A Deep Learning Empowered Framework for Enabling Energy Conservation and Machine Diagnosis via Non-Intrusive Load Monitoring

Andrew Charles Atilla Tanrıöver Connelly

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds

Faculty of Engineering

School of Electronic and Electrical Engineering

September 2023

Intellectual Property

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

List of Publications

Excerpts of this thesis have been peer-reviewed and published in the following as part of the project outcomes:

Chapter 3 is based on the results and reports of the work done in the publication: Connelly, A.C., Zaidi, S.A.R. and McLernon, D., 2023. Autoencoder and Incremental Clustering-Enabled Anomaly Detection. Electronics, 12(9), p.1970. The author contributions are as follows:

Conceptualization, A.C.C. and S.A.R.Z.; methodology, A.C.C. and S.A.R.Z.; software, A.C.C.; validation, A.C.C. and D.M.; formal analysis, A.C.C. and D.M.; investigation, A.C.C.; resources, A.C.C. and S.A.R.Z.; data curation, A.C.C.; writing—original draft preparation, A.C.C.; writing—review and editing, A.C.C., S.A.R.Z. and D.M.; visualization, A.C.C.; supervision, S.A.R.Z. and D.M.; and project administration, S.A.R.Z. All authors have read and agreed to the published version of the manuscript.

Chapter 4 has been accepted for presentation and publication in the 2023 IEEE International Smart Cities Conference (ISC2), track 6-IoT and Smart X Services, under the title: LSTM-Powered Point Process Modelling for Idle State Detection in Cyclic Energy Data. The submission is currently in review. Connelly, A.C., Zaidi, S.A.R. and McLernon, D.

Conceptualization, A.C.C. and S.A.R.Z.; methodology, A.C.C. and S.A.R.Z.; software, A.C.C.; validation, A.C.C. and D.M.; formal analysis, A.C.C. and D.M.; investigation, A.C.C.; resources, A.C.C. and S.A.R.Z.; data curation, A.C.C.; writing—original draft preparation, A.C.C.; writing—review and editing, A.C.C., S.A.R.Z. and D.M.; visualization, A.C.C.; supervision, S.A.R.Z. and D.M.; and project administration, S.A.R.Z. All authors have read and agreed to the published version of the manuscript.

Chapter 5 is, at the time of writing, drafted for publication under the title: Multi-Classifier for Fault Prognosis of Electrical Appliances using Gradient Boosting. Connelly, A.C., Zaidi, S.A.R., McLernon, D, and Mhamdi, L.

Conceptualization, A.C.C. and S.A.R.Z.; methodology, A.C.C., S.A.R.Z., and L.M.; software, A.C.C.; validation, A.C.C. and L.M.; formal analysis, A.C.C. and L.M.; investigation, A.C.C.; resources, A.C.C. and S.A.R.Z.; data curation, A.C.C.; writing—original draft preparation, A.C.C.; writing—review and editing, A.C.C., S.A.R.Z. and D.M.; visualization, A.C.C.; supervision, S.A.R.Z. and D.M.; and project administration, S.A.R.Z. All authors have read and agreed to the published version of the manuscript.

Abstract

Despite the immense success and powerful capability of machine learning and its subsets, there remain areas in which the technology has not been as thoroughly researched. The recent advent of Industry 4.0 has enabled new applications of machine learning - deep learning in particular, which require a vast amount of training data. The objective of this thesis is to investigate the more recent innovations in machine learning in the field of smart meter data through non-intrusive load monitoring (NILM): a technique for analysing gradual change in the energy draw and deducing what is used and how. Specifically, we explore machine learning-enabled challenges toward the health monitoring and proactive repair of electrical appliances, increasing the operational lifetime and inherently privacy-preserving. Machine learning can allow us to identify hidden indicators in data that the electrical appliance is deviating from its known, normal working pattern. Our dataset originates from retrofitted power outlets with metering functionality, and we are looking to investigate similar techniques based on energy consumption alone. Many existing techniques involved in IoT condition monitoring enjoy access to feature-rich sensor data, with a large basis of data on which to train. The research within is a natural complement to the data already collected by the smart appliances we are to support. This thesis explores the key challenges in implementing machine learning algorithms and the lack of research on those that look at appliances with cyclic load patterns (e.g., laundry appliances or dishwashers) before offering proposed discoveries. This thesis proposes and evaluates two deep learning algorithms and one ensemble-based machine learning algorithm in solving three distinct challenges.

The first proposed model aims to identify anomalous behavior in the power signatures of household electrical appliances using an incremental clustering algorithm which first classifies the cycle, then trains an autoencoder to reconstruct the signature. The second model in this work aims to predict the idle time until the next usage following a period of activity of an electrical appliance, which can inform other systems to conserve power, prolonging the appliance. The model is based on temporal point processes, a classical statistical field. We architect an LSTM neural network capable of outputting direct time deltas. Finally, we look to identify specific faults in an appliance, this time knowing more of its nature, based on known failure conditions. We propose a gradient boosting model to classify a machine failure. Both anomaly-related models competently account for the expectedly skewed performance in the literature due to a steep imbalance in the data.

Acknowledgements

Foremost, I would like to thank my supervisor Dr Syed Ali Raza Zaidi for his trust, accommodation, and guidance as I pursued my research into—at the time—the unknown. Dr Zaidi's support has been key in turning the daunting prospect of a doctorate into a deeply satisfying and fruitful endeavour. I would like to extend this thanks to my co-supervisor Dr Des McLernon for his guidance, deep expertise, and support in exercising my research and scientific capabilities.

My next thanks go towards both UK Research and Innovation (UKRI) for the Industrial CASE EPSRC studentship, as well as the University of Leeds for what has been the most fascinating and rewarding research endeavour on which I have ever embarked. To have worked on industrial matters and have innovated on a small part in the United Kingdom has been nothing short of a pleasure.

Finally, a special word of thanks to my family for grounding me in reality and constant reminders of your unwavering love and of of the world out there.

Contents

List of Figures

List of Tables

Nomenclature

0.1 Notation

In general, matrices are denoted by upper case letters (X), vectors by lower case bold letters ($\mathbf{x}$), and scalars by lower case italics (x). Below are the definitions of several symbols used within the body of this thesis.

$\mathbf{a}$	Activation vector
$\mathbf{b}$	Bias vector
$\circ$	Output of left operand pipes into right operand
f	Activation function
$\dot{f}$	derivative of activation function
K	Count of clusters
t	Time series observation
θ	Model parameter set (encapsulating weights and bias)
$\sigma(\cdot)$	sigmoid function
X	Matrix of data instances
x	Data instance
y	Target
$\hat{y}$	Output

0.2 List of Abbreviations

A	Amps
ADASYN	Adaptive synthetic (oversampling)
AE	Autoencoder
AI	Artificial Intelligence
API	Application programming interface
ARIMA	Autoregressive integrated moving average
ANN	Artificial neural network
CE	Cross-entropy
CNN	Convolutional neural network
CSV	Comma-separated values
DART	Dropout meet multiple additive regression trees
ECG	Electrocardiographic
ENN	Edited nearest neighbour
ETL	Extract, transform, load
GRU	Gated recurrent unit
HMM	Hidden Markov model
IoT	Internet of Things
KL	Kullback–Leibler
KNN	K-nearest neighbours
LSTM	Long short-term memory (network)
MAE	Mean absolute error
ML	Machine Learning
MSE	Mean squared error
NILM	Non-intrusive load monitoring

PAM	Partition around medoids
PII	Personally identifiable information
RegEx	Regular expression(s)
ReLU	Rectified linear unit
RMSE	Root mean squared error
RNN	Recurrent neural network
RUL	Remaining useful life
ROC	Receiver operating characteristic
SGD	Stochastic gradient descent
SMOTE	Synthetic minority oversampling technique
SVM	Support vector machine
TSDB	Time series database
XGB	Extreme gradient boosting
V	Volts

Chapter 1

Introduction

1.1 Project Background

The means to our NILM solution come from a retrofitted, IoT-connected BS 1363 smart plug, running on bespoke firmware and powered by an ESP32-S3 microcontroller. The plug is capable of remote operation and scheduling, controllable via an in-house cloud, and is equipped with a metering chip that records and transmits its energy readings at minute intervals, as well as into the ESP32, equipped with the capability to run machine learning algorithms on edge. This serves as our target platform for the models described in this thesis.

This research is the product of an industrially-driven project with a requirement to implement NILM on the data collected from smart plugs. Several key industrial stakeholders are exploring the potential of such technologies deployed retrospectively on appliances without smart functionality. The energy data collected from several thousand plugs over the years by the industrial partner remains unexplored from a NILM perspective. The work looks to develop a suite of machine learning technologies for this data to monitor the usage, health, and likelihood of failure of an electrical appliance based on its energy readings in real time. Development of an enhanced version of the smart plug capable of capturing 50 energy readings per second as opposed to 1 per minute was underway throughout this research, which we eventually received. This new frequency should offer greater precision in the behaviour analysis of the connected appliance than the existing archives, but it must be investigated as to whether and how the archived readings are best utilised in this machine learning pipeline for the new hardware.

1.1.1 Objectives

This project looks to develop a comprehensive, intelligent suite of machine learning solutions to gain a new depth of understanding in appliances, from failure detection to energy conservation. The primary objective of this project is to develop an overall system capable of recognising various appliances and their behaviour, then identifying when such an appliance is developing faults or is likely to fail based on deviation from expected behaviour. The objectives based on this goal are defined as:

1. Develop a deep learning framework for monitoring laundry appliance energy consumption and identifying anomalous behaviour, relying on power consumption without the use of sensor data in accordance with NILM, yet still be able to account for multistate devices;

2. Develop a deep learning empowered framework for deriving energy use patterns of a variety of appliances and deciding when to shift the appliance into a low power mode for energy conservation in a NILM context, using only power consumption;

3. Develop a machine learning framework for predictive maintenance in the context of laundry appliances, capable of recognising both normal operating healthy states and classifying known modes of failure from a dataset markedly low in features and volume.

1.2 Origination and Definition of Dataset

Our dataset originates from a cloud time series database of energy data collected by the industrial partner with readings of the power, voltage, and current readings of each plug, sampled each minute. The partner is equipped with the means to continue developing this dataset it in far richer detail, yet currently has no way of leveraging it for their customers.

Data collected for each plug are treated as time series, T: an ordered sequence of scalar data points $T = (t_0, t_i, \ldots, t_n), t_i \in \mathbf{R}$ indexed by fixed time intervals [1]. This multivariate time series data encapsulates 3 distinct scalar features with each observation, therefore $t_i = X = (x_0, x_j, \ldots, x_m)$. How each variable is best utilised is a decision made by both the programmer and the model. Time series is a prevalent data format across multiple sectors, including medicine, finance, and industry. Applying machine learning to this has been a large field in research, for example, [2] developed a convolutional neural network (CNN) to detect heart arrhythmias in electrocardiographic (ECG) time series data, which outperformed human cardiologists.

Little work has been done in combining prognostic health monitoring of appliances with machine learning to run on edge devices with typical IoT constraints (e.g., storage, resources, processing power). This renders much of the existing research unsuitable for such an application, as vast training data and computationally expensive operations are typical in training neural networks; many devices are simply unable to support such requirements. The dataset that motivates this thesis is entirely unlabelled, making unsupervised learning a core focus.

The data is formatted as temporal energy data recorded at minute intervals, which points the overall direction of this thesis toward time series. However, much of the work cannot rely on the offline availability of this data. In deployment, models are evaluated on the continuous, unseen arrival of new data and must run their predictions on experience and history alone before receiving the next input. Further, the nature of time series data, with energy/sensor readings in particular, is of high velocity and volume, which makes it difficult to store onboard edge hardware (the target platform for this work). To overcome this, relevant models must train themselves in an online fashion, using the latest data point—or a small reservoir of previous readings.

The first stage in developing our dataset is to extract and process the energy data from the private cloud of the industrial partner into a local document-based dataset, before transforming it into data frames for a model at runtime. Energy readings are stored on cloud infrastructure using InfluxDB: a specialised database for time series data. The open-source database project allows for a more efficient population and interrogation of timestamped, sequential data and cites IoT sensor data as a core supported use case. One row of data consists of a timestamp (usually in milliseconds), a corresponding value—an integer, floating point, string, or boolean—and a series of optional tags used for grouping [3]. Beyond highly optimised storage and write performance, key user attractions of time series databases (TSDB) include range queries and aggregations [4], which make for easier interrogation and analysis.

Due to the cloud infrastructure, identifying what and where to query from the InfluxDB realm cannot be determined from within InfluxDB itself. Data concerning smart plugs themselves are housed separately inside a traditional relational database. Their energy readings are bundled as a metering service, containing a device-specific identifier for each axis of energy reading, referred to internally as a ‘characteristic’: voltage, current, and wattage, all expressed in 64-bit floating point. Querying their private application programming interface (API) for devices with a user-defined name containing either ‘washing’ or ‘laundry’ returned 15 plugs and their corresponding characteristic identifiers.

For a more efficient development workflow and processing, the data is pulled down in its entirety onto a local disk and simulated as a stream, where each reading is received incrementally as input to models. These identifiers are grouped with the device name and a unique identifier in a row of a five-column output CSV file for further processing,

```
Timestamp,Wattage,Current,Voltage
2019-05-24T18:10:00Z,246.8125,0.22265625,40.0
2019-05-24T18:11:00Z,247.375,0.265625,41.0
2019-05-24T18:12:00Z,246.96875,0.30078125,29.0
2019-05-24T18:13:00Z,246.15625,0.11328125,7.0
2019-05-24T18:14:00Z,239.25,9.30859375,2000.0
...
```

Figure 1.1: Head of a sample of a laundry machine cycle once all featured aggregated.

exampled in Figure 1.1, with no relationship to the personal user account of the device. Although a more convoluted route to data generation, it requires only a single API request to the cloud, conserving bandwidth and egress costs during these experimental stages.

To begin extracting raw energy data, each energy characteristic of a device is queried for the graph of its entire history, and each data point within the graph is appended to an in-memory dictionary under its title. Eventually, one entry in the dictionary will be keyed by an observation timestamp, along with an array of the three characteristic readings as the corresponding value.

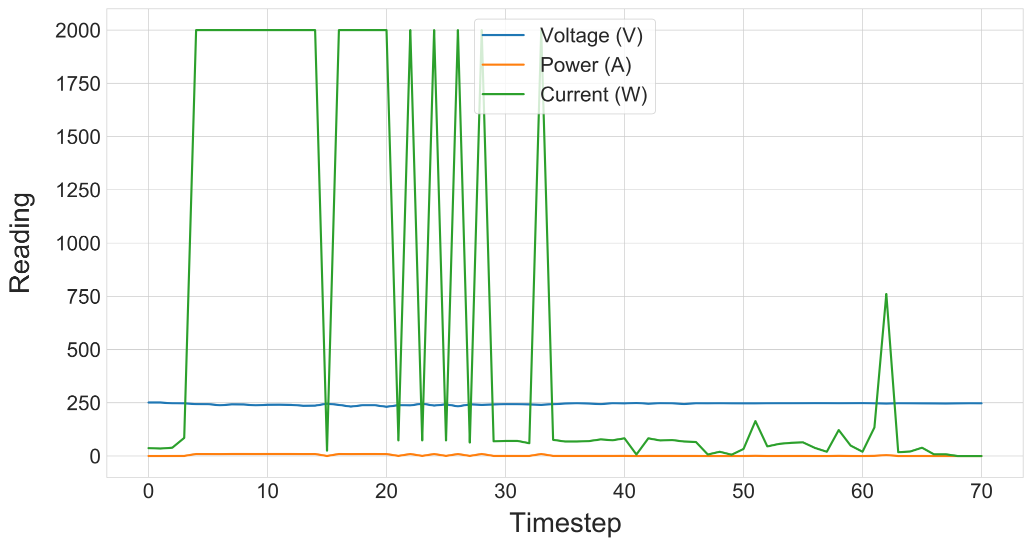


Figure 1.2: Power consumption graph of an isolated laundry machine wash cycle.

The historical power consumption time series must be extracted into individual samples as input data for a model, which begins with light manual processing. We observe that when the appliances are inactive, the smart plugs themselves tend to idle just below 0.04A. We carry this value as a minimum threshold and consider any readings above as metered activity, in which case it is appended to the sample in situ. Once the readings bottom out at the 0.04A threshold for 5+ consecutive minutes, we trim and export the collection as an individual sample.

```
Timestamp,Wattage,Current,Voltage
2019-09-12T10:13:00Z,243.5,0.07421875,10.0
2019-09-12T10:14:00Z,245.1875,0.07421875,11.0
2019-09-12T10:15:00Z,244.859375,0.0703125,11.0
2019-09-12T10:16:00Z,242.125,3.3359375,473.0
2019-09-12T10:17:00Z,242.078125,3.32421875,472.0
2019-09-12T10:18:00Z,242.921875,3.375,500.0
2019-09-12T10:19:00Z,243.8125,3.375,486.0
2019-09-12T10:20:00Z,241.328125,3.34375,498.0
2019-09-12T10:21:00Z,242.359375,3.375,502.0
2019-09-12T10:22:00Z,241.5625,3.296875,476.0
2019-09-12T10:23:00Z,241.015625,3.3515625,507.0
2019-09-12T10:24:00Z,241.75,3.3359375,493.0
2019-09-12T10:25:00Z,242.953125,3.40234375,509.0
2019-09-12T10:26:00Z,243.09375,3.38671875,510.0
2019-09-12T10:27:00Z,243.140625,3.30859375,478.0
2019-09-12T10:28:00Z,243.390625,3.36328125,490.0
2019-09-12T10:29:00Z,243.125,3.421875,522.0
2019-09-12T10:30:00Z,245.015625,3.46484375,557.0
...
```

Figure 1.3: Head of a sample of a laundry machine cycle.

As readings are at a fixed minute interval, two or more consecutive idle readings will write out the sample into a four-column CSV, including the timestamp, wattage, current, and voltage as illustrated in Figure 1.2. The head of an example wash cycle is shown in Figure 1.3.

1.3 Contributions

The overarching basis for this research is to derive as much actionable insight from the online, evolving dataset we are presented with on each plug. From this task, we begin to form our contributions on the nature of what we can feasibly achieve with such a dataset.

Contribution 1: We first propose an unsupervised anomaly detection system that operates on a per-cycle basis of a single white appliance (e.g., laundry machine or dishwasher), without prior knowledge of the machine make or model. For example, the signature of one machine's eco cycle is considerably different from another, thus eliminating the possibility of a universally trained model, despite access to a substantial energy consumption dataset. Our first challenge is to address this with a design able to differentiate between known, similar, and unknown wash cycles. Using a novel adaptation of partition clustering and appropriate preprocessing, the algorithm designed operates on an evolving dataset, compared to conventional clustering algorithms that rely on full access to the dataset. We must then be able to predict normal behaviour as a benchmark for anomalies without known anomalous samples in our dataset. To achieve this, the cycle is then submitted to an ensemble of autoencoders for reconstruction, which swaps its training parameters ad hoc depending on the cluster matched to the cycle. The key attraction of this approach is being able to account for operation cycles that differ in nature, helping prevent the system from falsely classifying the normal behaviour of one cycle as the anomalous behaviour of another.

Contribution 2: Using the same dataset, we look to explore the concept of interval prediction to estimate the next time to use, allowing us to conserve power amid global energy challenges. We design a second system to accept meter readings, this time to predict the interval between the current time and its next use. With the result of this, the system can potentially be powered down to conserve energy and lifespan. The system developed encloses an online load-sensing machine learning model capable of adapting to new environments. Leveraging deep learning, our approach relegates all aspects of classical temporal point processes to a recurrent neural network. Although there is existing work in this field, no proposed solution we have investigated to date fully addresses the unique nature of our challenges. Unlike the majority of solutions that work in likelihoods of occurrence, ease of interpretability is a core consideration in the design of this model, our most significant contribution offers a method to sample intervals directly from the model. As usage patterns change, the model must be recalibrated to maintain its accuracy and relevance to the user. As new data is collected, the model must be retrained or adjusted to account for changes in patterns, behaviours, or environmental factors.

Contribution 3: Returning to the idea of health and increasing the lifespan of an appliance in the anomaly detection space, we finally look toward the field of predictive maintenance and bringing it from typically heavy industrial machinery to domestic appliances. Through remote diagnosis of appliance failure or even normal wear and tear, manufacturers can heavily optimise service operations to the benefit of immense cost savings. Here, we suggest an approach toward conditioning highly imbalanced time series data of markedly small size into an efficient gradient boosting model, capable of running on edge hardware. Unlike rich sensor data, as with the majority of similar work, our model performs solely on statistical features of energy data from a retrofitted power supply. This model was architected and trained under a dataset of hourly readings to serve as a basis for comparison. We then adapt the same model for the similar purpose of fault detection to a new, more intricate dataset.

1.4 Thesis Outline

Chapter 2 offers a background review of the technologies concerned in this project, primarily deep learning. The chapter begins with the concept of machine learning and how deep learning advances it, before explaining the core principles of deep learning. More sophisticated variants of traditional multilayer perceptron networks (including recurrent neural networks and autoencoders) with more specific use cases are ideal for this project. This chapter also reviews existing work relevant to our challenges and approaches. We find some areas are well researched and their approaches well evidenced, however, many are simply not viable solutions when evaluated against our problems.

Chapter 3 documents the first framework developed to adaptively learn from a cycle-based appliance and detect faults in a cycle. The framework addresses the challenges of grouping cycles, both normal and anomalous, into individual datasets through a novel incremental clustering algorithm, which is trained and executed in parallel using a set of autoencoder neural networks working in ensemble, each running on a distinct feature mapping.

Chapter 4 describes the architecture and approach toward the second model developed as part of the energy conservation goal. A series of LSTM neural networks are employed in an online fashion to predict the interval between the last cycle and the time to the next. Higher-level software can use this prediction to power the appliance down into a low-energy state. Traditional LSTM solutions are explored before looking toward temporal point processes, which have seen recent interest in modelling them with neural network techniques.

Chapter 5 documents the work of the third model: a gradient boosting-enabled system designed to predict when a machine is likely to fail and designate the failure to one of four known root causes. Unlike other models, this particular model has access to labelled data and is, therefore, a supervised learning problem from the onset. We process the data into moving windows, which the model is fed (along with the state—normal operating or type of failure). The work is then adapted to be compatible with our new high-frequency data. Within, changes to both the model architecture and the data preprocessing stages are detailed.

Chapter 6 then concludes the thesis, summarising the work of the previous chapters, and the contributions made, and examines potential routes for future work and further development.

Chapter 2

Background

This chapter presents some background literature and existing work on deep learning techniques, before turning toward those more concentrated on time series data. Section 2.1 introduces the core concepts behind machine learning, which Section 2.2 elaborates on with a focus on deep learning core foundations, its uses and evolution, and its suitability for the goal of this project. Sections 2.3 and 2.4 review specialised variants of neural networks better suited to handling sequential data, e.g., time series, and how they are all scored in Section 2.5. Section 2.6 covers the challenges faced in the vast and rapidly developing field of deep learning, and how others have attempted to overcome them. Sections 2.9 and 2.10 review other technologies explored in this project.

2.1 Introduction to Machine Learning

Machine learning empowers machines to approximate patterns and correlations between data points and their labels, leveraging predictions from past data points as experience to train its accuracy for the next with as minimal human intervention as possible. Such a task is known as supervised learning when working with labelled data [5]. Based on these training sets, the model trains to derive and utilise the underlying structure of the data.

A machine learning model is a trained representation of the process used to map the input data into an output, regardless of shape. Within most models exists a series of parameters used to form and refine this mapping, which must be learned repeatedly over data for optimal performance [6]—a process known as training. In this stage, the performance of a model is measured in its ability to evaluate against data withheld from the training process. Having to reserve a portion of the overall dataset to quantify this is one of the long-running challenges of machine learning, as models require an abundance of data to learn from and generalise well. Collecting sufficient volume and variety of data can be difficult, though several well-researched techniques can help to compensate for lacking or imbalanced data, such as cross-validation and data augmentation (particularly useful when working with image datasets).

Certain intelligent technologies, such as natural language and computer vision, are difficult to program manually, more so with sufficient generalisation for a production, real-world environment. Machine learning has demonstrated extraordinary capability in many such tasks, often outperforming leading manual implementations in the fields concerned with relative ease. A recent example covered in mainstream media is Google DeepMind's AlphaZero, a chess-playing deep reinforcement neural network model, which outperformed world champions in the strategy board games of chess and shogi [7].

