Journal of Control, Automation and Electrical Systems (2022) 33:1457–1465
https://doi.org/10.1007/s40313-021-00882-y

An Incremental Learning Approach Using Long Short-Term Memory Neural Networks

Álvaro C. Lemos Neto[1] · **Rodrigo A. Coelho**[1] · **Cristiano L. de Castro**[1]

Received: 20 April 2021 / Revised: 21 July 2021 / Accepted: 13 December 2021 / Published online: 10 April 2022

Abstract
Due to Big Data and the Internet of Things, machine learning algorithms targeted specifically to model evolving data streams have gained attention from both academia and industry. Although most of the proposed solutions in the literature have reported being successful in learning from non-stationary streaming settings, their complexity and the need for extra resources may be a constraint for their deployment in real applications. Aiming at less complexity without losing performance, this article proposes an incremental variant of the original LSTM neural networks with minor changes, that can tackle evolving data stream problems such as concept drift and the elasticity–plasticity dilemma without neither needing a dedicated drift detector nor a memory management system. Results achieved from benchmark datasets have shown that the proposed method is competitive in comparison with other incremental methods from the literature.

Keywords Machine learning · Incremental learning · Neural networks · Long short-term memory

1 Introduction

In recent years, Big Data changed from just a *buzzword* to becoming a real problem for many companies. The increase in processing and memory capabilities of computers, more sensors generating data, and the rise of the Internet of Things gave businesses the opportunity to generate more insights by understanding their products and clients better, but also brought a burden to IT departments. More data require more storage, more processing power, and the need for techniques to process this data in a distributed manner because in many cases they do not fit into a single machine (De Francisci Morales et al., 2016; Mehta, 2017; Žliobaitė et al., 2016).

This problem reflects in the machine learning (ML) field as well. Usually, ML models are designed to handle finite datasets so that the learning process involves solving complex optimization problems, such as support vector machines (Cortes & Vapnik, 1995). In addition, most ML models are trained with chunks of data that are supposed to be stationary, which in the real world it is rarely the case. Instead, information is always coming in an infinite stream of data that may change over time, so the ML researchers should take that into account (Ditzler et al., 2015).

This paper tackles the problem of supervised learning from streaming data by proposing a novel Incremental Learning technique based on the long short-term memory (LSTM) neural networks (Hochreiter & Schmidhuber, 1997). The main idea is to use the LSTM's intrinsic characteristics for handling concept drifts and learning new concepts (of course, keeping the old relevant ones) in an automatic manner using minimal memory resources.

Our approach was tested on benchmark datasets from the literature and compared with state-of-the-art online learning methods. In terms of overall accuracy through time, the proposed method achieved competitive performance, being more effective for datasets having some dependency among samples.

The rest of this paper is organized as follows: Sect. 2 brings the theory behind the incremental learning problem and the

An early version of this paper was presented at XXIII Congresso Brasileiro de Automática (CBA 2020).

✉ Álvaro C. Lemos Neto
alvarolemos@ufmg.br

Rodrigo A. Coelho
toluenotnt@gmail.com

Cristiano L. de Castro
crislcastro@ufmg.br

[1] Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-901, Brazil

LSTM neural networks; also presented is the state-of-the-art of online learning solutions. Section 3 describes the proposed approach; In Sect. 4, the experimental methodology is presented; Sect. 5 shows the findings of the comparative analysis of online methods. Final discussions and conclusions are presented in Sect. 6.

2 Theoretical Background

2.1 Incremental Learning and the Stability–Plasticity Dilemma

An ML supervised model can be trained in two distinguished learning modes: *offline learning* and *incremental* or *online learning*. In the former, the whole dataset is available at the time of training, whereas in the latter, the model process data as they come in a real-time stream that may be infinite. According to Gama et al. (2014), in this online scenario two main issues must be considered:

1. If the dataset is infinite, the learning algorithm should have infinite memory to accommodate it, which is unreal;
2. Even if the algorithm was able to keep a long history of past data, the data distribution may change as time passes by, which would make past data stale regarding the current data distribution.

Thus an incremental learning algorithm is subject to a trade-off where it has to keep past information so the model does not learn *outliers*, but it cannot keep too much information, because the system has memory constraints and it also needs to be able to learn new concepts (Gepperth & Hammer, 2016; Hoi et al., 2018).

Furthermore, when dealing with non-stationary data streams, a change in the relation between the input data and the target variable $P(\mathbf{x}, y)$ (the joint probability function) can occur at any time. This event, known in the literature as a *concept drift* (Gepperth & Hammer, 2016), represents a major concern for ML applications that need to process data and make decisions in very short periods of time. Models that operate in this online scenario require mechanisms for adapting to the evolving streaming of data, otherwise their performance tends to degrade. How a model adapts to a *concept drift* is also a challenge: if it reacts too quickly, old information is lost; if it waits too long, concept drifts are not caught at all (Mermillod et al., 2013).

Many approaches have been proposed to address the aforementioned challenges, which can easily be divided into two major categories: *active approaches*, which explicitly detect concept drifts, and *passive approaches*, those that continuously adapt themselves without explicit awareness of occurring drifts (Losing et al., 2016).

