



M Ű E G Y E T E M 1 7 8 2

**Budapest University of Technology and Economics**  
Faculty of Electrical Engineering and Informatics  
Department of Networked Systems and Services

Dániel László Vajda

# **MACHINE LEARNING-BASED ANOMALY DETECTION**

BSc thesis

SUPERVISORS:

**Dr. Adrián Pekár**  
**Dr. Károly Farkas**

BUDAPEST, 2020



## SZAKDOLGOZAT FELADAT

**Vajda Dániel László**  
szigorló villamosmérnök hallgató részére

# Machine Learning-Based Anomaly Detection

Detecting anomalous behavior on time series data is under active research today. It has potentials in several domains, such as network performance monitoring for abnormal network behavior detection, preferably in real-time. However, most of the existing anomaly detection approaches require manual intervention and extensive domain knowledge to appropriately tune the detection engine, which is far from being a handy and generic solution.

NETvisor Ltd., an SME active in ICT systems integration in Hungary, has been pursuing research in network telemetry and anomaly detection related topics. As part of this activity, a network telemetry analytics system has been developed and implemented recently to receive, process, store, and visualize telemetry data. The next step is to build an intelligent anomaly detection engine, which can automatically adapt to pattern changes in the collected time series data and detect abnormal behavior.

The candidate's assignment is to develop and implement such an intelligent anomaly detection engine. Thus, in the frame of the bachelor thesis work, the following tasks have to be completed:

- Overview the state-of-the-art anomaly detection approaches;
- Design and implement a machine learning-based anomaly detector;
- Test and validate the developed solution;
- Compare the performance of the developed solution to other state-of-the-art anomaly detection approaches using test data sets collected from different domains;
- Examine and discuss the possibilities for further improvements.

**Tanszéki konzulensek:** Dr. Farkas Károly, egyetemi docens  
Dr. Pekár Adrián, egyetemi adjunktus

Budapest, 2020. szeptember 20.

Dr. Imre Sándor  
egyetemi tanár  
tanszékvezető

### Konzulensi vélemények:

Tanszéki konzulens:  Beadható,  Nem beadható, dátum: aláírás:

# Table of contents

<b>Hallgatói nyilatkozat</b> .....	<b>4</b>
<b>Kivonat</b> .....	<b>5</b>
<b>Abstract</b> .....	<b>6</b>
<b>1 Introduction</b> .....	<b>7</b>
<b>2 Related Work</b> .....	<b>9</b>
<b>3 Neural Networks</b> .....	<b>12</b>
3.1 Concept .....	12
3.2 Recurrent Neural Networks .....	14
3.2.1 Vanishing Gradient.....	14
3.3 Long Short-Term Memory.....	15
<b>4 State-of-the-art Algorithms Based on LSTM</b> .....	<b>16</b>
4.1 RePAD .....	16
4.1.1 LSTM Model .....	16
4.1.2 Details of the Algorithm .....	16
4.2 ReRe.....	19
4.3 Limitations of ReRe.....	20
<b>5 Alter-Re<sup>2</sup></b> .....	<b>22</b>
5.1 Ageing.....	22
5.2 Window-mode.....	23
5.3 Automated hyperparameter tuning .....	25
5.3.1 Devising a metric for the size of data points used .....	25
5.3.2 Automated tuning algorithm steps .....	27
5.3.3 Detection of extreme parameter values .....	30
<b>6 Experiments</b> .....	<b>33</b>
6.1 Preliminaries .....	33
6.2 Reference Measurement.....	35
6.3 ReRe vs. Alter-Re <sup>2</sup> .....	36
6.4 Performance Maintenance through Different Data Types .....	41
6.5 Evaluation of automated hyperparameter tuning.....	43
<b>7 Discussion</b> .....	<b>46</b>
<b>8 Research Implications</b> .....	<b>48</b>
<b>9 Conclusions</b> .....	<b>50</b>

<b>10 Acknowledgements .....</b>	<b>51</b>
<b>11 List of Figures.....</b>	<b>52</b>
<b>12 Acronyms .....</b>	<b>53</b>
<b>13 Bibliography .....</b>	<b>54</b>

## Hallgatói nyilatkozat

Alulírott **Vajda Dániel László**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. 12. 11.

.....  
Vajda Dániel László

## Kivonat

Ahogy egyre több és több eszköz csatlakozik számítógépes hálózatokhoz, úgy lesz azok infrastruktúrája egyre komplexebb. Ezen hálózati eszközök, rendszerek és szolgáltatások folyamatos felügyelete manapság lényegesebb, mint valaha. Ennek számos haszna lehet, mint például üzemzavar előrejelzése; leállások elkerülése azáltal, hogy előre azonosítjuk azok jeleit; rendszerek teljesítményének monitorozása; továbbá rendszerek biztonságának felügyelete és az esetleges támadások észlelése.

Hagyományos módszerekkel azonban ezeket a funkciókat megbízhatóan, hatékonyan, valós időben megvalósítani koránt sem egyszerű feladat. Ezt segíti elő a hálózati telemetria paradigmája, mely egy modern eljárás a hálózati eszközökből kinyerhető, idősor alapú telemetria adatok gyors, hatékony és automatikus begyűjtésére. A begyűjtött adatokat azonban fel kell dolgozni, hogy képesek legyünk detektálni a helytelen működésre utaló jeleket, amit gépi tanuló algoritmusok segítségével lehet hatékonyan megvalósítani. Ezt a folyamatot nevezzük anomáliadetekciónak.

Ez a dolgozat kifejezetten az anomáliadetekcióra összpontosít, új megvilágításba helyezve annak idősor alapú telemetria adatokon történő használatát. Az irodalomkutatás során az ún. Long Short-Term Memory (LSTM) alapú, ReRe elnevezésű algoritmust azonosítottuk, mint a jelenleg elérhető leghatékonyabb eljárás. Azonban vizsgálataink azt mutatták, hogy még ez az eljárás is számos limitációval rendelkezik. Ezért a dolgozatban bemutatjuk az algoritmus általunk továbbfejlesztett, Alter-Re<sup>2</sup>-nek elnevezett változatát, melyben az eredeti eljárást az ún. öregítés módszerével, illetve az adatok egy csúszóablakban való feldolgozásával egészítettük ki. Egy, a bevezetett módszerek hiperparamétereit automatikusan beállító algoritmust is kidolgoztunk. Az így elért teljesítményjavulás ígéretes, az Alter-Re<sup>2</sup> algoritmus átlagosan háromszor jobban, de legalább úgy teljesített, mint a ReRe tíz különböző adatsoron végzett vizsgálatainkban.

Továbbá a dolgozatban kitérünk arra, hogyan függ a ReRe és az Alter-Re<sup>2</sup> algoritmusok megbízhatósága és pontossága az elemzett adatsor típusától. Kategóriákba soroljuk a feldolgozott adatsorokat az adatok mintázatai alapján, majd elemezzük az algoritmus működését kategóriánként.

Meggyőződésünk, hogy az Alter-Re<sup>2</sup> előnyösen használható számos területen, ahol gyors és pontos anomáliadetekcióra van szükség, mint például a hálózati telemetria, IoT szenzorfolyamok, behatolók, hibák, csalások észlelése esetén.

# Abstract

The complexity of network infrastructure is growing in tandem with the number of connected devices. Infrastructure monitoring is essential as many managerial-related tasks highly depend on it. These include alerting partial or total system malfunction, outage prevention based on predictive identification of such situations, performance tracking, and, last but not least, security detection of system penetration.

However, it has become far from obvious how to achieve timely, reliable, and sound infrastructure monitoring using traditional approaches. Telemetry has been developed to streamline this process. Network devices using it generate time series telemetry data that is streamed to a central collector. This streaming time-series data has to be analysed, plausibly by machine learning-based algorithms, to detect indications of abnormal behaviour and notify administrators. This process is called anomaly detection.

In this study, we focus on anomaly detection on time-series telemetry data. We rigorously examined state-of-the-art anomaly detection methods. Specifically, we assessed ReRe, a Long Short-Term Memory-based (LSTM-based) algorithm. Although the algorithm was claimed to achieve high efficacy, our experiments revealed several limitations when applied on time-series data. Motivated by these findings, we propose in this study a modified version of ReRe, called Alter-Re<sup>2</sup>, to overcome these limitations. We introduce the concepts of ageing and sliding window as the key enablers of our extensions. Additionally, we also automated the hyperparameter tuning of our approach that can adjust the initial configuration of the algorithm. Consequently, the configuration converges to the most optimal values in just a few steps, whereas without this automation the values would need to be set manually. The resulted performance improvements are promising; compared to ReRe, our Alter-Re<sup>2</sup> can achieve three times better, but never worse performance on average when evaluating using ten different datasets.

Furthermore, we also discuss how the data type being analysed impacts the reliability and precision of ReRe and Alter-Re<sup>2</sup>. We were able to create categories based on specific data patterns and draw conclusions on ReRe for each. We hope to shed new light on time-series anomaly detection and stimulate further research in the field.

Our approach is advantageous in application domains where timely and accurate anomaly detection is essential, such as in network telemetry, IoT sensor streams, intrusion, fault, and fraud detection related tasks.

# 1 Introduction

Nowadays, infrastructure monitoring, including networks, systems, and services, is more critical than ever before. It is essential for several reasons, such as alerting partial or total system malfunction, outage prevention based on predictive identification of such situations, performance tracking, and, last but not least, security detection of system penetration.

However, with the exponential increase in the number of interconnected devices and traffic volume, it has become far from obvious how to achieve timely, reliable, and sound infrastructure monitoring. It requires understanding the details of processes inside the systems and recognize how they influence each other or the whole infrastructure. The concept of network telemetry has been introduced to streamline this goal. It allows automated, fast, and simultaneous collection of a wide variety of time-series data from a large number of devices.

Machine learning techniques can process, understand, and classify problematic infrastructure behaviours, even in massive data volumes. Despite recent significant advances in machine learning, their application to anomaly detection remains poorly understood and investigated in the network telemetry domain. This work focuses specifically on that, i.e., it attempts to shed new light on anomaly detection on time-series telemetry data.

Broadly speaking, measurement data are created by a generating process. If this generating process behaves unusually due to the system's abnormal behaviour or the entity that impacts the generating process, it produces anomalies. The manifestation of anomalous behaviour can be identified by observing the generated time-series data. Anomaly detection is a critical component of network and services management as it can provide useful insights into the operation of the network and its components.

Our survey of anomaly detection on time-series data yielded ReRe [2], a Long Short-Term Memory (LSTM) based algorithm, as the most efficient state-of-the-art approach. It is claimed to achieve high accuracy in detecting abnormal behaviour while minimizing false positives and re-trainings. However, our evaluation revealed several limitations when ReRe was applied on time-series data. As such, we extended ReRe with two additional features to improve its efficacy. We named our improved algorithm Alter- $\text{Re}^2$ . The first feature ensures that the collected data ages out; thus, its weight decreases

as time passes, allowing faster adaption to short-term history and more precise anomaly detection. The second feature serves the purpose of a sliding window that reduces the anomaly detector’s resource demands. Furthermore, we also introduce two hyperparameters to the algorithm whose adjustment is automated; eventually leading to optimal efficiency with no need for manually configuration.

The evaluation of Alter-Re<sup>2</sup> has shown promising results. We could achieve approximately three times higher, but never worse accuracy in detecting anomalies compared to the original ReRe algorithm experimenting with ten different time-series datasets. Not only could we eliminate issues preventing real-time use, but we were also able to enhance sensitivity to smaller amplitude and length anomalies. Our automated parameter tuning algorithm was shown to be successful at setting appropriate values after some timesteps for two hyperparameters – the ageing and sliding window. Furthermore, we also observed that this LSTM based anomaly detection algorithm was only directly applicable to certain data patterns. Presumably, the patterns of normal and abnormal behaviour depend on the type of data being analysed. However, the validity of this hypothesis has yet to be further investigated and will be part of our future work. Nevertheless, we argue the strong applicability of our approach in real-world scenarios.

The rest of this study is organized as follows. In Section 2, we discuss related works from the field of anomaly detection on time-series, especially network telemetry-related, data. In Section 3, we explain the basic principles of neural networks and their key concepts, including Recurrent Neural Networks (RNNs) and Long Short-Term Memory. Section 4 describes two state-of-the-art real-time anomaly detection algorithms on time-series data from 2020. Specifically, we discuss RePAD [1] and its improved version called ReRe [2] in detail. In Section 5, we present Alter-Re<sup>2</sup>, our approach to address the issues discussed above. In Section 6, we present a number of experiments to compare and contrast the original ReRe and our Alter-Re<sup>2</sup> algorithms. Section 7 is a discussion on the different types of datasets on which these algorithms can perform well. In Section 8, we discuss further research implications and lay out possible future work directions. And finally, we draw conclusions in Section 9.

## 2 Related Work

We originally focused on anomaly detectors deployed in network telemetry streams. However, we observed that there is only a handful of research in this area. Putina et al. [3] at Cisco developed a streaming telemetry-based anomaly detection engine for BGP anomalies; however, it uses a legacy clustering algorithm called DenStream [7], published in 2006, with limited performance. Furthermore, this detector only deals with BGP telemetry data. DenStream classifies incoming data into clusters of arbitrary shape. These are made up of core and outlier microclusters. An anomaly is detected when a data point is merged into an outlier microcluster. DenStream requires the calculation of certain parameters based on the whole dataset, so this method is not adapted for real-time use.

We found other, less applicable works as well related to anomalies and computer networks such as [9], [10], [11]. Ye et al. [9] use a statistical approach to detect zero-day attacks and malicious intent. The paper claims that machine learning techniques miss the bigger picture of network behaviour. Kaiafas et al. [10] use multiple unsupervised machine-learning algorithms in an ensemble to identify fraudulent private exchange phone calls, yet their approach only works on offline data. Lazaris and Prasanna [11] aim to predict fine-grained network traffic using an LSTM neural network model based on Software-Defined Networking.

As we did not come across any research directly applicable, we made a shift towards generic real-time anomaly detection algorithms. AnomalyDetectionTs (ADT) and AnomalyDetectionVec (ADV) are two anomaly detection algorithms, developed by Twitter, and available in their GitHub repository [8]. ADT works on time-series data, while ADV is designed for vectors without timestamp information. These algorithms employ statistical-based approaches, therefore require many data points. This makes them less applicable for streaming time-series data. ADV and ADT are parameter-sensitive algorithms and require a human expert to set appropriate values to achieve proper detection.

Tan et al. [6] developed a fast anomaly detection algorithm for streaming data. Their method, published in 2011, makes use of Half-Space Trees for machine learning. It processes data in one pass, requires constant memory and performs fast model updates. The algorithm has to be trained on normal data to be able to identify anomalies; that way, it cannot perform unsupervised learning. It operates using two consecutive windows, data is handled in batches instead of constantly updating.

Ahmad et al. [5] developed an unsupervised real-time anomaly detector for streaming data. In their paper, published in 2017, the authors use the Hierarchical Temporal Memory (HTM) algorithm to detect anomalies in real-time streaming data. HTM is based on neuroscientific research. It is claimed to be extremely tolerant of noisy data and to adapt to changes in the statistics of the stream. Additionally, it is said to detect subtle temporal anomalies while minimizing false positives. There are additional statistical calculations needed to adapt HTMs to our task, which makes it domain-dependent.

Maciag et al. [4] published their work in 2019. Their method performs unsupervised anomaly detection in stream data. It uses evolving Spiking Neural Networks (eSNN) for online unsupervised anomaly detection. The concept of eSNNs heavily rely on the way a human brain works. The input layer transforms the data into spikes that can be sent between neurons. The output layer is a repository; neurons are added or merged to an existing one while training. Anomaly detection is based on data classification like in [7]. It works only on univariate data streams.

Greenhouse [12] is an algorithm that combines state-of-the-art machine learning and data management techniques for anomaly prediction over high volumes of time-series data. The term ‘zero-positive’ means that the algorithm has to be trained on normal data but does not require labelled anomalies. Greenhouse uses a look-back, predict-forward approach to detect anomalies. This means, it employs an LSTM model [13] to predict new values based on old ones, then compares the prediction to the actual data point.

