Condition monitoring of the cooling of variable speed drives using artificial intelligence

Pere Izquierdo

Energy Technology, PED4-1050, May 2020

Master's Thesis

Energy Engineering
Aalborg University
http://www.aau.dk

Title:
Condition monitoring of the cooling of variable speed drives using artificial intelligence

Theme:
Reliability in power electronics

Project Period:
Spring Semester 2020

Project Group:
PED4-1050

Participant(s):
Pere Izquierdo

Supervisor(s):
Huai Wang
Norbert Hanigovszki

Copies: 1

Number of Pages: 70

Date of Completion:
May 28, 2020

Abstract:

Artificial intelligence (AI) has been successfully applied to find solutions to problems in a large array of fields. At its core, machine learning serves to develop statistical models that are able to analyze or predict the behavior of a given system. This study aims to make use of AI techniques to monitor the health condition of the cooling system of a variable speed drive. To do so, measurements are collected from a physical testing setup at different operating conditions, and these data samples are then used to train artificial neural networks. Two main approaches to this condition monitoring are detailed in this thesis, developing models to predict the behavior of the system at future times and to directly derive the value of a health indicator. This thesis also aims to detail the modeling of the drive and its cooling system and to provide a theoretical background on artificial neural networks.

Preface

This project has been carried out within the Energy Engineering Master's degree programme at Aalborg University, by a student of the Power Electronics and Drives specialization. This report aims to document the insights and results obtained during the development of the project. It has been written in the period comprised between the months of February and May of the year 2020.

I would like to thank Huai Wang, Yashar Khadem Sabaz, Norbert Hanigovszki, Shuai Zhao, Jörg Dannehl and Luzia Lopes Almeida for their support, advice, and feedback during the development of this project. Special acknowledgement is due for Yashar and Norbert to have taken the time to prepare and run experimental tests that would otherwise have been unfeasible.

Aalborg University, May 28, 2020

Pere Izquierdo
<pizqui18@student.aau.dk>

Contents

Nomenclature

Symbols

$\boldsymbol{\theta}$	Neural network internal parameters.	[-]
δ	Small constant for numerical stability.	[-]
$\dot{m}$	Mass flow rate.	[kg/s]
ϵ	Learning rate for Adam.	[-]
ϵ_{rough}	Internal surface absolute roughness.	[m]
η	Learning rate for SGD.	[-]
$\mathbf{b}$	Layer bias vector.	[-]
$\mathbf{g}$	Switching device gate signals.	[-]
$\mathbf{W}$	Layer weight matrix.	[-]
μ	Statistical mean.	[-]
μ_{avg}	Average dynamic viscosity.	[kg/(m·s)]
ω_{ref}	Reference rotor speed.	[rad/s]
ρ_1	First moment estimate decay rate of the Adam optimizer.	[-]
ρ_2	Second moment estimate decay rate of the Adam optimizer.	[-]
ρ_R	Density of air at the inlet conditions.	[kg/m^3]
σ	Sigmoid function.	[-]
θ	Rotor angle.	[rad]
a	Neuron activation.	[-]

C	Covariance matrix.	[-]
C_1	Module thermal capacitance.	[J/K]
C_2	Heat sink thermal capacitance.	[J/K]
c_p	Specific heat capacity.	[J/(kg·K)]
D	Diameter.	[m]
E_{OFF}	Turn-off device switching losses.	[J]
E_{ON}	Turn-on device switching losses.	[J]
f_{Darcy}	Darcy friction factor.	[-]
i_α	α-axis current.	[A]
i_β	β-axis current.	[A]
i_{abc}	Three-phase current.	[A]
I_{cond}	Conducting current of a single device.	[A]
i_d	Direct-axis current.	[A]
i_q	Quadrature-axis current.	[A]
J	Cost function.	[-]
k	Thermal conductivity.	[W/(m· K)]
Nu	Nusselt number.	[-]
P_{cond}	Power loss during to device conduction.	[W]
P_{loss}	Total power loss.	[W]
Pr	Prandtl number.	[-]
Q	Heat flow rate.	[W]
q_R	Volumetric air flow rate.	[m^3/s]
Q_a	Heat flow absorbed by the ambient.	[W]
Q_{conv}	Heat flow due to convection.	[W]
Q_{loss}	Heat flow due to the module losses.	[W]
R_1	Thermal resistance between the heat sink and the ambient.	[K/W]

R_1	Thermal resistance between the module and the heat sink.	[K/W]
$r_{p,P}$	Ratio of fan static pressure rise.	[-]
S	Surface.	[m^2]
T_a	Ambient temperature.	[T]
T_h	Heat sink temperature.	[T]
T_j	Device junction temperature.	[T]
T_{ref}	Reference machine torque.	[Nm]
v_α	α-axis voltage.	[V]
v_β	β-axis voltage.	[V]
V_{DC}	DC-link voltage.	[V]
v_d	Direct-axis voltage.	[V]
v_q	Quadrature-axis voltage.	[V]
V_{sat}	On-state saturation voltage of a single device.	[V]
x	Neuron state.	[-]

Abbreviations

AC	Alternating Current
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
BCE	Binary Cross-Entropy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma-Separated Values
DC	Direct Current

FOC	Field-Oriented Control
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	HyperText Transfer Protocol
IGBT	Insulated-Gate Bipolar Transistor
IoT	Internet of Things
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MCT	Motion Control Tool
MD	Mahalanobis Distance
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
mRMR	Minimum Redundancy, Maximum Relevance
MSE	Mean Squared Error
PWM	Pulse-Width Modulation
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RMS	Root Mean Square
RNN	Recurrent Neural Network
RPM	Revolutions Per Minute
SGD	Stochastic Gradient Descent
SSH	Secure Shell
SVPWM	Space Vector Pulse-Width Modulation

Chapter 1

Introduction

Artificial intelligence (AI) has been a fundamental part of many research breakthroughs in recent years. By making use of machine learning, a subset of artificial intelligence, computers can be made to solve challenging problems without any task-specific programming. Through machine learning, processes such as data analysis and system modelling can be automated with highly accurate results. Given enough data and processing capacity, systems based on machine learning can replicate the behavior of any arbitrarily complex system.

Modeling physical systems to predict potential failures can be a daunting task to perform through more traditional modeling techniques: failures can occur and interact in many different ways, and are often multi-domain in nature [1]. Machine learning can be used to develop such prediction models based on data rather than on explicit rules, allowing for accurate rules to be learnt implicitly.

1.1 Problem analysis

The reliability of power electronic converters is an important concern in industrial applications. According to [2], active cooling systems are the most common and effective solution to improve converter reliability, as temperature is one of the most significant stress factors in components [3]. However, this means that the reliability of the cooling system itself also becomes an important concern. Inadequate cooling can lead to the premature failure of power electronic components. One way to ensure that the cooling system of a power converter remains healthy is through the use of condition-based maintenance, which can drive down maintenance costs by

predicting when faults will occur and thus enabling operators to act accordingly.

Most approaches to the condition monitoring of cooling systems require additional sensors. Among others, the condition monitoring of cooling fans (and rotating machines in general) is often based on vibration data analysis This is currently a barrier for implementation in industrial applications, due to the increased cost, size, and complexity that such a solution would introduce. As such, the goal of the present project is to develop an approach to monitor the condition of the cooling system of an industrial drive that does not require any additional sensors. To do so, artificial intelligence techniques—more specifically, neural network models—are used to obtain similar performance without making use of additional sensors.

1.2 State of the art

The degradation of cooling systems can be broadly classified into two main categories: failures in the cooling fan and failures in the thermal interface material. The metallic heat sink should be robust enough to only fail in very extreme cases, and as such hard failures in this element will not be considered in this project.

The traditional method for cooling system fault detection, still widely in use, involves human diagnosticians performing periodic inspections based on warning signs such as increased temperature, smell or acoustic noise, or decreased performance [4]. However, these physical symptoms are prone to being unreliable. A potentially more reliable and less costly approach is to automate the task by replacing such physical inspections by signal processing techniques.

Such techniques are commonly based on fan vibration, speed, current, voltage, acoustic emission, and, more recently, stray flux signals [5]. Out of these, the most widespread method for cooling fans, and rotary machines in general, is vibration signal analysis [6] [7]. Acquiring data for accurate diagnosis of the thermal interface material can be more challenging. Current methods can be classified into two kinds: direct observation of the degradation by making use of transparent materials, and estimating the thermal resistance of the interface based on temperature measurements [8].

Visual inspection of the thermal interface material is often not a real possibility in industrial applications, and therefore temperature measurements are more commonly used. If accurate measurements of both heat sink and switching device temperatures can be obtained, as well as reliable estimates of device power losses (which is often challenging), the calculation of thermal resistance becomes trivial.

Otherwise, assuming that the condition of the cooling fan is known, a degradation of the overall cooling system performance can be safely attributed to degradation in the thermal interface material.

The condition of the cooling system or either of its subsystems can be quantified using a generalized health indicator. Such an indicator should be as simple as possible in order to be easily interpretable. Distance metrics are most common, as they can be used to represent the status of the whole system as a single scalar value. Euclidean distance is often unfit for such a task, making other metrics such as Mahalanobis distance [9] or K-nearest neighbor distance [5] more commonly favored. In essence, these methods are used to find the distance between collected data samples and their global centroid.

Distance metrics applied on their own can usually not be used to directly infer the condition of a given system, as the metric would reflect parameters of the system uncorrelated with health status. Therefore, such methods require feature selection and engineering to ensure that the distance metric correlates with the health status of the system. More details on generalized health indicator metrics are provided in Chapter 2.

In this context, artificial intelligence can be used to predict health indicator values where part of the used data is missing. In a more direct approach, AI can automate the feature engineering or directly predict the health status of the system. This allows for increased flexibility in determining distance metrics, as AI can automatically find nonlinear combinations of features that correlate with system health but would not be considered in manual feature engineering.

A common approach is to extract the most significant features from measured data by making use of autoencoders, a type of neural network topology. This has been demonstrated in, among others, the papers by Sun et al. [10], in the context of induction machine fault classification, and by Roy et al. [11], which applies autoencoder-based feature selection to achieve a reported 100% accuracy in the NASA bearing health data set.

Direct approaches to health status predictions can be found in publications such as the papers by Jin et al. [12], where a support vector machine is used to predict the remaining life of fan bearings, or by Tallam et al. [13], where a deep neural network is used to detect a type of induction machine fault.

Another more empirical approach is to analyze the transient behavior of the system to estimate health indicators that would not be easily obtainable during normal operation. For example, the 2019 paper by Zhang et al. [8] involves measuring the time it takes for a power converter system to cool down after it is shut down

from normal operation to estimate the natural frequency of the thermal network. This parameter can reportedly be used to reliably assess the health condition of the cooling system, but cannot be directly predicted from real-time measurements during steady-state operation, requiring extensive testing instead.

Another common approach to condition monitoring is, rather than finding a single global health metric, predicting future values of measurable parameters of the system, building what is commonly called a surrogate model or digital twin. This predictions can then be compared to the real measurements, with their difference providing information on the health of the system. For example, in a cooling system, a surrogate model can be built to predict heat sink temperature in healthy operation. If measured temperature became significantly higher than the one predicted by the healthy surrogate model, it could be concluded that the cooling system would be experiencing degradation.

Such a surrogate model could be based on known mathematical descriptions (such as state-space models), on statistical models (such as linear regression, curve fitting, or autoregressive models), or on artificial intelligence models. The latter have the advantage of being able to represent arbitrarily complex relationships in data, both linear and nonlinear, time-dependent and time-independent. On the other hand, training a complex AI model will often require large amounts of data and computational resources.

Some publications demonstrating this approach are the papers by Xie et al. [14], where an autocorrelation-based model is used to evaluate cooling fan vibration data, by Venkatesan et al. [15], where several metrics of a permanent magnet synchronous machine are predicted using an artificial neural network and fuzzy logic, or by Su and Chong [16], where a neural network-based surrogate model is used to predict induction machine vibration data.

1.3 Problem statement and methodology

The main goal of this project is to develop machine learning-based models that can be used to monitor the condition of the active cooling system of an industrial motor drive.

To do so, a model of the drive and its cooling system was developed, in order to obtain a deeper understanding of the system. This model was developed in the *Matlab/Simulink* environment. Due to the multi-domain nature of the system, which contains electrical, rotating, air flow, and thermal elements, the *Simscape*

library of *Simulink* was used extensively. Simulation results were then used as a starting point to develop artificial intelligence models.

These models were developed in *Python 3*, after performing data analysis and processing. Two approaches were considered. The first involves training models to predict the future temperature values of the system using only data from healthy operation. By observing deviations between the predictions made by such a model and the behavior of the monitored system, the condition of the cooling system can be evaluated. Thresholds for these deviations can be determined based on experimental data. The second approach involves determining performance labels for healthy and unhealthy operation, and training an AI model to make predictions on this label. In its simplest form, a binary indicator can be used as this performance label, training the model to classify measurements as falling into either healthy or faulty operation.

The implementation of neural networks is based on the *Keras* API, in turn based on the *Tensorflow* backend. The models were tuned to obtain adequate performance metrics on simulation data. There is a large degree of freedom in the design of neural network models, mostly involving their internal structure. The developed models make use of simple fully-connected layers and of recurrent layers, which are typically able to obtain better performance on time-dependent data.

Experimental tests were then ran on a test setup at Danfoss, in Gråsten, with data being gathered in real time through an internet of things (IoT) device. Experiments were performed both for healthy and degraded operation of the cooling system.

This experimental data was then used to validate the simulation model and to train the developed neural network models. Finally, the performance of the neural network models was evaluated using data obtained in tests excluded from the training data sets.

1.4 Scope

The scope of this project has been influenced by the closing of Danish universities due to the 2020 COVID-19 outbreak. Initially, experimental measurements were planned to be collected early in the development process on an experimental setup at Aalborg University. Instead, experiments had to be ran at a Danfoss laboratory in Gråsten, at a later date than originally intended.

The experimental validation of the condition monitoring techniques developed in this project is limited to the test setup provided by Danfoss. As such, it is not

guaranteed that these techniques can be applied to other systems without loss of performance.

1.5 Project outline

The contents of this report are structured as follows:

- Chapter 1 aims to provide the motivation for this project, as well as summarize the current state of the art in the condition monitoring of cooling systems. It also includes a brief description of the process behind the development of this project.
- Chapter 2 includes a simplified description of the drive system, with special attention being given to its cooling system. The reader is assumed to be versed in the field of power electronics and drives; the bulk of the chapter is therefore focused on thermal modelling, which is in a closer relationship with the goals of this project. This is followed by an introduction to the development of quantitative health indicator metrics.
- Chapter 3 aims to provide a theoretical background in the field of artificial neural networks, describing their main components and training process. Besides the simpler deep fully-connected feedforward neural networks, recurrent neural networks are also introduced, as they are specially well-suited to processing time-series data, the type of data collected in this project.
- Chapter 4 provides details on the more practical parts, summarizing the process of data collection, the two suggested approaches to condition monitoring, and the training process of the artificial intelligence models.
- Chapter 5 summarizes the main difficulties found in the project and the compromises that have had to be taken, as well as the insights gained from these.
- Chapter 6 presents the main conclusions to be drawn from the obtained results.
- Chapter 7 aims to suggest future research possibilities based on the insights gained in the present project.

Chapter 2

System modelling

The first step towards developing a reliable method to monitor the condition of the drive cooling system is to acquire a deeper understanding of both the system itself and of how its health status can be evaluated. This chapter aims to provide this information, by including a general overview of the target system, of how its internal cooling process can be modelled, and of how system health is commonly quantified in approaches that do not involve artificial intelligence.

2.1 The drive system

In a simplified way, the drive system to be modelled can be represented as the block diagram shown in Figure 2.1. This block diagram shows information signals—i.e. control and measurement—in black. Physical signals are shown in different colors: blue for electrical, orange for thermal, and purple for air flow. For the purposes of this project, a thermal signal refers to a heat flow dominated by conduction, while an air flow signal models a mostly convective one. Arrows point in the direction of the flows expected during normal operation.

The first block of the model is the controller of the inverter, which is tasked with synthesizing the inverter gate signals that allow the load to be driven according to desired commands. Although industrial implementations are typically more complex, the controller can be considered to be a field-oriented one, in order to simplify the modelling process. Such a controller takes as its input measurements (or estimates) of the rotor position of the driven machine (θ_{meas}), the three-phase inverter output current ($i_{abc,meas}$), DC-link voltage ($V_{DC,meas}$), as well as either a

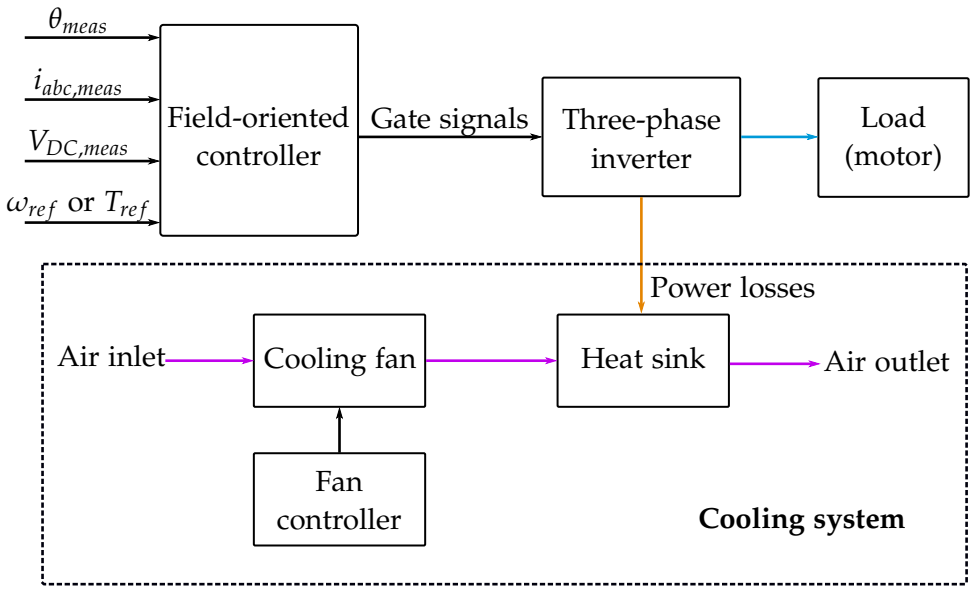

Figure 2.1: General block diagram of the drive system.

speed reference (ω_{ref}) or a torque reference (T_{ref}), depending on which of these two variables is to be controlled.

