MUSE-RNN: A Multilayer Self-Evolving Recurrent Neural Network for Data Stream Classification

Monidipa Das
NTU, Singapore
monidipadas@ntu.edu.sg

Mahardhika Pratama
NTU, Singapore
mpratama@ntu.edu.sg

Septiviana Savitri
NTU, Singapore
septivianasavitri37@gmail.com

Jie Zhang
NTU, Singapore
ZhangJ@ntu.edu.sg

***Abstract*—In this paper, we propose MUSE-RNN, a multilayer self-evolving recurrent neural network model for real-time classification of streaming data. Unlike the existing approaches, MUSE-RNN offers special treatment towards capturing temporal aspects of data stream through its novel recurrent learning approach based on the teacher forcing policy. Novelties here are twofold. First, in contrast to the traditional RNN models, MUSE-RNN has intrinsic ability to self-adjust its capacity by growing and pruning hidden nodes as well as layers, to handle the ever-changing characteristics of data stream. Second, MUSE-RNN adopts a unique scoring-based layer adaptation mechanism, which makes it capable of recalling prior tasks, with minimum exploitation of network parameters. The performance of MUSE-RNN is evaluated in comparison with a number of state-of-the-art techniques, using seven popular data streams and continual learning problems under prequential test-then-train protocol. Experimental results demonstrate the effectiveness of MUSE-RNN in stream classification scenario.**

***Index Terms*—Recurrent neural network, Data stream, Online learning, Evolving network, Classification**

I. Introduction

Real-time classification of data streams [1], [2], such as phone conversations, sensor data, video surveillance, IP network traffic etc., is of high importance in many of the application areas demanding on-the-fly processing of the generated data. However, a stream classification problem is not a trivial task as it imposes several new challenges which are quite uncommon to the traditional classification scenario [3]. First of all, a stream classification algorithm must work on infinite length data, but process it under restricted time and memory bound. Accordingly, the algorithm should be able to learn new data in a single-pass and sequential manner in the absence of old data. Secondly, since the underlying concepts of the data are subject to change over time, the algorithm should possess self-evolving property and learn in incremental fashion, without suffering from the catastrophic forgetting of previous knowledge [4]. In consequence, stream classification has gained growing research attention, and a number of advanced models based on incremental and continual learning approaches [5], [6], [7] have been proposed recently. However, despite the significant achievements of these systems in terms of concept drift detection and adaptation over time, we observe the following research gaps:

- Majority of these models are based on feed-forward computation which assume that the data is distributed 'not identically, but still *independently*' [3], and thus, do not account for the time dependencies in data stream. However, there are many real-world data streams, such as 'Electricity-pricing', 'Weather', 'Ozone' etc. [3], [8], that show prominent temporal dependence, which can be observed through autocorrelation analysis of the corresponding data.
- Although recurrent neural networks (RNNs) have intrinsic ability to account for temporal dependencies in sequence learning, the existing works mostly focus on improving the memorization aspects of RNNs through variants of control structures as in LSTM (long short term memory) [9] and GRU (gated recurrent units) [10], which often make the model too complex to learn through single scanning of data. Further, the rigid-structure (fixed no. of nodes and layers) of these models often limits their capacity, and consequently, they show deterioration in the presence of concept drift or distributional change in data.

Motivated by these observations, in this paper we attempt to develop MUSE-RNN, a RNN model with dynamic network capacity achieved through *self-evolving multi-layered structure* to suit stream classification scenario. The idea is to utilize the recurrent learning power of RNN to model the temporal aspects of the data streams, and at the same time, to make the recurrent model adaptive to evolving characteristics of data.

A. Contribution

MUSE-RNN structure is *automatically* developed from scratch by growing and pruning hidden units to fit the characteristics of the data. MUSE-RNN is also capable of adding hidden layers to its architecture to cope with concept drifts in the data stream. Further, instead of using complex control structures, the model adopts '*hidden layer voting*' and '*winning layer adaptation*' techniques to help recall prior tasks with the minimum exploitation of network parameters. Eventually, MUSE-RNN becomes fit for sequential learning in the restricted resource environment of streaming data processing. Furthermore, the recurrent learning strategy of MUSE-RNN is driven by '*teacher forcing*' principle that resolves vanishing and exploding gradient problems while making the model capable of capturing temporal nature of data streams. Overall, the proposed MUSE-RNN is well featured for classifying data streams with optimal usage of network parameters. The dynamically adjustable recurrent structure of the model helps

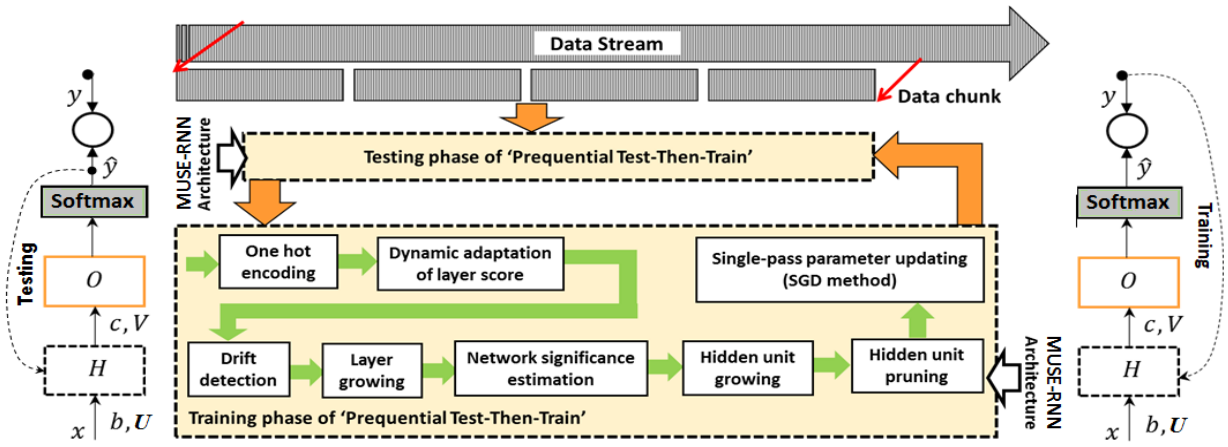

Fig. 1: Learning mechanism of the proposed MUSE-RNN. [In the training phase, the model utilizes the actual output y (see MUSE-RNN at right) to guide the classification as per teacher forcing policy, while this is replaced by predicted output $\hat{y}$ during testing phase (see left).]

it to efficiently capture the temporal dependencies in the data without leveraging complex control structure scattering much less network parameters than those of popular RNN models: LSTM, GRU. As far as we concern, this is the first teacher forcing based autonomous RNN model for data stream processing. We show empirically that MUSE-RNN is able to achieve average 12% improvement over the state-of-the-art accuracy under single-pass learning scenario of stream classification, evaluated using the standard data stream evaluation protocol of ***prequential-test-then-train***. Thus, our major contributions in this work are as follows:

- We propose MUSE-RNN, a novel RNN model which is able to *dynamically construct/adjust* its network structure by growing/pruning hidden units as well as hidden layers;
- We define the feature learning and layer growing in such a way that it maintains *teacher-forcing policy* and relieve MUSE-RNN from gradient calculation for hidden-hidden recurrent connection;
- We incorporate *dynamic layer voting* and *winning layer adaptation* mechanism in RNN model to appropriately recall relevant knowledge from past, with minimal usage of network parameters;

The rest of the paper is organized as follows. Section II, summarizes the recent and relevant works on streaming data classification. Section III presents the architectural as well as learning details of the proposed MUSE-RNN model. Section IV provides a thorough description of the empirical study along with the details of datasets, experimental set up, results, and major findings. Finally, we conclude in Section V.

