FERNN: A Fast and Evolving Recurrent Neural Network Model for Streaming Data Classification

Monidipa Das
School of CSE
NTU, Singapore
monidipadas@ntu.edu.sg

Mahardhika Pratama
School of CSE
NTU, Singapore
mpratama@ntu.edu.sg

Andri Ashfahani
School of CSE
NTU, Singapore
andriash001@e.ntu.edu.sg

Subhrajit Samanta
ERI@NTU
NTU, Singapore
sama0021@e.ntu.edu.sg

***Abstract*—With the recent explosion of data navigating in motion, there is a growing research interest for analyzing streaming data, and consequently, there are several recent works on data stream analytics. However, exploring the potentials of *traditional* recurrent neural network (RNN) in the context of *streaming data classification* is still a little investigated area. In this paper, we propose a novel variant of RNN, termed as FERNN, which features *single-pass learning capability* along with *self-evolution property*. The online learning capability makes FERNN fit for working on streaming data, whereas the self-organizing property makes the model adaptive to the rapidly changing environment. FERNN utilizes *hyperplane activation* in the hidden layer, which not only reduces the network parameters to a significant extent, but also triggers the model to work by default as per *teacher forcing mechanism* so that it automatically handles the vanishing/exploding gradient issues in traditional RNN learning based on back-propagation-through-time policy. Moreover, unlike the majority of the existing autonomous learning models, FERNN is free from *normal distribution assumption* for streaming data, making it more flexible. The efficacy of FERNN is evaluated in terms of classifying *six* publicly available data streams, under the *prequential test-then-train protocol*. Experimental results show encouraging performance of FERNN attaining state-of-the-art classification accuracy with fairly reduced computation cost.**

***Index Terms*—Data stream, RNN, Online learning, Classification, Hyperplane, Teacher forcing**

I. Introduction

Online classification of data stream plays a crucial role in the present era of data explosion where massive volume of data is continuously generated by several data intensive applications and requires an on-the-fly treatment of the same to meet the demand of real-time analytics. Typically, a data stream can be defined as an unbounded (infinite length), ordered sequence of structured or unstructured data item, that arrives at a continuous and rapid manner and is quite susceptible to distributional change over time [22]. Because of the high throughput and potentially infinite size, the data streams are often not feasible to store first and then process as per the requirements [24]. Consequently, the traditional classification techniques that assume the data to be static and repeatedly accessible at any point of time are not at all fit for analysing the streaming data. To be more specific, any standard stream classification technique is expected to satisfy the following constraints: i) restrictive processing time, ii) bounded memory usage, iii) single-scan of data, and iv) sequential data access [9]. Additionally, it should have incremental learning capability and adaptive nature towards time varying data. Accordingly, devising adaptive and real-time classification models to cope up with streaming data have attained significant research interest in recent years.

A. Related Works

Of late, considerable research efforts have been made to develop appropriate models for classifying streaming data. The existing models can be broadly categorized into Bayesian approach-based, decision tree-based, KNN-based, neural network-based, support vector machine-based, and ensemble-based classifiers. A comprehensive overview of all these techniques can be found in [9], [10], [16]. In the present paper, we mainly concentrate on the recently proposed incremental learning-based and ensemble-based models for data stream classification. Among the various incremental models, the Learn++.NC [14], Learn++.NSE [7], and the DEVDAN [19] are worth mentioning. Learn++.NC is able to learn even new classes that may be introduced later with additional data. However, it is not at all suitable for learning in non-stationary environment which is common for a stream classification scenario. Contrarily, Learn++.NSE can accommodate a wide variety of drift scenarios and has the capability of learning in non-stationary environments. However, Learn++.NSE suffers from high structural complexity and also requires significantly long time for execution. DEVDAN is a recently proposed incremental learning algorithm which features self-organizing and single-pass working capability and overcomes the limitation of rigid-structured denoising autoencoders in handing data stream. However, DEVDAN is crafted on single-layered architecture and also not free from normal distribution assumption on data. Beside the incremental learning models, recently ensemble-based approaches have also gained growing research attention in the domain of data stream analytics [3]. Most of the ensemble-based models, like incremental bagging and boosting [17], are decision-tree based, and hence, cannot be adopted for the neural networks. These are also based on static ensemble structure which cannot adapt to the time varying nature of data stream. Further, though there has been a recent progress in adapting ensemble-based classifiers to the non-stationary environment of streaming data, most of these ensemble models [9] suffer from considerably high structural complexity and also require input features to be selected in

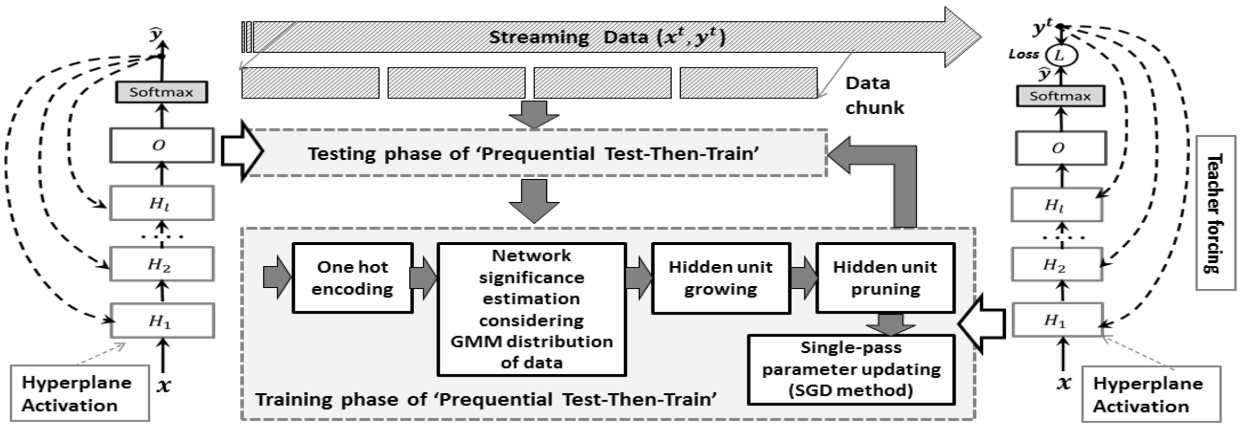
Streaming Data (x^t, y^t)
Data chunk
Testing phase of 'Prequential Test-Then-Train'
One hot encoding
Network significance estimation considering GMM distribution of data
Hidden unit growing
Hidden unit pruning
Single-pass parameter updating (SGD method)
Training phase of 'Prequential Test-Then-Train'
Hyperplane Activation
Softmax
Loss
Teacher forcing

