Neural Computing and Applications
https://doi.org/10.1007/s00521-019-04288-5

BRAIN INSPIRED COMPUTING &MACHINE LEARNING APPLIED RESEARCH-BISMLARE

Identifying data streams anomalies by evolving spiking restricted Boltzmann machines

Lining Xing[1] · Konstantinos Demertzis[2] · Jinghui Yang[3]

Received: 1 February 2019 / Accepted: 31 May 2019

Abstract
Data streams are characterized by high volatility, and they drastically change in an unpredictable way over time. In the typical case, newer data are the most important, as the concept of aging is based on their timing. These flows require real-time processing in order to extract meaningful information that will allow for essential and targeted responses to changing circumstances. Knowledge mining is a real-time process performed on a subset of the data streams, which contains a small but recent part of the observations. Timely security requirements call for further quest of optimal approaches, capable of improving the reliability and the accuracy of the employed classifiers. This research introduces a real-time evolving spiking restricted Boltzmann machine approach, for efficient anomaly detection in data streams. Testing has proved that the proposed algorithm maximizes the classification accuracy and at the same time minimizes the computational resources requirements. A comparative analysis has shown that it outperforms other data flow analysis algorithms.

Keywords Big Data · Data streams analysis · Evolving spiking neural networks · Restricted Boltzmann machines · Deep learning · Real-time anomaly detection

1 Introduction

Information generated by complex environments such as the Internet of Things (IOT) ecosystem, has increased exponentially [1]. The result is the inefficient management and storage of the total volume of generated information. This requires the adoption of complex real-time data mining and analysis architectures [2]. These architectures should incorporate specialized real-time processing algorithms that dynamically adapt to new standards or data, or to scaled data production as a function of time [3].

Although mining data streams is an emerging area, it poses enormous challenges to the data mining community. High transmission speed, change of data distribution and high volume raise the following issues that should be addressed [4]:

- *High velocity* Online data streams arrive at a very high speed. Thus, it has become almost practically infeasible to scan all of them. This is also the case for the off-line ones.
- *Concept drift* Frequent patterns keep changing, as data streams are time varying in nature. During the mining process, as new incoming data are added to the existing ones, some frequent patterns may change their status to become infrequent and vice versa. This issue is known as the concept drift problem.
- *Unbounded size* Data streams are unbounded in size. Their size is unknown to the user in advance unlike the static data.

✉ Konstantinos Demertzis
kdemertz@fmenr.duth.gr

Lining Xing
xinglining@gmail.com

Jinghui Yang
jhyang@sspu.edu.cn

1 School of Logistics and Transportation, Central South University of Forestry and Technology, Changsha 410004, People's Republic of China

2 Department of Civil Engineering, School of Engineering, Democritus University of Thrace, University Campus, Kimmeria, Xanthi, Greece

3 College of Engineering, Shanghai Polytechnic University, Shanghai 201209, People's Republic of China

- *Enormous space requirement* Huge amount of data are generated both in online and off-line applications. There might not be enough space to store the data stream before processing.
- *Unsteady analysis results* High-speed as well as varying data distribution may affect the analyzed results, due to which the mining outcome may be declined. In order to cope with this, data streams mining must be an incremental process.

Anomaly detection [5] over multiple data streams is initially determined by the real-time observation of a single (multivariable) time-series frequency, which constitutes the systems' quantitative performance parameters.

A data stream s_i coming from a system of sensors consists of numerical values determined by a function of time (t), where $t \in [0, +\propto]$. The values of (t) must be in the closed interval [0, 1]. If n sensor flows are synchronized periodically to report their values, the whole of the multivariable information at each time t is represented by the following frame vector (Eq. 1) [6]:

$$\Delta\Pi_t = (s_1(t), s_2(t), \ldots, s_n(t) \in R^n) \tag{1}$$

In practice, each flow forms a one-dimensional time series, while the frame vector flow (FVF) represents a multivariable time series. Event detection on data streams is intended to determine the values s_i (t), which represent abrupt changes within an FVF.

Particularly, each FVF of length n is converted to a binary vector of the same length, where each value represents a possible change in the corresponding sensor flux. Such deviations from the normal behavior are called events, and binary vectors are called event vectors. An event may be an observation that does not conform to an expected standard in the dataset (anomaly). Incidents may have been caused by a variety of reasons, like sensor failure or malfunction, or deviations and substantial changes that may affect the system's behavior, such as cyber-attacks. Therefore, a vector of events in time t is represented by Eq. 2 [7, 8]:

$$\Delta\Sigma_t = \left(e_1^t, e_2^t, \ldots, e_n^t\right) \in [0, 1] \tag{2}$$

Finally, $e_i^t = e_i(t)$ is the binary value which represents the occurrence of an abnormal flow behavior, and it is equal to (t) = 1 in time t.

The error is calculated at each iteration, as data characteristics can change drastically and in an unpredictable way changing the typical, normal behavior. An object that may be considered abnormal can then be included in the set of normal observations due to rapid developments in the data stream. Due to the fact that the data volume is unlimited, data mining is performed on a subset of the flow, called a sliding window, which obviously contains a small but recent percentage of the observations. The goal of the data flow processing algorithms is to minimize the cumulative error for all iterations, which can be calculated by Eq. 3 [7, 8]:

$$I_n[w] = \sum_{j=1}^{n} V\left(w, x_j, y_j\right) = \sum_{j=1}^{n} \left(x_j^{\mathrm{T}} w - y_j\right)^2 \tag{3}$$

where $x_j \in R^d, w \in R^d$ and $y_j \in R$. We consider that $i \times d$ is a data matrix and $i \times 1$ is a matrix with target values, after the arrival of the first i data points. Assuming that the covariance matrix $\Sigma_i = X^{\mathrm{T}}X$ is reversible, the optimal solution $f^*(x) = w^*, x$ is given by Eq. 4 [7, 8]:

$$\langle w^*, X\rangle = \left(X^T X\right)^{-1} X^T \Upsilon = \Sigma_i^{-1} \sum_{j=1}^{i} x_j y_j \tag{4}$$

First, the covariance table is calculated $\Sigma_i = \sum_{j=1}^{i} x_j x_j^{\mathrm{T}}$. The initial time complexity (CM) is calculated to be of $\mathrm{O}(id^2)$ order, but after we inverse the $X^{\mathrm{T}}X$ $(d \times d)$ table, it increases to $\mathrm{O}(d^3)$, while the rest of the required multiplications have an $\mathrm{O}(d^2)$CM. This produces an overall CM of $\mathrm{O}(id^2 + d^3)$ [7, 8].

It is conceivable that robust systems ensuring reliability and high accuracy rates without requiring high availability of resources are required, to safely approach problems stemming from knowledge mining processes. The above argument is further supported as follows: Let's suppose that the set of data points is equal to n, and after the arrival of each new data point i = 1, 2, …, n, recalculation of the solution is required. In this case, the total time complexity is equal to $\mathrm{O}(n^2d^2 + nd^3)$ [8].

Summarizing, we suggest that a successful model requires good preparation and methodological determination of the operating parameters related to the data processing algorithms. This can avoid long-term convergence, unwanted variations in precision that may be associated with frequent model updates and instability or loss of generalization, which may be due to corrupted and noisy data.

1.1 Innovation of the proposed approach

This research introduces the real-time e-SREBOM algorithm, which performs successful anomaly detection in data streams. This is a hybrid, sophisticated, innovative and highly effective anti-malware detection method.

To the best of our knowledge, the evolving spiking restricted Boltzmann machine algorithm is introduced to the literature for the first time, as a hybrid anomaly detection approach (combining two well-established algorithms). Our research team was inspired by the need to maximize the classification accuracy and to simultaneously minimize the computational resources requirements.