II. Related Works

Streaming data classification is one of the hot topics in recent years and has attracted considerable research attentions. Most of the existing stream classification methods are based on incremental or continual learning which ensures learning of new information without perturbing previously acquired knowledge [8]. Diverse approaches supporting incremental learning can be found in the literature. For example, the incremental learning in progressive neural network (PNN) [11] is crafted on the creation of fresh network structure whenever a new task arrives, whereas the dynamically expandable network (DEN) [12] handles the situation by dynamically growing neurons in the hidden layer and timestamping these. Recently, the task-based hard attention mechanism (HAT) [13] is proposed as continual learning model which is a masking technique to preserve the knowledge of previous tasks without affecting the learning of current tasks. The 'attentional mechanism' is also employed by Liu et al. [14] for learning personalized classifier weights with respect to different instances. However, all these models are built upon fixed-layered network architecture, thus contributing less in adjusting the network capacity compared to dynamically changing network layer or depth structure [15]. The recent progress in incremental learning based stream analytics can also be noticed in the emergence of various ensemble models [16]. An ensemble method has advantage over single classifier model because it controls the bias-variance problem better, by providing sufficient diversity of its base classifiers. Incremental bagging and incremental boosting [17] are two well-known ensemble learning algorithms for streaming data prediction. Recently, Jung et al. [18] have proposed a variant of online boosting technique, based on ensemble of decision trees, which is fit for multiclass classification scenario. A CNN-based prototype ensemble model has been proposed by Wang et al. [19] for novel class detection and correction in incremental learning scenario. Learn++.NSE [4] is another popular ensemble-based classifier model which trains a new classifier for each batch of data it receives, and combines these using dynamically weighted majority voting.

Nevertheless, none of the above-discussed models is designed to capture the temporal dependency from input sequence, and definitely, they lose generalization power to some extent [3]. Though the internal memory of RNN is inherently capable of handing this issue, the existing RNN models for data stream analytics are rigid-structured, and thus, inflexible to address drifts and shifts of data streams. Moreover, the vanishing and exploding gradient problem, from which RNN suffers, are addressed using various gating mechanisms [20],

[21] which, however, incur expensive network parameters, and are hard to train in online learning situation.

III. Proposed Model: MUSE-RNN

This section covers the architectural as well as algorithmic details of MUSE-RNN, with respect to stream classification.

A. Problem Formulation

The data stream classification problem can be defined as the classification of continuously generated data chunks $\mathbb{D} = [\mathbb{D}_1, \mathbb{D}_2, \cdots, \mathbb{D}_C]$ where the number of chunks (C) and the distribution of the data are not known apriori. Each chunk may contain one or more data points/samples $\left[x^{(1)}, \cdots x^{(T)}\right]$ such that $T \geq 1$ and $x^{(i)} \in \mathbb{R}^D$, where D denotes the input feature dimension. Further, the sample size may depend in a random manner on the accumulating data [22]. This limits the feasibility of direct train-test partitioning methods and requires the model to be executed under *prequential test-then-train* protocol, where the data is first used to test the generative power of the classifier and then is exploited to update the model. Moreover, the streaming data is usually ***temporal in nature***, which is ignored by feed forward network architectures. This salient trait is addressed here by using a teacher forcing based recurrent network structure, featuring computationally wise solution (low network parameters) without any compromise of vanishing/exploding gradient problem. The fundamental working principle of MUSE-RNN is visualized in Fig. 1. We assume that the labels of the current samples become available when the next data chunk arrives.

B. Architecture

Primarily, the concept of MUSE-RNN uses a combination of dynamic number of self-configurable hidden layers ($1 \leq l \leq K$; K is the total number of hidden layers), all of which have connections to the output layer and have the ability to produce multi-class probability ($y_{(l)}$) by following the teacher forcing principle of a recurrent model. The recurrent architecture and the corresponding unfolded computational graph of the proposed MUSE-RNN at a particular timestamp t is depicted in Fig. 2. For each time step t, the D-dimensional input is $x^{(t)}$, the hidden layer activation for any layer l is $H_{(l)}^{(t)}$, un-normalized output is $O_{(l)}^{(t)}$ which is further updated through *softmax* layer to achieve the predicted output $\widehat{y}_{(l)}^{(t)} = softmax\left(O_{(l)}^{(t)}\right) \in \mathbb{R}^S$, where S is the number of classes, and the cross-entropy loss for layer l is $L_{(l)}^{(t)}$.

Each layer (l) is either rewarded or penalized based on its classification performance at each timestamp, and accordingly, the layer is assigned a voting score (ψ_l) in a dynamic manner (refer to Section III.E). The winning layer (l_W) in the earlier timestamp is considered for classifying the current data and is allowed to update its parameter to acquire the new knowledge. Thus, the predicted output at t becomes:

$$\widehat{y}^{(t)} = softmax\left(O_{(l_W)}^{(t)}\right) = softmax\left(c + V_{(l_W)} H_{(l_W)}^{(t)}\right) \tag{1}$$

where $c \in \mathbb{R}^S$ is the bias for the output layer, $V_{(l_W)} \in \mathbb{R}^{S \times e_{l_W}}$ is the hidden-to-output layer weight matrix, and $l_W = \underset{l}{argmax}\ (\psi_l)$ i.e. the layer corresponding to the ***highest voting score*** which varies dynamically as per classification performance of the layer throughout all the elapsed time stamps till $(t-1)$. This, in turn, helps to maintain the 'plasticity' of the model [5]. Further, in the form of old knowledge, MUSE-RNN keeps a record for the parameter configuration of the layer (l_M) producing latest ***best ever classification performance*** and utilizes the same to initialize the parameters for the newly introduced hidden layer in the forthcoming drift detection scenario (refer Section III.E). This helps the model to preserve its 'stability' towards retaining existing knowledge. Thus, MUSE-RNN adopts recurrent learning along with 'layer voting' and 'layer adaptation' mechanisms to achieve an improved recall power that can even maintain the 'stability-plasticity' trade off for overcoming the catastrophic forgetting situation [13] in a stream classification problem.

