Neural Networks 139 (2021) 118–139

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Unsupervised Anomaly Detection in Stream Data with Online Evolving Spiking Neural Networks

Piotr S. Maciąg [a,*], Marzena Kryszkiewicz [a], Robert Bembenik [a], Jesus L. Lobo [b], Javier Del Ser [b,c]

[a] Warsaw University of Technology, Institute of Computer Science, Nowowiejska 15/19, 00-665, Warsaw, Poland
[b] TECNALIA, Basque Research and Technology Alliance (BRTA), Parque Tecnológico de Bizkaia, E-700, 48160 Derio, Spain
[c] University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain

ARTICLE INFO

Article history:
Received 25 June 2020
Received in revised form 7 January 2021
Accepted 12 February 2021
Available online 25 February 2021

Keywords:
Evolving Spiking Neural Networks
Outliers detection
Online learning
Time series data
Unsupervised anomaly detection
Stream data

ABSTRACT

Unsupervised anomaly discovery in stream data is a research topic with many practical applications. However, in many cases, it is not easy to collect enough training data with labeled anomalies for supervised learning of an anomaly detector in order to deploy it later for identification of real anomalies in streaming data. It is thus important to design anomalies detectors that can correctly detect anomalies without access to labeled training data. Our idea is to adapt the Online evolving Spiking Neural Network (OeSNN) classifier to the anomaly detection task. As a result, we offer an Online evolving Spiking Neural Network for Unsupervised Anomaly Detection algorithm (OeSNN-UAD), which, unlike OeSNN, works in an unsupervised way and does not separate output neurons into disjoint decision classes. OeSNN-UAD uses our proposed new two-step anomaly detection method. Also, we derive new theoretical properties of neuronal model and input layer encoding of OeSNN, which enable more effective and efficient detection of anomalies in our OeSNN-UAD approach. The proposed OeSNN-UAD detector was experimentally compared with state-of-the-art unsupervised and semi-supervised detectors of anomalies in stream data from the Numenta Anomaly Benchmark and Yahoo Anomaly Datasets repositories. Our approach outperforms the other solutions provided in the literature in the case of data streams from the Numenta Anomaly Benchmark repository. Also, in the case of real data files of the Yahoo Anomaly Benchmark repository, OeSNN-UAD outperforms other selected algorithms, whereas in the case of Yahoo Anomaly Benchmark synthetic data files, it provides competitive results to the results recently reported in the literature.

1. Introduction

Unsupervised anomaly discovery in stream data is a research topic that has important practical applications. For example, an Internet system administrator may be interested in recognition of abnormally high activity on a web page potentially caused by a hacker attack. Unexpected spiking usage of a CPU unit in a computer system could be another example of anomalous behavior that may require investigation. Correct detection and classification of such anomalies may enable optimization of the performance of the computer system. However, in many cases, it is not easy to collect enough training data with labeled anomalies for supervised learning of an anomaly detector in order to use it later for identification of real anomalies in streaming data. It is thus particularly important to design anomalies detectors that can correctly detect anomalies without access to labeled training data. Moreover, since the characteristic of an input data stream may be changing, the anomaly detector should learn in an online mode.

In order to design an effective anomaly detection system, one can consider adaptation of evolving Spiking Neural Networks (eSNNs) (Kasabov, 2014; Lobo et al., 2018, 2020b; Maciąg et al., 2020) to the task. eSNN is a neural network with an evolving repository of output neurons, in which learning processes, neuronal communication and classification of data instances are based solely on transmission of spikes from input neurons to output neurons (Kasabov, 2014). The spikes increase so called post-synaptic potential values of output neurons, and directly influence the classification results. The input layer of neurons in eSNN transforms input data instances into spikes. Depending on the type of input data, the transformation can be carried out by means of the temporal encoding methods such as Step-Forward or Threshold-Based (Maciąg et al., 2019; Petro et al., 2019) or,

* Corresponding author.
E-mail addresses: pmaciag@ii.pw.edu.pl (P.S. Maciąg), mkr@ii.pw.edu.pl (M. Kryszkiewicz), r.bembenik@ii.pw.edu.pl (R. Bembenik), jesus.lopez@tecnalia.com (J. L. Lobo), javier.delser@tecnalia.com (J. Del Ser).

https://doi.org/10.1016/j.neunet.2021.02.017

alternatively, with the use of Gaussian Receptive Fields (Lobo et al., 2018). The distinctive feature of the eSNN is that its repository of output neurons evolves during the training phase based on candidate output neurons that are created for every new input data sample (Kasabov, 2015; Kasabov et al., 2013). More specifically, for each new input value, a new candidate output neuron is created and is either added to the output repository or, based on the provided similarity threshold, is merged with one of the output neurons contained in the repository.

Recently, an online variant OeSNN of eSNN was proposed for classification of stream data (Lobo et al., 2018). Contrary to the eSNN architecture, the size of the evolving repository of output neurons in OeSNN is limited. When the repository of output neurons is full and a new candidate output neuron is significantly different from all of the neurons in the repository, an oldest neuron is replaced with the new candidate output neuron. It was claimed in Lobo et al. (2018) that OeSNN is able to make fast classification of input stream data, while preserving restrictive memory limits. Considering all the positive features of eSNN and OeSNN, in this article, we offer a novel Online evolving Spiking Neural Network for Unsupervised Anomaly Detection (OeSNN-UAD) in stream data.

Our main contributions presented in this article are as follows:

- We offer a new OeSNN-UAD anomaly detector working online in an unsupervised way. It adapts the architecture of OeSNN, which also works in an online way, but, unlike OeSNN-UAD, requires supervised training. The main distinction between our detector and OeSNN lies in applying different models of an output layer and different methods of learning and input values classification. While output neurons of OeSNN are divided into separate decision classes, there is no such separation of output neurons in our approach. Rather than that, each new output neuron is assigned an output value, which is first randomly generated based on recent input values and then is updated in the course of learning of OeSNN-UAD.
- As a part of the proposed OeSNN-UAD detector, we offer a new anomaly classification method, which classifies an input value as anomalous only in the following two cases:
  1. if none of output neurons in the repository fires, or otherwise,
  2. if an error between an input value and its OeSNN-UAD prediction is greater than the average prediction error plus user-given multiplicity of the standard deviation of the recent prediction errors.

  This two-step approach to classification of an input value as anomalous or not enables more effective detection of anomalies in input stream data and to the best of our knowledge was not previously used in the literature.
- We derive the important theoretical property of the OeSNN neuronal model that shows that the values of post-synaptic potential thresholds of all output neurons are the same. This property is inherited by our OeSNN-UAD detector. The obtained result eliminates the necessity of recalculation of these thresholds when output neurons of OeSNN, as well as of OeSNN-UAD, are updated in the course of the learning process, and increases the speed of classification of input stream data. Moreover, we also prove that firing order values of input neurons do not depend on values of TS and β parameters, which were previously used in OeSNNs for input value encoding with Gaussian Receptive Fields.
- We prove experimentally that in the case of stream data from the Numenta Anomaly Benchmark repository (Ahmad et al., 2017a) as well as from the Yahoo Anomaly Datasets repository (Webscope, 2015) the proposed OeSNN-UAD detects anomalies in unsupervised way more effectively than other state-of-the-art unsupervised and semi-supervised detectors proposed in the literature.
- Eventually, we argue that the proposed OeSNN-UAD is able to make fast detection of anomalies among data stream input values and works efficiently in environments with imposed restrictive memory limits.

The paper is structured as follows. In Section 2, we overview the related work. In Section 3, we present the architecture of Online evolving Spiking Neural Networks, whose adaptation proposed by us will be then used in OeSNN-UAD. In Section 4, we provide new theoretical properties of neuronal model and input layer encoding of OeSNN, which enable more effective and efficient detection of anomalies in our OeSNN-UAD approach. In Section 5, we offer our online method to unsupervised anomaly detection in stream data OeSNN-UAD. Section 6 presents and discusses the proposed OeSNN-UAD algorithm in detail. In Section 7, we present the results of comparative experimental evaluation of the proposed OeSNN-UAD detector and state-of-the-art unsupervised and semi-supervised detectors of anomalies. We conclude our work in Section 8.

2. Related work

A number of solutions to the task of unsupervised anomaly detection in time series data was offered in the literature. The state-of-the-art algorithms for unsupervised and semi-supervised anomaly detection are:

- Numenta and NumentaTM (Ahmad et al., 2017c) — two slightly different algorithms that consist of the following modules: (i) a Hierarchical Temporal Memory (HTM) network for predicting the current value of an input stream data, (ii) an error calculation module, and (iii) an anomaly likelihood calculation module, which classifies the input value as an anomaly or not based on the likelihood of the calculated error. Both algorithms are implemented in Python and are available as a part of the Numenta Anomaly Benchmark repository. NumentaTM and Numenta differ in implementation of the HTM network and its parameters initialization.
- HTM JAVA (Hawkins & Ahmad, 2016) — a JAVA implementation of the Numenta algorithm.
- Skyline (Stanway, 2015) — an algorithm based on ensembles of several outliers' detectors, such as e.g. Grubb's test for outliers or a simple comparison of the current input value of a data stream against the deviation from the average of past values. In Skyline, a given input value of a data stream is classified as an anomaly if it is marked as anomalous by the majority of ensemble detectors. Skyline is implemented in Python and is available as a part of the Numenta Anomaly Benchmark repository.
- TwitterADVec (Kejariwal, 2015) — a method for anomaly detection based on the Seasonal Hybrid ESD (S-H-ESD) algorithm (Chandola et al., 2009). For given time series values, the S-H-ESD algorithm first calculates extreme Student deviates (Rosner, 1983) of these values and then, based on a statistical test, decides which of these values should be marked as outliers. The TwitterADVec method is currently implemented as an R language package and is a part of the Numenta Anomaly Benchmark repository.
- Yahoo EGADS (Extensible Generic Anomaly Detection System) (Laptev et al., 2015) — an algorithm consisting of the following modules: (i) a time-series modeling module,

(ii) an anomaly detection module, and (iii) an alerting module. Yahoo EGADS is able to discover three types of anomalies: outliers, sudden changepoints in values and anomalous subsequences of time series. To this end, the following three different anomaly detectors were implemented in Yahoo EGADS: (i) time series decomposition and prediction for outliers' detection, (ii) a comparison of values of current and past time windows for changepoint detection, and (iii) clustering and decomposition of time series for detection of anomalous subsequences.

- DeepAnT (Munir et al., 2019b) — a semi-supervised anomaly detection method based on either convolutional neural networks or Long–Short Term Memories networks. DeepAnT consists of a time series prediction module and an anomaly detection module. DeepAnT uses the first part of a time series as a training and validation data and detects anomalies in the remaining part of the time series.
- Bayesian Changepoint (Adams & MacKay, 2007) — an online algorithm for sudden changepoint detection in time series data by means of the Bayesian inference. This method is particularly suited to such time series data, in which it is possible to clearly separate partitions of values generated from different data distributions. The algorithm is able to detect the most recent changepoint in the current input values based on the analysis of probability distributions of time series partitions, which are created from changepoints registered in the past values.
- EXPected Similarity Estimation (EXPoSE) (Schneider et al., 2016) — an algorithm that classifies anomalies based on the deviation of an input observation from an estimated distribution of past input values.
- KNN CAD (Burnaev & Ishimtsev, 2016) — a method of anomaly detection in univariate time series data based on nearest neighbors classification. KNN CAD method first transforms time series values into its Caterpillar matrix. Such a matrix is created both for the most recent input value (which is classified as an anomaly or not) and for a sequence of past values, which are used as reference data. Next, the Non-Conformity Measure (NCM) is calculated both for the classified value and for the reference values using the created Caterpillar matrix. Eventually, the anomaly score of the classified input value is obtained by comparing its NCM with NCMs of the reference values.
- Relative Entropy (Wang et al., 2011) — a method, which uses a relative entropy metric (Kullback–Leibler divergence) of two data distributions to decide if a series of input values can be classified as anomalies.
- ContextOSE (Smirnov, 2016) — an algorithm that creates a set of contexts of time series according to the characteristics of its values. A subsequence of most recent input values is classified as anomalous if its context differs significantly from the contexts of past subsequences of values.

The above presented methods and algorithms are directly compared to our approach in the experimental evaluation provided in Section 7. In addition to these approaches, other non-online unsupervised methods of anomaly detection in time series data were proposed. In Munir et al. (2019a), an unsupervised detector of anomalies in time series, which combines the ARIMA (Auto-regressive Moving Average) method and convolutional neural networks, was provided. Ergen and Kozat (2019) introduced an unsupervised anomaly detection method integrating Long–Short Term Memory networks and One-class Support Vector Machines. Yet another non-online unsupervised approach to anomaly detection was offered in Lin and Su (2019). It uses a sliding window data stream sampling algorithm based on data elements to sample the sensor network data stream. The sampling result is used as the sample set of the clustering algorithm to detect anomalies in the data stream.

A supervised eSNN approach to anomaly detection, called HESADM, was proposed in Demertzis and Iliadis (2014). In this approach, the eSNN network is first taught based on a training part of data, and then is used for detection of anomalies in the remaining part of data. In Demertzis et al. (2019), a semi-supervised approach to anomaly detection with one-class eSNN was offered and dedicated to intrusion detection systems. Contrary to the approaches presented in Demertzis and Iliadis (2014), Demertzis et al. (2019), OeSNN-UAD approach offered in this work learns to recognize anomalies in an unsupervised mode, in which anomaly labels are not assigned to data samples.

The distinguishing feature of our proposed OeSNN-UAD anomaly detector compared to the above-mentioned methods and algorithms is that it is the only eSNN-based detector operating both in an online and unsupervised way.

The anomaly detectors proposed in Zhang et al. (2019) and Bovenzi et al. (2011) are also online unsupervised ones. They detect whether a current data stream value is an anomaly or not based only on a given number of recent input values.

- The SPS algorithm, presented in Bovenzi et al. (2011), uses a window of recent values to calculate statistics that enable dynamic determination of a lower bound and an upper bound on the expected value of the current data point. The real current value is defined as anomalous if it is outside the current bounds. The SPS algorithm was found in Zoppi et al. (2019) as inferior to non-online unsupervised algorithms kMeans and kHOBS on all datasets tested there (KDD Cup 99 (1999), NSL-KDD (2009), ISCX (2012), UNSW-NB15 (2015)).
- The algorithm offered in Zhang et al. (2019) splits the window of recent values into disjoint subwindows. A vector (PDD) of Probability Density-based Descriptors is calculated for each subwindow. Checking if the current value is an anomaly or not is based on the distances between PDDs of each of two consecutive subwindows. The algorithm was tested in Zhang et al. (2019) on a number of data streams from the Numenta Anomaly Benchmark repository and was found an effective anomaly detector there. In the experimental evaluation provided in Section 7, we compare its quality with the quality of OeSNN-UAD.

An important feature distinguishing OeSNN-UAD from the two above mentioned algorithms is that the predictions made by OeSNN-UAD are not only based on the contents of a sliding window of recent input values, but also on the state of an evolving spiking neural network, which plays a role of an adaptable memory of historical input values.

Overview of anomaly detection techniques for stream data can be found in Chalapathy and Chawla (2019), Chandola et al. (2009, 2012, 2008), Izakian and Pedrycz (2013), Izakian and Pedrycz (2014), Kwon et al. (2017), Pimentel et al. (2014) and Ergen and Kozat (2019).

3. Online evolving Spiking Neural Networks

Our approach to unsupervised anomaly detection adapts the architecture of OeSNN networks previously introduced in Lobo et al. (2018). Thus, the presentation of our approach is preceded with an overview of the architecture of OeSNN network and its classification principles. Input values encoding method and output neuronal model used in that type of network are given as well.

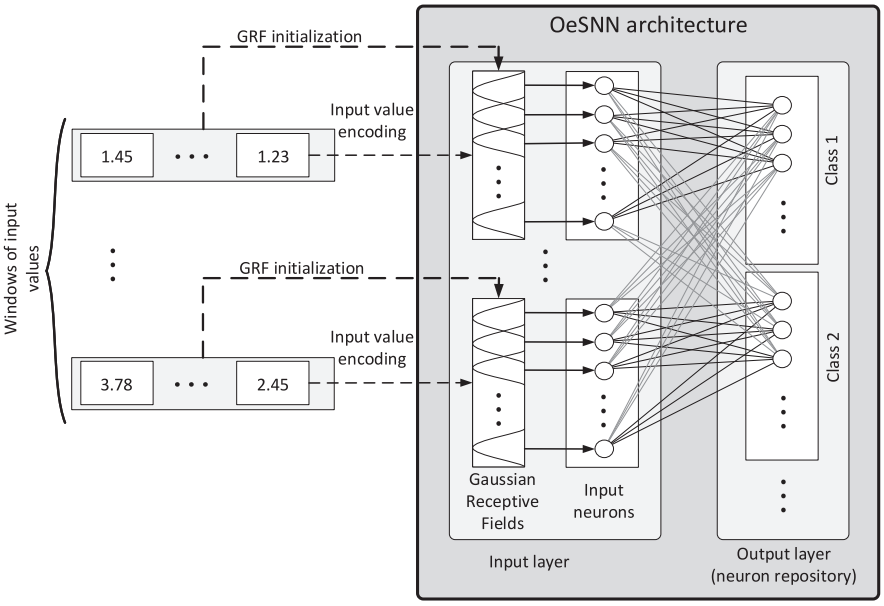

Fig. 1. OeSNN architecture introduced in Lobo et al. (2018).