2.1.1 Supervised Learning

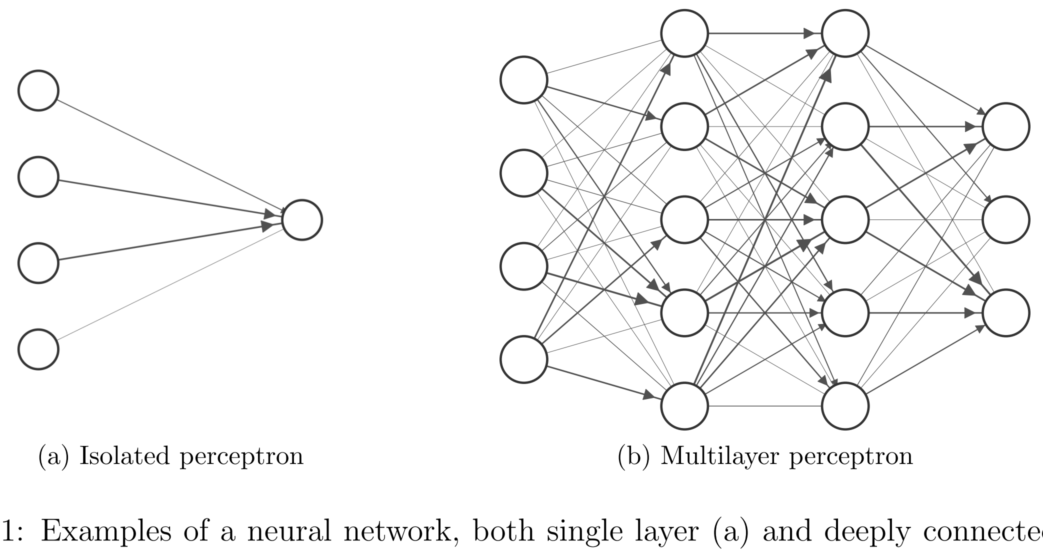
Supervised learning matches a series of example inputs with one or more corresponding outputs, also referred to as targets. These outputs can be well-defined labels in a form known as classification, e.g., a type of failure in a machine. Alternatively, the desired outputs may be continuous, e.g., a delay interval or time to failure, known as regression. The resurgence and success of deep learning solutions are thanks to supervised learning challenges and the recent computation power, as well as the data availability that followed, have enabled. Deep learning requires vast training datasets to attain adequate performance, which even today are often difficult to procure and annotate until the advent of big data.

2.1.2 Unsupervised Learning

Unsupervised learning concerns data where targets (or *labels*) are unidentified or simply unavailable. Like supervised learning, the objective is to discover underlying existing patterns in the data. Perhaps the most common use case of unsupervised learning is clustering, which organises a dataset into clusters of similar, statistically coherent examples [8]. This thesis explores combinations of supervised and unsupervised approaches to machine learning in classification and regression. Concerning deep learning specifically, its renewed interest comes from the success of promising techniques of autoencoder architectures and generative modelling [9], used to capture coherent latent representations of data.

2.2 Introduction to Deep Learning

2.2.1 Layered Perceptron



(a) Isolated perceptron (b) Multilayer perceptron

Figure 2.1: Examples of a neural network, both single layer (a) and deeply connected layer (b). Arrowhead density indicates the strength of the synapse (i.e., connection weight).

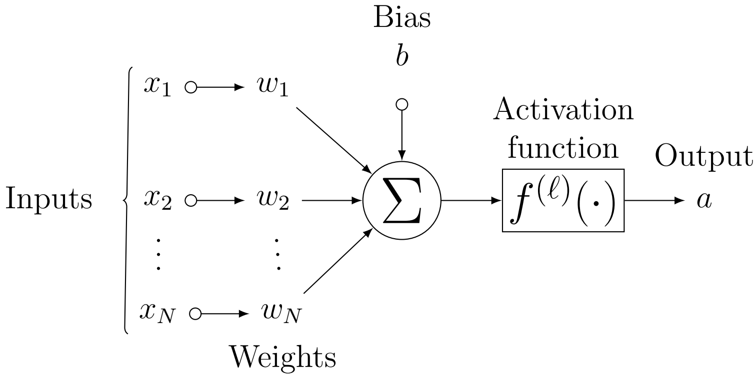


Figure 2.2: A classic neuron isolated from part of a larger network. Inputs can be given as either from the input data at layer 0, or activations of the previous layer for all subsequent layers.

Neural networks, loosely inspired by the neural architecture of the human brain, are implemented as a network of interconnected functional elements, each with multiple inputs mapping to one output. Figure 2.2 illustrates a neuron in its most basic form: a series of inputs $\mathbf{x}$ fed individually from $1, 2, \ldots, N$, each with a corresponding weight w. This neuron is a computational unit that produces from its weighted inputs, along with a separately trainable bias term $b_j^{(\ell)}$ [10], a non-linear output guaranteed from passing through an activation function $f(\cdot) : \mathbb{R} \to \mathbb{R}$.

Consider layer ℓ of an L-layered neural network with N_ℓ neurons $\mathbf{x}_1^{(\ell)}, \mathbf{x}_2^{(\ell)}, \ldots, \mathbf{x}_{N_\ell}^{(\ell)}$, each with some differentiable activation function $f^{(\ell)}(\cdot)$. The inputs to layer ℓ are given by the preceding layer $\ell - 1$, and each neuron $x_j^{(\ell)}$ is weighted by weight matrix $\mathbf{W}$, whose elements are given by $w_{ij}^{(\ell)}$, for $i = 1, 2, \ldots, N_{\ell-1}$ and $j = 1, 2, \ldots, N_\ell$. A trainable offset, or bias $\mathbf{b}$ is maintained alongside the model weights. The net input to $x_j^{(\ell)}$ is given by

$$n_j^{(\ell)} = \sum_{i=1}^{N_{\ell-1}} x_i^{(\ell-1)} w_{ij}^{(\ell)} + b_j^{(\ell)}, \quad j = 1, 2, \ldots, N_\ell. \tag{2.1}$$

Where $\ell = 0$, $x_j^{(\ell)}$ is simply the j-th element of the input data itself. Where $\ell = L$, the net activation is the final output y for the network. The activation $a_j^{(\ell)}$ of neuron $\mathbf{x}_j^{(\ell)}$ is given by

$$a_j^{(\ell)} = f^{(\ell)}(\sum_{i=1}^{N_{\ell-1}} n_i^{(\ell-1)} w_{ij}^{(\ell)} + b_j^{(\ell)}). \tag{2.2}$$

The summation is passed to an activation function, which transforms the weighted inputs to produce the end firing intensity (i.e., activation) $a_j^{(\ell)}$ of the neuron.

The activation result $0 \geq a_j^{(\ell)} < 1$ is received as input by one or more neurons of the next layer. The dense connections between many of the neurons as shown in Figure 2.2, where the output of one layer serves as input to another, form a neural network. Layer $\ell = 0$ is considered the input layer and is sized according to the shape of the input data, given as its initial activations at each execution. Likewise, layer L is considered the output layer, and its final activations are considered its prediction and in its evaluation. Any layers between these are known as hidden layers, where the *learning* takes place.

Training and label vectors $\mathbf{x}$ and $\mathbf{y}$ are presented individually for processing through the network. Using t as the time step of the training process, input vector $\mathbf{x}_t$ propagates through the network per the previous equations, eventually outputting $\hat{\mathbf{y}}_t$. In our application, we derive our label vectors from the next reading, which the model is tasked with predicting.

Following a complete forward propagation through the network, a loss function measures the deviation between the model's prediction $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$, i.e., target. As with activation functions, several loss functions have been developed in response to varying challenges, and each particular loss function is characterised by different measures and interpretations of what exactly constitutes loss. An example of loss functions includes Root-Mean-Square Error (RMSE) given in Equation 2.9. This particular loss function is well suited when even a few large error margins are unacceptable in machine learning applications, as its penalisation of these major errors can significantly affect the overall RMSE score [11]. This lends particularly well in the anomaly detection space and we leverage RMSE in the evaluation of our first model in Chapter 3.

$\mathbf{W}$ and $\mathbf{b}$ of each layer ℓ are updated individually according to their degree of responsi-

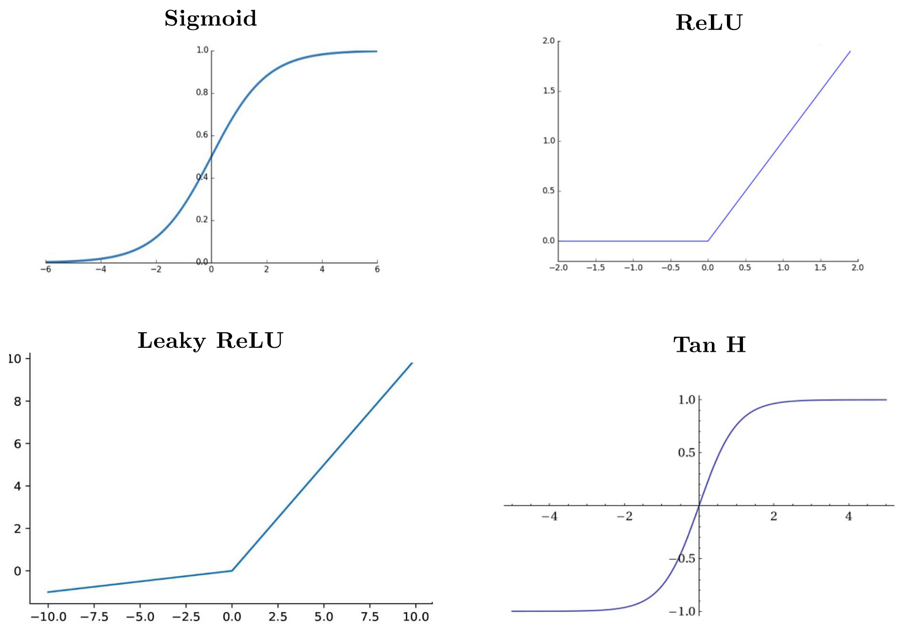


Figure 2.3: Plots of common activations functions. The x-axis represents the given input and its corresponding activation is given on the y-axis. Inputs are clamped to the bounds of the axes.

bility for the overall loss. This responsibility for loss is calculated using the backpropagation algorithm. Readers interested in more about backpropagation and the training processes of artificial neural networks are directed to [12, 13] and the references therein. This gradient is next multiplied by a learning rate $\alpha \in [0, 1]$ for the network: a shared hyperparameter that governs the step size taken in converging toward ideally the global minima. Although the learning rate is shared in this implementation, the weight and bias updates can be driven by independent learning rates. Choosing a value for this hyperparameter can significantly impact the performance of the model. The learning rate should be small enough to converge toward the minima, but not so excessively small that it may become trapped in a suboptimal minima [14]. Likewise, too high a value may cause the model to continually overshoot [15] and the weights may never settle towards convergence.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.3}$$

$$R(x) = \max(0, x) \tag{2.4}$$

$$S(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.5}$$

$$tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{2.6}$$

The most common activation functions include sigmoid (σ), rectified linear unit (R) commonly abbreviated as $ReLU$, softmax (S), and hyperbolic tangent ($tanh$). These activation functions, illustrated in Figure 2.3, are expressed respectively in Equations 2.3 through 2.6. The result of the preactivation sum is then passed to an activation function, which transforms the weighted inputs $z_j^{(\ell)}$ to produce the end firing intensity (i.e., activation) $a_j^{(\ell)}$ of the neuron.

Selecting an activation function for the layer depends on the position and purpose of the layer in the network, as each activation function contributes independently to the overall performance of the network. For example, ReLU tends to perform better in feature space, converging several times faster than equivalents using otherwise [16]; softmax is better suited to output layers of multi-label classification models as it enforces a probability-like outcome, summing to 1. Softmax (Equation 2.5) accepts a vector as input, as opposed to a single value, given its use for probabilistic output.

ReLU and its variants are particularly favoured in modern deep learning endeavours for their consistently high performance in a broad range of applications [17] and their immunity to the vanishing gradient phenomenon. It plays a significant role in mitigating the vanishing gradient problem bu its nature of providing a linear response for positive inputs and clamping negative inputs to 0. In anomaly detection systems, ReLU is a frequent choice of activation function in combination with autoencoders [18, 19]. Further, its efficient runtime computation and gradient propagation characteristics suit our purposes well, working in low-compute environments with scarce data where faster convergence is a strong requirement.

Also illustrated in Figure 2.3 is Leaky ReLU, a variant that attempts to counter the "dying ReLU" problem whereby neurons with negative bias may never activate, ensuring all neurons part of the network may contribute to the final output regardless of their sign.

Connecting multiple perceptron systems into layers, as illustrated in Figure 2.1 (b) can produce higher-order, non-linear decision hyperplanes, which often better generalise around more diverse, complex datasets in a more automated fashion. These intermediate *hidden* layers advance a model from logistic regression to a neural network.

A neural network comprises an input and output layer (sometimes known as the logits layer in classification problems), with multiple hidden layers between. This design is referred to as a feedforward neural network, or classically, a multilayer perceptron. The architecture is named feedforward for its unidirectional data flow, as there is no feedback loop between the outputs and inputs of one part of the model to the next [8]. A deep neural network comprises multiple hidden layers, which eventually map the feature space of a problem based on the output of the last, thereby relegating the task of feature engineering to the model, traditionally undertaken by human domain experts. These hidden layers can be fully connected (i.e., *dense*), stacked sequentially, and comprised of different counts of neurons. Figure 2.1 (b) illustrates this concept of a neural network comprised of four input features, two hidden layers, and a final output layer of three output neurons.

Layer $\ell = 0$ is considered the input layer, sized of N components and producing activations $a_i^{(0)} = x_i, i = 1, 2, \ldots, N_0$. N should equal the size of the input vector $\mathbf{x}$. Similarly, the output layer of M components should match the output vector $\mathbf{y}$, given by $y_j = a_j^{(L)}, j = 1, 2, \ldots, M$.

2.2.2 Measuring Network Loss: Minimising Risk

Training and label vectors $\mathbf{x}$ and $\mathbf{y}$ are instances of the training set, presented one at a time to the network. The final output of the network is given as $\hat{\mathbf{y}}$ and used in evaluating the performance of the model. For example, using time step t of the training process, input vector $\mathbf{x}^{(m)}, m = 1, 2, \ldots M$ propagates through the network per the previous equations, eventually outputting $\hat{\mathbf{y}}^{(t)}$.

Following a complete forward propagation through a multilayer perceptron, a loss function measures the deviation between the model's prediction and the ground truth, known as the loss or *risk*. As with activation functions, several loss functions are available, and each particular function produces different measures of loss, such as Root Mean Squared Error (RMSE), given in Equation 2.9. To simplify the explanation, we use the standard square of error on the target vector $\mathbf{y}$ and network output $\hat{\mathbf{y}}$ (i.e., activations of the final layer $\mathbf{a}^{(L)}$).

$$\mathrm{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.7}$$

$$\mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y_i * log(\hat{y}_i) \tag{2.8}$$

$$\mathrm{RMSE}(\mathbf{y}, \hat{\mathbf{y}}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{2.9}$$

where $n =$ the size of the label vector, which should equal the target vector. Selecting a loss function depends on the nature of the data as well as the form of output from the model. Commonly used loss functions include mean squared error (MSE) loss, typically used in regression tasks, binary cross-entropy (CE), used in classification tasks, and root mean squared error, which features in the models produced as part of this project. These functions are given respectively in Equations 2.7 through 2.9.

Loss functions offer a quantifiable measure of the model's performance and its internal parameters. Therefore, the smaller the loss, the more accurately the parameters of the model reflect the natural truth of the data.

2.2.3 Training Through Backpropagation

Regardless of the selected loss function, the objective of the majority of machine learning models is to minimise its output through learning. Learning is optimised through the repeated refinement of the model weights and biases through some optimisation process, such as gradient descent, to define and update the change in $\Delta\mathbf{W}^{(\ell)}$ and $\Delta\mathbf{b}^{(\ell)}$ on a per-layer (ℓ) basis during training.

Several more sophisticated variants of optimisation algorithms have recently appeared which exhibit more efficient convergence toward the global minima of the learning space—especially in non-convex and complex geometric space—such as the Adam optimisation algorithm, which maintains an adaptive learning rate for each parameter of the network [20].

Backpropagation is one of the most renowned training algorithms used in neural networks. Its objective is to minimise the output of a given loss function, quantifying the difference between actual versus expected output through iterative adjustment of the network's weights. Supervised training requires a dataset typically structured as two vectors: one for the training data itself and another for the corresponding target (i.e., label), as classification problems can be multi-label.

$$w_{ij}^{(\ell)}(q+1) = w_{ij}^{(\ell)}(q) - \eta \frac{\partial J}{\partial w_{ij}^{(\ell)}(q)} \tag{2.10}$$

As training progresses through the dataset, the weights and biases within the network are updated incrementally based on the provisional output for the sample at position q of the training set against the ground truth counterpart. Equation 2.10 shows the change for the weight in the next parameter series deriving from the negative gradient of the previous weight (i.e., the steepest descent). This gradient is multiplied by η, the learning rate for the network: a hyperparameter that governs the stride taken in converging toward the minimum - local or global. The learning rate must be delicately balanced, as weights will continually overshoot the minima with too high a learning rate. Similarly, an unnecessarily low learning rate will take an unreasonable amount of time to converge to any reasonably reachable global optima and may get trapped in local optima. It should be noted that bias parameters undergo the same process described alongside Equation 2.10.

In the context of a single unit, one eventual value will produce the minimum possible output from the loss function, which is what the backpropagation algorithm seeks to reach. Backpropagation is an efficient algorithm used to compute this gradient, which is used in conjunction with an optimiser, such as stochastic gradient descent (SGD), for parameter adjustment in the network to minimise the network's overall error [6].

Output from the neural network is then calculated into a sensitivity vector $\mathbf{s}$, which typically refers to the gradient of the loss function with respect to neuron $X_n^{(\ell)}$ and its responsibility for the overall loss in the forward pass. The sensitivity is particularly important in the backpropagation algorithm for its ability to express how a particular output unit (i.e., neuron) changes its output with respect to changes to the input vector [21], and goes on to assist in identifying and shaping the synapses of the network.

$$a^{(\ell)} = f^{(\ell)}(\mathbf{n}^{(\ell)}) \tag{2.11}$$

Working backward from layer L, the sensitivity s for the network can be computed directly for each neuron activation a using the chain rule [10]

$$s_n^{(L)} = 2\left(a_n^{(L)} - t_n(t)\right) \dot{f}^{(L)}(n_n^{(L)}) \qquad n = 1, 2, \ldots N_L \tag{2.12}$$

where $\dot{f}$ denotes the derivative of activation function f.

$$w_{ij}^{(L)}(t+1) = w_{ij}^{(L)}(t) - \alpha \frac{\partial E}{\partial w_{ij}^{(L)}(t)} \tag{2.13}$$

$$b_{j}^{(L)}(t+1) = b_{j}^{(L)}(t) - \alpha \ \frac{\partial E}{\partial b_{j}^{(L)}(t)} \tag{2.14}$$

Based on the steepest descent, the weights and biases are individually updated according to Equation 2.13-2.14. This gradient is multiplied by a learning rate α for the network: a hyperparameter that governs the stride taken in converging toward a local minima.

2.2.4 Network Regularisation

Backpropagation in feedforward networks carries the risk of vanishing or exploding gradients, whereby gradients calculated < 1 shrink exponentially toward zero as training propagates through the network, delaying the learning process of earlier layers; or where gradients calculated > 1 *explode* over the course of training, destabilising the model. This phenomenon is often observed in neural networks beginning training from randomly initialised weights and in those opting for the hyperbolic tangent ($tanh$) as an activation function.

$$W \sim U\left[-\frac{1}{\sqrt{n}}, \frac{\sqrt{1}}{\sqrt{n}}\right] \tag{2.15}$$

Several techniques have emerged that help prevent such cases. The most common of which is to initialise weights within a monitored variance throughout the network in a strategy developed by [22], given by Equation 2.15, where U is the uniform distribution and n is the size of layer $\ell - 1$.

A model performing well during training but considerably poorer in testing has likely been designed overly complex and inadvertently learned the training data, thus it cannot generalise unseen data outside training. This observation is known as overfitting [23], [24]. Given the enormous number of weights in a deep neural network, which can span well into the thousands, deep neural networks require additional precautions against overfitting [25]. One simple defence mechanism is to reduce the total trainable parameters of the model. The inverse of this case is known as underfitting; observed when a model delivers consistently poor results, it may be unable to capture the underlying concept of the training data [24]. In such a case, the model may be too simple in design, too biased, or the dataset may be unrepresentative of the problem tasked to the model. Both cases are addressed by seeking to minimise the training error (from inappropriately preprocessed data) and the gap between the test error [8].

Limiting the error in the generalisation of a model is known as regularisation. Many techniques designed to achieve this are implemented as configurable hyperparameters during development stages. One of the most effective techniques developed by [26] is known as dropout, which randomly severs a defined proportion of neurons within a network during each training batch. Dropping neurons can help desensitise the model from the training data, preventing it from forming overly-complex connections. These neurons are restored after training. Dropout is implemented as a mask and is supplied with a probability factor of disabling each neuron.

$$R(x) = \begin{cases} \max(0, x), & \text{for } x < 0 \wedge P(d) \\ 0, & \text{otherwise, or for } x \geq 0 \end{cases} \tag{2.16}$$

For example, when incorporated with the ReLU activation function as previously seen in Equation 2.4, where $P(d)$ is the probability of dropout of a given rate $0 < d < 1$.

Overfitting due to excessive model complexity in a model is typically identified by its large weights within, as a network with smaller weights tends to generalise better than a similar network with larger weights [27]. This weight magnitude can be quantified and regularised through L_2 parameter regularisation, also known as weight decay.

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \left(\sum_{j=1}^{n}\|w_j\|^2\right)\frac{\lambda}{2m} \tag{2.17}$$

This technique penalises excessive weights and instead encourages lighter values closer to 0 (but not 0 explicitly) and is implemented alongside the chosen loss function for the model. Equation 2.17 demonstrates the L_2 loss function appended to the MSE loss function as previously defined in Equation 2.7, where m represents the input count and w_j represents the weight matrix for the layer. An additional hyperparameter λ governs the strength of the penalty.

Whichever techniques are employed in regularising a model, if any, must be carefully chosen. Applying these regularisation strategies pre-emptively on a model that does not yet display a need for them can limit its performance. Likewise, fusing every form of regularisation can yield a similar result, as investigated by [28], who found pairing L_2 regularisation with the Adam optimiser on an otherwise well-performing model often leads to poorer results than when paired alone with some optimisation strategy (e.g., gradient descent).

2.3 Recurrent Neural Networks

Despite the power of traditional feedforward neural networks, they are bound by fixed-size input for training and evaluation and struggle to consider past input of potentially varying length, temporally-dependent data (i.e., time series), such as natural language or time series. As with most classical machine learning models, these networks assume input samples are independent of one another and are not built to capture any underlying temporal relationships. The simple approach: to architect and feed the entire dataset to the input of a network, is unfeasible when capturing long-range data dependencies.

In classical machine learning, the Hidden Markov Model (HMM), a doubly stochastic model, can model well discrete auto-correlated processes [29] so long as the conditional probability of the following state is dependent on the present state [30]. This hard dependency can severely inhibit the model from learning long-term dependencies. The recurrent neural network is one solution toward considering the potential nuances found in sequential data over long ranges, operating in distinct timesteps $X = (t_0, t_i, \ldots, t_n), t_i \in \mathbf{R} > 0$ as input.

In this architecture, neurons now retain a *memory* and can then consider past observations when computing for the current timestep. Therefore, recurrent neural networks do not make assumptions of independence over time. This feedback loop distinguishes a recurrent neural network from a traditional feedforward neural network. As a result, with its ability to decisively retain and discard past observations, this architecture is particularly suited to classifying and predicting from temporal/sequential data.

The training of recurrent neural networks is similarly undertaken through backpropagation to feedforward neural networks, known as backpropagation through time. Recurrent neural networks are first unfolded through time to resemble a traditional, extremely deep feedforward neural network to apply the conventional backpropagation process through the network on both the input and output layers at each timestep [12]. However, recurrent neural networks are still susceptible to vanishing gradients. Similarly to feedforward neural networks, learning primarily occurs in later layers (i.e., later timesteps), with the weights decaying and vanishing entirely around earlier timesteps [31].

2.3.1 Long Short-Term Memory Units

Whilst recurrent neural networks hold the unique capability to interpret sequential data, vanilla recurrent neural networks struggle to connect long-term dependencies, given their low short-term memory capacity and propensity for gradients to vanish or explode during training, as with vanilla neural networks. An alternative model developed by [31] aims to address these issues by swapping a traditional neuron with a long short-term memory (LSTM) cell, which extends the cell of a recurrent neural

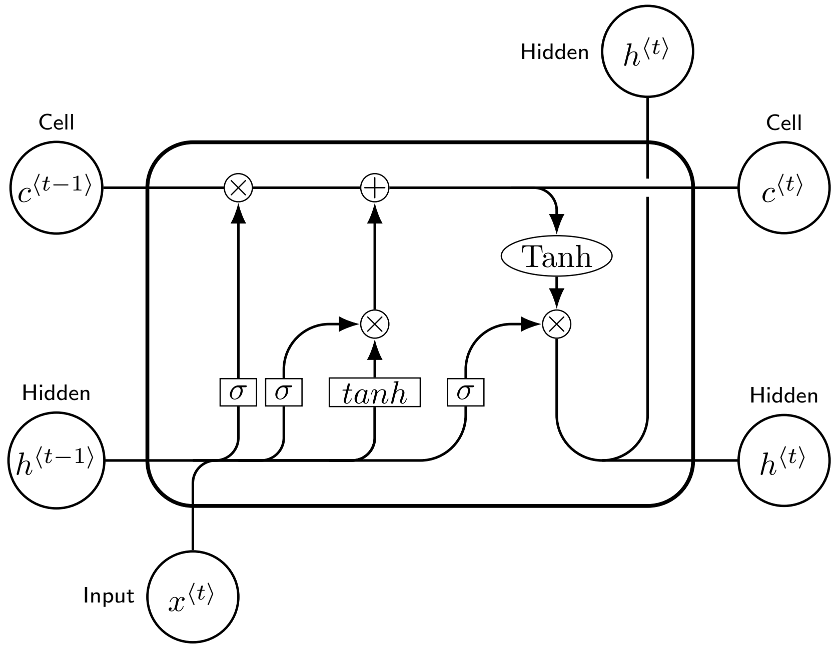


Figure 2.4: A long short-term memory cell. Symbols $\otimes$ and $\oplus$ denote pointwise multiplicative and additive operations between gates respectively.

network. LSTM cells protect the error flow fading from backpropagation during training through multiplicative input and output gates, which learn to regulate internal access and update only to relevant inputs [31] based on its own set of learned weights.

