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

Piotr S. Maciag, Marzena Kryszkiewicz, Robert Bembenik, Jesus L. Lobo, Javier Del Ser, Senior Member, IEEE

Abstract—In this work, we propose a novel OeSNN-UAD (Online evolving Spiking Neural Networks for Unsupervised Anomaly Detection) approach for online anomaly detection in univariate time series data. Our approach is based on evolving Spiking Neural Networks (eSNN). Its distinctive feature is that the proposed eSNN architecture learns in the process of classifying input values to be anomalous or not. In fact, we offer an unsupervised learning method for eSNN, in which classification is carried out without earlier pre-training of the network with data labeled anomalies. Unlike in a typical eSNN architecture, neurons in the output repository of our architecture are not divided into known a priori decision classes. Each output neuron is assigned its own output value, which is modified in the course of learning and classifying the incoming input values of time series data. To better adapt to the changing characteristic of the input data and to make their classification efficient, the number of output neurons is limited: the older neurons are replaced with new neurons whose output values and synapses’ weights are adjusted according to the current input values of the time series. The proposed OeSNN-UAD approach was experimentally compared to the state-of-the-art unsupervised methods and algorithms for anomaly detection in stream data. The experiments were carried out on Numenta Anomaly Benchmark and Yahoo Anomaly Datasets. According to the results of these experiments, our approach significantly outperforms other solutions provided in the literature in the case of Yahoo Anomaly Benchmark. Also in the case of real data files category of Yahoo Anomaly Benchmark, 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.

Index Terms—Evolving Spiking Neural Networks, Anomaly detection, Outliers detection, Online learning, Time series data.

I. 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 caused by a hacker attack. An unexpected spiking usage of CPU unit in a computer system is another example of an anomalous behaviour 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 stream data. It is thus particularly important to design such anomalies detectors that can properly classify anomalies from data where none of the input values is labeled as being anomalous or not. Moreover, since the characteristic of the input data stream is typically varying, the designed anomaly detector should learn in an online mode, in which the classification of current input values adjusts the state of the detector for better anomaly detection in future input data.

In order to design an effective anomaly detection system, one may consider the adaptation of evolving Spiking Neural Networks (eSNN) to the task. eSNN are a subclass of Spiking Neural Networks (SNN), in which learning processes, neuronal communication and classification of data instances are based solely on spike exchange between neurons [1]. The architecture of an eSNN network consists of two layers of neurons: input and output. The aim of the input layer of neurons is to transform 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 [2], [3] or Gaussian Receptive Fields [4]. The distinctive feature of an eSNN is the evolving repository of output neurons, which in the training phase of the network is updated with a new output neuron that is created for each new input data sample presented to eSNN [5], [6]. In particular, each newly created output neuron can be either added to the output repository or, based on the provided similarity threshold, merged with one of the neurons already existing in the repository.

Recently an extension OeSNN of eSNN for online classification of stream data was proposed in [4]. Contrary to the eSNN architecture, the size of the output neurons repository in OeSNN is limited: older neurons are removed from the repository and are replaced with new neurons. It was presented in [4], 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 OeSNN-UAD (Online evolving Spiking Neural Networks for Unsupervised Anomaly Detection) approach for unsupervised anomaly detection in stream data.

Our main contributions presented in this article are as follows:

• we introduce an unsupervised learning model of OeSNN,
in which output neurons do not have classes assigned according to class labels present in the input stream data. Instead, each output neuron: 1) has an output value that is generated based on the current and past characteristic of data stream when the neuron is created and added to the repository, 2) output values of output neurons are corrected based on the classification of input values as being anomalous or not. Hence, to correctly detect anomalies in stream data the proposed OeSNN-UAD approach does not need any input values labeled as anomalies or not.

- as a part of the proposed OeSNN-UAD architecture, we offer a new anomaly classification method, which classifies each input value as anomalous or not according to the comparison of the prediction error obtained for that value with the average and the standard deviation of the past prediction errors of a given time window.
- we derive an important property of the eSNN neuronal model, which shows that the values of actual post synaptic potential thresholds of all output neurons are the same. This property eliminates the necessity of recalculation of these thresholds when output neurons of eSNN are updated in the course of the learning process and increases the speed of classification of input stream data.
- we prove experimentally that the proposed approach is more effective in the unsupervised detection of anomalies in stream data of Numenta Anomaly Benchmark and Yahoo Anomaly Datasets than other state-of-the-art approaches proposed in the literature.
- eventually, we argue that the proposed anomaly detection architecture is able to make fast classification of input values and work in environments with restrictive memory limits.

The paper is structured as follows. In Section II we provide description of the related work. The proposed OeSNN-UAD approach is offered in Section III which also contains theoretical properties of OeSNN-UAD and proofs. In Section IV we discuss experimental evaluation. First we give an overview of the used datasets and characterize the experimental setup. Then, we provide the results of anomaly detection with the proposed approach and with the state-of-the-art solutions. We conclude our work in Section V.

II. RELATED WORK

Unsupervised anomaly detection in time series data is an important task, which attracts attention of researchers and practitioners. A number of solutions of the task was offered in the literature to date. The state-of-the-art algorithms for anomaly detection are:

- Numenta and NumentaTM [7] - 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 available in the NAB set of algorithms. NumentaTM and Numenta differ in a way of the HTM implementation and its parameters initialization.
- HTM JAVA [8] - a JAVA implementation of the Numenta algorithm.
- Skyline [9] - 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 time series against the deviation from the average of past values. In Skyline, a given input value of time series is classified as an anomaly if it is marked as anomalous by the majority of ensemble detectors. Skyline is implemented in Python and available as a part of the NAB benchmark.
- TwitterADVec [10] - a method for anomaly detection based on the Seasonal Hybrid ESD (S-H-ESD) algorithm [11]. For given time series values, the S-H-ESD algorithm first calculates extreme Student deviates [12] 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 both an R language package and as a part of the NAB benchmark.
- Yahoo EGADS (Extensible Generic Anomaly Detection System) [13] - an algorithm consisting of the following modules: (i) a time-series modeling module, (ii) an anomaly detection module, and (iii) an alarming module. EGADS is able to discover three types of anomalies: outliers, sudden change points in values and anomalous subsequences of time series. To this end, the following three different anomaly detectors were implemented in 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 [14] - a semi-supervised deep learning method. DeepAnT operates with both Convolutional Neural Networks and Long-Short Term Memories networks and consists of several modules, such as: a time series prediction module and anomaly detector module. Contrary to the approach proposed here, DeepAnT divides classified time series values into training and testing parts. First, DeepAnT learns from the training data, and then it is used for the classification of the test data. The advantage of our OeSNN-UAD method over DeepAnT is its ability to learn the correct classification of anomalies based on the whole provided time series, rather than only from its training part.
- Bayesian Changepoint [15] - an online algorithm for sudden changepoint detection in time series data by means of the Bayesian inference. This method is particularly suited to time series data in which it is possible to clearly separate partitions of values generated from different probability distributions. The algorithm is able to infer 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) [16] - an algorithm that classifies anomalies based on the deviation of an input observation from an estimated distribution of past input values.
- KNN CAD [17] - a method for 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 for both 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 for both the classified value and 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 [18] - 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 [19] - 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, which are stored in the operating memory of ContextOSE.

In addition to the above presented methods and algorithms directly compared with our approach in the experimental evaluation, other approaches to unsupervised anomaly detection in time series data were proposed. In [20], an unsupervised approach to anomaly detection, which combines ARIMA (Auto-regressive Moving Average) method and Convolutional Neural Networks (CNN) was provided. [21] introduced an unsupervised anomaly detection method integrating Long-Short Term Memory (LSTM) networks and One-class Support Vector Machines. The method proposed in [21] uses online learning of LSTM offered in [22]. A supervised eSNN approach to anomaly detection, called HESADM, was proposed in [23]. In this approach, the eSNN network first learns based on the training data, and then it is used for anomaly detection. All data samples presented to the detector are labeled as being either anomalous or not and there is a clear distinction between training and testing phases. In [24], a semi-supervised approach to anomaly classification with one-class eSNN was offered and dedicated to intrusion detection systems. Contrary to the approaches presented in [23], [24], our OeSNN-UAD approach learns to recognize anomalies in an unsupervised mode, in which anomaly labels are not assigned to data samples.

