Universiteit
Leiden
The Netherlands

Machine learning and deep learning approaches for multivariate time series prediction and anomaly detection

Thill, M.

Citation
Thill, M. (2022, March 17). *Machine learning and deep learning approaches for multivariate time series prediction and anomaly detection*. Retrieved from https://hdl.handle.net/1887/3279161

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/3279161

Note: To cite this publication please use the final published version (if applicable).

Universiteit Leiden

Machine Learning and Deep Learning Approaches for Multivariate Time Series Prediction and Anomaly Detection

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op donderdag 17 maart 2022
klokke 16.15 uur

door

Markus Thill

geboren te Patan, Nepal
in 1987

Promotiecommissie

Promotores:	Prof.dr. T.H.W. Bäck	
	Prof.dr. W. Konen	(Cologne University of Applied Sciences)
Co-promotor:	Dr. H. Wang	
Promotiecommissie:	Prof.dr. A. Plaat	
	Prof.dr. K.J. Batenburg	
	Dr. M. Baratchi	
	Prof.dr. H. Trautmann	(University of Münster)
	Prof.dr. E. Alba	(University of Malaga)

Abstract

In many real-world applications today, it is critical to continuously record and monitor certain machine or system health indicators in order to discover malfunctions or other abnormal behavior at an early stage and prevent potential harm. The demand for such reliable monitoring systems is expected to increase in the coming years. Particularly in the industrial context, in the course of ongoing digitization, it is becoming increasingly important to analyze growing volumes of data in an automated manner using state-of-the-art algorithms. In many practical applications, one has to deal with temporal data in the form of data streams or time series. The problem of detecting unusual (or anomalous) behavior in time series is commonly referred to as time series anomaly detection. Anomalies are events observed in the data that do not conform to the normal or expected behavior when viewed in their temporal context.

In the era of 'industry 4.0', 'cyber-physical systems' and 'big data', data-driven AI (artificial intelligence) approaches (designed to learn solely through data) from the fields of machine learning (ML), or deep learning (DL) – a sub-field of ML – have gained tremendous popularity. The majority of ML models are trained in a supervised fashion and require a labeled training data set. However, a particular problem in (time series) anomaly detection is that labeled data are usually relatively sparse and, as a consequence, supervised learning methods are mostly not feasible.

This thesis focuses on unsupervised machine learning algorithms for anomaly detection in time series. In an unsupervised learning setup, a model attempts to learn the normal behavior in a time series – which might already be contaminated with anomalies – without any external assistance. The model can then use its learned notion of normality to detect anomalous events. This work presents four unsupervised anomaly detection algorithms for multivariate time series, which can be used in different contexts: 1. SORAD learns an autoregressive prediction model and maintains a distribution of the prediction errors to detect anomalies. It can operate fully online and is up-and-running after a very short transient phase. Due to its online adaptability, it is advantageous for non-stationary time series. 2. DWT-MLEAD uses discrete wavelet transforms (DWT) to analyze a time series signal at different frequency scales to detect short-range and longer-range anomalies. It is also online-adaptable and computationally efficient. 3. LSTM-AD is a DL approach designed for longer time series, which uses a stack of recurrent long short-term memory (LSTM) neural networks to predict normal time-series behavior. Due to several extensions, it is particularly well-suited for quasi-periodic time series. 4. TCN-AE is a deep temporal autoencoder architecture, which – similarly to DWT-MLEAD – analyzes time series at different frequency scales and can learn short- and long-term relationships. It uses so-called dilated convolutional layers with learnable filters. TCN-AE is usually more compact in terms of model size than other DL models and can be trained faster due to its convolutional architecture.

We evaluate the algorithms presented in this work on challenging synthetic and real-world time series anomaly detection benchmarks, such as Mackey-Glass time series or electrocardiogram recordings, and compare them to other state-of-the-art algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background & Motivation

We often find ourselves in situations that seem unusual to us, and we get surprised by events or observations that do not seem to correspond to our expectations. But what is the reason for us reacting surprised? One possible explanation could be that we get used to continually recurring events and observations and memorize (although often not consciously) the relevant patterns that characterize them. Due to our cognitive abilities, we can process new information, match it with already existing patterns in our memory, and formulate predictions or expectations based on it (usually, also subconsciously). And as long as the new information matches our expectations, everything is fine. But as soon as we observe a new piece of information containing patterns that deviate from the norm, we run into a moment of surprise because we cannot connect these new patterns to the information stored in our long-term memory and our expectations (built up in repeated experiences) are not met. Oddly enough, we can – in many cases – naturally learn to identify what is novel or unusual without requiring any form of external assistance. As we will see later in the context of learning algorithms, this characteristic is often linked to a concept called "unsupervised learning". Our ability to identify the unusual helps us to cope with our daily life. Often, an unusual situation is also associated with a certain degree of concern for us, and we understand the situation as a warning of an impending problem.

Ignoring unusual or unexpected observations may have fatal consequences. In 1998, a high-speed train derailed and crashed into a bridge near the town of Eschede in Germany, causing the death of 101 people [42]. The accident's leading cause (followed by a sequence of unfortunate events) was a fatigue crack in one of the train's wheels, which was not discovered during inspections of the train in the weeks prior to the accident. Remarkably, already two months before the crash, several train staff members noticed and reported unusual noises and vibrations in the train compartment with the defective wheel – however, the complaints were not considered a safety issue. And although train wheels had been previously inspected with advanced testing devices, by 1998, the equipment was no longer in use due to its high false alarm rates (we will call these alarms false-positive errors and discuss the issue of false alarms in more detail later on). Instead, maintenance staff only performed visual inspections of the wheels with simple flashlights, which would not allow

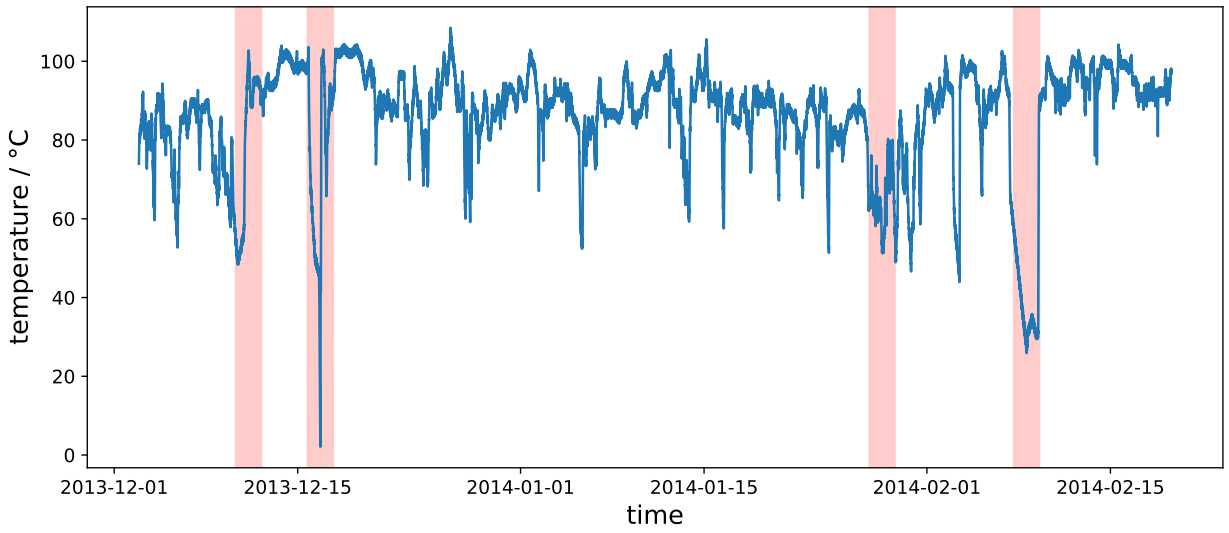

Figure 1.1: An example time series taken from the Numenta Anomaly Benchmark (NAB) [89].The graph shows the temperature sensor data of an internal component of a large industrial machine over its last few months of operation. The second anomaly (mid of December) is a planned shutdown of the machine. The catastrophic failure occurs end of February when the recordings end.

detecting fatigue cracks reliably. In the aftermath, it was discussed how microphones usage, in particular, could have possibly prevented the crash [10]. Today, so-called wayside monitoring points have been installed across the German rail network, which monitor passing trains with acoustic, optic, and other sensors and, for example, check the noise of passing trains for irregular patterns [38].

The previous example highlights the necessity of monitoring critical systems and detecting and responding appropriately to unexpected or abnormal behavior at an early stage in order to prevent failure and consequential damage. Similar to the wayside monitoring points mentioned before, many critical systems such as IT infrastructures, industrial machines, or power plants can be equipped with specialized sensors and software to record relevant health indicators and detect abnormal behavior. If the monitored data is visualized/presented appropriately, one could manually inspect the recorded data. An example is given in Figure 1.1, where one health indicator (temperature) of a large industrial machine is plotted over time. Although not all readers will have sufficient background knowledge about the monitored machine, most will spot at least two points in time that appear to deviate from the norm significantly.

However, while in the past, one could rely on humans' aforementioned extraordinary capabilities to analyze data and to recognize unusual and possibly problematic patterns, this has become increasingly challenging in recent years.

For example, one major challenge in many domains is the enormous amounts of data that have to be processed continuously. In the past years, we could observe the increasing degree of digitalization in companies and their processes, households, and many other public institutions and an extensive interconnection of systems. Today, companies are equipping even the smallest devices with sensors that consistently supply various measurements and other indicators. New technologies for the internet of things, cyber-physical systems (CPS), and other related domains enable manufacturers to equip such devices with considerable computing power, networking capabilities, and numerous sensors. These devices can consistently record and distribute various measurements and other information. The sheer endless amount of data and its inherent complexity can even overwhelm experienced human experts when analyzing and interpreting it manually. For this reason, among others, the necessity for automated, intelligent, and adaptable analytical approaches arose, and researchers invested much effort into developing new methods and algorithms. Especially in the field of machine learning (ML), the research community has made notable progress in recent years. In particular, deep learning (DL) – a subfield of ML – has recently received much interest. Essentially, machine learning is the generic term for computer algorithms designed to learn solely through data. There is no need for programming hard-coded rules into a model. Instead, an ML model is trained with a set of examples (the so-called training data set) and autonomously learns to identify the intricate underlying patterns within the data, which later allows the model to generalize to examples it has not seen before.

While ML/DL models can be and are used in a wide range of applications, such as computer vision[174], natural language processing (NLP) [125], or general game playing (GGP) [155], in this thesis, we will focus on one particular task in ML, which is concerned with the detection of unusual events in time series. In the scientific literature, this task is commonly referred to as *time series anomaly detection*.

1.1.1 Introduction to Time Series Anomaly Detection

Generally, there is no clear definition of the term "anomaly" since the categorization into anomalous or normal is heavily dependent on the application domain and the problem context. The Oxford Language dictionary defines an anomaly as

"something that deviates from what is standard, normal, or expected" [126].

Other (similar) definitions can be found in [26, 64, 23]. In the literature, also the terms *outlier detection*, *novelty detection* or sometimes *abnormality detection* can be found. Especially the term "outlier detection" is often used synonymously with "anomaly detection". While this understanding might be reasonable in many domains, especially in the time series analysis domain, which is the main focus of this work, the notion of anomaly and outlier is mostly different: outliers are single points that deviate significantly from all other points in the time series. Outliers are typically considered as single-point anomalies. However, not all anomalies in time series are outliers. Most anomalies in time series are temporal patterns

– containing a range of points – that are unusual given their temporal context. A possible definition that considers the temporal aspect of time series and which might be useful for this work is:

> *An anomaly is an observed pattern that, when viewed in its temporal context, does not conform to the normal or expected behavior due to its characteristics.*

There is a wide range of possible applications for time series anomaly detection: As already mentioned, an important application is system health monitoring, and fault detection, which is, for example, often embedded in predictive maintenance (PdM) approaches [119]. Other relevant applications are intrusion detection systems, patient monitoring systems in medical domains, or event detection in sensor networks [26]. This thesis will focus less on the details of different application domains of anomaly detection algorithms. Instead, as discussed in more detail below, this thesis aims to investigate more general aspects of time series anomaly detection. Nevertheless, we will investigate a few examples coming from some of the domains mentioned above.

1.1.2 Challenges

Up till today, anomaly detection in general and more specifically in time series has remained a complicated task. There are several challenges in the field of time series anomaly detection, which we believe are still not sufficiently addressed in the literature:

Availability of labeled data Many machine learning approaches require so-called *labeled (annotated) data* for their training to learn a general model that can later predict the outputs for new inputs. In these cases, the training algorithm processes examples with pairs consisting of an input and the desired output. After training (during the inference phase), the model can predict the unknown outputs for new given inputs. Since a "teacher" or "supervisor" is needed to pass the desired outputs to the training algorithm, this type of learning is called *supervised learning*. In the context of time series anomaly detection, "labeled data" means that each data point in a time series has been classified as either normal or anomalous by a domain expert. However, in a vast amount of real-world anomaly detection problems, no or insufficient labeled data exists. This increases the difficulty of the training and benchmarking of anomaly detection algorithms. One reason for this situation is that the data set's annotation process is usually very cumbersome and expensive; a human expert has to analyze and judge every piece of data. Since most anomalies in time series are not single points but longer sub-sequences, a particular problem in the annotation process is to identify the anomalous regions. Often, it is not evident at which instance of time an anomaly begins and where it ends. While large publicly available datasets for benchmarking new methods are available in other domains, such as visual object recognition (ImageNet [37], Open Images [84]), the amount of benchmark data for time series anomaly detection is somewhat limited.

Anomalies are rare events Another critical challenge for many anomaly detection problems is the simple fact that anomalies are rare events. Even if large amounts of data are available, they will mostly represent normal (nominal) instances. Due to the high class-imbalance (many examples which are normal and only very few which are anomalous), it is hard to train a supervised machine learning model.
An alternative machine learning type is so-called unsupervised learning. No annotations are needed in such setups, and the algorithms attempt to learn patterns and structure in the data without external supervision.

Finding suitable boundaries between normal and anomalous behavior In many applications, it is not trivial to define where to draw the line between normal and anomalous behavior. Especially since anomalies are only encountered very rarely, algorithms do not have many examples (sometimes not any examples at all) in order to learn a decision boundary. A common problem is that not all anomalous patterns which might occur can be known beforehand, and almost every anomaly has a unique pattern or fingerprint. Often, models are trained under the assumption that the data only contains normal behavior. The models attempt to learn a boundary that surrounds this presumably normal data. This task is often referred to as one-class classification [114] and is considered more difficult than other classification tasks. In practice, the data itself usually contains noise, and also the supposedly normal data is regularly "contaminated" with anomalies, which adds extra complexity to the learning task.

Non-stationary environments In many real-world problems, one has to deal with environments which dynamically change over time. Such environments undergo so-called concept drifts or concept changes [47, 170, 177], where certain statistical properties of variables continuously change over time. Time series with constantly changing means, variances, or trends are simple examples of such behavior. In practice, one often encounters dynamically changing time series in which also the regions of normality/abnormality permanently change over time. The challenge for anomaly detection algorithms is progressively updating their models, generalizing the data, and adapting their understanding of normality/abnormality. Especially if an algorithm has to operate on a data stream, the necessity arises to adapt quasi "on-the-fly" to new situations or concepts. Machine learning tasks involving such streaming data are considered rather challenging since many classical learning approaches are not applicable due to their offline character: offline learning algorithms are trained on a static batch of data. They require a complete repetition of the training procedure if new examples are added to the data set. Due to memory and time constraints, such approaches are typically infeasible for problems with streaming data. In such cases, it is necessary to operate in an online (an extreme case is "in real-time", which adds time constraints to the problem) setting on the data and process the data in an example-by-example manner and incrementally learn from every new example, without re-training a completely new

model each time. However, online approaches are associated with several additional new challenges, as discussed in the next chapter(s).

High false alarm rates Since the misclassification of true anomalies in the real-world usually is associated with severe consequences, most anomaly detection algorithms are configured very sensitively. In effect, the algorithms will trigger many alarms and are thus much more likely to classify normal data points as anomalous. A normal data point incorrectly classified as anomalous is called false positive. Since all alarms have to be investigated in real-world applications, false alarms cause much overhead (e.g., for machine operators). It might also happen that the responsible persons become less attentive and, as a consequence, ignore actual anomalous situations. If the false-positive rate gets too large, the expected beneficial effect of the anomaly detection algorithm has to be questioned. As described in the example mentioned earlier about the tragic derailment accident in Eschede, the use of advanced technical equipment for testing train wheels was discontinued since too many false-positive errors were reported.

High-dimensional Time Series While many might consider the detection of anomalies in univariate (1-dimensional) time series as relatively simple, the detection in their multivariate counterparts is typically of tremendous complexity since anomalies cannot be identified in individual series generally. Many anomalies only become apparent by analyzing the time series in all dimensions simultaneously. In other cases, the anomalous patterns range over very few or even only one dimension. However, they are very hard to spot since they are embedded in a high-dimensional space. While it might be possible to perform the anomaly detection in reduced lower-dimensional space for the latter case, it is much more challenging to detect high-dimensional temporal anomalies.

1.1.3 Underlying Research Questions

Based on the aforementioned challenges in anomaly detection, we motivate the work in this thesis with the following central research questions:

> **Q1:** Is it possible to successfully train and apply novel unsupervised machine learning models for anomaly detection and do they advance the state of the art?

This first research question is very general and will accompany us throughout the thesis. There are several sub-questions and assumptions linked to this question. For example, how can "successful" be quantified? The following chapter will describe several well-known performance measures (such as the F_1-score) which indicate an algorithm's performance on an individual time series or even a whole set of time series. By "successful", we mean achieving a performance (according to the specified measures) at least similar to other (unsupervised) state-of-the-art algorithms, ideally even surpassing them.
A central point of this work will be the investigation of *unsupervised* approaches. We will

assume that the training data is not annotated (no anomaly labels will be passed to the algorithms). Only for assessing the performance, the real anomaly labels will be used. We will make the mild assumption that the majority of the training data contains normal instances and that anomalies are relatively rare. However, if not mentioned otherwise, we will not remove any anomalies (or other noisy instances) from the training data, which increases the difficulty of learning and requires noise-resilient algorithms.
Although mentioned before, another aspect that should be highlighted again is the algorithms' requirement to operate on higher-dimensional time series. Considering the inherently temporal nature of the considered data and the multidimensionality increases the anomaly detection task significantly.

> **Q2:** Given certain characteristics of the time series data, can we advise which algorithm is most suited for detecting anomalies?

The second research question is a continuation of the first one. However, it focuses on whether it is possible to classify time series according to various properties and subsequently recommend which anomaly detection algorithm is likely to work best for the given problem. In an ideal case, *one* unique algorithm can be found that is applicable to a wide range of time series and outperforms all other algorithms on the considered problems. However, since this question is framed particularly in light of the data's complex temporal nature and great diversity, it is unlikely that a universally good method exists which performs equally well on all kinds of time series data. For example, such an algorithm would have to be able to deal with complex short- and long-term patterns, with periodic (seasonal) or quasi-periodic behavior and predictable or unpredictable time series. In order to answer this research question, it is necessary to identify distinct characteristics of different time series and find algorithmic approaches to address them.

> **Q3:** How can online learning methods be successfully used for anomaly detection in time series or data streams?

This last research question especially (but not exclusively) addresses problems with non-stationary environments. It has to be investigated how to design online algorithms that can incrementally adapt to new concepts in the data and learn the new notion of "normal". A particular challenge in online setups is the so-called stability-plasticity dilemma [21]: The learning process requires stability in order to protect the already acquired knowledge, but also plasticity, for the integration of new knowledge. A suitable balance has to be found to prevent that too much plasticity causes catastrophic forgetting on the one hand and that too much stability hinders the model from learning new concepts.

In the following chapters, we will address these research questions and propose several approaches in an attempt to answer these questions.

1.2 Thesis Outline

Chapter 2 gives an overview of the state of the art in the field of time series anomaly detection and introduces several benchmarks and algorithms that we use in this work. Furthermore, we describe the general benchmarking/scoring process for the remaining chapters. Chapter 4 is mainly concerned with research questions RQ1 & RQ3 and presents an unsupervised simple online regression anomaly detection algorithm (SORAD) and investigates the importance of its online elements on different types of time series. In chapter 5, we introduce the DWT-MLEAD algorithm. The algorithm uses the discrete wavelet transform (DWT) to analyze time series and detect anomalies at different time scales. We first propose an offline variant of the algorithm and subsequently discuss and implement necessary modifications to obtain a fully online algorithm. A deep learning (DL) approach, called LSTM-AD, based on stacked long short-term memory (LSTM) networks, is introduced in chapter 6. Like SORAD, LSTM-AD predicts a time series's normal behavior, and the prediction errors on several prediction horizons are used to detect anomalous behavior. We apply LSTM-AD to a set of electrocardiogram (ECG) recordings containing various types of arrhythmias. TCN-AE, a temporal convolutional autoencoder based on dilated convolutional neural networks, is presented in chapter 7. The model encodes time series into significantly shorter sequences and uses its decoder's reconstruction error as an indicator for anomalies. TCN-AE is evaluated on anomalous Mackey-Glass time series and, further on, on a more extensive ECG signal set. Finally, in chapter 8, we conclude this thesis and give an outlook on possible future work. In all chapters, we investigate research question RQ1 and implicitly also RQ2. Additionally, chapters 4 & 5 attempt to find an answer for research question RQ3.

Chapter 2

Fundamentals

2.1 Online Anomaly Detection

Especially in an online setting, which is often a requirement in practice, anomaly detection is usually a more difficult task than in the offline case, for several reasons: an online algorithm cannot process a time series several times (e.g., using batch learning approaches). Instead, the algorithm has to gradually update its knowledge with every new data instance arriving. Every instance is processed at maximum once. Also, often stability issues, such as numerical overflows or diverging behavior, arise when models have to be learned incrementally, with one example at a time, especially if the algorithm operates on an infinite data stream. Furthermore, back-dating of anomalies is usually not possible in an online setup and, hence, anomalous events have to be detected instantaneously. This would also require the algorithm to be computationally feasible in order to run in real-time. On the other side, online algorithms can also have advantages. The most notable advantage might be the adaptivity capabilities, which allow online algorithms to learn in dynamically changing environments and to adapt to concept drifts or concept changes. In general, online learning approaches have several properties that are convenient for many learning tasks, especially if streaming data is involved:

- Online approaches can incrementally adapt their models with each new example. In contrast to offline methods, it is not necessary to keep the whole dataset in the memory.
- Online models can be used simultaneously to make predictions and learn.
- Concept drift adaptation: While offline machine learning approachescannot track concept drifts, online learning algorithms generally can adapt to new situations. Also, in situations where the data generating process adjusts its behavior to the model (e.g., spammers who change their e-mails according to the spam filter's current configuration), online-adaptable models can be preferable.

Next to the mentioned advantages, online learning methods are also challenging in several aspects:

- As mentioned before, one main issue that is often faced is a trade-off between the online learning algorithm's adaptation speed and its stability. For example, a fast learning algorithm can quickly adapt to new concepts in the data and might become

unstable easily, e.g., in the presence of random noise. On the other side, an algorithm that adapts itself less aggressively could be less sensitive to noise but might not fully trackconcept changes in the data. This problem is also known as the stability-plasticity dilemma [21].

- Many of the existing machine learning algorithms are inherently offline and might require significant adjustments to achieve an online formulation.
- Additional complications arise if the rate of change and frequency of the concept drifts varies over time.

2.2 Scoring Process for Anomaly Detection Tasks

In order to compare the performance of the different algorithms on the described benchmarks (Section 2.3), suitable performance metrics are required. Similarly to binary classification tasks, an algorithm can classify every instance in a time series either as normal or anomalous. Subsequently, one can compare the classes predicted by the algorithm to the ground truth labels. For time series taken from real-world problems, the anomalies are often labeled manually. This might lead to some inaccuracies in the labeling process since experts (e.g., based on knowledge in the domain) might interpret certain instances differently. For artificial time series, the labeling process can generally be automatized and is mostly very accurate. The exact scoring procedure using anomaly windows is described in the following.

2.2.1 Anomaly Score & Anomaly Threshold

In practice, there are two approaches to report anomalous behavior: 1. The algorithms directly assign a binary label to each data point or, 2. they output a so-called *anomaly score* for each data point, as an intermediate indicator for the degree of abnormality of the corresponding point. This anomaly score is a continuous (usually between 0 and 1) value and reflects the algorithm's predicted probability of a data point being anomalous. Low values are an indicator of nominal behavior and high values indicate that an unusual situation has been observed. The first approach has the disadvantage that there is no possibility to control the sensitivity of the algorithm since labels are directly assigned to each point. The second approach allows to specify a so-called *anomaly threshold*, with which a sensitivity level can be set. Points with a score above the threshold are classified as anomalous, all other points are classified as nominal. By reducing the threshold the algorithm will get more sensitive and detect more anomalies. However, at the same more false alarms will be issued. Finding a suitable threshold is heavily dependent on on the application. For example, in critical medical health monitoring systems the anomaly detection algorithms might have to be more sensitive (having a relatively low threshold).

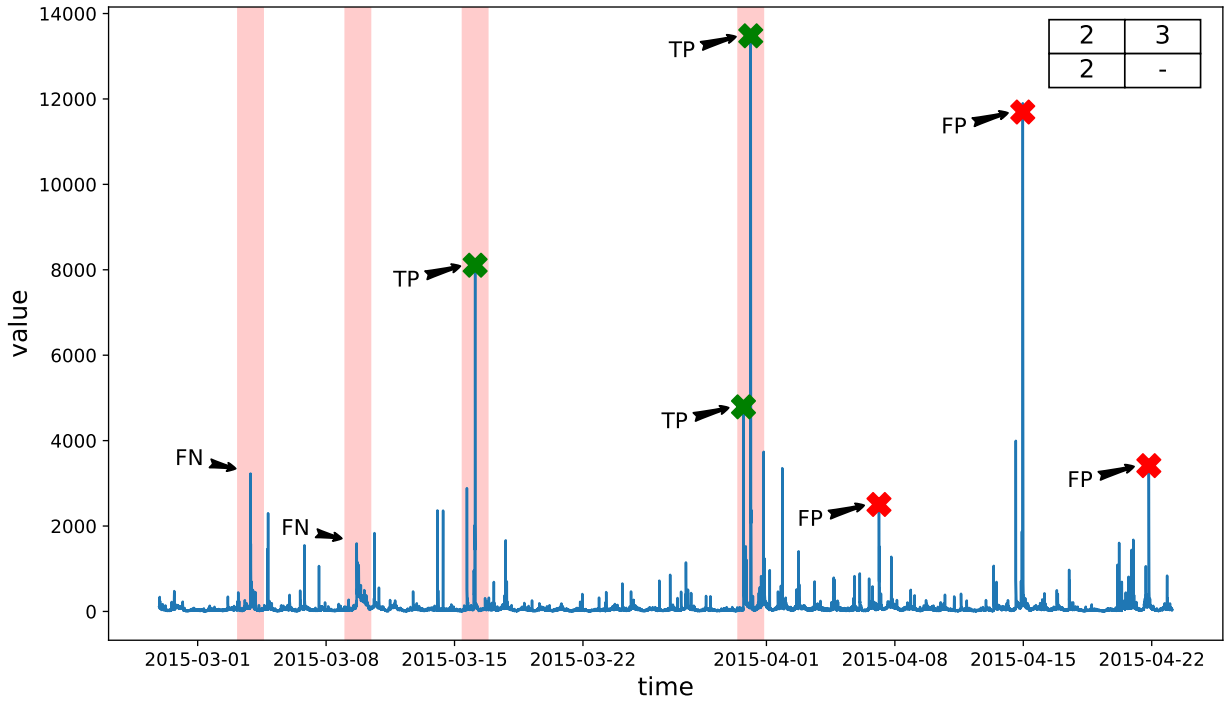

Figure 2.1: A time series taken from the Numenta Anomaly Benchmark (NAB) with in total 4 anomalies (anomaly windows indicated by red shaded vertical bars). In this plot, we illustrate the construction of the confusion matrix (Section 2.2.2) for some sample detections. The confusion matrix for this particular example is depicted in the top-right corner. The last anomaly window contains two detections, however, only the first detection will be counted as TP. In this example, we do not show TN in the confusion matrix since TN usually is of limited interest for anomaly detection tasks.

2.2.2 Anomaly Windows & Confusion Matrix

Usually, anomalies in time series are not single points and certain instances of time. Most anomalies span a longer range in the time series. In many cases, it is also not entirely clear, where exactly an anomaly begins and where it ends. A useful approach used in practice is to simply specify a so-called anomaly window of appropriate length, which is centered around the ground truth label(s). In order to correctly detect an anomaly, an algorithm would have to classify at least one point within the anomaly window as anomalous. Detections that fall into the anomaly windows are considered as true positives (TP). However, only the first true positive in each window is counted. The remaining detections in the same window are ignored. If an algorithm fails to flag any instance inside an anomaly window as anomalous, this will be considered as one false negative (FN). All detections outside the window are considered as false positives (FP). All other points, which are not marked as anomalous, are considered as true-negatives (TN). However, TNs play only a minor role in the evaluation of the algorithms, as they represent the vast majority of all points. In some cases, especially

for very long time series (such as ECG data), we will permit to group a small range of false positives (e.g., if they span less than one heart beat) into a single false positive, in order to prevent situations where false positives will dominate the scoring process. We will use the anomaly windows throughout this thesis to denote anomalies. Table 2.1 summarizes the construction of the confusion matrix.

Table 2.1: Confusion matrix for time series anomaly detection tasks.

		Actual	
		Anom.	Norm.
Predicted	Anom.	TP	FP
	Norm.	FN	TN

The scoring process using anomaly windows is illustrated in Figure 2.1.

2.2.3 Algorithm Performance Measures

In our setup, the real anomaly labels are not passed to the algorithms (i.e., the task is treated as an unsupervised learning task like in a real-world application setting). Hence, each anomaly detection algorithm has to learn the normal structure of a time series and has to be able to identify anomalies among the observed time series values. The ground truth labels are used only for evaluation purposes, or to obtain anomaly thresholds for all algorithms (for example, using the EAC-criterion, as described below), which allow a fair comparison.

From the aforementioned quantities TP, FP, FN, and TN, additional metrics can be derived: Precision (also called positive predictive value) takes into account how many false-positive errors were produced and is defined as the number of correctly identified anomalies divided by the number of all predicted anomalies:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.1}$$

Recall (also called sensitivity) measures the detection rate of true anomalies and is defined as the number of correctly identified anomalies divided by the overall number of true anomalies:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.2}$$

Both measures are independent of TN, which usually only plays a minor role in anomaly detection tasks due to the typically highly unbalanced class labels. Depending on the anomaly threshold of the algorithm, there is a tradeoff between precision and recall. While precision and recall are important metrics, it is not always possible to compare two algorithms

based on these measures. For example, one algorithm might achieve a high recall but a low precision and another algorithm, vice versa, a high precision and low recall. In such situations, a direct comparison is infeasible. In order to fairly assess algorithms based on these two objectives (precision and recall), the algorithms are typically compared based on the so-called equal accuracy (EAC) where precision and recall are approximately equal. We will use EAC as one performance indicator. Another possibility is to generate a multi-objective plot, where precision and recall are plotted against each other for a large range of anomaly thresholds.

The F_1-score combines precision and recall into a single value and is defined as their harmonic mean:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}} \tag{2.3}$$

Since the F_1-score considers both, precision and recall, and weights both equally, the F_1-score will be the most important measure during our later experiments. A perfect algorithm would achieve a precision and recall of 1, which would also result in $F_1 = 1$. Two additional metrics, which are used less commonly for anomaly detection (since they are based on TN) are the false-positive rate (FPR) and the positive likelihood ratio (PLR):

$$FPR = \frac{FP}{FP + TN}, \tag{2.4}$$

$$PLR = \frac{\text{recall}}{FPR}. \tag{2.5}$$

Finally, other relevant performance indicators of an algorithm are its computation time (for training and inference) and its memory requirements. Usually, we will report average computation times on a certain computer and the number of trainable parameters of the models.

In the literature, another commonly reported metric is the area under the ROC (Receiver Operation Characteristic) curve, in short AUC-ROC, which summarizes the performance of an algorithm when the anomaly threshold is varied. However, we do not use AUC-ROC for our scoring process, since it is not particularly well suited for imbalanced data sets (ROC curves plot recall against FPR). In such cases, Precision-Recall (PR) curves, as used by us, give better information. Simiarly to AUC-ROC, it is also possible to compute the AUC-PR, however, we usually plot the PR curve, which is more intuitive than a single metric.

2.3 Time Series Anomaly Detection Benchmarks

In the following, we describe several time series anomaly detection benchmarks that we use in this work for benchmarking purposes. All benchmarks are publicly available and serve as a common reference point for comparing the investigated algorithms.

2.3.1 Yahoo's Webscope S5 Dataset

The Webscope S5 dataset is a relatively large anomaly detection benchmark which is publicly available [87]. It consists of 367 time series (with overall 572,966 data points). On average, each time series has approximately 1,500 instances. The Webscope S5 benchmark is split again into the 4 datasets A1, A2, A3 and A4 containing 67, 100, 100 and 100 time series. While class A1 has real data from computational services, classes A2, A3, and A4 contain synthetic anomaly data with increasing complexity. Example plots are shown in Figs. 2.2–2.5. The ground truth anomaly information is available for all time series. For Yahoo's Webscope S5 data we place anomaly windows of size 10 around the labeled anomalies. In our setup, the ground truth anomaly labels are not provided to the anomaly detection algorithms, which have to learn to separate anomalies from normal behavior in an unsupervised fashion.

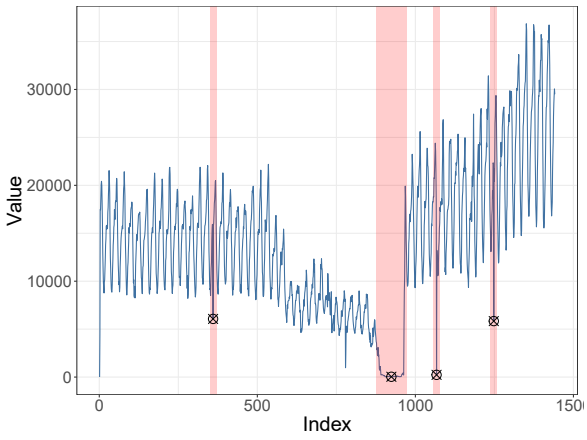

Figure 2.2: Example plot for the A1 data. Black crosses mark the true anomalies and the red vertical bars indicate their anomaly windows. The time series of the A1 data were taken from real world applications and anomalies were manually labeled.

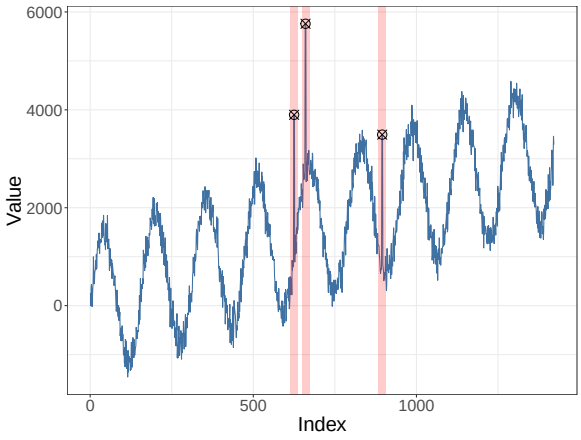

Figure 2.3: Example plot for the A2 data. The time series of the A2 data were generated synthetically. The time series include a trend, a seasonal (periodic) component and noise. The anomalies were added at random instances.

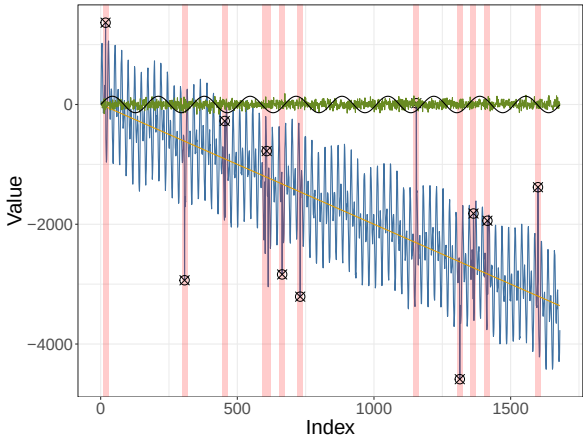

Figure 2.4: Example plot for the A3 data. The time series of the A3 data (blue) were generated synthetically. The time series include a trend, three seasonal (periodic) components and noise (green). The anomalies were added at random instances.

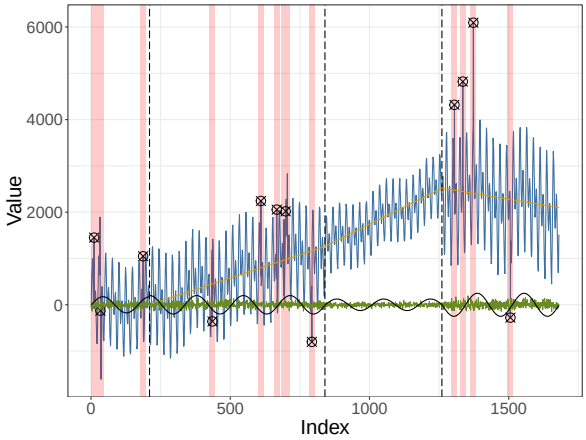

Figure 2.5: Example plot for the A4 data (blue). Similar to the A3 data. Additionally, change point anomalies are introduced (indicated by the vertical dashed lines) in which the characteristics (such as slope or frequency) of the individual components are changed.

2.3.2 The Numenta Anomaly Benchmark

The Numenta Anomaly Benchmark (NAB) [89] is a publicly available dataset that consists of 58 time series with in total 365,558 data points – the shortest series containing 1,127, the longest containing 22,695 and the average series containing approx. 6,300 instances. The majority of the time series are real-world data coming from application areas such as server monitoring, network utilization, sensor readings from industry and social media statistics [89]; 11 time series were generated artificially, from which 5 are anomaly-free. In total, over all 58 time series, 115 anomalies were labeled, most of which were identified and documented manually by a human expert. The ground truth anomaly labels are available for all considered time series. An example taken from NAB is shown in Figure 2.1.

2.3.3 The MIT-BIH Arrhythmia Database

The MIT-BIH arrhythmia database [52, 112, 113] contains two-channel electrocardiogram (ECG) signals of 48 patients of the Beth Israel Hospital (BIH) Arrhythmia Laboratory. Each signal was recorded with a sampling frequency of 360 Hz and has a length of approximately half an hour, which corresponds to 650 000 data points each. The two channels recorded are the modified limb lead II (MLII) and a modified lower lead V1, in a few cases V5. Each recording contains on average 2160 ± 365 heartbeats, and in total, there are 54 087 heartbeats. The individual signals have a quasi-periodic behavior, with differing heartbeat patterns for each subject. To understand the complexity of anomaly detection in quasiperiodic ECG data, we show in Fig 2.6 an anomaly example. The normal signal is quasiperiodic (peak height and other small details may differ from period to period to some extent). The anomalous region has a very similar peak, yet it comes earlier than expected. The model which predicts anomalies thus has to learn not only the shape of the peak but when to expect it. This requires to process long-range information since the local neighborhood of the anomalous peak looks absolutely normal.

There are many different events in all ECG time series, which were labeled by human experts. The whole list of heartbeat annotations is found in [112]. InTable 2.2, we summarize nine event classes, which are considered as anomalous.
The unsupervised anomaly detection approaches investigated in this thesis rest on the assumption that most time-series data is normal (nominal) and that anomalous events are relatively seldom.

Since there are time series in the database which contain many events (in several cases, the vast majority of heart beats are anomalous events) we limit ourselves to time series with a small to moderately number of events.

ECG-13 For our initial experiments in Chapter 6, we consider only those time series with 50 or fewer events. Overall, 13 out of 48 time series will be used. We call this data set the *ECG-13* data. The selected time series contain 130 anomalous events from 6 anomaly classes, which are listed in Table 2.2.

ECG-25 Later, in Chapter 7, we will extend the benchmark and select those 25 time series with 250 or fewer anomalous events and call this data set the *ECG-25* data. The ECG-25 data contains 721 events. Additionally, for the ECG-25 data, we also perform some simple pre-processing steps, which are commonly done in practice: Since several of the raw 2-dimensional ECG signals contain a lot of noise and exhibit simple baseline drifts, each signal is filtered with a bandpass filter parameterized with the cutoff frequencies of 2 and 20 Hz. These are values which are commonly used for the processing of ECG signals in practice (cf. Pan-Tompkins algorithm [127]). This bandpass filter removes most of the high-frequency noise in the signal as well as drifts in the baseline. In order to reduce the training time of the models, we down-sample (using an anti-aliasing filter) each ECG signal in ECG-25 by a factor of $n_{samp} = 5$. This shortens the length of the ECG signal from originally 650 000 time-steps to just $T = 130\,000$ time-steps without losing too much information from the signal. The down-sampling was done to speed-up the training time of the individual algorithms.

For the ECG-13 and ECG-25 data, we normalize each input time series to zero mean and unit variance before generating the training samples. In the MIT-BIH benchmark, only the R-peaks (the tops of the main spikes on the ECG line) of the QRS complex are labeled (either as normal or with the corresponding arrhythmia type). However, it is not possible to locate the position of an anomaly in exactly one point of the ECG signal.

Thus, as for the other benchmarks, we specify an anomaly window for each anomaly. The anomaly window is centered around the given label and contains 400 (80 after downsampling the time series in ECG-25) data points before and after the label, which corresponds to approximately 2 seconds (or roughly one heartbeat before and after the label). A more detailed database description can be found in [113].

Table 2.2: Anomaly types in the ECG-13 and ECG-25 data considered for the experiments. The descriptions are taken from [112]. The second column shows the overall number of the various anomaly types for ECG-13 and the third column for ECG-25 considered ECG signals.

Code	# ECG-13	# ECG-25	Description
a	6	11	Aberrated atrial premature beat
A	44	235	Atrial premature beat
e	–	10	Atrial escape beat
f	–	22	Fusion of paced and normal beat
F	2	25	Fusion of ventricular and normal beat
J	–	3	Nodal (junctional) premature beat
V	53	374	Premature ventricular contraction
x	8	19	Non-conducted P-wave (blocked APC)
|	17	22	Isolated QRS-like artifact
Σ	130	721	

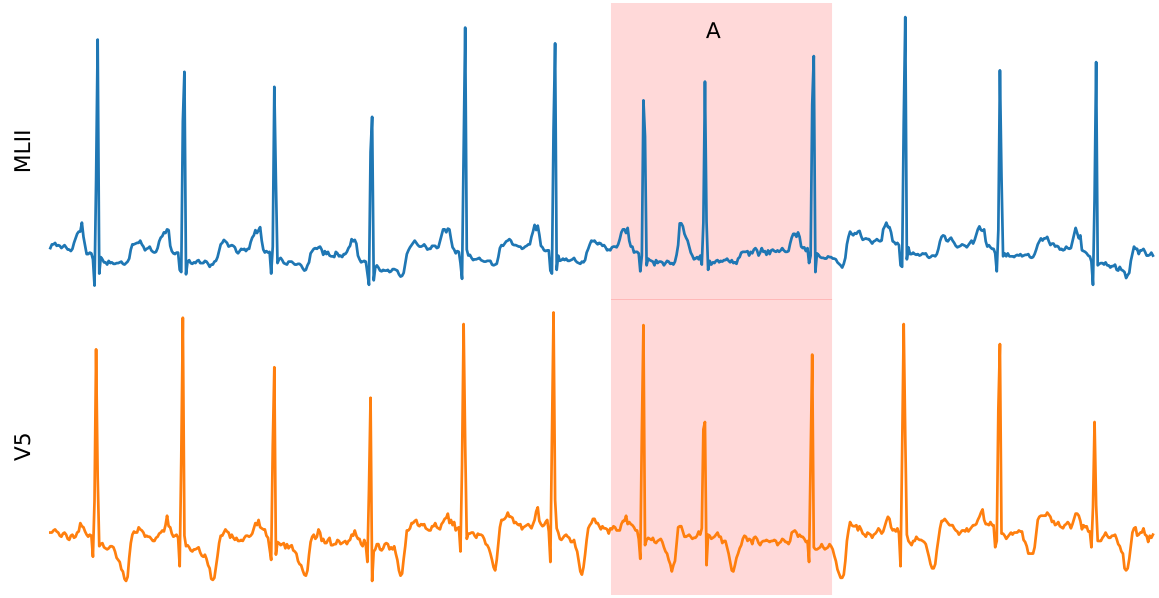

Figure 2.6: Example ECG signal from the MIT-BIH Arrhythmia database. This is just one type of anomaly out of the set of 9 different anomaly types (to be discussed later in more detail, see Table 2.2).

2.3.4 The Mackey-Glass Anomaly Benchmark

During our work on a paper [166], we developed a non-trivial synthetic benchmark, named Mackey-Glass anomaly benchmark (MGAB) [167]. The motivation for this benchmark was to be able to generate fairly long synthetic time series (with 100 000 points) with well-defined, yet non-trivial anomalies, in order to investigate and evaluate larger deep learning anomaly detection models. MGAB allows creating an unlimited amount of quasi-periodic time series data with steerable difficulty by simply adjusting a few parameters of the MGAB generation process (e.g. time delay, smoothness parameters). In Figure 2.7, an excerpt of ab MGAB time series is shown.

Mackey-Glass (MG) time series are known to exhibit chaotic behavior under certain conditions. MGAB contains 10 MG time series of length $T = 10^5$. Into each time series 10 anomalies are inserted with a procedure described in Section 2.3.4.2. In contrast to other synthetic benchmarks, the introduced anomalies are for the human eye nearly invisible in the context of the normal (chaotic) behavior. Overall, we generate 100 anomalies in 10^6 time series points. The benchmark data and the detailed procedure for generating these and similar benchmark data are publicly available at GitHub [167].[1]

[1]GitHub repository: `https://github.com/MarkusThill/MGAB/`

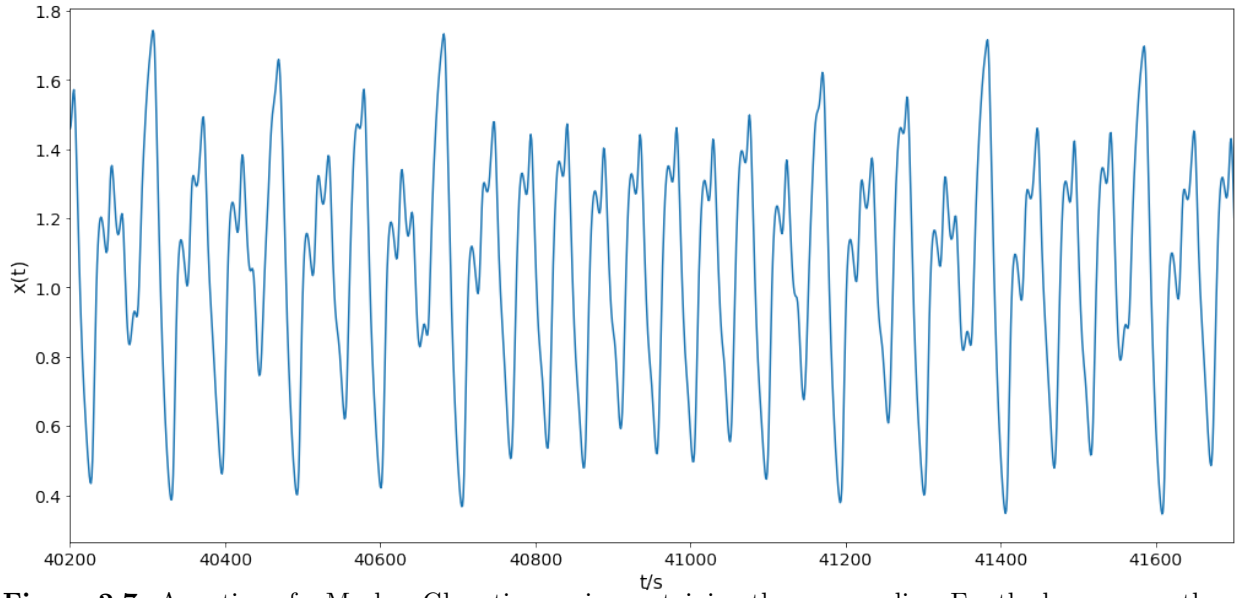

Figure 2.7: A section of a Mackey-Glass time series containing three anomalies. For the human eye, these anomalies might be hard to spot. Fig. 2.10 unveils the location of the anomalies.

2.3.4.1 The Mackey-Glass Equation

The Mackey-Glass equation [106] is a so-called non-linear time delay differential equation (DDE), which is defined as

$$\frac{dx}{dt} = \beta \cdot \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t), \tag{2.6}$$

where τ, n, β and γ are real numbers. Additionally, an initial condition and a past (history) are required in order to integrate the equation. Commonly, initial condition and past are chosen to be constant with $x(t \leq 0) = h$. Depending on the parameterization of Eq. (2.6), the resulting time series develops various patterns of chaotic and periodic dynamics. One possibility to visualize the chaotic behavior of a Mackey-Glass attractor is a time delay embedding graph, which plots the delayed series $x(t-\tau)$ against the actual Mackey-Glass time series $x(t)$. An example is shown in Fig. 2.8. In order to generate all Mackey-Glass time series used in this work, we use the JiTCDDE package for Python [11], which is designed to accurately integrate delay differential equations (DDEs).

2.3.4.2 Generating Anomalies in Mackey-Glass Time Series

In order to create the Mackey-Glass Anomaly Benchmark, we first generate a sufficiently long time series having a dimension of $d = 1$ using the JiTCDDE [11] solver with the parameters $\tau = 18, n = 10, \beta = 0.25, \gamma = 0.1, h = 0.9$. The integration step size is set

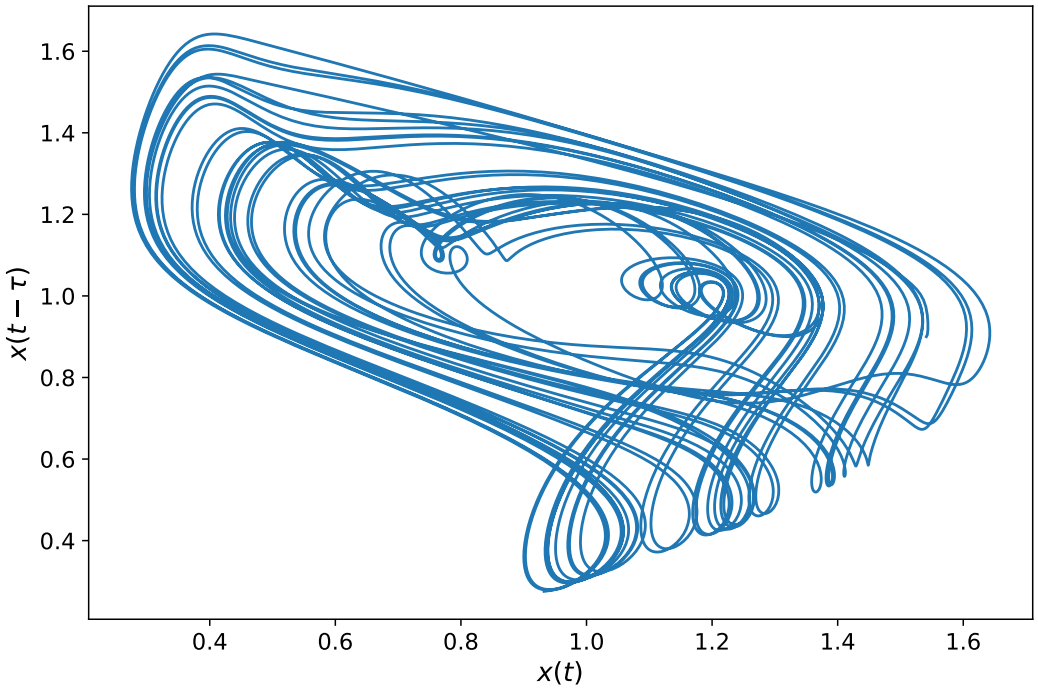

Figure 2.8: Time delay embedding of the Mackey-Glass attractor, which is described in Section 2.3.4.2, for $\tau = 10.0$, $\beta = .25$, $\gamma = .1$, $h = .9$ (history).

to 1. The maximal Lyapunov exponent (MLE) of $\lambda_{mle} = 0.0061 \pm 0.0002$ suggests that the generated time series is (mildly) chaotic. Subsequently, we split this series into ten same-sized individual time series and insert 10 anomalies into each time series.

The basic idea is to split the time series in two places and remove the segment in between. If the two split points are carefully chosen, the manipulation will be hardly visible later, without the knowledge of the dynamics of the MG series. This procedure is repeated to insert 10 anomalies in each time series. It is ensured that a new anomaly does not remove a previously inserted anomaly. Finally, the 10 time series are truncated to a length of $T = 10^5$. In order to make the benchmark data more similar to real-world data, we add uniform random noise to each of the 10 time series after inserting the anomalies. For this purpose, we sample 10^5 values uniformly from $[-0.01, 0.01]$. Overall, 96% of each time series are normal data points and 4% anomalous.

2.4 Time Series Characteristics

Based on certain characteristics that time series may or may not exhibit, some algorithms might be better suited for anomaly detection tasks than others. In the following, we list

several characteristics that we found important in deciding which algorithm to choose. In Section 8.1, we will summarize how well the anomaly detection algorithms investigated in this work are suited for particular time series, after classifying them according to the following characteristics.

Small data sets Some algorithms require many data for their training process. Especially DL approaches usually require significant amounts of data for training. If no suitable pre-trained model is available, it is mostly not recommended to train a DL model on small data sets. Algorithms with less trainable parameters are more suited for these cases.

Multivariate time series Many time series in practice are multivariate. However, not all algorithms are applicable to time series with more than one dimension.

Anomalies appear at different time scales In many cases, it is necessary to analyze time series at different time scales in order to detect short-term and longer-term anomalies.

Long-term correlations Some problems require that the anomaly detection algorithm learns the time series's behavior over very long ranges of time to detect anomalous patterns reliably (see Section 2.3.3 for an example). Algorithms with only a limited temporal memory or a small temporal receptive field are only partially or not suited for these types of problems.

Unpredictable time series Prediction-based anomaly detection algorithms assume that future instances of a time series can be predicted (forecasted) to some extent. However, some time series in practice are inherently unpredictable, although they show nominal behavior, which can be learned with other methods. A common approach to handle unpredictable time series is to use reconstruction-based algorithms (based on autoencoders or similar architectures).

Weak non-stationary behavior In many time series recorded under real-world conditions (such as ECG signals), one can observe weak forms of non-stationary behavior such as baseline wandering or changes (gradual or sudden) in the signal noise or quality.

Strong non-stationary behavior For strong non-stationary behavior, online-adaptable algorithms are required which can adapt to new concepts in the data. Examples for such behavior are (sudden) notable changes in the trend, in the signal frequency or in the recurring signal patterns.

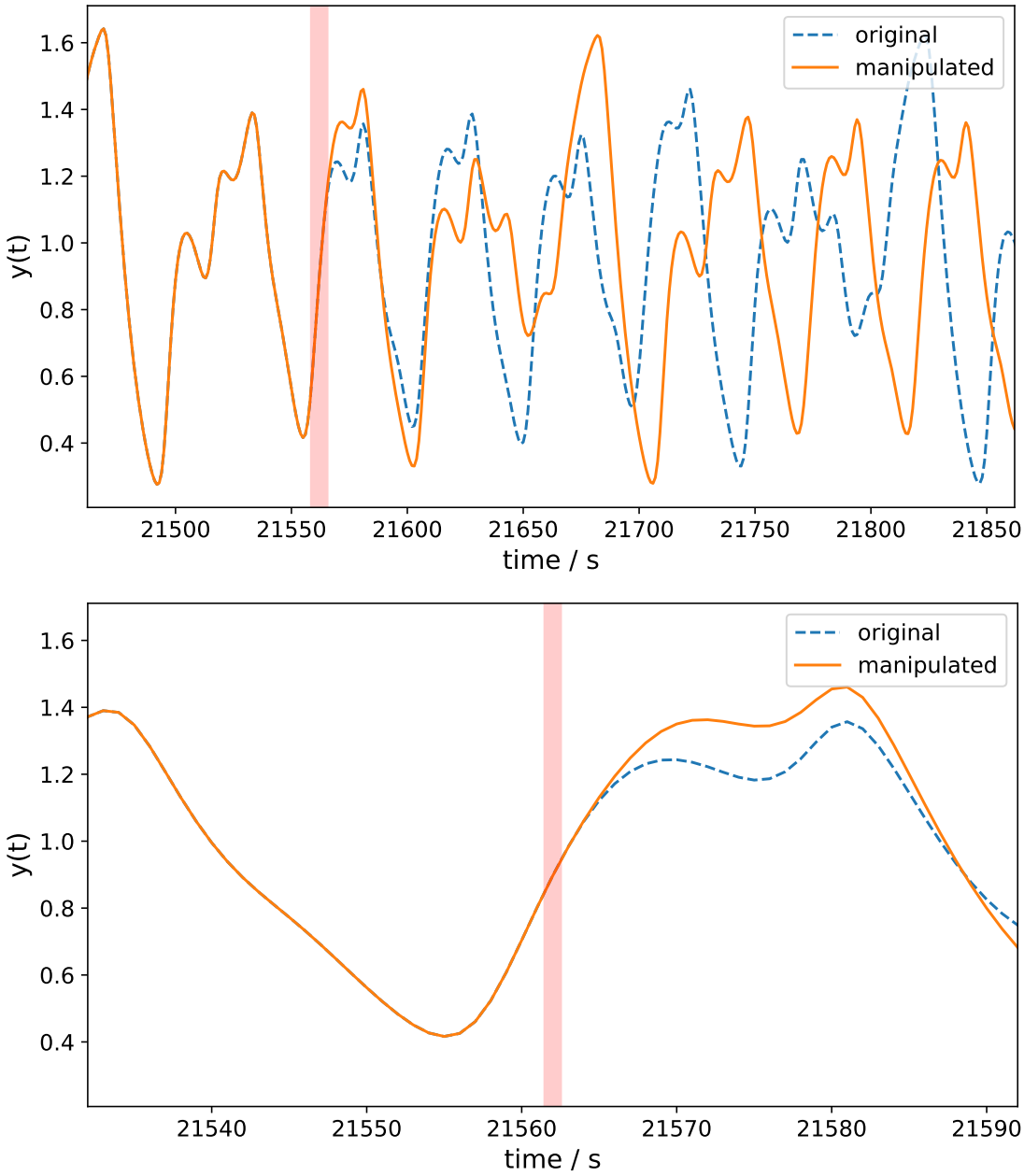

Figure 2.9: Top: Example for the creation of a Mackey-Glass time series with a temporal anomaly. The original time series (dashed line) is manipulated in such a way, that a segment is removed and the two remaining ends are joined together. In this example, the interval $[21562, 21703]$ is removed from the original curve. The resulting manipulated time series (solid line) has a smooth point of connection, but significantly differs from the original. Bottom: Zoomed-In. The red shaded area indicates the position where the anomaly was inserted.

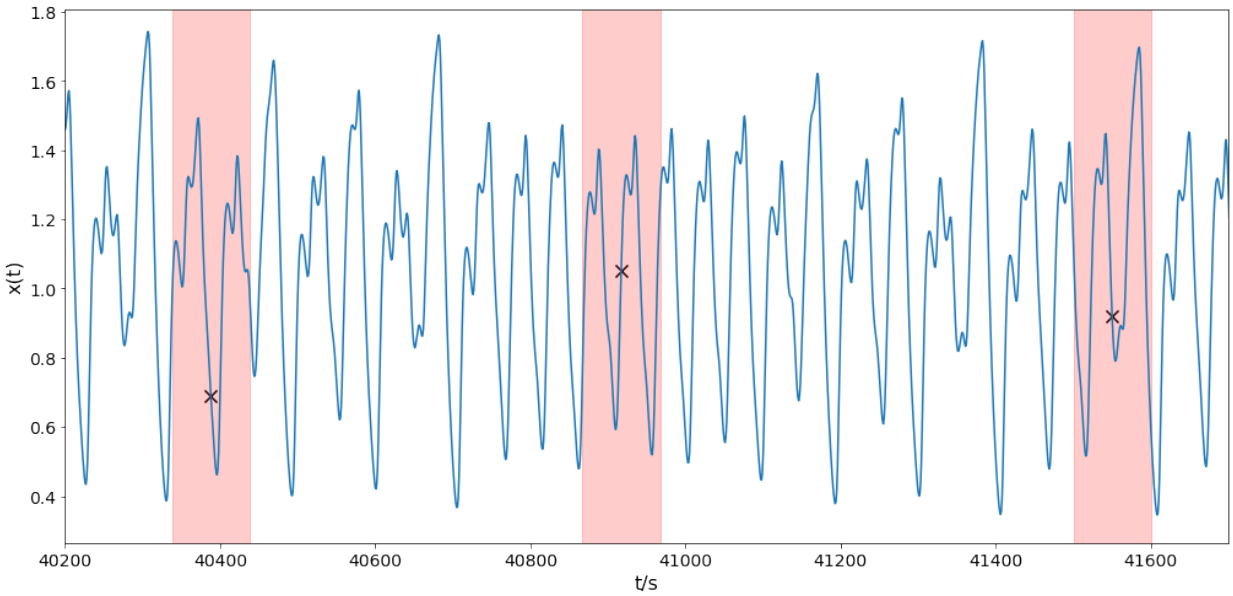

Figure 2.10: This graph shows the same section of a Mackey-Glass time series as Fig. 2.7, but now reveals the location of the anomalies in the time series. The anomalies are at $t_1 = 40388$, $t_2 = 40917$ and $t_3 = 41550$. The positions are indicated by the black crosses in the plot.

