the overall learning performance.

A prominent contribution was made by Pao et al. with the random vector functional link network (RVFLN) [19]. This work presents a specific case of the functional link neural network [20], which embraces the concept of randomness in the functional link network. The universal approximation capability of the RVFLN is proven in [7] by formalising the Monte Carlo method approximating a limit-integral representation of a function. To attain the universal approximation capability, the hidden node should be chosen as either absolutely integrable or differentiable function. In practise, the region of random parameters should also be chosen carefully and the number of hidden nodes should be sufficiently large. There also exists another research direction in this area, namely reservoir computing (RC), which puts forward a recurrent network architecture in order to take into account temporal dependencies between subsequent patterns and in order to avoid dependencies on time-delayed input attributes [16]. Recent advances in the area of randomness in neural network are found in the seminal work by Wang and Li, Stochastic Configuration Networks (SCNs) [29]. This work presents theoretical contribution of random selection of neural network parameters under selective and constructive manner using a supervisory mechanism. This work starts from the fact that random sampling of neural network parameters highly influence the stability and convergence of neural network training. Improper scope settings for the random parameters may cause a neural network to lose its learning power. It is confirmed in analysis of robustness in Section 6.4 of this paper. Comprehensive survey of randomness in neural network can be found in [25].

Table 1: Key Mathematical Notations

Symbol	**Description**
$A_t \in \Re^n$	The input weight vector
β_t	The output of expansion layer
$X_t \in \Re^n$	The input attribute
$T_t \in \Re^m$	The target attribute
$x_e \in \Re^{(2n+1)\times 1}$	The expanded input vector
$w_i \in \Re^{(2n+1)\times 1}$	The output weight vector
$\tilde{G}_{i,temporal}$	The interval-valued temporal firing strength
$q \in \Re^m$	The design factor
$\lambda \in \Re^R$	The recurrent weight vector
$\widetilde{\mu}_i \in \Re^n$	The interval-valued local mean
$\widetilde{\Sigma}_i \in \Re^n$	The interval-valued mean square length
$\delta_i \in \Re^n$	The uncertainty factor
$H(N\|X_n)$	The entropy of neighborhood probability
$I_c(\tilde{\mu}_i, X_t)$	The input coherence
$O_c(\tilde{\mu}_i, X_t)$	The output coherence
$\zeta()$	The correlation measure
$\zeta(\tilde{G}_{i,temp}, T_t)$	The mutual information between $i-th$ rule and the target concept
$\Psi_i \in \Re^{(2n+1)\times(2n+1)}$	The output covariance matrix

The vast majority of RVFLNs in the literature are not compatible with online real-time learning situations. This issue led to the development of online learning in RVFLNs, which follows a single-pass learning concept [34]. The original version of RVFL is also applicable for online learning setting because it makes use of the conjugate gradient algorithm. Some modification need to be implemented and involve the use of stochastic gradient principle where the gradient is obtained for every sample and iteration over a number of epoch is not permitted. Nevertheless, this work is still built upon a fixed network structure which cannot evolve in accordance with up-to-date data trends. A concept of dynamic structure was offered in [12] by putting forward the notion of a growing structure. Notwithstanding their dynamic natures, concept drift remains an uncharted territory in these works because all parameters are chosen at random without paying close attention to the true data distribution. RC aims to address temporal system dynamics [16] but still does not consider a possible dramatic change of system behaviour.

3. Basic Concepts

This section outlines the foundations of pRVFLN encompassing the basic concept of RVFLN [19], the use of the Chebyshev polynomial as the functional expansion block [21] and the concept of data clouds [4].

3.1. Random Vector Functional Link Network

The idea of RVFLN was studied by Pao, Park and Sobajic in [19] and is one of the forms of the functional link network combined with the random vector approach [20]. It features the enhancement node performing the nonlinear transformation of input attributes as well as the direct connection of input attributes to the output node. The activation degree of the enhancement node along with the input attributes is combined with a set of output weights to generate the final network output. The RVFLN only leaves the weight vector to be fine-tuned during the training process while the other parameters are randomly sampled from a carefully selected scope. Suppose that there are R enhancement nodes and n input attributes, the size of the output weight vector is $W \in \Re^{(R+n)}$. The quadratic optimization problem is then formulated as follows:

$$E = \frac{1}{2N}\sum_{p=1}^{N}(t^{(p)} - Wd^{(p)})^2 \qquad (1)$$

where $W \in \Re^{(R+n)}$ is the output weight vector. $d^{(p)}$ is the output of the enhancement node and N is the number of samples. The RVFLN is similar to a single hidden layer feedforward network except for the fact that the hidden nodes function as an enhancement of the input feature and there exists direct connection from the input layer to the output layer. The steepest descent approach can be used to fine-tune the output weight vector. If matrix inversion using pseudo-inverse is feasible, a closed-form solution can be formulated. The generalization performance of RVFLN was examined in [19] and the RVFLNs convergence is also guaranteed to be attained within a number of iterations.

The RVFLN is a derivation of the functional link network [21]. That is, the hidden node or the enhancement node can be replaced by the functional expansion block generating a set of linearly independent functions of the entire input pattern. The functional expansion block can be formulated as trigonometric expansion [13], Chebyshev expansion, Legendre expansion, etc. [21] but our scope of discussion is limited to the Chebyshev expansion only due to its relevance to pRVFLN. Given the n-dimensional input vector $X = [x_1, x_2, ..., x_n] \in \Re^{1\times n}$ and its corresponding target variable y, the output of RVFLN with the Chebyshev functional expansion block is expressed as follows:

$$y = \sum_{j=1}^{2n+1} W_j \nu_j(A_n^T X_n + b_n) \tag{2}$$

where W_j is the output weight and $\nu_j()$ is the Chebyshev functional expansion mapping the n-dimensional input attribute and the input weight vector to the higher $2n+1$ expansion space. As with the original RVFLN, the output weight vector W_j can be learned using any optimization method while other parameters, A_n and b_n, are randomly generated. The $2n+1$ here results from the utilisation of the Chebyshev series up to the second order. The Chebyshev series is mathematically written as follows:

$$\nu_{order+1}(x) = 2(x)\nu_{order}(x) - \nu_{order-1}(x) \tag{3}$$

Because we are only interested in the Chebyshev series up to the second order, this results in $\nu_0(x) = 1, \nu_1(x) = x, \nu_2(x) = 2x^2 - 1$. Suppose that we deal with two dimensional input vector $_X = [x_1, x_2]$, the Chebyshev function expansion leads to $\nu = [1, \nu_1(x_1), \nu_2(x_1), \nu_1(x_2), \nu_2(x_2)]$. The advantage of the Chebyshev functional link compared to other popular functional links such as trigonometric [13], Legendre, power function, etc. [21] lies in its simplicity of

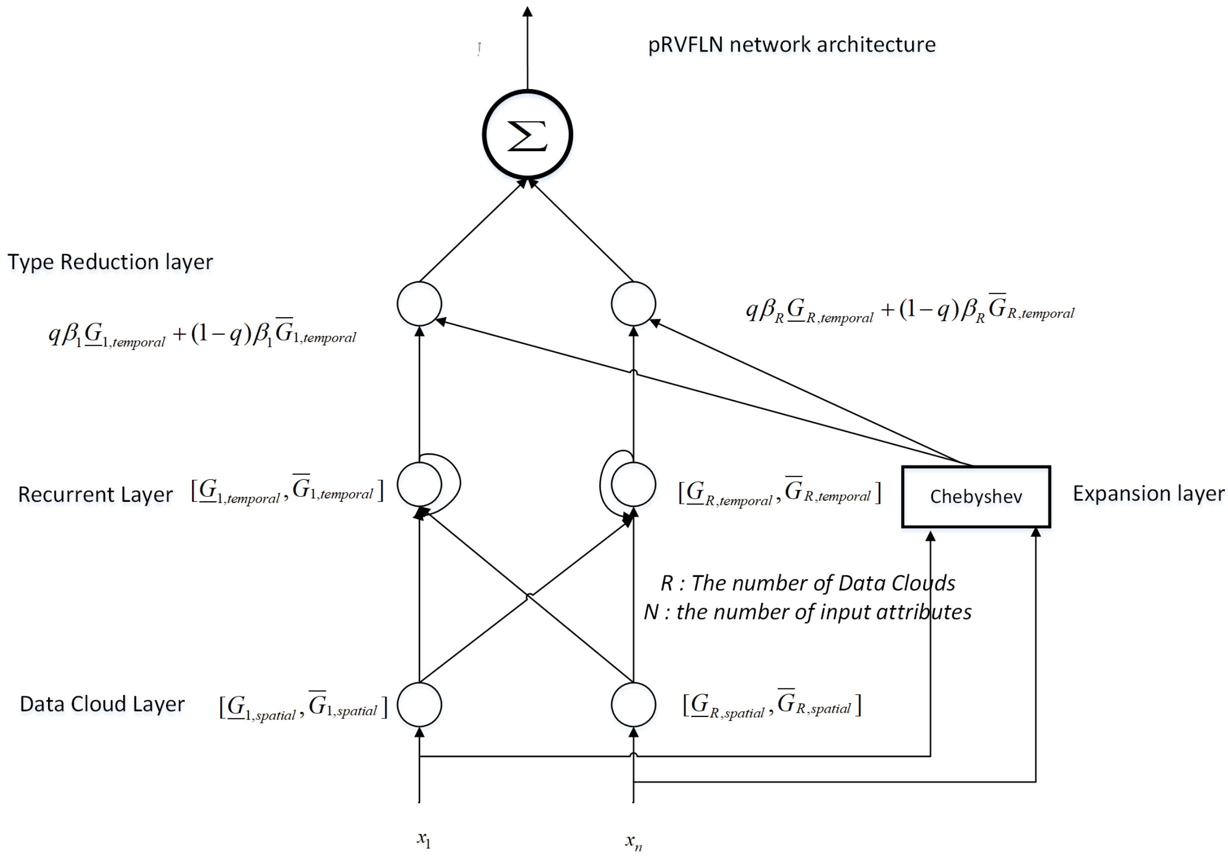

pRVFLN network architecture
Σ
Type Reduction layer
$q\beta_1\underline{G}_{1,temporal}+(1-q)\beta_1\overline{G}_{1,temporal}$
$q\beta_R\underline{G}_{R,temporal}+(1-q)\beta_R\overline{G}_{R,temporal}$
Recurrent Layer
$[\underline{G}_{1,temporal},\overline{G}_{1,temporal}]$
$[\underline{G}_{R,temporal},\overline{G}_{R,temporal}]$
Chebyshev
Expansion layer
R : The number of Data Clouds
N : the number of input attributes
Data Cloud Layer
$[\underline{G}_{1,spatial},\overline{G}_{1,spatial}]$
$[\underline{G}_{R,spatial},\overline{G}_{R,spatial}]$
x_1
x_n

Figure 1: Network Architecture of pRVFLN

computation. The Chebyshev function scatters fewer parameters to be stored into memory than the trigonometric function, while the Chebyshev function has a better mapping capability than the other polynomial functions of the same order. In addition, the polynomial power function is not robust against an extrapolation case. The functional expansion block can be also formed by using the Wavelet function [24] but it must be noted that the Wavelet function is sensitive to its initial values. It also requires a reliable tuning strategy to produce a good mapping of original input space.