An overview of anomaly detection techniques for stream data can be found in [23], [11], [25], [26], [27], [28], [29], [30], [31], [21].

### III. OeSNN-UAD - THE PROPOSED ANOMALY DETECTION MODEL BASED ON ONLINE EVOLVING SPIKING NEURAL NETWORKS

In this section, we present our online approach to unsupervised anomaly detection in stream data called OeSNN-UAD, which is based on the eSNN network. First, we overview the proposed architecture of OeSNN-UAD. Then, we describe encoding of input values applied by us and the used neuronal model. Eventually, we present our algorithm for anomaly detection in stream data.

#### A. The Architecture of OeSNN-UAD

The eSNN network of the proposed OeSNN-UAD architecture consists solely of an input layer and output layer. The input layer contains input neurons and their Gaussian Receptive Fields. The output layer is an evolving repository of output neurons. The proposed architecture of OeSNN-UAD is presented in Fig. 1.

The set of input neurons is denoted by NI, while the set of output neurons by NO. The number of input neurons is fixed and determined by user-given parameter N\textsubscript{size}, whereas the maximal number of output neurons is given by NO\textsubscript{size}, which is also a user-specified parameter value.

The input stream data is denoted by X, and is defined as a series of real values \(\{x_1, x_2, \ldots, x_T\}\). A window with regard to \(t\) is denoted by \(\mathcal{W}_t\) and is defined as \(\{x_{t-(W\text{size}-1)}, x_{t-(W\text{size}-2)}, \ldots, x_t\}\), where \(W\text{size}\) is a user-specified parameter called window size. Clearly, window \(\mathcal{W}_{t+1}\) can be obtained from \(\mathcal{W}_t\) by removing the first value from \(\mathcal{W}_t\), shifting the remaining values by one position to the left and adding \(x_{t+1}\) at the last position.

When value \(x_t\) occurs in stream data, and thus becomes subject to classification, values in window \(\mathcal{W}_t\) are used to determine GRFs of input neurons. Next, \(x_t\) is encoded by means of GRFs into a sequence of \(N\text{I}\text{size}\) spikes, which are then used to update the repository of output neurons. Eventually, \(x_t\) is classified as an anomaly or not. To this end, errors of the eSNN network predictions for non-anomalous values in \(\mathcal{W}_t\) are used. In the remainder of the article, in the case when \(t\) denotes the current time, we may also write briefly \(\mathcal{W}\) instead of \(\mathcal{W}_t\).

In Table 1 we list notation of parameters used in the algorithms presented in the article.

#### B. Input Layer of the Proposed OeSNN-UAD Approach

The aim of the input neurons of the eSSN network and their GRFs is to encode input values of a data stream into firing orders of these neurons. The firing orders are then used for learning of the eSNN network and for the detection of anomalies. The encoding of the input value \(x_t\) into firing orders of input neurons is performed in several steps. First, based on the actual time window \(\mathcal{W}\), GRFs of input neurons are recalculated. In particular, the maximal value in window \(\mathcal{W}\) (denoted by \(I^{\mathcal{W}}_{\text{max}}\)) and the minimal value (denoted by \(I^{\mathcal{W}}_{\text{min}}\)) are used to calculate centers \(I^{\text{GRF}}_{\text{max}}\) and widths \(I^{\text{GRF}}_{\text{size}}\) of GRFs. For each \(j\)-th input neuron, where \(j = 0, \ldots, N\text{I}\text{size} - 1\), the center \(I^{\text{GRF}}_{nj}\) and width \(\sigma_{nj}\) of its GRF are defined according to Eq. (1) and Eq. (2), respectively:

\[
\mu_{nj}^{\text{GRF}} = I^{\mathcal{W}}_{\text{min}} + \frac{2}{2} \left( \frac{I^{\mathcal{W}}_{\text{max}} - I^{\mathcal{W}}_{\text{min}}}{N\text{I}\text{size} - 2} \right) + \frac{1}{N\text{I}\text{size} - 1} \sum_{i=1}^{N\text{I}\text{size} - 1} I_{i}^{\mathcal{W}} - \frac{1}{N\text{I}\text{size} - 1} \sum_{i=1}^{N\text{I}\text{size} - 1} I_{i}^{\mathcal{W}}.
\]
The firing times of input neurons are used to calculate their firing ordering. Let $SNI$ denote a list of all input neurons in $NI$ that have the same firing times and such that $j < k$ input neuron $n_j$ precedes input neuron $n_k$ on the list. Then, the firing order $order(j)$ of an input neuron $n_j$, where $j = 0, \ldots, NI_{size} - 1$, equals the position of this input neuron on the $SNI$ list decreased by 1.

Let us consider an example of encoding of value $x_t = 0.5$ given in Fig. 2. The size of window $W$ is $W_{size} = 14$. The GRFs parameters $I_{min}^W$ and $I_{max}^W$ are 0.1 and 1.0, respectively. Seven neurons in the input layer are used with seven associated GRFs. In Fig. 2, the firing times are calculated with synchronization time $TS$ equal to 1.0. The input value $x_t = 0.5$ translates into the firing times of input neurons:

- $Exc_{n_0}(0.5) = 0.001 \rightarrow T_{n_0}(0.5) = 0.999$,
- $Exc_{n_1}(0.5) = 0.024 \rightarrow T_{n_0}(0.5) = 0.976$,
- $Exc_{n_2}(0.5) = 0.227 \rightarrow T_{n_0}(0.5) = 0.773$,
- $Exc_{n_3}(0.5) = 0.770 \rightarrow T_{n_0}(0.5) = 0.230$,
- $Exc_{n_4}(0.5) = 0.962 \rightarrow T_{n_0}(0.5) = 0.038$,
- $Exc_{n_5}(0.5) = 0.442 \rightarrow T_{n_0}(0.5) = 0.558$,
- $Exc_{n_6}(0.5) = 0.074 \rightarrow T_{n_0}(0.5) = 0.926$.

According to the obtained firing times of input neurons, their firing ordering is as follows: order(4) = 0, order(3) = 1, order(5) = 2, order(2) = 3, order(6) = 4, order(1) = 5, order(0) = 6.

C. Neuronal Model of Output Neurons

In our approach, we apply a simplified Leaky Integrate and Fire (LIF) neuronal model of output neurons as presented in [4]. According to this model, an output neuron accumulates its Postsynaptic Potential (PSP) until it reaches an actual postsynaptic potential threshold $\gamma$. Then the output neuron fires and its PSP value is reset to 0. The accumulation of PSP potential of an output neuron $n_i \in NO$ is given in Eq. (5):

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

where $w_{n_i n_j}$ represents the synapse’s weight from input neuron $n_j \in NI$ to output neuron $n_i \in NO$, $\text{mod}$ is a user-given modulation factor within range $[0, 1)$, and $\text{order}(j)$ is the firing order of the input neuron $n_j$ for the encoding of the value of $x_t$. In the proposed approach, the postsynaptic potential of each output neuron $n_i \in NO$ is reset to 0 (regardless if $n_i$ is actually fired or not) and re-calculated for each input value $x_t \in X$ being classified.
The distinctive feature of (O)eSNN 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 \( x_t \), its synapses weights are initialized according to the firings of input neurons for the \( x_t \) encoding. The initial weights of synapses between each input neuron in NI and the candidate output neuron \( n_c \) are calculated according to Eq. (6):

\[
w_{n_j n_c} = \text{mod} \text{order}(j),
\]