Chapter 3

Related Work

In this chapter, we shortly describe several state-of-the-art anomaly detection algorithms that we use in this work for comparison purposes. Additionally, we give an overview of the current relevant state of the art in the field of time series anomaly detection. Since this field is very wide, the algorithms and benchmarks discussed in this chapter are in no way intended to represent a comprehensive list. For a detailed overview and classification of the different algorithms and datasets, the reader is referred to general anomaly (novelty) detection survey papers [26, 64, 131, 135, 109, 110, 23, 128] or survey papers with the focus on time series [31, 8, 16, 55]. Additional, more specific related work is also presented in the following chapters.

3.1 Algorithms used for Comparison Purposes

This section describes all time series anomaly detection algorithms used in this thesis for comparison purposes. Our own contributions will be described in the course of the thesis: Chapter 4 – 7 introduce SORAD, DWT-MLEAD (offline & online variant), LSTM-AD, and TCN-AE, respectively.

3.1.1 ADVec

Twitter's ADVec algorithm [173] is a robust online anomaly detection algorithm designed to detect local and global anomalies in time series with seasonal/periodic behavior and underlying trend. ADVec uses a method called Seasonal Hybrid ESD (S-H-ESD), which is based on the Generalized ESD (extreme Studentized deviate) test [141], combined with robust statistical approaches and piecewise approximation. It is available as open-source R package "AnomalyDetection" from Github.[1] The algorithm requires three main parameters: The first parameter α describes the level of statistical significance with which to accept or reject anomalies. As in the other algorithms, this parameter can be interpreted as an anomaly threshold. ADVec requires a second parameter, a period-length. The third parameter, $\max_{anoms}$, determines the maximum number of anomalies that the algorithm will

[1] `http://github.com/twitter/AnomalyDetection`

detect as a percentage of the data. We found that the setting of the parameter $\max_{anoms}$ is crucial (but not difficult to select) for the performance of ADVec.

3.1.2 NuPIC

Numenta's online anomaly detection algorithm NuPIC [160] is based on the hierarchical temporal memory (HTM) model [49] which is biologically inspired by the neocortex of the brain. NuPIC is also open-source and can be obtained from GitHub[2]. To verify the correctness of our NuPIC installation, we applied it to the Numenta Anomaly Benchmark (NAB) ([89], section 2.3.2) using the standard parameter settings and confirmed that we could exactly reproduce the results[3] published in [89]. The anomaly detection algorithm behind NuPIC, similarly to other approaches, is based on the assumption [160, 89] that time series are predictable to a certain extent. At each time step t, NuPIC performs several predictions for time step $t+1$. These predictions are compared with the time series's actual value at $t+1$ and based on the prediction errors, an anomaly score is formed. NuPIC also tracks and adapts the estimated mean and variance for the anomaly scores in an online fashion and can thus handle dynamically changing behavior to some extent. Furthermore, it is supposed to work on a large variety of datasets without manual parameter tweaking [89]. Although the parameters can be tuned with an internal swarming tool [3], a tool to aid automatic parameter search for a given dataset, we decided to use the standard parameter settings recommended in [89] for all investigated problems. For some problems, the time-expensive tuning process is not feasible, and for the other problems, we found that the results were very similar, so we list the results for the standard parameter settings in all chapters. NuPIC outputs an anomaly likelihood in the interval [0,1] for each time series point, which is suitably thresholded to control the algorithm's sensitivity.

3.1.3 LSTM-ED

LSTM-ED [108] attempts to reconstruct sub-sequences of time series in order to detect anomalies. It addresses the problem that many time series are not predictable (in a sense that it is impossible to accurately forecast the time series's next values). Internally, it uses an encoder-decoder LSTM [63, 54, 62] network, which takes sub-sequences from the time series and encodes them into vectors of fixed length and, subsequently, attempts to reconstruct the whole input sequence from the encoded vector. For each reconstructed point, a reconstruction error is computed. For the reconstruction errors, a Gaussian distribution is estimated, and the probability density of each point is used as anomaly score. The algorithm's main parameters are batch size B, the number of training epochs n_{epochs}, sequence length T_{train}, hidden size $h = 100$ and $\%_{Gaussian}$, which specifies the fraction of the data

[2] `https://github.com/numenta/nupic`

[3] The result files can be obtained from `https://github.com/numenta/NAB/tree/master/results/numenta`.

used to estimate a Gaussian distribution for the anomaly detection task. Both encoder and decoder use a stacked LSTM network with two layers.

3.1.4 DNN-AE

DNN-AE [46] is similar to LSTM-ED in that it takes short sequences from a time series and attempts to encode and reconstruct these. However, it is a conventional deep autoencoder architecture improving an earlier approach based on replicator neural networks [58]. Contrary to the replicator neural network approach, DNN-AE uses a significantly deeper architecture, trains its weights with the more powerful ADAM optimizer [82], and replaces the step-wise (staircase) activation function[4] at the bottle-neck layer with $\tanh(\cdot)$. The algorithm requires several parameters similar to LSTM-ED: batch size B, number of training epochs n_{epochs}, sequence length T_{train} and a hidden size of h for the bottle neck (which results in a compression factor of T_{train}/h for each sequence) and $\%_{Gaussian}$ (same as for LSTM-ED). The number of densely connected layers L depends on T_{train} and h: The number of units in all hidden layers (except for the bottle neck) are powers of two which are smaller than T_{train} and h. For example, for $T_{\text{train}} = 50$ and $h = 10$ we have $L = 6$ layers and the number of units for the individual layers is $(32, 16, 10, 16, 32, 50)$.

3.2 Other Anomaly Detection Algorithms

In recent years much effort was put into the design of time series anomaly detection algorithms, and researchers proposed many new methods. In this section, we give an overview of different approaches. Due to the great variety in this field, this overview is not exhaustive.

3.2.1 Online Algorithms

Next to the already introduced algorithms ADVec [173] and NuPIC [160, 49] many very different approaches exist for online anomaly detection in time series, for example: Yao et al. [181] present a simple method, called Segmented Sequence Analysis (SSA), which uses the similarity of piecewise linear representations of univariate time series and a reference model to detect anomalous behavior in sensor data. Ma & Perkins derived an online support vector regression (SVR) training algorithm [104] which they use to predict time series and detect anomalies (novelties) based on the prediction error [102]. In [175], Wang et al. introduce an online algorithm based on two statistical approaches, the Tuckey method, and the relative entropy statistic. Their algorithm is designed to detect anomalies in large-scale cloud services. Ahmed et al. [4] use the kernel recursive least squares (KRLS) algorithm to maintain a relatively small dictionary of feature vectors forming a cluster in a high-dimensional feature space that approximately describes the normal behavior of traffic measurements in a

[4]We also experimented with the step-wise activation function introduced in [58] but could not observe any improvements over using $\tanh(\cdot)$.

network. The maintained dictionary can adapt itself over time and detects anomalous network traffic by computing the distance of new data points to the normal cluster. Talagala et al. [157] present an online algorithm for univariate time series based on extreme value theory and a kernel density estimation. In [158], an adaptable approach for streaming data based on so-called half-space (HS) trees is proposed. Wei et al. developed an online approach based on symbolic time series representations (Symbolic Aggregate approXimation, SAX [94]) which can – according to its authors – be applied to a wide range of applications without domain-specific customization.

We did not find many online-adaptable DL algorithms for time series anomaly detection in the literature: Saurav et al. [146] introduce an online DL model for (multivariate) time series. A stacked GRU (gated recurrent unit) network is used for multistep ahead prediction, and the mean of the prediction errors is used as anomaly score and is compared to a threshold in order to detect anomalies. The learning algorithm is based on a stochastic gradient descent (SGD) approach with an anomaly-score-dependent learning rate, which can perform the backpropagation through time (BPTT) in a fully online fashion.

3.2.2 Deep Learning Approaches

Although some DL algorithms exist, which can be used in an online-adaptable fashion [146], we found that most DL anomaly detection algorithms for time series are trained offline. Due to the temporal nature of the data, common building blocks for DL algorithms are convolutional neural networks or recurrent neural networks (RNNs) such as the long short-term memory (LSTM) [63] or gated recurrent units [29], which are generally applicable to multivariate time series. In some cases also regular fully-connected neural networks are used. Most algorithms that we found in the literature can be classified either as prediction-based or reconstruction-based or are combinations of both.

3.2.2.1 Prediction-based DL Algorithms

The most common approach is to train a DL model to perform a multistep-ahead prediction (forecasting) and compare the predictions to the observed values. Typically, the prediction errors are used as an indicator for anomalous behavior. Although these approaches are trained in a (self-) supervised fashion to predict the time series, they are still considered unsupervised as long as the actual anomaly labels are not used. Malhotra et al. [107] describe such an anomaly detection algorithm based on LSTMs. It predicts over several horizons and estimates a multivariate Gaussian distribution for the prediction errors, and uses the probability density function of the estimated Gaussian as anomaly score. [28, 45, 44, 67, 18, 51] are similar to and partially based on [107], but apply their algorithms to different problems. Partially, also the anomaly thresholds are computed in different ways, for example, simply with the mean squared error (MSE) [45, 44], or using exponentially-weighted error averages [67]. However, most of these models ([107, 28, 45, 44, 67, 18]) are only trained with anomaly-free data and partially also require that the time series data is

manually split into training, validation, and test sets (which requires labeled data or expert knowledge). In chapter 6 (based on [161]), we present a prediction-based LSTM model which can be trained with contaminated data (time series with anomalies) and extends [107] in several aspects.

Instead of LSTMs or GRUs, in [61], He & Zhao use a temporal convolutional network (TCN) [13] for predicting time series, while Munir et al. [115] use regular CNNs. Zhu & Laptev [187] present an algorithm that is a prediction-based approach but uses a reconstruction-based approach (described in the following section) for pre-training: during the pre-training phase, an LSTM encoder-decoder network is trained to extract useful embeddings from a sequence. Subsequently, the decoder is discarded. The encoder network generates features for the prediction network (a multi-layer densely connected network), predicting the next few time-series steps. Additionally, the authors describe an approach to assess the prediction uncertainty, which can help to detect anomalies.

3.2.2.2 Reconstruction-based Algorithms

Other popular approaches for time series anomaly detection are based on compressing and reconstructing time series (or segments thereof) and detecting anomalies using the reconstruction error (or reconstruction probabilities). Reconstruction-based algorithms can be applied when a time series contains normal patterns which are inherently unpredictable. Generally, they are also applicable to predictable time series. The previously introduced DNN-AE ([58], Section 3.1.4) and LSTM-ED ([108], Section 3.1.3) are examples for such approaches. Similarly to [58], other early approaches apply regular (deep) autoencoders (AE) to windows of fixed length and slide the AE over the whole time series. Although relatively simple, these approaches demonstrate their effectiveness in various applications [144, 58, 35]. Oh & Yun [123] present an autoencoder architecture based on CNNs, which is trained to encode and reconstruct segments of a spectrogram. If the reconstruction SSE (sum of squared errors) of a segment lies above a predefined threshold, the corresponding segment is flagged as anomalous. Kieu et al. [80] use an autoencoder architecture based on 2D CNNs and another LSTM encoder-decoder approach similar to [108]. Additionally, they enrich time series with additional features before passing them to the autoencoder. In [79], Kieu et al. present an autoencoder approach based on an ensemble of sparse recurrent neural networks. Zong et al. present with DAGMM an architecture [188] where the parameters of a deep autoencoder and a Gaussian mixture model are simultaneously learned during training. Zhang et al. [184] construct so-called signature matrices of time series segments, capturing the correlations between different dimensions of the time series, and subsequently use an attention-based convolutional LSTM (convLSTM) autoencoder to encode and decode the signature matrices. The residual signature matrices are used as an indicator for anomalous behavior.

Variational approaches Variational autoencoders [83] have a wide range of applications. In recent years, they are also becoming increasingly popular in the field of (time

series) anomaly detection. Generally, variational anomaly detection approaches attempt to compute reconstruction probabilities instead of reconstruction errors [9]. Pereira & Silveira [133] propose a variational Bi-LSTM autoencoder with variational self-attention (VSAM), which takes short segments as input. But instead of reconstructing the original points of the input sequence, the decoder attempts to compute the reconstruction probability of each point by estimating the parameters of a Laplace distribution (again, for each point of the sequence). Given the parameters of the Laplace distribution, the probability (density) of the corresponding input point can be used as anomaly score (here, lower values indicate a larger degree of abnormality). The authors apply their VSAM algorithm to univariate solar photovoltaic generation time series. In later work, they apply similar approaches to ECG data [132, 134]. The Donut algorithm [180] uses sliding windows and applies a variational autoencoder to windows of fixed length. The algorithm is studied on a dataset from a large internet company and outperforms several baseline algorithms. The dataset is not publicly available. Su et al. propose the OmniAnomaly algorithm [153], which is based on stochastic recurrent neural networks and is designed to encode and reconstruct short sub-sequences of a multivariate time series. Similar to most other approaches, the authors use the reconstruction probability as anomaly score. Additionally, they present an approach for an automatic threshold selection. Other similar variational anomaly detection algorithms were introduced by Park et al. [129] (LSTM variational autoencoder) and Su et al. [154] (echo state conditional variational autoencoder).

GAN-based approaches In recent years, the generative adversarial network (GAN), invented by Goodfellow et al. [53], is being applied to anomaly detection tasks [39]. For time series anomaly detection, often encoder-decoder architectures are used for the generator network [186, 71]. In other approaches, the original sequence is reconstructed from an incrementally updated latent space representation (the gradients of an error function are used to improve an initially random latent space vector iteratively) [91, 92]. Most GAN-based approaches compute the anomaly score for a given sample as a weighted average of a reconstruction loss and a discriminator loss [91, 92, 71]. In [186], the authors only consider the reconstruction loss for the anomaly score. Remarkably, all GAN-based approaches that we studied [186, 71, 91, 92] require that the training data is exclusively normal and does not contain any anomalous sequences.

3.2.2.3 Other relevant Algorithms

The HOT SAX algorithm [77, 78] by Keogh et al. is based on Symbolic Aggregate ApproXimation (SAX) [94, 95] and is designed to detect time series discords, i.e., anomalous subsequences in a longer time series. Other worth-mentioning approaches are based on one-class classification (e.g., using support vector machines [103] or using neural networks [24, 149]), artifical immune-system approaches [33, 34], or energy based models [183]. Notable examples from industry – next to ADVec [173] and NuPIC [160] – are Yahoo's EGADS [88], LinkedIn's luminol [97] and Expedia.com's adaptive alerting [43].

3.3 Benchmarks

As described before in Section 2.3, we use Yahoo's Webscope S5 dataset, the Numenta Anomaly Benchmark (NAB) [89], the Mackey-Glass Anomaly Benchmark [167, 166] and the MIT-BIH Arrhythmia database [52, 112, 113] in our work for benchmarking and comparison purpoes.

Apart from these four datasets, other benchmarks can be found in the literature: Filonov et al. generated the synthetic Gasoil Heating Loop (GHL) dataset [45], using a model of a real gasoil plant to simulate hacker attacks. During the attacks, the simulated intruders adjust the setpoints of various process variables. In order to prevent a fault of the system, an anomaly detection algorithm has to detect the unauthorized changes. In total, the benchmark contains 49 19-dimensional time series. One time series contains only normal behavior and is used as the training data. However, many anomalies can be detected by applying a simple threshold to one of the time series' dimensions. A similar benchmark [44] uses the Tennessee Eastman Process (TEP) with simulated cyberattacks to generate industrial multivariate (59-dimensional) time series. In [20], a benchmark containing 276 univariate time series is introduced. The time series are taken from real-world sensor data (such as temperature or light). However, the authors only inserted artificial anomalies. Each time series has a length between 2000 and 18 000 data points and contains 5 to 23 anomalies of various types (random, malfunction, bias, drift, polynomial drift, and combinations). Additionally, a Java program is available to generate new anomalous time series. However, we did not use this benchmark for our work as the anomalies were artificial and partially appeared to be too trivial. Other popular time series anomaly benchmarks are the Soil Moisture Active Passive satellite (SMAP) and Mars Science Laboratory rover (MSL) data by NASA [67] and the Server Machine Dataset (SMD) [153]. A more recent dataset is Skoltech's anomaly benchmark (SKAB) [75], which currently contains more than 30 multivariate time series. Each time series contains about 1 200 points. The data was collected in a testbed consisting of a water circulation system with eight different sensors. In [18], Bontemps et al. describe how they converted the KDD Cup 1999 dataset [76], a network instrusion detection benchmark (initially designed for point-based anomaly detectors), into a time series version.

In our work, we mostly use the standard metrics precision, recall, and F_1-score to compare the performance of algorithms. Additionally, we use precision-recall curves to compare algorithms over a large range of anomaly thresholds. More advanced – but less commonly used – evaluation metrics for time series are introduced in [159] and [89].

Chapter 4

SORAD: A Simple Online Regression Anomaly Detection Algorithm

4.1 Introduction

A challenging aspect of time series anomaly detection is working in environments with non-stationary behavior or unknown or vaguely defined nature of future anomalies. Most of the current anomaly detection algorithms follow the general idea to classify an anomaly as a significant deviation from the prediction. This chapter presents a comparative study where several online anomaly detection algorithms are compared on the Yahoo Webscope S5 anomaly benchmark. We show that a relatively *S*imple *O*nline *R*egression *A*nomaly *D*etector (SORAD) is quite successful compared to other anomaly detection algorithms. We discuss the importance of several adaptive and online elements of the algorithm and their influence on the overall anomaly detection accuracy.

In many applications, data is collected continuously sequentially in the form of data streams or time series. A common characteristic of data streams is the inherent non-stationarity. In a vast amount of real-world problems, data is generated by non-stationary processes. Typically, certain properties of streams or time series (such as trends, noise, periodicity, and other parameters) change over time.

Machine learning tasks involving such non-stationary streaming data are considered challenging since many classical learning approaches are not applicable due to their offline character: offline learning algorithms are trained on a whole batch of data. They require a complete repetition of the training procedure if new examples are added to the data set. Due to memory and time constraints, such approaches are typically infeasible for problems with streaming data. In such cases, it is necessary to operate in an online setting on the data and process the data in an example-by-example manner and incrementally learn from every new example without re-training an entirely new model each time.

Especially in non-stationary environments, an anomaly is difficult to define. In its most general form, it is the absence of normality, but „normality“ depends mainly on the (current) context and cannot be expressed in *one* standard formula. This is the reason why anomaly detection algorithms are very difficult to benchmark. An anomaly detection algorithm that performs very well on a particular benchmark dataset might perform surprisingly poorly on

another benchmark. As an example, we will consider in this chapter Yahoo's well-known Webscope S5 dataset [87] with labeled anomalies of various kinds.

This chapter's purpose is twofold: (1) We show that benchmarking anomaly detection algorithms is difficult with currently well-established benchmark datasets. As an example, we found that the well-known algorithm NuPIC (based on Hierarchical Temporal Memory - HTM), which shows remarkable successes on other benchmarks (e.g., NAB [89]), performs not so well on Yahoo's S5 dataset. From this point, we will argue that there is a need for benchmark datasets better capturing the variety of anomalies in time series data. (2) Numerous applications call for fast yet reliable anomaly detection algorithms. We present here, as a preliminary study, a relatively simple one, the **S**imple **O**nline **R**egression **A**nomaly **D**etector (SORAD), which nevertheless shows good performance on the Yahoo Webscope S5 dataset. It is preliminary because SORAD needs to be refined, extended, and tested on more diverse anomaly data benchmarks.

In this chapter, we address the following research questions:

1. How does a simple online regression algorithm perform on the Webscope S5 dataset?
2. How do other algorithms perform on the same benchmark? In particular, we compare our new algorithm's performance to Numenta's NuPIC algorithm[160, 49] and Twitter's ADVec algorithm [173].
3. How important is the online capability?

4.1.1 Related Work

In this chapter, we compare with two developments in online anomaly detection where the frameworks are available as open-source: (a) Hierarchical Temporal Memory (HTM) [49, 89], which is available as software NuPIC from Numenta [1], and (b) Twitter's ADVec Algorithm [173], which is available as open-source R package AnomalyDetection from Github [2]. Both algorithms are described in more detail in chapter 3.

The Yahoo S5 anomaly detection benchmark has been investigated in [68, 154, 69, 115]. While [68] uses it for detecting whether a whole time series is anomalous, [154] presents an anomaly detection approach with echo state networks to which we will later compare. The algorithm introduced in [69] gives only results for FP (false positives) and is thus not well comparable. In [115], the CNN-based algorithm DeepAnT is introduced and evaluated on the Yahoo S5 benchmark. The performance of DeepAnT for A1–A4 is later compared to SORAD's.

[1] `http://www.numenta.com`
[2] `http://github.com/twitter/AnomalyDetection`

4.2 Methods

In the following sections (4.2.1 – 4.2.2), we present the methods necessary for the offline variant of SORAD, which we call Offline-RAD. Sections 4.2.3 – 4.2.4 cover the methods necessary to extend this offline variant to the online algorithm SORAD. Having both variants available allows us to measure the effect of online adaptivity precisely.

4.2.1 Feature Generation using Sliding Windows

In order to model temporal relationships in machine learning, a common approach is to employ a so called sliding window of a fixed length ℓ, which creates feature (input) vectors of length ℓ. When applied to a time series or sequence of length N in the form $(y_0, y_1, \ldots, y_{N-1})$, the sliding window creates for each instance y_k a feature vector $\mathbf{x}_{k+1}$ consisting of a bias term and the ℓ previous instances, i. e. $\mathbf{x}_k = (1, y_k, y_{k-1}, \ldots, y_{k-\ell+1})^T$. During the transient phase ($k < \ell$) we pad values with negative indices with y_0. Matrix $\mathbf{X}$ is composed of the transposed inputs $\mathbf{x}_k$, one vector per row. The number of rows is the size K of the training set. Likewise, matrix $\mathbf{X}_{test}$ has $N - K$ rows, starting with vector $\mathbf{x}_K$.

4.2.2 Offline Regression Anomaly Detection (Offline-RAD)

The offline regression algorithm divides each time series into a training phase $t \leq K$ and a test or detection phase $t > K$. The general procedure is described in Algorithm 1. This algorithm requires a matrix inversion for each pass through the training data to build the model vector $\boldsymbol{\theta}$. When the parameters $\boldsymbol{\theta}$ are estimated, the prediction for new examples is computed with $\tilde{y}_k = \boldsymbol{\theta}^T \mathbf{x}_k$. The tuning of the regularization parameter ρ is done as follows: The $K = 500$ training data are divided further: For various values of ρ, the training phase is done on the first 400 training data, and the mean prediction error is measured on the remaining 100 validation data. After choosing the value ρ with the smallest prediction error, the final model is trained on all $K = 500$ training data. We perform two passes through the data in order to make the training data approximately anomaly-free.

4.2.3 Online Estimation of a Distribution's Mean and Variance

The naive approach for online estimation of a distribution's mean and variance from the sum of squares suffers from numeric instability. The Welford algorithm [176] proposes a numerically stable variant. A variant of it is used in this chapter to estimate the parameters of the normal distribution.

As a new element, we introduce a forgetting factor for mean and variance. The forgetting is realized by weighting the elements, e. g., for the sample variance we use

$$s_n^2 = \sum_{i=1}^{n} \frac{w_i (x_i - \mu_n)^2}{\sum_{i=1}^{n} w_i}, \tag{4.1}$$

Algorithm 1 Offline-RAD: Offline anomaly detection algorithm. **Input**: Time series (y_k), anomaly threshold ϵ, training set size $K = 500$. **Output**: Anomaly flags $\mathbf{a}_{flags}$.

Initialize:
Anomaly flags $\mathbf{a}_{flags} \leftarrow 0$ ▷ Binary Vector
Tune regularization parameter ρ (Sec. 4.2.2).

Training phase:
Create training data $\mathbf{X}$ and $\mathbf{y}$ from the K first time steps.
for $i = 1 \dots 2$ **do** ▷ Two passes through training data
    $\boldsymbol{\theta} \leftarrow (\mathbf{X}^T\mathbf{X} + \rho\mathbf{I}_{\ell+1})^{-1}\mathbf{X}^T\mathbf{y}$
    $\boldsymbol{\delta} \leftarrow \mathbf{y} - \mathbf{X}\,\boldsymbol{\theta}$ ▷ Compute train prediction error
    Compute mean μ_δ and standard deviation s_δ for $\boldsymbol{\delta}$
    Calculate the ϵ-quantile z_ϵ of $\mathcal{N}(0, s_\delta^2)$ (see Fig. 4.1)
    $E \leftarrow [\mu_\delta - z_\epsilon,\, \mu_\delta + z_\epsilon]$
    for all $\{k \,|\, \boldsymbol{\delta}_k \notin E\}$ **do**
        Remove k-th row from $\mathbf{X}$ and $\mathbf{y}$

Detection phase:
Create test data $\mathbf{X}_{test}$ and $\mathbf{y}_{test}$
$\boldsymbol{\delta}_{test} \leftarrow \mathbf{y}_{test} - \mathbf{X}_{test}\,\boldsymbol{\theta}$
for all $\{k \,|\, \boldsymbol{\delta}_{test,k} \notin E\}$ **do**
    $a_{flags,k} \leftarrow 1$ ▷ Flag $y_{test,k}$ as anomalous

where x_i is the ith instance in the sample and μ_n is the current sample mean. The weights

$$w_i = \lambda^{n-i} \tag{4.2}$$

decay exponentially so that historic elements contribute less to the sample variance s_n^2. A similar formula can be obtained for the sample mean μ_n. Both formulas can be combined with the modified Welford algorithm to get an online formulation of Eqs. (4.1) and (4.2). This is presented in Algorithm 2.

One can estimate the memory (the number of time steps after an observation is "forgotten" again) with the formula

$$n_{mem} \approx \frac{1+\lambda}{1-\lambda},$$

as shown in Appendix B.2.4. A thorough derivation of the online weighted estimation of mean and (co-) variance can be found in Appendix B.2.

4.2.4 SORAD

The Offline-RAD algorithm has two main disadvantages: (a) It needs a training period (500 time steps in our application) before it can perform predictions, and (b) it does not learn any further in the detection phase. Both aspects call for an online version of this algorithm.

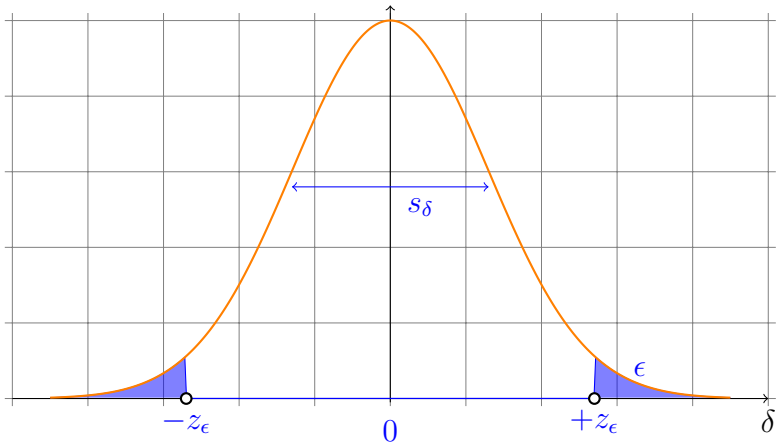

Figure 4.1: The ϵ-quantiles $\pm z_\epsilon$ of $\mathcal{N}(0, s_\delta^2)$ which define the border between anomalous and normal data in Offline-RAD and SORAD.

One of the most popular online models of the past few decades is the recursive least-squares (RLS) algorithm [59] from adaptive filter theory. The standard RLS formulation often includes a forgetting factor $\lambda \in (0, 1]$ that allows to deal with non-stationary systems to some extent [172, 17]. RLS is used in Algorithm 3 to estimate the model vector $\boldsymbol{\theta}$ recursively [17]:

$$\mathbf{P} \leftarrow \frac{1}{\lambda}\mathbf{P} - \frac{1}{\lambda} \cdot \frac{\mathbf{P}\mathbf{x}_k\mathbf{x}_k^T\mathbf{P}}{1 + \mathbf{x}_k^T\mathbf{P}\mathbf{x}_k} \tag{4.3}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta_{k+1}\mathbf{P}\mathbf{x}_k \tag{4.4}$$

with prediction error δ_{k+1} and forgetting factor λ. Each input vector $\mathbf{x}_k$ is used only once. A RLS variant for mini-batch updates (updating the parameters $\boldsymbol{\theta}$ with more than one example) is derived in Appendix B.1 and can also be used in practice.

Having an online estimation of $\mathbf{P}$ makes it possible to introduce a forgetting factor λ which causes the algorithm to slowly fade out the long-ago parts of history and adapt the model to non-stationary elements in the time series. This is not possible in the offline version. Of course, a careful balancing between stability and plasticity of the model is necessary.

Finally, the modified weighted Welford algorithm (Sec. 4.2.3) provides an online estimation for mean and variance of the error δ_{k+1} (Algorithm 3, step 20).

SORAD has a short transient phase for $k < \ell$. In this period, where the input vector $\mathbf{x}_k$ needs to be partially padded (see Sec. 4.2.1), the model vector $\boldsymbol{\theta}$ is left in its initial state. But the changes $\Delta\boldsymbol{\theta}$ according to Eq. (4.4) are accumulated separately and added to $\boldsymbol{\theta}$ at $k = \ell$. Likewise, the standard deviation s_δ is kept at ∞, leading to an inhibition of anomaly detection. The changes Δs_δ are accumulated and applied to s_δ at $k = \ell$. After the transient phase, $\boldsymbol{\theta}$ and s_δ are updated normally (Algorithm 3 and 2).

Algorithm 2 Online estimation of sample mean and sample variance for the prediction errors. A more detailed derivation of the update rules can be found in Appendix B.2.

Initialize:
$\mu_\delta = 0$, $s_\delta^2 = 0$, $M = 0$, $W = 0$

function UPDATEESTIMATION(k, δ_{k+1})
  $W \leftarrow \lambda W + 1$
  $\Delta \leftarrow \delta_{k+1} - \mu_\delta$
  $\mu_\delta \leftarrow \mu_\delta + \frac{\Delta}{W}$
  $M \leftarrow \lambda M + \Delta \cdot (\delta_{k+1} - \mu_\delta)$ ▷ Use new value of μ_δ
  $s_\delta^2 \leftarrow \frac{M}{W}$ ▷ Without bias-correction

4.3 Experimental Setup

4.3.1 Algorithm Setup

All algorithms (except Offline-RAD) operate online on the time series. We evaluate all algorithms on the Yahoo Webscope S5 benchmark (introduced in Section 2.3).

Offline-RAD We use a window size of $\ell = 10$. The first $K = 500$ instances of each time series are used for training, the remaining 1000 instances for testing (detection).

Setup of Numenta's NuPIC (HTM) To verify the correctness of our NuPIC installation, we applied it to the Numenta Anomaly Benchmark (NAB) [89] using the standard

Algorithm 3 Pseudo code of SORAD. **Input:** Time series (y_k), anomaly threshold $\epsilon \in (0, 1)$, forgetting factor $\lambda \in (0, 1]$. Additionally, there is a short transient phase (see Sec. 4.2.4). **Output:** Anomaly flags $\mathbf{a}_{flags}$.

Initialize and Transient Phase
$\boldsymbol{\theta} \leftarrow (\theta_{Bias} \quad 2^{-1} \quad 2^{-2} \quad \cdots \quad 2^{-\ell})^T$, with $\theta_{Bias} = 0$
$(\mu_\delta, s_\delta^2) \leftarrow (0, \infty)$
$\mathbf{P} \leftarrow 500\mathbf{I}_{\ell+1}$, with the identity matrix $\mathbf{I}_{\ell+1}$
Anomaly flags $\mathbf{a}_{flags} \leftarrow 0$ ▷ Binary Vector

Set instance counter $k \leftarrow 0$
while instance y_{k+1} available **do**
    $\mathbf{x}_{k+1} \leftarrow (1, y_k, y_{k-1}, \ldots, y_{k-\ell+1})^T$
    $\tilde{y}_{k+1} \leftarrow \boldsymbol{\theta}^T \mathbf{x}_{k+1}$ ▷ Predict next step
    Observe y_{k+1}
    $\delta_{k+1} \leftarrow y_{k+1} - \tilde{y}_{k+1}$ ▷ Compute prediction error
    Calculate the ϵ-quantile z_ϵ of $\mathcal{N}(0, s_\delta^2)$ (see Fig. 4.1)
    $E \leftarrow [\mu_\delta - z_\epsilon,\ \mu_\delta + z_\epsilon]$
    if $\delta_{k+1} \notin E$ **then**
        $a_{flags,k+1} \leftarrow 1$ ▷ Flag y_{k+1} as anomalous
        $k \leftarrow k + \ell - 1$ ▷ Skip next ℓ instances
    else
        Update $\mathbf{P}$ and $\boldsymbol{\theta}$ according to Eq. (4.3) and (4.4)
        Update μ_δ, s_δ^2 with updateEstimation(k, δ_{k+1})
    $k \leftarrow k + 1$
end while

parameter settings and confirmed that we can exactly reproduce the results[3] published in [89]. In a first round of experiments we used the same parameter setting for the S5 datasets. In a second round we used NuPIC's so called swarming algorithm [3], a tool to aid automatic parameter search for a given dataset. The results were very similar, so we list in the following only the results for the standard parameter settings.

Setup of Twitter's ADVec Algorithm The ADVec algorithm has two parameters. The first parameter α describes the level of statistical significance with which to accept or reject anomalies. Similar to an anomaly threshold, this parameter trades off false-positives and false-negatives. The period-length is set to the value 40 in all experiments[4]. Finally, the parameter $\max_{anoms}$ is left at its default ($\max_{anoms} = 2\%$).

4.4 Results

Table 4.1 summarizes the results of all algorithms running on the four datasets A1–A4. To have a fair comparison to Offline-RAD, we include for all algorithms only time steps $t > 500$ (after the Offline-RAD training phase) into the anomaly detection phase. Apparently, SORAD has a better performance than Offline-RAD, and both are on most datasets significantly better than NuPIC and ADVec in nearly all performance measures.

The great advantage of online algorithms is, of course, that they are much faster up-and-running. We show in Table 4.2 the results when including all time steps $t > \ell$ beyond the transient phase in the anomaly detection. Basically, the results are very similar. This means that the ‚anytime-ready' feature of online-algorithms does not severely influence accuracy.

Fig. 4.2 shows an example time series from the A4 dataset with the anomalies detected by the various algorithms included. It is seen that NuPIC and ADVec produce a number of false-positives (FP).

Anomaly detection algorithms always have a threshold parameter that allows the user to control the trade-off between FP and FN. Fig. 4.5 shows that the performance measure F_1 is very stable over several threshold decades.[5] The multiobjective plot (where the measures FP and FN are plotted against each other) in Fig. 4.3 varies these thresholds and shows the multiobjective front for each algorithm. It is seen that both RAD algorithms dominate the others irrespective of the chosen threshold. SORAD is significantly better than Offline-RAD.[6] This might be due to the fact that the online algorithm continues to learn, which is advantageous for non-stationary environments.

[3]The result files can be obtained from: `https://github.com/numenta/NAB/tree/master/results/numenta`.

[4]We tuned this parameter over a wide range for all four datasets.

[5]Fig. 4.5 shows the stability of F_1 for SORAD (without forgetting). However, it holds the same way for the SORAD-variants with forgetting.

[6]Additionaly, the variation of the threshold produces for SORAD a more diverse population on the multiobjective front with no 'holes' as in Offline-RAD.

We test in Fig. 4.3 the one-pass and the two-pass variant of Offline-RAD, showing that there is no significant difference between both. This means that the poorer performance of Offline-RAD (as compared to SORAD) is *not* due to the occasional presence of anomalies in the training phase.

All of these results were obtained with a forgetting factor $\lambda = 1$, that is, with no forgetting. If a time series is non-stationary, then anomaly detection might benefit from a certain degree of forgetting the past values. We show in Fig. 4.6 the effect of varying the forgetting factor in the range [0.8,1.0]. Additionally, Table 4.2 shows the overall results for two SORAD variants with forgetting: While forgetting in the RLS part (SORAD-F) does not change much, a forgetting mechanism additionally for the online estimation of the error distribution (μ_δ, s_δ) is of great benefit (SORAD-FMS), especially for the datasets A1 and A4.

Table 4.3 shows the computation times for all algorithms.

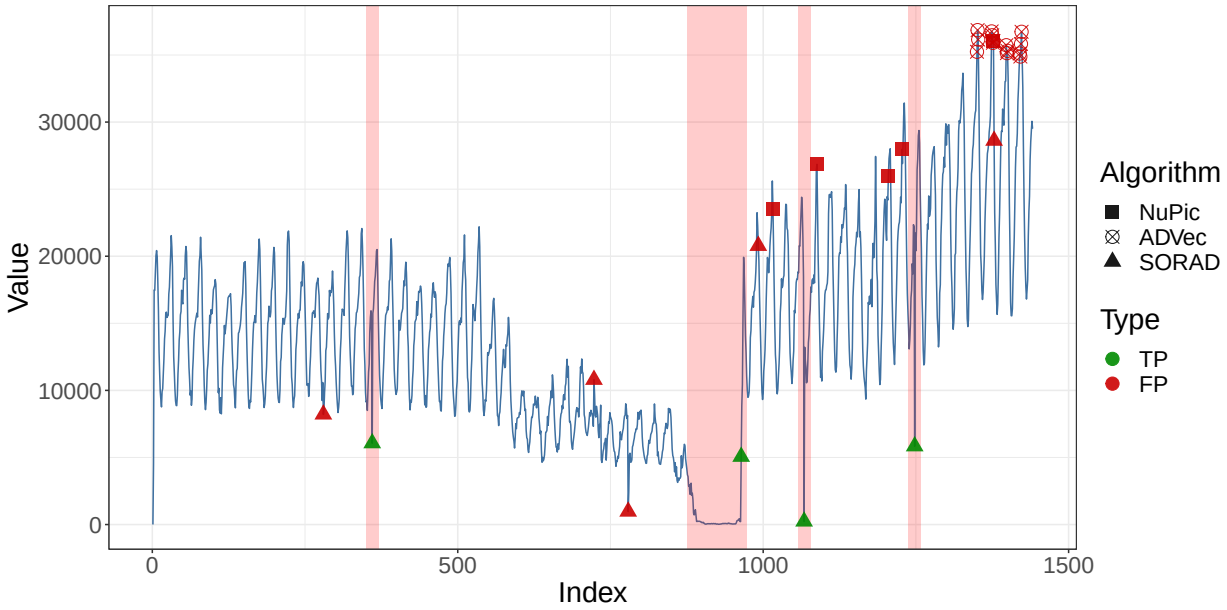

Figure 4.2: Example time series taken from data set A1 with the anomalies detected by the various algorithms SORAD, HTM (NuPIC), and ADVec. The red vertical bars in the plot indicate the true anomaly windows.

4.5 Discussion

4.5.1 Transient Phase

As described in Sec. 4.2.4, algorithm SORAD needs a short transient phase. During this phase, the model vector changes are accumulated, and no anomaly detection is allowed. It

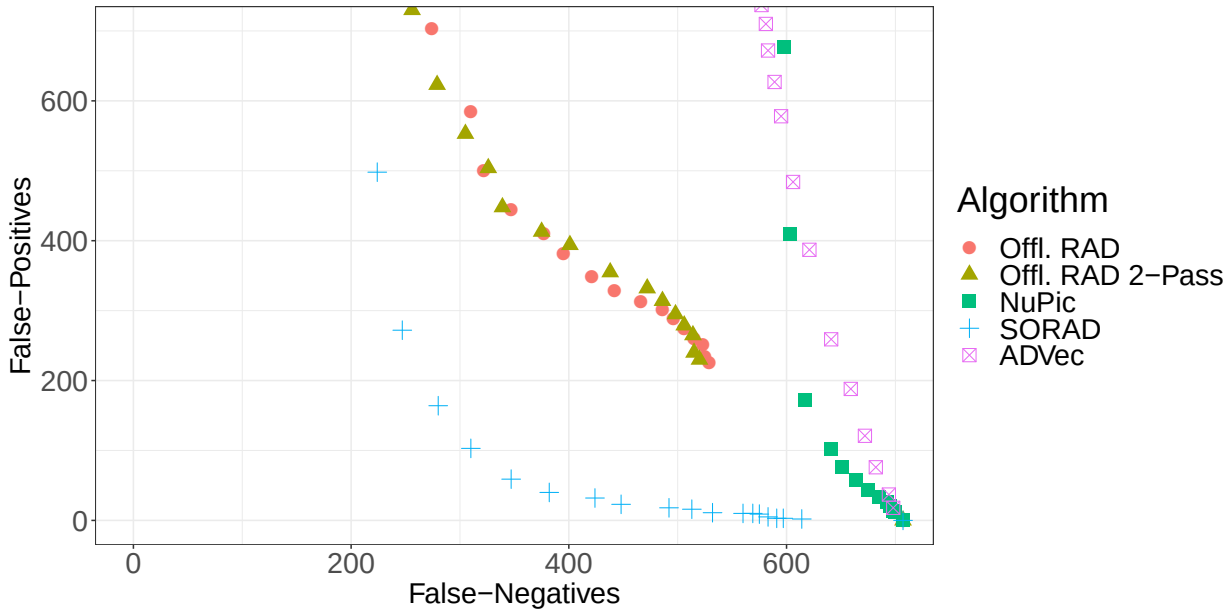

Figure 4.3: Multiobjective plot for different algorithms and thresholds for $t > 500$. Here, the results for the A4 data are shown. The FNs and FPs are the sums over the 100 time series of the A4 data.

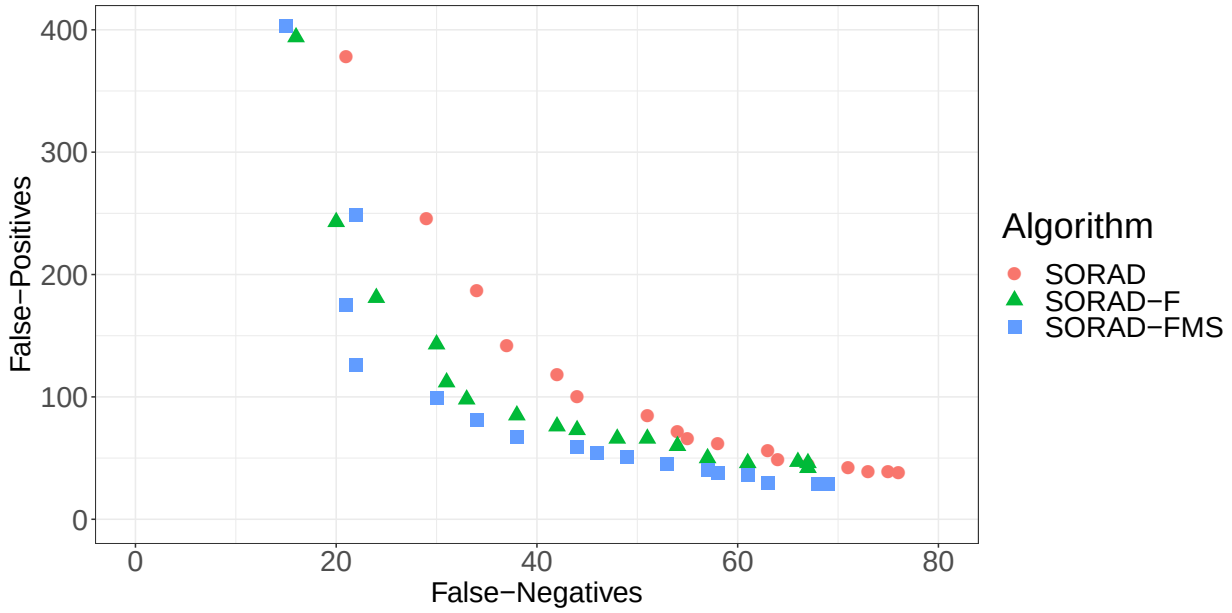

Figure 4.4: Multiobjective plot for different SORAD variants and thresholds for all t after the transient phase. Here, the results for the A1 data are shown. The FNs and FPs are the sums over the 100 time series of the A1 data.

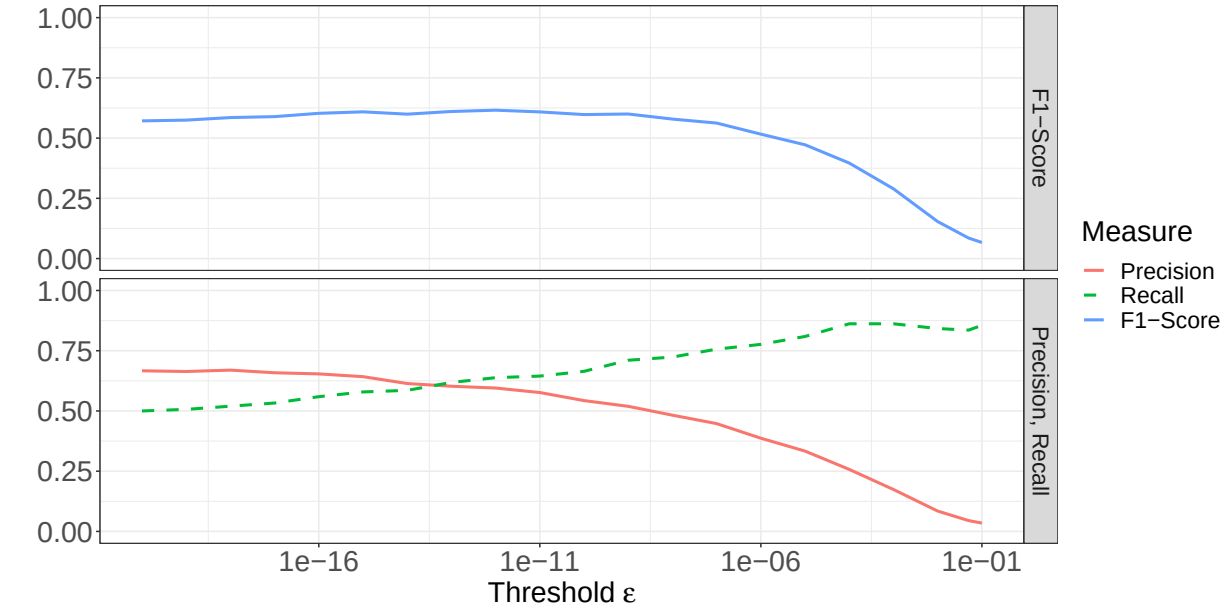

Figure 4.5: The performance of SORAD (without forgetting) over a wide range of thresholds for the A1-data. The F_1 score is virtually constant for a wide range $\epsilon \in [1e-17, 1e-9]$.

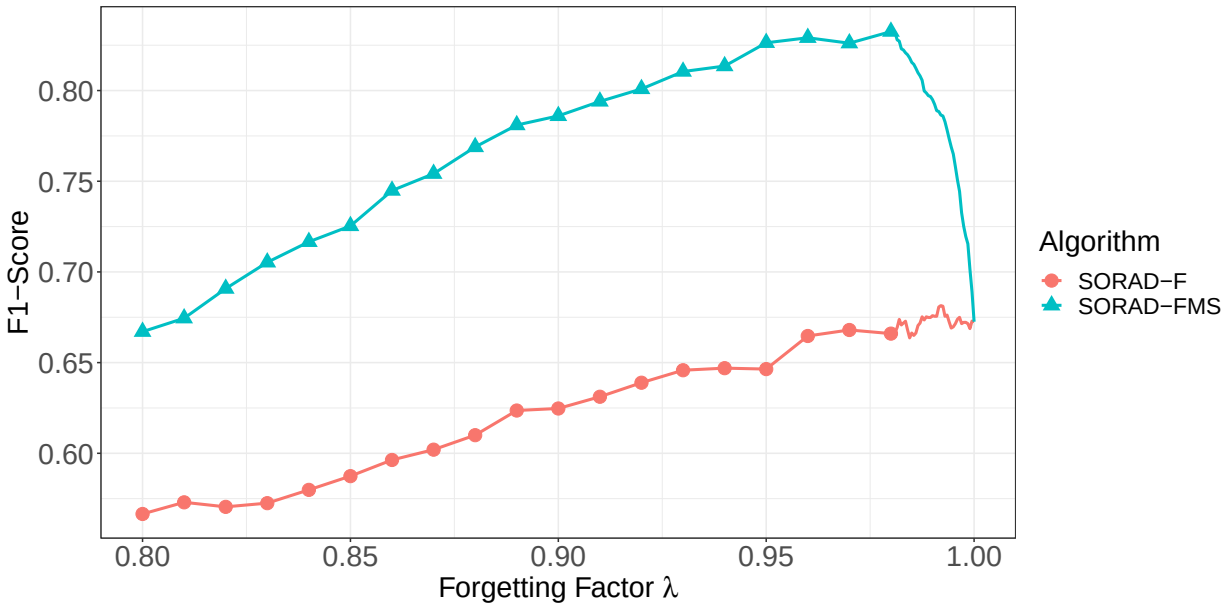

Figure 4.6: Comparing results on A4 for various forgetting factors of SORAD. The first curve SORAD-F shows the results for SORAD with forgetting in RLS; the forgetting factor $\lambda = 0.992$ results in the highest F_1 score of $F_1 = 0.68$ in this case. For the second curve SORAD-FMS, the forgetting is applied to the estimation of μ_δ and s_δ of the error signal distribution as well. The best F_1 score on the A4-data with a value of $F_1 = 0.83$ is reached for a forgetting factor of $\lambda = 0.9805$.

Table 4.1: Results for various algorithms on the Yahoo S5 datasets A1–A4. Shown are TP, FP, FN for each dataset (sum over all time series, $t > 500$) and the quantities precision, recall, and F_1 for each dataset (over all time series, $t > 500$). All algorithms have their threshold chosen such that FP $\approx$ FN. (Only for the F_1 score in brackets, the threshold is chosen such that F_1 is maximized.)

		Dataset			
Algorithm	Measure	A1	A2	A3	A4
Offline RAD	TP, FP, FN	73, 101, 54	197, 3, 3	603, 18, 37	312, 382, 395
	Precision, Recall	0.42, 0.57	0.98, 0.98	0.97, 0.94	0.45, 0.44
	F_1 score	0.49 (0.53)	0.98 (0.99)	0.96 (0.96)	0.45 (0.48)
Offline RAD 2-Pass	TP, FP, FN	73, 107, 54	197, 3, 3	615, 36, 25	306, 394, 401
	Precision, Recall	0.41, 0.57	0.98, 0.98	0.94, 0.96	0.44, 0.43
	F_1 score	0.48 (0.5)	0.98 (0.99)	0.95 (0.96)	0.43 (0.49)
SORAD	TP, FP, FN	85, 43, 42	197, 0, 3	627, 16, 13	460, 272, 247
	Precision, Recall	0.66, 0.67	1, 0.98	0.98, 0.98	0.63, 0.65
	F_1 score	**0.67** (0.67)	**0.99** (0.99)	**0.98** (0.98)	**0.64** (0.66)
NuPIC	TP, FP, FN	67, 57, 60	91, 102, 109	151, 465, 489	109, 677, 598
	Precision, Recall	0.54, 0.53	0.47, 0.46	0.25, 0.24	0.14, 0.15
	F_1 score	0.53 (0.55)	0.46 (0.48)	0.24 (0.26)	0.15 (0.19)
ADVec	TP, FP, FN	60, 62, 67	114, 65, 86	165, 458, 475	112, 578, 595
	Precision, Recall	0.49, 0.47	0.64, 0.57	0.26, 0.26	0.16, 0.16
	F_1 score	0.48 (0.48)	0.6 (0.64)	0.26 (0.29)	0.16 (0.17)

has proven to be adversarial to start directly with the recursive procedure while the sliding window is not entirely filled with true data. Our experiments have shown that in such a case, the estimation of the vector $\boldsymbol{\theta}$ tends to be unstable, and the anomaly error rate increases.

4.5.2 Forgetting Factor

Online algorithms open the possibility to add a certain degree of forgetting. We investigated different forms of forgetting in SORAD. SORAD-F only uses forgetting in RLS. For SORAD-FMS, the forgetting is applied to the estimation of μ_δ and s_δ of the error signal distribution as well. Our results indicate that the usual forgetting element in the RLS part (SORAD-F) does not play a significant role. However, the new element to add forgetting to the estimation of the error distribution (SORAD-FMS) has proven beneficial in datasets A1 and A4. The increase in F_1 score (13%) is most prominent for dataset A4, which is the dataset with the most significant non-stationary elements. Note that all datasets in the Yahoo S5 benchmark are relatively short (1500 time steps), thus putting a boundary on the

Table 4.2: Same as Table 4.1, but now the detection phase is larger (all time steps after the transient phase), and only the online algorithms (SORAD in different variants, NuPIC, ADVec) are compared. Similar accuracy as in Table 4.1, but the online algorithms are faster up-and-running. For SORAD-F and SORAD-FMS the forgetting factor is fixed to $\lambda = 0.98$ for all experiments.

		Dataset			
Algorithm	Measure	A1	A2	A3	A4
SORAD	TP, FP, FN	94, 62, 58	197, 0, 3	877, 24, 28	648, 301, 331
	Precision, Recall	0.6, 0.62	1, 0.98	0.97, 0.97	0.68, 0.66
	F_1 score	0.61 (0.62)	**0.99** (0.99)	**0.97** (0.97)	0.67 (0.67)
SORAD-F	TP, FP, FN	95, 59, 57	197, 1, 3	888, 51, 17	672, 331, 307
	Precision, Recall	0.62, 0.62	0.99, 0.98	0.95, 0.98	0.67, 0.69
	F_1 score	0.62 (0.63)	**0.99** (0.99)	0.96 (0.96)	0.68 (0.68)
SORAD-FMS	TP, FP, FN	98, 52, 54	197, 1, 3	855, 24, 50	796, 289, 183
	Precision, Recall	0.65, 0.64	0.99, 0.98	0.97, 0.94	0.73, 0.81
	F_1 score	**0.65** (0.66)	**0.99** (0.99)	0.96 (0.96)	**0.77** (0.83)
NuPIC	TP, FP, FN	69, 110, 83	91, 102, 109	177, 816, 728	129, 1034, 850
	Precision, Recall	0.39, 0.45	0.47, 0.46	0.18, 0.2	0.11, 0.13
	F_1 score	0.42 (0.5)	0.46 (0.48)	0.19 (0.2)	0.12 (0.15)
ADVec	TP, FP, FN	81, 171, 71	114, 93, 86	241, 668, 664	147, 844, 832
	Precision, Recall	0.32, 0.53	0.55, 0.57	0.27, 0.27	0.15, 0.15
	F_1 score	0.4 (0.4)	0.56 (0.59)	0.27 (0.3)	0.15 (0.16)

Table 4.3: Computation times of the algorithms on datasets A1–A4. Shown are the average and standard deviation from 20 runs each. The runs were performed on a PC with an i7-3520M CPU and 8GB of RAM.

	Computation Time (s)			
Algorithm	A1	A2	A3	A4
Offline RAD	7.7 ± 0.1	11.6 ± 0.1	12.9 ± 0.1	12.7 ± 0.1
SORAD-FMS	21.1 ± 0.1	31.8 ± 0.1	35.8 ± 0.1	36.3 ± 0.1
NuPIC	368 ± 5	693 ± 2	813 ± 3	828 ± 4
ADVec	3.3 ± 0.1	4.8 ± 0.2	5.6 ± 0.4	6.0 ± 0.7

degree of non-stationarity one can observe. For longer time series, the effect of having or not having a forgetting factor can be much larger.

4.5.3 Detection Accuracy

This comparative study's most striking result is that our relatively simple regression anomaly detection works better on all datasets A1–A4 than NuPIC or ADVec. It has to be said that NuPIC has a large number of parameters, and we cannot exclude with certainty the possibility that there might be another parameter set leading to better results for NuPIC. But we can say that it must be hard to find since even the parameter optimization procedure built into NuPIC (swarming algorithm) did not reveal such a parameter set.

4.5.4 Other algorithms

Suh et al. [154] propose an echo state network approach to anomaly detection and test it on Webscope S5, but only for the A1 dataset. Besides methodological issues (they devote 44 complete time series to training, 11 to validation, omit the 45th time series, and perform their evaluation only on the remaining 11 time series, without cross-validation), their results for precision / recall / $F_1 = 0.54$ / 0.51 / 0.52 are inferior to the results of SORAD. In [115], the F_1 scores of an CNN-based algorithm (DeepAnT) on A1–A4 are reported: The obtained F_1-scores for A1, A2, A3 and A4 are 0.46, 0.94, 0.87, and 0.68, respectively, which are all slightly lower than the reported values for SORAD-FMS in Table 4.2.

4.5.5 Limitations of SORAD

Initially, we intended SORAD to act as a baseline algorithm (to show how far simple algorithms get on a particular benchmark and how much further advanced algorithms would lead). It was a surprise for us that SORAD performed better on A1–A4. We are led to the conclusion that anomaly detection benchmarks with a larger variety of anomalies are essential for proper benchmarking.

We do *not* claim that SORAD is for all time series the better anomaly detector. It has not enough memory for more complicated long-term interactions. On the Numenta anomaly detection benchmark NAB [89], where NuPIC achieves good results (F_1=0.51), SORAD does not perform too well in its current form (F1=0.24). Further research effort is needed here.

4.6 Conclusion

In concluding this chapter, we refer to our research questions from 4.1 by providing the following summarizing answers:

(1) A simple online regression anomaly detector (SORAD) performs surprisingly well on the Webscope S5 anomaly benchmark dataset.

(2) It outperforms other anomaly detection algorithms (NuPIC, ADVec) on these datasets. This is at least true if those algorithms are used with their standard parameter

settings. A search for better parameter settings for NuPIC using its parameter optimization routine did not reveal significantly different results.

(3) Our third research question on the importance of the online capability is answered as follows: We compared algorithm SORAD with its offline sibling (Offline-RAD) and could thus assess in this comparative study the differences between both: The online variant is superior to the offline variant (around 40% increase in F_1 score on datasets A1 and A4). We showed that it is crucial to make *all* elements of the algorithm adaptive, including the parameters of the error distribution. This underpins the importance of building *online and adaptive* anomaly detection algorithms to cope successfully with today's large data streams.

4.6.1 Possible Future Work

It is necessary to build up more diverse anomaly detection benchmarks and to collect comprehensive results of algorithms working on them. As said before, our algorithm SORAD is not yet good on all anomaly benchmarks. For example, it does not work too well on the Numenta Anomaly Benchmark (NAB). Simple extensions of the algorithm, such as multi-step ahead prediction (predict several horizons instead of just one), additional non-linear feature transformations, replacing the RLS algorithm with an kernel-based RLS approach [41], and the usage of other online approaches for modeling the prediction errors (instead of a Gaussian distribution) could likely improve the performance of SORAD.

In the following chapter, we will introduce the DWT-MLEAD algorithm, which – contrary to SORAD – can analyze time series at different time scales and thus, performs better on the time series (with longer-range anomalies) of the Numenta Anomaly Benchmark.

Chapter 5

An Anomaly Detection Algorithm based on Discrete Wavelet Transforms

5.1 Introduction

In the previous chapter, we introduced the SORAD algorithm which demonstrated its effectiveness on the Yahoo Webscope S5 benchmark, outperforming other state-of-the-art algorithms. However, the benchmark time series mostly contained only short-term anomalies. Initial experiments on the Numenta Anomaly Benchmark (NAB) revealed that SORAD has difficulties with time series with longer-term patterns and anomalies. A common problem in practice is that anomalies can appear on quite different time scales: they can be spikes (short-time events) or broader structures (mid- or long-term irregularities). Many anomaly detection algorithms available today have their strength either in shorter or in longer time scales, but not in both. However, it is from great practical relevance to have algorithms which work robustly on diverse time-series data. Thus, the underlying research question for this chapter is: Is it possible to propose an online-adaptable anomaly detection algorithm which works robustly on a very *diverse* set of benchmarks?

We believe that wavelet-based methods could be a suitable approach to answer this question. Wavelet transforms [111] are commonly used to decompose a time series signal into accurate time-localized frequency information. This makes them ideally suited to detect anomalies on different time scales where the time scale is a priori unknown.[1]

In the context of the above research question, a new algorithm for unsupervised anomaly detection in time series, called DWT-MLEAD, is introduced in this Chapter. The approach is based on the discrete wavelet transform (DWT) of time series and the identification of abnormal behavior on individual frequency scales using a Mahalanobis-distance-based method. Wavelets allow feature extraction on different frequency levels in different representations and allow a timely precise localization of anomalies in time series. Initially, we

[1]We note in passing that the visual or auditory system of higher vertebrates contains information-processing structures similar to wavelets [72], thus underpinning the importance of wavelets for *natural computing*.

experimented with an offline version of our algorithm and found that it performs well on a set of (mostly) stationary time series. On a diverse set of 158 time series (taken from NAB and the Yahoo Webscope S5 benchmark), the algorithm is compared with three other state-of-the-art anomaly detectors and it is shown to outperform the other approaches. Thanks to the linear time complexity of the DWT, the algorithm is also computationally efficient.

Later in this chapter, we extend DWT-MLEAD to operate fully online. Given streaming data or time series, the algorithm iteratively computes the (causal and decimating) discrete wavelet transform. For individual frequency scales of the current DWT, the algorithm now detects unusual patterns across frequency scales by computing the Mahalanobis distance in an online fashion. The online DWT-MLEAD algorithm is tested on all 425 time series from the Yahoo Webscope S5 benchmark and NAB. A comparison to the other state-of-the-art online anomaly detectors shows that our algorithm can mostly produce results similar to the best algorithm on each dataset. It produces the highest average F1-score with one standard parameter setting. That is, it works more stable on high- and low-frequency-anomalies than all other algorithms. We believe that the wavelet transform is an important ingredient to achieve this.

This chapter is based on our three earlier publications [162, 164, 165].