A block diagram showing how field-oriented control (FOC) can be implemented is shown in Figure 2.2. It is based on the fact that AC synchronous and induction motors produce torque due to the component of stator magnetic field aligned perpendicularly with that of the rotor. This flux is directly proportional to the stator currents. Therefore, by making use of the Park and Clarke transformations, and controlling the d-axis and q-axis current components, FOC is able to achieve high dynamic performance. Space vector pulse-width modulation (SVPWM), or a different PWM scheme, can be used to obtain the switch gate signals that correspond to the desired voltages. Field-oriented control typically requires measurements or estimates of the stator currents, rotor position, and rotor speed, as well as references for torque or speed and d-axis current. After PWM, its output are each of the gate signals $\mathbf{g}$.

The gate signals are then used to drive the switching devices of a three-phase inverter, which typically uses rectified voltage as its power source. The output of the inverter is then used to drive an electrical machine or another type of load. Loading the inverter generates switching losses every time that a switching device changes its state, and conduction losses while current flows through the devices and conductors. These power losses flow conductively through a thermal interface that connects the devices to a heat sink, thus cooling down the devices. The heat sink is in turn cooled by means of a fan, which forces air to flow from an inlet to

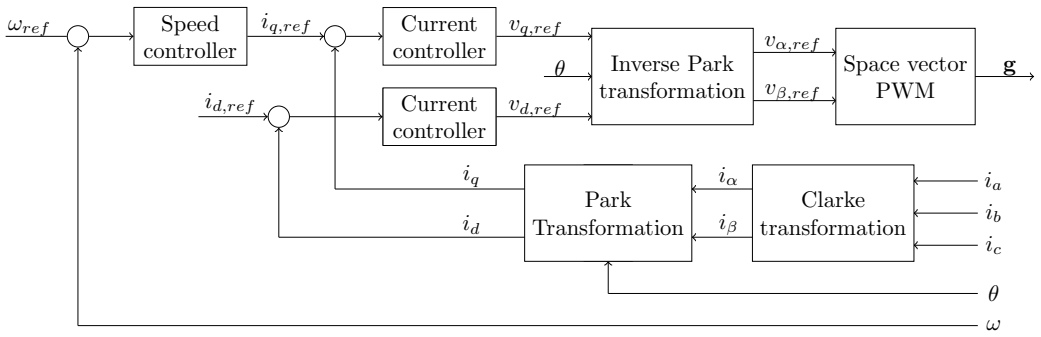

Figure 2.2: Block diagram of field-oriented control.

an outlet, interfacing with the heat sink in this process. For low-to-medium power drives, this fan is typically moved by a small DC motor, which is controlled by monitoring the system temperature of the system.

The main object of this project is the cooling system of the drive, and therefore most of the analysis and modelling is centered around it. More in-depth analysis of the inverter and motor systems is thus omitted from this report.

2.2 Modelling thermal losses and heat flow

A power module is formed by a switching device—typically an IGBT or MOSFET—and a freewheeling diode. Its power losses can be divided into conduction losses and switching losses.

Conduction losses are relatively straight-forward to calculate, as they can be found from the product between conducting current I_{cond} and on-state saturation voltage V_{sat}, typically given as a function of temperature in data sheets. The average dissipated power due to conduction losses over a period of time T can be calculated as:

$$P_{cond,avg} = \int_0^T \big(V_{sat}(t) I_{cond}(t)\big) dt \tag{2.1}$$

V_{sat} will have different values for diodes and for switching devices. Conduction losses therefore depend mostly on load current, duty cycle, and junction temperature. Switching losses, on the other hand, do not depend on the duty cycle, instead being directly proportional to switching frequency and DC-link voltage.

The typical behavior of switching device voltage and current is shown in Figure 2.3. Switching losses occur during turn-on (E_{ON}) and turn-off (E_{OFF}) of the device. Due

to the internal capacitances of the device and the free-wheeling diode, switching transients are not ideal, instead showing periods during which there is both non-zero voltage and current, resulting in power losses.

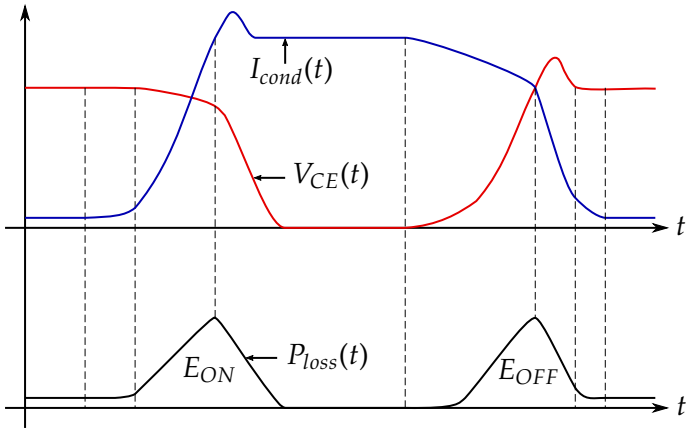

Figure 2.3: Waveforms of switching device current (blue), collector-emitter voltage (red), and instantaneous power losses (black).

Typically, the values of E_{ON} and E_{OFF} are provided in manufacturer data sheets, as functions of conducting current and device temperature. This data can then be introduced directly to device models in simulation software such as *PLECS* or *Simscape*.

Once the power losses of the power module are known, they are transmitted to the outside environment to maintain the temperature of the devices within an adequate range. The thermal network linking the power losses of the module with the external environment can be represented as shown in Figure 2.4.

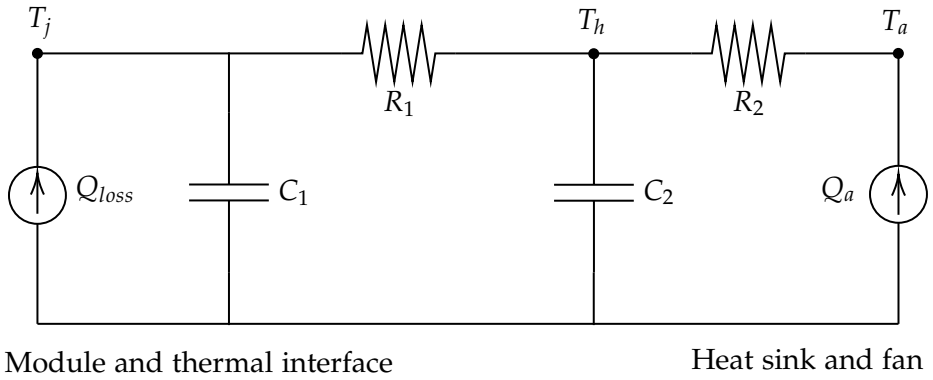

Figure 2.4: Circuit diagram of the cooling system thermal network.

Q_{loss} corresponds to the heat flow provided by the module losses, with device

junction temperature T_j. The parallel thermal capacitance C_1 represents the heat storage capability of the module, while the resistor R_1 is determined by the resistivity of the thermal interface material that connects the module to the heat sink. In turn, the thermal capacitance of the heat sink is represented by C_2, with temperature T_h. Another thermal resistor R_2 represents the thermal resistance between the heat sink and the external environment, which is modelled with a heat source extracting Q_a from the system and with a constant temperature T_a.

It should be noted that every component of this thermal network is variable, as all of them are at least temperature-dependent. This is especially true of R_2, as it models the performance of the cooling fan, heat-sink design, and air flow as a single parameter. R_2 is, for example, inversely proportional to fan speed, as higher cooling fan speeds allow for a larger flow rate of air and thus of heat dissipation.

In a way analogous to electric circuits, the heat flow between two nodes A and B connected by a thermal resistor is given by Fourier's law, as follows:

$$Q = R(T_A - T_B) \tag{2.2}$$

$$R = k\frac{A}{D} \tag{2.3}$$

Where Q is the heat flow from A to B, R is the thermal resistance, T_A, T_B are the temperatures of the nodes, k is the thermal conductivity of the material linking the nodes, A is the area perpendicular to the direction of heat flow, and D is the distance between the nodes.

For two nodes connected by a thermal capacitance, heat flow is given by:

$$Q = c \cdot m\frac{dT}{dt} \tag{2.4}$$

Where c is the specific heat of the thermal mass material, and m is its mass. Every parameter of the thermal circuit can be considered to remain constant for modeling purposes, except for R_2. In place of a simple thermal resistance, the heat exchange between the heat sink and the air flow driven by the cooling fan. The heat exchange modeling is based on the dynamics of pipe flow as described in [17]. A simple diagram showing the model of the heat exchange process is shown in Figure 2.5.

Four nodes are defined to describe the behavior of the heat exchange: the two ends of the pipe A and B (the sides of the heat sink perpendicular to air flow), an internal node I, and the surface of the pipe H (receiving the heat sink heat flow signal). Analysis is based on mass, energy, and momentum balance. The total heat transfer between the pipe wall and the internal gas surface—between the heat sink walls and the air flow—is found by adding together the conductive and convective

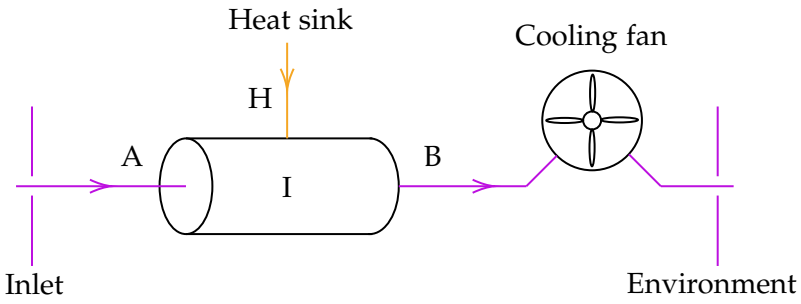

Figure 2.5: Model of the heat exchange process between the heat sink and the external environment.

flows:

$$Q_H = Q_{conv} + \frac{k_I S_{surf}}{D_h}(T_H - T_I) \tag{2.5}$$

Where Q_H is the heat flow rate at port H, Q_{conv} is the convective heat transfer, k_I is the internal thermal conductivity, S_{surf} is the equivalent pipe surface area, D_h is the equivalent hydraulic diameter of the pipe, and T_H, T_I are the temperatures at ports H and I, respectively. Assuming an exponential temperature distribution along the pipe—the length of the heat sink—heat transfer due to convection is given by:

$$Q_{conv} = |\dot{m}_{avg}| c_{p,avg}(T_H - T_{in})(1 - \exp\left(\frac{h_{coeff} S_{surf}}{|\dot{m}_{avg}| c_{p,avg}}\right)) \tag{2.6}$$

Where $\dot{m}_{avg} = (\dot{m}_A - \dot{m}_B)/2$ is the average mass flow rate between the inlet and the outlet, T_{in} is the inlet temperature (port A in the configuration shown in Figure 2.5), and $c_{p,avg}$ is the specific heat of the fluid (air) evaluated at its average temperature. The heat transfer coefficient h_{coeff} is defined as:

$$h_{coeff} = Nu \frac{k_{avg}}{D_h} \tag{2.7}$$

Where k_{avg} is the thermal conductivity evaluated at the average temperature. The Prandtl number evaluated at the average temperature Pr_{avg} can be considered constant, with a value of around 0.71 for air. The Nusselt number Nu depends on the flow regime. For forced, laminar flow, it can be considered to be a constant dependant on the geometry of the surface perpendicular to the flow—for a circular pipe, it has a value of 3.66. For turbulent flow, it can be found using:

$$Nu = \frac{\frac{f_{Darcy}}{8}(Re_{avg} - 1000)Pr_{avg}}{1 + 12.7\sqrt{\frac{f_{Darcy}}{8}}(Pr_{avg}^{2/3} - 1)} \tag{2.8}$$

Where the Darcy friction factor f_{Darcy} is given by:

$$f_{Darcy} = \left[-1.8 \log\left(\frac{6.9}{Re_{avg}} + \left(\frac{\epsilon_{rough}}{3.7 D_h}\right)^{1.11}\right)\right]^{-2} \tag{2.9}$$

With the constant ϵ_{rough} being internal surface absolute roughness. The Reynolds number evaluated at the average temperature can be found using:

$$Re_{avg} = \frac{|\dot{m}_{avg}|D_h}{S_{surf}\mu_{avg}} \tag{2.10}$$

Where μ_{avg} is the average dynamic viscosity of the fluid. The Reynolds number also determines the type of flow, usually from substance-dependent limits: an upper-limit for purely laminar flow and a lower limit for purely turbulent flow. If the Reynolds number lies in-between these thresholds, the flow is considered to be mixed, and its Nusselt number can be approximated by assuming a smooth transition between the two types of flow.

The modeling of the fan is carried out as a look-up table according to specifications. Specifically, the tabulated data used in simulations is volumetric flow rate and total efficiency against shaft speed and static pressure ratio. The mass flow rate $\dot{m}$ is obtained through:

$$\dot{m} = \rho_R q_R \tag{2.11}$$

Where ρ_R is the density of air at the inlet fan conditions, and q_R is the volumetric flow rate, computed from tabulated data using the ratio of static pressure rise to its maximum rated value $r_{p,P}$, as well as angular fan speed ω:

$$q_R = q(\omega, r_{p,P}) \tag{2.12}$$

Fan speed ω is given by a DC motor driven by the fan controller, which can be modeled as having ideal dynamics.

2.3 Quantifying system health

The simplest approach to defining system health is to use a binary indicator, i.e. to attach a value of 1 to samples taken where the system is known to be healthy and of 0 to samples where it is unhealthy, or vice versa. This is a valid strategy to employ for data collection in controlled environments, both physical and simulated, where faults in the system are either present or missing. Such a binary indicator can be sufficient for some tasks, but may prove too simplistic for others, where component degradation can degrade system performance without necessarily causing a complete fault.

A common approach to increase the robustness of such a health indicator without unnecessarily increasing complexity is to assess the condition of the system

through a single distance metric. The main intuition behind this is that by joining available information into a single scalar value, ranges can be defined for both healthy and unhealthy operation. This method is illustrated in Figure 2.6.

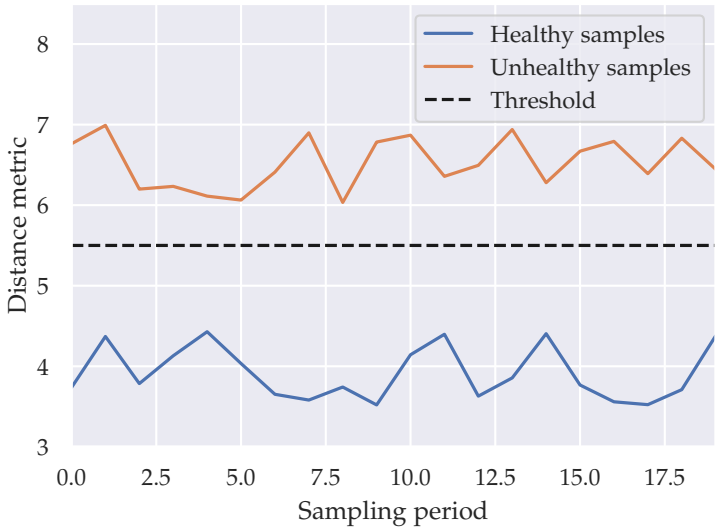

Figure 2.6: Intuition behind the use of distance-based metrics in condition monitoring.

Figure 2.6 shows the values taken by a distance metric in two hypothetical time-series of data taken during healthy operation (in blue) and unhealthy operation (in orange). A constant distance metric threshold (in black) is defined to separate both types of data. The more data that is used to define this threshold, the more confidently it can be determined. It is important to note, however, that in this case the distinction between healthy and unhealthy condition is not a strict binary. Deterioration can be observed through trends that bring the distance metric closer to unhealthy operation, before every sample necessarily falls within the unhealthy region.

Two main challenges must be overcome when developing a distance metric-based condition monitoring method. Firstly, if every feature—measurement—is considered equally when calculating the metric, it will likely not reflect the condition of the system. In such a way the metric would be prone to over-represent certain features: as an example, the switching frequency of a power converter is typically much larger in magnitude than phase current amplitude, which would make the former hold a much larger influence over the generalized health indicator. This problem can be solved, or at least mitigated, by normalizing measurements before including them in the calculation of the metric. Even so, there would be no guarantee that the distance metric would reflect the health status of the system.