A candidate output neuron, say \( n_c \), is characterized also by two additional parameters: the maximal post-synaptic threshold \( \text{PSP}^{\text{max}}_{n_c} \) and the actual post-synaptic potential threshold \( \gamma_{n_c} \). The definition of \( \text{PSP}^{\text{max}}_{n_c} \) is given in Eq. (7):

\[
\text{PSP}^{\text{max}}_{n_c} = \sum_{j=0}^{N_1 \times \text{size} - 1} w_{n_j n_c} \cdot \text{mod} \text{order}(j),
\]

where \( \text{order}(.) \) is the firing order function calculated for this input value \( x_t \) for which candidate output neuron \( n_c \) was created. The definition of the actual post-synaptic potential threshold \( \gamma_{n_c} \) is given in Eq. (8):

\[
\gamma_{n_c} = \text{PSP}^{\text{max}}_{n_c} \cdot C,
\]

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

**Example 1:** To illustrate how synapses weights of a new candidate output neuron are calculated, let us consider an example of encoding the value of \( x_t = 0.5 \) with seven input neurons presented in Fig. 2 and \( \text{mod} \) as well as \( C \) parameters of neuronal model set to 0.5 and 0.8, respectively. The previously calculated firing orders of the seven input neurons for the encoded value 0.5 are as follows: order(4) = 0, order(3) = 1, order(5) = 2, order(2) = 3, order(6) = 4, order(1) = 5, order(0) = 6. In such a case, the weights of synapses between input neurons and neuron \( n_c \) are initialized as follows:

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

For the encoding of Example 1 the value of maximal postsynaptic potential threshold \( \text{PSP}^{\text{max}}_{n_c} \) calculated according to Lemma 2 is \( 1^2 + 0.5^2 + 0.25^2 + 0.125^2 + 0.0625^2 + 0.03125^2 + 0.015625^2 = 1.333251953 \).

Each candidate output neuron, say \( n_i \), is either added to the repository NO or is merged with some output neuron in NO. An additional parameter \( M_{n_i} \) provides the information from how many candidate output neurons \( n_i \) was created. \( M_{n_i} \) equal to 1 means that \( n_i \) is a former candidate output neuron and preserves values of its parameters. 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 \( n_c \), each weight of \( w_{n_j n_i} \) of the synapse between output neuron \( n_i \) and input neuron \( n_j \) is recalculated as shown in Eq. (9). \( \text{PSP}^{\text{max}}_{n_i} \) is recalculated as shown in Eq. (10) and \( \gamma_{n_i} \) is recalculated according to Eq. (11):

\[
w_{n_j n_i} \leftarrow \frac{w_{n_j n_i} + M_{n_i} \cdot w_{n_j n_i}}{M_{n_i} + 1},
\]

\[
\text{PSP}^{\text{max}}_{n_i} \leftarrow \frac{\text{PSP}^{\text{max}}_{n_c} + M_{n_i} \cdot \text{PSP}^{\text{max}}_{n_i}}{M_{n_i} + 1},
\]

Fig. 2. The encoding and the input layer of the proposed network architecture. \( W_{\text{size}} \) denotes the size of window \( W \). All current values in \( W \) are used to construct GRFs, while only \( x_t \) value is encoded and propagated to neurons in the output repository NO.
TABLE I
NOTATIONS AND PARAMETERS USED IN OESNN-UAD

| Notation | Description | Value |
|----------|-------------|-------|
| X        | Stream of input data |       |
| W_t      | Time window of input values |       |
| W_{size} | Size of time window |       |
| x_t      | Input value at time t |       |
| y_t      | OESNN-UAD prediction of x_t |       |
| Y        | Vector of predicted values |       |
| u_t      | Result of anomaly detection for input value x_t |       |
| U        | Vector of results of anomaly detection for input values |       |
| NO       | Input neurons |       |
| N_{size} | Number of input neurons |       |
| TS       | Synchronization time of input neurons firings |       |
| n_j      | j-th neuron in the set NO of input neurons |       |
| p_{GRF}  | GRF center for input neuron n_j |       |
| C_{GRF}  | GRF width for input neuron n_j |       |
| y_{max}  | Maximal input value in window W_t |       |
| y_{min}  | Minimal input value in window W_t |       |
| E_{exc_t}(x_t) | Excitation of GRF of neuron n_j for value x_t |       |
| T_{inj}(x_t) | Firing time of input neuron n_j for value x_t |       |
| \gamma_{NI} | Mean and variance of input values in W_t |       |
| N        | Normal distribution |       |
| NO       | Repository of output neurons |       |
| N_{size} | Number of output neurons in repository NO |       |
| s_t      | User-specified threshold |       |
| n_{ij}   | i-th output neuron from repository NO |       |
| w_{ni}   | Weight of a synapse between n_j \in NO and n_i \in NO |       |
| \gamma    | Actual post-synaptic threshold of output neurons |       |
| \psi_{ni} | Output value of output neuron n_i |       |
| \tau_{ni} | Initial or update time of output neuron n_i |       |
| M_{ni}   | Number of updates of output neuron n_k |       |
| PSP_{max} | Maximal post-synaptic threshold of output neurons |       |
| \gamma_{ni} | Fraction of \gamma_{PSP_{max}} for calculation of \gamma_{ni} |       |
| \epsilon  | Error between input value x_t and its prediction y_t |       |
| E         | Vector of error values between X and Y |       |
| \varepsilon | Anomaly classification factor | \geq 2 |

\[
\gamma_{ni} = \frac{\gamma_{ni} + M_{ni} \cdot \gamma_{ni}}{M_{ni} + 1}. \tag{11}
\]

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

D. Properties of a Neuronal Model of Output Neurons

In this subsection, we formulate and prove properties of the candidate output neurons and the output neurons in NO.

Lemma 1: For each candidate output neuron \( n_{nc} \), the following holds:

(i) the sum of its synaptic weights equals \( N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \),

(ii) \( \text{PSP}_{max} = \frac{N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk}}{M_{ni}} \),

(iii) \( \gamma_{ni} = C \cdot \frac{N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk}}{M_{ni}} \).

Proof. By Eq. (6), the vector of synaptic weights of any candidate output neuron, say \( n_{nc} \), consists of the following \( N_{size} \) elements: \( \text{mod}^{0}, \text{mod}^{1}, \ldots, \text{mod}^{N_{size}-1} \) (which may be stored in different order in distinct candidate vectors), and \( \text{PSP}_{max} \) value of \( n_{nc} \) is the sum of the squares of these elements by Eq. (7). Thus, the sum of all synaptic weights of each candidate output neuron trivially equals \( N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \) and \( \text{PSP}_{max} = \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \). By Eq. (8),

\( \gamma_{nc} = C \cdot \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \).

Theorem 1: For each output neuron \( n_i \in NO \), the following holds:

(i) the sum of its synaptic weights equals \( N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \),

(ii) \( \text{PSP}_{max} = \frac{N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk}}{M_{ni}} \),

(iii) \( \gamma_{ni} = C \cdot \frac{N_{size} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk}}{M_{ni}} \).

Proof. The proof follows immediately from Lemma 1 in the case when an output neuron, say \( n_i \), is composed of one candidate output neuron; that is, when \( M_{ni} = 1 \). Now, we will focus on the case when output neuron \( n_i \) is constructed from \( M_{ni} \), where \( M_{ni} > 1 \), candidate output neurons: \( n_{i1}, n_{i2}, \ldots, n_{iM_{ni}} \). Then, by Eq. (9), the vector of synaptic weights of output neuron \( n_i \) is the average of the vectors of synaptic weights of these candidate vectors. Hence and by Lemma 1(i), the sum of the synaptic weights of output neuron \( n_i \) equals \( \frac{1}{M_{ni}} \sum_{l=1}^{M_{ni}} (\text{PSP}_{max})_{n_{i_l}} = \frac{1}{M_{ni}} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \).