3.1. Architecture of Online evolving Spiking Neural Networks

OeSNN networks were designed to perform classification tasks in streaming data. OeSNN extends eSNN architecture, which was designed for classification of batch data with separate training and testing parts (Kasabov, 2007). Both eSNN and OeSNN architectures consist of input and output layers, however the number of output neurons in OeSNN is limited, while in eSNN it is unlimited. The limitation on the number of neurons in OeSNN is motivated by requirements for the classification of data stream values, where usually a large number of input data is processed and strict memory requirements exist. Both eSNN and OeSNN create a candidate output neuron for each new data sample and either insert it to the output repository when the candidate output neuron is significantly different from all output neurons in the repository or, otherwise, merge it with the most similar output neuron in the repository. However, OeSNN unlike eSNN, controls the size of the repository and when the repository is full, OeSNN inserts the candidate output neuron into it only after the removal of an oldest output neuron from the repository. The input layer of OeSNN consists of so-called Gaussian Receptive Fields (GRFs) and input neurons. The aim of GRFs is to encode input values into firing times and firing order values of input neurons (Lobo et al., 2020a).[1] The firing order values of input neurons are further used to initialize synapses weights between each input neuron and a candidate output neuron and then to classify an input data sample. The classification in OeSNN is performed by calculation of so-called Post-synaptic Potential (PSP) values of output neurons in the output repository. For each input sample to be classified, first GRFs are initialized, and then they are used for calculation of order values of input neurons. Next, given that encoding the PSP values of output neurons are updated. A decision class assigned to the output neuron whose PSP value first exceeds PSP threshold is returned as a decision class of the input sample. The architecture of OeSNN is presented in Fig. 1.

OeSNN network enables classification of both univariate as well as multivariate time series data. The OeSNN classifies each new input sample of data, which consists of only the newest value of each time series. However, the encoding of an input sample is carried out by the input layer using the contents of windows with predefined number of recent values of respective time series.

In the following subsections, we overview the input values encoding technique used in OeSNN as well as its output neuronal model, as the OeSNN-UAD approach proposed in this article adopts both of these principles in its learning and classification working schema.

3.2. Input layer of the OeSNN network

Since our approach to anomaly detection, which is presented in the following sections, operates on a single time series (that is, a single data stream of values), in this subsection we recall the encoding technique of OeSNN for a single time series, as it was presented in Lobo et al. (2018). Please note however, that the notions presented here can be easily extended for more than one time series.

The OeSNN network consists solely of an input layer and an output layer. The input layer contains a fixed number of input neurons and their *Gaussian Receptive Fields* (*GRFs*). The output layer, called an *evolving repository*, contains output neurons, whose maximal number is limited. The set of input neurons is denoted by **NI**, while the set of output neurons by **NO**. The number of input neurons is determined by user-given parameter NI_{size}, whereas the maximal number of output neurons is given by NO_{size}, which is also a user-specified parameter value. Each input neuron is linked by a synapse to each output neuron. Let **X** denote an input stream of values to be classified and x_t denote the newest value of that stream, which will be subject to classification.

By $\mathcal{W}$ we denote a window containing the newest value x_t of data stream **X**, as well as previous $\mathcal{W}_{size} - 1$ values of that data stream. $\mathcal{W}_{size}$ is a user given parameter, which denotes the *size of window* $\mathcal{W}$. Clearly, $\mathcal{W}$ contains $[x_{t-(\mathcal{W}_{size}-1)}, x_{t-(\mathcal{W}_{size}-2)}, \ldots, x_t]$

[1] Several published studies have widely agreed on the suitability of the Gaussian Receptive Fields encoding (and its variants) for streaming scenarios (please see, for example, (Lobo et al., 2020a) or (Petro et al., 2019)). Other alternatives include temporal encoding techniques, such as *Threshold-based Representation algorithm*, *Bens Spiker algorithm* or *Moving Window algorithm*, which were reviewed e.g. in Petro et al. (2019) and in Maciąg et al. (2019). The temporal encoding techniques can be especially useful in the case of time series data with significant changes of input values, such as EEG or fMRI time series data. Recently, a new encoding technique was proposed in Maciąg et al. (2020). This encoding technique directly calculates firing order values of input neurons (without calculation of exact firing times of input neurons).

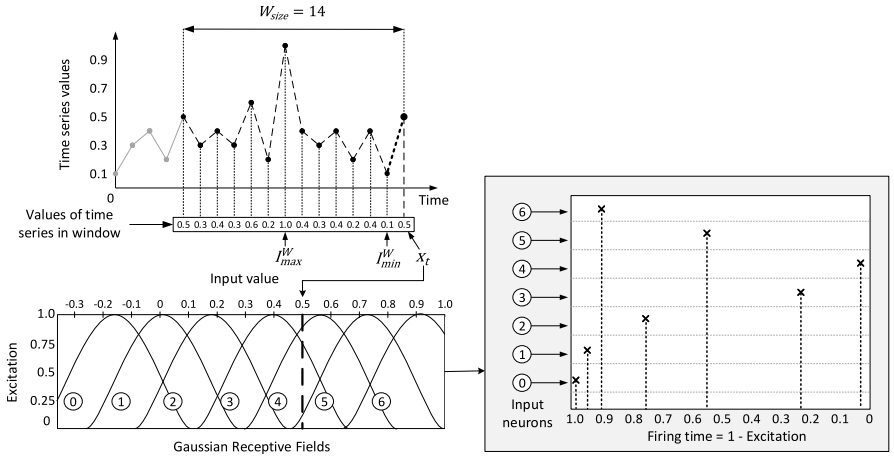

Fig. 2. The encoding and the input layer of the proposed network architecture. $\mathcal{W}_{size}$ denotes the size of window $\mathcal{W}$. Current values in $\mathcal{W}$ are used to construct GRFs, while only x_t value is encoded and propagated to neurons in the output repository **NO**.

values of data stream. Two values in the window denoted by $I^{\mathcal{W}}_{min}$ and $I^{\mathcal{W}}_{max}$ play an important role in the classification of the input value x_t. $I^{\mathcal{W}}_{min}$ is defined as the minimal value in window $\mathcal{W}$, whereas $I^{\mathcal{W}}_{max}$ is defined as the maximal value in $\mathcal{W}$. The two values are used to initialize NI_{size} distinct GRFs *excitation* functions — one function per one input neuron. The values of excitation functions obtained for x_t are used to calculate *firing times* and *firing order values* of input neurons, and thus influence the classification result.

The *excitation function* of *j-th* GRF, where $j = 0 \ldots, NI_{size} - 1$, for input value x_t is denoted by $Exc^{GRF}_j(x_t)$ and is defined as the following Gaussian function:

$$Exc^{GRF}_j(x_t) = \exp\left(-\frac{1}{2}\left(\frac{x_t - \mu^{GRF}_j}{\sigma^{GRF}_j}\right)^2\right), \tag{1}$$

where μ^{GRF}_j stands for *j-th* GRF's *mean* which is expressed by Eq. (2), and σ^{GRF}_j stands for *j-th* GRF's *standard deviation* which is expressed by Eq. (3):

$$\mu^{GRF}_j = I^{\mathcal{W}}_{min} + \frac{2j-3}{2}\left(\frac{I^{\mathcal{W}}_{max} - I^{\mathcal{W}}_{min}}{NI_{size} - 2}\right), \tag{2}$$

$$\sigma^{GRF}_j = \frac{1}{\beta}\left(\frac{I^{\mathcal{W}}_{max} - I^{\mathcal{W}}_{min}}{NI_{size} - 2}\right), \text{where } \beta \in [1, 2]. \tag{3}$$

The parameter β that occurs in the equation defining σ^{GRF}_j is used to control the degree to which Gaussian Random Fields overlap. μ^{GRF}_j is also called a *center value of jth GRF*, while σ^{GRF}_j is also called its *width*.

Please note that the excitation function $Exc^{GRF}_j(x_t)$ takes greatest values for those GRFs whose center values are closest to x_t. Input neurons associated with such GRFs will have the greatest impact on prediction results. In an approach in which, e.g. for efficiency reasons, only some of input neurons should be used for prediction rather than all, a subset of input neurons related to GRFs with highest excitation values will be fired. A *firing time function* defined in Eq. (4) assigns earlier firing time values to input neurons associated with GRFs having higher excitation values.

The *firing time function* for input neuron n_j, where $j = 0 \ldots, NI_{size} - 1$, is denoted by $T_{n_j}(x_t)$, and is defined as follows:

$$T_{n_j}(x_t) = TS \cdot \left(1 - Exc^{GRF}_j(x_t)\right), \tag{4}$$

where TS is a user-given basic synchronization time of firings of input neurons in OeSNN and $TS > 0$.

The firing times of input neurons imply their distinct *firing order values*; namely, input neurons with shorter firing times are assigned smaller firing order values, which are integers in $\{0, \ldots NI_{size} - 1\}$. The firing order value of input neuron n_j is denoted by $order(n_j)$.

Example 1. Given the window of input values and the input layer as shown in Fig. 2, let us consider an example of encoding of value $x_t = 0.5$ into excitation values of GRFs and then into firing times and firing order values of input neurons associated with corresponding GRFs. We assume that the size of window $\mathcal{W}$ is $\mathcal{W}_{size} = 14$ and the GRFs parameters $I^{\mathcal{W}}_{min}$ and $I^{\mathcal{W}}_{max}$ are equal to 0.1 and 1.0, respectively. The input layer contains seven neurons each of which is associated with one distinct Gaussian Random Field. The excitation values of GRFs are determined with GRF overlapping parameter β = 1.0 and the firing times of input neurons are calculated with synchronization time TS equal to 1.0. The resulting encoding of input value $x_t = 0.5$ is presented beneath:

- $Exc^{GRF}_0(0.5) = 0.001 \rightarrow T_{n_0}(0.5) = 0.999 \rightarrow order(n_0) = 6$,
- $Exc^{GRF}_1(0.5) = 0.024 \rightarrow T_{n_1}(0.5) = 0.976 \rightarrow order(n_1) = 5$,
- $Exc^{GRF}_2(0.5) = 0.227 \rightarrow T_{n_2}(0.5) = 0.773 \rightarrow order(n_2) = 3$,
- $Exc^{GRF}_3(0.5) = 0.770 \rightarrow T_{n_3}(0.5) = 0.230 \rightarrow order(n_3) = 1$,
- $Exc^{GRF}_4(0.5) = 0.962 \rightarrow T_{n_4}(0.5) = 0.038 \rightarrow order(n_4) = 0$,
- $Exc^{GRF}_5(0.5) = 0.442 \rightarrow T_{n_5}(0.5) = 0.558 \rightarrow order(n_5) = 2$,
- $Exc^{GRF}_6(0.5) = 0.074 \rightarrow T_{n_6}(0.5) = 0.926 \rightarrow order(n_6) = 4$.

3.3. Neuronal model of output neurons and network learning

The distinctive feature of OeSNN is the creation of a candidate output neuron for each value x_t of the input data stream. When a new candidate output neuron n_c is created for x_t, weights of its synapses are initialized according to input neurons' firing order values obtained as a result of the x_t encoding. The initial weights of synapses between each input neuron n_j in **NI** and the candidate output neuron n_c are calculated according to Eq. (5):

$$w_{n_j n_c} = mod^{order(n_j)}, \tag{5}$$

where *mod* is a user-given modulation factor within range (0, 1) and $order(n_j)$ is n_j's firing order value obtained as a result of the x_t encoding.

Vector $[w_{n_0 n_c}, \ldots, w_{n_{NI_{size}-1} n_c}]$ of weights of synapses connecting the input neurons in **NI** with candidate output neuron n_c will be denoted by $\mathbf{w}_{n_c}$.

A candidate output neuron, say n_c, is characterized also by two additional attributes: the maximal post-synaptic potential $PSP^{max}_{n_c}$ and the post-synaptic potential threshold γ_{n_c}. The definition of $PSP^{max}_{n_c}$ is given in Eq. (6):

$$PSP^{max}_{n_c} = \sum_{j=0}^{NI_{size}-1} w_{n_j n_c} \cdot mod^{order(n_j)}, \tag{6}$$

where $order(n_j)$ is n_j's firing order value obtained as a result of the x_t encoding.

Property 1 follows immediately from Eqs. (5) and (6).

Property 1. $PSP^{max}_{n_c} = \sum_{j=0}^{NI_{size}-1} mod^{2 \cdot order(n_j)}$.

The definition of the post-synaptic potential threshold γ_{n_c} is given in Eq. (7):

$$\gamma_{n_c} = PSP^{max}_{n_c} \cdot C, \tag{7}$$

where C is a user fixed value from the interval (0, 1).

Example 2. Let us consider again Example 1, which illustrates encoding of input value $x_t = 0.5$ with seven input neurons, as presented in Fig. 2. We will show now how synapses weights of a new candidate output neuron are calculated given neuronal model parameters: $mod = 0.5$ and $C = 0.8$. As calculated in Example 1: order(4) = 0, order(3) = 1, order(5) = 2, order(2) = 3, order(6) = 4, order(1) = 5, order(0) = 6. In consequence, the weights of synapses between input neurons and candidate output neuron n_c would be initialized as follows:

- $w_{n_4 n_c} = 0.5^0 = 1$,
- $w_{n_3 n_c} = 0.5^1 = 0.5$,
- $w_{n_5 n_c} = 0.5^2 = 0.25$,
- $w_{n_2 n_c} = 0.5^3 = 0.125$,
- $w_{n_6 n_c} = 0.5^4 = 0.0625$,
- $w_{n_1 n_c} = 0.5^5 = 0.03125$,
- $w_{n_0 n_c} = 0.5^6 = 0.015625$.

Given these weights of synapses of candidate output neuron n_c, the value of its maximal post-synaptic potential $PSP^{max}_{n_c}$ calculated according to Eq. (6) is $1^2 + 0.5^2 + 0.25^2 + 0.125^2 + 0.0625^2 + 0.03125^2 + 0.015625^2 = 1.333251953$.

In OeSNN, each candidate output neuron, say n_i, is either added to the repository **NO** or is merged with some output neuron in **NO**. In fact, each output neuron in **NO** is either an extended copy of a candidate output neuron or is an aggregation of a number of candidate output neurons. In comparison with candidate output neurons, output neurons in **NO** are characterized by one more attribute — *update counter* M_{n_i}, which provides the information from how many candidate output neurons n_i was created. If $M_{n_i} = 1$, then the values of the remaining attributes of n_i are the same as of a former candidate output neuron. Now, each time when an output neuron n_i built from M_{n_i} former candidate output neurons is merged with a current candidate output neuron n_c, weight $w_{n_j n_i}$ of the synapse between output neuron n_i and each input neuron n_j is recalculated as shown in Eq. (8), $PSP^{max}_{n_i}$ is recalculated as shown in Eq. (9) and γ_{n_i} is recalculated according to Eq. (10):

$$w_{n_j n_i} \leftarrow \frac{w_{n_j n_c} + M_{n_i} \cdot w_{n_j n_i}}{M_{n_i} + 1}, \tag{8}$$

$$PSP^{max}_{n_i} \leftarrow \frac{PSP^{max}_{n_c} + M_{n_i} \cdot PSP^{max}_{n_i}}{M_{n_i} + 1}, \tag{9}$$

$$\gamma_{n_i} \leftarrow \frac{\gamma_{n_c} + M_{n_i} \cdot \gamma_{n_i}}{M_{n_i} + 1}. \tag{10}$$

In addition, M_{n_i} is increased by 1 to reflect the fact that one more candidate output neuron was used to obtain an updated version of output neuron n_i.

In the remainder of the article, vector $[w_{n_0 n_i}, \ldots, w_{n_{NI_{size}-1} n_i}]$ of weights of synapses connecting the input neurons in **NI** with output neuron n_i in **NO** will be denoted by $\mathbf{w}_{n_i}$.

In the OeSNN approach, which uses a simplified Leaky Integrate and Fire (LIF) neuronal model of output neurons, output neuron n_i fires for x_t only when its, so called, *post-synaptic potential* is not less than its post-synaptic potential threshold γ_{n_i} (Lobo et al., 2018).

The *post-synaptic potential* of output neuron n_i in repository **NO** for input value x_t is denoted by PSP_{n_i} and is defined by Eq. (11):

$$PSP_{n_i} = \sum_{j=0}^{NI_{size}-1} w_{n_j n_i} \cdot mod^{order(n_j)}, \tag{11}$$

where $w_{n_j n_i}$ represents the weight of the synapse linking input neuron $n_j \in$ **NI** with output neuron $n_i \in$ **NO**, and $order(n_j)$ is n_j's firing order value obtained as a result of the x_t encoding.

Classification of an input value, say x_t, in OeSNN is performed based on output neurons' post-synaptic potentials obtained for x_t. As presented in Fig. 1, output neurons of the output layer are organized into decision classes. Input value x_t is assigned the decision class of the first output neuron whose post-synaptic potential value exceeded its own post-synaptic potential threshold.