3.2. Data Cloud

The concept of the data cloud offers an alternative to the traditional cluster concept where the data cloud is not shape-specific and evolves naturally in accordance with the true data distribution. It is also easy to use because it is non-parametric and does not require any parameterization. This strategy is desirable because parameterization per scalar variable often calls for complex high-level approximation and/or optimization. This approach was inspired by the idea of RDE and was integrated in the context of the TSK fuzzy system [4]. Unlike a conventional fuzzy system where a degree of membership is defined by a point-to-point distance, the data cloud computes an accumulated distance of the point of interest to all other points in the data cloud without physically keeping all data samples in the memory similar to the local data density. This notion has a positive impact on the memory and space complexity because the number of network parameters significantly reduces. The data cloud concept is formally written as:

$$\gamma_t^i = \frac{1}{1 + ||x_t - \mu_t^L||^2 + \Sigma_t^L - ||\mu_t^L||^2} \tag{4}$$

where γ_t^i denotes the *i-th* data cloud at the *t-th* observation. The data cloud evolves by updating the local mean μ_t^L and square length of *i-th* local region Σ_t^L as follows:

$$\mu_t^L = \frac{N_t^i - 1}{N_t^i}\mu_{t-1}^L + \frac{x_{t,N_i}}{N_t^i}, \mu_1^L = x_1 \tag{5}$$

$$\Sigma_t^L = \frac{N_t^i - 1}{N_t^i}\Sigma_{t-1}^L + \frac{||x_{t,N_i}||^2}{N_t^i}, \Sigma_1^L = ||x_1||^2 \tag{6}$$

where N_t^i denotes the number of samples associated to i-th cluster at the t-th observation. It is worth noting that these two parameters correspond to statistics of the *i-th* data cloud and are computed recursively with ease using standard recursive formulas. They do not impose a specific optimization or a specific setting to be performed to adjust their values.

4. Network Architecture of pRVFLN

pRVFLN utilises a local recurrent connection at the hidden node which generates the spatiotemporal activation degree. This recurrent connection is realized by a self-feedback loop of the hidden node which memorizes the previous activation degree and outputs a weighted combination between previous and current activation degrees spatiotemporal firing strength. In the literature, there exist at least three types of recurrent network structures

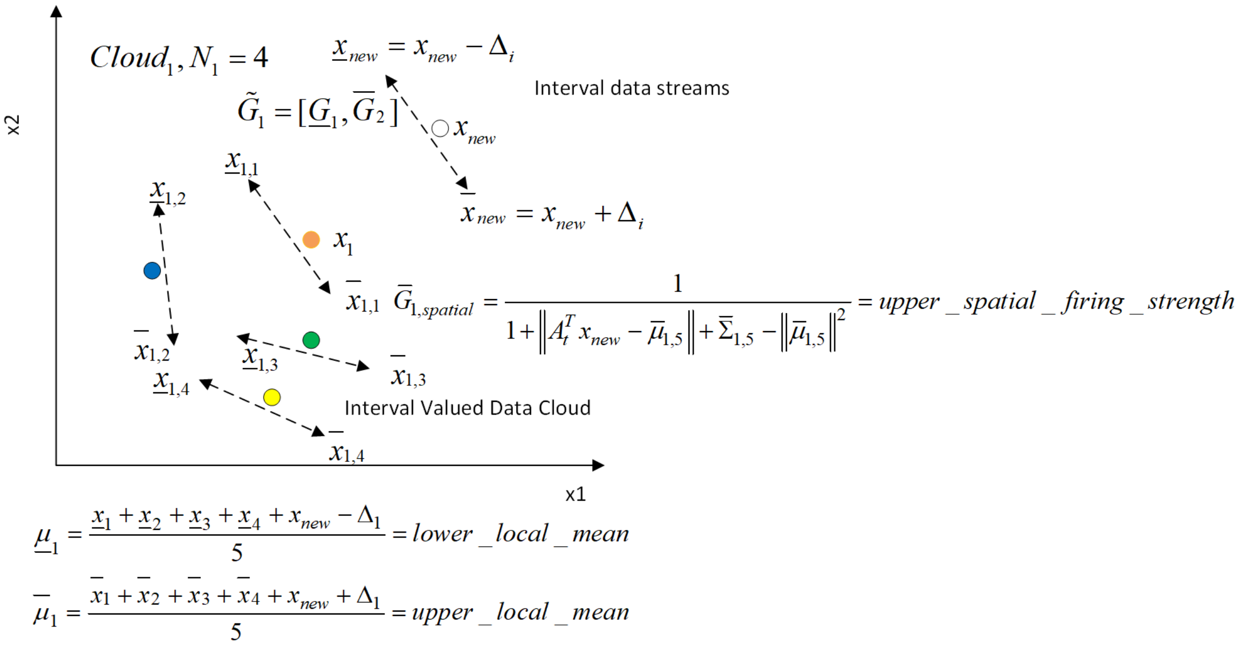

$Cloud_1, N_1 = 4$
$\underline{x}_{new} = x_{new} - \Delta_i$
Interval data streams
$\tilde{G}_1 = [\underline{G}_1, \overline{G}_2]$
x_{new}
$\underline{x}_{1,1}$
$\underline{x}_{1,2}$
$\overline{x}_{new} = x_{new} + \Delta_i$
x_1
$\overline{x}_{1,1}$
$\overline{G}_{1,spatial} = \frac{1}{1 + \left\| A_t^T x_{new} - \overline{\mu}_{1,5} \right\| + \overline{\Sigma}_{1,5} - \left\| \overline{\mu}_{1,5} \right\|^2} = upper\_spatial\_firing\_strength$
$\overline{x}_{1,2}$
$\underline{x}_{1,3}$
$\overline{x}_{1,3}$
$\underline{x}_{1,4}$
Interval Valued Data Cloud
$\overline{x}_{1,4}$
x2
x1
$\underline{\mu}_1 = \frac{\underline{x}_1 + \underline{x}_2 + \underline{x}_3 + \underline{x}_4 + x_{new} - \Delta_1}{5} = lower\_local\_mean$
$\overline{\mu}_1 = \frac{\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4 + x_{new} + \Delta_1}{5} = upper\_local\_mean$

Figure 2: Interval Valued Data Cloud

referring to its recurrent connections: global [9], interactive [13], and local [10], but the local recurrent connection is deemed to be the most compatible recurrent type in our case because it does not harm the local property, which assures stability when adding, pruning and fine-tuning hidden nodes. pRVFLN utilises the notion of the functional-link neural network where the expansion block is created by the Chebyshev polynomial up to the second order. Furthermore, the hidden layer of pRVFLN is built upon an interval-valued data cloud [4] where we integrate the idea of an interval-valued local mean into the data cloud.

The input coherence explores the similarity between new data and existing data clouds directly, while the output coherence focusses on their dissimilarity indirectly through a target vector as a reference. The input and output coherence formulates a test that determines the degree of confidence in the current hypothesis:

$$I_c(\tilde{\mu}_i, X_t) \leq \alpha_1, \;\; O_c(\tilde{\mu}_i, X_t) \geq \alpha_2 \qquad (7)$$

Suppose that a pair of data points (X_t, T_t) is received at t-th time instant where $X_t \in \Re^n$ is an input vector and $T_t \in \Re^m$ is a target vector, while n and m are respectively the number of input and output variables. Because pRVFLN works in a strictly online learning environment, it has no access to previously seen samples, and a data point is simply discarded after being learned. Due to the pre-requisite of an online learner, the total number of data N is assumed to be unknown. The output of pRVFLN is defined as follows:

$$y_o = \sum_{i=1}^{R} \beta_i \tilde{G}_{i,temporal}(A_t^T X_t + B_t), \widetilde{G}_{temporal} = [\underline{G}, \overline{G}] \tag{8}$$

where R denotes the number of hidden nodes and β_i stands for the i-th output of the functional expansion layer, produced by weighting the weight vector with an extended input vector $\beta_i = x_e^T w_i$. $x_e \in \Re^{(2n+1)\times 1}$ is an extended input vector resulting from the functional link neural network based on the Chebyshev function up to the second order [21] as shown in (3) and $w_i \in \Re^{(2n+1)\times 1}$ is a connective weight of the i-th output node. The definition of β_i is rather different from its common definition in the literature because it adopts the concept of the expansion block, mapping a lower dimensional space to a higher dimensional space with the use of certain polynomials. This paradigm produces the extended input vector x_e and here the Chebyshev polynomial expansion block up to the second order is used to produce the extended input vector as aforementioned in Section 3.1. Suppose that three input attributes are given $X = [x_1, x_2, x_3]$, the extended input vector is expressed as the Chebyshev polynomial up to the second order $x_e = [1, x_1, \nu_2(x_1), x_2, \nu_2(x_2), x_3, \nu(x_3)]$. Note that the term 1 here represents an intercept of the output node to avoid going through the origin, which may risk an untypical gradient. $A_t \in \Re^n$ is an input weight vector randomly generated from a certain range. The bias B_t is removed for simplicity. $\widetilde{G}_{i,temporal}$ is the *i-th* interval-valued data cloud, triggered by the upper and lower data cloud $\underline{G}_{i,temporal}, \overline{G}_{i,temporal}$. Note that recurrence is not seen in (8) because pRVFLN makes use of local recurrent layers at the hidden node. By expanding the interval-valued data cloud, the following is obtained:

$$y_o = \sum_{i=1}^{R} (1 - q_o)\beta_i \overline{G}_{i,temporal} + \sum_{i=1}^{R} q_o \beta_i \underline{G}_{i,temporal} \tag{9}$$

where $q \in \Re^m$ is a design factor to reduce an interval-valued function to a crisp one. It is worth noting that the upper and lower activation functions $\underline{G}_{i,temporal}, \overline{G}_{i,temporal}$ deliver spatiotemporal characteristics as a result of a local recurrent connection at the i-th hidden node, which combines the spatial and temporal firing strength of the i-th hidden node. These temporal activation functions output the following.