By Eq. (10), \( \text{PSP}_{max} \) of output neuron \( n_i \) is the average of \( \text{PSP}_{max} \) of candidate vectors \( n_{i1}, n_{i2}, \ldots, n_{iM_{ni}} \).

Hence, and by Lemma 1(ii), \( \text{PSP}_{max} = \frac{1}{M_{ni}} \sum_{l=1}^{M_{ni}} (\text{PSP}_{max})_{n_{i_l}} = \frac{1}{M_{ni}} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \). By Eq. (11), \( \gamma_{ni} \) is the average of \( \gamma \) of candidate vectors \( n_{i1}, n_{i2}, \ldots, n_{iM_{ni}} \). Hence, and by Lemma 1(iii), \( \gamma_{ni} = \frac{1}{M_{ni}} \sum_{l=1}^{M_{ni}} (\gamma_{n_{i_l}}) = \frac{1}{M_{ni}} \sum_{l=1}^{M_{ni}} (C \cdot \text{PSP}_{max})_{n_{i_l}} = \frac{1}{M_{ni}} \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \).

Corollary 1 follows immediately from Theorem 1 and the fact that \( \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \) and \( \sum_{k=0}^{N_{size}-1} \text{mod}^{dk} \) are sums of \( N_{size} \) consecutive elements of geometric series:

Corollary 1: For each output neuron \( n_i \in NO \), the following holds:

(i) the sum of its synaptic weights equals \( \frac{1-\text{mod}^{N_{size}}}{1-\text{mod}^{dk}} \),

(ii) \( \text{PSP}_{max} = \frac{1-\text{mod}^{N_{size}}}{1-\text{mod}^{dk}} \),

(iii) \( \gamma_{ni} = C \cdot \frac{1-\text{mod}^{N_{size}}}{1-\text{mod}^{dk}} \).

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...
and their $\gamma$, respectively. The property related to the $\gamma$ threshold will be used in our proposed algorithm for detecting anomalies.

E. Learning Algorithm of OeSNN-UAD

In the course of learning of (O)eSNN as presented in [4], the weights of synapses between input neurons and output neurons are shaped during the supervised learning of the network. The proposed learning method of OeSNN-UAD does not use supervised learning. The OeSNN-UAD approach to anomaly classification can be summarized by its two phases performed for each input value $x_t$ of stream data obtained at time $t$:

1) The sliding window $W$ is updated with value $x_t$ and GRFs of input neurons are initialized. The value $x_t$ of $W$ is used to calculate firing times and firing orders of neurons in NO. Next, the input value $x_t$ is classified as anomalous or not in three steps: a neuron $n_f \in \text{NO}$, which fired first is obtained. The output value of $n_f$ is not classified as anomalies. In order to obtain the first firing output neuron $n_f$, its postsynaptic potential $PSP_{n_f}$ is calculated according to Algorithm 4. After the first iteration in which $PSP$ of at least one output neuron is greater than the $\gamma$ threshold (and by this, its $PSP$ is also greater than the $\gamma$ threshold), no other iterations are carried out. In such a case, each output neuron identified so far whose $PSP$ is greater than $\gamma$ is added to the ToFire list. $n_f$ is found as this output neuron in ToFire that has the greatest value of $PSP$, and is returned as the result of the $\text{FIREFIRST}$ 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 $PSP > \gamma$ is found, the $\text{FIREFIRST}$ returns None to indicate that no output neuron in NO was fired. In this case, 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 $\text{NoN}$ and $+\infty$, respectively. Otherwise, the prediction of network $y_t$ is equal to $v_{n_f}$ and the absolute difference between $x_t$ and $y_t$ is set as the value of error $e_t$.

The $\text{CLASSIFYANOMALY}$ function, given in Algorithm 8, performs anomaly classification (which we describe in more detail in the next subsection) It returns $u_t$ Boolean value indicating presence or absence of an anomaly for the input value $x_t$. If the anomaly is not detected ($u_t$ is $\text{False}$), then the $\text{VALUECORRECTION}$ function is called with parameters $n_f$ and $x_t$. The $\text{VALUECORRECTION}(n_f, x_t)$ function, presented in Algorithm 9, adjusts the output value $v_{n_f}$ (reported as a prediction $y_t$ at time $t$) to the input $x_t$ value. Specifically, the value $v_{n_f}$ is increased or decreased by the factor $\xi \in (0, 1]$ of the difference $(x_t - v_{n_f})$.

In step 22 of Algorithm 1 a new candidate output neuron $n_c$ is created, and initialized by function $\text{INITIALIZENEURON}$, presented in Algorithm 5. Function $\text{INITIALIZENEURON}$ first creates synapses between the new candidate neuron $n_c$ and each input neuron in NI. Then, the initial synapses weights are calculated according to the firing orders of input neurons in NI obtained for an input value $x_t$. Next, the output value $v_{n_c}$ of $n_c$ is generated from a normal distribution created based on input values currently falling into window $W$ and finally the initialization time $\tau_{n_c}$ is set to current input value time $t$.

Step 26 of Algorithm 1 calls function $\text{FINDMOSTSIMILAR}$, presented in Algorithm 8, which finds output neuron $n_s \in \text{NO}$ such that the Euclidean distance $D_{n_s,n_c}$ between vectors of synapses weights of $n_c$ and $n_s$ is the smallest. If $D_{n_s,n_c}$ is less than or equal to the value $\text{sim}$ threshold, then $n_s$ is updated according to the function $\text{UPDATENEURON}$ presented in Algorithm 5 and $n_c$ is discarded. Otherwise, if the number of output neurons in repository NO did not reach $NO_{size}$ yet, then $n_c$ is added to NO and counter $\text{CNO}_{size}$ is incremented. If both the similarity condition is not fulfilled and the NO
repository is full, then candidate output neuron \( n_c \) replaces the oldest neuron in NO.

Function UpdateNeuron, presented in Algorithm 5, updates the vector of synapses weights of output neuron \( n_s \) as well as its output value and initialization time. The updated values are weighted averages of all previous \( (M_{n_s}) \) values of output neuron \( n_s \) and the respective values of candidate output neuron \( n_c \). Eventually, the procedure increments the update counter \( M_{n_s} \), of output neuron \( n_s \).