Forget Gate: It determines the amount of information to discard from the cell state. This is crucial for the LSTM's ability to "forget" irrelevant or outdated information from the past, making room for more relevant, new information.

In an LSTM (Long Short-Term Memory) cell, each of the three gates—input, forget, and output—has a specific function:

LSTM cells each maintain a mutable cell state c_t. Each of the four activations (σ and $tanh$) are, in essence, their learned network layers acting as gates, denoted as Γ, regulating the flow of information into, across, and out of the cell. Figure 2.4 illustrates the layout of an LSTM cell. On input, the cell first discerns at the t-th timestep of the input $\mathbf{x}$, what segments of information to discard from the cell state by passing through the forget gate. Cells follow a similar forward propagation to vanilla neural networks through a trainable parameter set of weights $\mathbf{W}$ and ancillary bias $\mathbf{b}$.

$$\Gamma_f^t = \sigma(\mathbf{W}_f[a^{(t-1)}, X^t] + b_f) \tag{2.18}$$

which modulates the cell's existing memory, where $a^{(t-1)}$ carries the activation at the previous timestep, and by considering the hidden state—or activation a—at the previous timestep $a_{(t-1)}$ along with the cell input (i.e., recurrent connections) X^t.

$$\Gamma_i^t = \sigma(\mathbf{W}_i[a^{(t-1)}, X^t] + b_i) \tag{2.19}$$

The input gate (Γ_i) evaluates which values to update into a candidate cell state $\tilde{c}^t$. The existing state $c_{(t-1)}$ is then mutated to integrate the new values. It decides how much of the new information to add to the cell state. This gate filters the incoming information from the current input and the previous hidden state, allowing the LSTM to update its cell state with relevant new information.

$$\tilde{c}^t = tanh(\mathbf{W}_c[a^{(t-1)}, X^t] + b_c) \tag{2.20}$$

$$c^t = \Gamma_f^t c^{(t-1)} + \Gamma_i^t \tilde{c}^t \tag{2.21}$$

Finally, the output gate determines whether the cell should emit the activation or not

$$o^t = \sigma(\mathbf{W}_o[a^{(t-1)}, X^t] + b_o) \tag{2.22}$$

where $\mathbf{W}, b$ are specific to each gate, before passing the cell state for activation. This gate controls the extent to which the value in the cell state is used to compute the output activation of the LSTM unit. The output gate filters the cell state based on the current input and the previous hidden state, producing the final output of the LSTM

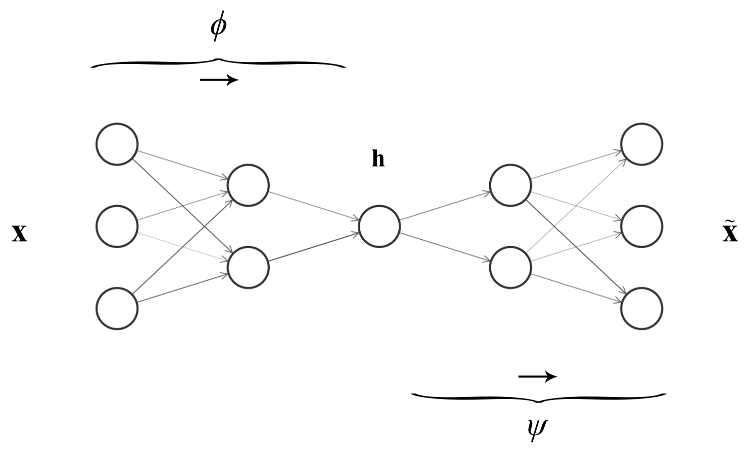


Figure 2.5: Example of an autoencoder neural network, illustrating the gradual compression from the encoder ϕ, to latent space $\mathbf{h}$, to the decoder ψ.

cell, which can be final or piped as input to another cell.

$$h^t = o^t \tanh(c^t) \tag{2.23}$$

Another variant of the recurrent neural network cell, the gated recurrent unit (GRU), was developed as an alternative to the LSTM. This cell functions similarly to that of the LSTM cell but instead combines gates Γ_f and Γ_i, merges the cell and hidden states (c, h), then exposes this combined state as the output gate to the next section of the network [32]. The result is a more computationally efficient cell but at a lesser learning capacity, for which we favour the LSTM. Eliminating expert human knowledge is one of the key appeals of deep learning, and research into combining deep learning techniques with classical point process modelling is relatively new in and of itself.

2.4 Autoencoder Neural Networks

As with other machine learning models, autoencoders are a form of neural network which strive to minimise the discrepancy between input and expected output. Their defining characteristic is that their output value is trained to approximate their input value identically. This mirroring-like behaviour is achieved by encoding the input across narrowing layers down to a latent space lower in dimension than the original input, then decoding from this latent space the original input at the output layers. Their architecture can be considered as two back-to-back neural networks, where the objective of the first is to compress the input as representatively as possible, which is received by the second network and tries to reconstruct it back to the original input. Figure 2.5 illustrates this architecture and concept, showing independent neural networks conjoined by the latent space.

When training an autoencoder, the entire architecture is considered as a single network, forcing this design of neural network to derive a compressed underlying identity function from the latent space, making this an unsupervised model as the data are unlabelled. The size of the latent space, i.e., by how much compression the input can withstand without losing its representative meaning, is configurable, and we express it in a range of 0–100% of the original input. Optionally, the latent space can be decoupled from the network once trained before continuing to the decoding stage, where the input is reconstructed, making it ideal for discriminative tasks such as anomaly detection.

This architecture comprises two trainable functions accompanied with individual training parameters (encapsulated as θ): the encoder $\phi_\theta : \mathbf{x} \to \mathbf{h}$, and the decoder $\psi_{\theta'} : \mathbf{h} \to \mathbf{x}$. Note that the encoder stage has separate training parameters from the decoder (for clarity, differentiated as θ'). Latent space $\mathbf{h}$ is the compressed representation of the input, from which the decoder will try to approximate. The encoder and decoder operate as parallel neural networks, the inputs for which undergo the same standard forward propagation found in the deep neural networks that were previously detailed. The decoding stage accepts as input the latent space, which then undergoes a similar feedforward propagation with its own separately trainable set of weights and biases.

$$\phi_\theta *, \psi_{\theta'} * = \arg \min_{\phi_\theta, \psi_{\theta'}} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \phi_\theta \circ \psi_{\theta'}(x_i))^2} \tag{2.24}$$

Outputs are then subjected to one of the same series of activation functions previously described in Section 2.2 in training respective parameters θ and θ' in both components of the network through the shared optimisation problem (2.24). Here, $\circ$ denotes the result of the encoder stage piping the entirety of $\mathbf{h}$ directly as an argument to the decoder stage.

Autoencoders are popularly incorporated as part of a machine learning system, commonly as a dimensionality-reduction technique when working with a complex dataset, or in a generative application to enrich small or sparse datasets (achieved with another variant of autoencoder, known as a variational autoencoder) based on an underlying trend. We favour the autoencoder architecture for its reconstruction characteristic. It has been shown that autoencoders that generalise well on normal, non-anomalous data will struggle to reproduce data unseen during training [33]. Carrying the reconstruction error forward, we can analyse and determine if, where, and by what margin the autoencoder struggled to identify anomalies.

2.5 Measuring Model Performance

The performance metrics classifier machine learning models are derived from the confusion matrix: a 2-dimensional (actual outputs and predicted) form of a contingency table, where both dimensions comprise sets of classes relative to the problem. For example, a classification task with $N = 5$ potential classes is measured in an $N \cdot N$ confusion matrix, where correct classes of cell i are added to position (i, i) of the matrix and incorrect classifications (predicting j-th class instead of i) are added to cell (i, j). From this confusion matrix, we derive metrics more informative of the model's performance, including accuracy, precision, recall, sensitivity, specificity, and F_1-score. For one case, these rates require, at a minimum, the counts of true positives TP, true negatives TN, false positives FP (i.e., type I error), and false negatives FN (i.e., type II error).

Accuracy, the most initially intuitive metric, is an absolute performance measure given as the proportion of results correctly classified as their known, labelled class amongst the entire training set. It is given by

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} \tag{2.25}$$

$$= \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.26}$$

where P = count of positive predictions made and $mathrmN$ = the total count of positive cases in the dataset, and N = the total count of negative cases in the dataset. In multi-classification evaluations, both of these are relative to the particular class using

$$\mathrm{ACC} = \frac{\sum_{i=1}^{K} C_{i,i}}{\sum_{i=1}^{K} \sum_{j=1}^{K} C_{i,j}}. \tag{2.27}$$

However, accuracy alone can be misleading in the common situation where datasets are not symmetric. For example, in a situation where false negatives are simply unacceptable, an imbalanced dataset may have a low P and still return a high accuracy. Therefore, we look to relative performance measures.

Precision (positive predictive value, PPV) is given as the proportion of results classified as their labelled class. It can be calculated from the discovery or false discovery rate (FDR), the proportion of results incorrectly classified as their expected class. These are given respectively as

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} \tag{2.28}$$

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}. \tag{2.29}$$

Recall (true positive rate, TPR) is given as the proportion of results correctly classified as their labelled class, given by

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}. \tag{2.30}$$

Specificity (true negative rate, TNR) is given as the proportion of results correctly designated as not belonging to the labelled class and is given by

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}. \tag{2.31}$$

Sensitivity (false negative rate, FNR) is given as the proportion of results incorrectly designated as belonging to the labelled class, given by

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}. \tag{2.32}$$

The last metric part of the evaluation of our models is the F_1-score: a harmonic mean (i.e., weighted average) of the precision and recall, both of equal contribution to the score. It is given as

$$\mathrm{F}_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{2.33}$$

We favour this metric as our primary learning target as precision/recall alone can each improve at the expense of the other. When computing in multi-class applications, the F_1-score is the average of each class of some weighting strategy, e.g., the unweighted mean for each label.

2.6 Challenges in Deep Learning

It is often difficult to interpret and debug machine learning models. In deep learning especially, where the training process is highly abstracted from the engineer to the equivalent of a *black box*, it can be difficult to deconstruct and surmise the rationale behind a model's evaluation [34]. Various tools can aid the development, training, and interpretation of deep neural networks. The work of [35] develops such a tool to introduce a novel visualisation technique for use in the diagnostics of CNNs, another variant of neural network best cased for image data. This technique allows engineers to visualise and inspect the contribution and performance of intermediate feature layers and the operation of the classifier in the network. The developer may revise the architecture of a network using the insights gained.

Many implementations of machine learning technologies strive to benefit human life, whether in health, lifestyle, or out of convenience. Each of these endeavours requires vast, thorough training data, which can often contain sensitive information revolving around the habits or profile of a person. This personal identifiable information (PII) can be recoverable from a compiled trained model, as demonstrated by [36], who were able to recover images of human faces used in training within the network from a facial recognition system. However, sanitising such datasets from personal or identifiable information can considerably diminish the performance of the model, which leads to an important balancing act. Several techniques have emerged aiming to address this, such as differential privacy, which introduces in the data calculated statistical noise and constraints to limit a model's reliance on sensitive and personal information. Such techniques have been successfully employed by large technology companies, such as Apple, who incorporate differential privacy at a local device level in features across their operating systems [37], such as predictive text.

Arguably the most substantial challenge in most areas of machine learning is procuring sufficient data for training—in both quality and quantity. Deep neural networks, in particular, require vast amounts of data to generalise well [38]. Likewise, a dataset that does not provide sufficient data variety will impact the model's ability to generalise for different scenarios. Here, we enter the challenge of imbalanced data: the problem faced by even modern machine learning approaches whereby the response to inputs is heavily weighted toward particular classes by uneven distribution in the training data.

Such a state is easily identified in the data treatment stages. An imbalanced model will demonstrate a higher competence in the majority class and little to none in the minority class(es).

Oversampling and undersampling are well-researched strategies to compensate for an imbalanced dataset. Both sampling methods modify the distribution of a dataset to redress the balance between minority and majority classes. Oversampling artificially synthesises instances of the minority class, the simplest technique of which is random. Using repeated data points will restore the balance at a higher propensity to overfit. One of the most popular approaches is using the Synthetic Minority Oversampling Technique (SMOTE): a process that selectively generates and inserts instances of the minority class at a distance between its nearest neighbour [39]. Undersampling, the converse, deletes instances of the majority class. Unless the dataset is sufficiently abundant in its majority class, this approach can have a major impact on model performance. Most often, we see both techniques applied in tandem. For example, SMOTE is combined with the Edited Nearest Neighbour (ENN) [40] undersampling technique, which oversamples the minority class before pruning excessively represented instances of the dataset.

One of the largest challenges to date in the deep learning space is optimising its intense computational complexity requirements. Within this problem lies the secondary challenge of porting the output of such a training regime into an efficient model, computable on lower-grade hardware. The calculations performed with each training run, which can run into the billions, have recently become achievable with modern compute power and have inspired purpose-built hardware, e.g., tensor processing units. At the microcontroller level, such as the ESP32 we are targeting, quantisation is one technique available to reduce model size. The high-precision floating point representations within the weights and bias of a trained model can be reduced to lower precision, potentially unsigned types at a substantial performance benefit. This technique is not without cost, as lowering the precision will likely impact model performance. It is for the architect to decide whether the loss is acceptable. This technique was explored in [41], selectively quantising a deep neural network to 8-bit precision at a performance drop of only 0.7%.

An important challenge in deploying machine learning models lies in the implications of their failure and the effects on their trust. The widespread adoption of machine learning presents a new attack surface and increased security risk across a broad range of applications. It is therefore imperative that training and testing be as accurate as possible. A model could have been the victim of adversarial attack: a set of deliberate methods to poison a model with adversarial examples [42]. For example, in an autonomous vehicle domain, failure to correctly recognise a stop sign could lead to road traffic collisions. Whilst some can only be exercised with knowledge of the model weights, others can interface directly with the model itself. The work in [43, 44] demonstrates through calculated noise and pixel swapping, the performance of classifiers can be decimated.

2.7 Non-Intrusive Load Monitoring

Recent advances in society, energy, and climate have led to increasing concerns on the demand for energy and efficiency. Supporting efforts on energy consumption at the consumer level has led to the systematic deployment of the smart grid: an electrical grid enhanced by advanced technologies to monitor and manage the transport of electricity from origin to end user [45]. The result is a more economical, available, and efficient electrical grid. To enable this from the end user side requires some integral infrastructure to assess and monitor energy demands, which gave rise to the smart meter ubiquitous in households today.

Load monitoring defines the task of measuring energy consumption at the appliance level, i.e., usage tracking. Given a device or appliance equipped with the appropriate sensor, its energy consumption can be probed and transmitted across networks as a feedback system; this is the intrusive approach, where the primary drawbacks include setup and maintenance costs, whether retrofitted or integrated, as well as the privacy concerns surrounding personal usage being used to construct profiles and learn habits of end users. This strategy has proven effective in harvesting organic datasets for load monitoring, such as UK-DALE [46] and Blued [47], which have been used to drive NILM solutions. Cost-effectiveness and low customer acceptance have led to the need for a less invasive solution [48].

NILM is a low-cost alternative aiming to address both challenges without the need for distributed direct sensing that is difficult to install in ordinary households, lessening the invasiveness of the original solution. Easily installed and removed, NILM approaches are most often based on a single channel source, such as the total power consumption of an entire property. By using various statistical techniques, providers can disaggregate the power signatures of different appliances from a single source when measuring power consumption. Electrical appliances display unique signatures during their normal working cycles, which may contain information on steady or transient states [48].

Employing supervised machine learning-based models for privacy-preserving NILM solutions is a thoroughly researched field. From classical models such as SVM or K-means clustering [49], this technology can interpret well the dynamics of load monitoring data. Unsupervised or semi-supervised strategies that do not require labels in training are preferable in a real-time NILM context but generally will underperform compared to pretrained, supervised equivalents. For example, [50] proposes a temporal convolutional network to segment and generate pseudo-labels of a load monitoring dataset in a semi-supervised solution. In an unsupervised approach, [51] employ clustering and classification using clustering and a variation of the on-off appliance model to detect events without pretraining.

Our objectives draw several consistencies with the ambitions of and approaches toward NILM. However, NILM solutions do not typically account for or support the particular devices we are interested in: those with a finite count of transient states [52], [53]. As with NILM, we rely solely on the aggregate energy consumption of an entity (in its case, a household) to isolate different behaviours, before undertaking fault detection. There are no sensors, probes, or user input to supplement learning, and we must disaggregate the unknown count of states of a machine of a finite number of states, which does not feature in the problem space of NILM [54].

Challenges Outstanding

- Existing classification approaches do not account for multistate devices (e.g., laundry machines);
- Measuring the energy consumption of a classified state of a device is not supported in conventional NILM approaches.

2.8 Anomaly Detection

Anomaly detection, sometimes known as *outlier* detection, refers to the analysis and identification of unexpected occurrences in items, events, or observations [55] in otherwise normal data. Regular use cases of anomaly detection include fraud detection in financial contexts, network intrusion, and medical diagnosis. Data used in this actively researched field predominantly feature more positive examples than anomalous ones, complicating the challenge of classifying such anomalies. The general strategy, therefore, is to define some boundary for positive examples in the dataset, using what negative examples are available within the dataset to validate.

Selecting an anomaly detection approach requires multiple considerations. Figure 2.6, elaborated further in [56], illustrates this decision process, where the architect must be conscious of the form of the data available and the definition of an anomaly in this particular application. These factors dictate the technologies and mechanisms available for their use case. Our application will receive multivariate power consumption data, allowing more complex learning models with more powerful generalisation abilities.

Between the outlier types to be identified by the system, point and sequential, heavily influence the design of the system. Point-based anomalies are the simplest: searching for a point in the data sufficiently different from the rest. Sequential (sometimes referred to as contextual) terms a point of data anomalous in a specific context, but considered

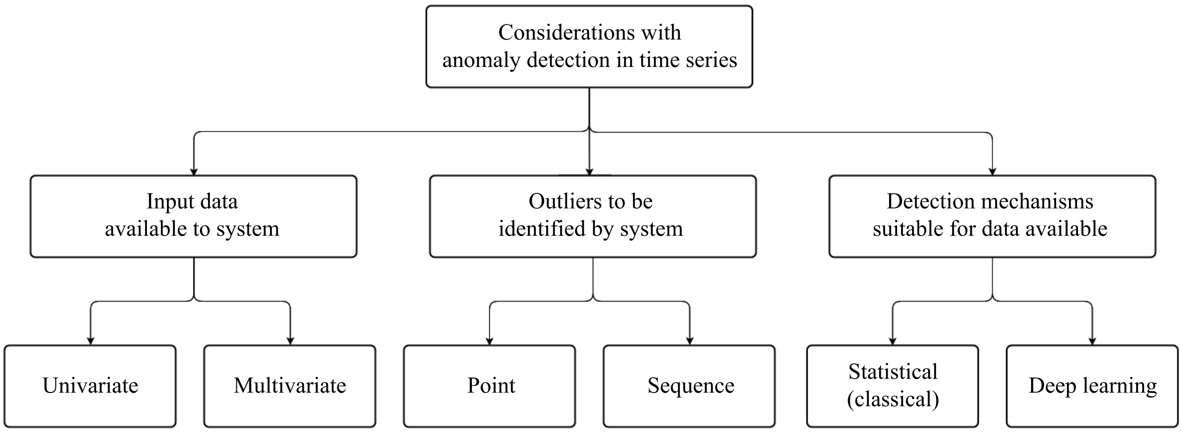


Figure 2.6: Tree of considerations when working in anomaly detection, including the available data and nature of anomalies to be detected. These are still decisions driven by domain expertise.

otherwise normal in a different context. For example, in the context of a financial system, a gift card purchase may be considered normal during a holiday season than in year-round buying patterns. A third term, a collective anomaly, is a series of points that, when grouped, are considered an anomaly.

Conventional statistical approaches, such as ARIMA or exponential smoothing, rely on the assumption that there is some underlying origin function to the data generated and will aim to discover the parameters used to define it [57]. Alternatively, machine learning-based methods for anomaly detection make far less explicit assumptions about the underlying origin of the data and attempt to optimise a series of complex functions mapping input data to the desired output.

As outlined by [58], there are several approaches toward anomaly detection, and the correct one is highly dependent on domain and data. Figure 2.7, adapted from [59], visualises this set of approaches into a decision tree of suitable approaches depending on the data available for the task.

The first is probabilistic detection, based on estimating some probability density function to define an underlying distribution in the data. This statistical approach assumes the data is generated using some origin function, which this approach attempts to ap-

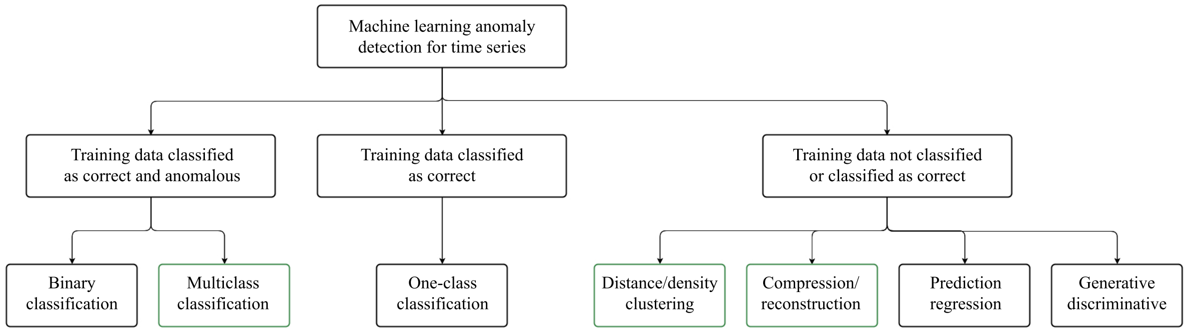


Figure 2.7: Decision tree of machine learning approaches available to an architect depending on the nature of the data. Those highlighted in green are options selected for the work described in this thesis.

proximate. An example of this work today is by [60], who construct a density function based on Kullback–Leibler (KL) divergence and on minimal assumptions of the data in a similar approach to domain-based detection.

Another approach to anomaly detection is distance-based detection, using clustering and nearest-neighbour techniques to form implicit boundaries within datasets, using distance or density to measure an anomaly. Positive data points are assumed to lie in close proximity to neighbours, and the task of distance-based algorithms is to provide a subset of data points inconsistent with this defined positive boundary. The work of [61] explores a distance-based algorithm measuring a point's outlier factor, designating the furthest as anomalies, and testing on a shuttle dataset whether to prefer auto or manual pilot of a spacecraft landing.

Information-theoretic approaches, based on the principles of information theory, compute the information content of a dataset using measures such as (relative) entropy under the assumption that the characteristics of a positive dataset are significantly disturbed by introducing an anomalous data point. This assumption is evaluated in the work of [62], who employ an entropy-based information-theoretic anomaly detection algorithm to identify forged traffic inside in-vehicle controlled area networks. It is found that a blanket coverage identifies well in high volumes of anomalies, and single-class targeted algorithms are better suited to fewer anomalies.

Reconstruction-based approaches are commonly used in safety-critical applications [58] and introduce us to the deep learning strategies of anomaly detection, where autoencoders are the most common choice. Many recent techniques employ some form of neural network, which, in essence, is trained to approximate the input data as closely as possible in their output. The ruling theory is that positive examples will reconstruct with relatively low error; however, the network will struggle with anomalous data not experienced during training. Learning occurrs inside a latent space, forcing the network to learn a compressed interpretation of the data in the hope it recognises its core characteristics. The degree of deviation from the original reconstruction can then be used in a number of techniques to decide whether to designate the point as anomalous. We favour this approach as, from the beginning, it allows us to refrain from any presumptions regarding the data. One deep learning-enabled example of this is the work found in [63], proposing a recurrent neural network architecture to detect and classify types of anomalies in spacecraft sensor data, coupled with a higher level framework which thresholds identified anomalies and, in some cases, mitigates false positives.