2.1.1 State-of-the-Art of Incremental Learning

Most methods of the *active approach* are agnostic to the drift detection mechanism so that they consider drift detectors to be independent of the learning model. This is the case of many state-of-the-art methods, like the *Oza Bagging ADWIN Classifier*, which is a mixture of the Online Bagging algorithm (Oza, 2005) with the ADaptive sliding Window (ADWIN) (Bifet & Gavalda, 2007) drift detection mechanism. Another method that follows the same structure is the *Adaptive Random Forests* (ARF) (Gomes et al., 2017b), which is an incremental version of the original Random Forest algorithm (Breiman, 2001). For every incoming sample from a data stream, ARF runs a drift detection mechanism for each tree of the ensemble and, depending on the value obtained, it either warns a possible drift (which triggers the training of a background tree) or detects it; in which case the trained background tree is used to replace the existing one. A similar recent work is the *Adaptive XGBoost* (Montiel et al., 2020), which adapts the *XGBoost* algorithm (Chen & Guestrin, 2016) for evolving data streams. Adaptive XGBoost and ARF were reported to achieve good results when combined with the ADWIN drift detector.

For the *passive approach*, *Self-adjusting Memory k Nearest Neighbors* (SAM-kNN) (Losing et al., 2016) was reported to show good performance over data stream classification problems. SAM-kNN is an ensemble of two models, one trained with the most recent samples from the stream, called the *short-term memory*, and another one trained with past samples, called *long-term memory*. The drift detection occurs passively through the training of the short-term memory model with different subsets of a window of recent samples; this process is analogous to a hyperparameter tuning taking place for every incoming sample. For a more detailed review on ensemble-based incremental learning methods, one can recommend the studies of Gomes et al. (2017a) and Krawczyk et al. (2017).

In the scope of artificial neural networks (ANNs), online extensions of the ELM (Extreme Learning Machines) topology have been proposed (Li et al., 2019; Shao & Er, 2016). The forgetting parameters extreme learning machine (FP-ELM) method has an incremental training with regularization (L2-norm) and employs a forgetting factor on the subset of observations that were learned at the previous time instant so that this subset can be reused along with the current sample. A more recent study on the theme of recurrent nets achieved promising results by introducing the covariance of the present and one-time step past input vectors into the gating structure of LSTMs and GRUs (gated recurrent unit) (Mirza et al., 2020).

One characteristic that all the aforementioned methods have in common is that they are all structured in a complex way around classic supervised learning batch algorithms.

On the other hand, LSTM neural networks seem to fit perfectly for streaming scenarios: its *hidden state* and, more importantly, the *memory cell* was built to deal exactly with *stability–plasticity dilemma* and avoid *catastrophic forgetting* (Gepperth & Hammer, 2016). By applying minor changes in its architecture, we propose in this paper an incremental learning algorithm that does not need neither explicit drift detection, nor memory management. In addition, our strategy does not modify the weight update rule of recurrent networks, as reported in other past studies (Marschall et al., 2020; Mirza et al., 2020; Xu et al., 2020).

2.2 Long Short-Term Memory Neural Networks

Long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) is a type of *recurrent neural network* (RNN) and, as such, it is suited to learn sequence problems by tracking dependencies along the time. This is accomplished by feeding back the layer hidden state, from a time step t, as an input to the next step $t+1$, along with the input for that step. The weights are updated with the backpropagation through time optimization algorithm (Goodfellow et al., 2016).

The problem with regular RNNs arises when they are used to learn sequence problems that have long-term dependencies. This happens due to a phenomenon known as *vanishing gradients* (Pascanu et al., 2013), in which the network's weight updates at the beginning of the sequence tend to lose their contribution due to small gradient values. To overcome this, the LSTM introduces the *memory cell* in its architecture, so it not only tracks dependencies at time step t with the hidden state h_{t-1} from the previous step $t-1$, but also has a *memory cell* c_{t-1}, which does not vanish during the learning process.

The computations performed by an LSTM cell are listed in Eq. 1, which are better understood along with Fig. 1.

$$
\begin{aligned}
\Gamma_{f,t} &= \sigma(\boldsymbol{W_f}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_f) \\
\Gamma_{u,t} &= \sigma(\boldsymbol{W_u}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_u) \\
\widetilde{\boldsymbol{c}}_{\boldsymbol{t}} &= \tanh(\boldsymbol{W_c}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_c) \\
\boldsymbol{c_t} &= \Gamma_{f,t} \odot \boldsymbol{c_{t-1}} + \Gamma_{u,t} \odot \widetilde{\boldsymbol{c}}_{\boldsymbol{t}} \\
\Gamma_{o,t} &= \sigma(\boldsymbol{W_o}[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + b_o) \\
\boldsymbol{h_t} &= \Gamma_{o,t} \odot \tanh(\boldsymbol{c_t}) \\
\hat{\boldsymbol{y}}_{\boldsymbol{t}} &= softmax(\boldsymbol{W_y h_t} + b_y).
\end{aligned}
\tag{1}
$$