The structure of the rest of the paper is the following: Sect. 2 is the literature review, which briefly describes some of the most well-known machine learning approaches used in processing of data streams. Section 3 is an overall description of the methodology employed herein, whereas Sect. 4 presents the proposed e-SREBOM algorithm. Sections 5 and 6 present the datasets and the evaluation metrics, respectively, and Sect. 7 shows and illustrates the results and the last chapter discusses the conclusions.

2 Literature review

Data mining in nonstationary data streams is gaining more attention recently, especially in the context of Internet of Things and Big Data. To cope with evolving data streams, ensembles are often coupled with drift detectors. The leveraging bagging for evolving data streams [9] and the adaptive windowing are characteristic algorithms [10]. The diversity for dealing with drifts online ensemble learning [11] employs the early drift detection method [12] to detect concept drifting. Another approach to deal with concept drift while using an ensemble of classifiers is to constantly reset the low-performance classifiers [13, 14]. This reactive approach is useful to recover from gradual drifts, while methods based on drift detectors are more appropriate for rapidly recovering from abrupt drifts. Moreover, Yang et al. [15] present a novel method for concept drift detection, based on: (1) the development and continuous updating of online sequential extreme learning machines and (2) the quantification of how much the updated models are modified by the newly collected data. The results show the superiority of the proposed method with respect to alternative state-of-the-art concept drift detection methods. Furthermore, updating the prediction model when the concept drift has been detected is shown to allow improving the overall accuracy of the energy prediction model and, at the same time, minimizing the number of models updating. A major issue is the fact that the method is time-consuming and needs significant computational resources. Domingos et al. [16] have developed the varying fast decision tree algorithm which is constructed on Hoeffding trees. The split point is found by using Hoeffding bound, which satisfies the statistical measure. This algorithm also drops the nonpotential attributes. Hence, the cost of acquiring the confidence interval is sublinear in terms of confidence level and quadratic in terms of precision. Note that there are more efficient methods of estimating a confidence interval. Aggarwal introduced the idea of microclusters included in CluStream, following on-demand classification [17]. This technique exploits clustering results to classify data, using statistics of class distribution in each cluster. But, as the number of records increases, the performance of algorithm goes decreasing and time for execution increases. Therefore, for the same amount of data, micro-clustering will take quadratic amount of time. Peng and Zhang [18] proposed a model that categorizes concept drift in data streams based on the following two scenarios, namely the loose concept drifting and the rigorous one. They proposed solutions to handle each one of them, by using a weighted instance, plus a weighted classifier ensemble framework, such that the overall accuracy of the built classifier ensemble framework can reach the minimum value.

Losing et al. [19] proposed the self-adjusting memory (SAM) model for the *k*-nearest neighbor (k-NN) algorithm, since *k*-NN constitutes a proven classifier within the streaming setting. SAM-*k*NN can effectively deal with heterogeneous concept drift, i.e., different drift types and rates, using biologically inspired memory models and their coordination. Rani and Sumathy [20] used the *k*-NN algorithm to determine the optimal subset.

Some researchers have employed the primal estimated sub-gradient solver for support vector machines (SVM) algorithm, known as "Pegasos". Shalev-Shwartz et al. [21] described and analyzed a simple and effective stochastic sub-gradient descent algorithm for solving the optimization problem cast, by using SVM. Their algorithm was particularly well suited for large text classification problems, where authors demonstrated an order-of-magnitude speedup, over all previous SVM methods.

Random forests is currently one of the most popular ML algorithms in the nonstreaming (batch) setting. This preference is attributed to its high learning performance and to its low demands with respect to input preparation and hyper-parameter's tuning. The adaptive random forests (ARF) algorithm [22] is often employed for the classification of evolving data streams. ARF includes an effective resampling method and adaptive operators that can cope with different types of concept drifts without performing complex optimizations.

The e-SREBOM algorithm (proposed herein) optimizes the run-time performance of ARF by limiting the number of detectors, as maintenance of several detectors that often trigger simultaneously, would be a redundant consumption of resources.

3 Methodology

The hybrid architectural standardization and development of the proposed e-SREBOM data flow analysis algorithm is employed by an intelligent cybersecurity monitoring, modeling and management system. It is based on intelligent methods that have been widely used by our research team [23–26]. This study has emerged after extensive and long-term research on stream analysis, and it performs actions on real-time data. This paper exploits and considers some of the most important suggestions and innovations of our prior research [27–31].

3.1 Evolving spiking neural network (e-SNN)

Spiking neural networks (SNNs) [32] have been developed to cover and overcome the simplifying assumptions and generalizations of existing technologies. These models aim at simulating (in the most realistic way) the complex structures of the human brain and the way neuronal information is processed and transmitted. SNNs use spike sequences, as neural coding and internal presentation mechanisms, as opposed to the usual continuous variables, and they have better computational cost than traditional neural networks (ANN).

Neural encoding is the domain that analyzes and characterizes the following [32]: (a) the relationship between a stimulus with the individual or overall neuronal responses, (b) the relationship between the overall electrical activities of the neurons and (c) the processes of information transformation and representation in a form suitable for transmission and analysis. It is considering the cases of potential actions that vary in terms of duration, width, intensity and length of intervals between two successive spikes in a serial spike sequence. It is the process of encoding the input data in a suitable form, which is transmitted as serial signals (spike trains).

The e-SNNs [33] are based on the "Thorpe" neural model. They are modular connectionist-based systems that evolve their structure and functionality in a continuous, self-organized, online, adaptive and interactive way, by exploiting incoming information. In order to classify real-valued datasets, each data sample is mapped into a sequence of spikes using the rank order population encoding (ROPE) technique. In this encoding method, neurons are organized into neuronal maps which share the same synaptic weights. Whenever the synaptic weight is modified, the same modification is applied to the entire population within the neuronal map, where inhibition is also present. If a neuron spikes, it inhibits all the neurons in the other maps with neighboring positions. This prevents all neurons from learning the same pattern. When propagating new information, neuronal activity is initially reset to zero. As the propagation goes on (each time one of their inputs fires), neurons are progressively desensitized. This is making their responses dependent upon the relative order of firing of the neuron's afferents. Neural plasticity is used to monitor the learning algorithm by using the one-pass learning method. The aim of this scheme is to create a repository of trained output neurons during the presentation of training samples.

3.2 Restricted Boltzmann machines

The restricted Boltzmann machines (REBOM) [34] belong to the family of energy-based models, where each configuration of the variables of interest corresponds to a finite scalar energy value used for training. The learning process is performed by modifying the energy function (ENF), so that its shape is desirable [35]. An energy model could be trained to output a low value, when the variables' configuration appears frequently in the experimental data, thus giving a generator model. Moreover, the ENF offers the potential to define probability distributions of the following form [34, 35]:

$$P(x) = \frac{e^{-\text{Energy}(x)}}{Z} \tag{5}$$

where x is the variables set and the value of the partition function Z is determined by normalization Eq. 6:

$$Z(x) = \sum_{x} e^{-\text{Energy}(x)} \tag{6}$$

In several problems, it is not possible to observe the values of all variables at the same time. A popular option in these cases is to divide the variables into the following two groups, namely the visible group x and the hidden one denoted as h. The common probability distribution is defined by Eq. 7:

$$P(x,h) = \frac{e^{-\text{Energy}(x,h)}}{Z} \tag{7}$$

4 The proposed model

4.1 The REBOM

This research paper proposes a highly efficient hybrid computing intelligence mechanism for analyzing data flows. This approach stems from the innovative hybrid combination of the e-SNN and REBOM algorithms. Specifically, REBOM is a symmetric graphical model with a visible and a hidden layer described in the [34, 35] and analytical model presented in the following functions.