2.8.1 Related Work

Consumer-oriented fault detection in the domestic space is largely unexplored, particularly in the context of a monitoring device fitted retrospectively. At a higher level of time-series clustering and forecasting, anomaly detection within highly constrained environments offers a greater wealth of research in each field.

Forecasting is a powerful application of neural networks which can model sequential data; a network's ability to generalise even the most complex relationships and patterns in data makes it an attractive alternative to traditional, statistical univariate approaches (e.g., ARMIA and its variants). However, vanilla neural networks often struggle beyond single-step forecasting, as investigated by [64], who found the architecture of the network, data preprocessing stages, and abundance of samples required for training all to be considerable barriers to model performance. Some work attempts to increase model performance through intelligent feature engineering, such as [65], which leverages sparse filtering to learn features from bearing data in identifying faults, though our dataset does not comprise mechanical features.

The work of [66] adopts a similar approach to this stage of the project, using the minimum message length clustering algorithm by [67] to first cluster a time-series dataset before training each cluster on a counterpart LSTM network. Although the results of this system were competent, forecasting can only be performed when the entire dataset is well-clustered. LSTMs have proven immensely successful in time-series forecasting, given their ability to factor in past readings and experience when making future predictions. An example is the fully connected, convolutional LSTM by [68], which extends the traditional operations inside the LSTM cell into the convolutional processes. Ensemble architectures for time-series forecasting are another choice of model design, acting on the combined results of several forecasting models. In [69], they implement this concept between multiple LSTM networks, using a weighted system at the output.

One of the most relevant recent works in the context of NILM by [51], who model event detection using an approach combining heuristics and clustering of household power data—much like our own—to detect the toggling of specific appliances within the property in a non-intrusive manner. The system processes the data into estimated and delta power readings, along with maximum duration, and performs well in clustering activations. [70] continue research in this field of energy disaggregation by combining autoencoder reconstruction-based predictions trained on features provided by a form of popular adversarial networks. In our context of anomaly detection, we see similar approaches in NILM solutions, including the work in [71] which trains a series of classic statistical and deep learning models on smart meter household energy data to detect deviations in user behaviour.

Complex networks require vast amounts of data in specific formats and are significantly more expensive to (re)train computationally, which requires unrolling the entire LSTM network. This unrolling is not an ideal process given that we see and train inputs incrementally. Autoencoders are a frequent choice in anomaly detection, trained to capture normal structural patterns. The premise of reconstructing an anomalous input should result in a much higher error against a normal input has proven effective in unsupervised anomaly detection. [72] also follows this approach in a network intrusion detection system, this time powered by an ensemble of autoencoders—a mature theory in anomaly detection. Unlike others, this particular system is designed to work online under similar hardware constraints but requires additional training to derive a feature mapping from one continuous data stream. Further, the autoencoders do not retrain once executed, which, despite the model training online, still leaves them susceptible to model decay, though there are mechanisms explored to combat this. Similarly, [73] develop an autoencoder for anomaly detection, with a transformer network sitting in the latent space, which performed well in benchmarking.

Challenges Outstanding

- Datasets driving these existing works are single-instance with no requirement to account for unknown environments;
- Deep neural networks themselves often act as 'black boxes', making it difficult to understand and explain the rationale behind certain predictions of a model;
- Deep learning models are computationally expensive to train and interrogate, requiring far more training data than what we have available.

2.9 Temporal Point Processes for Interval Prediction

To train a model, particularly an LSTM, capable of producing actionable results from continuous raw timestamps alone is not feasible, as we cannot rely on neural networks in their current state to consider the nature of this challenge. Temporal point processes are probabilistic, statistical models suited for discrete event data, particularly applicable in scenarios where events occur at unpredictable times. The goal of these models is to accurately describe the likelihood of these events occurrence in the future. These models are conditioned on discrete event sequences, with the task of predicting the occurrence (or likelihood) of the next event. This output comes strictly from event history, with no learned behaviour in its model.

Temporal point process model a discrete event sequence, where t denotes an interval in localised time $0 < t_i < t_{i+1}$, from some intensity function $\lambda(t)$ describing the probability of a new arrival, written to capture the underlying distribution of event sequence, i.e.

$$\lambda(t) = \prod_{n} \lambda(t_n | \ldots, t_{(n-2)}, t_{(n-1)}) \tag{2.34}$$

A higher intensity expects a higher probability of arrival. Classical intensity functions, such as Hawkes or Poisson processes.

$$\lambda(t) = \int_0^t \phi(t - t_i) dN(u) = \mu + \sum_{t_i < t} \phi(t - t_i) \tag{2.35}$$

Hawkes processes are a form of self-exciting processes, such that new arrivals to the process *excite* the intensity function, decaying toward 0 as time elapses until the next arrival, given by Equation 2.35, where dN denotes differential of the counting process for events occurring at times $t \in A$, and $\mu > 0$ is the background rate of the process to date, sometimes referred to as the response or excitation function.

The decay rate is calculated by ϕ: a kernel function modulating the interval between the current and past arrivals (e.g., $\phi(t) = \exp(-\alpha t)$, where $\alpha \geq 0$ regulates the decay term). The form of $\phi(\cdot)$ chosen, although it has no asymptotic upper bound, typically decreases such that more recent events pose a stronger influence on the intensity than those more distant. For example, in seismology, aftershocks are far more likely immediately following the primary quake. Event t_i contributes to the intensity at a rate of α.

In cases where future point processes are inhibited immediately following a new occurrence, known as a self-correcting process [74], the intensity function can be modelled as

$$\lambda^*(t) = \exp\left(\mu t - \sum_{t_i < t} \alpha\right) \tag{2.36}$$

where $\mu > 0$ is another hyperparameter, and new points are multiplied by a constant $e^{-\alpha}$ to decrease the likelihood of a future *point*. Identifying pattern behaviours with the potential to evolve between behaviours modellable by point processes such as these—or none at all—is one of our core motivations.

Sampling point process models depend on the historical records and the conditional intensity function. One of the more known techniques is Ogata's modified thinning algorithm [75], which simulates Poisson processes with excessively high intensity before thinning those designated as such by the defined conditional intensity function.

Arrival patterns that do not conform to this behaviour are better modelled on a conditional intensity function $\lambda^*(t|\mathcal{H}_t)$, which uses the leading history $\mathcal{H}_t = \{t_0, \cdots, t_i\}$ to the event in calculating a rate of events per time unit. $\lambda^*(t) = \mu \geq 0$ where μ is a constant parameter, are constant or deterministic, assuming events are independent of their leading history $\mathcal{H}_t$.

Unknown real-world arrival patterns are not guaranteed to fall under assumptions made by traditional models, or indeed those of any point process model. Applying classical temporal point process techniques requires extensive domain knowledge for tuning and is therefore at risk of model misspecification: where the model does not account for some significant nonlinearities. Leveraging deep learning techniques is a recent approach toward more flexible implementations of point processes modelling, and has taken research into this has branched into various approaches. Given their demonstrated power in long-term sequence modelling, recurrent neural networks are particularly suited to this task.

Some point process theories translate well into select real-world scenarios, such as retaliatory or aftershock events in self-exciting models, and can be extremely useful in scenarios where events occur at discrete times. These parametric options require domain expertise before modelling. Eliminating expert human knowledge is one of the key attractions of deep learning, and research into combining deep learning techniques with temporal point processes, however, is relatively new in and of itself. Further, employing point processes, in general, is largely unexplored in the application of IoT-specification hardware, energy data, and online learning.

2.9.1 Related Work

The work in [76] was one of the earliest attempts, proposing a Recurrent Marked Temporal Point Process method to model the type and timings between discrete events by encoding event history through a recurrent neural network, creating a more flexible model toward data that does not follow parametric assumptions. This modelling considers the intensity function of point process-based models as a nonlinear function—the essence of deep learning—of the event history, which has proven effective on real-world datasets. Their model produces a tuple prediction of an event classification and likelihood of occurrence given a sequence of events analogous to simulating from classical point process models.

In the context of commerce, the work of [77] explores a recurrent spatiotemporal network in estimating a person's check-in time to a particular region or location given their previous visit history—regardless of whether or not the person has previously visited the target location. By converging the historical check-in data (if available) with location history, a modified LSTM reached considerable performance in this objective. Whilst this work demonstrates the combination of recurrent neural networks and temporal point process modelling, neither the features nor nature of the dataset discussed is transferable to our problem statement, which leaves us unable to base our work on this architecture.

More recently, [78] takes the event history from a recurrent neural network and combines it with another neural network to model the cumulative intensity function, constrained to non-negative weights. The output of their model is a log-likelihood of a proposed interval, which requires submitting several samples and deriving the next interval from the predictive distribution. The work in [79] builds on this by proposing models which alternatively learn some conditional distribution factor historical events $\mathcal{H}$, as opposed to the more researched conditional intensity function $\lambda^*(t|\mathcal{H}_t)$.

Challenges Outstanding

- Capturing long-range dependencies and intricate temporal patterns is especially challenging when sequences are sparse, long and irregular;
- Majority of models are architected as dual-output, with an event classification alongside an interval prediction;
- Output is not easily interpretable, offering a likelihood score against a proposed interval;
- In high-dimensional feature spaces, it becomes challenging for neural point processes to process and learn from such data without overfitting;
- The potentially high volume and velocity of event data makes it difficult to scale neural point processes effectively.

2.10 Gradient Boosting in Fault Detection

Traditional maintenance strategies are far from optimal; the most common is preventative maintenance: an approach whereby maintenance is undertaken at predetermined intervals, such as a component's average lifespan or mean time-to-failure. This interval may be decided from manufacturer recommendation or statistical inference. The most prominent drawbacks to this strategy are:

- Offering little to no assurance in the event of catastrophic failure.
- Under-utilisation of components, which may have otherwise had considerable remaining useful life (RUL) and risk premature replacement at an unnecessary cost to the owner.

The alternative, reactive maintenance, is a more straightforward approach where maintenance is performed only after a confirmed failure, fully expending the component(s) lifespan, but requires rapid diagnostic and repair as failure may occur at any time, potentially during operations, and incur heavier repair and downtime costs.

Between both approaches lies predictive maintenance: a condition-driven preventative maintenance strategy [80] relying on factual data over the machine's condition and health of its components within, typically in a non-invasive fashion through sensor data. A comprehensive predictive maintenance program provides reliable insight on which managers (human or otherwise) can base and schedule necessary maintenance, allowing early identification of potentially serious problems, easier and cheaper remedied when repaired early [81]. The vast majority of predictive maintenance research is focussed on industrial applications, e.g., factory machinery [82] or aviation [83]. Such work is largely unexplored in the context of domestic applications.

Gradient boosting is a technique part of ensemble learning, a machine learning strategy that arrives at decisions from the collective outputs of multiple learning algorithms working in tandem. Boosting refers to a learning strategy combining multiple *simpler* models, often referred to as base models, trained using a base learner or *weak learner*. A base model alone would be relatively weaker in performance, whereas forming multiple models using boosting in an ensemble can produce a single, composite model with often far stronger performance. Two of the most common strategies used in ensemble learning are each known as boosting and bootstrap aggregation (known as bagging).

- The former trains multiple models in series in a form of evolution, where the errors and learnings from one training run are given as the base of the next model, used to improve prediction scores of the overall model in producing a stronger result.

- The latter, bagging, instead trains multiple independent models, combining their capabilities (i.e., learned features) and outputs into an aggregate, such as averaging or voting, which serves the final output.

Extreme gradient boosting (XGBoost) [84] is a highly scalable, efficient, portable, and parallelisable technique of gradient tree boosting—ideal for our intended host platform where runtime and output performance are critical. It has demonstrated state-of-the-art results in many real-world challenges and environments. It accomplishes this with powerful optimisation and regularisation routines, in an extremely efficient implementation largely abstracted from the user.

Given N training vectors $\{(\mathbf{x}_i, y_i) : i = 1 \ldots n, \mathbf{x}_i \in \mathrm{I\!R}^{\mathrm{m}}\}$, each with M examples + 1 target $\hat{y}_i$, given by basic model

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i) \tag{2.37}$$

using scores $f_k(\mathbf{x}_i)$ of K independent regression trees, its overall objective function becomes

$$\mathcal{L} = \sum_{i=1}^{N} l(\hat{y}_i, y) + \sum_{k=1}^{K} \gamma T + \frac{1}{2}\Lambda\|w\|^2 \tag{2.38}$$

where l herein is some configurable loss function, e.g., mean square error $(\hat{y}_i - y_i)^2$, T is the number of leaves and w their corresponding weight. The second term, given onward as $\Omega(f_t)$ where t is the current training iteration, is a form of regularisation against model complexity as a preventative measure of overfitting, where γ, Λ are two hyperparameters governing regularisation strength. Training additively, as the function out of Euclidean space and allowing traditional optimisation techniques [84] $\mathcal{L}$ now becomes

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} \left[\iota_i f_t(\mathbf{x}_i) + \frac{1}{2} \upsilon_i f_t(\mathbf{x}_i)^2 \right] + \Omega(f_t) \tag{2.39}$$

and with second-order Taylor expansion, it becomes possible to solve other objective functions using

$$\mathcal{L}^{(t)} = \sum_{i=1}^{M} l \left[y_i, \hat{y}_i^{(t-1)} + \iota_i f_t(\mathbf{x}_i) + \frac{1}{2} \upsilon_i f_t^2(\upsilon_i) \right] + \Omega(f_t) \tag{2.40}$$

$$\iota_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{2.41}$$

$$\upsilon_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{2.42}$$

where ι_i, υ_i represents the first and second-order gradient on loss function l. The overall function is then rewritten, expanding Ω as

$$\begin{aligned} \mathcal{L}^{(t)} &= \sum_{i=1}^{N} \left[\iota_i f_t(\mathbf{x}_i) + \frac{1}{2} \upsilon_i f_t(\mathbf{x}_i)^2 \right] + \Omega(f_t) && (2.43) \\ &= \sum_{i=1}^{T} \left[(\sum i \in I_j \iota_i) w_j + \frac{1}{2} (\sum i \in I_j \upsilon_i + \Lambda) w_j^2 \right] + \gamma T && (2.44) \end{aligned}$$

where $I_j = \{i | q(\mathbf{x}_i = j)\}$ is defined as the j-th leaf point of the given structure q. Its optimal weight w_j^* is computed by

$$w_j^* = -\frac{\sum_{i \in I_j} \iota_i}{\sum_{i \in I_j} \upsilon_i + \gamma} \tag{2.45}$$

taking an overall gain into the objective function for tree q:

$$\mathcal{L}'(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j}\iota_i\right)^2}{\sum_{i\in I_j}\upsilon_j+\Lambda}+\gamma T \tag{2.46}$$

Whether or not a particular tree structure improves the overall model (i.e., the gain) is given by

$$\mathrm{G} = \frac{1}{2}\left[\frac{\left(\sum\limits_{i\in I_L}\iota_i\right)^2}{\sum\limits_{i\in I_L}\upsilon_i+\Lambda}+\frac{\left(\sum\limits_{i\in I_R}\iota_i\right)^2}{\sum\limits_{i\in I_R}\upsilon_i+\Lambda}+\frac{\left(\sum\limits_{i\in I}\iota_i\right)^2}{\sum\limits_{i\in I}\upsilon_i+\Lambda}\right]-\Lambda \tag{2.47}$$

on the basis that q is optimised whereby $G > \gamma > 0$. The algorithm begins from a single leaf for each structure and continues to add new branches, so long as the gain is positive. In a full implementation, a model (those built using XGBoost in particular) is comprised of an ensemble of these, each acting as a base learner. Independently, they can be weak performers; ensemble learning combines the outputs of all base learners to generate one overall result [85]. This arrangement allows for more complex relationships between input features and output targets.

2.10.1 Related Work

The vast majority of predictive maintenance research is focused on industrial applications, e.g., factory machinery, aircraft components, and jet engines. Such work is largely unexplored in the context of domestic applications. In fault detection using machine learning, gradient boosting has proven a popular and highly effective technique.

Ensemble learning is a well-researched approach in problems where the results of a single machine learning model may not be sufficient, systematically combining the predictive power of multiple learners working to produce a single output. XGBoost, in particular, has demonstrated success in many fields. The authors in [86] apply the same gradient boosting framework in the cyber security domain in detecting falsified data in network traffic (a common threat in grid/IoT networks). The authors in [87] propose a similar model used to detect degradation in piston pumps, outperforming complex support vector machines (SVM) and artificial neural networks [88].

Deep learning approaches have shown immense success in the same task. For example, [63] develop a complex recurrent neural network framework for detecting (and relabelling false classifications) types of anomalies in spacecraft. In predicting remaining useful life in machinery, a task similar to this, [89] architect a convolutional temporal (recurrent) neural network in calculating the point of degradation in industrial machinery.

Most promising solutions are focussed on industrial challenges and rely on obtaining highly relevant system data, e.g., precise sensor readings, a core requirement in machine learning-based predictive maintenance solutions [90]. Deep learning approaches require particularly vast quantities of training data to generalise well. Our limited dataset inhibits these complex models from learning, and classic oversampling techniques forfeit the temporal nature of our data—a key attraction in recurrent neural networks in particular.

Finally, datasets in most predictive maintenance challenges are often formed of rich, high-frequency sensor readings of vibration, rotation, etc. [91], whereas we are working solely with energy data, which cannot as easily provide the same insight.

Challenges Outstanding

- Datasets are formed of sensor readings of vibration, rotation, etc., whereas we are working solely with energy data, which cannot as easily provide the same insight.

Chapter 3

Autoencoder and Clustering Based Unsupervised Online Anomaly Detection in Cyclic Time Series Energy Data

This chapter presents the design of an anomaly detection system designed to support the intricacies of processing time series energy data from a multistate machine. Section 3.2 provides background on the requirements for such a system, which begins with minimal preprocessing stages described in 3.4. Using a novel incremental clustering algorithm, a complete wash cycle is matched with its closest cycles, if any, in an unsupervised technique detailed in Section 3.3. The cycle, in the form of multivariate time series, is then submitted to an autoencoder system, described previously in Section 2.4, for reconstruction. From there, analysis of the reconstruction and its error can then be undertaken with thresholding explained in Section 3.6 to identify and prune anomalies based on the reconstruction error.

3.1 Introduction

Many machine learning-enabled approaches toward anomaly detection depend on the availability of vast training data, an inherent requirement for most neural network-enabled solutions. Our data is formed from power readings of cycles from domestic appliances, such as dishwashers or laundry machines, and contains no known examples of anomalous behaviour. Moreover, we are limited to the machine's voltage, power, and current readings, drawn from a retrofitted power outlet in 60-second samples. No rich sensor data or previous insight is available as a training basis, limiting our ability to leverage existing work. Our objective is to identify anomalies in power cycles of domestic appliances with no given definition of anomalous behaviour.

We design a system to monitor the behaviour of an electrical appliance with no prior background or knowledge of its manufacturer or make. Before we can begin to address the challenge of online anomaly detection, we must first consider that this system requires special consideration for our data as different power cycles from the same machine can exhibit radically different behaviour. We account for this requirement by clustering unseen cycle patterns into siloed training datasets and corresponding learned parameters. They are then passed in real-time to an autoencoder ensemble for reconstruction-based anomaly detection, using the error in reconstruction as a means of flagging anomalous points in time. The system correctly identifies and trains appropriate cycle clusters of data streams on a real-world machine dataset injected with stochastic, proportionate anomalies.

3.2 System Overview

Our objective with this system is to design a reliable machine learning-based monitoring environment for electrical appliances with varying power cycle patterns, e.g., a laundry machine or dishwasher. We are interested in identifying *anomalies* where we have not formally defined the context and type of anomalous behaviour. We propose using an ensemble of autoencoder neural networks to observe the power consumption of an electrical appliance over long periods, thus learning its normal behaviour between different modes of operation while also identifying anomalies and where they occur. The objective is to anticipate any potential appliance failure, allowing the owner or manufacturer to take remedial action before total device failure occurs.

Our data originates from a retrofitted power outlet that records the voltage, power, and current of a machine sampled every minute. So, given no prior knowledge of the machine's function or capabilities and no access to rich sensor readings commonly leveraged in applications such as predictive maintenance, we must be able to make inferences about anomalous behaviour without defining any context. For example, consider the power the signature of one laundry machine's eco-friendly cycle; it will significantly differ from another machine of the same make, thus eliminating any universally trained model or globally recognised failure patterns as potential solutions. Our working dataset originates from discrete power consumption readings in 60-second samples from several laundry machines of unknown make and model, automatically separated into cycles (i.e., periods of some continuous power activity).

Supervised training requires labelled data, a laborious process to compile manually [92] before considering the vast count of observations prevalent in time series datasets. The heterogeneity of intended hosts for this framework means an anomalous example found in one electrical appliance may be normal behaviour in another. Our requirements begin to form from these challenges, which involve designing a system that operates using unlabelled data streams. Specifically, the system must capably recognise the different patterns of cycles from an unspecified appliance and predict from a set of similar cycles when an appliance may exhibit anomalous behaviour.

Neural networks have become the mainstream approach for many machine learning problems. Anomaly detection has been extensively researched in machine learning, specifically deep learning in the context of time series data.

Although several complex machine learning models have recently shown to work well in time series anomaly detection [63, 93, 94], many of these would simply require excessive processing power and storage than what is otherwise available to most microcontrollers and edge hardware. This task carries its unique challenges, including:

- Cycles and their characteristics are specific to each host machine, eliminating the possibility of a universally trained model.
- Cycles will arrive over time and unlabelled, with no immediately available samples of failures as a basis for development.

Given these constraints, a supervised, deep learning-empowered anomaly detection approach is unfeasible. Our work proposes a novel approach to the combination of unsupervised clustering and autoencoder-enabled anomaly detection. Our solution combines similar wash cycles into their own evolving parameter sets, swappable in and out of the autoencoder ensemble for more accurate learning.

Figure 3.1 illustrates the process of data ingestion through to model execution. Readings of the voltage, power, and current from the cycle are received. The current, offering the most distinct pattern, is selected to compute a fixed-length representation of the variable-length wash cycle for clustering, which, in turn, provides a corresponding parameter set for the autoencoders trained on previous matching cycles. Broken into stages, our system's process for a wash cycle is as follows:

1. A complete wash cycle is received with separate readings for the voltage, power, and current taken once every 60 seconds;
2. The wash cycle input is reduced to a fixed-length vector, where it is clustered into an incremental cluster space using an online adaptation of K-medoid clustering;
3. The cluster associated with the wash cycle maintains its own set of hyperparameters for an autoencoder ensemble, into which the original cycle is fed;
4. The autoencoder ensemble reconstructs the input to the best of its ability, with each parameter set assumed to be trained on samples with no anomalous be-

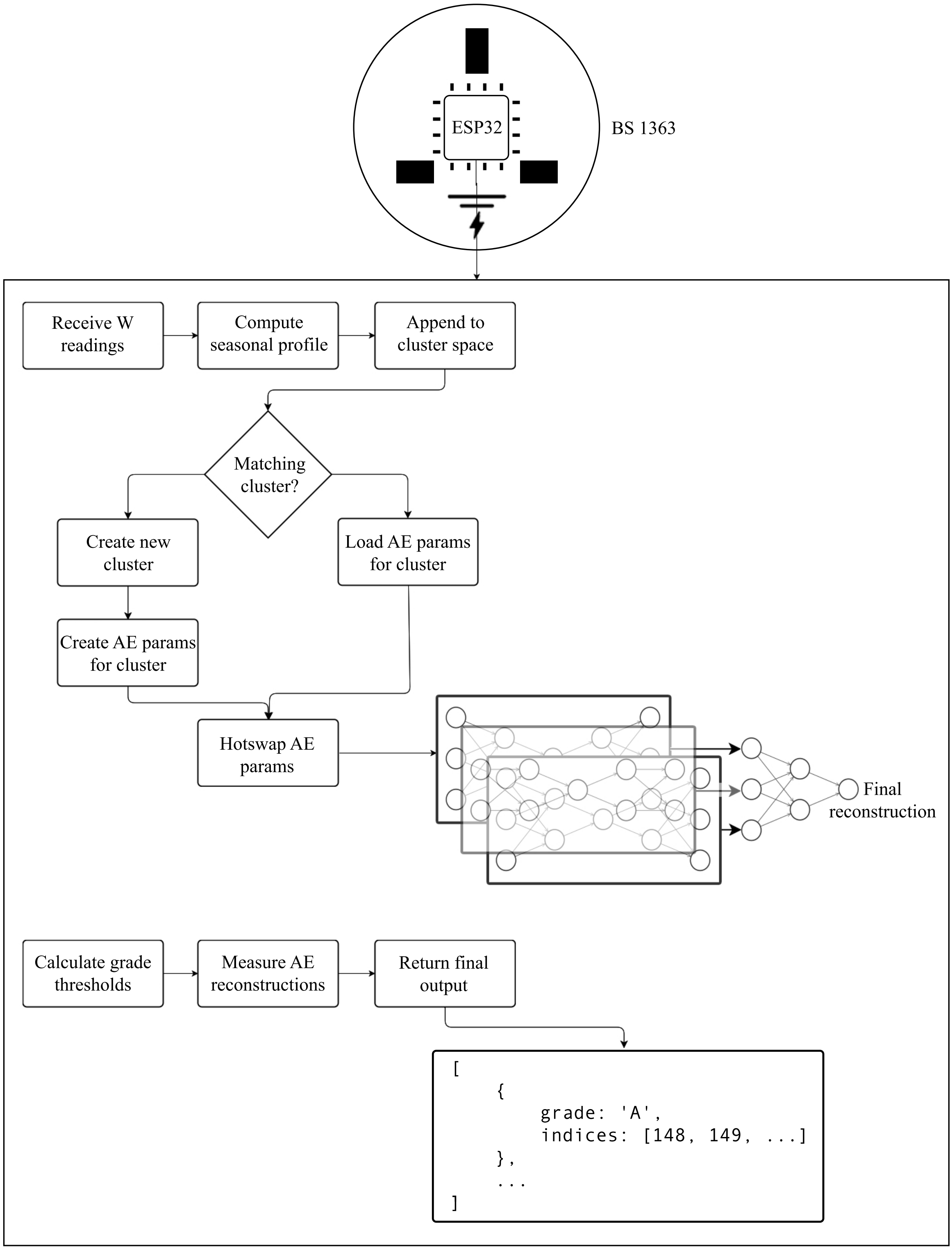


Figure 3.1: Flowchart of the proposed system. We first pass the wattage readings through the clustering process to retrieve or create the matching parameter set, then install them in the ensemble stage. All values are eventually submitted raw to the ensemble.

haviour;

5. Deviation between the original cycle and the reconstruction is measured, where large reconstruction errors indicate abnormal behaviour, returned as points in time;

6. Higher-level parts of the system can assess output and make decisions on informing users and manufacturers.

This chapter first presents a novel adaptation of Partitioning Around Medoid (PAM) clustering [95]: a clustering algorithm that maps a distance matrix into K clusters. The algorithm is adapted to operate on an evolving dataset without relying on complete access to the dataset or any predefined notion of K to cluster effectively. Instead, we introduce a tolerance parameter to determine the maximum allowable distance from a cluster before a sample is considered a medoid of its own. The cycle, then clustered and matched to a learned parameter set, is forwarded to an ensemble of autoencoders for reconstruction, which swaps its training parameters ad hoc depending on the cluster matched to the cycle. The key attraction of this approach is being able to account for operation cycles that differ in nature, helping prevent the system from falsely classifying normal behaviour of one cycle as the anomalous behaviour of another.