Fig. 1. Learning mechanism of proposed FERNN

pre-processing steps. Recently proposed pENsemble [21] and pENsemble+ [20] models overcome these limitations by their intrinsic ability to dynamically handle structural complexity and also through online feature learning capability. However, none of these models is designed to make use of temporal information from input sequence, and certainly, they lose generalization power to some extent [11].

Nevertheless, in spite of the fact that the recurrent neural network (RNN) models are inherently capable of efficiently using temporal information, adapting RNN models for data stream classification is still a little explored area. Though the works of Ma et al. [12], Neverova et al. [15], Mocanu et al. [13] etc. can be treated as some recent and pioneering researches on RNN-based data analytics, neither of these is adaptive to the time varying nature of data stream, and consequently, these become unfit for analyzing streaming data. In addition, the conventional RNNs suffer from the exploding gradient problem and is slow because of the nature of back propagation through time (BPTT) method.

B. Major contributions

This work is motivated by the remembrance power and sequential learning capability of RNN, which can be effectively exploited to meet the restricted processing time/memory constraints and single-pass data scanning requirement in data stream classification scenario. In this paper, we attempt to propose a variant of RNN with an aim to explore inherent potentials of traditional RNN in data stream classification, and also to fill some of the research gaps as discussed above. Our key contributions can be summarized in the following manner:

- We have proposed FERNN, a *new variant of recurrent neural network* (RNN), capable of classifying streaming data in a fast and cost efficient way;
- In our proposed FERNN, we have exploited the concept of hyperplane clustering [8] in the form of a new activation on the hidden layer. The use of hyperplane activation significantly *reduces the number of network parameters* and eventually brings down the computational costs;
- We have designed FERNN in such a way that it intrinsically learns as per the teacher forcing policy, and thereby, *overcomes the expensive computation and gradient vanishing/exploding issues* in traditional back propagation through time (BPTT)-based RNN learning.
- The proposed FERNN features *self-evolution power*, according to which the FERNN starts learning with a single hidden unit and automatically can construct/refine the network to cope up with the streaming data characteristic.
- Finally, the structural adaptation mechanism in FERNN is followed from *extended network significance method* dealing with *Gaussian mixture model* (GMM) of data distribution. Therefore, the proposed FERNN is released from the normal distribution assumption as adopted by most of the existing autonomous learning models.

The effectiveness of proposed FERNN has been empirically evaluated considering six real-world and synthetic data sets, popularly used as benchmark in stream classification problems. FERNN performance is compared with a number of recently developed ensemble-based and incremental learning models. The experimental results reveal that even a traditional RNN, when adapted appropriately like FERNN, is able to produce comparable accuracy while classifying streaming data.

The rest of the paper is structured as follows. Section II presents the conceptual as well as algorithmic details of the proposed FERNN model. Section III provides a thorough description of the empirical study along with the details of datasets, experimental set up, results, and major findings. Finally, the concluding remarks are drawn in Section IV.

II. Proposed FERNN: A Fast and Evolving RNN

In this section, we cover the architectural as well as algorithmic details of the proposed FERNN, with respect to data stream classification problem.

A. Problem Formulation

Considering a completely supervised environment, the streaming data classification problem can be formally defined as follows. The data points or instances in the stream appear as a sequence of labeled examples $\{x^t, y^t\}$ at each time stamp $t = 1, 2, \cdots, T$, in continuous and incremental manner, where x^t is a D-dimensional vector of attribute values, y^t is the class

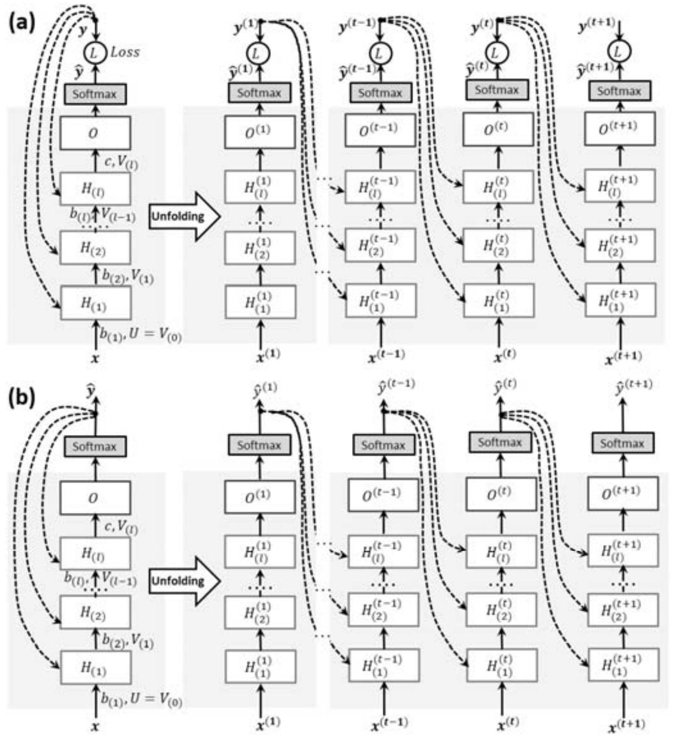

Fig. 2. Proposed FERNN architecture: (a) Training phase, (b) Test phase

label for the t-th sample, and T (the total number of time stamps) is unknown, in practice. The classifier C needs to predict the class label for each example x^t, which arrives either in 'instance-by-instance' or in 'block-by-block' mode. In case of single instance mode, the classification algorithm needs to process the samples one by one, whereas in case of block processing, the classifier is updated when all the examples in the current block are available. Usually, the data blocks, also called 'data chunks', are considered to be equal-sized in this paper but in practise may vary in practise.