To achieve this, feature engineering and extraction are to be performed. Commonly, the first step is to obtain transformations of the available data. For example, instantaneous phase current values are not likely to help quantify the condition of the system, but using the Fourier transform to find the amplitude of certain harmonics could prove very useful. Values that are normally stationary, such as amplitude or RMS values can also be significant. After this feature engineering phase, feature selection is performed to extract the most valuable parameters. Methods for feature extraction are often based on the selection of parameters that show the largest absolute correlation with the previously defined binary health indicator. Some more specific methods are kurtosis analysis [5] and minimum redundancy, maximum relevance (mRMR) analysis [6].

The "Predictive Maintenance Toolbox" of the Matlab software [18], and more specifically its "Diagnostic Feature Designer" application can be used to accelerate the data processing and feature mining processes. The toolbox includes resources to automate the extraction of metrics based on the available data. These metrics can be classified into three main categories, for which some examples are provided here:

- Time-domain features: mean, standard deviation, skewness, kurtosis, root-mean-square values.
- Frequency-domain features: mean frequency, power bandwidth, peak values, peak frequencies, harmonics.
- Time-frequency-domain features: spectral kurtosis, spectral entropy. This type of features are useful to characterize changes that occur over time in the frequency spectrum of a signal.

The second main challenge is the choice of the distance metric itself: this metric should aggregate relevant extracted features into a single value. A first approach would be to simply use Euclidean distance between each sample and the centroid of all samples, which in the present context could be defined as follows:

$$\begin{aligned} D_j(\mathbf{x}_j, \boldsymbol{\mu}) &= \sqrt{(x_{j,1} - \mu_1)^2 + (x_{j,2} - \mu_2)^2 + \cdots + (x_{j,n} - \mu_n)^2} \\ &= \sqrt{\sum_{i=1}^{n} (x_{j,i} - \mu_i)^2} \end{aligned} \tag{2.13}$$

Where D_j is the Euclidean distance associated with the j-th data sample, $\mathbf{x}_j = (x_{j,1}, x_{j,2}, \cdots, x_{j,n})$ are the features (variables) of the sample, $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_n)$ are the arithmetical means of each feature, or centroid vector, and n is the number of features of each sample. Directly calculating this distance between each sample

and the centroid of the features will, however, usually lead to misleading results. To illustrate this, consider the example provided in Figure 2.7.

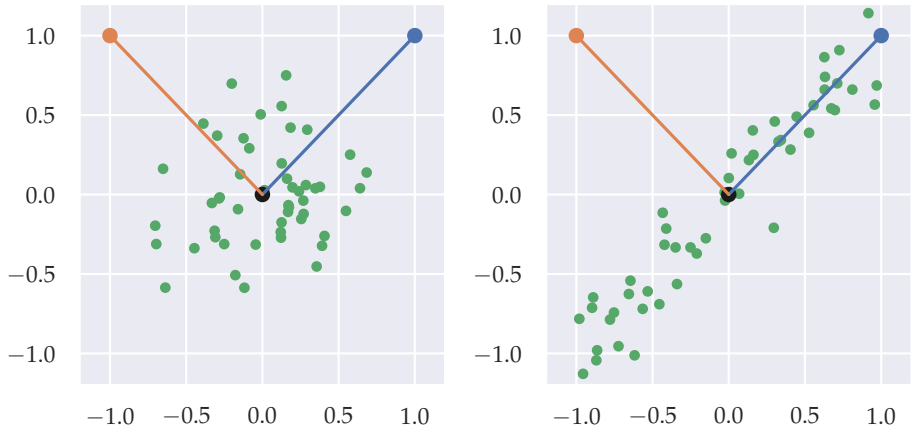

Figure 2.7: Euclidean distance to the centroid of a distribution with two features, when these are uncorrelated (left) and correlated (right).

Figure 2.7 shows two different distributions of sets of data with two features, which are represented as the axes of the figure. Points that are known to belong to these distributions are shown in green, with their centroid represented as a black circle. The two distributions are very different from each other: on the left, the two features are uncorrelated, whereas on the right, there is a clear correlation between them. Consider then the results that the previous definition of Euclidean distance would provide in such a context for two data samples, shown in blue and orange. The Euclidean distance between these points and the centroid are the same for both distributions, which implies that both points are equally likely to belong to them. This is true for the case on the left, with uncorrelated features, but it seems clear that on the right the blue dot should be much more likely to belong to the data distribution than the orange one.

In the context of condition monitoring, it is almost always the case that several features show strong correlations between them: consider e.g. switching device and heat sink temperatures. Moreover, in real applications more than two variables are usually analyzed, which increases the chances of some of them being correlated with one another.

A common distance metric that is often used to overcome this problem is Mahalanobis distance (MD), which can be formulated as:

$$MD_j(\mathbf{x}_j, \boldsymbol{\mu}) = \sqrt{(\mathbf{x}_j - \boldsymbol{\mu})^T C^{-1} (\mathbf{x}_j - \boldsymbol{\mu})} \tag{2.14}$$

Where MD_j is the Mahalanobis distance corresponding to the j-th sample, and C is the covariance matrix. Mahalanobis distance is a multi-dimensional generalization of the common idea in statistics of measuring how many standard deviations away a sample lies from the mean of a distribution [19]. By introducing the inverse of the covariance matrix in the distance calculation, the distance metric becomes unitless and scale invariant, and is made to take into consideration the correlations of features within the data set. Another notable aspect of MD is the fact that when every feature is uncorrelated with every other one, their associated covariance matrix is the identity matrix (assuming normalized features). In such a case, it can be seen that Equation 2.14 is equivalent to Equation 2.13, which translates to Mahalanobis distance being equivalent to Euclidean distance.

Going back to the example in Figure 2.7, the Euclidean distance metric remains constant for both added samples and both distributions, at around $\sqrt{2}$. Calculating MD, on the other hand, results in values around 5 for both samples on the left, while on the right the blue point shows a Mahalanobis distance of around 1.5 and the orange one is around 12. This example shows that MD is much better suited to the task of predicting whether a sample belongs to a distribution.

One shortcoming of Mahalanobis distance is that it makes the assumption that data follows Gaussian distributions, which may hinder its performance in situations where the data is distributed in a much different way. Some metrics, such as k-nearest neighbor distance, can be successfully applied in such situations [5].

Chapter 3

Artificial neural networks

Artificial intelligence (AI) can be broadly defined as the set of techniques that aim to autonomously interpret data, learn from it, and use these learnings to solve specific tasks. Many statistical learning algorithms fall under this definition, such as decision trees, support vector machines, or Kalman filters. The main strength of AI methods is their ability to solve tasks for which explicit algorithms either do not exist, are not known, or not enough information is available for one to be developed. Many tasks that would be extremely complex to describe algorithmically have been successfully solved by AI.

To illustrate this, consider the classic example of recognizing handwritten digits given as images. This task would be trivial for a human, but writing a program that can solve such a problem is not a simple goal. One approach could be to attempt to detect edges and use their shapes and relative positions to make a guess on the digit's value, but this would at best result in low accuracy and long processing time. This is currently one of the most common examples to solve as an introduction to AI classification techniques, which shows how seemingly complex tasks can be made comparatively trivial by making use of machine learning. The current state of the art of AI regarding image pattern recognition is far beyond this, consider e.g. real-time video data analysis for autonomous driving.

A very flexible and widely used approach to AI is that of artificial neural networks (ANN), which also fall under the category of statistical learning methods. Most of the AI techniques developed in this project are based on ANN structures. As such, this chapter aims to provide a theoretical background on the principles of artificial neural networks.

3.1 Motivation

Artificial neural networks have proven to be a fundamental part of many research breakthroughs in recent years, especially in fields such as computer vision, speech recognition, machine translation, and data analysis. ANNs possess several properties that can set them apart from other AI methods and motivate their widespread use [20]:

- Artificial neural networks are universal function approximators, as proven formally in 1989 by Hornik, Stinchcombe, and White [21]. Any arbitrarily complex function, be it linear or nonlinear, can be accurately approximated by an artificial neural network containing at least one hidden layer and making use of non-polynomial activation functions. More details on these concepts can be found in Section 3.2.
- The highly parallel structure of artificial neural networks allows for processing at a higher speed than many other AI algorithms. They are computationally very cheap, which enables complex structures to be executed in short processing times.
- Different ANN structures have been developed with the intention of fine-tuning their performance to specific tasks. Most notably, convolutional neural networks (CNN) are currently achieving state of the art results in image pattern recognition, while recurrent neural networks (RNN) are used with similar results in natural language processing, machine translation, and—relevantly to the present project—time series data analysis [22]. There is currently great research interest in the field of artificial neural networks, which makes resources to develop ANN-based models widely accessible and rapidly improving.
- In the same manner as many—but not all—AI techniques, training an ANN does not require any external information other that the training data itself. As an example of methods that do not fulfill this property, many adaptive algorithms that can be considered to be AI, such as state observers, require previous knowledge of the general rules governing the system they model, and only modify the parameters attached to these rules to fit observations.

Considering these properties together, the advantages provided by artificial neural networks are quite clear: they are able to model arbitrarily complex relationships in data in an accurate and computationally efficient way, and have a large community supporting their development.

On the other hand, the main shortcoming of artificial neural networks is their lack of interpretability. More traditional statistical learning methods such as curve fitting, autorregressive, or moving average models, have parameters serving functions that can be understood by their engineer. Neural networks, however, are often formed by large amounts of parameters in constant interaction with each other and include nonlinear transformations, which often makes their internal operation hard to interpret—although certainly not impossible.

ANNs are thus usually considered to be black boxes, as understanding their internal operation does not necessarily provide any insights on the process they model. However, neural network structures can hardly be properly designed unless their engineer knows where their strengths and weaknesses lie, and which design decisions result in which benefits and shortcomings. Because of this, the fundamentals of ANNs, including their basic operation and training process, will be covered here.

3.2 Neurons and layers

Artificial neural networks are loosely based on the biological neural networks forming the brains of animals. They are constituted by interconnected units commonly referred to as artificial neurons. To illustrate the general structure of a fully-connected ANN, an example graph is shown in Figure 3.1.

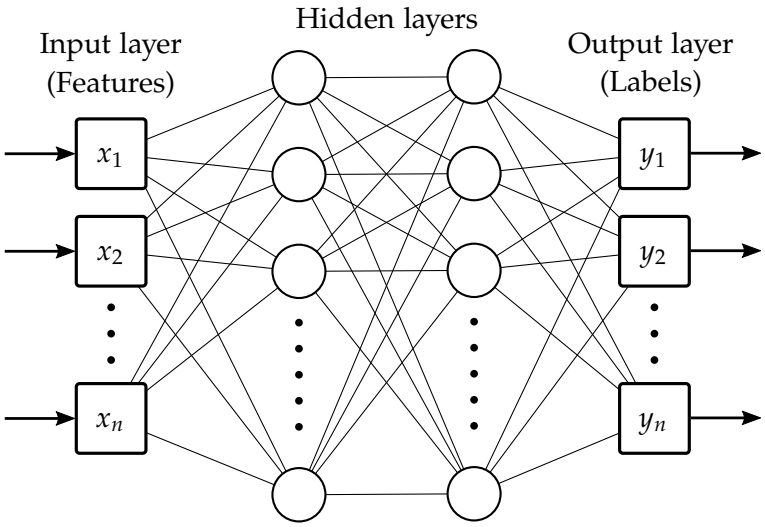

Figure 3.1: General diagram of the layout of a fully-connected ANN with two hidden layers.

Fully-connected ANNs are a common structure where each neuron is connected to

every neuron in their previous layer. In this sense, the input layer of the network receives some input data (often called features), denoted here as $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. This data is then transferred to the neurons in the following layer. Since neither the inputs or the outputs of these intermediate layers are known beforehand, they are often referred to as hidden layers. Each neuron then performs a transformation on its inputs. Due to its simplicity, the most common transformation is a multivariate linear function:

$$x_{l,k} = \mathbf{w}_{l,k} \cdot \mathbf{x}_{l-1} + b_{l,k} \tag{3.1}$$

Where $x_{l,k}$ is the state of the k-th neuron in layer l, $\mathbf{w}_{l,k}$ is the weight vector linking the neuron with the states of the previous layer $\mathbf{x}_{l-1}$, and $b_{l,k}$ is called the bias of the neuron. In this way, each of the states of the previous layer is scaled by some weight coefficient, the results of this scaling are added together, and a bias is finally added to the value of the state. Using simple matrix algebra, the states of a whole layer can be computed at once, assuming the states of the previous layer to be known:

$$\mathbf{x}_l = \mathbf{W}_l \cdot \mathbf{x}_{l-1} + \mathbf{b}_l \tag{3.2}$$

In this way, the weight vector $\mathbf{w}_{l,k}$ becomes a matrix $\mathbf{W}_l$ describing the connections of all neurons in the layer, and the scalar bias $b_{l,k}$ becomes a bias vector $\mathbf{b}_l$. The fact that computing the states of each layer requires knowledge of the states of the previous one makes this type of neural network often be referred to as feedforward ANN. Equation 3.2 already suggests how evaluating a neural network is computationally very efficient: it can be reduced to simple linear algebra operations, which can be very well optimized in most hardware implementations.

The goal of training an artificial neural network is then to obtain the combination of weight matrices $\mathbf{W}_l$ and bias vectors $\mathbf{b}_l$ for each of its layers l that minimizes the error between the states of its output layer $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ and their targets. As such, training data must be structured in such a way that every combination of inputs $\mathbf{x}$ has one or several labels $\mathbf{y}$ attached to them.

Such a model has, however, an important limitation. As every layer performs a linear mapping of the states of its previous one, only linear input–output relationships can be learnt by the neural network, which makes its capability equivalent to that of simple multivariate linear regression. By far the most common solution to overcome this is to apply an element-wise nonlinear function to the state of each unit. This function is commonly called an **activation function**. In this way, neural networks become able to represent nonlinear and complex relationships in data.

However, this also means that the training process of a neural network almost always becomes a non-convex optimization problem, which means that obtaining

the values of weights and biases that best fit a given task is not a problem that can be solved explicitly. Therefore, numerical optimization approaches—commonly gradient-based—must be used, where convergence to the global optimum point is not guaranteed.

Activation functions

For an artificial neural network to possess universal function approximator properties its activation functions must be non-polynomial, as proven formally in [23]. If the ANN is to be trained using gradient-based methods, as it is most often the case, its activation functions should also be continuous and differentiable.

Two similar and commonly used activation functions are the logistic curve or sigmoid function:

$$g(x) = \sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{3.3}$$

And the hyperbolic tangent function:

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.4}$$

In both cases, the values of x would be the states of the units in a given layer, with the functions being applied element-wise. Graphs of these two functions are shown in Figures 3.2 and 3.3.

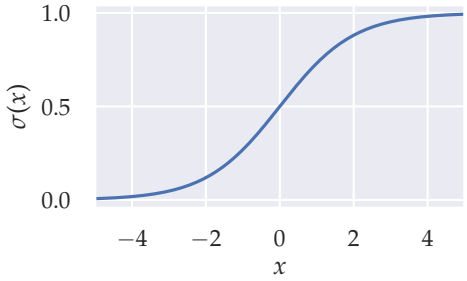

Figure 3.2: The sigmoid function.

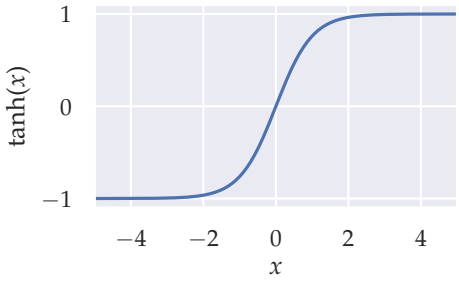

Figure 3.3: The hyperbolic tangent function.

The sigmoid and hyperbolic tangent function have traditionally been favored over other activation functions, as their saturating behavior has long been considered to be biologically plausible, especially in the case of the sigmoid function—although this is currently a point of debate. In artificial neural networks, the hyperbolic tangent is often preferred over the sigmoid, as its behavior near the zero point very

closely resembles the identity function, which can lead to improvements during training.

However, both of these functions present a property commonly referred to as the vanishing gradient problem. For values of x that fall away from the 0-point, their shape becomes increasingly flat, rapidly reaching saturation. Evaluating the gradient of the function at such points yields vanishingly small values, which can greatly hinder gradient-based learning.

In order to mitigate this problem, the rectified linear unit function, often shortened to ReLU, is becoming increasingly widespread. The ReLU function can be expressed as:

$$g(x) = \text{ReLU}(x) = \max(0, x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3.5}$$

Put into words, the ReLU function returns its input value for all positive inputs and 0 for negative ones, as shown in Figure 3.4.

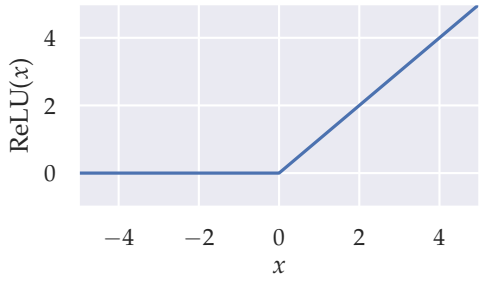

Figure 3.4: The rectified linear unit (ReLU) function.

The similarity between the ReLU function and the identity function makes training neural networks that make use of it more efficient than those with other activation functions, as its gradient can be simply defined as 1 for all positive values of x and 0 otherwise. The ReLU function is then not strictly differentiable, as its derivative contains a jump discontinuity at $x = 0$. When performing numerical optimization, this can be overcome by arbitrarily defining the gradient at this point as either 1, 0, or 0.5.