RePAD (Real-time Proactive Anomaly Detection for Time Series) [1] is an improvement of Greenhouse that eliminates the need for normal training data. ReRe (A Lightweight Real-time Ready-to-Go Anomaly Detection Approach for Time Series) [2] is an upgrade of RePAD by the same authors that aims to eliminate false positive anomaly detections. We give a detailed description of these two methods in Section 4.

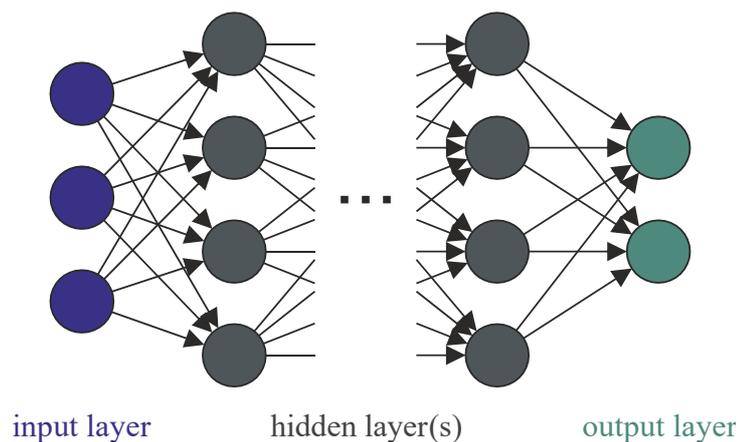
In summary, we were unable to identify any approach directly aimed at anomaly detection in streaming telemetry data using modern detection algorithms. Generic real-time anomaly detectors, however, have seen a major improvement in the last decade. Nevertheless, most of these methods fall short in a few important details, as some cannot perform anomaly detection unsupervised, others need domain knowledge when setting certain parameters and some cannot adapt to changing behaviours. We chose ReRe as the basis of our research, as it fulfils all of our requirements.

## 3 Neural Networks

We start with a brief overview of artificial neural networks, emphasizing on one subcategory, the Long Short-Term Memory Recurrent Neural Networks. This information is necessary to gain a better understanding of algorithms that build neural network models, often used for data prediction.

### 3.1 Concept

A neural network<sup>1</sup> consists of layers of interconnected neurons that form a circuit, often represented as vertices and edges of a graph. A simplified version of a feedforward neural network can be seen in Figure 3-1. The term ‘feedforward’ refers to the fact that connections in such networks never form a cycle. Refer to the book “An Introduction to Neural Networks” [14] for a more detailed explanation.



**Figure 3-1: Simplified layout of a feedforward neural network.**

As it is apparent from Figure 3-1, one neuron can have many input and output connections. Each connection has a weight value assigned to it that represents how important or emphasized that connection is. Information propagates through these networks the following way: Data enters the neural network at the input layer that encodes it into the values of nodes. A so-called propagation function then calculates an input value for each neuron. This value is the weighted sum of the output of all the neurons from the previous layer that have a connection to the given neuron. The output layer then transforms information from a set of numbers in the layer into the desired format.

---

<sup>1</sup> This study refers to artificial neural networks as simply ‘neural networks’.

When training a neural network model, we need to have a training dataset (also referred to as labels or ground truth) that assigns correct output values to input values. Using such a dataset is called supervised learning, and the main task is to adjust connection weights (and other possible values) in order to minimize some form of an error in the output (e.g., how close a prediction is to the real value).

This adjustment is made through the algorithm of backpropagation using a predefined loss function that determines a way to evaluate each output compared to the training dataset. Backpropagation works by starting at the output layer and computing the gradient of the loss function with respect to each weight, then it moves iteratively layer by layer towards the input tuning weights and calculating gradients trying to minimize the loss function usually by the method of gradient descent [20]. Thus, training is done in three steps:

1. A forward pass on the network, where a prediction is made.
2. The prediction is compared to the ground truth using the loss function.
3. A backwards pass is performed, where the weights are adjusted to minimize the loss function.

In neural networks, the number of epochs is a key factor that determines how quickly a model can be trained. However, estimating the optimal number of epochs is not simple and yields a trade-off between underfitting and overfitting. On the one hand, if the epoch number is too low, the neural network does not have enough passes to learn patterns in the training data, thus leading to a poor performing model. This is called underfitting. On the other hand, if it is too high, the neural network will be tuned too exactly to the training data and will not function that well on other data points. This is called overfitting. This trade-off is one of the key challenges in machine learning.

## 3.2 Recurrent Neural Networks

The main difference between RNNs and simple feedforward neural networks is a so-called hidden state. The main purpose of it is to enable the network to ‘remember’ previous states by creating a feedback loop in the hidden layer(s). Figure 3-2 displays the recursive and unfolded representation of RNNs [19], where each timestep can be considered a new network model with input from the previous one.

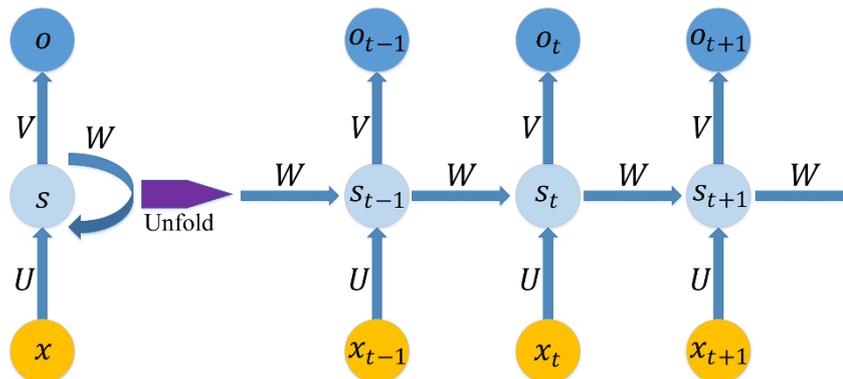


Figure 3-2: Structure of RNNs. [19]

A hidden state is a memory unit that is capable of storing information from the previous step. That is why RNNs are well-suited for applications on time-series data, where it is crucial to maintain some understanding of previous timesteps to produce a meaningful output. For example, when attempting speech recognition, it is important to keep in mind previous words in a sentence to understand the meaning of the whole sentence instead of just words one by one. Similarly, when using RNNs for time-series prediction, it is critical to ‘remember’ the influence of older training samples, not just the last one.

### 3.2.1 Vanishing Gradient

There is one issue with RNNs, however, that makes them almost unusable in their original form for real-world applications, namely the vanishing gradient problem. Because of the way neural networks are trained (refer back to Section 3.1), RNNs have what is called a ‘short-term memory’. This can be understood using the unfolded representation visible in Figure 3-2, where each timestep is thought of as a layer in the network. That is why, with RNNs, the training algorithm is called backpropagation through time. Gradient values shrink exponentially as it moves back through time, which

means that older timesteps are almost not at all taken into account when adjusting weights (the network ‘forgets’ older samples).

For more information on RNNs and LSTM networks refer to [15], titled “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. This work explains all important concepts and methods related to these neural networks.

### 3.3 Long Short-Term Memory

The concept of LSTM RNNs was first developed by S. Hochreiter and J. Schmidhuber in 1997 [13]. Their goal was to mitigate the effects of the vanishing gradient problem to enable more widespread applicability of RNNs. The key difference in the structure of a simple RNN and an LSTM is in the neurons (also referred to as units or cells). By introducing gates within the units, LSTMs are capable of controlling which pieces of information to ‘remember’, and for how long. The structural comparison of an RNN and an LSTM can be seen in Figure 3-3 [21]. The symbols within the gates denote the different activation functions used by the gates that determine how the input of the gate is processed ( $\sigma$ : sigmoid activation,  $\phi$ : tanh activation).

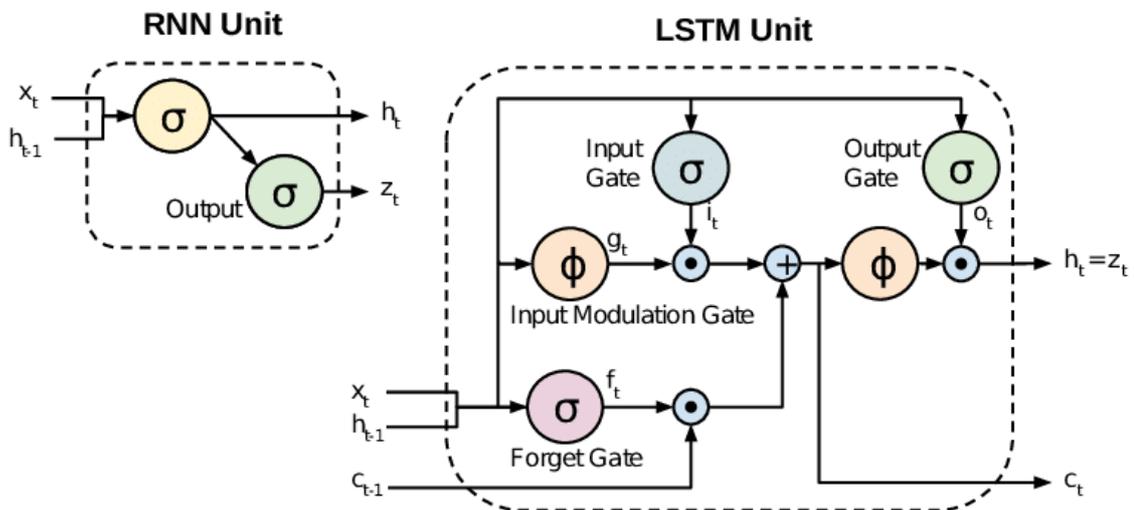


Figure 3-3: Comparison of the structures of RNNs and LSTMs. [21]

LSTMs managed to reduce the effects of the vanishing gradient to such a great extent, that current machine learning research is very often based on the original or an improved version of these neural networks. This is supported by the fact that 14k of the total 17k citations of the original paper [13] were received in the previous two years.

## 4 State-of-the-art Algorithms Based on LSTM

### 4.1 RePAD

RePAD [1] is a cutting-edge LSTM RNN-based algorithm designed for time-series anomaly detection. It was published in March 2020 by Ming-Chang Lee et al. The authors claim it is capable of detecting anomalies proactively in real-time, without any domain knowledge.

RePAD uses short-term historical data points to predict the upcoming value; then, it compares this prediction with the real value to determine if an anomaly is likely to happen in the near future. RePAD can adjust detection thresholds dynamically, making it well-suited to tolerate minor pattern changes as well. Its fast convergence (i.e., it can detect anomalies soon after being turned on) and unsupervised training (i.e., it does not require a labelled training dataset) sets it apart from previous approaches.

#### 4.1.1 LSTM Model

One key part of RePAD is the LSTM model used for data prediction. If an LSTM model has a complicated structure or the training data is large, training time increases significantly, which limits real-time use. That is why the LSTM model used in RePAD only has one hidden layer with 10 hidden units. Additionally, a fast-learning speed is guaranteed by a learning rate of 0.15. As mentioned in Section 3.1, the number of epochs is a key factor in determining the precision and speed of a neural network model. RePAD employs the algorithm of Early Stopping [16] to choose the number of epochs dynamically, to prevent overfitting and underfitting.

#### 4.1.2 Details of the Algorithm

RePAD is based on a so-called ‘look back, predict forward’ approach. This means that it takes the previous  $b$  data points ( $b$  is the look-back parameter) and uses them to predict the next  $f$  data points ( $f$  is the predict-forward parameter). In paper [1], the RePAD algorithm is stated with  $f = 1$  always. The algorithm can be broken down into four steps that follow each other.

Variables used in the explanation:

- $b$  or  $B$ : look-back parameter
- $t$ : current timestep, starts from  $t = 0$
- $v_x$ : data point at timestep  $x$
- $\widehat{v}_x$ : predicted data point for timestep  $x$
- $M, M'$ : LSTM models
- $AARE_x$ : Average Absolute Relative Error at timestep  $x$
- $thd_x$ : threshold value at timestep  $x$
- $\mu_{AARE_x}$ : the average of  $AARE_x$  values at timestep  $x$
- $\sigma_{AARE_x}$ : the standard deviation of  $AARE_x$  values at timestep  $x$

Equations for the algorithm:

$$AARE_t = \frac{1}{t - b + 1} \cdot \sum_{y=b}^t \frac{|v_y - \widehat{v}_y|}{v_y} \quad (1)$$

$$thd_t = \mu_{AARE_t} + 3 \cdot \sigma_{AARE_t} \quad (2)$$

$$\mu_{AARE_t} = \frac{1}{t - b + 1} \cdot \sum_{y=b}^t AARE_y \quad (3)$$

$$\sigma_{AARE_t} = \sqrt{\frac{\sum_{y=b}^t (AARE_y - \mu_{AARE_t})^2}{t - b + 1}} \quad (4)$$

### Step 1

if  $t < b - 1$

In this step, RePAD collects data points passively.

Collected values:  $v_0, v_1, \dots, v_{b-2}$ , so exactly  $b - 1$  pieces.

### Step 2

if  $t == b - 1$

When time reaches the value  $b - 1$ , RePAD trains an LSTM model  $M$  using the first  $b$  data points ( $v_0, \dots, v_t = v_{b-1}$ ). RePAD then uses  $M$  to predict  $\widehat{v}_{t+1} = \widehat{v}_{b+1}$ .

### Step 3

if  $b - 1 < t < 2b - 1$

For every timestep, RePAD calculates a so-called average absolute relative error using Eq. (1).

Then it trains the LSTM model  $M$  using the previous  $b$  data points:  $v_{t-b+1}, \dots, v_t$ .

$M$  then is used to predict  $\widehat{v_{t+1}}$ .

### Step 4

if  $t \geq 2b - 1$

Once RePAD gets to this step, it is already capable of detecting anomalies, for it has at least  $b$  number of  $AARE_x$  values. Since  $b$  is a small integer, RePAD has a short preparation period. The following then happens with each new timestep:

RePAD calculates  $AARE_t$  using Eq. (1).

It then calculates the  $thd_t$  threshold value using Eq. (2), which takes the average of all  $AARE_x$ 's, then adds their standard deviation three times as seen in Eq. (3) and (4).

These two values ( $AARE_t$  and  $thd_t$ ) are then compared.

If  $AARE_t \leq thd_t$ , that means the current data point  $v_t$  is similar to the previous ones, as  $M$  was able to predict it to an appropriate precision level, there is no anomaly. RePAD then uses  $M$  to predict  $\widehat{v_{t+1}}$ .

However, if  $AARE_t > thd_t$ , there may be two different reasons for that. Either the data pattern is slightly changing, or there is an anomaly. To make a decision, RePAD trains  $M'$  using the previous  $b$  data points, then recalculates the  $AARE_t$  error and  $thd_t$  threshold values. If  $AARE_t > thd_t$  still holds, RePAD signals an anomaly to the user, then uses  $M$  to predict  $\widehat{v_{t+1}}$ , it discards  $M'$ . But if  $AARE_t \leq thd_t$  using the new model, RePAD concludes that the data pattern is changing and replaces  $M$  with  $M'$ , so that it can predict new data points accurately.

Using this mechanism RePAD can adapt to small pattern changes in the data stream. Additionally, the LSTM model only needs to be retrained when a pattern change is detected, thus greatly reducing CPU load.

## 4.2 ReRe

ReRe [2] is an improvement of RePAD [1], developed by the same authors. The paper is currently available only as a preprint, its latest version uploaded in June 2020. According to experiments in [1], RePAD suffers from a great number of false positive anomaly detections (see Section 7 for our comments on false positives). ReRe attempts to target exactly this issue.

ReRe employs two LSTM models instead of one that provide two levels of detection sensitivity. These two models are deployed in two detectors (detector 1 and 2). An anomaly or pattern change is only detected and signalled by the ReRe algorithm if both detectors return the same detection for the given timestep.

Detector 1 works as RePAD (refer to Section 4.1.2 for more details). It acquires  $t$  and  $v_t$  values from the ReRe algorithm and produces signals ‘normal’, ‘pattern change’ and ‘anomaly’ as outputs. Internally, it stores and calculates  $AARE_t$  and  $thd_t$  using RePAD methods and equations. When it detects a pattern change, it retrains its own LSTM model,  $M_1$  that it uses for data value prediction.