5.1.1 Related Work

Although wavelet transforms are widely used in many fields of signal and image processing and in many data mining tasks such as time series classification, clustering, prediction & forecasting and similarity search [27], their potential for time series anomaly detection is only partially exploited. From those techniques found in the literature, most are designed for high-frequency anomaly detection (e.g. in network traffic data), such as [81, 85] and [100]. The early work of [6, 5] describes anomaly detection based on non-decimating wavelet transforms. [74] developed an anomaly detection algorithm for time series, based on wavelets, neural networks and Hilbert transforms. The algorithm was tested on a relatively simple benchmark, including two synthetic time series. Shahabi et al. [148] developed an anomaly detection approach which is based on the visual inspection of the DWT of time series data. In this chapter, we test our algorithm on two large anomaly benchmarks, one being the well known Numenta Anomaly Benchmark (NAB, 58 time series, most of them real-world) [89] and the other being a subset of the Yahoo's S5 Webscope benchmark [87]. We compare our algorithm with the already introduced state-of-the-art anomaly detectors: Numenta's NuPIC, based on Hierarchical Temporal Memory (HTM) [49], our previous algorithm SORAD (Chapter 4 & [163]) which is specialized on short-term anomalies, and Twitter's ADVec algorithm [173].

In the following Section 5.2, we describe the unsupervised algorithm which uses **D**iscrete **W**avelet **T**ransform and **M**aximimum **L**ikelihood **E**stimation for **A**nomaly **D**etection in

time series. Note, however, that the name might be slightly misleading. Initially, we assumed that the data-generating distributions are Gaussian. Hence, we used maximum likelihood estimation (MLE) to estimate the parameters of a Gaussian distribution (sample mean and covariance) which are required to compute the Mahalanobis distance. However, we noticed later that the assumption of normality is not generally necessary since the Mahalanobis distance does not impose any conditions on the distribution of the data.
For each time series the DWT-MLEAD algorithm (i) computes a *decimating* DWT using Haar wavelets, (ii) estimates a mean and covariance matrix for the individual frequency scales of the DWT, (iii) flags unusual data points in each frequency scale based on a (Mahalanobis-distance-based) quantile estimate, and finally (iv) aggregates the unusual data points of all frequency scales and detects anomalies in the original time series. We perform a few initial experiments with the offline DWT-MLEAD algorithm and shortly discuss the results.

In Section 5.3, we describe the necessary modifications in order to make DWT-MLEAD fully online. In order not to violate causality-constraints, the resulting algorithm is partially quite different from its offline counterpart. We compare online DWT-MLEAD on all time series taken from the Yahoo S5 data and NAB. Section 5.2.5 concludes this chapter and gives an outlook on possible future work.

5.2 The Offline DWT-MLEAD Algorithm

5.2.1 Methods

5.2.1.1 Wavelet Transforms

Wavelet transforms [111] allow to represent a time series signal in terms of waves (the so called wavelets) with little local support. While (short-time) Fourier transforms always have a trade-off between accuracy in the frequency domain and accuracy in the time domain, wavelet transforms are used to retrieve accurate time-localized frequency information. The wavelet transform of a time series signal is composed with scaling and shifting functions. They take a mother wavelet and stretch and shrink it (scaling), dilate it along the time axis (shifting), and finally form the scalar product with the time series. For sampled time series data, often the so called discrete wavelet transform (DWT) is applied, which has linear time complexity and can be implemented efficiently for time series streams using finite impulse response (FIR) filters. Usually a decimating DWT is performed, in which the filtered series are downsampled. The DWT decomposes the original time series into so called approximation and detail coefficients which are arranged in different levels. Due to the decimating (downsampling) property of the DWT one can represent both coefficient sets in two binary tree structures.

In its current form, DWT-MLEAD performs a decimating DWT using Haar wavelets (other wavelets are also applicable, but require some additional considerations) on each time

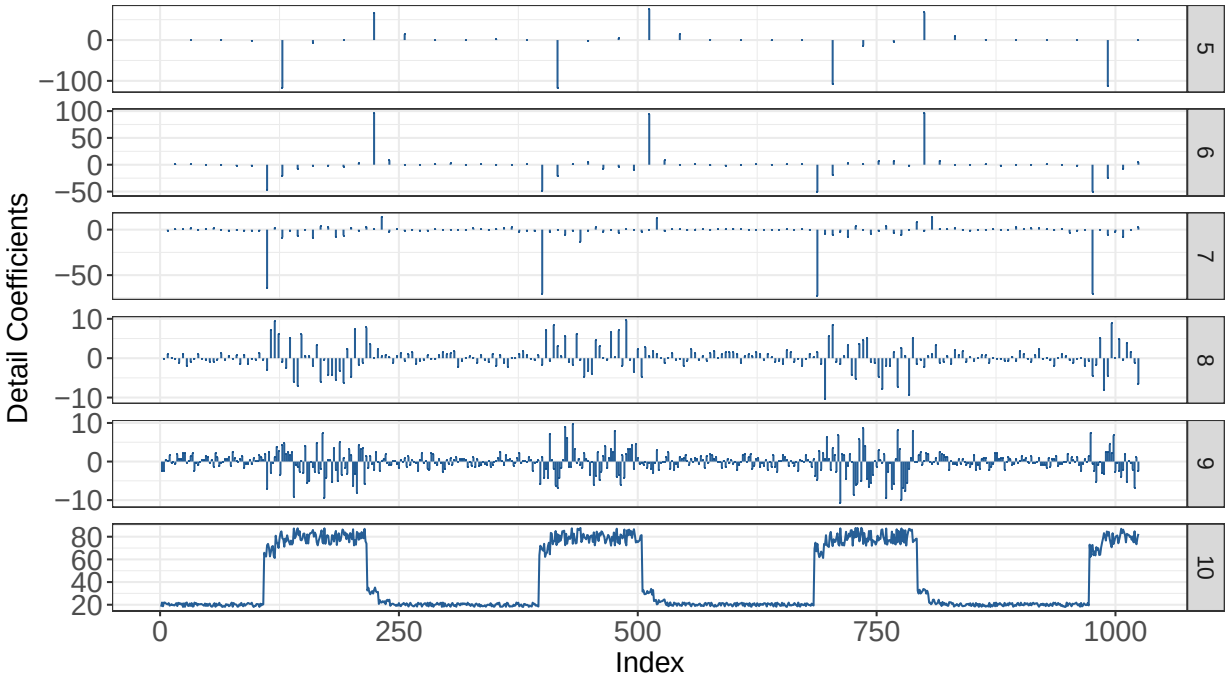

Figure 5.1: Example of a decimating DWT using Haar Wavelets for a time series of the NAB data. The original time series is depicted on scale 10. On the scales 5–9 the detail coefficients of the DWT are shown. While we move towards lower scales, the number of coefficients is decimated (i.e. halved in each step), with 32 coefficients left on scale 5.

series. For this purpose, the R-package *wavetresh* [117] is used. Since the package requires the time series to have a length equal to a power of two, we currently artificially extend – where required – a time series of length n to a length $N = 2^{\lceil \log_2(n) \rceil}$, by mirror copying the last segment of the original time series into the extended area. However, we do not consider anomalies which are detected at instances $> n$. DWT-MLEAD utilizes both the detail coefficients $d_{k,\ell}$ and the approximation coefficients $c_{k,\ell}$, computed by the DWT (lines 7–8 in Algorithm 4), where ℓ addresses the level and $k \in 1, \ldots, N$ the time index.

The lowest level $\ell = \log_2(N)$ contains only one coefficient. The highest level $\ell = 0$ has no approximation coefficients but only detail coefficients $d_{k,0}$ which represent the original time series. Since lower levels of the DWT usually do not contain patterns which are useful for anomaly detection, only the L highest levels (L is a parameter of the algorithm) are considered, where $\ell = L - 1$ describes the lowest considered level and $\ell = 0$ addresses the highest possible level (the original time series). In Fig. 5.1 the DWT of a time series from NAB is illustrated.

5.2.1.2 Sliding Windows

In order to express temporal relationships, a simple and common approach in many machine learning tasks involving time series is to employ sliding windows of a certain size w (e.g. $w = 10$), which are used to generate fixed-sized input vectors for a model. Our algorithm employs an individual sliding window for the detail and approximation coefficients at each level of the DWT tree.

By stacking the transposed input vectors, we obtain a matrix $\mathbf{X}$ with w columns which can be used to train a model. In the DWT-MLEAD algorithm (Algorithm 4, lines 10–11), a window of size w_ℓ is slid over the detail and approximation coefficients $d_{k,\ell}$ and $c_{k,\ell}$ at each DWT level $\ell \in \{\ell', \ldots, L\}$ in order to generate the matrices $\mathbf{D}^{(\ell)}$ and $\mathbf{C}^{(\ell)}$. Subsequently, for each matrix a mean vector and a covariance matrix are estimated, as described in the following.

5.2.1.3 Detecting Unusual Patterns on Individual Frequency Scales

In order to distinguish between normal and unusual patterns in the individual levels of the DWT, our algorithm estimates two parameters for each considered level: a mean vector $\bar{\mathbf{x}}$ and a covariance matrix $\bar{\mathbf{\Sigma}}$. This is done separately for the approximation and detail coefficients ($c_{n,\ell}$ and $d_{n,\ell}$). The function ESTIMATE in Algorithm 5 estimates $\bar{\mathbf{x}}$ and $\bar{\mathbf{\Sigma}}$ for a given matrix $\mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{n \times w_\ell}$, with $n = N - w + 1$ being the number of input vectors generated by sliding the window over the time series, $\bar{\mathbf{x}} \in \mathbb{R}^{w_\ell}$ is a w-dimensional vector, which indicates the center of the distribution, and $\bar{\mathbf{\Sigma}} \in \mathbb{R}^{w_\ell \times w_\ell}$ describes the covariances between individual dimensions. The dimensions of both quantities depend on the length of the sliding window w_ℓ used in each level of the DWT. In Algorithm 4, line 13, DWT-MLEAD estimates both parameters for each $\mathbf{D}^{(\ell)}$ and $\mathbf{C}^{(\ell)}$.

For a given estimate $(\bar{\mathbf{x}}, \bar{\mathbf{\Sigma}})$, one can compute the so-called squared Mahalanobis distance for a new observation $\mathbf{x}_i \in \mathbb{R}^{w_\ell}$:

$$m(\mathbf{x}_i) = (\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T} \bar{\mathbf{\Sigma}}^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}) \tag{5.1}$$

The Mahalanobis distance is a measure which can be used as an indicator for unusual behavior. If the Mahalanobis distance is large, it is likely that the observed point $\mathbf{x}$ does not represent a normal instance.

Note that using the Mahalanobis distance in this sense as anomaly indicator does not require any assumption about the distribution of the data $\mathbf{x}_i$. It is often found in the literature that Mahalanobis distance requires the data to be normally distributed, but this is not necessary to make $m(\mathbf{x}_i)$ a useful norm. (If the data were Gaussian distributed, then $m(x_i)$ would follow a Chi-Squared distribution, as derived in Appendix B.3, but we do not need this fact in our algorithm.)

For every entry in $\mathbf{D}^{(\ell)}$ and $\mathbf{C}^{(\ell)}$ we compute and collect the squared Mahalanobis distances in a vector $\mathbf{m}$ using the previously determined parameters $\bar{\mathbf{x}}$ and $\bar{\mathbf{\Sigma}}$. This is done in function MAHALANOBIS of Algorithm 5.

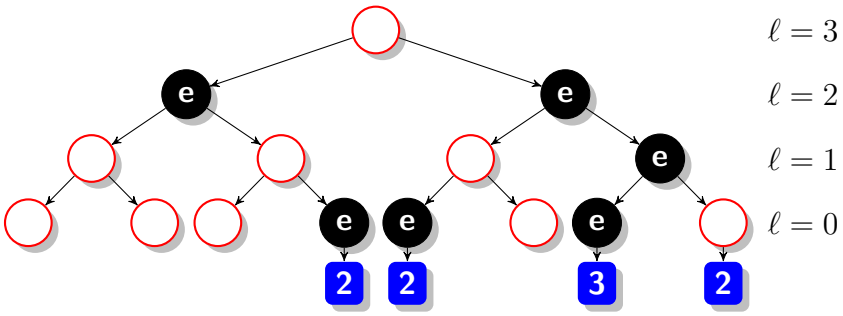

Figure 5.2: Detecting anomalies with leaf counters. Along the vertical axis are the DWT levels ℓ, along the horizontal axis are the time indices k. The leftmost event e thus comes from either an unusual $c_{1,L-2}$ or $d_{1,L-2}$. Each event increases the leaf counters (blue rectangles) connected with the e node. Only counters with count ≥ 2 are shown.

5.2.1.4 Quantile Boundaries

In order to separate unusual from usual window patterns in $\mathbf{D}^{(\ell)}$ and $\mathbf{C}^{(\ell)}$, one has to find a suitable threshold for the squared Mahalanobis distance. We use an empirical ϵ-quantile m_ϵ (e.g. the 99^{th} percentile) to determine the threshold. After determining m_ϵ in Algorithm 4, line 15, instances are flagged as "unusual" in a binary vector $\mathbf{a}$ if their (squared) Mahalanobis distance m_i lies above m_ϵ (line 16).

5.2.1.5 Leaf Counters

For each instance in the original time series the DWT-MLEAD algorithm maintains a leaf counter h_i. If an instance $c_{k,\ell}$ or $d_{k,\ell}$ on a certain level ℓ of the DWT is flagged as unusual (has a flag $a_k = 1$) then an event e – marked as a black node in Fig. 5.2 – is passed down the DWT tree to all leaf nodes connected with the e node. Each leaf node has a counter h_i (blue rectangles in Fig. 5.2) which counts all such events (Algorithm 4, line 17). After all events are processed, all counters with a count $h_i < 2$ are deleted (line 18).

5.2.1.6 Detecting the Anomalies

Once all the leaf counters are updated, DWT-MLEAD forms clusters C_j of all leaf counters h_i having a neighbor not more than d_{max} apart (Algorithm 4, line 19). Specifically, a cluster C_j is here a set of counters, each counter carrying its leaf position in the original time series and its event count. For each cluster C_j a sum s_j over all event counts is computed. In Fig. 5.2 for example, all counters form *one* cluster with sum $s_j = 9$. If a sum s_j exceeds the predefined threshold B, then the center of cluster C_j is labeled as anomaly event (line 24). The center $\mu(C_j)$ of cluster C_j is the weighted center of mass of all leaf positions, where the weights are the event counts.

Algorithm 4 Offline DWT-MLEAD, an anomaly detection algorithm based on the Discrete Wavelet Transform (DWT).

Define:
    L: Number of levels considered in the DWT
    ϵ for computation of quantiles (e.g., the 1st percentile)
    d_{max}: maximum distance for same-cluster points
    B: threshold for the counter sum in a cluster that triggers an anomaly
function MLEANOMALY($y = (y_1, y_2, \ldots, y_N)$) ▷ N is a power of 2
    Compute DWT of y for levels $\ell \in \{0, \ldots, L-1\}$
    Get detail coefficients $d_{k,\ell}$ and approximation coefficients $c_{k,\ell}$ of DWT
    Initialize a leaf counter $h_i = 0$ for each y_i, counting the events it receives
    Set window sizes for each level: $w_\ell = \max\{2, \log_2(m) - \ell\}$
    $\forall \ell \in \{0, \ldots, L-1\}$: Build $\mathbf{D}^{(\ell)}$, $\mathbf{C}^{(\ell)}$ by sliding window of size w_ℓ over $d_{k,\ell}$, $c_{k,\ell}$
    for all $\mathbf{X} \in \{\mathbf{D}^{(\ell)}, \mathbf{C}^{(\ell)} \mid \ell = 1, \ldots, L-1\} \cup \mathbf{D}^{(0)}$ **do**
        $(\bar{\mathbf{x}}, \bar{\mathbf{\Sigma}}) =$ ESTIMATE($\mathbf{X}$) ▷ Defined in Algorithm 5
        $\mathbf{m} =$ MAHALANOBIS($\mathbf{X}, \bar{\mathbf{x}}, \bar{\mathbf{\Sigma}}$) ▷ Defined in Algorithm 5
        Determine threshold m_ϵ as empirical ϵ-quantile or with $m_\epsilon = \chi^2_{1-\epsilon}(w_\ell)$
        $\mathbf{a} =$ PREDICT($\mathbf{m}, m_\epsilon$) ▷ Defined in Algorithm 5
        For all $\mathbf{a}_i = 1$: Trigger an event moving down the tree to any connected leaf
    When all events are processed: Delete all event counters with count $h_i < 2$
    Form clusters C_j of leaf counters having a neighbor not more than d_{max} apart
    $\mathbf{S} = \{\}$ ▷ Set of detected anomalies
    for all C_j **do**
        $s_j =$ sum of counter values in C_j
        if $s_j > B$ **then**
            $\mathbf{S} = \mathbf{S} \cup \{\mu(C_j)\}$ ▷ Add center $\mu(C_j)$ of C_j to anomaly set
    return S

Algorithm 5 Helper functions for Algorithm 4.

function ESTIMATE($\mathbf{X}$)
  $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$ ▷ Vetor $\mathbf{x}_i \in \mathbb{R}^w$ is the ith row of matrix $\mathbf{X} \in \mathbb{R}^{n\times w}$
  $\bar{\mathbf{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T}$
  return $(\bar{\mathbf{x}}, \bar{\mathbf{\Sigma}})$

function MAHALANOBIS($\mathbf{X}, \bar{\mathbf{x}}, \bar{\mathbf{\Sigma}}$)
  $\mathbf{m}$: vector of size $\mathbb{R}^n$ ▷ n is the number of rows in $\mathbf{X}$
  for each row $\mathbf{x}_i$ of $\mathbf{X}$ **do**
    $m_i = (\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T}\bar{\mathbf{\Sigma}}^{-1}(\mathbf{x}_i - \bar{\mathbf{x}})$
  return m

function PREDICT($\mathbf{m}$, m_ϵ)
  $\mathbf{a}$: vector of same size as $\mathbf{m}$
  for all $\mathbf{a}_i$ **do**
    $\mathbf{a}_i = \begin{cases} 1, & \textbf{if } m_i > m_\epsilon \\ 0, & \textbf{otherwise} \end{cases}$ ▷ Binary anomaly flag vector
  return a

5.2.2 Algorithms and their Settings

In the following, we compare DWT-MLEAD with three online anomaly detection algorithms, namely SORAD, NuPIC, and ADVec. We compare the algorithms on the A3 data of Yahoo's Webscope S5 benchmark and on NAB (introduced in 2.3). Although we did not systematically tune the parameters of each algorithm, we empirically determined for each algorithm and each dataset the best parameters from an informal search.

DWT-MLEAD Overall, three main parameters in Algorithm 4 have to be set, which are fixed for the whole dataset: a threshold $\epsilon \in [0, 1]$ for the ϵ-quantiles, which is varied to adjust the tradeoff between precision and recall, a parameter B (threshold for counter sum), and the number of considered levels L. From Sec. 5.2.1.4, we use the empirical quantiles for the NAB data and the χ^2-based quantiles for the A3 data. We empirically determined the setting $B = 3.5$, $L = \log_2(N) - 5$ for the NAB data and $B = 1$, $L = \log_2(N) - 7$ for the A3 data (longer time series also consider more DWT levels). The window size w_ℓ is set by Algorithm 4 in a level-dependent fashion. In its current form the DWT-MLEAD algorithm

Table 5.1: Results for various algorithms on the A3 and NAB dataset. Shown are the sums of TP, FP, FN over all time series and the metrics precision, recall and F_1, cf. Eqs. (2.1)–(2.3), derived from these sums. All algorithms have their threshold chosen such that F_1 is maximized (in brackets: F_1 for threshold such that FP $\approx$ FN).

Dataset	Algorithm	Threshold	TP	FP	FN	Precision	Recall	F_1 Score
A3	DWT-MLEAD	0.015	806	8	44	**0.99**	**0.95**	**0.97** (0.95)
	NuPIC	0.4	172	267	678	0.39	0.2	0.27 (0.26)
	SORAD	10^{-4}	810	22	40	0.97	**0.95**	0.96 (**0.96**)
	ADVec	20	190	216	660	0.47	0.22	0.3 (0.26)
NAB	DWT-MLEAD	0.02	69	65	46	**0.51**	0.6	**0.55** (**0.55**)
	NuPIC	0.55	76	113	39	0.4	**0.66**	0.5 (0.47)
	SORAD	10^{-9}	57	313	58	0.15	0.5	0.24 (0.21)
	ADVec	100	66	164	49	0.29	0.57	0.38 (0.34)

operates offline on each time series, the remaining algorithms investigated in this section are all online.

SORAD The algorithm's parameters which are set as follows for the experiments: We set the forgetting factor of the algorithm to $\lambda = 0.98$, the anomaly threshold ϵ will be varied over a larger range, and the window-size is set to $w = 10$ for the A3 data and to $w = 200$ for the NAB data.

NuPIC NuPIC is described in Section 3.1.2. We use the standard parameter setting for all experiments. The only parameter which is adjusted by us is an anomaly threshold that can be varied in the interval [0,1] and – similar to ϵ in SORAD and DWT-MLEAD – trades off precision and recall.

ADVec Twitter's ADVec Algorithm (Section 3.1.1) is the last algorithm which we will review in this section. The algorithm requires three main parameters, which are as follows: The first parameter α is used as anomaly threshold. The second parameter, the period-length, is fixed to the value 40, which has shown to give the best results on the investigated data. Finally, we found that the setting of the parameter $\max_{anoms}$ is crucial for the performance of ADVec, especially on the NAB dataset. We choose $\max_{anoms} = 1\%$ for the A3 data and $\max_{anoms} = 0.1\%$ for the NAB data.

Table 5.2: Computation times of the algorithms on datasets A3 and NAB. Shown is the average and standard deviation from 20 runs each. The runs were performed on a PC with an i7-3520M CPU and 8 GB of RAM.

	Computation Time (s)			
Dataset	DWT-MLEAD	SORAD	NuPIC	ADVec
A3	13.6 ± 0.3	34.6 ± 0.1	810.9 ± 1.3	2.6 ± 0.2
NAB	12.2 ± 0.2	111.6 ± 0.2	1636.4 ± 2.7	5.8 ± 0.5

5.2.3 Results for the Offline DWT-MLEAD Algorithm

Table 5.1 summarizes the results for the four algorithms on the A3 and NAB data. On the A3 data with short-term anomalies, DWT-MLEAD and SORAD both clearly outperform the other algorithms NuPIC and ADVec, achieving both, a high precision and recall. NuPIC and ADVec produce a large amount of FP and at the same time miss most of the true short-term anomalies. For the NAB data we observe rather different results: while DWT-MLEAD still outperforms the remaining algorithms according to the overall F_1 score, SORAD now performs the worst according to all metrics. In particular, the precision is rather low for SORAD, due to the large number of FP. NuPIC delivers similar results as DWT-MLEAD, with a slight advantage for DWT-MLEAD.

Two example time series from the NAB data with the detections of the individual algorithms are shown in Fig. 5.3. In the first example it can be clearly seen that SORAD produces many FP at the recurring spikes in the time series. This is due to the fact that SORAD has no long-term memory so that such recurring spikes appear to be anomalous. Only DWT-MLEAD and ADVec detect both anomalies in both examples, although ADVec produces a few more false-positives.

All algorithms examined in this section have a threshold which can be varied in a certain range and which trades off FP and FN (as well as precision and recall) to a certain extent. In Fig. 5.4 the precision is plotted against the recall for different thresholds. For the A3 data the recorded points of DWT-MLEAD and SORAD clearly dominate those of NuPIC and ADVec. For the NAB data the results are more diverse: while SORAD shows the worst performance of all algorithms, DWT-MLEAD and NuPIC show the best performance, with NuPIC having a slightly higher precision in larger recall ranges (recall > 0.6) and DWT-MLEAD in the lower recall ranges (recall < 0.6).

In Table 5.2, the computation times for the four algorithms on the A3 and NAB data are shown (mean and standard deviation from 20 runs). Overall, ADVec shows the best results regarding the computation time. On the A3 (NAB) data DWT-MLEAD is faster by a factor of 2.5 (9) than SORAD and 60 (134) than NuPIC.

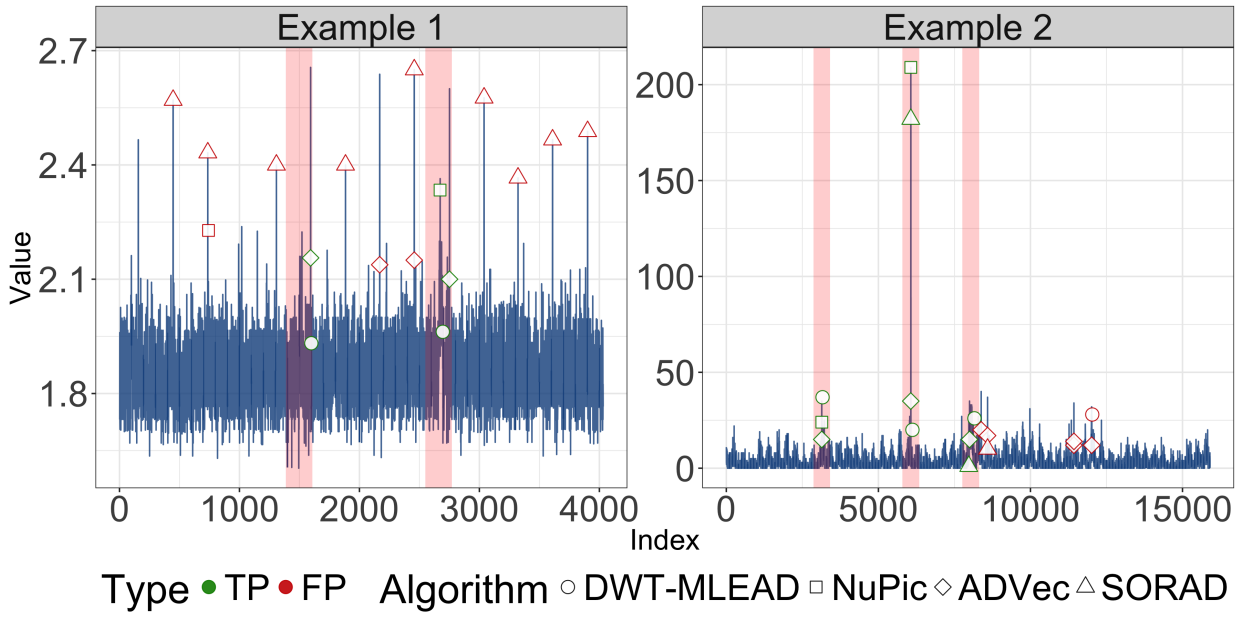

Figure 5.3: Example time series taken from the NAB data with the anomalies detected by the algorithms DWT-MLEAD, NuPIC, ADVec, and SORAD. The red vertical bars in the plot indicate the true anomaly windows. True-positives are indicated by green colors while False-positives are colored red.

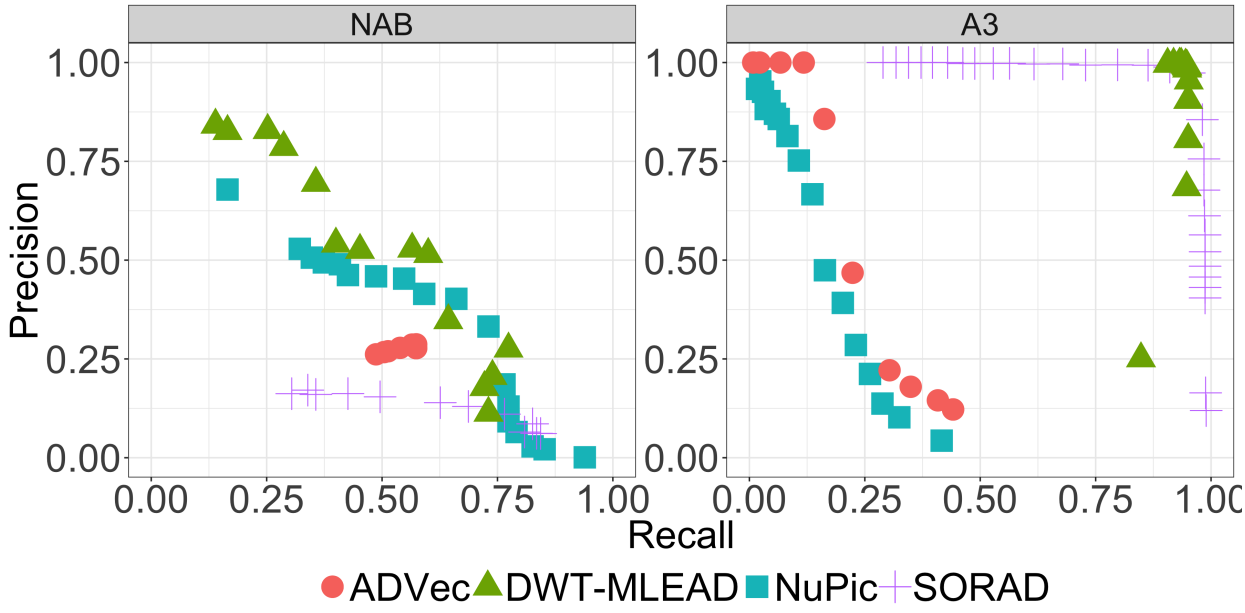

Figure 5.4: Multiobjective plot for the NAB and A3 dataset. Precision and recall are computed based on the results of *all* time series of the corresponding data set.

5.2.4 Discussion

The wavelet transform allows to capture features of the time series on different frequency levels. This is beneficial for detecting both long- and short-term anomalies. It is thus not unexpected that DWT-MLEAD is the only algorithm in our comparison which performs equally well on both benchmarks A3 and NAB. The event pooling mechanism shown in Fig. 5.2 with a minimum event count of 2 in each leaf counter is effective in shielding against noise which may produce an unusual event in just one frequency level. As expected, SORAD operates only well on short-term anomalies, since it analyzes only a short-term window in the original time series.

The algorithm DWT-MLEAD in its current form has these limitations:

- It is offline, i. e. the anomaly detection is undertaken when the whole time series is available. (It is still unsupervised since no information about prior anomalies is given to the algorithm.). This limitation will be lifted in Section 5.3.
- We assume a certain degree of stationarity for the algorithm to work. Trends and change-points cannot be handled well in the offline form. A (semi-)online version could offer more flexibility in the sense that trends and change-points can be learned.
- If a time series has long-term periodic structures, not all anomalies might be detected correctly. This can happen if the frequency of the long-term periodic structure is lower than the lowest wavelet level ℓ considered in Algorithm 4. In such cases it might help to extend the algorithm by a periodicity detector and subtract such a periodicity prior to analyzing the time series with DWT-MLEAD.

5.2.5 Summary

We have shown that the discrete wavelet transform (DWT) is beneficial for detecting anomalies in time series on various time scales. Specifically, our new algorithm DWT-MLEAD shows consistently good results on two larger benchmarks, one containing short-term anomalies (A3) and the other containing long-term anomalies (NAB). We tested this algorithm against three other state-of-the-art anomaly detectors and found DWT in first place on both benchmarks. It is remarkable that a single algorithmic principle works well over such a diverse set of time series. Due to the efficient implementation available for DWT, our algorithm is computationally efficient (fast) as well. However, as mentioned before, DWT-MLEAD is not online yet. In the following section, we add several modifications to the algorithm to operate fully online.

5.3 Online-Adaptable DWT-MLEAD Algorithm

In order to turn the offline DWT-MLEAD algorithm, presented in the previous section, into a fully online algorithm, several points have to be taken into consideration: In an online setup, a causal implementation of all algorithm's components is required. The DWT, the sliding

window approach, the estimation of mean and covariance, and the event detection on the individual layers of the DWT tree have to be modified to satisfy the causality requirement. Furthermore, not all components can be directly translated into an incremental (online) version. For example, it is not straight-forward to estimate a weighted sample mean and covariance matrix incrementally, and an online version has to be derived (see Appendix B.2). Also, stability issues in online settings have to be considered while ensuring that the model acquires new knowledge as fast as possible (stability-plasticity dilemma). In the following we present a solution for an online DWT-MLEAD algorithm and study it on a comprehensive benchmark.

5.3.1 Online and Causal DWT & Sliding Windows

The online DWT-MLEAD algorithm also utilizes both the detail coefficients $d_{k,\ell}$ and the approximation coefficients $c_{k,\ell}$. For the online implementation of the algorithm, a strictly causal computation scheme is adhered to: For example, two data points in the original time series have to be collected first before the next coefficient in level $\ell = 1$ can be computed. Similarly, 2^ℓ data points from the original time series are necessary to compute the next coefficient in level ℓ.

We experimented with the length of the sliding windows and found that for the online version of DWT-MLEAD, window sizes in the form of $w_\ell = \max\{1, \lfloor b^{o-\ell} \rfloor\}$ appear to be the best choice. $b, o \in \mathbb{R}$ are two additional hyperparameters of the algorithm. As soon as a new coefficient in level ℓ is available ($c_{n,\ell}$ or $d_{n,\ell}$), the corresponding window is slid one step further and the new window embedding is collected and passed to a model, which estimates the likelihood of observing such a vector. Unlikely vectors would indicate unusual behavior on the corresponding DWT level. The sliding windows at lower levels are moved with a slower rate than those on higher levels, since new coefficients are only generated after every 2^ℓ time steps in the original time series. This is necessary, to ensure causality of the system.

Anomaly detection starts after an initial transient phase, when the sliding windows can be completely filled.

5.3.2 Online Estimation of the Mean and Covariance Matrix

Since the DWT-MLEAD algorithm operates in an online fashion, the parameter estimations also have to be updated incrementally for each new data point. For this purpose we use an exponentially decaying weighted estimator with an forgetting factor $\lambda \in (0, 1]$. The forgetting factor controls at which rate past observations fade out over time. A value of λ close to 1 results in an algorithm with a very long memory, whereas small values (usually not smaller than 0.9) can significantly limit the memory of the estimator. By allowing the estimator to gradually forget historic information, the algorithm can adapt to new concepts in the data stream. Furthermore, with $\lambda < 1$ we can prevent (under most conditions) a numeric overflow of the required accumulator (the sum of squares of differences from the current mean). However, forgetting can also lead to a higher variance in the parameter

estimates. The pseudo-code of the estimator can be found in Alg. 7, lines 1 – 8. Note that it is not actually necessary to compute the covariance matrix, since only its matrix inverse is required in later steps. Therefore, we directly estimate the inverse of the sum of squares of differences from the current mean $\bar{\mathbf{M}}_n^{-1}$. Since the inverse $\bar{\mathbf{M}}_n^{-1}$ has to be re-computed for every new data point, which can be computationally expensive for larger dimensions, we use the Sherman-Morrison formula [150] to incrementally update $\bar{\mathbf{M}}_n^{-1}$. The inverse of the covariance matrix is given by $\bar{\mathbf{\Sigma}}_n^{-1} = W_n \bar{\mathbf{M}}_n^{-1}$. A detailed derivation of the exponentially decaying estimator of the mean and covariance matrix is given in Appendix B.2.

5.3.3 Detecting Events in the DWT Tree and Anomaly Detection

Since DWT-MLEAD estimates a mean and covariance matrix for every set of DWT-coefficients on the levels $\ell \in [0, 1, \ldots, L]$, it is possible to examine each newly observed value $c_{n,\ell}$ and $d_{n,\ell}$ in the context of its current sliding window in order to detect unusual patterns. For each new data point the current window embed vector is determined and the squared Mahalanobis distance $m_{\mathbf{x}_n}$ to the center of the distribution is computed for this vector. Subsequently, this distance is compared to a threshold m_ϵ. We assume that $\mathbf{x}_n$ is roughly Gaussian distributed. Since a Gaussian random variable has a squared Mahalanobis distance to its mean, which is Chi-squared (χ^2) distributed with w_ℓ degrees of freedom, we set m_ϵ by simply computing the $(1-\epsilon)$-quantile of the χ^2-distribution (function PREDICT in Algorithm 7, lines 10–15). Although the assumption of a Gaussian distribution is often violated in practice, we found that the threshold m_ϵ, determined based on the χ^2-distribution, usually is a reasonable choice. We also experimented with other online approaches, such as the P^2 algorithm [70], designed to incrementally track quantiles, but found the χ^2-quantiles to deliver better results.

If the Mahalanobis distance $m_{\mathbf{x}_n}$ exceeds the threshold m_ϵ, the current instance $c_{n,\ell}$ or $d_{n,\ell}$ is flagged as unusual and an event e is passed down the DWT tree, as illustrated in Fig. 5.5. Events arriving at the leaf nodes are summed up in a global, exponentially decaying event counter E_i (Algorithm 6, line 25). If the activity in a subtree of the DWT exceeds a certain limit, hence, if many events are produced in a short time, E_i will increase fast. As soon as E_i is larger than a specified threshold B, an anomaly will be fired and the instance i in the time series will be flagged. In order to avoid many detections in a short time, a new anomaly cannot be fired again until E_i has faded away and falls below threshold $\frac{2}{3}B$.

In order to detect extreme outlier events, a simple heuristic is used: The algorithm flags a point as anomalous, if it exceeds the current minimum/maximum by more than 20% of the min-max range.

5.3.4 Algorithmic Setup

In this section, we compare online DWT-MLEAD to two other online anomaly detection algorithms. For each algorithm *one* standard parameter setting is chosen which is then used for all experiments across all datasets. Only an anomaly threshold parameter is varied

Algorithm 6 An online version of DWT-MLEAD, an anomaly detection algorithm using the Discrete Wavelet Transform.

Define parameters:
  L: maximum number of levels considered in the DWT
  b, o: for the computation of the sliding window sizes w_ℓ
  λ: forgetting factor for the estimation of the Gaussian distributions
  ϵ: quantile of χ^2-distribution
  B: threshold for global event counter that triggers an anomaly

Initialize:
  Set window sizes for each level: $w_\ell = \max\{1, \lfloor b^{o-\ell} \rfloor\}$
  Global event counter: $E_0 = 0$
  Discount factor: $\gamma = \frac{w_L - 1}{w_L + 1}$
  Allow to trigger anomaly with: $A = \text{true}$
  Initialize all $P_0^{(c,\ell)}$ and $P_0^{(d,\ell)}$ with the tuple $(W_0, \bar{\mathbf{x}}_0, \bar{\mathbf{M}}_0^{-1}, \bar{\mathbf{M}}_0)$, where:
    $W_0 \in \mathbb{R}$, $\bar{\mathbf{x}}_0 \in \mathbb{R}^{w_\ell}$ and, $\bar{\mathbf{M}}_0^{-1}, \bar{\mathbf{M}}_0 \in \mathbb{R}^{w_\ell \times w_\ell}$
    $W_0 = 0$, $\bar{\mathbf{x}}_0 = \mathbf{0}$, $\bar{\mathbf{M}}_0^{-1} = \bar{\mathbf{M}}_0 = \mathbf{I}$

function DWTMLEAD(i, y_i) ▷ where $y = (y_1, y_2, \ldots)$ is a streaming time series
  Determine $\ell' = \min(L - 1, \max\{\ell^* \in \mathbb{N}_0 \mid i \mod 2^{\ell^*} = 0\})$
  for all $\ell \in \{0, \ldots, \ell'\}$ **do**
    $n = i/2^\ell$
    Compute DWT coefficients $c_{n,\ell}$ and $d_{n,\ell}$ ▷ if not already present
    $\mathbf{x}_n^{(c)} = (c_{n-w_\ell+1,\ell} \quad \cdots \quad c_{n,\ell})^\intercal$, $\mathbf{x}_n^{(d)} = (d_{n-w_\ell+1,\ell} \quad \ldots \quad d_{n,\ell})^\intercal$ ▷ sliding windows
    $P_n^{(c,\ell)} = \text{UPDATE}(P_{n-1}^{(c,\ell)}, \mathbf{x}_n^{(c)}, \lambda)$, $P_n^{(d,\ell)} = \text{UPDATE}(P_{n-1}^{(d,\ell)}, \mathbf{x}_n^{(d)}, \lambda)$
    $e_\ell = \text{PREDICT}(P_n^{(c,\ell)}, \mathbf{x}_n^{(c)}, \epsilon) + \text{PREDICT}(P_n^{(d,\ell)}, \mathbf{x}_n^{(d)}, \epsilon)$
  $E_i = \gamma E_{i-1} + \sum_{j=0}^{\ell'} e_j$ ▷ Adjust global event counter
  $a_i = \begin{cases} \text{true}, & \textbf{if } A \wedge E_i \geq B \\ \text{false}, & \textbf{otherwise} \end{cases}$ ▷ Flag anomaly at time step i, if threshold is exceeded
  if a_i **then** $A = \text{false}$
  if $E_i < \frac{2}{3}B$ **then**
    $A = \text{true}$ ▷ Allow new anomaly, if event-counter value falls below threshold
  return a_i

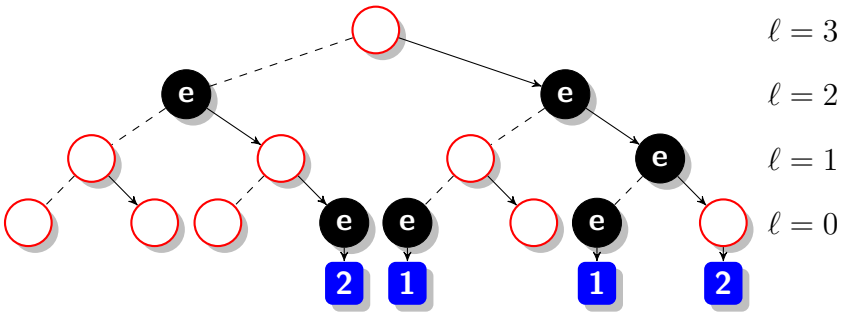

Figure 5.5: Detecting anomalies online with leaf counters. All coefficients (except on the leafs) are always computed bottom-up, based on two child nodes (connected with one dashed and one solid edge). Along the vertical axis are the DWT levels ℓ, along the horizontal axis are the time indices n of the coefficients of the DWT. E.g., the leftmost event e comes from either an unusual $c_{n,2}$ or $d_{n,2}$. Each event is passed down the tree only along the solid edges (causal computation) and increases the right-most leaf counter (blue rectangle) connected with the e node.

for each algorithm and dataset in order to balance precision and recall in a way that the F_1-score is maximized.

DWT-MLEAD As described in Sec. 5.3, in total 6 parameters have to be selected by the user. In order to find an appropriate setting, we did not systematically tune the parameters. Instead, we generated 60 design points using latin hypercube sampling (LHS) and evaluated the algorithm on all time series for these points. The setting $B = 2.20$, $b = 2.27$, $o = 6$, $L = 5$, $\lambda = 0.972$ achieved the highest average F_1-score and will be used throughout the rest of this paper. The parameter ϵ is used as anomaly threshold and is adjusted in the range $\epsilon \in [10^{-6}, 10^{-1}]$. Additionally, to exclude the possibility of overtuning on the data sets, we made the following experiment: We separated the set of all time series in a training and a test set (each containing 50% of the time series) and tuned the parameters of DWT-MLEAD only on the training data. Then the F_1-score was established only on the test set. The results will be given below under the name TRAIN-TEST-SEP.

NuPIC Similarly to the previous section, we run NuPIC with its standard parameter settings. Similarly to DWT-MLEAD, an anomaly threshold can be varied in the interval $[0, 1]$ to control the sensitivity of the algorithm.

ADVec The main three parameters were tuned to achieve the highest average F_1-score. The period-length is set to 40. The second parameter $\max_{anoms}$ is set to $\max_{anoms} = 0.003$. The last parameter α is used as anomaly threshold for ADVec and is adjusted in the range $\alpha \in (10^{-6}, 1)$.

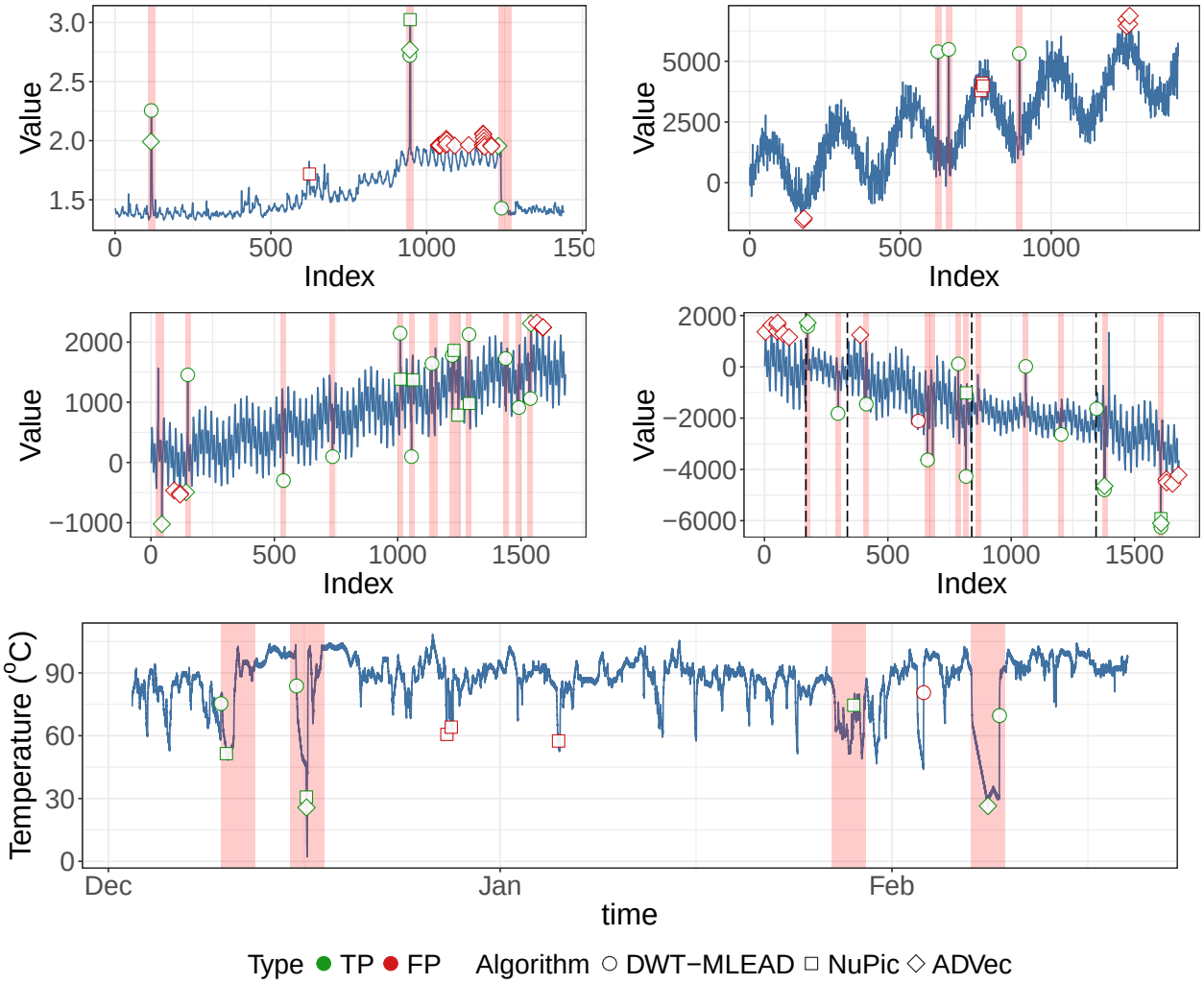

Figure 5.6: Example time series taken from the Yahoo Webscope S5 data and the Numenta Anomaly Benchmark (NAB). In each graph the real anomalies are indicated by the light-red shaded areas. Three algorithms are tested on this data and the individual detections are shown with different symbols. The color of the symbol indicates if the detections were correct (green) or false (red).
Top two rows: One example each from the A1–A4 data. The dashed vertical lines in the A4 data indicate concept changes which should also be detected by the anomaly detectors.
Bottom: Example time series taken from the NAB data. The graph shows the temperature sensor data of an internal component of a large industrial machine over its last few months of operation. The second anomaly (mid of December) is a planned shutdown of the machine. The catastrophic failure occurs end of February when the recordings end.

Algorithm 7 Helper functions for Algorithm 6.

function UPDATE($P_{n-1}, \mathbf{x}_n, \lambda$) ▷ $\mathbf{x}_n \in \mathbb{R}^{w_\ell}$, w_ℓ is the size of the window at scale ℓ

  $(W_{n-1}, \bar{\mathbf{x}}_{n-1}, \bar{\mathbf{M}}_{n-1}^{-1}, \bar{\mathbf{M}}_{n-1}) = P_{n-1}$ ▷ Matrix $\bar{\mathbf{M}}_{n-1}$ is optional (debugging purposes)

  $W_n = \lambda W_{n-1} + 1$

  $\boldsymbol{\Delta}_n = \mathbf{x}_n - \bar{\mathbf{x}}_{n-1}$

  $\bar{\mathbf{x}}_n = \bar{\mathbf{x}}_{n-1} + \frac{1}{W_n}\boldsymbol{\Delta}_n$

  $\bar{\mathbf{M}}_n = \lambda\bar{\mathbf{M}}_{n-1} + \boldsymbol{\Delta}_n(\mathbf{x}_n - \bar{\mathbf{x}}_n)^\intercal$ ▷ Optional, since only inverse $\bar{\mathbf{M}}_n^{-1}$ is required later

  $\bar{\mathbf{M}}_n^{-1} = \frac{1}{\lambda}\bar{\mathbf{M}}_{n-1}^{-1} - \frac{\frac{1}{\lambda}\bar{\mathbf{M}}_{n-1}^{-1}\boldsymbol{\Delta}_n(\mathbf{x}_n-\bar{\mathbf{x}}_n)^\intercal\bar{\mathbf{M}}_{n-1}^{-1}}{\lambda+(\mathbf{x}_n-\bar{\mathbf{x}}_n)^\intercal\bar{\mathbf{M}}_{n-1}^{-1}\boldsymbol{\Delta}_n}$ ▷ Sherman-Morrison Formula

  return $(W_n, \bar{\mathbf{x}}_n, \bar{\mathbf{M}}_n^{-1}, \bar{\mathbf{M}}_n)$ ▷ Return updated parameters

function PREDICT($P_n, \mathbf{x}_n, \epsilon$) ▷ $\mathbf{x}_n \in \mathbb{R}^{w_\ell}$, where w_ℓ is the size of the window at scale ℓ

  $(W_n, \bar{\mathbf{x}}_n, \bar{\mathbf{M}}_n^{-1}, \bar{\mathbf{M}}_n) = P_n$

  $m_{\mathbf{x}_n} = W_n(\mathbf{x}_n - \bar{\mathbf{x}}_n)^\intercal\bar{\mathbf{M}}_n^{-1}(\mathbf{x}_n - \bar{\mathbf{x}}_n)$ ▷ Mahalanobis distance of $\mathbf{x}_n$ to $\bar{\mathbf{x}}_n$

  $m_\epsilon = \chi^2_{1-\epsilon}(w_\ell)$ ▷ Threshold: upper ϵ-quantile of χ^2-distribution

  $e_n = \begin{cases} 1, & \textbf{if } m_{\mathbf{x}_n} > m_\epsilon \\ 0, & \textbf{otherwise} \end{cases}$ ▷ Binary event flag

  return e_n ▷ Unusual data points will cause an event in the DWT-tree

SORAD We tried to manually select the parameters of SORAD. However, we could not find a good compromise for the parameters, to obtain good results on the Yahoo data and NAB. Hence, we choose the same setting as in Section 5.2.2 and set the forgetting factor of the algorithm to $\lambda = 0.98$, and the window-size is set to $w = 10$.

5.3.5 Results

The main results of our experiments are summarized in Tab. 5.3. DWT-MLEAD achieves on all datasets the highest F_1-score (the same as SORAD on A1,A2, and A4). NuPIC has a slightly better precision on A1, but on A2, A3 and A4 the difference in all three metrics is large in favor of DWT-MLEAD. One reason, among others, for the weak performance of NuPIC and ADVec could be that the time series in both datasets contain many anomalies, occurring in part at the very beginning of each time series. Hence, the algorithms have to be up-and-running much faster and have to be able to detect anomalies in short time intervals. Furthermore, the A4 time series contain many concept changes, where amplitudes, seasonalities and noise abruptly change. In order to handle such concept changes, a strong online

Table 5.3: Results for various algorithms on the datasets A1–A4 and NAB. Shown are the metrics precision (how many percent of the detected events are true anomalies), recall (how many percent of the true anomalies are detected) and F_1. All algorithms have their threshold for each dataset chosen such that F_1 is maximized. Each algorithm uses otherwise *one standard parameter setting* for all data sets. The values in square brackets show the F_1-score on the test data of the experiment TRAIN-TEST-SEP.

	Precision, Recall F_1-Score					
Algorithm	**A1**	**A2**	**A3**	**A4**	**NAB**	**Avg**
DWT-MLEAD	0.60, 0.65 0.62 [**0.66**]	1, 0.98 **0.99** [**0.99**]	0.96, 0.97 **0.97** [**0.97**]	0.92, 0.75 **0.83** [**0.83**]	0.66, 0.45 **0.54** [**0.52**]	0.8, 0.76 **0.79** [**0.80**]
SORAD	0.65, 0.67 **0.66**	1, 0.98 **0.99**	0.97, 0.95 0.96	0.9, 0.77 **0.83**	0.09, 0.57 0.15	0.7, 0.8 0.72
NuPIC	0.62, 0.45 0.52	0.59, 0.42 0.49	0.39, 0.20 0.27	0.41, 0.11 0.18	0.40, 0.66 0.5	0.32, 0.37 0.39
ADVec	0.51, 0.56 0.54	0.66, 0.6 0.63	0.54, 0.20 0.29	0.29, 0.15 0.2	0.11, 0.72 0.2	0.32, 0.45 0.37

adaptability is required. For the NAB data, the difference in F_1-score between NuPIC and DWT-MLEAD is not that apparent, although there is a slight advantage for our algorithm. Overall, we can observe in column **Avg** that DWT-MLEAD achieves the highest average values for all three metrics.

The results in Tab. 5.3 are for tuning on all data. The additional experiment TRAIN-TEST-SEP (see Sec. 5.3.4) revealed very similar F_1-scores (less than 1% deviation in the **Avg** score). This observation confirms that DWT-MLEAD operates well on new data and is not overtuned to its parameters.

Since Tab. 5.3 only captures the results for one specific setting of the algorithms anomaly thresholds, we also measured precision and recall for a wide range of thresholds and plotted them against each other, as shown in Fig. 5.7. The overall picture mostly corresponds to the results shown in Tab. 5.3. Only for the NAB data we can observe that for recall values in the range $[0.5, 0.75]$ NuPIC achieves a higher precision and outperforms DWT-MLEAD. Finally, a look on Tab. 5.3 shows that the NAB dataset is a tough benchmark: All tested algorithms are far from being perfect on that dataset, having $F_1 < 0.55$, i.e. there is still room for improvement.

5.3.6 Discussion

Although algorithm DWT-MLEAD could produce good results on the investigated benchmarks, it still has several limitations which leave room for improvement:
(1) For our experiments we only used the relatively simple Haar wavelet. This leads to the limitation that anomalies manifesting themselves in complex frequency patterns might be difficult to detect. Wavelets with stronger localization in the frequency domain (e.g. Gabor

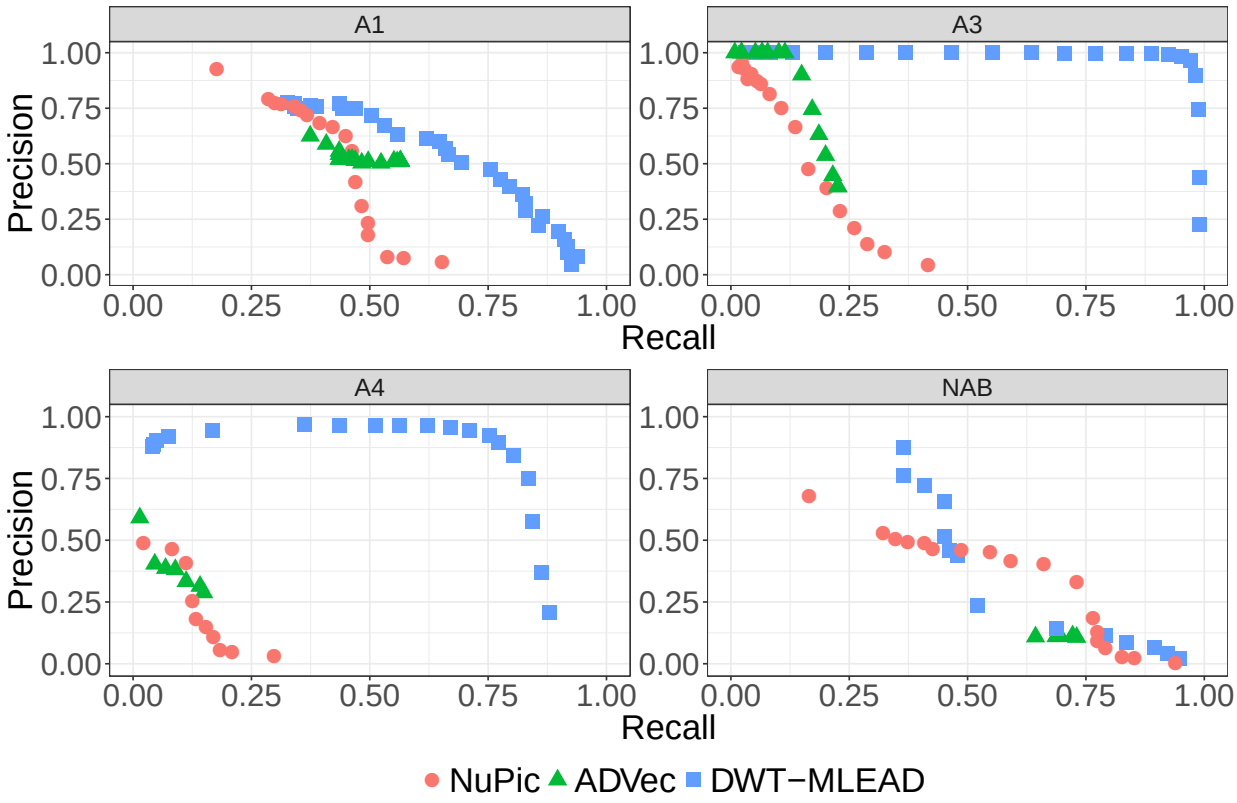

Figure 5.7: Multiobjective plot for Yahoo's Webscope S5 benchmark and the Numenta Anomaly benchmark. The graph for the A2 data is not shown here since the results are very similar to the A3 data.

wavelets or ensembles of such wavelets) might allow to detect frequency changes more reliably.

(2) Due to the strictly causal design of the algorithm, events occurring in the DWT-tree might be asymmetrically distributed along the leaf counters (Fig. 5.5). More events will tend to arrive at the leaf nodes on the right side of each sub-tree, which might lead to undesired effects.[2]

(3) Using the Mahalanobis-distance-based metric for the sample mean and covariance matrix to detect unusual behavior might not be the best choice. Other (perhaps multimodal) distributions might be more effective. To test this, we made some runs with Gaussian mixture models (GMM) which are capable to model more complex distributions. So far, however, these runs resulted in only marginal improvements.

[2]We note in passing that we performed runs with an algorithmic variant where we treated each leaf symmetrical: We wait until an L-subtree is complete, then we collect all events (along the dashed lines in Fig. 5.5 as well) and process them. The price to pay is a certain delay for some leafs and a deviation from the strict online scheme. The results in terms of precision-recall-metrics are a bit better for NAB and a bit worse for A4. Overall, the difference is only marginal.

The NAB dataset is a challenging benchmark, as it includes mostly real world data from many different applications. The time series contain anomalies in high and low frequencies in a large variety of forms. Many anomalies are also very hard to detect for the human eye without suitable domain knowledge.
It is worth mentioning that DWT-MLEAD proved to perform robustly on all time series, without ever showing numerical instabilities from the matrix updates (function UPDATE in Algorithm 7).

5.4 Conclusion & Possible Future Work

In this chapter, we introduced the relatively simple but effective DWT-MLEAD algorithm for offline and online anomaly detection in time series. Generally, the development of widely applicable anomaly detection algorithms is possible when suitable features can be obtained. We found that especially the discrete wavelet transform (DWT) can be an important tool to generate meaningful features across many different frequency scales. Empirical results on a large dataset with 425 time series containing both long-term and short-term anomalies show that DWT-MLEAD is more robust than other state-of-the-art anomaly detectors: Using only *one* fixed parameter setting, (online) DWT-MLEAD achieved an average F_1 twice as large as ADVec's & NuPIC's scores and 10% higher than SORAD's. Furthermore, DWT-MLEAD significantly outperformed SORAD on the NAB data, obtaining a 260% higher F_1-score. Furthermore, the online adaptability of the DWT-MLEAD algorithm appears to be beneficial in the presence of concept drifts and/or changes, as the results on the A4 data of Yahoo's Webscope S5 benchmark suggest. Our anomaly detection algorithm does not require labeled training data; it is unsupervised and infers from the unlabeled data of each time series what is normal and what is anomalous.

As future work several aspects of our algorithm can be improved: Currently, only simple Haar wavelets are used for the offline and online version of the algorithm; experiments with other wavelets or ensembles of wavelets might lead to a significantly increased performance. Furthermore, it might be possible to further reduce the sensitivity of DWT-MLEAD towards its parameters, for example with automatic parameter tuning methods. Finally, also an extension to multivariate time series should only require a few additional steps.

Chapter 6

Learning Quasi-periodic ECG Time Series with LSTM Networks

6.1 Introduction

The previous chapter introduced the DWT-MLEAD algorithm and demonstrated its effectiveness when applied to time series where anomalies appear on different frequency scales.

However, even with DWT-MLEAD, it is challenging to process time series with complex periodic or quasi-periodic behavior where an anomaly might be a time-shifted peak, a peak with a different shape, or other patterns. For such cases, an algorithm has to learn long-term correlations in the data accurately to detect anomalies reliably. A prominent real-world example where quasi-periodic signals have to be analyzed is in electrocardiography. In an electrocardiogram (ECG), the heart's activity is monitored over time by measuring voltages with several electrodes placed on the chest and limbs. A common task in health care and medical diagnosis is to monitor a patient's heart activity and detect abnormal heartbeat patterns, indicating cardiac arrhythmias or other heart-related problems. There are well-maintained databases, e.g., the MIT-BIH database [52], where medical experts annotate a large body of data with numerous types of anomalies. Automated anomaly detection in such ECG data is still challenging because the deviations from nominal behavior are often subtle and require long-range analysis. Furthermore, there are considerable signal variations from patient to patient or even within an ECG time series (noise, baseline wandering, etc.). A common issue is the heart rate variability (even for healthy hearts, there is a variation in the time interval between consecutive heartbeats of several milliseconds), leading to quasi-periodic signals and increasing the complexity of the analysis.