4. Properties of an OeSNN neuronal model

In this section, we derive theoretical properties of input and output layers of the OeSNN neuronal model based on definitions provided in Sections 3.2 and 3.3, respectively:

- The obtained properties of input layer show that firing order values of input neurons depend neither on values of Gaussian Random Fields overlapping parameter β nor on values of basic synchronization time TS of firing of input neurons, and, in consequence, the selection of output neurons to fires does not depend on values of these parameters.
- The derived properties related to output neurons show that for any recent input value x_t, the values of maximal post-synaptic potential of all output neurons are the same and equal to $\sum_{k=0}^{NI_{size}-1} mod^{2k} = \frac{1-mod^{2 \cdot NI_{size}}}{1-mod^2}$ and, in consequence, the values of post-synaptic potential thresholds of all output neurons are the same and equal to $C \cdot \sum_{k=0}^{NI_{size}-1} mod^{2k} = C \cdot \frac{1-mod^{2 \cdot NI_{size}}}{1-mod^2}$. The latter finding enables calculation of the values of the two parameters of output neurons only once; namely, at the beginning of the whole detection anomaly process rather than for each input value of the data stream. As a result, the anomaly detection becomes faster.

In fact, the obtained properties are used in our proposed OeSNN-UAD model for efficient detection of anomalies in a data stream (please see the next two sections).

Let us start the presentation of the obtained theoretical results with Proposition 1, which concerns properties of GRFs and input neurons.

Proposition 1. *For any most recent value x_t of window $\mathcal{W}$ and any input neurons $j_1, j_2 \in \{0, \ldots, NI_{size} - 1\}$, the following hold:*

(i) $\sigma^{GRF}_{j_1} = \sigma^{GRF}_{j_2}$ *for any values of parameters β and TS.*

(ii) $\mu^{GRF}_{j_1}$ *does not depend on values of parameters β and TS.*

(iii) The truth value of statement "$Exc^{GRF}_{j_1}(x_t)$ is greater than or equal to $Exc^{GRF}_{j_2}(x_t)$" does not depend on values of parameters β and TS.

(iv) *The following statements are equivalent for any values of parameters β and TS:*

- $Exc_{j_1}^{GRF}(x_t) \geq Exc_{j_2}^{GRF}(x_t)$.
- $T_{n_{j_1}}(x_t) \leq T_{n_{j_2}}(x_t)$.
- $order(n_{j_1}) \leq order(n_{j_2})$.

(v) *Firing order values of input neurons do not depend on values of parameters β and TS.*

Proof.

Ad (i). Follows trivially from Eq. (3).
Ad (ii). Follows trivially from Eq. (2).
Ad (iii). Follows from Eq. (1), Proposition 1.(i) and Proposition 1.(ii).
Ad (iv). By Eq. (1), $Exc_j^{GRF}(x_t)$ takes values from interval [0, 1], and by definition of the firing time function for input neuron (see Eq. (4)), parameter *TS* may take only a value greater than 0. Hence:

$$Exc_{j_1}^{GRF}(x_t) \geq Exc_{j_2}^{GRF}(x_t) \iff$$
$$1 - Exc_{j_1}^{GRF}(x_t) \leq 1 - Exc_{j_2}^{GRF}(x_t) \iff$$
$$TS \cdot \left(1 - Exc_{j_1}^{GRF}(x_t)\right) \leq TS \cdot \left(1 - Exc_{j_2}^{GRF}(x_t)\right) \iff$$
$$T_{n_{j_1}}(x_t) \leq T_{n_{j_2}}(x_t) \iff$$
$$order(n_{j_1}) \leq order(n_{j_2}).$$

Ad (v). Follows immediately from Proposition 1.(iii) and Proposition 1.(iv). ■

In Lemma 1, we provide properties of candidate output neurons.

Lemma 1. *For each candidate output neuron n_c, the following hold:*

(i) *Vector $\mathbf{w}_{n_c} = [w_{n_0 n_c}, \ldots, w_{n_{NI_{size}-1} n_c}]$ of synapses weights of candidate output neuron n_c is a permutation of vector $[mod^0, mod^1, \ldots, mod^{NI_{size}-1}]$.*
(ii) *The sum of all synapses weights of n_c equals $\sum_{k=0}^{NI_{size}-1} mod^k$.*
(iii) *n_c's maximal post-synaptic potential $PSP_{n_c}^{max} = \sum_{k=0}^{NI_{size}-1} mod^{2k}$.*
(iv) *n_c's post-synaptic potential threshold $\gamma_{n_c} = C \cdot \sum_{k=0}^{NI_{size}-1} mod^{2k}$.*

Proof. Let n_c be a candidate output neuron.

Ad (i). By Eq. (5), the weight of the synapse linking n_c with input neuron n_j equals $mod^{order(n_j)}$, where $order(n_j)$ is a firing order value of n_j. Taking into account that at any time firing order values of input neurons are integers from the set {0, 1, …, $NI_{size} - 1$} and are distinct for each input neuron, vector $\mathbf{w}_{n_c} = [w_{n_0 n_c}, \ldots, w_{n_{NI_{size}-1} n_c}]$ of synapses weights of candidate output neuron n_c is a permutation of vector $[mod^0, mod^1, \ldots, mod^{NI_{size}-1}]$.
Ad (ii). By Lemma 1.(i), the sum of all synapses weights of candidate output neuron n_c equals $\sum_{k=0}^{NI_{size}-1} mod^k$.
Ad (iii). By Property 1, $PSP_{n_c}^{max}$ is the sum of the squares of all synapses weights of candidate output neuron n_c. Hence, and by Lemma 1.(i), $PSP_{n_c}^{max} = \sum_{k=0}^{NI_{size}-1} mod^{2k}$.
Ad (iv). By Eq. (7) and Lemma 1.(iii), threshold $\gamma_{n_c} = C \cdot \sum_{k=0}^{NI_{size}-1} mod^{2k}$. ■

In the remainder of Section 4, we focus on properties of output neurons in repository **NO**. The derivation of these properties was based on the fact that each output neuron is in fact constructed from one or more candidate output neurons.

Theorem 1. *Let n_i be an output neuron n_i in repository **NO** that was constructed from M_{n_i}, where $M_{n_i} \geq 1$, candidate output neurons: n_{c_1}, n_{c_2}, …, $n_{c_{M_{n_i}}}$. The following hold for output neuron n_i:*

(i) *Vector $\mathbf{w}_{n_i} = [w_{n_0 n_i}, \ldots, w_{n_{NI_{size}-1} n_i}]$ of synapses weights of n_i is the average of the vectors of synapses weights of candidate output neurons n_{c_1}, n_{c_2}, …, $n_{c_{M_{n_i}}}$; that is,*

$$\mathbf{w}_{n_i} = \left[\frac{\sum_{l=1}^{M_{n_i}} w_{n_0 n_{c_l}}}{M_{n_i}}, \ldots, \frac{\sum_{l=1}^{M_{n_i}} w_{n_{NI_{size}-1} n_{c_l}}}{M_{n_i}}\right].$$

(ii) *The sum of synaptic weights of n_i equals $\sum_{k=0}^{NI_{size}-1} mod^k$.*
(iii) *n_i's maximal post-synaptic potential $PSP_{n_i}^{max} = \sum_{k=0}^{NI_{size}-1} mod^{2k}$.*
(iv) *n_i's post-synaptic potential threshold $\gamma_{n_i} = C \cdot \sum_{k=0}^{NI_{size}-1} mod^{2k}$.*

Proof.

Ad (i). It follows from Eq. (8) that the weight $w_{n_j n_i}$ of the synapse linking output neuron n_i with input neuron n_j is the average of the weights of synapses linking candidate output neurons n_{c_1}, n_{c_2}, …, $n_{c_{M_{n_i}}}$ with input neuron n_j; that is, $w_{n_j n_i} = \frac{\sum_{l=1}^{M_{n_i}} w_{n_j n_{c_l}}}{M_{n_i}}$. Hence, $\mathbf{w}_{n_i} = [w_{n_0 n_i}, \ldots, w_{n_{NI_{size}-1} n_i}] = \left[\frac{\sum_{l=1}^{M_{n_i}} w_{n_0 n_{c_l}}}{M_{n_i}}, \ldots, \frac{\sum_{l=1}^{M_{n_i}} w_{n_{NI_{size}-1} n_{c_l}}}{M_{n_i}}\right]$.
Ad (ii). By Theorem 1.(i) and Lemma 1.(ii), the sum of synapses weights of output neuron n_i is equal to $\sum_{k=0}^{NI_{size}-1} w_{n_k n_i} = \sum_{k=0}^{NI_{size}-1} \left(\frac{\sum_{l=1}^{M_{n_i}} w_{n_k n_{c_l}}}{M_{n_i}}\right) = \frac{1}{M_{n_i}} \sum_{l=1}^{M_{n_i}} \left(\sum_{k=0}^{NI_{size}-1} w_{n_k n_{c_l}}\right) = \frac{1}{M_{n_i}} \sum_{l=1}^{M_{n_i}} \left(\sum_{k=0}^{NI_{size}-1} mod^k\right) = \sum_{k=0}^{NI_{size}-1} mod^k$.
Ad (iii). It follows from Eq. (9) that PSP^{max} of output neuron n_i is the average of PSP^{max} of candidate output neurons n_{c_1}, n_{c_2}, …, $n_{c_{M_{n_i}}}$. Hence, and by Lemma 1.(iii), $PSP_{n_i}^{max} = \frac{1}{M_{n_i}} \sum_{l=1}^{M_{n_i}} (PSP_{n_{i_l}}^{max}) = \sum_{k=0}^{NI_{size}-1} mod^{2k}$.
Ad (iv). It follows from Eq. (10) that γ_{n_i} is the average of γ thresholds of candidate output neurons $n_{c_1}, n_{c_2}, \ldots, n_{c_{M_{n_i}}}$. Hence, and by Lemma 1.(iv), $\gamma_{n_i} = \frac{1}{M_{n_i}} \sum_{l=1}^{M_{n_i}} (\gamma_{n_{i_l}}) = \frac{1}{M_{n_i}} \sum_{l=1}^{M_{n_i}} (C \cdot PSP_{n_{i_l}}^{max}) = C \cdot \sum_{k=0}^{NI_{size}-1} mod^{2k}$. ■

Corollary 1 follows immediately from Theorem 1 and the fact that $\sum_{k=0}^{NI_{size}-1} mod^k$ and $\sum_{k=0}^{NI_{size}-1} mod^{2k}$ are sums of NI_{size} consecutive elements of geometric series.

Corollary 1. *For each output neuron $n_i \in$ **NO**, the following hold:*

(i) *the sum of synaptic weights of n_i equals $\frac{1-mod^{NI_{size}}}{1-mod}$,*
(ii) *n_i's maximal post-synaptic potential $PSP_{n_i}^{max} = \frac{1-mod^{2 \cdot NI_{size}}}{1-mod^2}$,*
(iii) *n_i's post-synaptic potential threshold $\gamma_{n_i} = C \cdot \frac{1-mod^{2 \cdot NI_{size}}}{1-mod^2}$.*

As follows from Lemma 1, Theorem 1 and Corollary 1, all candidate output neurons and output neurons in **NO** have the same values of the sum of their synaptic weights, their maximal post-synaptic potentials, and their maximal post-synaptic potential thresholds, respectively. The first two attributes characterizing (candidate) output neurons depend only on the number of input neurons NI_{size} and the value of parameter *mod*, while their third attribute depends also on the value of parameter *C*. The property related to post-synaptic potential thresholds will be used in our proposed algorithm for detecting anomalies.

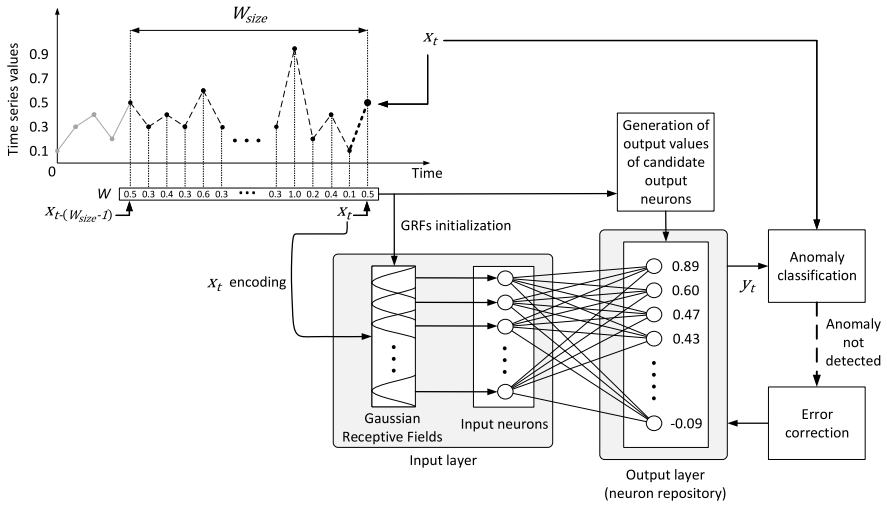

Fig. 3. The proposed OeSNN-UAD architecture.

5. OeSNN-UAD — the proposed anomaly detection model based on Online evolving Spiking Neural Networks

In this section, we offer our online OeSNN-UAD approach to unsupervised anomaly detection in stream data. We strove to design it in such a way so that the following postulates were fulfilled:

- Whenever possible, each new input value of a data stream should be correctly classified as either anomalous or non-anomalous.
- Each new input value should be used to train OeSNN-UAD for better future classification of input values.

The following subsections present the architecture and working principles of OeSNN-UAD. A step by step presentation of the OeSNN-UAD algorithm is given in Section 6.

5.1. The architecture of OeSNN-UAD

The proposed architecture of OeSNN-UAD, which is presented in Fig. 3, consists of a modified version of OeSNN and the following three new modules: *Generation of output values of candidate output neurons*, *Anomaly classification* and *Value correction*.

As it can be noted in Fig. 3, the output layer of our adapted version of OeSNN network consists of output neurons, which, unlike in OeSNN, are assigned output values, rather than decision classes. In OeSNN-UAD, an output value of each candidate output neuron is first randomly taken from a normal distribution, which is created based on values of the average and standard deviation of input values of window $\mathcal{W}$ and then, in the course of learning of OeSNN-UAD, such candidate is used to update the repository of output neurons. The output value of an output neuron may be modified in the course of learning of the network in up to two possible ways:

- it may be corrected in order to be better adjusted to a current input value of the data stream,
- it may be updated with the output value of a current candidate output neuron.

The main idea behind this approach is to store the output neurons whose output values correspond to previous non-anomalous values in the output repository **NO** of OeSNN-UAD. In consequence, when non-anomalous input value x_t occurs in a data stream, network prediction value y_t is expected to be similar to value x_t, while when anomalous input value x_t occurs, network prediction value y_t is expected to be significantly different from x_t.

The OeSNN-UAD anomaly detector works in two phases: in the anomaly detection phase and in the learning phase, which are performed for each input value x_t of the data stream:

1. In the anomaly detection phase, window $\mathcal{W}$ is updated with value x_t and GRFs of input neurons are initialized. The value x_t of $\mathcal{W}$ is used to calculate firing times and firing orders of input neurons in **NI**. Next, the input value x_t is classified as anomalous or not in the following steps. First, output neuron $n_f \in$ **NO** that fired as first is obtained. If none of output neurons fired, then input value x_t is immediately classified as an anomaly. Otherwise, the output value of the output neuron that fired as first is reported as network prediction value y_t for input value x_t. Finally, the *Anomaly classification* module classifies input value x_t as anomalous or not using a prediction error being the absolute difference between x_t and y_t and a threshold value calculated based on errors of recent $\mathcal{W}_{size}$ prediction values.
2. In the network learning phase, new candidate output neuron n_c corresponding to value x_t is created and initialized. The initialization procedure of n_c is performed in three steps: first, the synapses linking n_c with all input neurons in **NI** are created and their weights are calculated according to Eq. (5). Next, *update time* τ_{n_c} of the candidate output neuron is set to value t. Finally, the *Generation of values of candidate output neurons* module assigns initial output value v_{n_c} to the candidate output neuron and update counter M_{n_c} is set to 1. Additionally, if x_t is not classified as anomalous, then the generated initial output value v_{n_c} of the candidate output neuron is corrected by the *Value correction* module.
   After initialization of n_c, it is used to update repository **NO** of OeSNN-UAD in a similar (but not identical) way as OeSNN repository is updated. n_c is merged with the most similar output neuron n_s already present in **NO** for which the Euclidean distance between vectors of synapses weights $\mathbf{w}_{n_c}$ and $\mathbf{w}_{n_s}$ is minimal and less than or equal to the user given threshold *sim*, provided such an output neuron exists. If so, the vector of synapses weights, output value, update time and update counter of n_s are modified according to the formulae given in Eq. (12):

$$\mathbf{w}_{n_s} \leftarrow (\mathbf{w}_{n_s} \cdot M_{n_s} + \mathbf{w}_{n_c})/(M_{n_s} + 1),$$
$$v_{n_s} \leftarrow (v_{n_s} \cdot M_{n_s} + v_{n_c})/(M_{n_s} + 1),$$

$$\tau_{n_s} \leftarrow (\tau_{n_s} \cdot M_{n_s} + \tau_{n_c})/(M_{n_s} + 1),$$
$$M_{n_s} \leftarrow M_{n_s} + 1. \quad (12)$$

Otherwise, if current size of **NO** is less than NO_{size}, then n_c is simply added to **NO**. However, if **NO** is full, then n_c replaces the output neuron in **NO** whose update time is minimal.

