Embrik Tvenge Einang and Alfred Sollie Rønning

Anomaly detection for industrial time series

Master's thesis in Informatics
Supervisor: Ole Jakob Mengshoel
June 2020

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Carbon anodes are a crucial component in the aluminium electrolysis process. Monitoring the current draw through the hanger the carbon anode is positioned on allows operators to track and control the electrolysis environment, potentially avoiding catastrophic failure. The IACM-sensor designed in-house at Hydro regularly feeds current draw readings to be further analyzed manually or to be used in control loops, emphasizing the importance of data integrity. By applying machine learning anomaly detection techniques to historic IACM data, we can detect and separate process and sensor anomalies, giving early detection of potentially compromised data.

After performing a hybrid Systematic Literature Review, the architectures presented in the literature was rated according to a set of architectural requirements. Hierarchical Temporal Memory (HTM), LSTM/GRU, and Yet Another Segmentation Algorithm (YASA) with a One-Class Support Vector Machine (OCSVM) were implemented and tested. A qualitative approach was taken, given the lack of available anomaly descriptions and an event log of previous anomalies. Results show promising regression and anomaly detection results for LSTM/GRU and HTM, while the YASA with OCSVM struggled to correctly segment and model the noisy data. To prevent this, smoothing techniques for YASA and data set cleaning from export knowledge was performed.

Anomaly detection for all models was explored. Two methods for separation of sensor and process anomalies were tested, where the comparison of the standard deviation in the current draw and cell voltage yielded a separation.

Future work outlines the importance of re-visiting the problem with a formal anomaly description or the ability to synthetically inject anomalies with comparable signatures. The explainability of the models should be explored and weighted more during the evaluation process, as the results are manually interpreted by operators.

Sammendrag

Karbonanoder er en avgjørende komponent i elektrolyseprosessen for aluminiumsproduksjon. Overvåkning av strømtilførselen gjennom hengeren karbonanoden er festet på gjør det mulig for operatører å spore og kontrollere elektrolysemiljøet, som muliggjør tidlig oppdagelse av potensielle prosessrelaterte svikter. IACM-sensoren designet internt i Hydro avgir kontinuerlig strømavlesninger som blir analysert manuelt eller brukt i reguleringssløyfer, og understreker viktigheten av dataintegritet. Ved å anvende anomali maskinlæringsteknikker på historisk IACM-data kan vi oppdage og skille prosess- og sensoranomalier, noe som kan gi tidlige faresignal på potensielt kompromitterte data.

Etter å ha utført en hybrid Systematic Literature Review, ble arkitekturene presentert i litteraturen vurdert etter et sett med arkitektoniske krav. Hirarchical Temporal Memory (HTM), LSTM/GRU, og Yet Another Segmentation Algorithm (YASA) med en One Class Support Vector Machine (OCSVM) ble implementert og testet. En kvalitativ tilnærming ble tatt, gitt mangelen på tilgjengelige avviksbeskrivelser og/eller en hendelseslogg over tidligere anomalier. Resultater viser lovende regresjons- og anomali-deteksjonsresultater for LSTM / GRU og HTM, mens YASA med OCSVM slet med å segmentere og modellere dataen korrekt. For å forbedre YASA ble det utført glatteteknikker og rengjøring av datasett basert på eksportkunnskap.

Anomalipåvisning for alle modeller ble undersøkt. To metoder for separasjon av sensor- og prosessanomalier ble testet, der sammenligningen av standardavviket i strømtrekk og cellespenning ga en separasjon.

Framtidig arbeid skisserer viktigheten av å besøke problemet på nytt med en formell anomalibeskrivelse eller evnen til å syntetisere anomalier med sammenlignbare signaturer. Modellenes forklarbarhet bør utforskes og kanskje vektes mer under evalueringsprosessen, ettersom resultatene blir tolket manuelt av operatører.

Preface

This Master thesis was written for the Department of Computer Science at NTNU Trondheim, and supervised by Ole Jakob Mengshoel. The thesis was written in cooperation with Norsk Hydro ASA.

We would like to thank our supervisor and master thesis discussion group for enlightening questions and guidance. We would also like to thank Geir Sandnes, Even Gundersen Vee and Steinar Kolås from Norsk Hydro ASA in Årdal for their continual contribution of data, guided tour of the electrolysis hall, domain knowledge and encouraging discussions.

Trondheim, June 2020
Embrik Tvenge Einang and Alfred Sollie Rønning

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides an introduction to the background and motivation for the thesis. Section 1.2 introduces the research goal and questions that lay the foundation for the research, and Section 1.3 presents the structure for the rest of the thesis.

1.1 Background and Motivation

Aluminium is a widely used metal found everywhere in modern life. From building facades to packaging and the automotive industry, aluminium is used and recycled continually to meet the global demand. Being a highly energy-intensive material, averaging 13 kWh per produced kilo of aluminium, optimizing the road from raw material to finished product can provide benefits in reducing the carbon footprint while meeting the increased global demand for aluminium.

In addition to being highly energy-intensive, the aluminium production ecosystem is heavily regulated and data-driven. Intensive use of different sensory is crucial to correctly model and regulate the process. The custom sensor named *IACM* was developed in-house at Hydro to monitor the current draw of each hanger located in an electrolysis cell, providing information about the electrolysis process to ensure correct operating parameters. This information is further used in control loops. Ensuring that the information relayed by the sensor is accurate can avoid decisions made on a false basis, potentially avoiding costly downtime and increased carbon footprint from prolonged power use or re-heating of the cell.

To ensure correct readings, anomaly detection methods can help with locating and classifying anomalies in the time series IACM data. As Hydro has a good understanding of the anomalies related to the electrolysis process and aluminium production in general, the lion's share of the work will be dedicated to detecting sensor anomalies. Sensory anomalies can range from faulty readings produced by low battery or a loose connection, to manual interference with the sensor or its components. No logs of when sensory or process anomalies occurred are available to us. However, we assume most of the data to be of *normal* operation, incentivizing the exploration of unsupervised anomaly detection techniques.

1.2 Goal and Research Questions

The following goal has guided the thesis:

> **Goal** *Detect and categorize sensor-anomalies found in time-series IACM sensor data*

In order to guide the work toward the research goal, two research questions were created.

> **Research Question 1** *How does unique machine learning models perform on the IACM time series data?*

The IACM sensory data provides a unique time series directly related to the current drawn from a total of 40 hangers in an electrolysis cell. Exploring different architectures and learning paradigms can help categorize how efficient they are on the given data set. Also, the industry setting might have special constraints, such as low computational complexity or a limitation on the available normal data, and that will directly influence the relevance of the methods. It is hypothesized that models incorporating and learning temporal events with a non-fixed lag will perform well, due to the seasonal change of the anode every 22-30 days. See Section 2.1.4.

> **Research Question 2** *What implications do these solutions have in regards to detecting and categorizing IACM-sensor anomalies?*

Anomalies in data often translate to significant actionable information. Detecting and reacting to such information is essential to prevent escalation, unexpected maintenance or misguided actions in a control loop. Separating process and sensor anomalies should provide greater insight into how it happens when it happens and how the sensory data readings react accordingly.

1.3 Thesis structure

The following chapter will provide the necessary theory about both the aluminium production process and machine learning methodology to understand the problem formulation and research goal. Chapter 3 provides a hybrid structured literature review(SLR) approach to related work. It also includes a set of Inclusion and Quality criteria used when evaluating the research. Chapter 4 presents the anomaly detection methods and machine learning models implemented and tested. It presents the different data sets used throughout the thesis and the anomaly detection techniques. Chapter 5 presents the results for each method, as well as the separation of process and sensor anomalies. Chapter 6 presents the discussion, evaluation and conclusion of our work. Lastly, future work is outlined.

Chapter 2

Theory

The following chapter will provide a general introduction to the theory and methods implemented. Section 2.1 gives an overview of the aluminium production life cycle, Section 2.2 describes the data notion used and Section 2.3 presents the in-house IACM sensor. Section 2.4 will present a definition of anomaly detection, types of anomalies and the domains represented. Section 2.5 contains a description of machine learning algorithms, and Section 2.6 gives an in-depth view of machine learning architectures used in anomaly detection.

2.1 Aluminium production

In this section, we will give a short introduction of how aluminium is made, the components required and introduce the anode effect.

2.1.1 Alumina production

Aluminium oxide (Alumina or Al_2O_3) is the main raw material required for the production of primary aluminium. It is a white powder extracted and refined from bauxite ore through the Bayer process [24]. Crushed bauxite is digested in a caustic solution at high temperature in pressure tanks named digesters. The insoluble impurities, called red mud, are separated and filtered out from the mixture. The resulting solution goes through precipitation and a calcination step to form alumina crystals and remove excess water—the result is a white powder known commercially as pure alumina.

2.1.2 Electrolyte

The aluminium atoms in alumina are bonded to oxygen and need the bond to be broken by electrolysis to produce aluminium metal. Alumina has a melting temperature of over 2000° C, requiring enormous amounts of energy. To lower the operation costs, it is dissolved in an electrolyte consisting mainly of cryolite, resulting in a lowered melting temperature of 960° C. Cryolite (Na_3AlF_6) usually comprises more than 75%wt, of the electrolyte. It also typically contains 6-13% AlF_3, 4-7% CaF_2, and 2-4% Al_2O_3. The alumina is consumed in the process of producing aluminium, so alumina has to be continuously added to the electrolyte. It is important to control the concentration of alumina. If it is too high, the solution will be oversaturated, resulting in undissolved alumina, and if it is too low, there is a risk the anode-effect explained in section 2.1.5. The electrolyte also works as an electrical conductor, conducting current between the anode and the cathode.

2.1.3 Carbon anodes

For every kilogram aluminium produced, between 0.40-0.45 kilogram carbon anode material is consumed in the process [36], fulfilling the carbon requirement as presented in 2.1.4. The carbon anode consists mainly of calcined petroleum coke, a refined crude oil byproduct from oil refineries. Additionally, coal tar pitch is used as a binder, a liquid hydrocarbon consisting of around 90% carbon. Impurities such as vanadium and phosphorus, present in the petroleum coke, can contaminate either the aluminium process or the electrolyte, causing unwanted behaviour. Therefore, the coke undergoes a calcining process at around 1200 °C to remove impurities.

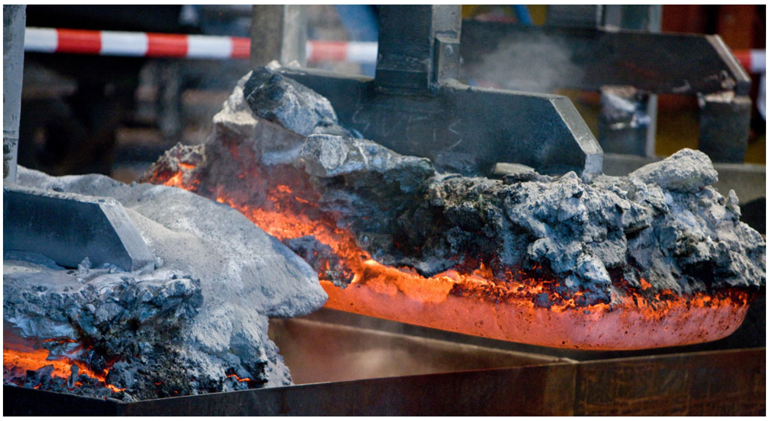


Figure 2.1: Carbon anode at the end of its life cycle.

Furthermore, remnants from previously damaged or used nodes, coke and pitch is mixed and baked at about 1150-1200 °C, causing it to carbonize and harden. To increase electrical contact and physical support, an apparatus consisting of either iron or cobber rods with an iron yoke and stud is attached to the carbon anode as shown in figure 2.1 [36].

2.1.4 The Hall-Hèroult process

In the Hall-Hèroult process, aluminium is reduced from alumina in an electrolytic reaction. In this reaction, the aluminium is separated from the oxygen in the alumina. Aluminium ions receive electrons from the negative cathode, and form molten aluminium, while the oxygen ions react with the anodes to mainly form carbon dioxide. This reaction can be written as:

$$2\,\mathrm{Al_2O_3(dissolved)} + 3\,\mathrm{C(s)} \longrightarrow 4\,\mathrm{Al(l)} + 3\,\mathrm{CO_2(g)} \qquad (2.1)$$

The Hall-Hèroult process requires a large amount of energy. The amount of energy needed to produce 1kg aluminium through the Hall-Hèroult process in modern smelters is close to 13kWh [36].

The electrochemical process happens in steel shells called reduction cells, shown in Figure 2.2. The reduction cells are thermally insulated to reduce heat flux out of the cell, to lower the energy needed to keep the cell temperature at an optimal $960°C$. The temperature is right above the melting temperature of the electrolyte. Keeping the electrolyte viscous will protect the sidewalls from corrosive action from the electrolyte [35]. According to Joule's first law

$$Q = I^2 \cdot R \cdot t \qquad (2.2)$$

Where Q is the amount of heat, I is the electric current flowing through a conductor, R is the amount of electrical resistance in the conductor, and t is the amount of time, the heat generated from passage of electric current through a conductor is proportional with the product of its resistance. By decreasing the Anode Cathode Distance(ACD), one reduces the total resistance on the current passage through the electrolyte, and thus reduces the heat generated. The temperature of the reduction cell is controlled by adjusting the ACD of the carbon anodes.

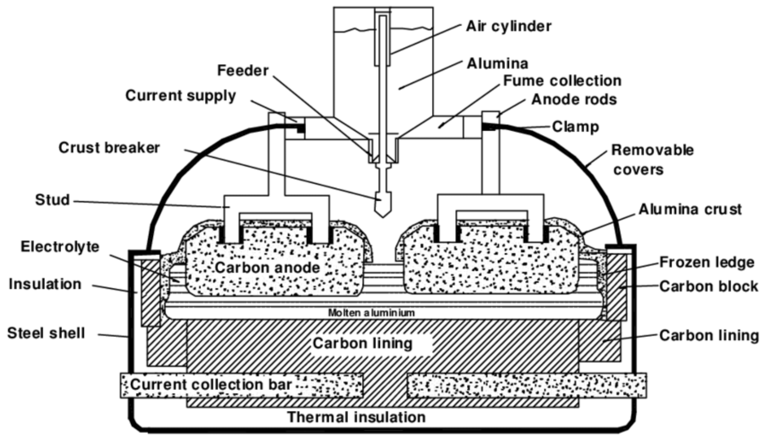


Figure 2.2: Cross sectional scheme of a Hall-Hèroult cell

The reduction cells in Hydro Aluminium's Reference Centre, located in Årdal, where our data is gathered, contains 40 anodes per cell. The pre-baked carbon anodes are consumed during the electrolytic process according to Equation 2.1, causing the anodes to shrink in size. The anode rods are, therefore, gradually lowered downwards into the cell to maintain a constant ACD. When the anodes have been reduced to approximately 1/4 of its size, which occurs every 22-30 days, they are replaced [35].

2.1.5 Anode effect

The anode effect takes place when external factors in the aluminium process, such as an alumina deficit, occurs. An electrically insulating gas layer is created underneath the anodes, causing sudden spikes in the cell voltage, up to 30 to 40 V [36]. This results in a change in the anode gas composition. Instead of the desired outcome of $\mathrm{CO_2}$ from Equation 2.1, $\mathrm{CO(g)}$, and gaseous perfluorocarbon compounds, $\mathrm{CF_4(g)}$ and $\mathrm{C_2F_6(g)}$ [36] are created and emitted to the atmosphere. The perfluorcarbon compounds are undesired due to their high global warming potential, respectively around 6.530 GWP and 11.100 GWP [55], and the following reduction in production. For reference, CO_2 has a GWP of 1.

2.2 Data notation

In this section, we will introduce a standard notation for our data set in addition to a simpler notation. The data set can be written on the form:

$$(t_1, t_2, c, s_1, s_2, ...s_{40}, v) \tag{2.3}$$

where t_1 and t_2 are two unique timestamps following the Unix time/POSIX time, i.e elapsed time since 00:00:00 UTC on January 1970, in milliseconds. The amperage reading is represented by s_x for a total of 40 sensors, v is the cell voltage and c is the total current.

A simpler notation will also be used, describing only the use of one sensor-amperage pair on the form:

$$(t_1, t_2, c, s_x, v) \tag{2.4}$$

Where s_x represents a specified amperage value from one of the 40 available sensors.

2.3 IACM-sensor

2.3.1 Introduction

The IACM-sensor is used to measure the current going through the carbon anode. There is one sensor installed on each anode rod, as shown in Figure 2.3, to capture the current going through all the anodes in the cell. Due to the high temperature and corrosive nature of the electrolyte, a sensors life will be short-lived inside the electrolyte and is therefore placed on the anode rod. The state of the electrolysis process is highly dependent on the amount of currency going through the electrolyte, and unwanted events such as the anode effect from Section 2.1.5 will affect the current. Analyzing the current draw can be used to detect and react to such events to prevent them from escalating.

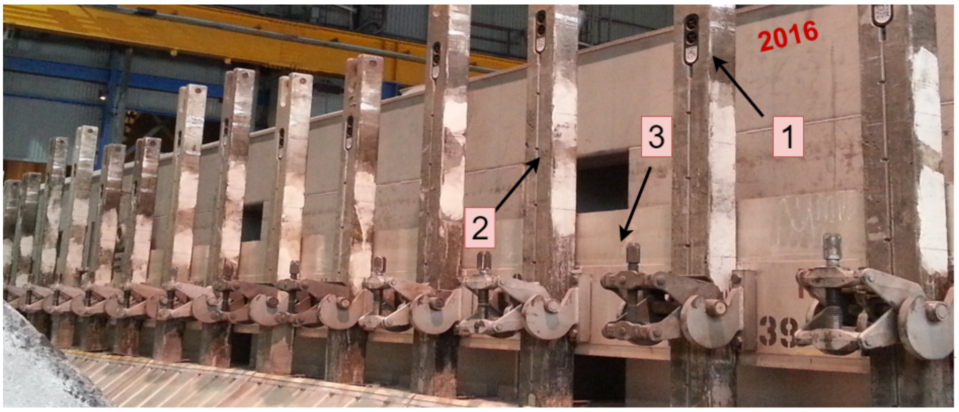


Figure 2.3: Installed IACM-sensors on hangers attached to an electrolysis cell. 1) The black box is connected to the sensor. It contains battery, communication hardware and some on-board processing power. 2) The hanger/anode rod with a groove milled into it for housing the temperature gauge going into the cell. 3) The cell itself. In our case one cell contains a total of 40 anode rods.

2.3.2 Sensor description

The sensor is battery powered and based on SmartMesh IP. SmartMesh IP is an intelligent protocol stack for wireless sensor networks based on the 6LoWPAN and 802.15.4e standards. The packages are routed in a multi-hop network where each wireless node knows when to listen, talk or sleep, resulting in very low power usage [70]. The data on the form presented in Section 2.3 is then sent to and stored in a Cassandra cloud database. The hierarchy of the network is shown in Figure 2.4.

The IACM-sensor reads the voltage drop over 10cm, and temperature from the anode rod. The temperature is used to determine the electrical resistance of the metal, and the current I going through the anode is calculated from the resistance R and the voltage V using Ohm's law from Equation (2.5).

$$\mathrm{I = V/R} \tag{2.5}$$

The temperature at the sensor-module is approximately $100°$C. The sensor and the battery are specifically designed to survive this environment, but unexpected failures still occur. Examples of such shortcomings are failing hardware, loose contacts, slowly degrading/ageing sensors and loss of power. Sensor data is used by algorithms in several control loops, determining everything from cell health to early detection of abnormal behaviour, so it is crucial to detect unreliable readings.

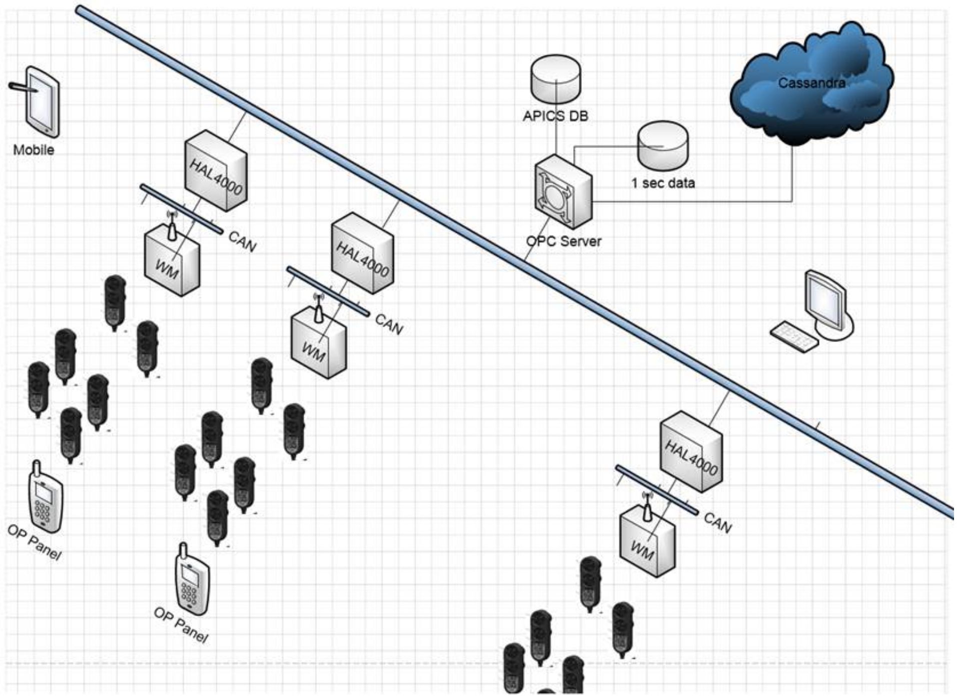


Figure 2.4: Overview of the sensor ecosystem. Multiple sensors are attached to each cell. The communication is handled through a wireless medium and used in the cells control system, HAL4000. An OPC-UA server receives and stores the 1 Hz sensor reading to a Cassandra cloud DB and a local DB.

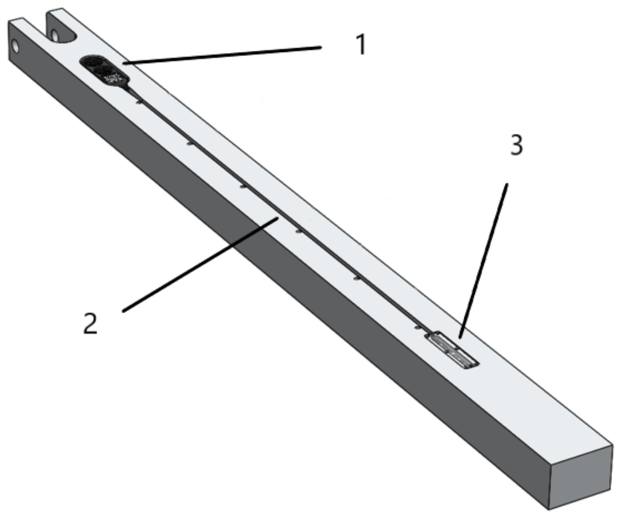


Figure 2.5: Overview on an installed IACM-sensor. The sensor is attached to a milled slot in the carbon rod. 1) Processing unit and battery. 2) Wire connected to the sensor. 3) Sensor, measuring the voltage drop over 10cm.