The proposed system aims to address these concerns by assigning the complete reading to an evolving data cluster of similar cycles—or forming a new cluster unsupervised if the cycle is sufficiently distinct—which maintains its own evolving dataset, without requiring the entire historical energy readings of the cluster. The system is interoperable between clusters of different cycles, and the prediction weights can be hot-swapped ad hoc. The implementation is prototyped in Python 3.*, with support from standard scientific computing libraries (NumPy, Pandas).

3.3 Incremental Online Clustering

The first stage of the system when given a wash cycle is to assign it a cluster of similar cycles—or form a new cluster where the cycle is sufficiently distinct. Different wash cycles will have different power consumption patterns; and laundry machines will have different power consumption patterns on the same wash cycle. One key challenge in our objective is to account for this in an unsupervised fashion. The majority of clustering algorithms—although technically unsupervised models—frequently require some predefined constant for the cluster count (K).

How similarity is measured and defined is a major decision for this system. The algorithm must be sufficiently robust to account for sometimes critical anomalies in a signature that would have a place in an existing cluster if deemed normal. Acting too inflexibly on an anomalous wash cycle may needlessly cast it into a new cluster of its own; likewise, being too flexible may miscategorise an anomalous cycle as a regular cycle of a completely different cluster.

Associated with each cluster is a parameter bank (weights, biases, training records, etc.) on which the autoencoder trains and executes.

Although later stages of the system are capable of processing in real-time, we first require a matching cluster of similar cycles on which to evaluate, where each cluster maintains its own learned parameter set to use in the autoencoder ensemble. We receive our data for this experiment as an array of distinct wash cycles, containing the voltage, power, and current, and voltage for each minute the cycle is in progress, ranging from 30-240+ minutes in length.

3.3.1 Existing Techniques

Many existing techniques assume a complete dataset or predefined cluster count before clustering. Conventional offline K-enabled clustering techniques cannot guarantee reproducible results when repeated, which disrupts the later stages of this system; we may

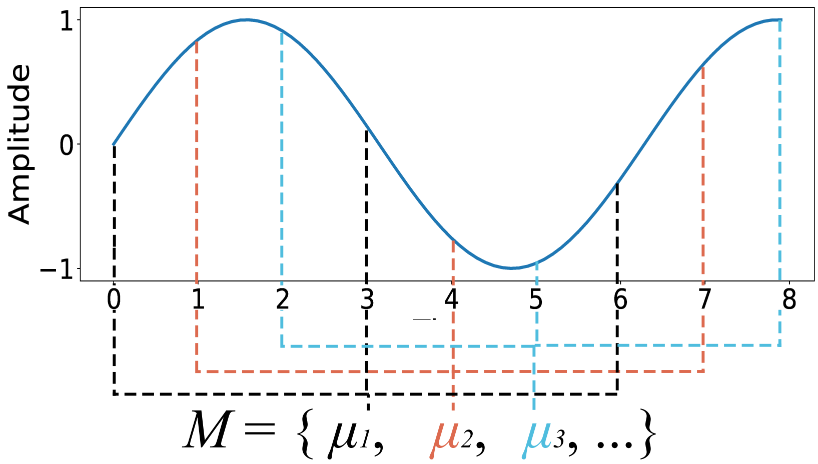


Figure 3.2: Computing the mean seasonal profile of a time series of variable length. For a dimensionality of N, each $(i \cdot N)$-th point of the time series is averaged into the $i = 0, 1, \ldots, N$-th index of the fixed length vector.

lose or rearrange clusters in the future with these techniques. PAM, an adaptation of K-means clustering, developed by [96], selects actual instances from the data as medoids, whereas K-means generates artificial points in the cluster space as centroids [97]. With its flexibility toward how objects are defined as close [98], a favourable technique for our use case.

3.3.2 Tolerance-Based PAM

Each observation is first scaled using zero-mean normalisation (or z-score) to help prevent one feature of a data point with a range different from another from becoming dominantly influential during training. More importantly, it limits noise impact and makes for accelerated convergence during training.

A time-series representation is next computed and piped directly to the clustering stage. Opting for a statistical representation of the time series is a promising approach, as it can account for disturbance in the series and still draw out the essential characteristics of data [99]. Our system uses the *mean seasonal profile* algorithm, which converts a time series to a length of set seasonality, adapted from [100].

The process compresses the z-scored vector to a fixed size N, then processes according to Figure 3.2, illustrating the algorithm against a series of variable length. This mean seasonal profile features the mean of every $i \cdot N$-th element at index $i = 0, 1, \ldots, N$ throughout the series. The representation can also be considered as a form of dimensionality reduction, which benefits memory requirements and computational complexity.

The mean seasonal profile approach is a model-based algorithm, which assumes the data processed originates from the same source [100]. Given the nature and origin of our data, we can safely base our assumptions on a similar premise.

We modify the process of PAM, which typically begins by building a set S, composed of k objects, as the initial medoids in the cluster space. The selection process for medoids can be randomised or conditional. In parallel, another object D records and tracks the distances between the medoids and child clusters of S. Each addition to the cluster space requires D to be recalculated to include the new distances after assigning the data to the nearest cluster. The next phase clones the cluster space, selecting an alternative set of k medoids, and computes the inter-cluster distances D of candidate medoids S. Should this version offer a sufficient gain in affinity over the existing cluster space, it replaces the existing arrangement. This process is repeatable any number of times with each addition.

The system described in this section implements an adaptation of PAM, where predefined K is replaced with a tolerance parameter to, depending on the distance from the nearest existing cluster, selectively partition the dataset into new clusters. Unlike many clustering algorithms, data are instead submitted incrementally to the cluster space. Its distance is evaluated against that of the closest medoid. As with original PAM clustering, distance is flexible in how it is measured, and the tolerance can be adjusted to the sensitivity of the expected data.

We apply ℓ^2 norm, or Euclidean distance, as our distance function d between two points in the cluster space, given as

$$\mathrm{dist}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{\frac{1}{2}}, \tag{3.1}$$

where $\mathbf{x}, \mathbf{y}$ are two data points of equal length N to compare; both are z-scored, mean-seasonal profile vectors previously described in Section 3.4. When appending a wash cycle to the cluster space, the new distance after matching its nearest medoid is evaluated against the predefined tolerance parameter. If the wash cycle is within distance, it is appended to the cluster. Exceeding the tolerance, or if in an empty cluster space, the data point is designated as a new medoid. D and S are each recomputed as with traditional PAM implementations to include the new data.

The tolerance value becomes another hyperparameter of the system, as it determines the flexibility of the cluster space. The value must be forgiving enough to provide sufficient variety for clustering without losing the representation of the cycle. Likewise, setting too low a tolerance may cast a true anomalous as an outlier and into a new cluster.

3.3.3 Operational Complexity

PAM has a runtime complexity of $O(k(n-k)^2)$, where n represents non-medoid data points and k the count of medoids in the cluster space, as $n-k$ data points must be distanced k times to compute the lowest change in cost. With our online adaptation, the runtime complexity becomes $O(k + (nk))$ per iteration, where k medoids are first traversed to find the nearest medoid, and then each data point is iterated to again find its nearest medoid, before evaluating the gain in fit were it placed in each candidate medoid. Although acceptable in our experiments and evaluation, the runtime will increase as the cluster space continues to populate over time.

3.4 Data Preprocessing

This section presents the stages of our proposed system and details its implementation. In complex applications, raw data alone is often unsuccessful in modelling some phenomenon from its features. Data preprocessing can significantly impact the generalisation performance of an algorithm—particularly in more complex models. This step is not unique in anomaly detection challenges; data pre-treatment is a core component of machine learning and a field of research in and of itself [101].

The input stage begins with a complete cycle, submitted as triaxial readings of voltage, power, and current in 60-second intervals. We do not consider or derive any features from timestamps or idle periods between. The readings of a wash cycle are first passed through a clustering process, which identifies the cycle, designates an identifier for later stages, and installs a corresponding learned parameter set into the autoencoders for reconstruction. To enable this clustering process, this stage first builds a representation of the cycle, the process for which is elaborated in Section 3.3. If a matching cluster exists, the cycle in question is evaluated against, or contributes to, the learned behaviour of that cluster from an autoencoder ensemble; if not, a new cluster is created along with a counterpart parameter set, consisting of the following:

- Weights, initialised randomly when first instantiated;
- Biases, initialised as zeroes;
- Variables for normalised minimum and maximum of data observed during training phases for the parameter set.

The system maintains the above set of learned parameters for each cluster, which are hot-swapped in and out of the subsystem for a given wash cycle. The clustering process described in Section 3.3 outputs an identifier (ID) for a corresponding cluster. Control then flows to a manager to fetch the parameters and data stream counterparts for the given cluster. In a newly formed cluster, these learned parameters are instantiated randomly.

The autoencoder's readings at each sample of the power profile are then fed individually to the ensemble, which outputs an approximate reconstruction of the signal. Following a complete forward pass, the reconstruction is compared against the original, using the theory that anomalous cycles will thwart the autoencoder's ability to reconstruct the input accurately.

Each cluster begins its corresponding parameter set under a fixed grace period, given as a hyperparameter to the model and constant between clusters, during which all data submitted during this time are used solely as training data for the autoencoders of the particular cluster. A cluster still in its training phase, i.e., within the grace period, will not offer any predictions.

The learned parameters of the autoencoder ensemble, described in Section 2.4, are hot-swapped with those of the cluster matching the wash cycle.

We measure the reconstruction against a set of user-defined severities, given as degrees of deviation from a threshold calculated using light statistical techniques on the errors made by the autoencoder's reconstruction, described in Section 3.6.

$$\mathbf{x}_{z\text{-score}} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \tag{3.2}$$

Both inputs (coordinates of the cycle to be clustered and those of the existing cycle within the cluster space as comparate) are z-scored outputs of mean-seasonal profile vectors, previously described in Section 3.4. When appending a wash cycle to the cluster space, the distance calculated after matching its nearest medoid is evaluated against the predefined tolerance. If the wash cycle is within distance, it is appended to the cluster; if it exceeds the tolerance or if the cycle is placed in an empty cluster space, the data point is then designated as a new medoid. Selecting an appropriate tolerance value, therefore, becomes a hyperparameter of this solution. D and S are both recomputed as with traditional PAM implementations to include the new data.

The z-scored vector, equal in size to $\mathbf{x}$ then undergoes a final processing stage to approximate the original time series by transforming the time series into a new representation: the mean seasonal profile, adapted from [100]. Vector $\mathbf{z}$ is compressed into a uniform size as the mean of the elements, as calculated in Algorithm 1. The mean seasonal pro-

Algorithm 1 Calculate median seasonal profile $\mathbf{r}$ using time series $\mathbf{ts}$ and frequency p

Require: $p > 0 \vee p \leq \text{len}(\bar{t})$

\# normalise series
$\hat{\mathbf{ts}} \leftarrow \sum_{i=1}^{n} \text{ts}_i^2 \,/\sqrt{\mathbf{ts}}$
\# result of size p, filled with zeroes
$\mathbf{r} \leftarrow [0] \cdot ||\hat{\mathbf{ts}}||$
\# declare indices to target
$\mathbf{ind} \leftarrow []$
\# calculate step size across time series
$\text{frq} \leftarrow ||\hat{\mathbf{ts}}||/p$

for $i \leftarrow 0$ to p **do**
    for $j \leftarrow 0$ to frq **do**
        \# push adjusted index for i-th stage of total length p
        $\text{ind}_j \leftarrow (j \cdot p) + 1$
    end for
    \# populate i-th stage of profile with mean of series at adjusted indices
    $\text{r}_i \leftarrow \sum_{i=1}^{p} \hat{\text{ts}}_i / p$
end for

return $\mathbf{r}$

file approach is a model-based algorithm, which assumes the data processed originates from the same basic source: an ideal assumption we can safely make given the origin and domain of our cyclic data.

3.5 Autoencoder-Based Reconstruction

Associated with each cluster is a parameter bank (weights, biases, training records) on which the autoencoder trains and executes. With a parameter set from the cycle's belonging cluster, each multivariate reading is submitted to the autoencoder ensemble incrementally. Where the parameter set is sufficient in size to execute the autoencoder, the outputs are returned in a vector for analysis. A parameter set still in its training phase will not make any prediction. Each cluster begins under a fixed grace period for each clustered parameter set, during which all data submitted are reserved as training data for the autoencoders. Selecting the grace period is an important decision, as the consuming system must balance allowing a representative dataset for a particular cycle to grow organically with the need to begin making predictions and assessments as soon as possible.

Autoencoders trained well on non-anomalous data will approximate the target with relatively low error rates but struggle to reconstruct an anomalous signal to the same accuracy [58]. This phenomenon is the underlying theory of this stage of the detection system. Each autoencoder in the ensemble trains in silo on at least one feature of the stream, which is compressed down to a latent representation in an intermediate layer. This compression ratio is a configurable parameter to be supplied by the user. Features are repeatable across multiple autoencoders, i.e., the i-th feature of a multivariate series observation may be input to ≥ 1 autoencoders.

Each observation is passed raw directly into the system one at a time and normalised to a range of $[0, 1]$ using min-max scaling. To avoid storing the entire observation set, a record of the smallest and largest values is maintained for each feature observed during training stages to date.

The output stage of the ensemble described is itself another autoencoder, trained on the collective outputs of all preceding autoencoders within the ensemble. We assume for our system that a wash cycle submitted for training on a new or developing cluster is not anomalous. Following the grace period, the system executes the autoencoders on all future data received, returning the reconstructed value and the measured loss according to the RMSE score, given previously in Equation(2.9). In training, the RMSE is used to drive the classical backpropagation supervised training technique for only the autoencoders.

As clusters form at different rates, one cluster with sufficient training will transfer into execution. Swapping to a cluster still in training automatically toggles training mode for the autoencoder ensemble.

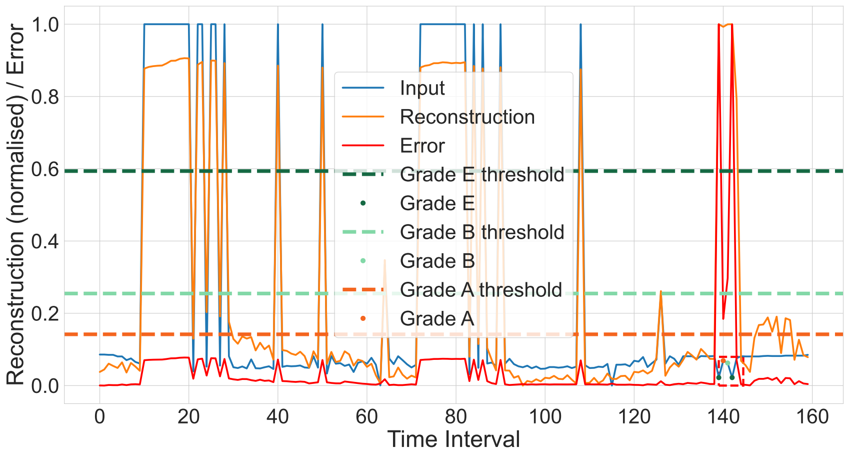


Figure 3.3: Calculated thresholds for identified grades respective to the range of the inputs, dashed horizontally in their respective colours, for a wash cycle against an artificial anomaly, outlined in red. The autoencoder stage should struggle to reconstruct (orange) this region of the input (blue). Points along the input mark the exact samples found anomalous and by what grade.

3.6 Post-processing Identification and Grading

The final stage of the system is where anomalies are identified and flagged. Here, we receive the output of the autoencoder ensemble for each observation in the wash cycle and compare it to the original input. Treating the scores as face-value indications of anomalies produces excessive false-positive identifications, diminishing the reliability of the system. This stage is introduced as a measure to prune and grade the ensemble scores into definitively ranked anomalies for review.

We begin by calculating the absolute error between the input and reconstructed output at each time step into a separate vector, e. Then, its severity is measured using a set of severity grades, calculated by $\mu(e) + (\sigma(e) \cdot g)$ where $g = 1, 2 \ldots N$ and N denotes the maximum degree of separation from the input. For interpretability, we return grades as an uppercase letter A – Z. Any absolute error score from e exceeding the threshold appropriate to the current grade is recorded as a dictionary wherein object keys are the grades, if any, and the corresponding values a set of intervals considered anomalous at a particular grade. Figure 3.3 illustrates this process, showing the relative thresholds for each severity grade. A cycle deemed normal can be resubmitted for retraining the cluster's parameter set at the user's discretion.

This grading mechanism does not consider transposition in its decision, allowing reconstruction to focus on the overall energy pattern rather than relying on exact matches. We are particularly concerned on noise and significant deviation from the ensemble's reconstruction in our detection criteria.

3.7 Evaluation

Table 3.1: Total number of readings in the dataset, which will be submitted incrementally to the system.

	Point evaluation	Strict evaluation
Population	1961	2324
True positive	51	70
True negative	1910	2254
Outcome P	95	135
Outcome N	1866	2189
False P	44	65
False N	0	0

Real-world power consumption data served throughout the design, development, and evaluation of this system. Our dataset for the detailed evaluation is a set of 62 individual wash cycles from the lifetime of a single laundry machine, collected organically over 6 months. As this system is designed to monitor a single machine, in this evaluation we do not introduce or aggregate multiple machines or their cycles. Each 60-s reading of an in-progress cycle contains the timestamp, voltage, power, and current. The system receives each wash cycle incrementally and not as a batch. In total, we have 2,324 readings, with each wash cycle running 41 min on average. Our final data population counts, shown in Table 3.1, are inclusive of anomalies injected.

Data are submitted raw as one complete cycle where it enters the clustering stage. When the autoencoders are swapped to the parameters of the appropriate cluster, the cycle is reconstructed reading by reading. The results in this section are produced on a dishwasher power graph of 75 uniquely identified cycles across 314,464 min-interval timesteps. The majority of this is standby power—idle readings are not submitted to the system.

We set our tolerance for clustering wash cycles to 0.05 and found a seasonality of size 20 to be effective in calculating a representative mean seasonal profile, the result of which is what is used in the cluster space. The autoencoders are each assembled into a 3-layer mapping and given a grace period of 320 observations for each cluster. This length was determined as the rounded average of 2-3 complete wash cycles of training per cluster before execution. The autoencoders are configured to compress input to a latent representation of 80%, using a learning rate of 0.03, as seen in other works.

3.7.1 Data Setup

The system expects disturbances in the power consumption characterised by prominent distortions; therefore, we include the novel ability to inject stochastic noise into some segments of a wash cycle in our testing framework. Anomalies are artificially injected at a rate proportional to the rest of the signal to carry our example, with a configurable variable to adjust the noise intensity to resemble the anticipated behaviour of their real-world domain.

Faced with a highly imbalanced dataset and scarce examples of true anomalous data, white noise injection is a familiar training tactic in the anomaly detection space. For example, the work in [102] explores the condition monitoring of UAV motors by injecting white Gaussian noise into their dataset as a training label, at a configurable signal-to-noise (S/N) ratio. Using noise in training is also seen in [103], where the authors assess the robustness of training deep learning models under synthetic noise to develop an anomaly-detection system into an unsupervised setting, again with a configurable S/N ratio, ranged proportion, and evaluation strategy. More similarly, the work in [104] explores datasets of a similar nature, bearing local fluctuations as formulaic anomalous periods injected into multivariate time-series datasets. Their work develops a new algorithm to extract feature vectors for training on existing machine-learning-enabled anomaly detection frameworks.

Our approach follows similarly to these works, first defining a probability of a wash cycle containing an anomaly, set at 40%. If selected, a wash cycle is injected with noise across 15 ± 3 readings of the cycle. We use additive white Gaussian noise as our noise model and a S/N ratio of -0.2 dB. The wash cycle is then modified in place and recorded in an external data store inaccessible to the system, marking the timestep in the cluster and the breakpoint of the anomaly for evaluation, forming our label for the cycle in the evaluation process. The "input" line in the graphs of Figure 3.4 shows the original power-consumption pattern, with an anomalous region outlined in red to illustrate the nature of the disturbances the system should identify.

3.7.2 Evaluation Strategy

This injection strategy does not necessarily mean that 40% of our preprocessed data becomes anomalous. With an ad hoc source of truth of injected anomalies, measuring detection performance is essentially reduced to binary classification. There are two potential strategies of performance evaluation, depending on the context of the anomalies expected and how they are characterised. First, the presence of any anomalous point alone may be sufficient. In this point-based approach, any one point within the range of the complete anomalous period found to be anomalous by the system is considered a success.

Alternatively, another application of this system may require a series of anomalies together first be identified before considering any action. In this approach, we require each point within the entire anomalous period be individually identified as anomalous by the system; points not correctly identified in this range are considered false negatives

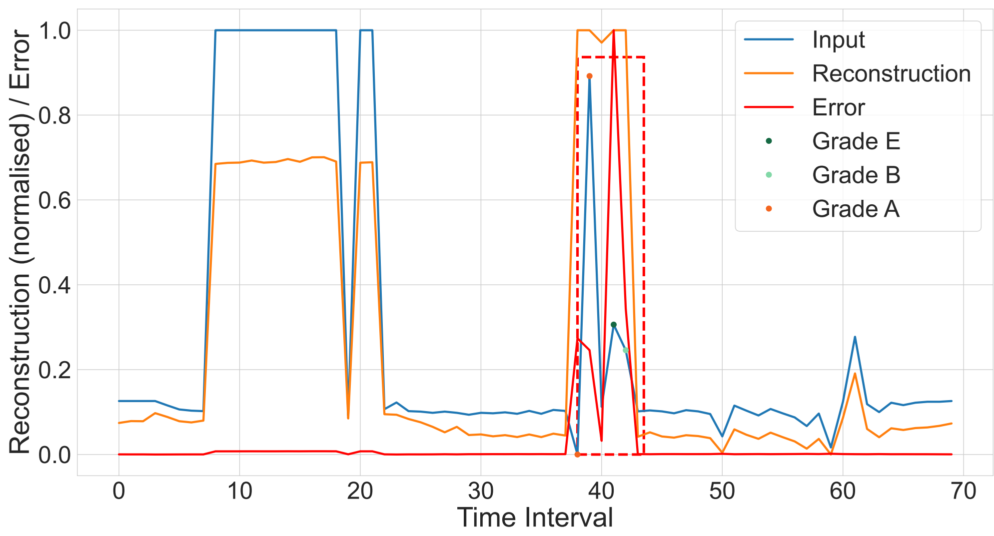


Figure 3.4: Reconstruction of a wash cycle, shown in Watts. The anomaly injected is marked by the dashed red border. The autoencoder should struggle to reconstruct the anomalous region marked in red—which the grading mechanism should identify.

and will negatively impact the detection score. Therefore, we present dual evaluation strategies as:

- Point, requiring any one of the points in the injected anomaly be identified as anomalous;
- Strict, requiring all points in the injected anomaly be identified as anomalous.