Long short-term memory (LSTM) networks [63], which are a particular form of recurrent neural networks (RNN) and thus belong to the class of deep learning methods, have proven to be particularly useful in learning sequences with long-range dependencies. They avoid the vanishing gradient problem [62] and are more stable and better scalable [54] than other RNN architectures. LSTMs have been successfully advanced the state-of-the-art in many application areas like language modeling and translation, acoustic modeling of speech, analysis of audio data, handwriting recognition, and others [54]. We will use stacked LSTMs as the building block for our ECG time series prediction.

This chapter presents an unsupervised time series anomaly detection based on LSTM networks that learn to predict the normal time series behavior. The prediction error on several prediction horizons is used to build a statistical model of normal behavior. We propose new methods (window-based error correction and outlier removal) essential for a successful model-building process and a high signal-to-noise-ratio. We apply our method to the well-known MIT-BIH ECG data set and present initial results. We obtain a good recall of anomalies while having a very low false alarm rate (FPR) in a fully unsupervised procedure. We also compare with other anomaly detectors (NuPIC, ADVec) from the state-of-the-art.

In Section 6.2, we will describe an LSTM prediction model that is trained to predict over multiple horizons and is applied to time series containing nominal and rare anomalous data. We observe multidimensional error vectors (one vector for each point in time) and estimate a mean and covariance matrix. Based on the Mahalanobis distance, we can assign a probability of being anomalous at each point in time. Section 6.3 describes our experimental setup and the MIT-BIH Arrhythmia Database used in our experiments. Section 6.4 presents and discusses our results, while Section 6.5 concludes.

6.1.1 Related Work

Publicly, there are only relatively few benchmarks for ECG arrhythmia detection available: The MIT-BIH Arrhythmia database [52, 112, 113], the CU Ventricular Tachyarrhythmia Database [121], and the St. Petersburg INCART 12-lead Arrhythmia Database [52]. We will use the MIT-BIH benchmark in this and the following chapter since it contains the most patients (47 patients) and sufficiently long ECG recordings (30 minutes), and it is the most commonly used benchmark in the literature.

Much work is devoted to anomaly detection in ECG readings: Several authors use multi-resolution wavelet-based techniques [139, 162]. A novelty-search approach on ECG data is taken in [90] in order to perform unsupervised anomaly classification. Sivaraks et al. [151] use motif discovery for robust anomaly detection.

Many articles are concerned with the detection of arrhythmias in ECG signals. However, work on unsupervised approaches appears to be less common in this field. The presented approaches are mostly supervised algorithms that are trained to classify different arrhythmia types. Luz et al. give a comprehensive overview of various classification approaches for ECGs in [101]. In [56], Hannun et al. designed a 34-layer convolutional deep neural network to classify 12 different heart arrhythmia types. However, due to its nature, the architecture is supervised and requires annotated data for training. The authors use a massive labeled dataset containing 91,232 single-lead ECGs from 53,549 patients (not publicly available). The trained model achieves very high accuracy on a cardiologist level. Several researchers base their approaches on the discrete wavelet transform (DWT) [171, 143, 168, 7]. Thomas et al. [168] extract, next to other features (some of which might partially require expert knowledge, such as the RR-intervals), dual-tree complex wavelet-based features from the ECG signal and train a neural network for four arrhythmia classes. We found several

works that introduce anomaly detection methods in ECG readings [107, 161, 28, 2, 22, 151].Sivaraks et al. [151] use motif discovery and propose an approach for robust anomaly detection in ECG data.

The works of Malhotra et al. [107] and Chauhan & Vig [28] are the closest to our current approach. They describe the general idea of LSTM-based prediction and their application to ECG, motor sensor, or power-consumption time series. However, the big drawback of [107, 28] is that they need a manual and supervised separation into up to six data sets: training, validation, test sets which are further subdivided into nominal & anomalous subsets. This means that for a real-world application, the ECG data for a new person would need to undergo an expert anomaly classification prior to any training, which would be highly impractical in most application scenarios. Instead, our method aims to use the whole body of data for a person and train the LSTMs without the necessity to have supervised anomaly information.

While Chauhan & Vig [28] only apply their method to one-minute recordings of the ECG signals in the MIT-BIH [52, 112, 113] corpus, we test our approach on the full-length signals (30 minutes each). Additionally, we only consider those 13 time-series signals with less than 50 abnormal events (ECG-13 data, as described in section 2.3) for our initial tests. In the following chapter 7, we extend the benchmark to 25 time series (ECG-25 benchmark, < 250 events per time series). In [28], also ECG signals are used where the vast majority of heartbeats are considered anomalous (e.g., signals with paced beats). These cases lead to a trivial anomaly detection task since an algorithm can simply flag each heartbeat as anomalous.

6.2 Methods

6.2.1 LSTM for Time Series Prediction

The learning task is formulated as a time series forecasting problem. Hence, we attempt to train a model which can predict future values in the time series. This approach's intuition is that the usual quasi-periodic patterns in the ECG time series should be predictable with only minor errors. At the same time, abnormal behavior should lead to significant deviations in the predictions. Although the presented methodology is only applied to ECG data in this chapter, it is sufficiently general to be applied to other (predictable) time series as well.

6.2.1.1 Data Preparation

Consider a d-dimensional time series of length T. In a first step, it is often recommendable to scale or normalize the time series's dimensions. In our setup, each ECG signal dimension is scaled to the range $[-1, 1]$. The training and test samples are generated by extracting subsequences of suitable length from the original time series. This is done by sliding a window of length W with a lag of 1 over the time series and collecting the windowed data in a tensor

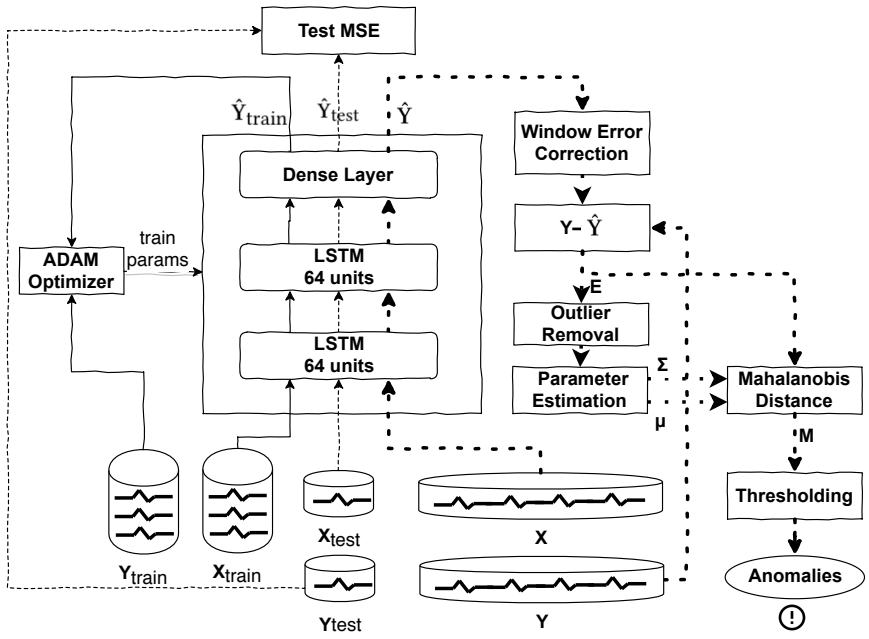

Figure 6.1: LSTM Architecture. The LSTM model is trained with $\mathbf{X}_{\text{train}}$ and $\mathbf{Y}_{\text{train}}$. $\mathbf{X}_{\text{test}}$ and $\mathbf{Y}_{\text{test}}$ are used to verify if the model is overfitting. In order to detect anomalies, the algorithm passes the whole time series $\mathbf{X}$ through the LSTM model and computes residuals for the predicted tensor $\hat{\mathbf{Y}}$.

$\mathcal{D} \in \mathbb{R}^{T' \times W \times d_{in}}$, where T' is the number of sub-sequences. We select all d dimensions of the time series so that $d_{in} = d$. Then, the first 80% of the samples of $\mathcal{D}$ are selected to form the training data $\mathbf{X}_{\text{train}}$. The remaining 20% are used as a test set $\mathbf{X}_{\text{test}}$ to compute an unbiased error estimate later. While the inputs are d_{in}-dimensional, the output-targets for each time step have the dimension m, since one can select for one time series multiple (m) prediction horizons. Technically, it is also possible to predict several time series dimensions with $d_{out} \leq d$ simultaneously in one model; however, in our experiments, the results do not improve in this case (for the investigated ECG-13 time series data). The targets $\mathbf{y}_t \in \mathbb{R}^m$ are future values of the selected signal at times $t + h_i$ for $i \in \{1, ..., m\}$, where the horizons are specified in $H = (h_1, h_2, \ldots, h_m)$. Since we follow a many-to-many time series prediction approach, where the algorithm performs a prediction at each instance of time t, the tensor containing the target signals has the shape $\mathbb{R}^{T' \times W \times m}$ with $T' = T - W - \max(H) + 1$. As before, the first 80% of the targets are used for training ($\mathbf{Y}_{\text{train}}$) and the remaining targets for the test set ($\mathbf{Y}_{\text{test}}$).

6.2.1.2 Model Architecture and Training

A stacked LSTM architecture [63] shown in Figure 6.1 with $L = 2$ layers is used to learn the prediction task. Each layer consists of $u = 64$ units. A dense output layer with m units and a linear activation generates the predictions for the specified prediction horizons in H.

The net is trained with the sub-sequences of length W taken in mini-batches of size B from the training inputs $\mathbf{X}_{\text{train}}$ and targets $\mathbf{Y}_{\text{train}}$. 10% of the training data is held out for the validation set. The LSTM model is trained by using the Adam optimizer [82] to minimize the mean-squared-error (MSE) loss. Other loss functions, such as log-cosh (logarithm of the hyperbolic cosine) and MAE (mean absolute error), were tested as well and produced similar results for our data. We could obtain similar results with the loss functions log-cosh (logarithm of the hyperbolic cosine) and MAE (mean absolute error). Early stopping is applied to prevent overfitting of the model and to reduce the overall time required for the training process. For this purpose, the MSE on the validation set is tracked. For most of the investigated time series, 10-20 epochs are sufficient to reach a minimum of the validation error.

6.2.2 Modeling the Residuals

After the LSTM prediction model is trained, the whole time series $\mathbf{X} \in \mathbb{R}^{T \times d_{in}}$ of length T is passed through the model, and a tensor $\hat{\mathbf{Y}}$ of shape $\mathbb{R}^{T \times m}$ is predicted. Then, the prediction errors $\mathbf{E} = \mathbf{Y} - \hat{\mathbf{Y}}$ are calculated, where $\mathbf{Y}$ contains the true target values for the horizons in H. Now, each row i in the matrix $\mathbf{E}$ represents an error vector $\mathbf{e}_i \in \mathbb{R}^m$.

We noticed in our initial experiments that it was not possible to find good Gaussian fits for the individual dimensions of the prediction errors in $\mathbf{E}$. An example for this is shown in Figure 6.2, upper part. This was due to the fact that the tails of the error distributions contained many outliers, which significantly distorted the estimated fit. Hence, we decided to remove the outliers in each dimension's tails (only during the Gaussian modeling phase). We could find good solutions by discarding the upper and lower 3% quantile in each dimension. We observed that this approach usually removes slightly more than 20% of the data records in our experiments. Note that the Mahalanobis distance itself does not require any particular assumptions about the data distribution (such as normality). However, we found that better results are obtained when removing the outliers in the distribution tails. Experimentally, we also tried using the minimum covariance determinant estimator (MCD) [142] (a robust estimator of mean and covariance matrix) but found that the results are similar to the simple heuristic described above. Since MCD is computationally less efficient, we decide to use our heuristic instead.

After removing the outliers from the prediction errors $\mathbf{E}$, the covariance matrix $\bar{\mathbf{\Sigma}}$ and mean vector $\bar{\mathbf{e}}$ are computed for the cleaned matrix $\mathbf{E}$. Then, the squared Mahalanobis distance

$$M(\mathbf{e}_i) = (\mathbf{e}_i - \bar{\mathbf{e}})^\intercal \bar{\mathbf{\Sigma}}^{-1} (\mathbf{e}_i - \bar{\mathbf{e}}) \tag{6.1}$$

to the mean vector $\bar{\mathbf{e}}$ is determined for each error vector $\mathbf{e}_i$ in $\mathbf{E}$.

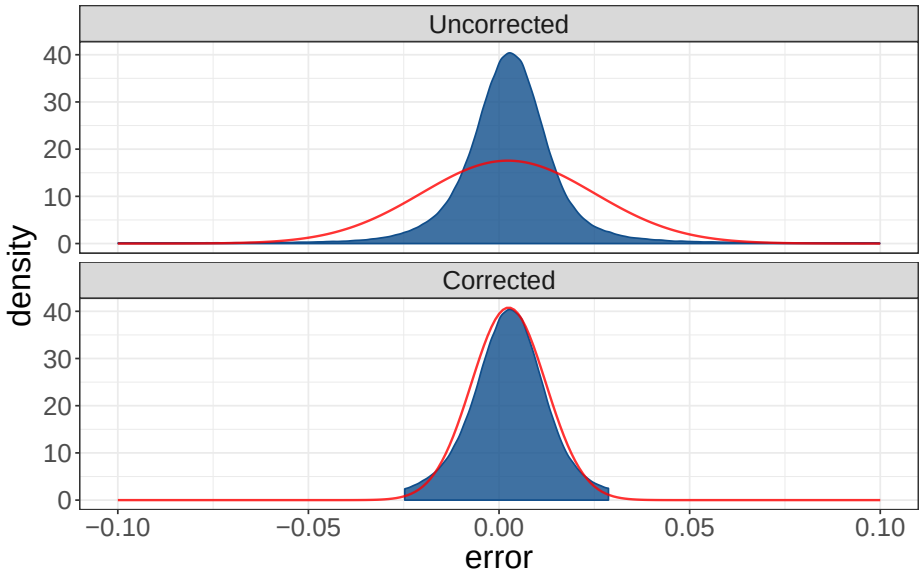

Figure 6.2: Gaussian fit without and with the removal of outliers in the tails of the error distribution. Exemplary, this is shown for one dimension of the overall error distribution. The blue curve shows the empiric error distribution. The red curve depicts the estimated Gaussian. Although our anomaly score metric, the Mahalanobis distance, does not require the assumption of Gaussian-distributed data, this plot nevertheless demonstrates that there are likely many outliers in the error distribution tails which might distort the estimations of mean and covariance matrix.

6.2.3 Anomaly Detection

For most data points in the time series, the corresponding Mahalanobis distance will be comparably small since they are located close to the distribution's mean. On the other side, unusual patterns in the error vectors $\mathbf{e}_i$ – such as large errors in one or more dimensions – will result in large values in the Mahalanobis distance. Therefore, the Mahalanobis distance can be used as an indicator for anomalous behavior in the time-series signal. In our LSTM-AD algorithm, points with a Mahalanobis distance larger than a specified anomaly threshold will be flagged as anomalous. Figure 6.3 shows exemplarily the Mahalanobis distance over time for a selected ECG signal. Depending on the choice of threshold, more or fewer points will be classified as anomalous. If the threshold is set too small, the algorithm will likely produce many false detections. If the threshold is chosen too large, many anomalies might be missed. Ideally, a threshold can be found which allows identifying all anomalies without any false detections. However, in practice, one usually has to trade-off true and false detections and select the threshold according to the own requirements.

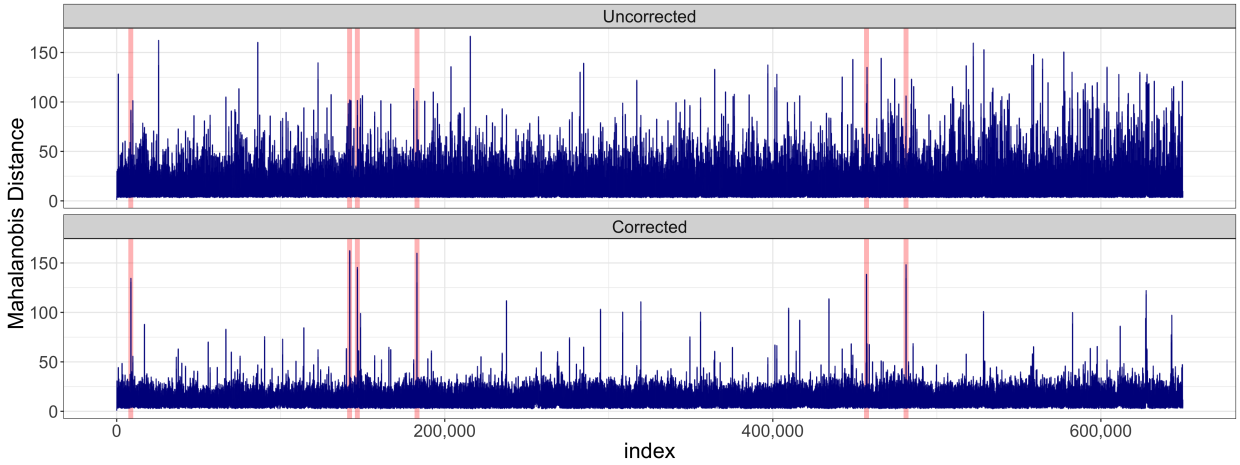

Figure 6.3: Mahalanobis distance over an ECG time series, before and after the window-based error correction method is applied.

6.2.4 Window-Based Error Correction

Initially, we could not obtain very good results when running our algorithm on several example ECG time series. The Mahalanobis distance signal was rather noisy and could not be used to distinguish between nominal and abnormal patterns in the data. In Figure 6.3 (top) this problem is visualized for one example time series. Further investigation showed that the LSTM network predictions were good in general but not sufficiently accurate near the heartbeat peaks. The prediction had been slightly shifted (up to ten time steps) forwards or backward compared to the actual signal. We could identify that the quasi-periodic character of most ECG signals (small but ubiquitous frequency changes in the heartbeat, often referred to as heart rate variability) is the primary source of this problem. The following solution for this problem is proposed: In order to address the variability in the frequency of the signal, small corrections in the predictions of the individual horizons will be permitted. For each output dimension $k \in \{1, \ldots, m\}$ the target values $y_{t,k}$ are compared to the neighbored predictions $\hat{y} \in \hat{Y}_{t,k}^{(win)}$ and the prediction with the smallest absolute error is effectively taken:

$$\hat{y}_{t,k} \leftarrow \arg\min_{\hat{y} \in \hat{Y}_{t,k}^{(win)}} |y_{t,k} - \hat{y}| \tag{6.2}$$

$$\hat{Y}_{t,k}^{(win)} = [\hat{y}_{t-c_k,k}, \ldots, \hat{y}_{t,k}, \ldots, \hat{y}_{t+c_k,k}] \tag{6.3}$$

We found that reasonable results can be achieved with window parameters c_k up to a length of 10, depending on the prediction horizon h_k:

$$c_k = \min(h_k, 10). \tag{6.4}$$

The window-based error correction is applied right after the LSTM prediction of $\hat{\mathbf{Y}}$ and before the prediction errors $\mathbf{E}$ are computed in subsection 6.2.2.

Although this approach corrects the predictions of the LSTM network explicitly with the actual target values $\mathbf{Y}$ of the time series, it is not supervised in the sense that no anomaly labels are presented to the algorithm at any time of the training and correction process. As will be shown in Sec. 6.4, we could significantly improve the performance of LSTM-AD utilizing this correction step.

6.3 Experimental Setup

6.3.1 The ECG-13 Benchmark

In this chapter, we use the ECG-13 dataset, as described in more detail in Section 2.3for our initial experiments. This dataset contains 13 time series taken from the MIT-BIH database [52], each with 650 000 2-dimensional points (which corresponds to roughly 30 minutes). The selection criterion for the 13 time series (patients) and the abnormal classes was such that each time series has 50 or fewer abnormal heartbeats of that class. In total, the ECG-13 dataset contains 130 anomalies from six anomaly classes. Apart from the normalization of the data, no further preprocessing is performed. In the following chapter 7, we extend the benchmark and use all MIT-BIH ECG time series with less than 250 events, resulting in a set of 25 time series (called the ECG-25 dataset).

6.3.2 Parameterization of the Algorithms

All algorithms compared in this work require a set of parameters, which are – if not mentioned otherwise – fixed for all experiments. For each algorithm, an anomaly threshold can be set, which specifies the algorithm's sensitivity towards anomalies and trades off false detections (false positives) and missed anomalies (false negatives). This threshold is usually set according to the anomaly detection task's requirements – allowing either a higher precision or a higher recall.

LSTM-AD We implemented our proposed algorithm using the Keras framework [30] with a TensorFlow [1] backend. For our LSTM-AD algorithm, both the MLII and V5 signal will be used as inputs of the model, and only the MLII signal is predicted for the specified horizons. The parameters of the algorithm are summarized in Table 6.1. Most of the parameters are related to the stacked LSTM network. The parameters were selected after a few informal pre-experiments, but we did not systematically tune the parameters.

ADVec There are mainly three parameters (described in Section 3.1.1) which have to be provided for Twitter's ADVec algorithm: The first parameter α represents the level of statistical significance with which to accept or reject anomalies. This parameter is used as anomaly threshold. The second parameter $\max_{anoms}$ specifies the maximum number of

Table 6.1: Summary of the the parameters used for the LSTM anomaly detector.

Parameter	value	Parameter	value
H	$(1, 3, \ldots, 47, 49)$	$\mathcal{L}$	MSE
L	3	u	$(64, 64)$
B	2048	W	80
optimizer	ADAM	α_{init}	0.001
d_{in}	2	d_{out}	1 (MLII)

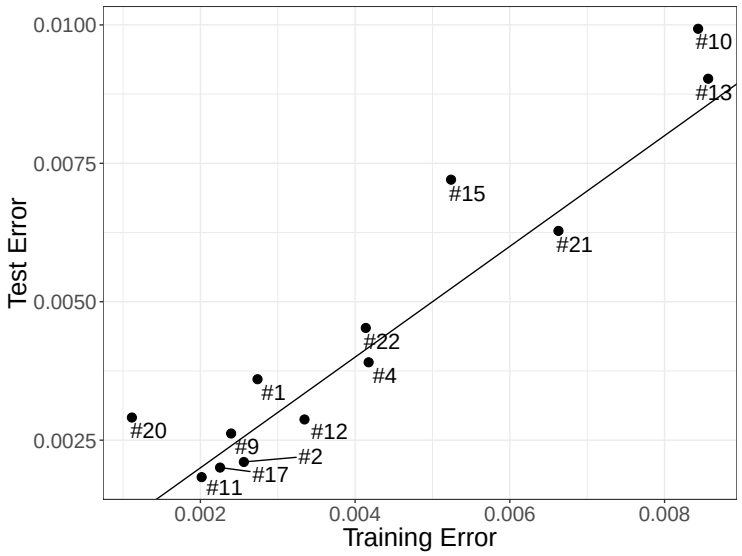

Figure 6.4: Training vs. test errors (MSE) for the individual time series. The median of all such training and test errors is $2.91 \cdot 10^{-3}$ and $3.43 \cdot 10^{-3}$, respectively. The black line depicts the identity line.

anomalies that the algorithm will detect as a percentage of the data. Although we did not tune the parameter extensively, we found the $\max_{anoms} = 0.05$ to deliver the best results.

NuPIC Numenta's anomaly detection algorithm [49] has a large set of parameters which have to be set. As in other experiments, we decided to use the standard parameter settings recommended in [89]. NuPIC outputs an anomaly likelihood for each time series point in the interval [0,1], which is suitably thresholded to control the sensitivity of the algorithm.

6.4 Results & Analysis

Firstly, we confirm with Fig. 6.4 that our LSTM models do not overfit since the training and test set errors are for all time series nearly the same. We note in passing that the time series with a somewhat larger test set error (No. 10, 15, and 20) have – as visual inspection shows

– a test set that varies from the bulk of the training data due to larger non-stationarities in the data set.

As seen in Table 6.2, the Mahalanobis distance is generally a good indicator for separating nominal from anomalous behavior in the heartbeat signals if a suitable threshold is known. For all time series, a recall value of 0.5 or larger can be observed, and with one exception, the F_1-score exceeds the value 0.5. On average, a F_1-score of approximately 0.81 can be achieved for all time series. Note that all FPRs are smaller than $3 \cdot 10^{-5}$.

Table 6.2: Results for the ECG-13 dataset (all ECG time series with less than 50 anomalies; in total 13 time series). For these results, the anomaly threshold is chosen for each time series individually so that the F_1-score is maximized. The row Σ represents the metrics for the sum of TP, FN and FP over all 13 time series.

No.	threshold	TP	FN	FP	Prec	Rec	F_1	FPR $*10^5$	PLR $/10^5$
1	49.31	17	17	14	0.55	0.50	0.52	2.15	0.23
2	71.31	6	0	4	0.60	1.00	0.75	0.62	1.62
4	11.25	1	1	0	1.00	0.50	0.67	0.00	Inf
9	27.12	15	13	3	0.83	0.54	0.65	0.46	1.16
10	14.96	33	7	6	0.85	0.82	0.84	0.92	0.89
11	60.75	1	0	0	1.00	1.00	1.00	0.00	Inf
12	59.18	2	0	0	1.00	1.00	1.00	0.00	Inf
13	116.93	6	0	0	1.00	1.00	1.00	0.00	Inf
15	99.74	5	0	5	0.50	1.00	0.67	0.77	1.30
17	40.02	1	0	0	1.00	1.00	1.00	0.00	Inf
20	75.40	1	0	0	1.00	1.00	1.00	0.00	Inf
21	30.30	1	0	0	1.00	1.00	1.00	0.00	Inf
22	121.42	3	0	7	0.30	1.00	0.46	1.08	0.93
mean	–	7	2	3	0.82	0.87	0.81	0.46	Inf
Σ	–	92	38	39	0.70	0.71	0.70	0.46	1.53

Since the anomaly threshold is used for trading off false-positive and false-negatives (precision and recall), one can vary the threshold in a specific range and collect the results for different values. This is done in Figure 6.5, which also shows the results for different thresholds for ADVec and NuPIC. It has to be noted that ADVec accounts only for fixed-length seasonalities, it is not built for quasi-periodic signals as they occur in ECG readings, so it is quite understandable that it has only low performance here.

In Figure 6.6, two excerpts of time series No. 13 (left) and No. 10 (right) with the detections of our LSTM-AD algorithm, NuPIC and ADVec are exemplarily shown. In both examples, it can be seen that LSTM-AD detects all indicated anomalies, while NuPIC and ADvec only detect two and one anomalies, respectively. Additionally, the other two algorithms produce several false positives.

The importance of our proposed window-based error correction method (Sec. 6.2.4) is illustrated in Figure 6.3 for ECG signal No. 13: If no window-based error correction is applied, the obtained Mahalanobis distance cannot be suitably used to distinguish between

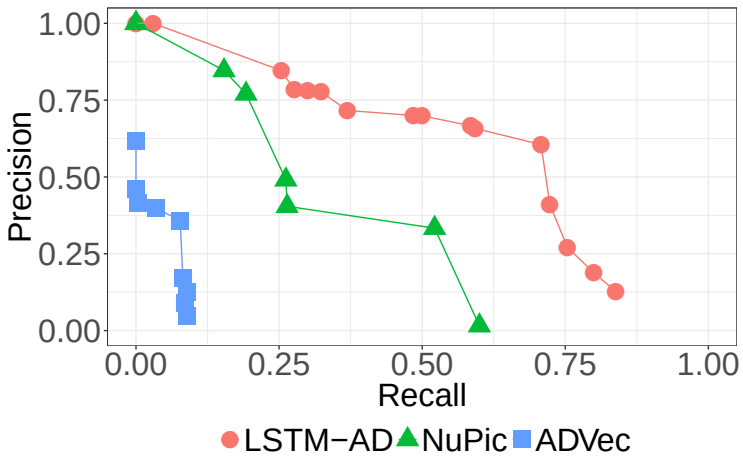

Figure 6.5: Precision-Recall plot for LSTM-AD, NuPIC and ADVec on the ECG-13 data. Precision and recall are computed over the sum of TP, FP, FN of the 13 ECG time series. For all three algorithms, each point is generated by scaling the individual best thresholds up and down by a common factor. For LSTM-AD, the best thresholds are reported in Table 6.2.

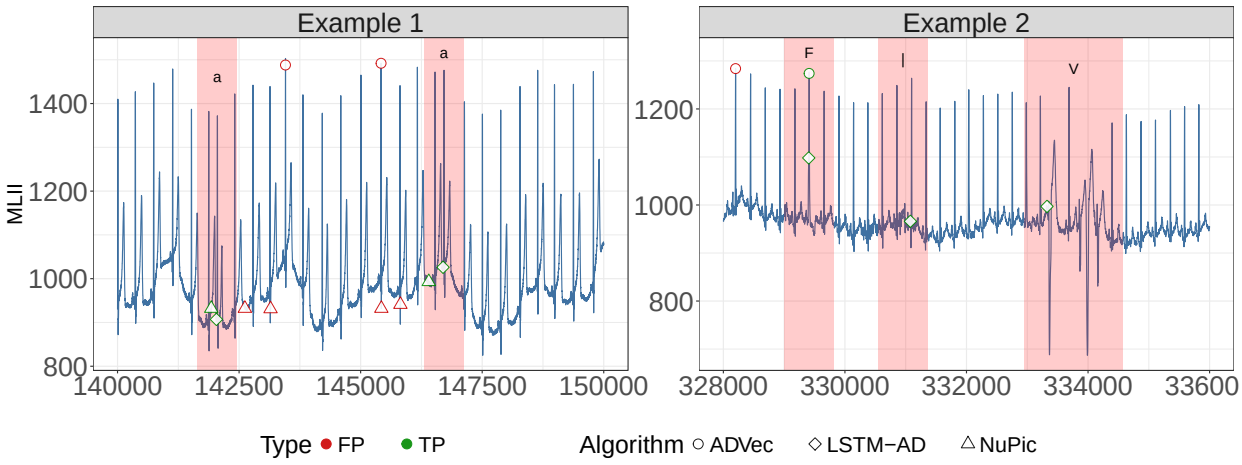

Figure 6.6: Subsets of two example time series taken from the MIT-ECG data with the anomalies detected by the algorithms LSTM-AD, NuPIC, and ADVec. The red rectangles in the plot indicate the true anomaly windows. Green colors indicate True-positives while False-positives are colored red.

Table 6.3: Comparison of the results on the ECG-13 data, with and without the window-based error correction, as described in subsection 6.2.4. The last column F_1(Corr) is copied from Table 6.2. The remaining columns depict the quantities obtained if no window-based error correction is applied (thresholds chosen such that F_1 is maximized).

ECG No.	threshold	TP	FN	FP	Prec	Rec	F_1	F_1(Corr)
1	20.60	12	22	23	0.34	0.35	0.35	0.52
2	5.83	2	4	2	0.50	0.33	0.40	0.75
4	7.03	1	1	0	1.00	0.50	0.67	0.67
9	16.83	13	15	10	0.57	0.46	0.51	0.65
10	16.21	28	12	2	0.93	0.70	0.80	0.84
11	40.60	1	0	0	1.00	1.00	1.00	1.00
12	28.48	1	1	1	0.50	0.50	0.50	1.00
13	93.83	5	1	130	0.04	0.83	0.07	1.00
15	87.97	2	3	5	0.29	0.40	0.33	0.67
17	35.90	1	0	38	0.03	1.00	0.05	1.00
20	25.10	1	0	1	0.50	1.00	0.67	1.00
21	32.31	1	0	0	1.00	1.00	1.00	1.00
22	77.33	2	1	37	0.05	0.67	0.10	0.46
mean	–	5	4	19	0.52	0.67	0.50	0.81
Σ	–	70	60	249	0.22	0.54	0.31	0.70

nominal and anomalous patterns in the displayed ECG data. Only after applying our approach, a better signal-to-noise ratio is established, which perfectly separates the anomalies from the nominal points. For most of the investigated ECG readings, we found that the window-based error correction significantly improves the signal-to-noise ratio in the Mahalanobis distance. Table 6.3 shows: The average F_1-score increases from F_1=0.50 when no window-based error correction is applied to F_1=0.81.

In Table 6.4, various measures are listed for the individual anomaly classes. The anomaly types a, F, and x, can all be detected by LSTM-AD. Also, for the anomaly class V, a high recall can be achieved. However, the two remaining types appear to be hard to detect for our algorithm.

Table 6.4: Various metrics for 5 different anomaly classes. The threshold was tuned individually for each time series by maximizing the F_1-score.

	TP	FN	Prec	Rec	F_1	FPR $*10^5$	PLR $/10^5$
A	23	21	0.65	0.52	0.58	0.14	3.64
V	44	9	0.73	0.83	0.78	0.19	4.36
|	9	8	0.63	0.53	0.58	0.06	8.49
a	6	0	0.79	1.00	0.88	0.02	53.44
F	2	0	0.65	1.00	0.79	0.01	80.16
x	8	0	0.73	1.00	0.85	0.03	29.15

Table 6.5: Same as Table 6.2. However, now the stacked LSTM is trained only with nominal sequences. Hence, training sequences containing anomalies are removed. The results slightly better in the F_1-score in comparison to Table 6.2.

ECG No.	threshold	TP	FN	FP	Prec	Rec	F_1	FPR	TPR/FPR
1	60.60	18	16	2	0.90	0.53	0.67	0.31	1.72
2	87.84	5	1	2	0.71	0.83	0.77	0.31	2.71
4	8.78	1	1	0	1.00	0.50	0.67	0.00	Inf
9	30.52	14	14	0	1.00	0.50	0.67	0.00	Inf
10	18.80	32	8	5	0.86	0.80	0.83	0.77	1.04
11	68.91	1	0	0	1.00	1.00	1.00	0.00	Inf
12	60.03	2	0	0	1.00	1.00	1.00	0.00	Inf
13	92.32	6	0	1	0.86	1.00	0.92	0.15	6.50
17	84.21	1	0	0	1.00	1.00	1.00	0.00	Inf
21	28.57	1	0	0	1.00	1.00	1.00	0.00	Inf
22	123.27	3	0	5	0.38	1.00	0.55	0.77	1.30
15	130.13	4	1	2	0.67	0.80	0.73	0.31	2.60
20	50.11	1	0	0	1.00	1.00	1.00	0.00	Inf
mean	–	6	3	1	0.88	0.84	0.83	0.20	Inf
Σ	–	89	41	17	0.84	0.68	0.75	0.20	3.40

Our method is unsupervised in the sense that no anomaly class labels are needed for training the algorithm. In fact, it is even not necessary that anomalous events are present at all in the training data, i.e., our algorithm can operate as a one-class classifier. We checked this by repeating the experiment leading to Table 6.2, but this time removing all data around anomalies during LSTM training. When using the trained model as an anomaly detector on all data, it worked as accurately as in Table 6.2, the mean F_1-score being now $F_1 = 0.83$. The results for this experiment are listed in Table 6.5.

6.5 Conclusion & Possible Future Work

We have presented a fully unsupervised method to detect anomalies in ECG readings. This method relies on an accurate LSTM predictor to learn the nominal behavior of the ECG for several prediction horizons. For the prediction errors, a mean vector and covariance matrix is estimated. Anomalous events have a high probability of being detected through an unusually high Mahalanobis distance.

We achieve for the ECG readings these high precision, recall, and F_1-values (on average higher than 80%, see Table 6.2), if we tune the final threshold for the Mahalanobis distance such that F_1 is maximized. Admittedly, this last step is not unsupervised since we calculate the confusion matrix based on the true anomaly labels.

We have shown that the window-based error correction is essential to achieve a Mahalanobis distance graph where the anomaly cases clearly stand out (Fig. 6.3 and Table 6.3).

Our LSTM-AD algorithm outperformed two state-of-the-art anomaly detection algorithms (NuPIC and ADVec) on the investigated ECG readings, achieving higher precision and recall over a large range of anomaly thresholds.

We have presented initial results of an unsupervised anomaly detector suitable for ECG readings or other quasi-periodic signals in this work. The results are encouraging, but there is still room for improvement. Possible future works include:

- Addressing the problem of concept-drifts in the ECG readings, e. g. by applying suitable preprocessing steps to reduce the effect of signal quality changes.
- Enrich the model by multi-resolution approaches to span larger prediction horizons on a coarser scale.
- Finding better hyper-parameters for the LSTM model in an automized fashion.

In the next chapter, we will introduce a temporal convolutional network autoencoder (TCN-AE), an architecture based on dilated convolutional layers, and compare TCN-AE to the LSTM-AD algorithm of this chapter on the Mackey-Glass Anomaly Benchmark (MGAB) and the extended ECG-25 benchmark.

Chapter 7

The Temporal Convolutional Autoencoder TCN-AE

7.1 Introduction

In this chapter, we present TCN-AE, a **t**emporal **c**onvolutional **n**etwork **a**utoencoder based on dilated convolutions. Similar to the other anomaly detection algorithms discussed in this thesis, TCN-AE trains completely unsupervised. In contrast to SORAD and LSTM-AD, TCN-AE is a reconstruction-based algorithm instead of a prediction-based one.

In the first part of the chapter, we describe a relatively simple baseline version of the algorithm (baseline TCN-AE) and demonstrate its capabilities by comparing it to other state-of-the-art algorithms on a Mackey-Glass (MG) anomaly benchmark (MGAB). Furthermore, we will see that our autoencoder is capable of learning interesting representations in latent space.

In the second part of this chapter, we analyze the architecture of the baseline TCN-AE, and we propose several enhancements that lead to TCN-AE in its current form. The final algorithm shows its efficacy on the real-world MIT-BIH [52, 112, 113] data of patients with cardiac arrhythmia, which we previously used in Chapter 6. Contrary to Chapter 6, we now consider the data of 25 patients (ECG-25) instead of 13 (ECG-13), almost doubling the amount of the benchmark data. TCN-AE also significantly outperforms several other unsupervised state-of-the-art anomaly detection algorithms on this comprehensive anomaly benchmark. Moreover, we investigate the contribution of the individual enhancements and show that each new ingredient improves the overall performance on the investigated benchmark.

As we have already seen in the previous chapter, it is rather challenging to learn the underlying structure of a system's normal behavior, especially if one has to deal with periodic or quasi-periodic signals with complex temporal patterns. In such environments, anomalies may be hard-to-detect deviations from the regular recurring pattern. Especially, for prediction-based algorithms it is challenging to predict small, but random variations in frequency/periodicity in quasi-periodic time series. ECG signals are a good example for such time series. Due to the heart rate variability (HRV), which is the variance in time between two heartbeats, it is rather difficult for a prediction-based algorithm to accu-

rately predict the exact position (in time) of, for example, the next R-peak (typically, the most characteristic peak in the signal). Even small shifts in predicted and actual R-peak cause large prediction errors which might falsely be interpreted as anomalous. To account for this variability in ECG signals, we introduced the window-based error correction for LSTM-AD in the previous chapter. Another way to approach this problem could be to use reconstruction-based (using an encoder and decoder architecture) anomaly detection algorithms instead of prediction-based ones. The advantage of reconstruction-based algorithms is that they do not require a time series to be predictable (in the sense of forecastable). Instead, they have a bottleneck which forces them to learn the underlying patterns of (nominal) time series. Most approaches found in the literature (see Chapter 3) are based on (temporal) autoencoders which take short sub-sequences from a time series, encode them into a latent-space vector, and attempt to reconstruct the sub-sequence based on the latent vector. However, these kind of reconstruction-based algorithms have the disadvantage that they are commonly limited to detecting anomalies in relatively short sub-sequences.

In this chapter, we propose a novel autoencoder architecture for sequences (time series), called TCN-AE, which is inspired by temporal convolutional networks [13] and shows its efficacy in unsupervised learning tasks. Contrary to other approaches, it does not encode and reconstruct short sub-sequences. Instead, it can compress a whole time series into a significantly shorter one, before reconstructing the original time series again. TCN-AE uses so-called dilated convolutional layers to naturally create a large receptive field and process a time series signal at different time scales. It consists of two parts, an encoder, and a decoder, which are both trained simultaneously and learn to find a compressed representation of the input time series (encoder) and reconstruct the original input again (decoder). Initially, we study a baseline version of TCN-AE. Our experiments show that the baseline architecture can learn interesting representations of sequences in latent space. When trained on (mostly) normal data, the approach can also be used for anomaly detection tasks by using reconstruction errors to predict anomaly scores. Only a small fraction of labeled data is needed to find a suitable threshold for the anomaly score. This can also be fine-tuned in operation with an already trained model.

For the initial benchmarking and comparison of our baseline algorithm, we use a synthetic benchmark based on Mackey-Glass (MG) time series [106]. In its current form, the Mackey-Glass Anomaly Benchmark (MGAB) consists of 10 MG time series in which anomalies were inserted using a clearly defined procedure. Although the anomalies are inserted synthetically, spotting them is rather difficult for the human eye. Due to the structured insertion process and the clear labeling of nominal and anomalous data, no domain knowledge is required to label the data correctly.

In the second part of the chapter we propose several enhancements for the baseline TCN-AE architecture. Then, we test our model on the more challenging ECG-25 dataset (introduced in Section 2.3.3), an anomaly benchmark consisting of 25 electrocardiogram time series with a length of half an hour.

We formulate the following research questions for this chapter:

- Can *unsupervised* deep learning models learn to detect anomalies?
- Which models are best to process the complex and long-range temporal patterns observed in periodic or quasiperiodic time series data?

The key findings of the research described in this chapter can be formulated as follows:

- Under certain (mild) assumptions, it is possible to train unsupervised Deep Learning (DL) models for anomaly detection. The novel autoencoder approach is essential for achieving this.
- It is essential to process the data on different time scales (like TCN and wavelets) and utilize the information from different time scales in the anomaly detection process.
- TCN-AE outperforms all other considered state-of-the-art algorithms by more than 10% on the ECG-25 benchmark (Table 7.6).

Several earlier works inspired the TCN-AE architecture that is presented in this chapter: While Holschneider et al. applied dilated convolutions in their "algorithme à trous" algorithm in the field of wavelet decomposition already in 1990 [65], more recently, they have also been applied to deep learning architectures, where the parallels to the non-decimating/stationary discrete wavelet transform (DWT) are still apparent: van der Oord et al. [124] introduced the WaveNet architecture, which uses dilated convolutions for the generation of raw audio. Yu & Koltun [182] successfully employed dilated convolutions to the task of semantic image segmentation. Later, Bai et al. [13] proposed a more general temporal architecture for sequence modeling, which they named temporal convolutional network (TCN). Our work is built upon the work of Bai et al. To the best of our knowledge, there is no earlier work that employs TCNs in an autoencoder-like architecture. We only found one approach for time series anomaly detection that is based on TCNs [61]. However, it does not use autoencoders. Its general idea is more similar to [107] and [161], which use forecasting errors as an indication for anomalous behavior. Further related work concerened with electrocardiography and anomaly detection in ECG signals is described in the previous Chapter 6.

The rest of this Chapter is organized as follows: Section 7.2 introduces the dilated convolution operation while Section 7.3 presents the baseline TCN-AE architecture and describes several experiments with Mackey-Glass time series. Section 7.4 proposes several enhancements for TCN-AE and discusses the results for extensive experiments on the ECG-25 dataset. Finally, we conclude this work in Section 7.5.

7.2 Methods

In the following, we introduce the Temporal Convolutional Network Autoencoder (TCN-AE), describe its main components, and discuss a few of its properties and application areas for time series analysis. We will start with a baseline architecture and then later successively

add several enhancements to this architecture. As the name suggests, TCN-AE is a convolutional neural network architecture. Convolutional neural networks (CNNs) are broadly and with great success used in computer vision applications, where other fully connected/dense architectures commonly suffer from the curse of dimensionality. Convolutional nets have several useful properties such as translation invariance, weight (parameter) sharing, and computational efficiency, making them especially beneficial for computer vision tasks such as image recognition, segmentation, or object detection. Their properties are also helpful for time series processing, where typically 1D-convolutions are employed. Several architectures, such as WaveNet [124], or temporal convolutional networks [13] take advantage of the convolutional approaches developed for computer vision and adopt some ideas into the time domain.

7.2.1 Intuition

TCN-AE is designed to learn how to encode or compress a sequence into a significantly shorter sequence (using an encoder network) and subsequently reconstruct the original sequence from the compressed representation (using a decoder network).

The central idea is to create a bottleneck in the architecture that forces the network to identify and capture the most useful (temporal) patterns in the raw input data and translate them into efficient encodings. The encoded data should contain all the essential information, allowing an accurate reconstruction of the original input. Ideally, the autoencoder learns to ignore signal noise, redundancies, and other irrelevant information.

Conceptually, TCN-AE is similar to other classical (deep) autoencoder architectures. The most common autoencoder architectures encode fixed-sized inputs into a latent space representation and then use the latent variables to reconstruct the original input. Similarly, the TCN-AE encodes sequences along the temporal axis into representation and then attempts to reconstruct the original sequence. However, it differs from regular autoencoders in so far in that it replaces the fully connected/dense layers with dilated 1D-convolutional layers. Thus, the network can consider temporal relationships in the data more naturally and flexibly regarding variable-size inputs. Furthermore, the temporal receptive field of TCN-AE can be easily scaled and grows exponentially with an only linear increase in the number of weights, which is especially important for time series containing long intricate temporal patterns. Another advantage over other autoencoders is that TCN-AE (due to the shared weights) potentially has fewer weights than dense AE architectures.

This idea can be used for several applications. The applications of (temporal) autoencoders are very diverse. In this work, we will focus on anomaly detection in time series. Other applications could be time series (sequence) compression or representation learning [166], as we will investigate in Section 7.3.2.2.

7.2.2 Dilated Convolutions

Convolutional layers in neural networks comprise digital filters, which remove or amplify individual components (frequencies) in a presented signal (for example, an image or a time series). Formally, the filtering process can be described by the convolution operation. For a one-dimensional signal $x[n]$ ($x[n]$ being the n-th element in the signal), $x : \mathbb{T} \to \mathbb{R}$, where $\mathbb{T} = \{0, 1, \ldots, T-1\}$, the convolution with a (finite impulse response) filter $h[n]$, $h : \{0, 1, \ldots, k-1\} \to \mathbb{R}$ is usually defined as:

$$y[n] = (x * h)[n] = \sum_{i=0}^{k-1} h[i] \cdot x[n-i], \tag{7.1}$$

where $y[n] \in \mathbb{R}$ is the output of the filter, $h[i] \in \mathbb{R}$ is the i-th filter weight and k specifies the length of the filter. The convolution operation can be thought of as sliding a window of length k, which contains the filter weights $h[i]$, over the input sequence $x[n]$ and computing a weighted average of $x[n]$ with the weights $h[i]$ in each time step. The resulting output signal is one-dimensional and of length $T - k + 1$. In order to obtain an output signal of the same length, the input sequence is usually padded with zeros before applying the filter. Since the filter is only slid along the time axis, the operation is usually referred to as one-dimensional convolution. The behavior of the filter is determined by $h[n]$ (e.g., low-pass or high-pass characteristics), and the central idea of convolutional neural networks is not to pre-determine $h[n]$ but rather to learn suitable filter weights based on the learning task.

Convolutional layers in neural networks usually deal with multivariate input signals $\mathbf{x}[n]$ of dimension d, with $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$. In this case, each dimension $\mathbf{x}_j[n]$ is convolved separately with its own sub-filter $\mathbf{h}_j[n]$, $\mathbf{h} : \{0, 1, \ldots, k-1\} \to \mathbb{R}^d$, and $y[n]$ (remaining one-dimensional) is a dot product:

$$y[n] = (\mathbf{x} * \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}[n-i]. \tag{7.2}$$

In contrast to the regular convolution operation (as specified above), the dilated convolution [182] has an additional parameter, the so called dilation rate $q \in \mathbb{N}$. It defines how many elements in the input signal $\mathbf{x}[n]$ are skipped between filter tap $\mathbf{h}[i]$ and filter tap $\mathbf{h}[i+1]$. The dilated convolution is written as:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}[n - q \cdot i]. \tag{7.3}$$

For $q = 1$ the original convolution operation is obtained.

In many applications also acausal convolutions are used. In this case, future values of a sequence $x[n]$ will be processed to generate output $y[n]$:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}\big[n - q \cdot \lfloor i - k/2 \rfloor\big] \tag{7.4}$$

In this work, we experimented with causal and acausal convolutions for TCN-AE and found acausal convolutions to produce slightly better results for the investigated anomaly detection tasks (MGAB & ECG-25). Note that this comes at the cost of slight delays in online settings.

When using dilated convolutions, one has to consider that a few properties of the frequency response of the system also change: the system function of a regular convolution ($q = 1$) for a one-dimensional input signal is given as

$$H(z) = \sum_{m=0}^{k-1} h[m] \cdot z^{-m}.$$

If we evaluate the system function $H(z)$ on the unit circle described by $z = \mathrm{e}^{j\hat{\omega}}$, we get the frequency response of the system. The parameter $\hat{\omega} = \frac{\omega}{f_s}$ is the so-called normalized radian frequency, which is obtained by dividing the radian frequency ω by the sampling frequency f_s. In this case, the Nyquist frequency is π. For the dilated convolution, the system function becomes:

$$H(z^q) = \sum_{m=0}^{k-1} h[m] \cdot z^{-q \cdot m}. \tag{7.5}$$

This means that a dilation rate $q > 1$ implicitly increases the filter order and adds extra poles and zeroes to the system. For example, if $q = 2$, then the filter order is doubled from $M = k - 1$ to $M = 2(k - 1)$ and at the same time $2(k - 1)$ zeroes/poles are added to the system. For $q = 4$, the filter order is multiplied by a factor of 4 and so on. Effectively, the frequency response becomes "sharper" for larger q and the filter is more sensitive to small changes in the frequency. In fact, the frequency response of the filter has a periodicity of $\frac{2\pi}{q}$. This can be easily verified if we insert $z = \mathrm{e}^{j(\hat{\omega} + \frac{2\pi}{q})}$ into Eq. (7.5):

$$\begin{aligned} H(z^q)\Big|_{z=\mathrm{e}^{j(\hat{\omega}+\frac{2\pi}{q})}} &= \sum_{m=0}^{k-1} h[m] \cdot \left(\mathrm{e}^{j(\hat{\omega}+\frac{2\pi}{q})}\right)^{-q \cdot m} \\ &= \sum_{m=0}^{k-1} h[m] \cdot \mathrm{e}^{-j\hat{\omega}qm} \mathrm{e}^{-j(2\pi)m} \end{aligned}$$

$$= \sum_{m=0}^{k-1} h[m] \cdot \mathrm{e}^{-j\hat{\omega}qm}$$
$$= H(z^q)\Big|_{z=\mathrm{e}^{j\hat{\omega}}}$$

Figure 7.1 illustrates the discussed properties of dilated convolutions for several filters that TCN-AE (described below) learned during its training. One can clearly see how the frequency response evolves for increasing q. Note that the periodicity of the frequency responses, as shown in Figure 7.1, are solely defined by the dilation rate q. For example, this means that for $q = 8$ we will – independently from the filter length and the filter weights – always have a periodicity of $\frac{\pi}{4}$ for the frequency response, resulting in a "rougher" landscape than for $q = 1$. This realization was one of the reasons for introducing skip connections in the architecture in order to reuse the outputs of different dilated convolutional layers (as discussed in more detail in Section 7.4.1.1).

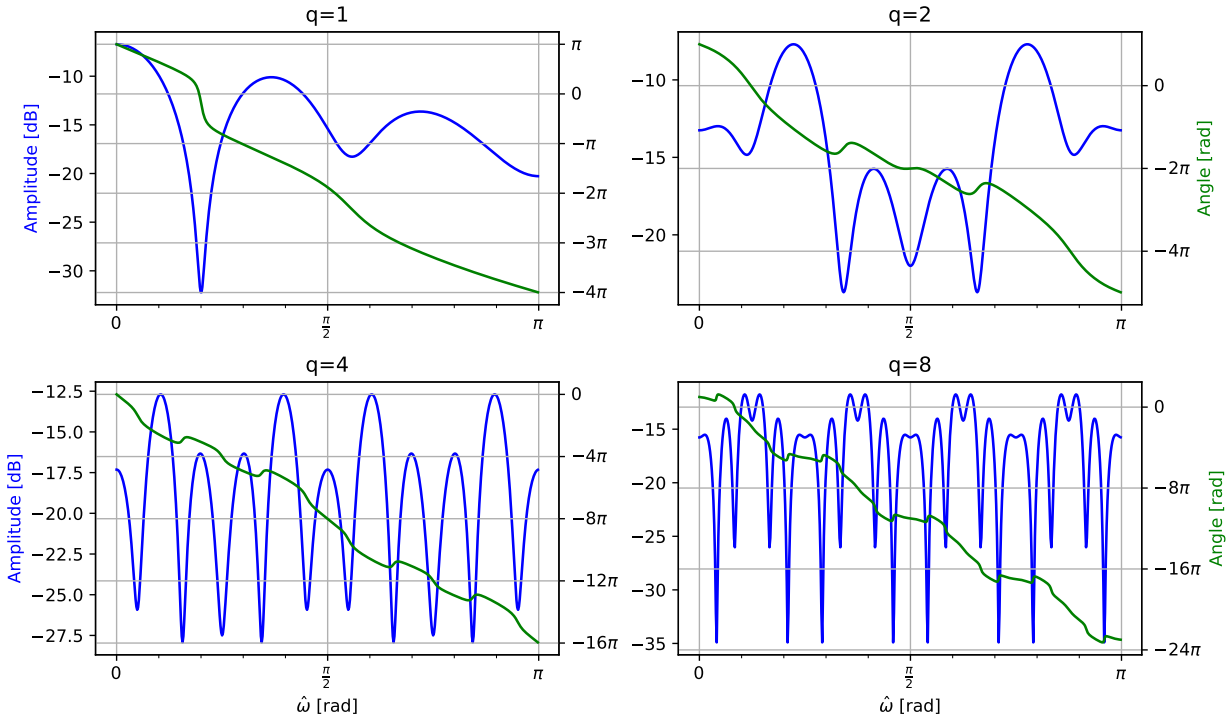

Figure 7.1: Illustration of the frequency responses of several filters taken from the first four dilated convolutional layers of TCN-AE (which is described in more detail below). Several properties, mentioned in Section 7.2.2, can be observed: For example, with increasing dilation rate q, the amplitude of the frequency response becomes "sharper" and the filter is more sensitive to small changes in the frequency of the input signal. This is because dilation rates larger than $q > 1$ implicitly increase the order of the filter, and additional zeros/poles are introduced in the complex frequency-domain representation of the filter. Also, the phase (angle) of the displayed filters is as expected not linear since the symmetry of the filter weights is generally not given in a dilated convolutional layer.

7.2.3 Dilated Convolutional Layers in Neural Networks

The previous section described how a one-dimensional output signal $y[n]$ is computed using a filter. In practice, a convolutional layer is typically comprised of many discrete filters, and the individual outputs $y[n]$ are stacked into a so-called feature map. If a signal $\mathbf{x}[n]$ of length T_{train} is passed through a convolutional layer with n_{filters} filters, the resulting feature map has the dimension $T_{\text{train}} \times n_{\text{filters}}$ (for a padded signal). The weights $\mathbf{h}[i]$ of each filter are considered learnable parameters, commonly trained using variants of the back-propagation algorithm.

Many neural network architectures for sequence modeling (e.g., [13, 124]) utilize dilated convolutions to create a hierarchical temporal model with a large receptive field, capable of learning long-term temporal patterns in the input data. The main idea is to build a stack of dilated convolutional layers, where the dilation rate increases with each added layer. A common choice is to start with a dilation rate of $q = 1$ for the first layer of the network and double q with every new layer. With this approach, we can increase the receptive field of the model exponentially. In general, the receptive field r for the causal and acausal case is given by:

$$r_{causal} = k \cdot 2^{L-1}, \tag{7.6}$$

$$r_{acausal} = \lfloor k/2 \rfloor \cdot (2^{L+1} - 2) + 1, \tag{7.7}$$

where $L > 0$ is the number of layers. If, for example, we build a stack of $L = 5$ dilated convolutional layers with a kernel size of $k = 3$, the receptive field's size will be $3 \cdot 2^4 = 48$ for the causal case and $2^6 - 1 = 63$ for the acausal setting. The size of the receptive field should be considered when choosing the length of the training sequences. For example, the receptive field should not be larger than the length of the training sequences.

In summary, a convolutional layer can be mainly described by three parameters: The dilation rate q, the number of filters n_{filters}, and the kernel size (filter length) k. A convolutional layer maps an input sequence $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$ to an output sequence $\mathbf{y} : \mathbb{T} \to \mathbb{R}^{n_{\text{filters}}}$. Note, that the shape of the output does not depend on k.

Relation between Dilated Convolutions and the DWT The non-decimating discrete wavelet transform (DWT) is, in some sense, related to dilated convolutional neural network architectures. The regular DWT decomposes a time series into so-called approximation and detail coefficients. By repeated filtering of the input with low-pass and high-pass filters, one obtains a hierarchical representation of the original signal on different frequency scales.

While the regular DWT downsamples (decimates) the signal after every low-pass filter by a factor of two, the non-decimating DWT removes all downsampling units. In turn, the filters have to be dilated. The dilation rate (which is a power of two) specifies the gaps between the filter taps. The non-decimating DWT is usually used in applications where one wants to achieve translation invariance (at the cost of redundancy). Holschneider et al. [65]

proposed an efficient algorithm for computing the non-decimating DWT using dilated convolutions. Current deep learning architectures [124, 182, 13] based on dilated convolutional layers are inspired by the earlier work of Holschneider et al. Dilated convolutional nets also repeatedly filter a signal (e.g., time series or image) in a stack of convolutional layers. The dilation rate q is usually doubled with every further layer.

There are also apparent differences: (a) The DWT filter weights depend on the mother-wavelet choice and are fixed, while the weights of convolutional layers are learnable parameters. (b) The DWT does not use non-linear activation functions such as rectified linear units (ReLU).

The baseline TCN-AE consists of two temporal convolutional neural networks (TCNs) [13], one for encoding and one for decoding. Additionally, a downsampling and upsampling layer are used in the encoder and decoder, respectively. The individual components will be described in more detail in the following.

7.2.4 Temporal Convolutional Networks

The temporal convolutional network (TCN) [13] is inspired by several convolutional architectures [36, 48, 73, 124], but differs from these approaches insofar as it combines simplicity, auto-regressive prediction, residual blocks, and very long memory. Essentially, a TCN is a stack of n residual blocks. Each block consists of two serial sub-blocks, and each sub-block is comprised of the following layers: a dilated convolutional layer, followed by a weight normalization layer [145], a ReLu activation function [116], and a spatial dropout layer [152]. Furthermore, a skip (residual) connection [60] bypasses the residual block and is added to the residual block's output. A TCN is mainly described by three parameters: a list of dilation rates $(q_1, q_2, \ldots, q_L)$, the number of filters n_{filters}, and the kernel size k. The output of each residual block and the final output is a sequence $\mathbf{y} : \mathbb{T} \to \mathbb{R}^{n_{\text{filters}}}$. A full description of TCN would be out of scope for this chapter. The reader is referred to [13] for the details.

7.2.5 Unsupervised Anomaly Detection with TCN-AE

A natural application of TCN-AE is the anomaly detection in time series. Although we have not yet discussed the architecture of TCN-AE in detail, we can already describe how its reconstruction errors are used to detect anomalous patterns. Due to the bottleneck in the architecture, the training procedure forces TCN-AE to learn compressed encodings of the input sequences, which capture the underlying structure of the data and allow accurate reconstruction. Intuitively, we expect that TCN-AE reconstructs recurring nominal patterns in a time series with only small errors. It focuses on minimizing the reconstruction error of the nominal data that are in the vast majority during training. On the other hand, when TCN-AE observes patterns that significantly differ from the norm, we expect higher reconstruction errors.

To discover abnormal behavior, we slide a window of length ℓ over the reconstruction error and collect the ℓ-dimensional vectors in an error matrix $\mathbf{E}$. The purpose of the sliding

window is to smoothen noisy events that might occasionally appear. The error matrix $\mathbf{E}$ is passed to the outlier detection algorithm, which identifies unusual points in the ℓ-dimensional space. The outlier detection algorithm outputs an anomaly score, which is later thresholded. We experimented with different outlier detection algorithms and found that a simple approach based on the Mahalanobis distance (line 15 in Algorithm 8) delivers the best results. An advantage of the Mahalanobis distance is that it is parameter-free and does not require any particular assumptions about the data distribution (such as normality). The Mahalanobis distance only requires the invertibility of the covariance matrix. However, in practice, there are rarely situations where the covariance matrix is non-invertible. We summarize the anomaly detection algorithm for TCN-AE in Algorithm 8.

Note that although we train TCN-AE with the complete time series, the overall anomaly detection algorithm consisting of TCN-AE and Mahalanobis distance calculation is entirely unsupervised. The training procedure does not pass anomaly labels to the algorithm at any time. Only for selecting an appropriate anomaly threshold on the Mahalanobis distance, we permit all algorithms to use 10% of the anomaly labels, as described later in Sec. 7.4.2.2.

7.3 A Baseline Version of TCN-AE

7.3.1 The Baseline TCN-AE Architecture

For our initial experiments, we use TCN as a building block for a baseline temporal autoencoder, referred to as baseline TCN-AE. In later sections, we will modify the baseline TCN-AE, add further enhancements to the architecture, and analyze their contribution to the final TCN-AE architecture. The baseline TCN-AE consists of an encoder network $\text{enc}(\cdot)$ and a decoder network $\text{dec}(\cdot)$.