2.4 Anomaly definition, types and domains

2.4.1 Definition

Before we can present machine learning architectures used in anomaly detection, we have to agree upon a definition. The literature often use novelty detection [40] and outlier detection [6] as two synonyms for anomaly detection. They originate from different domains of application, and there is no universally agreed-upon terminology. Pimentel et al. define novelty detection as 'the task of recognizing that test data differ in some respect from the data that are available during training' [54], while anomaly detection and outlier detection can be defined as 'a pattern that does not conform to expected normal behaviour" [8] and 'the patterns of the data that do not comply with the general expected behaviour' [39], respectively. As shown, the definitions closely resemble each other. Going forward, we will use Pimenetel's definition as our basis for anomaly detection and include methods from all three domains.

2.4.2 Types of anomalies

The nature of the desired anomalies is essential to consider when choosing a detection method. Anomalies can be broadly divided into three categories [8]; Point anomalies, contextual anomalies and collective anomalies.

Point anomalies

Point anomalies are the most straightforward category. They are single data instances that differ from the rest of the data. A boundary typically defines the normal data, and all individual data points outside of this boundary are considered anomalous.

Contextual anomalies

Contextual anomalies are data instances that deviate from the rest of the data instances in the same context. Contextual anomalies require access to contextual attributes that infer the context. In time-series data, time is one such contextual attribute that determines the position of an instance in the entire sequence. Anomalies in outdoor temperature is an example of contextual anomalies. A measurement of 20°C might be completely normal in the summer but anomalous in the winter. When dealing with contextual anomalies, examining the anomalies in various contexts is a significant aspect. Contexts are often very domain-specific, and expert knowledge is needed to formalize these contexts.

Collective anomalies

Collective anomalies are collections of several data instances that together deviate from the entire data set. Individual data instances in a collective anomaly might or might not be normal, but their collective occurrence is considered anomalous. Such types of anomalies can only occur in data sets where instances are related to each other.

2.4.3 Process and sensor anomalies

Differentiating from the previous sensor categories, process and sensor anomalies are specific to our problem. Process anomalies are anomalous behaviour directly linked to the electrolysis process. For example, the anode effect explained in 2.1.5, where the low concentration of alumina causes a current spike. Sensor anomalies are, in contrast, changes in a particular sensor value without any correlation to the process. Low battery, faulty connections and manual interference with the sensor could be potential sources of sensor anomalies. Both of these can be placed into the anomaly categories previously mentioned. Hydro already has a good understanding of process anomalies, and want a more in-depth focus to be applied to sensor anomalies if applicable.

2.4.4 Domains

Anomaly detection is used in several different domains. They can broadly be be categorized into six different categories according to Pimentel et al. [54]:

1. Electronic IT security
2. Healthcare informatics/medical diagnostics and monitoring
3. Industrial monitoring and damage detection
4. Image processing/video surveillance

5. Text mining
6. Sensor networks

For our problem description, item 3 and 6 are of particular relevance. This is because these domains encompass our problem description and should give us the most significant insight when exploring solutions. The mentioned domains are discussed in some more detail below.

Industrial monitoring and damage detection

Industrial machinery is exposed to heavy use and deteriorate at different rates. Detecting anomalous behaviour early can reduce the cost associated with repairing machinery and the cost of operation. Usually, high-end machinery is fitted with a multitude of sensors reporting on numerous parameters. These may, for example, be wattage, temperature and vibration.

Sensor networks

Sensor networks often consist of multiple low-cost sensors distributed in a network, with one or more central nodes gathering and forwarding sensor readings. As implied, the sensors and nodes have sensing abilities, processing and wireless capabilities. Anomaly detection can be used to find faulty readings or anomalies in the processes they monitor.

2.5 Machine Learning techniques

Determining the learning technique applicable to a data set is dependent on the availability of data labels. The data label associated with a data point identifies it as *normal* or *anomalous* behaviour. In almost every instance, it is easier to get access to unlabeled data as the cost associated with labelling data is high. It is usually a manual process requiring human expert domain knowledge. As the industry embraces Industry 4.0 [37], the availability and size of data sets increase, thus further increasing the manual labour needed to label a data set. Furthermore, detecting and labelling anomalies is a dynamic process. There is no guarantee that the set of possible anomalies are represented in the data, and new instances can occur that would need labelling. As with the IACM-sensor presented in section 2.3, the probability of sensor-specific anomalies are not known, and the set of possible anomalies may not be represented fully in the data set. Depending on the access to labelled data, anomaly detection methods can be divided into three categories:

2.5.1 Supervised anomaly detection

Supervised learning is a training paradigm that assumes a data label with each point in the data set, for both normal and anomalous classes. In a nutshell, they are trained by example. For each instance of data used in training, the algorithm learns to distinguish between the classes, learning a general model for each category. Given a set of new instances, the algorithm compares them to the previously learned classes and determines where it belongs. Chandola *et al.* [8] raises two major issues for supervised anomaly detection methods:

- The anomalous instances are few compared to the normal instances in the training data
- Obtaining accurate and representative labels, especially for the anomaly class is usually challenging

The imbalance in available anomalous classes compared to normal training data is called the bias-variance tradeoff [17]. Both items are relevant for the data set we obtained from the IACM-sensor.

2.5.2 Semi-supervised anomaly detection

Semi-supervised anomaly detection differs from supervised anomaly detection by assuming only normal data in the training data set. The algorithm will learn the behaviour of the normal class, and anomalies can be detected by how they deviate from the normal class. This form of classification is called one-class classification [67] and is more flexible compared to supervised techniques. The bias-variance tradeoff is not applicable to semi-supervised techniques, and there is no need for human expert knowledge for labelling anomalous classes.

2.5.3 Unsupervised anomaly detection

Unsupervised anomaly detection methods do not require any labels, thus being the most flexible learning technique of the three. Instead, it assumes normal instances are far more frequent in the data set and uses the intrinsic properties of the data [18] to give it an anomaly score. Distance-based and clustering methods are commonly used. Many semi-supervised anomaly detection methods can be used unsupervised by using a subset of the data set as training data. This assumes a low density of anomalies in the set, and that the model is robust for the anomalies present in the training set.

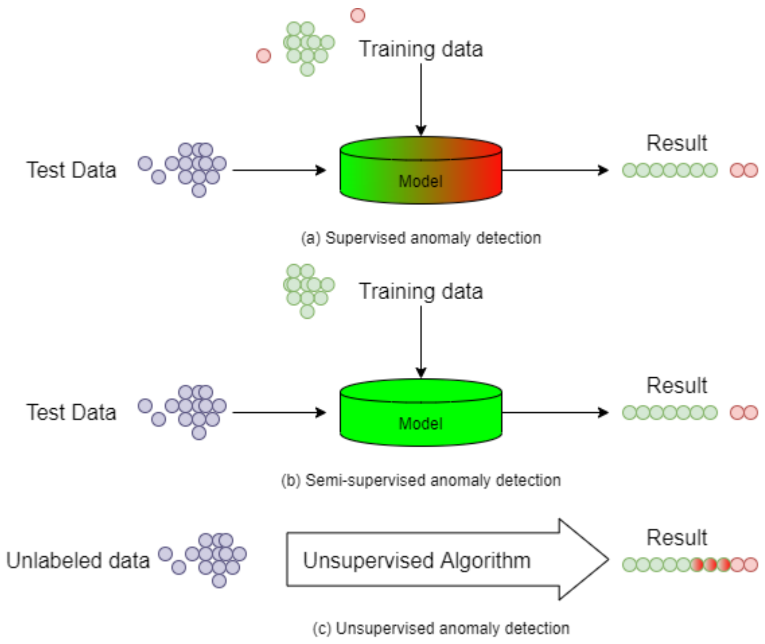


Figure 2.6: Three categories of machine learning techniques based on the availability of labels in the data set. (a) Training data contains labels for normal and anomalies. The model can detect both classes and the test data is correctly classified. (b) Training data is free of anomalies. After training, the difference between normal and anomalous behaviour is used to detect anomalies. (c) No labeled training or test set. Based on the intrinsic properties of the data, data points differentiating from the norm is classified as anomalies. However, the results are on a spectrum and not defined in classes. Adapted from [18].

2.6 Machine Learning Architectures

In this section we will present the different machine learning architectures used in anomaly detection. Each section contains:

- A brief overview of the architecture
- Advantages and disadvantages for the given architecture

2.6.1 Probabilistic

Probabilistic methods are mostly based on fitting a statistical model to the given data based on its statistical properties. One of the methods commonly used is estimating the probability density function [51]. Using statistical inference, one can then indicate if test samples belong to the model or not. This is based on the assumption that normal data occurs in high probability regions in the stochastic model, while anomalies occur with low probability regions.

Given this assumption, we can associate an anomaly threshold to the model of normality, usually estimated by the training set. Test instances that have a low chance of being generated by the learned model, and is over the anomaly threshold set, will be classified as anomalies. Both parametric and non-parametric techniques have been applied to probabilistic models. While parametric techniques make stringent assumptions about the nature of the population and the origin from where they were drawn, non-parametric techniques do generally not [63]. A more in-depth description of probabilistic methods used in anomaly detection is given in the literature [54][8][40].

The advantages of probabilistic methods are:

+ Provide an explainable and statistically justifiable solution, given that the assumption about the training data distribution remains true.
+ The anomaly score given by parametric statistical methods is often bound with a confidence interval, which can be integrated in a later decision making process.
+ Can operate in unsupervised mode without the need for labeled data, given that the distribution estimation is robust to anomalies in the data.
+ Encompasses a wide variety of methods with different computational complexity.

The disadvantages of probabilistic methods are:

- It is difficult to determine an anomaly threshold balancing the chance of false positives and false negatives.
- The assumption that data is generated according to a particular distribution is one of the major drawbacks of probabilistic methods. Higher likelihood of not being true in higher dimensional data sets.

2.6.2 Distance-based

Distance-based methods use a distance function to measure the distances between data points in feature space, where similar data lay in close proximity. For continuous data, Euclidean distance is a popular choice for this function. In this section, we will discuss two categories of distance-based methods, namely nearest-neighbour and clustering-based methods.

Nearest neighbor

Nearest neighbor based methods can broadly be divided into distance-based methods such as kth nearest neighbour, and local density-based methods, which considers the density of the neighbourhood around each data point.

Assumption: Normal data occurs in dense neighbourhoods, and anomalies occur far from its neighbours.

The K-nearest-neighbour (kNN) [53] algorithm uses the distances of the k closest points in feature space as input and calculates the anomaly score based on these distances. Fast

Outlier Detection in High Dimensional Spaces [5] proposes using the average distance of the nearest neighbours as the anomaly score. The distance to the k-nearest neighbours can be seen as the radius of a hypersphere centred at the data instance, containing k other instances. If the radius of this sphere increases, the density of data instances in the sphere decreases. The inverse of the distance to the k-nearest neighbour, therefore, represents an estimate of the local density of the neighbourhood. Local Outlier Factor (LOF) [7] is a technique computing the density of an instance relative to the densities of its neighbours. This technique is proposed to handle issues of varying densities in the data.

The advantages of nearest neighbour methods are:

+ They are unsupervised by nature, and are purely data driven.
+ Adapting the method to other domains is uncomplicated, and primarily requires an appropriate distance function. However, coming up with the distance function may be nontrivial.

The disadvantages of nearest neighbour methods are:

- If the normal instances in the data set do not have enough similar normal instances, the false positive rate will be substantial.
- The computational complexity of the testing phase is a significant challenge, because it involves calculating the distance of each test instance.

Clustering

Clustering-based methods group similar data instances into a small number of clusters. The centre of these clusters are points that characterize normal data. The distance from a data instance to its nearest cluster centre is often used to detect anomalies. These methods are primarily used for unsupervised learning. Clustering methods are based broadly on three different assumptions depending on which clustering algorithm used.

Assumption 1: Normal data belongs to a cluster, anomalies generally do not belong to any cluster.
Assumption 2: Normal data lies close to its nearest cluster centroid, while anomalies lies far away from it's closest cluster centroid.
Assumption 3: Normal data belongs to large dense clusters, while anomalies belong to small parse clusters.

The advantages of clustering methods are:

+ Clustering methods can operate in an unsupervised mode.
+ Generally low time complexity.

The disadvantages of clustering methods are:

- The performance is highly dependent on the clustering algorithm used.

- Several clustering algorithms try to force each instance into a cluster. These algorithms are therefore not optimal for anomaly detection, based on the assumption that the anomalies do not belong to any cluster.

A comprehensive survey done by Tian Y. and Xu D. is referred to for further reading [72].

2.6.3 Neural network-based approaches

Artificial Neural Network

Artificial Neural Network (ANN) is a computing system [26] that is inspired by the biological neural network that constitutes the brain of animals. In general, ANN can be characterized as a graph built on units called artificial neurons shown in Figure 2.7. The network consists of connections, where each connection provides the output of one neuron as the input to another neuron. Each connection has an assigned weight that represents the strength of influence of that particular input. Equation 2.6 shows how the neuron calculates the output from its inputs.

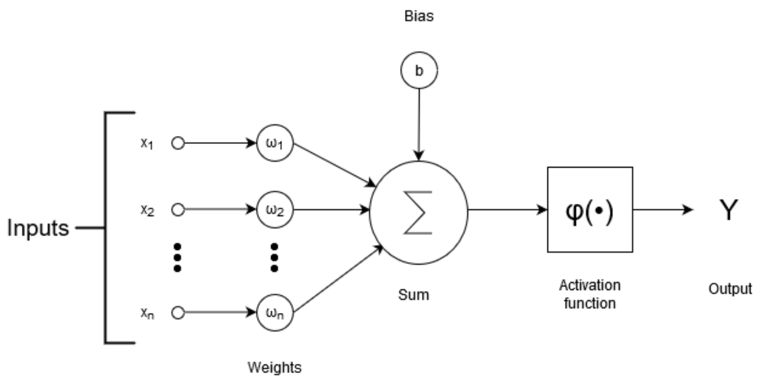


Figure 2.7: An artificial neuron. x_i represents the input values, and ω_i their respective weight values. The bias b is added to the weighted sum, and a non-linear activation function φ is applied on this sum to archive the neurons output.

$$y = f(\sum_i w_i x_i + b) \tag{2.6}$$

The neurons are typically organized in layers, where each neuron is connected to all neurons in its neighbouring layers as shown in Figure 2.8. The network works as a function that maps input to output.

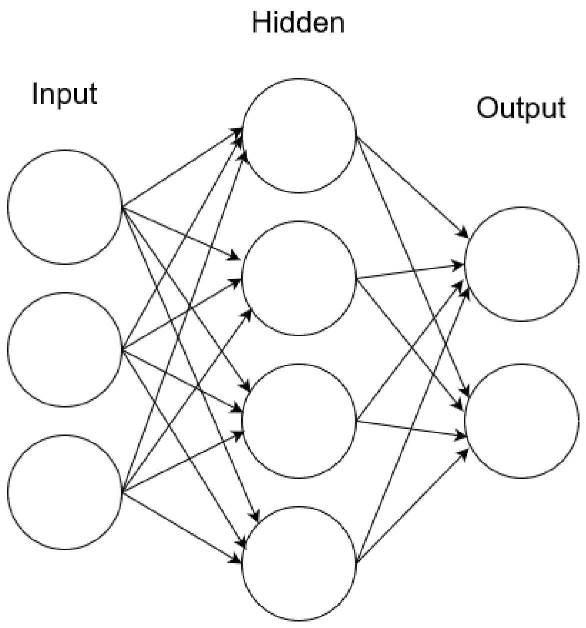


Figure 2.8: Fully connected Feed Forward Artificial Neural Network. This figure shows how each neuron has a forward connection to all the neurons in the succeeding layer.

The learning is the adaptation of the network to better handle a task by considering sample observations. It involves adjusting the parameters to increase the accuracy of the result. A random subset from the test data set is used for predictions, and a loss metric is calculated between the predicted and the expected result. The weights and biases, which are collectively called parameters, are then updated to minimize this loss metric. The most common way of doing this is a method called gradient descent which uses the gradient of the loss to update the parameters. The parameters are iteratively updated by moving in the steepest descent of the loss function, defined as the negative of the gradient. The slope of the steepest descent on the loss plane is visualized in figure 2.9. The parameters are then updated using the following equation:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta) \tag{2.7}$$

where θ is the parameters, α is a learning rate, and $\nabla_{\theta} J(\theta)$ is the gradient. The learning rate is a value that controls how much the parameters are updated with respect to the gradient for each iteration.

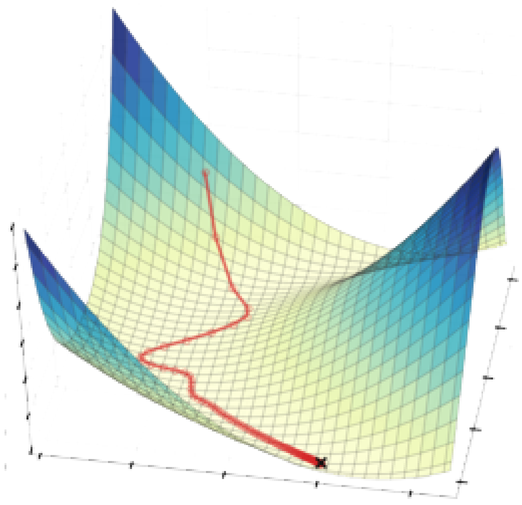


Figure 2.9: Gradient Slope. The surface here represents the loss with respect to the parameters. The red line follows the slope of the steepest descent of the gradient, and ends up in a local minimum [16].

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a modified version of the Feed Forward Artificial Neural Network (FFANN) shown in Figure 2.8, allowing feedback connections. The feedback connections enable the architecture to maintain information from previous iterations to persist, creating internal memory. Recurrent Neural Networks are best visualized in Figure 2.10 as multiple copies of the same network, each passing its output to the successor, resulting in a deep structure.

The deep structure and the internal state created from it allows RNNs to process sequences of inputs, making it suitable for time series. Unlike Feed-Forward Networks, all inputs are related to each other.

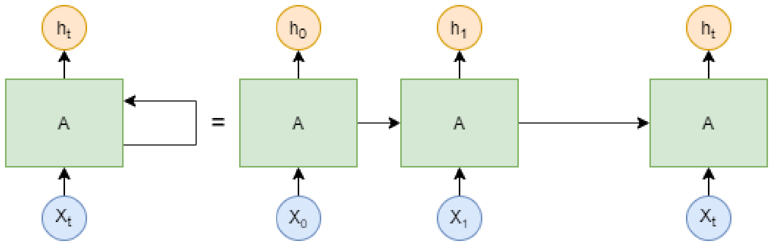


Figure 2.10: Recurrent Neural Network Unrolled. On the right hand side you can see how the rolled out representation of the network looks. Here the recurrent network is visualized as a chain of identical feed forward networks, one for each input. Adapted from [13]

Long Short Time Memory Networks

The Long Short Time Memory (LSTM) method was proposed by Hochreiter et al. in 1997 to deal with the vanishing gradient problems in RNNs [25]. In the standard RNN, the input is propagated through the recurrent connections over time. The same recurrent weight is multiplied several times to calculate the error. If this weight is small, the gradient will exponentially decay, and converge to zero. The further you go back, the lower the gradient will be, and thus harder to train. The RNN, therefore, struggles at learning long time dependencies.

To combat this, LSTM deploys several neural networks acting as *gates*, effectively determining how the internal cell state and hidden state are updated. The cell state is best explained as the long term internal memory of the LSTM cell, while the hidden state acts as the output and gate controller. Both states are updated each iteration through the *input*, *forget* and *update* gate, small neural network using the sigmoid activation function. The sigmoid activation function produces a value between 0 and 1. A value of zero means nothing will be let through, and a value of one means everything will be let through. The hyperbolic tangent used before combining the cell state with the hidden state outputs a value between -1 and 1, adjusting the cell state.

Figure 2.11 show the internal workings of a LSTM cell with the forget gate f, the input gate *i* and the output gate o. Additionally, they have their own weights and biases, denoted by W_f, W_i, W_o and b_f, b_i, b_o. The input is the previous cell state C_{t-1}, previous hidden state h_{t-1} and the current input x_t. W_c and b_c donates the weight and bias for updating the cell state. The three equations below show the forget, input and output gate, with σ denoting the sigmoid activation function.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{2.8}$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.9}$$

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{2.10}$$

The internal state is updated with the calculated candidate values and the hyperbolic tangent function, expressed by $\tanh$ and shown in Equation 2.11. $\tilde{C}$ are the new candidate values that should be added to the state.

$$\tilde{C} = \tanh(W_c * [h_{t-1}, x_t] + b_c) \tag{2.11}$$

The final output $h_t$will be based on the updated cell state C_t pointwise multiplied with the output gate:

$$C_t = f_t * C_{t-1} + i_t * C_t \tag{2.12}$$

$$h_t = o_t * \tanh(C_t) \tag{2.13}$$

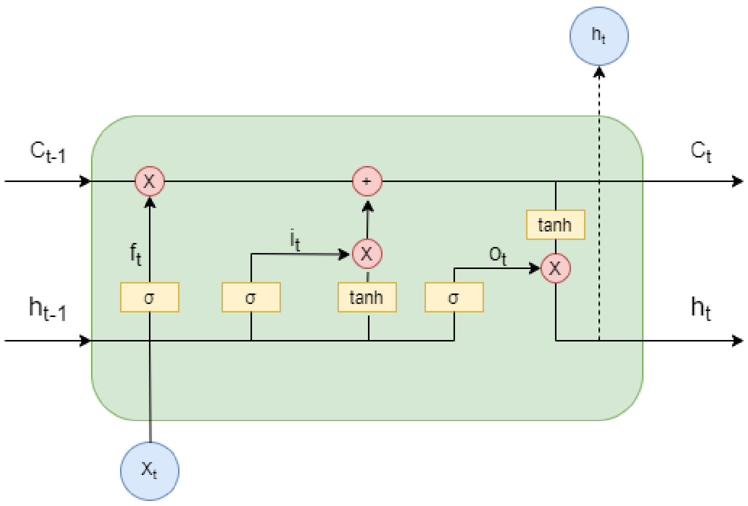


Figure 2.11: A single LSTM cell. The previous cell state(C_{t-1}) is first multiplied by the forget gate(f_t), deciding what information to retain and what to remove. The input gate(i_t) is then multiplied with the tanh activation of the current input(x_t) and the previous output(h_{t-1}), and added to the cell state, adding new information in the cell state. Finally the output gate(o_t) is multiplied with the cell state, deciding what to output from the current cell state. The current cell state(C_t) and the output(h_t) are used in the next time step. Adapted from [13].