In (1), the hidden state $\boldsymbol{h_{t-1}}$ from the previous time step $t-1$ is concatenated with the current step t input, $\boldsymbol{x_t}$, which is represented by $[\boldsymbol{h_{t-1}}, \boldsymbol{x_t}]$. It is weighted by $\boldsymbol{W_f}$, added with b_f and then applied to a sigmoid function $\sigma(\cdot)$, resulting in the *forgetting gate* $\Gamma_{f,t}$, which is ranged between 0 and 1, as shown in (2). In (3), the *forgetting gate* $\Gamma_{f,t}$ weights the previous step's cell memory, $\boldsymbol{c_{t-1}}$. Its role is to forget or not this cell memory, depending if its value is closer to 0 than to 1. In (4), the current time step's cell memory's contribution $\widetilde{\boldsymbol{c}}_{\boldsymbol{t}}$ is computed. The amount that it will contribute to the next step's cell memory is calculated in (6) and will depend on the *update gate* $\Gamma_{u,t}$, which is computed similarly to the *forgetting gate*, but with its own weights $\boldsymbol{W_u}$ and bias b_u, as shown in (5).

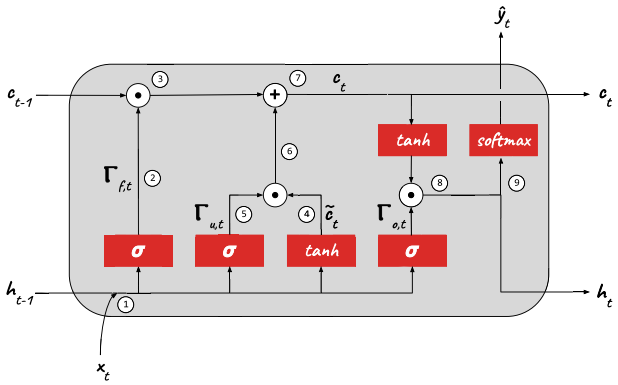


Fig. 1 An LSTM cell.. Adapted from Olah (2015)

The network's memory cell $\boldsymbol{c_t}$ is updated in (7) based on the previous and current steps' memory cell contributions ($\Gamma_{f,t} \cdot \boldsymbol{c_{t-1}}$ and $\Gamma_{u,t} \cdot \widetilde{\boldsymbol{c}}_{\boldsymbol{t}}$, respectively). Finally, it is weighted by the *output gate* $\Gamma_{o,t}$ in (8), which provides the *hidden state* $\boldsymbol{h_t}$, which is weighted by $\boldsymbol{W_y}$, followed by an element-wise addition of b_y, then fed to a *softmax* activation[1] in (9), providing the prediction $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$ of the current step.

An LSTM layer is composed of a sequence of single cells, as the one illustrated in Fig. 1. Also, LSTM layers can be connected, by feeding the hidden state as the input of the next layer, as shown in Fig. 2. It is important to note that both initial hidden states ($\boldsymbol{h_0^0}$ and $\boldsymbol{h_0^1}$) and initial memory cells ($\boldsymbol{c_0^0}$ and $\boldsymbol{c_0^1}$) are recommended to be assigned at random. In other words, the context carried by both memory cells and hidden states is exclusive for a single sequence. This is the property that the proposed method focuses on, as described next.

3 Incremental LSTM

Before going into the details of the learning algorithm, it is important to define how the proposed method prepares the received data into sequences suited for LSTMs. As it will

[1] This is the case for a Neural Network designed to address a multiclass classification task, otherwise, other activation function should be used. Also, if this is not the output layer, the hidden state is passed to the next layer, i.e., without the need of an activation function, having $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$ replaced by $\boldsymbol{h_t}$ in (9).

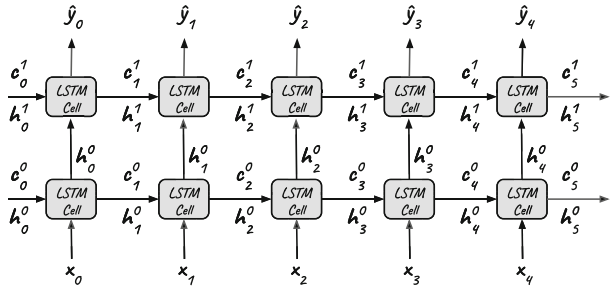


Fig. 2 A neural network composed of two LSTM layers design for sequences with five time steps

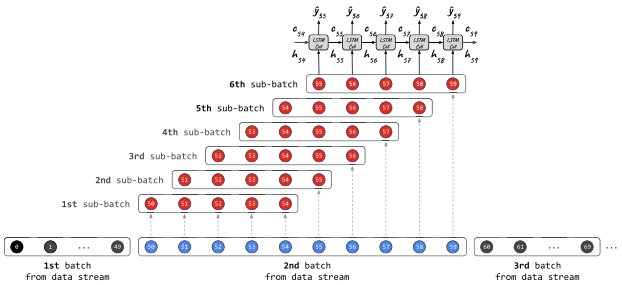


Fig. 3 Data stream sub-batching

be seen in Sect. 4, batches with a specific number of samples are accumulated in the data stream and then provided to the model, so that it can predict, and then train (*prequential* schema). On the other hand, LSTMs expect a batch of sequences as input for training.