Specifically, the units of one layer are connected (and thus dependent) only with units of the next one. This approach makes it easier to train its layout. The REBOM's ENF with V visible units and H hidden ones is defined according to Eq. 8:

$$\mathrm{E}(v,h) = -\sum_{i=1}^{V}\sum_{j=1}^{H} v_i h_j w_{ij} - \sum_{i=1}^{V} v_i b_i^v - \sum_{j=1}^{V} h_j b_j^h \tag{8}$$

where v and h are binary vectors related to the state of visible units and to the state of hidden units, respectively. Moreover, v_i and h_j correspond to the individual state of each visible unit (VU) i, and each hidden unit (HU) j, respectively, and w_{ij} is the weight assigned to their connection. Finally, b_i^v and b_j^h are the bias of the VU i and the bias of the HU j. The reserved probability $p(v|h)$ is given by Eq. 9:

$$p(v|h) = \frac{e^{-E(v,h)}}{\sum_g e^{-E(g,h)}} \tag{9}$$

In the specific case of examining a unit i of the visible level, the assigned reserved probability distribution (if we know the status of the hidden layer h) is estimated by Eq. 10:

$$p(v_k = 1|h) = \frac{1}{1 + e^{-\sum_{j=1}^{H} h_j w_{kj} + b_k^v}} \tag{10}$$

The reserved probabilities $p(h|v)$ and $p(h_k = 1|v)$ are defined by Eqs. 11 and 12, respectively:

$$p(h|v) = \frac{e^{-E(v,h)}}{\sum_g e^{-E(v,g)}} \tag{11}$$

and

$$p(h_k = 1|v) = \frac{1}{1 + e^{-\sum_{i=1}^{V} v_i w_{kj} + b_k^h}} \tag{12}$$

These relations express the independence of the units of the two levels from the units of the same level.

The proposed energy model can be trained if appropriate energy-specific limitations for each configuration are given. For example, in the case of a set of observations, we aim to maximize the likelihood that these data will appear in our model, giving them less energy. More specifically, let us give a set of training cases $\{v^c | c \in C\}$. Training the e-REBOM is the process of finding values for its parameters that maximize the mean logarithmic probability of the occurrence of set C (Eq. 13):

$$\sum_{c=1}^{C} \log_p \frac{\sum_g e^{-E(v^c,g)}}{\sum_u \sum_g e^{-E(u,g)}} \tag{13}$$

Several engineering learning problems are usually optimized by using gradient descent, where at each execution step we are approaching the final solution by utilizing the information given by the gradient function at that point. Let us suppose that we want to find the function for the renewal of the weights w_{ij}. The calculation of the partial derivative of the cost function to w_{ij} and the use of Eq. 14 result in the following:

$$\frac{\partial}{\partial w_{ij}} \sum_{c=1}^{C} \log_p (v^c) = \frac{\partial}{\partial w_{ij}} \sum_{c=1}^{C} \log \sum_g e^{-E(v^c,g)} - \log \sum_u \sum_g e^{-E(u,g)} \tag{14}$$

The first term is calculated by the average values of v_i^c, g_j when the visible level of e-REBOM is led by the data (v^c), whereas the second term corresponds to the values of v_i, g_j when the data are "produced" by the model. An equivalent way of formulating would suggest that every weight w_{ij} should change to become equal to Δw_{ij} (Eq. 15):

$$\Delta w_{ij} = \varepsilon_w \left(E_{\text{data}}\left[v_i h_j\right] - E_{\text{model}}\left[v_i h_j\right] \right) \tag{15}$$

The proposed e-REBOM model starts approaching the actual values of the data. The first term $E_{\text{data}}\left[v_i h_j\right]$ can be calculated easily, since knowing the values of the units at the visible level, we can calculate the reserved probability for each unit at the hidden level. Calculation of the second term, which presupposes the existence of samples from the model itself, is more difficult. An obvious solution to the problem is to initiate the units at the visible level at random values and then to take samples by the Gibbs sampling. This can be achieved through the relevant functions, so that after several iterations the model will be forced to "forget" the original (random) state.

Gibbs sampling is a Markov chain Monte Carlo algorithm [36] that repeatedly samples from the conditional distribution of one variable of the target distribution p, given all of the other variables. Gibbs sampling works as follows:

1. Initialize $x^{(t)} = \left(x_1^{(t)}, \ldots, x_k^{(t)}\right)$ (14) for $t = 0$
2. For $t = 0, 1, \ldots$
   - (a) Pick index i uniformly at random from 1, …, k
   - (b) Draw a sample $a \sim p\left(x_i^{'} | x_{-i}^{(t)}\right)$ where $x_{-i}^{(t)}$ is the set of all variables in $x^{(t)}$ except for the ith variable.
   - (c) Let

$$x^{(t+1)} = \left(x_1^{(t)}, x_2^{(t)}, \ldots, x_{i-1}^{(t)}, a, x_{i+1}^{(t)}, \ldots, x_k^{(t)}\right) \tag{16}$$

Gibbs sampler [37] simulates cases of multidimensional target distributions, which are already known. Thus, the overall problem is transformed to a simulation of cases following a low-dimensional distribution. Gibbs sampler is valid under the assumption that we can compute conditional distributions of one variable, based on the rest of the variables and by sampling these distributions. In graphical models, the conditional distribution only depends on the variables in the Markov blanket of that node. Let us show that Gibbs sampling is a special case of Metropolis–

Hastings algorithm (a Markov chain Monte Carlo method) [37] where the proposed moves are always accepted (the acceptance probability is 1).

Let x_i denote the ith variable, and let x_{-i} denote the set of all variables except x_i. Let

$$Q\left(x_i^{'}, x_{-i}|x_i, x_{-i}\right) = \frac{1}{k} p\left(x_i^{'}|x_{-i}\right). \tag{17}$$

Let

$$A\left(x_i^{'}, x_{-i}|x_i, x_{-i}\right) = \min(1, a) \tag{18}$$

where

$$a = \frac{p\left(x_i^{'}, x_{-i}\right) Q\left(x_i, x_{-i}|x_i^{'}, x_{-i}\right)}{p(x_i, x_{-i}) Q(x_i^{'}, x_{-i}|x_i, x_{-i})} \rightarrow$$
$$a = \frac{p\left(x_i^{'}, x_{-i}\right) p(x_i|x_{-i})}{p(x_i, x_{-i}) p(x_i^{'}|x_{-i})} \rightarrow$$
$$a = \frac{p\left(x_i^{'}|x_{-i}\right) p(x_{-i}) p(x_i|x_{-i})}{p(x_i|x_{-i}) p(x_{-i}) p(x_i^{'}|x_{-i})} \rightarrow$$
$$a = 1$$

As it can be seen, the full implementation of this method is not efficient because it requires a long execution time [37]. Hinton [38] showed that if the Markov chain is only run for a few steps, the learning can still work well, and it approximately minimizes a function called contrastive divergence (CODI). The optimal solution in these cases involves optimization through CODI [38]. The basic, single-step contrastive divergence procedure for a single sample can be summarized as follows [39]:

1. Take a training sample v, compute the probabilities of the hidden units and sample a hidden activation vector h from this probability distribution.
2. Compute the outer product of v and h and call this the *positive gradient*.
3. From h, sample a reconstruction v' of the visible units and then resample the hidden activations h' (Gibbs sampling step).
4. Compute the outer product of v' and h' and call this the *negative gradient*.
5. Let the update to the weight matrix W be the positive gradient minus the negative gradient, times some learning rate: $\Delta W = \varepsilon(uh^{\mathrm{T}} - u'h'^{\mathrm{T}})$.
6. Update the biases a and b analogously: $\Delta a = \varepsilon(u - u')$, $\Delta b = \varepsilon(h - h')$.

CODI uses the following two tricks to speed up the sampling process:

A. Since we eventually want $p(u) \approx p_{\text{train}}(u)$ (the true underlying distribution of the data), we initialize the Markov chain with a training example (i.e., from a distribution that is expected to be close to p, so that the chain will be already close to having converged to its final distribution p).
B. CODI does not wait for the chain to converge. Samples are obtained after only k-steps of Gibbs sampling. In practice, k = 1 has been shown to work surprisingly well.