Gated Recurrent Unit

Gated Recurrent Unit (GRU) was introduced in 2014 by Kyunghyun Cho el al. [10]. It is similar to LSTM, but has fewer parameters, as it has one less gate. GRU has a reset gate and an update gate. The update gate acts similar to the forget and input gates of LSTM. It decides what to throw away and what to add to the hidden state. The reset gate decides on how much of the past information to from the hidden state is added to the current input. As it has fewer parameters than LSTM, it is more computationally efficient. The LSTM allows disabling of writing to a cell, by turning off the input gate to prevent changes over many iterations. It means that longer temporal dependencies can be learned [31]. Equation 2.14 shows the update gate, and Equation 2.15 shows how the output of the reset gate is calculated. The logic of the gates is similar to the gates in LSTM. The calculation of the candidate activation vector and the new hidden state is shown in Equation 2.16 and Equation 2.17.

$$z_t = \sigma(W_z * [h_{t-1}, x_t] + b_z) \tag{2.14}$$

$$r_t = \sigma(W_r * [h_{t-1}, x_t] + b_r) \tag{2.15}$$

$$\tilde{h}_t = \tanh(W_h * [r_t * h_{t-1}, x_t] + b_h) \tag{2.16}$$

$$h_t = (1 - z_t) * h_{t-1} + \tilde{h}_t * z_t \tag{2.17}$$

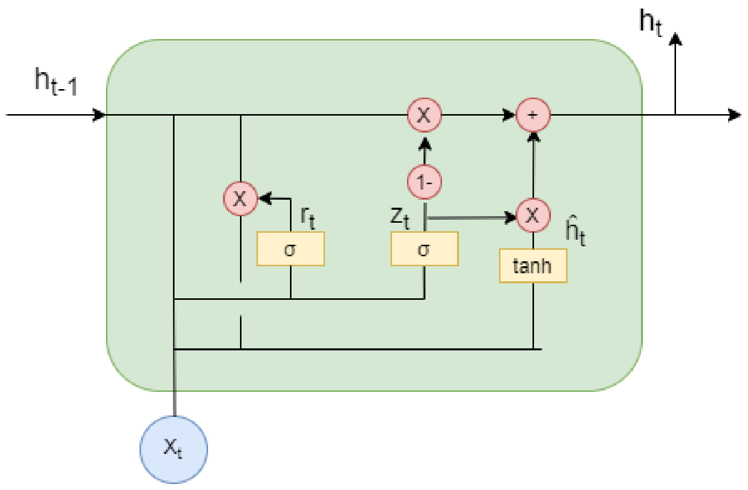


Figure 2.12: A single GRU cell with two gates, the reset gate (r_t) and the update gate (z_t). It combines the forget and input gates into a single update gate. Adapted from [13]

The advantages of neural network methods are:

+ High availability of open source libraries.
+ The ability to work with inadequate knowledge. The loss of performance depends on the importance of the missing information.
+ Storing information on the entire network. The disappearance of some information in one place does not prevent the network from functioning.
+ Parallel processing capability. ANNs can perform more than one job at the same time.

The disadvantages of neural network methods are:

- It is often difficult to understand why or how the network produced the output.
- Appropriate network structure is achieved though experience and trial and error. They can be hard to tune and debug.
- They do not perform as well on small data sets.
- Extensive training time is required for deep networks and large data set.

2.6.4 Domain-based

Domain-based methods require a boundary to be created, separating the target class(es). Inherently, they describe the domain's boundary and are usually impervious to the sample size of the target class or its density in the feature space. This is because domain-based methods evaluate unknown data points against the distance from the boundary, and not the class itself. The set of points from the training set that make up the distinguishing class boundary are called support vectors. All other data points from the training set are not evaluated when setting the boundary.

Support Vector Machines (SVM)

The SVM is a supervised, domain-based technique that works by assuming that there is some unknown and nonlinear dependency between some high-dimensional input vector and the scalar output [69]. Given a set of training examples, each from two distinct classes, SVMs performs *distribution-free learning* [12] to create a non-probabilistic linear classifier as shown in Figure 2.13. SVM incorporate the use of hyperplanes to maximize the margin between the two classes and improve accuracy. Maximizing the margin of two defined hyperplanes amounts to minimizing the normal vector $\overrightarrow{w}$ to the hyperplanes because the distance between the hyperplanes can be written as $\frac{2}{\|\overrightarrow{w}\|}$. The problem can be defined as in 2.18, with x_i being the input sample, and y_i the label(-1 or 1) for the training samples.

$$min \, \|\overrightarrow{w}\| \, (y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} - b) \geq 1, i = 1, 2, ..., n) \tag{2.18}$$

One crucial ingredient of SVMs is the kernel trick. The kernel trick is introduced to transform the feature space of the original problem in some way specified by a kernel function. By allowing the transformation defined by the kernel function to be nonlinear and the transformed feature space to be of a high dimension, the classifier may be found to be a hyperplane in the transformed feature space even though it may be nonlinear in the original input space. The kernel trick allows you to operate in the feature space without calculating the coordinates for a higher dimensional space, saving computational cost.

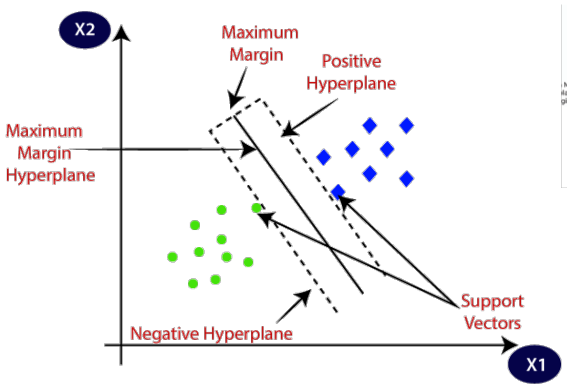


Figure 2.13: Illustration of a solution space with linear separation of two classes, maximum, negative and positive hyper plane and support vectors.

One Class Support Vector Machines (OCSVM)

OCSVM differentiate from standards SVMs by being trained on data only belonging to one class, i.e. normal data in the case of anomaly detection. The OCSVM maps the data into feature space and tries to describe the data in the feature space by using a hypersphere. The goal is to put most of the data into this hypersphere, leaving only the outliers outside. You want the hypersphere to be as small as possible, while at the same time including most of the training data. This can be formulated into an optimization problem [75]. The trade-off between the size of of the hypersphere and the number of training samples it can hold is tuned with the ν parameter. A small ν results in a larger hypersphere, and a significant ν results in a smaller hypersphere. The classification is performed on a query sample based on which side of the hypersphere the sample is located after being mapped to the same feature space as the training samples.

This is useful in anomaly detection as labelling anomalies is time consuming, complicated and requires expert knowledge. This allows OCSVM to be run in a semi-supervised fashion.

The advantages of domain-based methods are:

+ Good generalization without the need for a big data set.
+ Flexible: Can handle supervised and semi-supervised. Both linear and non-linear classification.

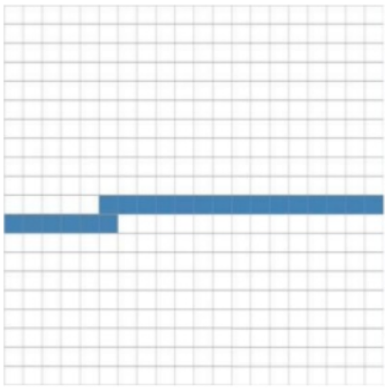
The disadvantages of domain-based methods are:

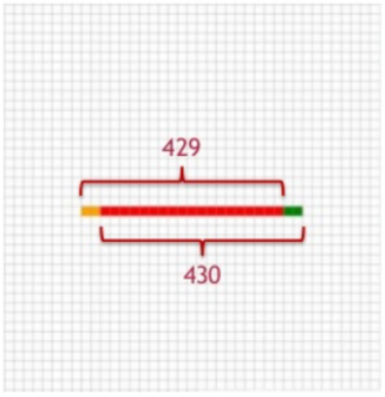
- Kernel functions can be computationally expensive.
- Parameterization in some SVM-based techniques can be difficult and severely impact performance.
- Sensitive to skewed data, i.e unbalanced data set [65]

2.6.5 Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) is a model of intelligence based on principles of neuroscience, and the interaction of pyramidal neurons in the neocortex of the human brain. It is a streaming algorithm that builds a predictive model of the world. Every time it receives an input, it attempts to predict what is going to be the next input. As it does this, it continuously updates the parameters of the model to improve future predictions. The models' forecast and the actual value is then used to calculate the anomaly score at that specific time step.

For the sequence learning part of HTM to be able to process the data, the input is encoded into a Sparse Distributed Representation (SDR). The SDR is a data sample represented as a bit array of zeros and ones. It is called sparse, because only around 2% of the bits are ones, and the rest are zeros. The SDR is encoded in two steps. First, the data is encoded into an array of all zeros, except a continuous series of ones, as explained in Figure 2.14a. It's vital that the semantic meaning of the data is preserved after this transformation so that similar data have a similar encoding in the input space. Figure 2.14b shows an example of how two similar numbers have overlapping bits in their encoded arrays.


(a) Scalar Encoder



(b) Similar overlap

Figure 2.14: Visualization of encoded bit arrays. Figure 2.14a shows the a bit array consisting of zeros except a series of consecutive ones, which represent a value after being encoded by the scalar encoder. Figure 2.14b shows how the numbers 429 and 430 have a similar encoding, where the red cells are their overlapping bits.

The second step is transforming this encoded array into a sparse representation of itself. The SDR is a three-dimensional structure, where two of the dimensions are used to describe the nature of the input, and the third dimension is used to represent the context of the input. The goal of spatial pooling is maintaining a small, fixed amount of sparsity, so that every SDR have the same amount of active columns, while still maintaining the overlapping properties from the input space. Each column in the ”Spatial pooler” has a set of connections to the input space. These connections are bound to specific positions in the input space and will trigger if the input has a ”one” on the position of the connection. The columns with the highest amount of connections that overlap a specific input will be active columns for that specific SDR. The connections updates run time, so new connections are learned, and old connections are forgotten. The spatial pooler reduces a large number of possible inputs to a manageable number of known coincidences. Figure 2.15 shows the connections of one single column to the input space.

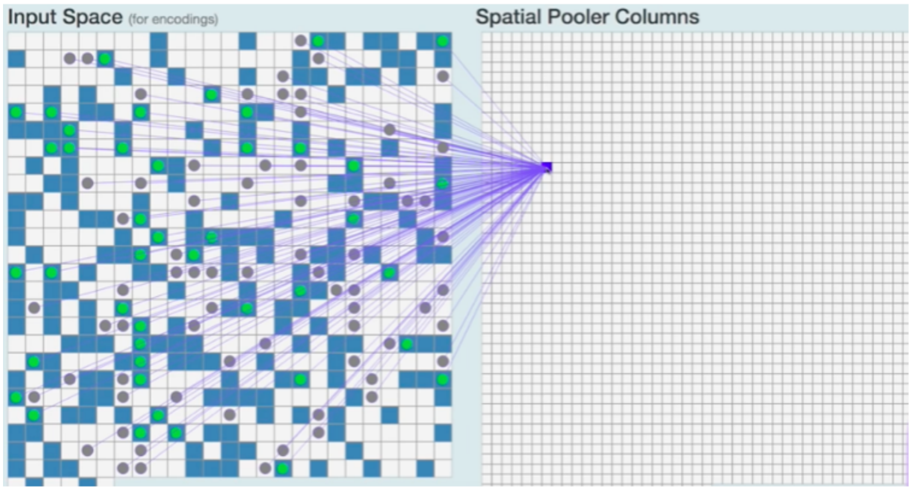


Figure 2.15: Example spatial pooler connections from one single column to the input space. The blue squares in the input space represents ones, and the white square represents zeros. The circles represent the connections. The overlap score of this column is 42, since 42 of the connections overlap with the specific input. Overlapping connections are colored green. [47]

HTM uses a ”temporal pooler” algorithm to learn the sequences of the input SDRs over time and predict what pattern is coming next. It does this by activating individual cells in the minicolumns of an SDR, based on the series of SDRs it previously received. The context of the input is stored in the cells of each column in the SDR. Two identical inputs will have the same active columns in its SDR, but might different active cells within the column depending on its context.

The temporal pooler algorithm forms weighted connections in the SDR structure. A cell within the SDR can form connections with any other cell. The connections are formed between the cells in the SDR that tend to be active during previous time steps. These connections are used to predict what cells in which columns are going to be active in the next time step. If a predicted cell were correctly predicted, the weights of the connections making this prediction would be slightly increased. If a predicted cell were wrongly predicted, the connections to this cell would be decayed.

The advantages of HTM are:

+ General purpose, can be used for a wide range of data.
+ Can be used with both stored and real-time data.
+ Works well for global outlier and level change anomalies.

The disadvantages of HTM are:

- May require some preprocessing and/or configuration for different data sets.
- Can be slow for batch processing of large data sets.
- Can detect anomalies for only one data set at a time [65].

Chapter 3

Related work

To create an overview of the state-of-the-art in sensor anomaly detection, we conducted a hybrid *systematic literature review* (SLR) approach. By this, we mean we used SLR to conduct an initial screening of the research domain and thus produce a set of relevant articles. In addition, a more traditional literature search was performed, hoping to negate some of the search term bias found in SLR. A manual literature search, including relevant conferences and reference crawling, was conducted in parallel to expand the search. Table 3.5 lists the articles found with both approaches.

The SLR was performed once at the start of the project, while the traditional approach was an iterative process taking place throughout the master thesis. We will give an introduction to SLR in this chapter, propose architectural requirements and an in-depth look at related work.

3.1 Systematic Literature Review

A systematic literature review is structured and well-defined way of identifying, evaluating and interpretation primary studies in relation to a set of research questions. The process consists of 3 main phases [33]: *i*) planning the review, *ii*) conducting the review and *iii*) reporting the review. Stage *i* and *ii* will be described in greater detail below. Part *iii* is not relevant for this exercise. Kitchenham et al. [33] emphasizes that traditional literature reviews are of little scientific value due to the difficulty of reproducing the results, bias interference and duplication of work. Adhering to SLRs strict framework of well-defined steps carried out in accordance with a pre-defined protocol enables reproducibility of the results. Carried out correctly, it should alleviate the disadvantages described above and gain scientific value.

3.1.1 Planning the review

The planning phase consists of five steps:

1. Identification of the need for a review
2. Commissioning a review
3. Specifying the research question(s)
4. Developing a review protocol
5. Evaluating the review protocol

For the sake of this master thesis step 1 and step 2 should be self-explanatory and are not mandatory in conducting a SLR. We will go into greater detail for step 3-5.

Specifying the research question(s)

Formulating the research questions is the most important part of any systematic review. They should be formulated in a fashion that discourages yes/no answers and be specific enough that they target the problems presented in the project. Given these constraints we compiled 3 literature research questions:

- What are the existing machine learning solutions to finding sensor anomalies in time series data?
- How do these different ML models affect accuracy, training time and explainability in relations to finding sensor anomalies?
- What implications do these solutions have in regards to detecting and categorizing sensor anomalies?

3.1.2 Conducting the review

Conducting the review consists of 5 steps, where we combined step 4 and 5 into an analysis step:

- Identification of research
- Selection of primary studies
- Study quality assessment
- Data analyzation and comparison

  1. Data extraction and monitoring
  2. Data synthesis

Step 1: Identification of research

The objective of this step is to find as many [33] or all [34] the primary studies relating to the stated research questions using a two step search strategy. This strategy consists of first defining *which sources* to be used in the literature search and *how to* search them e.g. which search terms to use.

(A) Defining which sources to be used was determined by comparing different search engines, databases, journals and conferences. Individually searching journals and conferences can be exhausting due to the time involved, difficulty mapping relevancy and how spread the information is. We, therefore, focused on finding scientific search engines and databases with wide coverage for the computer science field.

Based on these criteria and the limitations mentioned, we ended up with three main sources described in Table 3.1.

Table 3.1: Search engines and digital libraries used in SLR

Related work sources		
Name	Type	Website
Google Scholar	Search Engine	https://scholar.google.com/
ACM Digital Library	Digital Library	https://dl.acm.org/
IEEE Xplore	Digital Library	https://ieeexplore.ieee.org/

(B) Determining how to search the sources was done by building a search string from a set of key terms. This was done by grouping key terms together. Each key term group consists of terms that are either synonyms, different forms of the same word or have close semantic meaning [34]. By applying the AND($\wedge$) and OR($\vee$) operator on Table 3.2, where the OR operator is used within a group, and the AND is used to combine groups, we can compile the following search string:

$$([G1, T1] \vee [G1, T2] \vee [G1, T3]) \wedge ([G2, T1]) \wedge ([G3, T1] \vee [G3, T2])$$
$$\wedge([G4, T1] \vee [G4, T2]) \wedge ([G5, T1])$$

Table 3.2: Key terms used in literature search

	Group 1	Group 2	Group 3	Group 4	Group 5
Term 1	Novelty	Detection	Machine Learning	Sensor	Time series data
Term 2	Anomaly		Neural network	Sensor network	
Term 3	Outlier				

Step 2: Selection of primary studies

The search strategy returns a sizeable set of relevant articles, far too many to be manageable. The goal of the selection phase it to return a smaller set of relevant articles.

To broadly narrow it down, a few criteria was developed and used during the selection process:

- Duplicates (Keep the highest ranking/most cited)
- Same study published in different sources
- Studies published before 01.01.1990

In addition to the three general items listed above, a set of screening questions was used to determine if the article was relevant. These are listed in Table 3.3

Table 3.3: Title, abstract and conclusion screening questions used in the screening process of selecting primary studies

Screening domain and question	**Yes - include**	**No - exclude**	**Unclear - include**
Research design: Does the title/abstract describe an anomaly detection primary research study?	Yes. The study shows clear signs of direct data collection.	No. The study depends on data collected in previously done research	It is unclear if the study design is primary research from the abstract or title.
Publication type: Does the title and abstract come from a published study, conference study, scientific publications, government study or non-government study?	Yes.	No.	Unclear: It is unclear if it is a published study, conference study, scientific publication, government study or non government study from the title and abstract.
Outcome: Does the abstract and/or conclusion describe a solution to the given problem i.e an architecture or model?	Yes: The abstract and/or conclusion presents a solution to the given problem	No: The abstract and/or conclusion does not include any mentions of methods used to solve the initial problem.	Unclear: It is unclear if a solutions is presented in the title or abstract.

Table 3.4: Inclusion and quality criteria used in quality assessment. Taken from [34] and modified.

Criteria identification	**Criteria**
IC 1	The study's main concern is anomaly detection
IC 2	The study is a primary study presenting empirical results
IC 3	The study focuses on or describes some form of accuracy, training time complexity or explainability of the model
IC 4	The study describes the implications of implementing the solutions in regard to detection ability and/or categorization of anomalies.
QC 1	There is a clear statement of the aim of the research
QC 2	The study is put into context of other studies and research

Step 3: Study quality assessment

The purpose of the quality assessment is to filter away articles not relevant to the chosen field and create a final set of articles. By using the inclusion (IC) and quality criteria (QC) listed in Table 3.4 we avoided personal prejudice and achieved objectivity during the quality assessment. Table 3.5 shows the selected articles after the quality assessment that in turn, will create the bases for related work.

Table 3.5: Final set of articles selected including an arbitrary ID, title, author(s) name(s) and published year

ID	Title	Author(s)	Published year
ID001	Adaptive fuzzy clustering based anomaly data detection in energy system of steel industry	Zhao J. et al. [76]	2013
ID002	Anomaly detection in aircraft data using recurrent neural networks (RNN)	Nanduri A. and Sherry, L. [46]	2016
ID003	Detection and Characterization of Anomalies in Multivariate Time Series	Cheng H. et al. [9]	2009
ID004	Symbolic time series analysis of ultrasonic data for early detection of fatigue damage	Gupta S. et al. [23]	2005
ID005	Anomaly Detection Based on Sensor Data in Petroleum Industry Applications	Martí L. et al. [41]	2015
ID006	Fault diagnosis of ball bearings using machine learning methods	Kankar P. et al. [28]	2011
ID007	Gear fault detection using artificial neural networks and support vector machines with genetic algorithms	Samanta B. [60]	2004
ID008	dLSTM: a new approach for anomaly detection using deep learning with delayed prediction	Maya S. et al. [43]	2019
ID009	Toward Supervised Anomaly Detection	Görnitz N. et al. [20]	2013
ID010	LSTM-based encoder-decoder for multi-sensor anomaly detection	Malhotra P. et al. [38]	2016
ID011	Centered Hyperspherical and Hyperellipsoidal One-Class Support Vector Machines for Anomaly Detection in Sensor Networks	Rajasegarar S. et al. [57]	2010
ID012	Unsupervised real-time anomaly detection for streaming data	Ahmad S. et al. [3]	2017

Step 4: Data analyzation and comparison

The final step consists of analyzing the last set of articles in regards to the research questions specified in 3.1.1. The architectures and models presented in the papers will be evaluated and compared. In addition, we will assess how they can be used to solve the initial problem of anomaly detection in IACM-sensors, eventually ranking them based on our specified demands for a solution.

3.1.3 Proposing an architecture

Based on the findings from the structured literature review, we tailored a set of architectural requirements for the anomaly detection in time-series data of the IACM-sensor. The requirements are based on what we saw necessary to achieve a high detection rate based on our problem. All architectures presented was evaluated against the set of architectural requirements and presented in Table 3.5.

- AR1: The architecture for IACM anomaly detection should be able to perform well in unsupervised or semi-supervised mode; i.e. without the need for labelled data. At most, a set of normal data.
- AR2: The architecture for IACM anomaly detection should be suitable for real-time detection.
- AR3: The architecture for IACM anomaly detection should be able to handle a large, continuous, time-series data set, i.e. one cell consisting of 40 sensors operating at 1 Hz for around 30 days.
- AR4: The architecture for IACM anomaly detection should be feasible to implement and test for a team of two students for a Master thesis.

Dynamic time warping(DTW) combined with KNN-AFCM

Zhao J. et al. presents a method involving the use of dynamic time warping combined with k-nearest neighbour adaptive fuzzy C means (KNN-AFCM) to find and detect both trend anomalies in pseudo-periodic data and deviant anomalies in general data [76] generated in the energy system of a steel plant. As the same operation is performed in succession with variable time length, a DTW based sequence stretching method is introduced to transform variable-length time series into equal length given the euclidean distance. This allows calculating the degree of similarity and direct comparison between pseudo-periodic data. AFCM can then be used to cluster and calculate the proposed anomaly index to detect anomaly data with a run time of O(n^2).
Furthermore, they present a new clustering method, called KNN-AFCM, in which the correlative information of the neighbouring points are introduced and combined into a new objective function for data clustering. The first part of the function aims to estimate the deviation between the points and the neighbours, the second affects the clustering result. KNN-AFCM showed an improved capability of detecting local deviations. Computational complexity is similar to AFCM [76] and mildly suitable for real-time detection.

Recurrent Neural Networks(RNN)

Nanduri A. and Sherry L. describes the application of Recurrent Neural Networks(RNN) with Long Term Short Term(LSTM) and Gated Recurrent Units(GRU) [46]. The algorithm was used for anomaly detection in multivariate, time-series data collected from the aircraft's Flight Data Recorder (FDA), without the need for dimensional reduction, and shows robust detection in latent features. As with KNN-AFCM, RNNS can also be implemented for real-time detection by accepting sequences of multivariate data into a trained model.
The data set was artificially generated in the X-Plane flight simulator and artificially injected with a set of 11 known anomalies. Comparing it to a Multiple Kernel Anomaly Detection (MKAD) algorithm based on One-Class Support Vector Machine (SVM) [15], MKAD detected 6/11 anomalies while RNN with LSTM or GRU managed 8/11.

Graph based algorithms

To address several difficulties such as (1) finding a concise definition for an anomaly, (2) unexpected changes in relationships among a set of variables in multivariate time-series data and (3) presence of noise in one or more series, Cheng H. et al. created a robust graph-based algorithm for detecting anomalies in multivariate time series data [9]. This is archived by employing a kernel matrix alignment method to learn the dependence relations among variables, and thus reduces noise and retains anomalies present in the target time series that is dependent on observations in other time series. The anomalies are detected by performing a random walk traversal on the graph constructed from the aligned kernel matrix framework.
Comparing it to other multivariate time-series anomaly detection algorithms such as random walk without alignment [44], probability-based [73], LOF [7], and K-distance based methods [58] as introduced in 3.1.3, the graph-based algorithm, named Align, outperforms them on a lions share of the training data sets tested [9].