$$\underline{G}_{i,temporal}^{t} = \lambda_i \underline{G}_{i,spatial}^{t} + (1-\lambda_i)\underline{G}_{i,temporal}^{t-1},$$
$$\overline{G}_{i,temporal}^{t} = \lambda_i \overline{G}_{i,spatial}^{t} + (1-\lambda_i)\overline{G}_{i,temporal}^{t-1} \tag{10}$$

where $\lambda \in \Re^R$ is a weight vector of the recurrent link. The local feedback connection here feeds the spatiotemporal firing strength at the previous time step $\widetilde{G}_{i,temporal}^{t-1}$ back to itself and is consistent with the local learning principle. This trait happens to be very useful in coping with the temporal system dynamic because it functions as an internal memory component which memorizes a previously generated spatiotemporal activation function at $t-1$. Also, the recurrent network is capable of overcoming over-dependency on time-delayed input features and lessens strong temporal dependencies of subsequent patterns. This trait is desired in practise since it may lower the input dimension, because prediction is done based on the most recent measurement only. Conversely, the feedforward network often relies on time-lagged input attributes to arrive at a reliable predictive performance due to the absence of an internal memory component. This strategy at least entails expert knowledge for system order to determine the suitable number of delayed components.

The hidden node of the pRVFLN is an extension of the cloud-based hidden node, where it embeds an interval-valued concept to address the problem of uncertainty. Instead of computing an activation degree of a hidden node to a sample, the cloud-based hidden node enumerates the activation degree of a sample to all intervals in a local region on-the-fly. This results in local density information, which fully reflects real data distributions. This concept was defined in AnYa [4]. This concept is also the underlying component of TEDA-Class [11], all of which come from Angelov sound work of RDE [3]. This paper aims to modify these prominent works to the interval-valued case. Suppose that N_i denotes the support of the *i-th* data cloud, an activation degree of *i-th* cloud-based hidden node refers to its local density estimated recursively using the Cauchy function:

$$\widetilde{G}_{i,spatial} = \frac{1}{1+\sum\limits_{k=1}^{N_i}(\frac{\widetilde{x}_k - x_t}{N_i})},\ \widetilde{x}_k = [\underline{x}_{k,i}, \overline{x}_{k,i}],\ \widetilde{G}_{i,spatial} = [\underline{G}_{i,spatial}, \overline{G}_{i,spatial}] \tag{11}$$

where $\widetilde{x}_k$ is *k-th* interval in the *i-th* data cloud and x_t is *t-th* data sample. It is observed that (11) requires the presence of all data points seen so far. Its recursive form is formalised in [4] and is generalized here to the interval-valued case:

$$\begin{aligned}\overline{G}_{i,spatial} &= \frac{1}{1+||A_t^T x_t - \overline{\mu}_{i,N_i}||^2 + \overline{\Sigma}_{i,N_i} - ||\overline{\mu}_{i,N_i}||^2},\\ \underline{G}_{i,spatial} &= \frac{1}{1+||A_t^T x_t - \underline{\mu}_{i,N_i}||^2 + \underline{\Sigma}_{i,N_i} - ||\underline{\mu}_{i,N_i}||^2}\end{aligned} \tag{12}$$

where $\underline{\mu}_i, \overline{\mu}_i$ signify the upper and lower local means of the *i-th* cloud:

$$\begin{aligned}\underline{\mu}_{i,N_i} &= (\frac{N_i-1}{N_i})\underline{\mu}_{i,N_i-1} + \frac{x_{k,N_i}-\Delta_i}{||N_i||},\ \underline{\mu}_{i,1} = x_{1,N_1} - \Delta_i,\\ \overline{\mu}_{i,N_i} &= (\frac{N_i-1}{N_i})\overline{\mu}_{i,N_i-1} + \frac{x_{k,N_i}+\Delta_i}{||N_i||},\ \overline{\mu}_{i,1} = x_{1,N_1} + \Delta_i\end{aligned} \tag{13}$$

where Δ_i is an uncertainty factor of the *i-th* cloud, which determines the degree of tolerance against uncertainty. The uncertainty factor creates an interval of the data cloud, which controls the degree of tolerance for uncertainty. It is worth noting that a data sample is considered as a population of the *i-th* cloud when resulting in the highest density. Moreover, $\overline{\Sigma}_{i,N_i}, \underline{\Sigma}_{i,N_i}$ are the upper and lower mean square lengths of the data vector in the *i-th* cloud as follows:

$$\begin{aligned}\underline{\Sigma}_{i,N_i} &= (\frac{N_i-1}{N_i})\underline{\Sigma}_{i,N_i-1} + \frac{||x_{k,N_i}||^2-\Delta_i}{||N_i||},\ \underline{\Sigma}_{i,1} = ||x_{1,N_i}||^2 - \Delta_i,\\ \overline{\Sigma}_{i,N_i} &= (\frac{N_i-1}{N_i})\overline{\Sigma}_{i,N_i-1} + \frac{||x_{k,N_i}||^2+\Delta_i}{||N_i||},\ \overline{\Sigma}_{i,1} = ||x_{1,N_i}||^2 + \Delta_i\end{aligned} \tag{14}$$

Although the concept of the cloud-based hidden node was generalized in TeDaClass [11] by introducing the eccentricity and typicality criteria, the interval-valued idea is uncharted in [11]. Note that the Cauchy function is asymptotically a Gaussian-like function, satisfying the activation function requirement of the RVFLN to be a universal approximator.

Unlike conventional RVFLNs, pRVFLN puts into perspective a nonlinear mapping of the input vector through the Chebyshev polynomial up to the second order. Note that recently developed RVFLNs in the literature mostly are designed with a zero-order output node [1]. The functional expansion

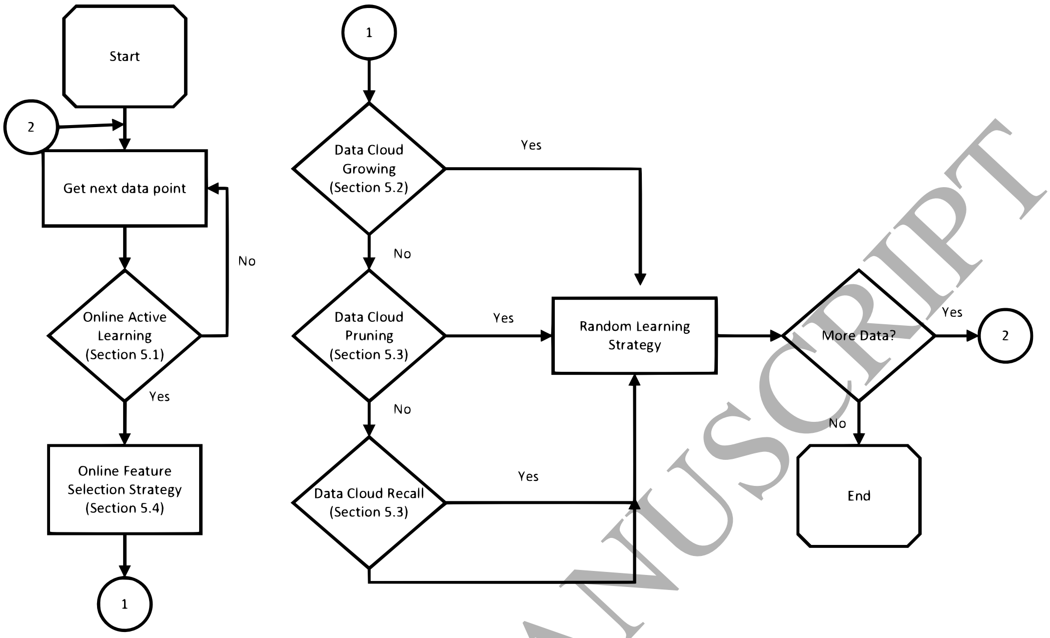

Start
2
Get next data point
No
Online Active Learning (Section 5.1)
Yes
Online Feature Selection Strategy (Section 5.4)
1
1
Data Cloud Growing (Section 5.2)
Yes
No
Data Cloud Pruning (Section 5.3)
Yes
No
Data Cloud Recall (Section 5.3)
Yes
Random Learning Strategy
More Data?
Yes
2
No
End

Figure 3: Fundamental working principle of pRVFLN

block expands the output node to a higher degree of freedom, which aims to improve the local mapping aptitude of the output node. pRVFLN implements the random learning concept of the RVFLN, in which all parameters, namely the input weight A, design factor q, recurrent link weight λ, and uncertainty factor Δ, are randomly generated. Only the weight vector is left for parameter learning scenario w_i. Since the hidden node is parameter-free, no randomization takes place for hidden node parameters. This trait helps to improve consistency of random network in which bad random values lead to poor performance. The network structure of pRVFLN and the interval-valued data cloud are depicted in Figs. 1 and 2 respectively.

5. Learning Policy of pRVFLN

This section discusses the learning policy of pRVFLN structured as follows: Section 5.1 outlines the online active learning strategy, which actively samples relevant training samples for model updates; Section 5.2 deliberates the hidden node growing strategy of pRVFLN; Section 5.3 elaborates the hidden node pruning and recall strategy; Section 5.4 details the online feature selection mechanism; Section 5.5 explains the parameter learning scenario of pRVFLN; Section 5.6 discusses the effect of ranges of random parameters in RVFLN. Algorithm 1 shows the pRVFLN learning procedure.