The encoder $\text{enc}(\cdot)$, shown in Fig. 7.2, left, attempts to generate a compact representation that captures the main characteristics of the input sequences and allows a reasonably good reconstruction in later steps. In order to learn the important features in a sequence, it is necessary to identify short-term as well as long-term patterns. The encoder takes an input sequence, passes it through a TCN network, reduces the dimension of the feature map by applying a 1×1 convolutional layer[1] [96, 156] and finally down-samples the series along the time axis by a specified factor using an average-pooling layer. It does so by averaging groups of size s along the time axis. The number of filters c in the 1×1 convolution layer specifies the dimension of the encoded representation and the sample rate s determines the factor, by which the length T of the series is reduced. Hence, the original input $\mathbf{x}[n]$ will be compressed into an encoded representation $\mathbf{H}[n] = \text{enc}(\mathbf{x}[n])$, where $\mathbf{H} : \{0, 1, \ldots, T/s - 1\} \to \mathbb{R}^c$.

The decoder $\text{dec}(\cdot)$, shown in Fig. 7.2, right, attempts to reconstruct the original input sequence, using the output of the encoder as input. First, the length of the original series has to be restored. We use a simple sample-and-hold interpolation for this purpose, which

[1] A 1×1 convolution is a weighted average over all feature maps, taken at every time point. The weights are learnable parameters.

Algorithm 8 General anomaly detection algorithm using the TCN-AE architecture. The estimation of $\bar{\mathbf{x}}$ and $\mathbf{\Sigma}$ might also have to be computed in batches according to the method described in Sec. B.2.1.

Adjustable parameters:
  $\mathcal{M}_\tau$: anomaly threshold (see Sec. 7.4.2.1), ℓ: error window length
  T_{train}: length of training sub-sequences, B: batch size

function ANOMALYDETECT($\mathbf{x}[n]$) ▷ $\mathbf{x} : \mathbb{T} \to \mathbb{R}^d$, $\mathbb{T} = \{0, 1, \ldots, T-1\}$
  Construct model TCNAE() and **Initialize** the trainable parameters
  $\mathbf{X}_{\text{train}} \leftarrow$ { Sub-sequences of length T_{train} taken from $\mathbf{x}[n]$ }
  for $\{1 \ldots n_{\text{epochs}}\}$ **do**
    TRAIN(TCNAE, $\mathbf{X}_{\text{train}}$) ▷ Train with random mini-batches of size B
  $\hat{\mathbf{x}}[n] \leftarrow$ TCNAE($\mathbf{x}[n]$) ▷ Encode and reconstruct $\mathbf{x}[n]$
  $\mathbf{e}[n] \leftarrow \mathbf{x}[n] - \hat{\mathbf{x}}[n]$ ▷ reconstr. error $\mathbf{e} : \mathbb{T} \to \mathbb{R}^d$
  $\mathbf{E}[n] \leftarrow$ SLIDINGWINDOW($\mathbf{e}[n]$, ℓ) ▷ $\mathbf{E} : \mathbb{T} \to \mathbb{R}^{\ell \times d}$
  $\mathbf{E}'[n] \leftarrow$ RESHAPE($\mathbf{E}[n]$) ▷ $\mathbf{E}' : \mathbb{T} \to \mathbb{R}^{\ell \cdot d}$
  $\boldsymbol{\mu}, \mathbf{\Sigma} \leftarrow$ ESTIMATE($\mathbf{E}'[n]$) ▷ Mean $\boldsymbol{\mu} \in \mathbb{R}^{\ell \cdot d}$, Cov. Mat. $\mathbf{\Sigma} \in \mathbb{R}^{\ell \cdot d \times \ell \cdot d}$
  $M[n] \leftarrow (\mathbf{E}'[n] - \boldsymbol{\mu})^\intercal \mathbf{\Sigma}^{-1} (\mathbf{E}'[n] - \boldsymbol{\mu})$ ▷ Mahalanobis distance
  $a[n] \leftarrow \begin{cases} 0 & \text{if } M[n] < \mathcal{M}_\tau \\ 1 & \text{else} \end{cases}$ ▷ Binary anomaly flags
  return $a[n]$ ▷ Return anomaly flag for each time series point

duplicates each point in the series s times. Subsequently, the upsampled sequence is passed through a second TCN block, which has the same structure as the TCN block in the encoder (but untied/independent weights). Finally, to restore the original dimension d, another 1×1-convolutional layer with d filters is used to obtain the reconstruction (the output) $\hat{\mathbf{x}}[n] = \text{dec}(\mathbf{H}[n])$, $\hat{\mathbf{x}} : \mathbb{T} \to \mathbb{R}^d$. The architecture of the baseline TCN-AE is depicted in Figure 7.2. Once TCN-AE is trained, the input sequence and its reconstruction will be used for detecting anomalies, as described in the next section.

7.3.2 Initial Experiments

7.3.2.1 Experimental Setup

Anomaly Detection Algorithms As before, all training algorithms are unsupervised, i. e. they do not need the true anomaly labels during the training process. Only in order

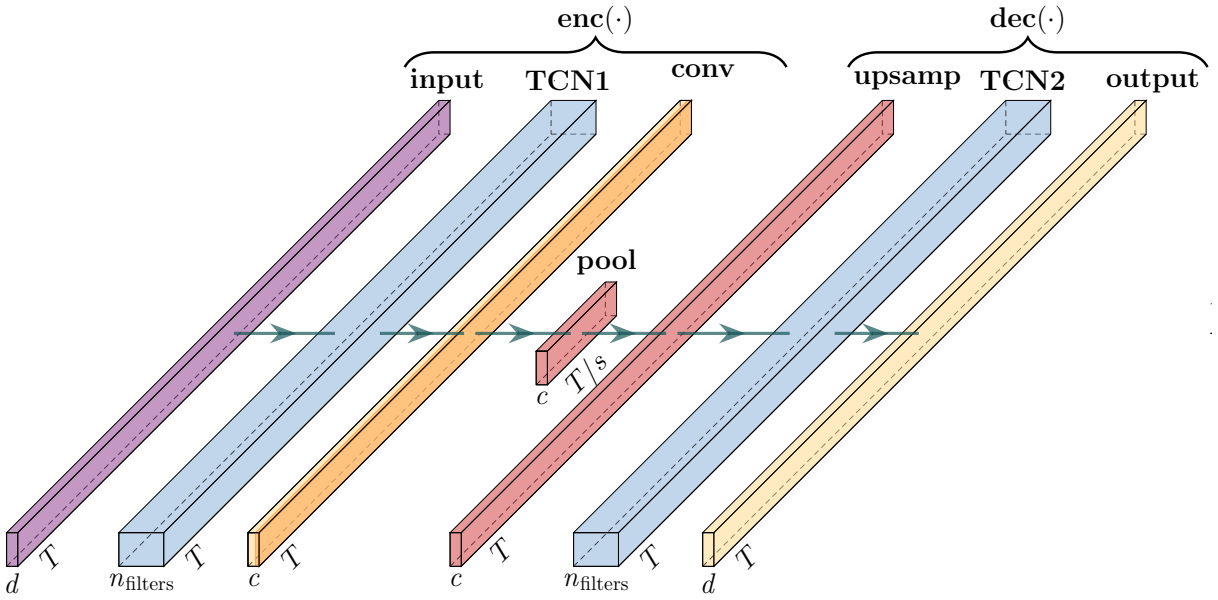

Figure 7.2: Architecture of the baseline TCN-AE as described in Section 7.3.1. The input of TCN-AE is a sequence $\mathbf{x}[n]$ with length T and dimensionality d. The layers "conv" and "output" are 1×1 convolutions with c and d filters, respectively. The TCNs have the dilation rates q . The layer "pool" downsamples the sequence by a factor s. *Configuration for MGAB:* $d = 1$, $c = 8$, $n_{\text{filters}} = 20$, and $s = 42$. *Configuration for ECG-25 benchmark:* $d = 2$, $c = 4$, $n_{\text{filters}} = 32$, and $s = 32$.

to find a suitable anomaly threshold, a small fraction of labels is used, as described below. Otherwise, the anomaly labels are only used at test time to evaluate the performance of the individual algorithms. In one run, each algorithm is trained for ten rounds: in the i-th round, the algorithms are trained on the i-th time series and evaluated on the time series $\{1, \ldots, 10\} \setminus \{i\}$. In total, we perform ten runs with different random seeds. In order to find suitable hyper-parameters for each algorithm, we use the HYPEROPT library [14] and optimize the F_1-score on a separate MG time series. For all neural networks, we use the Adam optimizer [82] to train the weights by minimizing the MSE loss. Additionally, all time series (having a dimension of $d = 1$) are standardized to zero mean and unit variance.

DNN-AE [46]: we use a PyTorch [130] implementation for the anomaly detection algorithm based on a deep autoencoder [58]. The algorithm requires several parameters, which we choose as follows: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 40$, sequence length $T_{\text{train}} = 150$ and a hidden size of $h = 10$ for the bottle neck (which results in a compression factor of $T_{\text{train}}/h = 15$ for each sequence). Finally, we set $\%_{Gaussian} = 1\%$, which specifies that 99% of the data is used to estimate a Gaussian distribution for the anomaly detection task.

LSTM-ED [108] is also implemented using PyTorch and uses the following parameter setting: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 20$, sequence length

$T_{\text{train}} = 300$, hidden size $h = 100$ and $\%_{Gaussian} = 1\%$. Both, encoder and decoder use a stacked LSTM network with two layers.

NuPIC [160]: Numenta's anomaly detection algorithm has a large range of hyper-parameters which have to be set. We use the parameters recommended by the authors in [89]. It is possible to tune the parameters with an internal swarming tool [3]. However, this is a time-expensive process which is not feasible for the large MGAB dataset.

LSTM-AD [161]: here we select the following parameters: batch size $B = 1024$, number of training epochs $n_{\text{epochs}} = 30$, and sequence length $T_{\text{train}} = 128$. A 2-layer LSTM network with 256 units in the first layer and 128 units in the second layer is used. The target horizons are chosen to be $H = (1, 3, \ldots, 51)$.

TCN-AE (baseline): The main TCN-AE parameters are given in Fig. 7.2. Additionally we use the sequence length $T_{\text{train}} = 1050$, batch size $B = 32$ and $n_{\text{epochs}} = 40$. For baseline TCN-AE, we use an existing TCN implementation in Keras [140]. The dilation rates are $q = (1, 2, \ldots, 16)$ and the kernel size of the TCNs is set to $k = 20$.

We determine this threshold for all algorithms as follows: A sub-sequence containing 10% of the data is taken, and the anomaly threshold is optimized on this short sequence, such that the F_1-score is maximized. The optimal threshold is then fixed for the complete time series, and the overall results are obtained. Since the results can vary depending on which sub-sequence is used for the threshold adjustment, we repeat the above procedure, similarly to k-fold cross validation, for ten different 10% sub-sequences of the considered time series and record the results for the ten different sub-sequences.

7.3.2.2 Learning Time Series Representations

In our first experiment, we want to assess the capabilities of the TCN-AE architecture to learn representations of time series. For this purpose, we train a TCN-AE model using many different MG time series with a varying time delay parameter τ. Ideally, TCN-AE should learn the main characteristics of the individual time series and find suitable compressed representations. In our experiment, we use TCN-AE on 10^5 different Mackey-Glass time series (10^4 for each τ in the range of $\tau = 11 \ldots 20$). Each time series of length 256 is encoded into a 2-dimensional compressed representation. The algorithm is trained in an unsupervised manner. Hence, τ is not passed to the algorithm at any time. Surprisingly, even with this large compression rate of 128, TCN-AE can find an interesting embedding for the MG time series, as depicted in Fig. 7.3 (top). For a certain τ, all samples are placed in *only one* connected cluster (except for a few satellites), and these clusters are mostly – with a few small exceptions – *non-overlapping*.

For comparison, we repeated the same experiment with the popular t-SNE [105] clustering algorithm. We executed t-SNE on a GPU with the help of a certain CUDA implementation [25]. We tried different parameter settings and finally fixed the perplexity parameter to 200, the learning rate to 10, and the number of iterations to 10^4. The results for t-SNE in Fig. 7.3 (bottom) indicate that it is not a trivial task to find suitable representations

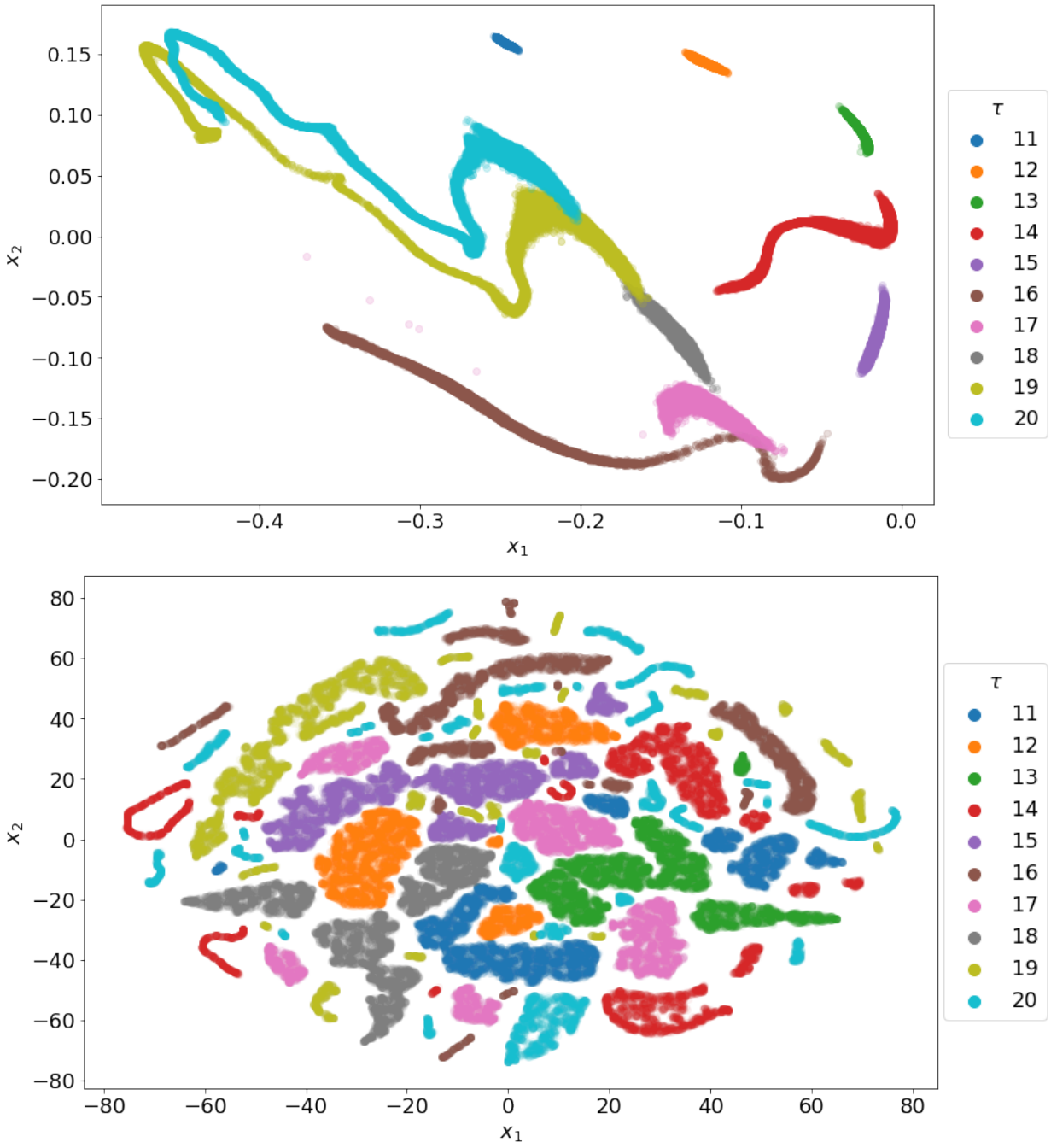

Figure 7.3: Top: 2d-representation of 10^5 (10^4 for each τ) different Mackey-Glass time series using TCN-AE (baseline). The (unsupervised) algorithm is capable of learning an encoding which separates the MG time series fairly well according to their τ value.
Bottom: 2d-representation of the same MG time series, but now using t-SNE [105] to find suitable encodings.

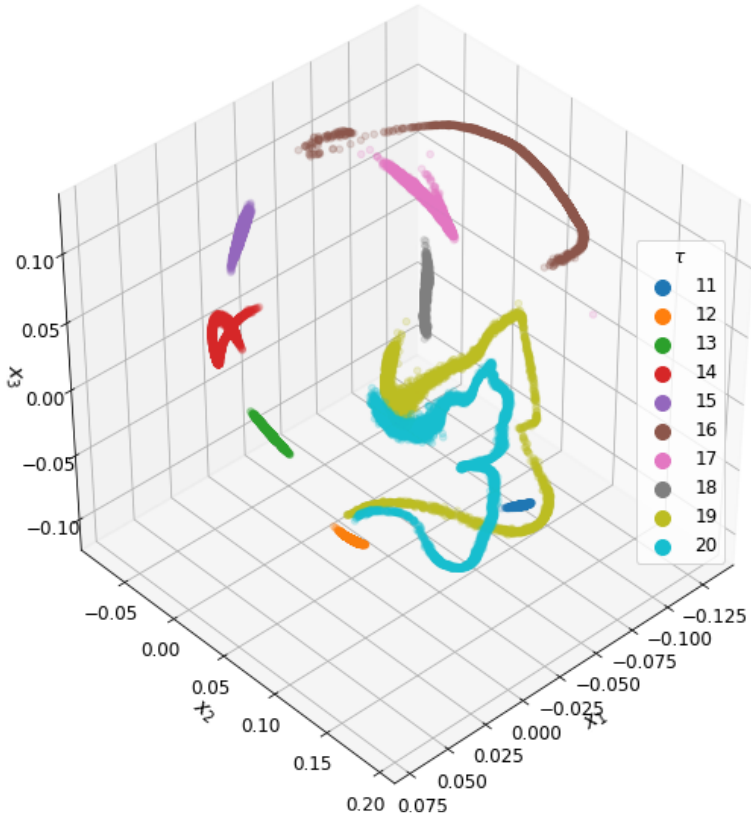

Figure 7.4: Similar to Fig. 7.3 (top). But now we encode each MG time series into a 3d-vector.

for MG time series. t-SNE has more difficulties in comparison to TCN-AE to cluster all sequences with a particular time delay parameter τ in only *one* connected region.

7.3.2.3 Anomaly Detection on the Mackey-Glass Anomaly Benchmark

In a second experiment, we compare TCN-AE to several state-of-the-art anomaly detection algorithms on the Mackey-Glass Anomaly Benchmark. For each algorithm, except NuPIC, ten runs were performed. Hence, for each algorithm and time series, ten different models are trained, and each model is evaluated on the other nine time series. NuPIC is entirely deterministic and does not require several runs. Additionally, as described in Section 7.3.2.1, the anomaly threshold for each algorithm and time series is tuned on ten different subsequences. We add up the TP, FN, and FP over all ten time series and summarize the results in Tab. 7.1. Up to 100 anomalies can be detected in total. We can see that the (deep) DNN-AE detects most of the anomalies (approx. 92), missing only about eight on average. However, this result is achieved at the expense of producing many false-positives. Overall, DNN-AE produces more than 60 false positives on average, while TCN-AE produces

less than one. Hence, DNN-AE achieves the highest recall among all algorithms but ranks only 3$^{\text{rd}}$ in F_1-score, due to its low precision. TCN-AE scores best in F_1-score and precision. NuPIC has the poorest performance in all measures.

Table 7.1: Results for MGAB. The results shown here (mean and standard deviation of 10 runs and ten sub-sequences, are for the sum of TP, FN, and FP over all ten time series. For each algorithm and time series, the anomaly threshold was tuned on 10% of the data using a cross-validation approach: the threshold is tuned on ten different 10%-sequences of the data.

Algorithm	TP	FN	FP	Precision	Recall	F_1-score
NuPIC [160]	3.00	97.00	132.00	0.02	0.03	0.03
LSTM-ED [108]	14.60 ± 5.86	85.40 ± 5.86	57.00 ± 20.43	0.21 ± 0.08	0.15 ± 0.06	0.17 ± 0.06
DNN-AE [58]	91.79 ± 1.22	8.21 ± 1.22	62.58 ± 13.65	0.60 ± 0.06	0.92 ± 0.01	0.72 ± 0.04
LSTM-AD [161]	88.80 ± 2.59	11.20 ± 2.59	0.62 ± 0.61	0.99 ± 0.01	0.89 ± 0.03	0.94 ± 0.01
TCN-AE [this work]	90.54 ± 1.72	9.46 ± 1.72	0.20 ± 0.47	1.00 ± 0.01	0.91 ± 0.02	0.95 ± 0.01

7.3.3 Discussion

The initial results that we obtained with our new TCN-AE architecture are promising. The learned representations (Fig. 7.3) on different MG time series appear to be useful and may reveal interesting insights. For anomaly detection, we achieve with TCN-AE and LSTM-AD the highest F_1-score on the non-trivial MG benchmark. Remarkably, all algorithms except NuPIC require many trainable weights. TCN-AE had 164 451 parameters, DNN-AE 241 526, LSTM-ED 244 101 and LSTM-AD 464 537. That is, the other high-performing algorithms require 50%–300% more trainable weights than TCN-AE.

Generally, we would expect TCN-AE to perform better than, for example, DNN-AE on tasks where a larger receptive field (memory) is required in order to detect anomalies since its hierarchical architecture allows to exponentially increase the receptive field while the number of parameters scales linearly.

Although the initial results of TCN-AE on MGAB look promising and although we could observe that the algorithm is capable of learning representations of MG time series, there are several limitations of the algorithm, which leave room for improvement and which we are planning to address in the future work: (1) Many parameters (approximately 160 000) are required for satisfactory MGAB results. TCN-AE's performance significantly drops if the number of filters n_{filters} and/or the kernel size k is reduced. (2) Baseline TCN-AE is somewhat sensitive towards the maximum dilation rate $q_{\max}$. For example, if we add a dilated convolutional layer with $q_{\max} = 32$ to both TCNs in the architecture, the performance significantly drops. (3) The net requires relatively many epochs until it learns the subtle differences between nominal and abnormal MG time-series patterns. TCN-AE requires ≥ 40 training epochs to learn to detect anomalies on the MG time series. It has to

be investigated if this holds for other (real-world) applications as well and if optimizations of the training-configuration might reduce the required epochs.

7.3.4 Summary

In this section, we proposed with TCN-AE an autoencoder architecture for multivariate time series. We evaluated it on various Mackey-Glass (MG) time series for two relevant tasks: representation learning and anomaly detection. The initial results on various Mackey-Glass (MG) time series are promising. TCN-AE could learn a very interesting representation in only two dimensions, which accurately distinguishes MG time series differing in their time delay values τ (Section 7.3.2.2). On the Mackey-Glass Anomaly Benchmark (MGAB), TCN-AE achieved better anomaly detection results than other state-of-the-art anomaly detectors (Section 7.3.2.3).

In the following section, we will address the limitations of baseline TCN-AE mentioned before and propose several extensions to improve the overall performance of TCN-AE.

7.4 An Improved TCN-AE Architecture

The goal of this section is to improve the baseline TCN-AE architecture based on the limitations discussed in the previous section. Overall, we suggest six modifications. We analyze, discuss, and compare the capabilities of the improved TCN-AE architecture on a challenging real-world HMS application, namely the detection of arrhythmias in electrocardiogram (ECG) signals of heart patients.

7.4.1 Enhancements of the Baseline TCN-AE

7.4.1.1 Skip Connections

While experimenting with the encoder and decoder's dilation rates, we noticed that the performance of the baseline TCN-AE is somewhat sensitive to the choice of the maximum dilation rate q_{max}. We believe that this problem occurs because only the TCN's final dilated convolutional layer is passed on to the following layer, i.e., the original TCN does not provide any mechanisms for feature reuse. However, especially for TCNs, which process a time series signal at different time scales, it might be detrimental to solely use the last dilated convolutional layer's output, since other time scales might also carry essential information. Instead, it should be possible to access the features at all time scales.

To provide for the possibility of feature reuse in TCN-AE, we add so-called skip connections to our architecture. A skip connection copies the output of a particular layer and concatenates it to the input of a subsequent layer of the network. In our setup, we use a concatenation layer in the end of encoder and decoder, which collects the outputs of all previous dilated convolutional layers.

In the encoder shown in Fig. 7.5, we add skip connections from every dilated convolutional layer to the encoder's bottleneck (after reducing the number of channels to 16), where the outputs of the individual layers are concatenated along the channel axis. The bottleneck reduces the number of channels of the concatenated outputs with a 1×1-convolution and downsamples the resulting signal to obtain a compressed encoding.

In the decoder shown in Fig. 7.6 we also place skip connections from each dilated convolutional layer to the output, where lastly, a 1×1 convolution reconstructs the time series with the original dimension d.

Relation to other Architectures Many modern DL architectures adopt skip connections. In ResNets [60], for example, shortcut connections perform an identity mapping, skipping one or more layers. Their outputs are then added to the skipped layers' outputs (not concatenated as in our approach). ResNets were among the first architectures that address the so-called degradation problem [60] (a problem observed in practice, where very deep neural networks surprisingly produce higher training errors than shallow nets) and have shown to improve the results on many problems.

In a DenseNet [66], each layer uses the output of all preceding layers as input and passes on its output to all subsequent layers. Due to this structure, many direct connections are necessary (in a network with L layers, there are $L(L+1)/2$ direct connections). Nonetheless, the authors could significantly reduce the number of required parameters in the overall network, since the number of filters in all layers could be decreased. DenseNets address similar problems as ResNets and are insofar more similar to our TCN-AE in that they also concatenate the feature maps of previous layers and do not add them (as in ResNets).

7.4.1.2 Dilation Rate Ordering

In the setup of the baseline TCN-AE, we use the identical TCN architecture for the encoder and decoder, with the same number of filters n_{filters}, filter lengths k and dilation rates q_i. In the decoder of the baseline TCN-AE, right after the upsampling layer, the first dilated convolutional layer has a dilation rate of $q = 1$. However, if we keep in mind that the upsampling layer uses sample-and-hold interpolation, which repeats each sample $s = 32$ times, a dilation rate of $q = 1$ might be ineffective. Due to the upsampled signal's coarse structure, the filters are mostly moved over ranges of identical values. A straightforward yet beneficial enhancement is to reverse the dilation rates in the decoder. Hence, now the last dilated convolutional layer before the output layer will have a dilation rate $q = 1$, the penultimate layer $q = 2$, and the first layer (after the upsampling layer) a dilation rate of 2^{L-1}. With this approach, larger dilation rates are used on coarser levels. In our architecture with $L = 7$ dilated convolutional layers, we use the dilation rates $(1, 2, 4, \ldots, 64)$ for the encoder, and the dilation rates $(64, 32, 16, \ldots, 1)$ for the decoder (see the green sticks in Fig. 7.5 and 7.6).

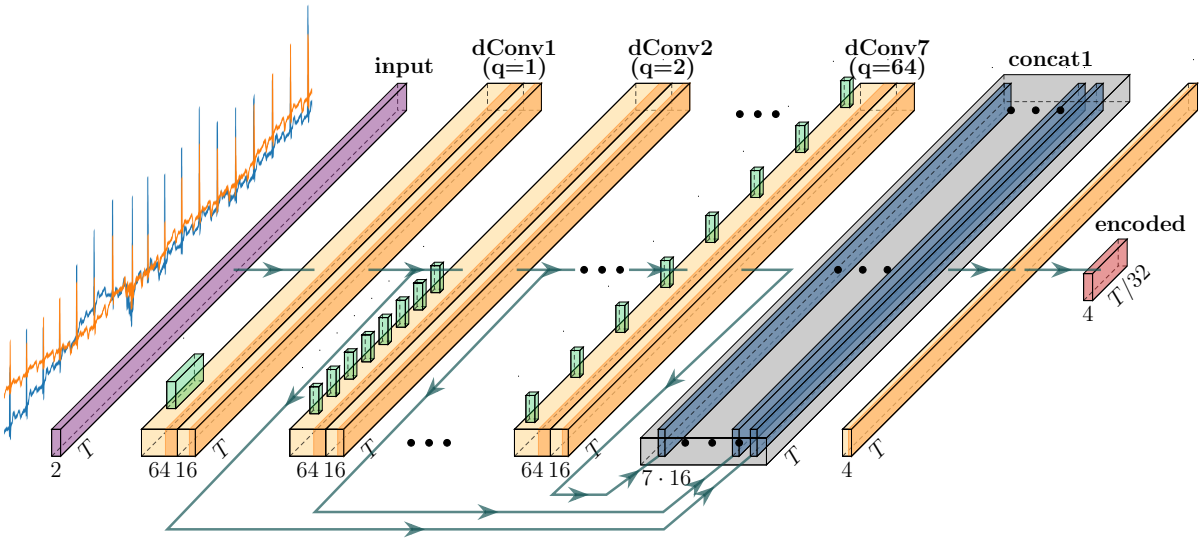

Figure 7.5: The architecture of TCN-AE's encoder. The two-dimensional input ECG-signal (purple) of length T, is passed through a stack of dilated convolutional layers (light orange, dConv1 – dConv7). The light green boxes represent the filters of the dilated convolutions. Each dilated convolutional layer is followed by a 1×1 convolution, which reduces the number of channels to 16. The outputs of the 1×1 convolutions are also concatenated in the block concat1 (blue). The dark blue blocks are identity mappings, not altering the tensors. Overall, seven tensors are concatenated, resulting in $7 \cdot 16 = 112$ channels. Finally, the concatenated tensor is compressed into the final encoded representation (red). The compressed representation of the original input is then passed to the decoder (Figure 7.6).

7.4.1.3 Utilizing Hidden Representations for the Anomaly Detection Task

While studying the relation of dilated convolutions to the DWT (Section 7.2.3), we noticed some similarities to our prior work [164]: In that work, we used the DWT to analyze a time series signal on different frequency scales to detect anomalous behavior. Each frequency scale was analyzed independently, and the aggregated results then led to an anomaly score for each data point of the time series. Similarly, transferred to the TCN-AE architecture, one could imagine that each dilated convolutional layer's output corresponds to an individual frequency/time scale, which already might carry useful information for the anomaly detection task. Hence, it could be sensible to look at the reconstruction error signal of TCN-AE and also individual hidden representations of the network to identify anomalies.

We take the output of each map-reduction layer (see section 7.4.1.4) in the encoder and reduce the feature map channels with a 1×1-convolution to size one. This is like taking each blue bar from Fig. 7.5 and reducing it to one output channel. We then stack each of the reduced outputs onto the reconstruction error signal. If there are seven dilated convolutional layers in the encoder ($q = 1 \ldots 64$) and the reconstruction error signal is two-dimensional, seven additional hidden representations will be stacked onto the reconstruction error signal. We end up with a 9-dimensional signal to which we apply Algorithm 8. With this approach,

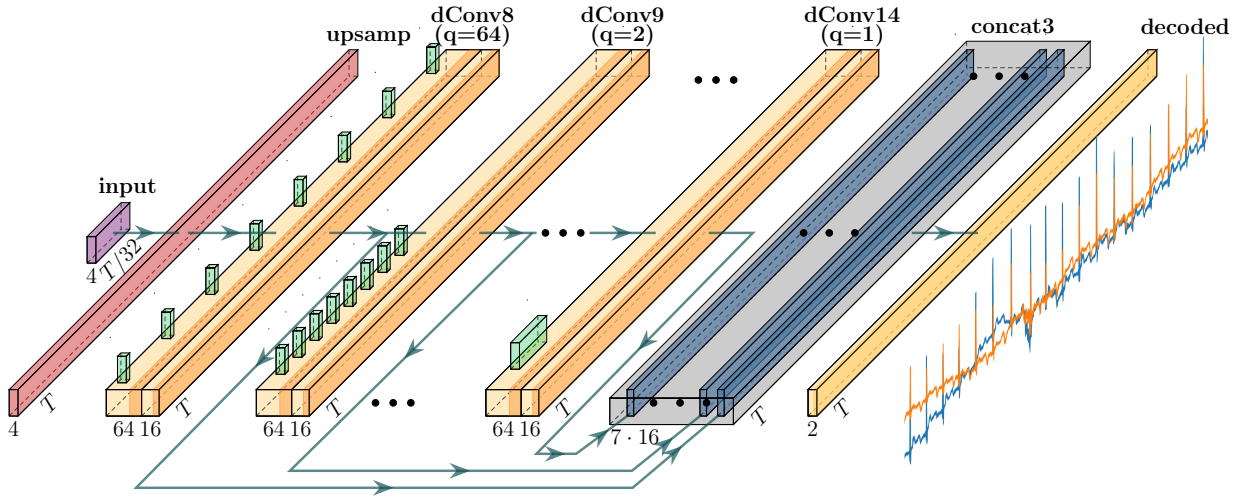

Figure 7.6: The architecture of TCN-AE's decoder. A compressed representation is given as input (purple) and then upsampled (red layer) to the original length T. Similar to Figure 7.5, a stack of dilated convolutional layers operates on the upsampled signal and the outputs of the 1×1 convolutions are concatenated in the concat3 block. Finally, the output layer (convolution with linear activation), reconstructs the original two-dimensional ECG sequence.

we can not only search for anomalies in the reconstruction error but also find irregularities in various hidden feature representations of the input time series.

Note that this enhancement is not shown in Figs. 7.5 and 7.6 to keep the complexity of the figures manageable.

7.4.1.4 Feature Map Reduction

A more technical enhancement of TCN-AE is the introduction of convolutional map reduction layers (commonly referred to as 1×1 convolutional layers) [96, 156], which are regularly used in practice to reduce the dimensionality (the number of channels) of feature maps and effectively reduce the number of trainable parameters in the overall architecture. We experimented with 1×1 convolutional layers and found that they allow reducing the overall number of parameters in the network, without sacrificing performance. Additionally, we could observe a slight improvement in the training time. We place 1×1 convolutions after each dilated convolutional layer, which reduces the number of channels from 64 to 16.

7.4.1.5 Anomaly Score Baseline Correction

While visualizing the anomaly score of the TCN-AE model for a few time series, we noticed that the anomaly score did not always have a constant baseline, as one would expect. We observed slight drifts in the baseline, which made it hard in some cases to find a suitable

threshold value. One reason for this phenomenon could be that certain statistical properties of the signal (such as the random noise) change over time. Since these drifts correspond to low-frequency components in the anomaly score, a simple way to remove them is to filter the anomaly score. We decided to use a second-order Butterworth filter with a cutoff frequency of 1Hz to remove the baseline wandering.

7.4.2 Experimental Setup

In this chapter, we compare all considered algorithms on the ECG-25 benchmark, described in Section 2.3. If not stated otherwise, we sum TP, FN, and FP over all 25 ECG time series. From these three quantities, the well-known metrics precision (Prec), recall (Rec), and F_1-score are derived.

7.4.2.1 Algorithmic Setup

We compare our unsupervised TCN-AE algorithm to four other unsupervised anomaly detection algorithms: DNN-AE [46], LSTM-ED [108], LSTM-AD [161], and NuPIC [160]. They are based on deep autoencoders (DNN-AE), LSTM networks (LSTM-ED and LSTM-AD), and hierarchical temporal memory, HTM (NuPIC).

All anomaly detection algorithms are trained in an unsupervised fashion. The actual anomaly labels are only used at test time. The training process passes the complete time series to the anomaly detection algorithm, and the algorithm learns a model for the provided data and returns an anomaly score for each data point of the time series. We trained all algorithms, except NuPIC (which does not support GPU capabilities), on a Tesla P100 GPU. All algorithms require a set of hyperparameters, which we will describe in the following. Parameters common to all algorithms are summarized in Tab. 7.2. We tuned the parameters (except for TCN-AE and NuPIC) using the HYPEROPT library [14]. For TCN-AE, we manually investigated different parameter settings, and for NuPIC, we use the recommended parameter settings [89].

To obtain statistically sound results, we run each anomaly detection algorithm ten times on all 25 ECG time series.

DNN-AE [46]: We use a PyTorch [130] implementation for the anomaly detection algorithm based on a deep autoencoder [58]. The algorithm requires several parameters, which we choose as follows: hidden size of $h = 6$ for the bottle neck (which results in a compression factor of $T_{\text{train}}/h = 25$ for each sequence). Finally, we set $\%_{Gaussian} = 1\%$, which specifies that 99% of the data is used to estimate a Gaussian distribution for the anomaly detection task.

LSTM-ED [108] is also implemented using PyTorch and has the following parameter setting: $\%_{Gaussian} = 3\%$. Both, encoder and decoder use a stacked LSTM network with two layers, each having LSTM layer having 50 units.

NuPIC [160]: Numenta's anomaly detection algorithm has a broad range of hyperparameters that have to be set. We use the parameters recommended by the authors

in [89]. It is possible to tune the parameters with an internal swarming tool [3]. However, this is a time-expensive process which is not feasible for the large benchmark.

LSTM-AD (Chapter 6, [161]): A 2-layer LSTM network with 256 units in the first layer and 128 units in the second layer is used. The target horizons are chosen to be $H = (1, 3, \ldots, 49)$.

SORAD (Chapter 4, [163]) We use SORAD with mini-batch RLS – using small batches to update the model instead of single examples (see Appendix B.1) – as a simple baseline method. The batch size is set to $\mu = 256$. The target horizons $H = (1, 3, \ldots, 49)$ are the same as for LSTM-AD. The forgetting factor is set to $\lambda = 0.98$. The window length for the sliding window is $w = 128$. Hence, the number of trainable weights of the model is 129 (including one bias weight). The regularization parameter was set to $\beta = 10^{-5}$.

TCN-AE (baseline): The settings of the baseline TCN-AE model (Figure 7.2) mostly correspond to the settings of the final variant. Only the maximum dilation rate is chosen smaller so that $q = (1, 2, \ldots, 32)$ and the number of filters for each dilated convolutional layer is reduced to $n_{\text{filters}} = 32$. Nonetheless, the number of trainable parameters of the baseline TCN-AE is larger due to the two consecutive layers which are created for each individual dilation rate. For baseline TCN-AE, we use an existing TCN implementation in Keras [140].

TCN-AE (final): We implemented TCN-AE using the Keras [30] & TensorFlow framework [1]. An overview of the architecture with its parameters is given in Figures 7.5 and 7.6. In both encoder and decoder we use 7 dilated convolutional layers each, with the dilation rates $q = (1, 2, \ldots, 64)$ (encoder) and $q = (64, 32, \ldots, 1)$ (decoder), $n_{\text{filters}} = 64$ filters with a kernel size of $k = 8$, and a ReLU activation. Each dilated convolutional layer is followed by a 1×1 convolutional layer with $n_{\text{filters}} = 16$ filters, which reduces the feature maps from 64 to 16. The sample rate of the average pooling layer is $s = 32$ and the error window length for the anomaly detection in Algorithm 8 is $\ell = 128$. For MGAB, we use $q = (1, 2, \ldots, 16)$, $n_{\text{filters}} = 32$, $k = 25$, $B = 64$, $n_{\text{epochs}} = 10$, $T_{\text{train}} = 1050$, $n_{\text{filters}} = 16$ filters for the skip connections, $s = 6$, and $\ell = 128$.

Table 7.2: Summary of the common parameters of the neural-network-based anomaly detection algorithms used in this work.

Algorithm	B	n_{epochs}	T_{train}	Loss	Optimizer	Initializer
TCN-AE	64	10	1024	logcosh	Adam	Glorot Normal [50]
DNN-AE	100	25	150	MSE	Adam	$\mathcal{U}(-\sqrt{k}, \sqrt{k}), k = \frac{1}{fan_{in}}$
LSTM-ED	100	10	30	MSE	Adam	$\mathcal{U}(-\sqrt{k}, \sqrt{k}), k = \frac{1}{fan_{in}}$
LSTM-AD	512	25	256	MAE	Adam	Glorot Uniform [50]

7.4.2.2 Evaluation

For most results presented in this section, we use the EAC criterion to determine precision, recall and F_1-score. To evaluate the performance of the algorithms over a wide range of anomaly thresholds, we also generate a precision-recall curve. In the other cases, we select an optimal threshold in a supervised manner for a small fraction of the time series data and then apply this threshold to the overall time series. If not stated otherwise, we select a segment containing 10% of a time series and find the threshold which maximizes the F_1-score on this small subset. Since the results may vary, depending on which 10%-segment is used, we repeat the whole evaluation procedure 10 times and average the results: adjust the threshold on 10% of the data, evaluate on the remaining 90% of the data. We assess the significance of the results with the non-parametric Wilcoxon signed-rank test [178] and report the p-values.

7.4.3 Evaluation of TCN-AE on MGAB

Before performing the experiments on the ECG-25 benchmark, we first run our enhanced TCN-AE algorithm on MGAB and compare the results with the results reported in the previous section. The anomaly threshold is determined by using 10% of the anomaly labels supervisedly, as described in the experimental setup (Sec. 7.3.2.1) of the previous section. The results for the final TCN-AE model are summarized in Table 7.3 and compared to the three other best models. The results for TCN-AE (baseline), LSTM-AD and DNN-AE are copied from Table 7.1. We can see that the performance of TCN-AE (final) is similar to TCN-AE (baseline) and LSTM-AD. However, instead of originally 164 451 trainable weights, TCN-AE (final) now only has 38 423 weights. Also the computation time could be drastically reduced from an aaverage time of 65 seconds (baseline) to about 17 seconds / time series.

Table 7.3: Similar to Table 7.1. Here, we also list the results for TCN-AE (final). Additionally, we also list the p-values, indicating the significance of the results.

Algorithm	TP	FN	FP	Precision	Recall	F_1
DNN-AE [58]	91.8 ± 1.2	8.2 ± 1.2	62.6 ± 13.6	0.600 ± 0.058	0.918 ± 0.012	0.724 ± 0.043
LSTM-AD (Ch. 6,[161])	88.8 ± 2.6	11.2 ± 2.6	0.6 ± 0.6	0.993 ± 0.007	0.888 ± 0.026	0.937 ± 0.014
TCN-AE (final)	90.6 ± 1.9	9.4 ± 1.9	0.4 ± 0.7	0.995 ± 0.008	0.906 ± 0.019	0.948 ± 0.010
TCN-AE (baseline)	90.5 ± 1.7	9.5 ± 1.7	0.2 ± 0.5	0.997 ± 0.011	0.905 ± 0.021	0.949 ± 0.010

7.4.4 Experiments, Results & Discussion for the ECG-25 Benchmark

We started our experiments with the baseline TCN-AE model (Section 7.3.1). The initial results on the ECG-25 benchmark were already promising, but the algorithm still performed

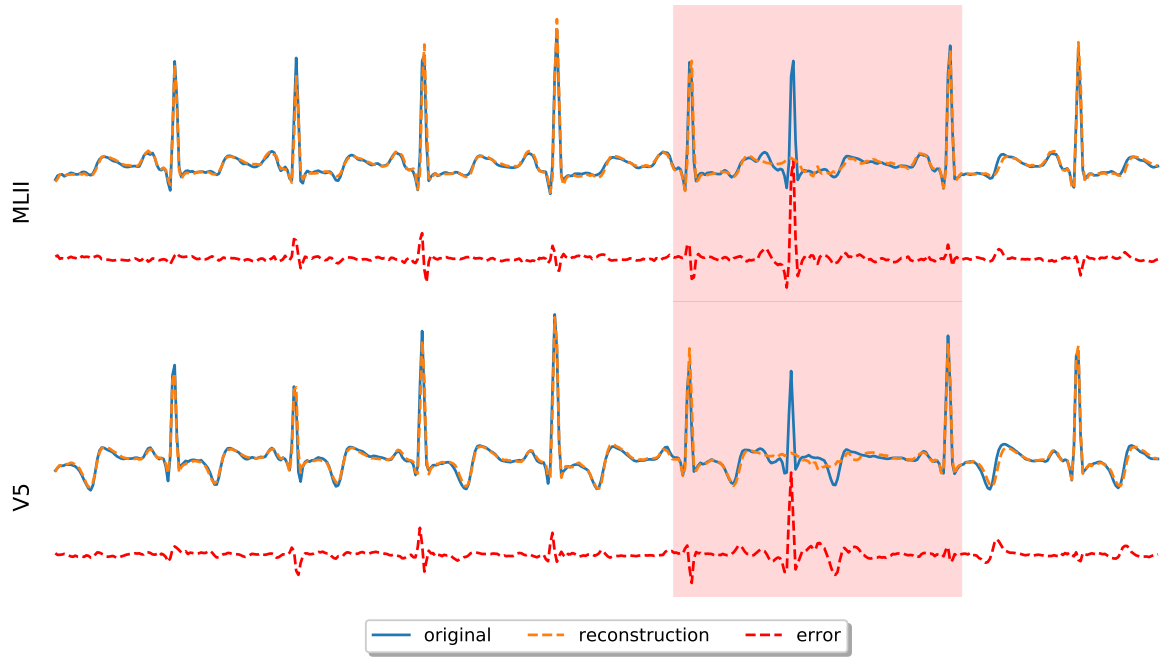

Figure 7.7: Excerpt showing how the final TCN-AE model reconstructs the modified limb lead II (MLII) and the modified lead V1 of ECG signal #1. TCN-AE has difficulties in reconstructing actual anomalous behavior (highlighted with the red shaded area). Due to the resulting large error, the algorithm later correctly detects an anomaly (true positive).

similar to LSTM-AD and DNN-AE concerning the $F1$-score (Table 7.6), leaving room for improvements. While analyzing the baseline TCN-AE, we developed several ideas for improvements, which we introduced in Section 7.4.1. The resulting (final) TCN-AE showed significantly improved performance, achieving a higher precision and recall on 15 out of 25 time series of the benchmark. We perform a more detailed analysis of the contribution of the individual enhancements in the following Section.

Figure 7.7 depicts an example for ECG signal #1, where the TCN-AE algorithm has difficulties reconstructing the original time series due to an actual anomaly present. In this case, the large construction error is correctly interpreted as anomalous behavior. For the same example, we visualize selected activations of several layers inside the trained TCN-AE in Figure 7.8. While the ECG's general patterns are still visible in the initial layers of the encoder, these disappear in later layers, and the activations do not seem to carry any information appearing useful to the human eye. After being passed through the bottleneck and upsampled again, only a 4-channel (from which one is depicted in the graph) step-shaped signal remains. However, remarkably, the decoder can almost accurately restore the original input sequence solely from this coarse representation (sixth row in the plot). Only

the anomalous pattern is incorrectly reconstructed, which results in a large reconstruction error that can easily be detected.

7.4.4.1 Contribution of the Individual TCN-AE Components

In the following, we describe the impact of individual enhancements on the final TCN-AE, which we introduced in Sec. 7.4.1.

Variant	Section	Comment
baseline	7.3.1	Baseline algorithm based on TCNs without any enhancements
noSkip	7.4.1.1	Skip-Connections removed from the architecture
noInvDil	7.4.1.2	Use same dil. rate ordering for encoder & decoder
noLatent	7.4.1.3	Do not use hidden represenations for anomaly detection
noRecon	7.4.1.3	Only use hidden representations of encoder for anomaly detection
noMapReduc	7.4.1.4	Do not use the Map reduction layers
noAnomScoreCorr	7.4.1.5	Do not correct the baseline of the anomaly score
final	7.4.1	Final TCN-AE with all enhancements

Table 7.4: Summary of all TCN-AE variants

Although it is challenging to accurately measure each element's effect on the final result (since there might be some interaction effects between elements[2]), we can approximately quantify the improvements with the following approach: In order to measure the contribution of component C on the final result, we run TCN-AE on the benchmark with this specific component turned off. If the component has a positive impact on the model, we expect a poorer result, and the differences in precision, recall, and F_1-score serve as a rough indicator for the contribution of the component. Additionally, the p-value of the one-sided Wilcoxon test signalizes the significance of the result. In Table 7.5, we summarize the different variants of the TCN-AE algorithm. Further results are listed in Appendix A.

Overall, all the individual enhancements significantly improve the performance of TCN-AE on the ECG-25 benchmark. As summarized in Table 7.5, the precision, recall and F_1-score all improve by around 10%. All enhancements have a significant impact on the increase in performance, as indicated by the corresponding p-values. The p-values are also illustrated graphically in a heat map in Figure 7.9 for all 25 time series of the benchmark. We can see that the algorithm achieved an improvement for most time series. The exact F_1-scores for each TCN-AE variant and time series can be found in Table A.4. This table also highlights the time series for which the p-values are above the significance level of 0.05.

Skip Connections The skip connections in TCN-AE allow the last layers of the encoder and decoder to access all prior layers (having different dilation rates) directly. As shown

[2]We assume that the overall contribution of the individual components is larger than our estimations due to the interaction effects which we cannot measure.

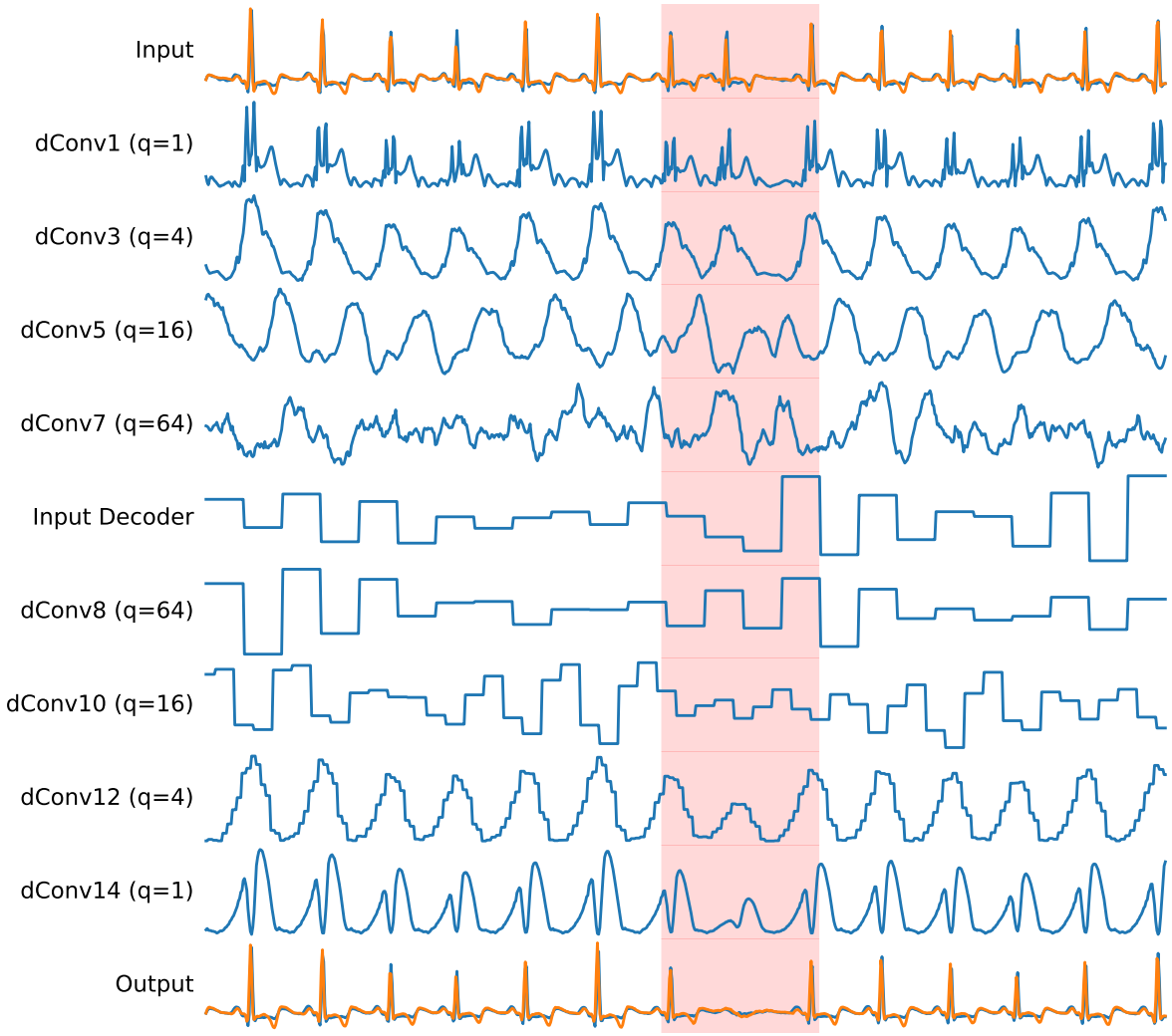

Figure 7.8: Activations inside the trained final TCN-AE model for several layers of the network. For each dilated convolutional layer, we plot the channel (signal) with the largest mean absolute activation. If we compute and plot the mean over all channels, we get structurally relatively similar results. The rows dConv1 – dConv7 refer to the activations of the dilated convolutions inside the encoder, while dConv8 – dConv14 are dilated convolutional layers inside the decoder. The input signal contains an anomaly (atrial premature beat), highlighted with the red-shaded vertical bar. The decoder fails in reconstructing this segment of the time series, which results in a significant deviation between the original and reconstructed signal.

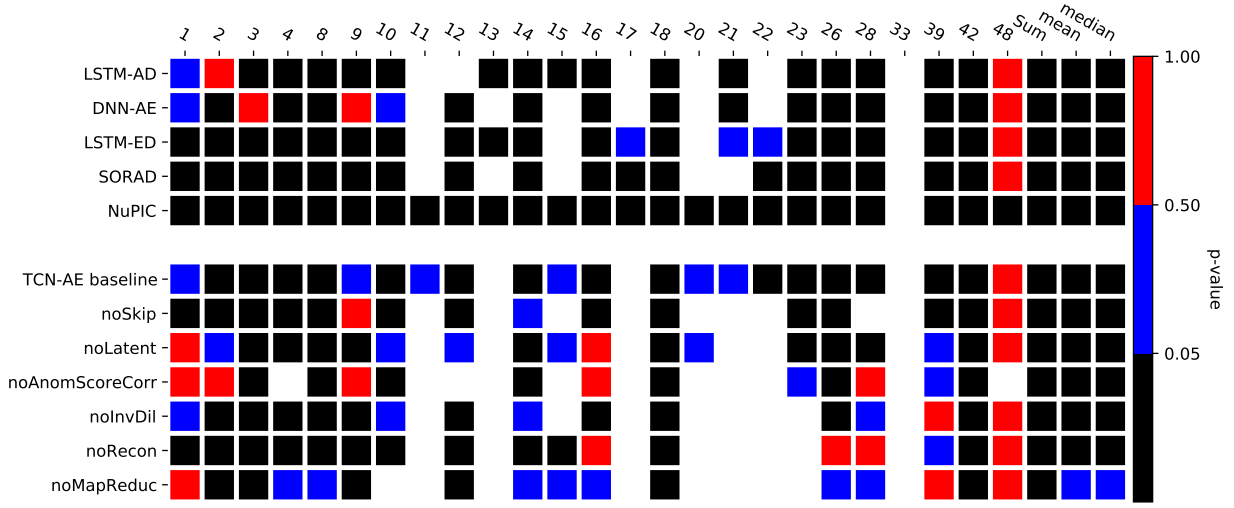

Figure 7.9: Heatmap, showing the p-values for the comparison of our final TCN-AE model to various algorithms and other variants of TCN-AE for all 25 ECG signals of our benchmark. We use a one-sided Wilcoxon signed-rank test for the F_1-scores of ten runs each. The test has the null hypothesis that the median F_1-score of our final TCN-AE is smaller (than the compared algorithm) against the alternative that the median F_1-score is larger. The first four rows represent anomaly detection algorithms from the literature, while the remaining rows are for different variants of the TCN-AE algorithm. In the cases where the p-value is below the significance level of $\alpha = 0.05$, the tiles are colored black (indicating a significantly higher performance of TCN-AE). White tiles indicate that the F1-scores of TCN-AE and the compared algorithm are exactly the same. The exact F_1-scores are given in Table A.4 and Table 7.9.

Table 7.5: Impact of the individual TCN-AE components for the ECG-25 data (mean and standard deviation of 10 runs). The results shown here are for the sum of TP, FN and FP over all 25 time series and were obtained such that an approximate (difference of less than 1% in precision and recall) equal accuracy (EAC) is achieved for each algorithm and time series.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
baseline	597.5 ± 5.1	123.5 ± 5.1	129.0 ± 5.2	0.822 ± 0.007	0.829 ± 0.007	0.826 ± 0.007	2.531e-3
noSkip	622.4 ± 5.7	98.6 ± 5.7	102.9 ± 5.7	0.858 ± 0.008	0.863 ± 0.008	0.861 ± 0.008	2.531e-3
noLatent	629.5 ± 5.2	91.5 ± 5.2	95.3 ± 5.4	0.869 ± 0.007	0.873 ± 0.007	0.871 ± 0.007	2.531e-3
noAnomScoreCorr	644.2 ± 3.6	76.8 ± 3.6	83.2 ± 3.7	0.886 ± 0.005	0.893 ± 0.005	0.890 ± 0.005	2.531e-3
noInvDil	653.6 ± 1.9	67.4 ± 1.9	72.2 ± 1.9	0.901 ± 0.003	0.907 ± 0.003	0.904 ± 0.003	2.531e-3
noRecon	656.9 ± 2.9	64.1 ± 2.9	69.9 ± 2.9	0.904 ± 0.004	0.911 ± 0.004	0.907 ± 0.004	2.531e-3
noMapReduc	660.7 ± 1.3	60.3 ± 1.3	65.1 ± 1.4	0.910 ± 0.002	0.916 ± 0.002	0.913 ± 0.002	8.302e-3
final	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003	–

in Table 7.5 and Figure 7.9, this improvement has the highest impact on the performance of TCN-AE: Without skip connections, the F_1-score drops from $F_1 \approx 0.93$ to $F_1 \approx 0.86$. However, for the model without the skip-connections, we had to decrease the number of filters from $n_{\text{filters}} = 64$ to $n_{\text{filters}} = 32$; otherwise the results would be even worse.

We also experimented with different variants of a dense TCN-AE similar to a DenseNet [66] (connecting all dilated convolutional layers with the preceding ones). However, we decided to no longer pursue this approach, since the dense connections increased the number of parameters and the computation time significantly without considerably improving the results.

Although our primary purpose for the introduction of skip connections is to reduce the sensitivity towards the maximum dilation rate and to enable the encoder/decoder to reuse the features at different time scales, as a side effect, our TCN-AE architecture might also benefit from other advantages associated with ResNets [60] or DenseNets [66]. Some of the observed improvements might be due to a smoother curvature of the loss landscape [93], alleviation the vanishing/exploding gradient problem & degradation problem due to gradient shortcuts through the identity mappings of the skip connections, reduction of parameters, and others.

We also tested higher dilation rates up to $q_{max} = 1024$ and found (apart from the higher computation time and memory requirements[3]) that the results remained almost the same. Only for $q_{max} = 1024$ we could observe a slight drop in the overall F_1-score, from $F_1 \approx 0.93$ to $F_1 \approx 0.91$. Since the results do not change significantly with a larger stack of dilated convolutional layers, this might imply that the TCN-AE model is capable of learning to select the suitable features from the important time scales and to ignore the remaining time scales which do not carry directly relevant features. In the baseline version, where no skip connections were employed, the results significantly deteriorated for inappropriate (too large/small) choices of q_{max}.

Dilation Rate Ordering In Figure 7.9, we can see that this reversed dilation rate scheme improves the results on most of the considered 25 time series. While the reason for the improvement is not entirely apparent yet, we assume that a larger dilation rate is beneficial for the coarse step-shaped signal we have in the first layers of the decoder, and a lower dilation rate is essential when we attempt to reconstruct the details of the original signal.

Detecting Anomalies in Hidden Representations of Time Series As discussed in Section 7.4.1.3, we noticed that there are some similarities between the (stationary) discrete wavelet transform (DWT) and DL architectures, which use stacks of dilated convolutional layers. In [164], we used the DWT to decompose a time series and look for anomalous behavior on different frequency scales. Similarly, we can also utilize the outputs of the individual dilated convolutional layers, which process the time series on different time scales. The general idea is that anomalies might become more apparent on some time scales than on others and that one can already detect anomalous behavior on these hidden representations rather than relying solely on the reconstruction error. In our first experiment, we used the

[3]Since the receptive field of the model increases with larger dilation rates, also longer training sequences are required. (We assume that this is also partially due to the artifacts induced when the filters move within the zero-padded areas.)

encoder's outputs of the dilated convolutional layers, reduced their number of channels to one (with a 1×1 convolution), and stacked them on top of the reconstruction error $\mathbf{e}[n]$ (line 11 in Algorithm 8). Although we observe a drop in the F_1-score for 3 ECG signals (#1, #16, #48) in Figure 7.9 (Table A.4), the overall results suggest that this approach generally improves the results (Table 7.5). The overall F_1-score increases from $F_1 = 0.89$ to $F_1 = 0.93$ and the mean (median) F_1-score is significantly higher (Figure 7.9 & Table A.4).

Similarly, in our next experiment, we tried also to utilize the decoder's hidden representations. However, this did not have any further effect on the algorithm's performance, and we discarded this approach again.

We made another interesting observation: If we entirely remove the decoder after training TCN-AE and solely use the outputs of the encoder's dilated convolutional layers to detect anomalies in the time series, still decent results can be obtained. In Table 7.5, we can see that the F_1-score only drops by about 0.02, although the size of the model (and the computational cost for inference) is effectively halved. This observation might be interesting for practical applications, where memory and computational resources are constrained.

Map Reduction Layers Although the primary purpose for the map reduction layers (Section 7.4.1.4) was to reduce the number of parameters in the overall model, as a side effect, we could observe a slight improvement in the overall performance, when considering the sum over all TP, FP, and FN. However, the improvement is smaller than for the other previously discussed enhancements. The mean (median) F_1-score does not increase, as shown in Figure 7.9 (Table A.4).

Anomaly Score Baseline Correction Also, the correction of the anomaly score baseline using a Butterworth filter, as described in Section 7.4.1.5, has a notable impact on the final results. Although there is only a significant improvement for 7 out of 25 time series (Figure 7.9), the overall F_1 drops by around 4% if we turn off the baseline correction of the anomaly score, as reported in Table 7.5. Instead of using a filter, we also tested the more advanced baseline correction algorithm by Zhang et al. [185], and obtained results which did not significantly differ (F_1-score of 0.920 ± 0.003).