Detector 2 has a few key differences from RePAD. It uses the same algorithm structure; it also acquires  $t$  and  $v_t$  from ReRe; the only change is in the input values for  $AARE_t$  and  $thd_t$  calculations. Namely, detector 2 uses its own  $M_2$  LSTM model to predict its own values.  $M_2$  is structurally identical to  $M_1$  but due to different retrain timesteps it will produce different predictions. And the reason for the different pattern change detection timesteps is the different calculation of the threshold  $thd_t$ . Detector 2 uses only  $AARE_x$  values for threshold calculation, where  $v_x$  is considered ‘normal’ (no ‘pattern change’ or ‘anomaly’) by itself. This one key difference means that it produces different results and is able to disable anomalies detected by detector 1 (RePAD). We give a more detailed evaluation of ReRe in Section 6.2.

### 4.3 Limitations of ReRe

As described in Section 4, ReRe can only detect anomalies, when both detectors have an  $AARE_t$  value higher than  $thd_t$ , and both detectors decide, there is no pattern change. That means, normally,  $AARE_y$  values are lower than  $thd_y$ .

In the design of RePAD and subsequently ReRe,  $AARE_t$  and  $thd_t$  values are computed using Eq. (1) and (2). Both of these values include a sum that starts at a fixed value, namely timestep  $y = b$  and ends at the current timestep  $y = t$ . All terms in these sums have an equal weight of 1, which means all terms have an equal priority in determining current  $AARE_t$  and  $thd_t$  values.

This results in an unfortunate consequence to the speed of anomaly detection. Figure 4-1 illustrates this issue<sup>2</sup>. When the algorithm has started just recently,  $AARE_t$  calculation has only a few (absolute relative error) terms to take the average of. This means when an anomaly occurs in the data, and error terms instantly get large (due to the big difference in the predicted and actual data points),  $AARE_y$  values also get large fairly quickly. This is the way how it should work for all timesteps.

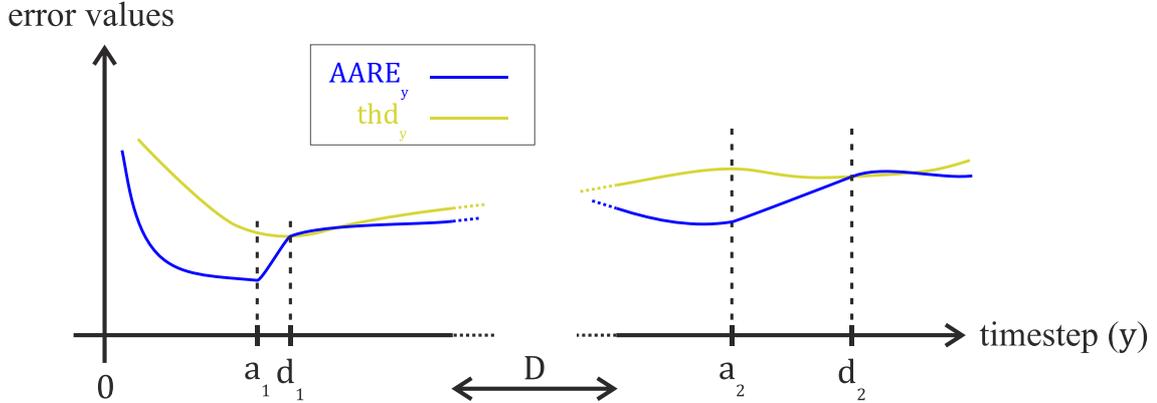


Figure 4-1: The slope of  $AARE$  depends on the current timestep.

In Figure 4-1, this situation is illustrated on the left side of the graph. At timestep  $a_1$ , there is an anomaly in the dataset.  $AARE_y$  values start to rise quickly as new absolute relative error terms are high, and there are only a few error terms to take the average of. At timestep  $d_1$ , the  $AARE_y$  curve crosses the  $thd_y$  curve, and an anomaly is detected.

<sup>2</sup> Note that the figure is exaggerated for better visibility. In reality,  $a_1$  and  $d_1$  are closer to each other.

However, when a considerable time passes ( $D$  is at least a few thousand timesteps), there are many more terms in  $AARE_t$  calculation to take the average of. This is the reason why when an anomaly occurs at timestep  $a_2$ , the  $AARE_y$  curve starts to rise only slowly, as averages increase slower with more terms (the influence of one new high term is proportionally less). As the beginning timestep for the average is always  $y = b$ , the slope gets less and less steep as  $y$  increases. That is why the anomaly at  $a_2$  is only detected at  $d_2$ .

In the extreme case, as the algorithm is planned to run real-time uninterrupted, anomalies start to get unnoticed after a certain timestep as the  $AARE_y$  curve never reaches the  $thd_y$  curve, it normalises before normal data points arrive again. This is a major drawback when ReRe is planned to run persistently.

There is another issue with continuous real-time use. In the current implementation of ReRe, all original data points ( $v_y$ ), all predicted data points ( $\widehat{v}_y$ ), all AARE error values ( $AARE_y$ ), all threshold values ( $thd_y$ ), moreover all detected anomalies and pattern changes have to be stored from the moment they are generated for as long as ReRe is running. This means that in order for ReRe to run for  $T$  total timesteps,  $6T$  data points have to be stored locally. Consequently, there is no upper bound on the memory requirements of the algorithm, and if it runs out of space, ReRe is going to stop with an error. Admittedly, this would still involve a presumably long operation time, as modern computers have terabytes of storage, and a timestep usually takes up around a few hundred bytes, but this approach is far from being optimal.

## 5 Alter-Re<sup>2</sup>

We introduce two extensions, such as ageing and window-mode, to the ReRe algorithm (refer to Section 4.2) in this section to address important limitations in the design of ReRe. Then, we present our implementation for tuning both parameters of the proposed upgrades (i.e., age power and window-mode).

### 5.1 Ageing

To address the issue described in Section 4.3, we decided to implement the ageing of the terms in  $AARE_y$  calculation. This involves devising a method to place greater emphasis on a few previous data points instead of averaging them with the same weight. So as time elapses, the weight of a given data point decreases.

This means that an extra ageing coefficient  $C_y$  is introduced to Eq. (1), which results in the following modified formula:

$$AARE_{t, \text{aging}} = \frac{1}{t - b + 1} \cdot \sum_{y=b}^t C_y \cdot \frac{|v_y - \widehat{v}_y|}{v_y} \quad (5)$$

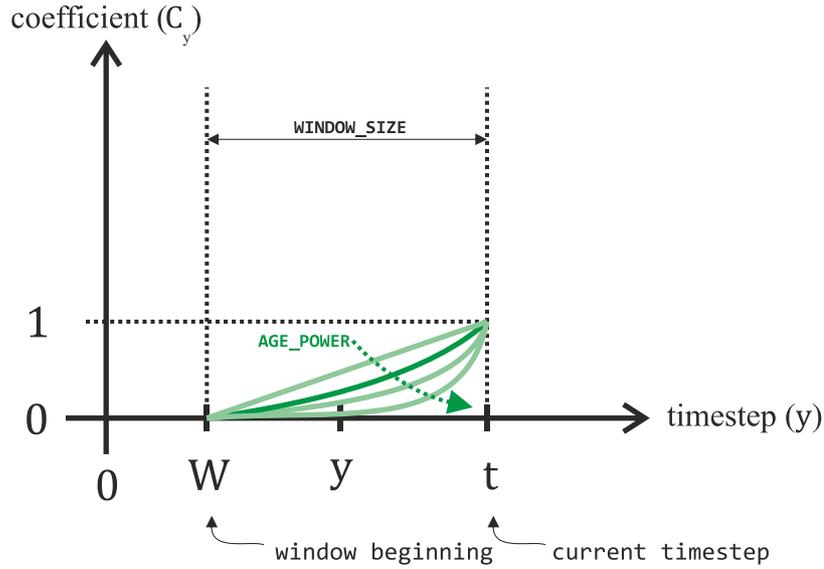
This coefficient  $C_y$  is calculated using the following equation:

$$C_y = \left( \frac{y - W}{t - W} \right)^{AP} \quad (6)$$

Variables used (beyond the ones introduced in Section 4.1.2):

- $y$ : timestep running variable in the sum
- $W$ : beginning timestep of the Window
- $AP$ : Age Power
- $C_y$ : ageing coefficient

The visual aid for understanding the concept of ageing can be seen in Figure 5-1. The figure incorporates the design of a sliding window as well, introduced in Section 5.2. To see the improvements of ageing only, settings  $W = 0$  and  $WINDOW\_SIZE = t$  should be used. As it is apparent from Figure 5-1 and Eq. (6), the age power variable  $AP$  determines the aggressiveness of ageing, i.e., how strongly the algorithm should consider the previous few data points. If  $AP = 1$ , there is a linear ageing. Negative numbers are not recommended, as they result in inverse ageing. Thus,  $AP$  becomes an additional hyperparameter of the algorithm.



**Figure 5-1: The operational principle of ageing.**

Equation (6) always produces a number between 0 and 1 for  $C_y$  if  $y \in [W, t]$ . This way, the last few data points will remain approximately the same, while the ones closer to the start will be scaled down.

The issue of slow or no reaction to anomalies after a considerable time, discussed in Section 4.3, can be addressed with the ageing of the terms in the sum, as new high error terms influence  $AARE_y$  values much more than older smaller ones. We assess further this improvement in Section 6.3.

## 5.2 Window-mode

The other limitation of ReRe discussed at the end of Section 4.3 is the need to store all data from the point in time they were generated. We can mitigate this issue by storing only the previous value of the average error terms, and the number of data points it was calculated from. This recursive method formalized below:

$$AARE_t = \frac{1}{t - b + 1} \cdot \left( AARE_{t-1} \cdot (t - b) + \frac{|v_t - \hat{v}_t|}{v_t} \right) \quad (7)$$

However, in order to calculate the threshold value  $thd_t$ , we need to know the standard deviation of the  $AARE_y$  values. Unfortunately, this  $\sigma_{AARE_t}$  value cannot be expressed only using values from the previous timestep  $t - 1$  and the number of timesteps due to the changing  $\mu_{AARE_y}$  values in every timestep. Therefore every  $AARE_y$  value has to be stored from the start.

To address the need for this ‘unlimited’ storage, and to entirely eliminate the necessity to take into account very old data points, we implemented a sliding window in the algorithm (see Figure 5-1 for a visual representation). The sliding window has one parameter, the WINDOW\_SIZE or  $WS$ . The window beginning timestep  $W$  is calculated in Eq. (8). Data points before the beginning of the window are discarded, and the equations for ReRe are modified as follows in Eq. (9) – (12). These equations also include the implementation of ageing. If ageing is disabled, all values of  $C_y$  are set to 1. If window-mode is disabled, the window size parameter  $WS$  is set to the current timestep  $t$ . This way, Eq. (8) always chooses the timestep  $b$  as the beginning of the window because that is when the first value of  $AARE_y$  is produced.

$$\begin{cases} W = t - WS + 1 & \text{if } t - WS + 1 > b \\ W = b & \text{if } t - WS + 1 \leq b \end{cases} \quad (8)$$

$$AARE_t = \frac{1}{t - b + 1} \cdot \sum_{y=W}^t C_y \cdot \frac{|v_y - \widehat{v}_y|}{v_y} \quad (9)$$

$$thd_t = \mu_{AARE_t} + 3 \cdot \sigma_{AARE_t} \quad (10)$$

$$\mu_{AARE_t} = \frac{1}{t - b + 1} \cdot \sum_{y=W}^t AARE_y \quad (11)$$

$$\sigma_{AARE_t} = \sqrt{\frac{\sum_{y=W}^t (AARE_y - \mu_{AARE_t})^2}{t - b + 1}} \quad (12)$$

As shown in Figure 5-1, the introduction of ageing (see Section 5.1) has been adapted for the sliding window. Eq. (6) produces values of 0 at the beginning of the window (timestep  $W$ ) and produces values of 1 at the current timestep  $t$ .

The implemented sliding window has an extra benefit.  $thd_y$  values are calculated using  $AARE_y$  values only from within the window. Therefore, the detection threshold adjusts faster and more precisely, and high  $AARE_y$  values from a few thousand timesteps before do not distort the performance until the very end of operation.

## 5.3 Automated hyperparameter tuning

In this section, we first discuss the effects of the WINDOW\_SIZE and AGE\_POWER parameters on detection. Then, we introduce our proposed algorithm for automated adjustment of these two parameters. We show that this automation can help optimizing the values of the window size and age power, which normally would need to be set manually.

### 5.3.1 Devising a metric for the size of data points used

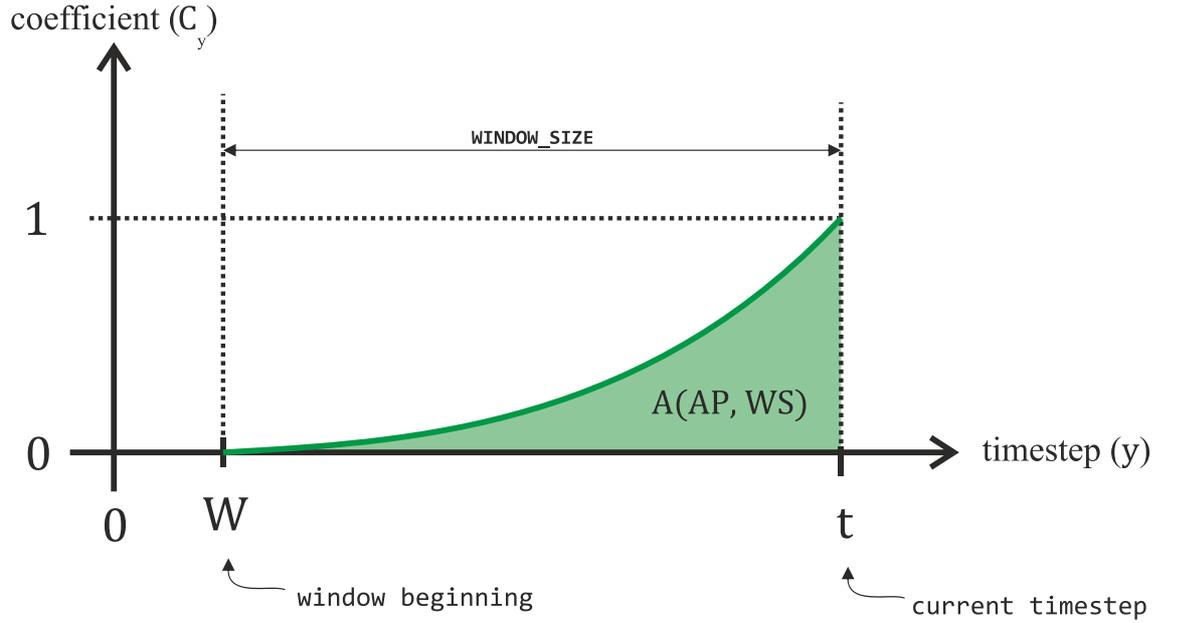
Both the WINDOW\_SIZE and AGE\_POWER parameters have an indirect relation to the accuracy of anomaly detection. Precisely, they impact the volume of data points and their weights that serve as inputs when Alter-Re<sup>2</sup> calculates the  $AARE_y$  and  $thd_y$  values. This relationship is qualitatively showed in Figure 5-1.

From the figure, we find that an increase of  $AP$  results in fewer (and newer) data points being taken into consideration. Decreasing it, on the other hand, has the opposite effect, enabling older data points within the window as well to have an impact on the calculated  $AARE_y$  value.

The opposite is true for  $WS$ . As we increase the size of the window, the number of data points used for calculation also increases, and reducing the size also reduces the number of values taken into consideration.

The above discussed qualitative analysis, however, does not allow for precise tuning of these parameters; their effect has to be evaluated numerically. To achieve this, we propose a metric that relates to the number of data points and the weights introduced by ageing. Namely, the area under the  $C_y$  curve,  $A(AP, WS)$ , as shown in Figure 5-2.