More detailed description of the modules: *Generation of output values of candidate output neurons*, *Anomaly classification* and *Value correction* specific to OeSNN-UAD is provided in the following subsections.

5.2. Anomaly classification

Given input value x_t of the data stream and its prediction y_t made by OeSNN-UAD, the aim of the *Anomaly classification* module (see Fig. 3) is to decide whether x_t should be classified as an anomaly or not. The approaches proposed in the literature, such as those presented in Malhotra et al. (2015), Munir et al. (2019b) and Munir et al. (2019a), simply calculate an error between the predicted and the real value, compare it against a fixed threshold value and decide if an anomaly occurred. Alternatively, a window of recent predictions errors is used to construct a statistical distribution and obtain the probability of actual prediction error for value x_t (Ahmad et al., 2017c). If the probability of the actual prediction error is low, the observation is classified as an anomaly. Carrera et al. (2019) take a different approach and propose to adapt an error threshold for anomaly classification according to the changing characteristic of the stochastic process generating input data.

In our approach, a vector of error values calculated between predicted values (network responses) and input values of window $\mathcal{W}$ is used to decide if observation x_t should be classified as anomalous or not. The error e_t between x_t and its prediction y_t is calculated as the absolute difference between these two values: $e_t = |x_t - y_t|$. Let $\mathbf{e}$ be a subset of set $\{e_{t-(\mathcal{W}_{size}-1)}, \ldots, e_{t-1}\}$ of those $\mathcal{W}_{size}$ prediction error values that were obtained for input values classified as non-anomalous. If $\mathbf{e}$ is not empty, then the mean $\overline{x}_{\mathbf{e}}$ and the standard deviation $s_{\mathbf{e}}$ of error values in $\mathbf{e}$ are calculated and used to classify x_t as either an anomaly or not. If the difference between e_t and $\overline{x}_{\mathbf{e}}$ is greater than $\varepsilon \cdot s_{\mathbf{e}}$, where ε is a user-specified anomaly classification factor, then x_t is classified as an anomaly, otherwise it is not.

If $\mathbf{e}$ is empty, then x_t is classified as non-anomalous. This classification can be justified as follows. Empty set $\mathbf{e}$ indicates that all previous $\mathcal{W}_{size}$ input values were classified as anomalies. This case may indicate the presence of a long anomaly window or the occurrence of a new trend in input values of the data stream. After classifying all values of previous $\mathcal{W}$ window as anomalies, the detector should no longer assume that input values are anomalies, but instead should treat them as normal values and should not indicate the presence of anomalies.

The crucial step of the above procedure is the selection of the value of parameter ε. Usually, the value of that parameter should be experimentally adjusted to the domain of the input data stream. In our experimental evaluation, we performed tuning of values of that parameter.

5.3. Generation of an initial output value of a candidate output neuron

The module called *Generation of initial output value of a candidate output neuron* in Fig. 3 is responsible for generating initial output values of candidate output neurons when they are created. The proposed method of generating output values is based on the assumption that network prediction y_t for non-anomalous input value x_t should be relatively similar to x_t, whereas network prediction y_t for anomalous x_t should differ from x_t significantly. Following this assumption, we propose to generate initial output values of candidate output neurons based on recent window $\mathcal{W}$ of input values. More specifically, in our approach, the initial output value of the candidate output neuron created for x_t is taken randomly from a normal distribution whose mean and standard deviation are determined based on input values present in $\mathcal{W}$. The aim of this approach is to generate values, which will correspond to the current fluctuations in the data stream.

5.4. Correction of an output value of an output neuron

The *Value correction* module (see Fig. 3) is responsible for correcting the output value of the candidate output neuron n_c created for input value x_t. The value correction is performed only when that input value was classified as non-anomalous. Intuitively, one can perceive n_c as a neuron representation of x_t, since initial synapses weights of n_c were determined based on firing order values of input neurons for value x_t. When x_t is not classified as an anomaly, the initial output value of the candidate neuron n_c is adjusted as follows: $v_{n_c} \leftarrow v_{n_c} + (x_t - v_{n_c}) \cdot \xi$. In this formula, ξ is a user given value correction factor within the range [0, 1]. If $\xi = 0$, the initial output value of n_c will not change. If $\xi = 1$, then v_{n_c} value will be equal to x_t.

Let us assume that output neuron n_c with its output value corrected as described above became output neuron n_i in **NO** or was merged with output neuron n_i in **NO** and that new input value x_{t_1} arrives at time t_1 slightly later than time t and becomes subject to classification. If x_{t_1} is similar to previously classified x_t value, then output neuron n_i corresponding to x_t is very likely to fires as first and its output value v_{n_i} will be reported as network prediction y_{t_1}. Since that v_{n_i} output value was previously adjusted to non-anomalous x_t value, then also x_{t_1} value will be likely classified as non-anomalous provided prediction error e_{t_1} is relatively small in comparison with the errors present in $\mathbf{e}$.

Overall, the aim of correcting an output value of candidate output neuron n_c is to prevent future incorrect classification of input values that could be caused by possible fluctuations of input values in a data stream.

6. The OeSNN-UAD algorithm for unsupervised anomaly detection

In this section, our proposed OeSNN-UAD algorithm (please refer to Table 1 for notation used in it), working principles of which were described in Section 5, is presented and discussed in detail. The main procedure of OeSNN-UAD is presented in Algorithm 1. All OeSNN-UAD input parameters, that is $\mathcal{W}_{size}$, NI_{size}, NO_{size}, mod, C, sim, ξ, ε are constant during the whole process of anomaly detection and learning. First, the current counter CNO_{size} of output neurons in output repository **NO** is set to 0. Next, based on the fact that the values of post-synaptic potential thresholds γ_{n_i} are the same for all output neurons n_i in **NO** and depend only on constants NI_{size}, mod and C (as follows from Theorem 1.(iv) and Corollary 1.(iii) provided in Section 4), their common post-synaptic potential threshold, denoted by γ, is calculated only once. Then, window $\mathcal{W}$ is initialized with input values $x_1, \ldots, x_{\mathcal{W}_{size}}$ from data stream $\mathbf{X}$. These values are not classified as anomalies (the assumption that the first $\mathcal{W}_{size}$ values of the data stream are not treated as anomalies is similar to the approach taken in the Numenta Anomaly Benchmark repository, which we use in our experimental evaluation).

The detection of anomalies among input values x_t of $\mathbf{X}$, where $t \geq \mathcal{W}_{size} + 1$, starts in step 7 of Algorithm 1 and for each of these input values is carried out as follows. First, window $\mathcal{W}$ is

Table 1
Notations and parameters used in OeSNN-UAD.

Notation	Description	Value
$\mathbf{X}$	Stream of input data	
$\mathcal{W}$	Window of recent input values	
$\mathcal{W}_{size}$	Window size (the number of recent input values in $\mathcal{W}$)	
x_t	Input value at time t	
y_t	OeSNN-UAD prediction of x_t	
$\mathbf{Y}$	Vector of predicted values	
u_t	Boolean value indicating anomaly presence or absence for input value x_t	
$\mathbf{U}$	Vector of results of anomaly detection for input values	
NI	Set of input neurons	
NI_{size}	Number of input neurons	
TS	Synchronization time of input neurons firings	
n_j	jth neuron in the set **NI** of input neurons	
μ_j^{GRF}	GRF center for input neuron n_j	
σ_j^{GRF}	GRF width for input neuron n_j	
$I_{max}^{\mathcal{W}}$	Maximal input value in window $\mathcal{W}$	
$I_{min}^{\mathcal{W}}$	Minimal input value in window $\mathcal{W}$	
$Exc_j^{GRF}(x_t)$	Excitation of jth GRF for value x_t	
$T_{n_j}(x_t)$	Firing time of input neuron n_j for value x_t	
$\bar{x}_{\mathcal{W}}$, $s_{\mathcal{W}}$	Mean and standard deviation of input values in $\mathcal{W}$	
$\mathcal{N}$	Normal distribution	
NO	Repository of output neurons	
NO_{size}	Number of output neurons in repository **NO**	
mod	Modulation factor of weights of synapses	(0, 1)
sim	User-given similarity threshold	
n_i	ith output neuron from repository **NO**	
$\mathbf{w}_{n_i}$	Vector of synaptic weights of output neuron n_i	
w_{n_j,n_i}	Weight of a synapse between $n_j \in$ **NI** and $n_i \in$ **NO**	
γ	Post-synaptic potential threshold of output neurons	
v_{n_i}	Output value of output neuron n_i	
τ_{n_i}	Update time of output neuron n_i	
M_{n_i}	Number of updates of output neuron n_k	
PSP^{max}	Maximal post-synaptic potential of output neurons	
C	Fraction of $PSP_{n_i}^{max}$ for calculation of γ_{n_i}	(0, 1]
n_c	New candidate output neuron	
D_{n_c,n_i}	Euclidean distance between weights vectors $\mathbf{w}_{n_c}$ and $\mathbf{w}_{n_i}$	
ξ	Error correction factor	[0, 1]
e_t	Error between input value x_t and its prediction y_t	
$\mathbf{E}$	Vector of error values between $\mathbf{X}$ and $\mathbf{Y}$	
ε	Anomaly classification factor	≥ 2

updated with input value x_t which becomes subject to anomaly classification, and GRFs as well as firing order values of input neurons are determined based on the content of window $\mathcal{W}$, as presented in Algorithm 2. Next, output neuron $n_f \in$ **NO** that fires as first is obtained (see Algorithm 3).

The determination of the first output neuron n_f to fires is carried out according to our proposed procedure FiresFirst, which is presented in Algorithm 3. To this end, for efficiency reasons, the algorithm uses lower approximation $\underline{PSP}_{n_i}$ of post-synaptic potential PSP_{n_i} for each output neuron $n_i \in$ **NO** instead of PSP_{n_i}. Lower approximation $\underline{PSP}_{n_i}$ differs from PSP_{n_i} in that PSP_{n_i} is obtained after firing all input neurons, while $\underline{PSP}_{n_i}$ sufficiently approximates PSP_{n_i} after firing only a few most significant input neurons, whose firing order values are lowest.

Specifically, output neuron n_f firing as first is obtained as follows: initially, $\underline{PSP}_{n_i}$ of each output neurons in **NO** is reset to 0. Next, in the loop in which variable j iterates over identifiers of input neurons starting from the one with the least order value (0) to the one with the greatest order value ($NI_{size}-1$), $\underline{PSP}_{n_i}$ of each output neuron n_i in **NO** is calculated in an incremental way. As a result, after k iterations, where $k \in \{1, 2, \ldots NI_{size}\}$, $\underline{PSP}_{n_i}$ is equal to $w_{n_{j_0}n_i} \cdot mod^{order(j_0)} + w_{n_{j_1}n_i} \cdot mod^{order(j_1)} + \cdots w_{n_{j_{k-1}}} n_i \cdot mod^{order(j_{k-1})}$, where n_{j_l} is the input vector whose *order* is equal to l, $l = 0 \ldots k-1$; that is, $\underline{PSP}_{n_i} = w_{n_{j_0}n_i} \cdot mod^0 + w_{n_{j_2}n_i} \cdot mod^1 + \cdots w_{n_{j_{k-1}}n_i} \cdot mod^{(k-1)}$, and $order(j_0) = 0$, $order(j_1) = 1$, …, $order(j_{k-1}) = k-1$.

After the first iteration, in which $\underline{PSP}$ (and by this, PSP) of at least one output neuron is greater than the γ threshold, no other iterations are carried out. In such a case, each output neuron whose current $\underline{PSP}$ value is greater than γ is added to the `ToFire` list. n_f is found as this output neuron in `ToFire` that has the greatest value of $\underline{PSP}$, and is returned as the result of the FiresFirst function. Please note that the method we propose to calculate more and more precise lower approximations of PSP of output neurons guarantees that n_f is found in a minimal number of iterations. If within NO_{size} iterations no output neuron with $\underline{PSP} > \gamma$ is found, the FiresFirst returns NULL to indicate that no output neuron in **NO** was fired.

If FiresFirst returns NULL, then, in steps 12 to 14 of Algorithm 1, value x_t is classified as being anomalous and the prediction of network y_t as well as error value e_t are set to NULL and $+\infty$, respectively. Otherwise, in steps 16 to 18 of Algorithm 1, the prediction of network y_t is assigned output value v_{n_f}, error e_t is set to the absolute difference between x_t and y_t, and our proposed ClassifyAnomaly procedure is invoked.

ClassifyAnomaly is given in Algorithm 4. Its description was provided in Section 5.2. The procedure returns Boolean value u_t indicating presence or absence of an anomaly for input value x_t.

In step 22 of Algorithm 1, new candidate output neuron n_c is created, and then initialized in our proposed InitializeNeuron procedure, which is presented in Algorithm 5. InitalizeNeuron first creates synapses between candidate output neuron n_c and each input neuron in **NI**. Then, the weights of the created synapses are calculated according to the firing order values of input neurons in **NI** obtained for input value x_t. Next, output value v_{n_c} of n_c is generated from a normal distribution created based on input values currently falling into window $\mathcal{W}$ (as it was

Algorithm 1 OeSNN-UAD

Input: $\mathbf{X} = [x_1, x_2, \ldots, x_T]$ - stream of input data.
Assure constant:
$\mathcal{W}_{size}$, NO_{size}, NI_{size}, mod, C, sim, ξ, ε
Output: **U** - a vector with classification of each $x \in \mathbf{X}$ as an anomaly or not.

1: $CNO_{size} \leftarrow 0$
2: $\gamma \leftarrow C \cdot \frac{1-mod^{2 \cdot NI_{size}}}{1-mod^2}$
3: Initialize $\mathcal{W}$ with $x_1, \ldots, x_{\mathcal{W}_{size}} \in \mathbf{X}$
4: Initialize $y_1, \ldots, y_{\mathcal{W}_{size}}$ with random values from $\mathcal{N}(\bar{x}_{\mathcal{W}}, s^2_{\mathcal{W}})$ and add to **Y**
5: Intialize **E** with $e_l \leftarrow |x_l - y_l|$, $l = 1, \ldots, \mathcal{W}_{size}$
6: Set $u_1, \ldots, u_{\mathcal{W}_{size}}$ to *False* and add to **U**
7: **for** $t \leftarrow \mathcal{W}_{size} + 1$ to T **do**

{*———————— OeSNN-UAD anomaly detection ————————*}
8: Update window $\mathcal{W}$ with value x_t
9: InitializeGRFs($\mathcal{W}$)
10: $n_f \leftarrow$ FiresFirst(CNO_{size})
11: **if** n_f is NULL **then** ▷ If none of output neurons fired
12: $y_t \leftarrow$ NULL; append y_t to **Y**
13: $e_t \leftarrow +\infty$; append e_t to **E**
14: $u_t \leftarrow True$ ▷ Immediately classify x_t as anomaly
15: **else**
16: $y_t \leftarrow v_{n_f}$; append y_t to **Y**
17: $e_t \leftarrow |x_t - y_t|$; append e_t to **E**
18: $u_t \leftarrow$ ClassifyAnomaly(**E**, **U**)
19: **end if**
20: Append u_t to **U**

{*———————— OeSNN-UAD learning ————————*}
21: Create a candidate output neuron n_c
22: $n_c \leftarrow$ InitializeNeuron($\mathcal{W}, t$)
23: **if** $u_t = False$ **then** ▷ Anomaly for x_t not detected
24: $v_{n_c} \leftarrow v_{n_c} + (x_t - v_{n_c}) \cdot \xi$ ▷ Correct generated output value of n_c
25: **end if**
26: $n_s \leftarrow$ FindMostSimilar(n_c)
27: **if** $D_{n_c,n_s} \leq sim$ **then**
28: UpdateNeuron(n_s, n_c)
29: **else if** $CNO_{size} < NO_{size}$ **then**
30: Insert n_c to **NO**; $CNO_{size} \leftarrow CNO_{size} + 1$
31: **else**
32: $n_{oldest} \leftarrow$ an output neuron in **NO** such that $\tau_{n_{oldest}} = min\{\tau_{n_i} \mid i = 0, \ldots, NO_{size} - 1\}$
33: Replace n_{oldest} with n_c in **NO**
34: **end if**
35: **end for**
36: **return U**

presented in Section 5.3), and finally the update time τ_{n_c} is set to current input time t. Additionally, if the anomaly is not detected (u_t is *False*), then the value correction operation is performed which adjusts output value v_{n_c} of the candidate output neuron n_c. Specifically, the value v_{n_c} is increased or decreased by the factor $\xi \in [0, 1]$ of the difference $x_t - v_{n_c}$ (please see Section 5.4 for details).

In step 26 of Algorithm 1, the FindMostSimilar procedure, presented in Algorithm 6, is called. The procedure finds an output neuron $n_s \in$ **NO**, such that the Euclidean distance D_{n_c,n_s} between vectors of synapses weights of n_c and n_s is the smallest. If D_{n_c,n_s} is less than or equal to the similarity threshold value sim, then n_s is merged with n_c according to the UpdateNeuron procedure, presented in Algorithm 7. The updated values of synapses weights, output value, update time and update counter of output neuron n_s are calculated according Eq. (12).

Algorithm 2 InitializeGRFs($\mathcal{W}$)

Input: $\mathcal{W} = \{x_{t-(\mathcal{W}_{size}-1)}, \ldots, x_t\}$ window of input values of **X**.