Algorithm 1 The proposed learning algorithm of OeSNN-UAD

**Input:** \( X = [x_1, x_2, \ldots, x_T] \) - stream of input data.

**Assure constant:**
- \( W_{size}, NO_{size}, TS, NI_{size}, mod, C, sim, \xi, \epsilon \)

**Output:** \( U \) - a vector with classification of each \( x \in X \) as an anomaly or not.

1: \( CNO_{size} \leftarrow 0 \)
2: \( \gamma \leftarrow C \cdot \frac{1}{1 - \text{mod}^2} \cdot \frac{1}{1 - \text{mod}} \)
3: \( \text{Initialize } W_i \) with \( x_1, \ldots, x_{W_{size}} \in X \)
4: \( \text{Initialize } y_1, \ldots, y_{W_{size}} \) with random values from \( N(\mu_W, \sigma_W^2) \) and add to \( Y \)
5: \( \text{Initialize } E \) with \( e_t \leftarrow |x_t - y_i| \), \( l = 1, \ldots, W_{size} \)
6: \( \text{Set } u_1, \ldots, u_{W_{size}} \to False \) and add to \( U \)
7: \( \text{for } t \leftarrow W_{size} + 1 \) to \( T \) do
8: \( \text{Update time window } W_i \) with value \( x_t \)
9: \( \text{InitializeGRFs}(W_i) \)
10: \( n_f \leftarrow \text{FireFirst}(CNO_{size}) \)
11: \( \text{if None of output neurons in NO fired then} \)
12: \( y_t \leftarrow \text{NaN}; \) append \( y_t \) to \( Y \)
13: \( e_t \leftarrow +\infty; \) append \( e_t \) to \( E \)
14: \( u_t \leftarrow \text{False} \)
15: \( \text{else} \)
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: \( u_t \leftarrow \text{ClassifyAnomaly}(E, U) \)
19: \( \text{end if} \)
20: \( \text{Append } u_t \) to \( U \)
21: \( \text{Create a candidate output neuron } n_c \)
22: \( n_c \leftarrow \text{InitializeNeuron}(W_i, t) \)
23: \( \text{if } u_t \leftarrow \text{False} \) then
24: \( v_n \leftarrow \text{ValueCorrection}(n_c, x_t) \)
25: \( \text{end if} \)
26: \( n_s \leftarrow \text{FindMostSimilar}(n_c) \)
27: \( \text{if } D_{n_c, n_s} \leq \text{sim} \) then
28: \( \text{UpdateNeuron}(n_s, n_c) \)
29: \( \text{else if } CNO_{size} < NO_{size} \) then
30: \( \text{Insert } n_c \) to \( NO; CNO_{size} \leftarrow CNO_{size} + 1 \)
31: \( \text{else} \)
32: \( n_{\text{oldest}} \leftarrow \text{an output neuron in NO such that} \)
33: \( \tau_{n_{\text{oldest}}} \leftarrow \text{min}\{\tau_{n_i}|i = 0, \ldots, NO_{size} - 1\} \)
34: \( \text{Replace } n_{\text{oldest}} \) with \( n_c \) in \( NO \)
35: \( \text{end if} \)
36: \( \text{return } U \)

Algorithm 2 Initialize GRFs

**Input:** \( W_i = \{x_1 - (W_{size} - 1), \ldots, x_t\} \) window of input values of \( X \).

1: \( \text{procedure InitializeGRFs}(W_i) \)
2: \( \text{Obtain current } t^W_{\text{min}} \) and \( t^W_{\text{max}} \) from \( W \)
3: \( \text{for } j \leftarrow 0 \) to \( NI_{size} - 1 \) do \( \triangleright \) For all neurons in \( NI \)
4: \( \text{Calculate } \mu^n_{GRF} \) and \( \sigma^n_{GRF} \)
5: \( \text{Calculate excitation } E \cdot \text{c}(n_j(x_t)) \)
6: \( \text{Calculate firing time } T(n_j(x_t)) \)
7: \( \text{end for} \)
8: \( \text{for } j \leftarrow 0 \) to \( NI_{size} - 1 \) do
9: \( \text{Calculate order}(j) \)
10: \( \text{end for} \)
11: \( \text{end procedure} \)

Algorithm 3 Initialize neuron \( n_c \)

**Input:** \( W_i \) - current time window, \( t \) - time of current input value \( x_t \)

**Output:** \( n_c \) - a newly created and initialized candidate output neuron

1: \( \text{function InitializeNeuron}(W_i) \)
2: \( \text{Create new neuron } n_i \)
3: \( \text{for } j \leftarrow 0 \) to \( NI_{size} - 1 \) do
4: \( \text{Create synapse between } n_j \in NI \) and \( n_c \)
5: \( \text{end for} \)
6: \( \text{for } j \leftarrow 0 \) to \( NI_{size} - 1 \) do \( \triangleright \) Calculate \( w_{n_j n_c} \)
7: \( w_{n_j n_c} \leftarrow \text{mod}(\text{order}(j)) \)
8: \( \text{end for} \)
9: \( v_{n_c} \leftarrow \text{Generate output value from } N(\mu_W, \sigma_W^2) \)
10: \( \tau_{n_c} \leftarrow t \)
11: \( M_{n_c} \leftarrow 1 \)
12: \( \text{return } n_i \)
13: \( \text{end function} \)

Algorithm 4 Find the most similar neuron to \( n_c \)

**Input:** \( n_c \) - a candidate output neuron.

**Output:** \( n_s \) - the neuron in NO such that Euclidean distance between \( W_{n_c} \) and \( W_{n_s} \) is least.

1: \( \text{function FindMostSimilar}(n_c) \)
2: \( \text{for } i \leftarrow 0 \ldots CNO_{size} - 1 \) do
3: \( D_{n_c, n_i} \leftarrow \text{dist}(W_{n_c}, W_{n_i}) \)
4: \( \text{end for} \)
5: \( n_s \leftarrow \text{an output neuron in NO such that } D_{n_c, n_s} = \min\{D_{n_c, n_i}|i = 0, \ldots, CNO_{size} - 1\} \)
6: \( \text{return } n_s \)
7: \( \text{end function} \)

F. Anomaly Classification

Given input value \( x_t \) at moment \( t \) obtained from the input stream and prediction \( y_t \) of this value made by OeSNN-UAD, the aim of the anomaly classification module is to decide if either \( x_t \) should be classified as an anomaly or not. The approaches proposed in the literature such as presented in [32], [17], [14], [20] either simply calculate an error between the predicted and the real value, compare it against a threshold value and decide if an anomaly occurred or a window of
Algorithm 5 Update neuron $n_s$

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

1: procedure UPDATENEURON($n_s, n_e$)
2: \hspace{1em} $w_{n_s} \leftarrow (w_{n_s} + M_{n_s} \cdot w_{n_s})/(M_{n_s} + 1)$
3: \hspace{1em} $v_{n_s} \leftarrow (v_{n_e} + M_{n_e} \cdot v_{n_e})/(M_{n_e} + 1)$
4: \hspace{1em} $\tau_{n_s} \leftarrow (\tau_{n_e} + M_{n_e} \cdot \tau_{n_e})/(M_{n_e} + 1)$
5: \hspace{1em} $M_{n_s} \leftarrow M_{n_s} + 1$
6: end procedure

Algorithm 6 Returns an output neuron which fires first

Input: $CNO_{\text{size}}$ - current size of output repository NO

Output: $n_f$ - an output neuron $\in$ NO which fires first

1: function FIREFIRST($CNO_{\text{size}}$)
2: \hspace{1em} ToFire $\leftarrow \emptyset$
3: \hspace{1em} SNIID $\leftarrow$ the list of identifiers of input neurons in NI obtained by sorting input neurons increasingly according to their order value
4: for $i \leftarrow 0$ to $CNO_{\text{size}} - 1$ do
5: \hspace{2em} $PSP_n_i \leftarrow 0$
6: end for
7: for $j$ $\leftarrow$ first to last input neuron identifier on list SNIID do
8: \hspace{2em} for $i \leftarrow 0$ to $CNO_{\text{size}} - 1$ do $\triangleright$ output neuron ids
9: \hspace{3em} $PSP_{n_i} \leftarrow PSP_{n_i} + w_{n_j n_i} \cdot \text{modorder}(j)$
10: \hspace{3em} if $PSP_{n_i} > \gamma$ then
11: \hspace{4em} Insert $n_i$ to ToFire
12: \hspace{3em} end if
13: end for
14: if ToFire $\neq \emptyset$ then
15: \hspace{2em} $n_f$ $\leftarrow$ an output neuron in ToFire such that $PSP_{n_f} = \max\{PSP_{n_i} | n_i \in \text{ToFire}\}$
16: \hspace{1em} return $n_f$
17: end if
18: end for
19: return None
20: end function

Algorithm 7 Value correction function

Input: $n_f$ - a neuron which output value needs to be corrected; $x_t$ - input value at moment $t$

Output: $v_{n_f}$ - corrected output value of neuron $n_f$

1: function VALUECORRECTION($n_e, x_t$)
2: \hspace{1em} $v_{n_e} \leftarrow v_{n_e} + (x_t - v_{n_e}) \cdot \xi$
3: \hspace{1em} return $v_{n_e}$
4: end function

past errors is used to construct a statistical distribution and obtain the probability of predicting an error for $x_t$. With low probability of such an error, the observation is classified as an anomaly. In both approaches, a constant error threshold value for anomaly classification is used. [33] takes a different approach and proposes to adapt an error threshold for anomaly classification according to the changing characteristic of the stochastic process generating input data.