Since in the realm of streaming data the data points arrive with absence of true class labels, the proposed FERNN-based classifier is designed to work based on the 'prequential test-then-train' mechanism. That is, first we use the data to test the generative power of the classifier and then the data is exploited to update the model (refer to the Fig. 1). FERNN is constructed under the concept of hyperplane clustering [8], applied while learning features in the hidden layers. The idea is to learn a set of features which can classify/map each data point into a hyperplane-shaped data class/cluster, and this is achieved by applying a new, hyperplane-based activation function on the hidden layer, the details of which are illustrated in the subsequent subsection. The section ends with a detailed description of how FERNN evolves its network structure and remains adaptive to the time variant nature of streaming data.

B. FERNN Model architecture

The recurrent architecture and the corresponding unfolded computational graph of the proposed FERNN is shown in the Fig. 2(a)-2(b), considering both training and testing phase. As shown in Fig. 2(a), for each time step t, the input is $x^{(t)}$ (where $x^{(t)^\top} \in \mathbb{R}^D$, D is input feature dimension), the hidden layer activation are $H^{(t)}_{(k)}$ (where $H^{(t)^\top}_{(k)} \in \mathbb{R}^{e_k}$, e_k is no. of hidden units at layer k, where $1 \leq k \leq l$), un-normalized outputs are $O^{(t)}$ which are further updated through *softmax* layer to achieve the predicted output $\widehat{y}^{(t)}$, (where $\widehat{y}^{(t)\top} \in \mathbb{R}^S$, S is the number of classes) and the loss is $L^{(t)}$. The input-to-hidden (i.e. between $x^{(t)}$ and $H^{(t)}_{(1)}$) connections are parametrized by weight matrix $\mathbf{U}_{[e_1 \times D]}$; the hidden-to-hidden forward connections between layer k and $k+1$ are parameterized by weight matrix $\mathbf{V_{(k)}}_{[e_{k+1} \times e_k]}$ where $(1 \leq k < l)$; and the hidden-to-output connections (i.e between $H^{(t)}_{(l)}$ and $O^{(t)}$) are parameterized by $\mathbf{V_{(k)}}_{[S \times e_k]}$, where $k = l$. To be noted, e_k $(1 \leq k \leq l)$ is not fixed for any k. These change dynamically along with the flow of streaming data during training phase, depending on the distributional change in the input data stream. Moreover, in FERNN, there are recurrent connections between output and hidden layers as denoted by dotted arrows in the Fig. 2. However, these connections are maintained implicitly which is achieved through hyperplane-based activation in the hidden layer (details in the next subsection). That is, in FERNN, no external weight matrix is maintained for output-to-hidden recurrent connection. Consequently, this reduces the number of network parameters to a great extent, especially when the network is deep. Further, the use of output-to-hidden layer connection eventually helps the model to learn from exact inputs of earlier time stamps (as per teacher forcing [11]) and also allows greater parallelization during the training phase.

C. FERNN Learning

This section provides the details of FERNN learning through forward and backward propagation computation.

*1) **Forward-propagation computation**:* The forward propagation computation in FERNN is constructed on the concept of hyperplane, applied in the form of hidden layer activation:

$$H^{i(t)}_{(k)} = e^{\left(-\eta \frac{d^{i(t)}_{(k)}}{max\left(d^{i(t)}_{(k)}\right)}\right)} \tag{1}$$

where, $i = 1, 2, \cdots e_k$ is the number of hidden units in the k-th hidden layer $H^{(t)}_{(k)}$ at time t, and η is a control parameter adjusting the strength of activation. In our experiment we fixed η to 0.05 for all the datasets. $d^{i(t)}_{(k)}$ represents the distance from the $(t-1)$-th data point to the i-th feature in the feature hyperplane at the k-th layer and we define it as follows:

$$d^{i(t)}_{(1)} = \frac{|y^{(t-1)}|_1 - S \cdot \left(b^i_{(1)} + U_i x^{(t)}\right)}{\sqrt{1 + \sum_{j=1}^{D} U^2_{ij}}} \tag{2}$$

where $k = 1$, $|\mathbf{y}|_1$ indicates the 1-norm of $\mathbf{y}$, S is the output dimension, and $b_{(1)}{}^\top \in \mathbb{R}^{e_1}$ is equivalent to the bias (at level 1), associated with the added dimension to the feature plane.

Similarly, for $1 < k \leq l$,

$$d^{i(t)}_{(k)} = \frac{|y^{(t-1)}|_1 - S \cdot \left(b^i_{(k)} + V_{(k-1)i} H^{(t)}_{(k-1)}\right)}{\sqrt{1 + \sum_{j=1}^{e_{k-1}} V^2_{(k-1)ij}}} \tag{3}$$

where ${b_{(k)}}^{\top} \in \mathbb{R}^{e_k}$ is the bias for k-th level and is associated with the added dimension to the $(k-1)$-th feature plane.
Overall, it can be noted that this new hyperplane-based activation leads each hidden layer to be implicitly influenced by the output from previous time stamp without involvement of external weights or parameters, and further, this makes the model learn by default as per the teacher forcing technique during the training phase. Incidentally, in case of testing, the original output $y^{(t-1)}$ can be replaced by the predicted output $\widehat{y}^{(t-1)}$, as indicated in Fig. 2 (b).