To transform a batch of samples into a batch of sequences, the batch is ordered by the time they are received. Then, it is broken down into multiple sub-batches in a sliding window manner. Figure 3 illustrates this procedure. In blue, we have the current batch, with length 10, which is split into sub-batches of length 5 (in red), each of which is presented to the model as a sample sequence. Hence, according to this example, a batch of ten samples became a batch of six sample sub-batches.

After the sub-batching procedure, the incremental learning process here proposed relies on using LSTM's components to determine when to forget past concepts and when to learn new ones. As mentioned earlier, the hidden state and memory cell are reset for each sample sequence received by the model. So, for the batch B_t, we have $\boldsymbol{c_0} = \boldsymbol{0}$ and $\boldsymbol{s_0} = \boldsymbol{0}$. The trick here is to keep the last value of both $\boldsymbol{c}$ e $\boldsymbol{t}$, so that at the next batch, they are provided, instead of vectors of zeros. By doing so, the following problems are solved:

1. No need to keep a history of past sample that may violate the system's memory constraints, since only the LSTM cell's weights ($\boldsymbol{W_f}$, $\boldsymbol{W_u}$, $\boldsymbol{W_c}$, $\boldsymbol{W_o}$ and $\boldsymbol{W_y}$), biases (b_f, b_u, b_c, b_o and b_y) and the last values (meaning, at time t) of the cell and hidden state (c_t and a_t) should be kept in memory;
2. No need to implement a specific mechanism to detect a *concept drift*, since they should be captured automatically by the *memory cell*.

This approach can be applied to distinct supervised machine learning problems, such as *regression*, *classification* and *forecasting*.

4 Experimental Methodology

Experiments were conducted to evaluate the effectiveness of the proposed algorithm (hereafter referred to as ILSTM) in evolving data stream scenarios. ILSTM was compared with well-known methods in the literature: *Oza Bagging ADWIN* (Oza, 2005), *Adaptive Random Forest* (Gomes et al., 2017b) and *SAM-kNN* (Losing et al., 2016), using Prequential Evaluation (Gama et al., 2004). In this experimental setting, data comes in batches over time. The initial batch, usually larger than the subsequent ones, is used to pre-train the model. In the following ones, the batch data is first presented to the model (without labels) which makes predictions. This is illustrated in Fig. 3, where the first batch contains 50 samples, whereas the second and third batches have 10 samples each. After the model makes predictions for the unlabeled data, a prequential error is calculated using a metric best suited for the given experiment. As will be shown next in Sect. 4.1, tests involved only classification datasets with reasonable class balance and hence, the metrics *accuracy*, *precision* and *recall* were chosen for performance evaluation of the methods.

ILSTM had the same hyperparameter setting for all experiments: 3 LSTM layers, with 150, 200 and 50 units, respectively, all of them using the *hyperbolic tangent* activation function; 300 for the initial batch's number of epochs and 60 for the subsequent batches; Adam was chosen as the optimizer with $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $epsilon = 1e^{-7}$; the loss function was set to binary cross-entropy. The other incremental learning methods were configured with their default hyperparameters as suggested in the *Scikit-multiflow* framework.[2] All experiments were repeated multiple times to increase statistical significance.

[2] Available online on: https://www.scikit-multiflow.github.io.