This area changes exactly as expected based on the discussion above. Increasing  $AP$  reduces it while decreasing  $AP$  increases its value. On the other hand, expanding the window size results in a larger area, whereas a smaller window yields a smaller area. Therefore, we consider it a useful metric for the size of data taken into consideration. However, its calculation has to utilize only a small portion of the available resources to justify its usefulness. An area under a curve from a certain timestep to another can be calculated using definite integrals. This derivation is present in Equation (13).



**Figure 5-2: The area under the  $C_y$  curve.**

$$\begin{aligned}
 A(AP, WS) &= \int_W^t C_y(y) \cdot dy = \int_W^t \left( \frac{y - W}{t - W} \right)^{AP} \cdot dy = \dots = \\
 &= \left[ \frac{(y - W)^{AP+1}}{(t - W)^{AP} \cdot (AP + 1)} \right]_W^t = \frac{t - W}{AP + 1}
 \end{aligned} \tag{13}$$

Substituting  $W = t - WS + 1$  after enough timesteps, as shown by Equation (8) results in Equation (14).

$$A(AP, WS) = \frac{t - (t - WS + 1)}{AP + 1} = \frac{WS - 1}{AP + 1} \tag{14}$$

This relationship (derived above in Equation (14)) between the area under the  $C_y$  curve and the  $WS$  and  $AP$  parameters meets all the criteria defined above. It is lightweight in terms of computation complexity, contains only the two important parameters, and behaves exactly like the qualitative analysis predicted.

If the goal is to preserve this area ( $A(AP, WS)$ ) while changing both  $WS$  and  $AP$ , we can use the following formula (displayed in Equation (15)).

$$\frac{WS_y - 1}{AP_y + 1} := \frac{WS_{y+1} - 1}{AP_{y+1} + 1} \tag{15}$$

We can rearrange Equation (15) to express  $AP_{y+1}$  using the other variables. This can be useful if our aim is to adjust  $AP$  exactly so that the new area using  $WS_{y+1}$  is the same as the old one. This rule for  $AP$  setting is displayed in Equation (16).

$$AP_{y+1} = \frac{(WS_{y+1} - 1) \cdot (AP_y - 1)}{WS_y - 1} \quad (16)$$

### 5.3.2 Automated tuning algorithm steps

Utilizing the metric described in Section 5.3.1 and the findings on signs of extreme parameter values described in Section 5.3.3, we devise an algorithm that is capable of automatically adjusting the  $WS$  and  $AP$  parameters. The algorithm consists of four steps.

#### Step 1 – default values

if t == 0

In this step, Alter-Re<sup>2</sup> sets the default values for the parameters. The window size is determined based on the  $B$  parameter (see Section 4.1), as shown in Equation (17). Since the  $B$  parameter determines how many data points the LSTM model uses for training and prediction, it is well suited to be the basis for the window size, also a parameter that determines the number of data points to be used, but for another purpose.

$$WS_{def} = 5 \cdot B \quad (17)$$

We consider this a small window based on our experiments, but since it can be both increased and decreased during various steps of the algorithm, this bears little significance. The default value for the AGE\_POWER parameter results in linear ageing at the beginning, as shown in Equation (18). This can also be considered a smaller value.

$$AP_{def} = 1 \quad (18)$$

These default values determine a fixed area metric, which can be calculated as shown in Equation (19).

$$A_{def} = \frac{WS_{def} - 1}{AP_{def} + 1} = \frac{5B - 1}{2} = 2,5B - 0,5 \approx 2,5B \quad (19)$$

## Step 2 – initial increase

```
if t == 2B - 1 + 2B*i
```

The second step is the initial increase of both  $WS$  and  $AP$ . At this stage, the  $WINDOW\_SIZE$  and the  $AGE\_POWER$  parameters are most likely too small for the dataset given the default values (see Step 1 – default values above). Additionally, these original values determine an area metric calculated in Equation (19) that corresponds to only a few data points taken into consideration. As the optimal area metric depends on the actual dataset, the most straightforward approach seems to be starting from a small one initially and increasing it until satisfactory. Therefore, Alter-Re<sup>2</sup> only checks if the window size is too small, using the so-called  $WS\_MIN$  criterion (see Section 5.3.3). If it is true (i.e., the window needs to be larger), both the  $WS$  and  $AP$  parameters are increased as shown in Equation (20) and Equation (21). ( $[x]$  means  $x$  rounded to the nearest integer).

$$WS_{y+1} = [2 \cdot WS_y] \quad (20)$$

$$AP_{y+1} = 1,5 \cdot AP_y \quad (21)$$

These formulas have three distinct effects. Trivially, both  $WS$  and  $AP$  are increased, getting them closer to their optimal values. The third effect is on the area metric. The coefficients in the formulas are chosen so that the area under the  $C_y(y)$  curve also increases, allowing more data points at once to have an impact on detection.

The number  $i$  stores how many times this step was performed, starting from 0. As long as  $WS\_MIN$  is true,  $i$  is incremented by one, and this step is performed again. When  $WS\_MIN$  returns false, a Boolean storing whether this initial step is over is set to true, and  $i$  is fixed. After this, Alter-Re<sup>2</sup> never returns to this step, alternating instead between steps 3 and 4 until the algorithm is stopped.

## Step 3 – adjusting the window

```
if t == 2B - 1 + 2B*i + B^2*j
```

After the initial increase of the parameters, this step is performed at every  $B^2$  timesteps, as seen in the code above. Alter-Re<sup>2</sup> performs three checks at the beginning. First, it determines if a parameter adjustment has occurred in the last  $B$  timesteps. If it has, execution of the step stops. If not, the algorithm performs two additional checks,

namely  $WS\_MIN$  and  $WS\_MAX$  (see Section 5.3.3) to determine whether the  $WINDOW\_SIZE$  needs to be adjusted.

If the two criteria return different logical values (i.e., true and false or false and true), Alter-Re<sup>2</sup> proceeds to perform the adjustment. If the  $WS\_MIN$  condition is signalled,  $WS$  and  $AP$  are increased. The  $WINDOW\_SIZE$  is determined by taking the average of the last appropriate higher  $WS$  and the current one. If there is none, it is doubled. Similarly, in the other case (when  $WS\_MAX$  is signalled),  $WS$  is reduced by taking the average of the last appropriate lower  $WS$  and the current one. Also like in the previous case, if there is no lower value applicable, it is halved. The  $AGE\_POWER$  in both cases is modified using Equation (16), to maintain the area metric. This is done so that only the window is adjusted, and not the number of data points taken into consideration (see Step 4 – adjusting ageing).

If the two criteria return the same logical value (i.e., true, true or false, false), that means that the size of the window is set and no additional changes are performed in this step. The running index  $j$  stores the number of times this step was initiated, starting from 0. Every time the initial checks are performed, besides  $j$ , the  $k$  parameter (see Step 4 – adjusting ageing) is also incremented by one.

#### **Step 4 – adjusting ageing**

```
if t == 2B - 1 + 2B*i + B^2 + floor(B^2/4)*k
```

When Alter-Re<sup>2</sup> reaches this step, it begins similarly as Step 3 – adjusting the window starts; it checks three conditions. The first one is the same: whether there has been an adjustment in the last  $B$  timesteps. If yes, execution of this step stops. If not, the two additional criteria, namely  $AP\_MIN$  and  $AP\_MAX$  are evaluated. Also, similarly, if they both return the same Boolean value, the  $AP$  parameter is set, no additional adjustments are performed. Otherwise, Alter-re<sup>2</sup> adjusts the  $AGE\_POWER$  (and only that variable) as the criteria suggest. If  $AP\_MIN$  is true,  $AP$  is increased by 25%, and if  $AP\_MAX$  is true, it is decreased by 25%.

This step follows Step 3 – adjusting the window initially, and then is performed four times more frequently. This is done to enable fine tuning of detection by only adjusting  $AP$ . The variable  $k$  is an integer that shows how many times this step was started. It starts with 0, and is incremented by one every time the initial checks are performed.

### 5.3.3 Detection of extreme parameter values

To facilitate the detections listed below, a new database is established during operation. It only stores the following three pieces of information: timestep, anomaly detected (True or False), pattern change detected (True or False). The length of this database is not determined by the WINDOW\_SIZE parameter, thus allowing independent operation. Its size is computed instead using Equation (22).

$$L_{database} = B^2 \quad (22)$$

We also refer to this parameter ( $L_{database}$ ) by the size of the signal window. The following subsections list the criteria for determining whether a certain parameter ( $WS$  or  $AP$ ) is at an extreme value (too small or large).

#### Too large window

$$WS\_MAX = \text{long\_anomalies}$$

Detecting a too large  $WS$  parameter can be achieved through measuring the number of continuous timesteps an anomaly is signalled for. Due to the reasons discussed in Section 4.3, when there are too many data points taken into consideration, the slope of the  $AARE_y$  curve gets small. That means, that Alter- $Re^2$  adjusts to new data patterns slowly in this case, resulting in anomalies signalled for a significantly longer time than usually (based on our experience, the assessed anomalies never lasted more than just a few timesteps). Thus, the detector's only job is to count the number of timesteps an anomaly is signalled for. If it gets larger than  $2 \cdot B$ , a long anomaly is reported.

#### Too small window

$$WS\_MIN = \text{anomaly\_flapping or frequent\_signals}$$

If a window is too small, Alter- $Re^2$  does not have enough information to reliably determine long-term patterns and thresholds, and detection becomes unstable. This results in two different noticeable phenomena. The first one is anomaly flapping, i.e., anomaly signalling turning on and off rapidly. This condition is true when an anomaly is signalled for  $n$  timesteps, then no detection is made for no longer than  $1,5 \cdot n$  timesteps, then another anomaly is signalled. This behaviour is the result of the  $AARE_y$  and  $thd_y$  curves crossing frequently due to instability introduced by the small window.

The other criterion checked is frequent signalling of anomalies or pattern changes. The reason for this is the same as discussed above. Too few data points in a window result in unstable operation, that manifests in abnormally large number of pattern changes and anomaly signals. This, however, can only be determined based on the patterns in the original dataset, as different time series will require different anomaly and pattern change detections. To achieve this dataset-dependent detection of too frequent signals, a threshold percentage is calculated based on the changes in the dataset using Equation (23), where  $ST$  is referred to as the signal frequency threshold.

$$ST = \frac{10}{t - W} \cdot \sum_{y=W}^{t-1} (v_{y+1} - v_y)^2 \quad (23)$$

This metric is useful as a threshold for the frequency of signals, as when there are many high-amplitude changes, more anomaly and pattern change signals are expected, and  $ST$  also evaluates to a higher value. After calculating  $ST$ , Alter- $Re^2$  counts the actual number of signals in the database discussed in Section 5.3.3. Signals (i.e., anomaly or pattern change detections) count as one even if they are present for multiple timesteps. This is achieved by counting only the starts (false to true transitions) of the signals. When the enumeration is over, the algorithm divides the number of signals by the length of the database ( $L_{database} = B^2$ ) to get the current signal ratio. It then compares the signal threshold ( $ST$ ) to this current signal ratio. If the current ratio is higher than  $ST$ , the number of signals is considered abnormal, and the frequent signals condition is set to true, otherwise it is set to false.

### **Too high age power**

`AP_MAX = WS_MIN = anomaly_flapping or frequent_signals`

As discussed in Section 5.3.1, when  $AP$  and  $WS$  are changed in opposite directions have similar effects. So much so, that their extreme values can be detected using the same set of criteria. The too large  $AGE\_POWER$  (if nothing else changed) results in fewer data points taken into consideration, thus producing anomaly flapping and too frequent signals. Therefore,  $AP\_MAX$  returns the same logical value as  $WS\_MIN$  (see the section entitled ‘Too small window’).

### **Too small age power**

$AP\_MIN = WS\_MAX$  or  $AP < 1 = \text{long\_anomalies}$  or  $AP < 1$

Similarly, when the AGE\_POWER is too small, many older data points will have an effect on the algorithm, reducing the slope of the  $AARE_y$  curve, and thus the speed of anomaly detection. This can be observed by searching for too long anomalies.

An additional condition is introduced to AP\_MIN compared to WS\_MAX (see the section entitled 'Too large window'),  $AP < 1$ . This is done in order to prevent AP values falling to extreme low values even if the WINDOW\_SIZE needs to be reduced (see Step 3 – adjusting the window).

## 6 Experiments

In this section, we present our experiments comparing different settings of ReRe and Alter-Re<sup>2</sup>, and present the experimental parameters and setup used. This is then followed by the evaluation of the automated *WS* and *AP* parameter tuning algorithm integrated in Alter-Re<sup>2</sup>.

### 6.1 Preliminaries

The source of the datasets presented in this study is the Numenta Anomaly Benchmark (NAB) [17], using their publicly accessible GitHub repository [18]. NAB has a wide variety of different types of datasets related to computer networks and data traversing them or originating from them. It has flags for real anomalies to help evaluate real-time anomaly detection algorithms.

To set ReRe parameters, we started with values recommended and used in the original paper [2]. This means having one hidden LSTM layer and setting the number of neurons to 10 within it. This proved to be insufficient, as the neural network model was making unreasonable predictions due to not being able to learn data patterns with enough complexity at the beginning. Unfortunately, we could not compare our implementation to the authors' one, as there is to this day no publicly available code of either RePAD or ReRe. Therefore, it is entirely possible that a slightly different way of deploying the LSTM model and implementing training and prediction functions might result in different operation. Nonetheless, we concluded that a setting of 30 neurons in the one hidden layer with 30 epochs produced the best results. On the one hand, increasing these numbers further dramatically increases training time, on the other hand, decreasing them degrades prediction performance.

We conducted similar experiments to determine the value of  $b$ , the look-back parameter. The authors of ReRe used a very low  $b = 3$ , that for us did not produce satisfying results. After experimenting with the effect of all parameters on the overall performance, we concluded a setting of  $b = 30$ . Surprisingly, a higher or lower number sometimes introduced a constant offset between the original and predicted data points.

As mentioned in Section 4.1.2, the predict-forward parameter is set to  $f = 1$  constantly (ReRe predicts only the very next datapoint  $\widehat{v}_{t+1}$ ).

With regard to the WINDOW\_SIZE ( $WS$ ) parameter, we found that a too small value results in an unstable operation, as there are not enough data points to learn long-term dependencies. The upper bound on  $WS$  is the storage space designated for the algorithm. Additionally, CPU and time constraints might also have to be considered, as computational difficulty and time increases with the larger number of available data points. For these reasons, we concluded based on experiments that  $WS$  has to be set to at least 500 timesteps. In the following experiments, we use the setting of  $WS = 1000$ .

Finally, we found that the AGE\_POWER ( $AP$ ) parameter should be set to approximately 2, resulting in a quadratic ageing equation. If  $AP$  is significantly lower, old data points have a strong influence on the current  $AARE_t$  value, which is undesirable. On the contrary, if  $AP$  is much higher than 2, only the very few last data points have any effect on the algorithm’s performance, which results in unstable operation. In the following experiments, we use the setting of  $AP = 2$ .

As we were working with offline datasets, we did not have to implement real-time normalization of data points (more discussion on this aspect can be found in Section 8). Since LSTM models require the training and prediction data to fall between 0 and 1 for appropriate operation, we simply divided the whole dataset by the value of the largest data point.

All figures presented in the following three experiment sections (Section 6.2, Section 6.3 and Section 6.4) have the same layout. They consist of three graphs that show the details of ReRe in operation. The top graph shows the original data points ( $v_y$ ) in green, and the LSTM-predicted data points ( $\widehat{v}_y$ ) in red. Recall, that the original and predicted values have been scaled down to the  $[0, 1]$  interval for the LSTM model to work. In the middle graph, we draw the curves of the absolute average relative error ( $AARE_y$ ) in blue and the threshold ( $thd_y$ ) in yellow. The bottom figure displays detections made by ReRe. Anomaly detections are drawn in purple, while pattern changes have a turquoise colour. As a reminder, when the  $AARE_y$  value gets higher than the  $thd_y$  value, ReRe determines whether there is a pattern change or an anomaly (always one of these, but never both). For more information on ReRe operation, refer to Section 4.