C. Parameter Learning

Typically, the hidden layer activation in MUSE-RNN is measured in terms of hyperplane activation [23], [24] as: $H_{(l)}^{i(t)} = \exp\left(-\eta \frac{d_{(l)}^{i(t)}}{max\left(d_{(l)}^{i(t)}\right)}\right)$ where, $i = 1, 2, \cdots e_l$ is the number of hidden units in the l-th hidden layer $H_{(l)}^{(t)}$ at time t, η is a parameter controlling the activation strength, and $d_{(l)}^{i(t)} = \frac{|y^{(t-1)}|_1 - S \cdot \left(b_{(l)}^i + U_{(l)i} H_{(0)}^{(t)}\right)}{\sqrt{1 + \sum_{j=1}^{D} U_{(l)ij}^2}}$ represents the *distance* from the $(t-1)$-th data point to the i-th feature in the feature hyperplane at the l-th layer. Here, $|y|_1$ indicates the 1-norm of y, $U_{(l)} \in \mathbb{R}^{e_l \times D}$ is the input to l-th hidden layer weight matrix, and $H_{(0)}^{(t)} \equiv x^{(t)}$. It may be noted that the *hyperplane-based activation* leads each hidden layer to be implicitly influenced by the output or summarized information from previous time stamp without involvement of external weights or parameters. ***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 multi-layered. Further, the use of output-to-hidden layer connection eventually helps the model to learn from exact inputs from earlier time stamps as per ***teacher forcing policy*** [25]. In case of testing, the original output $y^{(t-1)}$ is replaced by the predicted output $\widehat{y}^{(t-1)}$ (refer to Fig. 1), to maintain the constraint of unlabeled data. Since the output-to-hidden recurrent connection in MUSE-RNN is not explicit, during parameter updating, the back-propagation algorithm is applied in isolation to each time stamp. This not only reduces the computational time but 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 (using stochastic gradient descent or SGD) in MUSE-RNN for internal nodes at each time stamp becomes as follows:

$$\frac{\partial L}{\partial O_i^{(t)}} = \left(\nabla_{O^{(t)}} L\right)_i = \left(\widehat{y}_i{}^{(t)} - y_i^{(t)}\right) \tag{2}$$

$$\nabla_{H_{(l)}^{(t)}} L = V_{(l)}^{\top} \left(\nabla_{O^{(t)}} L\right) \tag{3}$$

Once the gradients on the internal nodes are obtained, we compute the gradients for bias parameters (b,c) and weight parameters ($V_{(l)}$,$U_{(l)}$) as follows:

$$\nabla_c L = \left(\frac{\partial O^{(t)}}{\partial c}\right)^{\top} \left(\nabla_{O^{(t)}} L\right) = \left(\nabla_{O^{(t)}} L\right) \tag{4}$$

$$\nabla_{b_{(l)}} L = \left(\frac{\eta}{mv} \cdot H_{(l)}^{(t)}\right) \circ \left(\nabla_{H_{(l)}^{(t)}} L\right) \tag{5}$$

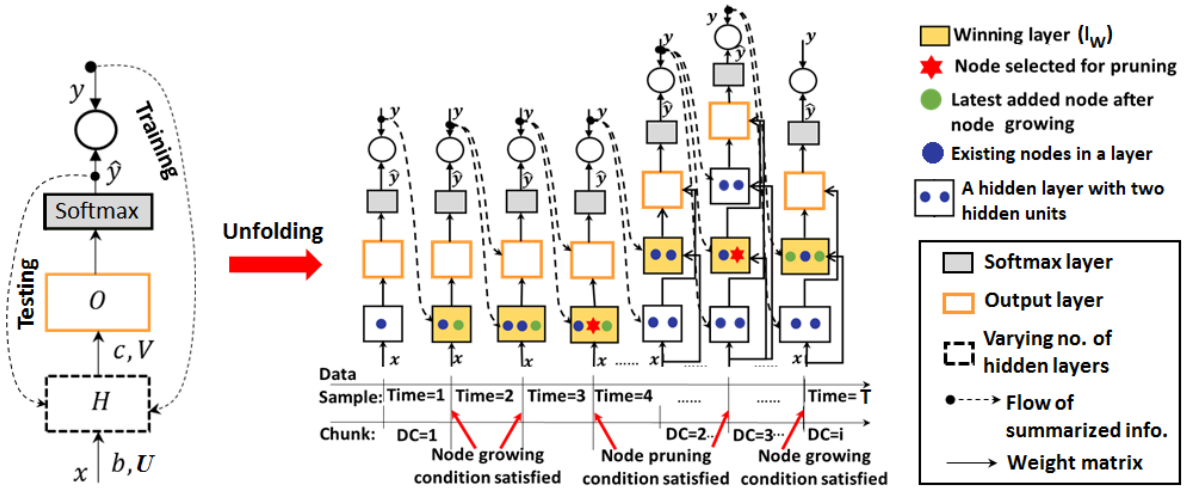

Fig. 2: MUSE-RNN: Recurrent architecture and unfolded computational graph

TABLE I: Summary of notations used to describe MUSE-RNN parameter learning

Notations	Meaning
D	Input feature dimension
S	No. of class
$x^{(i)}$	i-th datapoint or sample in a chunk
K	Total number of layers in the MUSE-RNN architecture
e_l	Number of hidden units in the l-th hidden layer
$H_{(l)}^{(t)}$	Activation of l-th hidden layer for timestamp t
$O_{(l)}^{(t)}$	Un-normalized output
$\widehat{y}_{(l)}^{(t)}$	Predicted normalized output using layer l
$\widehat{y}^{(t)}$	Predicted normalized output of MUSE-RNN
$y^{(t)}$	Actual output
ψ_l	Score of layer l
l_W	Wining layer
l_M	Layer producing best ever classification performance
η	Activation strength controlling parameter
L	Cross-entropy loss
$d_{(l)}^{i(t)}$	Distance from the previous data-point to the i-th feature in the l-th feature hyperplane
b	bias for the input layer
c	bias for the output layer
U_l	input layer to l-th hidden layer weight matrix
V_l	l-th hidden layer to output layer weight matrix

$$\nabla_{V_{(l)}} L = \sum_i \left(\frac{\partial L}{\partial O_i^{(t)}}\right) \nabla_V O_i^{(t)} = \left(\nabla_{O^{(t)}} L\right) \cdot \left(H_{(l)}^{(t)}\right)^\top \quad (6)$$

$$\nabla_{U_{(l)}} L = \left(\left(\frac{\eta}{mv} \cdot H_{(l+1)}^{(t)}\right) \circ \left(\nabla_{H_{(l+1)}^{(t)}} L\right)\right) \left(H_{(l)}^{(t)}\right)^\top \quad (7)$$

where mv corresponds to the maximum of the *distances* from sample to the features on the l-th feature plane, at time t. Table I summarizes the notations used in Sections III-B-III-C.

D. Self-construction of Within Layer Structure

MUSE-RNN features flexible layer structure. The number of the nodes e_l per layer l $(1 \leq l \leq K)$ is not fixed. It starts from scratch with a single hidden node in a single layer, and then, the count of hidden node is dynamically adjusted during training phase, depending on the distributional change in the input data stream. The hidden node growing and pruning condition in MUSE-RNN is driven by network significance (NS) [7] which represents the generalization power of the network in terms of bias and variance: $NS = Var(O) + Bias(O)^2 = (E[O^2] - E[O]^2) + (E[O] - y)^2$ where, $E[O]$ represents the expectation of un-normalized output from MUSE-RNN. Note that the NS method in MUSE-RNN has to be redefined because its original form only covers the case of sigmoid activation function. For any time instant t, the $E[O]$ for the MUSE-RNN network can be recursively estimated as follows: $E[O] = \int_{-\infty}^{\infty}(c + V \cdot H)p(H)dH = c + V \cdot E[H]$, where $E[H] = \int_{-\infty}^{\infty} e^{\left(-\frac{d}{max(d)}\right)} p(d)dd = e^{\left(-\frac{E(d)}{max(E(d))}\right)}$ and $E[d^i]_{k=1} = \frac{1}{S}\sum_{m=1}^{S} \frac{|y|_1 - \left(b^i + U_i \cdot \mu\right)}{\sqrt{1+\sum_{j=1}^{D} U_{ij}}}$, for $1 \leq i \leq e_l$; μ is the mean corresponding to the data distribution.