After hidden layer activations are obtained, the un-normalized output at time t is determined based on eq. (1) as below:

$$O^{(t)} = c + V_{(k)} H_{(k)}^{(t)} \tag{4}$$

where $k = l$, and $c^{\top} \in \mathbb{R}^{S}$ is the bias for the output layer. Primarily this produces the un-normalized log probabilities which are further normalized using $softmax$ function [11] to obtain the predicted output $\widehat{y}^{(t)}$ as follows:

$$\widehat{y}^{(t)} = softmax(O^{(t)}) \tag{5}$$

Once the predicted output is obtained, we calculate the cross-entropy loss in following manner:

$$L(y, \widehat{y}) = -\sum_i y_i \cdot log(\widehat{y_i}) \tag{6}$$

where $\widehat{y}$ is the predicted value and y is the observed value at a particular time, and L is the associated loss.

*2) **Back-propagation through Gradient computation**:* The gradient computation in FERNN is performed by simply employing back-propagation algorithm (stochastic gradient descent/ SGD) to the unrolled computational graph. However, it is to be noted that since the output-to-hidden recurrent connection in FERNN is not explicit, the back-propagation algorithm is applied in isolation to each time stamp which eventually diminishes the computational time and also overcomes the problem of exploding/vanishing gradient, as encountered in the generalized back-propagation-through-time algorithm which is applied on RNN with hidden-to-hidden recurrent connection. Accordingly, the gradient computation in FERNN for each time stamp is described below:

The recursive computation starts with $\frac{\partial L}{\partial L^{(t)}} = 1$ and proceeds for the output nodes and hidden nodes as described below.

$$\frac{\partial L}{\partial O_i^{(t)}} = \left(\nabla_{O^{(t)}} L\right)_i = \left(\widehat{y_i}^{(t)} - y_i^{(t)}\right) \tag{7}$$

$$\nabla_{H_{(k)}^{(t)}} L = V_{(k)}^{\top} \left(\nabla_{O^{(t)}} L\right) \tag{8}$$

when $k = l$, and

$$\begin{aligned}\nabla_{H_{(k)}^{(t)}} L &= \left(\frac{\partial H_{(k+1)}^{(t)}}{\partial H_{(k)}^{(t)}}\right)^{\top} \left(\nabla_{H_{(k+1)}^{(t)}} L\right) \\ &= V_{(k)}^{\top} \left(\left(\frac{\eta}{M} \cdot H_{(k+1)}^{(t)}\right) \circ \left(\nabla_{H_{(k+1)}^{(t)}} L\right)\right)\end{aligned} \tag{9}$$

when $1 \le k < l$.
Here '$\circ$' denotes element-wise product and M corresponds to the maximum of the distances from sample to the features on the $(k+1)$-th feature plane.

Once the gradients on the internal nodes are obtained, we compute the gradients for parameter nodes as follows:

Gradient computation for bias parameters:

$$\nabla_c L = \left(\frac{\partial O^{(t)}}{\partial c}\right)^{\top} \left(\nabla_{O^{(t)}} L\right) = \left(\nabla_{O^{(t)}} L\right) \tag{10}$$

$$\nabla_{b_{(k)}} L = \left(\frac{\partial H_{(k)}^{(t)}}{\partial b_{(k)}^{(t)}}\right)^{\top} \left(\nabla_{H_{(k)}^{(t)}} L\right) = \left(\frac{\eta}{M} \cdot H_{(k)}^{(t)}\right) \circ \left(\nabla_{H_{(k)}^{(t)}} L\right) \tag{11}$$

where $1 \le k \le l$. Here M corresponds to the maximum of the distances from sample (at a particular time stamp) to the features on the (k)-th feature plane.

Gradient computation for weight parameters:

$$\nabla_{V_{(k)}} L = \sum_i \left(\frac{\partial L}{\partial O_i^{(t)}}\right) \nabla_V O_i^{(t)} = \left(\nabla_{O^{(t)} L}\right) \cdot \left(H_{(k)}^{(t)}\right)^{\top} \tag{12}$$

when $k = l$, and

$$\begin{aligned}\nabla_{V_{(k)}} L &= \sum_i \left(\frac{\partial L}{\partial H_{(k+1)}^{i(t)}}\right) \left(\nabla_{V_{(k)}^{(t)}} H_{(k+1)}^{i(t)}\right) \\ &= \left(\left(\frac{\eta}{M} \cdot H_{(k+1)}^{(t)}\right) \circ \left(\nabla_{H_{(k+1)}^{(t)}} L\right)\right) \left(H_{(k)}^{(t)}\right)^{\top}\end{aligned} \tag{13}$$

when $1 \le k < l$. Here, $\nabla_{V^{(t)}}$ denotes the contribution of the weights at time step t to the gradient.

D. Online Adaptation of Hidden Layers

In this section we formalize the self-evolution strategy of FERNN, as followed from the network significance (NS) method [19]. Primarily, NS is derived from the expectation of mean squared error (MSE) in prediction and mathematically expressed in terms of bias and variance formula as follows:

$$NS = Var(O) + Bias(O)^2 \tag{14}$$

Unlike the standard system error index, NS is capable of measuring the quality of the predictive model by direct inspection of its possible underfitting or overfitting condition. Further, NS can also capture the reliability of the model across the entire data space, given a particular data distribution. A large value of NS specifies either a high variance problem, indicating overfitting of the model, or a high bias problem, indicating the underfitting of the model.