In our approach presented in Algorithm 8, a vector of error values calculated between predicted and observed values of window $W_t$ is used to decide if either observation $x_t$ should be classified as anomalous or not. The error between $x_t$ and its prediction $y_t$ is calculated as an absolute difference between these two values: $e_t = |x_t - y_t|$. Given vector $E = [e_1, \ldots, e_t]$ of error values obtained for all input values of $X$ already presented to the network and their predictions in $Y$, a vector $e$ of such past $W_{size}$ error values of $E$, whose respective input values $x$ were not classified as anomalies is obtained. If $e$ is empty, then the procedure returns False, which indicates absence of an anomaly for $x_t$. Otherwise, the mean $\bar{e}$ and the standard deviation $s^2_e$ of error values in $e$ are calculated and used to classify $x_t$ either as an anomaly or not. If the difference between values $e_t$ and $\bar{e}$ is greater than $\epsilon \cdot s^2_e$, where $\epsilon$ is a user-defined parameter, then $x_t$ is classified as an anomaly, otherwise it is not.

Algorithm 8 Anomaly classification in OeSNN-UAD

Input: $E = [e_1, \ldots, e_t]$ - vector of error values; $U = [u_1, \ldots, u_{t-1}]$ - 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 anomaly or not.

1: function CLASSIFYANOMALY($E$, $U$)
2: \hspace{1em} $e \leftarrow \emptyset$
3: \hspace{1em} Append to $e$ all $e_k$ such that: $k = t - (W_{size} - 1), \ldots, t - 1$ and $u_k$ is False
4: \hspace{1em} if $e = \emptyset$ then
5: \hspace{2em} $u_t = False$
6: \hspace{2em} else
7: \hspace{3em} Calculate $\bar{e}$ and $s^2_e$ over $e$
8: \hspace{3em} if $e_t - \bar{e} \geq \epsilon \cdot s^2_e$ then
9: \hspace{4em} $u_t = True$
10: \hspace{3em} else
11: \hspace{4em} $u_t = False$
12: \hspace{3em} end if
13: \hspace{2em} end if
14: \hspace{1em} return $u_t$
15: end function

IV. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments. First we overview used anomaly benchmark datasets, then we describe experimental setup in detail with emphasis on the eSNN parameter tuning method. Then we provide anomaly detection results for the used benchmarks: Numenta Anomaly Benchmark and Yahoo Anomaly Dataset.

A. Selected Anomaly Benchmark Datasets

1) Numenta Anomaly Benchmark: Numenta Anomaly Benchmark (NAB) consists of 7 categories of datasets, both artificial and real, each of which contains 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 series of a input values. The number of input values in data files varies between 1000 and 22,000. Overall, there
are 58 data files in NAB. The current version (1.0) of NAB consists of the following categories of data files:

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

Only data files in artificialNoAnomaly category do not contain anomalies. The rest of data files contain at least one anomaly window. The anomalies occurring in data files are given in a separate file in the form of anomaly windows. Each anomaly window consists of multiple input values and each data file can have several anomaly windows. The labeling of anomaly windows in data files was conducted both by humans and algorithms. It should be noted that not necessarily all anomalies which occur in a data file are labeled. In fact, Numenta encourages volunteers to perform additional anomaly labeling of data files [34]. It was also reported in [35], that some data files in NAB contain missing values or differences in input values distributions. These reasons make NAB particularly challenging for anomaly detection algorithms.

2) **Yahoo Anomaly Dataset**: As the second benchmark for our experiments, we selected Yahoo Anomaly Dataset [36], which consists of four categories of data files:

- **A1Benchmark** - contains 67 data files with real input time series values. Both point and window anomalies occur in these data files.
- **A2Benchmark** - consists of 100 synthetic data files, which contain anomalies in the form of single outliers (point anomalies). Most of input time series values in this category have their own periodicity.
- **A3Benchmark** - which has 100 synthetic data files, with anomalies in the form of single outliers. In comparison to A2Benchmark, input values time series in this category are more noisy.
- **A4Benchmark** - contains 100 synthetic data files, where anomalies are mainly sudden step changes in input values of time series.

**B. Experimental Setup**

In the experimental phase, we aim to compare anomaly detection quality of our approach with other state-of-the-art methods and algorithms provided in the literature. In particular, we report three measures of detection quality: precision, recall and F-measure. Precision provides information on how many of input values detected as anomalies by the detector are actually labeled as anomalies in data files. On the other hand, recall tells how many of the labeled anomalies in the data file are properly detected by the detector. F-measure is defined as a harmonic mean of precision and recall. In Eqs. (12)-(14) we give formulas for precision, recall and F-measure. In these equations, TP (True Positive) refers to the number of input values which were both classified as anomalies by the detector and labeled as being such in the data file, FP (False Positive) is the number of input values incorrectly classified as being anomalies by the detector, while FN (False Negative) is the number of input values labeled in the data file as anomalies but not properly classified by the detector as being such.

\[
\text{Precision} = \frac{TP}{TP + FP}. \quad (12) \\
\text{Recall} = \frac{TP}{TP + FN}. \quad (13) \\
\text{F-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (14)
\]

For the correct detection of anomalies with OeSNN-UAD, it is especially important to correctly select values of window size \(W_{\text{size}}\) and anomaly classification \(\varepsilon\) parameters. The selection of proper learning parameters for data stream processing algorithms is not a trivial task [37]. We used the grid search procedure to find the best values of parameters \(W_{\text{size}}\) and \(\varepsilon\) for each data file separately. The implemented grid search iterates over all given combinations of learning parameters to find a set of parameters (in particular, \(W_{\text{size}}\) and \(\varepsilon\)), which provides the best anomaly detection results for a given data file. Such an approach to grid search over learning parameters is motivated by the following objectives:

- a similar grid search procedure is used in Numenta Anomaly Benchmark and Yahoo Anomaly Benchmark. In particular, Numenta Anomaly Benchmark optimizes anomaly detection threshold of each implemented algorithm for each data files category separately.
- a series of input values (even for data files in the same category) often has its own characteristic and for most data files it is not possible to optimize the learning parameters on a selected subset of data files in a given category and use these parameters for other data files in the same category.
- for most of the data files in both benchmarks it is not possible to divide time series of each data file into a validation part (which can be possibly used for grid search over a given combination of parameters) and a test part (which is used with the parameters set, which gives the best results on the validation part). This is because the input time series usually contain only a few anomalies or anomaly windows, so splitting the data file into validation and test parts can result in incorrectly labeled anomalies or the lack of them in either validation or test parts.

Most of the parameters of Numenta Anomaly Benchmark and Yahoo Anomaly Dataset are set to single values. These are namely: \(NO_{\text{size}} = 50, NI_{\text{size}} = 10, TS = 1000, \text{sim} = 0.17, \text{mod} = 0.6, C = 0.6, \xi = 0.9\). The possible values of parameters \(W_{\text{size}}\) and \(\varepsilon\) for Numenta Anomaly Benchmark are: \(W_{\text{size}} = \{100, 200, \ldots, 600\}\) and \(\varepsilon = \{2, 3, \ldots, 7\}\), respectively, while for Yahoo Anomaly Datasets the possible
values of these parameters are: $W_{\text{size}} = \{20, 40, \ldots, 500\}$ and $\varepsilon = \{2, 3, \ldots, 17\}$. The total number of neurons in the network is limited to 60 (10 input neurons and 50 output neurons) to minimize the learning and response times of eSNN.