Table 1 Overall performance achieved by all tested methods

Dataset	i.i.d.	Type	Model	Accuracy (%)	Precision (%)	Recall (%)
Chess	No	Real	ILSTM	**78.93 ± 4.06**	**81.76 ± 3.58**	84.26 ± 3.41
			SAM-KNN	74.50 ± 2.96	77.50 ± 5.72	82.06 ± 4.46
			ARF	77.67 ± 3.17	78.46 ± 2.87	**86.93 ± 4.47**
			OB-ADWIN	76.67 ± 4.19	77.69 ± 4.55	85.97 ± 4.68
Electricity	No	Real	ILSTM	73.34 ± 11.94	77.75 ± 13.43	80.35 ± 15.65
			SAM-KNN	64.75 ± 9.27	71.19 ± 12.85	72.29 ± 17.08
			ARF	**77.97 ± 10.39**	**82.57 ± 12.18**	**83.62 ± 16.76**
			OB-ADWIN	69.58 ± 7.78	73.67 ± 11.26	77.88 ± 11.53
Weather	No	Real	ILSTM	**75.52 ± 3.02**	69.69 ± 7.34	**62.64 ± 8.49**
			SAM-KNN	73.43 ± 5.62	72.20 ± 10.73	49.88 ± 12.67
			ARF	73.90 ± 3.84	**73.57 ± 11.11**	49.77 ± 10.27
			OB-ADWIN	75.32 ± 3.47	71.35 ± 7.36	57.39 ± 8.72
Sine 1	No	Synthetic	ILSTM	**97.41 ± 8.64**	**97.95 ± 9.63**	96.68 ± 9.51
			SAM-KNN	96.80 ± 13.75	96.91 ± 13.65	96.74 ± 13.63
			ARF	96.97 ± 8.29	97.00 ± 8.46	**96.90 ± 8.67**
			OB-ADWIN	92.90 ± 20.75	92.78 ± 20.69	93.06 ± 20.78
Sine 2	No	Synthetic	ILSTM	**94.36 ± 8.26**	**94.49 ± 8.33**	**94.33 ± 8.16**
			SAM-KNN	86.01 ± 17.56	86.18 ± 17.78	85.84 ± 17.93
			ARF	89.44 ± 10.07	89.56 ± 10.18	89.36 ± 10.62
			OB-ADWIN	86.63 ± 13.73	86.78 ± 13.81	86.61 ± 13.64
Spam	Yes	Real	ILSTM	86.84 ± 18.30	77.56 ± 31.34	**81.88 ± 28.39**
			SAM-KNN	**91.42 ± 8.29**	**83.11 ± 29.32**	69.00 ± 31.61
			ARF	90.15 ± 10.31	82.08 ± 25.02	71.73 ± 27.68
			OB-ADWIN	89.89 ± 9.16	81.31 ± 26.02	67.28 ± 29.25
SEA	Yes	Synthetic	ILSTM	84.19 ± 4.11	85.48 ± 4.10	90.00 ± 4.28
			SAM-KNN	87.14 ± 2.74	87.14 ± 3.36	93.12 ± 3.12
			ARF	**87.39 ± 3.61**	**87.35 ± 4.23**	**93.45 ± 4.09**
			OB-ADWIN	86.44 ± 2.69	86.98 ± 3.31	92.00 ± 3.38
Stagger	Yes	Synthetic	ILSTM	98.12 ± 7.55	97.79 ± 10.19	97.80 ± 10.99
			SAM-KNN	99.08 ± 5.90	99.16 ± 6.09	98.60 ± 9.78
			ARF	**99.64 ± 3.13**	**99.71 ± 2.53**	**99.40 ± 6.02**
			OB-ADWIN	97.98 ± 9.15	98.35 ± 8.00	97.31 ± 13.07

Bold values indicate the metrics that obtained the higher average value for each group of: "Dataset", "i.i.d.", "Type"

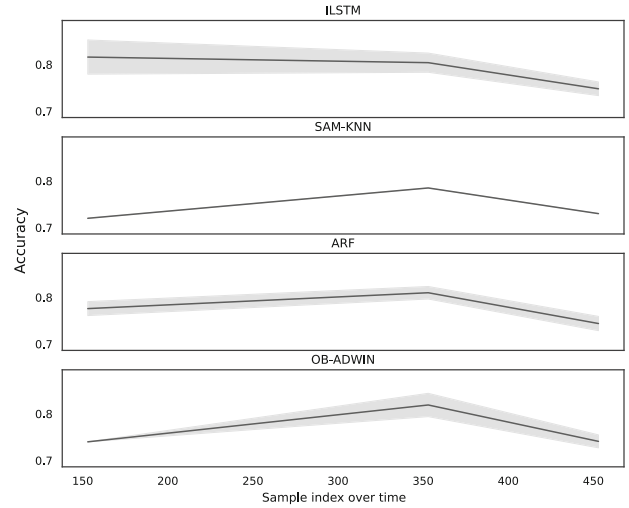
4.1 Datasets

Experiments were performed over four synthetic and four real datasets. Stream generators[3] were used to create the synthetic datasets and the real datasets are available at the GitHub repository.[4]

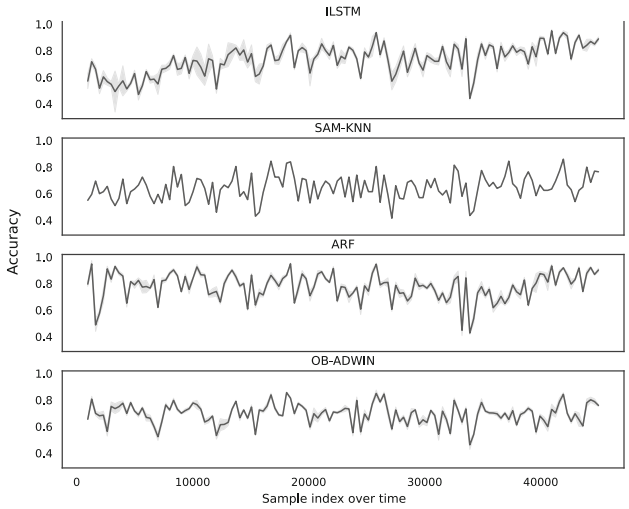
- **Chess**: consists of real game records of one player over a period from 2007 December to 2010 March. Since a chess player learns from its previous games, the dataset is considered to be non-*independent and identically distributed* (non-i.i.d.). This dataset has 8 attributes, 2 classes and 503 instances, which were divided into a first batch of 153 samples and subsequent batches of 50 samples. The sub-batch size was set to 30 samples for ILSTM.
- **Electricity**: is formed from real data collected in the Australian electricity market. In this market, prices fluctuate according to the demand, and data is sampled every 30 min. Since electricity demand is seasonal, this is a non-i.i.d. dataset and such characteristic was taken into account in the experimental setup: sub-batch size of 48 samples (1 day worth of data), batches of 336 samples (1 week of data), except for the first batch with 960 sam-

[3] Available at: https://scikit-multiflow.github.io/.