In this case, the contrastive divergence method attempts to approximate the quantity $E_{\text{model}}\left[v_i h_j\right]$, by performing the sampling for a small number of iterations. Units at the visible level are initialized to a sample from the existing actual data, and we perform N iterations, taking some "reconstructed" data, due to model's contribution. The CODI method gives lower energy to the actual data and much higher energy to the "reconstructions" resulting from them. This helps the model approach the actual data distribution.

The primary disadvantage of the contrastive divergence [37, 38] is that the samples from P_n do not necessarily explain the whole state space. Hence, some of the modes in the model distribution are not explored. Even after learning has converged, the model distribution possesses the modes that are not included in the data distribution as it is defined by the training dataset. This problem is illustrated in Fig. 1.

The left figure shows the model distribution (blue) and the training samples (black dots). The blue dots in the right figure indicate the fantasy particles obtained by CODI learning. It is apparent that the fantasy particles failed to explain the whole space by missing the mode at the top. In order to overcome the above problem and to make the training process much faster and more efficient, this research suggests that the e-REBOM training should be performed based on the operation and training mode of the e-SNN algorithm.

4.2 The e-SNN

Spiking neural networks (SNNs) fall into the third generation of neural networks models, increasing the level of realism in a neural simulation. In addition to neuronal and synaptic state, SNNs also incorporate the concept of time into their operating model. The idea is that neurons in the SNN do not fire at each propagation cycle (as it happens with typical multilayer perceptron networks) but they rather fire only when a membrane potential (an intrinsic quality of the neuron related to its membrane electrical charge) reaches a specific value. When a neuron fires, it generates a signal which travels to other neurons which, in turn, increases or decreases their potentials in accordance with this signal [32].

The topology of the e-SNN is strictly feed-forward, organized in several layers, and weight modification occurs on the connections between the neurons of the existing

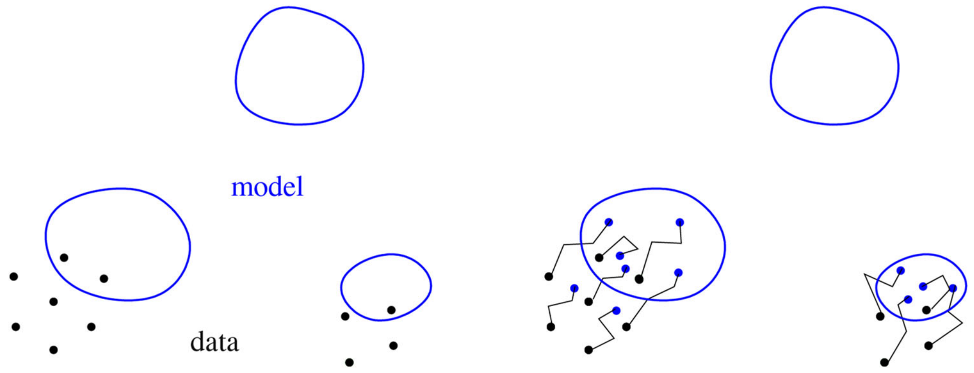

Fig. 1 Problem of the contrastive divergence (color figure online)

layers. The encoding is performed by the rank order population encoding (ROPE) technique with 20 Gaussian Receptive Fields (GRF) per variable [33]. The ROPE method is an alternative to conventional rate coding scheme that uses the order of firing neuron's inputs to encode information. This allows the mapping of vectors of real-valued elements into a sequence of spikes. In addition, the e-SNN [33] uses the one-pass learning method. The aim of this method is to create a repository of trained output neurons during the presentation of training samples. After presenting a certain input sample to the network, the corresponding spike train is propagated through the network which may result in the firing of certain output neurons. It is also possible that no output neuron is activated and in this case the network remains silent and the classification result is undetermined. If one or more output neurons have emitted a spike, the neuron with the shortest response time among all activated output neurons is determined. The label of this neuron represents the classification result for the presented input sample.

The CODI is used to train the REBOM model, by optimizing the weight vector. In the spiking version of this algorithm, Hebbian rule is used to calculate the weight change in forward and reconstruction phase. Weight changes from data layers result in potentiation of synapses while those in model layers result in depreciation. Also, weight change is calculated only when hidden layer neuron fires. The proposed e-SREBOM implementation is based on the following two rules:

- Any synapse that contributes to the firing of a postsynaptic neuron should be made strong.
- Synapses that do not contribute to the firing of a postsynaptic neuron should be demised.

Following the above rules, this is the algorithm for updating weights.

- If a presynaptic neuron fires before a postsynaptic neuron, then corresponding synapse should be made strong by a factor proportional to the time difference between the spikes. The smaller the time difference between a postsynaptic and a presynaptic spike, the higher is the contribution of that synapse in postsynaptic firing and hence the greater the weight change (positive).
- If a presynaptic neuron fires after a postsynaptic neuron, the corresponding synapse should be diminished by a factor proportional to the time difference between the spikes. The smaller the time difference between a postsynaptic and a presynaptic spike, the smaller is the contribution of that synapse in postsynaptic firing and hence the greater the weight change (negative).

The weights' update mechanism is performed by the e-SNN algorithm. Specifically, neurons are organized into neuronal maps which share the same synaptic weights. Whenever the synaptic weight of a neuron is modified, the same modification is applied to the entire population of neurons within the map. Inhibition is also present between each neuronal map. If a neuron spikes, it inhibits all the neurons in the other maps with neighboring positions. This prevents all the neurons from learning the same pattern. When propagating new information, neuronal activity is initially reset to zero. Then, as the propagation goes on, neurons are progressively desensitizing each time one of their inputs fire, thus making neuronal responses dependent upon the relative firing order of the neuron's afferents. More precisely, let $A = \{a_1, a_2, a_3, \ldots, a_{m-1}, a_m\}$ be the ensemble of afferent neurons of neuron i and $W = \{w_{1,i}, w_{2,i}, w_{3,i} \ldots w_{m-1,i}, w_{m,i}\}$ the weights of the m corresponding connections; let mod $\in [0, 1]$ be an arbitrary modulation factor. The activation level of neuron i at time t is given by Eq. 19 [33]:

$$\text{Activation}\ (i, t) = \sum_{j \in [1,m]} \text{mod}^{\text{order}(a_j)} w_{j,i} \tag{19}$$

where order(a_j) is the firing rank of neuron a_j in the ensemble A. By convention, if a neuron a_j does not fire at time t, the corresponding term order(a_j) in the above sum is set to zero. This kind of desensitization function could correspond to a fast shunting inhibition mechanism. Whenever a neuron reaches its threshold, it spikes and inhibits neurons at equivalent positions in the other maps so that only one neuron will respond at any particular location. Every spike also triggers a time-based Hebbian-like learning rule that adjusts the synaptic weights. Let t_e be the date of arrival of the excitatory postsynaptic potential at synapse of weight W, and let t_a be the date of discharge of the postsynaptic neuron:

$$\text{if } t_e < t_a \text{ then } \mathrm{d}W = a(1 - W)e^{-|\Delta o|\tau} \tag{20}$$

$$\text{else} \qquad \mathrm{d}W = -aWe^{-|\Delta o|\tau} \tag{21}$$

where Δo is the difference between the date of the EPSP and the date of the neuronal discharge (expressed in terms of order of arrival instead of time) and a is a constant that controls the amount of synaptic potentiation and depression [2].

4.3 The e-SREBOM

This rule of weight update has been used in the CODI algorithm in order to train the e-SREBOM. In this implementation, the change in weight is kept constant in the entire window. The hyper-parameters of this e-SREBOM version are described below:

- *Learning rate* This parameter determines the size of a weight update when a hidden layer neuron spikes, and it controls how quickly the system changes its weights to approximate the input distribution. It is considered to be the most basic parameter of any neural network. There is a trade-off associated with this parameter, and it can be explained by the same experiment done above. Higher learning rate develops fast receptive fields but in an improper way. Accuracy increases quickly, but it reaches a plateau much earlier. Lower learning rate results in better training but it requires more samples (more time) to reach the highest accuracy.
- *Spikes per sample* This parameter, also known as luminosity, defines the spiking activity of the network quantitatively. It is preferred to keep the activity as low as possible (enough to change the weights). We have kept a maximum bound on the number of spikes that an input can generate. Without this moderation, there will be no uniformity in the input activity across all the patterns.
- *Weight initialization* The range of uniformly distributed weights used to initialize the network play a very significant role in training, which most of the times is not considered properly. Properly initializing the weights can save significant computational effort, and it can have drastic results on the eventual accuracy. Generally, the weights are initialized between 0 and 1.