1: Obtain current $I^{\mathcal{W}}_{min}$ and $I^{\mathcal{W}}_{max}$ from $\mathcal{W}$
2: Calculate $\sigma^{GRF} \leftarrow \frac{I^{\mathcal{W}}_{max} - I^{\mathcal{W}}_{min}}{NI_{size} - 2}$
3: **for** $j \leftarrow 0$ to $NI_{size} - 1$ **do** ▷ For all input neurons in **NI**
4: $\sigma^{GRF}_j \leftarrow \sigma^{GRF}$ ▷ By Eq. (3) and Proposition 1.(i)
5: Calculate μ^{GRF}_j ▷ By Eq. (2)
6: Calculate excitation $Exc^{GRF}_j(x_t)$ ▷ By Eq. (1)
7: Calculate firing time $T_{n_j}(x_t)$ ▷ By Eq. (4)
8: **end for**
9: **for** $j \leftarrow 0$ to $NI_{size} - 1$ **do**
10: Calculate $order(j)$
11: **end for**
12: **return**

Algorithm 3 FiresFirst(CNO_{size})

Input: CNO_{size} - current size of output repository **NO**
Output: n_f - an output neuron $\in$ **NO** which fires first

1: ToFire $\leftarrow \emptyset$
2: **SNIID** $\leftarrow$ the list of identifiers of input neurons in **NI** obtained by sorting input neurons increasingly according to their *order* value
3: **for** $i \leftarrow 0$ to $CNO_{size} - 1$ **do**
4: $\underline{PSP}_{n_i} \leftarrow 0$
5: **end for**
6: **for** $j \leftarrow$ first to last input neuron identifier on list **SNIID** **do**
7: **for** $i \leftarrow 0$ to $CNO_{size} - 1$ **do** ▷ output neuron ids
8: $\underline{PSP}_{n_i} \leftarrow \underline{PSP}_{n_i} + w_{n_j n_i} \cdot mod^{order(j)}$
9: **if** $\underline{PSP}_{n_i} > \gamma$ **then**
10: Insert n_i to ToFire
11: **end if**
12: **end for**
13: **if** ToFire $\neq \emptyset$ **then**
14: $n_f \leftarrow$ an output neuron in ToFire such that $\underline{PSP}_{n_f} = max\{\underline{PSP}_{n_i} | n_i \in$ ToFire$\}$
15: **return** n_f
16: **end if**
17: **end for**
18: **return** NULL

Algorithm 4 ClassifyAnomaly(**E**, **U**)

Input: $\mathbf{E} = [e_1, \ldots, e_t]$ - vector of error values; $\mathbf{U} = [u_1, \ldots, u_{t-1}]$ - vector of input values classified as anomalies or not; e_t - error between predicted y_t and input x_t values.
Output: u_t - a Boolean value being classification of x_t as either an anomaly or not.

1: **e** $\leftarrow \emptyset$
2: Append to **e** all e_k such that: $k = t - (\mathcal{W}_{size} - 1), \ldots, t - 1$ and u_k is *False*
3: **if e** $= \emptyset$ **then**
4: $u_t = False$
5: **else**
6: Calculate $\bar{x}_{\mathbf{e}}$ and $s_{\mathbf{e}}$ over **e**
7: **if** $e_t - \bar{x}_{\mathbf{e}} \geq \varepsilon \cdot s_{\mathbf{e}}$ **then**
8: $u_t = True$
9: **else**
10: $u_t = False$
11: **end if**
12: **end if**
13: **return** u_t

Otherwise, if the number of output neurons in repository **NO** is still below NO_{size}, then n_c is added to **NO** and counter CNO_{size} is incremented.

Algorithm 5 InitializeNeuron($\mathcal{W}$)

Input: $\mathcal{W}$ - current window of input values, t - time of current input value x_t
Output: n_c - a newly created and initialized candidate output neuron

Create new neuron n_c
for $j \leftarrow 0$ to $NI_{size} - 1$ **do**
    Create synapse between $n_j \in$ **NI** and n_c
end for
for $j \leftarrow 0$ to $NI_{size} - 1$ **do** ▷ Calculate $\mathbf{w}_{n_c}$
    $w_{n_j n_c} \leftarrow mod^{order(n_j)}$
end for
$v_{n_c} \leftarrow$ Generate output value from $\mathcal{N}(\bar{x}_{\mathcal{W}}, s^2_{\mathcal{W}})$
$\tau_{n_c} \leftarrow t$
$M_{n_c} \leftarrow 1$
return n_c

If both the similarity condition is not fulfilled and the **NO** repository is full, then candidate output neuron n_c replaces the oldest neuron n_{oldest} in **NO** (that is, neuron n_{oldest} in **NO** whose update time $\tau_{n_{oldest}}$ is minimal).

Algorithm 6 FindMostSimilar(n_c)

Input: n_c - a candidate output neuron.
Output: n_s - the neuron in **NO** such that Euclidean distance between $\mathbf{w}_{n_s}$ and $\mathbf{w}_{n_c}$ is least.

for $i \leftarrow 0 \ldots CNO_{size} - 1$ **do**
    $D_{n_c,n_i} \leftarrow dist(\mathbf{w}_{n_c}, \mathbf{w}_{n_i})$
end for
$n_s \leftarrow$ an output neuron in **NO** such that $D_{n_c,n_s} = min\{(D_{n_c,n_i}) \mid i = 0, \ldots, CNO_{size} - 1\}$
return n_s

Algorithm 7 UpdateNeuron(n_s, n_c)

Input: n_s - a neuron from **NO** to be updated; n_c - a newly created candidate output neuron

$\mathbf{w}_{n_s} \leftarrow (\mathbf{w}_{n_c} + M_{n_s} \cdot \mathbf{w}_{n_s})/(M_{n_s} + 1)$
$v_{n_s} \leftarrow (v_{n_c} + M_{n_s} \cdot v_{n_s})/(M_{n_s} + 1)$
$\tau_{n_s} \leftarrow (\tau_{n_c} + M_{n_s} \cdot \tau_{n_s})/(M_{n_s} + 1)$
$M_{n_s} \leftarrow M_{n_s} + 1$
return

7. Experiments

In this section, we present the results of the comparative experimental evaluation of the proposed OeSNN-UAD method and state-of-the-art methods and algorithms for unsupervised anomaly detection. For comparison, we use the following methods and algorithms: Numenta (Ahmad et al., 2017c), NumentaTM (Ahmad et al., 2017c), HTM JAVA (Hawkins & Ahmad, 2016), Skyline (Stanway, 2015), TwitterADVec (Kejariwal, 2015), Yahoo EGADS (Extensible Generic Anomaly Detection System) (Laptev et al., 2015), DeepAnT (Munir et al., 2019b), Bayesian Changepoint (Adams & MacKay, 2007), EXPected Similarity Estimation (EXPoSE) (Schneider et al., 2016), KNN CAD (Burnaev & Ishimtsev, 2016), Relative Entropy (Wang et al., 2011) and ContextOSE (Smirnov, 2016) and the unsupervised anomaly detection method offered in Zhang et al. (2019). The experiments were carried out on two anomaly benchmark repositories: Numenta Anomaly Benchmark (Ahmad et al., 2017a) and Yahoo Anomaly Dataset (Webscope, 2015). The experimental results concerning both our proposed OeSNN-UAD and all other anomaly detectors used for comparative assessment were obtained after tuning their parameters.

The section starts with an overview of both of the aforementioned benchmark repositories. Then, it is followed by the description of the experimental setup. The OeSNN-UAD parameter tuning method is presented as well. Next, we present the extensive experimental evaluation of the compared methods and algorithms is provided. Finally, we provide and discuss the results of the experiments examining influence of different values of $\mathcal{W}_{size}$ and ε parameters on the quality of the anomaly detection.

7.1. Anomaly benchmark repositories used in the experiments

Both repositories (Numenta Anomaly Benchmark and Yahoo Anomaly Dataset), which we use for experiments, contain time series with labeled anomalies. Both of them are commonly used to asses the quality of unsupervised anomaly detection for various methods and algorithms (please see, for example (Ahmad et al., 2017c; Munir et al., 2019b; Zhang et al., 2019)).

7.1.1. The Numenta Anomaly Benchmark repository

The Numenta Anomaly Benchmark (NAB) repository contains 7 categories of datasets, both artificial and real, each of which has multiple CSV data files. Each CSV data file consists of two time series, one of them being a series of timestamp values and the second one being a series of input values. The number of input values in data files varies between 1000 and 22 000. Overall, there are 58 data files in NAB. All time series in the NAB repository are imbalanced with the average percentage of input values being anomalies in a time series less than 10% on average. In the current version (1.0) of NAB, the following categories of data files are distinguished:

- `artificialNoAnomaly` — contains data files artificially generated, which do not have anomalies;
- `artificialWithAnomaly` — contains data files which consist of artificial data with anomalies;
- `realAdExchange` — contains data files with online advertisements clicks recordings;
- `realAWSCloudwatch` — contains data files with metrics from AWS servers;
- `realKnownCauses` — contains real data files, such as hourly registered taxi schedules in New York City or CPU utilization;
- `realTraffic` — contains data files with freeway traffic recordings, such as speed or travel time;
- `realTweets` — contains data files with Tweeter volume statistics.

Only data files in `artificialNoAnomaly` category do not contain anomalies. The data files in the remaining categories contain at least one anomaly window. Each anomaly window consists of multiple input values and each data file can have several anomaly windows. The labeling of anomaly windows in the data files was conducted manually or by means of algorithms (Ahmad et al., 2017b). However, as follows from the documentation of NAB, it is not guaranteed that all anomalies in a data file are labeled (Ahmad et al., 2017b). In fact, potential users of NAB are encouraged to perform additional anomaly labeling of data files (Ahmad et al., 2017b), which can be released in future versions of NAB. It was also reported in Singh and Olinsky (2017) that some data files in NAB contain missing values or differences in input values distributions. For these reasons, NAB is particularly challenging for anomaly detection algorithms. In Fig. 4, we illustrate data file `ec2_cpu_utilization_ac20cd` from `realAWSCloudwatch` category. It can be noted that input values around timestamp 500, which can intuitively be perceived as anomalous, are not labeled as such, whereas all input values in window starting at timestamp 3375 and ending at timestamp 3777, which are labeled in the dataset as anomalies, in the majority of cases seem not to be anomalous. This kind of incorrect anomaly labeling or its lack can negatively influence the measures describing anomaly detection quality, as well as make learning of a detector less effective.

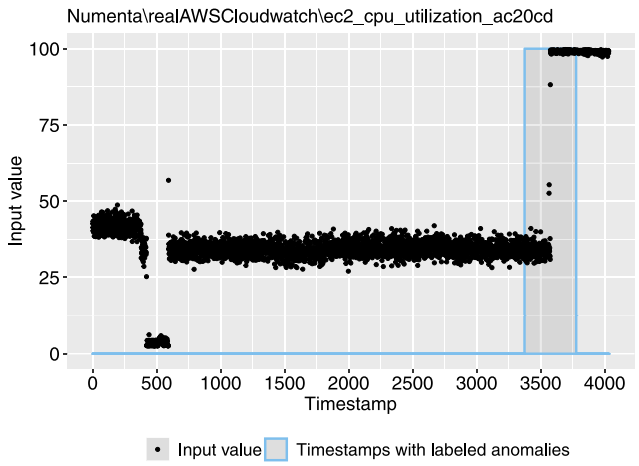

Fig. 4. Input values and labeled anomalies in data file `ec2_cpu_utilization_ac20cd` of `RealAWSCloudwatch` category in Numenta Anomaly Benchmark. Anomalous input values, which occur around timestamp 500 are not labeled as such, whereas anomaly window around timestamp 3500 incorrectly identifies input values as anomalies.

7.1.2. The Yahoo Anomaly Dataset repository

The Yahoo Anomaly Dataset (Webscope, 2015) repository consists of four categories of data files:

- `A1Benchmark` — contains 67 data files with real input time series values. Both single anomalous values and windows of anomalies occur in these data files. Each data file consists of three time series: timestamps, input values and a label for each input value as being either anomalous or not.
- `A2Benchmark` — consists of 100 synthetic data files, which contain anomalies in the form of single anomalous values. Most of input time series values in this category have their own periodicity. Similarly to `A1Benchmark`, each data file contains only three time series: timestamp, input values and labels indicating the presence or absence of anomalies.
- `A3Benchmark` — has 100 synthetic data files with anomalies in the form of single anomalous values. In comparison to `A2Benchmark`, input values time series in this category are more noisy. In addition to three standard time series (timestamps, input values and anomalies labels), data files in this category contain also other time series (trend, noise, seasonality and changepoint), which are not used by our OeSNN-UAD model either for anomaly detection or for the calculation of the detection quality.
- `A4Benchmark` — contains 100 synthetic data files with anomalies. The majority of the anomalies correspond to sudden transitions from an input data trend to another significantly different input data trend. `A4Benchmark` consists of the same time series types as `A3Bechnmark`. Again, in our detector we used only first three time series in each data file: timestamp, input values and anomalies labels. The former two are used for anomaly detection by OeSNN-UAD, while the third time series is used for calculation of its detection quality.

All time series in the Yahoo repository are imbalanced or strongly imbalanced with the average percentage of input values being anomalies in a time series less than 1% on average. Similarly to the NAB repository, data files in the Yahoo Anomaly Dataset repository are provided as CSV files.

7.2. Experimental setup and optimization

To run our OeSNN-UAD detector one has to provide values of the following parameters: NI_{size} — number of input neurons, NO_{size} — the maximal number of output neurons, *mod* — modulation factor, C — fraction of PSP^{max} required to fires an output neuron, *sim* — threshold value for similarity between a candidate output neuron and an output neuron in terms of their weights vectors, which is required for merging these neurons, ξ — output value correction factor, W_{size} — window size, ε — anomaly classification factor. The values of the β and TS parameters do not need to be determined, because the classification result does not depend on them (please see, Proposition 1.(v)). In the reported experiments, values of the first 6 parameters were set as follows: $NI_{size} = 10$, $NO_{size} = 50$, $sim = 0.17$, $mod = 0.6$, $C = 0.6$, $\xi = 0.9$. In our preliminary experiments, we observed that values of parameters $\mathcal{W}_{size}$ and ε have the greatest impact on anomaly detection (similar observation follows from the experiments reported in Ahmad et al. (2017c) and Munir et al. (2019b), in which approaches to anomaly detection that also use window size and anomaly classification threshold are proposed). Hence, the selection of $\mathcal{W}_{size}$ and ε parameters values of OeSNN-UAD was optimized. In the case of data files from the Numenta Anomaly Benchmark repository, OeSNN-UAD was run multiple times with $W_{size} \in \{100, 200, \ldots, 600\}$ and $\varepsilon \in \{2, 3, \ldots, 7\}$, while in the case of data files from the Yahoo Anomaly Datasets repository, it was executed with $W_{size} \in \{20, 40, \ldots, 500\}$ and $\varepsilon \in \{2, 3, \ldots, 17\}$.

Following Kasabov (2014), Maciąg et al. (2019), Tu et al. (2017), we implemented the grid search procedure to find the best values of parameters W_{size} and ε for each data file separately. The grid search procedure iterates over all given combinations of input learning parameters to find a set of parameters values (in particular, W_{size} and ε), which provides the best anomaly detection results for an input data file.

The implementation of OeSNN-UAD is prepared in C++ and its source code is publicly available (https://github.com/piotrMaciag32/eSNN-AD). The compiled executable file is lightweight (it consumes around 2 MB of RAM memory), which makes it additionally suitable for environments with very strict memory constraints, such as sensor microcontrollers or IoT devices.

7.3. Obtained anomaly detection results

In the experimental phase, we compare anomaly detection quality of our approach to the other state-of-the-art methods and algorithms provided in the literature. To this end, we use five measures of detection quality: *precision*, *recall*, *F-measure*, *balanced accuracy* (BA) and *Matthews correlation coefficient* (MCC).

Precision provides information on how many of the input values detected as anomalies by the detector are actually labeled as anomalies in data files, while *recall* indicates how many of the labeled anomalies in the data file are properly detected by the detector. *F-measure* (F1) is a harmonic mean of precision and recall. Moreover, since most of the datasets in the NAB and Yahoo repositories are strongly imbalanced, we additionally computed *balanced accuracy*. *Balanced accuracy* (BA) is defined as the average of recall and the equivalent of recall calculated with respect to the category of non-anomalous input values. This measure is typically used when dealing with imbalanced datasets, such as time series in the NAB and Yahoo repositories. *Matthews correlation coefficient* (MCC) is defined as a correlation coefficient between real and predicted labels of both anomalous and non-anomalous input values.

In Eqs. (13), (14), (15), (16), (17) we give formulae for precision, recall, F-measure, balanced accuracy and Matthews correlation coefficient. In these equations, |TP| (True Positives) denotes

Table 2
Average F-measure values obtained for Numenta Anomaly Benchmark stream data using the unsupervised anomaly detectors (marked with *) presented in Munir et al. (2019b) and using our proposed OeSNN-UAD detector. The bolded results are the best for each data files category. The results for the detectors marked with * were reported in Munir et al. (2019b).