The experiments were performed on a computer equipped with Intel Core i7-8750H CPU and 16.0 GB of RAM memory. The implementation of OeSNN-UAD is prepared in C++ and source code of the implementation is publicly available\footnote{https://github.com/piotrMaciag32/eSNN-AD}. The compiled executable file is very lightweight (it consumes around 2 MB) of RAM memory, which makes it additionally suitable for environments with very limited memory constraints, such as sensor microcontrollers or IoT devices.

C. Overview of the obtained anomaly detection results

In Fig. 3 we present charts showing the results obtained for the selected data files for both benchmarks. For each data file, the first chart shows input values time series ($X$), the second one presents predicted values $Y$, the third one illustrates an error between the input and the predicted values ($E = |X - Y|$) and the last chart shows the detected and labeled anomalies. For example, it can be noted that anomalies for the file ec2_cpu_utilization_ac2cd presented in Fig. 3 are correctly detected by our approach. OeSNN-UAD is able to detect anomalies which occur both around timestamp 500 and are not labeled in the data file, as well as the labeled anomalies present after timestamp 3500.

In Table [1], we give the results of OeSNN-UAD anomaly detection for Numenta Anomaly Benchmark as compared to other unsupervised anomaly detection methods and algorithms. Similarly to the comparison of the results presented recently in [14], we report the mean F-measure obtained for each category of data files. It can be noted, that OeSNN-UAD significantly outperforms the results obtained by other approaches for each category of data files. Table [3] presents the obtained precision and recall values for the selected data files from Numenta Anomaly Benchmark. For some data files eSNN is able to provide much higher values of both precision and recall than other methods. The lower precision value of OeSNN-UAD obtained for some data files compared to other methods is caused by two reasons:

- Most of other methods and algorithms for which the results are presented in Table [3] discover very few anomalies (which is indicated by high values of precision and very low values of recall) and is described in detail in [35].
- As it was indicated above, some input values in data files are not labeled as anomalies.

In Table [4] we present comparison of the obtained F-measure values for each category of data files in the Yahoo Anomaly Dataset. For the real data files category (ALBenchmark), the proposed OeSNN-UAD approach provides better results than recent results reported in the literature, while for the other three categories of data files OeSNN-UAD is competitive to the reported results.

V. Conclusions

In this article, we offered a novel approach to unsupervised and online detection of anomalies in data streams called OeSNN-UAD. The proposed detector is designed for univariate stream time series data and extends evolving Spiking Neural Networks architecture and their learning principles. To the best of our knowledge, for the first time we introduced an eSNN architecture, which can be trained in an unsupervised mode, in which input values do not need to be labeled with decision classes. Several distinctive features of OeSNN-UAD allow for proper learning and classification of input values:

- the proposed eSNN architecture learns within classification process based on the recent window of input values.
- with each new input value presented to OeSNN-UAD, a new output neuron is created and initialized. Contrary to a typical eSNN architecture, the newly created neuron is not assigned an output decision class. Instead, it is initialized with a random real output value. The output value of an output neuron is then modified in the course of learning of the eSNN network.
- we introduced a method of anomaly classification based on the history of past prediction errors of input values.

In the article, we have also proved that all output neurons have the same values of the sum of their synaptic weights, their maximal post-synaptic thresholds, and their actual post-synaptic potential thresholds, respectively. The last property eliminate the necessity of recalculating of the actual post-synaptic potential thresholds when output neurons of eSNN are updated within the learning process.

In the experimental part, we compared the proposed OeSNN-UAD approach to the results of unsupervised and semi-supervised anomaly detection methods and algorithms recently reported in the literature. Experiments were conducted on two anomaly benchmark datasets: Numenta Anomaly Benchmark and Yahoo Anomaly Dataset, which consist of more than 500 data files grouped into several categories. For the assessment of the quality of anomaly detection of our approach and other state-of-the-art approaches, we used three indicators: F-measure, precision and recall. For Numenta Anomaly Benchmark, OeSNN-UAD is able to provide significantly better results in terms of F-measure for all categories of data files. Detailed analysis of precision and recall obtained for the selected data files in Numenta Anomaly Benchmark shows that OeSNN-UAD is competitive in terms of both of these measures to the results reported in the literature. For the second selected benchmark, the Yahoo Anomaly Dataset, OeSNN-UAD provides higher F-measure value for real data files category, while for the other three synthetic data categories the obtained values of F-measure are competitive to the results reported in the literature.

The important property of OeSNN-UAD approach is its ability to learn and classify input values efficiently. Moreover, OeSNN-UAD is suitable for environments with very restricted memories. As we prove in the experimental part, the network is able to outperform other methods when it consists of only 10 neurons in the input layer and the size of output neuron repository is limited to 50.
Fig. 3. The results of anomaly detection with OeSNN-UAD for four example data files in Numenta Anomaly Benchmark and Yahoo Anomaly Dataset.

ACKNOWLEDGMENT

P. Maciag acknowledges financial support of the Faculty of the Electronics and Information Technology of the Warsaw University of Technology (grant no. II/2019/GD/1). J. L. Lobo and J. Del Ser would like to thank the Basque Government for their support (ELKARTEK and EMAITEK programs).

REFERENCES

[1] F. Ponulak and A. Kasinski, “Introduction to spiking neural networks: Information processing, learning and applications.” Acta neurobiologiae experimentalis, vol. 71, no. 4, pp. 409–433, 2011.
[2] B. Petro, N. Kasabov, and R. M. Kiss, “Selection and optimization of temporal spike encoding methods for spiking neural networks.” IEEE Transactions on Neural Networks and Learning Systems, pp. 1–13, 2019.
[3] P. S. Maciag, N. Kasabov, M. Kryszkiewicz, and R. Bembenik, “Air pollution prediction with clustering-based ensemble of evolving spiking neural networks and a case study for London area,” Environmental Modelling & Software, vol. 118, pp. 262 – 280, 2019.
[4] J. L. Lobo, I. Laña, J. Del Ser, M. N. Bilbao, and N. Kasabov, “Evolving spiking neural networks for online learning over drifting data streams,” Neural Networks, vol. 108, pp. 1 – 19, 2018.
[5] N. Kasabov, K. Dholbe, N. Nuntalid, and G. Indiveri, “Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition,” Neural Networks, vol. 41, pp. 188 – 201, 2013, special Issue on Autonomous Learning.
[6] N. K. Kasabov, “Evolving connectionist systems for adaptive learning and knowledge discovery: Trends and directions,” Knowledge-Based Systems, vol. 80, pp. 24 – 33, 2015, 25th anniversary of Knowledge-Based Systems.
[7] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, “Unsupervised real-time anomaly detection for streaming data,” Neurocomputing, vol. 262, pp.
| Dataset category       | BayesAn Changepoint | Context OSE | EXPSEQ | HTM JAVA | KNN CAD | NumentaTM | 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       | 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   |           |

**Table II**

Comparison of precision and recall obtained for the state of the art methods and the proposed OeSNN-UAD approach for selected data files from the Numenta Anomaly Benchmark (the results for methods marked with * are given in [14], DeepAnt was also introduced there). The bolded results are the best for each data files category.