When signals propagate from visible to hidden, the input layer (i.e., the data sample) is multiplied by the weight matrix W and it is added with the bias vector b of the hidden layer. It finally goes through the sigmoid function to be translated in the interval [0, 1], which corresponds to the probabilities for each hidden neuron to be on. However, it is very important to keep the hidden states binary (0 or 1), rather than using the probabilities itself. Only during the last update of the Gibbs sampling, should we use probabilities for the hidden layer. During backward pass, or reconstruction, the hidden layer activation becomes the input, which is multiplied by the same weight matrix W, added with visible biases, and then it either goes through the sigmoid function or it is sampled from a multivariate Gaussian distribution. The model is adjusting its weights, during training, such that it could best approximate the training data distribution p with its reconstruction distribution q. For any pair of x and h, we are able to calculate energy function $E(x, h)$. Its value is scalar. The higher the value of energy function, the lower the joint probability $p(x, h)$. Generally, the energy function, which is also scalar, is exactly what we need for testing data, from which we will use the distribution to detect anomalies. The higher the energy, the higher the chance of x being some anomalies [34, 39].

The process to tune hyper-parameters is described below:

1. Split the data into training and validation sets with ROPE technique. ROPE technique with receptive fields allows the encoding of continuous values by using a collection of neurons with overlapping sensitivity profiles [33]. Each input variable is encoded independently by a group of one-dimensional receptive fields. For a variable n, an interval $[I^n_{\min}, I^n_{\max}]$ is defined. The Gaussian receptive field of neuron i is given by its center μ_i:

$$\mu_i = I^n_{\min} + \frac{2i-3}{2}\frac{I^n_{\max} - I^n_{\min}}{M-2} \tag{22}$$

   The width σ is given by the following function (23):

$$\sigma = \frac{1}{\beta}\frac{I^n_{\max} - I^n_{\min}}{M-2} \tag{23}$$

   where $1 \le \beta \le 2$ and the parameter β directly controls the width of each Gaussian receptive field.
2. Train the model on the training dataset with one-pass learning method, while evaluating the performance on validation. The aim of the one-pass learning method is to create a repository of trained output neurons, during the presentation of the samples. After presenting a certain input sample to the network, the corresponding spike train is propagated through the spiking neural network, which may result in the firing of certain output neurons. It is also possible that no output neuron is activated, and in this case, the network remains silent and the classification result is undetermined. If one or more output neurons have emitted a spike, the one with the

shortest response time among all is activated. The label of this neuron represents the classification result for the presented input sample. For each training sample i with class label $l \in$ L a new output neuron is created and fully connected to the previous layer of neurons This results in a real-valued weight vector $w^{(i)}$ with $w_j^{(i)} \in R$ denoting the connection between the presynaptic neuron j and the created neuron i. In the next step, the input spikes are propagated through the network and the value of weight $w_j^{(i)}$ is computed according to the order of spike transmission through a synapse $j : w_j^{(i)} = (m_l)^{\mathrm{order}(j)}$, $\forall j|j$ presynaptic neuron of i. Parameter m_l is the modulation factor of the Thorpe neural model. Differently labeled output neurons may have different modulation factors m_l. Function order(j) represents the rank of the spike emitted by neuron j. The firing threshold $\theta^{(i)}$ of the created neuron I is defined as the fraction $c_l \in R$, $0 < c_l < 1$, of the maximal possible potential $u_{\max}^{(i)}$:

$$\theta^{(i)} \leftarrow c_l u_{\max}^{(i)} \tag{24}$$

$$u_{\max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{\mathrm{order}(j)} \tag{25}$$

The fraction c_l is a parameter of the model, and for each class label $l \in$ L, a different fraction can be specified. The weight vector of the trained neuron is then compared to the weights corresponding to neurons already stored in the repository. Two neurons are considered too "similar", if the minimal Euclidean distance between their weight vectors is smaller than a specified similarity threshold s_l. (The e-SNN object uses optimal similarity threshold s = 0.6.) In this case, both the firing thresholds and the weight vectors are merged according to functions (26) and (27):

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1+N}, \quad \forall \; j\,|\,j \text{ pre-synaptic neuron of } i \tag{26}$$

$$\theta^{(k)} \leftarrow \frac{\theta^{(i)} + N\theta^{(k)}}{1+N} \tag{27}$$

The merging is implemented as the (running) average of the connection weights and the (running) average of the two firing thresholds. After the merging, the trained neuron i is discarded and the next sample is processed. If no other neuron in the repository is similar to the trained neuron i, the neuron i is added to the repository as a new output neuron.The overall process of the e-SREBOM training updates using CODI is presented in Algorithm 1:

Algorithm 1. The training update of the eSREBOM using Contrastive Divergence

```
Inputs: Input: training pair ($y_t$,$x_t$) and learning rate $\lambda$.
    for a class label $l \in L_d$
        initialize neuron repository $R_l = \{\}$
    for all samples $X^{(i)}$ belonging to class $l$ do
        $w_j^{(i)} \leftarrow (m_l)^{order(j)}$, $\forall$ j | j pre-synaptic neuron of i
        $u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)}$
        $\theta^{(i)} \leftarrow c_l u_{max}^{(i)}$
    if min(d($w^{(i)}$, $w^{(k)}$)) < $s_l$, $w^{(k)} \in R_l$ then
        $w^{(k)}$ ← merge $w^{(i)}$ and $w^{(k)}$ according to $u_{max}^{(i)} : \theta^{(i)} \leftarrow c_l u_{max}^{(i)}$
        $\theta^{(k)}$ ← merge $\theta^{(i)}$ and $\theta^{(k)}$ according to $u_{max}^{(i)} \leftarrow \sum_j w_j^{(i)} (m_l)^{order(j)}$
    else
        $R_l \leftarrow R_l \cup \{w^{(i)}\}$
    end if
    end for
    # Notation: a ← b means a is set to value b and a ~ p means a is sampled from p
    # Positive phase
    $y^o \leftarrow y_t, x^o \leftarrow x_t, \widehat{h^0} \leftarrow sigm(c + Wx^0 + Ue_{y^0})$
    # Negative phase
    $h^0 \sim p(h|y^o, x^o), y^1 \sim p(y|h^o), x^1 \sim p(x|h^o)$
    $\widehat{h^1} \leftarrow sigm(c + Wx^1 + Ue_{y^1})$
    # Update
    for $\theta \in \Theta$ do
        $\theta \leftarrow \theta - \lambda\left(\frac{\partial}{\partial\theta}E(y^o, x^o, \widehat{h^0}) - \frac{\partial}{\partial\theta}E(y^1, x^1, \widehat{h^1})\right)$
    end for
```

3. Start the hidden layer with a smaller dimension than the input layer (e.g., 5, 10). Set the learning rate to have a small value (such as 0.001), and monitor the validation dataset reconstruction error (not the actual error against labels).
4. The reconstruction error is basically the mean squared of the difference between the predicted x' and the actual data value x, averaged over the entire mini-batch.
5. If the reconstruction error stops decreasing, that would be a sign for early stopping.