Activation functions can also be used to shape the outputs of a neural network in order to improve its performance. For instance, if an ANN is used to make predictions on a variable that cannot take negative values, such as temperature (in degrees Kelvin), a ReLU activation function can be applied to the output layer of the network. The sigmoid function is very often applied to the output of neural net-

works performing binary classification tasks—where they are to predict whether a set of input data either belongs to a certain class or it does not. Such a model would then output values within the continuous range between 0 and 1, which can be interpreted as the confidence of the network in either a negative or a positive prediction. In principle, an adequately trained ANN model would still be able to learn such relationships regardless of its output activation function, but properly selecting it will improve performance and eliminate potential physically impossible outputs, which could occur for some input outliers.

Many other activation functions have been proposed in literature and successfully applied in practice, with this being an active area of research. Many of these, however, are modifications to the sigmoid, hyperbolic tangent, and ReLU functions, and offer only slight performance increases over their more basic versions. Most state of the art results in classification problems are obtained using ANNs making use of the ReLU function in their hidden layers, together with a sigmoid output for binary classification or a softmax—a multivariate generalization of the sigmoid function—for multi-class classification [24].

3.3 Back-propagation

Up to this point, this chapter has been concerned with forward-propagation: that is, how a neural network obtains the states of its output layer from known inputs. During training, a cost function is introduced to penalize deviations between the predictions made by the neural network and its desired outputs. Back-propagation [25] is an algorithm that allows this information to flow backwards from the network's output layer towards its input, computing the gradients of cost with respect to each of its internal parameters (weights and biases).

More formally, back-propagation obtains the gradient of a cost function J with respect to the network's internal parameters $\boldsymbol{\theta}$ (formed by the weight matrices and bias vectors). The goal of back-propagation is then to obtain the values of this gradient, which can be expressed as:

$$\nabla J(\boldsymbol{\theta}) = \left(\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \cdots, \frac{\partial J}{\partial \theta_{n+m}}\right) = \left(\frac{\partial J}{\partial w_1}, \cdots, \frac{\partial J}{\partial w_n}, \frac{\partial J}{\partial b_1}, \cdots, \frac{\partial J}{\partial b_m}\right) \tag{3.6}$$

Where n is the total number of weight and m the total number of bias parameters. Once these gradients are known, they can be used in numerical optimization algorithms to update the parameters of the ANN in a way that minimizes the deviation between predictions and desired outputs (labels). As such, back-propagation forms the backbone of gradient-based training of neural networks, which is currently the most widespread approach.

3.3.1 Cost functions

The choice of a cost function largely depends on the particular task that the neural network is to perform. This project is centered on two types of problem: regression (making predictions on the future values of one or several variables, such as temperature) and binary classification (predicting whether or not a system is operating in a faulty manner).

Mean squared error (MSE) and mean absolute error (MAE) are two of the most common loss functions for regression problems. Mean squared error can be defined for a single variable as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.7}$$

Similarly, mean absolute error would be:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.8}$$

In both cases, N is the number of samples from which error is calculated, y_i is the known target value of the variable y of the i-th sample, and $\hat{y}_i$ is the corresponding prediction made by the ANN. To compare these two very similar functions, both are plotted in Figure 3.5.

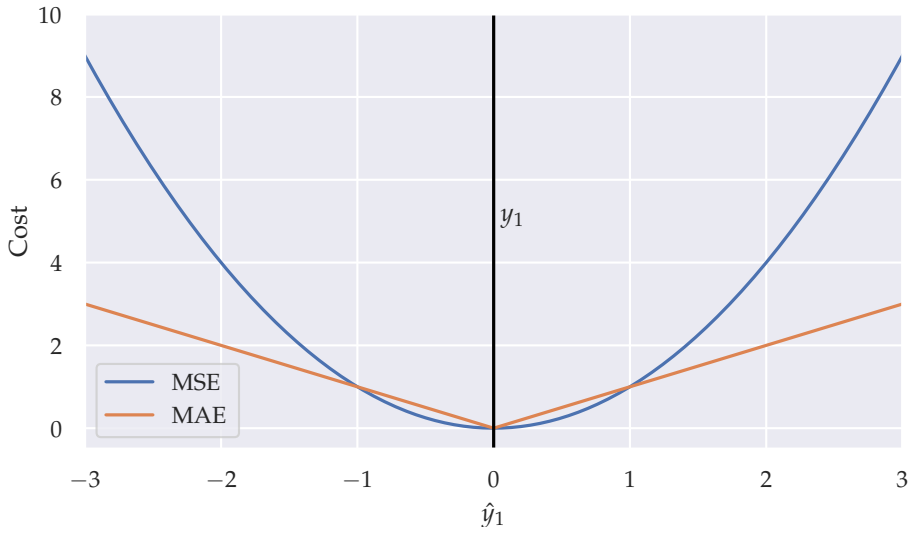

Figure 3.5: The mean squared error (MSE) and mean absolute error (MAE) cost functions.

Figure 3.5 shows a simple example where the cost functions are evaluated for a single sample with a target $y_1 = 0$. The graph shows how MSE is lower than MAE

for deviations from the target between -1 and 1, while MSE is larger for all other values. This means that MSE is more sensitive to outliers, while MAE is more sensitive to accumulated small errors. As such, MSE should be chosen for regression problems where the main priority is to remove single large prediction errors, giving more weight to outlier samples, whereas MAE is preferable in situations where fine tuning predictions is more important than large errors on single outliers.

Both of them share two important properties: a completely accurate prediction would yield a cost value of 0, while—theoretically—an infinitely large error would also result in an infinite cost.

For binary classification the most common cost function is cross-entropy loss (also called logistic loss or log loss). For a single sample, binary classification implies that it either belongs to the target class or it does not. Binary cross-entropy (BCE) can therefore be calculated from a single value using complimentary probability, following its basic definition [26]:

$$BCE = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y) \log\big(1-\hat{y}\big) \tag{3.9}$$

Where $p_i = \{y, 1-y\}$ denotes each possible target value, where y is either 0 or 1, and $q_i = \{\hat{y}, 1-\hat{y}\}$ denotes each possible prediction, with $\hat{y}$ being interpreted as the predicted probability of the sample belonging to the class. In the context of machine learning, cost functions are often evaluated from batches of samples. For this purpose, binary cross-entropy is defined as the average cross-entropy loss:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \hat{y}_i + (1-y_i) \log\big(1-\hat{y}_i\big)) \tag{3.10}$$

Where N is the number of samples used to find the loss. This function is shown in Figure 3.6 for a single positive-class sample ($y_1 = 1$), as a function of the predicted probability $\hat{y}_1$.

Figure 3.6 illustrates how binary cross-entropy fits with binary classification: a completely accurate prediction results in a cost of 0, while a completely wrong prediction would yield infinite cost.

3.3.2 Obtaining gradients

As introduced above, the goal of back-propagation is to obtain the values of the gradient vector of the cost function with respect to each of the parameters of the ANN. These gradients are then to be used in an optimization algorithm to update the parameters in such a way that cost is made to decrease. It must be kept in mind

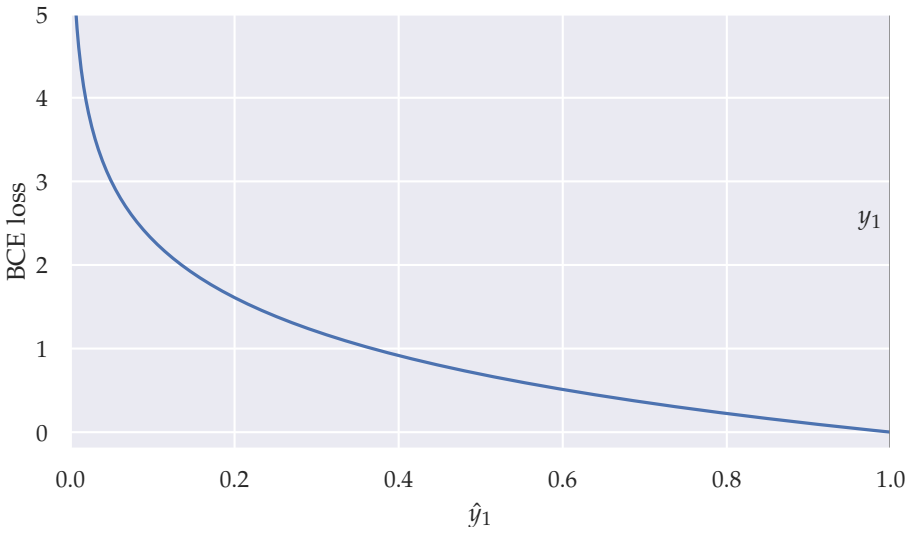

Figure 3.6: The binary cross-entropy (BCE) cost function.

that back-propagation is only concerned with the computation of these gradients, and is a separate process from the optimization of the network.

In order to simplify notation, the back-propagation algorithm developed in this section will be attached to a single training sample, as generalization to batches—the most common way to train ANNs—is relatively straightforward. Let J denote the cost function, and $w_{i,j}$ denote the weight connecting the i-th neuron in layer $L-1$ and the j-th neuron in layer L. The gradient of the cost function with respect to such a weight can then be found using the chain rule of calculus:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial x_j}{\partial w_{ij}} \frac{\partial a_j}{\partial x_j} \frac{\partial J}{\partial a_j} \tag{3.11}$$

Where x_j denotes the state of the j-th unit before its activation function g has been applied, and $a_j = g(x_j)$ denotes the state of the j-th unit after applying this activation. The goal of back-propagation is then to obtain the values of every derivative in the right-hand side of Equation 3.11. Therefore, from Equation 3.1:

$$\frac{\partial x_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \Big(\sum_{k=1}^{n} w_{kj} a_k + b_j \Big) = \frac{\partial}{\partial w_{ij}} (w_{ij} a_i) = a_i \tag{3.12}$$

Where $k = (1, 2, \cdots, n)$ denotes each neuron in the layer $L-1$. The next term in Equation 3.11 depends on the chosen activation function for the j-th neuron. For a rectified linear unit activation function, the term becomes:

$$\frac{\partial a_j}{\partial x_j} = g'(x_j) = \mathrm{ReLU}'(x_j) = \begin{cases} 1 & x_j > 0 \\ 0 & x_j < 0 \end{cases} \tag{3.13}$$

In software implementations, the undefined gradient at $x_j = 0$ is instead set at 0, 0.5 or 1, rather than raising an error. For the other two activation functions used in this project, the sigmoid and hyperbolic tangent functions, their gradients are, respectively:

$$\frac{\partial a_j}{\partial x_j} = g'(x_j) = \sigma'(x_j) = \sigma(x_j)(1 - \sigma(x_j)) \tag{3.14}$$

$$= \tanh'(x_j) = 1 - (\tanh^2(x_j)) \tag{3.15}$$

From these three equations, it is clear that all three activation functions have easily computable gradients, which translates to faster ANN training processes. The last gradient in Equation 3.11 depends on the choice of activation function. Assume first that the layer L, to which the j-th unit belongs, is the output layer of the network, and consider a mean squared error cost function:

$$\frac{\partial J}{\partial a_j} = \frac{\partial}{\partial a_j}(y_j - a_j)^2 = 2(a_j - y_j) \tag{3.16}$$

With y_j being the label attached to a_j. For a mean absolute error cost function, the gradient would be:

$$\frac{\partial J}{\partial a_j} = \frac{\partial}{\partial a_j}|y_j - a_j| = \begin{cases} -1 & y_j > a_j \\ +1 & y_j < a_j \end{cases} \tag{3.17}$$

Similarly to the ReLU activation function, software implementations of this gradient do not maintain its undefinedness when $y_j = a_j$, and instead return a value of 1, 0, or -1. Finally, the gradient corresponding to binary cross-entropy can be found to be:

$$\frac{\partial J}{\partial a_j} = -\frac{\partial}{\partial a_j}(y_j \log a_j + (1 - y_j)\log(1 - a_j)) = -\frac{y_j}{a_j} + \frac{1 - y_j}{1 - a_j} \tag{3.18}$$

These expressions have been developed assuming that the unit j belongs to the output layer. If j is instead set to belong in an arbitrary layer, this derivation is less straightforward. Every gradient in Equation 3.11 remains the same except for $\partial J/\partial a_j$. To derive it, consider first the cost function $J(a_j)$ to be a function of every neuron belonging to the next layer $(L + 1)$ that is connected to j. The gradient of J is then given by the total derivative of the cost function with respect to each of these neurons.

$$\frac{\partial J}{\partial a_j} = \sum_{l \in L+1} \Big(\frac{\partial J}{\partial a_l} \frac{\partial a_l}{\partial x_l} \frac{\partial x_l}{\partial a_j} \Big) = \sum_{l \in L+1} \Big(\frac{\partial J}{\partial a_l} \frac{\partial a_l}{\partial x_l} w_{jl} \Big) \tag{3.19}$$

The gradient $\partial a_l/\partial x_l$ depends on the activation function of neuron l, and can be found as developed above. Equation 3.19 means that the gradient $\partial J/\partial a_l$ can be

found assuming that the gradients have already been obtained for every neuron in the layer immediately closer to the output. This is where the algorithm gets its name from, as gradients propagate backwards from the network's output layer towards its input.

At this point, every gradient on the right-hand side of Equation 3.11 can be found for any layer in a fully connected ANN, and thus the goal of the algorithm—the partial derivative of the cost function with respect to each weight—can be obtained. To obtain this gradient with respect to a bias instead of a weight, only a simple change must be applied to Equation 3.11, resulting in:

$$\frac{\partial J}{\partial b_j} = \frac{\partial x_j}{\partial b_j} \frac{\partial a_j}{\partial x_j} \frac{\partial J}{\partial a_j} \tag{3.20}$$

Where the only different gradient on the right-hand side is:

$$\frac{\partial x_j}{\partial b_j} = \frac{\partial}{\partial b_j}\Big(\sum_{k=1}^{n} w_{kj} a_k + b_j \Big) = \frac{\partial}{\partial b_j}(b_j) = 1 \tag{3.21}$$

Combining these expressions, the values of the complete gradient vector $\nabla J(\boldsymbol{\theta})$ can be computed. This gradient can then be used in an optimization algorithm to update the weights and biases of the ANN in a way that decreases cost. In a general way, the back-propagation algorithm can be summarized as:

Algorithm 1: The back-propagation algorithm for a fully-connected ANN.

Require: ANN parameters at iteration k, $\boldsymbol{\theta}^k$.
Require: Number of ANN layers N_L.
Require: Activation functions $\mathbf{g}$ of each unit.
Require: Cost function J.
Require: Batch of training input data $\mathbf{z}$ and its attached labels $\mathbf{y}$.
Compute the neural network's predictions $\hat{\mathbf{y}}$ from $\boldsymbol{\theta}^k$, $\mathbf{g}$, and $\mathbf{z}$.
Evaluate the loss attached to the batch using J, $\mathbf{y}$, and $\hat{\mathbf{y}}$.
Initialize a layer counter $l = N_L$.
while $l > 0$ **do**
    Compute the gradients of the cost function with respect to each parameter in layer l: $\nabla_l^k J(\boldsymbol{\theta}^k)$.
    Move one layer closer to the input: $l = l - 1$.
end
return $\nabla^k J(\boldsymbol{\theta}^k)$, the gradients attached to every parameter in the ANN.

3.4 Optimization algorithms

Once every gradient is known for a given training iteration, an optimization algorithm must be used to update the parameters of the neural network. For example, using a simple steepest descent algorithm, the parameter update step would be:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \nabla^k J(\boldsymbol{\theta}^k) \tag{3.22}$$

Typically, the optimization—training—of a neural network is performed on batches of data. This means that the true gradient $\nabla^k J(\boldsymbol{\theta}^k$ is often unknown, and is instead estimated for each batch. Using all the available training data at once in every training step would result in an accurate calculation of the gradient vector, but this is often undesirable. By using smaller batches of data, a certain degree of randomness is introduced in the training process, which in many cases prevents the optimization process from stopping at globally high local minima. It is important, however, that the estimate of the gradient remains unbiased: for this reason, the samples used in each batch should be chosen randomly. For this reason, optimization algorithms applied in this context become stochastic.

In general, the larger the batch size, the faster the training, as fewer steps of the back-propagation and optimization algorithms need to be computed. On the other hand, smaller batch sizes introduce more stochasticity to the process, which can lead to convergence to better cost function minima.

Steepest descent becomes mini-batch stochastic gradient descent, or stochastic gradient descent (SGD) for short. Pseudo-code for this is provided in Algorithm 2 [26].

Algorithm 2: Stochastic gradient descent (SGD)

Require: Learning rate η.
Require: Initial parameters $\boldsymbol{\theta}$.
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ with their corresponding targets $\{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$.
  Compute gradient estimate $\hat{\mathbf{g}}$ using the back-propagation algorithm.
  Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \hat{\mathbf{g}}$.
end

As convergence criteria can be hard to define in the context of ANN training, stopping criteria are usually defined in terms of the number of iterations to be performed. As this number typically takes on very large values, iterations are

commonly grouped in *epochs*, which are defined as the optimization process with sampling in batches of all of the available training data a single time.