5.1. Online Active Learning Strategy

The active learning component of the pRVFLN is built on the extended sequential entropy (ESEM) method, which is derived from the SEM method [35]. The ESEM method makes use of the entropy of the neighborhood probability to estimate the sample contribution. The underlying difference from its predecessor [35] lies in the integration of the data cloud paradigm, which greatly relieves the effort in finding the neighborhood probability because the data cloud itself is inherent with the local data density, taking into account the influence of all samples in a local region. Furthermore, it handles the regression problem which happens to be more challenging than the classification problem because the sample contribution is estimated in the absence of a decision boundary. The concept of neighborhood probability refers to the probability of an incoming data stream sitting in the existing data clouds:

$$P(X_i \in R_i) = \frac{\sum\limits_{k=1}^{N_i} \frac{M(X_t,x_k)}{N_i}}{\sum\limits_{i=1}^{R} \sum\limits_{k=1}^{N_i} \frac{M(X_t,x_k)}{N_i}} \tag{15}$$

where X_T is a newly arriving data point and x_n is a data sample, associated with the *i-th* data cloud and R_i is the number of data clouds. $M(X_{T,xk})$ stands for a similarity measure, which can be defined as any similarity measure. The bottleneck is however caused by the requirement to revisit already seen samples. This issue can be tackled by formulating the recursive expression of (15). we would like to clarify that in [24], recursive update as usually done in realm of EIS [3, 2] is formed to compute (15) but the recursive update must be calculated per rule or locally. In the context of the data cloud, this issue becomes even simpler, because it is derived from the idea of local density and is computed based on the local mean [4]. (15) is then written as follows:

$$P(X_i \in R_i) = \frac{\Lambda_i}{\sum\limits_{i=1}^{R} \Lambda_i} \tag{16}$$

Algorithm 1. Learning Architecture of pRVFLN

```
Algorithm 1: Parsimonious Random Vector Functional Link Network
Given a data tuple at $t-th$ time instant $(X_t, T_t) = (x_1, ..., x_n, t_1, ..., t_m)$, $X_t \in \Re^n, T_t \in \Re^m$;
set predefined parameters $\alpha_1, \alpha_2$
/*Step 1: Online Active Learning Strategy/*
For i=1 to R do
    Calculate the neighborhood probability (16) with spatial firing strength (12)
End For
Calculate the entropy of neighborhood probability (17)
IF (18) Then
/*Step 2: Online Feature Selection/*
IF Partial=Yes Then
    Execute Algorithm 3
Else IF
    Execute Algorithm 2
End IF
/*Step 3: Data Cloud Growing Mechanism/*
For j=1 to n do
    Compute $\xi(x_j, T_0)$
End For
For i=1 to R do
    Calculate input coherence and output coherence (19),(20)
    For o=1 to m do
        Calculate $\xi(\widetilde{\mu}_i, T_0)$ (21)
    End For
    IF (23) Then
        Assign a new sample to the winning data cloud, with the highest input coherence $i^*$
    Else IF
        Create a new data cloud based on a new sample (24)
    End IF
End For
/*Step 4: Data Cloud Pruning and Recall Mechanism/*
For i=1 to R do
    For o=1 to m do
        Calculate $\xi(\widetilde{G}_{i,temp}, T_0)$
    End For
    IF (26) Then
        Discard i-th data cloud
    End IF
End For
IF (27) Then
    Recall previously pruned rule $i^*$ (28)
End IF
/*Step 5: Adaptation of Output Weight/*
For i=1 to R do
    Update output weights using FWGRLS
End For
```

where Λ_i is a type-reduced activation degree $\Lambda_i = (1-q)\overline{G}_{i,spatial} + q\underline{G}_{i,spatial}$. Once the neighbourhood probability is determined, its entropy is formulated as follows:

$$H(N|X_i) = -\sum_{i=1}^{R} P(X_i \in R_i)\mathrm{log}P(X_i \in N_i) \tag{17}$$

The entropy of the neighbourhood probability measures the uncertainty induced by a training pattern. A sample with high uncertainty should be admitted for the model update, because it cannot be well-covered by an existing network structure and learning such a sample minimises uncertainty. A sample is to be accepted for model updates, provided that the following condition is met:

$$H \geq thres \tag{18}$$

where *thres* is an uncertainty threshold. The higher the value of this paper the higher the number of training samples are to be discarded and vice versa. This parameter can be made adaptive rather than constant by dynamically adjusting its value to suit the learning context as done in [24]. Nevertheless, this scenario has to integrate a budget determining the maximum number of training samples. Otherwise, it often overspends and is very sensitive to the step size.

5.2. Hidden Node Growing Strategy

pRVFLN relies on the T2SCC method to grow interval-valued data clouds on demand. This notion is extended from the so-called SCC method [36] to adapt to the type-2 hidden node working framework. The significance of the hidden nodes in pRVFLN is evaluated by checking its input and output coherence through an analysis of its correlation to existing data clouds and the target concept. Let $\widetilde{\mu}_i = [\underline{\mu}_i, \overline{\mu}_i] \in \Re^{1\times n}$ be a local mean of the *i-th* interval-valued data cloud (5), $X_t \in \Re^n$ is an input vector and $T_t \in \Re^n$ is a target vector, the input and output coherence are written as follows:

$$I_c(\tilde{\mu}_i, X_t) = (1-q)\zeta(\overline{\mu}_i, X_t) + q\zeta(\underline{\mu}_i, X_t) \tag{19}$$

$$O_c(\tilde{\mu}_i, X_t) = (\zeta(X_t, T_t) - \zeta(\tilde{\mu}_i, T_t)),\ \zeta(\tilde{\mu}_i, T_t) = (1-q)\zeta(\overline{\mu}_i, T_t) + q\zeta(\underline{\mu}_i, T_t) \tag{20}$$

where $\zeta()$ is the correlation measure. Both linear and non-linear correlation measures are applicable here. However, the non-linear correlation measure is rather hard to deploy in the online environment, because it usually calls for the Discretization or Parzen Window method. The Pearson correlation measure is a widely used correlation measure but it is insensitive to the scaling and translation of variables as well as being sensitive to rotation [17]. The maximal information compression index (MCI) is one attempt to tackle these problems and it is used in the T2SCC to perform the correlation measure $\zeta()$[17]:

$$\begin{aligned}\zeta(X_1, X_2) = \frac{1}{2}(&\mathrm{var}(X_1) + \mathrm{var}(X_2) \\ &- \sqrt{(\mathrm{var}(X_1) + \mathrm{var}(X_2))^2 - 4\mathrm{var}(X_1)\mathrm{var}(X_2)(1 - \rho(X_1, X_2)^2)})\end{aligned} \tag{21}$$

$$\rho(X_1, X_2) = \frac{\mathrm{cov}(X_1, X_2)}{\sqrt{\mathrm{var}(X_1)\mathrm{var}(X_2)}} \tag{22}$$

where (X_1, X_2) are substituted with $(\overline{\mu}_i, X_t), (\underline{\mu}_t, X_t), (\overline{\mu}_i, T_t), (\underline{\mu}_t, T_t), (X_t, T_t)$ to calculate the input and output correlation (19), (20). respectively stand for the variance of X, covariance of X_1 and X_2, and Pearson correlation index of X_1 and X_2. The local mean of the interval-valued data cloud represents a data cloud because it represents a point with the highest density. In essence, the MCI method indicates the amount of information compression when ignoring a newly observed sample. The MCI method features the following properties: 1) $0 \leq \zeta(X_1, Y_2) \leq 0.5(\mathrm{var}(X_1) + \mathrm{var}(X_2))$, 2) a maximum correlation is given by $\zeta(X_1, X_2) = 0$, 3) a symmetric property $\zeta(X_1, X_2) = \zeta(X_2, X_1)$, 4) it is invariant against the translation of the dataset, and 5) it is also robust against rotation.

The input coherence explores the similarity between new data and existing data clouds directly, while the output coherence focusses on their dissimilarity indirectly through a target vector as a reference. The input and output coherence formulates a test that determines the degree of confidence in the current hypothesis:

$$I_c(\tilde{\mu}_i*, X_t) \leq \alpha_1, O_c(\tilde{\mu}_i*, X_t) > \alpha_2 \tag{23}$$

where $\alpha_1 \in [0.001, 0.01], \alpha_2 \in [0.01, 0.1]$ are predefined thresholds. If a hypothesis meets both conditions, a new training sample is assigned to a data cloud with the highest input coherence i^*. Accordingly, the number of intervals Ni^*, local mean and square length $\tilde{\mu}_{i^*}, \tilde{\Sigma}_{i^*}$ are updated respectively with (21) and (22) as well as $N_{i^*} = N_{i^*} + 1$. A new data cloud is introduced, provided that the existing hypotheses do not pass either condition (7) , that is, one of the conditions is violated. This situation reflects the fact that a new training pattern conveys significant novelty, which has to be incorporated to enrich the scope of the current hypotheses. Note that if a larger α_1 is specified, fewer data clouds are generated and vice versa, whereas if a larger α_2 is specified, larger data clouds are added and vice versa. The sensitivity of these two parameters is studied in the section V.E of this paper. Because a data cloud is non-parametric, no parameterization is committed when adding a new data cloud. The output node of a new data cloud is initialised:

$$W_{R+1} = W_{i^*},\ \Psi_{R+1} = \overline{\omega} I \tag{24}$$

where $\overline{\omega} = 10^5$ is a large positive constant. The output node is set as the data cloud with the highest input coherence because this data cloud is the closest one to the new data cloud. Furthermore, the setting of covariance matrix Ψ_{R+1} leads to a good approximation of the global minimum solution of batched learning.