Dataset category	*Bayesian Changepoint**	*Context OSE**	*EXPoSE**	*HTM JAVA**	*KNN CAD**	*Numenta**	*NumentaTM**	*Relative Entropy**	*Skyline**	*Twitter ADVec**	*Windowed Gaussian**	*DeepAnT**	*OeSNN-UAD*
Artificial no Anomaly	0	0	0	0	0	0	0	0	0	0	0	0	0
Artificial with Anomaly	0.009	0.004	0.004	0.017	0.003	0.012	0.017	0.021	0.043	0.017	0.013	0.156	**0.427**
Real Ad Exchange	0.018	0.022	0.005	0.034	0.024	0.040	0.035	0.024	0.005	0.018	0.026	0.132	**0.234**
Real AWS Cloud	0.006	0.007	0.015	0.018	0.006	0.017	0.018	0.018	0.053	0.013	0.06	0.146	**0.369**
Real Known Cause	0.007	0.005	0.005	0.013	0.008	0.015	0.012	0.013	0.008	0.017	0.006	0.2	**0.324**
Real Traffic	0.012	0.02	0.011	0.032	0.013	0.033	0.036	0.033	0.091	0.020	0.045	0.223	**0.340**
Real Tweets	0.003	0.003	0.003	0.010	0.004	0.009	0.010	0.006	0.035	0.018	0.026	0.075	**0.310**

the number of input values that were both classified as anomalies by the detector and labeled as being such in the data file, |FP| (False Positives) denotes the number of input values that were classified as anomalous by the detector, but were not labeled as anomalies in the data file, while |FN| (False Negatives) denotes the number of input values labeled as anomalous in the data file, but not classified as anomalies by the detector.

$$\text{Precision} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|}. \tag{13}$$

$$\text{Recall} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}|}. \tag{14}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{15}$$

$$\text{BA} = \frac{1}{2} \cdot \left(\frac{|TP|}{|TP| + |FN|} + \frac{|TN|}{|TN| + |FP|} \right). \tag{16}$$

$$\text{MCC} = \frac{|TP| \cdot |TN| - |FP| \cdot |FN|}{\sqrt{(|TP| + |FP|)(|TP| + |FN|)(|TN| + |FP|)(|TN| + |FN|)}}. \tag{17}$$

In Fig. 5, we present charts showing anomaly detection results obtained for example data files in categories `artificialWithAnomaly`, `realAdExchange` and `realTraffic` of the NAB repository. Fig. 6 shows similar charts for example datasets in categories `A1Benchmark`, `A2Benchmark` of the Yahoo Anomaly Dataset repository. The charts present the occurrences of true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN) found by the proposed OeSNN-UAD algorithm. It can be observed on the charts, especially in the case of the data files from the NAB repository, that there are some input values in these data files that are not correctly labeled as anomalies. For example, it can be seen on the first chart of Fig. 5, which presents anomaly detection in `art_daily_jumpsdown` data file from `artificialWithAnomaly`, that many input values that occur around time period 2800–3200 are labeled in the data file as anomalies, while they should be perceived as non-anomalous (please see the green groups of input values). In this case, OeSNN-UAD classifies them, as we believe, correctly as non-anomalous, but they are treated as false negatives with respect to available labeling of the data file. On the other hand, it may happen that true anomalous input values are not labeled as anomalies in these data files. Such a phenomenon may result in the decrease in the value of the detection quality measures, such as F-measure, even though a detector correctly classifies input data as anomalous or non-anomalous, respectively.

Our proposed OeSNN-UAD is able to detect point anomalies (see e.g. Fig. 6: `synthetic_13` and `synthetic_44`), contextual anomalies and collective anomalies (see e.g. Fig. 6:`real_19`).[2] Proper detection of these types of anomalies, however, is often dependent on the used values of parameters (especially on window size $\mathcal{W}_{size}$ and anomaly factor ε). When it is possible, one should adjust the values of these parameters based on analysis of a subset of available data stream.

In Table 2, we show the results of OeSNN-UAD anomaly detection for the Numenta Anomaly Benchmark repository as well as for the other unsupervised anomaly detection methods and algorithms. As in Munir et al. (2019b), we report the mean F-measure obtained for each category of data files for each compared detector. As follows from Table 2, OeSNN-UAD outperforms the results obtained by the other detectors in terms of F-measure for each category of data files.

Table 3 presents the obtained precision and recall values for the selected data files from the Numenta Anomaly Benchmark repository. For some data files, OeSNN-UAD is able to provide much higher values of both precision and recall than the other detectors. Most of the detectors compared with OeSNN-UAD, for which the results are presented in Table 3, have very high values of precision, but very low values of recall. High values of precision imply that very few cases which were not labeled as anomalies are detected as anomalous. Low recall, on the other hand, implies that these detectors do not discover a large number of cases labeled as anomalies. In fact, the recall values below 0.01 obtained for many data files presented in Table 3 by these detectors indicate their very limited ability to detect anomalies. To the contrary, our OeSNN-UAD detector, in the majority of cases, is characterized by much larger values of the recall, and thus it is much more efficient in detecting anomalies than the compared detectors. The only case, when OeSNN-UAD has low recall value can be observed for `exchange-2-cpc-results` data file. Nevertheless, as follows from Fig. 7, which presents this data file and the results of anomaly detection obtained with OeSNN-UAD, a number of input values in this data file were labeled in a counter-intuitive way. For example, the majority of input values in the time interval 245–407 were found by OeSNN-UAD as non-anomalous (namely, input values marked as green, which seem not to be real anomalies). Surprisingly, these values are labeled as anomalies in the data file. This incoherence between given labeling of input values and their expected labeling results in increase in the number of false negatives and, in consequence,

[2] The characteristics of point, collective and contextual types of anomalies can be found in Chandola et al. (2009).

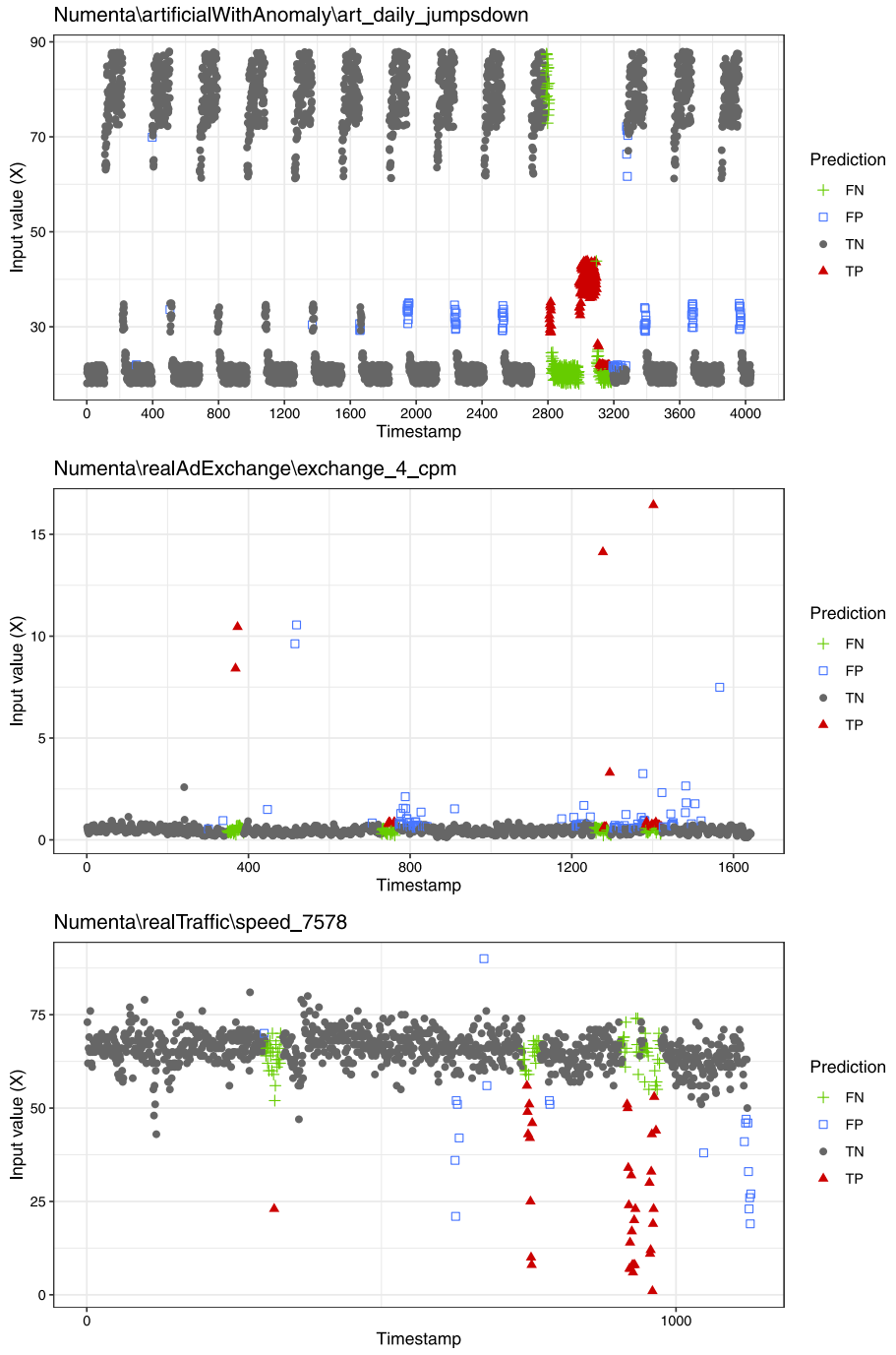

Fig. 5. Anomaly detection results for selected data files from the Numenta Anomaly Benchmark repository with OeSNN-UAD.

in decrease in the recall value for OeSNN-UAD. In addition, high input values (marked as blue ones) in the time interval 1000–1600 were found by OeSNN-UAD as anomalies (and seem to be real anomalies), but are not labeled as such in the data file. This results in an increase in the number of false positives and, in consequence, in a decrease in the precision value for OeSNN-UAD.

In Table 4, we present the obtained optimal values of $\mathcal{W}_{size}$ and anomaly classification factor ε of OeSNN-UAD for data files, which were used in the experiments the results of which are presented in Table 3.

In Table 5, we present the comparison of the obtained F-measure values for each category of data files in the Yahoo Anomaly Dataset repository. For the real data files category (`A1Benchmark`), the proposed OeSNN-UAD approach provides higher values of the F-measure than the recent results reported in the literature (Munir et al., 2019b), while for the other three categories of data files, OeSNN-UAD's results are competitive to the results reported there.

In Table 6, we provide average values of precision, recall, F-measure (F1), balanced accuracy (BA) and Matthews correlation coefficient (MCC), obtained with OeSNN-UAD for all categories of data files in the NAB and Yahoo repositories.

Table 7 presents a comparison of precision, recall, F-measure, balanced accuracy and MCC for OeSNN-UAD and the unsupervised anomaly detection method offered in Zhang et al. (2019) for the data files in Real Known Cause and Real Tweets categories of the NAB repository which were used there for validation. The results presented in Table 7 were derived from the information

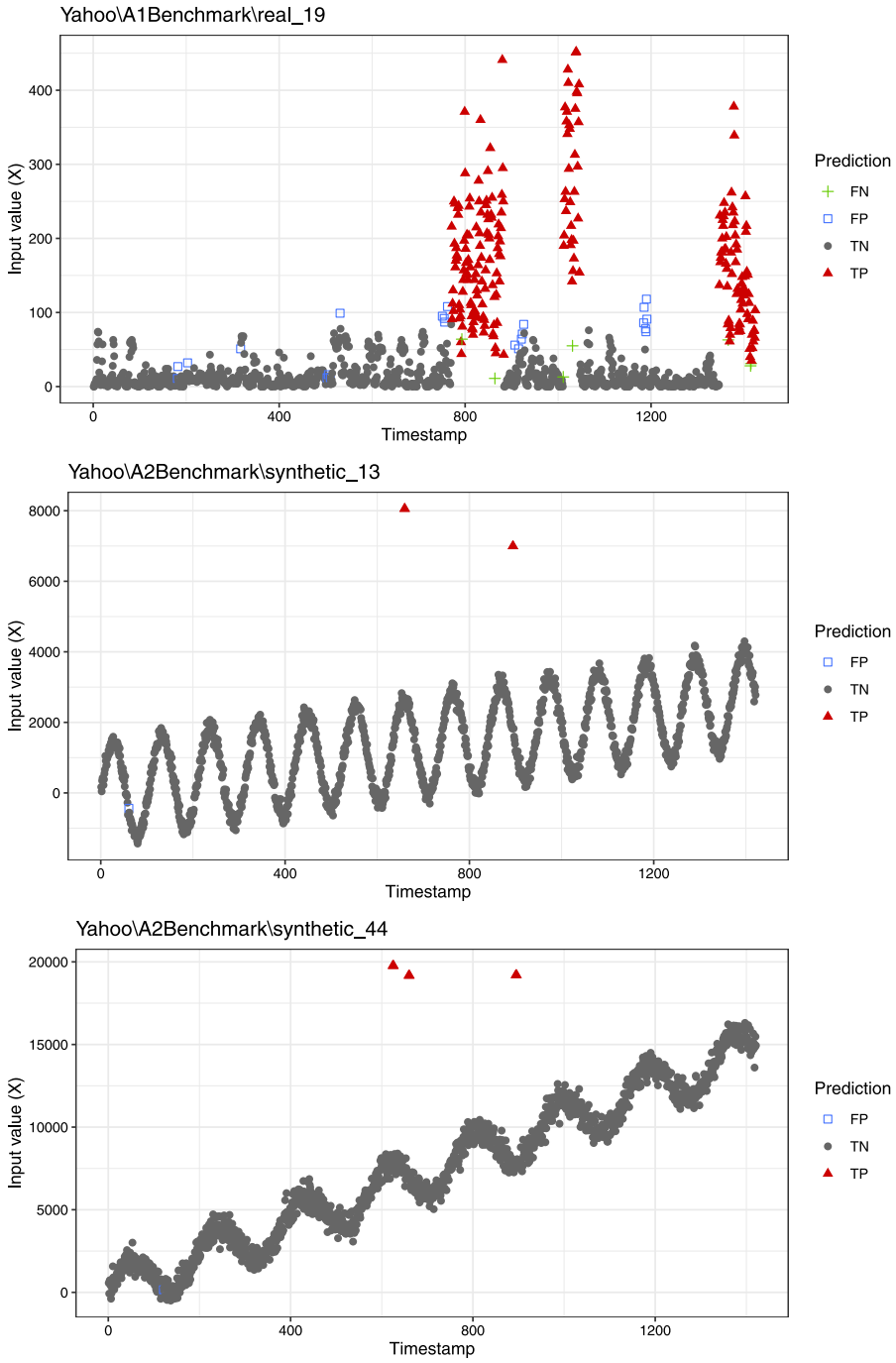

Fig. 6. OeSNN-UAD anomaly detection results for selected Yahoo data files. The triangles on the chart `real_19` present collective anomalies (timestamp ∼1000) as well as both collective and contextual anomalies (timestamps ∼800 and ∼1400); the triangles on the other two charts present point anomalies.

provided in Zhang et al. (2019) about the number of true positives and false positives that were discovered by the anomaly detector offered there and based on the number of anomalous and non-anomalous input values in data files of the NAB repository.

7.4. Experiments with different values of window size and anomaly classification factor

In this subsection, we analyze the impact of different values of window size $\mathcal{W}_{size}$ and classification factor ε on the anomaly detection results obtained by OeSNN-UAD for the data files that were presented in Figs. 5 and 6, as well as the `rds_cpu_uti .e47b3b` data file (see Fig. 10) from the NAB repository. The first six examined data files contain anomalies of different nature, as discussed in Section 7.3, while the last data file contains a time series covering four data trends and anomalies. In these experiments, the values of all parameters of OeSNN-UAD, except for $\mathcal{W}_{size}$ and ε, are set as given in Section 7.2.

In Fig. 8, we present the example plots of the obtained values for the selected performance measures (precision, recall, F1, BA and MCC) for anomaly detection using different values of the window size $\mathcal{W}_{size}$ for data files presented in Figs. 5 and 6. Analogous plots for different values of the ε parameter are presented in Fig. 9.

Data files `art._daily_jumpsdown` (Fig. 5), `synthetic_13` (Fig. 6) and `synthetic_44` (Fig. 6) can be conceived as periodic

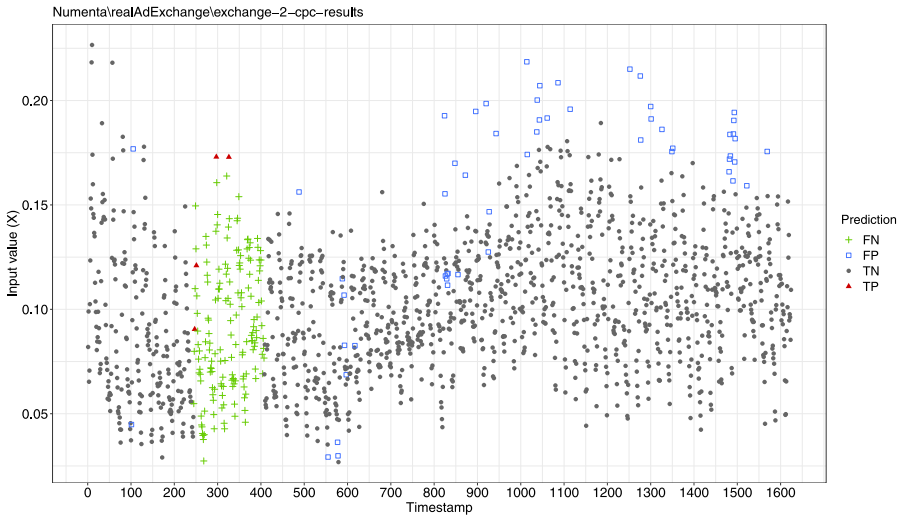

Fig. 7. Anomaly detection for data file `exchange-2-cpc-results` in `realAdExchange` category of data files of NAB with OeSNN-UAD.