Using the wash cycle graphed in Figure 3.4 as an example, the strict evaluation approach requires all of the data points within the known anomalous region (bordered in dashed red) to be identified at any grade to be considered completely successful, whereas the former approach will accept any anomaly identified within the region as an overall success.

3.7.3 Results

This evaluation runs both strategies under two versions of the system: the original, as proposed in this section, and another that bypasses the clustering stage, i.e., all of the wash cycles fed to the same autoencoder parameter set. Each version is executed in silo on the same dataset, meaning it develops its parameters separately from the other versions of the system. The clustering stage will reserve some of this data as training for each identified cluster, whereas the continuous examples required will train their sole cluster once.

Table 3.2: Results of the simulation experiment ran on all versions.

	Point evaluation	Strict evaluation
TP	1	1
TN	0.977	0.971
FP	0.023	0.029
FN	0	0
Discovery	0.463	0.482
Precision	0.537	0.519
Accuracy	0.978	0.972

From the results logged in Table 3.2, we observe the strict evaluation strategy expectedly produces relatively lower results than its more forgiving point counterpart, given that a single anomalous point within the injection range qualifies as successful, whereas requiring all anomalous points detected in the range injected is naturally more challenging for the framework. However, the results are still marginal in comparison. Both approaches demonstrate together that separating data streams offered by the clustering stage is beneficial to overall performance, which is beneficial under strict evaluation. However, clustering will require a fixed series of initial readings for each data stream before it can begin to offer predictions; hence a lower population in the results ran without the clustering stage.

This evaluation was conducted with a 0.2 clustering tolerance and a mean seasonal profile length of 20. Autoencoders were constructed as a 3-layer mapping with a learning rate $\eta = 0.09$, and a latent representation of 80%, a common ratio in autoencoder applications which displayed the most optimal performance in our ranged cross-search. Figure 3.3 dashes a red box around the injected anomalous area, showing the exact

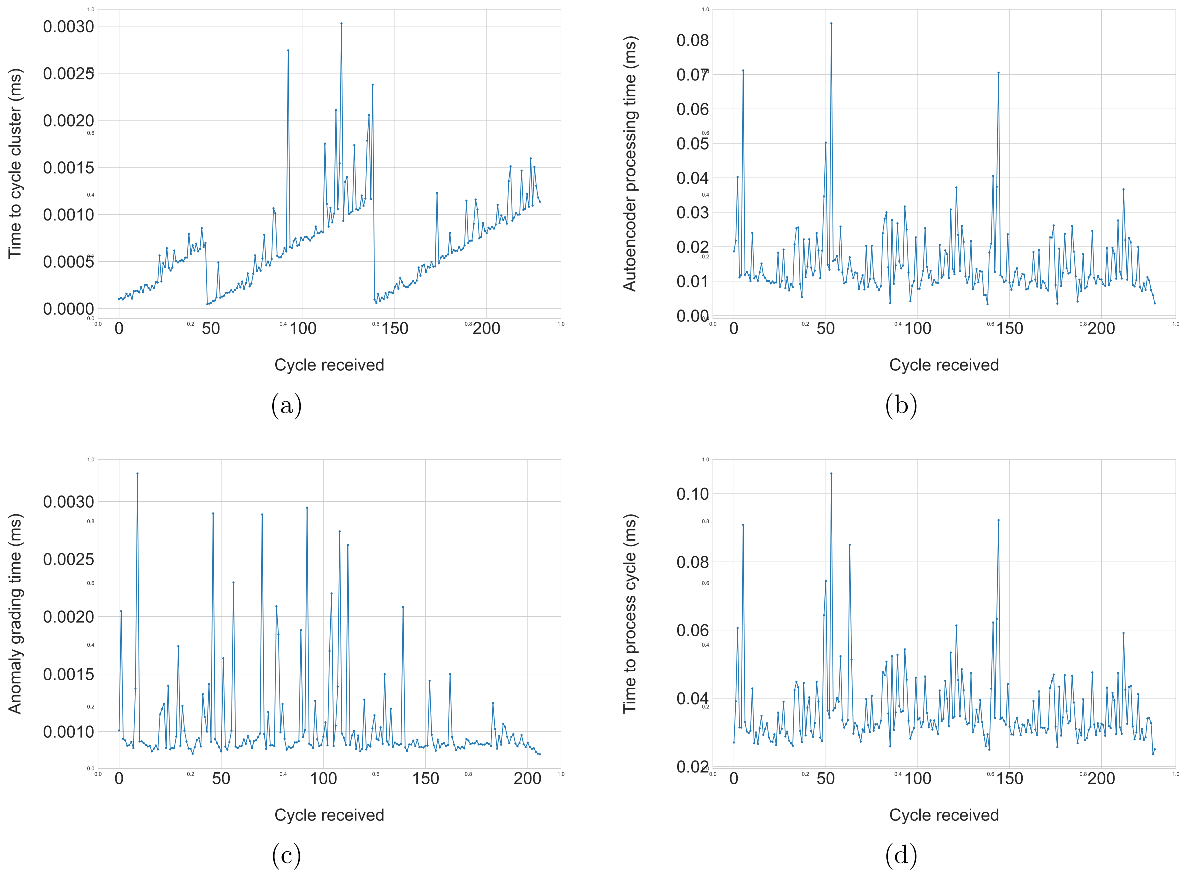


Figure 3.5: Performance graphs of the system recorded during the evaluation. (a) shows the processing time taken to cluster a cycle into the existing cluster space, (b) shows time taken to run the cycles through the autoencoder ensemble, (c) shows the time to grade anomalies received from the autoencoders, and (d) shows the total runtime between receiving a cycle and returning identified anomalies.

reconstruction points marked as anomalous. The sharp increase along an expected idle period sufficiently disrupted the autoencoder's reconstruction, reflected in the RMSE scores.

Figure 3.4 illustrates an excerpt of the post-processed output of a wash cycle injected with an anomaly at a random point, bordered in dashed red. The system should identify from the input (Watts, in the case of Figure 3.4) any or all points within this region as anomalous, depending on the evaluation strategy. The output shows the different lengths of each cycle, demonstrating that the system is not confined to rigid data shaping commonly found in classic machine learning approaches. In both examples, the sharp increase along an expected steadier period in the cycles sufficiently disrupts the autoencoder's reconstruction, reflected in the running red error line. Figure 3.5 illustrates the system performance during the experiment, showing the evaluation metrics used in both point and collective evaluation strategies. The grading mechanism identifies each of these points well. Owing to the bespoke problem we have aimed to solve, it has proven challenging to source or synthesise a comparable public dataset to use in these more complex networks that can still make good use of our incremental clustering stage.

This evaluation runs both strategies under two versions of the system: the original, as proposed in this section, and another bypassing the clustering stage, treating data received as one continuous stream (i.e., a single cluster). Each version executes upon the same data stream in silo, where anomalies are injected externally. It should be noted the clustering variants must reserve training data for each identified cluster, whereas the continuous requires no training for additional clusters. Both approaches demonstrate a worthwhile improvement offered by the clustering stage to overall performance, which is particularly beneficial under strict evaluation.

Runtime performance is another prominent consideration of the system. To evaluate time and space performance, the system was non-obtrusively monitored throughout its execution of various stages by wrapper blocks timing the code's performance in two core stages of the workflow: the clustering of a cycle and the overall processing throughout the rest of the system (including autoencoder reconstruction). These measurements are illustrated in the timing performance graph in Figure 3.5.

We observe that the majority of the system scales well as input continues, yet a slight incline appears as the system's parameter sets evolve. The cause of the performance scaling is the clustering stage—an expected outcome given its runtime complexity, described previously in Section 3.2. As wash cycles continue to populate the cluster space, new submissions will take expectedly longer to place. A potential remedy we are interested in exploring is a mechanism of selectively thinning intense clusters. Should performance become noticeably impacted, we could assess the largest cluster(s) of the feature space and delete some of the members within, freeing up space with limited impact on the cluster's significance and strength. As it has already trained the parameter set, the data and learning experienced by the autoencoders from these cycles would not be, only their presence in the earlier cluster space would be deleted.

3.8 Conclusions and Future Directions

This work describes the design and implementation of an approach toward the identification of unlabelled, undefined anomalies within cyclic time-series data—in our case, wash cycles. Supporting the inherent heterogeneity of different laundry machines, wash cycles, and their varying power-consumption patterns disqualifies classical training approaches on even a rich offline dataset, as the system must begin training from the first reading with few to no prior assumptions.

Through the hot-swapping mechanic, triggered by unsupervised clustering, the system designed can monitor and predict from sometimes radically varied data from the same machine. The system is easily optimised for the general nature of any data with high-level parameter tweaks, as shown in our evaluation. On evaluating the system based on both point and collective anomaly detection, the system demonstrates competent performance despite running through only a single pass of incoming data.

In considering the transfer detection of the proposed method between different work conditions or different electrical appliances, the user has the option to adjust the training grace period and cluster threshold tolerance within the system, or the traditional hyperparameters of the autoencoder ensemble, should the need arise. From our experiments, including the above, we observe similar performance between different machines without adjustment to these parameters. As part of our future directions, we would be interested in obtaining a similar dataset from, e.g., a dishwasher on which to rerun this experiment.

In terms of future work, our attention would next be paid to the time complexity of the incremental clustering algorithm. Although lightweight and storing cycles only in their condensed form for the cluster space, it may eventually become too intensive to process on edge hardware. A preliminary thought is to develop a thinning mechanism, which prunes high-intensity clusters to reduce the dataset size whilst preserving the identity and strength of the cluster space. Additionally, we would be interested in exploring the performance increase when threading the autoencoder ensemble over multiple cores. For machine-learning-related endeavours, we would be interested in appending a classifier to the end of this pipeline, which would classify anomalies in the context where they are known.

Chapter 4

Online LSTM-Enabled Temporal Point Process Modelling for Idle State Detection

The objective of this research is to develop an online load-sensing machine learning (ML) model capable of adapting to new types of appliances and applications, e.g., from laundry machines to dishwashers. Given the end of some discrete event, this model should predict the interval between the beginning of the next in a task commonly known as *time to interval.* Event sequencing is a common application in areas such as commerce, social networking, and finance [105]. For example, an e-commerce store may be interested in the likelihood and time of the next purchase to be made by a customer, given their buying history, or a financial institution may be interested in studying the dynamics of high-frequency historical trading prices [106].

Our objective is to estimate the time elapsed between using an electrical appliance which typically enters into a standby state when idle, such as a laundry machine, television, or dishwasher. Based on the interval forecast by the model, the appliance can be placed into a low power state—or shut down completely—to conserve energy and prolong its lifespan, reducing short and long-term costs to both the owner and the manufacturer. Although environmental impact is a factor in buyers' decisions, inciting change in consumer behaviour and attitude toward energy consumption has little effect, even with an emphasis on the financial and environmental benefits [107] it brings. Our model strives to facilitate this change toward lower energy consumption, fostering more energy-efficient households.

Recurrent neural networks appear initially the strongest candidates to drive such a model, given their inherent and powerful ability to consider long-term dependencies from past observations when presented with unseen input. Such capability is likely needed to uncover the complex relationships and dynamics in a user's typical habits. For example, they may begin a cycle each morning, every second afternoon, each Wednesday of the week, or during weekends, etc. However, no proposed solution we have investigated to date addresses the unique needs of our challenge, to:

- Recondition the model over time as usage patterns change with value/concept drift;
- Operate locally on low-resource edge hardware;
- Predict without assumptions on the underlying dynamics of the host machine.

4.1 Toy LSTM-Powered Time Delta Forecast

Recurrent neural networks appear initially the strongest candidates to drive such a model, given their inherent ability to consider long-term dependencies from past observations when faced with unseen input. Such capability is likely needed to uncover complex relationships in a user's typical habits. For example, they may begin a cycle each morning, every second afternoon, each Wednesday of the week, or during weekends, etc. Further, for the model to reside on edge hardware, it must be computationally lightweight and retrained adaptively. This is a challenge for recurrent neural networks and LSTM cells in particular, as the training algorithm (backpropagation through time) is considerably more intensive than that of a traditional neural network and should be minimised. There exists similar research into collective prediction, including the work of [108], who predict the daily routine of a user over 24 hours using LSTMs and activity classification to estimate the time and class of the next activity. [109] also employs an LSTM in business process monitoring to predict the commencement time of an event by including temporal factors as input to the LSTM.

To validate the concept of using recurrent neural networks in a point process context to sample intervals, a stateful LSTM network was trained and executed online using the elapsed intervals between a 90-cycle dataset over a single epoch, ranging from 1–1,425 minutes in elapsed durations. The model produces overall poor results, attributed to the vast data required to effectively generalise an LSTM. Were we to pursue this model further, our attention would turn to the feature engineering stages. Instead of raw time or deltas, we would explore exogenous time data (minute of hour, of day, of week, of month, of quarter, etc.) as input with an expected idle period as output, though this still may not be sufficient to compensate for the lacking data.

This novel exploration established our basis for going forward in the design of this system.

4.2 System Design and Architecture

The model is architected as a multilayer set of 3 stacked LSTM units each of size 32, formulated previously in Equations 2.18 through 2.23, where the input to layer i is given by $x_t^i = h_t^{(i-1)}\delta_t^{(i-1)}$ where $i \geq 2, \delta = 0.1$. We find this offers sufficient complexity in the model for it to learn effectively and at a quicker pace. The last sequence output from the final LSTM is then propagated to a layer to a mapping size of 16 before passing through a rectification unit. This output is trained as the suggested time interval until the next event, allowing for direct sampling from the model. To derive a timestamp from the interval output by the system, it is simply added to the timestamp of the last input sequence. This process is illustrated in Figure 4.1, showing a timestamp outputted along the continuum, derived from its event history. The system catalogues

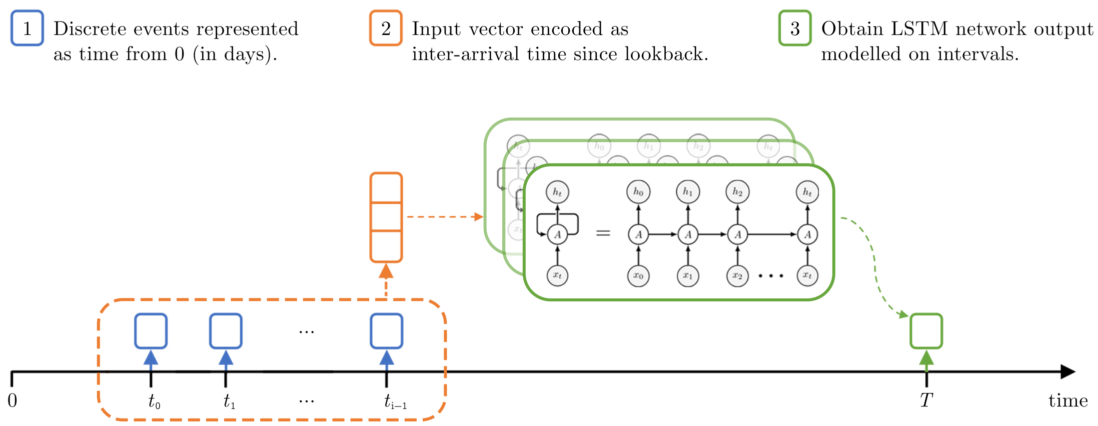


Figure 4.1: Overview of system. Discrete events in time are indexed from 0, collected into vectors and processed throught the LSTM networks, outputting an interval.

events along a continuous timeline, indexing them as having occurred since time elapsed from 0 (the beginning of observation). When the system is asked to make an assessment, the previous arrival times, ranging from the lookback interval until present time, are first loaded and encoded into a vector.

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \epsilon \frac{\{l_1, \ldots, l_N\}^\top}{N}, \quad l_n = |x_n - y_n|, \tag{4.1}$$

MAE, as given in Equation 4.1 where $\mathbf{x}$ and $\mathbf{y}$ are arbitrarily sized vectors and N is the batch size, is favoured as the loss function of this model for its gentler approach toward excessive outliers in the overall training batch. The model should not penalise outliers with , as usage patterns are prone to change.

Training is scheduled in batches of 64 across five epochs, optimised using the Adam stochastic optimiser [20] with a learning rate 0.001, an L2 penalty term of $1.0 \cdot 10^3$, and a dropout rate of 0.1 between all stacked LSTM layers. Given that base training is undertaken offline, we schedule a batch size of 64 for the volume of the dataset and to allow the model larger steps in converging.

4.3 Data Preprocessing

Our global dataset in this application comprises energy signatures from over 950 electrical appliances collected in minute intervals. Each appliance is formatted as a series of incrementing timestamp intervals between uses and contains approximately 100 timestamps. We do not combine or aggregate cycles from other machines in our training or evaluation stages; examples follow a single machine throughout its lifetime. Part of our objective includes training for platforms of a similar nature under loads varying from infrequent to continuous. We make no distinction between electrical appliances in the dataset, and therefore type of appliance (laundry machine, dishwasher, etc.) or any information regarding its manufacturer or components within are withheld from the model.

Conscious of the intended host platform of this model, we undertake as little pre and post-processing as possible—another reason for having the model directly output raw intervals.

Algorithm 2 Preparing dataset $\mathbf{X}$ for batch training, constructed according to lookback l.

$\bar{\mathbf{X}} \leftarrow []$
for $i \leftarrow 0$ to $\|\mathbf{X}\|$ **do**
  for $j \leftarrow 0$ to $\|\mathbf{x}_i\| - (l+1)$ **do**
    \# declare interval sequence
    $\mathbf{s} \leftarrow []$
    \# push time delta between i-th interval until $i+l$ into sequence
    for $k \leftarrow 0$ to $l+1$ **do**
      $s_k \leftarrow [\mathbf{x}_{(j+k)} - \mathbf{x}_j]$
    end for
    \# push sequence into final set
    $\bar{\mathbf{x}}_i \leftarrow \mathbf{s}$
  end for
end for
return $\bar{\mathbf{X}}$

To keep the input size small without resorting to classical batch normalisation techniques, the lookback vector of size $\ell = 16$ is preprocessed according to Algorithm 2, which produces inter-arrival times beginning from the first event within the bounds of the lookback. This strategy allows us to omit the typical (batch) normalisation stage within the model—particularly favourable in an online learning context where we may not make assumptions about the nature of the data post-transfer and cannot guarantee the range of the training set is perfectly representative. When the system is deployed, only lines $4 \ldots 7$ are required to maintain a size l vector with the input sequence.

4.4 Integrated Memory Consolidation

Offline training naturally enjoys full access to the dataset and, traditionally, can expect data of a similar nature in the future. Adding new tasks to a pre-trained neural network can result in a phenomenon known as catastrophic forgetting, which is a significant impairment in incremental learning [110]. To facilitate the transfer into online learning and as a preventative measure against catastrophic forgetting, the model receives a memory system of a specified capacity ζ to retain previous input sequences and their outcome. This design is reminiscent of declarative memory consolidation found in biological cognitive systems, one of several techniques in mitigating catastrophic forgetting and fostering continual learning [111].

The memory bank is implemented within the model in the form of a double-ended sequence container. Sequences built by higher layers in the system are maintained in memory up to a specified capacity and retrained alongside execution on unseen data for reinforcement until they are progressively discarded at the tail end when a new sequence is received. This implementation operates at $O(1)$ constant time and is addressable in code similarly to a vector; the input to the model is submitted following the memory's retraining. It should be noted that memory consolidation is disabled during offline training.

Table 4.1: MAE scores (days) of online outputs, each of varying episodic memory capacity.

Capacity of memory bank	0	10	50
Active machine MAE	0.212	**0.157**	0.198
Infrequent machine MAE	0.274	**0.146**	0.155

4.5 Online Transfer

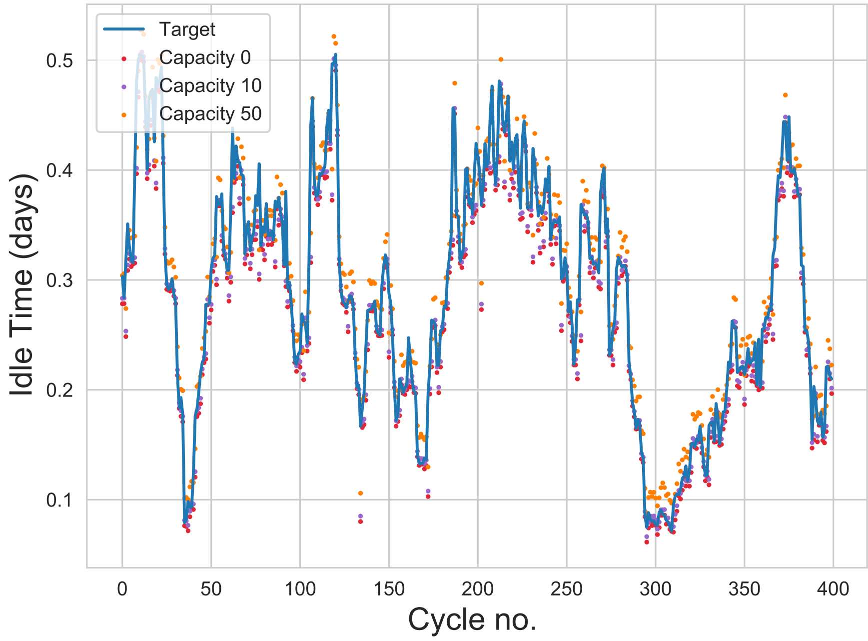
This sectiThis section presents the migration of the model described previously to an online learning environment. To evaluate the model's diversity, two machine samples are reserved from the same real-world dataset from offline training: one with over 1,000 cycles throughout the month and another with less than 100. To carry forward the model, its parameters are serialised and exported to disk after offline training. These can be deserialised from disk when the model should execute.

4.5.1 Environment

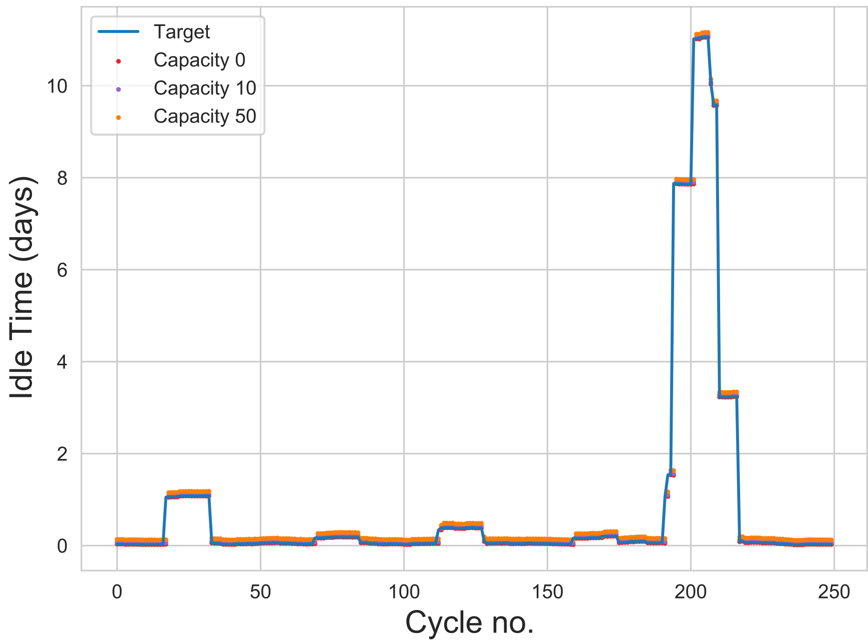
To simplify deployment and execution, the model is encapsulated in a manager framework responsible for the rolling input buffer, interrogation, and reporting from the model. A single method is accessible to the consuming platform, where it may submit a UNIX timestamp and receive a suggested interval until the next event, assuming data is sufficient in quantity. No inference is made when the input buffer is below the lookback size. However, this does not apply to the memory system; training from memory is still conducted with an underpopulated store, as the sequences within are complete.

4.5.2 Results

We run the framework encapsulating the model individually on each cycle on shared hyperparameters and a single pass of the data stream. The direct outputs against the targets of both datasets are illustrated in Figure 4.2, which shows competent predictions in both sets. We repeat the experiment, varying the model's memory capacity at 10



(a)



(b)

Figure 4.2: Interval prediction (days) from machines of characteristically distinct usage patterns, where (a) shows interval predictions between cycles of a more active machine and (b) shows interval predictions between cycles of a less active machine. The scatter points represent the outputs from different models trained on the same blue baseline data but with varying episodic memory capacities, as given in the legends.

and 50 units, the MAE losses for which are detailed in Table 4.1. The model is then run once more, this time, with its episodic memory capabilities disabled.

From these results, we note that the memory capacity assumes a role as another hyperparameter of the model, as an excessive memory capacity leads the model to exhibit classical overfitting behaviour in its most recent observations. In balancing this hyperparameter, we must consider not only the potential to overfit as an upper bound but also the capabilities of the host platform and whether it can support this level of retraining, as backpropagation training is an especially intensive process with recurrent neural networks. However, the memory system still offers a substantial benefit and is a worthwhile performance enhancement in a problem and approach of this nature.