5.3. Hidden Node Pruning and Recall Strategy

pRVFLN incorporates a data cloud pruning scenario, termed the type-2 relative mutual information (T2RMI) method. This method was firstly developed in [6] for the type-1 fuzzy system. This method is convenient to apply here because it estimates mutual information between a data cloud and a target concept by analysing their correlation. Hence, the MCI method (21), (22) is valid to measure the correlation between two variables. Although this method has been well-established [6], to date, its effectiveness in handling data clouds and a recurrent structure as implemented in pRVFLN is an open question. Unlike both the RMI method that applies the classic symmetrical uncertainty method, the T2RMI method is formalised using the MCI method as follows:

$$\zeta(\tilde{G}_{i,temp}, T_t) = q\zeta(\underline{G}_{i,temp}, T_t) + (1-q)\zeta(\overline{G}_{i,temp}, T_t) \tag{25}$$

where $\underline{G}_{i,temp}, \overline{G}_{i,temp}$ are respectively the lower and upper temporal activation functions of the *i-th* rule. The temporal activation function is included in (25) rather than the spatial activation function in order to account for the inter-temporal dependency of subsequent training samples. The MCI method is chosen here because it possesses a significantly lower computational burden than the symmetrical uncertainty method but it is still more robust than a linear Pearson correlation index. A data cloud is deemed inconsequential, if the following is met:

$$\zeta_i > mean(\zeta) + 2std(\zeta) \tag{26}$$

where $mean(\zeta), std(\zeta)$ are respectively the mean and standard deviation of the MCI during its lifespan. This criterion aims to capture an obsolete data cloud which does not keep up with current data distribution due to possible concept drift, because it computes the downtrend of the MCI values during its lifespan. It is worth mentioning that mutual information between hidden nodes and the target variable is a reliable indicator for changing data distributions because it monitors significance of a local region with respect to the recent data context.

The T2RMI method also functions as a rule recall mechanism to cope with cyclic concept drift. Cyclic concept drifts frequently happen in relation to the weather, customer preferences, electricity power consumption problems, etc. all of which are related to seasonal change. This points to a situation where a previous data distribution reappears in the current training step. Once pruned by the T2RMI, a data cloud is not forgotten permanently and is inserted into a list of pruned data clouds $R^* = R^* + 1$. In this case, its local mean, square length, population, an output node, and output covariance matrix $\tilde{\mu}_{R^*}, \tilde{\Sigma}_{R^*}, N_{R^*}, \beta_{R^*}, \Psi_{R^*}$, are retained in memory. Such data clouds can be reactivated in the future, whenever their validity is confirmed by an up-to-date data trend. It is worth noting that adding a completely new data cloud when observing a previously learned concept catastrophically erases the learning history. A data cloud is recalled subject to the following condition:

$$\max_{i^*=1,\dots,R^*}(\zeta_{i^*}) < \max_{i=1,\dots,R}(\zeta_i) \tag{27}$$

This situation reveals that a previously pruned data cloud is more relevant than any existing ones. This condition pinpoints that a previously learned concept reappears again. A previously pruned data cloud is then regenerated as follows:

$$\tilde{\mu}_{R+1} = \tilde{\mu}_{R^*}, \tilde{\Sigma}_{R+1} = \tilde{\Sigma}_{R^*}, N_{R+1} = N_{R^*}, \beta_{R+1} = \beta_{R^*}, \Psi_{R+1} = \Psi_{R^*} \tag{28}$$

Although previously pruned data clouds are stored in memory, all previously pruned data clouds are excluded from any training scenarios except (18).

Unlike its predecessors, this rule recall scenario is completely independent from the growing process (please refer to Algorithm 1).

5.4. Online Feature Selection Strategy

A prominent work, namely online feature selection (OFS), was developed in [30]. The appealing trait of OFS lies in its aptitude for flexible feature selection, as it enables the provision of different combinations of input attributes in each episode by activating or deactivating input features (1 or 0) in accordance to the up-to-date data trend. Furthermore, this technique is also capable of handling partial input attributes which are fruitful when the cost of feature extraction is too expensive. OFS is generalized here to fit the context of pRVFLN and to address the regression problem.

We start our discussion from a condition where a learner is provided with full input variables. Suppose that B input attributes are to be selected in the training process and $B < n$, the simplest approach is to discard the input features with marginal accumulated output weights $\sum\limits_{i=1}^{R}\sum\limits_{j=1}^{2}\beta_{i,j}$ and maintain only B input features with the largest output weights. Note that the second term $\sum\limits_{j=1}^{2}$ is required because of the extended input vector $x_e \in \Re^{(2n+1)}$. The rule consequent informs a tendency or orientation of a rule in the target space which can be used as an alternative to gradient information. Although it is straightforward to use, it cannot ensure the stability of the pruning process due to a lack of sensitivity analysis of the feature contribution. To correct this problem, a sparsity property of the L1 norm can be analyzed to examine whether the values of n input features are concentrated in the L1 ball. This allows the distribution of the input values to be checked to determine whether they are concentrated in the largest elements and that pruning the smallest elements wont harm the models accuracy. This concept is actualized by first inspecting the accuracy of pRVFLN. The input pruning process is carried out when the system error is large enough $T_t - y_t > \kappa$. Nevertheless, the system error is not only large in the case of underfitting, but also in the case of overfitting. We modify this condition by taking into account the evolution of system error $|\overline{e}_t + \sigma_t| > \kappa|\overline{e}_{t-1} + \sigma_{t-1}|$ which corresponds to the global error mean and standard deviation. The constant κ is a predefined parameter and fixed at 1.1. The output nodes are updated using the gradient descent approach and then projected to the L2 ball to guarantee a bounded norm. Algorithm 2 details the algorithmic development of pRVFLN.

```
Algorithm 2. GOFS using full input attributes Input: $\alpha$ learning rate, $\chi$ regularization factor, $B$ the number of features to be retained
Output: selected input features $X_{t,selected} \in \Re^{1\times B}$
For t=1,., T
    /*Step 1: Check the reliability of model/*
    Make a prediction $y_t$
    IF $|\overline{e}_t+\sigma_t| > 1.1|\overline{e}_{t-1}+\sigma_{t-1}|$ // for regression case, check global system error $\hat{o} = \max_{o=1,\dots,m}(y_o) \neq T_t$ or // for classification, check whether a sample is correctly classified
        /*Step 2: Adapt the output weight vector and apply L2 projection/*
        $\beta_i = \beta_i - \chi\alpha\ \beta_i - \alpha\chi\frac{\partial E}{\partial\beta_i},\ \beta_i = \min(1, \frac{1/\sqrt{\chi}}{||\beta_i||_2})\beta_i$
        /*Step 3: Prune inconsequential input attribute/*
        Prune input attributes $X_t$ except those of $B$ largest $\sum_{i=1}^{R}\sum_{j=1}^{2}\beta_{i,j}$
    Else
        $\beta_i = \beta_{i,t-1}$
    End IF
End FoR
```

where α, χ are respectively the learning rate and regularization factor. We assign $\alpha = 0.2, \chi = 0.01$ following the same setting [30]. The optimization procedure relies on the standard mean square error (MSE) as the objective function and utilises the conventional gradient descent scenario:

$$\frac{\partial E}{\partial \beta_i} = (T_t - y_t)\left\{\sum_{i=1}^{R}(1-q)\overline{G}_{i,temporal} + \sum_{i=1}^{R} q\underline{G}_{i,temporal}\right\} \tag{29}$$

Furthermore, the predictive error has been theoretically proven to be bounded in [30] and the upper bound is also found. One can also notice that the GOFS enables different feature subsets to be elicited in each training observation t.

A relatively unexplored area of existing online feature selection is a situation where a limited number of features is accessible for the training process. To actualise this scenario, we assume that at most B input variables can be extracted during the training process. This strategy, however, cannot be done by simply acquiring any B input features, because this scenario risks having the same subset of input features during the training process. This problem is addressed using the Bernoulli distribution with confidence level ϵ to sample B input attributes from n input attributes $B < n$. Algorithm 3 provides an overview of feature selection procedure.

```
Algorithm 3. GOFS using partial input attributes Input: $\alpha$ learning rate, $\chi$ regularization factor, $B$ the number of features to be retained, $\epsilon$ confidence level
Output: selected input features $X_{t,selected} \in \Re^{1\times B}$
For t=1,., T
    /*Step 1: Generate partial input information/*
    Sample $\gamma$ from Bernoulli distribution with confidence level $\epsilon$
    IF $\gamma_t = 1$
        Randomly select $B$ out of $n$ input attributes $\widetilde{X}_t \in \Re^{1\times B}$
    End IF
    /*Step 2: Check reliability of the model/*
    Make a prediction $y_t$
    IF $|\overline{e}_t + \sigma_t| > 1.1|\overline{e}_{t-1} + \sigma_{t-1}|$ // for regression, check the global system error $\hat{o} = \max_{o=1,...,m}(y_o) \neq T_t$ or // for classification, check whether a sample is correctly classified
        /*Step 3: Adapt the output weight vector and apply L2 projection/*
        $\hat{X}_t = \widetilde{X}_t/(B/n\epsilon) + (1-\epsilon)$
        $\beta_i = \beta_i - \chi\alpha\ \beta_i - \alpha\chi\frac{\partial E}{\partial \beta_i},\ \beta_i = \min(1, \frac{1/\sqrt{\chi}}{||\beta_i||_2})\beta_i$
        /*Step 4: Prune inconsequential input attribute/*
        Prune input attributes $X_t$ except those of $B$ largest $\sum_{i=1}^{R}\sum_{j=1}^{2}\beta_{i,j}$
    Else
        $\beta_{i,t} = \beta_{i,t-1}$
    End IF
End FoR
```

As with Algorithm 2, the convergence of this scenario has been theoretically proven and the upper bound is derived in [30]. One must bear in mind that the pruning process in Algorithm 2 and 3 is carried out by assigning crisp weights (0 or 1), which fully reflect activation and deactivation of input features.