7.4.4.2 Comparison to other Algorithms

In Table 7.6, we summarize the results for all algorithms. The table is sorted according to the F_1-score and shows the p-values for comparing the F_1-scores of the individual algorithms with TCN-AE. The first observation we can make is that TCN-AE (baseline and final variant) outperforms the other five algorithms significantly (p-value < 0.05). On average, TCN-AE detects 81 anomalies more than the second-ranked algorithm, LSTM-AD, while at the same time also producing 80 fewer FPs. The overall F_1-score of TCN-AE is around 15% higher than of DNN-AE and even 20% higher than of LSTM-ED.

Table 7.6: Summary for the ECG-25 data (mean $\pm \sigma_{\text{mean}}$ of 10 runs, except for the deterministic NuPIC algorithm). The results shown here are for the sum of TP, FN and FP over all 25 time series and were obtained such that an approximate (difference of less than 1% in precision and recall) equal accuracy (EAC) is achieved for each algorithm.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
NuPIC	224.0	497.0	497.0	0.311	0.311	0.311	2.531e-3
SORAD	519.0	202.0	206.0	0.716	0.72	0.718	2.531e-3
LSTM-ED	557.3 ± 2.9	163.7 ± 2.9	168.9 ± 2.9	0.767 ± 0.004	0.773 ± 0.004	0.770 ± 0.004	2.531e-3
DNN-AE	583.7 ± 1.1	137.3 ± 1.1	142.9 ± 1.3	0.803 ± 0.002	0.810 ± 0.002	0.806 ± 0.002	2.531e-3
LSTM-AD	589.3 ± 0.8	131.7 ± 0.8	136.4 ± 0.5	0.812 ± 0.001	0.817 ± 0.001	0.815 ± 0.001	2.531e-3
TCN-AE (baseline)	597.5 ± 5.1	123.5 ± 5.1	129.0 ± 5.2	0.822 ± 0.007	0.829 ± 0.007	0.826 ± 0.007	2.531e-3
TCN-AE (final)	670.2 ± 2.4	50.8 ± 2.4	55.8 ± 2.4	0.923 ± 0.003	0.930 ± 0.003	0.926 ± 0.003	–

Precision-Recall Curves Since the anomaly threshold trades off FNs (recall) and FPs (precision), another way of showing the performance of an anomaly detection algorithm is to vary the threshold over a wide range of values and plot the precision and recall for different settings in a graph. In Figure 7.10, we generated such a precision-recall plot for all the compared algorithms. The precision-recall plot can be seen as a bi-objective optimization problem where one attempts to maximize both precision and recall. Each point in the graph is obtained for a specific threshold value. We fit one curve as a rough approximation to the points of the 10 runs. It can be seen that TCN-AE outperforms the other algorithms over a wide range of anomaly thresholds. Especially along the identity line (precision=recall), the difference of TCN-AE to the other algorithms becomes apparent.

Performance for individual Anomaly Types The ECG-25 benchmark contains nine different anomaly types, as summarized in Table 2.2. Since the anomaly types take very different shapes, we were interested in knowing how well TCN-AE can detect the individual types and how it compares to the other anomaly detection algorithms. Table 7.7 shows how many of the individual anomaly types were detected by the respective algorithms for an EAC setting. Although TCN-AE has the most detections for only five out of nine anomaly types, it is among the top three algorithms in each case. Furthermore, it produces significantly less FPs in the EAC setting (if we would permit TCN-AE to also have more FPs, similar to the other algorithms, it would detect even more anomalies). However, we still see room for improvement. A more thorough investigation of the anomaly types that appear to be hard for TCN-AE could lead to new insights and possible new enhancements.

Computing Anomaly Score with little Labled Data While we used EAC to determine all anomaly thresholds for the results presented in Table 7.6, we also investigated how the results change when all algorithms may use only a small fraction of each time series' labels to find a suitable threshold. This approach is more realistic for practical applications

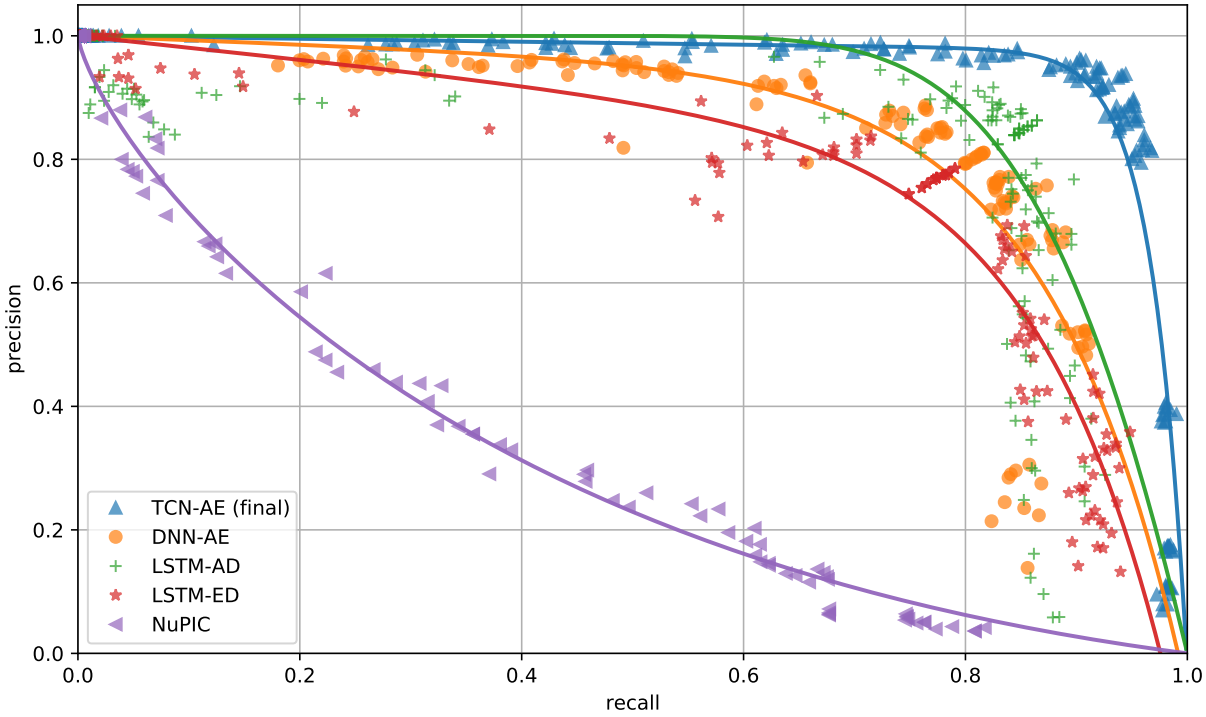

Figure 7.10: Precision-recall curves for the individual algorithms on the ECG-25 data. Shown are the fits through around 100 points which were generated by evaluating precision & recall for different thresholds. For each algorithm, except NuPIC, 10 runs were performed.

since, usually, only little labeled data is available. In our specific experiment the algorithms were only allowed to use 10% of the anomaly labels The detailed results are listed in the appendix in Table A.2. We observe that the F_1-score for all algorithms deteriorates compared to Table 7.6, where the EAC was used. Nevertheless, TCN-AE ($F_1 = 0.79$) still has the highest performance (having the highest F_1-score on 18 out of 25 time series), followed by LSTM-AD ($F_1 = 0.67$), LSTM-ED ($F_1 = 0.6$), and DNN-AE ($F_1 = 0.57$), while NuPIC performs the worst ($F_1 = 0.22$).

One interesting observation is that the recall of all algorithms is significantly higher than the precision. A possible explanation for this is that for many 10% intervals, on which the algorithms optimize their threshold, a threshold value can be found that results in a high recall and high precision. However, this threshold is then too low for the remaining time series, and many FPs are created as a consequence. This problem demonstrates that in practice, more sophisticated methods are necessary to determine a suitable threshold. We are planning to work on this issue in future work.

Finally, we compare in Table 7.8 the runtimes of all algorithms and the number of trainable parameters. All TCN-AE variants are considerably faster than the remaining

Table 7.7: Number of detected anomalies for the individual algorithms, broken down by anomaly type (in total 9 types, see Table 2.2). We count the true detections when an EAC is used. The last two columns (FN & FP) are the false negatives and false positives summed up over all anomaly types.The last row depicts optimal results for each type.

Algorithm	a	A	e	f	F	J	V	x	|	FN	FP
NuPIC	2.0	32.0	1.0	1.0	5.0	2.0	172.0	3.0	6.0	497.0	497.0
SORAD	9.0	108.0	5.0	17.0	17.0	2.0	337.0	6.0	11.0	207.0	210.0
LSTM-ED	10.1	150.5	7.2	18.6	20.0	2.0	328.1	9.1	11.7	163.7	168.9
DNN-AE	9.9	212.4	**7.4**	**21.5**	**21.8**	1.0	290.9	8.9	9.9	137.3	142.9
LSTM-AD	6.4	213.4	4.2	5.8	18.4	2.0	316.9	12.4	9.8	131.7	136.4
TCN-AE (baseline)	9.6	176.9	5.5	18.8	18.9	1.8	345.0	9.0	**12.0**	123.5	129.0
TCN-AE (final)	**10.4**	**213.7**	6.3	20.2	19.5	**2.4**	**369.8**	**16.5**	11.4	**50.8**	**55.8**
Ideal	11	235	10	22	25	3	374	19	22	0	0

algorithms. TCN-AE final, the fastest algorithm is 5× faster than the fastest non-TCN-AE algorithm (DNN-AE). LSTM-AD, ranked 2nd according to the F_1-score, is by far the slowest algorithm and required more than four days to complete ten runs. Furthermore, LSTM-AD has the most trainable parameters. Also, NuPIC is relatively slow (around $750s$ per time series) since no GPU acceleration is available.

Table 7.8: Computation times: average (per time series) and total (10 runs, all time series) for all algorithms evaluated in this chapter. *TCN-AE final* is faster than *baseline* since it effectively has less layers. SORAD and NuPIC are not directly compareable to the other approaches since they are not neural-network-based and run only on one CPU core.

Algorithm	mean (s)	total (h)	#Params/10^3
SORAD (Ch. 4,[163])	12.5±0.1	0.1	0.129
NuPIC [160]	752.0±23.4	52.4	–
TCN-AE final	**47.6±0.4**	**3.3**	**123.8**
TCN-AE baseline	97.0±1.1	6.7	208.4
DNN-AE [58]	237.5±10.3	16.5	242.1
LSTM-ED [108]	626.5±30.8	43.5	134.4
LSTM-AD [161]	1615.0±127.6	112.2	465.6

7.4.4.3 Additional Investigations

Outlier Detection Algorithms We experimented with different outlier detection algorithms using different values of ℓ: The isolation forest [98] and extended isolation forest [57], the one-class support vector machine (OCC-SVM [147]), local outlier factor (LOF) [19], an

Table 7.9: F_1-scores (mean $\pm$ σ_{mean}) of TCN-AE and the other algorithms on all 25 time series of the ECG-25 benchmark (highest values in boldface). The p-values are computed with the one-sided Wilcoxon signed-rank test, in which we compare the final TCN-AE algorithm with the other variants. We compare the F_1-scores of ten runs, which are obtained for an EAC. The Wilcoxon test has the null hypothesis that the median F_1-score of TCN-AE (final) is smaller than the compared algorithm against the alternative that the median F_1-score is larger. In all cases in which we fail to reject the null hypothesis at a confidence level of 5%, we add a grey background the corresponding field.

	NuPIC		LSTM-ED		DNN-AE		LSTM-AD		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1
1	0.03	0.002	0.572±0.022	0.002	0.907±0.008	0.118	0.904±0.007	0.06	**0.919±0.006**
2	0.5	0.029	0.350±0.017	0.003	0.267±0.045	0.003	**0.783±0.026**	0.5	0.733±0.083
3	0.196	0.002	0.831±0.010	0.002	**0.981±0.006**	0.997	0.376±0.009	0.002	0.933±0.008
4	0.0	0.001	0.5	0.001	0.0	0.001	0.250±0.083	0.002	**1.0**
8	0.088	0.002	0.923±0.012	0.003	0.724±0.010	0.002	0.913±0.007	0.002	**0.981±0.003**
9	0.218	0.002	0.498±0.014	0.002	**0.708±0.005**	0.983	0.582±0.005	0.002	0.674±0.011
10	0.725	0.001	0.811±0.016	0.002	0.975±0.005	0.096	0.907±0.016	0.002	**0.987**
11	0.0	0.001	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
12	0.0	0.001	0.5	0.001	0.650±0.077	0.004	**1.0**	–	**1.0**
13	0.0	0.001	0.950±0.026	0.042	**1.0**	–	0.833	0.001	**1.0**
14	0.524	0.002	0.925±0.004	0.004	0.693±0.005	0.002	0.918±0.002	0.002	**0.945±0.004**
15	0.0	0.001	**0.909**	–	**0.909**	–	0.727	0.001	**0.909**
16	0.33	0.001	0.877±0.013	0.003	0.675±0.010	0.003	0.945±0.003	0.03	**0.953±0.001**
17	0.0	0.001	0.9 ±0.1	0.159	**1.0**	–	**1.0**	–	**1.0**
18	0.24	0.002	0.519±0.010	0.003	0.843±0.003	0.003	0.902±0.005	0.003	**0.962±0.003**
20	0.0	0.001	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
21	0.0	0.001	0.9 ±0.1	0.159	0.1 ±0.1	0.001	0.500±0.167	0.013	**1.0**
22	0.0	0.001	0.900±0.071	0.09	**1.0**	–	**1.0**	–	**1.0**
23	0.19	0.001	0.814±0.025	0.004	0.910±0.009	0.003	0.919±0.010	0.01	**0.952**
26	0.169	0.002	0.724±0.008	0.002	0.646±0.005	0.002	0.240±0.005	0.003	**0.806±0.011**
28	0.507	0.001	0.893±0.009	0.049	0.874±0.006	0.003	0.882±0.010	0.007	**0.912±0.004**
33	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**
39	0.25	0.002	0.778±0.018	0.002	0.899±0.006	0.002	0.793±0.012	0.003	**0.952±0.003**
42	0.514	0.002	0.845±0.007	0.003	0.898±0.003	0.045	0.798±0.005	0.003	**0.909±0.006**
48	0.0	0.002	**1.0**	0.987	**1.0**	0.987	**1.0**	0.987	0.875±0.042
Σ	0.311	0.003	0.770±0.004	0.003	0.806±0.002	0.003	0.815±0.001	0.003	**0.926±0.003**
mean	0.179	0.003	0.757±0.008	0.003	0.746±0.005	0.003	0.767±0.006	0.003	**0.896±0.006**
median	0.129	0.002	0.837±0.009	0.003	0.883±0.003	0.003	0.882±0.006	0.003	**0.953±0.002**

elliptic envelope based on the minimum covariance determinant estimator (MCD) [142], and finally, a simple method using only the Mahalanobis distance. Overall, we found the simple Mahalanobis-distance-based approach to deliver the best results. The other algorithms could partially produce similar results but required significantly more computation time and mostly required additional hyper-parameters, which had to be tuned first. Using the Mahalanobis distance as the anomaly score has the advantage that the algorithm only has to compute a mean vector and a covariance matrix, which is computationally less expensive and does not require any additional hyper-parameters. One reason for the higher accuracy

of this method over the other outlier detection algorithms could be that the reconstruction errors are bell-shaped in every dimension (elliptic in higher dimensions), as we observed in visualizations of the error distributions.

Downsampling and Upsampling Approaches We experimented with different approaches to create a bottleneck along the time axis and to restore the original time resolution again. There are several possibilities to decimate/downsample a sequence by a factor s:

1. Resampling: Keep every s-th sample of the sequence. This is the simplest method but could lead to artefacts in the resulting sequence, since there might be aliasing effects if higher frequencies are present.
2. Average pooling: Use an average pooling layer, where groups of size s are averaged. Effectively, the operation is a moving average whose output is resampled. In some sense, average pooling acts as a crude low-pass filter, which removes higher frequencies and reduces aliasing effects.
3. Max-Pooling: Similar to average pooling, with the difference that the maximum value of groups of size s is selected.
4. Strided convolutions: One could use a convolutional layer where the filters are moved with a stride of s. At the same time, the number of filters n_{filters} in this layer can be used to reduce the dimension of the feature map to a desired size.
5. Stepwise downsampling with convolutional layers and (average) pooling: Smoothen the downsampling process by using a sequence of pooling and convolutional layers. For example, to achieve a downsampling rate of $s = 32$, one could use a series of 5 pooling and convolutional layers.
6. Applying a regularizer to the activations: Instead of downsampling the sequence along the time axis, sparsity can also be enforced by applying a regularizer to the activations of the last layer in the encoder. We experimented with L1- and L2-regularization [12] and the Kullback-Leibler [118] divergence as penalty terms.

Accordingly, we also tested several different upsampling techniques:

1. Sample-and-hold interpolation: This is the simplest upsampling approach, where each sample is copied s times in order to recover the original length
2. Linear interpolation: Linearly interpolate between adjacent samples.
3. Tranposed convolutional layer: Use transposed convolutions [99, 40] to obtain the original length of the time series. Transposed convolutions are still used often in practice, but can suffer from so-called checkerboard effects [122].
4. Stepwise upsampling: Analogous to method 5 described in the previous list.
5. Max-Unpooling in combination with Max-Pooling [120].

Surprisingly, we observed that the results for the simple methods average pooling (downsampling step) and sample-and-hold interpolation (upsampling step) are similar and, in some cases, even superior to the supposedly more advanced approaches.

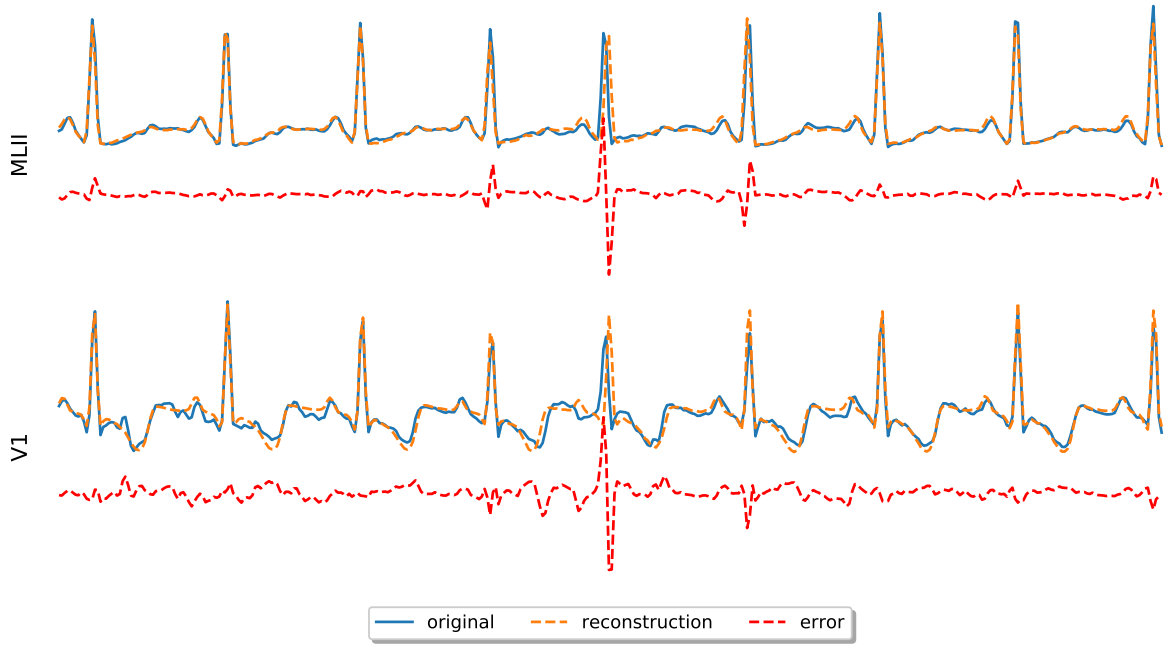

Figure 7.11: Similar to Figure 7.7, but now showing an excerpt of ECG signal #48. For both signals, one can observe a slight shift to the right in the reconstruction of the 5th R-peak, which results in an unusually large error and, ultimately, the TCN-AE algorithm falsely detects an anomaly (false positive) at this position. Taking the (ECG) signal's variability into account, in order to prevent situations like these, is addressed in our ongoing work.

However, it might be possible that more sophisticated down- and upsampling methods are beneficial in other situations, where the time-series data and algorithmic setup is different. Especially for the downsampling, one has to consider that, when using naive decimation approaches, aliasing effects might occur, which introduce artifacts into the compressed signal. We are planning to investigate this topic more thoroughly in the future.

7.4.4.4 Discussion

Only for the last ECG recording (#48), our final version of TCN-AE performs significantly worse most of the other algorithms. Surprisingly, for this time series, the final TCN-AE is also worse than most other variants without one of the enhancements, i.e., it is the combination of all additional modules that leads to the deterioration of the result for ECG signal #48. We found that the final TCN-AE algorithm produced an additional false-positive event, which reduces the overall precision on this time series. Exemplarily, we illustrate the cause of this FP in Fig. 7.11: one can see that a reconstructed R-peak appears slightly shifted, resulting in a large reconstruction error. The reason for this problem could be the quasi-periodic nature of the ECG signal, which is a major challenge for many algorithms.

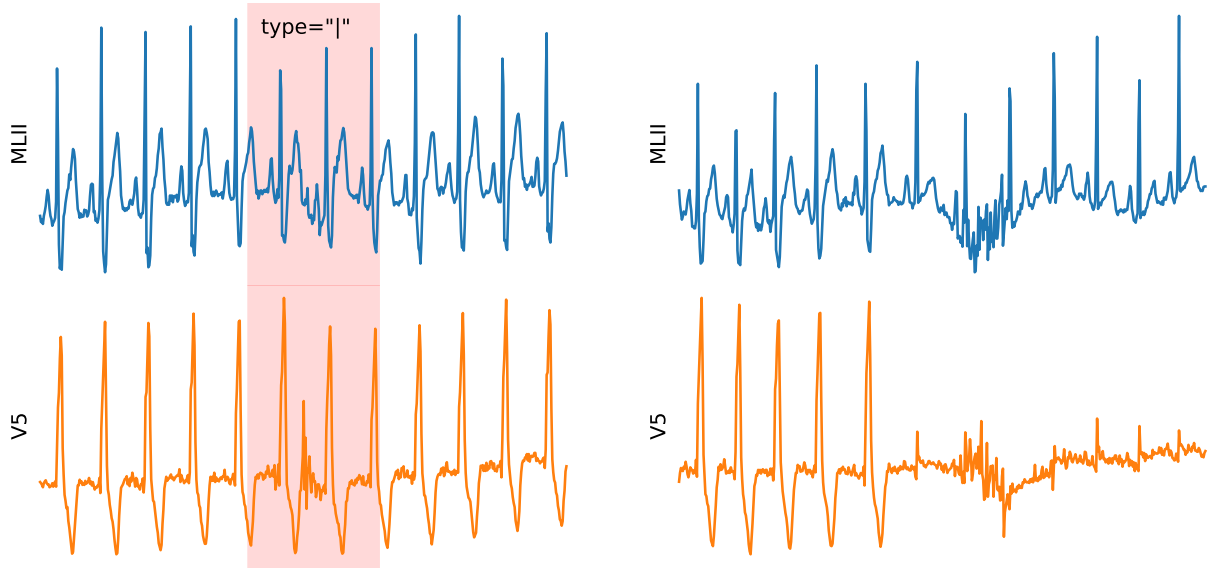

Figure 7.12: Two segments taken from ECG time series #33. This time series only contains one isolated QRS-like artifact (shown on the left side). The segment on the right side does not contain any anomalies, although there is a significant change in the signal. These signal quality changes occasionally occurring in this time series make it rather challenging for the unsupervised anomaly detection algorithms to detect the single anomaly.

We could observe similar events of this kind in a few other time series as well. Without the extra FP event in time series #48, TCN-AE would also achieve an F_1-score of $F_1 = 1.0$.

For time series #33 the results are unusual: we observe that all algorithms have $F_1 = 0$. This is because the time series contains only a single anomaly, which is rather hard to spot, even for the human eye. In Figure 7.12, the problem is illustrated: While the segment on the left contains the anomaly (an isolated QRS-like artifact), the segment shown on the right does not contain any anomaly, although there are significant changes in the signal quality. This makes it rather challenging for unsupervised algorithms to detect the single anomaly. We could observe many such signal quality changes in time series #33. If we discarded time series #33 from the benchmark, the average F_1-score would improve for all algorithms; for example, we would observe an increase from $F_1 = 0.896$ to $F_1 = 0.933$ for TCN-AE.

Also, there are a few other time series (#11, #17, #20, #22, #48) for which the majority of algorithms obtained a perfect F_1-score of $F_1 = 1.0$. These time series (except #48, which has four anomalies) only contain a single anomaly.

Although there are differences to supervised heartbeat classification algorithms, we found that our results are also roughly comparable to a few works in the literature: In [86], a F_1-score of $F_1 = 0.93$ is obtained, if we consider only the anomaly classes also used in this

work. In [169], the F_1-score on the test set for the overlapping anomaly classes is $F_1 = 0.84$, which is slightly lower than the score reported by us ($F_1 = 0.93$, Table 7.5).

Overall, we have shown that the TCN-AE architecture can produce competitive results on a challenging anomaly detection benchmark. We found that the TCN-AE architecture has several appealing properties which can be advantageous in time series anomaly detection, some of which we list in the following:

- Receptive field: Due to the hierarchical dilated convolutional structure, the size of the receptive field of the network can easily be scaled to the requirements of the problem.
- Skip Connections: Due to the introduced skip connections, TCN-AE is less sensitive towards the choice of the dilation rates (for example, we could choose dilation rates $1 \dots 64$ or $1 \dots 256$ and obtain similar results in both cases).
- Utilization of hidden Representations: Outputs of intermediate dilated convolutional layers are utilized, which allows using information processed at different time scales. This information is useful for obtaining an accurate reconstruction of the input and scanning for anomalies at different time scales.
- Fast Training: Due to the parallelizable convolution operation, time series can be processed very fast using GPUs (in this study, less than 50 seconds per time series, as shown in Table 7.8).
- Number of Weights: TCN-AE appears to potentially require less trainable weights than other architectures (e.g., recurrent neural networks) to obtain a good model accuracy. However, this claim has to be verified in more thorough studies in the future.

7.5 Conclusion and Possible Future Work

In this chapter, we introduced a novel temporal convolutional autoencoder (TCN-AE) architecture, which is designed to learn compressed representations of time series data in an unsupervised fashion. It is, to the best of our knowledge, the first work showing the combination of TCN and AE.

Starting with a baseline model and evaluating it on the Mackey-Glass anomaly benchmark (MGAB), we could already obtain promising results: The baseline TCN-AE performed best among 3 other state-of-the-art algorithms and could learn interesting representations of Mackey-Glass time series. However, we also noticed a few limitations which we addressed and led to a new TCN-AE algorithm with several modifications. We demonstrated the new algorithm's efficacy on a challenging real-world anomaly detection task, consisting of 30-minute electrocardiogram (ECG) readings of 25 patients, and could outperform several other unsupervised state-of-the-art algorithms on the investigated problem. Starting with a baseline model, we showed that several extensions are crucial to increase the overall performance. Particularly, we found that skip connections from the encoder's dilated convolutional layers to the bottleneck and skip connections from the decoder's dilated convolutional layers to the final reconstruction (output) layer improve the overall learning significantly.

Furthermore, the utilization of hidden representations (the outputs of the individual dilated convolutional layers) inside TCN-AE appears to be of considerable importance. The results suggest that temporal anomalies become more apparent on some time scales than on others. Another important finding was that TCN-AE was 5 times faster in our experiments than the second fastest algorithm (DNN-AE, see Table 7.8).

In summary, we demonstrated that it is possible to train a deep learning model without supervision, which can be used after training to detect anomalies in multivariate time series data. The novel TCN-AE model proposed in this work appears to be particularly well suited to learn long-range temporal patterns in complex quasi-periodic time series.

In our future research, we are planning to address several aspects of TCN-AE, which have not been thoroughly understood or investigated yet: (a) Application of the architecture to other challenging real-world anomaly detection problems. (b) Gaining more insights from the representations that TCN-AE learns unsupervisedly (Fig. 7.3). (c) Approaches for the determination of suitable anomaly thresholds with severely limited labeled data. (d) Analyzing time series with a higher ratio anomalous/normal data: In this work, we analyzed time series with not more than 250 anomalous events per patient. It is possible that TCN-AE also works for data with more anomalies. We plan to investigate our algorithm for settings with significantly larger anomaly ratios.

Chapter 8

Conclusion and Outlook

Time series anomaly detection is an intriguing and also challenging topic that will likely not lose any of its relevance in the future. Today, accurate anomaly detection algorithms are already critical in system health monitoring or predictive maintenance (PdM) applications or in detecting intrusions into organizational networks. The need for reliable anomaly detection methods is expected to increase in the coming years. Particularly in the industrial context, in the course of ongoing digitization, it will become necessary to analyze growing volumes of data in an automated manner using sophisticated and efficient algorithms.

During the work on this thesis, we had the opportunity to look into the broad domain of time series analysis and anomaly detection and add several contributions to this field. Our focus was on so-called *unsupervised* machine learning (ML) approaches, and we could introduce several novel algorithms with state-of-the-art performance. In unsupervised anomaly detection problems, a model attempts to learn the normal underlying behavior of a system without an external supervisor. The model's understanding of normality is then used to detect abnormal (anomalous) events. Unsupervised learning tasks are usually considered harder than their supervised counterparts since no target function exists, which defines the notion of normal. Instead, the algorithm has to learn to separate normal from anomalous behavior. The reason why unsupervised learning methods are used for anomaly detection is that labeled data are usually rather sparse. In the following, we will conclude this thesis with a short summary, by answering the research questions formulated at the beginning of the thesis and by discussing open questions, and giving an outlook on possible future work.

8.1 Discussion

We have presented four different unsupervised anomaly detection algorithms (some of them in several variants) in the course of this work: SORAD, DWT-MLEAD, LSTM-AD and, TCN-AE. SORAD learns a simple regression model to predict future values of a time series and simultaneously estimate the prediction errors' mean and variance. It operates fully online (or using small batches, or completely offline) and is up-and-running after a very short transient phase. Overall, due to its capabilities to adapt to new concepts, SORAD could outperform other state-of-the-art algorithms on Yahoo's Webscope S5 benchmark, which contains many non-stationary time series. Also, DWT-MLEAD can be either used in

online or offline settings. The main idea behind DWT-MLEAD is to examine a time series at different time scales using the discrete wavelet transform (DWT) to detect short-range and longer-range anomalies. For long-range anomalies, DWT-MLEAD performs better than SORAD. While SORAD and DWT-MLEAD were mostly tested on relatively short time series with less than 10 000 points, our deep learning approaches LSTM-AD and TCN-AE were applied to time series with length 100 000 or more. LSTM-AD uses a stack of recurrent long short-term memory (LSTM) neural networks. Similarly to SORAD, it is trained to predict normal time series behavior. However, LSTM-AD is extended in many aspects, which allows it to learn complex temporal patterns (such as in ECG data). For complex quasi-periodic time series, we found that the window-based error correction method introduced in Chapter 6 is crucial to improving the overall detection accuracy of LSTM-AD. While we did not test it yet, this method might also be beneficial for some of the other algorithms. It could also be integrated into the actual learning procedure (currently, it is an extra module applied after computing the prediction errors). LSTM-AD is trained offline and cannot adapt to new concepts in non-stationary time series. However, we found that it can be used for time series with weak non-stationary behavior, such as baseline wandering or signal quality changes.

For TCN-AE, we revisited the idea of analyzing time series at different time scales. This is achieved with a convolutional neural network architecture based on so-called dilated convolutions (which have their origin in DWT). TCN-AE is a reconstruction-based algorithm with a novel autoencoder architecture that can also be applied to unpredictable time series. It is possible to increase TCN-AE's receptive field exponentially with only a linear increase in the number of trainable weights. Due to the exponentially increasing receptive field in TCN-AE, it is the only algorithm that can learn long-term correlations in time series. Like LSTM-AD, TCN-AE can also work with time series that exhibit some baseline wandering or changes in the signal quality/noise. It significantly outperforms other state-of-the-art DL architectures in terms of precision & recall, computation time, and the number of trainable weights on the challenging ECG benchmark and MGAB (both introduced in Section 2.3).

Not surprisingly, our DL models LSTM-AD and TCN-AE require more training data than the other approaches, mainly due to the large number of trainable parameters. Hence, they are more likely to perform well when trained on longer time series with several ten-thousand points but not when trained on small data sets with less than a few thousand points. On the other hand, all algorithms can deal with very long time series and have no restrictions in this regard.

The main ingredient for DWT-MLEAD and TCN-AE to reliably detect short- and long-range anomalies simultaneously are their hierarchical temporal architecture and their capability to analyze time series at different time resolutions. The general idea in both approaches is that anomalies might become more apparent on some time scales than others. DWT-MLEAD realizes this with a decimating DWT. In practice, the DWT of a signal is computed by passing it through a series of filters and subsampling layers. The parameters of the filters are not trainable and depend on the choice of the mother wavelet. Similarly, TCN-AE uses dilated convolutional layers (which have their origin in DWTs), however, with

trainable filters. In both approaches, the temporal receptive field can easily be doubled by adding a new layer.

Theoretically, all our algorithms can process multivariate time series. However, in this work, DWT-MLEAD was only tested on univariate time series and would require some minor modifications for higher dimensions.

	Characteristics						
	Small Data Sets	Multivariate Time Series	Analysis at Different Time Scales	Long-term Correlations	Unpredictable Time Series	Weak Non-stationary Behavior	Strong Non-stationary Behavior
ADVec [173]	√					√	√
DNN-AE [46, 58]		√			√	√	
DWT-MLEAD (Ch. 5)	√	(√)	√		√	√	√
LSTM-AD (Ch. 6)		√	(√)	(√)		√	
LSTM-ED [108]		√	(√)		√	√	
NuPIC [160]	√	√				√	√
SORAD (Ch. 4)	√	√				√	√
TCN-AE (Ch. 7)		√	√	√	√	√	

Table 8.1: This table summarizes the suitability of various time series anomaly detection algorithms, given different time series characteristics. A checkmark (in parentheses) indicates that an algorithm can (partially) handle time series with the specified characteristic.

Based on the time series characteristics described in Section 2.4, we tried to indicate in Table 8.1 for all algorithms used in this work, which ones might be suited for which characteristics. For example, we expect that DL algorithms are generally less well suited for data sets that are relatively small (less than a few thousand points). Most algorithms investigated in this work should be applicable to multivariate time series or to time series with weak non-stationary behavior. Algorithms that can run entirely online, such as SORAD, generally tend to perform better than offline algorithms on time series with strong non-stationary behavior.

In order to learn long-term correlations, a long temporal memory is required. Algorithms, such as LSTM-ED or DNN-AE, designed to encode and reconstruct short sub-sequences, do not scale well for increasing sub-sequence lengths and usually cannot learn long-term correlations very well. On the other hand, TCN-AE exhibits a much longer memory than recurrent architectures (e.g., LSTM or GRU) with the same capacity (network size) and does not suffer from vanishing/exploding gradients. Hence, TCN-AE could be a reasonable choice if long-term correlations in time series have to be learned. As described before, SORAD, NuPIC, and LSTM-AD will likely not perform well on unpredictable (in the sense of forecastable) time series since all three approaches rely on single-step or multi-step ahead prediction.

8.2 Conclusions

In Sec. 1.1.3, we posed research questions that motivated our work in the following. At this point, we would like to revisit these questions and highlight the contributions that emerged in their context.

Q1: Is it possible to successfully train and apply novel unsupervised machine learning models for anomaly detection and do they advance the state of the art?

A1: Although it is challenging to analyze time series in an unsupervised fashion, especially when considering the data's temporal nature, we can answer this question mostly positively. Overall, we introduced four different anomaly detection algorithms for time series, which can be used in different contexts and show competitive performance when benchmarked against other state-of-the-art algorithms: The online SORAD could significantly outperform other algorithms [173, 160] on non-stationary time series with short-term anomalies. Our second online algorithm, DWT-MLEAD, is able to analyze time series on different time scales and detect short-term and longer-term anomalies. It is better than other algorithms [173, 160] when applied to benchmarks with anomalies being diverse in their time scales. LSTM-AD is a DL model with several enhancements that obtains state-of-the-art results on quasi-periodic time series such as ECG signals or the Mackey-Glass Anomaly Benchmark (MGAB). TCN-AE is a novel reconstruction-based DL approach that analyzes time series at different time scales and can learn long-term relationships. For a real-world anomaly detection task for ECG time series, TCN-AE outperforms other strong (DL) anomaly detection algorithms [160, 46, 108] with respect to detection accuracy, model size, and computation time, significantly improving the state of the art. All algorithms presented in this work are *unsupervised* and do not require any ground truth labels for the training process. Only for evaluation purposes, the anomaly labels are partly required, for example, to configure the anomaly threshold of all algorithms in a way that allows a fair comparison (e.g., by achieving equal accuracy, EAC). Furthermore, none of the algorithms requires solely normal data in order to learn the regular behavior of a time series (up to 5% of the data was anomalous in the examples used in our experiments), demonstrating the algorithms noise resilience. Contrary to other state-of-the-art algorithms, all of our algorithms are applicable to time series with two or more dimensions. However, due to the lack of suitable benchmarking data, we did not experiment with high-dimensional time series.

Q2: Given certain characteristics of the time series data, can we advise which algorithm is most suited for detecting anomalies?

A2: To answer this research question, we investigated two aspects: (1) What are general characteristics/properties of time series which are important in the context of anomaly detection and for choosing suitable algorithms? (2) How do existing state-of-the-art algorithms deal with these properties and can we find better approaches to analyze time series with different characteristics? Based on the different problems that we studied, we found several answers to both questions in the individual chapters, which are discussed on a higher level considering the overall context in Section 8.1.

In summary, we identified several recurring characteristics appearing in time series. Each proposed algorithm addresses one or more of these characteristics. However, there is no universally applicable algorithm. All algorithms have their justification for different problems. An important decision criterion for the choice of the algorithm is the data size. Deep learning approaches (TCN-AE, LSTM-AD) tend to be less well suited for small data sets. Another factor for many problems is to analyze a time series and to detect anomalies across different frequency scales. The capability to learn long-term dependencies is necessary for many types of problems and requires that the model has a large temporal receptive field (TCN-AE) or an efficient temporal memory (LSTM-AD). Online and offline algorithms should be able to process time series with weak forms of non-stationarity (e.g. baseline wandering, changes in signal quality) which are ubiquitous in real-world problems. For strong non-stationary behavior, algorithms with online adaptability (DWT-MLEAD, SORAD) are needed that are stable on the one side and, on the other side, can learn new concepts in the data. Finally, the dimension of a time series plays an important role in the choice of the algorithm. Some algorithms are better suited for multivariate time series than others. Although already mentioned (and although not directly related to the nature of a time series), we want to emphasize that the available time series data usually does not permit to train supervised learning models in practice, due to the sparse amount of labeled data. Hence, all our algorithms are trained in an unsupervised fashion.

Q3: How can online learning methods be successfully used for anomaly detection in time series or data streams?
A3: We investigated this research question mainly in Chapter 4 and Chapter 5. Note that while all our models can be run online on new data after training, only SORAD and DWT-MLEAD are online adaptable; hence, we do not have to train them offline. Instead, they only require a very short transient phase to be ready for use, and they can continually adapt to new concepts in the data. SORAD uses the recursive least squares (RLS) algorithm to learn its prediction model and estimates the mean and (co-) variance of the prediction errors online. Additionally, it is possible to add a certain amount of forgetting to the RLS model and to the estimation of the error distribution, which enables the online adaptability of the algorithm. Although the regression model is linear, it can be augmented with non-linear features, such as polynomials or radial basis functions (RBFs). DWT-MLEAD algorithm can compute the (causal and decimating) discrete wavelet transform entirely online. For individual frequency scales of the current DWT, the algorithm estimates a mean vector and covariance matrix in an online fashion. It is also possible to add forgetting to DWT-MLEAD so that the algorithm can adapt itself in non-stationary environments. The online adaptability of SORAD and DWT-MLEAD appears to be beneficial for time series with concept changes or drifts, as our experiments showed. Furthermore, both algorithms did not show any signs of numerical instabilities or other diverging behavior.

LSTM-AD and TCN-AE are not online-adaptable in the sense that they continue adjusting their weights after the (offline) training. However, both algorithms are capable of dealing with non-stationary artifacts such as baseline wandering.

So far it is an open question to us whether DL models can be trained online. Today, most online learning algorithms are designed to learn shallow models with convex optimization (e.g., linear least squares) techniques. In this work we did not investigate to which extent DL models, with highly non-convex objective functions, can be trained in an online fashion. Hence, this research question cannot yet be answered conclusively and has much potential for future investigations.

8.3 Future Work

Although we explored some challenges mentioned in the introduction and present novel approaches to time series anomaly detection, many challenges beyond this work remain that research could address in future work. Apart from the more detailed discussions in the previous chapters on possible further work on individual problems and improvements to particular algorithms, we see some more general points which are worth investigating in the future:

Need for Better Benchmarks Although several benchmarks for time series anomaly detection are publicly available and we could also design a challenging synthetic benchmark based on Mackey-Glass time series and construct a complex benchmark based on ECG data, we have the impression that many of the current data sets do not adequately reflect the reality that will prevail in the future. It will become increasingly difficult to assess algorithms' performance regarding their practicability for real-world problems based on the current benchmarks. Many currently popular benchmarks contain only relatively short (several thousand points) and only one-dimensional time series. However, applications today collect thousands of high-dimensional data points in a short time. Especially for benchmarking DL approaches, many current benchmarks are insufficient due to their somewhat limited size. Other issues are triviality (i.e., simple thresholding methods can detect some anomalies), insertion of unrealistic synthetic anomalies into normal data, or the translation of traditional classification problems into anomaly detection problems by simply re-labeling minority classes as anomalies. Another common problem is that not always guidelines exist on how to measure an algorithm's performance. For the same benchmark, researchers assess algorithms' performance in very different ways and report performance metrics that are incomparable to other works. For these reasons, we believe there is a need for better benchmarks in time series anomaly detection, which are relevant to practice and widely accepted among the research community and allow a fair and thorough evaluation of different approaches. These new benchmarks also require standardized rule sets that clearly state how to assess the performance of an algorithm.

Interpretability Another issue that will likely get more relevant in the context of (time series) anomaly detection is the *interpretability* of models. Interpretability is concerned with the problem of getting some explanation or understanding for the decisions made by our

models. In the context of anomaly detection, interpretability is linked to the question of whether an algorithm can also identify the underlying cause of a detected anomaly. Usually, an anomaly is associated with a problem in the monitored system or process. Hence, in many real-world applications, it is not sufficient to solely detect anomalous behavior. To fix the problem and possibly prevent further harm, it is also critical to identify the actual source of the problem and find a way to handle this problem. In some cases, human experts might be able to analyze the anomalous pattern and recognize the cause. However, especially in high-dimensional & high-frequency time series, where anomalies might occur only in a small subset of the high-dimensional space, this becomes increasingly difficult. In other cases, it might be of interest to understand an algorithm's decision to reduce algorithmic bias (e.g., caused by biased training data) or to justify individual decisions. While researchers have invested much effort in developing anomaly detection algorithms in recent years, interpretable approaches have not received as much attention yet. Overall, we see much potential in this area.

Integrating Expert Feedback All the anomaly detection algorithms presented in this work and many approaches described in the literature are trained in an unsupervised fashion. The main reason for this is that the available labeled data is sparse, and the few labeled instances are needed for tuning and validation purposes. Unsurprisingly, many anomaly detection models have a relatively low performance when initially deployed. That is, the models produce many false alarms or overlook real anomalous behavior. One idea to improve the overall prediction quality could be to integrate expert feedback into the model during operation since human experts (such as machine operators) are commonly involved in anomaly detection tasks and have to check and acknowledge alarms. With such a human-in-the-loop (HITL) approach, the model could gradually adapt itself and improve its predictions. However, one has to keep in mind that also expert feedback will mostly be sparse and has to be handled efficiently so that the improvement will become apparent in a reasonable time. Although some work exists on how to incorporate expert feedback into anomaly detection [32, 138, 136], it is a mostly open research question how such additional information can be utilized efficiently, apart from simple adjustments of the anomaly threshold.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., et al.: TensorFlow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. pp. 265–283. OSDI'16, USENIX Association, Berkeley, CA, USA (2016)

[2] Adler, A., Elad, M., Hel-Or, Y., Rivlin, E.: Sparse coding with anomaly detection. Journal of Signal Processing Systems 79(2), 179–188 (May 2015)

[3] Ahmad, S.: Running swarms (May 2017), `http://nupic.docs.numenta.org/0.6.0/guide-swarming.html`

[4] Ahmed, T., Coates, M., Lakhina, A.: Multivariate online anomaly detection using kernel recursive least squares. In: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM). pp. 625–633. IEEE (2007)

[5] Alarcon Aquino, V.: Anomaly Detection and Prediction in Communication Networks Using Wavelet Transforms. Ph.D. thesis, Imperial College London (2003)

[6] Alarcon-Aquino, V., Barria, J.A.: Anomaly detection in communication networks using wavelets. IET Proceedings-Communications 148(6), 355–362 (2001)

[7] Alickovic, E., Subasi, A.: Medical decision support system for diagnosis of heart arrhythmia using DWT and random forests classifier. Journal of Medical Systems 40(4), 108 (2016)

[8] Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. Knowledge and Information Systems (KAIS) 51(2), 339–367 (2017)

[9] An, J., Cho, S.: Variational autoencoder based anomaly detection using reconstruction probability. Special Lecture on IE 2(1), 1–18 (2015)

[10] Anon.: Auf dem Weg zur Lösung der ICE-Probleme. Eisenbahn-Revue International 10, 446–447 (1998)

[11] Ansmann, G.: Efficiently and easily integrating differential equations with JiTCODE, JiTCDDE, and JiTCSDE. Chaos 28(4), 043116 (2018)

[12] Arpit, D., Zhou, Y., Ngo, H., Govindaraju, V.: Why regularized auto-encoders learn sparse representation? In: Proceedings of the International Conference on Machine Learning (ICML). pp. 136–144. PMLR (2016)

[13] Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. CoRR abs/1803.01271 (2018)

[14] Bergstra, J., et al.: Hyperopt: A python library for model selection and hyperparameter optimization. Computational Science & Discovery 8(1), 014008 (2015)
[15] Bishop, C.M.: Pattern Recognition and Machine Learning. springer (2006)
[16] Blázquez-García, A., Conde, A., Mori, U., Lozano, J.A.: A review on outlier/anomaly detection in time series data. arXiv preprint arXiv:2002.04236 (2020)
[17] Bontempi, G., Ben Taieb, S.: Statistical foundations of machine learning, chapter: Recursive least squares. Université Libre de Bruxelles (2008)
[18] Bontemps, L., McDermott, J., Le-Khac, N.A., et al.: Collective anomaly detection based on long short term memory recurrent neural network. In: Proceedings of the International Conference on Future Data and Security Engineering (FDSE). pp. 141–152. Springer (2016)
[19] Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp. 93–104 (2000)
[20] de Bruijn, B., Nguyen, T.A., Bucur, D., Tei, K.: Benchmark datasets for fault detection and classification in sensor data. In: Proceedings of the 5th International Conference on Sensor Networks (SENSORNETS). pp. 185–195 (2016)
[21] Carpenter, G.A., Grossberg, S.: ART 2: Self-organization of stable category recognition codes for analog input patterns. Applied Optics 26(23), 4919–4930 (1987)
[22] Chakraborty, G., Kamiyama, T., Takahashi, H., Kinoshita, T.: An efficient anomaly detection in quasi-periodic time series data — A case study with ECG. In: Rojas, I., Pomares, H., Valenzuela, O. (eds.) Time Series Analysis and Forecasting, pp. 147–157. Springer International Publishing (2018)
[23] Chalapathy, R., Chawla, S.: Deep learning for anomaly detection: A survey. CoRR abs/1901.03407 (2019)
[24] Chalapathy, R., Menon, A.K., Chawla, S.: Anomaly detection using one-class neural networks. CoRR abs/1802.06360 (2018)
[25] Chan, D.M., Rao, R., Huang, F., Canny, J.F.: GPU accelerated t-distributed stochastic neighbor embedding. Journal of Parallel and Distributed Computing (JPDC) 131, 1–13 (2019)
[26] Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Surveys (CSUR) 41(3), 15 (2009)
[27] Chaovalit, P., Gangopadhyay, A., Karabatis, G., Chen, Z.: Discrete wavelet transform-based time series analysis and mining. ACM Computing Surveys (CSUR) 43(2), 6:1–6:37 (Feb 2011)
[28] Chauhan, S., Vig, L.: Anomaly detection in ECG time signals via deep long short-term memory networks. In: Proceedings of IEEE International Conference on Data Science and Advanced Analytics (DSAA). pp. 1–7. IEEE (2015)
[29] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
[30] Chollet, F., et al.: Keras. `https://keras.io` (2015)

[31] Cook, A.A., Mısırlı, G., Fan, Z.: Anomaly detection for IoT time-series data: A survey. IEEE Internet of Things Journal (IoT-J) 7(7), 6481–6494 (2019)
[32] Das, S., Wong, W.K., Dietterich, T., Fern, A., Emmott, A.: Incorporating expert feedback into active anomaly discovery. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 853–858. IEEE (2016)
[33] Dasgupta, D., Forrest, S.: Novelty detection in time series data using ideas from immunology. In: Proceedings of the 5th International Conference on Intelligent Systems (IS). pp. 82–87. Citeseer (1996)
[34] Dasgupta, D., Forrest, S.: An anomaly detection algorithm inspired by the immune system. In: Artificial Immune Systems and Their Applications, pp. 262–277. Springer (1999)
[35] Dau, H.A., Ciesielski, V., Song, A.: Anomaly detection using replicator neural networks trained on examples of one class. In: Asia-Pacific Conference on Simulated Evolution and Learning (SEAL). pp. 311–322. Springer (2014)
[36] Dauphin, Y.N., Fan, A., Auli, M., Grangier, D.: Language modeling with gated convolutional networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research, vol. 70, pp. 933–941. PMLR (2017)
[37] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 248–255. IEEE (2009)
[38] Deutsche Bahn AG: Zug-Check im Vorbeifahren: Wayside-Monitoring. `https://inside.bahn.de/wayside-monitoring/` (2017), last accessed: 30.30.2021
[39] Di Mattia, F., Galeone, P., De Simoni, M., Ghelfi, E.: A survey on GANs for anomaly detection. CoRR abs/1906.11632 (2019)
[40] Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. CoRR abs/1603.07285 (2016)
[41] Engel, Y., Mannor, S., Meir, R.: The kernel recursive least-squares algorithm. IEEE Transactions on Signal Processing 52(8), 2275–2285 (2004)
[42] Esslinger, V., Kieselbach, R., Koller, R., Weisse, B.: The railway accident of Eschede – technical background. Engineering Failure Analysis 11(4), 515–535 (2004)
[43] expedia.com: ExpediaDotCom/ adaptive-alerting. `https://github.com/ExpediaDotCom/adaptive-alerting` (2020)
[44] Filonov, P., Kitashov, F., Lavrentyev, A.: RNN-based early cyber-attack detection for the Tennessee Eastman process. CoRR abs/1709.02232 (2017)
[45] Filonov, P., Lavrentyev, A., Vorontsov, A.: Multivariate industrial time series with cyber-attack simulation: Fault detection using an LSTM-based predictive data model. CoRR abs/1612.06676 (2016)
[46] Fischer, M., et al.: Anomaly detection on time series: An evaluation of deep learning methods. `https://github.com/KDD-OpenSource/DeepADoTS` (2019)
[47] Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Computing Surveys (CSUR) 46(4), 44:1–44:37 (2014)

[48] Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional sequence to sequence learning. CoRR abs/1705.03122 (2017)

[49] George, D., Hawkins, J.: Towards a mathematical theory of cortical micro-circuits. PLOS Computational Biology 5(10), e1000532 (2009)

[50] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)

[51] Goh, J., Adepu, S., Tan, M., Lee, Z.S.: Anomaly detection in cyber physical systems using recurrent neural networks. In: 18th International Symposium on High Assurance Systems Engineering (HASE). pp. 140–145. IEEE (2017)

[52] Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation 101(23), e215–e220 (2000)

[53] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial networks. CoRR abs/1406.2661 (2014)

[54] Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: A search space odyssey. IEEE Transactions on Neural Networks and Learning Systems (TNNLS) 28(10), 2222–2232 (2017)

[55] Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: A survey. IEEE Transactions on Knowledge and Data Engineering (TKDE) 26(9), 2250–2267 (2013)

[56] Hannun, A.Y., Rajpurkar, P., Haghpanahi, M., Tison, G.H., Bourn, C., Turakhia, M.P., Ng, A.Y.: Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature Medicine 25(1), 65 (2019)

[57] Hariri, S., Kind, M.C., Brunner, R.J.: Extended isolation forest. IEEE Transactions on Knowledge and Data Engineering (TKDE) 33(4), 1479–1489 (2021)

[58] Hawkins, S., He, H., Williams, G.J., Baxter, R.A.: Outlier detection using replicator neural networks. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK). Lecture Notes in Computer Science, vol. 2454, pp. 170–180. Springer (2002)

[59] Haykin, S.: Adaptive Filtering Theory. Pearson, 5th Edition (2013)

[60] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)

[61] He, Y., Zhao, J.: Temporal convolutional networks for anomaly detection in time series. In: Journal of Physics: Conference Series. vol. 1213, p. 042050. IOP Publishing (2019)

[62] Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6(02), 107–116 (1998)
[63] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
[64] Hodge, V., Austin, J.: A survey of outlier detection methodologies. Artificial Intelligence Review 22(2), 85–126 (2004)
[65] Holschneider, M., Kronland-Martinet, R., Morlet, J., Tchamitchian, P.: A real-time algorithm for signal analysis with the help of the wavelet transform. In: Combes, J.M., Grossmann, A., Tchamitchian, P. (eds.) Wavelets, pp. 286–297. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
[66] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4700–4708 (2017)
[67] Hundman, K., Constantinou, V., Laporte, C., Colwell, I., Söderström, T.: Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In: Guo, Y., Farooq, F. (eds.) Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD). pp. 387–395. ACM (2018)
[68] Hyndman, R.J., Wang, E., Laptev, N.: Large-scale unusual time series detection. In: International Conference on Data Mining Workshop (ICDMW). pp. 1616–1619. IEEE (2015)
[69] Ibidunmoye, O., Metsch, T., Elmroth, E.: Real-time detection of performance anomalies for cloud services. In: International Symposium on Quality of Service (IWQoS). pp. 1–9. IEEE (2016)
[70] Jain, R., Chlamtac, I.: The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations. Communications of the ACM 28(10), 1076–1085 (1985)
[71] Jiang, W., Hong, Y., Zhou, B., He, X.: A GAN-based anomaly detection approach for imbalanced industrial time series. IEEE Access 7, 143608–143619 (2019)
[72] Jones, J., Palmer, L.: An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. Journal of Neurophysiology 2, 1233–1258 (1987)
[73] Kalchbrenner, N., Grefenstette, E., Blunsom, P.: A convolutional neural network for modelling sentences. In: Annual Meeting of the Association for Computational Linguistics (ACL). pp. 655–665. Baltimore, Maryland (2014)
[74] Kanarachos, S., Mathew, J., Chroneos, A., Fitzpatrick, M.: Anomaly detection in time series data using a combination of wavelets, neural networks and Hilbert transform. In: Proceedings of the International Conference on Information, Intelligence, Systems and Applications (IISA). pp. 1–6 (2015)
[75] Katser, I.D., Kozitsin, V.O.: Skoltech anomaly benchmark (SKAB) (2020)
[76] KDD cup 1999 (1999), http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[77] Keogh, E., Lin, J., Fu, A.: Hot SAX: Efficiently finding the most unusual time series subsequence. In: Fifth IEEE International Conference on Data Mining (ICDM'05). pp. 8–pp. IEEE (2005)

[78] Keogh, E., Lin, J., Lee, S.H., Van Herle, H.: Finding the most unusual time series subsequence: Algorithms and applications. Knowledge and Information Systems (KAIS) 11(1), 1–27 (2007)

[79] Kieu, T., Yang, B., Guo, C., Jensen, C.S.: Outlier detection for time series with recurrent autoencoder ensembles. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 2725–2732 (2019)

[80] Kieu, T., Yang, B., Jensen, C.S.: Outlier detection for multidimensional time series using deep neural networks. In: Proceedings of the International Conference on Mobile Data Management (MDM). pp. 125–134. IEEE (2018)

[81] Kim, S.S., Reddy, A.N., Vannucci, M.: Detecting traffic anomalies using discrete wavelet transform. In: International Conference on Information Networking. pp. 951–961. Springer (2004)

[82] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015)

[83] Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. In: Bengio, Y., LeCun, Y. (eds.) Proceedings of the International Conference on Learning Representations (ICLR) (2014)

[84] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J.R.R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., Ferrari, V.: The open images dataset V4. International Journal of Computer Vision (IJCV) 128(7), 1956–1981 (2020)

[85] Kwon, D., Ko, K., Vannucci, M., Reddy, A.N., Kim, S.: Wavelet methods for the detection of anomalies and their application to network traffic analysis. Quality and Reliability Engineering International (QREI) 22(8), 953–969 (2006)

[86] Lagerholm, M., Peterson, C., Braccini, G., Edenbrandt, L., Sornmo, L.: Clustering ECG complexes using Hermite functions and self-organizing maps. IEEE Transactions on Biomedical Engineering (T-BME) 47(7), 838–848 (2000)

[87] Laptev, N., Amizadeh, S.: Yahoo anomaly detection dataset webscope S5 (2015), `http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70`

[88] Laptev, N., Amizadeh, S., Flint, I.: Generic and scalable framework for automated time-series anomaly detection. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). p. 1939–1947. Association for Computing Machinery, New York, NY, USA (2015)

[89] Lavin, A., S. Ahmad, S.: Evaluating real-time anomaly detection algorithms – the Numenta anomaly benchmark. In: Proceedings of the International Conference on Machine Learning and Applications (ICMLA) (2015)

[90] Lemos, A.P., Tierra-Criollo, C., Caminhas, W.: ECG anomalies identification using a time series novelty detection technique. In: IV Latin American Congress on Biomedi-

cal Engineering 2007, Bioengineering Solutions for Latin America Health. pp. 65–68. Springer (2007)

[91] Li, D., et al.: Anomaly detection with generative adversarial networks for multivariate time series. CoRR abs/1809.04758 (2018)

[92] Li, D., et al.: MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In: Proceedings of the International Conference on Artificial Neural Networks (ICANN). pp. 703–716. Springer (2019)

[93] Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Advances in Neural Information Processing Systems (NIPS). pp. 6389–6399 (2018)

[94] Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD). pp. 2–11 (2003)

[95] Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: A novel symbolic representation of time series. Data Mining and knowledge discovery (DMKD) 15(2), 107–144 (2007)

[96] Lin, M., Chen, Q., Yan, S.: Network in network. In: Bengio, Y., LeCun, Y. (eds.) Proceedings of the 2nd International Conference on Learning Representations (ICLR) (2014)

[97] LinkedIn: linkedin/luminol. `https://github.com/linkedin/luminol` (2015)

[98] Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: Eighth IEEE International Conference on Data Mining (ICDM). pp. 413–422. IEEE (2008)

[99] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3431–3440 (2015)

[100] Lu, W., Ghorbani, A.A.: Network anomaly detection based on wavelet analysis. EURASIP Journal on Advances in Signal Processing 2009, 4 (2009)

[101] Luz, E.J.d.S., Schwartz, W.R., Cámara-Chávez, G., Menotti, D.: ECG-based heartbeat classification for arrhythmia detection: A survey. Computer methods and programs in biomedicine 127, 144–164 (2016)

[102] Ma, J., Perkins, S.: Online novelty detection on temporal sequences. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 613–618 (2003)

[103] Ma, J., Perkins, S.: Time-series novelty detection using one-class support vector machines. In: Proceedings of the International Joint Conference on Neural Networks. vol. 3, pp. 1741–1745. IEEE (2003)

[104] Ma, J., Theiler, J., Perkins, S.: Accurate on-line support vector regression. Neural Computation 15(11), 2683–2703 (2003)

[105] van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. Journal of Machine Learning Research (JMLR) 9, 2579–2605 (2008)

[106] Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science 197(4300), 287–289 (1977)

[107] Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN). pp. 89–94 (2015)

[108] Malhotra, P., et al.: LSTM-based encoder-decoder for multi-sensor anomaly detection. CoRR abs/1607.00148 (2016)

[109] Markou, M., Singh, S.: Novelty detection: A review – part 1: Statistical approaches. Signal Processing 83(12), 2481–2497 (2003)

[110] Markou, M., Singh, S.: Novelty detection: A review – part 2: Neural network based approaches. Signal Processing 83(12), 2499–2521 (2003)

[111] Meyer, Y., Salinger, D.: Wavelets and Operators, Cambridge Studies in Advanced Mathematics, vol. 1. Cambridge University Press (1995)

[112] Moody, G.B., Mark, R.G.: PhysioNet: The MIT-BIH arrhythmia database. `https://www.physionet.org/physiobank/database/mitdb/` (1992)

[113] Moody, G.B., Mark, R.G.: The impact of the MIT-BIH arrhythmia database. IEEE Engineering in Medicine and Biology Magazine (EMBS) 20(3), 45–50 (2001)

[114] Moya, M.M., Hush, D.R.: Network constraints and multi-objective optimization for one-class classification. Neural Networks 9(3), 463–474 (Apr 1996)

[115] Munir, M., et al.: DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. IEEE Access 7, 1991–2005 (2019)

[116] Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML). pp. 807–814 (2010)

[117] Nason, G.: wavethresh: Wavelets Statistics and Transforms (2016), `https://CRAN.R-project.org/package=wavethresh`, R package version 4.6.8

[118] Ng, A., et al.: Sparse autoencoder. CS294A Lecture Notes 72(2011), 1–19 (2011)

[119] Nguyen, D., Kefalas, M., Yang, K., Apostolidis, A., Olhofer, M., Limmer, S., Bäck, T.: A review: Prognostics and health management in automotive and aerospace. International Journal of Prognostics and Health Management 10(2), 35 (2019)