In case of FERNN, the bias-variance formula can be derived from eq. 14 as follows:

$$NS = (E[O^2] - E[O]^2) + (E[O] - y)^2 \tag{15}$$

where, $E[O]$ represents the expectation of un-normalized output from FERNN. For any time instant t, the $E[O]$ for the FERNN network can be recursively estimated as follows.

$$E[O] = \int_{-\infty}^{\infty} (c + V \cdot H) p(H) dH = c + V \cdot E[H] \tag{16}$$

where,

$$E[H] = \int_{-\infty}^{\infty} e^{\left(-\frac{d}{max(d)}\right)} p(d) dd = e^{\left(-\frac{E(d)}{max(E(d))}\right)} \tag{17}$$

Now, $E[d] = \left(E[d^1], E[d^2], E[d^3], \cdots, E[d^{e_k}]\right)^\top$, where

$$E[d^i]_{k=1} = \int_{-\infty}^{\infty} \frac{|y|_1 - \left(b^i_{(1)} + U_i \cdot x\right)}{\sqrt{1 + \sum_{j=1}^{D} U_{ij}}} p(x)dx \tag{18}$$

Majority of the existing literature assume normal distribution of the data stream and accordingly the $p(x)$ is considered as $N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} e^{\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\}}$, when x is a D-dimensional vector. However, the data streams (e.g. weather data [5]) are more likely to have mixture of distributions rather than following an unique density model. Therefore, in order to release our proposed FERNN from this rigid assumption of normal distribution, we define $p(x)$ in terms of Gaussian mixture model as $p(x) = \sum_{i=1}^{K} N(x|\mu_i, \Sigma_i)$, where K is the number of component, π_i are the mixing coefficients such that $0 \leq \pi_i \leq 1$ and $\sum_{i=1}^{K} \pi_i = 1$. In our work, we consider $K = S$ i. e. the number of classes corresponding to the data stream. The π_i values and the other parameters (mean and variance) are estimated and tuned on sample by sample basis by exploiting expectation maximization (EM) algorithm. Accordingly, the eq. (18) can be re-written as:

$$E[d^i]_{k=1} = \frac{1}{S} \sum_{m=1}^{S} \frac{|y|_1 - \left(b^i + U_i \cdot \mu_m\right)}{\sqrt{1 + \sum_{j=1}^{D} U_{ij}}} \tag{19}$$

where, $\mu \equiv [\mu_1, \mu_2, \cdots, \mu_S]$, and $\mu^\top_{m(m=1,\cdots S)} \in \mathbb{R}^D$.

On the other side, when the hidden layer level $k > 1$:

$$\begin{aligned} E[d^i]_{k>1} &= \int_{-\infty}^{\infty} \frac{|y|_1 - \left(b^i_{(k)} + V_{(k-1)i} \cdot H_{(k-1)}\right)}{\sqrt{1 + \sum_{j=1}^{D} V_{(k-1)ij}}} p(H)dH \\ &= \frac{|y|_1 - \left(b^i_{(k)} + V_{(k-1)i} \cdot E[H_{(k-1)}]\right)}{\sqrt{1 + \sum_{j=1}^{D} V_{(k-1)ij}}} \end{aligned} \tag{20}$$

$$= \frac{|y|_1 - \left(b^i_{(k)} + V_{(k-1)i} \cdot e^{-\frac{E[d_{k-1}]}{max\left(E[d_{k-1}]\right)}}\right)}{\sqrt{1 + \sum_{j=1}^{D} V_{(k-1)ij}}} \tag{21}$$

Once NS is calculated as per the eq. 14, it is utilized to update the structural configuration of hidden layer in FERNN, as thoroughly described in the subsequent subsections.

*1) **Hidden units growing strategy**:* The objective of growing hidden unit is to tackle the high bias problem. A high value of bias signifies underfitting situation which can be resolved by increasing the complexity of the network structure, or in other words, by introducing more units in the hidden layer. A new hidden unit is added if the following condition is satisfied:

$$\mu^t_{Bias} + \sigma^t_{Bias} \geq \mu^{min}_{Bias} + \pi\sigma^{min}_{Bias} \tag{22}$$

where μ^t_{Bias}, σ^t_{Bias} are respectively the mean and standard deviation of bias at the t-th time instant and μ^{min}_{Bias}, σ^{min}_{Bias} are the minimum bias till the time instant t, respectively. The value of π is selected as $1.3\exp(-bias^2) + 0.7$ which leads to attain confidence level in between 68.2% and 95.2%. Once a new hidden node is appended, the associated parameters (b and V) can be randomly sampled from the scope of [-1,1]. However, since the scope [-1, 1] may not always ensure the convergence of the model, one may also select the parameters using adaptive scope selection mechanism [25].

*2) **Hidden units pruning strategy**:* The underlying goal of pruning hidden unit is to deal with the high variance problem. A high value of variance signifies overfitting situation which can be resolved by decreasing the complexity of the network structure, or in other words, by reducing the number of units in the hidden layer. A high variance situation is detected when the following condition is satisfied:

$$\mu^t_{Var} + \sigma^t_{Var} \geq \mu^{min}_{Var} + 2\chi\sigma^{min}_{Var} \tag{23}$$

where μ^t_{Var}, σ^t_{Var} are respectively the mean and standard deviation of the variance at the t-th time instant and μ^{min}_{Var}, σ^{min}_{Var} are the minimum variance till the time instant t, respectively. The χ, estimated as $1.3\exp(-Var) + 0.7$, is a dynamic constant controlling the confidence level of the sigma rule.

Now, once the high variance is detected, the algorithm searches for the appropriate hidden unit in the top-most layer (l) for the purpose of pruning. The hidden unit associated with least significance value is chosen as the candidate. The significance (HS_i) of the i-th hidden unit in the layer is measured in terms of average activation degree for all possible data samples as follows:

$$HS_i = \lim_{T\to\infty} \sum_{t=1}^{T} \frac{H^{i(t)}_l}{T} \tag{24}$$

Thus, the pruning process of the hidden units from the top-most (l-th) hidden layer can be represented as follows:

$$\text{Pruning} \longrightarrow \min_{i=1,\cdots,e_l} HS_i \tag{25}$$

III. Experimentation

The details of experimental evaluation for FERNN is thoroughly described in the subsequent subsections. The code of FERNN has been shared online[1].