5.5. Random Learning Strategy

pRVFLN adopts the random parameter learning scenario of the RVFLN, leaving only the output nodes W to be analytically tuned with an online learning scenario, whereas others, namely A_t, q, λ, Δ, can be randomly generated without any tuning process. To begin the discussion, we recall the output expression of pRVFLN as follows:

$$y_o = \sum_{i=1}^{R} \beta_i \tilde{G}_{i,temporal}(X_t; A_t, q, \lambda, \Delta) \tag{30}$$

Referring to the RVFLN theory, the activation function $\tilde{G}_{i,spatial}$ should satisfy the following conditions.

$$\int_R G^2(x)dx < \infty, \ or \ \int_R [G'(x)]^2 dx < \infty \tag{31}$$

Furthermore, a large number of hidden nodes R is usually needed to ensure adequate coverage of data space because hidden node parameters are chosen at random [27]. Nevertheless, this condition can be relaxed in the pRVFLN, because the data cloud growing mechanism, namely the T2SCC method, partitions the input region in respect to real data distributions. The data cloud-based neurons are parameter-free and thus do not require any parameterization, which often calls for a high-level approximation or complicated optimization procedure. Other parameters, namely A_t, q, λ, Δ, are randomly chosen, and their region of randomisation should be carefully selected. Referring to [7], the parameters are sampled randomly from the following.

$$\begin{cases} b = -w_0 y_0 - \mu_0 \\ w_0 = \alpha c_0; \ c_0 \in V^d; \ V^d = [0; \Omega] \times [-\Omega; \Omega] \\ y_0 \in I^d \\ \mu_0 \in [-2\Omega, 2\Omega] \end{cases} \tag{32}$$

where μ, Ω, α are probability measures. Nevertheless, this strategy is impossible to implement in online situations because it often entails a rigorous trial-error process to determine these parameters. Furthermore, these ranges are derived to prove theoretically the universal approximation property of RVFL.

Assuming that a complete dataset $\Xi = [X, T] \in \Re^{N\times(n+m)}$ is observable, a closed-form solution of (7) can be defined to determine the output weights . Although the original RVFLN adjusts the output weight with the conjugate gradient (CG) method, the closed-form solution can still be utilised with ease [7]. The obstacle for the use of pseudo-inversion in the original work was the limited computational resources in 90's. Although it is easy to use and ensures a globally optimum solution, this parameter learning scenario however imposes revisiting preceding training patterns which are intractable for online learning scenarios. pRVFLN employs the FWGRLS method [22] to adjust the output weight. we also would like to clarify that FWGRLS can be seen as a derivation of FWRLS [3] where the weight decay term is added to retain the decay effect during the recursive updates. As the FWGRLS approach has been detailed in [22], it is not recounted here. The flowchart of pRVFLN is visualized in Fig. 3.

Table 2: Details of Experimental Procedure

Section	Mode	Number of Runs	Benchmark Algorithm	Pred. Parameters	NS	NI
A (Nox Emission)	Direct Partition	10 times	GENEFIS, eTS, simpeTS, DFNN, GDFNN, FAOS-PFNN, ANFIS, BARTFIS	$\alpha_1 = 0.002, \alpha_2 = 0.02$	826	170
	Cross Validation	5 times per fold	DNNE, Online RVFLN, Batch RVFLN	$\alpha_1 = 0.002, \alpha_2 = 0.02$		
B (Tool Cond. Mon.)	Direct Partition	10 times	GENEFIS, eTS, simpeTS, DFNN, GDFNN, FAOS-PFNN, ANFIS, BARTFIS	$\alpha_1 = 0.002, \alpha_2 = 0.02$	630	12
	Cross Validation	5 times per fold	DNNE, Online RVFLN, Batch RVFLN	$\alpha_1 = 0.002, \alpha_2 = 0.02$		
C (Nox E., Tool Cond. Mon.)	Cross Validation	5 times per fold	N/A	$\alpha_1 = 0.002, \alpha_2 = 0.02$	As above	As above
D (Mackey Glass)	Direct Partition	10 times	N/A	$\alpha_1 = 0.002, \alpha_2 = 0.02$	3500	4
E (BJ gas furnace)	Direct Partition	10 times	N/A	N/A	290	2

5.6. Robustness of RVFLN

The network parameters are usually sampled uniformly within a range of [-1,1] in the literature. A new finding of Li and Wang in [12] exhibits that randomly generating network parameters with a fixed scope $[-\alpha, \alpha]$ does not ensure a theoretically feasible solution or often the hidden node matrix is not full rank. Surprisingly, the hidden node matrix was not invertible in all their case studies when randomly sampling network parameters in the range of [-1,1] and far better numerical results were achieved by choosing the scope [-200,200]. This trend was consistent with different numbers of hidden nodes. How to properly select scopes of random parameters and its corresponding distribution still require in-depth investigation [26]. In practice, a pre-training process is normally required to arrive at a decent scope of random parameters. Note that the range of random parameters by Igelnik and Pao [7] is still at the theoretical level and does not touch the implementation issue. We study different random regions in Section 6.4 to see how pRVFLN behaves under variations of the scope of random parameters.

Table 3: Prediction of Nox emissions Using Time-Series Mode

Model	**RMSE**	**Node**	**Input**	**Runtime**	**Network**	**Samples**
pRVFLN (P)	**0.04±0.0009**	**1**	**5**	**3.4±0.14**	**11**	**596±0**
pRVFLN (F)	0.04±0.009	**1**	**5**	3.46±0.25	**11**	**596±0**
eT2Class	0.045	2	170	17.98	117304	667
GENEFIS	0.1	7	18	6.59	2268	667
RIVMcSFNN	0.05	1	146	6.59	128.62	667
Simp_eTS	0.14	5	170	5.5	1876	667
BARTFIS	0.11	4	170	5.55	52	667
DFNN	0.18	548	170	4332.9	280198+NS	667
GDFNN	0.48	215	170	2144.1	109865	667
eTS	0.38	27	170	1098.4	13797	667
FAOS-PFNN	0.06	6	170	14.8	2216+NS	667
ANFIS	0.15	2	170	100.41	17178	667

Table 4: Prediction of Nox emissions Using CV Mode

Model	NRMSE	Node	Input	Runtime	Network	Samples
pRVFLN (P)	**0.09±0.01**	**1.3±0.05**	**5**	4.78±0.48	**14.5±0.6**	**743.4±0.14**
pRVFLN (F)	0.094±0.01	1.3±0.17	**5**	**4.4±0.47**	14.96±1.9	743.4±0.2
DNNE	0.14±0	50	170	8.74±0.05	43600+NS	744
Online RVFLN	0.52±0.02	100	170	5.13±0.52	87200	744
Batch RVFLN	0.59±0.05	100	170	6.3±0.001	87200+NS	744

6. Numerical Examples

This section presents the numerical validation of our proposed algorithm using case studies and comparisons with prominent algorithms in the literature. Two numerical examples, namely modelling of Nox emissions from a car engine and tool condition monitoring in the ball-nose end milling process, are presented in Section 6.2 and 6.3 of this paper, and two other numerical examples, namely modeling of S&P 500 index time series and prediction of household electricity consumption, are placed in the supplemental document to keep the paper compact while Section 6.1 elaborates on experimental setup. We provide the analysis of robustness in Section 6.4 which offers additional results with different random regions and illustrates how the scope of random parameters influences the final numerical results. The influence of user-defined predefined thresholds are analysed in Section 6.5. Furthermore, additional numerical results across different problems are provided in the supplemental document.

6.1. Experimental Setup

Our numerical studies were carried out under two scenarios: the time-series scenario and the cross-validation (CV) scenario. The time-series procedure orderly executes data streams according to their arrival and partitions data streams into two parts, namely training and testing. Simulations were repeated 10 times and the numerical results were averaged from 5 runs to arrive at conclusive findings because of the random nature of pRVFLN. In the time-series mode, pRVFLN was compared against 11 state-of-the-art evolving algorithms: eT2Class [23], RIVMcSFNN [24], BARTFIS [18], GENEFIS [22], eTS [3], simp_eTS [2], DFNN [32], GDFNN [33], FAOSPFNN [31], ANFIS [8]. The CV scenarios were implemented in our experiment in order to follow the commonly adopted simulation environment of other RVFLNs in the literature where each fold is repeated five times to prevent the random natures of RVFLNs affecting numerical results. The numerical results were obtained from average numerical results over all folds. pRVFLN was benchmarked against the decorelated neural network ensemble (DNNE) [1], online and batch versions of RVFLN [26]. The MATLAB code of pRVFLN is provided in [1] while the MATLAB codes of DNNE and RVFLN are available online [2,3]. Comparisons were performed against five evaluation criteria: accuracy, data clouds, input attribute, runtime, and network parameters. The scope of the random parameters was set in the range [0,1] but the effect of this range on numerical results is explained in Section 6.4. For all simulations, the same setting of hyper-parameters was applied $\alpha_1 = 0.002, \alpha_2 = 0.02$ to show that these two parameters are not case-specific. It is worth mentioning that these two values are simply picked up and are not obtained from a pre-processing step - grid search, cross validation, etc. In other words, we do not fine-tune these two parameters to arrive at presented numerical results. One can explore different values that might lead to better numerical results than those reported. All the numerical studies were carried out using the original feature space without offline feature selection to check the effectiveness of the GOFS method. Moreover, two configurations of the GOFS method, partial and full, were simulated in the numerical study. For Nox emission problem, the desired number of input attributes was set as 5 for both time-series and CV modes while, for the tool wear prediction problem, the number of input variables was selected as 8 for both time-series and CV scenarios. Normalization was undertaken before carrying out the simulation. To ensure a fair comparison, all the consolidated algorithms were executed using the same computational resources under the MATLAB environment. Details of the experimental procedure are given in Table 2.

6.2. Modeling of Nox Emissions from a Car Engine

This section demonstrates the efficacy of the pRVFLN in modeling Nox emissions from a car engine [15]. This real-world problem is relevant to validate the learning performance, not only because it features noisy and uncertain characteristics similar to the nature of a car engine, it also characterizes high dimensionality, containing 170 input attributes. That is, 17 physical

[1] http://www.ntu.edu.sg/home/mpratama/Publication.html

[2] http://homepage.cs.latrobe.edu.au/dwang/html/DNNEweb/index.html

[3] http://ispac.ing.uniroma1.it/scardapane/software/lynx/

variables were captured in 10 consecutive measurements. Furthermore, different engine parameters were applied to induce changes to the system dynamics to simulate real driving actions across different road conditions. In the time-series procedure, 826 data points were streamed to consolidated algorithms, where 667 samples were set as training samples, and the remainder were fed for testing purposes. 10 runs were carried out to attain consistent numerical results. In the CV procedure, the experiment was run under the 10-fold CV, and each fold was repeated five times similar to the scenario adopted in [1]. This strategy checks the consistency of the RVFLNs learning performance because it adopts the random learning scenario and avoids data order dependency. Table 3 and 4 exhibit the consolidated numerical results of the benchmarked algorithms.