The settings of WINDOW\_SIZE ( $WS$ ) and AGE\_POWER ( $AP$ ) are shown in the captions underneath the figures, alongside with the dataset name and flagged anomalies (anomalies confirmed by the collector of the dataset).

## 6.2 Reference Measurement

The aim of this experiment is to prove that the original ReRe algorithm is capable of detecting anomalies. The dataset used here shows CPU utilization percentages from an Amazon Web Services (AWS) server. The results are depicted in Figure 6-1.

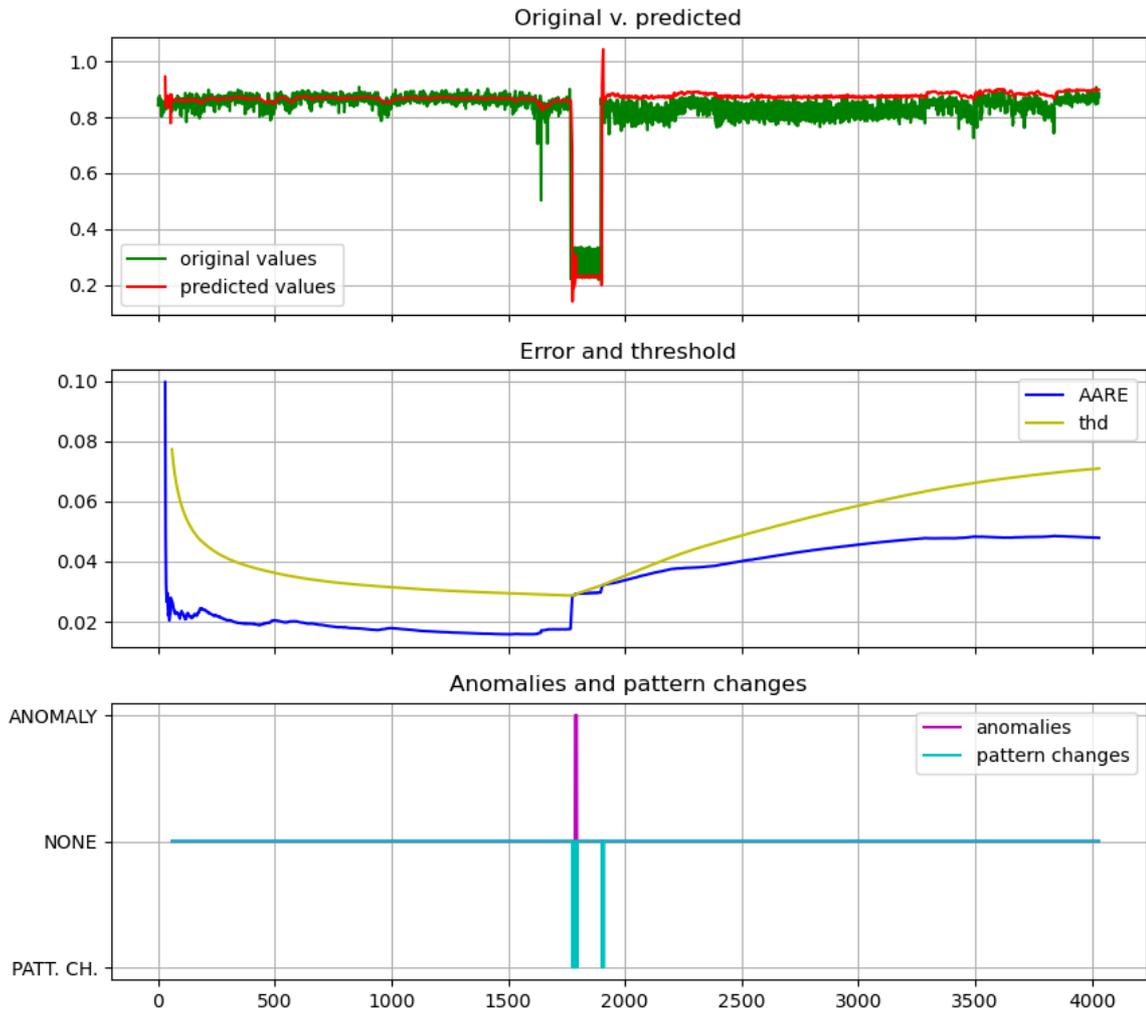


Figure 6-1: ReRe output for `ec2_cpu_utilization_825cc2.csv` (no ageing, no window).

Flagged anomalies at timesteps 1627, 1769.

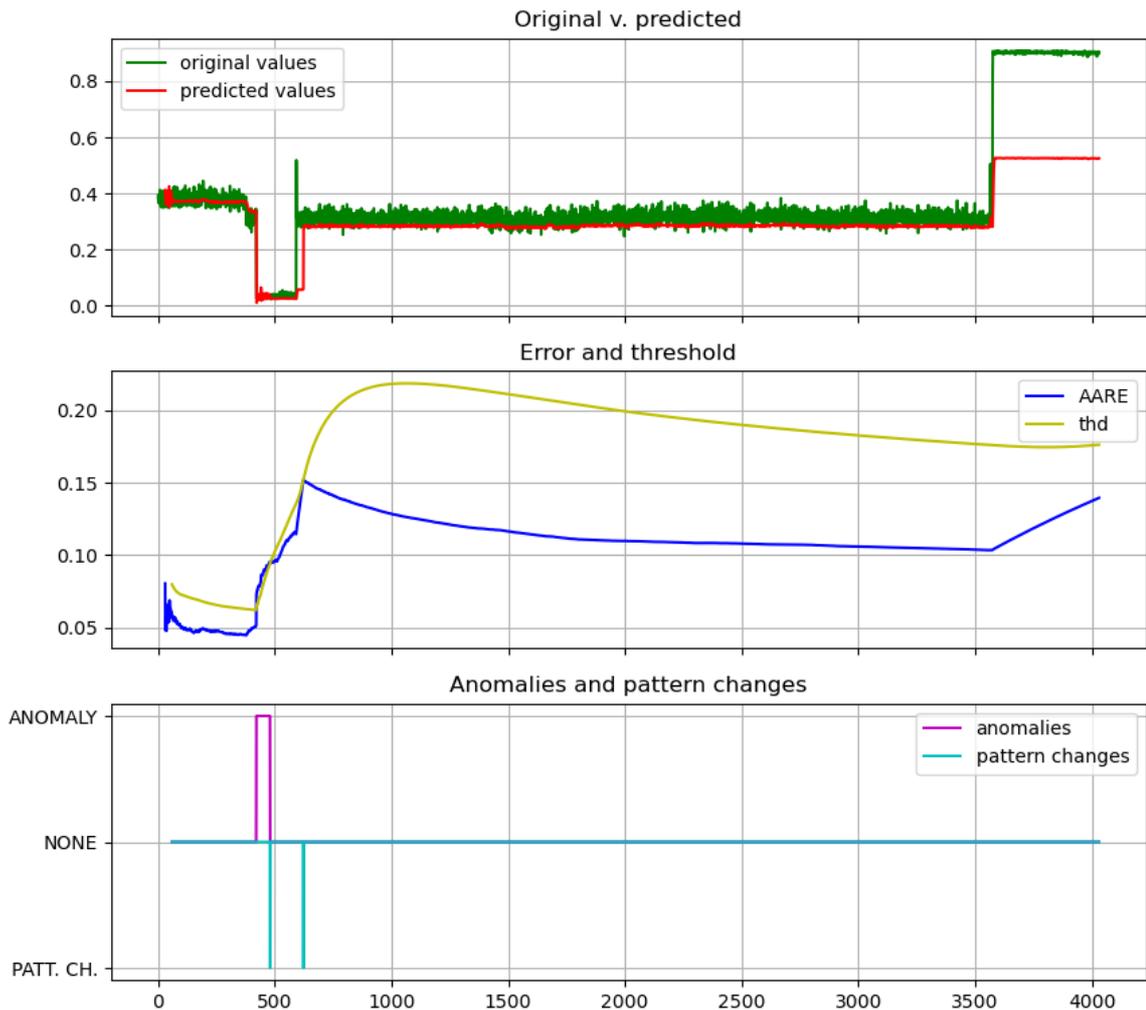
The second anomaly at timestep 1769 is detected by ReRe as the bottom graph displays. The middle graph shows that ReRe needs a few timesteps for the *AARE* curve to rise enough to cross the *thd* curve and begin the detection process. After the original data rises again at around timestep 1900, ReRe detects a second pattern change and retrains the LSTM model with the new data to adjust to the new pattern.

However, the first anomaly at timestep 1627 is not detected, since it is only a few timesteps long, and the *AARE* curve does not have the time to rise enough to cross *thd* values. This is due to the arrival of new normal data points that mitigate the increase of

AARE values. Nonetheless, ReRe is able to perform anomaly detection when the anomaly persists for a few timesteps, thus confirming the claims of its developers.

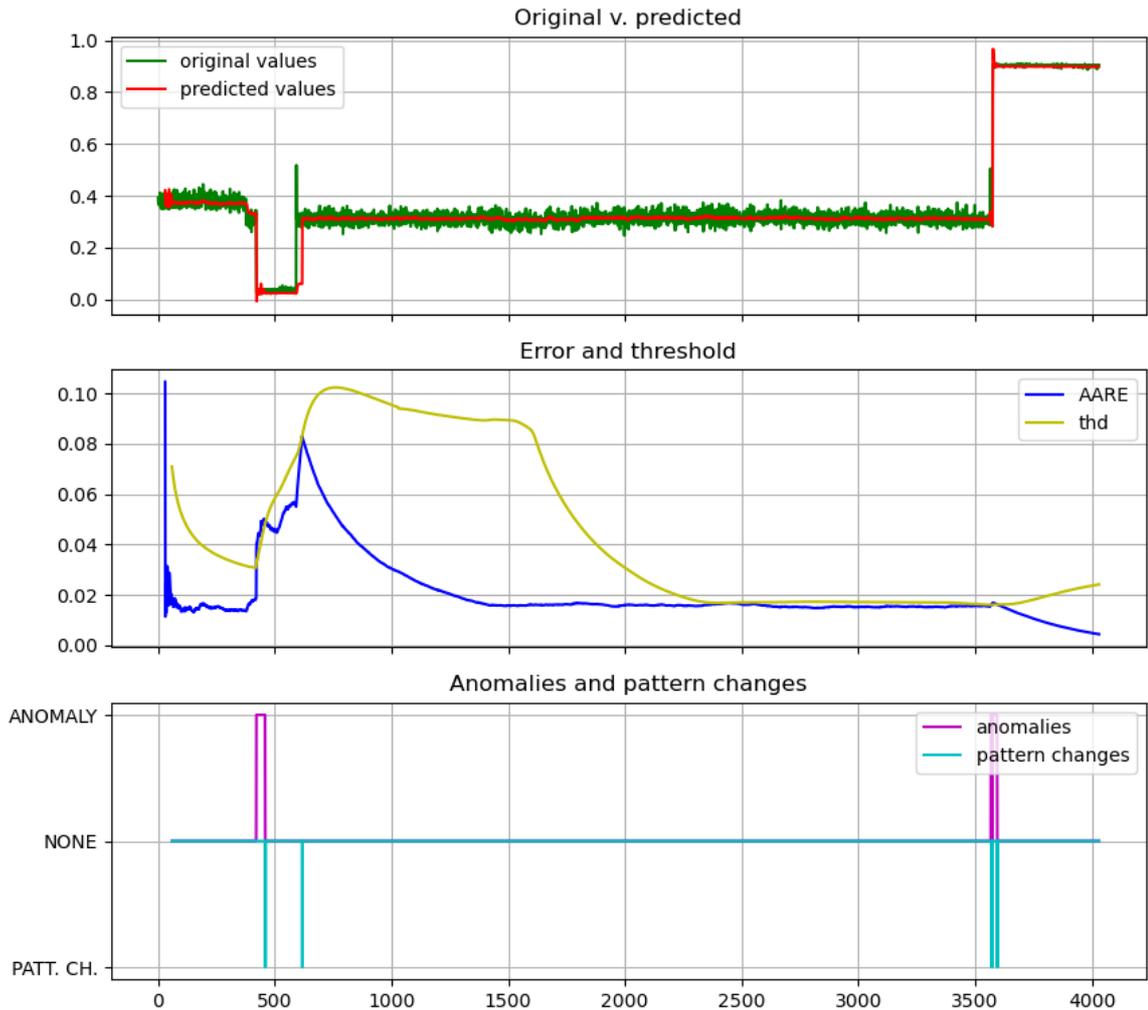
### 6.3 ReRe vs. Alter-Re<sup>2</sup>

In this experiment, our goal is to show the benefits of Alter-Re<sup>2</sup> compared to the original design of ReRe in two experiments. Both datasets used here show CPU utilization percentages collected from AWS servers using the CloudWatch monitoring tool.



**Figure 6-2: ReRe output for ec2\_cpu\_utilization\_ac20cd.csv (no ageing, no window).  
Flagged anomalies at timesteps 421, 3576.**

Figure 6-2 and Figure 6-3 depict the different detection results of ReRe and Alter-Re<sup>2</sup>, respectively. The first flagged anomaly is detected by both algorithms for similar reasons, as discussed in the previous experiment (see Section 6.2).



**Figure 6-3: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_ac20cd.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 421, 3576.**

However, the middle graph in Figure 6-2 indicates the shortcomings of ReRe. After the detection of the first anomaly, *thd* values rise significantly as they are calculated using the average and standard deviation of *AARE* values. Additionally, as the second anomaly comes at timestep 3576, a lot of data points have been collected, and the issue (the slope of the *AARE* curve is much lower) arises. These two issues combine in such a way that the anomaly goes entirely undetected.

Alter-Re<sup>2</sup>, on the other hand, solves both issues and detects the second anomaly as well, shown in Figure 6-3. The *thd* curve resets between timesteps 1500 and 2500 due to the use of sliding window, allowing much more precise detection. The slope of the *AARE* curve increases dramatically (although admittedly it is hard to spot in the figure), thanks to the implemented ageing.

The superiority of Alter- $Re^2$  over ReRe can be examined further by contrasting Figure 6-4 and Figure 6-5. Here, both anomalies last only for a few timesteps (the two spikes with the highest amplitude).

Figure 6-4 shows the operation of the original ReRe algorithm. As anomalies are only detectable for a few timesteps, AARE values barely reflect the anomalous behaviour. For this reason, ReRe does not detect either of them.

There is another issue visible in Figure 6-4. Namely, there is an almost constant offset between the averages of the predicted values and the original ones. This offset is present in most of the other experiments' figures as well, at least in part, though perhaps it is most visible here. This offset is due to the slightly abnormal training data the LSTM model learned from the data patterns. We discuss how to mitigate this issue in Section 8.