Table 3
Precision and recall values obtained for Numenta Anomaly Benchmark stream data using the selected unsupervised anomaly detectors (marked with *) presented in Munir et al. (2019b) and using our proposed OeSNN-UAD detector. The results for the detectors marked with * were reported in Munir et al. (2019b).

Time series		ContextOSE*		NumentaTM*		Skyline*		ADVec*		DeepAnT*		OeSNN-UAD	
		Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.
Real AWS Cloud Watch	ec2-cpu-utilization-5f5533	1	0.005	1	0.01	1	0.002	1	0.002	1	0.01	0.18	0.51
	rds-cpu-utilization-cc0c53	1	0.005	1	0.002	1	0.1	0.62	0.012	1	0.03	0.50	0.75
Real Known Cause	ambient-temperature-system-failure	0.33	0.001	0.5	0.006	0	0	0	0	0.26	0.06	0.21	0.75
	cpu-utilization-asg-misconfiguration	0.12	0.001	0.52	0.01	0	0	0.74	0.01	0.63	0.36	0.32	0.49
	ec2-request-latency-system-failure	1	0.009	1	0.009	1	0.014	1	0.02	1	0.04	0.38	0.40
	machine-temperature-system-failure	1	0.001	0.27	0.004	0.97	0.01	1	0.02	0.8	0.001	0.39	0.50
	nyc-taxi	1	0.002	0.85	0.006	0	0	0	0	1	0.002	0.17	0.47
	rouge-agent-key-hold	0.33	0.005	0.5	0.005	0	0	0	0	0.34	0.05	0.13	0.23
	rouge-agent-key-updown	0	0	0	0	0	0	0.11	0.002	0.11	0.01	0.25	0.43
Real Traffic	occupancy-6005	0.5	0.004	0.2	0.004	0.5	0.004	0.5	0.004	0.5	0.004	0.18	0.41
	occupancy-t4013	1	0.008	0.66	0.008	1	0.04	1	0.02	1	0.036	0.50	0.44
	speed-6005	0.5	0.004	0.25	0.008	1	0.01	1	0.01	1	0.008	0.36	0.34
	speed-7578	0.57	0.03	0.6	0.02	0.86	0.16	1	0.01	1	0.07	0.64	0.30
	speed-t4013	1	0.008	0.8	0.01	1	0.06	1	0.01	1	0.08	0.31	0.78
	TravelTime-387	0.6	0.01	0.33	0.004	0.62	0.07	0.2	0.004	1	0.004	0.22	0.34
	TravelTime-451	1	0.005	0	0	0	0	0	0	1	0.009	0.82	0.11
Real Ad Exchange	exchange-2-cpc-results	0.5	0.006	0	0	0	0	0	0	0.03	0.33	0.07	0.02
	exchange-3-cpc-results	0.75	0.02	1	0.007	0	0	1	0.02	0.71	0.03	0.21	0.23
Real Tweets	Twitter-volume-GOOG	0.75	0.002	0.38	0.005	0.59	0.02	0.81	0.01	0.75	0.01	0.25	0.43
	Twitter-volume-IBM	0.37	0.002	0.22	0.005	0.22	0.01	0.5	0.009	0.5	0.005	0.24	0.28

time series with a few anomalies. All selected anomaly detection measures obtain optimal values of $\mathcal{W}_{size}$ values approximately equal to:

- 1 × the time series period length in the case of `art._daily_jumpsdown`,
- 0.5 × the time series period length in the case of data file `synthetic_44`,
- m × the time series period length, where $m \in \{0.5, 1, 2, 3\}$, and for some even greater values in the case of data file `synthetic_13`.

In the case of data files `real_19` (Fig. 6) and `speed_7578` (Fig. 5), in which collective anomalies occur, the best results of our anomaly detection approach were obtained for $\mathcal{W}_{size}$ not less than the cardinality of the most numerous maximal cluster of collective anomalies (see Fig. 8). In particular, data file `real_19` contains three maximal clusters of collective anomalies (see Fig. 6), the first of which is most numerous. Thus, $\mathcal{W}_{size}$ is recommended to be set to the cardinality of this cluster for data file `real_19`.

In the case of data file `exchange_4_cpm_results` (Fig. 5), multiple values of $\mathcal{W}_{size}$ allowed obtaining comparably good quality measures (see Fig. 8). Unlike the data files discussed earlier in this subsection, `exchange_4_cpm_results` contains non-periodic time series with mainly single point anomalies rather than collective anomalies. The proper determination of the value of $\mathcal{W}_{size}$ for this data file is challenging, especially due to the fact that the current labeling of what is an anomaly and what

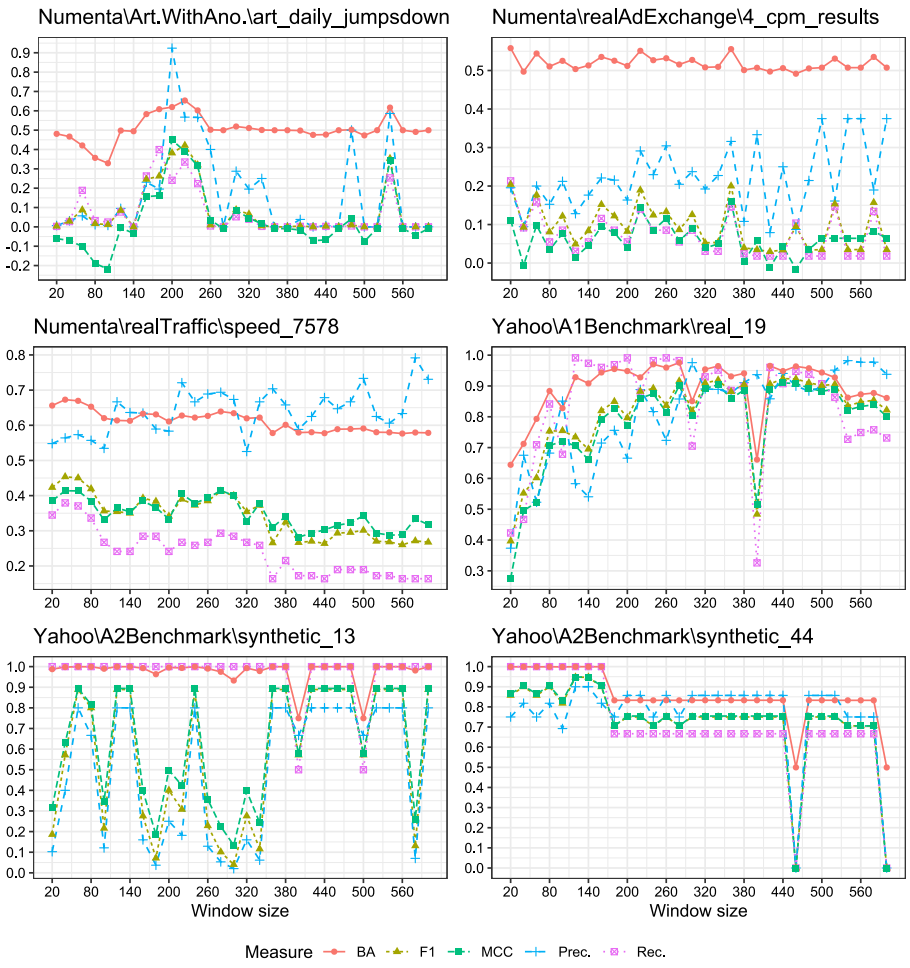

Fig. 8. The plots of the obtained measures (precision, recall, F1, BA and MCC) for anomaly detection experiments with OeSNN-UAD using different window sizes $\mathcal{W}_{size}$ for data files presented in Figs. 5 and 6. The applied values of ε for data files: `art_daily_jumpsdown`, `4_cpm_results`, `speed_7578`, `real_19`, `synthetic_13` and `synthetic_44` are as follows: 5, 3, 4, 4, 4 and 7, respectively.

is not happens to be counter-intuitive (this labeling issue was previously discussed in Sections 7.3 and 7.1.1).

In Fig. 9, we present the plots of obtained values of the selected quality measures when only the anomaly classification factor ε is subject to changes. As it can be noticed from the plots in this figure, the best values of the selected measures are usually obtained for smaller values of ε, such as 3, 4 or 5. However, in the case of `4_cpm_results` data file, the best values of the precision and MCC measures are obtained for larger values of the ε parameter (the best values of the other measures in the case of this data file were obtained for $\varepsilon = 2$).

Furthermore, in order to better assess the impact of $\mathcal{W}_{size}$ and ε on the anomaly detection results, we carried out experiments on `rds_cpu_uti.e47b3b`, which consists of four data trends and anomalies (see Fig. 10). Please note that input values marked as false negatives at the end of the first two trends presented in this figure are not proper anomalies, but were labeled by the NAB authors as such. Thus, the values of recall for anomaly detection using this data file are understated. Nevertheless, the first input values corresponding to newly observed trends are correctly recognized as anomalies.

The obtained values of anomaly detection quality measures for data file `rds_cpu_uti.e47b3b` and for different values of the $\mathcal{W}_{size}$ and ε parameters are presented in Fig. 11. We note that the best values of the selected measures were obtained in particular for $\mathcal{W}_{size}$ equal to 100, which corresponds to the number of input values labeled as anomalous that are present at the beginning of each new trend. In the case of ε, the best values for precision, F1 and MCC as well as values close to the best ones for the recall and BA measures were obtained for both $\varepsilon = 4$ and $\varepsilon = 6$.

The experiments conducted in this subsection suggest that: (i) in the case of a periodic time series, the best anomaly detection results are likely to be obtained for $\mathcal{W}_{size}$ parameter equal to a multiple of the period length of input values or of its half; (ii) in the case of a data file containing different data trends, it seems reasonable to set the value of $\mathcal{W}_{size}$ to the maximum number of consecutive atypical input values that alone are not treated as constituting a new trend in data, but as anomalies; (iii) in the remaining considered types of data files, it seems reasonable to set the value of $\mathcal{W}_{size}$ to not less than the cardinality of the most numerous maximal cluster of collective anomalies; (iv) the best anomaly detection results can be obtained for relatively small values of the ε parameter.

8. Conclusions

In this article, we offered a new detector of anomalies in data streams. Our proposed OeSNN-UAD detector is designed for univariate stream time series data and adapts Online evolving Spiking Neural Networks OeSNN. The distinctive feature of our proposed OeSNN-UAD anomaly detector is that it is the only detector based on (Online) evolving Spiking Neural Networks which operates in an online and unsupervised way.

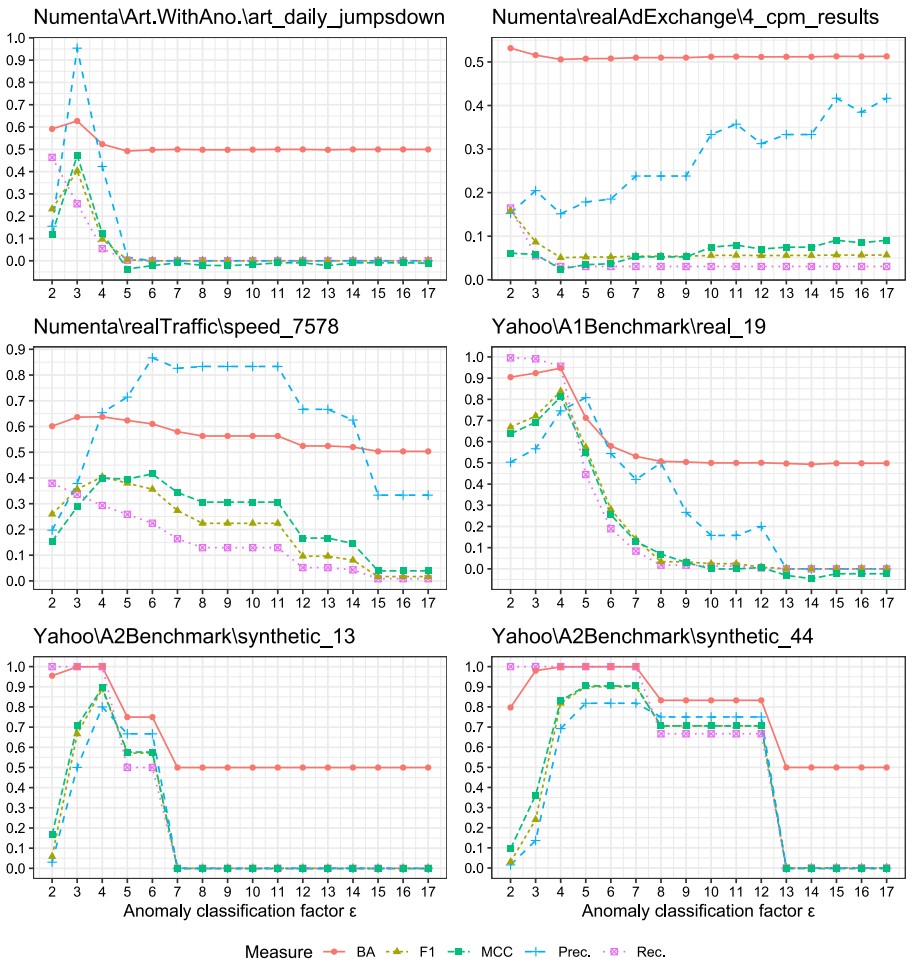

Fig. 9. The plots of obtained measures (precision, recall, F1, BA and MCC) for anomaly detection experiments with OeSNN-UAD using different values of anomaly classification factor ε for data files presented in Figs. 5 and 6. The applied values of $\mathcal{W}_{size}$ for data files: `art_daily_jumpsdown`, `4_cpm_results`, `speed_7578`, `real_19`, `synthetic_13` and `synthetic_44` are as follows: 300, 300, 100, 180, 480 and 80, respectively.

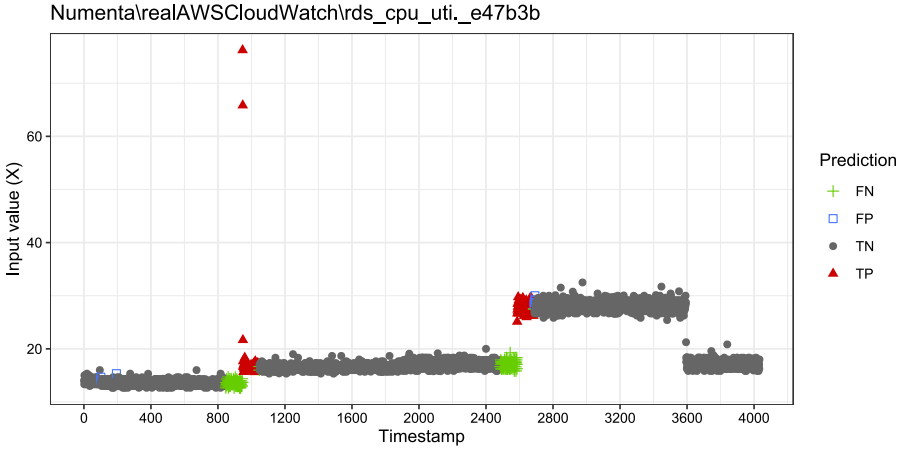

Fig. 10. Anomaly detection results for data file `rds_cpu_uti.e47b3b` (the results were obtained with parameters $\mathcal{W}_{size} = 100, \varepsilon = 6$.

OeSNN-UAD adapts the architecture of OeSNN for anomaly detection purposes. Nevertheless, unlike OeSNN, OeSNN-UAD applies a different model of an output layer and different methods of learning and input values classification. In particular, OeSNN-UAD does not separate output neurons into known in advance decision classes. Instead, each new output neuron on OeSNN-UAD is assigned an output value, which is randomly generated based on recent input values and then is updated in the course of learning of OeSNN-UAD to better adapt to changes in a data stream. As a part of the proposed OeSNN-UAD detector, we offered a new two-step anomaly classification method. Our method treats an input value as anomalous if either none of output neurons fires or, otherwise, if an error between the input value and its OeSNN-UAD prediction is greater than the average prediction

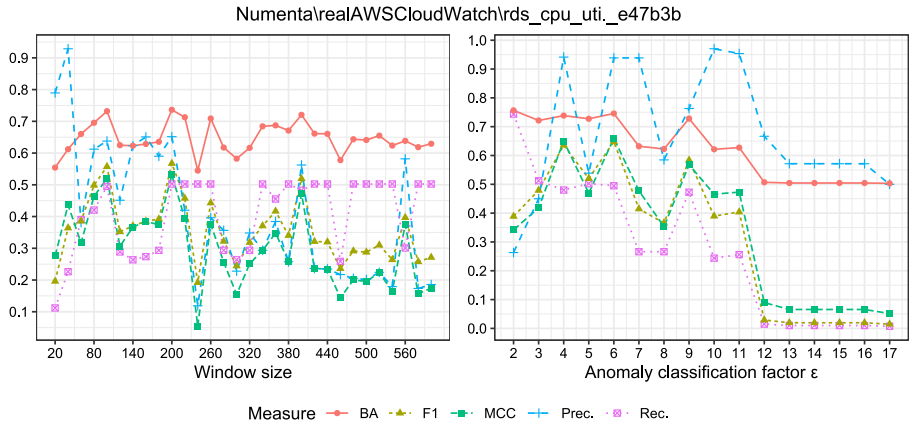

Fig. 11. The plots of obtained measures (precision, recall, F1, BA and MCC) for anomaly detection experiments with OeSNN-UAD using different values of window size $\mathcal{W}_{size}$ (left plot) and anomaly classification factor ε (right plot) obtained for data file `rds_cpu_util_e47b3b`. The left plot was obtained using constant value of $\varepsilon = 4$, while the right plot was obtained with constant value of $\mathcal{W}_{size} = 180$.