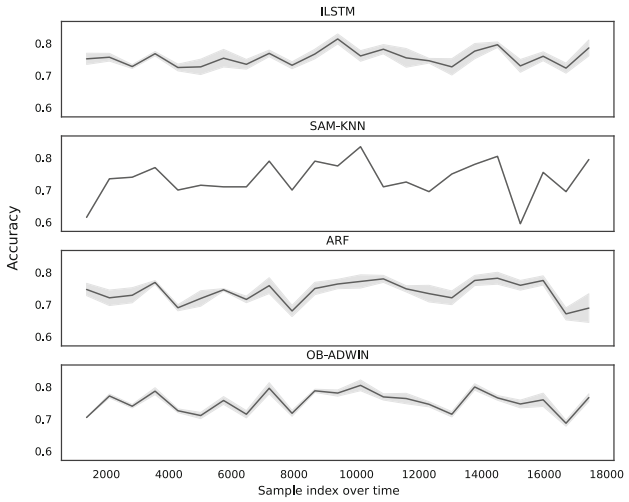
[4] Available at: https://github.com/ogozuacik/concept-drift-datasets-scikit-multiflow.



(a) *Chess* dataset

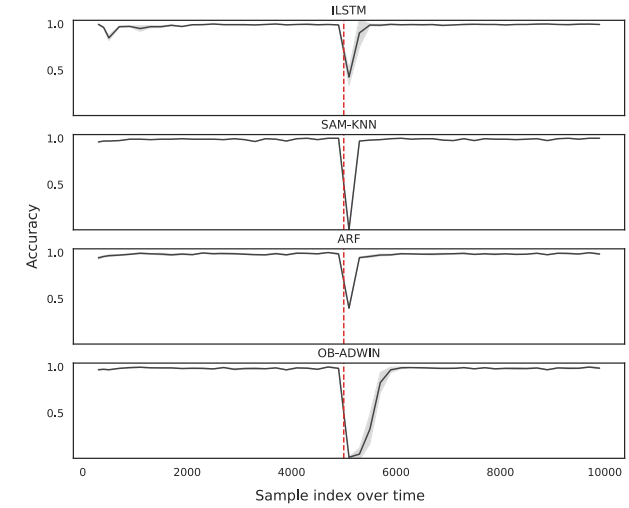


(b) *Electricity* dataset

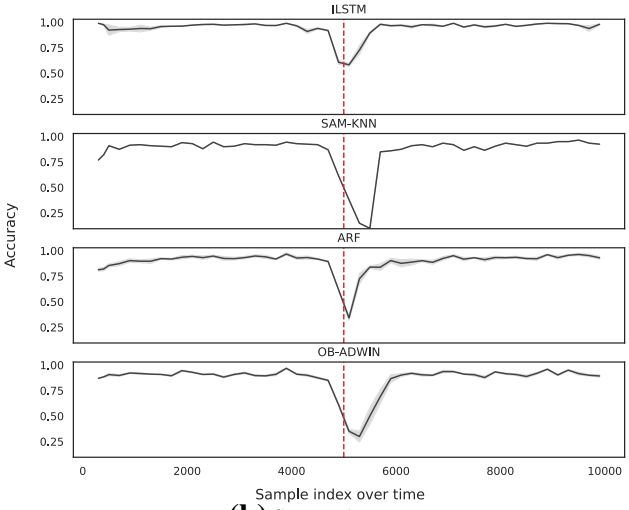


(c) *Weather* dataset

Fig. 4 Accuracy evolution over time for **real** and **non-i.i.d.** datasets



(a) *Sine 1* dataset

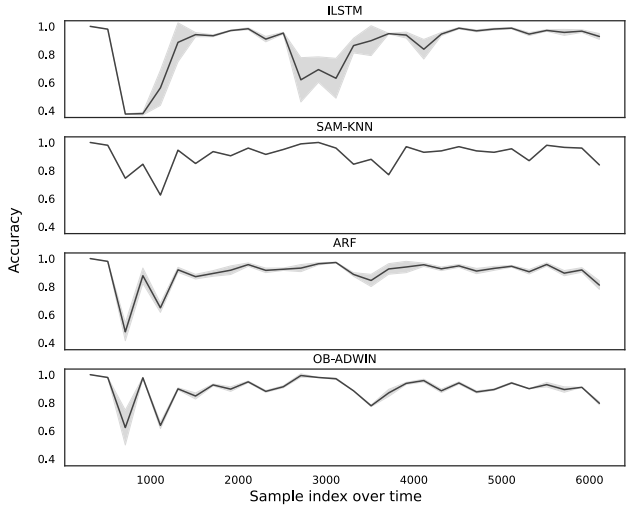


(b) *Sine 2* dataset

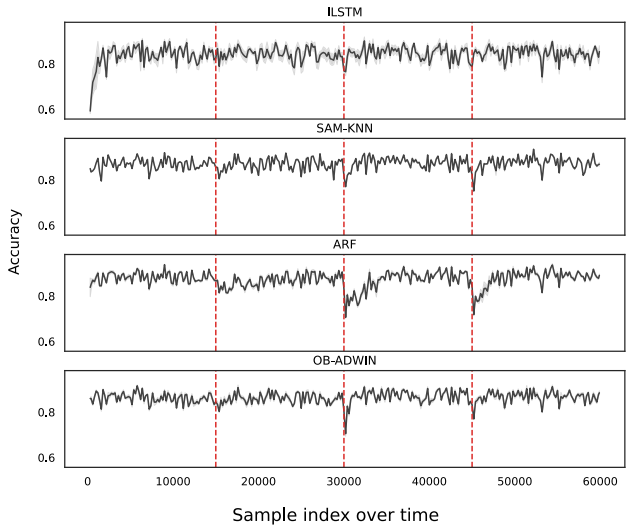
Fig. 5 Accuracy evolution over time for **synthetic** and **non-i.i.d.** datasets

ples, which adds up to all the 45,312 instances of the dataset, that has 8 dimensions and 2 classes.

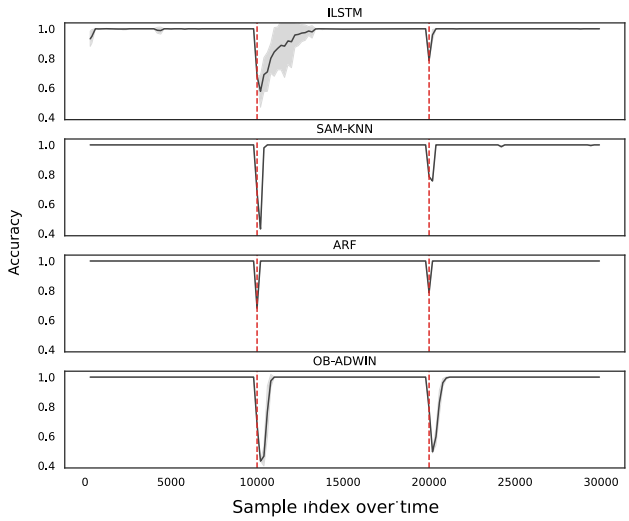
– **Weather**: is formed by 50 years of real daily measurements with several meteorological information, which characterizes this dataset as non-i.i.d. It has 2 classes, 8 attributes and 18,159 instances, divided into a first batch of 1369 samples and subsequent batches of 730 samples (2. years of data). For ILSTM, the sub-batch size was set to 365 samples (1 year of data).
– **Sine 1**: is a synthetic and non-i.i.d. dataset with a total of 10,000 instances, 2 dimensions, two different concepts with 5000 instances each and an abrupt change of concept. In the first concept, all points below the curve $y = \sin(x)$ are classified as positive and after the drift, the classification is reversed. The experimental setup was: 300 samples for the initial batch, 100 samples for the



(a) *Spam* dataset



(b) *SEA* dataset



(c) *Stagger* dataset