[120] Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: Proceedings of the IEEE international conference on computer vision (ICCV). pp. 1520–1528 (2015)

[121] Nolle, F., Badura, F., Catlett, J., Bowser, R., Sketch, M.: CREI-GARD, a new concept in computerized arrhythmia monitoring systems. Computers in Cardiology 13, 515–518 (1986)

[122] Odena, A., Dumoulin, V., Olah, C.: Deconvolution and checkerboard artifacts. Distill 1(10), e3 (2016)

[123] Oh, D.Y., Yun, I.D.: Residual error based anomaly detection using auto-encoder in SMD machine sound. Sensors 18(5) (2018)

[124] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A.W., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. In: Proceedings of the 9th ISCA Speech Synthesis Workshop. p. 125. ISCA (2016)

[125] Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning for natural language processing. IEEE Transactions on Neural Networks and Learning Systems (TNNLS) (2020)

[126] Oxford Dictionary of English: anomaly, n. Oxford University Press, Oxford, 3 edn. (2016), https://www.unilexids.de/

[127] Pan, J., Tompkins, W.J.: A real-time QRS detection algorithm. IEEE Transactions on Biomedical Engineering (T-BME) 32(3), 230–236 (1985)

[128] Pang, G., Shen, C., Cao, L., van den Hengel, A.: Deep learning for anomaly detection: A review. CoRR abs/2007.02500 (2020)

[129] Park, D., Hoshi, Y., Kemp, C.C.: A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. IEEE Robotics and Automation Letters 3(3), 1544–1551 (2018)

[130] Paszke, A., et al.: PyTorch: An imperative style, high-performance deep learning library. In: Wallach, H., et al. (eds.) Advances in Neural Information Processing Systems (NIPS), pp. 8024–8035. Curran Assoc. (2019)

[131] Patcha, A., Park, J.M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks 51(12), 3448–3470 (2007)

[132] Pereira, J., Silveira, M.: Learning representations from healthcare time series data for unsupervised anomaly detection. In: Proceedings of the International Conference on Big Data and Smart Computing (BigComp). pp. 1–7 (Feb 2019)

[133] Pereira, J., Silveira, M.: Unsupervised anomaly detection in energy time series data using variational recurrent autoencoders with attention. In: Wani, M.A., et al. (eds.) Proceedings of the International Conference on Machine Learning and Applications (ICMLA). pp. 1275–1282. IEEE (2018)

[134] Pereira, J., Silveira, M.: Unsupervised representation learning and anomaly detection in ECG sequences. International Journal of Data Mining and Bioinformatics 22(4), 389–407 (2019)

[135] Pimentel, M.A., Clifton, D.A., Clifton, L., Tarassenko, L.: A review of novelty detection. Signal Processing 99, 215–249 (2014)

[136] Pimentel, T., Monteiro, M., Veloso, A., Ziviani, N.: Deep active learning for anomaly detection. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2020)

[137] Price, G.R.: Extension of covariance selection mathematics. Annals of Human Genetics 35(4), 485–490 (1972)

[138] Raginsky, M., Willett, R.M., Horn, C., Silva, J., Marcia, R.F.: Sequential anomaly detection in the presence of noise and limited feedback. IEEE Transactions on Information Theory 58(8), 5544–5562 (2012)

[139] Rai, H.M., Trivedi, A., Shukla, S.: ECG signal processing for abnormalities detection using multi-resolution wavelet transform and artificial neural network classifier. Measurement 46(9), 3238–3246 (2013)

[140] Remy, P.: Temporal convolutional networks for Keras. GitHub repository: `https://github.com/philipperemy/keras-tcn` (2020)

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

[142] Rousseeuw, P.J., Driessen, K.V.: A fast algorithm for the minimum covariance determinant estimator. Technometrics 41(3), 212–223 (1999)

[143] Sahoo, S., Kanungo, B., Behera, S., Sabut, S.: Multiresolution wavelet transform based feature extraction and ECG classification to detect cardiac abnormalities. Measurement 108, 55–66 (2017)

[144] Sakurada, M., Yairi, T.: Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA). pp. 4–11 (2014)

[145] Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems (NIPS). pp. 901–909 (2016)

[146] Saurav, S., Malhotra, P., TV, V., Gugulothu, N., Vig, L., Agarwal, P., Shroff, G.: Online anomaly detection with concept drift adaptation using recurrent neural networks. In: Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (CODS-COMAD). pp. 78–87 (2018)

[147] Schölkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J., Platt, J.C.: Support vector method for novelty detection. In: Advances in Neural Information Processing Systems (NIPS). pp. 582–588 (2000)

[148] Shahabi, C., Chung, S., Safar, M.: A wavelet-based approach to improve the efficiency of multi-level surprise mining. In: PAKDD International Workshop on Mining Spatial and Temporal Data. vol. 2001 (2001)

[149] Shen, L., Li, Z., Kwok, J.: Timeseries anomaly detection using temporal hierarchical one-class network. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems (NeurIPS). vol. 33, pp. 13016–13026. Curran Associates, Inc. (2020)

[150] Sherman, J., Morrison, W.J.: Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. The Annals of Mathematical Statistics 21(1), 124–127 (1950)

[151] Sivaraks, H., Ratanamahatana, C.A.: Robust and accurate anomaly detection in ECG artifacts using time series Motif discovery. Computational and Mathematical Methods in Medicine 2015 (2015)

[152] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research (JMLR) 15(1), 1929–1958 (2014)

[153] Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W., Pei, D.: Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD). pp. 2828–2837 (2019)

[154] Suh, S., Chae, D.H., Kang, H.G., Choi, S.: Echo-state conditional variational autoencoder for anomaly detection. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN). pp. 1015–1022. IEEE (2016)

[155] Świechowski, M., Park, H., Mańdziuk, J., Kim, K.J.: Recent advances in general game playing. The Scientific World Journal 2015 (2015)

[156] Szegedy, C., et al.: Going deeper with convolutions. CoRR abs/1409.4842 (2014)

[157] Talagala, P.D., Hyndman, R.J., Smith-Miles, K., Kandanaarachchi, S., Muñoz, M.A.: Anomaly detection in streaming nonstationary temporal data. Journal of Computational and Graphical Statistics (JCGS) 29(1), 13–27 (2020)

[158] Tan, S.C., Ting, K.M., Liu, T.F.: Fast anomaly detection for streaming data. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI) (2011)

[159] Tatbul, N., Lee, T.J., Zdonik, S., Alam, M., Gottschlich, J.: Precision and recall for time series. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NeurIPS). pp. 1924–1934 (2018)

[160] Taylor, M., et al.: numenta/nupic: 1.0.5. `https://doi.org/10.5281/zenodo.1257382` (2018)

[161] Thill, M., Däubener, S., Konen, W., Bäck, T.: Anomaly detection in electrocardiogram readings with stacked LSTM networks. In: ITAT. CEUR Workshop Proceedings, vol. 2473, pp. 17–25 (2019)

[162] Thill, M., Konen, W., Bäck, T.: Anomaly detection in time series with discrete wavelet transforms and maximum likelihood estimation. In: Hoffmann, F., Hüllermeier, E. (eds.) Proceedings 27. Workshop Computational Intelligence (GMA-CI). pp. 67–71. Universitätsverlag Karlsruhe (2017)

[163] Thill, M., Konen, W., Bäck, T.: Online anomaly detection on the webscope S5 dataset: A comparative study. In: Proceedings of the IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS). pp. 1–8. Springer (2017)

[164] Thill, M., Konen, W., Bäck, T.: Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation. In: Valenzuela, O., Rojas, I., et al. (eds.) Proceedings of the International Conference on Time Series (ITISE). vol. 2, pp. 11–23 (2017)

[165] Thill, M., Konen, W., Bäck, T.: Online adaptable time series anomaly detection with discrete wavelet transforms and multivariate Gaussian distributions. Archives of Data Sciences (AoDSA), Series A (2018)

[166] Thill, M., Konen, W., Bäck, T.: Time series encodings with temporal convolutional networks. In: Filipic, B., Minisci, E.A., Vasile, M. (eds.) Bioinspired Optimization

Methods and Their Applications - 9th International Conference, BIOMA 2020, Brussels, Belgium, November 19-20, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12438, pp. 161–173. Springer (2020)

[167] Thill, M., Konen, W., Bäck, T.: MGAB: The mackey-glass anomaly benchmark (2020)

[168] Thomas, M., Das, M.K., Ari, S.: Automatic ECG arrhythmia classification using dual tree complex wavelet based features. AEU - International Journal of Electronics and Communications 69(4), 715–721 (Apr 2015)

[169] Tsipouras, M.G., Fotiadis, D.I., Sideris, D.: Arrhythmia classification using the RR-interval duration signal. In: Computers in Cardiology. pp. 485–488. IEEE (2002)

[170] Tsymbal, A.: The problem of concept drift: Definitions and related work. Computer Science Department, Trinity College Dublin 106(2), 58 (2004)

[171] Tuncer, T., Dogan, S., Pławiak, P., Acharya, U.R.: Automated arrhythmia detection using novel hexadecimal local pattern and multilevel wavelet transform with ECG signals. Knowledge-Based Systems 186, 104923 (2019)

[172] van Vaerenbergh, S., Santamaría, I., Lázaro-Gredilla, M.: Estimation of the forgetting factor in kernel recursive least squares. In: Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing. pp. 1–6. IEEE (2012)

[173] Vallis, O., Hochenbaum, J., Kejariwal, A.: A novel technique for long-term anomaly detection in the cloud. In: 6th USENIX Workshop on Hot Topics in Cloud Computing, Philadelphia, PA (2014)

[174] Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E.: Deep learning for computer vision: A brief review. Computational Intelligence and Neuroscience 2018 (2018)

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

[176] Welford, B.P.: Note on a method for calculating corrected sums of squares and products. Technometrics 4(3), 419–420 (1962)

[177] Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23(1), 69–101 (1996)

[178] Wilcoxon, F.: Individual comparisons by ranking methods. In: Breakthroughs in Statistics, pp. 196–202. Springer (1992)

[179] Woodbury, M.A.: Inverting modified matrices. Memorandum Report 42, Statistical Research Group, Princeton University, Princeton, NJ, (1950)

[180] Xu, H., et al.: Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications. In: Proceedings of The Web Conference (WWW). pp. 187–196 (2018)

[181] Yao, Y., Sharma, A., Golubchik, L., Govindan, R.: Online anomaly detection for sensor systems: A simple and efficient approach. Performance Evaluation 67(11), 1059–1075 (2010)

[182] Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions. In: Proceedings of the 4th International Conference on Learning Representations (ICLR) (2016)
[183] Zhai, S., Cheng, Y., Lu, W., Zhang, Z.: Deep structured energy based models for anomaly detection. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML). p. 1100–1109. ICML'16, JMLR.org (2016)
[184] Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H., Chawla, N.V.: A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1409–1416 (2019)
[185] Zhang, Z.M., Chen, S., Liang, Y.Z.: Baseline correction using adaptive iteratively reweighted penalized least squares. Analyst 135, 1138–1146 (2010)
[186] Zhou, B., Liu, S., Hooi, B., Cheng, X., Ye, J.: BeatGAN: Anomalous rhythm detection using adversarially generated time series. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 4433–4439 (2019)
[187] Zhu, L., Laptev, N.: Deep and confident prediction for time series at Uber. In: Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW). pp. 103–110. IEEE (2017)
[188] Zong, B., et al.: Deep autoencoding Gaussian mixture model for unsupervised anomaly detection. In: Proceedings of the International Conference on Learning Representations (ICLR) (2018)

Appendices

Appendix A

Extended Results for Chapter 7

Table A.1: Summary for the ECG-25 data (mean $\pm$ σ_{mean} of 10 runs, except for the deterministic NuPic algorithm). The results shown here are for the sum of TP, FN and FP over all 25 time series. For each algorithm and time series, the anomaly threshold was tuned on 10 % of the data, as described in Section 7.4.2.2.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
NuPIC	354.7 ± 5.4	366.3 ± 5.4	2214.5 ± 35.3	0.140 ± 0.003	0.492 ± 0.008	0.217 ± 0.004	1.948e-18
SORAD	566.6 ± 3.8	154.4 ± 3.8	743.2 ± 15.4	0.438 ± 0.006	0.786 ± 0.005	0.561 ± 0.005	1.948e-18
DNN-AE	551.3 ± 3.4	169.7 ± 3.4	672.5 ± 8.4	0.452 ± 0.004	0.765 ± 0.005	0.568 ± 0.004	1.948e-18
LSTM-ED	578.8 ± 3.2	142.2 ± 3.2	652.3 ± 17.6	0.480 ± 0.007	0.803 ± 0.004	0.597 ± 0.005	1.948e-18
TCN-AE (baseline)	640.4 ± 1.9	80.6 ± 1.9	630.9 ± 24.1	0.518 ± 0.008	0.888 ± 0.003	0.650 ± 0.006	2.480e-18
LSTM-AD	569.3 ± 1.9	151.7 ± 1.9	411.8 ± 8.7	0.585 ± 0.005	0.790 ± 0.003	0.671 ± 0.004	9.517e-18
TCN-AE (final)	691.3 ± 0.7	29.7 ± 0.7	352.8 ± 10.2	0.668 ± 0.006	0.959 ± 0.001	0.786 ± 0.004	–

Table A.2: F_1-scores (mean $\pm\, \sigma_{\text{mean}}$) of TCN-AE and the other algorithms (highest values in boldface). Same as Table 7.9, except that we permit the algorithms to determine an anomaly threshold from only 10% of the anomaly labels. In all cases in which we fail to reject the null hypothesis at the 5%-confidence level, we add a gray background to the corresponding field.

	NuPIC		LSTM-ED		DNN-AE		LSTM-AD		TCN-AE
	F1	p	F1	p	F1	p	F1	p	F1
1	0.077±0.012	1.94e-18	0.475±0.012	2.07e-18	0.809±0.016	0.00108	0.801±0.017	0.000857	**0.831±0.014**
2	0.164±0.038	8.63e-15	0.254±0.014	2.37e-10	0.276±0.012	4.85e-10	**0.561±0.014**	1.0	0.467±0.024
3	0.082±0.016	1.95e-18	0.77 ±0.01	4.6e-09	0.729±0.017	9.57e-08	0.345±0.014	1.95e-18	**0.845±0.011**
4	0.106±0.035	6.3e-06	0.112±0.018	1.94e-18	0.144±0.018	1.29e-17	**0.420±0.015**	1.0	0.278±0.031
8	0.294±0.041	1.95e-18	0.861±0.009	8.61e-06	0.538±0.013	7.07e-18	0.789±0.009	1.22e-11	**0.903±0.010**
9	0.108±0.027	2.01e-18	0.471±0.009	6.23e-13	0.544±0.013	0.0103	0.397±0.021	6.39e-11	**0.573±0.011**
10	0.509±0.064	3.79e-15	0.691±0.012	5.3e-06	0.644±0.024	9.3e-13	**0.796±0.011**	0.955	0.785±0.014
11	0.043±0.018	1.17e-18	0.515±0.034	0.0807	0.285±0.028	7.4e-15	**0.640±0.039**	0.967	0.556±0.023
12	0.09 ±0.06	1.8e-18	0.351±0.019	1.27e-15	0.343±0.027	1.25e-14	0.578±0.028	0.381	**0.593±0.022**
13	0.010±0.007	1.33e-18	0.718±0.022	4.99e-08	0.588±0.028	2.25e-15	0.624±0.029	9.26e-14	**0.817±0.022**
14	0.372±0.080	2e-18	0.793±0.015	4.31e-08	0.614±0.011	9.52e-18	0.684±0.019	1.42e-15	**0.869±0.010**
15	0.006±0.003	1.78e-18	0.574±0.018	4.46e-10	0.702±0.023	0.791	**0.781±0.008**	1.0	0.692±0.015
16	0.264±0.019	1.93e-18	0.816±0.010	1.1e-06	0.612±0.008	3.78e-17	0.873±0.010	0.00805	**0.883±0.013**
17	0.001±0.001	1.97e-19	0.097±0.029	4.66e-16	0.10 ±0.03	9.66e-16	0.10 ±0.03	9.66e-16	**0.791±0.029**
18	0.251±0.004	1.95e-18	0.482±0.007	1.95e-18	0.660±0.017	3.06e-18	0.837±0.006	4.07e-14	**0.917±0.005**
20	0.007±0.007	3.74e-18	0.368±0.032	0.000163	0.288±0.032	1.1e-07	0.329±0.036	0.00217	**0.436±0.032**
21	0.004±0.004	1.47e-18	0.361±0.033	2.1e-11	0.230±0.021	2.88e-12	0.391±0.021	0.000737	**0.558±0.038**
22	0.337±0.067	1.75e-17	0.500±0.024	1.1e-07	0.538±0.026	1.81e-05	0.621±0.022	0.212	**0.629±0.018**
23	0.281±0.032	1.94e-18	0.504±0.022	2.74e-17	0.769±0.017	0.000898	0.740±0.016	2.52e-05	**0.830±0.012**
26	0.135±0.023	1.95e-18	0.561±0.024	7.71e-11	0.460±0.019	5.74e-18	0.381±0.013	5.01e-14	**0.665±0.015**
28	0.426±0.038	1.94e-18	0.704±0.019	2.45e-06	0.723±0.020	0.00418	0.775±0.012	0.00466	**0.801±0.010**
33	0.0	1.34e-05	**0.092±0.014**	1.0	0.062±0.007	0.999	0.002±0.001	1.34e-05	0.033±0.007
39	0.150±0.019	1.95e-18	0.719±0.010	1.77e-17	0.781±0.019	2.93e-06	0.823±0.011	6.18e-08	**0.898±0.007**
42	0.468±0.021	1.95e-18	0.738±0.014	3.59e-13	0.736±0.023	6.34e-07	0.714±0.012	6.9e-16	**0.844±0.008**
48	0.008±0.004	1.8e-18	0.544±0.024	1.39e-08	0.498±0.032	2.97e-05	**0.703±0.024**	0.925	0.667±0.022
Σ	0.217±0.012	1.95e-18	0.597±0.005	1.95e-18	0.568±0.004	1.95e-18	0.671±0.004	9.52e-18	**0.786±0.004**
mean	0.168±0.009	1.95e-18	0.523±0.004	1.95e-18	0.507±0.004	1.95e-18	0.588±0.003	3.78e-18	**0.686±0.005**
median	0.120±0.015	1.95e-18	0.544±0.006	1.95e-18	0.531±0.008	1.95e-18	0.650±0.007	1.22e-17	**0.747±0.007**

Table A.3: Impact of the individual TCN-AE components for the ECG-25 Data (mean $\pm\, \sigma_{\text{mean}}$ of 10 runs). The results shown here are for the sum of TP, FN and FP over all 25 time series. For each algorithm variant and time series the anomaly threshold was tuned on 10 different segments containing only 10 % of the data.

Algorithm	TP	FN	FP	Prec	Rec	F1	p
noAnomScoreCorr	588.9 ± 2.6	132.1 ± 2.6	527.7 ± 14.0	0.536 ± 0.007	0.817 ± 0.004	0.645 ± 0.006	2.008078e-18
baseline	640.4 ± 1.9	80.6 ± 1.9	630.9 ± 24.1	0.518 ± 0.008	0.888 ± 0.003	0.650 ± 0.006	2.480521e-18
noSkip	655.6 ± 1.8	65.4 ± 1.8	537.3 ± 19.2	0.563 ± 0.008	0.909 ± 0.002	0.691 ± 0.006	8.453158e-18
noLatent	651.9 ± 1.8	69.1 ± 1.8	522.8 ± 19.5	0.570 ± 0.009	0.904 ± 0.002	0.695 ± 0.007	5.052100e-17
noRecon	678.0 ± 1.3	43.0 ± 1.3	414.2 ± 12.7	0.628 ± 0.007	0.940 ± 0.002	0.751 ± 0.005	1.226155e-10
noInvDil	680.2 ± 1.2	40.8 ± 1.2	417.1 ± 13.7	0.630 ± 0.008	0.943 ± 0.002	0.752 ± 0.005	8.187313e-13
noMapReduc	687.1 ± 0.8	33.9 ± 0.8	381.2 ± 11.1	0.650 ± 0.007	0.953 ± 0.001	0.771 ± 0.005	2.041980e-04
final	691.3 ± 0.7	29.7 ± 0.7	352.8 ± 10.2	0.668 ± 0.006	0.959 ± 0.001	0.786 ± 0.004	0.000000e+00

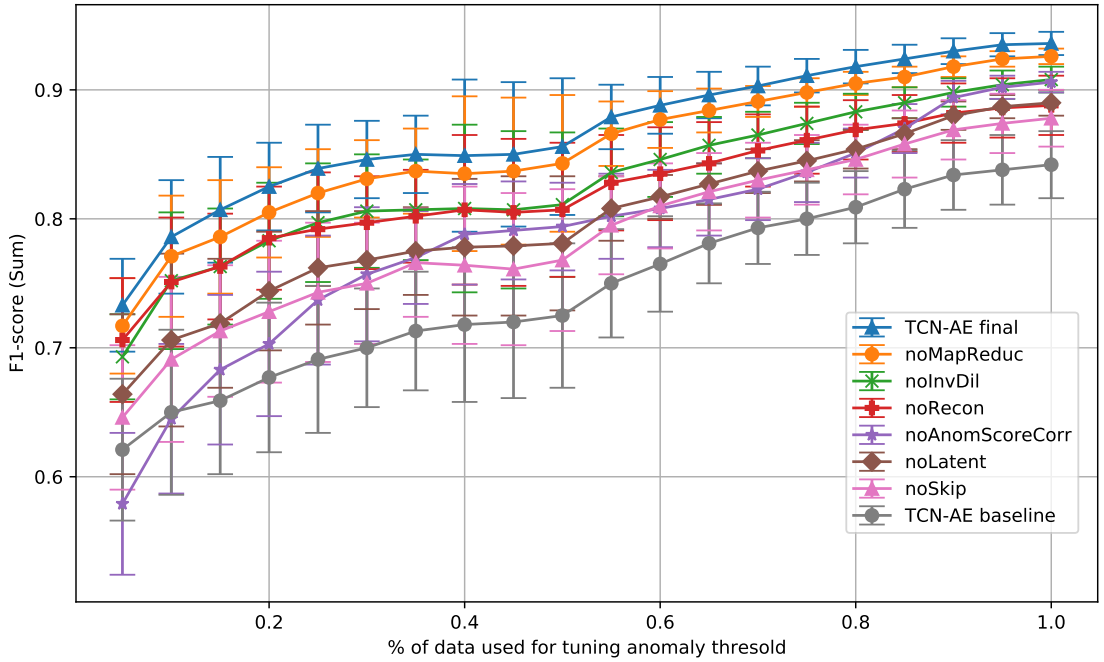

Figure A.1: Results when tuning various algorithmic variants of TCN-AE on different subsets of the ECG-25 data (mean and standard deviation of 10 runs).

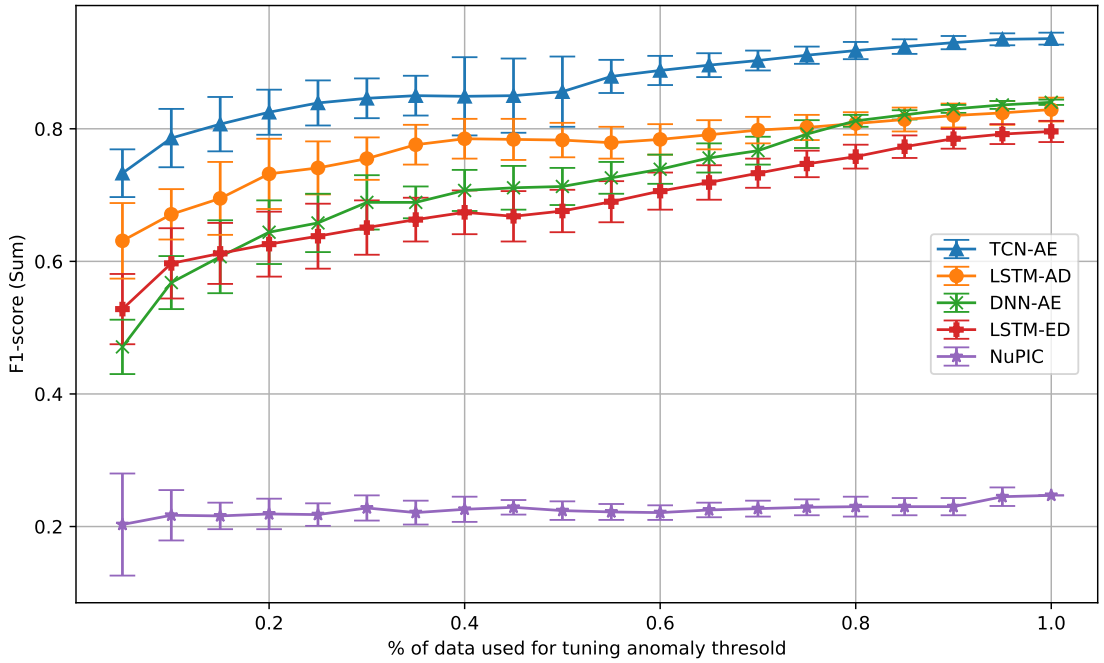

Figure A.2: Results when tuning various algorithms on different subsets of the ECG-25 data (mean and standard deviation of 10 runs).

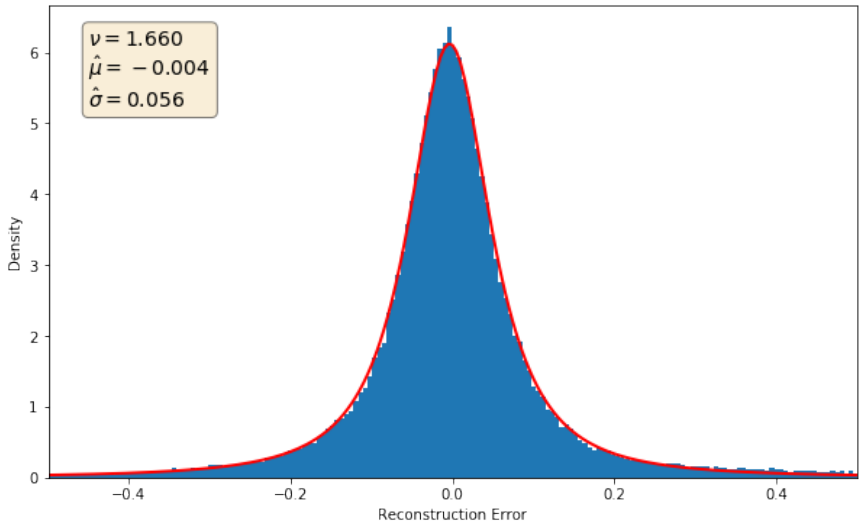

Figure A.3: Histogram of the reconstruction error (in blue) for one dimension of the error matrix $\mathbf{E}'$. We found that the reconstruction errors are bell-shaped (elliptic in higher dimensions). Although the reconstruction errors are not Gaussian, they closely follow a t-distribution with a mean $\hat{\mu}$, a standard deviation $\hat{\sigma}$, and ν degrees of freedom, as indicated by the estimated distribution (red line).

Table A.4: F_1-scores (mean $\pm\, \sigma_{\text{mean}}$) of the individual TCN-AE variants on all 25 time series of the ECG-25 benchmark. The p-values are computed with the one-sided Wilcoxon signed-rank test, in which we compare the final TCN-AE algorithm with the other variants. We compare the F_1-scores of ten runs, which are obtained for an EAC. The Wilcoxon test has the null hypothesis that the median F_1-score of TCN-AE (final) is smaller than the compared algorithm against the alternative that the median F_1-score is larger. In all cases in which we fail to reject the null hypothesis at a confidence level of 5%, we highlight the corresponding field.

Patient No	baseline F1	p	noSkip F1	p	noLatent F1	p	noInvDil F1	p	noRecon F1	p	noMapReduc F1	p	noAnomScoreCorr F1	p	final F1
1	0.768±0.079	0.069	0.844±0.039	0.018	0.940±0.001	0.995	0.913±0.005	0.051	0.846±0.014	0.002	0.937±0.002	0.978	**0.941**	0.995	0.919±0.006
2	0.617±0.043	0.043	0.667	0.042	0.667±0.078	0.168	0.600±0.067	0.035	0.662±0.005	0.046	0.600±0.067	0.035	**0.800±0.022**	0.841	0.733±0.083
3	0.877±0.008	0.003	0.883±0.009	0.002	0.831±0.014	0.002	0.862±0.006	0.002	0.882±0.013	0.002	0.909±0.011	0.041	0.846±0.006	0.002	**0.933±0.008**
4	0.300±0.082	0.002	0.750±0.083	0.013	0.5	0.001	0.5	0.001	0.55 ±0.05	0.001	0.900±0.067	0.079	**1.0**	–	**1.0**
8	0.811±0.016	0.003	0.837±0.015	0.002	0.811±0.012	0.003	0.958±0.004	0.003	0.970±0.004	0.006	0.977±0.002	0.111	0.839±0.015	0.003	**0.981±0.003**
9	0.613±0.045	0.077	**0.675±0.009**	0.556	0.643±0.007	0.007	0.643±0.011	0.025	0.602±0.011	0.004	0.618±0.012	0.007	0.673±0.007	0.556	0.674±0.011
10	0.948±0.016	0.014	0.962±0.007	0.008	0.980±0.005	0.09	0.982±0.003	0.079	0.979±0.004	0.029	**0.987**	–	0.975±0.004	0.023	**0.987**
11	0.9 ±0.1	0.159	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
12	0.750±0.083	0.013	0.600±0.067	0.002	0.95 ±0.05	0.159	0.650±0.077	0.004	0.600±0.067	0.002	0.750±0.112	0.029	**1.0**	–	**1.0**
13	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
14	0.903±0.009	0.009	0.938±0.005	0.088	0.938±0.001	0.039	0.938±0.006	0.124	0.922±0.003	0.003	0.939±0.005	0.2	0.924±0.005	0.005	**0.945±0.004**
15	0.894±0.010	0.079	**0.909**	–	0.894±0.010	0.079	**0.909**	–	0.841±0.008	0.001	0.894±0.010	0.079	**0.909**	–	**0.909**
16	0.905±0.009	0.002	0.932±0.002	0.002	0.956±0.005	0.764	0.949±0.002	0.029	0.962±0.001	0.994	0.950±0.001	0.051	**0.982±0.002**	0.998	0.953±0.001
17	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
18	0.777±0.031	0.003	0.809±0.011	0.003	0.865±0.017	0.002	0.942±0.004	0.002	0.946±0.003	0.002	0.943±0.005	0.006	0.899±0.003	0.003	**0.962±0.003**
20	0.800±0.133	0.079	**1.0**	–	0.967±0.033	0.159	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
21	0.9 ±0.1	0.159	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
22	0.867±0.054	0.023	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**	–	**1.0**
23	0.857±0.014	0.002	0.931±0.008	0.017	0.938±0.006	0.032	**0.952**	–	**0.952**	–	**0.952**	–	0.946±0.007	0.159	**0.952**
26	0.638±0.011	0.002	0.671±0.012	0.002	0.646±0.015	0.002	0.694±0.006	0.002	**0.817±0.013**	0.764	0.768±0.015	0.061	0.635±0.016	0.003	0.806±0.011
28	0.874±0.008	0.008	0.912±0.004	–	0.874±0.006	0.005	0.905±0.004	0.179	**0.926±0.005**	0.987	0.906±0.006	0.207	0.924±0.005	0.977	0.912±0.004
33	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**	–	**0.0**
39	0.859±0.012	0.003	0.897±0.010	0.002	0.945±0.003	0.029	0.952±0.003	0.5	0.950±0.003	0.159	**0.955±0.002**	0.921	0.95 ±0.01	0.405	0.952±0.003
42	0.825±0.009	0.002	0.853±0.006	0.002	0.831±0.009	0.002	0.882±0.002	0.002	0.868±0.005	0.002	0.885±0.003	0.004	0.849±0.011	0.003	**0.909±0.006**
48	0.950±0.033	0.958	**1.0**	0.987	0.889±0.039	0.841	**1.0**	0.987	**1.0**	0.987	0.975±0.025	0.949	0.875±0.042	–	0.875±0.042
Σ	0.826±0.007	0.003	0.861±0.008	0.003	0.871±0.007	0.003	0.904±0.003	0.003	0.907±0.004	0.003	0.913±0.002	0.008	0.890±0.005	0.003	**0.926±0.003**
mean	0.785±0.009	0.003	0.843±0.009	0.003	0.843±0.007	0.003	0.849±0.005	0.003	0.851±0.006	0.003	0.874±0.008	0.07	0.879±0.004	0.003	**0.896±0.006**
median	0.868±0.006	0.003	0.911±0.007	0.003	0.919±0.007	0.003	0.939±0.004	0.003	0.939±0.003	0.003	0.950±0.002	0.069	0.934±0.005	0.003	**0.953±0.002**

Universiteit Leiden
The Netherlands

Appendix B

Derivations

B.1 Batch Incremental Weighted Least Squares Estimator for multivariate Regression Tasks

In practice, the well-known recursive least squares (RLS) filter is often used to learn a linear function in a fully online setting. The RLS filter uses exponentially decaying weights in order to be able to adapt to new concepts. In this section, we derive a similar, however, slightly more complex, incremental version of the weighted least squares estimator which can be used for the SORAD algorithm introduced in Chapter 4. Specifically, we will derive a multivariate variant for batches (having a batch size $\mu \geq 1$) with exponentially decaying weights. This is beneficial in setups which do not have to be fully-online (batch processing usually allows to reduce the computation time if parallelization is supported) or in situations where the amount of data is too large to be processed in a single batch. For example, we use this approach to speed up SORAD for the experiments in Chapter 7.

We start with the original closed-form formulation of the weighted least squares estimator:

$$\boldsymbol{\theta} = \left(\mathbf{X}^\intercal \mathbf{W} \mathbf{X} + \beta \mathbf{I}\right)^{-1} \mathbf{X}^\intercal \mathbf{W} \mathbf{y}. \tag{B.1}$$

where $\mathbf{X}$ is a matrix containing n inputs of length k as row-vectors, $\mathbf{W}$ is a diagonal weight matrix, carrying a weight for each of the n observations, $\mathbf{y}$ is a n-dimensional target vector with one value for each input vector (as we will see later, we can easily extend our explications to multi-dimensional outputs, where we would instead use a matrix $\mathbf{Y}$). The term $\beta\mathbf{I}$ (regularization factor and identity matrix) is the so-called regularizer, which is used to prevent overfitting.

Since we have n observations we can also slightly modify our above equation, to later indicate the current iteration:

$$\boldsymbol{\theta}_n = \Big(\overbrace{\underbrace{\mathbf{X}_n^\intercal \mathbf{W}_n}_{=\mathbf{G}_n} \mathbf{X}_n + \beta \mathbf{I}}^{=\mathbf{A}_n}\Big)^{-1} \overbrace{\underbrace{\mathbf{X}_n^\intercal \mathbf{W}_n}_{=\mathbf{G}_n} \mathbf{y}_n}^{=\mathbf{b}_n} = \Big(\overbrace{\mathbf{G}_n \mathbf{X}_n + \beta \mathbf{I}}^{=\mathbf{A}_n}\Big)^{-1} \overbrace{\mathbf{G}_n \mathbf{y}_n}^{=\mathbf{b}_n} = \mathbf{A}_n^{-1} \mathbf{b}_n, \tag{B.2}$$

$$\boldsymbol{\theta}_n \in \mathbb{R}^k,\ \mathbf{X}_n \in \mathbb{R}^{n \times k},\ \mathbf{W}_n \in \mathbb{R}^{n \times n},\ \mathbf{I} \in \mathbb{R}^{k \times k},\ \boldsymbol{\beta} \in \mathbb{R},\ \mathbf{y}_n \in \mathbb{R}^n$$

$$\mathbf{G}_n \in \mathbb{R}^{k\times n},\ \mathbf{A}_n \in \mathbb{R}^{k\times k},\ \mathbf{b}_n \in \mathbb{R}^k.$$

If now a new batch of μ observation pairs $\mathbf{X}_\mu \in \mathbb{R}^{\mu\times k},\ \mathbf{y}_\mu \in \mathbb{R}^\mu$ arrives, some of the above matrices and vectors change as follows (the others remain unchanged):

$$\mathbf{X}_{n+\mu} = \begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix},\ \mathbf{W}_{n+\mu} = \begin{bmatrix}\lambda\mathbf{W}_n & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & \mathbf{W}_\mu\end{bmatrix},\ \mathbf{y}_{n+\mu} = \begin{bmatrix}\mathbf{y}_n \\ \mathbf{y}_\mu\end{bmatrix}, \tag{B.3}$$

where $\mathbf{X}_{n+\mu} \in \mathbb{R}^{(n+\mu)\times k},\ \mathbf{W}_{n+\mu} \in \mathbb{R}^{(n+\mu)\times(n+\mu)},\ \lambda \in \mathbb{R},\ \mathbf{W}_\mu \in \mathbb{R}^{\mu\times\mu},\ \mathbf{y}_{n+\mu} \in \mathbb{R}^{n+\mu}$. λ is a constant with which we can retroactively modify the old weights $\mathbf{W}_n$. Now let us insert the definitions in Eq. (B.3) into Eq. (B.2):

$$\begin{aligned}\boldsymbol{\theta}_{n+\mu} &= \left(\overbrace{\underbrace{\begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix}^\mathsf{T}\begin{bmatrix}\lambda\mathbf{W}_n & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & \mathbf{W}_\mu\end{bmatrix}}_{=\mathbf{G}_{n+\mu}}\begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix} + \beta\mathbf{I}}^{=\mathbf{A}_{n+\mu}}\right)^{-1}\overbrace{\underbrace{\begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix}^\mathsf{T}\begin{bmatrix}\lambda\mathbf{W}_n & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & \mathbf{W}_\mu\end{bmatrix}}_{=\mathbf{G}_{n+\mu}}\begin{bmatrix}\mathbf{y}_n \\ \mathbf{y}_\mu\end{bmatrix}}^{=\mathbf{b}_{n+\mu}} \\ &= \left(\overbrace{\mathbf{G}_{n+\mu}\begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix} + \beta\mathbf{I}}^{=\mathbf{A}_{n+\mu}}\right)^{-1}\overbrace{\mathbf{G}_{n+\mu}\begin{bmatrix}\mathbf{y}_n \\ \mathbf{y}_\mu\end{bmatrix}}^{=\mathbf{b}_{n+\mu}} = \mathbf{A}_{n+\mu}^{-1}\mathbf{b}_{n+\mu},\end{aligned} \tag{B.4}$$

where we identify:

$$\mathbf{G}_{n+\mu} = \begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix}^\mathsf{T}\begin{bmatrix}\lambda\mathbf{W}_n & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & \mathbf{W}_\mu\end{bmatrix} \in \mathbb{R}^{k\times(n+\mu)}, \tag{B.5}$$

$$\mathbf{A}_{n+\mu} = \mathbf{G}_{n+\mu}\begin{bmatrix}\mathbf{X}_n \\ \mathbf{X}_\mu\end{bmatrix} + \beta\mathbf{I} \in \mathbb{R}^{k\times k}, \tag{B.6}$$

$$\mathbf{b}_{n+\mu} = \mathbf{G}_{n+\mu}\begin{bmatrix}\mathbf{y}_n \\ \mathbf{y}_\mu\end{bmatrix} \in \mathbb{R}^k. \tag{B.7}$$

Now let us expand equation (B.5):

$$\begin{aligned}\mathbf{G}_{n+\mu} &= \begin{bmatrix}\overbrace{\mathbf{X}_n}^{n\times k} \\ \underbrace{\mathbf{X}_\mu}_{\mu\times k}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\overbrace{\lambda\mathbf{W}_n}^{n\times n} & \overbrace{\mathbf{0}}^{n\times\mu} \\ \underbrace{\mathbf{0}^\mathsf{T}}_{\mu\times n} & \underbrace{\mathbf{W}_\mu}_{\mu\times\mu}\end{bmatrix} = \begin{bmatrix}\overbrace{\mathbf{X}_n^\mathsf{T}}^{k\times n} & \overbrace{\mathbf{X}_\mu^\mathsf{T}}^{k\times\mu}\end{bmatrix}\begin{bmatrix}\overbrace{\lambda\mathbf{W}_n}^{n\times n} & \overbrace{\mathbf{0}}^{n\times\mu} \\ \underbrace{\mathbf{0}^\mathsf{T}}_{\mu\times n} & \underbrace{\mathbf{W}_\mu}_{\mu\times\mu}\end{bmatrix} \\ &= \begin{bmatrix}\lambda\mathbf{X}_n^\mathsf{T}\mathbf{W}_n + \mathbf{X}_\mu^\mathsf{T}\mathbf{0}^\mathsf{T} & \mathbf{X}_n^\mathsf{T}\mathbf{0} + \mathbf{X}_\mu^\mathsf{T}\mathbf{W}_\mu\end{bmatrix} = \begin{bmatrix}\lambda\mathbf{X}_n^\mathsf{T}\mathbf{W}_n & \mathbf{X}_\mu^\mathsf{T}\mathbf{W}_\mu\end{bmatrix} \\ &= \begin{bmatrix}\underbrace{\lambda\mathbf{G}_n}_{k\times n} & \underbrace{\mathbf{X}_\mu^\mathsf{T}\mathbf{W}_\mu}_{k\times\mu}\end{bmatrix} \in \mathbb{R}^{k\times(n+\mu)}.\end{aligned}$$

In the next step, let us evaluate $\mathbf{A}_{n+1}$ from Eq. (B.6):

$$\mathbf{A}_{n+\mu} = \mathbf{G}_{n+\mu} \begin{bmatrix} \mathbf{X}_n \\ \mathbf{X}_\mu \end{bmatrix} + \beta\mathbf{I} = \begin{bmatrix} \underbrace{\lambda\mathbf{G}_n}_{k\times n} & \underbrace{\mathbf{X}_\mu^\intercal \mathbf{W}_\mu}_{k\times\mu} \end{bmatrix} \begin{bmatrix} \overbrace{\mathbf{X}_n}^{n\times k} \\ \underbrace{\mathbf{X}_\mu}_{\mu\times k} \end{bmatrix} + \beta\mathbf{I} \tag{B.8}$$

$$= \lambda\mathbf{G}_n\mathbf{X}_n + \mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{X}_\mu + \beta\mathbf{I} = \lambda\underbrace{\mathbf{G}_n\mathbf{X}_n + \beta\mathbf{I}}_{=\mathbf{A}_n} + \mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{X}_\mu \tag{B.9}$$

$$= \lambda \underbrace{\mathbf{A}_n}_{k\times k} + \underbrace{\mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{X}_\mu}_{k\times k}, \quad \text{with } \mathbf{A}_0 = \beta\mathbf{I},\ \lambda_0 = 1. \tag{B.10}$$

Since we have to compute the inverse of $\mathbf{A}_{n+\mu}$, it might be helpful to find an incremental formulation, since the inverse is costly to compute. In this case, the Woodbury matrix identity [179] helps:

$$(\mathbf{A} + \mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{VA}^{-1} + \mathbf{U})^{-1}\mathbf{VA}^{-1}. \tag{B.11}$$

This gives us:

$$\begin{aligned} \mathbf{A}_{n+\mu}^{-1} &= (\lambda\mathbf{A}_n)^{-1} - \overbrace{(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big(\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big)^{-1}}^{=\boldsymbol{\Delta}_\mu}\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1} \\ &= (\lambda\mathbf{A}_n)^{-1} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}, \end{aligned} \tag{B.12}$$

with:

$$\boldsymbol{\Delta}_\mu = (\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big(\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big)^{-1} \in \mathbb{R}^{k\times\mu} \tag{B.13}$$

$$= \mathbf{A}_n^{-1}\mathbf{X}_\mu^\intercal\big(\lambda\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu\mathbf{A}_n^{-1}\mathbf{X}_\mu^\intercal\big)^{-1} \tag{B.14}$$

Then, we expand Eq. (B.7):

$$\mathbf{b}_{n+\mu} = \mathbf{G}_{n+\mu} \begin{bmatrix} \mathbf{y}_n \\ \mathbf{y}_\mu \end{bmatrix} = \begin{bmatrix} \underbrace{\lambda\mathbf{G}_n}_{k\times n} & \underbrace{\mathbf{W}_\mu\mathbf{X}_\mu}_{k\times 1} \end{bmatrix} \begin{bmatrix} \overbrace{\mathbf{y}_n}^{n\times 1} \\ \underbrace{\mathbf{y}_\mu}_{\mu\times 1} \end{bmatrix} \tag{B.15}$$

$$= \lambda\mathbf{G}_n\mathbf{y}_n + \mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu = \lambda\underbrace{\mathbf{b}_n}_{k\times 1} + \underbrace{\mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu}_{k\times 1} \tag{B.16}$$

Now let us insert the results of (B.12) and (B.16) into Eq. (B.4) and then simplify the expression:

$$\boldsymbol{\theta}_{n+\mu} = \mathbf{A}_{n+1}^{-1}\mathbf{b}_{n+1} \tag{B.17}$$
$$= \Big[(\lambda\mathbf{A}_n)^{-1} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\Big]\Big[\lambda\mathbf{b}_n + \mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu\Big] \tag{B.18}$$
$$= \underbrace{\mathbf{A}_n^{-1}\mathbf{b}_n}_{\boldsymbol{\theta}_n} + (\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu\underbrace{\mathbf{A}_n^{-1}\mathbf{b}_n}_{\boldsymbol{\theta}_n} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu \tag{B.19}$$
$$= \boldsymbol{\theta}_n + \Big[(\lambda\mathbf{A}_n)^{-1} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\Big]\mathbf{X}_\mu^\intercal\mathbf{W}_\mu\mathbf{y}_\mu - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu\boldsymbol{\theta}_n \tag{B.20}$$

Although we did a few rearrangements, it seems like Eq. (B.20) cannot be simplified further. However, we can find a more compact solution if we look closer at Eq. (B.13):

$$\boldsymbol{\Delta}_\mu = (\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big(\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big)^{-1} \tag{B.21}$$
$$\boldsymbol{\Delta}_\mu\big(\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal\big) = (\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal \tag{B.22}$$
$$\boldsymbol{\Delta}_\mu\mathbf{W}_\mu^{-1} + \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal = (\lambda\mathbf{A}_n)^{-1}\mathbf{X}_\mu^\intercal \tag{B.23}$$
$$\boldsymbol{\Delta}_\mu\mathbf{W}_\mu^{-1} = \big[(\lambda\mathbf{A}_n)^{-1} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1}\big]\mathbf{X}_\mu^\intercal \tag{B.24}$$

Interestingly, we can find the RHS of Eq. (B.24) also in Eq. (B.20). If we use above relation, we can therefore simplify (B.20) significantly:

$$\boldsymbol{\theta}_{n+\mu} = \boldsymbol{\theta}_n + \boldsymbol{\Delta}_\mu\mathbf{W}_\mu^{-1}\mathbf{W}_\mu\mathbf{y}_\mu - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu\boldsymbol{\theta}_n \tag{B.25}$$
$$= \boldsymbol{\theta}_n + \boldsymbol{\Delta}_\mu\mathbf{y}_\mu - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu\boldsymbol{\theta}_n \tag{B.26}$$
$$= \boldsymbol{\theta}_n + \boldsymbol{\Delta}_\mu\Big(\mathbf{y}_\mu - \mathbf{X}_\mu\boldsymbol{\theta}_n\Big) \tag{B.27}$$
$$= \boldsymbol{\theta}_n + \boldsymbol{\Delta}_\mu\mathbf{e}, \tag{B.28}$$

where $\mathbf{e}$ is the prediction error:

$$\mathbf{e}_\mu = \mathbf{y}_\mu - \mathbf{X}_\mu\boldsymbol{\theta}_n. \tag{B.29}$$

We can summarize our findings for the univariate case with arbitrary weight matrix W_μ as follows:

$$\mathbf{e}_\mu = \mathbf{y}_\mu - \mathbf{X}_\mu\boldsymbol{\theta}_n \tag{B.30}$$
$$\boldsymbol{\Delta}_\mu = \mathbf{A}_n^{-1}\mathbf{X}_\mu^\intercal\big(\lambda\mathbf{W}_\mu^{-1} + \mathbf{X}_\mu\mathbf{A}_n^{-1}\mathbf{X}_\mu^\intercal\big)^{-1} \tag{B.31}$$
$$\boldsymbol{\theta}_{n+\mu} = \boldsymbol{\theta}_n + \boldsymbol{\Delta}_\mu\mathbf{e}_\mu \tag{B.32}$$
$$\mathbf{A}_{n+\mu}^{-1} = (\lambda\mathbf{A}_n)^{-1} - \boldsymbol{\Delta}_\mu\mathbf{X}_\mu(\lambda\mathbf{A}_n)^{-1} \tag{B.33}$$

where

$$\lambda \in \mathbb{R},\ \mathbf{e}_\mu, \mathbf{y}_\mu, \boldsymbol{\theta}_n, \boldsymbol{\theta}_{n+\mu} \in \mathbb{R}^\mu,\ \mathbf{X}_\mu \in \mathbb{R}^{\mu \times k},\ \boldsymbol{\Delta}_\mu \in \mathbb{R}^{k \times \mu},\ \mathbf{A}_n, \mathbf{A}_{n+\mu} \in \mathbb{R}^{k \times k},\ \mathbf{W}_\mu \in \mathbb{R}^{n \times n},$$
$$\mathbf{A}_0 = \beta \mathbf{I},\ \lambda_0 = 1,\ \boldsymbol{\theta}_0 = \mathbf{0}.$$

Multivariate Batch Recursive Least Squares Algorithm Extending the above equations to the multivariate case with m dimensions is straightforward. We simply have to replace some vectors with matrices. Furthermore, for a simple setup with exponentially decaying weights we set $\lambda \in [0, 1]$ and $W_\mu = \mathbf{I}$. Hence, the final multivariate batch RLS algorithm can be described as follows:

$$\mathbf{E}_\mu = \mathbf{Y}_\mu - \mathbf{X}_\mu \boldsymbol{\Theta}_n \tag{B.34}$$
$$\boldsymbol{\Delta}_\mu = \mathbf{A}_n^{-1} \mathbf{X}_\mu^\mathsf{T} \left(\lambda \mathbf{I} + \mathbf{X}_\mu \mathbf{A}_n^{-1} \mathbf{X}_\mu^\mathsf{T}\right)^{-1} \tag{B.35}$$
$$\boldsymbol{\Theta}_{n+\mu} = \boldsymbol{\Theta}_n + \boldsymbol{\Delta}_\mu \mathbf{E}_\mu \tag{B.36}$$
$$\mathbf{A}_{n+\mu}^{-1} = \frac{1}{\lambda} \mathbf{A}_n^{-1} - \frac{1}{\lambda} \boldsymbol{\Delta}_\mu \mathbf{X}_\mu \mathbf{A}_n^{-1} \tag{B.37}$$

where

$$\lambda \in [0, 1],\ \mathbf{E}_\mu, \mathbf{Y}_\mu, \boldsymbol{\Theta}_n, \boldsymbol{\Theta}_{n+\mu} \in \mathbb{R}^{\mu \times m},\ \mathbf{X}_\mu \in \mathbb{R}^{\mu \times k},\ \boldsymbol{\Delta}_\mu \in \mathbb{R}^{k \times \mu},\ \mathbf{A}_n^{-1}, \mathbf{A}_{n+\mu}^{-1} \in \mathbb{R}^{k \times k},$$
$$\mathbf{A}_0^{-1} = \frac{\lambda}{\beta} \mathbf{I},\ \boldsymbol{\Theta}_0 = \mathbf{0}.$$

B.2 Online Estimation of the Sample Mean and Covariance

This section derives several formulas to incrementally compute the weighted mean and the weighted covariance matrix for a multivariate data set. This property is especially useful in online settings, where an algorithm constantly has to update its estimates and in situations where one expects the mean and the covariance matrix to drift over time. We present a fully online procedure and a procedure that can process mini-batches. The derived formulas are used in Chapter 5 for the DWT-MLEAD algorithm and in Chapter 7, for the SORAD algorithm, which was introduced earlier in Chapter 4.

B.2.1 The Weighted Mean and Covariance Matrix

In some cases, one might want to compute the weighted arithmetic mean and a weighted sample covariance (matrix) where particular sample points should contribute more to the final mean (covariances) than others. For example, it could be reasonable to give larger weight to more recent data points if the statistics (e.g., mean) of the underlying distribution

change over time (concept drifts). In such cases, we would want the estimator to track the data generating distribution changes and forget about older knowledge. But also other scenarios are possible, in which a weighting of the data points might be useful. In this section, we define the weighted arithmetic mean and weighted sample covariance, explore a few properties related to these weighted statistics, and then design an estimator for the sample mean and covariance exhibiting some forgetting.

In general, the weighted arithmetic mean is defined quite straightforward as:

$$\bar{\mathbf{x}}_n = \frac{\sum_{i=1}^{n} w_i' \mathbf{x}_i}{\sum_{i=1}^{n} w_i'}, \tag{B.38}$$

where $\mathbf{x}_i$ is the i-th data point (vector) and w_i' is the (unnormalized) weight assigned to the corresponding data point. If the weights are normalized to sum 1, then we get:

$$\bar{\mathbf{x}}_n = \sum_{i=1}^{n} w_i \mathbf{x}_i, \tag{B.39}$$

where the normalized weights are defined as:

$$w_i = \frac{w_i'}{W_n} = \frac{w_i'}{\sum_{i=1}^{n} w_i'}, \tag{B.40}$$

with the normalization factor

$$W_n = \sum_{i=1}^{n} w_i'.$$

The special case with $w_i = \frac{1}{n}$ results in the formulation of the conventional mean. According to [137], the biased weighted covariance matrix can be written similarly as:

$$\bar{\mathbf{\Sigma}} = \frac{\bar{\mathbf{M}}^{(n)}}{W_n} = \frac{\sum_{i=1}^{n} w_i'(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T}}{\sum_{i=1}^{n} w_i'}, \tag{B.41}$$

where $\bar{\mathbf{M}}^{(n)}$ is the so called weighted scatter matrix, with

$$\bar{\mathbf{M}}^{(n)} = \sum_{i=1}^{n} w_i'(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T}.$$

If the weights are frequency weights (each weight represents the count of a data point), then an unbiased estimator is found to be [137]:

$$\bar{\mathbf{\Sigma}} = \frac{\bar{\mathbf{M}}^{(n)}}{W_n - 1}. \tag{B.42}$$

Otherwise, the following unbiased estimator can be used [137]:

$$\bar{\mathbf{\Sigma}} = \frac{\bar{\mathbf{M}}^{(n)}}{W_n - W_n^{(2)}/W_n}, \tag{B.43}$$

where

$$W_n^{(2)} = \sum_{i=1}^{n} (w_i')^2. \tag{B.44}$$

B.2.1.1 Incremental (Batch) Estimation of the Weighted Mean and Covariance

It might become necessary to estimate the mean and covariance matrix of a distribution incrementally in practice. This could be the case if one operates on streaming data or has to process large amounts of data, which cannot be handled in one pass. In the general case, we want to handle batch sizes of $\mu \in \mathbb{N}^+$. For a fully online algorithm, we have the extreme case $\mu = 1$. If a new batch of size μ arrives, we have update our old estimates $\bar{\mathbf{x}}_{n-\mu}$ and $\bar{\mathbf{\Sigma}}_{n-\mu}$. For this purpose we have to process the new examples from $k = n - \mu + 1$ until n. We first note a recursive update rule for $\bar{\mathbf{x}}_n$:

$$\begin{aligned}
\bar{\mathbf{x}}_n &= \frac{W_{n-\mu}\bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_n} && \text{(B.45)}\\
&= \frac{\left(W_n - \sum_{i=k}^{n} w_i'\right)\bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_n} \\
&= \bar{\mathbf{x}}_{n-\mu} + \frac{-\sum_{i=k}^{n} w_i'\bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_n} \\
&= \bar{\mathbf{x}}_{n-\mu} + \frac{\sum_{i=k}^{n} w_i'(\mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu})}{W_n}, \\
&= \bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} \frac{w_i'}{W_n}\mathbf{\Delta}_i && \text{(B.46)}
\end{aligned}$$

with

$$k = n - \mu + 1 \tag{B.47}$$

$$\mathbf{\Delta}_i = \mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu} \tag{B.48}$$

Then, we derive an batch update rule for the estimator for the weighted covariance matrix $\bar{\boldsymbol{\Sigma}}_n$. We start with the weighted scatter matrix:

$$\begin{aligned}
\bar{\mathbf{M}}^{(n)} &= \sum_{i=1}^{n} w_i'(\mathbf{x}_i - \bar{\mathbf{x}}_n)(\mathbf{x}_i - \bar{\mathbf{x}}_n)^\mathsf{T} \\
&= \sum_{i=1}^{n} w_i'\big[\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - \mathbf{x}_i\bar{\mathbf{x}}_n^\mathsf{T} - \bar{\mathbf{x}}_n\mathbf{x}_i^\mathsf{T} + \bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T}\big] \\
&= \sum_{i=1}^{n} w_i'\big[\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - 2\mathbf{x}_i\bar{\mathbf{x}}_n^\mathsf{T} + \bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T}\big] \\
&= \sum_{i=1}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - 2\left(\sum_{i=1}^{n} w_i'\mathbf{x}_i\right)\bar{\mathbf{x}}_n^\mathsf{T} + \sum_{i=1}^{n} w_i'\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T} \\
&= \sum_{i=1}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - 2W_n\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T} + W_n\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T} \\
&= \sum_{i=1}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - W_n\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T}
\end{aligned}$$

The increment of the weighted scatter matrix from index $k = n-\mu+1$ to n can be computed as follows:

$$\begin{aligned}
\boldsymbol{\Delta}\bar{\mathbf{M}}^{(n)} &= \bar{\mathbf{M}}^{(n)} - \bar{\mathbf{M}}^{(n-\mu)} \\
&= \sum_{i=1}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - W_n\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T} - \sum_{i=1}^{n-\mu} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} + W_{n-\mu}\bar{\mathbf{x}}_{n-\mu}\bar{\mathbf{x}}_{n-\mu}^\mathsf{T} \\
&= \sum_{i=k}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - W_n\bar{\mathbf{x}}_n\bar{\mathbf{x}}_n^\mathsf{T} + W_{n-\mu}\bar{\mathbf{x}}_{n-\mu}\bar{\mathbf{x}}_{n-\mu}^\mathsf{T}
\end{aligned} \tag{B.49}$$

With the already known expression from Eq. (B.45) and with a rearranged formulation of Eq. (B.45)

$$\bar{\mathbf{x}}_{n-\mu} = \frac{W_n\bar{\mathbf{x}}_n - \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_{n-\mu}} \tag{B.50}$$

we can start simplifying Eq. (B.49). We insert Eq. (B.45) and Eq. (B.50) into Eq. (B.49):

$$\boldsymbol{\Delta}\bar{\mathbf{M}}^{(n)} = \sum_{i=k}^{n} w_i'\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - W_n\frac{W_{n-\mu}\bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_n}\bar{\mathbf{x}}_n^\mathsf{T} + W_{n-\mu}\bar{\mathbf{x}}_{n-\mu}\left(\frac{W_n\bar{\mathbf{x}}_n - \sum_{i=k}^{n} w_i'\mathbf{x}_i}{W_{n-\mu}}\right)^\mathsf{T}$$

$$
\begin{aligned}
&= \sum_{i=k}^{n} w'_i \mathbf{x}_i \mathbf{x}_i^\mathsf{T} - \left(\sum_{i=k}^{n} w'_i \mathbf{x}_i \right) \bar{\mathbf{x}}_n^\mathsf{T} - W_{n-\mu} \bar{\mathbf{x}}_{n-\mu} \bar{\mathbf{x}}_n^\mathsf{T} + W_n \bar{\mathbf{x}}_{n-\mu} \bar{\mathbf{x}}_n^\mathsf{T} - \bar{\mathbf{x}}_{n-\mu} \left(\sum_{i=k}^{n} w'_i \mathbf{x}_i \right)^\mathsf{T} \\
&= \sum_{i=k}^{n} w'_i \mathbf{x}_i \mathbf{x}_i^\mathsf{T} - \sum_{i=k}^{n} w'_i \mathbf{x}_i \bar{\mathbf{x}}_n^\mathsf{T} + \big(W_n - W_{n-\mu}\big) \bar{\mathbf{x}}_{n-\mu} \bar{\mathbf{x}}_n^\mathsf{T} - \bar{\mathbf{x}}_{n-\mu} \sum_{i=k}^{n} w'_i \mathbf{x}_i^\mathsf{T} \\
&= \sum_{i=k}^{n} w'_i \mathbf{x}_i \mathbf{x}_i^\mathsf{T} - \sum_{i=k}^{n} w'_i \mathbf{x}_i \bar{\mathbf{x}}_n^\mathsf{T} + \left(\sum_{i=k}^{n} w'_i \right) \bar{\mathbf{x}}_{n-\mu} \bar{\mathbf{x}}_n^\mathsf{T} - \bar{\mathbf{x}}_{n-\mu} \sum_{i=k}^{n} w'_i \mathbf{x}_i^\mathsf{T} \\
&= \sum_{i=k}^{n} \left(w'_i \mathbf{x}_i \mathbf{x}_i^\mathsf{T} - w'_i \mathbf{x}_i \bar{\mathbf{x}}_n^\mathsf{T} + w'_i \bar{\mathbf{x}}_{n-\mu} \bar{\mathbf{x}}_n^\mathsf{T} - w'_i \bar{\mathbf{x}}_{n-\mu} \mathbf{x}_i^\mathsf{T} \right) \\
&= \sum_{i=k}^{n} \left(w'_i \mathbf{x}_i \big(\mathbf{x}_i^\mathsf{T} - \bar{\mathbf{x}}_n^\mathsf{T}\big) - w'_i \bar{\mathbf{x}}_{n-\mu} \big(\mathbf{x}_i^\mathsf{T} - \bar{\mathbf{x}}_n^\mathsf{T}\big) \right) \\
&= \sum_{i=k}^{n} w'_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu}\big) \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^\mathsf{T} \\
&= \sum_{i=k}^{n} w'_i \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^\mathsf{T}
\end{aligned} \tag{B.51}
$$

In summary, with equations (B.40), (B.44), (B.48), (B.46), (B.51), and (B.41) we can update our estimates for a new batch (samples from $k = n - \mu + 1$ to n) with:

$$W_n = W_{n-\mu} + \sum_{i=k}^{n} w'_i \tag{B.52}$$

$$W_n^{(2)} = W_{n-\mu}^{(2)} + \sum_{i=k}^{n} (w'_i)^2 \tag{B.53}$$

$$\boldsymbol{\Delta}_i = \mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu} \tag{B.54}$$

$$\bar{\mathbf{x}}_n = \bar{\mathbf{x}}_{n-\mu} + \sum_{i=k}^{n} \frac{w'_i}{W_n} \boldsymbol{\Delta}_i \tag{B.55}$$

$$\bar{\mathbf{M}}^{(n)} = \bar{\mathbf{M}}^{(n-\mu)} + \sum_{i=k}^{n} w'_i \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^\mathsf{T} \tag{B.56}$$

$$\bar{\boldsymbol{\Sigma}}_n = \frac{\bar{\mathbf{M}}^{(n)}}{W_n}. \tag{B.57}$$

Note that Eq. (B.57) is a biased estimate of the covariance matrix. For an unbiased estimate one can use either Eq. (B.42) or Eq. (B.43). If all weights are set to $w'_i = 1$, we obtain the update rules for the unweighted case.