Table 5: Tool Wear Prediction Using Time Series Mode

Model	**RMSE**	**Node**	**Input**	**Runtime**	**Network**	**Samples**
pRVFLN (P)	0.14±0.02	1.4±0.5	**8**	**0.14±0.04**	23.8±9.3	295.6±28.4
pRVFLN (F)	0.14±0.03	**1±0**	**8**	**0.07±0.02**	**17**	**206.2±83.4**
eT2Class	0.16	4	12	1.1	1260	320
RIVMcSFNN	**0.11**	**1**	12	1.1	1260	315
Simp_eTS	0.22	17	12	1.29	437	320
eTS	0.15	7	12	0.56	187	320
BARTFIS	0.16	6	12	0.43	222	320
GENEFIS	0.14	14	12	0.41	2366	320
DFNN	0.27	42	12	2.41	1092+NS	320
GDFNN	0.26	7	12	2.54	259+ NS	320
FAOS-PFNN	0.38	7	12	3.76	1022+NS	320
ANFIS	0.16	8	12	0.52	296+ NS	320

It is evident that pRVFLN outperforms its counterparts in all the evaluation criteria. pRVFLN is equipped with an online active learning strategy, which discards superfluous samples. This learning module had a significant effect on predictive accuracy. Furthermore, pRVFLN utilizes the GOFS method, which is capable of coping with the curse of dimensionality. Note that the unique feature of the GOFS method is that it allows different feature subsets to be picked up in every training episode which avoids the catastrophic forgetting of obsolete input attributes, which are temporarily inactive

Table 6: Tool wear prediction using CV Mode

Model	**NRMSE**	**Node**	**Input**	**Runtime**	**Network**	**Samples**
pRVFLN (P)	0.16±0.3	1.08±0.23	**8**	**0.14±0.01**	25.1±0.88	**478.8±69.63**
pRVFLN (F)	**0.12±0.07**	**1.02±0.14**	**8**	**0.14±0.01**	**17.3±2.04**	493.8±63.8
DNNE	0.11±0	50	12	0.65±0.04	3310+NS	571.5
Online RVFLN	0.16±0.01	100	12	0.17±0.21	1400	571.5
Batch RVFLN	0.19±0.04	100	12	0.2±0.001	1400+NS	571.5

due to changing data distributions. The GOFS can handle partial input attributes during the training process and results in the same level of accuracy as that of the full input attributes. The use of full input attributes slowed down the execution time because it needed to deal with 170 input variables first, before reducing the input dimension. In this case study, we selected five input attributes to be kept for the training process. Our experiment shows that the number of selected input attributes is not problem-dependent and is set to the desired tradeoff between accuracy and simplicity. The fewer the number of input attributes to be selected the faster the training speed but at a cost of accuracy. We did not observe a significant performance difference when using either the full input mode or partial input mode. On the other hand, consistent numerical results were achieved by pRVFLN, although the pRVFLN is built on the random vector functional link algorithm, as observed in the CV experimental scenario. In addition, pRVFLN produced the most encouraging performance in almost all evaluation criteria. Note that the number of training samples, NS, has to be added in the network parameters for both DNNE and batch RVFLN because their learning procedures cannot be executed in a single scan rather it depends on iterating entire data samples over a number of epochs.

6.3. Tool Condition Monitoring of High-Speed Machining Process

This section presents a real-world problem from a complex manufacturing process [18]. The objective of this case study is to perform predictive analytics of the tool wear in the ball-nose end milling process frequently found in the metal removal process of the aerospace industry. In total, 12 time-domain features were extracted from the force signal and 630 samples were collected during the experiment. Concept drift in this case study is evident from changing surface integrity, tool wear degradation as well as varying machining configurations. For the time-series experimental procedure, the con-

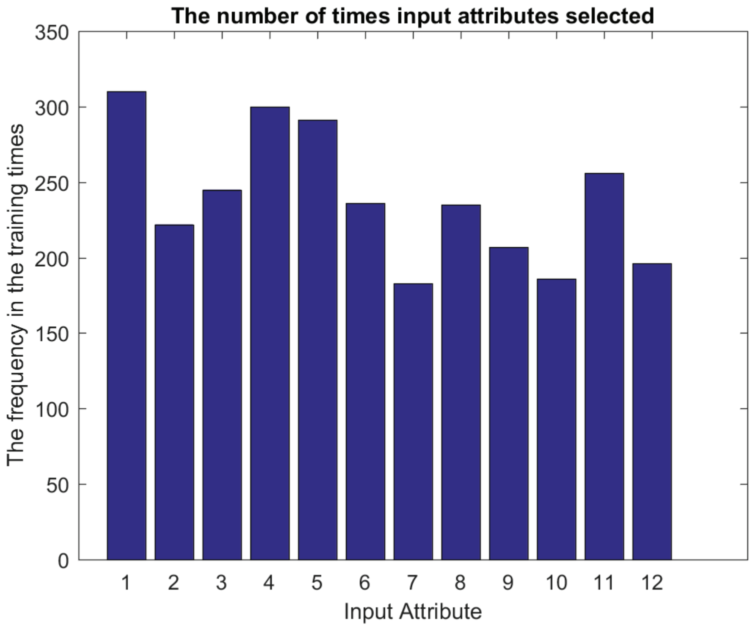


Figure 4: The frequency of input features

solidated algorithms were trained using data from cutter A, while the testing phase exploited data from cutter B. This process was repeated 10 times to achieve valid numerical results. For the CV experimental procedure, the 10-fold CV process was undertaken where each fold was undertaken five times to arrive at consistent findings. Tables 5 and 6 report the average numerical results across all folds. Fig. 4 depicts how many times input attributes are selected during one fold of the CV process.

It is observed from Tables 5 and 6 that pRVFLN evolved the lowest structural complexities while retaining a high accuracy. It is worth noting that although the DNNE exceeded pRVFLN in accuracy, it imposed considerable complexity because it is an offline algorithm revisiting previously seen data samples and adopts an ensemble learning paradigm. The efficacy of the online sample selection strategy can be seen, as it leads to a significant reduction in the training samples to be learned during the experiment. Using partial input information led to subtle differences to those with the full input information. It is seen in Fig. 4 that the GOFS selected different feature subsets in every training episode. Additional numerical examples are provided in the supplemental document. It is worth mentioning that the nature of RVFL-based algorithms such as pRVFLN, dnne is highly dependent on the initialization step. Recently, dnne has been extended in [28] where it incorporates the concept of SCN to minimize the effect of improper parameter initialization.

6.4. Analysis of Robustness

This section aims to numerically validate our claim in Section 5.6 that a range [-1,1] does not always ensure the production of a reliable model [12]. Additional numerical results with different intervals of random parameters are presented. Four intervals, namely [0,0.1], [0,0.5], [0,0.8], [0,3], [0,5], [0,10] were tried for two case studies described in Sections 6.1 and 6.2. Our experiments were undertaken in the 10-fold CV procedure as in previous sections. Table 7 displays the numerical results.

For the tool wear case study, the best-performing model was generated by the range [0,0.1]. The higher the range of the model, the more inferior the model, to the point where a model was no longer stable under the range [0,3]. On the other side, the range [0,0.5] induced the best-performing model with the highest accuracy while evolving comparable network complexity for the Nox emission case study. A higher scope led to a deterioration in the numerical results. Moreover, the range [0,0.1] did not deliver a better accuracy than the range [0,0.5] since this range did not generate diverse enough random values. These numerical results are interpreted from the nature of pRVFLN, a clustering-based algorithm. The success of pRVFLN is mainly determined by the compatibility of the zone of influence of hidden nodes on a real data distribution, and its performance worsens when the scope is not representative to cover the true data distribution. That is, the location of data clouds in the feature space with respect to true data distribution is influential to the success of pRVFLN since the data cloud will return very small or almost zero firing strength when a data sample is far from its coverage. This finding is complementary to Li and Wang [12] which relies on a sigmoid-based RVFLN network, and the scope of random parameters can be outside the applicable operating intervals. Its predictive performance is set by its approximation capability in the output space. It is worth-stressing that network parameters are randomly generated in a positive range since the uncertainty threshold setting the footprint of uncertainty is also chosen at random. Having negative values for this parameter causes invalid interval definitions and poor performance is returned as a result.

6.5. Sensitivity Analysis of Predefined Thresholds

This section examines the impact of two predefined thresholds, namely α_1, α_2, on the overall learning performance of pRVFLN. Intuitively, one can envisage that the higher the value of α_1, the fewer the number of data clouds are added during the training process and vice versa, whereas the higher the value of α_2, the higher the number of data clouds that are generated. To further confirm this aspect, the sensitivity of these parameters is analysed using the box Jenkins (BJ) gas furnace problem. The BJ gas furnace problem is a popular benchmark problem in the literature, where the goal is to model the CO2 level in off gas based on two input attributes: the methane flow rate $u(n)$, and its previous one-step output $t(n-1)$. From the literature, the best input and output relationship of the regression model is known as $\hat{y}(n) = f(u(n-4), t(n-1))$. 290 data points were generated from the gas furnace, 200 of which were assigned as the training samples, and the remainder were utilised to validate the model.α_1 was varied in the range of [0.002,0.004,0.006,0.008], while α_2 was assigned the values of [0.02,0.04,0.06,0.08]. Two tests were carried out to test their sensitivity. That is, α_1 was fixed at 0.002, while setting different values of α_2, whereas α_2 was set at 0.02, while varying α_1. Moreover, our simulation followed the time-series mode with 10 repetitions as aforementioned. The learning performance of pRVFLN was evaluated against four criteria: non-dimensional error index (NDEI), number of hidden nodes, execution time, number of training samples, and number of network parameters. The results are reported in Table 8.