The main challenge present in using stochastic gradient descent is tuning the learning rate parameter η. The learning rate must be a positive scalar, as decreasing the value of the cost function requires updating the parameters of the network in the direction opposite to the one given by the gradient vector. It should also be smaller than one, as otherwise gradients would tend to become indefinitely large. Moreover, a single scalar learning rate might be too simplistic for many situations, as cost is often highly sensitive to changes in some directions of the parameter space, and highly insensitive to others.

This type of algorithms commonly follow a simple underlying rule: if the partial derivative of the loss with respect to a parameter maintains the same sign during consecutive iterations, its attached learning rate should increase; if it changes sign, the learning rate should decrease.

Algorithm 3: The Adam algorithm [27]

Require: Learning rate ϵ (suggested default: 0.001).
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 (suggested defaults: 0.9 and 0.999 respectively).
Require: Small constant δ, for numerical stabilization.
Require: Initial parameters $\boldsymbol{\theta}$.
Initialize 1st and 2nd moment variables $\mathbf{s} = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$.
Initialize time step $k = 0$.
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ with their corresponding targets $\{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$.
  Compute gradient estimate $\hat{\mathbf{g}}$ using the back-propagation algorithm.
  $k \leftarrow k + 1$.
  Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$.
  Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$.
  Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^k}$.
  Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^k}$.
  Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\epsilon \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}}$ (element-wise operations).
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
end

A wide range of adaptive learning rate algorithms have been proposed and applied successfully to the training of ANNs, with some algorithms outperforming others in different situations. Some of the most popular optimization algorithms

actively in use are SGD with and without momentum, RMSProp with and without momentum, AdaDelta, and Adam; but there is no consensus on whether a single best algorithm exists [26]. Typically, SGD allows ANNs to converge to better optimum points than other algorithms, provided that the choice of learning rate and batch size is sufficiently adequate. However, since SGD keeps its learning rate constant, it is also among the slowest algorithms to converge, requiring a much larger number of iterations than adaptive learning rate algorithms. Adam [27], one such algorithm, is among the fastest, while also providing good performance and requiring little hyperparameter tuning. Pseudocode for Adam is provided in Algorithm 3.

Adam incorporates both first and second moment estimates. The first order moment is obtained by accumulating the gradient estimate $\hat{\mathbf{g}}$ in an exponentially decaying rate. The second moment is calculated in a similar manner, but using $\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$, i.e. the element-wise square of the gradient vector. Using the suggested default parameters, much more weight is given to the previous value of the moment estimates than to the gradient terms, especially for the second moment. This allows their values to remain more stable and less sensitive to sharp gradient changes.

These moment estimates are bias-corrected, to account for their initialization at zero. The parameter update is calculated as the ratio between the first moment estimate and the square root of the second, with a small constant δ included to prevent division by zero.

3.5 Recurrent neural networks

Recurrent neural networks, or RNNs, are a family of neural networks specialized for sequential data processing. They are based on one of the early ideas found in machine learning and statistical models of the 1980s: sharing parameters across different parts of a model [26]. Recurrent networks can also process sequences of variable length, by being able to account for both time-dependent and time-independent relationships in data. As such, they have been successfully used in time-series modeling and in language processing.

The application of RNNs to language processing can be used to illustrate the advantages that this type of neural network provides over the simpler feedforward ANNs described up to this point. As an example, consider the two sentences "I moved to Denmark in 2018" and "In 2018, I moved to Denmark". If a machine learning model is trained to extract the year in which the narrator moved to Denmark, 2018 should be recognized regardless of its position. A traditional fully-connected

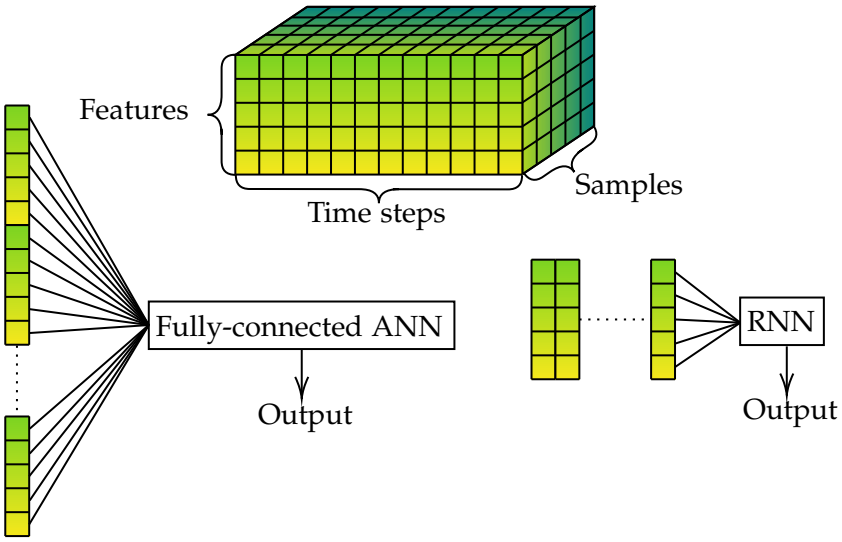

Figure 3.7: Comparison between how fully-connected ANNs and RNNs treat data.

feedforward network would have separate parameters for each position in the sequence of words, and so would have to learn all of the rules of the language separately for each word input, requiring a complex internal structure. An RNN model would instead apply the same calculations across different steps, learning to find the desired information regardless of its position.

In the context of this project, RNNs would be able to detect changes in parameters in a time independent-way. This means, for example, that a recurrent neural network could learn to interpret a change in heat sink temperature in an equivalent way regardless of the time in which this change occurs.

Figure 3.7 shows how fully-connected feedforward ANNs interpret data in a different way than recurrent neural networks do. Time series data is structured in a three-dimensional way, following the structure (samples, time steps, features). A fully-connected ANN requires unidimensional input data: this means that time-series data must be flattened to have a shape (samples, time steps * features) before it can be fed to the ANN. An RNN instead processes the data at each time step, making use of the similarities and differences between time steps to make its predictions. In this way, each member of the output of a recurrent layer is a function of the previous inputs, and is produced using the same update rule applied to the previous outputs. Recurrent neural networks usually incorporate fully connected layers as well as recurrent ones, to enable the model to find patterns in time-dependent data in a more efficient way.

3.5.1 Internal structure

The units of a recurrent layer make use of their previous values to obtain their current state. The states $\mathbf{h}^{(t)}$ of a recurrent layer at time step t can thus be generally defined as [26]:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}) \tag{3.23}$$

Where f is the transformation applied by the layer, a function of its previous states $\mathbf{h}^{(t-1)}$, current inputs $\mathbf{x}^{(t)}$, and internal parameters $\boldsymbol{\theta}$. In order to better understand how calculations are performed in a recurrent neural network, a computational graph of a recurrent layer is shown in Figure 3.8.

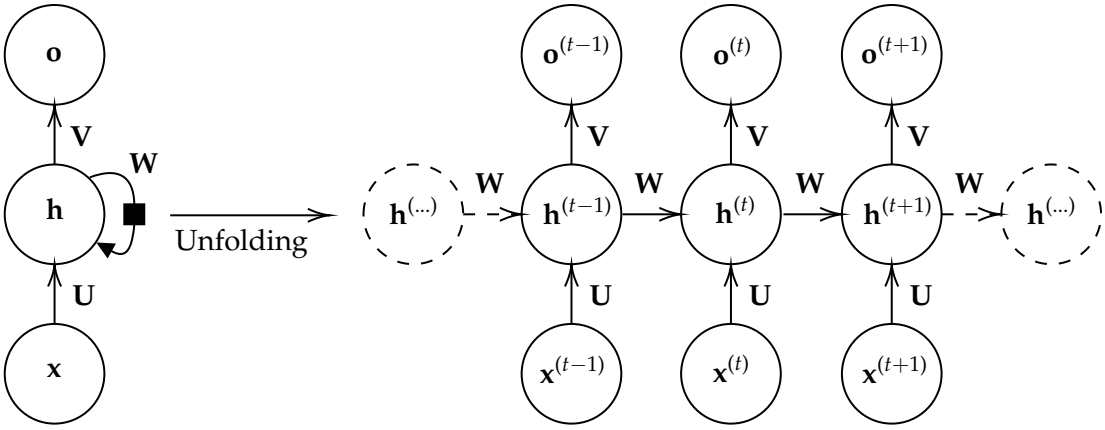

Figure 3.8: Unfolding the computational graph of a simple recurrent neural network.

The computational graph of Figure 3.8 shows two equivalent ways of representing a basic recurrent layer. On the left, its structure is given in a more compact way by making use of a delay block, shown as a black square. In this way, an input $\mathbf{x}$ is mapped to the state $\mathbf{h}$ of the layer using a weight matrix $\mathbf{U}$. This state is then stored through another weight matrix $\mathbf{W}$ and the delay block. Finally, the output $\mathbf{o}$ is obtained using the weight matrix $\mathbf{V}$. The state $\mathbf{h}$ at a time step t can then be found through:

$$\mathbf{h}^{(t)} = g(\mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}) \tag{3.24}$$

Where g is the layer's activation function. Unfolding this graph removes the need for a delay block. This helps illustrate the paths through which information flows by providing a exact description of the computations to perform. The unfolded graph can also be used to show how back-propagation is performed in a recurrent neural network. Here, gradients are not only back-propagated from the output to the input of the network, but also from the last time step to the first. At the final time step τ, the gradient of the cost function J (attached to the output $\mathbf{o}$) with

respect to the current state $\mathbf{h}^{(\tau)}$ is given by:

$$\nabla_{\mathbf{h}^{(\tau)}} J = \mathbf{V}^T \nabla_{\mathbf{o}^{(\tau)}} J \tag{3.25}$$

Iterations can then be performed from $t = \tau - 1$ down to $t = 1$, noting that $\mathbf{h}^{(t)}$ influences both $\mathbf{o}^{(t)}$ and $\mathbf{h}^{(t+1)}$. Therefore:

$$\nabla_{\mathbf{h}^{(t)}} J = \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{h}^{(t+1)}} J) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{o}^{(t)}} J) \tag{3.26}$$

$$= \mathbf{W}^T (\nabla_{\mathbf{h}^{(t+1)}} J) \mathbf{J}_g + \mathbf{V}^T (\nabla_{\mathbf{o}^{(t)}} J) \tag{3.27}$$

Where $\mathbf{J}_g$ indicates the Jacobian matrix of the layer's activation function at time $t + 1$. For a hyperbolic tangent activation function, for example, its Jacobian would be the diagonal matrix containing the elements $1 - (h_i^{(t+1)})^2$, where i indicates each hidden unit of the layer. The rest of the gradients can then be computed using the same back-propagation algorithm used for deep fully-connected networks.

This simple formulation of a recurrent layer has an important shortcoming in the form of the vanishing gradient problem. The more layers a fully-connected network has, the more prone to vanish are its gradients, as gradients close to the input are scaled down by the ones in outer layers. In an analogous way, for a simple recurrent layer, gradients close to the initial time step of a series are scaled down by the ones in later steps, resulting in them tending to vanish. For this reason, the simple layer shown in Figure 3.8 is very hard to train for long sequences, and would rarely be applied in practice.

3.5.2 Long short-term memory layers

To mitigate this problem, Hochreiter and Schmidhuber presented the long short-term memory (LSTM) model [28], which has since been found to be very successful in many applications. The core idea behind this type of layer is to introduce additional self-loops through which gradients can flow over long durations. This is implemented by making use of an additional internal loop besides the one described for a simple recurrent layer. The flow of information within each LSTM cell is controlled by a forget gate, an input gate, and an output gate, which allow gradients attached to relevant information to remain stored in the cell for longer times by removing unnecessary information.

A block diagram of an LSTM cell is shown in Figure 3.9. Each sigmoid (σ) and hyperbolic tangent (tanh) block have a weight matrix and bias vector attached to them, applied to their input before the corresponding nonlinear transformation.

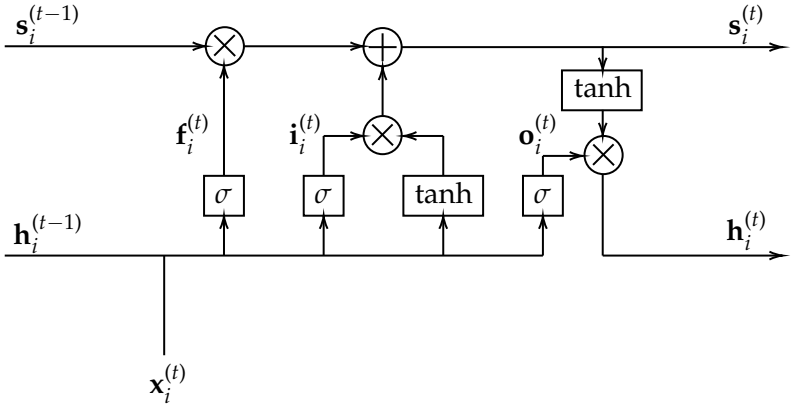

Figure 3.9: Internal structure of an LSTM cell at time step t.

These parameters are learnt through back-propagation in the same way as those belonging to other layers.

The inputs of the cell i at time t are the current layer input vector $\mathbf{x}_i^{(t)}$, the previous hidden layer vector $\mathbf{h}_i^{(t-1)}$, and the previous state vector $\mathbf{s}_i^{(t-1)}$; while its outputs are the current hidden layer vector $\mathbf{h}_i^{(t)}$ and state vector $\mathbf{s}_i^{(t)}$. Initially, $\mathbf{x}_i^{(t)}$ and $\mathbf{h}_i^{(t-1)}$ are concatenated and used as the inputs of the three gates of the cell: the forget gate (with output $\mathbf{f}_i^{(t)}$), the input gate ($\mathbf{i}_i^{(t)}$), and the output gate ($\mathbf{o}_i^{(t)}$). Each of this gates applies a sigmoid operation to force the values of its output vector to range between 0 and 1. In this way, element-wise multiplication involving these vectors results in a scaling that is able to remove information (multiplying by 0) or preserve it. During training, the weights associated to each gate will converge to values that result in relevant information being favored over time and irrelevant information being discarded.

The forget gate controls which information from the previous cell state is conserved into the current one, based on the previous hidden layer vector and the current input vector. The input gate determines how the previous hidden layer vector and current input vector are stored into the current state. Finally, the output layer uses this same information to determine which information from the current state vector will constitute the current hidden layer vector.

By adding these gating operations, LSTM cells are able to maintain the flow of relevant gradients for longer times than basic recurrent layers, making their training significantly easier and thus improving their performance. Many modifications on the LSTM structure have been proposed in literature, usually with the goal of simplifying the topology to obtain improvements in the training process. One such type of cell, the gated recurrent unit (GRU) presented in 2014 [29] appears to be

gaining interest, as it is often able to reach comparable results to LSTM with lower training requirements.

Chapter 4

Implementation

Having provided a basic theoretical framework for the considered drive system and for general neural network models, this chapter aims to describe the process followed to implement the proposed condition monitoring schemes. Its first section describes the steps taken for data collection and processing, followed by details on the two main considered AI approaches: developing a surrogate model of the cooling system and directly predicting the health status of the system.

4.1 Data acquisition

Experimental tests were carried out at the Danfoss facilities in Gråsten, Denmark, in order to collect data from a physical drive setup for use in this project. The testing setup consisted of an induction motor with the following nominal specifications:

- **Input voltage (three-phase, rms):** 400 V.
- **Input current (three-phase, rms):** 8.6 A.
- **Line frequency:** 50 Hz.
- **Mechanical power:** 4 kW.
- **Torque:** 26 Nm.
- **Speed:** 1475 rpm.

This motor was powered by a Danfoss VLT®FC302 drive, with the following specifications:

- **Output power:** 4 kW.
- **Maximum output current (three-phase, rms):** 16 A.
- **Nominal output current (three-phase, rms):** 10 A.
- **Nominal switching frequency:** 4 kHz.

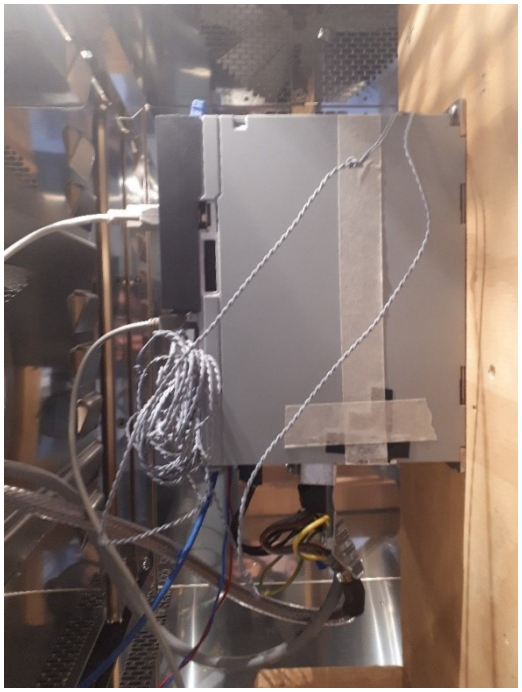

Figure 4.1: Photograph of the FC302 drive as used in experiments.

The load torque of the driven motor is provided by another electrical machine. The drive is placed in a cooling chamber in order to monitor and control its ambient temperature. It is connected to a computer through an FS-485 converter, and monitored using the Danfoss VLT®Motion Control Tool (MCT 10) software. The default sensors of the drive provide readings of internal heat sink temperature and control card temperature. Additional temperature sensors were installed in the drive to measure the temperatures of inlet air, outlet air, and the exterior of the heat sink. The other signals selected for logging are measured switching frequency, output power, output voltage, output current, motor torque, and motor speed. Additionally, the default software of the control card was modified to allow for access to fan speed reference readings. The physical drive setup is shown in Figure 4.1.