5 Datasets

Appropriate datasets were chosen that closely simulate Industrial Control Systems communication and transaction data. They were used in the development and evaluation of the proposed model. Network traffic analysis and feature extraction methodology were determined based on the functionality of ICS, namely on the verification of the reliable and error-free data upload and capture mechanisms. The sets also include data logs from flagged network transactions during the regular operation of certain ICSs, as well as transactions during 35 different cyberattacks. Finally, they also include measurements of physiological behavior as well as abnormalities detected during attacks, which were simulated in a virtual ICS environment. It should be noted that the configured virtual systems have no physical limits to the size of the simulation; therefore, the virtual platform on which the data were collected was formed by scaling the ICS. This was done in order to represent their operational situations in the most realistic way: the contained data from a laboratory-scale water tower, a gas pipeline and a laboratory-scale electric transmission system namely [40]:

- The *water_tower_dataset* includes 23 independent parameters and 236,179 instances, comprising 172,415 normal and 63,764 outliers. Totally 86,315 normal instances were used in the training phase (water_train_-dataset) whereas the rest 86,100 normal instances and 63,764 outliers comprised the water_test_dataset.
- The *gas_dataset* includes 26 independent features and 97,019 instances, comprising 61,156 normal and 35,863 outliers. The training of the algorithm was done with the gas_train_dataset that contains 30,499 normal instances, whereas the rest 30,657 normal instances and 35,863 outliers belong to the gas_test_dataset.
- Finally, the *electric_dataset* includes 128 independent variables with 146,519 instances, comprising 90,856 normal and 55,663 outliers. The training was performed based on the electric_train_dataset comprising of 45,402 normal instances, whereas the rest 45,454 normal and the 55,663 outliers belong to the electric_test_dataset.

The datasets include preprocessed network transaction data to strip lower layer transmission (e.g., TCP, MAC). In addition, our data include normal behavior measurements, as well as abnormalities detected during attacks that were simulated in a virtual ICS environment. The dataset is determined and normalized to the interval [− 1, 1] in order to phase the problem of prevalence of features with wider range over the ones with a narrower range, without being more important. Details regarding the dataset, their choice and assessment can be found in [40].

6 Evaluation metrics

A key point in any intelligent system is the evaluation methodology. Learning systems generate compact representations of what is being observable. They should be able to improve with experience and continuously self-modify their internal state. Their representation of the world is approximate. Evaluation is used in two contexts: inside the learning system to assess hypothesis and as a wrapper over the learning system to estimate the applicability of a particular algorithm in a given problem.

In most supervised tasks for machine learning, for each new example an exact loss function may be computed with respect to a previously made prediction. This is especially useful to assess the quality of online predictive models. For predictive learning tasks, the learning goal is to induce a function $\hat{y} = f(\vec{x})$. The most relevant dimension is the generalization error. It is an estimator of the difference between f and the unknown $\hat{f}$, and an estimate of the loss that can be expected when applying the model to future examples. In online learners, this estimate can be used not only to assess the quality of the model, but also to tune the model's parameters before applying it to future examples. Given the online setting of learning from data streams, the quality of a learning model is difficult to condense in a single value of loss or performance, since data are being produced with evolving concepts and the model itself is being continuously updated. The evaluation of online learners has been assuming at the end of the learning process by computing average losses in time windows.

The analysis of data streams is a specialized machine learning problem that requires specific metrics to measure the accuracy [41]. We evaluated the applied approaches by measuring the accuracy, based on the average values of the following indices, namely: Kappa statistic (K-Stats), Kappa temporal statistic (K-Temp-Stats) and Time to learn.

The data flow classifiers aim to capture the actual error. Comparison of classification performance is done in terms of *accuracy Kappa* statistic and *Kappa temporal* statistic, using the traditional immediate setting [42]. The true label is presented right after the instance has been used for testing, or after the delayed setting, where there is a delay between the presentation of an instance and the availability of its true label.

The Kappa statistic is a popular measure for validating classification accuracy under class imbalance. It is used in static classification scenarios, as well as in streaming data classification. The Kappa statistic or Cohen's kappa measures the agreement between two raters which classify N items into C mutually exclusive categories. The definition of κ is given by Eq. 28:

$$\kappa = \frac{p_0 - p_e}{1 - p_e} = 1 - \frac{1 - p_0}{1 - p_e} \tag{28}$$

where p_0 is the relative observed agreement among raters (identical to accuracy) and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly seeing each category. If the raters are in complete agreement, then $\kappa = 1$. If there is no agreement among the raters other than what would be expected by chance (as given by p_e), $\kappa \approx 0$. Considering the presence of temporal dependencies in data streams, the Kappa temporal statistic is defined by Eq. 29:

$$\kappa_{\mathrm{T}} = \frac{p - p_{\mathrm{per}}}{1 - p_{\mathrm{per}}} \tag{29}$$

where p_{per} is the accuracy of the persistent classifier.

The Kappa temporal statistic may take values in the interval $[1, -\infty)$. The interpretation is similar to that of κ. If the classifier is perfectly correct, then $\kappa_{\mathrm{per}} = 1$. If it is achieving the same accuracy as the persistent one, then $\kappa_{\mathrm{per}} = 0$. Classifiers that outperform the persistent one fall between 0 and 1. Sometimes it may happen that $\kappa_{\mathrm{per}} < 0$, which means that the reference classifier is performing worse than the persistent one's baseline.

Therefore, using κ_{per} instead of κ, we will be able to detect misleading classifier performance, for data that have temporal dependence. For highly imbalanced, but independently distributed data, the majority class classifier may beat the persistent and thus the use of κ_{per} will not be sufficient enough. Overall, κ_{per} and κ measures can be seen as orthogonal, since they measure different aspects of performance. Hence, for a thorough evaluation we recommend measuring and combining both.

It is important to mention that the efficiency of computing the Kappa statistic is an important reason why it is more appropriate for data streams than a measure such as the area under the ROC curve.

The evaluation parameters that have been selected are the following:

- *Accuracy* It refers to the reliability of the rule, usually represented by the proportion of correct classifications, although it may be that some errors are more serious than others, and it may be important to control the error rate for some key class.
- *Speed* In some circumstances, the speed of the classifier is a major issue. A classifier that is 90% accurate may be preferred over one that is 95% accurate if the former is 100 times faster in testing (and such differences in timescales are not uncommon in neural networks). Such considerations would be even more important when the number of samples to be processed is very large.
- Time to learn: In particular, in a rapidly changing environment, it may be necessary to learn a classification rule quickly, or to adjust an existing rule in real time. "Quickly" might imply also that we need only a small number of observations to establish our rule. In addition, retraining can be performed very fast, and this in turn allows it to be performed more often.

7 Results and discussion

In all simulations, the testing hardware and software conditions are listed as follows: Laptop Intel-i7 2.4 G CPU, 16 G DDR3 RAM, Ubuntu 18.04 LTS, Anaconda Python Data Science Platform and TensorFlow Python environment. All experiments are shown in Tables 1, 2 and 3. We also considered CPU time or speed (the total CPU time used by the process since it started, precise to the hundredths of a second) and memory consumption as RAM Hours (RAM-H) as estimates of computational resources usage. We have used a shell script based in the "*top*" command to monitor processes and system resource usage on the Linux OS.

The learning process used 10,000 instances, and the validation of the results was done by employing the prequential evaluation method [41]. For the sake of completeness, we reported the error rates of all window-based approaches with a window size of 1000 samples. The training window used 5000 instances. Window-based approaches were allowed to store 5000 samples but never more than 10% of the whole dataset. This rather large amount offers a high degree of freedom, and it prevents the concealment of their qualities with a very restricted window.