Hidden node growing policy: A hidden node is added to the winning layer (l_W) when the model goes into underfitting zone, identified by high bias condition: $\mu_{Bias}^t + \sigma_{Bias}^t \geq \mu_{Bias}^{min} + \pi\sigma_{Bias}^{min}$, where μ_{Bias}^t, σ_{Bias}^t are respectively the mean and the standard deviation of bias at the t-th time instant, and μ_{Bias}^{min}, σ_{Bias}^{min} correspond to the same for the minimum bias till the time instant t. The value of π is selected as $1.3\exp(-bias^2) + 0.7$, leading to attain confidence level in between 50% and 95.2%. The underfitting situation is resolved by increasing complexity of the network structure, by introducing more units in the hidden layer. Once a new hidden node is added to the layer, the associated parameters (b, U, V) are randomly sampled from the scope of [-1,1] so that it increases the interestingness (G) of the layer feature, i.e. $G(\widehat{H}_{(l_W)}) > G(H_{(l_W)})$, where $\widehat{H}_{(l_W)}$ and $H_{(l_W)}$ denote the activations for the winning layer l_W after and before the node growth, respectively. The interestingness G is estimated in terms of Gini index [26]. To be noted, the estimation of Gini index here is solely based on the current activation of the winning layer, considering its hidden state before and after the growing of hidden node. Therefore, this index value is not at all required to be computed in online fashion. Further, since the number of hidden units in the

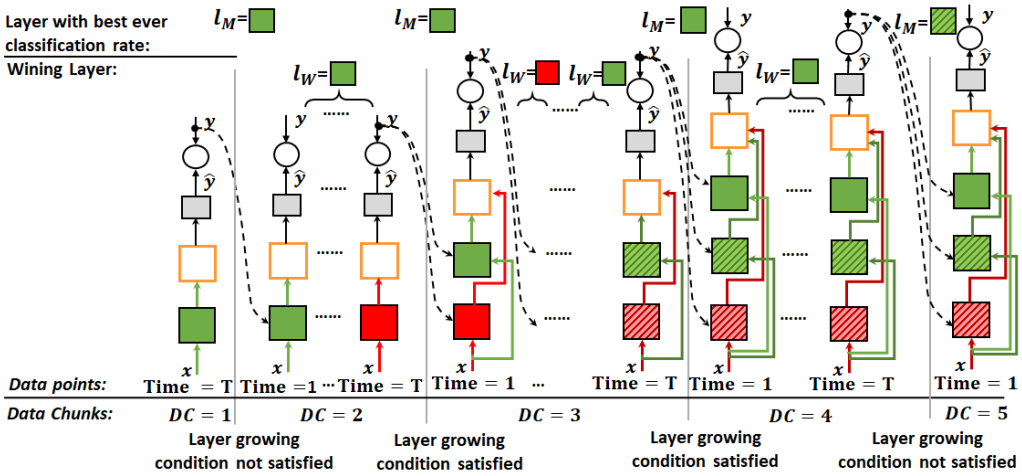

Fig. 3: A typical scenario of hidden layer adjustment in MUSE-RNN

layer at the given time stamp is not considerably high, the computation of the Gini index takes negligible time.

Hidden node pruning policy: The goal of pruning hidden unit is to reduce the network complexity to deal with overfitting situation which can be identified through high variance condition: $\mu^t_{Var} + \sigma^t_{Var} \geq \mu^{min}_{Var} + 2\chi\sigma^{min}_{Var}$, 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} correspond to 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. Once high variance is detected, the hidden unit associated with the least ever average activation value in the winning layer l_W is chosen for pruning. In other words, the pruning candidate is the k-th unit in l_W such that $k = \underset{i}{argmin}\left(\lim_{T\to\infty}\sum_{t=1}^{T}\frac{H^{i(t)}_{l_W}}{T}\right)$. Fig. 2 depicts a typical instance of unfolded MUSE-RNN structure along with hidden unit growing and pruning scenario.

E. Layer Adjustment to Adapt to Concept Drift

A drift situation indicates the inadequacy of acquired knowledge to describe the new data distribution. In order to maintain sequential learning, this issue needs to be resolved without sacrificing the already acquired knowledge over the observed data distribution. MUSE-RNN handles this situation by adding more layers in the network and only allowing the currently best performing layer to adapt with the new data. Other layers are kept unchanged to retain the old knowledge.

Hidden layer growing policy: The drift scenario is detected by means of Hoeffding's error bound technique [27]. When the drift status is signaled (see Algorithm 2), a new layer l_{new} is added in the network with direct feed forward connection from and to the input and output layer, respectively. Further, the new layer activations are computed in terms of hyperplane activation function which leads to implicit recurrent connection of l_{new} to the summarized information from the output of previous time stamp. This ensures multi-layer MUSE-RNN learning based on teacher forcing mechanism. The newly created layer parameters are initialized with the *parameter configuration* of the layer (l_M) producing best ever classification accuracy during the elapsed time period, i.e. $V_{l_{(new)}} \leftarrow V_{l_{(M)}}$ and $U_{l_{(new)}} \leftarrow U_{l_{(M)}}$. This helps in immediate recovery from drift situation if the drift is recurrent. Else, this triggers the model to go back to the stable situation and learn the new knowledge afresh, without affecting the present knowledge, since the other layers are kept frozen. To be noted, the best ever classification accuracy may not always be the one with the best parameters for a given specific recurring concept. However, our method still works desirably because of the following reason. If the best ever classification accuracy corresponds to the current recurring concept, definitely the reuse of parameter configuration helps retaining the performance. Otherwise, even if the best ever classification accuracy does not correspond to the current recurring concept, the reuse of its parameter configuration helps the model achieving divergence from the latest parameter configuration, and subsequently provides opportunity to search the solution space in a different direction, through the parameter updating process in the next step. MUSE-RNN layer growing policy is pictorially explained in Fig. 3. Incidentally, MUSE-RNN does not offer any layer pruning facility, since removal of a hidden layer may lead to catastrophic forgetting situation by sudden loss of significant amount of knowledge relevant to an old task.

Dynamic adjustment of layer score: The key objective of layer scoring is to properly recall the knowledge acquired from data. This is achieved in two ways: firstly, by assigning the highest score value to the layer producing best classification performance over the current data stream, and secondly, by allowing only this winning layer to back propagate the error derivative and accordingly update its associated parameters to learn the new knowledge while keeping the other layers unchanged to retain the old knowledge. This mechanism also helps MUSE-RNN to achieve diversity and ensures that each layer is assigned its own zone of influence. For each layer l, the score value $\psi_l \in [0, 1]$ is initialized with its classification rate when it appears first, and then, the value is dynamically

updated either by rewarding (as per eq. 8) or by penalizing the layer (as per eq. 9), to reflect its current performance.

$$\psi_l = min(1, \psi_l + \epsilon) \tag{8}$$

$$\psi_l = max(0, \psi_l - \epsilon) \tag{9}$$

where $\epsilon \in [0, 1]$ is the score adjustment factor which indicates the significance of reward or penalty. The eq. 8 and 9 ensure that the score for each layer remains in between 0 and 1. This scoring-based layer adaptation helps in recalling most-relevant knowledge without any gating mechanism, and thus, achieves improved memorization power with less exploitation of network parameter. The overall learning process of proposed MUSE-RNN is presented through Algorithm 1.