A. Datasets

FERNN is evaluated with respect to both synthetic and real-world datasets as described below:

- **Occupancy** [4] (A real-world multi-variate time series on room occupancy (binary) as per the environmental condition of the room): The data set contains 20560 instances and 7 attributes. Ground-truth occupancy was derived from time stamped pictures taken every minute.
- **Susy** [1] (An well-used dataset for big data problems): The dataset contains 5M instances, having 18 attributes.
- **Electricity-pricing** [6] (A real-world, non-stationary dataset): This is commonly used in literature under concept drift scenario of data stream analytics. It contains 45312 instances, each having 8 features.
- **Weather** [6] (A real-world dataset): It contains 18159 instances and 8 attributes. The dataset shows non-stationary property with recurring drift phenomena.
- **Hyperplane** [2] (A synthetic dataset widely used in data stream classification problems): The dataset contains 120000 instances (each instance having 4 attributes) and is created based on a moving hyperplane with an aim to incorporate gradual drifting concept.
- **Sea** [23] (A Synthetic dataset): It is comprised of 5 million records and 3 attributes. The dataset is well-used in stream classification problem.

[1]http://dx.doi.org/10.13140/RG.2.2.33768.72965

TABLE I
COMPARATIVE PERFORMANCE ANALYSIS OF FERNN-BASED DATA STREAM CLASSIFICATION

Data sets	Performance Metric	RNN_tanh	pEnsemble+	Learn++	Incremental Bagging	DEVDAN	FERNN (proposed)
Occupancy	Classification rate	74.24± 24.96	89.33± 23.64	**95.49±8.43**	84.15±27.39	85.73± 19.77	93.71± 10.11
	No. of Hidden node/Features	4	2	36	100	4	3
Susy	Classification rate	64.31± 1.53	76.99± 4.6	N/A	74.22±2.11	77.53± 3.22	**77.78± 1.96**
	No. of Hidden node/Features	9	9	N/A	100	21	9
Electricity-pricing	Classification rate	65.12± 7.21	72.60± 12.1	75.61±9	72.8±10.88	69.4± 8.74	**78.03± 5.62**
	No. of Hidden node/Features	4	1	180	100	11	4
Weather	Classification rate	69.14± 4.04	**78.80± 4**	76.51±4.13	75.49±3.44	74.04± 5.68	76.23± 2.91
	No. of Hidden node/Features	4	1	72	100	14	4
Hyperplane	Classification rate	76.55± 2.82	87.60± 6.2	90.87±2.67	81.39±2.2	92.12± 3.47	**92.59± 4.25**
	No. of Hidden node/Features	4	3	480	100	4	2
Sea	Classification rate	75.17± 2.94	92.00± 6	88.00±5.56	87.27±10.19	**92.29± 6.48**	90.02± 6.67
	No. of Hidden node/Features	4	3	400	100	18	2

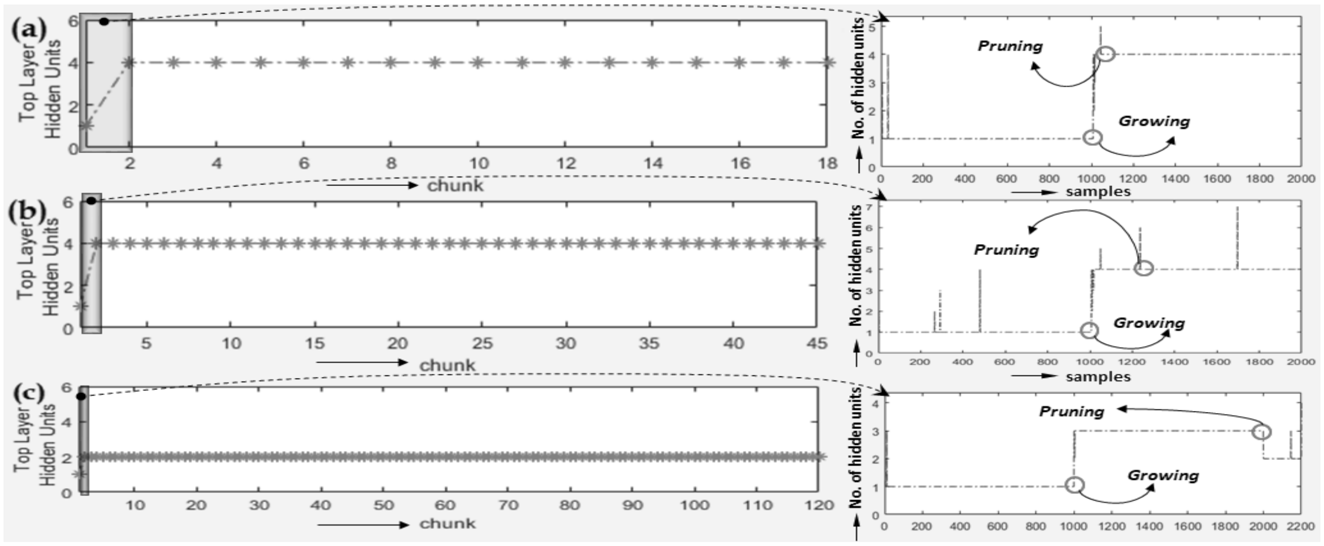

Fig. 3. Evolution of hidden layer units in FERNN for various datasets: (a) Weather, (b) Electricity pricing, (c) Hyperplane

B. Experimental Set-up

We compare FERNN against a number of state-of-the-art classifiers falling into three basic categories: traditional model, ensemble-based model, and incremental learning model. The properties of the underlying algorithms are summarized below:

1) **RNN_tanh**: As a representative of RNN we consider the original model [11] with $tanh$ activation and learning based on back propagation through time (BPTT) technique. This comparison is necessary to illustrate how the proposed FERNN, following teacher forcing policy with hyperplane activation, outperforms the classical BPTT-based RNN learning.
2) **pENsemble+**: The pENsemble+ [20] is an upgraded version of pENsemble, which, unlike pENsemble, is equipped with the online active learning and ensemble merging scenarios and further reduces operators annotation effort with reduced complexity.
3) **Learn++**: The Learn++ [18] is the mother/base algorithm of a number of incremental models (e.g. Learn++.NC [14], Learn++.NSE [7] etc.) for training ensemble of classifiers. It was primarily designed for incremental training of neural network-based classifiers, each trained using a different distribution of training samples. Though Learn++ has unique capability of remembering previous and relevant knowledge, it depends on quite a large number of parameters, high structural complexity, and considerably longer time for execution.
4) **Incremental bagging**: Incremental bagging [17] is the online version of 'Bagging', an well-known ensemble learning model. Even with low overhead it is capable of showing comparable result with respect to its batch counterparts. However, its performance substantially depends on the behaviour of the base learning algorithm.
5) **DEVDAN**: DEVDAN [19] is an incremental learning model constructed on denoising auto-encoder. It can operate in single-pass mode in both generative and discriminative phase. Moreover, it is able to automatically adapt its structure on demand and on-the-fly fashion.