The recorded data of the drive is uploaded in real time to the Danfoss DrivePro®Remote Monitoring cloud, from where it can be accessed online. Data can be accessed manually by logging into the platform and downloading the desired variables as comma-separated values (*.csv*) files. Alternatively, the representational state transfer application programming interface (REST API) can be used to access the data programmatically using the HTTP protocol. In Python, this can be done using the *Requests* library [30]. Each parameter logged in the cloud is attached to a unique datanode, which can be used in calls to the HTTP request *get* method. The *get* method then returns a JavaScript Object Notation (*.json*) file that can be parsed using the *json* Python library [31], and converted to a *DataFrame* object using the *pandas* library [32]. Formatting the data as *DataFrames* objects provides access to many efficient methods for data processing and analysis.

4.1.1 Design of experiments

Experimental setpoints were defined as combinations of some of the available variables, within the operating limits of the drive and motor:

- Motor speed: 1350 rpm, 1500 rpm.
- Ambient temperature (cooling chamber): 22°C, 30°C.
- Load torque: 0 Nm, 5 Nm, 24 Nm.
- Switching frequency: 4 kHz, 10 kHz.
- Simulated degraded conditions: replicating dust accumulation by placing cotton on the heat sink, and partially blocking the air passage of the cooling system. If feasible, some tests can also be performed after removing one or several of the fan blades. An image of the drive with a partially blocked air passage is shown in Figure 4.2.

A total of 16 tests were planned to be performed for healthy operation and 16 more for degraded operation, under simulated faulty conditions. This, however, ended up not being feasible. Each test requires around 10 minutes to reach thermal steady-state equilibrium. Sampling is performed at a frequency of 1 Hz, and timestamps for the beginning and the end of every test are recorded to be used for accessing the data. Sampling frequency is chosen in such a way that temperature transients are adequately represented in the recorded data, without placing much emphasis on electrical and mechanical transients, which are typically much faster. As the employed temperature sensors have a sensitivity of 1°C, not much information would be gained from increasing the sampling frequency.

Figure 4.2: Photograph of the FC302 drive with a partially blocked (slightly under 50%) air passage.

Due to practical limitations, it was not possible to complete all the desired tests within the available time frame of this project. The recorded experiments are thus limited to a single type of fault (50% blocked air intake). The testing points were also modified from the original design, including a wider range of motor speeds, switching frequencies, and torque loads for healthy operation of the drive. Table 4.1 shows the varying parameters of each recorded test.

The recording of measurements is affected by computational delays that make the sampling somewhat unreliable. As such, the different measured variables are each recorded at slightly differing times, with sampling frequencies slightly above or below 1 Hz. To account for this, the experimental data series are interpolated to estimate the values of each variable at exact periods of one second.

4.1.2 Data preprocessing

To visualize the shape of the temperature measurements, the recorded values for two tests are plotted in Figure 4.3. For the degraded test, the air passage of the setup was blocked by 50%. Both tests were performed at a motor speed of 1350 rpm and torque of 24 Nm. The healthy test was carried out at a switching frequency of 8 kHz, while the degraded one had its switching frequency reduced to 7 kHz.

The figure shows that heat sink internal (in blue) and external (in green) temperatures are very strongly correlated: their Pearson correlation coefficient is over

Test no.	Fault type	Motor speed (rpm)	Switching frequency (kHz)	Torque load (Nm)	Chamber temperature (°C)
1	None	1500	10	0 to 24	22
2	None	1500	4	24 to 0	22
3	None	1350	10	24	22
4	None	0	0	0	22
5	None	1500	10	12	22
6	None	1500	10	17	22
7	None	1100	10	17	22
8	None	1100	6	0	22
9	None	1500	6	0	30
10	None	1500	10	24	30
11	None	1500	4	24	30
12	None	1500	10	17	30
13	None	1500	10	7	30
14	None	1350	10	24	30
15	None	1350	10	17	30
16	50% block	1500	10	0	22
17	50% block	1500	7	24	22
18	50% block	1500	4	24	22
19	50% block	0	0	0	22
20	50% block	1350	7	24	22

Table 4.1: Setpoints of each recorded experimental test.

99% in both experiments. This high level of correlation indicates that, at least for these two experiments, the added external heat sink sensor does not provide much additional information on the behavior of the system. However, this additional measurement could prove very useful to monitor cases where the heat sink thermal interface material is degraded, as internal heat sink temperature would increase while external temperature would remain at a similar or lower level. As expected, these temperatures follow a negative exponential curve, reaching steady-state equilibrium at typical values between 45°C and 55°C, with internal heat sink temperatures always staying above external ones.

Control card temperature spikes at the very beginning of operation, where computational load is at its highest, and then decays to equilibrium. For these two examples, steady-state control card temperature was higher in the degraded test. As expected, air inlet temperature remains practically constant at the controlled temperature of 22°C. Air outlet temperature stays a few degrees higher, and also

appears to remain higher in the degraded test than in the healthy test.

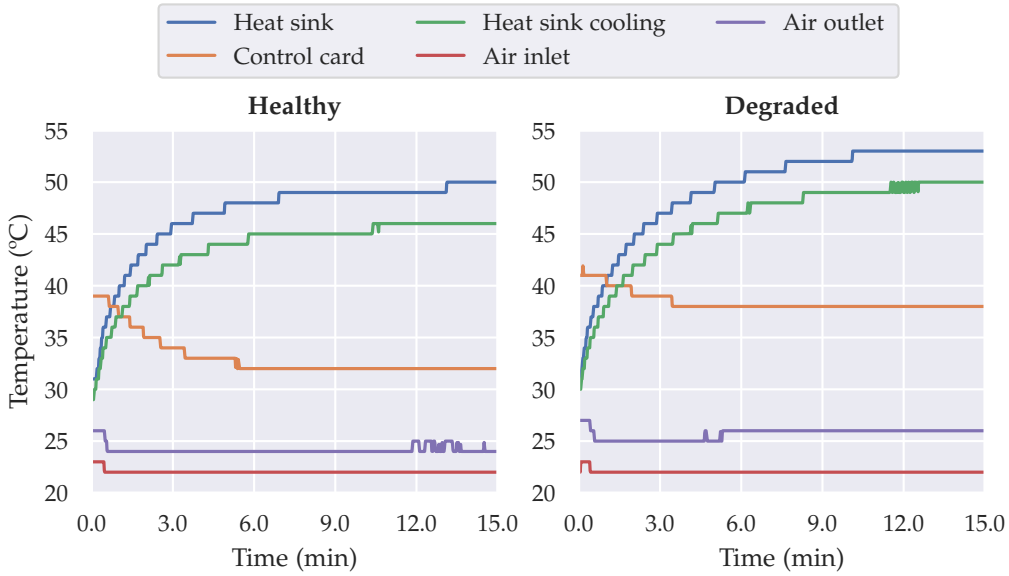

Figure 4.3: Temperature measurements over the course of two experimental tests.

From these two examples, it would appear that distinguishing between degraded and healthy operation should be quite straightforward: it would be enough to compare steady-state temperature measurements, as their values are higher in degraded operation than in healthy operation (except for air inlet temperature, which is expected to remain mostly constant). However, this would be an erroneous conclusion.

The temperatures of the drive are subject to many variables besides the condition of the cooling system: for example, an increase in load would also result in an increase in drive power losses and thus in temperature—a degraded system with a lower load could show similar temperature profiles to a healthy system with a higher load.

This illustrates the need for more complete data. A larger number of variables made available to the condition monitoring algorithm would improve its ability to separate the effects on system performance due to degradation from those caused by other sources. To obtain a reliable monitoring scheme, it is also fundamental to record data under many possible operating conditions: the more data that is made available to train the AI algorithm, the better its predictions will be.

Figure 4.4 shows other variables recorded during the same two experiments shown

in Figure 4.3, with blue lines corresponding to the healthy experiment and orange ones to the degraded one.

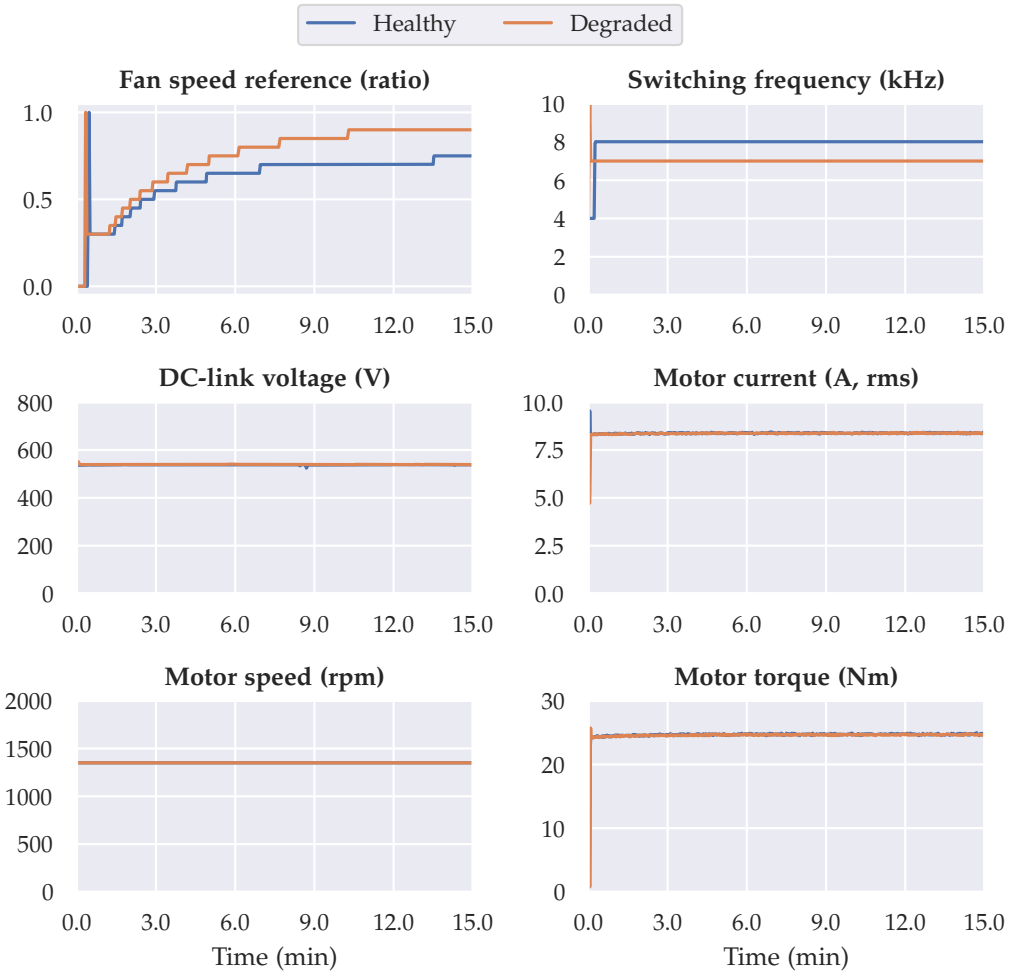

Figure 4.4: Plots of several recorded variables over the course of two experimental tests.

The speed reference of the cooling fan is given as the ratio between its reference and maximum speeds. The shape of this variable suggests that this control signal is proportional to the heat sink temperature measurements, with the cooling fan running faster at higher temperatures. Switching frequency remains at 8 kHz during the healthy test and at 7 kHz during the degraded one. In steady-state, the rest of the shown variables remain constant and at practically equal values for both tests.

The total list of recorded measurements for use in the training of ANN models is as follows: internal heat sink temperature (°C), external heat sink temperature (°C),

control card temperature (°C), air inlet temperature (°C), air outlet temperature (°C), switching frequency (Hz), fan speed reference (ratio), motor power (W), motor voltage (V, RMS), motor current (A, RMS), motor torque (Nm), motor speed (rpm), DC link voltage (V).

To maximize the performance of the condition monitoring algorithm, training data should not be provided as individual data points, as trends and rates of change of the recorded variables and their possible combinations might be relevant to assess the condition of the system. As an example, single sampling points taken during transient periods will likely not provide enough information to make predictions, while rates of change of temperature will probably prove to be relevant indicators. It is important to keep in mind, however, that neural network models are usually not easily interpretable, as the operations they perform are large combinations of nonlinear mappings. Therefore, although speculations can be made on which features the AI scheme could find to be relevant, it will often be impossible to interpret the transformations learnt by neural network models.

To preserve the time-dependant nature of the data while maximizing the number of training samples obtained from experimental tests, the recorded measurements are converted into smaller series of samples. A typical recorded test lasts for around 15 minutes, until the system reaches thermal equilibrium, although each individual test was recorded for somewhat different durations. At a sampling frequency of 1 Hz, this translates to around 900 data points per test. Such a data set would be converted, for example, into a set of 850 series with a length of 50 samples each. On the one hand, series made to contain more samples contain more information, which the AI model would in principle use to make more accurate predictions. On the other hand, increasing this length results in computationally more expensive—and therefore harder to train—models. If the obtained models were to be deployed in an embedded processor, memory requirements would also potentially limit the number of data samples that could be fed into the model at each execution step.

When training artificial neural networks, these sets of time series can either remain in a three-dimensional structure of the form (samples, time steps, features) if they are to be used to train a recurrent neural network, or they can be flattened into a two-dimensional structure of the form (samples, time steps * features) for training fully-connected feedforward networks. Rather than conserving every sample during the flattening process, principal component analysis can be applied to reduce the size of the data while retaining its most relevant features.

4.2 The surrogate model approach

The first considered approach aims to develop a model that mimics the behavior of the healthy cooling system as accurately as possible without being too computationally expensive. The main advantage of this approach is that less data is required for faulty operation, which means that most measurements can be obtained from healthy drives without affecting their performance. Some measurement data obtained from unhealthy drives is still required to adequately tune the evaluation scheme.

By predicting one or several of the system states and comparing these predictions to their corresponding measurements, the health of the system can be evaluated. Ideally, if the measured data comes from healthy operation, predictions should remain close to measurements. However, since the model is trained to represent only healthy operation, large deviations can mean that the measurements reflect degradation or faults in the cooling system.

The system state to be monitored through this surrogate model is selected to be the internal heat sink temperature, since it appears to experience the most variations due to system health condition, as shown in the previous section. Fan speed reference ratio could be another adequate health monitoring variable, but internal heat sink temperature has the added advantage that its readings are already monitored by default in Danfoss FC302 drives, and are accessible without any software modifications.

Once the healthy surrogate model has been obtained, it should be used to make predictions on both healthy and unhealthy data in order to tune the three main parameters of the scheme:

- Prediction horizon: how far into the future the model makes its predictions. If the surrogate model is accurate a large prediction horizon may be desirable. Changes in the states of the system will tend to accumulate over time, meaning that differences between predictions and faulty measurements will be amplified. On the other hand, too large a prediction horizon may make the predictions unreliable, as the conditions of the drive would be more likely to change in the period between every prediction and its corresponding measurement.
- Difference threshold: how far apart a prediction and its corresponding measurement should be for their difference to be considered significant. Increasing this threshold will result in the system being less sensitive to small variations, resulting in fewer false positives (classifying healthy measurements as

faulty) but also increasing the number of false negatives (classifying faulty measurements as healthy).

- Prediction time window: how many individual predictions deviating from their measurements are required to conclude that the health of the system is degraded. If a single deviation was considered sufficient to conclude degradation, outlier measurements resulting from transients such as load changes could be incorrectly classified as faults. Therefore, a more reliable approach is to define a window of a certain number of time periods in which a certain proportion of data points must be considered to be degraded. The time window should be long enough to prevent transients corresponding to nominal operation from being classified as faulty.

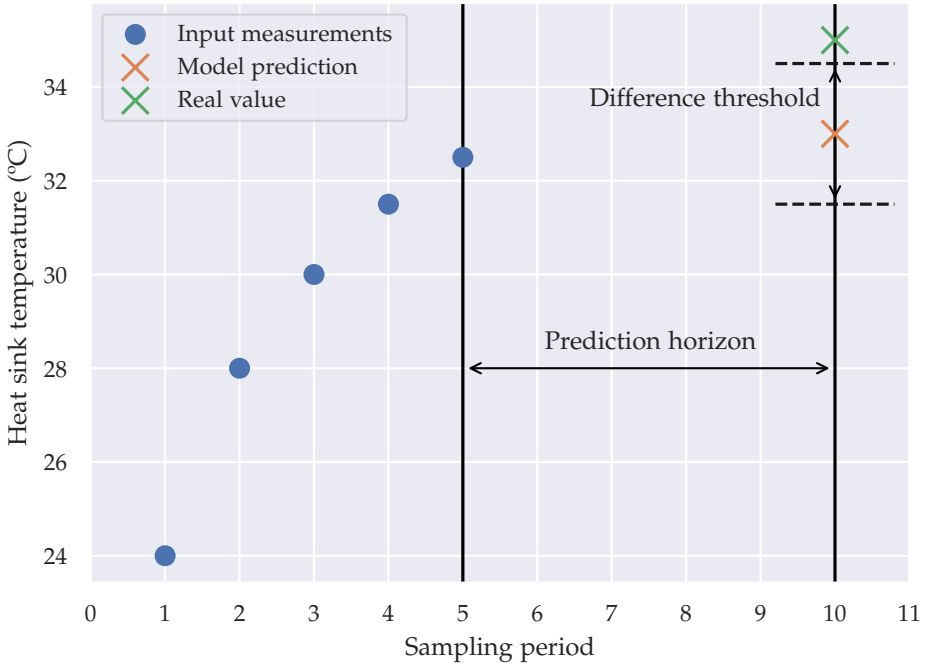

Figure 4.5: Example of the use of a heat sink temperature surrogate model, illustrating some of the parameters to be tuned for its use in condition monitoring.