Algorithm 1: MUSE-RNN Learning Model

Required: $x = [x_1, x_2, \cdot, x_D]^\top$, Input to process
Required: $y = [y_1, y_2, \cdot, y_S]^\top$, Target output /* $\left\{x^{(t)}, y^{(t)}\right\}$, $t = 1, 2, \cdots, T$ is the data stream

/* Variable definition and Initialization */
ψ_l: Score for the l-th layer
l_W: Winning hidden layer, or the layer with highest score; $l_W = 1$;
l_M: Hidden layer with best ever classification performance; $l_M = 1$;
/* Processing */
for $DC = 1$ *to the no.of data chunk in the stream* **do**
  for $t = 1$ *to the no.of training data in the chunk* DC **do**
    Testing Phase:
    Execute forward propagation computation for all l $(1 \leq l \leq K)$(refer Section III.C);
    Generate the output class label as $\widehat{y}^{(t)} = \widehat{y}^{(t)}_{l_W}$ (refer Section III.B);
    /* Dynamic updating of layer score */
    if *(DC==1 and t==1)* **then**
      for $l = 1$ *to* K **do**
        $\psi_l \leftarrow$ classification-rate of l;
      end
    end
    else
      $\psi_l \leftarrow$ Update score for l as per eqs. 8 and 9
    end
    Training Phase:
    /* Drift Detection */
    Perform drift detection based on the Hoeffding's error bound technique. (see **Algorithm 2**)
    drift-status=Drift-detection(y,$\widehat{y}$);
    if *(drift-status==DRIFT)* **then**
      /* Hidden layer growing */
      create layer l_{new} with feed forward connection from and to input layer and output layer;
      $V_{l_{new}} \leftarrow V_{l_M}$;
      $U_{l_{new}} \leftarrow U_{l_M}$;
      classification-rate of l_{new} $\leftarrow$classification-rate of l_M;
      $\psi_{l_{new}} \leftarrow \psi_{l_M}$;
      K$\leftarrow$K+1;
    end
    if $t > 1$ **then**
      $l_M \leftarrow$ layer l $(1 \leq l \leq K-1)$ performing the best till time $t-1$;
    end
    $l_W \leftarrow \underset{l}{argmax}\ (\psi_l)$
    Update l_W as per hidden unit growing and pruning policy (refer Section III.D)
    Calculate loss function and execute gradient computation as described in Section III.B-III.C;
    Update weights and biases (refer Section III.C);
  end
end

IV. Experimental Evaluation

The details of experimental evaluation for MUSE-RNN are thoroughly described in the subsequent subsections.

Algorithm 2: Drift-detection(y,$\widehat{y}$)

Required: $y \in \mathbb{R}^S$, Actual class labels
Required: $\widehat{y} \in \mathbb{R}^S$, Predicted class labels

/* Initialization */
cut : cutting point=data point position (initialized as 1) in the current chunk;
/* Determine switching point (Cutting point) */
for $t = 1$ *to* T **do**
  Estimate accuracy matrices $F \in \mathbb{R}^{T \times S}$ and $G \in \mathbb{R}^{cut \times S}$ such that these records 1 if corresponding $y \neq \widehat{y}$, else 0;
  $\widehat{F} \leftarrow mean(F)$;
  $\widehat{G} \leftarrow mean(G)$;
  Calculate Hoeffding error bound $errorb_{\widehat{F}}$ and $errorb_{\widehat{G}}$
  if $(\widehat{F} + errorb_{\widehat{F}} \leq \widehat{G} + errorb_{\widehat{G}})$ **then**
    $cut \leftarrow t$;
    Estimate accuracy matrices $H \in \mathbb{R}^{(T-cut) \times S}$ such that it records 1 if corresponding $y \neq \widehat{y}$, else 0;
    $\widehat{H} \leftarrow mean(H)$;
    break;
  end
end
if $|\widehat{H} - \widehat{G}| > error_{drift}$ **then**
  $driftStatus \leftarrow$ DRIFT /* $error_{drift}$ is predefined
end
else if $error_W < |\widehat{H} - \widehat{G}| < error_D$ **then**
  $driftStatus \leftarrow$ WARNING /* $error_W$ and $error_D$ are predefined
  Create data in buffer;
end
else
  $driftStatus \leftarrow$ STABLE
  Remove data from buffer;
end
return $driftStatus$

TABLE II: Specifications for the datasets used in experimentation

Datasets	Specifications				
	#Instance	#Attr.	#Target	#Task	Characteristics
SUSY	5000000	18	2	5000	Synthetic, stationary
ELECT.	45312	8	2	45	Real, non-stationary with covariate drifts
HYPER.	120000	4	2	120	Synthetic, non-stationary with gradual drifts
SEA	100000	3	2	100	Synthetic, non-stationary with recurring drifts
WEATHER	18000	8	2	18	Real, non-stationary with recurring concept drifts
R-MNIST	65000	784	10	65	Synthetic, non-stationary with abrupt concept drifts
P-MNIST	70000	784	10	70	Synthetic, non-stationary with recurrent drifts

A. Datasets

We evaluate MUSE-RNN with respect to seven datasets: *i) Susy* [28], *ii) Electricity-pricing* [4], *iii) Hyperplane* [29], *iv) Sea* [30], *v) Weather* [4], *vi) Rotated-MNIST* [31], and *vii) Permuted-MNIST* [31]. Susy is widely used as data stream for big data problem, whereas Electricity-pricing, Hyperplance, and Sea are well-used in literature as examples of data streams with variants of concept drifts (refer Table II). The Electricity-pricing and Weather data are real-world data showing prominent temporal aspects. As clearly stated in literature [3], if the electricity price goes up now, "it is more likely than by chance to go up again, and vice versa". This data is highly autocorrelated with "very clear cyclical peaks at every 24 hours, due to electricity consumption habits". On the other sides, the Weather data not only shows cyclical seasonal changes, but also has long-term climate change effect. Though the dataset Sea is artificial/synthetic, it is prepared along with induced recurring environment [4], and consequently, this also shows temporal dependence. We use Permuted (P)-MNIST and

Rotated (R)-MNIST as two variants of MNIST dataset where each task contains digits transformed by permutation of pixels or rotation by fixed angle $\in[0°,180°]$, respectively. These two problems are popular continual learning problems and are put forward to evaluate against a high input dimension. The details of all these datasets are summarized in Table II.

B. Baselines

We compare MUSE-RNN with eight recent baselines. All these algorithms deal with one or more of the various issues in concept drift scenario, including one-pass learning, learning of new knowledge and preserving the previous knowledge (i.e. handling the issue of catastrophic forgetting) and so on. Thus, these are appropriate as baselines for streaming data analysis.

- **PNN** [11]: Continual learning algorithm; uses the progressive network for each new task, and augments it with adapters and nonlinear lateral connections to keep the useful knowledge from previous tasks.
- **DEN** [12]: Extension of PNN; puts forward selective retraining, splitting and duplicating methods.
- **HAT** [13]: Continual learning algorithm; based on the task embedding, maintaining the previous task's information without involving the current task.
- **pENsemble+** [32]: Built on the concept of evolving fuzzy parsimonious classifier; Equipped with online active learning and ensemble merging scenarios, which reduces operator annotation effort with reduced complexity.
- **Incremental Bagging** [17]: Online version of 'Bagging', an well-known ensemble learning model, working with low overhead. However, it requires large execution time.
- **Learn++.NSE** [4]: Appropriate for dealing with variants of drift scenarios; Capable of learning in non-stationary environments. However, it suffers from high structural complexity and considerably long execution time.
- **Online Multiclass (OMC) Boosting** [18]: A variant of online boosting algorithm. Based on ensemble learning; Uses optimal no. of classifier in the ensemble to achieve desired accuracy with reduced computational cost.
- **RNN_tanh**: Conventional RNN model [25] with single layer using $tanh$ activation and learning based on back propagation through time (BPTT).