B.2.2 Incremental Estimation with Exponentially Decaying Weights

If we use constant batch sizes μ and also keep the weights w'_i among a batch constant, an useful incremental algorithm can be retrieved for a particular set of weights, where each weight is

$$w'_i = \lambda^{\lfloor \frac{n-i}{\mu} \rfloor} = \lambda^{M-j},$$

where $\lambda \in (0, 1]$ is the decay rate, $M = n/\mu$ ($\mu \mid n$) represents the number of batches for n examples, and $j = \lceil i/\mu \rceil$ is the index of the batch for the i-th example. With such a weighting, each example in the most recent batch will be weighted with $w'_k = \cdots = w'_{n-1} = w'_n = 1$, the penultimate batch will have the weights $w'_{n-2\mu+1} = \cdots = w'_{n-\mu-1} = w'_{n-\mu} = \lambda$ and so forth. Ultimately, the older batches will fade away exponentially, for $\lambda < 1$. The advantage of this approach is that we can usually prevent numerical overflows in W_n and the weighted scatter matrix $\bar{\mathbf{M}}^{(n)}$ and that we can adapt our estimates to new concepts in non-stationary environments.

The normalization factor W_n can be computed with:

$$W_n = \sum_{i=1}^{n} w'_i = \sum_{j=1}^{M} \mu \lambda^{M-j} = \mu \frac{1 - \lambda^M}{1 - \lambda}.$$

Note that the weightings $w'_{n-\mu}, w'_{n-\mu-1}, \ldots, w'_1$ of all previous examples $\mathbf{x}_{n-\mu}, \mathbf{x}_{n-\mu-1}, \ldots, \mathbf{x}_1$, as well as the normalization factor W_n, have to be adjusted if a new example $\mathbf{x}_n$ arrives. In this case, each weight $\{w'_i \mid i \leq n - \mu\}$ has to be multiplied with λ and the weights of the new examples are set to $w'_k = \cdots = w'_n = 1$. How does this change the estimation of the mean and the covariance matrix? Typically, if all weights are changed, one would have to re-compute both statistics from scratch. However, we can show that in this particular case, this is not necessary, as described in the following explications: First let us find an update rule for the normalization factor W_n. We can see that $W_{n-\mu}$ is given by:

$$W_{n-\mu} = \mu \sum_{j=1}^{M-1} \lambda^{M-1-j}. \tag{B.58}$$

Then we re-write W_n:

$$\begin{aligned} W_n &= \mu \sum_{j=1}^{M} \lambda^{M-j} \\ &= \mu \sum_{j=1}^{M-1} \lambda^{M-j} + \mu\lambda^0 = \lambda \cdot \mu \sum_{j=1}^{M-1} \lambda^{M-1-j} + \mu \end{aligned}$$

$$= \lambda \cdot W_{n-\mu} + \mu \tag{B.59}$$

Accordingly, for $W_n^{(2)}$we obtain:

$$W_n^{(2)} = \lambda^2 \cdot W_{n-\mu}^{(2)} + \mu.$$

Then, we find an recursive formulation of the estimated mean $\bar{\mathbf{x}}_n$. According to Eq. (B.38), we find:

$$\bar{\mathbf{x}}_n = \frac{\sum_{j=1}^{M} \lambda^{M-j} \sum_{i=k_j}^{n_j} \mathbf{x}_i}{\mu \sum_{j=1}^{M} \lambda^{M-j}} = \frac{\sum_{j=1}^{M} \lambda^{M-j} \mathbf{\Sigma}_j}{W_n}, \tag{B.60}$$

where

$$n_j = j\mu \ , \ k_j = n_j - \mu + 1 = \mu(j-1) + 1,$$
$$\mathbf{\Sigma}_j = \sum_{i=k_j}^{n_j} \mathbf{x}_i.$$

Then, with

$$\bar{\mathbf{x}}_{n-\mu} = \frac{\sum_{j=1}^{M-1} \lambda^{M-1-j} \mathbf{\Sigma}_j}{W_{n-\mu}}, \quad \text{and} \quad W_{n-\mu} = \frac{W_n - \mu}{\lambda},$$

we can obtain a recursive formulation of Eq. (B.60):

$$\begin{aligned}
\bar{\mathbf{x}}_n &= \frac{\sum_{j=1}^{M} \lambda^{M-j} \mathbf{\Sigma}_j}{W_n} \\
&= \frac{\sum_{j=1}^{M-1} \lambda^{M-j} \mathbf{\Sigma}_j + \lambda^0 \mathbf{\Sigma}_j}{W_n} = \frac{\lambda \sum_{j=1}^{M-1} \lambda^{M-1-j} \mathbf{\Sigma}_j}{W_n} + \frac{\mathbf{\Sigma}_M}{W_n} \\
&= \lambda \frac{W_n - \mu}{\lambda} \frac{1}{W_n} \bar{\mathbf{x}}_{n-\mu} + \frac{\mathbf{\Sigma}_M}{W_n} = \bar{\mathbf{x}}_{n-\mu} - \frac{\mu \bar{\mathbf{x}}_{n-\mu}}{W_n} + \frac{\mathbf{\Sigma}_M}{W_n} \\
&= \bar{\mathbf{x}}_{n-\mu} + \frac{\sum_{i=k}^{n} \mathbf{x}_i - \sum_{i=k}^{n} \bar{\mathbf{x}}_{n-\mu}}{W_n} = \bar{\mathbf{x}}_{n-\mu} + \frac{\sum_{i=k}^{n} \left(\mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu} \right)}{W_n} \\
&= \bar{\mathbf{x}}_{n-\mu} + \frac{\sum_{i=k}^{n} \mathbf{\Delta}_i}{W_n}
\end{aligned} \tag{B.61}$$

Finally, we only need a to find a recursive form for the weighted scatter matrix $\bar{\mathbf{M}}^{(n)}$. By expanding the recursion in (B.56) we get (with $k = n - \mu + 1$):

$$\begin{aligned}
\bar{\mathbf{M}}^{(n)} &= \sum_{j=1}^{M} \lambda^{M-j} \sum_{i=k_j}^{n_j} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}} \ , \quad n_j = j\mu \ , \ k_j = \mu(j-1) + 1 \\
&= \sum_{j=1}^{M-1} \lambda^{M-j} \sum_{i=k_j}^{n_j} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}} + \lambda^0 \sum_{i=k}^{n} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}} \\
&= \lambda \sum_{j=1}^{M-1} \lambda^{M-1-j} \sum_{i=k_j}^{n_j} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}} + \sum_{i=k}^{n} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}} \\
&= \lambda \bar{\mathbf{M}}^{(n-\mu)} + \sum_{i=k}^{n} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}}, \qquad \text{(B.62)}
\end{aligned}$$

since

$$\bar{\mathbf{M}}^{(n-\mu)} = \sum_{j=1}^{M-1} \lambda^{M-1-j} \sum_{i=k_j}^{n_j} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}}.$$

B.2.2.1 Batch and Online Estimation of the Inverse Covariance Matrix

In practice, it is often required to estimate the inverse of the covariance matrix of a sample, for example, when a Mahalanobis distance between points has to be computed. In a fully online setting it can be quite expensive to compute the inverse of the covariance matrix again for each new point arriving, since the complexity of the inverse of a $m \times m$ matrix is about $\mathcal{O}(m^3)$ (slightly less in some highly optimized algorithms). As we will see in the following, it is not necessary to estimate the covariance matrix, if only its inverse is needed and furthermore, the inverse can be adapted incrementally in an efficient manner. The results are shown for the general case with a weighted exponentially decaying estimator.

For a batch-setup ($\mu > 1$), one can use the Woodbury matrix identity [179] to find a recursive definition of the inverse of the scatter matrix $\bar{\mathbf{M}}$ and covariance matrix $\bar{\boldsymbol{\Sigma}}$. The Woodbury matrix identity is given as:

$$(\mathbf{A} + \mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{VA}^{-1} + \mathbf{U})^{-1}\mathbf{VA}^{-1}. \qquad \text{(B.63)}$$

First, we write Eq. (B.62) in vectorized form (to be consistent in the notation later, we move the superscript in $\bar{\mathbf{M}}$ to the index):

$$\bar{\mathbf{M}}_n = \lambda \bar{\mathbf{M}}_{n-\mu} + \sum_{i=k}^{n} \boldsymbol{\Delta}_i \big(\mathbf{x}_i - \bar{\mathbf{x}}_n\big)^{\mathsf{T}}$$

$$= \lambda \bar{\mathbf{M}}_{n-\mu} + \mathbf{D}_n^\mathsf{T} \mathbf{I} \boldsymbol{\mathcal{X}}_n, \tag{B.64}$$

where $k = n - \mu + 1$ and

$$\mathbf{D}_n = \begin{pmatrix} \boldsymbol{\Delta}_k & \boldsymbol{\Delta}_{k+1} & \cdots & \boldsymbol{\Delta}_n \end{pmatrix}^\mathsf{T}$$
$$\boldsymbol{\mathcal{X}}_n = \begin{pmatrix} \mathbf{x}_k - \bar{\mathbf{x}}_n & \mathbf{x}_{k+1} - \bar{\mathbf{x}}_n & \cdots & \mathbf{x}_n - \bar{\mathbf{x}}_n \end{pmatrix}^\mathsf{T}.$$

Then we identify:

$$\mathbf{A} = \lambda \bar{\mathbf{M}}_{n-\mu},\ \mathbf{U} = \mathbf{D}_n^\mathsf{T},\ \mathbf{C} = \mathbf{I},\ \mathbf{V} = \boldsymbol{\mathcal{X}}_n,$$

and find:

$$\begin{aligned} \bar{\mathbf{M}}_n^{-1} &= \frac{1}{\lambda} \bar{\mathbf{M}}_{n-\mu}^{-1} - \frac{1}{\lambda^2} \bar{\mathbf{M}}_{n-\mu}^{-1} \mathbf{D}_n^\mathsf{T} \Big(\mathbf{I}^{-1} + \frac{1}{\lambda} \boldsymbol{\mathcal{X}}_n \bar{\mathbf{M}}_{n-\mu}^{-1} \mathbf{D}_n^\mathsf{T} \Big)^{-1} \boldsymbol{\mathcal{X}}_n \bar{\mathbf{M}}_{n-\mu}^{-1} \\ &= \frac{1}{\lambda} \bar{\mathbf{M}}_{n-\mu}^{-1} - \frac{1}{\lambda} \bar{\mathbf{M}}_{n-\mu}^{-1} \mathbf{D}_n^\mathsf{T} \Big(\lambda \mathbf{I} + \boldsymbol{\mathcal{X}}_n \bar{\mathbf{M}}_{n-\mu}^{-1} \mathbf{D}_n^\mathsf{T} \Big)^{-1} \boldsymbol{\mathcal{X}}_n \bar{\mathbf{M}}_{n-\mu}^{-1}. \end{aligned}$$

Since we also have to compute an inverse here, it only makes sense to use the Woodbury matrix identity if the batch size μ is significantly smaller than the dimension of the examples $\mathbf{x}_i$. Similarly, in order to compute the inverse fully online ($\mu = 1$), we simply apply the Sherman-Morrison formula [150] – a special case of the Woodbury matrix identity [179] – to incrementally update $\bar{\mathbf{M}}_n^{-1}$. The formula is given by

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^\mathsf{T})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^\mathsf{T}\mathbf{A}^{-1}}{1 + \mathbf{v}^\mathsf{T}\mathbf{A}^{-1}\mathbf{u}}. \tag{B.65}$$

If we look at Eq. (B.62), we can identify:

$$\mathbf{A} = \lambda \bar{\mathbf{M}}^{(n-1)},\ \mathbf{u} = \boldsymbol{\Delta}_n,\ \mathbf{v} = \mathbf{x}_n - \bar{\mathbf{x}}_n$$

This then leads to:

$$\bar{\mathbf{M}}_n^{-1} = \frac{1}{\lambda} \bar{\mathbf{M}}_{n-1}^{-1} - \frac{\frac{1}{\lambda} \bar{\mathbf{M}}_{n-1}^{-1} \boldsymbol{\Delta}_n (\mathbf{x}_n - \bar{\mathbf{x}}_n)^\mathsf{T} \bar{\mathbf{M}}_{n-1}^{-1}}{\lambda + (\mathbf{x}_n - \bar{\mathbf{x}}_n)^\mathsf{T} \bar{\mathbf{M}}_{n-1}^{-1} \boldsymbol{\Delta}_n}. \tag{B.66}$$

With Eq. (B.83), we can finally compute the inverse of the covariance matrix with

$$\bar{\boldsymbol{\Sigma}}_n^{-1} = W_n \bar{\mathbf{M}}_n^{-1}. \tag{B.67}$$

In summary, we can write down the following rules for the recursive (iterative) estimation of the mean vector and the covariance matrix, which should be processed in the given order:

$$W_n = \lambda \cdot W_{n-\mu} + \mu \tag{B.68}$$

$$W_n^{(2)} = \lambda^2 \cdot W_{n-\mu}^{(2)} + \mu \tag{B.69}$$
$$\boldsymbol{\Delta}_i = \mathbf{x}_i - \bar{\mathbf{x}}_{n-\mu} \tag{B.70}$$
$$\bar{\mathbf{x}}_n = \bar{\mathbf{x}}_{n-\mu} + \frac{\sum_{i=k}^{n} \boldsymbol{\Delta}_i}{W_n} \tag{B.71}$$
$$\mathbf{D}_n = \begin{pmatrix} \boldsymbol{\Delta}_k & \boldsymbol{\Delta}_{k+1} & \cdots & \boldsymbol{\Delta}_n \end{pmatrix}^\mathsf{T} \tag{B.72}$$
$$\boldsymbol{\mathcal{X}}_n = \begin{pmatrix} \mathbf{x}_k - \bar{\mathbf{x}}_n & \mathbf{x}_{k+1} - \bar{\mathbf{x}}_n & \cdots & \mathbf{x}_n - \bar{\mathbf{x}}_n \end{pmatrix}^\mathsf{T} \tag{B.73}$$
$$\bar{\mathbf{M}}_n = \lambda \bar{\mathbf{M}}_{n-\mu} + \mathbf{D}_n^\mathsf{T} \boldsymbol{\mathcal{X}}_n \tag{B.74}$$
$$\bar{\mathbf{M}}_n^{-1} = \frac{1}{\lambda}\bar{\mathbf{M}}_{n-\mu}^{-1} - \frac{1}{\lambda}\bar{\mathbf{M}}_{n-\mu}^{-1}\mathbf{D}_n^\mathsf{T}\Big(\lambda\mathbf{I} + \boldsymbol{\mathcal{X}}_n\bar{\mathbf{M}}_{n-\mu}^{-1}\mathbf{D}_n^\mathsf{T}\Big)^{-1}\boldsymbol{\mathcal{X}}_n\bar{\mathbf{M}}_{n-\mu}^{-1} \tag{B.75}$$
$$\bar{\boldsymbol{\Sigma}}_n = \frac{\bar{\mathbf{M}}_n}{W_n}, \quad \bar{\boldsymbol{\Sigma}}_n^{-1} = W_n\bar{\mathbf{M}}_n^{-1}. \tag{B.76}$$

where μ is the batch size and $k = n - \mu + 1$ is the first index in the new batch. Again, for an unbiased estimate of $\bar{\boldsymbol{\Sigma}}$ one should either use Eq. (B.42) or Eq. (B.43).

For a fully online estimation (batch size $\mu = 1$, $k = n$), above update rules simplify to:

$$W_n = \lambda W_{n-1} + 1 \tag{B.77}$$
$$W_n^{(2)} = \lambda^2 W_{n-1} + 1 \tag{B.78}$$
$$\boldsymbol{\Delta}_n = \mathbf{x}_n - \bar{\mathbf{x}}_{n-1} \tag{B.79}$$
$$\bar{\mathbf{x}}_n = \bar{\mathbf{x}}_{n-1} + \frac{\boldsymbol{\Delta}_n}{W_n} \tag{B.80}$$
$$\bar{\mathbf{M}}_n = \lambda\bar{\mathbf{M}}_{n-1} + \boldsymbol{\Delta}_n(\mathbf{x}_n - \bar{\mathbf{x}}_n)^\mathsf{T} \tag{B.81}$$
$$\bar{\mathbf{M}}_n^{-1} = \frac{1}{\lambda}\bar{\mathbf{M}}_{n-1}^{-1} - \frac{\frac{1}{\lambda}\bar{\mathbf{M}}_{n-1}^{-1}\boldsymbol{\Delta}_n(\mathbf{x}_n - \bar{\mathbf{x}}_n)^\mathsf{T}\bar{\mathbf{M}}_{n-1}^{-1}}{\lambda + (\mathbf{x}_n - \bar{\mathbf{x}}_n)^\mathsf{T}\bar{\mathbf{M}}_{n-1}^{-1}\boldsymbol{\Delta}_n} \tag{B.82}$$
$$\bar{\boldsymbol{\Sigma}}_n = \frac{1}{W_n}\bar{\mathbf{M}}_n, \quad \bar{\boldsymbol{\Sigma}}_n^{-1} = W_n\bar{\mathbf{M}}_n^{-1}. \tag{B.83}$$

In Sec. B.2.4, we show that the memory of the fully online estimator is approximately:

$$n_{mem} \approx \frac{1+\lambda}{1-\lambda}.$$

B.2.3 The Covariance of Weighted Sample Means

In this section, we will derive a formula to compute the covariance of the weighted sample means. This formula will be important in the following section to roughly approximate the memory of the estimator for the sample mean and covariance matrix with exponentially decaying weights.

Typically, when one computes the conventional sample mean, the covariances of the sample mean vector $\bar{\mathbf{x}}_n$ (note: here we are talking about the covariance matrix for the sample mean $\bar{\mathbf{x}}_n$ and not the estimated covariance matrix of the sample itself. This is somewhat similar to the computation of the standard error of the mean.) will tend to zero as the sample size n grows larger. Similarly to the standard error of the mean, which can be expressed as

$$\sigma_{\bar{X}}^2 = \frac{\sigma^2}{n}, \tag{B.84}$$

the expected covariance matrix $\mathbf{\Sigma}_{\bar{X}}$ is

$$\mathbf{\Sigma}_{\bar{X}} = \frac{1}{n}\mathbf{\Sigma}, \tag{B.85}$$

which indicates, that for a sufficiently large sample n, the estimated mean $\bar{\mathbf{x}}_n$ will most likely be relatively close (under the typical conditions) to the real mean μ_X of the underlying distribution.

However, if the weighted sample means and covariances are computed, this is no longer necessarily the case. As we will see later, in the weighted case, the elements in the covariance matrix of the sample mean will not converge towards zero in certain situations, implying that the sample mean will not converge to the real mean. Some variance will remain in the estimation, and increasing the sample size will not change this. In such cases, we can say that the estimator has a "limited memory". This may be undesirable when estimating stationary distributions on the one hand. On the other hand, a limited memory can make sense if certain statistics of the underlying distribution (e.g., the means or variances) change over time (often called concept drift or concept change), since the estimator can then forget about the past and learn to adapt its parameters to the drifting distribution. To understand the effects that weighted sample statistics generate, we investigate in this section how the covariance of weighted means behave in different settings.

Let us first derive a formulation for the covariance of two weighted means $\bar{X}_n$ and $\bar{Y}_n$, which are computed for the two jointly distributed random variables X and Y. The covariance of two jointly distributed random variables is defined as:

$$\begin{aligned}\operatorname{cov}(X, Y) &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].\end{aligned}$$

Then, the covariance of the two sample means $\bar{X}_n$ and $\bar{Y}_n$ is:

$$\operatorname{cov}(\bar{X}_n, \bar{Y}_n) = \operatorname{cov}\left(\frac{\sum_{i=1}^n w_i' X_i}{\sum_{i=1}^n w_i'}, \frac{\sum_{i=1}^n w_i' Y_i}{\sum_{i=1}^n w_i'}\right)$$

$$= \operatorname{cov}\left(\frac{1}{W_n} \sum_{i=1}^{n} w_i' X_i, \frac{1}{W_n} \sum_{i=1}^{n} w_i' Y_i \right) \tag{B.86}$$

with random variables X_i and Y_i and where

$$W_n = \sum_{i=1}^{n} w_i'.$$

Let us assume that all weights are already normalized so that

$$w_i = \frac{w_i'}{W_n} = \frac{w_i'}{\sum_{i=1}^{n} w_i'}.$$

Then, Eq. (B.86), simplifies to:

$$\begin{aligned} \operatorname{cov}(\bar{X}_n, \bar{Y}_n) &= \operatorname{cov}\left(\sum_{i=1}^{n} w_i X_i, \sum_{i=1}^{n} w_i Y_i \right) \\ &= \mathbb{E}\left[\sum_{i=1}^{n} w_i X_i \cdot \sum_{j=1}^{n} w_j Y_j \right] - \mathbb{E}[\bar{X}_n]\mathbb{E}[\bar{Y}_n] \end{aligned} \tag{B.87}$$

Let the expected values $\mu_{\bar{X}_n} = \mathbb{E}[\bar{X}_n]$ and $\mu_{\bar{Y}_n} = \mathbb{E}[\bar{Y}_n]$ be the real means of the two jointly distributed random variables, so that Eq. (B.87) can be written as:

$$\begin{aligned} \operatorname{cov}(\bar{X}_n, \bar{Y}_n) &= \mathbb{E}\left[\sum_{i=1}^{n} w_i X_i \cdot \sum_{j=1}^{n} w_j Y_j \right] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\ &= \mathbb{E}\left[\sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j X_i Y_j \right] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\ &= \mathbb{E}\left[\sum_{i=1}^{n} w_i^2 X_i Y_j + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} w_i w_j X_i Y_j \right] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \end{aligned} \tag{B.88}$$

Note that in above Eq. (B.88), the original sum was split into two, so that the paired (mutually dependent) data points (X_i, Y_i) share the same sum. All other pairs $\{(X_i, Y_j) \mid i \neq j\}$ are statistically independent and can be treated differently in the following. With the relation

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y],$$

for independent random variables X and Y, we get for Eq. (B.88):

$$\begin{aligned}
\operatorname{cov}(\bar{X}_n, \bar{Y}_n) &= \mathbb{E}\left[\sum_{i=1}^{n} w_i^2 X_i Y_j + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j X_i Y_j\right] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \mathbb{E}\left[\sum_{i=1}^{n} w_i^2 X_i Y_j\right] + \mathbb{E}\left[\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j X_i Y_j\right] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathbb{E}[X_i Y_j] + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j \mathbb{E}[X_i Y_j] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathbb{E}[X_i Y_j] + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j \mathbb{E}[X_i]\mathbb{E}[Y_j] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n}. \qquad \text{(B.89)}
\end{aligned}$$

Since $\mathbb{E}[X_i] = \mu_{X_i} = \mu_{\bar{X}_n}$ and $\mathbb{E}[Y_i] = \mu_{Y_i} = \mu_{\bar{Y}_n}$ (the expected value of the sample mean is the same as the expected value of each element of the sample), we can continue with

$$\begin{aligned}
\operatorname{cov}(\bar{X}_n, \bar{Y}_n) &= \sum_{i=1}^{n} w_i^2 \mathbb{E}[X_i Y_j] + \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j \mathbb{E}[X_i]\mathbb{E}[Y_j] - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathbb{E}[X_i Y_j] + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \Big[\operatorname{cov}(X_i, Y_i) + \mu_{\bar{X}_i}\mu_{\bar{Y}_i}\Big] + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \Big[\operatorname{cov}(X_i, Y_i) + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\Big] + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \operatorname{cov}(X_i, Y_i) + \sum_{i=1}^{n} w_i^2 \mu_{\bar{X}_n}\mu_{\bar{Y}_n} + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j - \mu_{\bar{X}_n}\mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \operatorname{cov}(X_i, Y_i) + \underbrace{\mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n} w_i^2 + \mu_{\bar{X}_n}\mu_{\bar{Y}_n}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} w_i w_j}_{\text{both sums can be merged again}} - \mu_{\bar{X}_n}\mu_{\bar{Y}_n}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) + \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j - \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) + \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \sum_{i=1}^{n} w_i \underbrace{\sum_{j=1}^{n} w_j}_{=1} - \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) + \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \underbrace{\sum_{i=1}^{n} w_i}_{=1} - \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) + \mu_{\bar{X}_n} \mu_{\bar{Y}_n} - \mu_{\bar{X}_n} \mu_{\bar{Y}_n} \\
&= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) \qquad \text{(B.90)}
\end{aligned}
$$

Finally, since all pairs (X_i, Y_i) are independent and identically distributed (IID), we can write:

$$
\begin{aligned}
\mathrm{cov}(\bar{X}_n, \bar{Y}_n) &= \sum_{i=1}^{n} w_i^2 \mathrm{cov}(X_i, Y_i) \\
&= \mathrm{cov}(X, Y) \sum_{i=1}^{n} w_i^2. \qquad \text{(B.91)}
\end{aligned}
$$

This simple relation that we finally derived in Eq. (B.91) will help us later to determine the "memory" of exponentially decaying weighted estimators.

B.2.4 Memory of the Exponentially Decaying Estimator

Due to the exponentially decaying weights, the estimator described in Sec. B.2.2 has a limited historical memory since older observations fade out more and more with every new data point. With such an approach, it is possible to adapt the parameters to drifting (changing) distributions, however, at the cost of less accuracy when the data generating process is a stationary distribution. For example, this means that for forgetting factors $\lambda < 1$, the (co-) variances of the mean vector do not converge to zero for large sample sizes n. There will always be some fixed amount of noise left in the estimation, depending on the "rate" of forgetting (specified by λ). In this section, we attempt to answer the following question:
What is the memory n_{mem} of an estimator with exponentially decaying weights? Hence: For which sample size n_{mem}, when computing the ordinary mean (without forgetting) instead, can we obtain the same distribution parameters $\mu_{\bar{X}}$ and $\mathbf{\Sigma}_{\bar{X}}$? In other words, we are looking

for a corresponding ordinary estimator that only considers the last n_{mem} data points to compute the mean.

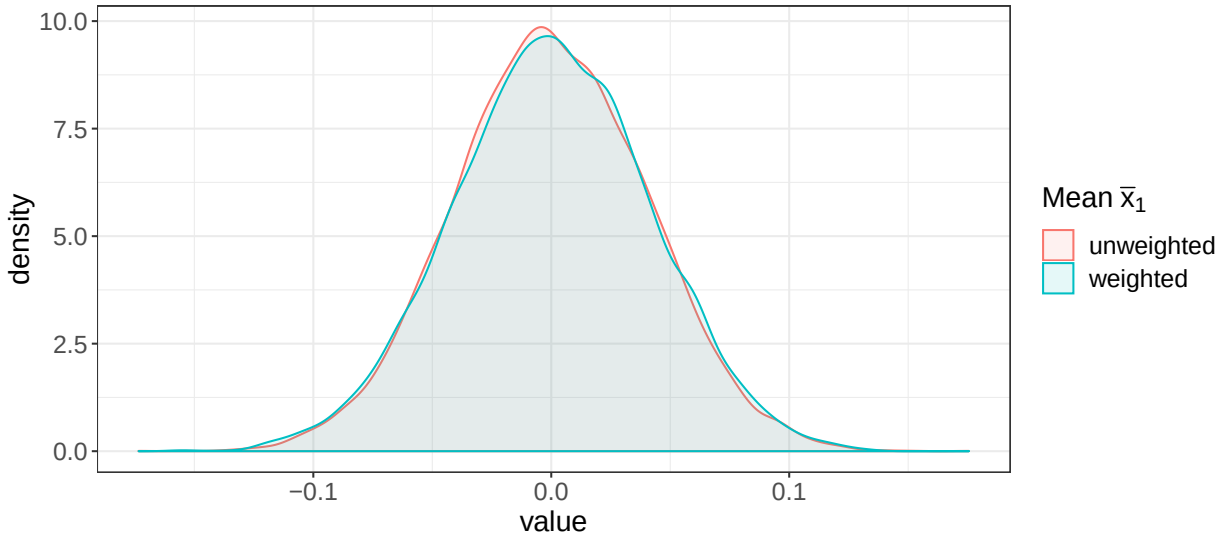

Figure B.1: Comparison of the distributions of the sample means (10^4 samples, sampled from a uniform distribution) for the unweighted and weighted case. The forgetting factor for the weighted estimator is set to $\lambda = 0.99$. The unweighted estimator computes the mean for samples of size $n_{mem} = 199$. Both distributions match closely, confirming our finding regarding the memory of the exponentially decaying estimator.

Intuitively, one might assume for the second question that the sample size n for the ordinary mean corresponds to the value of the normalization factor W_n since for large n, W_n converges – according to

$$W_n = \sum_{i=1}^{n} w_i' = \sum_{i=1}^{n} \lambda^{n-i} = \frac{1-\lambda^n}{1-\lambda}$$

– towards a fixed value (for $\lambda < 1$):

$$\lim_{n\to\infty} W_n = \lim_{n\to\infty} \frac{1-\lambda^n}{1-\lambda} = \frac{1}{1-\lambda}.$$

For example, for a forgetting factor of $\lambda = 0.99$, W_n converges towards $W_n = 100$. Hence, one would be tempted to assume that $n_{mem} = W_n = 100$. However, if we test this hypothesis in a small simulation, we find that memory must be larger. Hence, we have to find another way to estimate the memory n_{mem}. We can do this by actually computing the covariance matrix for the weighted mean. The derivations are shown in the previous

section, in which the following resulting equation is found:

$$\begin{aligned}\operatorname{cov}(\bar{X}_n, \bar{Y}_n) &= \sum_{i=1}^{n} w_i^2 \operatorname{cov}(X_i, Y_i) \\ &= \operatorname{cov}(X, Y) \sum_{i=1}^{n} w_i^2.\end{aligned}$$

If we extend the above equation to the setting with exponentially decaying weights, we obtain:

$$\begin{aligned}\sum_{i=1}^{n} w_i^2 &= \sum_{i=1}^{n} \Big(\frac{\lambda^{n-i}}{W_n}\Big)^2 = \sum_{i=1}^{n} \frac{(\lambda^{n-i})^2}{W_n^2} = \sum_{i=1}^{n} \frac{\lambda^{2(n-i)}}{\Big(\sum_{i=1}^{n} \lambda^{n-i}\Big)^2} \\ &= \sum_{i=1}^{n} \frac{\lambda^{2(n-i)}}{\Big(\frac{1-\lambda^n}{1-\lambda}\Big)^2} = \Big(\frac{1-\lambda}{1-\lambda^n}\Big)^2 \sum_{i=1}^{n} \lambda^{2(n-i)} \\ &= \Big(\frac{1-\lambda}{1-\lambda^n}\Big)^2 \frac{1-\lambda^{2n}}{1-\lambda^2}.\end{aligned}$$

For $0 < \lambda < 1$ and large n, the above expression converges towards

$$\lim_{n\to\infty} \sum_{i=1}^{n} w_i^2 = \lim_{n\to\infty} \Big(\frac{1-\lambda}{1-\lambda^n}\Big)^2 \frac{1-\lambda^{2n}}{1-\lambda^2} = \frac{(1-\lambda)^2}{1-\lambda^2}.$$

Due to the central limit theorem (CLT), we know that the ordinary mean vector $\bar{\mathbf{X}}_n$ is distributed as

$$\bar{\mathbf{X}}_n \sim \mathcal{N}\big(\mu_X, \frac{1}{n}\boldsymbol{\Sigma}\big),$$

so that we have to solve the following correspondence for n_{mem}:

$$\begin{aligned}\boldsymbol{\Sigma}\frac{1}{n_{mem}} &= \boldsymbol{\Sigma} \cdot \sum_{i=1}^{n} w_i^2 \\ n_{mem} &= \frac{1}{\sum_{i=1}^{n} w_i^2} = \Big(\frac{1-\lambda^n}{1-\lambda}\Big)^2 \frac{1-\lambda^2}{1-\lambda^{2n}},\end{aligned}$$

For large sample sizes n, n_{mem} converges to:

$$\begin{aligned}\lim_{n\to\infty} n_{mem} &= \frac{1-\lambda^2}{(1-\lambda)^2} = \frac{1-\lambda^2}{(1-\lambda)^2}\cdot\frac{1+\lambda}{1+\lambda} \\ &= \frac{1+\lambda}{1-\lambda}\cdot\frac{1-\lambda^2}{(1+\lambda)(1-\lambda)} = \frac{1+\lambda}{1-\lambda}\cdot\frac{1-\lambda^2}{1-\lambda^2} \\ &= \frac{1+\lambda}{1-\lambda}.\end{aligned}$$

For our previous example with $\lambda = .99$, this would mean that the memory of the estimator is approx. $n_{mem} \approx 199$. The distributions of the sample means for the weighted and unweighted estimator are visualized in Fig. B.1.

In this section, we found that the memory of the exponentially decaying estimator of the sample mean and sample covariance has a memory of approximately:

$$n_{mem} \approx \frac{1+\lambda}{1-\lambda}. \tag{B.92}$$

B.3 Relationship of the Mahalanobis Distance and the Chi-Square Distribution

In practice, sometimes (multivariate) Gaussian distributions are used for anomaly detection tasks (assuming that the considered data is approximately normally distributed): the parameters of the Gaussian can be estimated using maximum likelihood estimation (MLE) where the maximum likelihood estimate is the sample mean and sample covariance matrix. After estimating the distribution parameters, one has to specify a critical value (anomaly threshold) that separates the normal data from the anomalous data. One possibility is, to determine the critical value based on the probability density function (PDF). A new data point can then be classified as anomalous if the value of the PDF for this new point is below the critical value. In the univariate case, the boundary separates the lower and upper tails of the Gaussian from its center (mean). For a 2-dimensional distribution, the boundary is an ellipse around the center, and in higher dimensions, the boundary can be described by an ellipsoid. All points on the surface of such an ellipsoid have the same Mahalanobis distance to the center of the distribution. Hence, one can also specify a Mahalanobis distance as an anomaly threshold and separate normal and anomalous data points according to this distance metric. Generally, the Mahalanobis distance is parameter-free and does not require any particular assumptions about the data distribution (such as a normal distribution). However, as we will show in the following, for (roughly) Gaussian-distributed data, the squared Mahalanobis distance follows a Chi-square distribution. This relationship can also be verified empirically with a quantile-quantile plot.

B.3.1 Prerequisites

B.3.1.1 Matrix Algebra

Generally, the product of a $n \times \ell$ matrix $\mathbf{A}$ and a $\ell \times p$ matrix $\mathbf{B}$ is defined as:

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} \mathbf{A}_{ik}\mathbf{B}_{kj}$$

Then, the multiplication of a matrix $\mathbf{A}$ with its transpose $\mathbf{A}^\mathsf{T}$ can be written as:

$$\begin{aligned}(\mathbf{AA}^\mathsf{T})_{ij} &= \sum_{k=1}^{\ell} \mathbf{A}_{ik}\mathbf{A}^\mathsf{T}_{kj} = \sum_{k=1}^{\ell} \mathbf{A}_{ik}\mathbf{A}_{jk}, \\ \mathbf{AA}^\mathsf{T} &= \sum_{k=1}^{\ell} \mathbf{a}_k\mathbf{a}_k^\mathsf{T}, \end{aligned} \tag{B.93}$$

where $\mathbf{a}_k$ is the kth column vector of matrix $\mathbf{A}$.

B.3.1.2 Eigenvalues and Eigenvectors

When the eigenvectors of $n \times n$ matrix $\mathbf{A}$ are arranged in a squared $n \times n$ matrix $\mathbf{U}$, where the i-th column represents the i-th eigenvector $\mathbf{u}^{(i)}$, we have:

$$\begin{aligned}\mathbf{AU} &= \mathbf{U\Lambda} \\ \mathbf{A} &= \mathbf{U\Lambda U}^{-1}, \end{aligned} \tag{B.94}$$

commonly referred to as eigenvalue decomposition, where $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues λ_i of the corresponding eigenvectors. If $\mathbf{A}$ is a symmetric matrix, the eigenvectors are orthogonal (orthonormal) and the matrix $\mathbf{U}$ is orthogonal as well (the product with its transpose is the identity matrix). In this case $\mathbf{U}^{-1} = \mathbf{U}^\mathsf{T}$, and equation (B.94) can be written as $\mathbf{A} = \mathbf{U\Lambda U}^\mathsf{T}$. The square root of $\mathbf{A}$ (written here as $\mathbf{A}^{\frac{1}{2}}$) – such that $\mathbf{A}^{\frac{1}{2}}\mathbf{A}^{\frac{1}{2}} = \mathbf{A}$ – can be written as:

$$\mathbf{A}^{\frac{1}{2}} = \mathbf{U\Lambda}^{\frac{1}{2}}\mathbf{U}^\mathsf{T}, \tag{B.95}$$

$$\begin{aligned}\mathbf{A}^{\frac{1}{2}} \cdot \mathbf{A}^{\frac{1}{2}} &= \mathbf{U\Lambda}^{\frac{1}{2}}\mathbf{U}^\mathsf{T}\mathbf{U\Lambda}^{\frac{1}{2}}\mathbf{U}^\mathsf{T} = \mathbf{U\Lambda}^{\frac{1}{2}}\mathbf{I\Lambda}^{\frac{1}{2}}\mathbf{U}^\mathsf{T} = \mathbf{U\Lambda U}^\mathsf{T} \\ &= \mathbf{A}. \end{aligned}$$

The eigenvalue decomposition of the inverse of a matrix $\mathbf{A}$ can be computed as follows, using the associative property of the matrix product:

$$\begin{aligned}\mathbf{A}^{-1} &= \left(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1}\right)^{-1} = \left(\mathbf{U}^{-1}\right)^{-1}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1} = \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1} \\ &= \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{\mathsf{T}} \end{aligned} \tag{B.96}$$

Note that $\boldsymbol{\Lambda}^{-1}$ is again a diagonal matrix containing the reciprocal eigenvalues of $\mathbf{A}$.

B.3.1.3 Linear Affine Transform of a Normally Distributed Random Variable

If we apply a linear affine transform to a normal random variable $X \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ with a mean vector $\boldsymbol{\mu}_x$ and a covariance matrix $\boldsymbol{\Sigma}_x$, we obtain a new random variable Y:

$$Y = \mathbf{A}X + \mathbf{b}. \tag{B.97}$$

One can compute the new mean $\boldsymbol{\mu}_y$ and covariance matrix $\boldsymbol{\Sigma}_y$ for Y:

$$\begin{aligned}\boldsymbol{\mu}_y &= \mathbb{E}\{Y\} = \mathbb{E}\{\mathbf{A}X + \mathbf{b}\} = \mathbf{A}\mathbb{E}\{\mathbf{X}\} + \mathbf{b} \\ &= \mathbf{A}\boldsymbol{\mu}_x + \mathbf{b}, \end{aligned} \tag{B.98}$$

$$\begin{aligned}\boldsymbol{\Sigma}_y &= \mathbb{E}\{(Y - \boldsymbol{\mu}_y)(Y - \boldsymbol{\mu}_y)^{\mathsf{T}}\} \\ &= \mathbb{E}\{\left[(\mathbf{A}X + \mathbf{b}) - (\mathbf{A}\boldsymbol{\mu}_x + \mathbf{b})\right]\left[(\mathbf{A}X + \mathbf{b}) - (\mathbf{A}\boldsymbol{\mu}_x + \mathbf{b})\right]^{\mathsf{T}}\} \\ &= \mathbb{E}\{\left[\mathbf{A}(X - \boldsymbol{\mu}_x)\right]\left[\mathbf{A}(X - \boldsymbol{\mu}_x)\right]^{\mathsf{T}}\} \\ &= \mathbb{E}\{\mathbf{A}(X - \boldsymbol{\mu}_x)(X - \boldsymbol{\mu}_x)^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}\} \\ &= \mathbf{A}\mathbb{E}\{(X - \boldsymbol{\mu}_x)(X - \boldsymbol{\mu}_x)^{\mathsf{T}}\}\mathbf{A}^{\mathsf{T}} \\ &= \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^{\mathsf{T}} \end{aligned} \tag{B.99}$$

B.3.2 The Squared Mahalanobis Distance follows a Chi-Square Distribution

In this section we prove the conjecture: "The squared Mahalanobis distance of a Gaussian-distributed random vector $\mathbf{X}$ and the center $\boldsymbol{\mu}$ of this Gaussian distribution follows a Chi-square distribution."

B.3.2.1 Derivation Based on the Eigenvalue Decomposition

The Mahalanobis distance between two points $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{y})}. \tag{B.100}$$

Thus, the squared Mahalanobis distance of a random vector $\mathbf{X}$ and the center $\boldsymbol{\mu}$ of a multivariate Gaussian distribution is defined as:

$$D = d(\mathbf{X}, \boldsymbol{\mu})^2 = (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\mathbf{X} - \boldsymbol{\mu}), \tag{B.101}$$

where $\boldsymbol{\Sigma}$ is a $\ell \times \ell$ covariance matrix and $\boldsymbol{\mu} \in \mathbb{R}^\ell$ is the mean vector. In order to achieve a different representation of D one can first perform an eigenvalue decomposition on $\boldsymbol{\Sigma}^{-1}$ which is (with Eq. (B.96) and assuming orthonormal eigenvectors):

$$\boldsymbol{\Sigma}^{-1} = \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1} = \mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{T} \tag{B.102}$$

With Eq. (B.93) we obtain (cf. [15, p. 80]):

$$\boldsymbol{\Sigma}^{-1} = \sum_{k=1}^{\ell} \lambda_k^{-1} \mathbf{u}_k \mathbf{u}_k^\mathsf{T} \tag{B.103}$$

where $\mathbf{u}_k$ is the k-th eigenvector of the corresponding eigenvalue λ_k. Plugging (B.103) back into (B.101) results in:

$$\begin{aligned}
D = (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\mathbf{X} - \boldsymbol{\mu}) &= (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \left(\sum_{k=1}^{\ell} \lambda_k^{-1} \mathbf{u}_k \mathbf{u}_k^\mathsf{T} \right) (\mathbf{X} - \boldsymbol{\mu}) \\
&= \sum_{k=1}^{\ell} \lambda_k^{-1} (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \mathbf{u}_k \mathbf{u}_k^\mathsf{T} (\mathbf{X} - \boldsymbol{\mu}) \\
&= \sum_{k=1}^{\ell} \lambda_k^{-1} \left[\mathbf{u}_k^\mathsf{T} (\mathbf{X} - \boldsymbol{\mu}) \right]^2 = \sum_{k=1}^{\ell} \left[\lambda_k^{-\frac{1}{2}} \mathbf{u}_k^\mathsf{T} (\mathbf{X} - \boldsymbol{\mu}) \right]^2 \\
&= \sum_{k=1}^{\ell} Y_k^2
\end{aligned}$$

where Y_k is a new random variable based on an affine linear transform of the random vector $\mathbf{X}$. According to Eq. (B.98) , we have $\mathbf{Z} = (\mathbf{X} - \boldsymbol{\mu}) \sim N(\mathbf{0}, \Sigma)$. If we set $\mathbf{a}_k^\mathsf{T} = \lambda_k^{-\frac{1}{2}} \mathbf{u}_k^\mathsf{T}$ then we get $Y_k = \mathbf{a}_k^\mathsf{T} \mathbf{Z} = \lambda_k^{-\frac{1}{2}} \mathbf{u}_k^\mathsf{T} \mathbf{Z}$. Note that Y_k is now a random variable drawn from a univariate normal distribution $Y_k \sim N(0, \sigma_k^2)$, where, according to (B.99):

$$\sigma_k^2 = \mathbf{a}_k^\mathsf{T} \Sigma \mathbf{a}_k = \lambda_k^{-\frac{1}{2}} \mathbf{u}_k^\mathsf{T} \Sigma \lambda_k^{-\frac{1}{2}} \mathbf{u}_k \tag{B.104}$$

$$= \lambda_k^{-1} \mathbf{u}_k^\mathsf{T} \Sigma \mathbf{u}_k \tag{B.105}$$

If we insert $\mathbf{\Sigma} = \sum_{j=1}^{\ell} \lambda_j \mathbf{u}_j \mathbf{u}_j^\mathsf{T}$ into Eq. (B.105), we get:

$$\sigma_k^2 = \lambda_k^{-1} \mathbf{u}_k^\mathsf{T} \Sigma \mathbf{u}_k = \lambda_k^{-1} \mathbf{u}_k^\mathsf{T} \left(\sum_{j=1}^{\ell} \lambda_j \mathbf{u}_j \mathbf{u}_j^\mathsf{T} \right) \mathbf{u}_k = \sum_{j=1}^{\ell} \lambda_k^{-1} \mathbf{u}_k^\mathsf{T} \lambda_j \mathbf{u}_j \mathbf{u}_j^\mathsf{T} \mathbf{u}_k$$
$$= \sum_{j=1}^{\ell} \lambda_k^{-1} \lambda_j \mathbf{u}_k^\mathsf{T} \mathbf{u}_j \mathbf{u}_j^\mathsf{T} \mathbf{u}_k$$

Since all eigenvectors $\mathbf{u}_i$ are pairwise orthonormal the dotted products $\mathbf{u}_k^\mathsf{T} \mathbf{u}_j$ and $\mathbf{u}_j^\mathsf{T} \mathbf{u}_k$ will be zero for $j \neq k$. Only for the case $j = k$ we get:

$$\sigma_k^2 = \lambda_k^{-1} \lambda_k \mathbf{u}_k^\mathsf{T} \mathbf{u}_k \mathbf{u}_k^\mathsf{T} \mathbf{u}_k = \lambda_k^{-1} \lambda_k ||\mathbf{u}_k||^2 ||\mathbf{u}_k||^2 = \lambda_k^{-1} \lambda_k ||\mathbf{u}_k||^2 ||\mathbf{u}_k||^2$$
$$= 1,$$

since the the norm $||\mathbf{u}_k||$ of a orthonormal eigenvector is equal to 1. Thus, the squared Mahalanobis distance can be expressed as: $D = \sum_{k=1}^{\ell} Y_k^2$, where $Y_k \sim N(0, 1)$. Now the Chi-square distribution with ℓ degrees of freedom is exactly defined as being the distribution of a variable which is the sum of the squares of ℓ random variables being standard normally distributed. Hence, D is Chi-square distributed with ℓ degrees of freedom.

B.3.2.2 Alternative Derivation Based on the Whitening Property of the Mahalanobis Distance

Since the inverse $\mathbf{\Sigma}^{-1}$ of the covariance matrix $\mathbf{\Sigma}$ is also a symmetric matrix, its square root can be found – based on Eq. (B.95) – to be a symmetric matrix. In this case we can write the squared Mahalanobis distance as

$$D = (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \mathbf{\Sigma}^{-1} (\mathbf{X} - \boldsymbol{\mu}) = (\mathbf{X} - \boldsymbol{\mu})^\mathsf{T} \mathbf{\Sigma}^{-\frac{1}{2}} \mathbf{\Sigma}^{-\frac{1}{2}} (\mathbf{X} - \boldsymbol{\mu})$$
$$= \left(\mathbf{\Sigma}^{-\frac{1}{2}} (\mathbf{X} - \boldsymbol{\mu}) \right)^\mathsf{T} \left(\mathbf{\Sigma}^{-\frac{1}{2}} (\mathbf{X} - \boldsymbol{\mu}) \right) = \mathbf{Y}^\mathsf{T} \mathbf{Y} = ||\mathbf{Y}||^2$$
$$= \sum_{k=1}^{\ell} Y_k^2$$

The multiplication $\mathbf{Y} = \mathbf{W}\mathbf{Z}$, with $\mathbf{W} = \mathbf{\Sigma}^{-\frac{1}{2}}$ and $\mathbf{Z} = \mathbf{X} - \boldsymbol{\mu}$ is typically referred to as a whitening transform, where in this case $\mathbf{W} = \mathbf{\Sigma}^{-\frac{1}{2}}$ is the so called Mahalanobis (or ZCA) whitening matrix. $\mathbf{Y}$ has zero mean, since $(\mathbf{X} - \boldsymbol{\mu}) \sim N(\mathbf{0}, \Sigma)$. Due to the (linear) whitening transform the new covariance matrix $\mathbf{\Sigma}_y$ is the identity matrix $\mathbf{I}$, as shown in the following

(using the property in Eq. (B.99)):

$$\begin{aligned}\boldsymbol{\Sigma}_y &= \mathbf{W}\boldsymbol{\Sigma}\mathbf{W}^\mathsf{T} = \boldsymbol{\Sigma}^{-\frac{1}{2}}\boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}^{-\frac{1}{2}}\right)^\mathsf{T} = \boldsymbol{\Sigma}^{-\frac{1}{2}}\left(\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\right)\left(\boldsymbol{\Sigma}^{-\frac{1}{2}}\right)^\mathsf{T} \\ &= \boldsymbol{\Sigma}^{-\frac{1}{2}}\left(\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\right)\boldsymbol{\Sigma}^{-\frac{1}{2}} = \left(\boldsymbol{\Sigma}^{-\frac{1}{2}}\boldsymbol{\Sigma}^{\frac{1}{2}}\right)\left(\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\Sigma}^{-\frac{1}{2}}\right) \\ &= \mathbf{I}.\end{aligned}$$

Hence, all elements Y_k in the random vector $\mathbf{Y}$ are random variables drawn from independent normal distributions $Y_k \sim N(0,1)$, which leads us to the same conclusion as before, that D is Chi-square distributed with ℓ degrees of freedom.

Summary

Up until today, anomaly detection in general and especially for temporal data such as data streams or time series remains a challenging task. Nevertheless, anomaly detection in time series is becoming increasingly important in many domains. For example, more and more industrial machines are equipped with numerous sensors for condition monitoring and predictive maintenance applications, which produce large amounts of data in the shortest time. In such applications, it is crucial to continuously monitor these large data streams and detect anomalous behavior at an early stage to prevent critical systems' failure and possible further damage. Also, in many other application areas (e.g., server technologies & networking, high energy physics & astronomy, or health monitoring systems), vast amounts of data are collected, which have to be analyzed automatically with suitable algorithms. In this context, the necessity for accurate and robust unsupervised learning methods for anomaly detection tasks arose.

In this thesis, we present and discuss several unsupervised anomaly detection algorithms for time series.

Our first algorithm, called SORAD (Simple Online Regression for Anomaly Detection), is presented in Chapter 4. SORAD learns a linear regression model to predict future values of a time series and simultaneously estimate the prediction errors' mean and variance. Data points resulting in prediction errors that significantly deviate from the mean are considered anomalous. SORAD can operate fully online (or using small batches) on any time series after a short transient phase and adapt to changing concepts (due to the forgetting in the linear model and the estimates of the prediction errors). On the Webscope S5 anomaly benchmark [87] SORAD could significantly outperform other algorithms. However, it had some shortcomings on the Numenta Anomaly Benchmark (NAB) [89], where it did not perform well at first.

In Chapter 5, we introduced the DWT-MLEAD (Discrete Wavelet Transform with Maximum Likelihood Estimation for Anomaly Detection in Time Series) algorithm. This algorithm's central idea is to analyze a time series on different frequency/time scales using the discrete wavelet transform (DWT). DWT-MLEAD (i) computes a decimating DWT using Haar wavelets, (ii) slides a window over the individual frequency scales and estimates the mean and covariance matrix of the collected points, and (iii) identifies unusual points based on a Mahalanobis distance. Then (iv), the unusual points are flagged as events and passed down the DWT-tree. Finally (v), the events of all frequency scales are aggregated at the

original time scale and cause the algorithm to fire an anomaly if a specified threshold is exceeded. Using this multi-scale approach, it is possible to detect short-range and also longer-range anomalies. Initially designed as an offline algorithm, we showed that DWT-MLEAD could also be used in online settings after a few modifications. We found that DWT-MLEAD outperforms other state-of-the-art algorithms, the Webscope S5 and the NAB data using only one hyperparameter setting.

The LSTM-AD algorithm, described in Chapter 6, learns to predict normal time-series behavior using recurrent long short-term memory (LSTM) neural networks. From the general idea, LSTM-AD is similar to SORAD. However, it extends it in several aspects: Instead of using a simple linear regression model LSTM-AD uses a stack of LSTM layers to learn more complicated temporal patterns. Furthermore, now 25 prediction horizons are used instead of only one. The prediction errors for several prediction horizons are used to indicate anomalous behavior (using the Mahalanobis distance of the prediction errors as anomaly score). Several additional extensions, such as a window-based error correction and an unsupervised outlier removal method are introduced. LSTM-AD was evaluated on the well-known MIT-BIH ECG [52, 112, 113] data set and could obtain good initial results. Later, in Chapter 7, even better results are found with an improved hyper-parameter selection.

In Chapter 7, the temporal convolutional network autoencoder (TCN-AE) is introduced. The main idea is to use one TCN [13] as an encoder network and another TCN as a decoder network. In order to create a bottleneck, the encoder downsamples the time series and reduces its dimensionality. After the input time series is encoded into a compressed sequence, a temporal decoder network attempts to reconstruct the original input. TCN employs so-called dilated convolutions, which have their origin in discrete wavelet transforms and create an exponentially increasing receptive field with only a linear increase in the number of trainable weights. We performed our initial experiments with a relatively simple baseline TCN-AE architecture. The baseline TCN-AE demonstrated that it could already learn interesting representations of different Mackey-Glass time series, and it performed well on the Mackey-Glass Anomaly Benchmark (MGAB). However, we noticed a few shortcomings, which we addressed in the following. We showed that several extensions of the baseline model are crucial to improving the overall performance. TCN-AE in its final form produces significantly better results on the MIT-BIH ECG data, outperforming other state-of-the-art algorithms. At the same time, it reduces the computation time and the number of trainable parameters (in comparison to baseline TCN-AE).

Samenvatting

Tot op heden blijft anomaliedetectie in het algemeen en in het bijzonder voor temporele data zoals datastromen of tijdreeksen een uitdagende taak. Toch wordt anomaliedetectie in tijdreeksen steeds belangrijker in vele domeinen. Zo zijn bijvoorbeeld meer en meer industriële machines uitgerust met talrijke sensoren voor conditiebewaking en predictieve onderhoudstoepassingen, die grote hoeveelheden data produceren in de kortst mogelijke tijd. Bij dergelijke toepassingen is het van cruciaal belang deze grote gegevensstromen voortdurend te bewaken en afwijkend gedrag in een vroeg stadium te detecteren om uitval van kritische systemen en eventuele verdere schade te voorkomen. Ook in veel andere toepassingsgebieden (bijv. servertechnologieën & netwerken, hoge-energiefysica & astronomie, of systemen voor gezondheidsbewaking) worden enorme hoeveelheden gegevens verzameld, die automatisch moeten worden geanalyseerd met geschikte algoritmen. In deze context ontstond de noodzaak voor nauwkeurige en robuuste unsupervised learning methoden voor anomaliedetectie.

In dit proefschrift presenteren en bespreken we verschillende algoritmes voor onbewaakte anomaliedetectie voor tijdreeksen.

Ons eerste algoritme, genaamd SORAD (Simple Online Regression for Anomaly Detection), wordt gepresenteerd in Hoofdstuk 4. SORAD leert een lineair regressiemodel om toekomstige waarden van een tijdreeks te voorspellen en schat tegelijkertijd het gemiddelde en de variantie van de voorspelfouten. Gegevenspunten met voorspelfouten die significant anders zijn van het gemiddelde worden als afwijkend beschouwd. SORAD kan volledig online (of met kleine batches) werken op elke tijdreeks na een korte transiënte fase en zich aanpassen aan veranderende concepten (als gevolg van het vergeten in het lineaire model en de schattingen van de voorspelfouten). Op de Webscope S5 anomalie benchmark [87] kon SORAD aanzienlijk beter presteren dan andere algoritmen. SORAD vertoonde echter tekortkomingen op de Numenta Anomaly Benchmark (NAB) [89], waar het aanvankelijk niet goed presteerde.

In hoofdstuk 5 hebben we het DWT-MLEAD (Discrete Wavelet Transform with Maximum Likelihood Estimation for Anomaly Detection in Time Series) algoritme geïntroduceerd. Het centrale idee van dit algoritme is het analyseren van een tijdreeks op verschillende frequentie/tijdschalen met behulp van de discrete wavelet-transformatie (DWT). DWT-MLEAD (i) berekent een decimerende DWT met gebruikmaking van Haar wavelets, (ii) schuift een venster over de afzonderlijke frequentieschalen en schat het gemiddelde en

de covariantiematrix van de verzamelde punten, en (iii) identificeert ongebruikelijke punten op basis van een Mahalanobis-afstand. Dan (iv) worden de ongebruikelijke punten gemarkeerd als gebeurtenissen en door de DWT-boom geleid. Tenslotte (v) worden de gebeurtenissen van alle frequentieschalen samengevoegd op de oorspronkelijke tijdschaal en zorgt het algoritme ervoor dat een anomalie wordt afgevuurd als een gespecificeerde drempel wordt overschreden. Door gebruik te maken van deze multischaalbenadering is het mogelijk anomalieën op korte en ook op langere termijn te detecteren. DWT-MLEAD was in eerste instantie ontworpen als offline algoritme, maar na enkele aanpassingen is gebleken dat het ook in een online omgeving kan worden gebruikt. We ontdekten dat DWT-MLEAD beter presteert dan andere state-of-the-art algoritmen op de Webscope S5 en de NAB data met slechts één hyperparameter instelling.

Het LSTM-AD algoritme, beschreven in Hoofdstuk 6, leert normaal tijdreeksgedrag te voorspellen met behulp van recurrente long short-term memory (LSTM) neurale netwerken. Het algemene idee in LSTM-AD is vergelijkbaar met dat van SORAD. Het breidt het echter op verschillende punten uit: In plaats van een eenvoudig lineair regressiemodel te gebruiken, gebruikt LSTM-AD een stapel LSTM-lagen om meer ingewikkelde temporele patronen te leren. Bovendien worden nu 25 voorspellingshorizonnen gebruikt in plaats van slechts één. De voorspellingsfouten voor verschillende voorspellingshorizonten worden gebruikt om afwijkend gedrag aan te geven (waarbij de Mahalanobis-afstand van de voorspellingsfouten als anomaliescore wordt gebruikt). Verschillende extra uitbreidingen, zoals een venstergebaseerde foutcorrectie en een niet gesuperviseerde methode voor het verwijderen van uitschieter worden geïntroduceerd. LSTM-AD werd geëvalueerd op de bekende MIT-BIH ECG-gegevensset en kon goede eerste resultaten behalen. Later, in hoofdstuk 7, worden nog betere resultaten gevonden met een verbeterde hyper-parameter selectie.

In hoofdstuk 7 wordt de Temporal Convolutional Network Autoencoder (TCN-AE) geïntroduceerd. Het hoofdidee is om een TCN [13] te gebruiken als encoder netwerk en een ander TCN als decoder netwerk. Om een knelpunt te creëren, wordt in de encoder de tijdreeks gedownsamplesed en de dimensionaliteit gereduceerd. Nadat de ingevoerde tijdreeks is gecodeerd tot een gecomprimeerde reeks, tracht een temporaal decodernetwerk de oorspronkelijke invoer te reconstrueren. TCN maakt gebruik van zogenaamde gedilateerde convoluties, die hun oorsprong vinden in discrete wavelet transformaties en een exponentieel toenemend receptief veld creëren met slechts een lineaire toename van het aantal trainbare gewichten. Wij hebben onze eerste experimenten uitgevoerd met een relatief eenvoudige basis TCN-AE architectuur. De baseline TCN-AE toonde aan dat het al interessante representaties van verschillende Mackey-Glass tijdreeksen kon leren, en dat het goed presteerde op de Mackey-Glass Anomaly Benchmark (MGAB). We merkten echter een paar tekortkomingen op, die we in die we in later onderzoek hebben aangepakt. We toonden aan dat verschillende uitbreidingen van het basismodel van cruciaal belang zijn om de algemene prestaties te verbeteren. TCN-AE in zijn uiteindelijke vorm levert significant betere resultaten op op de MIT-BIH ECG data, en presteert beter dan andere state-of-the-art algoritmen. Tegelijkertijd vermindert het de rekentijd en het aantal trainbare parameters (in vergelijking met baseline TCN-AE).

About the Author

Markus Thill, born 1987 in Patan, Nepal, received his B.Sc. degree in computer engineering from TH Köln – University of Applied Sciences, Cologne, Germany, in 2012. He graduated with a master's degree in Automation & IT in 2015. Since 2012 he is a research associate at TH Köln. In 2017, he joined the Natural Computing Group of Prof. Dr. Thomas Bäck at the Leiden Institute for Advanced Computer Science (LIACS) as an external Ph.D. student. His research interests are in the fields of time series analysis, anomaly detection, Reinforcement Learning (e.g., for board games), applied Deep Learning (DL) and incremental (on-line) learning procedures.