4.6 Conclusions and Future Directions

This chapter proposes a novel LSTM model trained on temporal point processes from cyclic machine data, architected to perform and retrain on an unknown, evolving dataset. We enable this by preparing the dataset into sequences of local intervals and rebuilding the timestamp from the output of the model. The model is trained offline with full access to a wide range of appliances in preparation for online transfer, where it then adopts a consolidated memory system to converge quicker on new and changing habits in deployment.

The model demonstrated competent performance in an online environment, having tested in both inactive and frequent data streams. We note from the experiments that the memory capacity introduces an additional hyperparameter, which may be tuned when adopting this architecture into another context. It is an important parameter to balance, as allowing too high a memory capacity can hinder its training on future values by struggling to discard the irrelevant past, whereas too low would not harness the full potential of the model and mechanism, it may lead to wasting the compute used for retraining.

In terms of future work, we would be interested in regulating the capacity of the integrated memory consolidation feature of this model - or shutting it off completely. Given that this system is intended for edge hardware, we are conscious of its computational resources. Should a model perform online competently, the system could confidently reduce its memory capacity until it shows signs of struggle (e.g., an upward loss, a change point detected, or analysis of value drift). Further, given the intensity of model training, we would also be interested in training from memory during predicted idle periods to allow maximum resources for the host platform to perform its original purpose.

Chapter 5

Multi-Classifier for Fault Prognosis of Electrical Appliances using Gradient Boosting

The impact of unforeseen downtime in machinery assets, from domestic to industrial, bears a significant impact on owners. Any underlying mechanical issues or potential failures should be identified as early as possible. Predictive maintenance and prognostic analysis are well-researched fields motivated and bolstered by the Industry 4.0 revolution, aiming at anticipating equipment or component failure and recommending proactive maintenance with minimal downtime and impact on broader operations. Predictive maintenance assists in the management process, offering calculated degradation and health from data more insightful than ever, given the recent advancement of machine learning and deep learning techniques.

Through close monitoring of a machine's health, enriched by the new data and technology brought forward by the Internet of Things and Industry 4.0, performing maintenance pre-emptively at a time that maximises a component's lifetime without failing allows the owners to minimise the expense of downtime and potentially avoid catastrophic failure. Our objective is to introduce this same behaviour in a domestic setting by architecting a model that offers the same foresight as to a consumer's electrical appliance as with heavy industrial machinery.

The means to our NILM solution comes from a retrofitted, IoT-connected smart plug capable of remote control and scheduling. The plug is controllable via private cloud and is equipped with a metering chip that records and transmits its energy readings at minute intervals as triaxial readings of voltage, power, and current in 60-second intervals. The recording of these readings is largely powered by an ESP32-S3: a microcontroller capable of running machine learning algorithms on edge. For analysis and training, we sample from this hardware onto a desktop environment. We intend to deploy the model described in this paper onto this hardware.

We face a markedly imbalanced dataset, where an uneven class distribution in the training data thwarts representative learning. Due to scope and complications in oversampling temporal data [112], our attention turns to modern interpretations of classical machine learning approaches. In this instance, we favour gradient boosting with the popular XGBoost framework in designing a classifier capable of learning from oversampled data and engineering temporal features from static data points.

5.1 Introduction

With the rise of the Internet of Things and a new abundance of sensor data, condition monitoring, and its value, are becoming increasingly powerful tools, offering enriched insights into the performance and operation of machinery and equipment. Industrial applications are reaping the financial and operational benefits from the ability to preemptively schedule maintenance before a component fails to maximise its usefulness and mitigate unscheduled, expensive downtime. Our goal for this work is to retrofit a device inside a domestic electrical appliance next to its power source, using the power drawn to competently make the same predictions.

Due to scope and complexities of oversampling temporal data, we turn our attention to modern interpretations of classical machine learning approaches. We observe that the model we originally architect struggles to identify any trained failure, yet correctly recognises our data's normal, healthy, operational state. Expectedly, the cause of this is data imbalance: where an uneven class distribution in the training data thwarts representative learning. In this instance, we favour gradient boosting with the popular XGBoost framework in designing a classifier capable of learning from oversampled data and engineering temporal features from static data points.

We instead propose a gradient boosting-enabled ensemble, trained on statistical transformations of the rolling energy data, capable of informing higher levels of a monitoring system on failure. Further, the model attributes the fault to known failure modes in the device without access to rich sensor data, maintenance records, or failure history. We mitigate this imbalance by treating our dataset as non-temporal, allowing us to leverage oversampling techniques to redress the balance. We find the existing data pipeline and learned parameters of the model largely untransferable to this new data. We instead rework the data and feature engineering pipelines, followed by the hyperparameters and learnings of the model itself.

We adapt the same model, also trained for the similar purpose of fault detection, to a new, more intricate dataset evolved from the same origin as the data used in Chapters 3 and 4, adjusting the hyperparameters and introducing additional preprocessing for the new frequency of the dataset.

The design achieves an overall F-1 harmonic score of 97.3% in identifying its healthy operating state along with failure designations in our proprietary dataset and a similar score in a public predictive maintenance dataset. This chapter offers an approach toward the conditioning of highly imbalanced time series data of a relatively small size into training an efficient gradient boosting model, capable of running on edge hardware. Unlike rich sensor data, as with most similar existing work, our model performs on statistical features of energy data from a power supply.

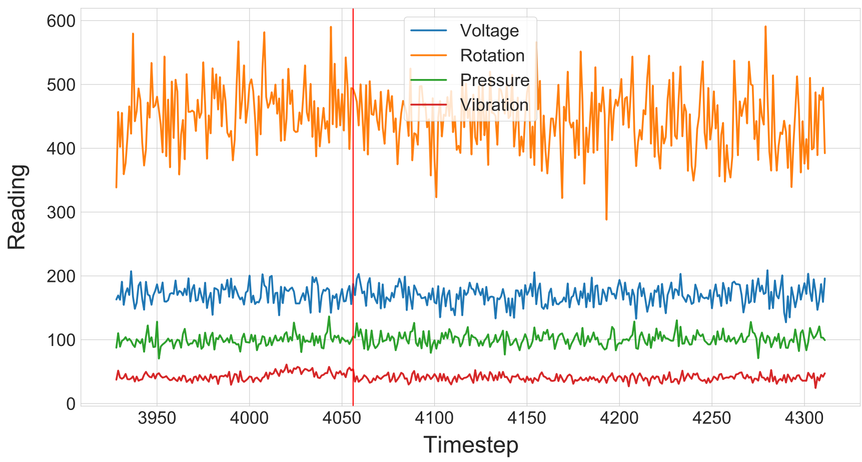


Figure 5.1: Sample graph in the public dataset leading to a distinct failure, the exact point of which is marked by the red vertical bar, and all subsequent readings become those of a faulty machine.

5.2 Materials

5.2.1 Datasets

Datasets of this nature are scarce and often protected. Therefore, as a benchmark and experimental platform, we turn to a dataset synthesised by the Azure platform to demonstrate cloud machine learning. This dataset, now no longer publicly available, is still inside the predictive maintenance problem space and comprises telemetry data from 100 machines' voltage, rotation, vibration, and pressure sensors over a span of 1 year for analysis, set in an industrial environment. For these 100 machines across four different machine models, individual telemetry readings from these sensors contain 876,100 hourly records, at 8,761 records per machine.

This dataset is supplemented with rich historical data including error and maintenance records, which can each be leveraged in the architecture and feature engineering of a machine learning model. The failure records contain 3,919 entries and the maintenance history contains 3,286 logs. Examples of machine failures are illustrated in Figure 5.1, showing the lead time and aftermath of a failure, marked by a solid vertical line.

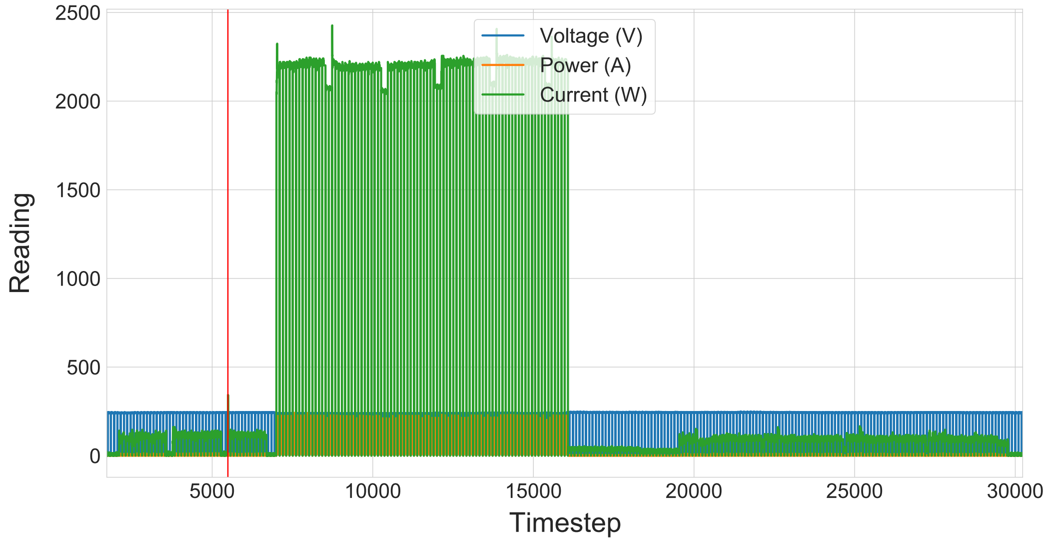


Figure 5.2: Example power consumption graph in the leading to a distinct failure of the new dataset; the exact point of invoked failure is marked by the red vertical bar.

Of the public dataset, there are four known failures the model must correctly categorise, one of which is illustrated in Figure 5.2, along with a normal operating state. In the figures within, we observe a noticeable change in at least one of the axes leading to a failure. This model should identify the correct failure based on the recent energy signature of the device. With known failure points in the dataset, the task transforms into a supervised learning problem, where we have corresponding labels for points of failure to train against (everything *normal* is categorised as such). However, the means of our dataset does not include any of these additional records. To ensure a more representative development, we opt to use only the telemetry in this dataset and omit any historical failure or condition records.

Our dataset is to be found from a device farm of a large domestic appliance manufacturer, comprising 100 machines running under high-intensity load, simulating several years of everyday usage over a span of 1 year. The energy readings are collected from its power outlet on an hourly basis, providing us with over 875,000 combined readings of four distinct features of voltage, pressure, vibration, and rotation. Meanwhile, we begin to curate our own, physically pulling the motor from inside a laundry machine, recording the time as the mark of failure, then reconnecting it.

Separate from the energy readings are failure records, recorded by devices equipped with a series of probes that can identify and deliberately invoke failures in real-time; this equipment is not feasible to integrate into the final product, and we cannot guarantee deliberate failures will be perfectly representative those of natural degradation, hence our requirement for a machine learning-enabled solution. In this case, we aim to designate motor failure in the machine. When such a failure occurs, the timestamp is logged along with its class.

Our data originates from a power outlet that records the voltage, power, and current of a machine sampled every minute, retrofitted to a laundry machine. The energy readings are sampled from a retrofitted power outlet with proprietary metering hardware at 50 Hz, providing us with almost 50,000 individual readings across the dataset. Each reading includes distinct readings of the device's voltage, power, and current at each given time step. The historical power consumption time series is stored on cloud infrastructure, which must be dumped then extracted into individual samples as input data for a model. This process begins with light manual processing. We observe that when appliances of interest are inactive, the smart plugs themselves tend to idle just below 0.04A. We carry this value as a minimum threshold and consider any readings above as metered activity, in which case it is appended to the sample in situ. Once the readings bottom out at the 0.04A threshold for 5+ consecutive minutes, we trim at that point and export the readings between as an individual sample.

Due to limitations in our data gathering, the model must categorise only one mode of failure, an example of which is illustrated in Figure 5.2. We observe a noticeable change in at least one of the three axes leading to a failure point. This model should identify the correct failure based on the recent energy signature of the device. Although we are presently working with a single mode of failure, we design with the intention that this model will eventually support multiple failure modes.

Our working dataset is organically formulated by the industrial partner, manually simulating catastrophic failures and recording the time of the action. In this case, the motor is pulled from the machine. The energy readings are collected from a retrofitted power outlet on a 50 Hz basis, providing us with almost 50,000 individual readings across 21 machines. Each reading includes distinct readings of the device's voltage, power, and current consumption at each given time step. There is, for now, only one mode of known failure in this dataset that the model must categorise, examples of which are illustrated in Figure 5.1. In the figures within, we observe a noticeable change in at least one of the three axes leading to a failure point. This model should identify the correct failure based on the recent energy signature of the device.

5.2.2 Data Ingestion

We receive our dataset in the form of a series of JSON documents, where each file contains an object representing the cycle from start to finish. The point of failure is marked in the file name with the minute count into the cycle, which are parsed using regular expressions (RegEx). There is also additional preprocessing to be undertaken in bringing the raw readings to their actual values.

Due to the nature of the metering hardware and how time series data is uploaded from edge to the cloud, we are given calibration offsets and scales for each feature of the data: voltage, power, and current. Each of these must be processed on a per-cycle basis to be compatible with our existing pipeline.

The preprocessing strategy calls for an additional stage to account for the new format and nature of the data. We receive a series of readings for voltage, power, and current, along with a set of constants defining a calibration offset and scale for each of these features, exampled above in Figure 5.3. These scales and offsets are calculated by the hardware to make uniform the readings we receive between metering chips. The novel nature of this hardware and its stability require these constants to be provided with each cycle, as the hardware is not fully stress-tested and cannot be presumed stable.

Per Algorithm 3, we receive voltage v and power p readings together. Using ν to denote

```
{
  "CalibrationVoltageOffset": 0.0,
  "CalibrationVoltageScale": 3.278749942779541,
  "CalibrationCurrentOffset": 0.0,
  "CalibrationCurrentScale": 473.06768798828125,
  "CalibrationPowerOffset": -0.019999999552965164,
  "CalibrationPowerScale": -0.20392952859401703,
  "Readings": [{
        "V": 0.0,
        "A": 0.0,
        "W": -3.0
    }, {
        "V": 0.0,
        "A": 0.0,
        "W": -1.0
    },
    // ...
  ]
}
```


Figure 5.3: Head of a given wash cycle. The offsets and scales provided with the payload are calibrated by the metering hardware of our smart plug.

the given voltage scale, ρ for the current scale, and ϱ for the current offset, we can then calculate the correct meter readings and derive the power. We clamp non-negative values to 0 as a pre-emptive measure in enforcing non-negative output in the model. These results, accompanied by an identifier for the specific cycle, are grouped as a tuple and appended to the total available training data.

5.2.3 Feature Engineering

To feed the model raw energy data as-is would not produce insightful results. We look to feature engineering to provide the model with more meaningful patterns for classification, referencing moments, and quantitative measures to the fourth ordinal, similar to the work in [113] for some of the features selected. These include: mean and variance

Algorithm 3 Preparing proprietary dataset meter readings into actual values, using supplied scales and offsets from the payload in Figure 5.3.

\# apply voltage scale
vtg $\leftarrow$ min(0, v/ν)
\# scale and offset current
crt $\leftarrow$ min(0, $(a-\rho)/\varrho$)
if $v > 0 \wedge w > 0$ **then**
    \# derive power if non-negative
    pwr $\leftarrow$ vtg / crt
else
    \# clamp to 0
    pwr $\leftarrow$ 0
end if
return vtg, crt, pwr

$$\mu(j) = \frac{\sum_i^N z_{ij}}{N} \tag{5.1}$$

$$v(j) = \frac{1}{N-1}\sum_i^N (z_{ij} - \bar{z}_j)^2 \tag{5.2}$$

as well as the skewness, standard deviation, and kurtosis

$$\mathrm{sk}(j) = \frac{N\Sigma_i^N (z_{ij} - \bar{z}_j)^3}{\sqrt{N\Sigma_i^N (z_{ij} - \bar{z}_j)^2}} \tag{5.3}$$

$$\sigma(j) = \sqrt{\frac{1}{N-1}\sum_i^N (z_{ij} - \mu(j))^2} \tag{5.4}$$

$$\mathrm{kt}(j) = \frac{N\Sigma_i^N (z_{ij} - \bar{z}_j)^4}{[\Sigma_i^N (z_{ij} - \bar{z}_j)^2]} - 3 \tag{5.5}$$

(deducting 3 for normal distribution as per the definition of [114]) as a measure of the asymmetry and the combined weight of the tails of a distribution relative to its centre.

5.2.4 Data Preprocessing

We receive a series of readings for voltage, power, and current, along with a set of constants defining a calibration offset and scale for each of these features. The novel nature of this hardware and its stability require these constants be provided with each cycle, as the hardware is not fully stress-tested and cannot be presumed stable. As a pre-emptive measure in enforcing non-negative output in the model, we clamp negative values to 0. These results, accompanied by an identifier for the specific cycle, are grouped as a tuple and appended to a wider matrix, forming our training data. Input data is then resampled into time windows of 3 hours and rolling time windows of 24 hours, both of which are fed collectively to the convolution layer. Each point in the window is calculated into new features using the statistical methods described in Equation 5.1-5.5 in addition to the minimum and maximum range of the window.

In our proprietary dataset, we use an equivalent strategy with 30-second and 5-minute windows. These windows are conjoined as a single input feature, where the 5-minute window lag is included alongside all 30-second windows elapsed of the sample as features. No form of normalisation is applied at any point in pre-processing stages or the model. When complete, the model will be given a total of 48 features, shared between 3-hour summations and 24-hour rolling windows. We adopt this approach based on piecewise linear degradation [115], where the component gradually exhibits signs of degradation rather than an abrupt failure. As the estimation of each component's exact lifetime is the problem itself, we use a general frame of tagging failures up to 24 hours prior in the pre-treated dataset.

Early training with an uncompensated dataset demonstrated strong performance in classifying a normal control state. The model displayed the same underperformance in correctly classifying our given test state. Through experimentation with model performance and imbalance ratios, we favour a combination of oversampling and undersampling techniques using the dual-strategy SMOTE-EEN.

We first considered an adaptive synthetic (ADASYN) approach [116] before opting for

SMOTE-ENN to overcome our data imbalance problem, the ratios for which are denoted in Table **??**. This oversampling strategy holds the more favourable characteristic of its ability to decide autonomously the required production volume of minority classes. This technique demonstrated its performance in reconciling an imbalanced dataset [117]. SMOTE-ENN begins with the standard SMOTE oversampling technique, generating calculated synthetic data points for the minority class to redress the balance. This data space is then piped to the Edited Nearest Neighbour (ENN) undersampling algorithm [40], an adaptation of KNN which deletes redundant or noisy instances from the dataset, which are:

1. Members of the majority class;

2. Not members of the K-nearest neighbour.

Each step is repeatable and, upon completion, delivers preferable results when used in noisier data spaces [118].

5.2.5 Training

Selecting optimal hyperparameters is a common challenge across many applications of machine learning. The model hyperparameters are obtained from an exhaustive grid search within predefined ranges for each parameter, drawn from other works, for our tuneable parameters, given in Table 5.1. Training of the model is conducted using softmax function, a loss function well suited in multiclass classification training. Simplicity in the design of the system was a core consideration for its intended purpose and environment, as illustrated by Figure 5.4, which shows the shared data preprocessing stage described in subsection 5.2.4 and the training process undertaken following. Selecting optimal hyperparameters is a challenge common across many applications of machine learning. The hyperparameters of the model are obtained from an exhaustive grid search within predefined ranges for each parameter, drawn from other works, for our tuneable parameters, given in Table 5.1. Training of the model is conducted using softmax, a loss function well suited in multiclass classification training.

Table 5.1: Hyperparameters selected for model training and evaluation in public and proprietary datasets.

Variable	Description	Value (sensor)	Value (energy)
η	The learning rate, or shrinkage. In extreme gradient boosting, η governs the rate at which new trees are created to correct the residual error and reduce the influence of overpowered trees within the ensemble, which betters the learning of newer trees in improving the overall model.	0.01	0.15
γ	A regularisation parameter governing the loss required to further partition a leaf node of a tree.	0.00015	0.0015
`max_depth`	Maximum depth of a tree in the model, directly influences model complexity and, by extension, its propensity to overfit.	25	50
`n_estimators`	The tree count available to the model. Generally, the tree count is heavily responsible for model complexity.	20	25
`one_drop`	At least one tree is always dropped during the dropout—used in conjunction with Dropout meet Multiple Additive Regression Trees (DART) booster: an approach which introduces dropout techniques from deep learning regularisation strategies to gradient boosting-enabled learning [119].	`true`	`true`
`objective`	Configures the model for multi-class classification, mutually exclusive outputs. In this case, we select softmax as our function, which is given by $y_i(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{k} e^{z_k}}$.	`softmax`	`softmax`

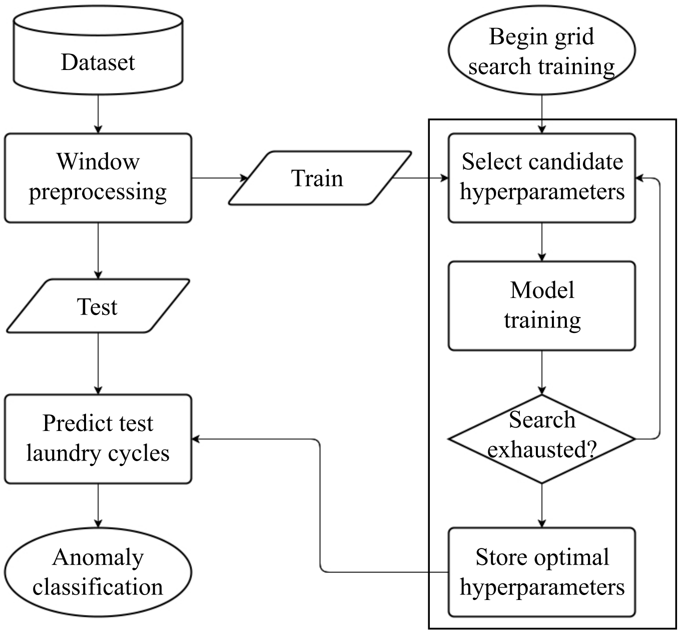


Figure 5.4: Flowchart of the workflow built for this application, showing both the hyperparameter tuning and test streams. The process is shared between the public and proprietary datasets alike.

Preliminary training observed promising output from the model in classifying normal operating state, but struggled to identify any form of degradation with a high false negative rate. Post-processing, the model shows only 719 failures against 30,000 normal samples in the original dataset. Such an imbalance is expected as a machine should be in a normal operating state for the majority of its lifetime. The model exhibited the same underperforming behaviour in correctly classifying our test state, recorded when physically removing power from the motor of a laundry machine. Through experimentation with model performance and imbalance ratios, we favour this time a combination of oversampling and undersampling techniques using the SMOTE-EEN sampler.

5.3 Results

The results of the training and testing runs are given in Table 5.2 for the original public dataset, and in Table 5.3 for our proprietary dataset. Each records performance evalua-

Table 5.2: Training evaluation on original dataset.

Designation	Normal	1.0	2.0	3.0	4.0
Precision	0.99	0.98	0.93	0.99	0.98
Recall	0.98	0.99	1.0	1.0	1.0
F1 score	0.97	0.99	0.99	0.99	0.98

Table 5.3: Training evaluation on proprietary dataset.

Designation	Normal	1.0 (pulled motor)
Precision	0.95	0.99
Recall	0.99	0.95
F1 score	0.97	0.97

tions for all respective failure designations as well as normal operating state. We note a consistently high F-1 score across both datasets of around 97% in failure identifications and a well-balanced hit rate despite the training size differential. As a baseline, we train a support vector machine (SVM) [120] classifier, a classical supervised learning model, on the same dataset. The SVM performs comparatively lower at average PPV = 0.9581, TPR = 0.9566, F-1 = 0.9565, as well as at a noticeably slower training rate. The XGBoost-enabled model has a higher overall reliability in designating failure states. Further, the feature engineering approach in this paper has shown its effectiveness, as the SVM baseline still attains a 95.6% overall accuracy.

5.4 Conclusions and Future Directions

To conclude, this paper explores the notion of applying predictive maintenance techniques, typically involving heavy industrial machinery, to a domestic setting, substituting intricate sensor readings for power signatures from a retrofitted smart plug. The model developed classifies multiple potential points of failure. Our work begins with omitting the maintenance and surrounding records of machines in a public dataset, relying solely on sensor readings to make informed fault diagnoses. We then detail the revisions and changes undergone to adapt this classifier to the new, real-world, higher-frequency dataset. Special care is paid to preprocessing stages, most notably the data imbalance—an inherent challenge in fault diagnosis where normal operating state vastly outweighs failure samples and model learning complexity. We employ an extreme gradient boosting ensemble, trained on rolling windows of statistical features running up to a single mode of known failure, demonstrating we can successfully develop according to the temporal nature of our data with a negligible effect on its overall performance of 97.3%.