While comparing with the baselines we tried our best to ensure fair comparison. We initially started with the same parameter settings as mentioned in their respective source files or manuscripts. If their result is surprisingly poor, we tuned the parameters empirically to achieve the best performance in every case. However, to be noted, our proposed model (MUSE-RNN) is not driven by any ad-hoc choice of parameters.

C. Experimental Settings

Our proposed MUSE-RNN and all the above mentioned benchmark models are executed under 'prequential test-then-train' environment in the same platform of 3.20 GHz Intel(R) Xeon(R) CPU E5-1650 processor and 16 GB RAM. The learning rate, control parameter (η), and layer score adjustment factor (ϵ) for MUSE-RNN are always fixed at 0.001, 0.05, and 0.1 for every dataset. The learning performance is evaluated on four criteria: *classification rate* (**CR**), *used parameter count* (**PC**), *no. of hidden layers* (**HL**), and *execution time* (**ET**). All the results are recorded as average of *five* random seeds. Additionally, we also confirm our model performance through *Wilcoxon* statistical tests, considering significance level of 5%.

TABLE III: Comparative performance study of MUSE-RNN

Data	Models	Metrics			
		CR	**HL**	**PC**	**ET** (sec.)
Susy	PNN	68.94 ± 4.08	3	424	345K
	DEN	63.15 ± 10.06	1	212	8K
	HAT	73.85 ± 3.18	2	342	16K
	pENsemble+	76.99 ± 4.6	19	3249	35K
	OMC-Boosting	77.13 ± 1.43	NA	NA	14K
	Learn++.NSE	—	NA	NA	—
	I-Bagging	72.8 ± 3.1	NA	NA	73K
	RNN_tanh	64.31± 1.5	1	312	289K
	MUSE-RNN	**78.14 ± 1.6**	10	1800	21K
Electricity	PNN	57.84 ± 4.52	3	1868	51.345
	DEN	56.54 ± 7.66	1	178	72.54
	HAT	56.63 ± 8.04	2	242	145.57
	pENsemble+	72.60± 12.1	1	243	170
	OMC-Boosting	77.27 ± 4.57	NA	NA	56.8
	Learn++.NSE	77.18 ± 9.3	NA	NA	169.88
	I-Bagging	72.8 ± 10.88	NA	NA	5K
	RNN_tanh	65.12± 7.21	1	104	2.5K
	MUSE-RNN	**78.10 ± 7.6**	4	184	117.06
Hyperplane	PNN	85.07 ± 7.12	3	560	190.196
	DEN	91.83 ± 4.17	1	58	202.57
	HAT	77.9 ± 10.76	2	98	370.8
	pENsemble+	87.60± 6.2	5	75	120
	OMC-Boosting	86.18 ± 3.73	NA	NA	111.74
	Learn++.NSE	90.35 ± 2.48	NA	NA	374
	I-Bagging	81.39 ± 2.2	NA	NA	1.7K
	RNN_tanh	76.55 ± 2.82	1	80	6.7K
	MUSE-RNN	**92.64 ± 2.15**	3	48	250.39
Sea	PNN	84.87 ± 6.52	3	353	152.46
	DEN	79.95 ± 19.28	1	38	169.72
	HAT	74.65 ± 10.1	2	72	327
	pENsemble+	92.00± 6	5	32	230
	OMC-Boosting	86.78 ± 3.85	NA	NA	77.44
	Learn++.NSE	90.17 ± 5.96	NA	NA	268
	I-Bagging	84.6 ± 13	NA	NA	1.5K
	RNN_tanh	75.17 ± 2.94	1	42	5.7K
	MUSE-RNN	**92.37 ± 6.11**	2	28	116.24
Weather	PNN	68.576 ± 1.05	3	1074	17.72
	DEN	68.6 ± 4.105	1	112	31.19
	HAT	68.54 ± 3.99	2	242	54.74
	pENsemble+	**78.78± 4**	2	81	28.0
	OMC-Boosting	65.32 ± 1.98	NA	NA	46.86
	Learn++.NSE	76.2 ± 3.82	NA	NA	26.64
	I-Bagging	75.49 ± 3.44	NA	NA	1.8K
	RNN_tanh	69.14 ± 4.04	1	64	20.15
	MUSE-RNN	76.96 ± 6.92	1	21	17.46
R-MNIST	PNN	56.19 ± 10.94	3	170K	128.91
	DEN	61.48 ± 21.75	2	290K	371.07
	HAT	64.52 ± 11.33	2	24.9K	190.59
	pENsemble+	NA	NA	NA	NA
	OMC-Boosting	26.07 ± 5.8	NA	NA	5K
	Learn++.NSE	NA	NA	NA	NA
	I-Bagging	NA	NA	NA	NA
	RNN_tanh	63.23 ± 7.54	1	198K	94K
	MUSE-RNN	**76.27 ± 4.9**	2	125K	190.01
P-MNIST	PNN	64.42 ± 8.77	3	170K	152.95
	DEN	52.08 ± 22.6	2	290K	399.83
	HAT	59.64 ± 18.88	2	24.9K	207.04
	pENsemble+	NA	NA	NA	NA
	OMC-Boosting	35.58 ± 20.51	NA	NA	5K
	Learn++.NSE	NA	NA	NA	NA
	I-Bagging	NA	NA	NA	NA
	RNN_tanh	69.62 ± 11.36	1	198K	101K
	MUSE-RNN	**83.87 ± 13.42**	4	200K	416.10

D. Results and Discussion

The performance of MUSE-RNN, in comparison with PNN, DEN, HAT, incremental bagging, OMC-boosting,