Fig. 6 Accuracy evolution over time for **i.i.d.** datasets

subsequent ones and 60 samples for the ILSTM's sub-batches.

- **Sine 2**: is a synthetic and non-i.i.d. dataset with a total of 10,000 instances of 4 dimensions so that two of them represent only noise; it also has two different concepts with 5000 instances each and gradual change. The gradual change of concept was carried out by a sigmoid function $f(t) = 1/(1 + e^{-4(t-p)/w})$, where $p = 5000$ is the position in which the change happens and $w = 500$ is the width of the transition. For this dataset, we have followed the same experimental setup of Sine 1.
- **Spam**: is a real dataset containing a ham/spam collection of legitimate messages with 6213 instances and 499 attributes. Since the messages are not correlated, this is an i.i.d. dataset. The experimental setup was: 313 samples for the initial batch, 100 samples for the subsequent ones and 60 samples for the ILSTM's sub-batches.
- **SEA**: is a synthetic and i.i.d. dataset with 60,000 instances, 3 dimensions and 2 classes. The dataset presents four different concepts with 15,000 instances each; 10% of the data are noisy. The experimental setup was the same for the Sine 1 dataset.
- **Stagger**: is a synthetic and i.i.d. dataset with 30,000 instances, 3 attributes and two abrupt drifts taking place in the transition regions of three different concepts with 10,000 instances each. It has Boolean values, generated by three different functions, where each concept represents one of these functions. The experimental setup was the same for the Sine 1 dataset.