Symbolic time series analysis(STSA)

In the case of detecting fatigue damage in polycrystalline alloys, specifically 7075-T6 aluminium, Gupta S. et al. [23] presents a novel analytical tool for early detection. They address the need for a solution that (1) take into account the initial defects in the materials, (2) has the ability to issue early warnings and estimate fatigue damage due to fatigue damages stresses stochastic nature and (3) real-time time-series analysis from mounted sensors for continuous structural health monitoring. The analytical tool mentioned consists of wavelet-based(WB) partitioning, followed by the extraction of pertinent information from the time-series data in probability distributions. This is in turn used for symbol sequence and symbol generation and used in a symbolic time series analysis [56].

Compared to excising pattern recognition tools such as multilayer perceptron neural network (MLPNN), principal component analysis(PCA) and radial basis function network (RBN), the STSA was able to detect an anomaly in fatigue damage at around 40- 44 kilocycles. MLPNN and PCA were able to detect the anomaly in the same range, but STSA yielded the greatest change in the anomaly growth rate and the earliest detection. As concluded by the authors, the striking feature of STSA with WB partitioning is the ability to take advantage of the vector information embedded in the histograms of probability distributions, to detect minor changes.

One-class support vector machine (OCSVM)

The domain of petroleum industry applications depends on correct sensor information to properly utilize machinery and ensure safe operation of pumps etc. An anomaly detection method needs to both be able to detect performance-related anomalies, in addition to sensor degradation anomalies. Mart L. et al introduces an architecture combining yet another segmentation algorithm [42] (YASA) with a OCSVM [41]. As with industrial systems of a large scale, a magnitude of sensors are involved. Mart L. et al. estimates a total of 1×10^{12} measurements per year, restricting computation complexity in order to support real-time detection and monitoring.

Furthermore, to deal with the amount of noisy data, time-series segmentation is used in the preprocessing stage to identify homogeneous data that can be analyzed separately, allowing OCSVM to focus on the most relevant parts of the time series. Comparing YASA to other segmentation algorithms such as bottom-up, top-down, adaptive top-down and sliding window and bottom-up shows it has a clear advantage in the time needed for the CPU to finish running the algorithm and the lowest RMSE tested on four different time series. The improved performance of YASA is credited to simple parametrization and usability. Usually, most other methods require having a priory number of segments as input, significantly reducing correct parameterization at the cost of negative bias on the outcome of the method [41].

Another approach building upon SVM for detecting sensor failures in a sensor network is presented by Ragasegarar S. et al. [57]. Two approaches for anomaly detection in sensor networks are presented: (1) centred hyperellipsoidal vector machine (CESVM) and (2) one-class quarter-sphere support vector machine (QSSVM). CESVM is based on a linear programming-based hyperellipsoidal formulation, showing advantages in parameter selection flexibility and computational complexity, but has limited scope for implementation in sensor networks. QSSM, on the other hand, works by capturing the normal data vectors in a higher dimensional space for each sensor, resulting in a hypersphere. The hyperspheres summary information is propagated inwards to a central global hypersphere, which is used to detect anomalous sensor readings.

The centralized scheme run time of the respective algorithms are $O(s^2n^2)$ for QSSVM and $O(n^2 + m^2n)$ for CESVM where n is the number of data vectors at a node, s is the number of sensors and m is the low rank approximation of the Gram matrix. The work shows that CESVM can generally attain a higher or comparable accuracy compared to QSSV in the four datasets tested. Nevertheless, it is a trade-off between communication overhead, computational complexity and detection accuracy in the end.

Artificial neural networks(ANN)

Using the time-domain vibration signals of a rotating machine with standard and defective gears, Samanta B. compares the use of ANN to SVM with and without using a genetic algorithm (GA) for hidden layer size, SVMs basis function kernel parameter and number of input features [60]. Without the use of a GA for parameter initialization, the standard ANN consisting of 24 neurons in the hidden layer and a constant width of 0.5 was used for the SVM, SVM scores significantly better at all 9 test cases with reduced training time. The same relationship was evident throughout, with or without feature selection and signal preprocessing.

However, with the use of GA-based selection, both classifiers were comparable at almost 100%. As evident from the results, parameter selection for ANN-based approaches plays a significant role in optimizing anomaly detection rate and minimizing error.

The same comparison has been made by Kankar P. et al. on fault diagnosis of ball bearings using ANN and SVMs [28], but without the use of a GA for parameterization. The difference between the two is the principle of risk minimization(RM) [22]. ANNs use traditional empirical risk minimization (ERM) to minimize the error on the training data, while SVMs employ structural risk minimization (SRM) to minimize an upper bound on the expected risk. This is supposed to make SVMs greater at generalization [60]. The results presented support the findings of Samanta B., reporting 71.2% correctly classified instances for the ANN and 73.9% for the SVM. Additionally, the SVM reported lower incorrectly classified cases and a higher RAE of 80.6% compared to ANNs 48.7%. However, it is worth noting that the availability of historical data has been limited in this case, and it was trained on a relatively small data set.

LSTM - EncDec-AD

Multiple sensor anomaly detection scenarios can be challenging due to unmonitored environmental conditions can affect sensor stability and health and produce unpredictable time-series. Standard anomaly detection techniques of mathematical models based on stationarity or predictive models based on prediction error are challenging to use in the specified scenario [38]. To solve this, Malhotra P. et al. propose an LSTM network-based encoder-decoder scheme for anomaly detection (EncDec-AD). EncDec-AD works by reconstructing typical time-series behaviour and compares the reconstruction error against the reported value to detect anomalies.

The authors tested EncDec-AD on four real-world datasets, concluding that the algorithm works well for detecting anomalies in both predictable and unpredictable time-series.

delayed Long Term-Short memory (dLSTM)

Maya S. et al. propose a new anomaly detection method called dLSTM [43] for time-series data to try and overcome the difficulty associated with sensor data, namely noise and multi-mode characteristics. The case of multi-mode is defined by that there are multiple outputs for the same input. These characteristics can make it difficult for one predictive model to properly encompass the difficulties and perform sufficiently. To avoid this, dLSTM operates by embedding multiple predictive models in the LTSTM and applying a gating function. The gating functions [45] purpose is to try and identify a predictive model for the input based on a collection of available models, the selection made possible by comparing prediction error. If successful, we can more flexibly predict future states and reduce prediction error [43]. Delayed prediction is incorporated and used for the selection phase, meaning the predicted value is delayed until the corresponding measured value is acquired, making use of the measured value to select the best-suited model.

The proposed method does well compared against other methods such as contractive autoencoders(CAE) [59], stacked autoencoder(SAE) [68] and variational autoencoder(VAE) [4]. By combining dLSTM with other predictive models for time-series data, it allows flexibility and increased robustness. However, LSTM requires sequential processing of time-series data when training to perform backpropagation and is thus time-consuming. Nevertheless, once a precise model is built, it can be used until an atypical event occurs.

Semi-supervised anomaly detection(SSAD)

Anomaly detection usually builds upon the unsupervised learning paradigm as anomalies stem from abnormal events and are highly difficult to classify beforehand. As a result, the predictive performance of these models suffers and fail to reach the required accuracy in some tasks. Grnitz N. et al. presents a novel semi-supervised algorithm grounded on the unsupervised learning paradigm [20] and allows for the incorporation of labelled data. They show that anomaly detection methods derived from a supervised technique are likely to miss out on novel and previously unseen classes of anomalies, as supervised techniques focus on classifying concept classes and not data characterization.

The proposed algorithm generalized the standard Support Vector Data Description (SVDD) [66] and is able to process both labelled and unlabeled data. This allows for the inclusion of prior and expert knowledge. By utilizing a small set of labelled data, the accuracy was significantly boosted. When tested in the domain of network intrusion, SSAD proved robust and beat SVM in execution time, proving useful in real-time scenarios.

Hierarchical Temporal Memory(HTM)

Contrasting the previous related work introduced, the use of HTM for anomaly detection in streaming time-series data presented by Ahmad S. et al. [3] poses a set of new challenges and solutions. HTM does not need a data set of previous measurements to train and establish a model but rather can work unsupervised on one or more stream for early detection. This is useful in cases where the sensor is new, or there is no access to previous measurements. In addition, the novel algorithm presented is capable of online prediction; i.e. determining if the state X_t is anomalous or not before accessing state X_{t+1}, continuous learning and is able to adapt in dynamic environments.

A modified version of HTM for anomaly detection using prediction error and anomaly likelihood outperformed the contest winners of IEEE WCCI NAB, an anomaly benchmark competition for real-time anomaly detection. Able to detect both spatial and temporal anomalies, as well as handle concept drift and is non-parametric.

3.1.4 Evaluating related work

We evaluated the architectures presented in related work against the set of architectural requirements from section 3.1.3. This was purely a subjective rating based on our understanding of the architecture and information presented in the paper. The scoring system is a trinary system, consisting of 0, 0.5 and 1. The score is based on these criteria:

- **0:** The architecture does not fulfil the architectural requirement(AR).
- **0.5:** The architecture does somewhat fulfil the AR.
- **1:** The architecture fulfils the AR.

In Table 3.6, a score of 0.5 was given for AR1 if the architecture worked in semi-supervised mode or normal data as a full score would be the architecture(s) best fitted for our initial data set.

ID	**Architecture**	**AR1**	**AR2**	**AR3**	**AR4**	**Total**
ID001	DTW with KNN-AFCM	1	0.5	0	0.5	2
ID002	**RNN with LSTM/GRU**	0.5	1	1	1	**3.5**
ID003	Align	1	0.5	0.5	0.5	2.5
ID004	STSA	1	1	0.5	0	2.5
ID005	**YASA with OCSVM**	0.5	1	1	1	**3.5**
ID011	CESVM	0.5	0.5	0.5	0.5	2
ID011	QVSSM	0.5	0.5	0.5	0.5	2
ID006 & ID007	ANN(s)	0	1	0.5	1	2.5
ID010	EncDec-AD	0.5	1	0.5	1	3
ID008	dLSTM	0.5	1	1	1	3.5
ID009	SSAD	1	0.5	0	0.5	2
ID012	**HTM**	1	1	1	1	**4**

Table 3.6: Final rating of previous work against the set of architectural requirements presented in section 3.1.3.

After performing the hybrid SLR we were left four highly ranked methods. Three were ultimately chosen based on method characteristics, evaluated data set, and how feasible it was to implement the method within the given time frame and for a master thesis. They are marked as bold in Table 3.6.

Chapter 4

Anomaly Detection and Separation Methods

Figure 4.1 introduces the system diagram for our anomaly detection implementation. This section will follow the diagram closely, firstly describing the data sets and frameworks used. It will then present the two baseline methods implemented, followed by the three methods implemented as a result of evaluating related work in Section 3.1.4. Lastly, it will introduce our anomaly distinguisher framework for separating process and sensor anomalies.

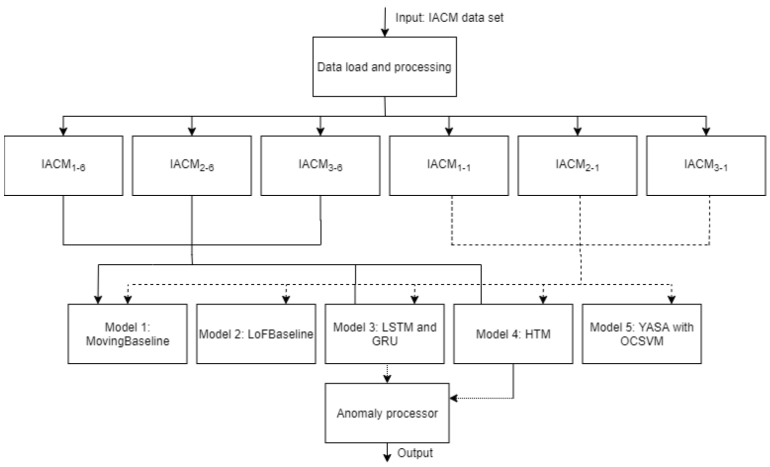


Figure 4.1: System diagram for our implementation. First, we process the data and separate it into several data sets. The black and dotted line indicates which model uses which data sets, as some architectures have limitation in how they process and handle multiple features. Results from YASA and OCSVM was not deemed good enough for further processing, as shown in the diagram and reasoned in Section 4.7

4.1 Frameworks

4.1.1 TensorFlow

TensorFlow [19] is an end-to-end open-source machine learning platform developed and operated by Google. It enables the user to create data-flow graphs, where each node in the graph represents a mathematical operation. The connecting edges is a multidimensional data array, called a tensor. Tensorflow is built around a C++ back-end and offers a plethora of language implementations acting as front-end APIs for creating, managing and running applications. Python was used in our implementations. Applications can be trained and run on both CPU and GPU, significantly increasing versatility and reducing training times through parallelization, or be run on Google's custom TensorFlow Processing Unit (TPU) for improved acceleration. In addition, it supports the use of training checkmarks where custom behaviour can be implemented, and fully customizable visualization options, making it a natural choice for ML research and training.

4.1.2 Keras

A disadvantage of using TensorFlow is the lines of code needed to create a running neural network model, and the amount of customization and individual implementation the user has to write or choose between. Keras [11] is an open-source neural-network library that solves these problems. Acting as an abstraction layer on top of TensorFlow, it enables users to use it as a high-level API. Developed by François Chollet, embracing core values such as ease of implementation, modularization and user-friendliness. It contains a wide variety of already implemented elements used in creating and training NN, such as layers, activation function and optimizers, enabling faster start-up times, fewer lines of code and less clutter when creating and running models on TensorFlow.

4.1.3 NuPIC

The NuPIC codebase [**?**] is a framework for developing and running HTM models, produced by Numenta. It consists of a network engine library, along with layered software including code examples and an anomaly detection framework. It is written in Python 2.7 and is particularly suited for anomaly detection and prediction of streaming data sources. HTM has several parameters to fine-tune, and NuPIC includes a swarming algorithm to find the correct parameters, to get a good model for a given data set [48]. By a good model, we mean a model that accurately produces the desired output.

4.1.4 Scikit-learn

Scikit-learn [2] is a free software machine learning library for python. It is designed to inter-operate with Python numerical and scientific libraries NumPy and SciPy and uses Numpy extensively for high-performance linear algebra and array operations. It features various classification, regression and clustering algorithms including a one-class support vector machine.

4.2 The IACM data set

The data set is over one year of historical data from 02.03.2016 to 30.03.2017. Section 2.2 describes the data notation we will use from this point on. The sensors are clocked at 1 hertz, performing one reading per second. All plots in this paper are normalized to preserve the sensitivity of the data.

Due to some of the models tested in this paper being computationally heavy, we decided to reduce the size of the data set when training our models. As mentioned in section 2.1.4, the life cycle of a carbon anode is approximately 22-30 days and is then replaced. We split the data set into three separate data sets with respectively 1, 2 and 3 months of historical data. This way, we capture both a single cycle and multiple cycles, so that the model potentially can adapt to seasonally repeated patterns. We also get to see the difference in the models' predictions from training on these three different time intervals. For each of these three intervals, we divide further into two different sets: (1) using one sensor and (2) using the neighbouring 5 sensors.

For all the data sets, anomalies are detected in Sensor 2. We chose Sensor 2 because it had five neighbours with a low amount of visible downtime. Figure 4.2 shows 3 months of data from sensor 2. Figure 4.3 shows how the neighbouring anodes are positioned in the cell.

When a carbon anode is replaced, the new anode is cold, so temperature around the anode will drop. The environment inside the cell is open, so this also affects the temperature and chemical process of the neighbouring anodes. The idea of using data from multiple sensors to detect anomalies for one single sensor is that giving the model additional information from the adjacent sensors, might result in better performance, and help to separate process and sensor anomalies. The separation of sensor and process anomalies will be explained in section 4.8.

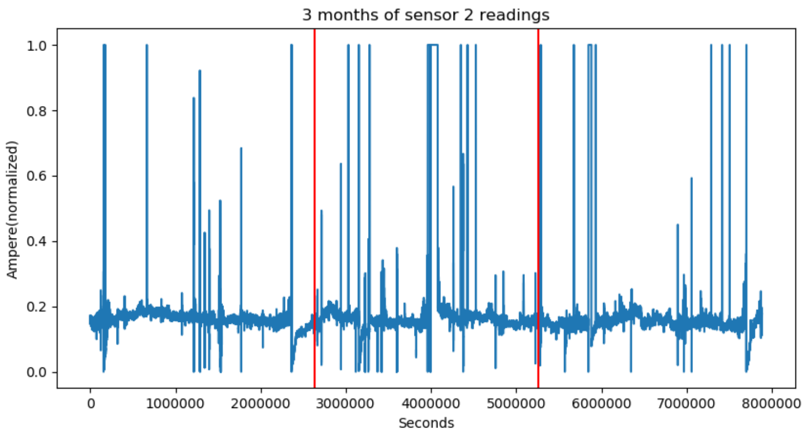


Figure 4.2: 3 Months of data from sensor 2. The red vertical lines show 1 and 2 months.

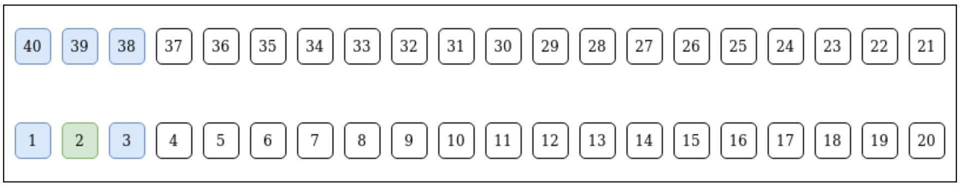


Figure 4.3: Anode positions. The squares represent the position of all 40 anodes in one cell. The blue anodes are the 5 neighbours of the green anode nr 2.

Data set name	Length	Number of sensor input
IACM_{1-1}	1 month	1
IACM_{2-1}	2 months	1
IACM_{3-1}	3 months	1
IACM_{1-6}	1 month	6
IACM_{2-6}	2 months	6
IACM_{3-6}	3 months	6

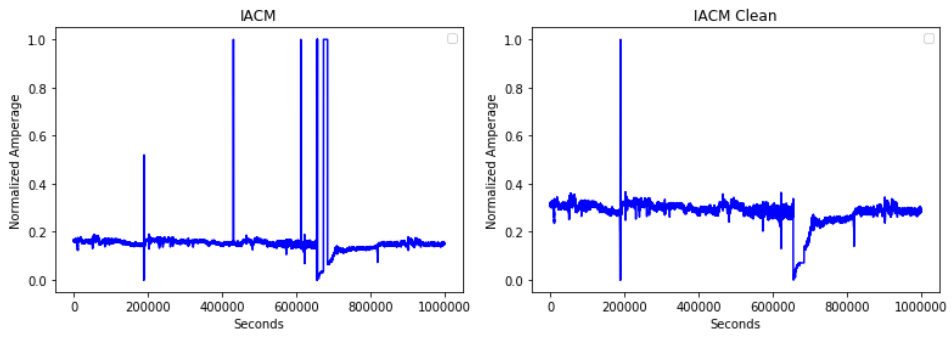
Table 4.1: The six different data sets used for training. The format for the name is IACM_{i-n}, where i is the time interval in months, and n is the number of sensor inputs.

The data set is unlabeled. That means the data examples all have an unknown classification. There are no available logs or records on when anomalous events occurred for this particular data set or if they occurred.

4.2.1 Data set cleaning

Data set cleaning is the removal of noise from the available sensor readings, whereby noise in a time series is classified as random fluctuations about the typical pattern. You can not reasonably model the noise and make predictions because of its inherently random nature. By removing noise from the time series, the model might have an easier time approximating the seasonal and trend components. In our case, what would look like noise could be sensor anomalies, which is something we want to detect. If possible, proceeding with a data cleaning method that still preserves these anomalies would be beneficial.

A domain expert from Hydro explained how the IACM sensor has several noisy faulty readings, and that every reading above X ampere can be safely discarded. The limit will not be mentioned in the paper given the sensitive nature. In the ”cleaned” data set, we, therefore, set every value that exceeds the given limit to the value of the previous time step. Figure 4.4 shows how a sample of the data set looks before and after cleaning. We will refer to the uncleaned data as IACM_{raw} and the cleaned data as $\text{IACM}_{cleaned}$.



(a) Before cleaning **(b)** After cleaning

Figure 4.4: Data before and after cleaning. The left figure shows a time series with one million data points from one sensor. The right figure shows the same time series after being cleaned.

4.3 Method 1: SimpleBaseline

An anomaly detection model was created to function as a simple, easy-to-understand baseline for evaluating further models against. A small set of requirements was developed for the baseline, and it should; (1) be easily explainable, (2) fast to implement and easy to maintain, and (3) be able to detect localized anomalies within a given span. It would not be realistic for a model with such novel characteristics to detect other than point anomalies, described in section 2.4.2. The anomaly detection accuracy would be unknown and not verifiable but assumed to be low. Given the nature of the model, only point anomalies from the last user-defined *span* in the moving average would be detected. Nevertheless, it would alert us if further models did not perform as expected and performed "worse" than the baseline. Since we have no labels and difficulty measuring anomaly detection percentage, by "worse" we mean a subjective review of the results. This method will be referred to as *SimpleBaseline*.

We chose a baseline model utilizing exponentially weighted moving average [49] with a *span* or N-day moving average of s = 20 and s = 100. *Span* describes a time series in the form presented in Equation 4.1, where X_t is a sensor reading at timestamp *t*. Given our objective (3), the values chosen for *span* was kept small to avoid aggressive smoothing with the potential of removing or diminishing single, anomalous sensor readings.

$$X_t, X_{t-1}, X_{t-2}, X_{t-3}...X_{t-(s-1)} \tag{4.1}$$

SimpleBaseline resulted in a weighted average of the last *span* measurements, where the weight decreases with each previous measurements. Weighted moving average is presented in Equation 4.2, where x_t is the input, y_t is the result and the w_i are the weights.

$$y_t = \frac{\sum_{i=0}^{t} w_i x_{t-i}}{\sum_{i=0}^{t} w_i} \tag{4.2}$$

The Pandas' exponentially weighted moving(EWM) [50] function supports two variants for exponential weights, were we use the default, shown in Equation 4.3.

$$w_i = (1 - \alpha)^i \tag{4.3}$$

α is calculated based on the *span*, giving us Equation 4.4.

$$\alpha = \frac{2}{s+1} \text{for span } s \geq 1 \tag{4.4}$$

It is worth noting that SimpleBaseline does not and can not take advantage of using several sensor-readings simultaneously. Also, the low span relative to the data set length results in the inability to detect slow changes in single sensor readings and a fast, anomalous change compared to other sensors e.g. low battery or manual operator interference. The pseudo-code for the novel exponentially weighted moving average part of the SimpleBaseline algorithm is depicted in Algorithm 1.