Figure 4.5 aims to illustrate what these parameters stand for in the condition monitoring scheme, as well as to provide an example showing how the surrogate model-based condition monitoring scheme may work in practice. In this example, the model is called at sampling period 5, where it receives information from the previous five periods. In the figure, only internal heat sink temperature is shown, but in reality the other recorded parameters would also be given as inputs to the model. Using this information, the model generates a prediction for the value of heat sink

temperature at some point in the future. The prediction horizon determines how far ahead into the future this prediction is made for: in the plot, this parameter is set to have a value of 5 periods.

At sampling period 10, the predicted value is compared to its corresponding temperature measurement. The difference threshold is used here to assess whether the difference between these two values can be considered to be significant: if this difference lies over the threshold, the system would conclude that, for this single sample, the system is not behaving as it would be expected if it were in a healthy condition. However, this would not be enough information to conclude that the system is really degraded, as measurement errors and other outliers could result in a healthy system performing unexpectedly at single points. This motivates the use of a prediction time window, determining how many single measurements have to exceed their prediction difference threshold for a conclusion on the global health of the system to be made.

4.2.1 Model design and training

The surrogate model was trained to predict heat sink temperature 3 minutes ahead in time (180 samples ahead) using as inputs the measurements obtained in the last 10 seconds (10 samples). The model is defined as a fully-connected ANN consisting of three middle layers formed by 60, 40, and 20 neurons, respectively, each making use of the ReLU activation function. This configuration appears to result in accurate predictions without leading to overfitting. The output of the model is defined as a single unit with a linear activation. Attempts were made to train recurrent models based on LSTM layers, but the available data appears to not be extensive enough to adequately train these significantly more complex models. Therefore, their performance was worse than that of the simpler fully-connected networks.

Training was performed using the Adam optimizer, with its default learning rate $\epsilon = 0.001$, decay rates $\rho_1 = 0.9$ and $\rho_2 = 0.999$, and numerical stability constant $\delta = 10^{-7}$. The cost function used for training is mean squared error, with mean absolute error also being recorded for further analysis.

The model was trained on data from tests 1 to 15 as shown in Table 4.1, which correspond to healthy operation of the drive cooling system. The data obtained from one or several of these experimental tests was removed from the training data set, and instead used for model testing. The processed data series were separated into two sets, training and validation, with the training set consisting on 70% of the available data and the validation set consisting on the remaining 30%. In total,

around 12000 samples were used for training and validation, depending on which experiment/s were removed from the set for use in testing. Each of these samples consists of a single label point, the heat sink temperature to be predicted, and of the measurements of the available variables in the corresponding 10 period series.

Only the data belonging to the training set is used to adjust the weights and biases of the neural network. Predictions on the validation set are evaluated in each training epoch, to ensure that the model is not overfitting to the training data and therefore unable to generalize to new data points. The most common way to diagnose overfitting is to check that validation and training errors remain close together during training: a spike in validation error with a low training error will usually mean that overfitting is occurring. In this way, however, the samples belonging to the validation set also belong to experimental tests that are represented in the training test. This is where the testing data set finds its use, to guarantee that the model can generalize to experimental tests for which none of its data has been included in the training process.

These design choices were mostly based on trial and error, as there is no completely reliable method to maximize the performance of neural networks. The main criterion for designing the layers of the model is to minimize error on the training and validation data sets while avoiding overfitting: a more complex model would be able to learn more and more complex relationships in the data, but would also be more prone to overfitting. The training process is very reliant on the quantity and quality of the available data—a more complex model requires more data for adjusting its internal parameters, and to avoid overfitting, this data should be adequately spread out over the design space.

The stochastic nature of the training process means that training the same network with the same data for several times will always result in different models; depending on the aggressiveness of the training, possibly with very different results. This makes fine-tuning the structure of the model even more unreliable, as a model may appear to be better fit to a task due to its stochastic training process rather than because of the model itself.

The number of measurement periods to be used as model inputs and the prediction horizon of the model were chosen in a similar way: the selected values appeared to result in adequate predictions without rendering the task trivial, as would happen if the prediction horizon was made too short. In this case, the model would be able to make accurate predictions on both healthy and degraded data, as changes in temperature would be minimal in the considered time frame.

Figure 4.6 shows a performance metric (mean absolute error) over the training

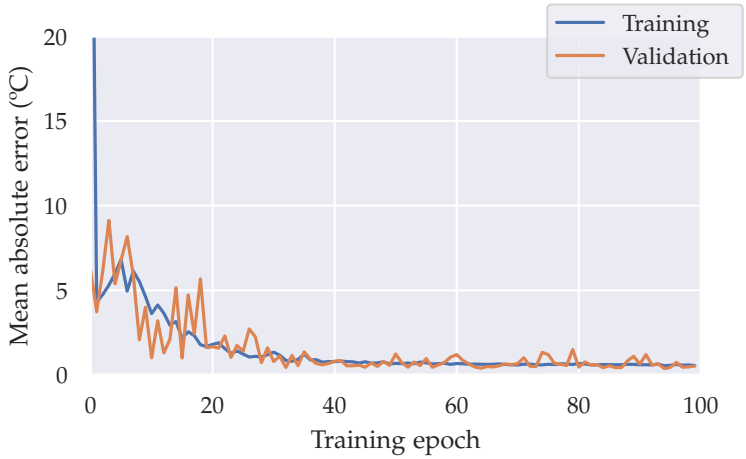

Figure 4.6: Training process for a temperature predictor surrogate model.

process, for both the training and validation data sets. Error tends to converge in about 50 epochs to values below 0.8 °C. The graph also shows how the stochasticity of the training process affects model performance, as oscillations are present in the error attached to both data sets. Overfitting does not appear to occur in this case, as validation error remains close to training error for the whole period.

4.2.2 Results

Figure 4.7 shows the results that can be obtained using the trained surrogate model. The graphs show the measured heat sink temperatures over the course of two experimental tests, as well as their corresponding predictions made by the ANN model. Predictions only start 3 minutes into the recorded data, as that is the selected prediction horizon.

Neither test had any of its data included in the training or validation data set: the model has not been exposed to either one during its training. As expected, the surrogate model is able to make more accurate temperature predictions for the healthy data set than for the degraded one, where it consistently predicts lower temperatures than their corresponding measurements. This can be interpreted as the model making predictions on how it would expect the system to behave if it was in a healthy condition, so it seems logical that a healthy system would be able to achieve lower heat sink temperatures under the same operating conditions.

For the healthy example, the prediction mean absolute error has a value of approximately 0.30 °C, while for the faulty one this value goes up to 1.66 °C. Distinguish-

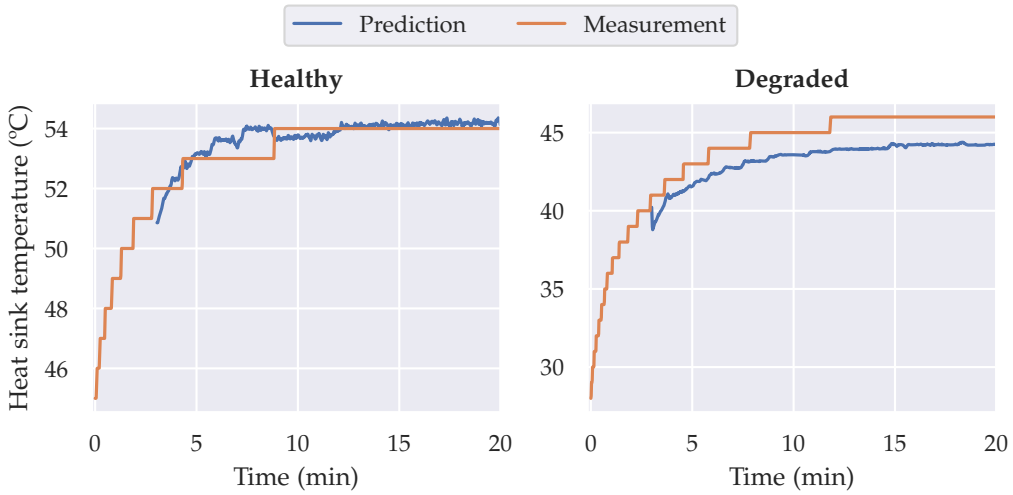

Figure 4.7: Comparison between the predictions made by the surrogate model on a healthy and a degraded test.

ing the health status of the two samples is then, for these two examples, relatively straightforward. A valid option would be to consider a difference threshold of 1 °C of absolute error and a prediction time window of 10 minutes, during which most temperature measurements of the degraded system would exceed the threshold. On more global terms, over the complete data set of degraded tests, mean absolute error has a value of 1.19 °C.

Rather than defining the difference threshold in terms of absolute error, squared error is likely to be a more appropriate metric in this context, as it amplifies absolute deviations of over 1 °C and reduces deviations of less than 1 °C. With this metric, error goes down to 0.16 for the healthy test, and up to 2.03 for the degraded data set, which suggests that squared error facilitates the distinction between operation modes.

Figure 4.8 illustrates the existing trade-off when tuning the difference threshold. The figure graphs the proportion of healthy samples that fall within the bounds defined by the threshold (and are therefore correctly classified as healthy), as well as the proportion of degraded samples that fall outside of these bounds (and are then correctly classified as degraded). For each data set, plots are provided for threshold definitions using both absolute and squared error. An ideal classification scheme would have at least one value of the difference threshold where both the degraded and healthy data sets had a correct classification proportion of 1; which is not the case here. An important caveat of the shown graph is that part of the healthy data has previously been used in training the model, and may therefore

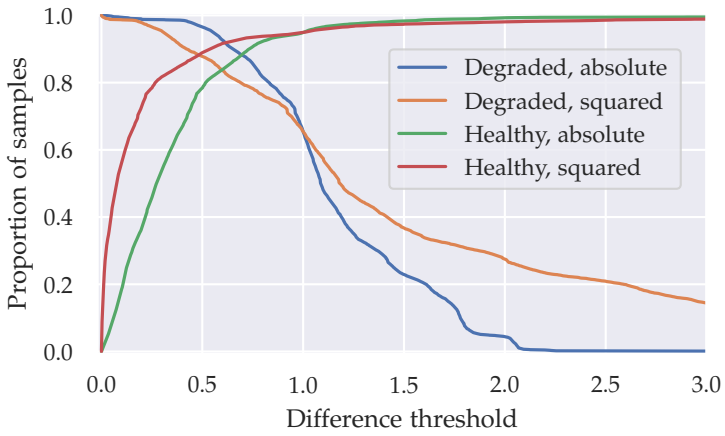

Figure 4.8: Number of samples within the degraded data set exceeding different values of the difference threshold, and samples within the healthy data set that fall underneath this same threshold. Curves are shown for thresholds defined using absolute and squared error.

present a misleadingly high proportion of correctly classified samples.

A larger value of the difference threshold means that more degraded samples are classified correctly, but it also means that more healthy samples are classified incorrectly. The values of difference thresholds that maximize both proportions are located at the intersections of the curves: at around 0.7 for absolute error and 0.5 for squared error. At this point, around 90% of samples are correctly classified. However, this might not always be the goal the scheme: in a practical application, it may be more desirable to minimize the classification of healthy measurements as degraded, in which case a higher threshold can be selected.

4.3 The health indicator approach

The second considered approach is to train a model that is able to predict the values of a previously defined health indicator by making use of available data measurements. As introduced in Chapter 2, the most common approach to develop this health indicator involves processing the available data into synthetic features that can be used to distinguish degraded from healthy performance in the analyzed system. These synthetic features can then be combined into a single health indicator using some distance metric, such as Mahalanobis distance, to increase the reliability of the monitoring scheme.

This approach has proven very successful in monitoring the condition of cooling

fans, typically through the analysis of fan vibration data sampled at a high frequency. Time-frequency analysis of this data using the wavelet transform allows for reliable assessment of the health status of the cooling fan.

In this project, however, the available data is of a quite different nature; it does not include any measurements with periodic oscillations and its sampling frequency is lower. This means that frequency and time-frequency analysis of the data does not result in synthetic features that add meaningful information for health monitoring. Instead, the condition of the system must be assessed by analyzing broader trends and interactions between the available variables. This is the main motivation behind the use of artificial neural networks: the simple parallel structure of these algorithms allows them to efficiently process large amounts of input data, while automatically finding nonlinear combinations of features that can be used to predict the condition of the system.

4.3.1 Model design and training

Figure 4.9 shows the main metrics obtained in the training process of a classifier to predict the binary health status of each available experimental test. Its y-axis represents both prediction accuracy and binary cross-entropy loss. In a similar way to what was done for the surrogate model, measurements from several experiments were excluded from the training and validation data sets. The input data of the neural network was also defined as batches of 10 consecutive measurement samples. The trained model was made to be simpler than that of the temperature predictor: it consists of two hidden layers of 10 and 5 units, and an output layer with a sigmoid activation. Since the task to be performed by the model is binary classification, this activation function is equivalent to the softmax function, which is typically used in multi-class classification.

The available data set is not a balanced one: it contains many more healthy samples than degraded ones. This explains why at the beginning of the training process the accuracy of the model was already over 80%. When training the model using ReLU activation functions in its hidden layers, the model tended to get stuck by finding a simple solution to the problem: it could achieve a relatively high accuracy by always predicting a healthy result. ReLU functions get saturated on their flat side, resulting in null gradients and preventing further training. This problem was solved by substituting the ReLU functions by one of their modifications, leaky ReLU, which maintains a small positive slope (0.1) on the negative side of its horizontal axis. This function prevents gradients from getting stuck at 0 while still presenting many of the desirable properties of the ReLU function. Substituting the ReLU layers by sigmoid or hyperbolic tangent functions also mitigates this

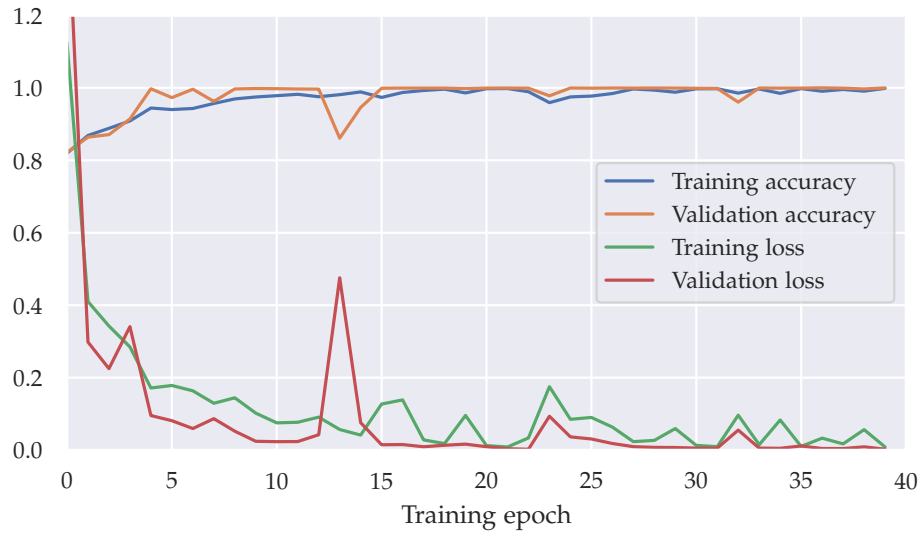

Figure 4.9: Training process for a health status classifier model.

problem, but their vanishing gradients result in slower training.

This problem is partially caused by the imbalance in the labels of the data set. Rather than modifying the model to better fit this data, data augmentation could have been used to improve this balance.

Training was also performed using the Adam optimizer with its default parameters, with categorical cross-entropy as its cost function and recording training and validation accuracy for analysis. These are the parameters shown in Figure 4.9: training and validation accuracy reach values of almost 1 relatively fast, and losses also decay quickly. There also appears to be no overfitting, as loss and accuracy maintain similar values during every epoch.

4.3.2 Results

On the testing data set, however, results paint a different picture: the model often predicts the wrong answer for whole experiments, and it does so with very high confidence. This suggests that the ANN has learnt to find the simplest patterns to make a binary prediction, and to always make predictions with a high confidence (with output values close to either 1 or 0). The model is therefore unable to generalize its predictions to tests that it has not been previously shown.

To understand how this is happening, it may be useful to visualize how a human

could approach this classification task in a simple way. Following Table 4.1 one may observe, for example, that a switching frequency of 6 kHz always means that the cooling system is healthy, although this is of course not true outside of this very reduced context. In this way, each combination of parameters could simply be memorized, rather than making predictions based on observations of temperature profiles according to the other relevant conditions. This is not an exact representation of how neural networks make predictions, but it may serve illustrate what can be amiss here.

A common solution to overfitting, to attempt to force the model to make generalizations, is to reduce the complexity of the model, which prevents the model from "memorizing" the specifics of its training data. In this case, the neural network was reduced down to a simple perceptron—a neural network containing only an input layer and a single output neuron—but the same situation persisted. A support vector machine (SVM) was trained using the same data. Again, however, the classifier model was unable to extrapolate to tests not included in its training data set.