A comparative analysis of the performance of various data stream classification algorithms was made with the proposed e-SREBOM approach. The traditional immediate setting approach where the true label is presented right after the instance, or the delayed setting method which

Table 1 Results *water_tower_dataset*

Performance metrics					
Classifier	K-Stats (%)	K-Temp-Stats (%)	Time to learn (s)	Speed (summary) (s)	RAM-H
Window size = 5000					
k-NN SAM	74.56	75.29	38	178	0.0135
SPegasos	72.07	72.94	41	165	0.0130
ARF	71.86	72.47	30	113	0.0126
e-SREBOM	75.95	76.87	29	111	0.0128
Window size = 1000					
k-NN SAM	79.22	79.96	12	81	0.0088
SPegasos	75.65	77.51	11	79	0.0080
ARF	75.24	77.72	14	82	0.0056
eS-REBOM	81.13	82.54	10	71	0.0060

Table 2 Results *gas_dataset*

Performance metrics					
Classifier	K-Stats (%)	K-Temp-Stats (%)	Time to learn (s)	Speed (summary) (s)	RAM-H
Window size = 5000					
k-NN SAM	72.03	72.73	51	185	0.0159
SPegasos	72.02	72.69	50	182	0.0164
ARF	71.83	72.41	52	148	0.0167
e-SREBOM	72.99	73.15	46	141	0.0158
Window size = 1000					
k-NN SAM	74.18	75.41	44	99	0.0101
SPegasos	73.94	75.01	47	101	0.0103
ARF	73.87	74.89	43	84	0.0097
e-SREBOM	74.31	75.24	42	89	0.0100

Table 3 Results *electric_dataset*

Performance metrics					
Classifier	K-Stats (%)	K-Temp-Stats (%)	Time to learn (s)	Speed (summary) (s)	RAM-H
Window size = 5000					
k-NN SAM	75.72	76.33	76	192	0.0191
SPegasos	75.63	76.12	81	198	0.0190
ARF	74.47	75.16	73	193	0.0196
e-SREBOM	75.86	77.09	70	189	0.0190
Window size = 1000					
k-NN SAM	80.63	81.96	21	93	0.0069
SPegasos	77.95	78.93	28	89	0.0072
ARF	76.18	77.97	23	83	0.0070
e-SREBOM	80.61	82.03	19	82	0.0071

applies a delay between the time an instance is presented and the time its true label becomes available, has been followed in testing [30]. The evaluation metrics used were accuracy Kappa statistic and Kappa temporal statistic.

Looking at the results, we conclude that the use of the e-SREBOM algorithm is an optimal practice, since it solves extremely quickly and with great precision and reliability, a real Information Systems' security problem. It is remarkable that this is achieved without requiring high availability of computational resources. Additionally, this technique employs a relatively smooth and fast learning rhythm, which determines how quickly learning converges. A high learning rate can lead to faster convergence and oscillation around optimal weight values, whereas low rate can result in slower convergence and it can lead to trapping at local extremes. The high rate of learning convocation is confirmed by the high accuracy of the model, considering the small magnitude of the examined data flow, for such a case of batch dataset evaluation.

The quality of the model's adaptation is proved by the fact that the model retains high accuracy levels or improves its forecasting percentages in a continuously changing environment, without been misled by temporary conditions. Specifically, the temporal bias that is developed in a certain time stamp is absorbed by keeping stable classification accuracy percentages, as they are presented in the results' table.

An additional important attribute resulting from the high accuracy of the proposed e-SREBOM algorithm is the relatively low "mutation" rate in the changes that characterize the data flows. The algorithm is not trapped in local extremes (minima/maxima) or general anomalies included in data flows or learning windows. This fact allows the discovery of local extremes that may be included in them, or in a learning window, as new areas of the multidimensional solution space are being explored.

On the contrary, if the mutation rate was too high, it could trap the system in solutions that do not generalize.

An important additional comment concerns the Kappa coefficient that links the level of the observed agreement to the level of the random agreement, and it estimates the variation in each observer–rater. This variability occurs when the same observer–evaluator evaluates differently the same case in repeated evaluations. The maximum value of the Kappa index, represents the full agreement between observers and markers, whereas the minimum value 0 means that there are only random agreement and thus no reliability between observers and markers.

The hybrid e-SREBOM model proposed herein has been proven reliable in all tests. The adaptive random forest approach generally needs a larger number of instances in order to achieve better accuracy with new data. Moreover, ARF works by combining some loose linear boundaries on the decision surface, as opposed to SPegasos which can achieve max margin in nonlinear boundaries. Therefore, given that the windows are characterized by a small amount of data, SPegasos yielded higher success rates than the ARF. Regarding the comparison between SPegasos and the *k*-NN self-adjusting memory, it is easy to understand why *k*-NN SAM performed better. This is due to the fact that the specific case examined herein is related to a high-dimensional space, where this algorithm is more efficient.

Also, the optimal combination of the two levels of memory, the employed different time intervals between memories and the transfer of knowledge, has minimized the errors and the increased the classification accuracy. The e-SREBOM algorithm manages to decode data from a specific distribution in a very prominent way, without this process requiring special computational resources. It also discovers accurately the high-level correlations contained in datasets, due to the way the hidden layers of the algorithm work, where the units of one layer depend only on units of the other.

Due to the above characteristics of the e-SREBOM, the general model is extremely fast, mainly because of the connection limitations between its hidden and its visible units. Another important advantage of the proposed algorithm is the ability to isolate and reject the random noise that may be contained in the training set.

8 Conclusions

This work presented a hybrid, innovative, reliable and highly effective algorithm for detecting data flow anomalies, based on sophisticated computational intelligence. The e-SREBOM is a clearly innovative effort to effectively analyze large-scale data flows. The proposed method is based on the optimal combination of the e-SNN and the REBOM algorithms, which ensures the adaptation of the system in new situations. It offers high level of generalization, by implementing a robust algorithm capable of responding to high-complexity problems. The performance of the proposed algorithm was tested on three-multidimensional datasets of high complexity. These datasets emerged as a result of an extensive research on the function of industrial control systems ICS (SCADA, DCS, PLC). They realistically state the operating modes of these devices in normal conditions and in situations where they are subject to cyberattacks. The results have proven the efficiency of the developed hybrid model.

Future research will focus in further optimization of the algorithm's parameters that may result in a faster and more accurate performance. We will work on the improvement of the e-SREBOM's complexity in a high understandable and adjustable level.

Further optimization by means of self-improvement and adaptive learning can be explored in order to fully automate the process of detecting abnormalities. Finally, a very important future improvement is the extension of the algorithm for incremental online learning, with long–short-term memory capabilities, in order to approach and model time sequences and their broader dependencies with greater accuracy and efficiency.