Algorithm 1: SimpleBaseline

```
input   : sensor_list: Set of time series sensor readings
output  : sensor_ewm: A list of the calculated exponentially weighted moving
          average for all sensor readings
Assume: calc_weights(s): Returns a list with exponentially moving weights based
          on Equation 4.3 where s = span
1  $sensor\_ewm \longleftarrow \emptyset$
2  $span \longleftarrow userdefined(20/100)$
3  $weights \longleftarrow calc\_weights(s)$
4  for $i \longleftarrow 0$ to $length(sensor\_list)$ do
5      $weighted\_sum \longleftarrow 0$
6      $added\_weights \longleftarrow 0$
7      for $j \longleftarrow 0$ to $min(span, i)$ do
8          $weighted\_sum$ += $weights[j]$ * $sensor\_list[i - j]$
9          $added\_weights$ += $weights[j]$
10     end
11     $y_i = \frac{weighted\_sum}{added\_weights}$
12     insert $y_i$ into $sensor\_ewm$
13 end
14 return $sensor\_ewm$
```

4.3.1 SimpleBaseline anomaly detection

However, the model presented so far only accounts for the forecasting part and did not contain anomaly detection logic. To check for anomalies, we implemented a standard check using the calculated residual, where a residual is defined by measuring how far the models' estimated value for X_{t+1} is from the measured sensor value.

The standard deviation of the residuals was then calculated as specified in Equation (4.5) and measured against an *anomaly_std_border* variable set to 5. This was arbitrarily set after manual testing and can be changed up or down to change anomaly detection sensitivity. Any residual values having a standard deviation of over 5 was marked as an anomaly. The formula for standard deviation is given by Equation 4.5.

$$std = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{4.5}$$

$x_1, x_2, ..., x_n$ is the observed values of the sample items, $\overline{x}$ is the mean value of these observations and *N* is the number of observations in the sample. Algorithm 2 presents the anomaly detection check in its entirety.

Algorithm 2: SimpleBaselineAnomaly

```
input : $sensor\_ewm$: A list containing the calculated exponentially weighted
        moving average for all sensor readings
        $sensor\_readings$: A list containing the sensor readings
output: $anomaly\_index$: A list specifying the anomalies index position in regard
        to $sensor\_readings$

1  $residuals = \emptyset$ $anomaly\_index = \emptyset$ $anomaly\_std\_border = 5$
2  for $prediction \in sensor\_ewm$ and $target \in sensor\_readings$ do
3      $residual$ = absolute($target$ - $prediction$)
4      insert $residual$ into $residuals$
5  end
6  $x\_avg$ = average($residuals$)
7  $N = length(residuals)$
8  $deviation\_sum$ = 0
9  for $i \longleftarrow 1$ to $N$ do
10     $deviation\_sum$ += $(residuals[i]$ - $x\_avg)^2$
11 end
12 $std \longleftarrow \sqrt{\frac{deviation\_sum}{n-1}}$
13 $anomaly\_check = anomaly\_std\_border * std$
14 for $index, residual \in residuals$ do
15     if $residual$ is greater than $anomaly\_check$ then
16         insert $index$ into $anomaly\_index$
17     end
18 end
19 return $anomaly\_index$
```

The baseline runs with a time complexity O(n). The red dots indicate anomalies. It is worth noting that with the current logic implemented, normal behaviour will be marked as anomalous if it comes after an extended period of anomalous behaviour.

4.4 Method 2: LoFBaseline

In addition to the rudimentary baseline in section 4.3, a more advanced baseline using the Local Outlier Factor(LoF) [7] algorithm was implemented. This was to reduce the possibility of erroneous results from the baseline, as well as provide an additional beam of support when comparing results against the methods implemented from the literature.

The implementation was from Scikits pre-written implementation [61] and is based around the concept of local density. Local density is given by the objects k-nearest neighbours and the distance to them, where distance can, for example, be Euclidean distance. Given an objects local density and that of its neighbours, regions of related density can be identified, where objects of significantly lower density than their neighbours are considered to be outliers. Algorithm 3 presents pseudo-code for LoF. However, Sci-kits implementation adds an outlier label to element, calculated based on the LoF and the $contamination$ parameter, i.e. the proportion of outliers in the set. As we do not have labels, this was left untouched.

Algorithm 3: LoF

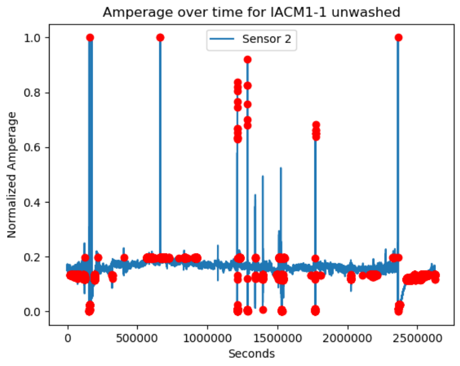
```
input  : k: Number of neighbors
         D: A set of data points
output : LOF: A list containing the calculated local outlier factors
assume: k_distance(D, k, p): Returns a matrix containing the $k\_distance(p)$.
         Denotes the distance the object $p$ has to its k-th nearest neighbours.
         reach_dist_k(D, k-nn, p): Returns the local reachability density(ldr) for
  each p in D. Lrd is defined in Equation 4.6
1 $LoF \longleftarrow \emptyset$
2 for $p \in D$ do
3     $KNNeighbors \longleftarrow k\_distance(D, k)$
4     $lrd \longleftarrow reach\_dist\_k(KNNeighbor, k, p)$
5     $t\_lof \longleftarrow$ calculate LoF using Equation 4.7
6     insert $t\_lof$ into $LoF$
7 end
8 return LoF
```

$$lrd_k(p) = [\frac{\sum_{O \in N_k(p)} reach - dist_k(p, o)}{|N_k(p)|}]^{-1} \tag{4.6}$$

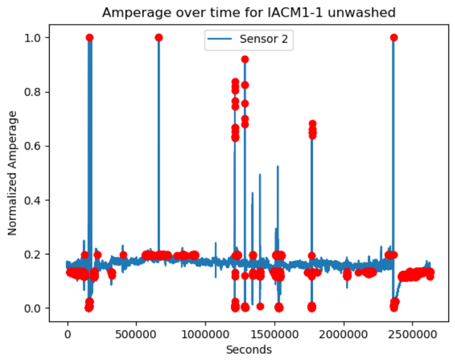
$$LOF_k(p) = \frac{\sum_{O \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|} \tag{4.7}$$

Figure 4.5 is an example of a finished run with n = 20 and n = 100. LoF closely resembles SimpleBaseline in the way it uses its neighbours to determine if a point is anomalous. However, the way they define a *neighbour* differs slightly. This will suffer the same shortcomings as discussed in Section 4.3.

Nevertheless, due to the local nature of LoF, it is able to distinguish and label anomalies in a part of the data set that normally would not be flagged as anomalous in other parts. For example, a point situated closed to a dense cluster of neighbours might be labelled as an anomaly, while a point far away a sparse cluster might not. LoF will not take advantage of the time series associated with our problem and data set but should provide an indication of anomalous areas and points.



(a) LoF with 20 neighbours



(b) LoF with 100 neighbours

Figure 4.5: A run with LoF on Sensor 2 with 1 months of data. Figure a operates on 20 neighbours and resulted in 9244 anomalies detected. The tail at the end indicates the change of an anode and is marked as anomalous behaviour. Figure b operates with 100 neighbours and resulted in 10519 anomalies, but the change of the anode was not marked as anomalous. Changing the number of included neighbours did not change the areas where anomalies were detected, as you can see in the similarity between the two graphs.

4.5 Recurrent neural networks

The original paper [46] presents a solution using a single layer LSTM and GRU. However, this was changed in our application. A single LSTM and GRU architecture had trouble modelling the IACM data, and we observed mirroring of the data input resulting in the LSTM and GRU predicting a straight line, and therefore changed in our application. The hyperparameters presented in the paper was used as a starting point and evaluated on the data set from section 4.2.

Both LSTM and GRU will be presented in the section below due to following the same preprocessing and training procedure.

4.5.1 Method 3: LSTM and GRU

Architecture

The full architecture of the stacked LSTM and GRU networks are shown in Figure 4.6. It consists of three stacked layers in the hope of potentially allowing the hidden state at each level to operate at a different timescale [52] and thus model the problem better given the large time frame of trends in the data set. As previously mentioned, one of the known seasonality changes associated with our problem is that an anode is changed after between 22-30 days. When talking to Hydro about this, they've had problems when internally testing ML architectures as they did not correctly model the unexpected change correctly. A deeper architecture would, in theory, lay the groundwork for an exponential increase in efficiency when trying to model and approximate the regression function [52].

This was also done with the notion that a deeper model would require less training time and increased modelling accuracy compared to a shallow one [52]. Graves et al. [21] found that increasing the depth of a RNN significantly impacted the architectures modelling precision, more so than the number of neurons in a single layer. This may allow for a deeper architecture with fewer total nodes, thus reducing training time. In addition, dropout is enabled in layer 1 and 3 to prevent overfitting during training, before entering a dense layer giving us the output.

ADAM [32] was used as the optimizer for all LSTM and GRU variants and Mean Square Error (MSE) [29] was used as the loss function over RMSE. This was done because RMSE has been reported as not being a good indicator of average model performance [71].

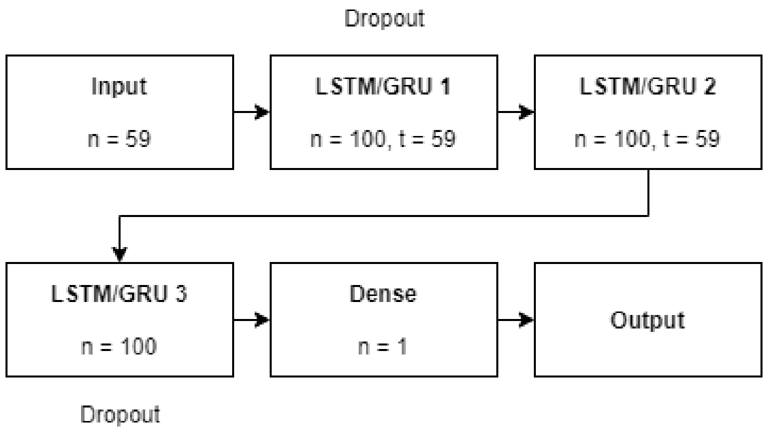


Figure 4.6: The LSTM and GRU architecture used. Consists of 3 stacked layers. The numbers represent the output for each layer and consists of neurons and the output timesteps. Layer 1 and 3 also has dropout enabled. The output is one number representing the models prediction.

Training hyperparameters

A total of 6 hyperparameters influence the training of the LSTM and GRU.

1. **Epochs:** The number of times all of the training samples pass through the learning algorithm. I.e 5 epochs results in the training algorithm seeing the whole training set 5 times.
2. **Batch size:** Number of training examples that will be propagated through the network.
3. **Dropout:** A regularization technique to prevent overfitting. Measured in percentage of nodes that are dropped during each iteration of training.
4. **Time steps:** The number of previous time steps the model is presented with each iteration.
5. **Validation split:** The training and validation split used during training. Used to calculate the validation loss during training.

6. **Neurons:** The number of input neurons for the given layer in the model.

The chosen values and their reasoning are presented in Table 4.2

Hyperparameter	**Value**	**Reasoning**
Epochs	5	A total of 5 epochs was chosen after conducting a small pilot study. Both models converged fast and there was little to no difference in accuracy after 2-3 epochs. For each run, a model was saved after each epoch, resulting in 5 models for one training run.
Batch size	128	Trade off between compute time and accuracy. In general, a higher batch size results in poorer model generalization [30] and the general consensus is that is should be a power of two. 128 was chosen as a sweet spot between generalization and accuracy according to the literature and was kept the same for the data sets presented in Table 2.2
Droput	0.2	Empirically chosen based on previous experience with LSTM and dropout.
Time steps	59	This was kept the same as the original paper. The model is fed 59 seconds of previous time steps in addition to the current time step for each training example.
Validation split	0.2	20% of the training data samples are used as a validation set.
Neurons	Layer 1, 2 and 3: 100	This was increased from the proposed 30 and 60 from the literature to allow for a wider model to improve generalization and accuracy given the increased size in data set and features.

Table 4.2: The hyperparameters used for training all LSTM and GRU models and their reasoning.

Normalization

The data set is normalized using Scikits MinMax-scaler [62] between the custom range of (-1, 1). This is done by scaling and translating each feature individually so that it fits in the given range. The transformation is given in equation 4.8

$$\begin{aligned} X_{std} &= (X - X_{min})/(X_{max} - X_{min}) \\ X_{scaled} &= X_{std} * (max - min) + min \end{aligned} \tag{4.8}$$

where min, max is given by our custom range, (-1, 1). X_{min} and X_{max} denote the range of possible values in the data set.

Training the model

After performing normalization as described in section 4.5.1, training follows a standard regression problem pattern where the main goal is to accurately find and describe the relationship between input parameters and the output. The data set does not contain any anomalous labels, and the use of expert knowledge to try and label anomalies are too time-consuming and not realistic.

While performing training, the use of a validation set to track and compare training progress is preferable. For all training scenarios with LSTM and GRU, 20% of the data set was used as a validation set. The validation set contains data coming last in the time series, i.e. the newest data time-wise.
A total of 12 different models for both LSTM and GRU were trained. The model names and difference in training parameters and data set is represented in table 4.3. All models were asked to predict based on Sensor 2.

Model name	**Data set**	**Number of sensors**
LSTM_{1-1}	1 month	1
LSTM_{2-1}	2 months	1
LSTM_{3-1}	3 months	1
LSTM_{1-6}	1 month	6
LSTM_{2-6}	2 months	6
LSTM_{3-6}	3 months	6
GRU_{1-1}	1 month	1
GRU_{2-1}	2 months	1
GRU_{3-1}	3 months	1
GRU_{1-6}	1 month	6
GRU_{2-6}	2 months	6
GRU_{3-6}	3 months	6

Table 4.3: Final set of models and their training differences for LSTM and GRU

4.5.2 Anomaly detection

As explained in section 4.2, the data we are working with does not contain any labels and thus prevents us from training a classifier to try and label anomalies. The method used for anomaly detection for LSTM and GRU is similar to the technique presented in Algorithm 2, except in how the anomaly border is found. Our implementation of LSTM and GRU differs from the literature [46] as we do not have a labelled classification problem and therefore have to use a different approach to finding anomalies.

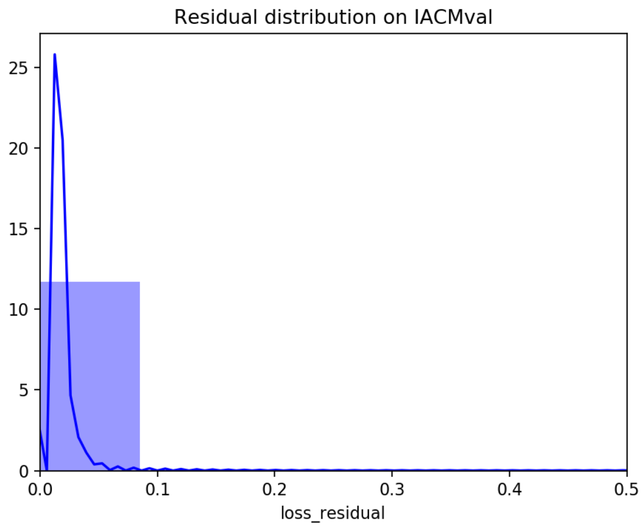
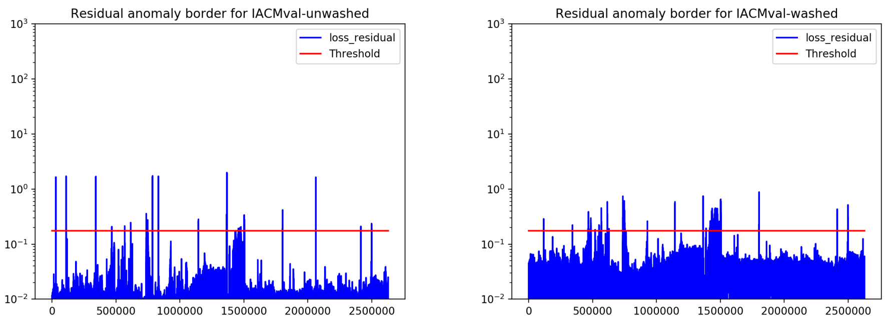


Figure 4.7: Example of plotting the residual distribution for IACM$_{val}$ used for determining the anomaly border. Distribution plot with 20 bins in the range of [0,2.5], showing a minimum of loss variance, where most loss is in the range [0, 0.1].

The method for finding the residual deviation border for LSTM/GRU is presented in Algorithm 4. Note that the border consists of manually finding a residual border and not a standard deviation border.

Algorithm 4: Residual anomaly border

```
input: Predictions: A list of predictions made by LSTM or GRU
1 $residuals \longleftarrow \emptyset$
2 for $prediction \in predictions$ and $target \in sensor\_readings$ do
3     $residual$ = absolute($target$ - $prediction$)
4     insert $residual$ into $residuals$
5 end
6 plot residual distribution for validation and training set like Figure 4.7
7 manually determine a residual border
```



(a) IACM$_{raw}$ **(b)** IACM$_{cleaned}$

Figure 4.8: (a) Residual anomaly border for IACM$_{raw}$, with the red line indicating the manually set anomaly border. (b) Residual anomaly border for IACM$_{cleaned}$. Notice the difference in residual spike caused by removing noisy sensor readings.

The result is a method that classifies anomalies based on the difference in model prediction and actual sensor output. However, the anomaly border is a question of user definition and results in the models' anomaly output being determined by a user and therefore, subject to change. The implications of this on results will be further discussed in Section 4.6.

4.6 Method 4: Hierarchical Temporal Memory

In the original paper [3] and the example implementations in the NuPIC framework, the timestamp is used together with the sensor values as an input. The seasonal dependencies in our data are mainly based on the anode changes that happen at varying time intervals. There is no exact correlation between the timestamps and the seasonality in the data because the seasonality varies depending on the speed of the chemical process and consumption of the carbon anode. The timestamps are therefore not used in the encoded input.

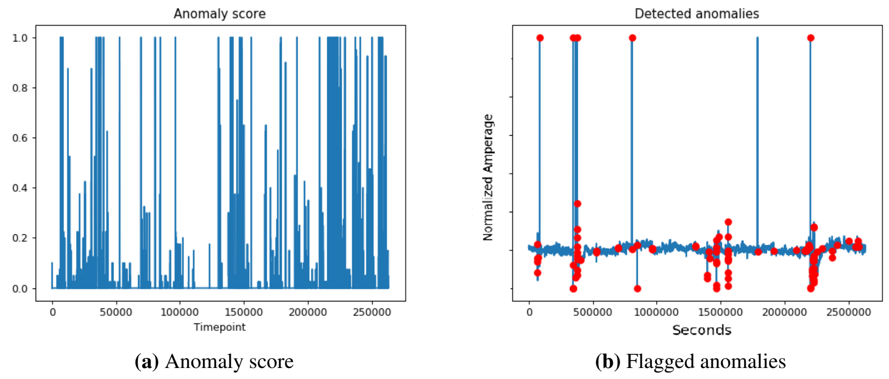
Architecture

The HTM library handles calculating an anomaly score for each example through the HTM pipeline described in section 2.6.5. HTMs "run function" uses data points from one timestamp as input, updates the state of the model, and makes a prediction for the SDR encoding of the trailing value. This can be decoded back to a scalar value to get a prediction of the next example, or you can use the predicted SDR directly to calculate an anomaly score. For our model, we used the anomaly score to detect anomalies.

The anomaly score is 0 if the pattern of the SDR was predicted, and 1 if the pattern was not predicted at all. The anomaly score is calculated based on the deviation between the models' prediction(π) and the actual input a given the previous input x:

$$s_t = 1 - \frac{\pi(x_{t-1}) \cdot a(x_t)}{|a(x_t)|} \tag{4.9}$$

We get an anomaly score for each point in the time series ranging from [0-1]. We then use a threshold of 0.99[1] to flag anomalies. Every sample with a score above 0.99 will be considered anomalies. Figure 4.9 shows an example of anomaly scores and anomalies detected from the HTM algorithm.



(a) Anomaly score **(b)** Flagged anomalies

Figure 4.9: Example run from HTM run on the raw IACM$_{1-1}$. The left figure shows the anomaly score for each data point from 0-1, and the right figure shows the anomalies flagged as red circles on the time series.

An alternative approach would be to use the anomaly likelihood instead of the anomaly score. The anomaly likelihood represents the current state of predictability. The anomaly likelihood is calculated from an estimate of the historical distribution of anomaly scores. Equation 4.10 shows how the the likelihood, L_t, is calculated. $\widetilde{\mu_t}$ is the mean of the anomaly score of a short term moving average, μ_t is the mean and σ_t is the deviation from the historical distribution. Q is a function deciding the anomaly score.

$$L_t = 1 - Q(\frac{\widetilde{\mu_t} - \mu_t}{\sigma_t}) \tag{4.10}$$

From early pilot studies, we did not see a significant change in the areas classified as anomalies from using anomaly likelihood and chose to proceed using the raw anomaly score.

Finding the hyperparameters

To find the hyperparameters, we used the swarming algorithm in the NuPIC library with a medium swarm size [48]. The swarming algorithm outputs a python file with a dictionary containing hyperparameters for running the HTM algorithm. The most essential hyperparameters from the run are listed in table 4.4.

[1]The value of 0.99 was recommended by one of the Numenta community managers on the Numenta forum.

Parameter	**Value**	**Explaination**
w	272	Size of the input space
n	21	Number of ”ones” generated from the scalar encoder
ColumnCount	2048	Number of columns in the SDR
numActiveColumnsPerInhArea	40	Number of active columns in the SDR($\sim$2%)
cellsPerColumn	32	Cells per column used by the temporal pooler
activationThreshold	14	Activation to make a cell predictive
permanenceDec	0.1	Speed of permanence decrementation in the TP
permanenceInc	0.1	Speed of permanence incrementation in the TP

Table 4.4: Some of the parameters from NuPIC swarming algorithm run on IACM_{1-1}

Training the model

A total of six different models for HTM was trained with the six data sets described in 4.2. The models are represented in table 4.5. All models were asked to predict based on Sensor 2.

Model name	**Data set**	**Number of sensors**
HTM_{1-1}	1 month	1
HTM_{2-1}	2 months	1
HTM_{3-1}	3 months	1
HTM_{1-6}	1 month	6
HTM_{2-6}	2 months	6
HTM_{3-6}	3 months	6

Table 4.5: Final set of models for HTM

4.7 Method 5: YASA with OCSVM

The fifth and final architecture is based on Yet Another Segmentation Algorithm(YASA) and One-Class Support Vector Machine (OCSVM). The following section describes YASA with OCSVM and how we implemented, trained and further developed the solution for our data set. The hyperparameters in the original paper [41] were tuned using evolutionary algorithms. It would be hard to define fitness function for our models, as we have no exact way to evaluate their performance without labelled data. The hyperparameters are, therefore, manually set.

4.7.1 YASA

YASAs main purpose is to segment and divide the time series into homogeneous or close to homogeneous series with low approximation errors, focusing on low computational cost and "easy parametrization". In general, a time series segmentation algorithm can be expressed as a function f that creates X number of segments, given a time series S, such that,

$$f : S \longrightarrow \{S_1, S_2, ..., S_x\}$$

where the resulting set of segments $\{S_1, S_2, ..., S_x\}$ display two properties: (1) $S = \cup_{i=1}^{X} S_x$, i.e the time series S can be reconstructed without loss and (2) $\cap_{i=1}^{X} S_x = \emptyset$, or in other words, no overlap between segments.