Table 4
Optimal window sizes and anomaly classification factors used by OeSNN-UAD for data files from Table 3.

	Time series	$\mathcal{W}_{size}$	ε
Real Ad Exchange	exchange-2-cpc-results	100	2
	exchange-3-cpc-results	100	4
Real AWS Cloud Watch	ec2-cpu-utilization-5f5533	300	2
	rds-cpu-utilization-cc0c53	600	7
Real Known Cause	ambient-temperature-system-failure	500	6
	cpu-utilization-asg-misconfiguration	600	3
	ec2-request-latency-system-failure	400	5
	machine-temperature-system-failure	300	4
	nyc-taxi	100	3
	rouge-agent-key-hold	100	5
	rouge-agent-key-updown	300	6
Real Traffic	occupancy-6005	300	2
	occupancy-t4013	600	2
	speed-6005	600	2
	speed-7578	100	4
	speed-t4013	400	3
	TravelTime-387	100	2
	TravelTime-451	100	5
Real Tweets	Twitter-volume-GOOG	200	3
	Twitter-volume-IBM	100	5

error plus user-given multiplicity of the standard deviation of recent prediction errors.

In the article, we proved that all candidate output neurons, as well as all output neurons in the repository, have the same values of the sum of their synaptic weights, their maximal post-synaptic potentials, and their post-synaptic potential thresholds, respectively. The last property eliminates the necessity of recalculation of these thresholds when output neurons of OeSNN-UAD are updated in the course of the learning process, and thus it allows increasing the speed of classification of input stream data. Moreover, we also proved that firing order values of input neurons do not depend on values of *TS* and β parameters, which were previously used in OeSNNs for input value encoding with Gaussian Receptive Fields.

In the experimental part, we compared the quality of the proposed OeSNN-UAD detector with 14 other anomaly detectors provided in the literature. The experiments were conducted on data files from two anomaly benchmark repositories: Numenta Anomaly Benchmark and Yahoo Anomaly Dataset. These two repositories cumulatively contain more than 500 data files grouped into several categories. For the assessment of the quality of anomaly detectors, we used five indicators: F-measure, precision, recall, balanced accuracy and Matthews correlation coefficient. For the Numenta Anomaly Benchmark repository, OeSNN-UAD is able to provide significantly better results in terms of F-measure for all categories of data files. The detailed analysis of the experimental results obtained for the data files in the Numenta Anomaly Benchmark repository that were considered in Munir et al. (2019b) also shows that OeSNN-UAD outperforms other compared detectors in terms of recall. In the case of the Yahoo Anomaly Dataset repository, OeSNN-UAD achieves higher F-measure values for the real data files category, while for the other three synthetic data categories the obtained values of the F-measure are competitive to the results reported in the literature (Munir et al., 2019b). In addition, in terms of recall, F-measure, balanced accuracy and MCC measures, OeSNN-UAD outperforms the method proposed in Zhang et al. (2019) on all NAB data files which were used there for evaluation.

As we discuss in Section 7.4, the quality measures of the OeSNN-UAD algorithm depend on properly selected values of its parameters, in particular, the window size $\mathcal{W}_{size}$ and classification factor ε. An inaccurate selection of these parameters can negatively affect the anomaly detection quality. Especially, too large values of the ε parameter could result in an increase in false negatives (which would result in low recall scores). Conversely, too small values of the ε parameter could increase the number of false positives (which would entail low precision values). Thus, the value of ε should be adjusted according to the observed dispersion of at least some representative subset of input values. The selection of proper values for $\mathcal{W}_{size}$ can be even more challenging, as shown in Section 7.4. The best value of this parameter can be linked to such factors as the periodicity of time series data, trends present in input values or the cardinalities of collective anomalies clusters. Thus, it is critical to conduct a proper tuning of this parameter that takes into account available characteristics of the input data file under study.

In the performed experiments, we used the OeSNN-UAD detector whose Online evolving Spiking Neural Network contained as few as 10 input neurons and at most 50 output neurons, and thus occupied very little operating memory. In spite of such a small number of neurons, OeSNN-UAD was able to outperform the compared detectors for most data files in each category of the Numenta Anomaly Benchmark repository and most real data

Table 5
Average F-measure values obtained for Yahoo Anomaly Dataset stream data using the unsupervised anomaly detectors (marked with *) presented in Munir et al. (2019b) and using our proposed OeSNN-UAD detector. The bolded results are the best for each data files category. The results for the detectors marked with * were reported in Munir et al. (2019b).

Dataset category	Yahoo EGADS*	Twitter Anomaly Detection, $\alpha = 0.05$*	Twitter Anomaly Detection, $\alpha = 0.1$*	Twitter Anomaly Detection, $\alpha = 0.2$*	DeepAnT (CNN)*	DeepAnT (LSTM)*	OeSNN-UAD
A1Benchmark	0.47	0.48	0.48	0.47	0.46	0.44	**0.70**
A2Benchmark	0.58	0	0	0	0.94	**0.97**	0.69
A3Benchmark	0.48	0.26	0.27	0.3	**0.87**	0.72	0.41
A4Benchmark	0.29	0.31	0.33	0.34	**0.68**	0.59	0.34

Table 6
Average values of F-measure, balanced accuracy (BA) and Matthews correlation coefficient (MCC) obtained with OeSNN-UAD for the NAB and Yahoo repositories.

Dataset category		Prec.	Rec.	F1	BA	MCC
Numenta Anomaly Benchmark	Artificial with Anomaly	0.500	0.457	0.427	0.690	0.391
	Real Ad Exchange	0.224	0.255	0.234	0.584	0.154
	Real AWS Cloud	0.358	0.445	0.342	0.683	0.369
	Real Known Cause	0.263	0.469	0.324	0.652	0.244
	Real Traffic	0.433	0.387	0.340	0.646	0.299
	Real Tweets	0.267	0.412	0.310	0.633	0.225
Yahoo Anomaly Dataset	A1Benchmark	0.657	0.791	0.697	0.869	0.706
	A2Benchmark	0.616	0.929	0.690	0.957	0.715
	A3Benchmark	0.557	0.374	0.409	0.686	0.432
	A4Benchmark	0.467	0.373	0.342	0.683	0.369

Table 7
Precision, recall, F-measure (F1), balanced accuracy, and Matthews correlation coefficient obtained for OeSNN-UAD and the method of Zhang et al. (2019) for the selected data files in the NAB repository. The results for the latter method were derived from the information provided in Zhang et al. (2019) about the number of true positives and false positives that were discovered by their anomaly detector and based on the number of anomalous and non-anomalous input values in the NAB data files.

Time series		The method of Zhang et al. (2019)					OeSNN-UAD				
		Prec.	Rec.	F1	BA	MCC	Prec.	Rec.	F1	BA	MCC
Real Known Cause	ambient-temperature-system-failure	0.833	0.007	0.014	0.503	0.07	0.207	0.752	0.325	0.716	0.27
	cpu-utilization-asg-misconfiguration	0.5	0.001	0.003	0.501	0.022	0.318	0.494	0.387	0.699	0.328
	ec2-request-latency-system-failure	1	0.006	0.011	0.503	0.073	0.38	0.402	0.39	0.67	0.332
	machine-temperature-system-failure	0.294	0.002	0.004	0.501	0.018	0.395	0.5	0.441	0.707	0.374
	nyc-taxi	0.722	0.013	0.025	0.506	0.087	0.166	0.471	0.245	0.604	0.138
	rouge-agent-key-hold	0.667	0.011	0.021	0.505	0.075	0.126	0.232	0.164	0.526	0.04
	rouge-agent-key-updown	0.667	0.008	0.015	0.504	0.064	0.251	0.432	0.317	0.645	0.23
Real Tweets	TV-AAPL	0.333	0.004	0.007	0.501	0.026	0.454	0.491	0.472	0.713	0.411
	TV-AMZN	0.233	0.004	0.009	0.501	0.019	0.17	0.41	0.24	0.594	0.132
	TV-CRM	0.615	0.005	0.01	0.502	0.049	0.313	0.551	0.399	0.708	0.328
	TV-CVS	0.6	0.002	0.004	0.501	0.03	0.201	0.543	0.293	0.656	0.21
	TV-FB	0.375	0.002	0.004	0.501	0.021	0.261	0.18	0.213	0.562	0.146
	TV-GOOG	0.4	0.003	0.006	0.501	0.027	0.248	0.429	0.314	0.65	0.236
	TV-IBM	0.429	0.002	0.004	0.501	0.023	0.241	0.284	0.261	0.592	0.172
	TV-KO	0.286	0.004	0.007	0.501	0.023	0.132	0.339	0.19	0.546	0.063
	TV-PFE	0.25	0.007	0.013	0.502	0.026	0.188	0.463	0.267	0.62	0.168
	TV-UPS	0.389	0.009	0.017	0.504	0.046	0.463	0.428	0.445	0.687	0.387

files in the Yahoo Anomaly Dataset repository. This proves that OeSNN-UAD is effective and suitable also for environments with restrictive memory limits.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

P. Maciąg acknowledges financial support of the Faculty of the Electronics and Information Technology of the Warsaw University of Technology, Poland (Grant No. II/2019/GD/1). J.L. Lobo and J. Del Ser would like to thank the Basque Government, Spain for their support through the ELKARTEK and EMAITEK funding programs. J. Del Ser also acknowledges funding support from the Consolidated Research Group MATHMODE (IT1294-19) given by the Department of Education of the Basque Government.

References

Adams, R. P., & MacKay, D. J. C. (2007). Bayesian Online changepoint detection. CoRR, abs/0710.3742, arXiv:0710.3742 URL: arXiv:abs/0710.3742.

Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017c). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, *262*, 134–147, Online Real-Time Learning Strategies for Data Streams.

Ahmad, S., Lavina, A., Purdya, S., & Agha, Z. (2017a). Numenta anomaly benchmark. https://github.com/numenta/NAB.

Ahmad, S., Lavina, A., Purdya, S., & Agha, Z. (2017b). Numenta anomaly benchmark - labeling instructions. https://drive.google.com/file/d/0B1_XUjaAXeV3YlgwRXdsb3Voa1k/view.

Bovenzi, A., Brancati, F., Russo, S., & Bondavalli, A. (2011). A statistical anomaly-based algorithm for on-line fault detection in complex software critical systems. In F. Flammini, S. Bologna, & V. Vittorini (Eds.), *Computer Safety, Reliability, and Security* (pp. 128–142). Berlin, Heidelberg: Springer Berlin Heidelberg.

Burnaev, E., & Ishimtsev, V. (2016). Conformalized density- and distance-based anomaly detection in time-series data. arXiv:1608.04585.

Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2019). Online anomaly detection for long-term ECG monitoring using wearable devices. *Pattern Recognition*, *88*, 482–492.

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. CoRR, arXiv:abs/1901.03407, arXiv:1901.03407.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 15:1–15:58.

Chandola, V., Banerjee, A., & Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *24*(5), 823–839.

Chandola, V., Mithal, V., & Kumar, V. (2008). Comparative evaluation of anomaly detection techniques for sequence data. In *2008 eighth IEEE international conference on data mining* (pp. 743–748).

Demertzis, K., & Iliadis, L. (2014). A hybrid network anomaly and intrusion detection approach based on evolving spiking neural network classification. In A. B. Sideridis, Z. Kardasiadou, C. P. Yialouris, & V. Zorkadis (Eds.), *E-democracy, security, privacy and trust in a digital world* (pp. 11–23). Cham: Springer International Publishing.

Demertzis, K., Iliadis, L., & Bougoudis, I. (2019). Gryphon: a semi-supervised anomaly detection system based on one-class evolving spiking neural network. *Neural Computing and Applications*.

Ergen, T., & Kozat, S. S. (2019). Unsupervised anomaly detection with LSTM neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15.

Hawkins, J., & Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, *10*, 23.

Izakian, H., & Pedrycz, W. (2013). Anomaly detection in time series data using a fuzzy c-means clustering. In *2013 Joint IFSA world congress and NAFIPS annual meeting (IFSA/NAFIPS)* (pp. 1513–1518).

Izakian, H., & Pedrycz, W. (2014). Anomaly detection and characterization in spatial time series data: A cluster-centric approach. *IEEE Transactions on Fuzzy Systems*, *22*(6), 1612–1624.

Kasabov, N. (2007). Evolving connectionist systems: The knowledge engineering approach. Springer.

Kasabov, N. K. (2014). Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, *52*, 62–76.

Kasabov, N. K. (2015). Evolving connectionist systems for adaptive learning and knowledge discovery: Trends and directions. *Knowledge-Based Systems*, *80*, 24–33, 25th anniversary of Knowledge-Based Systems.

Kasabov, N., Dhoble, K., Nuntalid, N., & Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition. *Neural Networks*, *41*, 188–201, Special Issue on Autonomous Learning.

Kejariwal, A. (2015). Introducing practical and robust anomaly detection in a time series. https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html.

Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., & Kim, K. J. (2017). A survey of deep learning-based network anomaly detection. *Cl uster Computing*.

Laptev, N., Amizadeh, S., & Flint, I. (2015). Generic and scalable framework for automated time-series anomaly detection. In *KDD '15, Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1939–1947). New York, NY, USA: ACM.

Lin, L., & Su, J. (2019). Anomaly detection method for sensor network data streams based on sliding window sampling and optimized clustering. *Safety Science*, *118*, 70–75.

Lobo, J. L., Laña, I. L., Del Ser, J., Bilbao, M. N., & Kasabov, N. (2018). Evolving spiking neural networks for online learning over drifting data streams. *Neural Networks*, *108*, 1–19.

Lobo, J. L., Oregi, I., Bifet, A., & Del Ser, J. (2020). Exploiting the stimuli encoding scheme of evolving spiking neural networks for stream learning. *Neural Networks*, *123*, 118–133. http://dx.doi.org/10.1016/j.neunet.2019.11.021, http://www.sciencedirect.com/science/article/pii/S0893608019303892.

Lobo, J. L., Ser, J. D., Bifet, A., & Kasabov, N. (2020). Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, *121*, 88–100.

Maciąg, P. S., Kasabov, N., Kryszkiewicz, M., & Bembenik, R. (2019). Air pollution prediction with clustering-based ensemble of evolving spiking neural networks and a case study for London area. *Environmental Modelling & Software*, *118*, 262–280.

Maciąg, P. S., Kryszkiewicz, M., & Bembenik, R. (2020). Online evolving spiking neural networks for incremental air pollution prediction. In *2020 international joint conference on neural networks* (pp. 1–8).

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *23rd european symposium on artificial neural networks, 2015*.

Munir, M., Siddiqui, S. A., Chattha, M. A., Dengel, A., & Ahmed, S. (2019). Fusead: Unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models. *Sensors*, *19*(11).

Munir, M., Siddiqui, S. A., Dengel, A., & Ahmed, S. (2019). Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, *7*, 1991–2005.

Petro, B., Kasabov, N., & Kiss, R. M. (2019). Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–13.

Pimentel, M. A., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, *99*, 215–249.

Rosner, B. (1983). Percentage points for a generalized ESD many-outlier procedure. *Technometrics*, *25*(2), 165–172.

Schneider, M., Ertel, W., & Ramos, F. (2016). Expected similarity estimation for large-scale batch and streaming anomaly detection. *Machine Learning*, *105*(3), 305–333.

Singh, N., & Olinsky, C. (2017). Demystifying Numenta anomaly benchmark. In *2017 international joint conference on neural networks* (pp. 1570–1577).

Smirnov, M. (2016). Contextual anomaly detector. https://github.com/smirmik/CAD.

Stanway, A. (2015). Etsy skyline. https://github.com/etsy/skyline.

Tu, E., Kasabov, N., & Yang, J. (2017). Mapping temporal variables into the neucube for improved pattern recognition, predictive modeling, and understanding of stream data. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(6), 1305–1317.

Wang, C., Viswanathan, K., Choudur, L., Talwar, V., Satterfield, W., & Schwan, K. (2011). Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE international symposium on integrated network management (IM 2011) and workshops* (pp. 385–392).

Webscope, Y. (2015). Labeled anomaly detection dataset - v. 1.0. http://research.yahoo.com/Academic_Relations.

Zhang, L., Zhao, J., & Li, W. (2019). Online and unsupervised anomaly detection for streaming data using an array of sliding windows and PDDs. *IEEE Transactions on Cybernetics*, 1–6.

Zoppi, T., Ceccarellli, A., & Bondavalli, A. (2019). An initial investigation on sliding windows for anomaly-based intrusion detection. In *2019 IEEE world congress on services (SERVICES)* vol. 2642-939X (pp. 99–104).