Compliance with ethical standards

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

References

1. Dedić N, Stanier C (2017) Towards differentiating business intelligence, Big Data, data analytics and knowledge discovery, vol 285. Springer, Berlin
2. Benjelloun F, Lahcen AA, Belfkih S (2015) An overview of Big Data opportunities, applications and tools. In: 2015 intelligent systems and computer vision (ISCV), pp 1–6. https://doi.org/10.1109/ISACV.2015.7105553
3. Kiran M, Murphy P, Monga I, Dugan J, Baveja SS (2015) Lambda architecture for cost-effective batch and speed Big Data processing. In: IEEE international conference on Big Data (Big Data), Santa Clara, CA, pp 2785–2792. https://doi.org/10.1109/bigdata.2015.7364082
4. Demchenko Y, de Laat C, Membrey P (2014) Defining architecture components of the Big Data Ecosystem. In: 2014 international conference on collaboration technologies and systems (CTS), Minneapolis, MN, pp 104–112. https://doi.org/10.1109/cts.2014.6867550
5. Sample C, Schaffer K (2013) An overview of anomaly detection. IEEE J Mag 15(1):8–11. https://doi.org/10.1109/MITP.2013.7
6. Borah A, Nath B (2017) Mining patterns from data streams: an overview. In: 2017 international conference on I-SMAC (IoT in social, mobile, analytics and cloud) (I-SMAC), pp 371–376. https://doi.org/10.1109/I-SMAC.2017.8058373
7. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002). Models and issues in data stream systems. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. PODS ‘02. ACM, New York, pp 1–16. CiteSeerX 10.1.1.138.190. https://doi.org/10.1145/543613.543615. ISBN 978-1581135077
8. Kushner HJ, Yin GG (2003) Stochastic approximation algorithms and applications, Springer, New York (1997). ISBN 0-387-94916-X; 2nd edn, titled Stochastic approximation and recursive algorithms and applications, ISBN 0-387-00894-2
9. Bifet A, Holmes G, Pfahringer B (2010) Leveraging bagging for evolving data streams. In: Balcázar JL, Bonchi F, Gionis A, Sebag M (eds) Machine learning and knowledge discovery in databases. ECML PKDD 2010, vol 6321. Lecture notes in computer science. Springer, Berlin
10. Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining, pp 443–448. https://doi.org/10.1137/1.9781611972771.42
11. Minku LL, White AP, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. IEEE Trans Knowl Data Eng 22(5):730–742. https://doi.org/10.1109/TKDE.2009.156
12. Baena-Garcia M, Del Campo-Avila J, Fidalgo R, Bifet A, Gavalda R, Morales-Bueno R (2006) Early drift detection method. In: 4th ECML PKDD international workshop on knowledge discovery from data streams, pp 77–86
13. Farid DM, Zhang L, Hossain A, Rahman CM, Strachan R, Sexton G, Dahal K (2013) An adaptive ensemble classifier for mining concept drifting data streams. Expert Syst Appl 40(15):5895–5906. https://doi.org/10.1016/j.eswa.2013.05.001
14. Wang L-Y, Park C, Choi H, Yeon K (2016) A classifier ensemble for concept drift using a constrained penalized regression combiner. Procedia Comput Sci 91:252–259. https://doi.org/10.1016/j.procs.2016.07.070
15. Yang Z, Al-Dahidi S, Baraldi P, Zio E, Montelatici L (2019) A novel concept drift detection method for incremental learning in nonstationary environments. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2019.2900956
16. Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 71–80. https://doi.org/10.1145/347090.347107
17. Aggarwal CC, Han J, Wang J, Yu PS (2004) On demand classification of data streams. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 503–508. https://doi.org/10.1145/1014052.1014110
18. Zhang P, Zhu X, Shi Y (2008) Categorizing and mining concept drifting data streams. In: Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, pp 812–820. https://doi.org/10.1145/1401890.1401987
19. 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. https://doi.org/10.1109/ICDM.2016.0040
20. Rani MS, Sumathy S (2016) Analysis of KNN, C5.0 and one class SVM for intrusion detection system. Int J Pharm Technol 8(4):26251–26259
21. Shalev-Shwartz S, Singer Y, Srebro N, Cotter A (2011) Pegasos: primal estimated sub-gradient solver for SVM. Math Program 127(1):3–30. https://doi.org/10.1007/s10107-010-0420-4
22. Gomes HM, Bifet A, Read J, Barddal JP, Enembreck F, Pfharinger B, Holmes G, Abdessalem T (2017) Adaptive random forests for evolving data stream classification. Mach Learn 106(9–10):1469–1495. https://doi.org/10.1007/s10994-017-5642-8
23. Demertzis K, Iliadis L (2014) Evolving computational intelligence system for malware detection. In: Iliadis L, Papazoglou M, Pohl K (eds) Advanced information systems engineering workshops. Lecture notes in business information processing. Springer, Berlin, pp 322–334
24. Demertzis K, Iliadis LS (2016) Ladon: a cyber threat bio-inspired intelligence management system. J Appl Math Bioinform 6(3):45–64
25. Demertzis K, Iliadis LS, Anezakis V-D (2018) An innovative soft computing system for smart energy grids cybersecurity. Adv Build Energy Res 12:3–24. https://doi.org/10.1080/17512549.2017.1325401
26. Demertzis K, Iliadis L (2018) A computational intelligence system identifying cyber-attacks on smart energy grids. In: Daras NJ, Rassias TM (eds) Modern discrete mathematics and analysis: with applications in cryptography, information systems and modeling. Springer optimization and its applications. Springer, Cham, pp 97–116. https://doi.org/10.1007/978-3-319-74325-7_5
27. Demertzis K, Iliadis L (2014) A hybrid network anomaly and intrusion detection approach based on evolving spiking neural network classification. In: Sideridis AB, Kardasiadou Z, Yialouris CP, Zorkadis V (eds) E-democracy, security, privacy and trust in a digital world, communications in computer and information science. Springer, Berlin, pp 11–23
28. Demertzis K, Iliadis L, Spartalis S (2017) A spiking one-class anomaly detection framework for cyber-security on industrial control systems. In: Boracchi G, Iliadis L, Jayne C, Likas A (eds) Engineering applications of neural networks, communications in computer and information science. Springer, Berlin, pp 122–134
29. Demertzis K, Kikiras P, Tziritas N, Sanchez SL, Iliadis L (2018) The next generation cognitive security operations center: network flow forensics using cybersecurity intelligence. Big Data and Cogn Comput 2:35. https://doi.org/10.3390/bdcc2040035
30. Demertzis K, Iliadis L, Anezakis V (2018) MOLESTRA: a multi-task learning approach for real-time Big Data analytics. In: 2018 innovations in intelligent systems and applications (INISTA). Presented at the 2018 innovations in intelligent systems and

applications (INISTA), pp 1–8. https://doi.org/10.1109/INISTA.2018.8466306
31. Demertzis K, Iliadis L, Anezakis V-D (2018) A dynamic ensemble learning framework for data stream analysis and real-time threat detection. In: Kůrková V, Manolopoulos Y, Hammer B, Iliadis L, Maglogiannis I (eds) Artificial neural networks and machine learning—ICANN 2018. Lecture notes in computer science. Springer, Berlin, pp 669–681
32. Ponulak F, Kasinski A (2011) Introduction to spiking neural networks: information processing, learning and applications. Acta Neurobiol Exp 71(4):409–433
33. Schliebs S, Kasabov N (2013) Evolving spiking neural network—a survey. Evolv Syst 4(2):87–98. https://doi.org/10.1007/s12530-013-9074-9
34. Zhang N, Ding S, Zhang J, Xue Y (2018) An overview on restricted Boltzmann machines. Neurocomputing 275:1186–1199. https://doi.org/10.1016/j.neucom.2017.09.065
35. LeCun Y, Chopra S, Hadsell R, Huang FJ et al (2006) A tutorial on energy-based learning. In: BakIr G, Hofmann T, Schölkopf B, Smola AJ, Taskar B, Vishwanathan SVN (eds) Predicting structured data. MIT Press, Cambridge
36. van Ravenzwaaij D, Cassey P, Brown SD (2018) A simple introduction to Markov chain Monte-Carlo sampling. Psychon Bull Rev 25(1):143–154. https://doi.org/10.3758/s13423-016-1015-8
37. Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans Pattern Anal Mach Intell 6(6):721–741. https://doi.org/10.1109/TPAMI.1984.4767596
38. Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14(8):1771–1800. https://doi.org/10.1162/089976602760128018
39. Liu J, Chi G, Luo X (2013) Contrastive divergence learning for the restricted Boltzmann machine. In 2013 9th international conference on natural computation (ICNC), pp 18–22. https://doi.org/10.1109/ICNC.2013.6817936
40. Morris T, Gao W (2014). Industrial control system traffic data sets for intrusion detection research. In: Butts J, Shenoi S (Eds) 8th international conference on critical infrastructure protection (ICCIP), Mar 2014, Arlington, United States, IFIP advances in information and communication technology, AICT-441. Critical infrastructure protection VIII. Springer, pp 65–78
41. Vinagre J, Jorge AM, Gama J (2014) Evaluation of recommender systems in streaming environments. In: Workshop on 'recommender systems evaluation: dimensions and design' (REDD 2014), held in conjunction with RecSys 2014. Oct 10 2014, Silicon Valley, USA. https://doi.org/10.13140/2.1.4381.5367
42. Žliobaitė I, Bifet A, Read J et al (2015) Evaluation methods and decision theory for classification of streaming data with temporal dependence. Mach Learn 98:455. https://doi.org/10.1007/s10994-014-5441-4