The pseudocode for the algorithm is presented in Algorithm 5 and is best understood as a recursive algorithm. It starts by checking whether the maximum number of recursive calls are met and if yes, returns the segment. If not, linear regression is performed on the segment. If the linearity test for the linear regression is passed, the segment is returned as a unique segment. However, failing the linearity test results in finding the point, t_s, with the largest residual error from the linear regression, and splitting the segment at this point. YASA is then called recursively on these two smaller segments.
The result is a set of segments that passes the linearity test and are deemed as "unique", allowing the classifier to focus the attention on the most relevant part of the time series. The standard SVM classifier is not adjusted to process large training sets, as the computational complexity can reach O(n^3). Segmenting the data makes the model able to efficiently

process small segments, fitting the model one segment at a time.

Algorithm 5: The pseudocode for YASA, as presented in the literature [42]

```
input : $S^{(j)}_{t_{max},t_0}$: time series data of sensor j corresponding to time interval
        $[t_0, t_{max}]$.
        $p_{min} \in [0,1]$: maximum significance for statistical hypothesis test of
        linearity.
        $l_{max} > 0$, maximum levels of recursive calls.
        $s_{min} > 0$, minimum segment length.
output: $\Phi$ :=$\{\phi_1, ..., \phi_m\}$, data segments

1  Function SegmentData($S^{(j)}_{t_{max},t_0}$, $p_{min}$, $l_{max}$, $s_{min}$):
2      if $l = l_{max}$ then
3          return $\Phi = \{S^{(j)}_{t_{max},t_0}\}$
4      end
5      Perform linear regression, $\{m, b\} \longleftarrow LinearRegression(\{S^{(j)}_{t_{max},t_0}\})$.
6      if $LinearityTest(S^{(j)}_{t_{max},t_0}, m, b) > p_{min}$ then
7          return $\Phi = \{S^{(j)}_{t_{max},t_0}\}$
8      end
9      Calculate residual errors, $\{e_0, ..., e_{max}\}$ = Residuals($S^{(j)}_{t_{max},t_0}, m, b$)
10     $t_s \longleftarrow t_0$
11     while $max(\{e_0, ..., e_{max}\}) > 0$ and $t_s \notin (t_0 + s_{min}, t_{max} - s_{min})$ do
12         Determine split point, $t_s = argmax_t\{e_t\}$
13     end
14     if $t_s \in (t_0 + s_{min}, t_{max} - s_{min})$ then
15         $\Phi_{left}$ = SegmentData($S^{(j)}_{t_s,t_0}, p_{min}, s_{min}, l + 1$)
16         $\Phi_{right}$ = SegmentData($S^{(j)}_{t_{max},t_s}, p_{min}, s_{min}, l + 1$)
17     end
18     return $\Phi = \{S^{(j)}_{t_{max},t_0}\}$
```

The segmentation is adjusted with; *minimum segment length*, *maximum levels of recursive calls*, and the p_{min} value. The minimum segment length and the maximum levels of recursive calls make sure the segments have a reasonable minimum length and keeping the algorithm efficient. The p-value sets the threshold for which segments are recursively split, and which segments are returned. A lower p-value will cause the linear check inline 5 in 5 to accept segments with higher approximation errors, causing fewer large segments.

As you can see in figure 4.10, when increasing the resolution of the plot, our data has a high variance from fluctuations and noisy sensor readings, causing a linear regression to have difficulties correctly modelling our data even for small segments.

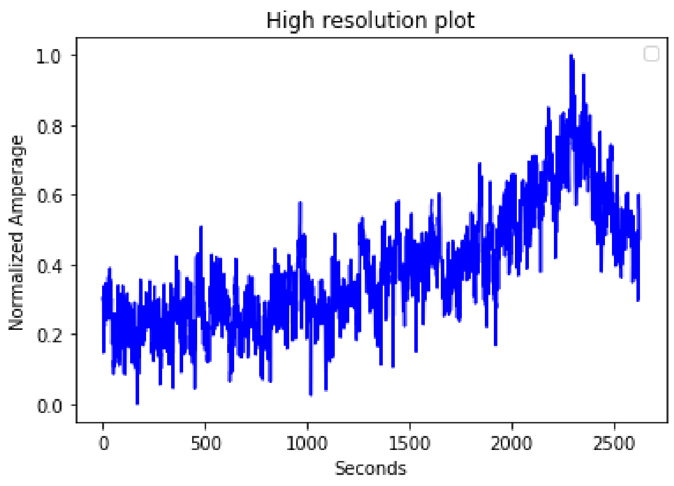


Figure 4.10: Zoomed in plot of data from Sensor 2, showing 40 minutes of data.

By using values for p_{min} in the range of [0.01-0.1], the linearity test is seldom passed because of the high variance in the data. Most segments are returned because the minimum segment length is reached. This results in the segments having diminished value since it doesn't differ much from manually setting the same fixed length for every segment. To get the algorithm to return segments based on the linearity test, we have to set p_{min} extremely low, accepting segments with a high error. In figure 4.11 you can see YASA segmentation on one week of data, where only 17 of the 91 segments passed the linearity test with $p_{min}=10^{-5}$.

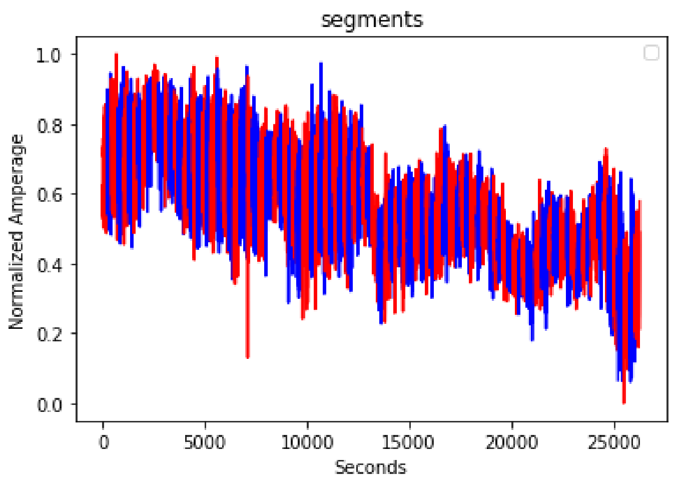


Figure 4.11: YASA segmentation of seven hours of data from Sensor 2. Blue and Red is used to differentiate between segments.

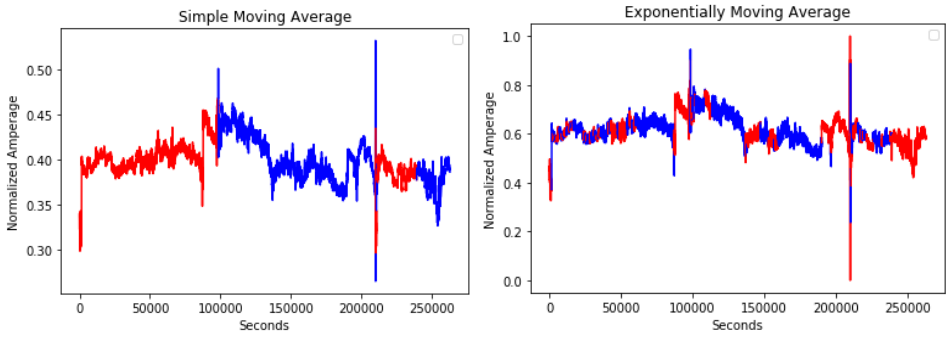
To combat accepting segments with high error residuals, the use of smoothing and aggregation was tested.

Smoothing

Two smoothing techniques were tested. A simple moving average, where each data point is calculated with equation 4.11 using span n, and an exponentially moving average the same way as explained in section 4.3. The exponentially moving average will preserve the tops and bottoms more than the simple moving average.

$$y_t = \frac{\sum_{i=t}^{t+n-1} x_i}{n} \tag{4.11}$$

Smoothing the data resulted in fewer larger segments. The the bigger the span, the fewer segments for both of the techniques. In figure 4.12a and 4.12b you can see the two smoothing techniques both using a span of 50 and and p$_{min}$=0.01.



(a) Simple moving average **(b)** Exponentially moving average

Figure 4.12: Results of segmentation after smoothing. Both techniques with of 3 days of data after using with a span of 50. p$_{min}$=0.01, l$_{max}$ = 30 and s$_{min}$ = 500 were used for YASA. Average segment for SMA and EMA is 44000 and 3300. The change in the y-axis occur because the normalization is done before the smoothing.

Aggregation

Aggregation in time series is collecting all data points over a specified period (the polling period) to a collective value. This value reflects a statistical view of the collected and aggregated data points. In the smoothed time series, each point represents a collective value. We down-sampled the result of the simple moving average to get the aggregation, using every nth value of the smoothed result. Figure 4.13 shows how the aggregation affect the segmentation.

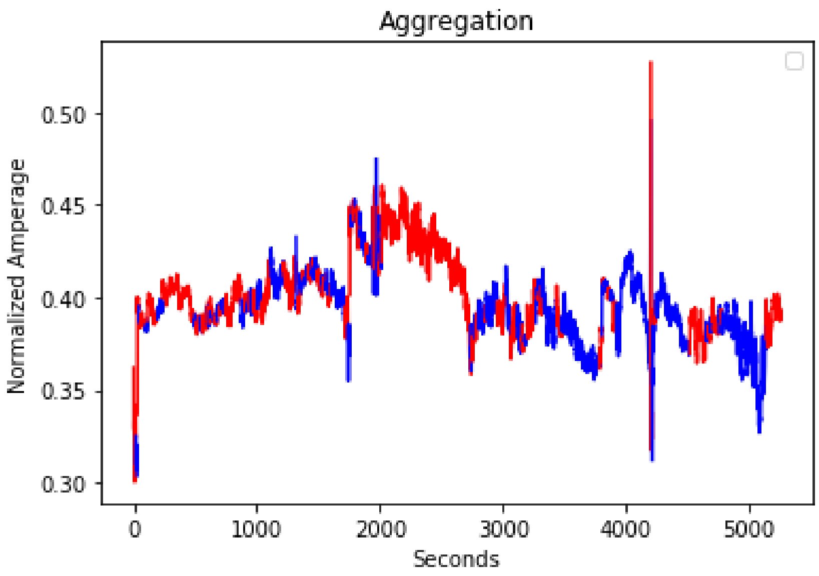


Figure 4.13: Segmentation of 3 days of data after using aggregation with polling period 50. p_{min}=0.01, l_{max} = 30 and s_{min} = 10 were used for YASA. Average segment length is 49

4.7.2 One Class Support Vector Machine(OCSVM)

For each segment returned from YASA, the model fits and predicts anomalies using an OCSVM as explained in the theory chapter in section 2.6.4. The non-linear kernel RBF [1] was used for all experiments.

Hyperparameters

Three hyperparameters were tuned for the OCSVM. The main goal of tuning these parameters was reducing the number of outliers, as the model classified a lot of what we assume to be false positives. Since we do not have a log of actual anomalies, and can not objectively measure the change in performance, these values were subjectively selected.

1. ν**:** An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. By default 0.5 will be taken.
2. γ**:** Kernel coefficient for RBF. Defines how far the influence of a single training example reaches.
3. ϵ**:** The stopping tolerance, affects the number of iterations used when optimizing the model.

The chosen values and their reasoning are presented in Table 4.6.

Hyperparameter	Value	Reasoning
ν	0.01	If the value is set to 0.5 you will allow up to 50 % of the data to be classified as outliers asymptotically. When we set this value too low, the number out outliers detected increased. We found this value to give a reasonable amount of detected anomalies.
γ	1	With a high gamma value, the support vectors are less affected by the data points far from the decision boundary. 1 is the same as the "auto" setting for our dataset with one single feature. This value gave a reasonable amount of detected anomalies.
ϵ	1e-3	Training the model longer that the default stopping criterion did not provide any detectable performance boost. This was concluded in early pilot studies.

Table 4.6: The hyperparameters used for OCSVM.

Training the model

As the YASA is limited to segmenting single time series, the IACM$_{i-6}$ data sets using data from 6 neighbouring sensors is not possible to run on this model. The data sets used are 1, 2 and 3 months from a single sensor with an addition of the cleaned data set for 1 month. The model names are represented in table 4.7.

Model name	Data set
YASA$_{1-1}$	1 month
YASA$_{2-1}$	2 months
YASA$_{3-1}$	3 months

Table 4.7: Final set of models for YASA OCSVM

Anomaly detection

The one class support vector machine classifies every data point that falls outside of the support vectors that make the boundary for normal data, as anomalies. For each of the segments from YASA, the one OCSVM trains and detects outliers for that specific segment. Figure 4.14 shows an example of the detected anomalies using data smoothed with a simple moving average with a span of 50, and YASA segmentation with p$_{min}$ of 0.01.

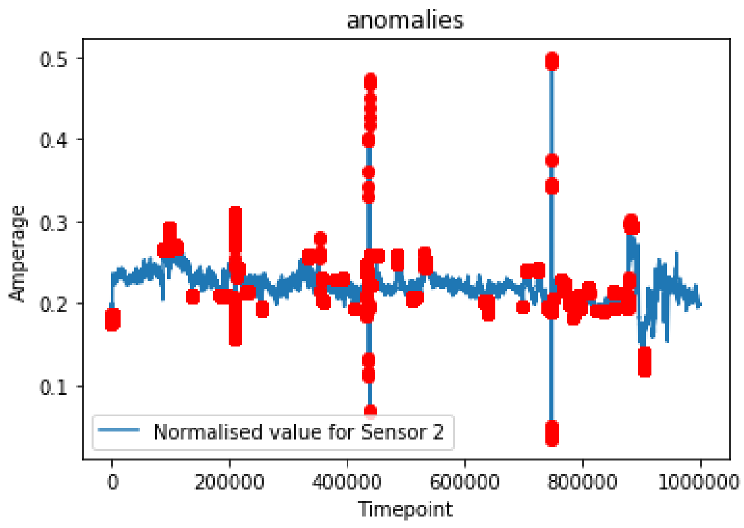


Figure 4.14: Example of anomalies detected from a run of the full YASA with OSCVM algorithm. This is just above 12 days of data from Sensor 2. A total of 17000 points were classified as anomalies. That is 1.7 % of the data.

4.8 Separating anomalies

So far, the issue of separating process and sensor anomalies in accordance with RQ 3 (1.1) has not been addressed. We came up with three possible strategies and proceeded to test two of these.

1. Shifting the data of the neighbouring sensors
2. Comparing with anomalies detected in neighbouring sensors
3. Comparing standard deviation in cell voltage

4.8.1 Shifting the data of the neighbouring sensors

This strategy is based on the assumptions:

1. Changes in the process of an anode affects the neighbouring anodes.
2. By giving the model data from the sensors neighbours at timestamp t+1, it will be able to forecast changes happening in the process.
3. One second of information from the neighbours ahead of time is enough to forecast these changes.

Both LSTM and GRU (section 4.5.1) and HTM (section 4.6) detects anomalies based on the error of the models forecast. Giving the model information from neighbouring anodes ahead of time could make the model better at predicting changes in the process. In equation 4.12 the model outputs a forecast of the sensor value at t=1, based on the sensor value at t=0 and the neighbours values at t=1. s is the sensor where the anomalies are detected, and n1-n5 are its nearest neighbours.

$$(s_{t+0}, n1_{t+1}, n2_{t+1}, n3_{t+1}, n4_{t+1}, n5_{t+1}) \xrightarrow{model} s_{t+1} \tag{4.12}$$

The idea was that making all three architectures able to model the process anomalies, the anomalies remaining would be sensor anomalies.

4.8.2 Comparing with anomalies detected in neighbouring sensors

This strategy is based on the assumptions:

1. Changes in the process of an anode affects the neighbouring anodes.
2. A process anomaly at one anode will be detected in the neighbouring anodes.

If an anomaly is detected at several neighbouring anodes at the same timestamp, it is probable that an anomalous event occurred in the process, and that the sensor reading is reliable. If an anomaly is detected on one anode, and the neighbouring anodes describe a normal process, the detected anomaly is classified as a sensor anomaly. In contrast, if there is a correlation between anomalies detected in neighbours at the same timestamp, the anomaly is classified as a process anomaly.

4.8.3 Comparing standard deviation in cell voltage

This strategy is based on the assumptions:

1. A process anomaly detected in one anode can also be detected in the total cell voltage.
2. Sudden increase in standard deviation is detected as anomalies by the model.

A domain expert at Hydro suggested that we could use the cell voltage to separate sensor anomalies. According to Ohm's law 2.5, there is a linear relationship between voltage and current, having constant resistance. The anodes are connected in parallel, so an increase in the standard deviation (equation 4.5) of an anode will cause increased standard deviation in the total cell voltage. If this is not the case, it is probable that the sensor is defect.
For each anomaly detected, we check the standard deviation in a span around where the anomaly occurred for both the current and the cell voltage. If they both have breached a set threshold, the anomaly is classified as a process anomaly. Pseudocode for this method

shown in Algorithm 6.

Algorithm 6: Standard deviation anomaly separation

```
input  : D_c: Time series data of current, corresponding to time interval $[t_0, t_{max}]$.
         D_v: Time series data of voltage, corresponding to time interval $[t_0, t_{max}]$.
         A: List of timestamps for detected anomalies in time interval $[t_0, t_{max}]$.
         s: Size of the window used for calculating the standard deviation.
         t_c: Standard deviation threshold for current.
         t_v: Standard deviation threshold for voltage.
output : R_s: A list containing sensor anomalies
         R_p: A list containing process anomalies
assume: $calc\_std$(D, a, s): Returns the standard deviation in a segment from time
         series D with span s, centered at time stamp a.

1  $R_s \longleftarrow \emptyset$
2  $R_p \longleftarrow \emptyset$
3  for $a \in A$ do
4      std_c ⟵calc_std($D_c$, a, s)
        std_v ⟵calc_std($D_v$, a, s)
        if $std_c \geq t_c$ AND $std_v \geq t_v$ then
5          insert a into R_p
6      else
7          insert a into R_s
8      end
9  end
10 return R_s, R_p
```

4.8.4 Final separating techniques

The detected anomalies for for both LSTM and GRU, and HTM are based on a threshold. We assumed it would be optimistic to think that in general, process anomalies were affecting the neighbours enough to breach the same threshold, and therefore chose to focus on strategy 1 and 3.

Chapter 5

Anomaly separation and detection results

In chapter 3, we experimented with several machine learning models and training techniques for anomaly detection in an industrial sensory-based time series. To try and answer research question 2, first presented in section 1.1 and re-iterated here for convenience:

> *RQ2: What implications do these solutions have in regards to detecting and categorizing sensor anomalies?*

The following chapter will present both empirical evidence and a qualitative approach in an attempt to recognize and separate sensor anomalies. The chapter will be divided into three parts: Section 5.2 presents the raw data results, Section 5.3 shows the cleaned results and Section 5.4 presents the separation of sensor anomalies.

5.1 Result approach

A common denominator for all the literature methods implemented [42][3][46] is the use of a labelled data set or an on-hand domain expert when comparing and evaluating methods. A labelled set opens up for a quantitative approach to analysis and testing, including the use of tools and plots such as ROC curves, F1 score, categorization accuracy, precision measures and more. However, given the raw, unlabeled data set and the lack of an event log for the sensors, our approach will differ from the ones presented in the literature. The method of injecting synthetic anomalies was deemed outside the scope of this thesis, as the extensive use of a domain expert to create anomaly characteristics and ensuring the data set contained only ”normal behaviour” is too time-consuming, and hard to realize. As a result, there is an uncertainty connected to the presence of sensor anomalies in the data set as a whole and the number of process anomalies.

The qualitative approach taken will be based on the assumption that a lions share of sensor readings are correct readings, i.e. the data set reflects normal procedure and sensor readings are within a normal range. To accurately compare calculated anomalies between models, all models will be asked to predict on the 4th month, as shown in Figure 5.1.

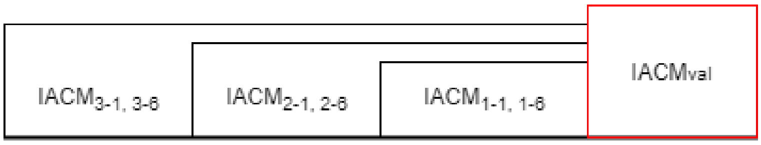


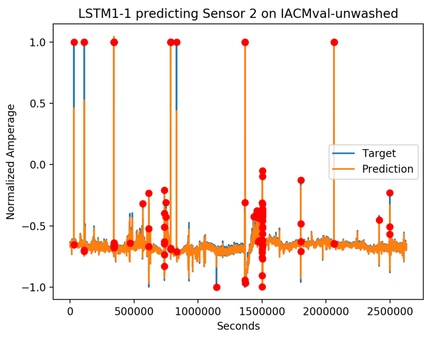
Figure 5.1: A graphic showing the different data sets and how they do not contain overlapping segments between the training data sets and the validation set, marked in red.

With the limitations previously mentioned, our approach will first present the IACM$_{raw}$ results. Here we attempt to highlight how the detection frameworks are affected by the noisy readings described by the domain expert, and how we decided which models to pursue further. Secondly, we will present the results for IACM$_{cleaned}$ along with a discussion on how models acted when presented with normal process irregularities, such as changing an anode. In addition, we will explore similarities between detected anomalies. Lastly, the results of separating between process and sensor anomalies will be explored.

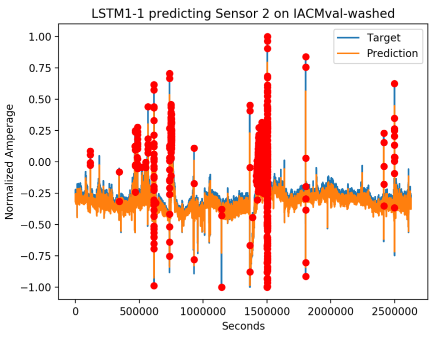
5.2 IACM-raw

In section 4.2.1 we explained how extreme values were filtered away from IACM$_{raw}$, as these were known to be faulty readings. The regression models of LSTM/GRU and HTM struggled to predict these spikes because they don't follow any pattern and are inherently random in nature. When the models miss-predicts a spike, the high loss will aggressively adjust the regression function of the models to better predict future spikes in the time series, not only affecting the anomalies detected, but also the training of the models. For the third method, YASA and OCSVM are both affected by spikes. A segment with values far from the median will lead to an increase in the residual of the linear regression for that segment, resulting in an increased number of segments, and the decision boundary of the OCSVM will be adjusted by these extreme values.

The spikes were in a vast majority of the cases classified as anomalies by all the models, causing a high amount of false positives. Figure 5.2 shows anomalies detected by LSTM$_{1-1}$ from IACM$_{val}$, where Figure 5.2a is trained and tested using the IACM$_{raw}$, and Figure 5.2b is trained and tested using IACM$_{cleaned}$.



(a) LSTM$_{1-1}$ raw



(b) LSTM$_{1-1}$ cleaned

Figure 5.2: Prediction and anomaly detecting of LSTM$_{1-1}$ trained on both the raw and cleaned IACM$_{val}$.

Adding noise to the data is a common way of regularizing a neural network to prevent over-fitting if you train until convergence [64]. In our case, the amount of noise was presented by Hydro as a challenge with the IACM data, and we assume the data still has a significant amount of noise after removing the extreme values. As the data cleaning is based on expert domain knowledge, all the results presented in the rest of this chapter will use IACM$_{cleaned}$.