5 Results

Overall performance for all tested methods is presented in Table 1, whereas a more detailed behavior of the models' accuracy over time can be seen in the individual plots of Figs. 4, 5 and 6. In such plots, we have a thick gray line representing the average accuracy for each training batch (across all experiments), in lighter gray, the standard deviation and in dashed red, the timestamps where a concept drift occurred for the synthetic datasets (Figs. 5a, b, 6b, c). As pointed out earlier, there is no information regarding drifts for the real datasets (Figs. 4a–c, 6a).

As can be observed in Table 1, the proposed method achieved competitive performance in comparison with other incremental methods from the literature. The results also suggest that ILSTM can be more effective for datasets having some dependency among samples (non-i.i.d.), which is the case for most real applications of evolving data streams.

The individual plots for SEA and SPAM datasets (Fig. 6a, b) show parts of ILSTM's accuracy graphs having larger variance (lighter gray area) when compared to other methods,

what suggests that the proposed method could benefit from a better hyperparameter tuning in such cases. It is also worth noting the ILSTM behavior during the occurrence of concept drifts in the synthetic datasets SINE 1, SINE 2 and SEA. In most cases, ILSTM had the smallest drops of accuracy after the corresponding time instants of concept drifts (dashed red lines in Figs. 5a, b, 6b).

6 Conclusion

The proposed method achieved promising results, having simplicity as its main strength. In contrast to other incremental learning algorithms, it does not need a complex memory structure, nor explicit mechanisms for detecting drifts or forgetfulness. Instead, it uses the ability to deal with the stability–plasticity dilemma of LSTM networks, through a simple but effective adaptation.

Our findings pointed out that incremental LSTM may be more suitable for datasets having some dependence among the samples (non-i.i.d.), which is reasonable for most real-world problems involving streaming data. Our results also suggest that sequence model-based incremental learning approaches should be further explored since sequence models, such as LSTMs and GRUs (Chung et al., 2014), already have inherent characteristics for handling problems with spatiotemporal context and, such context should be taken into account when learning from evolving data streams.

As future efforts, the authors believe that experimenting with different architectural setups, as well as other recurrent neural network variants, performance could be improved. Moreover, we intend to test the proposed method in different learning tasks, such as regression and forecasting.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (pp. 443–448). SIAM.

Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5–32.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794).

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273–297.

De Francisci Morales, G., Bifet, A., Khan, L., Gama, J., & Fan, W. (2016). IoT big data stream mining. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 2119–2120).

Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine, 10*(4), 12–25.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence* (pp. 286–295). Springer.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR), 46*(4), 1–37.

Gepperth, A., & Hammer, B. (2016). Incremental learning algorithms and applications. In *European symposium on artificial neural networks (ESANN)*.

Gomes, H. M., Barddal, J. P., Enembreck, F., & Bifet, A. (2017a). A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR), 50*(2), 1–36.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017b). Adaptive random forests for evolving data stream classification. *Machine Learning, 106*(9–10), 1469–1495.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge: MIT Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Hoi, S. C., Sahoo, D., Lu, J., & Zhao, P. (2018). Online learning: A comprehensive survey. arXiv preprint arXiv:1802.02871.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion, 37*, 132–156.

Li, L., Sun, R., Cai, S., Zhao, K., & Zhang, Q. (2019). A review of improved extreme learning machine methods for data stream classification. *Multimedia Tools and Applications, 78*(23), 33375–33400.

Losing, V., Hammer, B., & Wersing, H. (2016). KNN classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th international conference on data mining (ICDM)* (pp. 291–300). IEEE.

Marschall, O., Cho, K., & Savin, C. (2020). A unified framework of online learning algorithms for training recurrent neural networks. *Journal of Machine Learning Research, 21*(135), 1–34.

Mehta, S., et al. (2017). Concept drift in streaming data classification: Algorithms, platforms and issues. *Procedia Computer Science, 122*, 804–811.

Mermillod, M., Bugaiska, A., & Bonin, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology, 4*, 504.

Mirza, A. H., Kerpicci, M., & Kozat, S. S. (2020). Efficient online learning with improved LSTM neural networks. *Digital Signal Processing, 102*, 102742.

Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., & Bifet, A. (2020). Adaptive xgboost for evolving data streams. arXiv preprint arXiv:2005.07353.

Olah, C. (2015). Understanding LSTM networks. Understanding LSTM Networks-Colah's Blog. Github

Oza, N. C. (2005). Online bagging and boosting. In *2005 IEEE international conference on systems, man and cybernetics* (Vol. 3, pp. 2340–2345). IEEE.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Shao, Z., & Er, M. J. (2016). An online sequential learning algorithm for regularized extreme learning machine. *Neurocomputing, 173*, 778–788.

Xu, R., Cheng, Y., Liu, Z., Xie, Y., & Yang, Y. (2020). Improved long short-term memory based anomaly detection with concept drift adaptive method for supporting IoT services. *Future Generation Computer Systems, 112*, 228–242.

Žliobaitė, I., Pechenizkiy, M., & Gama, J. (2016). An overview of concept drift applications. In *Big data analysis: New algorithms for a new society* (pp. 91–114). Springer.