We achieved a similar score in our public dataset for benchmarking, despite omitting detailed failure, maintenance, and age records. This study faced the pervasive limitation of an imbalanced dataset. The failure records account for a vast minority in the dataset, which required thoughtful feature engineering and preprocessing techniques to balance out the dataset before training of the ensemble could commence. After discovering a large data imbalance, we compensate by oversampling according to the SMOTE-ENN technique.

In terms of future work, we would be interested in exploring how distant in time the model is configurable to accurately predict and differentiate component-level failures from a more diverse dataset. Further, we would be interested in gaining some example data of a machine experiencing multiple failures and then exploring the model's adaptability toward multi-label classification, i.e., designating multiple (imminent) failures.

Chapter 6

Conclusions and Future Directions

This thesis presents and evaluates the architecture, data processing, and performance of multiple distinct machine learning models, each engineered to solve unique challenges and performing with notable competence in their problem space. Although differing in purpose and technology, the models form strong links and, used in conjunction, can offer significant insight as to the usage and condition of a device from its power signature alone, along with its prolongation.

We first explore the use of autoencoder-based reconstruction as a means of anomaly detection in Chapter 3. This approach is well-researched, however, it does not fully address our challenges, the most prominent being that the system must operate in relatively unknown environments with different patterns of energy cycles. To address this, we first build an incremental clustering system based on K-medoids, which allocates complete cycles into what become individual data streams later in the system. These streams carry a matched parameter set for an autoencoder ensemble, hot-swapped in real time depending on the active cycle. In this design, the model first spends a predefined amount of time learning the track's normal behaviour for future reconstruction. When a parameter set is trained up to, or beyond, this grace period, future readings relevant to the parameter set are reconstructed through the ensemble, and any anomalous behaviour is flagged in the returned result. To continue training in an online fashion, completed cycles are resubmitted to the system as training data. The model demonstrates competent output on simulated (therefore known) injections, and the configurable grading mechanism can designate the anomalies we inject as our test data. We evaluate the model on both point and collective strategies, accepting a single marked failure as success or requiring the entire injection to be flagged as anomalous.

Then, in Chapter 4, we employ an online deep learning technique in predicting the interval between one event and the next—in this case, machine cycles. We begin with classic statistical methods of temporal point processes. However, certain models are better suited to different dynamics and complexities and require careful selection. Given this model is for an unknown environment, where usage may be infrequent or hyperactive, it is not possible to make such presumptions. Therefore, we look to relegating as much of the tasks of point processes to a neural network. In this case, we propose an LSTM recurrent neural network designed to produce direct outputs from its training data, given only as timestamps, processed into rolling lookback windows. We train on a rich and varied dataset and see strong performance in data samples of varying usage. To support this model's adaptability for transfer into new environments, the model is equipped with an episodic memory, in which previous interval tracks are stored for retraining.

Finally, in Chapter 5, we look to develop a machine learning model capable of identifying as well as designating known failures in electrical appliances. The original approach, trained on statistical features of the dataset, struggled to correctly classify anything beyond the normal operating state. We found this is due to data imbalance exceptionally severe for temporal data, and oversampling forfeited any benefits of temporal stages. We then looked toward extreme gradient boosting, a technique that has seen remarkable performance in recent studies, and applied it to the same problem. We then adapt the model to our newer real-world dataset collected from new metering hardware on a laundry machine. This new form of data required additional preprocessing stages, bringing the new data shape into a workable format, in the data pipeline. We then account for the slight difference in the problem it is tasked with solving in the model's hyperparameters, primarily by reducing the model's complexity. Following these adjustments, the model's performance was brought up to comparable competence with the original design and dataset.

6.1 Future Directions

The work of this thesis opens avenues to exciting future possibilities. The models proposed within, when deployed into a consumer environment, can facilitate and complement useful, higher-level features. For example, detecting faults can serve push notifications to both the manufacturer and the user of the machine's failure (and precise root cause) and organise a repair before catastrophic failure. Similarly, a higher-level framework may record the intervals and activity of a laundry machine to calculate, track, and reorder detergent; or it may detect an overload of components within based on the same output from the model.

In terms of technical improvements, for the anomaly detection system of Chapter 3, there is an inherent risk that the system begins training from an already-anomalous machine. We would be interested in researching an approach that could address this, perhaps based on transfer learning. As a preliminary thought, we suggest monitoring the autoencoder's learning progress during initial training, flagging a failure to converge as potentially anomalous. In terms of future work, our attention would immediately focus on the time complexity of the incremental clustering algorithm. Although lightweight, it may eventually become too intensive to process on edge hardware. A preliminary thought is to develop a thinning mechanism, which prunes excessively dense clusters to reduce the dataset size, whilst preserving the identity and strength of the cluster.

Similarly, the model described in Chapter 4 also requires consideration for its first installation in a new system. The episodic memory part of the LSTM encourages stronger consolidation in new environments. We would be interested in regulating its capacity as learning improves so as not to avoid overfitting and not forfeit the varied training it has received. Likewise, a high-performing model may have its memory capacity shut off completely, as we must still consider the model is running on edge hardware and be conscious of the intense training process of recurrent neural networks.

For the failure prognosis model in Chapter 5, our primary interest in its future development is to adapt the model to multi-label classification—offering multiple failures (if existing) from a single reading. To achieve this we must first understand how the target machines behave with multiple impending failures, as we were only able to physically pull the motor cord from inside the laundry machine to gather live fault data.

In a wider sense, part of the original objective of this research was to onboard each of the final models into an all-in-one metering system; one that could intelligently catalogue its usage (autoencoder-based reconstruction as a means of anomaly detection in Chapter 3), dynamically place itself into a low-power state during predicted periods of idleness (a novel LSTM model trained on temporal point processes from cyclic machine data in Chapter 4), and monitor and diagnose where available for a set of known faults based on the same dataset. Each model is designed and trained on the same nature of data, produced by the same source. It would be a fascinating endeavour to architect a framework that can bootstrap, on edge, to the firmware level of our data source's metering technology.

References

[1] Philippe Esling and Carlos Agon. “Time-series data mining”. In: *ACM Computing Surveys (CSUR)* 45.1 (2012), pp. 1–34.

[2] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. “Cardiologist-level arrhythmia detection with convolutional neural networks”. In: *arXiv preprint arXiv:1707.01836* (2017).

[3] Julian Ganz, Matthias Beyer, and Christian Plotzky. “Time-series based solution using InfluxDB”. In: *no. i* (2016).

[4] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. “Time series databases and influxdb”. In: *Studienarbeit, Université Libre de Bruxelles* (2017).

[5] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. “Machine learning from theory to algorithms: an overview”. In: *Journal of physics: conference series*. Vol. 1142. IOP Publishing. 2018, p. 012012.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Grae-

pel, et al. “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”. In: *arXiv preprint arXiv:1712.01815* (2017).

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[9] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. “Grammar variational autoencoder”. In: *arXiv preprint arXiv:1703.01925* (2017).

[10] Armando Vieira and Bernardete Ribeiro. *Introduction to Deep Learning Business Applications for Developers*. Springer, 2018.

[11] Tianfeng Chai and Roland R Draxler. “Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature”. In: *Geoscientific model development* 7.3 (2014), pp. 1247–1250.

[12] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[13] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology press, 1995.

[14] D Randall Wilson and Tony R Martinez. “The general inefficiency of batch training for gradient descent learning”. In: *Neural networks* 16.10 (2003), pp. 1429–1451.

[15] Robert A Jacobs. “Increased rates of convergence through learning rate adaptation”. In: *Neural networks* 1.4 (1988), pp. 295–307.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[17] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[18] Duc Tam Nguyen, Zhongyu Lou, Michael Klar, and Thomas Brox. "Anomaly detection with multiple-hypotheses predictions". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4800–4809.

[19] Guansong Pang, Chunhua Shen, and Anton van den Hengel. "Deep anomaly detection with deviation networks". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 353–362.

[20] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[21] YURDANUR Tulunay, E Tulunay, and ET Senalp. "The neural network technique—-1: a general exposition". In: *Advances in Space research* 33.6 (2004), pp. 983–987.

[22] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[23] Kaveh Mahdaviani, Helga Mazyar, Saeed Majidi, and Mohammad H Saraee. "A method to resolve the overfitting problem in recurrent neural networks for prediction of complex systems' behavior". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 3723–3728.

[24] Kostas Kouvaris, Jeff Clune, Louis Kounios, Markus Brede, and Richard A Watson. "How evolution learns to generalise: Principles of under-fitting, over-fitting

and induction in the evolution of developmental organisation". In: *arXiv preprint arXiv:1508.06854* (2015).

[25] David A Winkler. "11.3 Deep and Shallow Neural Networks". In: *Chemoinformatics: Basic Concepts and Methods* (2018), p. 453.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[27] Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In: *Advances in neural information processing systems*. 2001, pp. 402–408.

[28] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: (2018).

[29] Lawrence Rabiner and BiingHwang Juang. "An introduction to hidden Markov models". In: *ieee assp magazine* 3.1 (1986), pp. 4–16.

[30] Larkin Liu, Yu-Chung Lin, and Joshua Reid. "Comparing the Performance of the LSTM and HMM Language Models via Structural Similarity". In: *arXiv* (2019), arXiv–1907.

[31] Sepp Hochreiter and Jürgen Schmidhuber. "LSTM can solve hard long time lag problems". In: *Advances in neural information processing systems*. 1997, pp. 473–479.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[33] Niklas Renström, Pramod Bangalore, and Edmund Highcock. "System-wide anomaly detection in wind turbines using deep autoencoders". In: *Renewable Energy* 157 (2020), pp. 647–659.

[34] Jaegul Choo and Shixia Liu. "Visual analytics for explainable deep learning". In: *IEEE computer graphics and applications* 38.4 (2018), pp. 84–92.

[35] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[36] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1322–1333.

[37] Apple. "Learning with Privacy at Scale". In: (2017).

[38] Chao Shang, Fan Yang, Dexian Huang, and Wenxiang Lyu. "Data-driven soft sensor development based on deep learning technique". In: *Journal of Process Control* 24.3 (2014), pp. 223–233.

[39] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[40] Dennis L Wilson. "Asymptotic properties of nearest neighbor rules using edited data". In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), pp. 408–421.

[41] Yi Yang, Andy Chen, Xiaoming Chen, Jiang Ji, Zhenyang Chen, and Yan Dai. "Deploy large-scale deep neural networks in resource constrained iot devices with local quantization region". In: *arXiv preprint arXiv:1805.09473* (2018).

[42] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. "Efficient defenses against adversarial attacks". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 2017, pp. 39–49.

[43] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning". In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.

[44] A Ćiprijanović, D Kafkes, GN Perdue, K Pedro, G Snyder, FJ Sánchez, S Madireddy, SM Wild, and B Nord. "Robustness of deep learning algorithms in astronomy–galaxy morphology studies". In: *arXiv preprint arXiv:2111.00961* (2021).

[45] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. "Smart grid—The new and improved power grid: A survey". In: *IEEE communications surveys & tutorials* 14.4 (2011), pp. 944–980.

[46] Jack Kelly and William Knottenbelt. "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes". In: *Scientific data* 2.1 (2015), pp. 1–14.

[47] Adrian Filip et al. "Blued: A fully labeled public dataset for event-based nonintrusive load monitoring research". In: *2nd workshop on data mining applications in sustainability (SustKDD)*. Vol. 2012. 2011.

[48] EJ Aladesanmi and KA Folly. "Overview of non-intrusive load monitoring and identification techniques". In: *IFAC-PapersOnLine* 48.30 (2015), pp. 415–420.

[49] Attique Ur Rehman, Tek Tjing Lie, Brice Vallès, and Shafiqur Rahman Tito. "Comparative evaluation of machine learning models and input feature space for non-intrusive load monitoring". In: *Journal of Modern Power Systems and Clean Energy* 9.5 (2021), pp. 1161–1171.

[50] Zhenyu Zhang, Yong Li, Yixiu Guo, Ziming Tao, Chang Li, Gang Lin, Yijia Cao, Christian Rehtanz, and Mohammad Shahidehpour. "Achieving Sustained Improvement in Identification Accuracy with a Semi-Supervised Learning Approach for NILM Systems". In: *Authorea Preprints* (2023).

[51] Arfa Yasin and Shoab Ahmed Khan. "Unsupervised event detection and on-off pairing approach applied to NILM". In: *2018 international conference on frontiers of information technology (FIT)*. IEEE. 2018, pp. 123–128.

[52] Lucas Pereira and Nuno Nunes. "Performance evaluation in non-intrusive load monitoring: datasets, metrics, and tools—A review". In: *Wiley Interdisciplinary Reviews: data mining and knowledge discovery* 8.6 (2018), e1265.

[53] Anthony Faustine, Nerey Henry Mvungi, Shubi Kaijage, and Kisangiri Michael. "A survey on non-intrusive load monitoring methodies and techniques for energy disaggregation problem". In: *arXiv preprint arXiv:1703.00785* (2017).

[54] Sanket Desai, Rabei Alhadad, Abdun Mahmood, Naveen Chilamkurti, and Seungmin Rho. "Multi-state energy classifier to evaluate the performance of the nilm algorithm". In: *Sensors* 19.23 (2019), p. 5236.

[55] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[56] Sangeeta Oswal, Subhash Shinde, and M Vijayalakshmi. "A Survey of Statistical, Machine Learning, and Deep Learning-Based Anomaly Detection Techniques for Time Series". In: *International Advanced Computing Conference*. Springer. 2022, pp. 221–234.

[57] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. "Statistical and Machine Learning forecasting methods: Concerns and ways forward". In: *PloS one* 13.3 (2018), e0194889.

[58] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection". In: *Signal Processing* 99 (2014), pp. 215–249.

[59] Oscar Serradilla, Ekhi Zugasti, Jon Rodriguez, and Urko Zurutuza. "Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects". In: *Applied Intelligence* 52.10 (2022), pp. 10934–10964.

[60] Wei Wang, Baoju Zhang, Dan Wang, Yu Jiang, Shan Qin, and Lei Xue. "Anomaly detection based on probability density function with Kullback–Leibler divergence". In: *Signal Processing* 126 (2016), pp. 12–17.

[61] Ke Zhang, Marcus Hutter, and Huidong Jin. "A new local distance-based outlier detection approach for scattered real-world data". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2009, pp. 813–822.

[62] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms". In: *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE. 2016, pp. 1–6.

[63] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 387–395.

[64] Weizhong Yan. "Toward automatic time-series forecasting using neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.7 (2012), pp. 1028–1039.

[65] Yaguo Lei, Feng Jia, Jing Lin, Saibo Xing, and Steven X Ding. "An intelligent fault diagnosis method using unsupervised feature learning towards mechanical

big data". In: *IEEE Transactions on Industrial Electronics* 63.5 (2016), pp. 3137–3147.

[66] Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. "Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach". In: *Expert Systems with Applications* 140 (2020), p. 112896.

[67] Chris S Wallace and David L Dowe. "MML clustering of multi-state, Poisson, von Mises circular and Gaussian distributions". In: *Statistics and Computing* 10.1 (2000), pp. 73–83.

[68] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[69] Jae Young Choi and Bumshik Lee. "Combining LSTM network ensemble via adaptive weighting for improved time series forecasting". In: *Mathematical Problems in Engineering* 2018 (2018).

[70] Halil Çimen, Ying Wu, Yanpeng Wu, Yacine Terriche, Juan C Vasquez, and Josep M Guerrero. "Deep learning-based probabilistic autoencoder for residential energy disaggregation: an adversarial approach". In: *IEEE Transactions on Industrial Informatics* 18.12 (2022), pp. 8399–8408.

[71] Álvaro Hernández, Rubén Nieto, Laura de Diego-Otón, María Carmen Pérez-Rubio, José M Villadangos-Carrizo, Daniel Pizarro, and Jesús Ureña. "Detection of Anomalies in Daily Activities Using Data from Smart Meters". In: *Sensors* 24.2 (2024), p. 515.

[72] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. "Kitsune: an ensemble of autoencoders for online network intrusion detection". In: *arXiv preprint arXiv:1802.09089* (2018).

[73] Seyed Amirhossein Najafi, Mohammad Hassan Asemani, and Peyman Setoodeh. "Attention and Autoencoder Hybrid Model for Unsupervised Online Anomaly Detection". In: *arXiv preprint arXiv:2401.03322* (2024).

[74] Valerie Isham and Mark Westcott. "A self-correcting point process". In: *Stochastic processes and their applications* 8.3 (1979), pp. 335–347.

[75] Yosihiko Ogata. "On Lewis' simulation method for point processes". In: *IEEE transactions on information theory* 27.1 (1981), pp. 23–31.

[76] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. "Recurrent marked temporal point processes: Embedding event history to vector". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 2016, pp. 1555–1564.

[77] Guolei Yang, Ying Cai, and Chandan K Reddy. "Recurrent spatio-temporal point process for check-in time prediction". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management.* 2018, pp. 2203–2211.

[78] Takahiro Omi, Kazuyuki Aihara, et al. "Fully neural network based model for general temporal point processes". In: *Advances in Neural Information Processing Systems.* 2019, pp. 2122–2132.

[79] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. "Intensity-Free Learning of Temporal Point Processes". In: *arXiv preprint arXiv:1909.12127* (2019).

[80] Tiago Zonta, Cristiano André Da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. "Predictive main-

tenance in the Industry 4.0: A systematic literature review". In: *Computers & Industrial Engineering* 150 (2020), p. 106889.

[81] R Keith Mobley. *An introduction to predictive maintenance*. Elsevier, 2002.

[82] Martin Pech, Jaroslav Vrchota, and Jiˇrí Bednář. "Predictive maintenance and intelligent sensors in smart factory". In: *Sensors* 21.4 (2021), p. 1470.

[83] Wlamir Olivares Loesch Vianna and Takashi Yoneyama. "Predictive maintenance optimization for aircraft redundant systems subjected to multiple wear profiles". In: *IEEE Systems Journal* 12.2 (2017), pp. 1170–1181.

[84] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[85] Robi Polikar. "Ensemble learning". In: *Ensemble machine learning: Methods and applications* (2012), pp. 1–34.

[86] Wenli Xue and Ting Wu. "Active Learning-Based XGBoost for Cyber Physical System Against Generic AC False Data Injection Attacks". In: *IEEE Access* 8 (2020), pp. 144575–144584.

[87] Rui Guo, Zhiqian Zhao, Tao Wang, Guangheng Liu, Jingyi Zhao, and Dianrong Gao. "Degradation State Recognition of Piston Pump Based on ICEEMDAN and XGBoost". In: *Applied Sciences* 10.18 (2020), p. 6593.

[88] Shahbaz A Siddiqui, Kusum Verma, KR Niazi, and Manoj Fozdar. "Real-time monitoring of post-fault scenario for determining generator coherency and transient stability through ANN". In: *IEEE Transactions on Industry Applications* 54.1 (2017), pp. 685–692.

[89] Lahiru Jayasinghe, Tharaka Samarasinghe, Chau Yuenv, Jenny Chen Ni Low, and Shuzhi Sam Ge. "Temporal convolutional memory networks for remaining useful life estimation of industrial machinery". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE. 2019, pp. 915–920.

[90] Jihong Yan, Yue Meng, Lei Lu, and Lin Li. "Industrial big data in an industry 4.0 environment: Challenges, schemes, and applications for predictive maintenance". In: *IEEE Access* 5 (2017), pp. 23484–23491.

[91] Weiting Zhang, Dong Yang, and Hongchao Wang. "Data-driven methods for predictive maintenance of industrial equipment: A survey". In: *IEEE systems journal* 13.3 (2019), pp. 2213–2227.

[92] Alison Smith, Tak Yeon Lee, Forough Poursabzi-Sangdeh, Jordan Boyd-Graber, Niklas Elmqvist, and Leah Findlater. "Evaluating visual representations for topic understanding and their effects on manually generated topic labels". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 1–16.

[93] Mikel Canizo, Isaac Triguero, Angel Conde, and Enrique Onieva. "Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study". In: *Neurocomputing* 363 (2019), pp. 246–260.

[94] Gabriel Rodriguez Garcia, Gabriel Michau, Mélanie Ducoffe, Jayant Sen Gupta, and Olga Fink. "Time Series to Images: Monitoring the Condition of Industrial Assets with Deep Learning Image Processing Algorithms". In: *arXiv preprint arXiv:2005.07031* (2020).

[95] Kaufman Leonard and J Rousseeuw Peter. "Finding groups in data: an introduction to cluster analysis". In: *Probability and Mathematical Statistics. Applied Probability and Statistics* (1990).

[96] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. John Wiley & Sons, 2009.

[97] Saurabh Shah and Manmohan Singh. "Comparison of a time efficient modified K-mean algorithm with K-mean and K-medoid algorithm". In: *2012 International Conference on Communication Systems and Network Technologies*. IEEE. 2012, pp. 435–437.

[98] Mark Van der Laan, Katherine Pollard, and Jennifer Bryan. "A new partitioning around medoids algorithm". In: *Journal of Statistical Computation and Simulation* 73.8 (2003), pp. 575–584.

[99] Peter Laurinec and Mária Lucká. "Clustering-based forecasting method for individual consumers electricity load using time series representations". In: *Open Computer Science* 8.1 (2018), pp. 38–50.

[100] Peter Laurinec. "TSrepr R package: Time Series Representations". In: *Journal of Open Source Software* 3.23 (2018), p. 577.

[101] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. "Data preprocessing for supervised leaning". In: *International journal of computer science* 1.2 (2006), pp. 111–117.

[102] Farhad Pourpanah, Bin Zhang, Rui Ma, and Qi Hao. "Anomaly detection and condition monitoring of UAV motors and propellers". In: *2018 IEEE SENSORS*. IEEE. 2018, pp. 1–4.

[103] Xi Jiang, Jianlin Liu, Jinbao Wang, Qiang Nie, Kai Wu, Yong Liu, Chengjie Wang, and Feng Zheng. "Softpatch: Unsupervised anomaly detection with noisy data". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 15433–15445.

[104] Kiyotaka Matsue and Mahito Sugiyama. “Unsupervised Tensor based Feature Extraction and Outlier Detection for Multivariate Time Series”. In: *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2021, pp. 1–12.

[105] Shuai Xiao, Junchi Yan, Mehrdad Farajtabar, Le Song, Xiaokang Yang, and Hongyuan Zha. “Joint modeling of event sequence and time series with attentional twin recurrent neural networks”. In: *arXiv preprint arXiv:1703.08524* (2017).

[106] Alan G Hawkes. “Hawkes processes and their applications to finance: a review”. In: *Quantitative Finance* 18.2 (2018), pp. 193–198.

[107] Kathryn Buchanan and Riccardo Russo. “Money doesn’t matter! Householders’ intentions to reduce standby power are unaffected by personalised pecuniary feedback”. In: *PloS one* 14.10 (2019), e0223727.

[108] Kundan Krishna, Deepali Jain, Sanket V Mehta, and Sunav Choudhary. “An lstm based system for prediction of human activities with durations”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.4 (2018), pp. 1–31.

[109] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. “Predictive business process monitoring with LSTM neural networks”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2017, pp. 477–492.

[110] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. “Measuring catastrophic forgetting in neural networks”. In: *arXiv preprint arXiv:1708.02072* (2017).

[111] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* 113 (2019), pp. 54–71.

[112] Azim Ahmadzadeh, Maxwell Hostetter, Berkay Aydin, Manolis K Georgoulis, Dustin J Kempton, Sushant S Mahajan, and Rafal Angryk. “Challenges with extreme class-imbalance and temporal coherence: A study on solar flare data”. In: *2019 IEEE international conference on big data (Big Data)*. Ieee. 2019, pp. 1423–1431.

[113] C Rajeswari, B Sathiyabhama, S Devendiran, and K Manivannan. “Bearing fault diagnosis using multiclass support vector machine with efficient feature selection methods”. In: *Int. J. Mech. Mechatronics Eng* (2015).

[114] Ronald Aylmer Fisher. “Moments and product moments of sampling distributions”. In: *Proceedings of the London Mathematical Society* 2.1 (1930), pp. 199–238.

[115] Felix O Heimes. “Recurrent neural networks for remaining useful life estimation”. In: *2008 international conference on prognostics and health management*. IEEE. 2008, pp. 1–6.

[116] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning”. In: *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE. 2008, pp. 1322–1328.

[117] Adnan Amin, Sajid Anwar, Awais Adnan, Muhammad Nawaz, Newton Howard, Junaid Qadir, Ahmad Hawalah, and Amir Hussain. “Comparing oversampling techniques to handle the class imbalance problem: A customer churn prediction case study”. In: *IEEE Access* 4 (2016), pp. 7940–7957.

[118] Guillaume Lemaitre, Fernando Nogueira, and Christos K Aridas. "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 559–563.

[119] Rashmi Korlakai Vinayak and Ran Gilad-Bachrach. "Dart: Dropouts meet multiple additive regression trees". In: *Artificial Intelligence and Statistics*. PMLR. 2015, pp. 489–497.

[120] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. "Support vector regression machines". In: *Advances in neural information processing systems* 9 (1996), pp. 155–161.