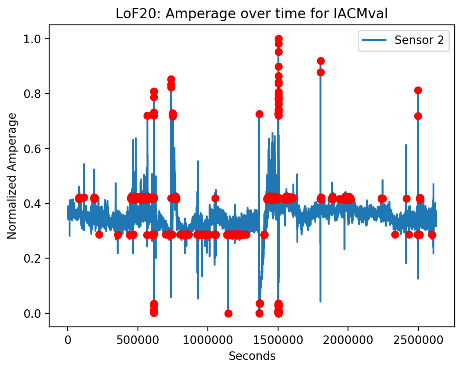
5.3 IACM-cleaned

In this section, we will present and compare the results from the three methods implemented from the literature and the two baselines. We will compare the detected anomalies, see how the data from the neighbours affected the models' predictions, and see how each method handles the anode change present in IACM$_{val}$ for Sensor 2. As explained in section 2.1.4, the anode change occurs approximately once per month, meaning that the models trained on IACM$_{1-n}$ have trained on this event in one specific context, while the models trained on IACM$_{3-n}$ have seen it in three different contexts.

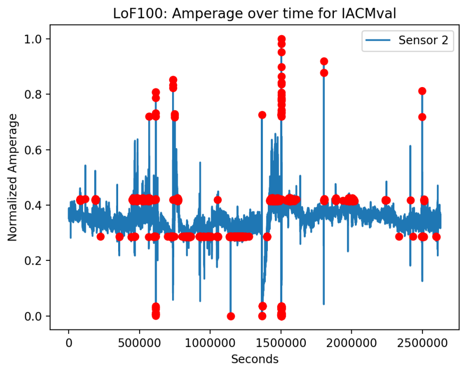
5.3.1 Baseline models

SimpleBaseline and LoFBaseline represent two architectures with low computational complexity, no internal memory or connection between temporal events and a local approach to data evaluation. Hence, the results are thereafter but should indicate anomalous areas that would need further deep-dive in subsequent methods. Both methods result are shown in Figure 5.3.
SimpleBaselines run on IACM$_{val}$ clearly mark the deviation spikes as anomalies, in accordance with its local EWM-average approach. It can not intelligently account for an anode change, or other process related events and therefore marks them anomalous if the residual difference is greater than the anomaly border, as discussed in section 4.3.

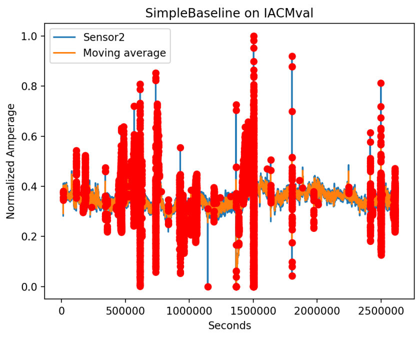
LoFBaseline clearly shows how data points within the ”normal” operating range given to us by Hydro are clustered together and act as neighbours with a short relative density. Effectively, it has created a linear border over and below this range, almost always labelling one or more points crossing the border as anomalous. This is certainly not correct, and fails to account for the variation in sensor readings, but gives us a good indication for the normal operating range of Sensor 2.
Together, the result from both baselines outlines an area of normal operating range and several anomalous spikes and outlier areas that will be further evaluated.



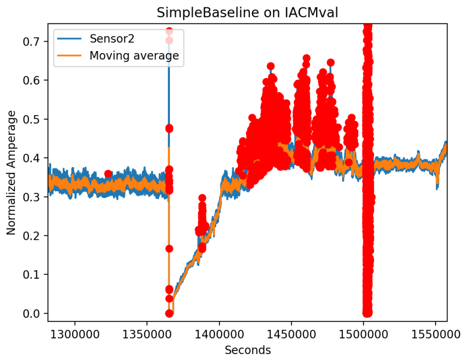
(a) LoFBaseline20



(b) LoFBaseline100



(c) SimpleBaseline



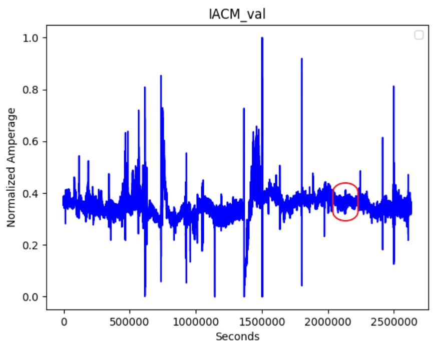
(d) SimpleBaseline anode change

Figure 5.3: LoF and Simple baseline comparison. (a) shows LoF with 20 neighbors for IACM$_{val}$, where the red circles represent marked anomalies. (b) presents LoF with 100 neighbours, not resulting in any visual differences in marked areas. (c) shows the SimpleBaseline with its numerous marked anomalous areas. (d) shows the mark of an anode change, and how SimpleBaseline does not mark the initial rise in amperage, but correctly marks the anomalous area afterwards, further discussed in section 5.3.3.

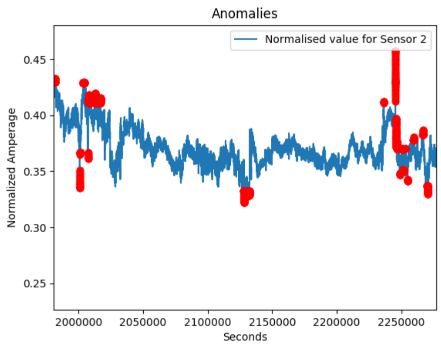
5.3.2 YASA with OCSVM

Figure 4.11 in section 4.7 shows how YASA produces small segments. As explained in section 4.7, three different smoothing approaches were carried out as an attempt to combat this, but it was a question of how much potential loss of detail in the data we were willing to sacrifice to brute force the method to work. Smoothing the data set would also make it harder to compare YASA and OCSVM with the two other methods, as high standard deviation parts of the data could disappear in the smoothing.

The YASA with OCSVM detected a high number of anomalies in sections we assume to be normal data. Figure 5.4 shows anomalies detected on a section of IACM$_{val}$ smoothed with a span of 30. This section looks like normal data, and both baseline models classified the entire section as normal, so it is safe to assume that this method has a lot of false positives. The OCSVM captures regions in the input space where high probability data lives, resulting in a binary anomaly classifier. Because this classifier is trained for each segment, sections with many small segments will have a lower threshold for detecting anomalies, because few points affect the decision boundary. The section in Figure 5.4b was split into many small segments by YASA, causing this result.



(a) IACM$_{val}$



(b) YASA with OCSVM zoomed

Figure 5.4: High resolution plot of anomalies from YASA with OCSVM. The red circle in (a) shows the section in IACM$_{val}$ that is zoomed in on the right figure. The red dots in the right figure shows detected anomalies in the area we assume to be normal data.

This method struggled to segment our data, having regions with a high amount of false positives, and not capturing anomalies in part of the data where we would expect from a method with no temporal memory. Figure 5.5 shows this method run on IACM$_{val}$, where few data points around the anode change are marked because YASA made this one big segment. Because of the unreliable results, we chose to focus our attention on comparing the LSTM/GRU and HTM methods throughout the remaining of this section.

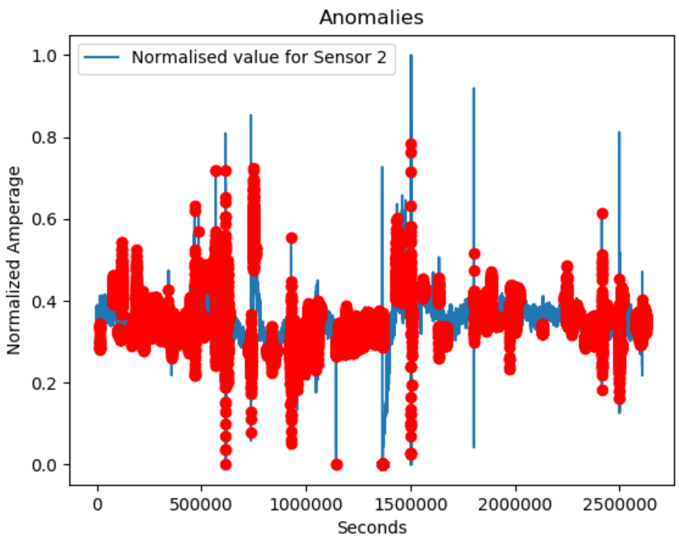
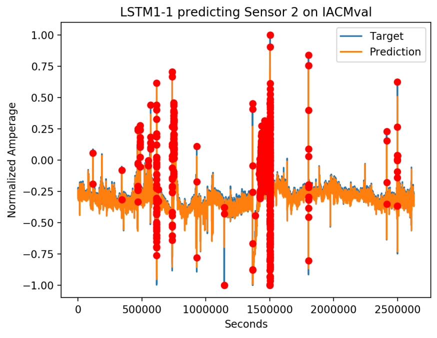


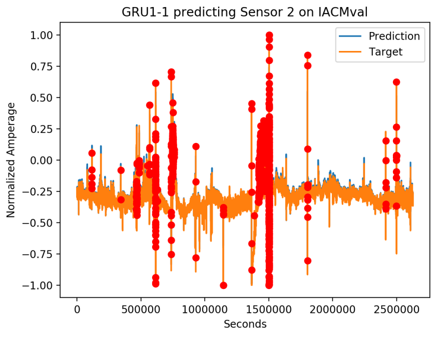
Figure 5.5: YASA OCSVM run on IACM$_{val}$. The anode change at 1.375M seconds and the trailing values was segmented as one big segment, causing a low amount of flagged anomalies in this section compared to the rest of the data set.

5.3.3 LSTM and GRU

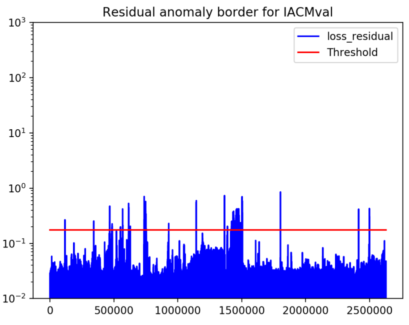
Both LSTM and GRU performed similarly when training and testing on IACM$_{cleaned}$, with the training complexity being the biggest difference, in accordance with the literature [27][74]. Reduced complexity associated with combining the LSTMs *update* and *forget* gate into GRUs single *update* gate is the underlying factor for this change, in addition to the hidden state changes, discussed in section 2.6.3. With the difference between the LSTM and GRU results being almost negligible, presented in Figure 5.6, and hard to differentiate as reasoned in section 5.1, no further distinction between the models will be pointed out, and the results from LSTM and GRU will be presented interchangeably.



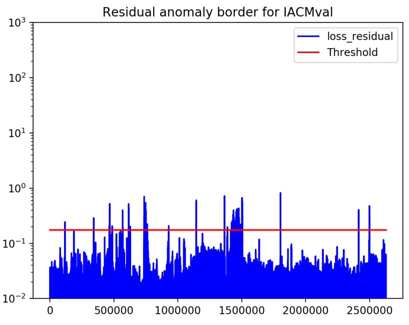
(a) LSTM$_{1-1}$



(b) GRU$_{1-1}$



(c) LSTM$_{1-1}$ anomaly border



(d) GRU$_{1-1}$ anomaly border

Figure 5.6: Comparison between LSTM$_{1-1}$ and GRU$_{1-1}$, where detected and marked anomalies are almost identical. They marked 1194 and 1945 anomalies respectively. However, the identified anomalous areas are almost identical and the difference in anomaly count can be contributed to the small differences in anomaly score, as shown in (c) and (d).

Figure 5.7 shows a close up of the anode change in IACM$_{val}$ for Sensor 2 and indicates that GRU$_{1-1}$ with *only* 5 epochs and one month of data correctly models the event without marking it as anomalous. All LSTM and GRU models exhibit this behaviour. Both architectures appear to possess the capability for long-distance relation modelling, in contrast to YASA with OCSVMs more local approach to anomaly detection and comparison. The RNNs appears to quickly and efficiently learn temporal patterns, especially the ones with a long temporal lag in-between, making it a good fit for the IACM data.

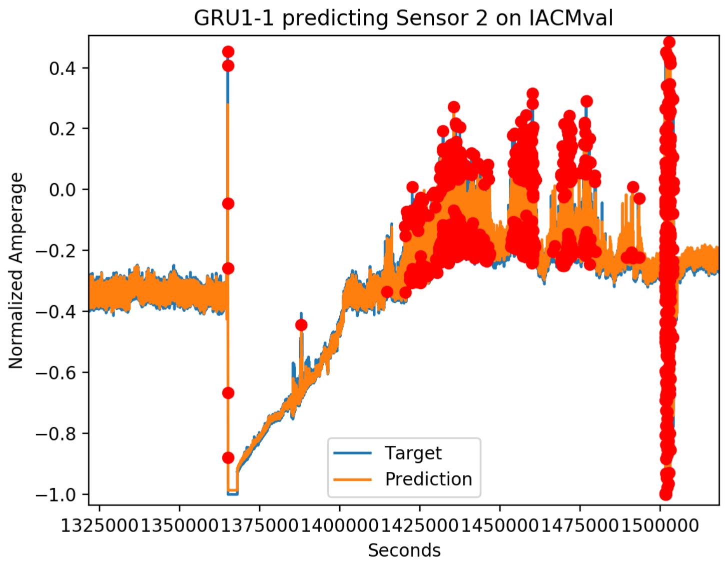
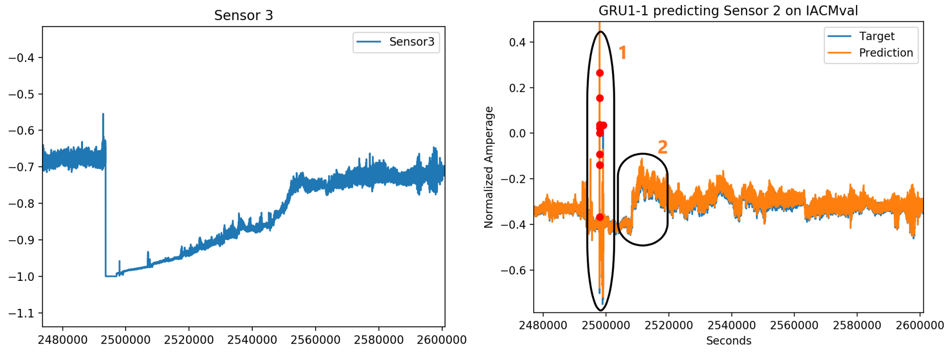


Figure 5.7: The figure shows an anode change and the anomalies outputted from GRU$_{1-1}$, with the red dots indicating anomalies. No anomalous behaviour is marked in the anode change happening from 1.375M seconds to 1.425M. However, some irregularities presumed to originate from the electrolysis process is heavily marked. The *tail* following the anode change has not been detected in previous events, and therefore correctly marked as anomalies given our process knowledge.

In addition, models presented with 1 and 6 sensors as input features correctly learn to account for an anode change in neighbouring anodes, regardless of the number of training months included. Figure 5.8 shows the anode change for the neighbouring Sensor 3 is correctly modelled and accounted for in GRU$_{1-1}$, causing the sudden rise in amperage around 2.51M seconds, marked by *2*. Nevertheless, each anode change results in a sudden, short spike none of the models were able to account for, and it is unclear if that is the normal operating procedure.



(a) Sensor 3 **(b)** GRU$_{1-1}$

Figure 5.8: Sensor 3 shows the sensor readings when performing an anode change. The marked area represented by *1* indicates the spike seen for all anode changes, while *2* details the increase in amperage in the neighboring anode, and how it is not marked as anomalous.

Comparing the basic LSTM$_{1-1}$ from Figure 5.6 with the most data-intensive LSTM architecture, LSTM$_{3-6}$ as shown in Figure 5.9, we see little to no difference in the areas marked as abnormal. Our assumption was that including the neighbouring sensors as features when training would allow the model greater insight into the local electrolysis process. And, thus account for neighbouring fluctuations and how it affects the predicted sensor. In addition, the increase in training sample size to include three months should allow for greater generalization. The lack of a quantitative approach to LSTM and GRU regression analysis makes it hard to safely determine an answer for our hypothesis, but given the initial results from calculating MSE, RMSE, MAE and R^2 for IACM$_{val}$ in Table 5.1, it does look like the increase in timespan and input features slightly increases regression performance. A deeper comparison is needed before concluding, but a regression analysis for LSTM and GRU is outside the scope of this thesis.

Model	**MSE**	**RMSE**	**MAE**	**R^2**
LSTM$_{1-1}$	0.000253	0.015918	0.011708	0.965772
LSTM$_{3-6}$	0.000204	0.014307	0.008170	0.972350

Table 5.1: Regression evaluation metrics for LSTM$_{1-1}$ and LSTM$_{3-6}$, hinting to a better regression performance for the data intensive model. R^2 is a regression score function and a measure of how close the data are to the fitted regression line given by the models predictions, where a higher number indicates a better fit.

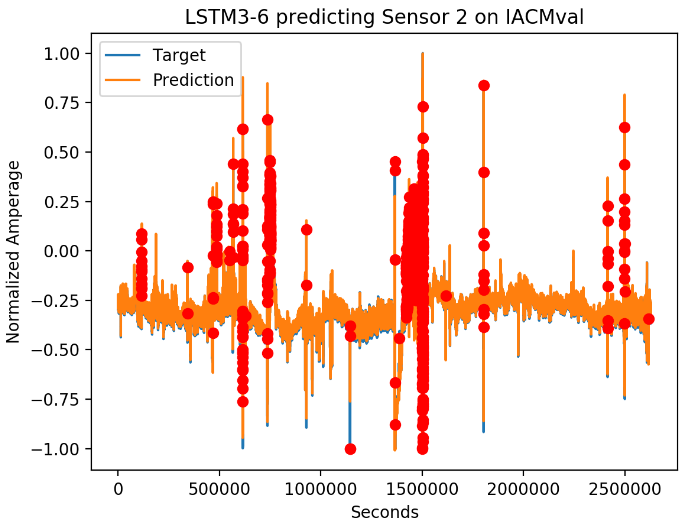


Figure 5.9: The anomalies marked by LSTM$_{3-6}$

5.3.4 HTM

The HTM method is similar to the LSTM/GRU in that it attempts to predict the next value in the time series, and detect anomalies based on the error of the aforementioned prediction. HTM is based on the structure of pyramidal neurons in the neocortex of the human brain. Where the LSTM requires hyperparameter tuning, the hyperparameters of HTM are tuned according to known properties of real cortical neurons, allowing it to perform well on a wide variety of problems using the same set of hyperparameters [14].

The temporal element represents the long term information of what the model has seen over time. In HTM, this information is stored in the active cells within the columns of the SDR. While in the LSTM, long term information is stored in the hidden state. HTM has the characteristic that it discovers temporal patterns very rapidly [14], but our HTM architecture does not appear to have the same capabilities as the LSTM/GRU architectures in learning the anode change shown Figure 5.10. Here you can see how HTM$_{3-1}$ handles the anode change in IACM$_{val}$, marking it as an anomalous event.

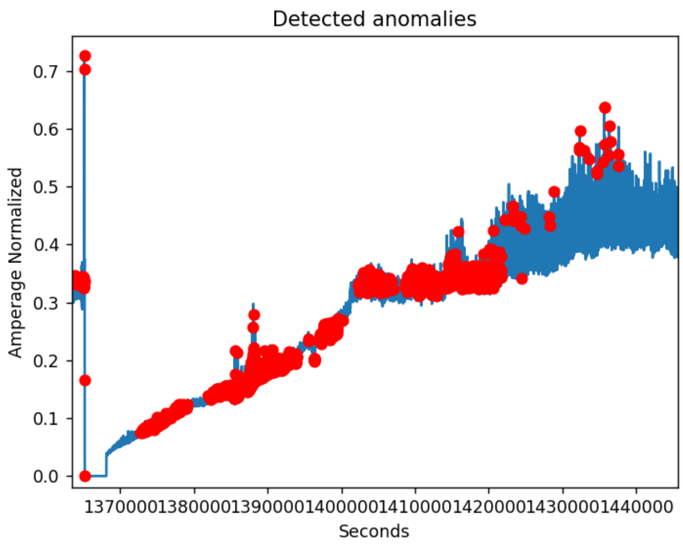
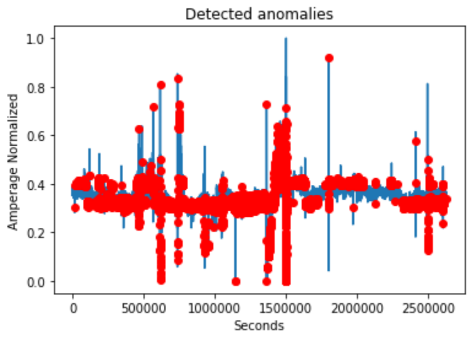
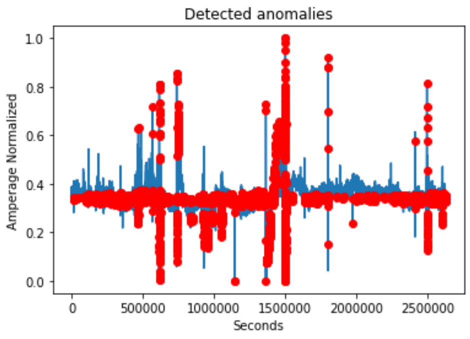


Figure 5.10: The figure shows an anode change and the anomalies outputted from HTM$_{3-1}$, with the red dots indicating anomalies. Several anomalies were detected after the anode change happening at 1.375M seconds.

Figure 5.11 shows a comparison of anomalies detected from HTM$_{1-1}$ with HTM$_{3-1}$ on IACM$_{val}$. The extra months of training data had some effect on the detected anomalies. However, both of the models were very sensitive. Like the YASA with OCSV, HTM was detecting anomalies in areas that look like a normal process. All the detected anomalies have an anomaly score of 1.0, so it would not help to tune down the threshold.



(a) HTM$_{1-1}$



(b) HTM$_{3-1}$

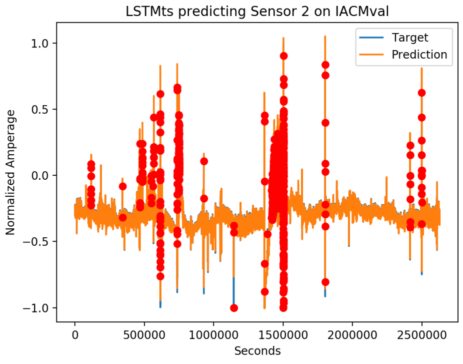
Figure 5.11: Anomalies from HTM trained for one and three mothns. Figure 5.11a shows HTM$_{1-1}$ and Figure 5.11b HTM$_{3-1}$, detecting anomalies in IACM$_{val}$.

For the HTM$_{i-6}$ models, with input from the neighbouring sensors, each input field is encoded and concatenated into one encoding for each timestamp. Even though the NuPIC framework allows to specify the predicted field, the anomaly output is still based on the entire input, making it hard to detect anomalies from one of the sensors specifically. This was not stated in the literature or the GitHub documentation but was something stated by a Numenta community manager on the HTM forum. This makes it hard to test the method of separating sensor anomalies based on shifted neighbours presented in Section 4.8.