Additionally, we have compared our proposed FERNN with pENsemble [21], Learn++.NSE [7], and incremental boosting [17] algorithms. However, due to page limitation, we could not include these in the main manuscript. An overview of all

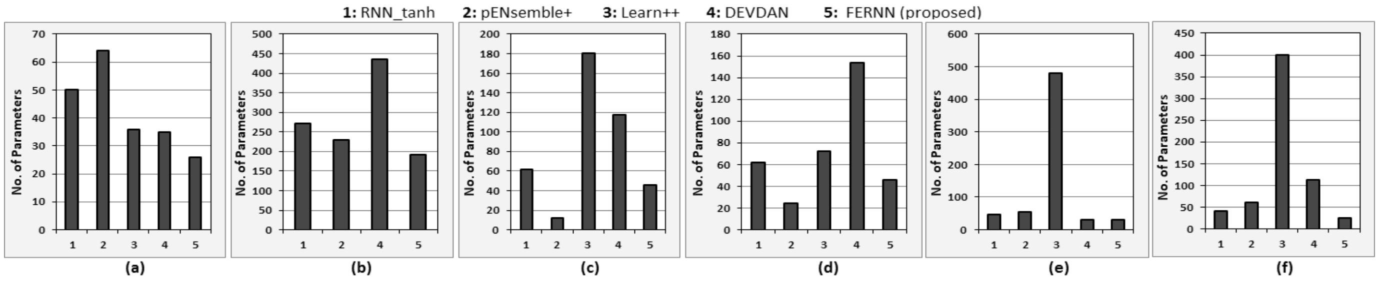

Fig. 4. Comparative study on parameter requirement of the models for various datasets: (a) Occupancy, (b) Susy, (c) Electricity pricing, (d) Weather, (e) Hyperplane, (f) Sea. [*Incremental bagging is not considered here, since it is based on decision tree. In case of 'Susy', Learn++ could not finish execution.*]

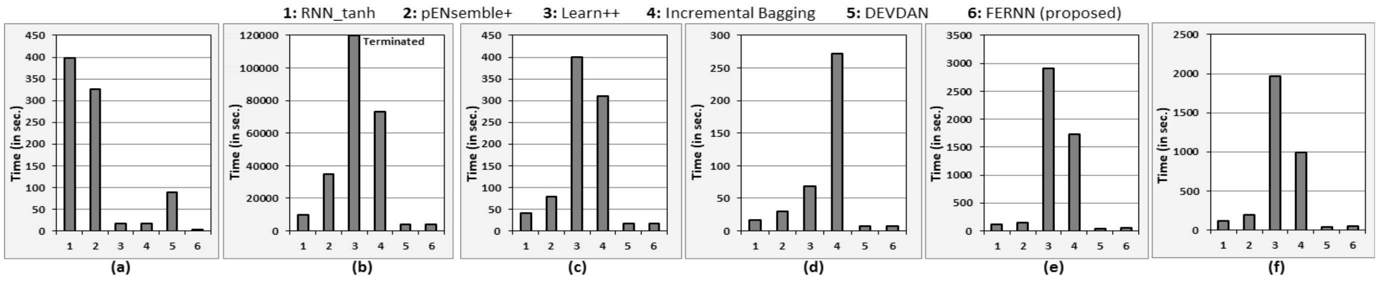

Fig. 5. Comparative study on execution time over various datasets: (a) Occupancy, (b) Susy, (c) Electricity pricing, (d) Weather, (e) Hyperplane, (f) Sea

the results can be found in our supplementary document[2].

In our overall experimentation, we have considered FERNN architecture to be composed of single layer of hidden units. The learning rate for FERNN is set as 0.001 and the value of control parameter η is fixed to be 0.05, for all the considered datasets. FERNN and all the above mentioned benchmark models are executed in the same platform of MATLAB environment with 3.20 GHz Intel(R) Xeon(R) CPU E5-1650 processor and 16 GB RAM. The learning performance has been evaluated based on four criteria, namely, classification rate, no. of used parameters, no. of hidden units (extracted features), and execution time. For all the models and all the performance metrics, we record the average estimation obtained by executing each model five times.

C. Results and Discussions

The comparative performance of FERNN, with respect to classification rate and hidden node requirement is summarized in Table I. We also show in Fig. 3, how the number of required hidden unit in proposed FERNN is evolved with the ever changing flow of data stream. The left part of the figure shows evolution (of hidden units) per data-chunk, whereas the right part of the figure shows evolution per data sample within a specific chunk of data. Moreover, the comparative study on model execution time and parameter count are depicted in the Fig. 5 and Fig. 4, respectively. On analysing the table and the figures, the following inferences can be drawn:

- It is evident that even with a minimum number of hidden units, the proposed FERNN produces the highest classification accuracy in most of the cases (refer Table I). Though in case of 'Weather' dataset, FERNN cannot outperform 'pENsemble+', the classification rate of FERNN is quite comparable and is achieved with appreciably low execution time (refer Fig. 5). Similarly, for 'Sea' dataset, though 'DEVDAN' outperforms FERNN, the highest accuracy of DEVDAN is obtained only by using considerably large number of parameters (refer Fig. 4). Likewise, in case of 'Occupancy' dataset though FERNN can not outperform 'Learn++' and 'Learn++.NSE' models (refer supplementary document[2]), it achieves over 93% accuracy by using quite a lesser no. of features and also with the least execution time.
- Further, though the incremental bagging algorithm (based on decision tree) is found to have advantage over parameter count, it suffers from high structural complexity and large computation time which is not suitable for streaming data (refer to Fig. 5). On the other side, though the incremental boosting algorithm (refer to supplementary document[2]) requires overall least time for execution, the classification accuracy of the model is significantly low compared to FERNN and the others, for each dataset.
- Moreover, it can be noted that in all the cases, the proposed FERNN produces substantially higher classification accuracy than that of traditional BPTT-based RNN model, and that is also by exploiting lesser parameters and minimum computation time. This proves the worth of applying hyperplane-based activation which not only helps the model to extract the essential features with reduced parameter but also leads the model to follow the teacher forcing policy and ultimately overcomes the gradient vanishing/exploding issues.
- Finally, as depicted in the Fig. 3, the execution of