| Dataset category       | Time series       | ContextOSE* Prec. | ContextOSE* Rec. | NumentaTM* Prec. | NumentaTM* Rec. | Skyline* Prec. | Skyline* Rec. | ADVec* Prec. | ADVec* Rec. | DeepAnt* Prec. | DeepAnt* Rec. | OeSNN-UAD Prec. | OeSNN-UAD Rec. |
|-----------------------|-------------------|-------------------|-----------------|------------------|-----------------|----------------|--------------|--------------|--------------|----------------|---------------|----------------|----------------|
| Real Ad Exchange      | exchange-2-cpc-results | 0.5               | 0.006           | 0                 | 0                | 0               | 0             | 0.03         | 0.33         | 0.07           | 0.02          |                |                |
| Real AWS Cloud Watch  | ec2-cpu-utilization-50533 | 1                 | 0.005           | 1                 | 0.002           | 1               | 0.002         | 1             | 0.01         | 0.18           | 0.51          |                |                |
| Real Known Cause      | ambient-temperature-system-failure | 0.33           | 0.001           | 0.5               | 0.006           | 0               | 0             | 0.26         | 0.06         | 0.21           | 0.75          |                |                |
| Real Known Cause      | cpu-utilization-asm-misconfiguration | 0.12             | 0.001           | 0.52              | 0.01            | 0               | 0.74          | 0.01         | 0.63         | 0.36           | 0.49          |                |                |
| Real Traffic          | occupancy-6005     | 0.5               | 0.004           | 0.4               | 0.004           | 0               | 0.004         | 0.5           | 0.004        | 0.18           | 0.41          |                |                |
| Real Traffic          | speed-6005         | 0.5               | 0.004           | 0.25              | 0.008           | 0               | 0.04          | 1             | 0.02         | 0.036          | 0.50          |                |                |
| Real Traffic          | speed-7578         | 0.5               | 0.037           | 0.6               | 0.02            | 0.86           | 0.16          | 1             | 0.01         | 0.07           | 0.64          |                |                |
| Real Traffic          | speed-64013        | 0.1               | 0.008           | 0.8               | 0.01            | 1               | 0.06          | 0             | 0.01         | 0.08           | 0.31          |                |                |
| Real Traffic          | TravelTime-387     | 0.6               | 0.01            | 0.33              | 0.004           | 0.62           | 0.07          | 0.2           | 0.004        | 0.04           | 0.22          |                |                |
| Real Traffic          | TravelTime-451     | 1                 | 0.005           | 0                 | 0               | 0             | 0             | 0             | 0.009        | 0.02           | 0.82          |                |                |
| 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          |                |                |
| Real Tweets           | Twitter-volume-IBM  | 0.37             | 0.002           | 0.22              | 0.005           | 0.22           | 0.01         | 0.5           | 0.009        | 0.5             | 0.005         |                |                |

134 – 147, 2017, online Real-Time Learning Strategies for Data Streams.

[8] J. Hawkins and S. Ahmad, “Why neurons have thousands of synapses, a theory of sequence memory in neocortex,” Frontiers in Neural Circuits, vol. 10, p. 23, 2016.

[9] A. Stanway. (2015) Etsy skyline. [Online]. Available: [https://github.com/etsy/skyline](https://github.com/etsy/skyline)

[10] A. Kejariwal. “Introducing practical and robust anomaly detection in a time series,” 2015. [Online]. Available: [https://blog.twitter.com/engineering/en_us/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).

[11] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” ACM Comput. Surv., vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[12] B. Rosner, “Percentage points for a generalized esd many-outlier procedure,” Technometrics, vol. 25, no. 2, pp. 165–172, 1983.

[13] N. Laptev, S. Amizadeh, and I. Flint, “Generic and scalable framework for automated time-series anomaly detection,” in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD ’15. New York, NY, USA: ACM, 2015, pp. 1939–1947.
| 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)* | OesSNN-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        |

[14] M. Munir, S. A. Siddiqui, M. A. Chattha, A. Dengel, and S. Ahmed, “Deepant: A deep learning approach for unsupervised anomaly detection in time series,” IEEE Access, vol. 7, pp. 1991–2005, 2019.

[15] P. R. Adams and D. J. C. MacKay, “Bayesian online changepoint detection,” CoRR, vol. abs/0710.3742, 2007. [Online]. Available: abs/0710.3742

[16] M. Schneider, W. Ertel, and F. Ramos, “Expected similarity estimation for large-scale batch and streaming anomaly detection,” Machine Learning, vol. 105, no. 3, pp. 305–333, Dec 2016.

[17] E. Burnaev and V. Ishmiev, “Conformalized density- and distance-based anomaly detection in time-series data,” 2016.

[18] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, “Statistical techniques for online anomaly detection in data centers,” in 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, May 2011, pp. 385–392.

[19] M. Smirnov. (2016) Contextual anomaly detector. [Online]. Available: https://github.com/smirmik/CAD

Piotr S. Maciag is a research assistant and a PhD student at the Warsaw University of Technology, Poland. His research interests are with data mining and machine learning: discovering sequential patterns from spatio-temporal data, mining frequent itemsets, spiking neural networks, time series processing, clustering and their applications to such areas as air pollution prediction or crime incidents analysis. In the academic year 2017/2018, he was a research visitor in Knowledge Engineering and Discovery Research Institute (KEDRI) at Auckland University of Technology. In January and February 2019, he visited OPTIMA research group at Tecnalia Research and Innovation in Basque Country. Currently, he conducts his research visit at the Robotics Institute of Carnegie Mellon University.

Marzena Bogumiła Kryszkiewicz received her PhD and DSc degrees in Computer Science from Warsaw University of Technology in 1995 and 2003, respectively. She was awarded the title of Professor by the President of the Republic of Poland in 2013. She was the Associate Dean for Academic Affairs at the Faculty of Electronics and Information Technology of Warsaw University of Technology in 2005-2008. Since 2009 she has been the head of the Division of Information Systems in Institute of Computer Science of this faculty. She was awarded with scientific prizes such as Award of the Foundation for Polish Science in 1995 and Award of Minister of Education of the Republic of Poland in 2004. In 2017, she was awarded the Medal by the Commission of National Education. Her research interests include: data mining, knowledge discovery, concise representations of knowledge, evaluation measures of association and decision rules, clustering, classification, reasoning under incompleteness and uncertainty, rough sets, privacy preserving. She has published more than 100 articles. She is a member of Program Committees and reviewer for several international conferences and journals.
**Robert Bembenik** obtained his PhD in computer science from Warsaw University of Technology in 2007, where he currently works as an assistant professor. His research interests include spatial data mining, social networks, mobile applications and text analysis. He is a co-author of over 30 scientific publications.

**Jesus L. Lobo** holds a PhD in Information and Communications Technologies in Mobile Networks from the University of the Basque Country (Spain). He works as AI researcher at the OPTIMA (Optimization, Modeling and Analytics) research area at TECNALIA RESEARCH & INNOVATION (www.tecnalia.com). His research interests are mainly focused on Machine Learning applied to data streams. Currently, he addresses the challenges of concept drift adaptation and detection for data streams, where he has so far authored several conference contributions and has published a number of papers in indexed scientific journals.

**Javier Del Ser** (M’07-SM’12) received his first PhD degree (cum laude) in Electrical Engineering from the University of Navarra (Spain) in 2006, and a second PhD degree (cum laude, extraordinary PhD prize) in Computational Intelligence from the University of Alcalá (Spain) in 2013. He is currently a Research Professor in Artificial Intelligence and leading scientist of the OPTIMA (Optimization, Modeling and Analytics) research area at TECNALIA, Spain. He is also an adjunct professor at the University of the Basque Country (UPV/EHU), an invited research fellow at the Basque Center for Applied Mathematics (BCAM), an a senior AI advisor at the technological startup SHERPA.AI. He is also the coordinator of the Joint Research Lab between TECNALIA, UPV/EHU and BCAM, and the director of the TECNALIA Chair in Artificial Intelligence implemented at the University of Granada (Spain). His research interests are in the design and application of Artificial Intelligence methods for data mining and optimization for problems emerging in Intelligent Transportation Systems, Smart Mobility and Logistics, among others. He has published more than 280 scientific articles, co-supervised 10 Ph.D. theses, edited 7 books, co-authored 9 patents and participated/led more than 40 research projects. He is an Associate Editor of tier-one journals from areas related to Data Science Artificial Intelligence, such as Information Fusion, Swarm and Evolutionary Computation and Cognitive Computation, and has served as a Guest Editor in journals from the ITS field such as IEEE Transactions on Intelligent Transportation Systems and IEEE ITS Magazine. He is a recipient of the Bizkaia Talent prize for his research career.