Referring to Table 8, it can be observed that pRVFLN can achieve satisfactory learning performance while demanding very low network, computational, and sample complexities. Allocating different values of α_1, α_2 did not cause significant performance deterioration, where the NDEI, runtime and the number of samples were stable in the range of [0.27,0.38], [0.5,0.79], and [10,30] respectively. Note that the slight variation in these learning performances was also attributed to the random learning algorithm of pRVFLN. On the other hand, the number of hidden nodes and parameters remained constant at 2 and 10 respectively and were not influenced by a variation of the two predefined thresholds. It is worth mentioning that the data cloud-based hidden node of pRVFLN incurred modest network complexity because it did not have any parameters to be memorised and adapted. In all the simulations in this paper, α_1 and α_2 were fixed at 0.02 and 0.002 respectively to ensure a fair comparison with its counterparts and to avoid a laborious pretraining step in finding suitable values for these two parameters.

7. Conclusion

A novel random vector functional link network, namely the parsimonious random vector functional link network (pRVFLN), is proposed. pRVFLN aims to provide a concrete solution to the issue of data streams by putting into perspective a synergy between adaptive and evolving characteristics and the fast and easy-to-use characteristics of RVFLN. pRVFLN is a fully evolving algorithm where its hidden nodes can be automatically added, pruned and recalled dynamically while all network parameters except the output weights are randomly generated in the absence of any tuning mechanism. pRVFLN is fitted by the online feature selection mechanism and the online active learning scenario which further strengthens its aptitude in processing data streams. Unlike conventional RVFLNs, the concept of interval-valued data clouds is introduced. This concept simplifies the working principle of pRVFLN because it neither requires any parameterization per scalar variables nor follows a pre-specified cluster shape. It features an interval-valued spatiotemporal firing strength, which provides the degree of tolerance for uncertainty. Rigorous case studies were carried out to numerically validate the efficacy of pRVFLN where pRVFLN delivered very low complexity. The ensemble version of pRVFLN will be the subject of our future investigation which aims to further improve the predictive performance of pRVFLN.

ACKNOWLEDGEMENT

The first author acknowledges the support of NTU start-up grant. The third author acknowledges the support of the Austrian COMET-K2 program of the Linz Center of Mechatronics (LCM), funded by the Austrian federal government and the federal state of Upper Austria. We thank Dr. Li Xiang, SIMtech, Singapore who provides cutter data used in Section 6.3 of this paper and Mr. MD Meftahul Ferdaus for his assistance for Latex typesetting of our manuscript. We thank Associate Professor Dianhui Wang from La Trobe University for his kind suggestion on the scope setting issue of RVFL networks, and robustness issue of RVFLN.

References

[1] M. Alhamdoosh and D. Wang. Fast decorrelated neural network ensembles with random weights. *Information Sciences*, 264:104–117, 2014.

[2] P. Angelov and D. Filev. Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models. In *Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on*, pages 1068–1073. IEEE, 2005.

[3] P. Angelov and D. P Filev. An approach to online identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):484–498, 2004.

[4] P. Angelov and R. Yager. A new type of simplified fuzzy rule-based system. *International Journal of General Systems*, 41(2):163–185, 2012.

[5] D. S. Broomhead and D Lowe. Multi-variable functional interpolation and adaptive networks. *Complex Systems*, 2(3):321–355, 1998.

[6] H. Han, X-L. Wu, and J-F. Qiao. Nonlinear systems modeling based on self-organizing fuzzy-neural-network with adaptive computation algorithm. *IEEE transactions on cybernetics*, 44(4):554–564, 2014.

[7] B. Igelnik and Y-H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6(6):1320–1329, 1995.

[8] J-SR Jang. ANFIS: adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3):665–685, 1993.

[9] C-F. Juang and C-T. Lin. A recurrent self-organizing neural fuzzy inference network. *IEEE Transactions on Neural Networks*, 10(4):828–845, 1999.

[10] C-F. Juang, Y-Y. Lin, and C-C. Tu. A recurrent self-evolving fuzzy neural network with local feedbacks and its application to dynamic system processing. *Fuzzy Sets and Systems*, 161(19):2552–2568, 2010.

[11] D. Kangin, P. Angelov, and J. A. Iglesias. Autonomously evolving classifier tedaclass. *Information Sciences*, 366:1–11, 2016.

[12] M. Li and D. Wang. Insights into randomized algorithms for neural networks: Practical issues and common pitfalls. *Information Sciences*, 382–383:170–178, 2017.

[13] Y-Y. Lin, J-Y. Chang, and C-T. Lin. Identification and prediction of dynamic systems using an interactively recurrent self-evolving fuzzy neural network. *IEEE Transactions on Neural Networks and Learning Systems*, 24(2):310–321, 2013.

[14] D. Lowe. Adaptive radial basis function nonlinearities, and the problem of generalisation. In *Artificial Neural Networks, 1989., First IEE International Conference on (Conf. Publ. No. 313)*, pages 171–175. IET, 1989.

[15] E. Lughofer, V. Macián, C. Guardiola, and E. P. Klement. Identifying static and dynamic prediction models for nox emissions with evolving fuzzy systems. *Applied Soft Computing*, 11(2):2487–2500, 2011.

[16] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[17] P. Mitra, C. A. Murthy, and S. K. Pal. Unsupervised feature selection using feature similarity. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):301–312, 2002.

[18] R. J. Oentaryo, M-J. Er, S. Linn, and X. Li. Online probabilistic learning for fuzzy inference system. *Expert Systems with Applications*, 41(11):5082–5096, 2014.

[19] Y-H. Pao, G-H. Park, and D. J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, 1994.

[20] Y-H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[21] J. C. Patra and A. C. Kot. Nonlinear dynamic system identification using chebyshev functional link artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(4):505–511, 2002.

[22] M. Pratama, S. G. Anavatti, and E. Lughofer. Genefis: toward an effective localist network. *IEEE Transactions on Fuzzy Systems*, 22(3):547–562, 2014.

[23] M. Pratama, J. Lu, and G. Zhang. Evolving type-2 fuzzy classifier. *IEEE Transactions on Fuzzy Systems*, 24(3):574–589, 2016.

[24] M. Pratama, E. Lughofer, M-J. Er, S. Anavatti, and C-P. Lim. Data driven modelling based on recurrent interval-valued metacognitive scaffolding fuzzy neural network. *Neurocomputing*, 262:4–27, 2017.

[25] S. Scardapane and D. Wang. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2), 2017.

[26] W. F. Schmidt, M. A. Kraaijveld, and R. PW Duin. Feedforward neural networks with random weights. In *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*, pages 1–4. IEEE, 1992.

[27] I. Y. Tyukin and D V Prokhorov. Feasibility of random basis function approximators for modeling and control. In *Control Applications,(CCA) & Intelligent Control,(ISIC), 2009 IEEE*, pages 1391–1396. IEEE, 2009.

[28] D. Wang and C. Cui. Stochastic configuration networks ensemble with heterogeneous features for large-scale data analytics. *Information Sciences*, 417(10):55–71, 2017.

[29] D. Wang and M. Li. Stochastic configuration networks: Fundamentals and algorithms. *IEEE transactions on cybernetics*, 47(10):3466–3479, 2017.

[30] J. Wang, P. Zhao, S. CH. Hoi, and R. Jin. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):698–710, 2014.

[31] N. Wang, M-J. Er, and X. Meng. A fast and accurate online self-organizing scheme for parsimonious fuzzy neural networks. *Neurocomputing*, 72(16–18):3818–3829, 2009.

[32] S. Wu and M-J. Er. Dynamic fuzzy neural networks-a novel approach to function approximation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 30(2):358–364, 2000.

[33] S. Wu, M-J Er, and Y. Gao. A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks. *IEEE Transactions on Fuzzy Systems*, 9(4):578–594, 2001.

[34] L. Xie, Y/ Yang, Z. Zhou, J Zheng, M. Tao, and Z. Man. Dynamic neural modeling of fatigue crack growth process in ductile alloys. *Information Sciences*, 364–365:167–183, 2016.

[35] S. Xiong, J. Azimi, and X. Z. Fern. Active learning of constraints for semi-supervised clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):43–54, 2014.

[36] R-F. Xu and S-J. Lee. Dimensionality reduction by feature clustering for regression problems. *Information Sciences*, 299:42–57, 2015.

Table 7: Analysis of Robustness

Scope	**Criteria**	**Tool Wear**	**Nox emission**
[0,0.1]	RMSE	0.13±0.008	1.9±0
	Node	1.8±0.25	1
	Input	8	5
	Runtime	0.2±0.1	0.1±0.02
	Network	30.9	11
	Samples	503.1	1
[0,0.5]	RMSE	0.14±0.02	0.1±0.01
	Node	1.92±0.2	1.98±0.14
	Input	8	5
	Runtime	0.18±0.008	5.7±0.3
	Network	32.6	21.8
	Samples	571.5	743.4
[0,0.8]	RMSE	0.47±0.42	0.18±0.3
	Node	1.4±0.05	1.96±0.19
	Input	8	5
	Runtime	0.19±0.13	5.56±0.96
	Network	23.8	21.6
	Samples	385.1	711.24
[0,3]	RMSE	Unstable	Unstable
	Node		
	Input		
	Runtime		
	Network		
	Samples		
[0,5]	RMSE	Unstable	Unstable
	Node		
	Input		
	Runtime		
	Network		
	Samples		
[0,10]	RMSE	Unstable	Unstable
	Node		
	Input		
	Runtime		
	Network		
	Samples		

Table 8: Sensitivity Analysis

PARAMETERS	**NDEI**	**HN**	**RUNTIME**	**NP**
$\alpha_1 = 0.002$	0.3	19.3	0.52	96.5
$\alpha_1 = 0.004$	0.3	19.3	0.49	96.5
$\alpha_1 = 0.006$	0.3	35.9	0.67	179.5
$\alpha_1 = 0.008$	0.3	7.3	0.4	36.5
$\alpha_2 = 0.02$	0.3	17	0.44	85
$\alpha_2 = 0.04$	0.31	143	1.41	715
$\alpha_2 = 0.06$	0.32	196.3	2.01	981.5
$\alpha_2 = 0.08$	0.32	196.3	2.01	981.5