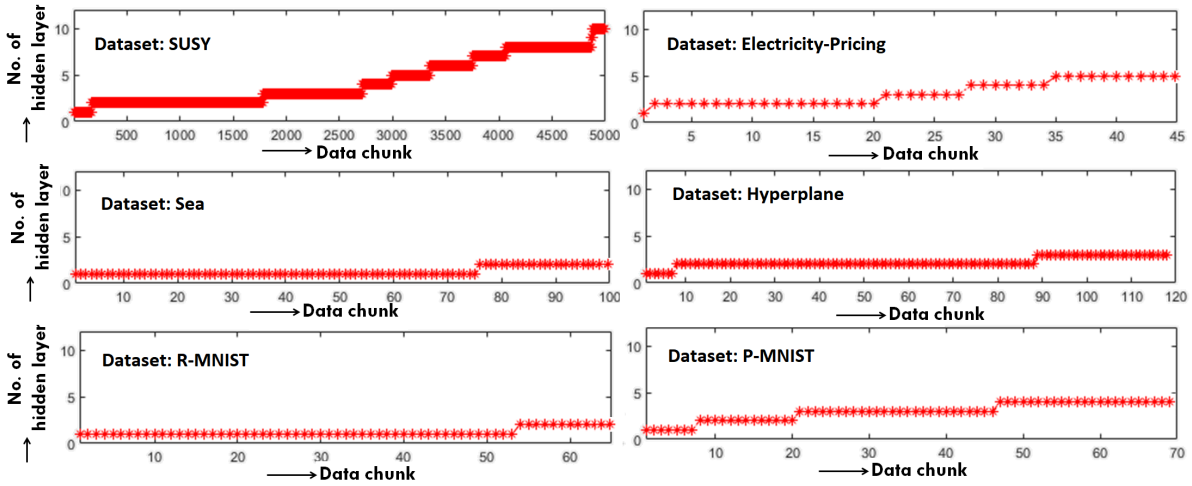

Fig. 4: Typical instance of hidden layer evolution in MUSE-RNN

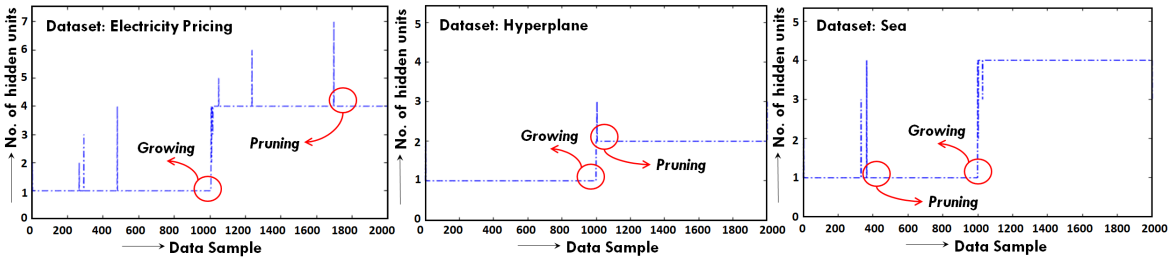

Fig. 5: Typical instance of hidden unit adjustment in MUSE-RNN

pENsemble+, Learn++.NSE, and traditional RNN_tanh is summarized in Table III. For I-bagging, OMC boosting, and Learn++.NSE, PC=NA and HL=NA, since these are based on decision tree model. Further, the I-bagging, pENsemble+, and Learn++.NSE models could not be executed on MNIST variants, because of the very high dimensionality of the data. In Figs. 4-5, we additionally show how the number of required hidden layer and hidden unit in proposed MUSE-RNN model are prudently adjusted with the ever changing flow of data stream. We also have performed **ablation study** considering MUSE-RNN without hidden unit growing/pruning and recurrent learning features (refer Table V). On analyzing the tables and the figures, we can infer the following.

Comparison on classification rate (CR): It is evident that even with the constraint of single-pass data scanning, the proposed MUSE-RNN is able to achieve state-of-the-art classification accuracy for each dataset. From Table III, it can be noted that the average percentage improvement of MUSE-RNN over state-of-the-art techniques in classifying Susy, Electricity-pricing, Hyperplane, Sea, Rotated (R)-MNIST, and Permuted (P)-MNIST, are more than 7%, 10%, 8%, 8%, 30%, and 23% respectively. The modelling of temporal dependencies through the dynamically adaptive recurrent architecture of MUSE-RNN leads the model to attain this encouraging performance. The 'dynamic layer voting' and 'winning layer adjustment' further assists MUSE-RNN to appropriately recall relevant knowledge. Although the classification rates of pENsemble+, OMC-boosting, and Learn++.NSE are sometimes very near to that of the MUSE-RNN for some datasets, these techniques either become intractable or produce substantially poor performance while dealing with high dimensional data or high volume of data. For example, though the performance of pENsemble+ in case of Weather dataset is slightly higher than MUSE-RNN, pENsemble+ performs considerably poor in case of other datasets, such as Electricity pricing, Hyperplane, P-MNIST, R-MNIST etc. To be noted, pENsemble+ outperforms in case of weather data primarily because of its intrinsic fuzziness for handling data-uncertainty. Hence, overall, the performance of MUSE-RNN is comparatively *more consistent* over all the datasets. Further, the average percentage improvement of MUSE-RNN over fixed structured RNN (using BPTT) model is more than 13%, and also, it is achieved with notably less execution time (refer Table III). This demonstrates the worth of dynamically evolving recurrent architecture of MUSE-RNN along with implicit teacher-forcing-based learning policy.

Comparison on parameter count (PC) and hidden layer (HL) requirement: As shown in Table III, though the number of parameters and layer count for MUSE-RNN are sometimes found to be higher, the beauty lies in the fact that these hidden layers and parameter requirements are not prefixed from the beginning. As depicted in Figs. 4 and 5, the execution of the proposed MUSE-RNN always starts with only one unit in a single hidden layer, and then, gradually the hidden layers

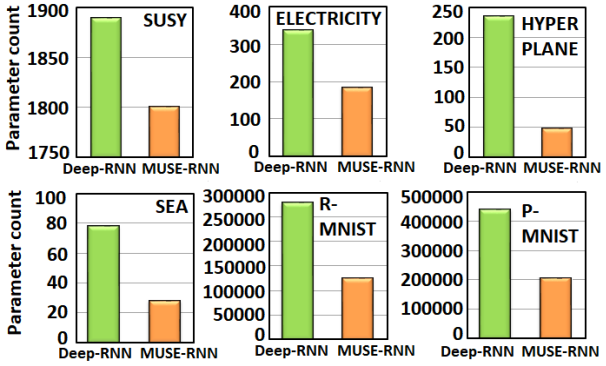

Fig. 6: Comparative study of parameter requirement in multilayer architecture: Traditional deep RNN architecture vs. MUSE-RNN

TABLE IV: MUSE-RNN performance validation using statistical test

Datasets	Models							
	PNN	DEN	HAT	Incr-Bagg	pENs emble+	OMC -Boost	Learn ++ NSE	RNN _tanh
SUSY	X	X	X	X	X	X	—	X
ELECT.	X	X	X	X	X	X	X	X
HYPER.	X	X	X	X	X	X	X	X
SEA	X	X	X	X	X	X	X	X
WEATHER	X	X	X	X	X	X	X	X
RMNIST	X	X	X	X	—	—	—	X
P-MNIST	X	X	X	X	—	—	—	X

X: Reject the null hypothesis that a model performs better than MUSE-RNN
—: Main experimentation could not be conducted

are increased and the hidden units are added or removed from the winning layer so as to cope with the time varying distribution and conceptual drift of the streaming data. This gradual and on-the-fly structural adjustment helps MUSE-RNN in achieving desired accuracy with optimal number of parameters in the classifier model.

For example, in order to use same number of hidden layers (as finally adjusted by MUSE-RNN for each dataset) a traditional RNN (with full forward as well as recurrent connections) needs to deal with substantially larger number of parameters, whereas MUSE-RNN is able to judiciously choose subset of these parameters to successfully accomplish the task (see Fig. 6).

Comparison on Execution Time (ET): From the results presented in Table III, it may also be noted that even with the dynamic layer adaptation overhead, MUSE-RNN is able to achieve acceptable accuracy within reasonable time. Though OMC-boosting needs comparatively less time to produce almost similar accuracy in case of Susy and Electricity datasets, the accuracy of OMC-boosting is considerably poor compared to MUSE-RNN for the other datasets, especially for those with high input feature dimension.

Results of statistical tests: The performance of MUSE-RNN is also validated using *Wilcoxon statistical test*, as summarized in Table IV. The results validate that, in every case, the proposed MUSE-RNN performs statistically better or similar classification compared to the other considered models with a significance level of 5%.