[2]Supplementary document: http://dx.doi.org/10.13140/RG.2.2.26428.69766

the proposed FERNN starts with only one unit in the hidden layer (below the output layer), and then, gradually adds/grows or removes/prunes hidden units to cope up with the time varying distribution of streaming data. This also aids FERNN in achieving the desired accuracy with optimal number of parameters in the classifier model.

Overall, FERNN demonstrates encouraging performance which is typically suitable for stream classification scenario. The study reveals that even a traditional RNN (i.e. not the special architecture like LSTM, GRU etc.) also holds enough potential to effectively perform in streaming data classification when adapted appropriately, like our proposed FERNN.

IV. Conclusions

This paper proposes FERNN, a fast and self-evolving model of recurrent neural network for online classification of streaming data. The novelties in this work are threefold:

1) First of all, FERNN uses a new kind of hidden layer activation, derived from the concept of hyperplane clustering which contributes to considerable reduction in the number of network parameters, and eventually, brings down the execution time. The use of hyperplane activation also persuades the model to learn in accordance with the teacher forcing principle and thereby ensures gradient computation in isolated fashion and reduces the chance of gradients exploding/vanishing problem as encountered in RNN learning with BPTT mechanism;
2) Secondly, FERNN possesses self-organizing property which makes it capable of on-demand constructing/discarding features through automatic growing/pruning of hidden layer units;
3) Finally, unlike many of the existing stream classification approaches, FERNN is not rigidly based on the unique and normal distribution assumption of data, which makes FERNN more flexible towards diverse categories of data streams.

Experimentation using real-world and artificial data stream classification problem demonstrates the effectiveness of FERNN in delivering state-of-the-art accuracy with comparatively less computational costs.

Nevertheless, the proposed FERNN model is crafted under the assumption of fixed number of layers in the architecture. Thus, ample scopes remain in further extending the present model with on-the-fly layer adaptation mechanism, which may be explored in future. We also plan to extend the proposed model with skipped recurrent connection for better modeling of the temporal dependencies in data stream.

References

[1] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[2] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[3] Hamid Bouchachia and Emili Balaguer-Ballester. Dela: A dynamic online ensemble learning algorithm. In *ESANN*, 2014.

[4] Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.

[5] Monidipa Das and Soumya K Ghosh. Data-driven approaches for meteorological time series prediction: A comparative study of the state-of-the-art computational intelligence techniques. *Pattern Recognition Letters*, 105:155–164, 2018.

[6] Gregory Ditzler and Robi Polikar. Incremental learning of concept drift from streaming imbalanced data. *ieee transactions on knowledge and data engineering*, 25(10):2283–2301, 2013.

[7] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *Trans. Neur. Netw.*, 22(10):1517–1531, October 2011.

[8] Md Meftahul Ferdaus, Mahardhika Pratama, Sreenatha G Anavatti, Matthew A Garratt, and Edwin Lughofer. Pac: A novel self-adaptive neuro-fuzzy controller for micro aerial vehicles. *arXiv preprint arXiv:1811.03764*, 2018.

[9] Joao Gama. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012.

[10] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, 50(2):23, 2017.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[12] Xiaolei Ma, Haiyang Yu, Yunpeng Wang, and Yinhai Wang. Large-scale transportation network congestion evolution prediction using deep learning theory. *PloS one*, 10(3):e0119044, 2015.

[13] Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Wil L Kling. Deep learning for estimating building energy consumption. *Sustainable Energy, Grids and Networks*, 6:91–99, 2016.

[14] Michael D Muhlbaier, Apostolos Topalis, and Robi Polikar. Learn ˆ{++}. nc: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes. *IEEE transactions on neural networks*, 20(1):152–168, 2009.

[15] Natalia Neverova, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham Taylor. Learning human identity from motion patterns. *IEEE Access*, 4:1810–1820, 2016.

[16] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and information systems*, 45(3):535–569, 2015.

[17] C Oza Nikunj and January Russell Stuart. Online bagging and boosting. jaakkola tommi and richardson thomas, editors. In *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 105–112, 2001.

[18] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

[19] Mahardhika Pratama, Andri Ashfahani, Yew Soon Ong, Savitha Ramasamy, and Edwin Lughofer. Autonomous deep learning: Incremental learning of denoising autoencoder for evolving data streams. *arXiv preprint arXiv:1809.09081*, 2018.

[20] Mahardhika Pratama, Eric Dimla, Tegoeh Tjahjowidodo, Witold Pedrycz, and Edwin Lughofer. Online tool condition monitoring based on parsimonious ensemble+. *IEEE transactions on cybernetics*, 2018.

[21] Mahardhika Pratama, Witold Pedrycz, and Edwin Lughofer. Evolving ensemble fuzzy classifier. *IEEE Transactions on Fuzzy Systems*, 2018.

[22] Jerzy Stefanowski and Dariusz Brzezinski. Stream classification. In *Encyclopedia of Machine Learning and Data Mining*, pages 1191–1199. Springer, 2017.

[23] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM, 2001.

[24] Mark Tennant, Frederic Stahl, Omer Rana, and João Bártolo Gomes. Scalable real-time classification of data streams with concept drift. *Future Generation Computer Systems*, 75:187–199, 2017.

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