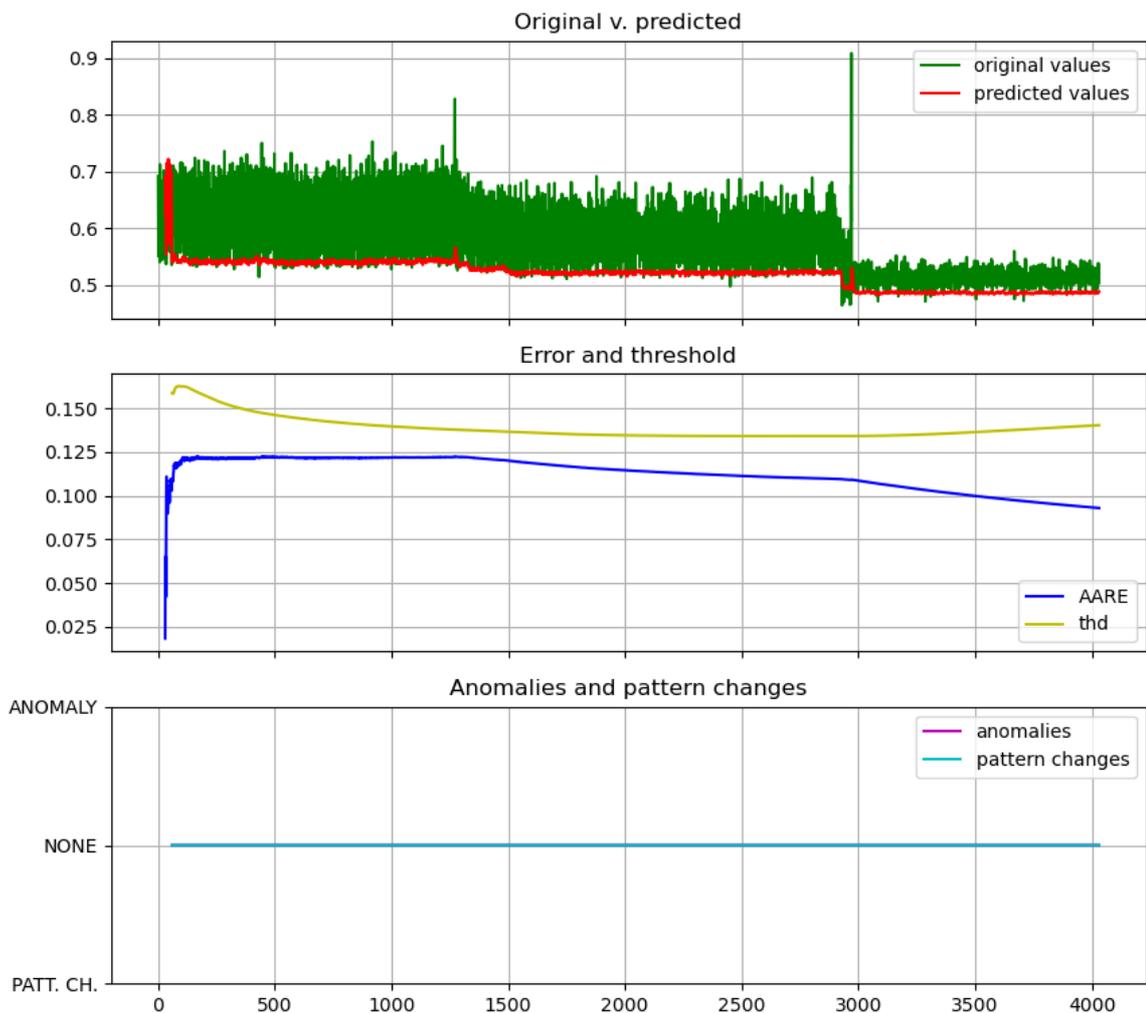
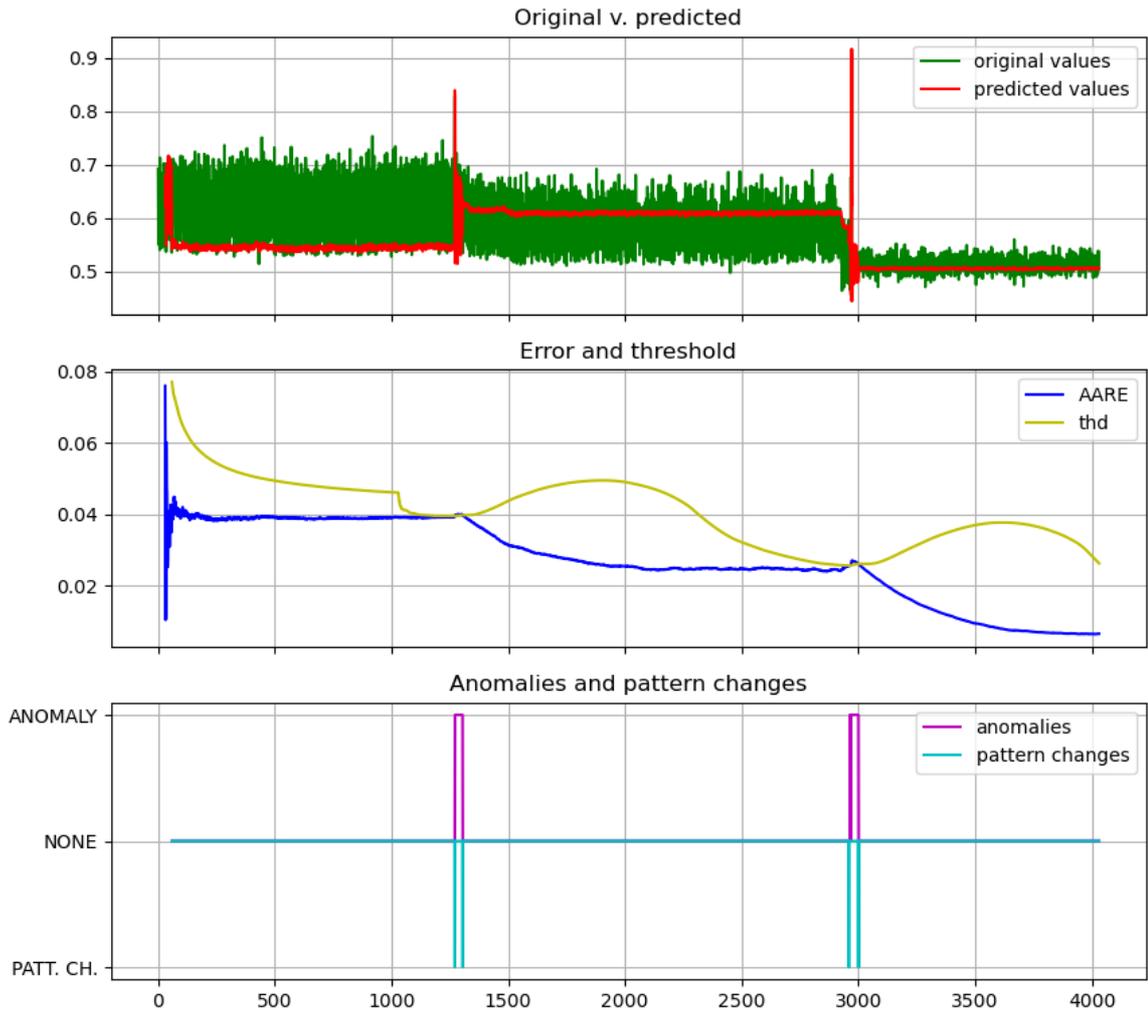


Figure 6-4: ReRe output for ec2\_cpu\_utilization\_5f5533.csv (no ageing, no window).

Flagged anomalies at timesteps 1272, 2931.



**Figure 6-5: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_5f5533.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 1272, 2931.**

In contrast with Figure 6-4, Figure 6-5 depicts that Alter-Re<sup>2</sup> successfully detected both anomalies. This is partly because of the adjustment of the *thd* curve to the *AARE* at around timestep 1040. The *thd* curve does this as the sliding window (of size 1000) leaves behind the usually high error timesteps at the beginning that are a result of the training phase of ReRe. The original algorithm never gets rid of these terms, and always takes them into account with the same weight as the newest timestep. Alter-Re<sup>2</sup>, on the contrary, throws these terms away after a *WS* number of timesteps.

Ageing is another important upgrade to ReRe. It allows even the shortest length anomalies (only present for a few timesteps) to be detected, as it places greater emphasis on newly arrived data than older points. This way, the *AARE* curve rises swiftly enough to cross the *thd* curve and initiate the detection process. We argue that Alter-Re<sup>2</sup>, our approach to address the limitations present in ReRe, is able to significantly increase the detection precision of the original algorithm.

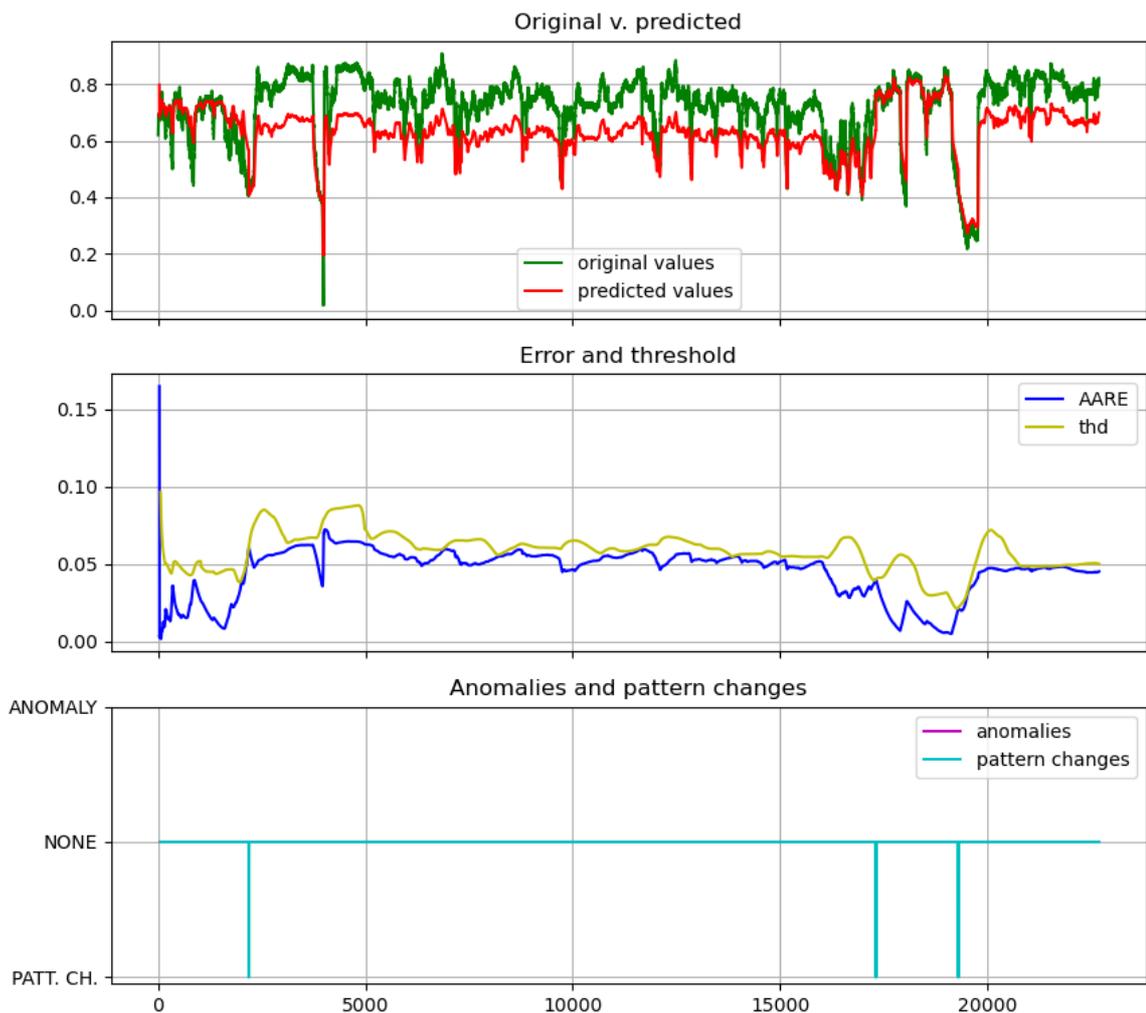
Apart from the ones shown in this section, we performed numerous further experiments to compare the performance of ReRe and Alter-Re<sup>2</sup>, using a wide variety of datasets. Ten datasets were from the type of data shown in this section (for more discussion on data types, see Section 7). These record various sources of information including CPU utilization, request latency, request count, and other network-related measurements. We downloaded all datasets from the NAB GitHub repository [18].

In these ten, similar type of datasets, Alter-Re<sup>2</sup> found 10 anomalies in total (this is the number of true positives), while ReRe managed to detect only 3. This three-fold increase in the number of true positives indicates that Alter-Re<sup>2</sup>, implementing even only relatively simple extensions, can substantially overperform ReRe, at least in case of certain dataset types. With regard to false positives (the number of signalled anomalies that are not flagged to be true), Ater-Re<sup>2</sup> detected only 4 anomalies (these came from only two datasets), while ReRe found only 1. We believe that these results prove the relevance of our improvements.

## 6.4 Performance Maintenance through Different Data Types

The goal of this experiment is to shed light on the operation of Alter-Re<sup>2</sup> on different kinds of datasets. So far, all previous experiments were performed on one certain type of data that ReRe was likely designed to perform well on. Here, we present two fundamentally different datasets and evaluate Alter-Re<sup>2</sup> on them. For a detailed discussion on the results presented in this section, please refer to Section 7.

The first dataset, depicted in Figure 6-6, shows machine temperature data. There are four anomalies according to the source of the data, the Numenta Anomaly Benchmark. These are extreme rises or drops in the temperature measured by the sensor. Alter-Re<sup>2</sup>, however, is unable to detect either one of the four.



**Figure 6-6: Alter-Re<sup>2</sup> output for machine\_temperature\_system\_failure.csv (AP: 2, WS: 1000). Flagged anomalies at timesteps 2410, 3987, 16341, 19516.**

The reason is, on one hand, that these anomalies do not have such a steep slope as in the previous datasets, so our algorithm simply declares a pattern change, the previous

few data points will not be that significantly different from the currently processed one. On the other hand, the previously mentioned constant offset is present here as well, which negatively impacts detection, as errors do not differ that much from normal behaviour. It starts at the first pattern change detection and lasts until the next one. The reason is the abnormal data points right before the first pattern change, since our algorithm retrains the LSTM model with those, and thus learns slightly wrong patterns (see Section 8).

The second dataset (shown in Figure 6-7) also shows CPU utilization data from an AWS server. The only difference from the dataset used in Section 6.3 is that correct behaviour here is a periodic spike in the data with the same amplitude. Therefore, abnormal data here means out-of-period or different amplitude spikes. Our algorithm is unable to detect these anomalies, as even if it signalled the individual spikes (as it sometimes did on similar datasets), these would not be correctly identified anomalies.

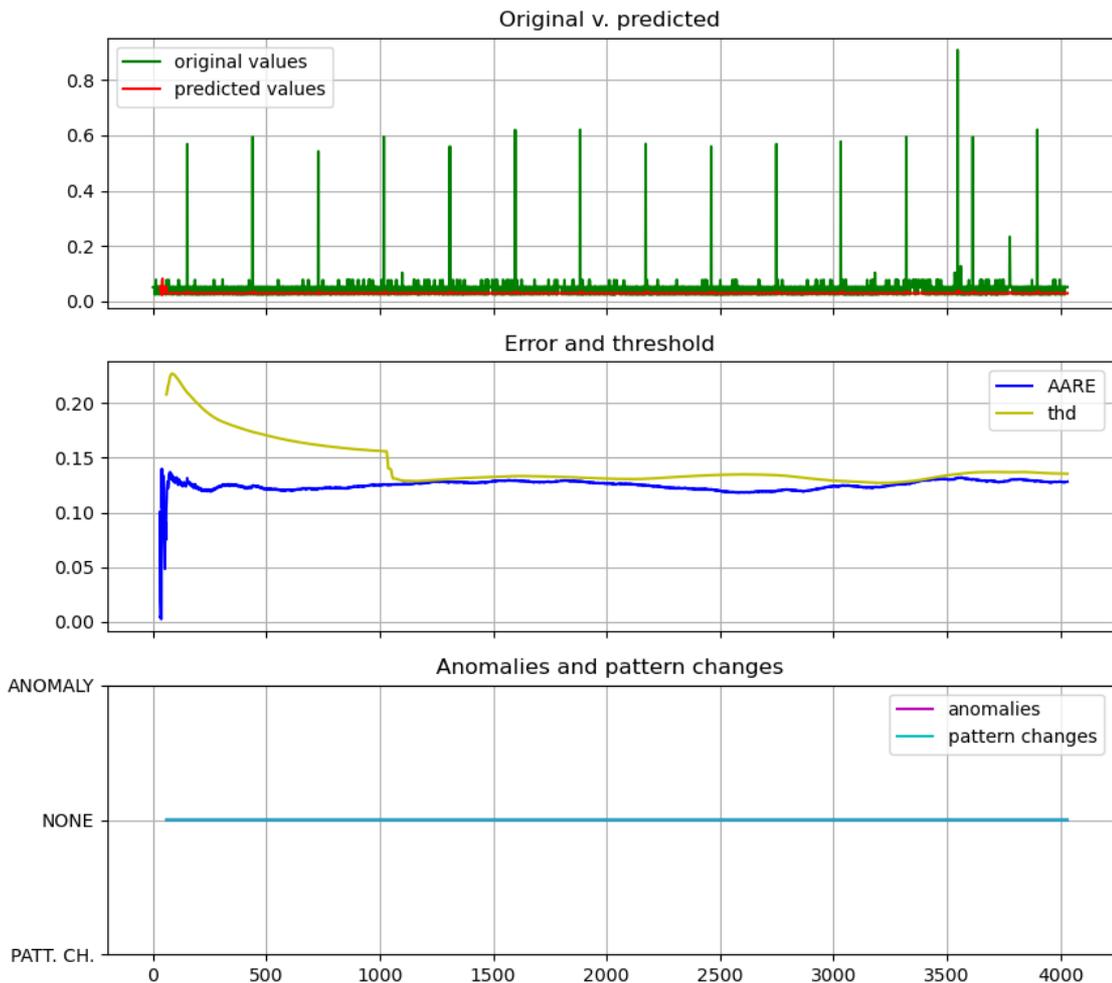


Figure 6-7: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_24ae8d.csv (AP: 2, WS: 1000).