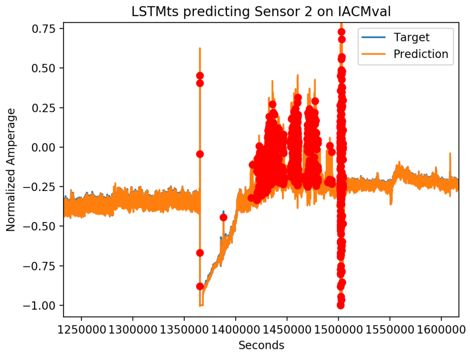
5.4 Sensor anomaly separation

5.4.1 Separation method 1: Data shift

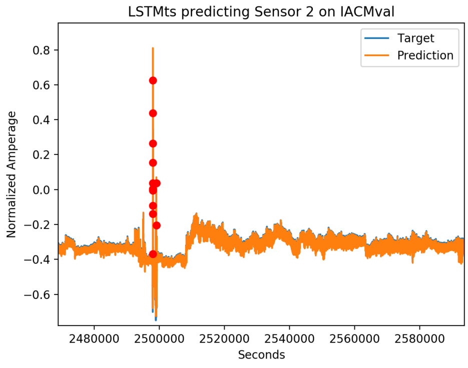
LSTM$_{3-6}$ was retrained with the change described in section 4.8.1, time-shifting neighbouring sensors t+1 and the new model is hereby referred to as LSTM$_{ts}$. If the assumptions presented were correct, we would see a decrease in anomalous areas related to known process failures, such as an anode change in Sensor 2 or related neighbours. Figure 5.12 shows a finished prediction run on IACM$_{val}$, in addition to a zoomed-in view of both the anode change for Sensor 2 and the neighbouring anode change first presented in Figure 5.8. The plot shows clear similarities between the areas marked as anomalous by IACM$_{val}$, and the areas marked by LSTM$_{1-1}$, GRU$_{1-1}$ and LSTM$_{3-6}$ from Section 5.3.3. The visual analysis shows no distinguishable difference between the marked anomalies, and we believe one or more of the assumptions made in Section 4.8.1 to be wrong. Mainly assumption 3. We believe the model might already learn much of the same relationship between neighbouring sensors and the prediction sensor without the extra time shift, making the shift redundant.



(a) LSTM$_{ts}$ on IACM$_{val}$



(b) Anode change

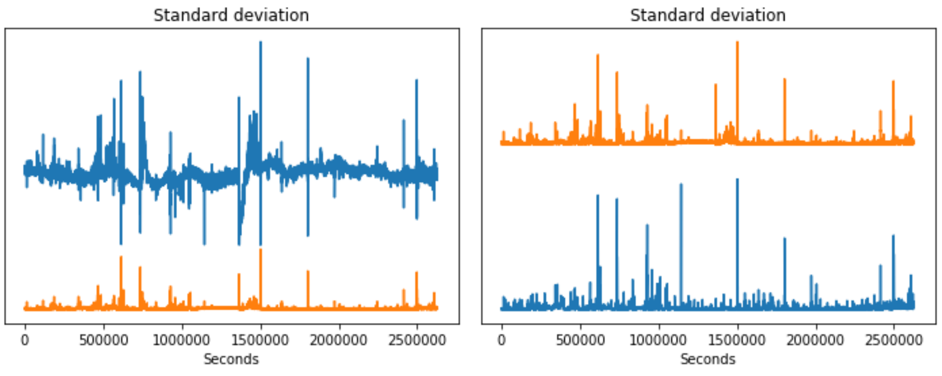


(c) Anode change in Sensor 3

Figure 5.12: (a) LSTM$_{ts}$ on IACM$_{val}$, marking similar areas as previously. No visual difference in the detected anomalies. (b) The anode change event for Sensor 2 in IACM$_{val}$, with an increase in marked anomalies for the process anomaly following the tail of the anode change, circled in 2 in Figure 5.8b. (c) Correctly modelled anode change, as previously shown. Time shifting neighbouring values with t+1 does not seem to increase process anomaly modelling performance.

5.4.2 Separation method 2: Comparing standard deviation

Figure 5.13a shows the normalized standard deviation of a 50 span window around each point in IACM$_{val}$. Figure 5.13b shows the comparison of the standard deviation comparisons from the current from Sensor 2 and the total cell voltage. You can, in many areas, see a clear correlation between the standard deviation in the current and the cell voltage. These are most likely caused by process-related events.



(a) IACM$_{val}$ and std **(b)** Std in current and voltage

Figure 5.13: Standard deviation visualized. (a) The blue plot shows IACM$_{val}$, and the orange plot the standard deviation around each point. (b) shows the standard deviation in the current and the pot voltage, where the current is shown in orange and the voltage in blue. The standard deviation was calculated with a span of 50 around each point.

The anomaly output was filtered using the standard deviation method described in Section 4.8.3. Using this method, around 15 % of the anomalies were filtered out as process anomalies, as the standard deviation in the cell voltage and the current at their specific time stamps both increased enough to breach their thresholds. Figure 5.14 shows how this method separated the anomalies detected by LSTM$_{3-6}$. The thresholds were for this run set to 0.015 for both the standard deviations because it captured the visible peaks you can see in Figure 5.13.

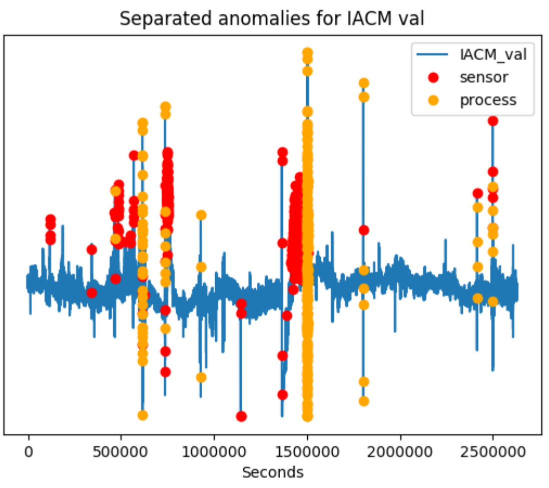


Figure 5.14: This figure shows how the standard deviation method separated the anomalies from LSTM$_{3-6}$ run on IACM$_{val}$. Sensor anomalies are marked as red points, and process anomalies are marked as yellow points.

We suspected the part shown in Figure 5.15, with high deviation and many classified anomalies, to be an event happening in the process since it occurred directly after the anode change. Even though there is an increase in the standard deviation in the current, there is little to no change in the standard deviation of the cell voltage in this area, and thus, this method does not flag them as process anomalies.

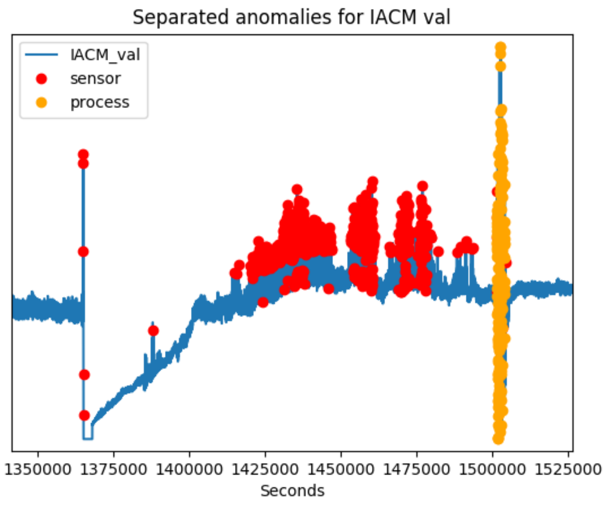


Figure 5.15: This figure shows a zoomed in version of the anode change from Figure 5.14

There are a few limitations worth mentioning. Firstly, the other 39 anodes influence the total cell voltage. If there is an event in another anode causing an increase in standard deviation at the same time as a detected anomaly, this might influence the standard deviation of the cell voltage, causing a wrong classification of the anomaly. Secondly, there could also be sensor anomalies that do not affect the standard deviation around the detected anomaly, but we don't have enough domain knowledge to know the exact nature of sensor anomalies.

Chapter 6

Conclusion and Future Work

6.1 Discussion and Evaluation

The main challenge has been the uncertainties of what constitutes normal data in the IACM data set. Along with examples from domain experts, most of our understanding has been achieved through examining and visualizing the data. The qualitative approach to evaluating the models in the result chapter have been carried out without much expert domain knowledge, and should therefore not be taken too literally.

The data set used for the training and testing were in total four months of data from the IACM electrolysis cell, which is only a minor excerpt of what Hydro have stored in their database. Some segments have been observed to have more variation than others, and it might be that segments of data are more complicated than the methods we tested. As for the quantity of training data, more data is usually better for generalizing. For our models, increasing the size of the models training set from one to three months did not have much of an effect on the detected anomalies. With deeper models, changing the parameters to slow down the learning and better preserve the long term temporal relations, could yield improved results. There is continuous research in the field of aluminium production, and electrolysis technology is continually changing. The data presented in this thesis is from a HAL4000 cell from 2016 and may be subject to change in different cell models.

Most hyperparameters in our models were set manually based on observations from the results of the methods, and are most likely not optimal. By using an event log, a labelled data set or having mathematical definitions of the sensor anomalies, one would be able to quantitatively measure the accuracy with regards to the selected parameters, and tune them accordingly. The sensitivity of the LSTM/GRU and HTM are tuned with thresholds for classifying anomalies. Knowing the nature of the anomalies, one could set these thresholds to balance the rate of false positives and false negatives.

Both LSTM/GRU and HTM are based on the error of comparing an expected result to what is observed. The predicted value can be seen as what the value was expected to look like, given that the system had behaved normally. That the anomaly detection models depend on comparing what is observed to what is expected, contribute to making them understandable, as the expected value can be compared to the observed. These values can be visualized and inspected to see what gave rise to an anomaly warning. If these methods were deployed in the control loops of the electrolysis process, it would give an operator the chance to inspect the deviations of the predictions, and maybe help understand what caused a detected anomaly.

To close the discussion, the research questions presented in the introduction are again revisited to see how far the research has come in answering them. Finally, the research goal is revisited, to see to what extent it has been achieved.

> ***RQ1:*** *How does unique machine learning models perform on the IACM time series data?*

YASA with OCSVM appeared to be a bad fit for our problem. The anomalies detected from this method did not seem reliable, and it would introduce a challenge implementing this for real-time detection, as the segmentation happens recursively, starting from one large segment. The HTM also classified a lot of anomalies in what appeared to be normal regions. A streaming algorithm learning real-time on unsupervised data sounds like a perfect match for our problem, but our experience was that HTM was slow and oversensitive on our data set. The anomalies detected from the LSTM were represented in sections with sudden extreme changes in the measured current, that looked like a deviation from the normal. The LSTM method also managed to learn the pattern of the rising current after the anode change after only one month of training.

> ***RQ2:*** *What implications do these solutions have in regards to detecting and categorizing IACM-sensor anomalies?*

Detecting and categorizing sensor anomalies was done by elimination of process anomalies. If the process shows normal behaviour at the time of a detected anomaly, an anomalous reading is likely caused by the sensor itself.

The inclusion of neighbouring data at time t+1 was tested for the LSTM/GRU method. The idea that the model would be able to forecast anomalies in the process from the neighbours "future" information might have been a bit optimistic and did not have any noticeable effect on the detected anomalies. As this required a lot of training time, the same parameters as the LSTM$_{3-6}$ were used, without much tuning and testing. With the correct depth and hyperparameters, one might be able to model this process relationship if there is one.

The second anomaly method was built as a second layer on top of any anomaly detection method. It seemed to work very well. Comparing the standard deviation of the sensor and the cell voltage showed a clear correlation in areas around detected anomalies. In anomalies from small local process events around the one anode in question, the cell voltage will likely not be affected enough for this method to classify it correctly.

> ***Goal:*** *Detect and categorize sensor-anomalies found in time-series IACM sensor data*

This work cannot claim to have found a method that can detect and categorize sensor anomalies, as we have no way to evaluate their specific performance. The best method of the ones we tested was the LSTM/GRU method separated using the comparison on standard deviation in the time series of current and voltage. It is merely the best of the ones we tested, but we think it showed promising results, and hope it is of value for further research.

6.2 Future work

The lack of synthetic sensor anomaly descriptions or a log over previous faults has made performing anomaly classification hard. For the future, incorporating the use of artificial anomalies if no real signatures are present, should yield promising classification results. In addition, it can help fine-tune the anomaly border parameter used in both baseline and LSTM/GRU anomaly detection. The same can be said for process anomalies, as a description of them would help avoid false positives.

We believe the most significant assumptions for the models trained in this thesis is the assumption that most of the data provided can be said to be *normal* data. Spending the time to clean the data according to a maintenance log over anomalies should give the models a better training foundation and avoid the models learning anomalous behaviour, and thus not mark it as such.

We also believe that future work should encompass further testing and training of the most promising method we have found, LSTM and GRU. A complete regression analysis should give indications on how to improve the models' ability to forecast. Furthermore, we utilized only six sensors out of the 40 available, potentially starving the model of valuable features. Expanding the months available during training may allow for models to pick up on temporal relations not present in the current data sets.

Additionally, the architectural requirements described in Section 3.1.3 applied guidelines to method selection and ranking. Our subjective interpretation of how we ranked the architectures from related work could potentially be skewed, and the three highest-rated architectures might not be the best choice given our problem definition. A reformulation of the architectural requirement may be worth exploring, or a more in-depth exploration of lesser rated architectures.

Finally, exploring the explainability of the models should be a priority. Since the anomalies detected might be analyzed or reported to operators, including the reasoning behind such a marking would be valuable.

Bibliography

[1] , 2020. brf. https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html.

[2] , 2020. sklearn. https://github.com/scikit-learn/scikit-learn.

[3] Ahmad, S., Lavin, A., Purdy, S., Agha, Z., 2017. Unsupervised real-time anomaly detection for streaming data. Neurocomputing 262, 134–147.

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

[5] Angiulli, F., Pizzuti, C., 2002. Fast outlier detection in high dimensional spaces. In: European Conference on Principles of Data Mining and Knowledge Discovery. Springer, pp. 15–27.

[6] Ben-Gal, I., 2005. Outlier detection. In: Data mining and knowledge discovery handbook. Springer, pp. 131–146.

[7] Breunig, M. M., Kriegel, H.-P., Ng, R. T., Sander, J., 2000. Lof: identifying density-based local outliers. In: ACM sigmod record. Vol. 29. ACM, pp. 93–104.

[8] Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR) 41 (3), 15.

[9] Cheng, H., Tan, P.-N., Potter, C., Klooster, S., 2009. Detection and characterization of anomalies in multivariate time series. In: Proceedings of the 2009 SIAM international conference on data mining. SIAM, pp. 413–424.

[10] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

[11] Chollet, F., 2020. Keras. https://keras.io/.

[12] Cocianu, C., 2013. Kernel-based methods for learning non-linear svm. Economic Computation and Economic Cybernetics Studies and Research 47 (1), 41–60.

[13] Colah, 2020. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[14] Cui, Y., Surpur, C., Ahmad, S., Hawkins, J., 2016. A comparative study of htm and other neural network models for online sequence learning with streaming data. In: 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1530–1538.

[15] Das, S., Matthews, B. L., Srivastava, A. N., Oza, N. C., 2010. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 47–56.

[16] dsdeepdive, 2020. Figure of gradient descent. `https://2.bp.blogspot.com/-w3EFnDuIyf4/V1KrbOK944I/AAAAAAAAFOk/BHszQSE-w5Q0i0aQEtIsuoaIclkaDuBowCLcB/s1600/rosenbrock-nag%2Bcopy.png`.

[17] Geman, S., Bienenstock, E., Doursat, R., 1992. Neural networks and the bias/variance dilemma. Neural computation 4 (1), 1–58.

[18] Goldstein, M., Uchida, S., 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one 11 (4), e0152173.

[19] Google, 2020. Tensorflow. `https://www.tensorflow.org/`.

[20] Görnitz, N., Kloft, M., Rieck, K., Brefeld, U., 2013. Toward supervised anomaly detection. Journal of Artificial Intelligence Research 46, 235–262.

[21] Graves, A., Mohamed, A.-r., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp. 6645–6649.

[22] Gunn, S. R., et al., 1998. Support vector machines for classification and regression. ISIS technical report 14 (1), 5–16.

[23] Gupta, S., Ray, A., Keller, E., 2007. Symbolic time series analysis of ultrasonic data for early detection of fatigue damage. Mechanical Systems and Signal Processing 21 (2), 866–884.

[24] Hind, A. R., Bhargava, S. K., Grocott, S. C., 1999. The surface chemistry of bayer process solids: a review. Colloids and surfaces A: Physicochemical and engineering aspects 146 (1-3), 359–374.

[25] Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9 (8), 1735–1780.

[26] Jain, A. K., Mao, J., Mohiuddin, K. M., 1996. Artificial neural networks: A tutorial. Computer 29 (3), 31–44.

[27] Kaiser, Ł., Sutskever, I., 2015. Neural gpus learn algorithms. arXiv preprint arXiv:1511.08228.

[28] Kankar, P. K., Sharma, S. C., Harsha, S. P., 2011. Fault diagnosis of ball bearings using machine learning methods. Expert Systems with applications 38 (3), 1876–1886.

[29] Keras, 2020. Mse. `https://www.tensorflow.org/api_docs/python/tf/keras/losses/MSE`.

[30] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P. T. P., 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.

[31] Khandelwal, S., Lecouteux, B., Besacier, L., 2016. Comparing gru and lstm for automatic speech recognition.

[32] Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[33] Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering.

[34] Kofod-Petersen, A., 2012. How to do a structured literature review in computer science. Ver. 0.1. October 1.

[35] Kolås, S., McIntosh, P., Solheim, A., 2015. High frequency measurements of current through individual anodes: some results from measurement campaigns at hydro. In: Light Metals 2015. Springer, pp. 729–734.

[36] Kvande, H., 2011. Production of primary aluminium. In: Fundamentals of Aluminium Metallurgy. Elsevier, pp. 49–69.

[37] Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., Hoffmann, M., 2014. Industry 4.0. Business & information systems engineering 6 (4), 239–242.

[38] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G., 2016. Lstm-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint arXiv:1607.00148.

[39] Malik, K., Sadawarti, H., G S, K., 2014. Comparative analysis of outlier detection techniques. International Journal of Computer Applications 97 (8), 12–21.

[40] Markou, M., Singh, S., 2003. Novelty detection: a review—part 1: statistical approaches. Signal processing 83 (12), 2481–2497.

[41] Martí, L., Sanchez-Pi, N., Molina, J., Garcia, A., 2015. Anomaly detection based on sensor data in petroleum industry applications. Sensors 15 (2), 2774–2797.

[42] Martí, L., Sanchez-Pi, N., Molina, J. M., Garcia, A. C. B., 2014. Yasa: yet another time series segmentation algorithm for anomaly detection in big data problems. In: International Conference on Hybrid Artificial Intelligence Systems. Springer, pp. 697–708.

[43] Maya, S., Ueno, K., Nishikawa, T., 2019. dlstm: a new approach for anomaly detection using deep learning with delayed prediction. International Journal of Data Science and Analytics, 1–28.

[44] Moonesignhe, H., Tan, P.-N., 2006. Outlier detection using random walks. In: 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06). IEEE, pp. 532–539.

[45] Nan, F., Saligrama, V., 2017. Dynamic model selection for prediction under a budget. arXiv preprint arXiv:1704.07505.

[46] Nanduri, A., Sherry, L., 2016. Anomaly detection in aircraft data using recurrent neural networks (rnn). In: 2016 Integrated Communications Navigation and Surveillance (ICNS). IEEE, pp. 5C2–1.

[47] numenta, 2020. Nupic. `https://github.com/numenta/nupic`.

[48] numenta, 2020. swarming. `https://nupic.docs.numenta.org/1.0.0/guides/swarming/algorithm.html`.

[49] Pandas, 2020. Exponentially weighted windows. `https://pandas.pydata.org/pandas-docs/stable/user_guide/computation.html#exponentially-weighted-windows`.

[50] Pandas, 2020. Pandas ewm. `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.ewm.html`.

[51] Parzen, E., 1962. On estimation of a probability density function and mode. The annals of mathematical statistics 33 (3), 1065–1076.

[52] Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y., 2013. How to construct deep recurrent neural networks. arXiv preprint arXiv:1312.6026.

[53] Peterson, L. E., 2009. K-nearest neighbor. Scholarpedia 4 (2), 1883.

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

[55] Protocol, G., 2016. Global warming potential values. `https://www.ghgprotocol.org/sites/default/files/ghgp/Global-Warming-Potential-Values%20%28Feb%2016%202016%29_1.pdf`.

[56] Rajagopalan, V., Ray, A., 2006. Symbolic time series analysis via wavelet-based partitioning. Signal processing 86 (11), 3309–3320.

[57] Rajasegarar, S., Leckie, C., Bezdek, J. C., Palaniswami, M., 2010. Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks. IEEE Transactions on Information Forensics and Security 5 (3), 518–533.

[58] Ramaswamy, S., Rastogi, R., Shim, K., 2000. Efficient algorithms for mining outliers from large data sets. In: ACM Sigmod Record. Vol. 29. ACM, pp. 427–438.

[59] Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y., 2011. Contractive auto-encoders: Explicit invariance during feature extraction. In: Proceedings of the 28th International Conference on International Conference on Machine Learning. Omnipress, pp. 833–840.

[60] Samanta, B., 2004. Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. Mechanical systems and signal processing 18 (3), 625–644.

[61] Scikit, 2020. Lof scikit. `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html`.

[62] Scikit, 2020. Minmax scikit. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`.

[63] Siegel, S., 1957. Nonparametric statistics. The American Statistician 11 (3), 13–19.

[64] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15 (1), 1929–1958.

[65] Tang, Y., Zhang, Y.-Q., Chawla, N. V., Krasser, S., 2008. Svms modeling for highly imbalanced classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39 (1), 281–288.

[66] Tax, D. M., Duin, R. P., 2004. Support vector data description. Machine learning 54 (1), 45–66.

[67] Tax, D. M. J., 2002. One-class classification: Concept learning in the absence of counter-examples.

[68] Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.-A., 2008. Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning. ACM, pp. 1096–1103.

[69] Wang, L., 2005. Support vector machines: theory and applications. Vol. 177. Springer Science & Business Media.

[70] Watteyne, T., Doherty, L., Simon, J., Pister, K., 2013. Technical overview of smartmesh ip. In: 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. IEEE, pp. 547–551.

[71] Willmott, C. J., Matsuura, K., 2005. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. Climate research 30 (1), 79–82.

[72] Xu, D., Tian, Y., 2015. A comprehensive survey of clustering algorithms. Annals of Data Science 2 (2), 165–193.

[73] Yamanishi, K., Takeuchi, J.-i., 2002. A unifying framework for detecting outliers and change points from non-stationary time series data. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 676–681.

[74] Yin, W., Kann, K., Yu, M., Schütze, H., 2017. Comparative study of cnn and rnn for natural language processing. arXiv preprint arXiv:1702.01923.

[75] Yunqiang Chen, Xiang Sean Zhou, Huang, T. S., 2001. One-class svm for learning in image retrieval. In: Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205). Vol. 1. pp. 34–37 vol.1.

[76] Zhao, J., Liu, K., Wang, W., Liu, Y., 2014. Adaptive fuzzy clustering based anomaly data detection in energy system of steel industry. Information Sciences 259, 335–345.

Anomaly detection for industrial time series

NTNU
Norwegian University of
Science and Technology