Results of ablation study: The results of ablation study for our proposed MUSE-RNN model is summarized in the Table

TABLE V: Summary of the ablation study for MUSE-RNN

Ablation Scheme	Datasets	Metrics			
		CR	HL	PC	ET (sec.)
Proposed Model WITHOUT hidden unit pruning feature	Susy	76.56 ±1.93	9	3.1K	22K
	Electricity	72.58 ± 10.84	2	20	70.15
	Hyperplane	91.73 ± 2.36	3	70	261.92
	Sea	87.39 ± 7.28	2	32	117.50
	Weather	74.45 ± 4.56	2	90	17.82
	R-MNIST	71.09 ± 3.10	3	220K	2.1K
	P-MNIST	82.72 ± 13.29	4	240K	1.9K
Proposed Model WITHOUT hidden unit growing feature	Susy	76.11 ± 1.35	10	40	20K
	Electricity	73.63 ± 7.16	4	18	132.11
	Hyperplane	88.46 ± 2.2	3	18	242.19
	Sea	84.40 ± 7.56	2	10	115.94
	Weather	74.18 ± 6.21	1	13	15.83
	R-MNIST	19.64 ± 1.85	3	809	151.4
	P-MNIST	18.81 ± 1.77	4	817	161.93
Proposed Model WITHOUT recurrent learning feature	Susy	78.04 ± 2.82	9	10.6K	15K
	Electricity	63.45 ± 8.82	3	304	28.74
	Hyperplane	90.62 ± 3.61	1	30	50.81
	Sea	89.08 ± 7.67	1	74	43.20
	Weather	71.10 ± 6.10	1	35	7.57
	R-MNIST	74.37 ± 8.56	2	200K	632.64
	P-MNIST	80.16± 14.27	3	220K	564.05

V. It can be noted from the table that, in case the hidden layer pruning feature is removed from MUSE-RNN, the number of parameter requirements, and hence the execution time is increased substantially. However, the classification rate of the model shows no improvement, and rather, it is deteriorated to some extent because of the over-fitting issue. On the other side, if the hidden unit growing feature is removed from the proposed MUSE-RNN, then the number of parameter requirement, and hence, the computational time is reduced to certain extent. However, this leads to a considerable deterioration of the model performance because of the insufficient usage of the parameters and the under-fitting issue. Thus, the ablation study further demonstrates the effectiveness of the fully autonomous and self-evolving property of the proposed MUSE-RNN.

In order to assess the **effectiveness of recurrent learning** for modeling **temporal aspects** of the data streams, we use another ablation scheme, where the recurrent learning feature of the model is removed. The recurrent learning capability of MUSE-RNN is primarily originated by default due to the use of hyperplane activation in the hidden layer. Hence, in this scheme, we simply replace the hyperplane activation with the standard sigmoid activation. Interestingly, it can be noted from the Table V that without recurrent learning feature, though the execution time is certainly reduced, the classification performance of the model is significantly deteriorated especially for those data streams that show high degree of temporal dependency, such as Electricity pricing, Weather, and Sea.

Overall, the empirical study reveals that the dynamically evolving recurrent network model, as adopted in our proposed MUSE-RNN, have enough potentials to perform desirably under the data stream classification scenario. Whether the improved classification performance of MUSE-RNN is due to its ability to do deal with concept drift or due to its ability to capture temporal dependencies can be decided based on the property of the considered dataset as well as the other algorithm with which we are comparing. For example, the DEN algorithm is primarily proposed to deal with online learning and catastrophic forgetting issues of concept drift han-

dling. Thus, the better performance of MUSE-RNN compared to DEN over the Electricity dataset seems due to temporal dependence modeling ability of MUSE-RNN. Contrarily, the better performance of MUSE-RNN compared to DEN over the R-MNIST dataset (showing no temporal dependence) seems due to better concept drift handling ability of MUSE-RNN.

V. Conclusions

This paper proposes a novel recurrent network model, called MUSE-RNN, for real-time classification of streaming data with a special treatment towards capturing the temporal aspects of the data as well. The proposed MUSE-RNN learns under teacher forcing environment in single-pass manner and features automatic layer construction as well as adjustment capability. Further, MUSE-RNN has an unique property of recalling prior tasks with minimum exploitation of network parameters. Comparative studies with recently proposed incremental and continual learning techniques show that MUSE-RNN is able to produce comparable or even better accuracy in stream classification scenario. Ample scopes remain in future to further extend MUSE-RNN with added convolution mechanism to deal with complex image streams. In order to support reproducible research initiative, source code of MUSE-RNN along with sample datasets has been shared[1].

Acknowledgment

This work is financially supported by the NTU-SUTD-ASTAR AI partnership grant (#RGANS1902).

References

[1] J. Gama, "A survey on learning from data streams: current and future trends," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 45–55, 2012.

[2] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream clustering and classification," *Knowledge and information systems*, vol. 45, no. 3, pp. 535–569, 2015.

[3] A. Bifet, J. Read, I. Žliobaitė, B. Pfahringer, and G. Holmes, "Pitfalls in benchmarking data stream classification and how to avoid them," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 465–479.

[4] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2283–2301, 2013.

[5] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[6] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 2990–2999.

[7] A. Ashfahani and M. Pratama, "Autonomous deep learning: Continual learning approach for dynamic environments," in *SIAM International Conference on Data Mining (SDM)*., 2019.

[8] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, vol. 31, no. 4, pp. 497–508, 2001.

[9] T. Ergen and S. S. Kozat, "Efficient online learning algorithms based on lstm neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3772–3783, 2018.

[10] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction." in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 3428–3434.

[11] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[12] J. Lee, J. Yun, S. Hwang, and E. Yang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[13] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *35th International Conference on Machine Learning*, 2018, pp. 4548–4557.

[14] H. Liu, Y. Du, and Z. Wu, "Aem: Attentional ensemble model for personalized classifier weight learning," *Pattern Recognition*, vol. 96, p. 106976, 2019.

[15] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Proceedings of the Conference on Learning Theory*, 2016, pp. 907–940.

[16] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 139–148.

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

[18] Y. H. Jung, J. Goetz, and A. Tewari, "Online multiclass boosting," in *Thirty-First conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[19] Z. Wang, Z. Kong, S. Changra, H. Tao, and L. Khan, "Robust high dimensional stream classification with novel class detection," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1418–1429.

[20] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[21] B. Zhao, X. Li, X. Lu *et al.*, "Video captioning with tube features." in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 1177–1183.

[22] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine learning*, vol. 90, no. 3, pp. 317–346, 2013.

[23] M. M. Ferdaus, M. Pratama, S. Anavatti, and M. A. Garratt, "Palm: An incremental construction of hyperplanes for data stream regression," *IEEE Transactions on Fuzzy Systems (in press)*, 2019.

[24] M. Das, M. Pratama, A. Ashfahani, and S. Samanta, "FERNN: A fast and evolving recurrent neural network model for streaming data classification," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019.

[25] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[26] F. Guillet and H. J. Hamilton, *Quality Measures in Data Mining*. Springer, 2007, vol. 43.

[27] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015.

[28] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 2014.

[29] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[30] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 377–382.

[31] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 6467–6476.

[32] M. Pratama, E. Dimla, T. Tjahjowidodo, W. Pedrycz, and E. Lughofer, "Online tool condition monitoring based on parsimonious ensemble+," *IEEE Transactions on Cybernetics*, 2018.

[1] https://github.com/MUSE-RNN/Share