Flagged anomalies at timesteps 3548, 3778.

## 6.5 Evaluation of automated hyperparameter tuning

In this section, we present our observations on Alter-Re<sup>2</sup> with automated parameter tuning turned on. We evaluate the efficacy hyperparameter tuning using two datasets. Specifically, we show the adjustment steps and criteria described in Section 5.3, Figures Figure 6-8 and Figure 6-9 have a slightly different layout to those in Section 6.1 to 6.4. Although the top diagram still shows the original and predicted values, anomaly and pattern change signals have been moved up directly below that. The third diagram records WINDOW\_SIZE and AGE\_POWER values, as these parameters are modified by the algorithm. Lastly, at the bottom, we show the results of evaluating the three detection criteria (see Section 5.3.3). The source of the datasets is NAB [18] as previously, and Alter-Re<sup>2</sup> parameters are set to the same values of  $B = 30$ , 30 neurons, 30 hidden units.

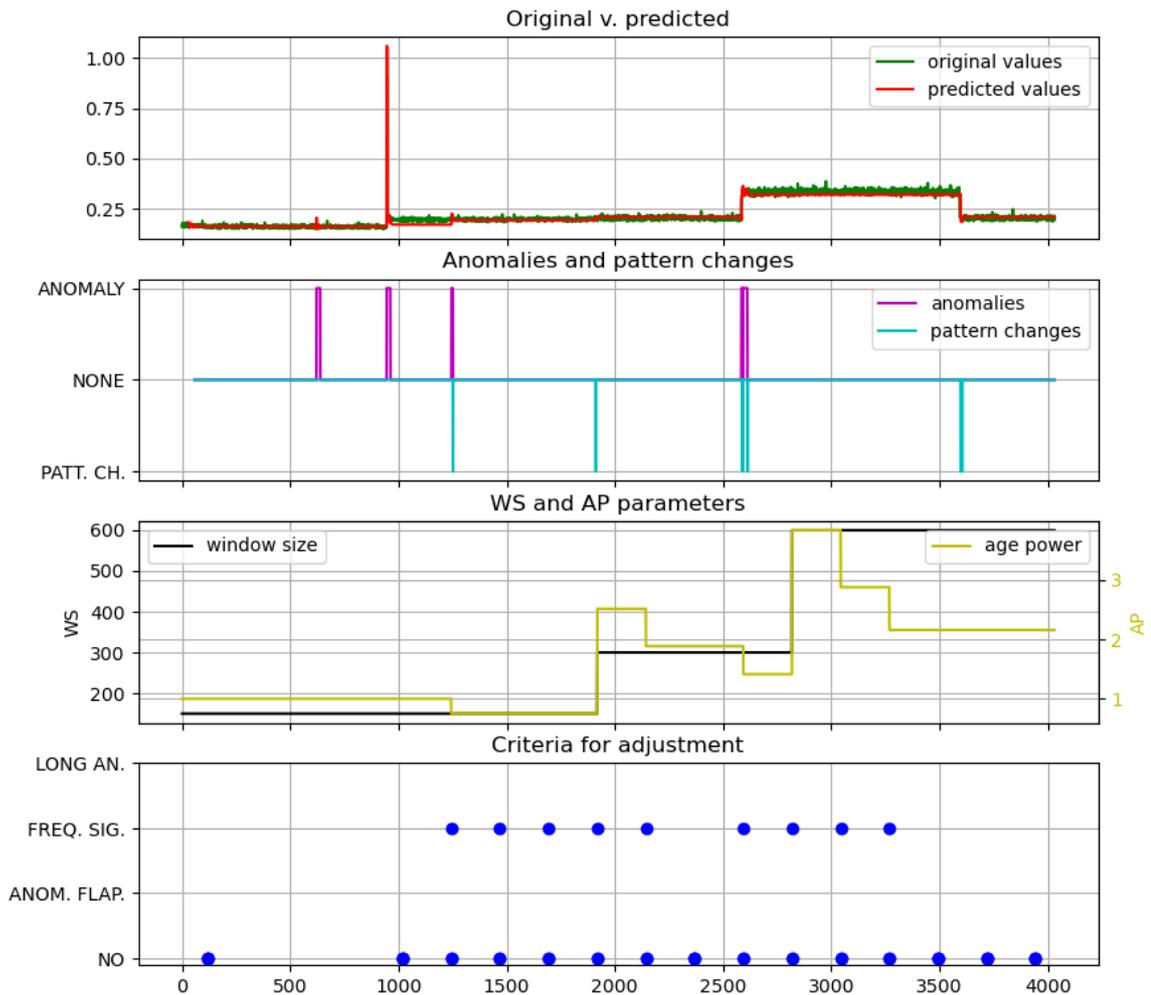


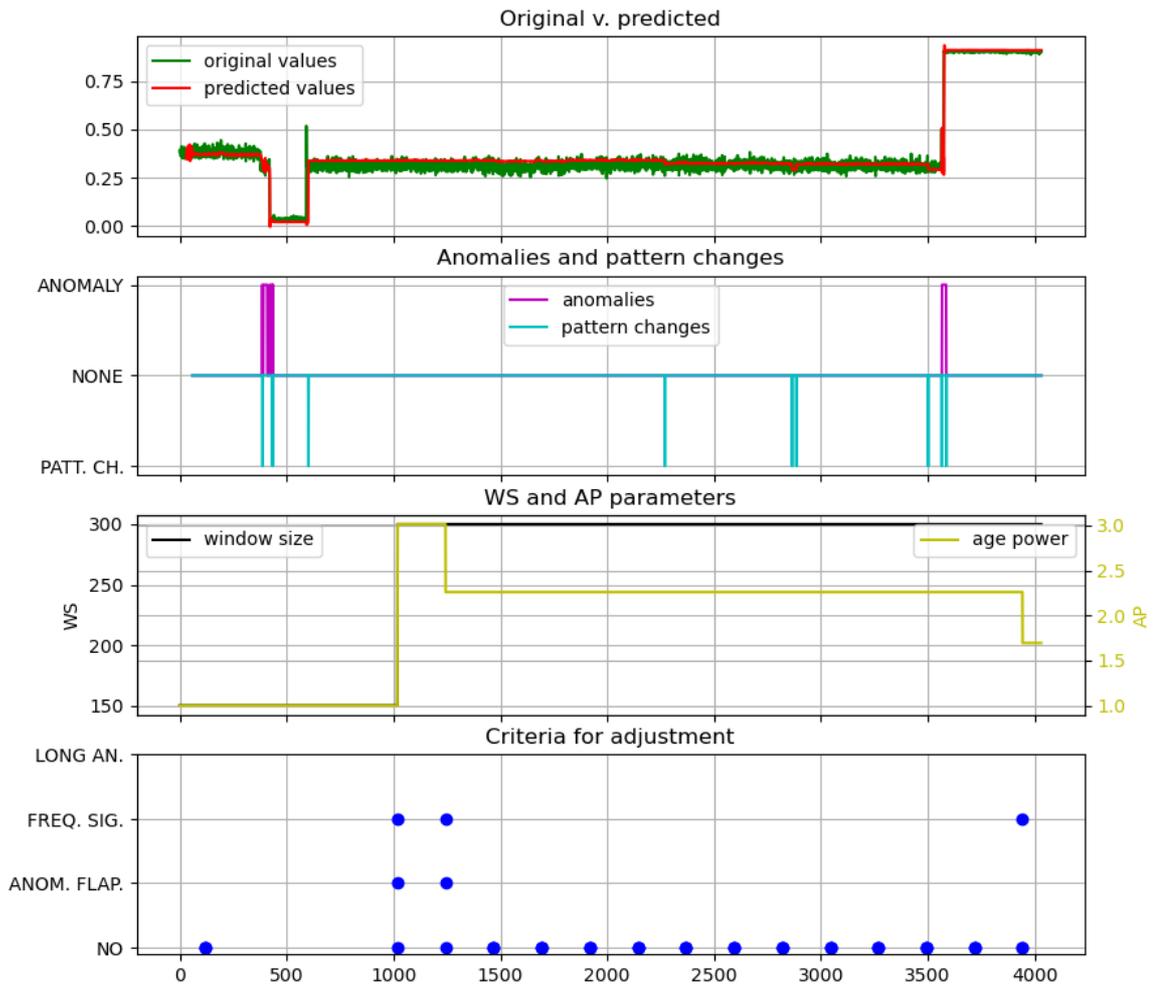
Figure 6-8: Alter-Re<sup>2</sup> output for rds\_cpu\_utilization\_e47b3b.csv (AP: auto, WS: auto).

Flagged anomalies at timesteps 947, 2586.

The first experiment, displayed in Figure 6-8, clearly demonstrates the operational principles discussed in Section 5.3. Alter-Re<sup>2</sup> starts with the default values of  $WS = 150$  and  $AP = 1$ . The initial check whether to increase the window returns false, as there have been no anomalies signalled up to that point. At around timestep 600, Alter-Re<sup>2</sup> reports a false anomaly, then, at 947, it signals a one. At about 1000 timesteps, the algorithm checks the conditions, and finds no errors, as the two anomaly signals are not frequent enough compared to the average change in the data. After the second check, however, frequent signalling is noticed, and the AGE\_POWER is lowered to give way to more datapoints (i.e., a larger area under the  $C_y$  curve). The following checks also return positive due to the introduction of another false alarm, and some pattern changes, but  $AP$  is not lowered further because of the condition  $AP < 1$ , to prevent complete disabling of ageing. Before 2000, the window size is tested again, and Alter-Re<sup>2</sup> can then resolve the frequent signalling issue by increasing  $WS$  and  $AP$ . As it persists, it decreases  $AP$  incrementally, and increases  $WS$  once again. During this, it signals another correct anomaly at timestep 2586. After all these adjustments, the parameters are set to correct values, as indicated by the fact that no check for extreme parameter values returns positive. Our experiments on the appropriate values for these parameters also suggested an  $AP$  value close to 2, and a  $WS$  value close to 1000 (in fact, these are the parameters used in the above experiment sections). This proves that in a few thousand timesteps, Alter-Re<sup>2</sup> is capable of adjusting the WINDOW\_SIZE and AGE\_POWER parameters to values enabling further precise detection.

The second experiment results are displayed in Figure 6-9. Just like in the previous one, the default values are set, and the initial checks return negative, as no signal was raised previously. When the first anomaly occurs in the dataset at timestep 421, Alter-Re<sup>2</sup> signals it immediately, but with a frequent state change. This behaviour is noticed by the first check at around 1000 timesteps, that determines that both the anomaly flapping and frequent signalling conditions are true. Consequently, Alter-Re<sup>2</sup> increases  $WS$  and  $AP$ . Since the window size is only checked and adjusted four times less frequently as  $AP$ , the next check allows the fine tuning of the AGE\_POWER parameter. It is lowered by 25% due to the same conditions present at the previous check. After this timestep, the algorithm operates normally, only signalling a few pattern changes that are not considered too frequent by the algorithm due to the changes from timestep to timestep in the original data (see Section 5.3.3). When the second anomaly occurs in the original dataset, Alter-

Re<sup>2</sup> signals it immediately. The last condition check shown in Figure 6-9 returns the frequent signalling criterion to be true. This is a result of the change in the pattern of the data at the second anomaly, as original values afterwards change much less between timesteps. This results in a smaller signal threshold, that no longer accepts the signals raised previously, and triggers the decrease of the *AP* parameter. As with the previous one, this experiment validates the designed algorithm for automatic parameter tuning.



**Figure 6-9: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_ac20cd.csv (AP: auto, WS: auto).  
Flagged anomalies at timesteps 421, 3576.**

In general, by analysing a few dozen more experiments, we conclude that the automated tuning algorithm chooses appropriate WINDOW\_SIZE and AGE\_POWER values for the type of data patterns shown in Section 6.3 (the same applies here about data patterns as discussed in Section 7). Although it yields a few false positive detections at the beginning, after approximately 2000 to 4000 timesteps, the parameter values stabilize, and major changes only occur with a change in the original data.

## 7 Discussion

In this section, first we comment on false positives produced by RePAD and the necessity of ReRe. Then we discuss the results of the experiments in this section. Finally, we present our detailed discussion on the types of data ReRe and Alter-Re<sup>2</sup> can perform well on. We also show other data patterns that are significantly harder to optimize for using these approaches strictly.

The purpose of ReRe, according to its authors, is to eliminate false positive anomaly detections (when the algorithm signals an anomaly, but there is no abnormal behaviour in reality) produced by RePAD. Yet, in our implementation, we did not perceive an excessive number of false positives. On the contrary, our experiments showed that RePAD failed to identify certain anomalies (false negative). As ReRe employs a second detector that can only disable anomalies detected by detector 1 (i.e., RePAD), ReRe cannot detect more anomalies than RePAD, only less or the same. For these reasons, we did not find a significant benefit of using ReRe instead of RePAD.

To understand how ReRe behaves on different datasets, first, we have to create categories of types of data. In our classification of certain patterns of data, we consider two main properties; what kind of pattern signals normal and abnormal behaviour. On this basis, we identified three main categories in the available datasets.

The first one has an almost constant average with data values appearing within a band around the average. Here, anomalies occur when there is a large spike in the data or a sudden and quick shift in the average. This is the type of data ReRe, and its improvement operates properly on. Experiments in Sections 6.2 and 6.3 all deal with this data pattern. The ‘look-back, predict-forward’ approach is ideal for the type even with a small look-back parameter ( $b$ ), as even a small number of training and predicting data will be sufficient for the LSTM model to infer the basic pattern from.

The second category of data types shows only activity at regular intervals. When working normally, periodic spikes are registered in the data. This might come from a periodic message processed or any other task that requires repetition after a preset time interval. Bytes written on or read from disks often fall into this category with network devices, as for most timesteps disks may be in the idle state. Anomalies here are aperiodic spikes, a longer width of the spikes, a different amplitude, or in essence every type of behaviour that differs from the periodic spikes with the same amplitudes. Figure 6-7 in

the third experiment shows exactly this kind of data. We believe ReRe is unable to detect these kinds of anomalies, as it is not prepared to deal with periodicity in any form. Even if we increase the look-back parameter  $b$ , the improved ReRe produces inconsistent and unreliable results. We conclude that this might be a consequence of the fundamental design of the algorithm.

The third and last type of data we have analysed is more similar to the first category (data with an almost constant average) but deviates from it in one significant aspect. With this type, normal behaviour is not constant; rather, data points rise and fall within acceptable limits and with an acceptable slope. When an anomaly occurs, it is most likely due to an extremely high or low value, or a very sudden change. We found that this type of data is mostly the output of temperature sensors, such as the first dataset in the third experiment in Section 6.4. Our experiments with such data patterns show that ReRe and its improvement are sometimes able to detect the most obvious and extreme anomalies, but they almost always mistake less striking ones for pattern changes or do not detect them at all. Measurement errors of these sensors make this task even harder, as data can fluctuate even if the temperature is not changing. Ultimately, the ability of our algorithm to detect these kinds of anomalies depends on the complexity of patterns the LSTM model can learn. Yet, increasing this would mean an increase in the number of neurons, epochs and the look-back parameter  $b$ , which are deliberately kept low to enable real-time use.

In conclusion, we would have to make fundamental changes to ReRe to enable the detection of these different kinds of anomalies, as it is originally best suited only for the first category of data types. Perhaps an intelligent detector built into the system could infer in the first few hundred timesteps which kind of data it is receiving and could then select the algorithm suited best or tune its parameters for the type. It could then monitor the data during the operation and choose a different algorithm or retune the parameters if necessary. Nonetheless, we have to conclude that the accuracy of ReRe is data type-dependent, still, with our improvements, though Alter-Re<sup>2</sup> produces convincing results in the domain it was designed for.

## 8 Research Implications

In this section, we briefly elaborate on our work’s implications and some of the directions for future research that stem from the analysis. Our ultimate goal is developing an anomaly detection algorithm that operates in real time on multivariate data embedded within a complex tool designed for network administrators.

As seen in most of the figures in Section 6 to a varying degree, and specifically mentioned in connection with Figure 6-4 and Figure 6-6, sometimes there is a constant offset between the predicted and original data values. Discussing Figure 6-6, we mentioned that this offset is the consequence of LSTM models being trained on slightly abnormal data that does not represent the whole dataset properly. In Figure 6-6, there are multiple pattern changes signalled by ReRe. Recall that then and only then is the LSTM model retrained. Looking at the figure, it is clear that if a pattern change is signalled at the right time, the offset is eliminated thanks to the normal and representative training data. Observing all figures of the experiments in Section 6 we can confirm this. Based on this finding, we plan to include an offset compensation component in the detection algorithm. It will observe the difference of the averages of the predicted and original values ( $|avg(v_y) - avg(\widehat{v}_y)|$ ), and if it notices a too high permanent difference, it will trigger an LSTM retrain with the last few data points.

Another indeterministic issue in our experiments was an excessive number of pattern changes. This results in many LSTM model retrains that are resource-intensive and time-consuming. To address this shortcoming, we plan to introduce a component that can modify the coefficient of  $\sigma_{AARE_t}$  that is currently set to 3. Through this, it is possible to modify the *thd* curve to have larger values, thus detecting fewer pattern changes.

We also plan to automate further hyperparameters of Alter-Re<sup>2</sup> in a similar approach as we did with the AGE\_POWER and WINDOW\_SIZE parameters. As we introduce new extensions of the algorithm, the usefulness of this automation will increase. Thus, we plan to enable setting as many parameters automatically as possible, based on the properties of the dataset.

In order for this algorithm to work in real-time applications, there are a few problems that need to be solved. As ReRe is in principle capable of online detection of

streaming data<sup>3</sup>, data preparation is the key area to focus on. More specifically, ReRe requires the input data to be scaled down and fall between 0 and 1. This has to be done in real time without knowing the boundaries of the input data. In some scenarios, we can make predictions on what is the possible highest and lowest input value we can observe (e.g., a CPU utilization percentage will be between 0 and 100), but if we strive for a general solution, this is not a viable option. All in all, we have to incorporate real-time normalization of the input data into Alter-Re<sup>2</sup>. We also have to adapt our implementation that currently works on offline datasets to manage the real-time receipt of input data.

Since we plan to deploy the algorithm as an anomaly detector for streaming telemetry data, we will have to solve the issue of multivariate data. A network device can produce a wide range of operational information. To find patterns of abnormal behaviour, we have to analyse multiple variables in real-time. Although ReRe was designed to run on univariate data, the LSTM model and its functions can handle extra dimensions. The only issue arises at signalling an anomaly. In future, we plan to devise a method to decide when to signal a collective anomaly based on the individual signals of variables. This may require domain knowledge, and specifically tuning the system for an application.

Finally, we briefly mention the issue of evaluating anomaly detection algorithms. The main reason for the complexity of this task comes from the very small frequency of anomalies compared to the frequency of normal data points. Depending on the application, there might also be different aspects that are more important than others. For example, one use case might prioritize the speed of anomaly detection, while another might place more emphasis on how many anomalies go unnoticed. The cost of a false positive or a false negative might also have to be considered. There are multiple metrics used in literature, such as precision  $\left(\frac{\# \text{ of true positives}}{\# \text{ of true and false positives}}\right)$ , recall  $\left(\frac{\# \text{ of true positives}}{\# \text{ of anomalies}}\right)$  and many other fractions, curves and diagrams. A possible solution is to create an aggregated metric that consists of application-specific variables and calculations.

Given that ageing and the sliding window mechanism can already introduce improvements in anomaly detection (as demonstrated in Section 6), it is very likely that incorporating the above discussed concepts can lead to further advances.

---

<sup>3</sup> This only depends on the capabilities of the machine it is deployed on and parameter settings.

## 9 Conclusions

Real-time anomaly detection in time-series data is an emerging area with approaches mostly based on neural networks, and increasingly often LSTMs. We surveyed state-of-the-art solutions and selected an LSTM-based real-time anomaly detection algorithm called ReRe.

We evaluated the selected method and found its performance limitations. Motivated by this fact, we developed Alter-Re<sup>2</sup>, an anomaly detection algorithm that seems to overperform ReRe. It provides a sliding window to limit memory and CPU stress below an upper bound. Furthermore, Alter-Re<sup>2</sup> implements also a mechanism for ageing of the data points, used for calculating error terms, to solve the issue of slow (or no) reaction to anomalies. We also introduced an algorithm capable of automating the tuning of the AGE\_POWER and WINDOW\_SIZE hyperparameters.

We rigorously evaluated Alter-Re<sup>2</sup> in several different scenarios. Our approach, implementing even only relatively simple extensions, achieved significantly better performance compared to ReRe in the investigated scenarios, detecting three times as many anomalies. Our algorithm showed reliable performance even in cases when ReRe fell short of detecting certain anomalies. We conclude that our automated parameter adjustment method has met our goals and can tune the values to a high accuracy.

Furthermore, we examined also how Alter-Re<sup>2</sup> performed on various types of data and drew conclusions for each category we divided data types into. We found that Alter-Re<sup>2</sup> worked appropriately only on data with an almost constant average as normal behaviour, nonetheless always overperforming ReRe. Other types of data require further investigation. In conclusion, our observations support the feasibility of our approach.

In future work, we plan to evaluate the applicability and usefulness of several other concepts. Our goals include offset compensation, adaptive threshold sigma-coefficient setting, automated tuning of further hyperparameters, real-time normalization, and multivariate data support.

As time-series data streams are now an integral part of almost every field of technology, the important tool of real-time anomaly detection deserves more attention. We are certain, that our contribution is valuable in facilitating the development of relevant techniques, yet we argue our approach has immediate real-world applicability as well.

## **10 Acknowledgements**

With a deep sense of gratitude, I would like to thank my thesis advisors Dr. Adrián Pekár and Dr. Károly Farkas (Department of Networked Systems and Services at Budapest University of Technology and Economics) for assisting me at every step of the process with their valuable guidance and continuous support.

I must also express my profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study.

# 11 List of Figures

Figure 3-1: Simplified layout of a feedforward neural network. *(original)*

Figure 3-2: Structure of RNNs. [19]

Figure 3-3: Comparison of the structures of RNNs and LSTMs. [21]

Figure 4-1: The slope of AARE depends on the current timestep. *(original)*

Figure 5-1: The operational principle of ageing. *(original)*

Figure 5-2: The area under the Cy curve. *(original)*

Figure 6-1: ReRe output for ec2\_cpu\_utilization\_825cc2.csv (no ageing, no window).  
Flagged anomalies at timesteps 1627, 1769. *(original)*

Figure 6-2: ReRe output for ec2\_cpu\_utilization\_ac20cd.csv (no ageing, no window).  
Flagged anomalies at timesteps 421, 3576. *(original)*

Figure 6-3: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_ac20cd.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 421, 3576. *(original)*

Figure 6-4: ReRe output for ec2\_cpu\_utilization\_5f5533.csv (no ageing, no window).  
Flagged anomalies at timesteps 1272, 2931. *(original)*

Figure 6-5: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_5f5533.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 1272, 2931. *(original)*

Figure 6-6: Alter-Re<sup>2</sup> output for machine\_temperature\_system\_failure.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 2410, 3987, 16341, 19516. *(original)*

Figure 6-7: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_24ae8d.csv (AP: 2, WS: 1000).  
Flagged anomalies at timesteps 3548, 3778. *(original)*

Figure 6-8: Alter-Re<sup>2</sup> output for rds\_cpu\_utilization\_e47b3b.csv (AP: auto, WS: auto).  
Flagged anomalies at timesteps 947, 2586. *(original)*

Figure 6-9: Alter-Re<sup>2</sup> output for ec2\_cpu\_utilization\_ac20cd.csv (AP: auto, WS: auto).  
Flagged anomalies at timesteps 421, 3576. *(original)*

## 12 Acronyms

<b>AARE</b>	Average Absolute Relative Error
<b>ADT</b>	AnomalyDetectionTs
<b>ADV</b>	AnomalyDetectionVec
<b>AP</b>	Age Power parameter
<b>AWS</b>	Amazon Web Services
<b>BGP</b>	Border Gateway Protocol
<b>eSNN</b>	evolving Spiking Neural Networks
<b>HTM</b>	Hierarchical Temporal Memory
<b>LSTM</b>	Long Short-Term Memory
<b>NAB</b>	Numenta Anomaly Benchmark
<b>RePAD</b>	Real-Time Proactive Anomaly Detection for Time Series
<b>ReRe</b>	A Lightweight Real-time Ready-to-Go Anomaly Detection Approach for Time Series
<b>RNN</b>	Recurrent Neural Network
<b>thd</b>	detection threshold
<b>WS</b>	Window Size parameter
<b>AP</b>	Age Power parameter
<b>ST</b>	Signal Threshold ratio

## 13 Bibliography

- [1] Ming-Chang Lee, Jia-Chun Lin, Ernst Gunnar Gran (2020): “*RePAD: Real-Time Proactive Anomaly Detection for Time Series*”. In: Advanced Information Networking and Applications. AINA 2020. Advances in Intelligent Systems and Computing, vol 1151. (pp. 1291-1302.). Springer, Cham. ISBN: 978-3-030-44040-4, DOI: [https://doi.org/10.1007/978-3-030-44041-1\\_110](https://doi.org/10.1007/978-3-030-44041-1_110).
- [2] Ming-Chang Lee, Jia-Chun Lin, Ernst Gunnar Gran (2020): “*ReRe: A Lightweight Real-time Ready-to-Go Anomaly Detection Approach for Time Series*”, preprint for Proceedings of the 44th IEEE Computer Society Signature Conference on Computers, Software, and Applications (COMPSAC 2020), IEEE. arXiv preprint arXiv:2004.02319. <https://arxiv.org/abs/2004.02319>.
- [3] Andrian Putina, Dario Rossi, Albert Bifet, Steven Barth, Drew Pletcher, Cristina Precup, Patrice Nivaggioli (2018): “*Telemetry-based stream-learning of BGP anomalies*”, In: Big-DAMA '18: Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (SIGCOMM '18), (pp. 15-20.). ISBN: 978-1-450-35904-7, DOI: <https://doi.org/10.1145/3229607.3229611>.
- [4] Piotr S. Maciag, Marzena Kryszkiewicz, Robert Bembenik, Jesus L. Lobo, Javier Del Ser (2019): “*Unsupervised Anomaly Detection in Stream Data with Online Evolving Spiking Neural Networks*”, preprint for 4th IEEE/IFIP International Workshop on Analytics for Network and Service Management. arXiv preprint arXiv:1912.08785. <https://arxiv.org/abs/1912.08785>.
- [5] Subutai Ahmad, Alexander Lavin, Scott Purdy, Zuha Agha (2017): “*Unsupervised real-time anomaly detection for streaming data*”, Elsevier: Neurocomputing volume 262 (pp. 134-147.). ISSN: 0925-2312, DOI: <https://doi.org/10.1016/j.neucom.2017.04.070>.
- [6] Swee Chuan Tan, Kai Ming Ting, Tony Fei Liu (2011): “*Fast Anomaly Detection for Streaming Data*”, IJCAI'11: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Two (pp. 1511-1516.). ISBN 978-1-57735-512-0, DOI: <https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-254>.
- [7] Feng Cao, Martin Ester, Weining Qian, Aoying Zhou (2006): “*Density-Based Clustering over an Evolving Data Stream with Noise*”, Proceedings of the Sixth SIAM International Conference on Data Mining (pp. 328-339.). ISBN: 978-0-89871-611-5, DOI: <https://doi.org/10.1137/1.9781611972764.29>.
- [8] *Twitter/AnomalyDetection* [Online code repository], Available: <https://github.com/twitter/AnomalyDetection> (accessed: 2020-10-28).
- [9] Nong Ye, Douglas Montgomery, Kevin Mills, Mark Carson (2019): “*Multivariate Metrics of Normal and Anomalous Network Behaviors*”, IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, (pp. 55-58.). ISBN: 978-1-7281-0618-2.

- [10] Georgios Kaiafas, Christian Hammerschmidt, Radu State, Cu D Nguyen, Thorsten Ries, Mohamed Ourdane (2019): “*An Experimental Analysis of Fraud Detection Methods in Enterprise Telecommunication Data using Unsupervised Outlier Ensembles*”, IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, (pp. 37-42.). ISBN: 978-1-7281-0618-2.
- [11] Aggelos Lazaris and Viktor K. Prasanna: “*An LSTM Framework For Modeling Network Traffic*”, 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, (pp. 19-24.). ISBN: 978-1-7281-0618-2.
- [12] Tae Jun Lee, Justin Gottschlich, Nesime Tatbul, Eric Metcalf, Stan Zdonik (2018): “*Greenhouse: A Zero-Positive Machine Learning System for Time-Series Anomaly Detection*”, preprint for SysML’18, February 2018, Stanford, CA, USA. arXiv preprint arXiv:1801.03168. <https://arxiv.org/abs/1801.03168>.
- [13] Sepp Hochreiter and Jürgen Schmidhuber (1997): “*Long Short-Term Memory*”, Neural Computation Volume 9, Issue 8 (pp. 1735-1780.). ISSN: 0899-7667, DOI: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [14] Kevin Gurney (1997): “*An Introduction to Neural Networks*”, Publisher: Taylor & Francis, Inc., CRC press. ISBN: 1-85728-673-1.
- [15] Alex Sherstinsky (2020): “*Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network*”, Physica D: Nonlinear Phenomena Volume 404, 132306. ISSN: 0167-2789, DOI: <https://doi.org/10.1016/j.physd.2019.132306>.
- [16] Early stopping. “*What is early stopping?*” <https://deeplearning4j.konduit.ai/tuning-and-training/early-stopping> (accessed: 2020-10-28).
- [17] Alexander Lavin and Subutai Ahmad (2015): “*Evaluating real-time anomaly detection algorithms – the Numenta Anomaly Benchmark*”, 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL (pp. 38-44.). ISBN: 978-1-5090-0286-3, DOI: <https://doi.org/10.1109/ICMLA.2015.141>.
- [18] *NAB: Numenta Anomaly Benchmark* [Online code repository], Redwood City, CA: Numenta, Inc. Available: <https://github.com/numenta/NAB> (accessed: 2020-10-28).
- [19] Bao, Wei & Yue, Jun & Rao, Yulei (2017): "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." PLoS ONE. 12. DOI: <https://doi.org/10.1371/journal.pone.0180944>
- [20] Sebastian Ruder (2016): “*An overview of gradient descent optimization algorithms.*”, arXiv preprint arXiv:1609.04747. <https://arxiv.org/abs/1609.04747>
- [21] Aliaa Rassem, Mohammed El-Beltagy and Mohamed Saleh (2017): “*Cross-Country Skiing Gears Classification using Deep Learning*”, arXiv preprint arXiv:1706.08924. <https://arxiv.org/abs/1706.08924>