Another approach to force the model to learn generalizations is to remove variables from its input space. This was attempted by removing inputs down to only temperature profiles. However, the obtained results suggest that once the number of input variables is made small enough, the model becomes unable to make reliable predictions and instead finds a local minimum of the cost function by always predicting that the system is healthy. It appears that with the available data, the classifier either "memorizes" parameter combinations that seem to correlate with the health status or is unable to make reliable predictions.

The remaining conclusion is, therefore, that the available data set is not rich enough to be used in this classification scheme. If experimental tests could be extended to include a wider range of operating conditions and different types of faults, the classification method may result in successful results.

Chapter 5

Discussion

This chapter aims to summarize the main issues encountered during the development of this project, discussing the reasoning behind the decisions taken to overcome them. Furthermore, it is intended to provide commentary on the obtained results and findings.

5.1 Modeling

Rather than attempting to develop a mathematical model of the drive and its cooling system from scratch, the modeling process was based on the *Simscape* library included in *Simulink*. This library has the main advantages of covering a wide range of physical system domains and being well documented. This documentation [33] often includes mathematical descriptions of its blocks, as well as their corresponding references, some of which have also been used in Chapter 2 of the present report. These sources have proven very useful in bridging the author's gaps in knowledge regarding thermal systems and fluid dynamics.

The complete drive system is quite complex: it comprises many interconnected elements belonging to different physical domains, which made its modeling quite challenging. Many simplifications were made, especially regarding the control scheme of the drive. Some other assumptions made during modeling were considering load torque and cooling fan dynamics ideal, as well as simplifying the interactions between the heat sink and the cooling air flow. Many variables, such as switching losses, were modeled as comparatively simple look-up tables.

Despite all of these simplifications, the simulation model still turned out to be

significantly complex, and required very long simulation times. On a laptop with a 3.5 GHz Intel i5-7300HQ CPU, simulations require around 1.5 minutes of real time for every second of simulation, even after some computational optimizations. This is due to the fact that every electrical transient must be simulated to a large degree of precision. Electrical and mechanical behaviors could be simplified to simpler large-scale energy-based models to simplify calculations, but this was considered to fall outside of the scope of this project. For the time frames required for the system to reach thermal steady state (around 15 minutes of simulation time), every simulation would require around 25 hours of real time.

For this reason, servers were set up in the AAU computational cloud (CLAAUDIA) to run simulations remotely and on more powerful computers. Using the secure shell (SSH) communication protocol, simulation files could be sent to Linux-based systems with an existing installation of Matlab with both *Simulink* and *Simscape*, where simulations could be run and their results sent back to any computer with access to the server. Although this allowed for simulations to be run remotely, they were not significantly sped up. This is due to the fact that *Simulink* simulations are mostly based on sequential operations. The CPUs contained in the available server machines mostly scale up their power by having an increased number of cores rather than by augmenting their sequential processing speed, making parallelizable computations significantly faster but not showing much improvement on sequential ones. Therefore, simulations still required long times to be completed, and with no guarantee that the simulation model would accurately reflect the performance of the physical drive (as no experimental data was available yet), no more effort was allocated to the simulation models.

5.2 Experimental tests

Initially, the planned time schedule for this project consisted on having the physical drive set up from sent from the Danfoss facilities to Aalborg University. Experimental tests would have been ran in the laboratory facilities of the University, to gather a large amount of data corresponding to healthy operation and to simulated degradation of the cooling system. Afterwards, the bulk of the project would consist on the analysis of this data and the development of corresponding condition monitoring schemes.

However, due to the COVID-19 outbreak, lockdown policies were enforced on Danish universities starting on March 13, 2020, preventing all access to their laboratories. Consequently, the project's focus had to be moved from experimental tests to simulation models. Most of the code used for the analysis of data and the training

of ANN models was developed based on data from simulations.

The physical drive system was then set up at the Danfoss laboratories in Grásten, where tests were performed by Yashar Khadem Sabaz and Norbert Hanigovszki, and their measurement data then made available to the author. However, due to time limitations and scheduling conflicts, partly due to the measures taken by Danfoss to prevent the spread of COVID-19, not all of the experimental tests could be performed as initially planned.

The design of experiments was intended to cover a wide range of operating conditions of the system, both for healthy and degraded operation. Degraded operation was to be simulated by blocking the cooling air inlet at several levels, as well as covering the heat sink with cotton and removing one or several blades from the cooling fan. It would not have been possible to simulate the degradation of thermal interface material between the heat sink and the switching devices in practice, as this component is not easily accessible. Due to the previously mentioned limitations, degradation tests could only be ran with a single type of fault, a 50% blocked air passage, with relatively few setpoints. Most of the planned experiments for healthy operation could be performed without any major issues.

Test setpoints were chosen according to the component specifications of the system. Other decisions, such as the chosen sampling frequency of 1 Hz, were made according to hardware availability and sensor resolution. Each test was ran until the system reached thermal equilibrium, usually at between 15 and 20 minutes from the beginning of the test.

Initially, it was intended to make use of the procedures detailed in Chapter 2 to develop a health indicator to be attached to each test. For this purpose, experimental data was imported in *Matlab*, for analysis using its Diagnostic Feature Designer application. However, it became clear that standard procedures for the development of a health indicator do not fit well to obtained data: they are typically based on statistical and frequency analysis of data. These methods are appropriate for measurements such as fan vibration data, sampled at high frequency, where Fourier and wavelet transforms can be used to extract high frequency subsignals with clear links to faulty operation. With the larger timeframes considered here, this approach to develop a health indicator did not prove fruitful. Therefore, rather than a continuous health indicator, tests were given a binary label representing whether they were recorded in healthy or in degraded operating conditions.

5.3 Neural networks

The artificial neural network models presented in this project were designed and trained using the Python 3 implementation of *TensorFlow*, making use of the *Keras* API. This environment was chosen over the neural network resources included in Matlab due to the libraries being open-source and more widely used in the machine learning community, as well as having high-quality documentation readily available. Python is also a more flexible environment, with many useful libraries for data analysis, processing, and visualization.

The servers previously set up for running simulations could have been useful for speeding up the training of neural network models, as computations performed for this process are highly parallelizable. However, with the relatively small size of the available data sets (usually under 15000 samples), each training epoch only required under 5s when ran locally using GPU acceleration on an NVIDIA GTX 1050 GPU. Training an ANN model for 100 epochs therefore only required a little over 5 minutes of computational time. Running the training on the remote servers would thus not result in very significant computational time improvements, but could prove to be useful if the training data set were to increase in size.

The artificial neural networks presented for use as temperature predictors are shown to perform as intended; however, the ones trained to make predictions on the value of the binary health indicator do not achieve acceptable levels of performance. This illustrates some of the strengths and weaknesses of artificial neural networks: they require large amounts of diverse enough data to be able to make generalizations on inputs outside of their training set, otherwise, they are prone to finding simple patterns that only fit the training set. For this reason, it is fundamental to maximize both the quantity and the quality of the available data, in order to train better models.

Another point to consider is that artificial neural networks are not necessarily the best fit for the problems at hand: simpler statistical models such as autoregressive models or AI methods like support vector machines or decision trees could yield similar results and be less computationally demanding. The most appropriate solution for each task will be closely tied to the available data and the desired goals of the model, which makes it hard to extract conclusions on the global performance of different solutions.

Chapter 6

Conclusion

This chapter documents the main conclusions drawn from the development and results of this project. As stated in Chapter 1, the main goal of this project was to study the feasibility of using artificial intelligence to monitor the health condition of the cooling system of a variable speed drive.

This report documents the steps taken to obtain such AI schemes, and aims to provide the required theoretical background to understand the principles behind them.

A simulation model of the drive and its cooling system was developed following its mathematical description as shown in Chapter 2. Although the model has not been completely tuned to the specifics of the physical drive used in experimental tests, it can be expected to perform in a similar manner to the real system. It therefore allows for observation of parameters that are not recorded in experimental tests, such as electrical transients, and also for the modification of elements outside of the bounds placed by the physical system. For example, degradation in heat sink thermal interface material, which would be hard to reproduce experimentally, can be represented in simulation by simply increasing the thermal resistance of this particular element.

The available data does not justify the use of more complex neural network structures such as LSTM layers, as their large number of parameters makes their training challenging with the relatively low number of samples that can be used for training. The trained models were therefore reduced to simpler fully-connected feedforward neural networks, which, despite their relative simplicity, can be made to converge to lower error values, as they can be fully trained using the available data.

The models appear to perform well when trained as temperature predictors. The surrogate model scheme is able to separate samples for the two operation modes in a quite reliable way. However, it is not guaranteed that its promising performance would be carried over to predictions on new experimental data, especially considering the poor performance shown by the health indicator classifier models. With a richer data set, more conclusive observations could be drawn.

Still, the obtained results show that the proposed approaches to condition monitoring using AI-based models, or at the very least the surrogate model approach, may be developed successfully. By making use of more extensive data for training and testing, it appears likely that the proposed models would be suitable for monitoring the condition of the cooling system of drives in practical applications.

Chapter 7

Future work

This chapter aims to catalog some possible modifications, improvements, and extensions of the presented work that, in the author's opinion, could prove fruitful if investigated. The scope of this project can be broadened and its development process has room for improvement. Some suggestions for future work based on the present project are listed as follows:

- The performance of models based on machine learning is almost completely reliant on the quantity and quality of the data that is made available for their training. The most obvious improvement for any machine learning project is therefore to obtain more data covering as much ground within the design space as possible. Improving the available data would result in more reliable and accurate predictions after the training process. Obtaining data from more diverse setpoints—with the test setup running under more varied conditions—would serve to ensure that the model is able to generalize to monitor the drive system under conditions not directly contained in the training data. With a larger data size, more complex neural networks, such as recurrent neural networks, may be successfully trained to improve the capabilities of the develop models.
- By making use of the REST API, data can be streamed directly from the target system to any computer with access to the internet, allowing for real-time monitoring of the test setup. Based on this, models could be made to run in real-time, continuously gathering data and assessing the condition of the system. This would represent more accurately the way that an AI-based condition monitoring scheme could be deployed in practice for drives connected to the IoT. Online training of the models could also become a

possibility, assuming that faults in the system could be observed, by manually labeling the streamed data and using it to periodically update the training status of the AI algorithms.

- Extending the ideas presented in the previous point, models prepared to be able to run in real-time could be deployed on a microcontroller and installed in the drive system, to locally monitor the condition of the system. The *TensorFlow* library for Python directly supports the deployment of pre-trained neural network models on microcontrollers, through its sub-library *TensorFlow Lite for Microcontrollers*. This library is implemented in C++ and can be used to convert existing ANN models to C or C++ code with lower requirements of memory and processing speed. This library is still under development, and therefore does not yet officially support operations such as recurrent layers or the training of models. For more powerful devices—for example, embedded Linux devices like the Raspberry Pi—the standard *TensorFlow Lite* library might prove easier to integrate and offer broader functionality.

- The developed neural network models may be streamlined by, rather than feeding the model every variable as a time-dependent input, selecting the input method of each feature depending on whether or not it remains constant in steady-state operation. More specifically, variables that are more prone to change over the considered time periods (such as heat sink temperatures) could be fed to a recurrent layer and then combined with time-independent features in a fully-connected or convolutional layer.

- This project has not drawn any conclusions on whether this particular application is significantly improved by relatively complex machine learning models such as artificial neural networks. Simpler statistical models such as auto regressive moving average (ARMA) or auto regressive integrated moving average (ARIMA) combined with classification methods such as decision trees could result in similar performance and be computationally more efficient. Further investigation could be based on developing different models and comparing their performance. The type of algorithm that performs best in a given environment is highly dependent on the nature of its available data, so conclusions on which algorithm is best for a broad type of tasks may not be reliable. For example, an LSTM-based ANN could perform well in a classification task with dense time data—with a high sampling frequency—while a support vector machine could be a better fit for the same task with a lower frequency.

- Before deployment in industrial applications, the performance of the developed AI-based condition monitoring schemes should be compared against

the more traditional approach to monitor the health of cooling fans, based on vibration data analysis. Vibration data would be expected to result in a more accurate assessment of the health status of the system, but also have a higher cost of implementation, as well as raising the additional reliability concerns attached to adding a vibration sensor.

- By making use of recorded data from laboratory experiments, the simulation models introduced in Chapter 2 can be tuned to more accurately reflect the physical system. If the simulation model can be made to closely mirror the system's performance, the model can be used to obtain data for training the condition monitoring models. Using a simulation model for data collection can be used to obtain data under extreme conditions that would be costly to reproduce in an experimental setup. Moreover, obtaining data from simulation models will almost always be less time-consuming than running the same tests in a laboratory.

- Another approach to the development a condition monitoring scheme would be to automate the obtention of synthetic features by making use of a type of ANN known as autoencoder. Such a model is trained to output the same data that is given to it as input, but inside it data is compressed by making use of one or several layers with a lower number of units than the input layer. In this way, input data is mapped to a lower-dimensional structure that allows for optimal decompression. This lower-dimensional mapping can then be extracted and used for obtaining synthetic features, which will retain information of time-dependence and feature interactions. These features could then be used to develop health indicators for any condition monitoring algorithm.

Bibliography

[1] Patrick D.T. O'Connor and Andre Kleyner. *Practical Reliability Engineering: Fifth Edition*. Chichester, UK: John Wiley and Sons, Nov. 2011. ISBN: 9780470979822. DOI: 10.1002/9781119961260. URL: http://doi.wiley.com/10.1002/9781119961260.

[2] Shaoyong Yang et al. "An industry-based survey of reliability in power electronic converters". In: *IEEE Transactions on Industry Applications* 47.3 (May 2011), pp. 1441–1451. ISSN: 00939994. DOI: 10.1109/TIA.2011.2124436.

[3] Jun Zhang et al. "Thermal Parameter Monitoring of IGBT Module Using Case Temperature". In: *IEEE Transactions on Power Electronics* 34.8 (Aug. 2019), pp. 7942–7956. ISSN: 08858993. DOI: 10.1109/TPEL.2018.2879825.

[4] Wilson Wang and Derek Dezhi Li. "Health Condition Monitoring of Induction Motors". In: *Induction Motors - Applications, Control and Fault Diagnostics*. InTech, Nov. 2015. DOI: 10.5772/61110.

[5] Jing Tian et al. "Motor Bearing Fault Detection Using Spectral Kurtosis-Based Feature Extraction Coupled with K-Nearest Neighbor Distance Analysis". In: *IEEE Transactions on Industrial Electronics* 63.3 (Mar. 2016), pp. 1793–1803. ISSN: 02780046. DOI: 10.1109/TIE.2015.2509913.

[6] Xiaohang Jin et al. "Health monitoring of cooling fans based on mahalanobis distance with mRMR feature selection". In: *IEEE Transactions on Instrumentation and Measurement* 61.8 (2012), pp. 2222–2229. ISSN: 00189456. DOI: 10.1109/TIM.2012.2187240.

[7] Jie Liu, Wilson Wang, and Farid Golnaraghi. "An extended wavelet spectrum for bearing fault diagnostics". In: *IEEE Transactions on Instrumentation and Measurement* 57.12 (2008), pp. 2801–2812. ISSN: 00189456. DOI: 10.1109/TIM.2008.927211.

[8] Jun Zhang et al. "Condition monitoring the health status of forced air cooling system using the natural frequency of thermal network". In: *IEEE Transac-*

tions on Power Electronics 34.11 (Nov. 2019), pp. 10408–10413. ISSN: 19410107. DOI: 10.1109/TPEL.2019.2923801.

[9] Xiaohang Jin and Tommy W.S. Chow. "Anomaly detection of cooling fan and fault classification of induction motor using Mahalanobis-Taguchi system". In: *Expert Systems with Applications* 40.15 (2013), pp. 5787–5795. ISSN: 09574174. DOI: 10.1016/j.eswa.2013.04.024. URL: http://dx.doi.org/10.1016/j.eswa.2013.04.024.

[10] Wenjun Sun et al. "A sparse auto-encoder-based deep neural network approach for induction motor faults classification". In: *Measurement: Journal of the International Measurement Confederation* 89 (July 2016), pp. 171–178. ISSN: 02632241. DOI: 10.1016/j.measurement.2016.04.007.

[11] Mohendra Roy et al. "A Stacked Autoencoder Neural Network based Automated Feature Extraction Method for Anomaly detection in On-line Condition Monitoring". In: *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*. Institute of Electrical and Electronics Engineers Inc., Jan. 2019, pp. 1501–1507. ISBN: 9781538692769. DOI: 10.1109/SSCI.2018.8628810. arXiv: 1810.08609.

[12] Shuangshuang Jin et al. "The Remaining Life Prediction of the Fan Bearing Based on Genetic Algorithm and Multi-Parameter Support Vector Machine". In: *2017 5th International Conference on Mechanical, Automotive and Materials Engineering, CMAME 2017*. Institute of Electrical and Electronics Engineers Inc., Nov. 2018, pp. 143–148. ISBN: 9781538604311. DOI: 10.1109/CMAME.2017.8540112.

[13] Rangarajan M. Tallam et al. "Neural network based on-line stator winding turn fault detection for induction motors". In: *Conference Record - IAS Annual Meeting (IEEE Industry Applications Society)* 1 (2000), pp. 375–380. ISSN: 01972618. DOI: 10.1109/IAS.2000.881138.

[14] Lei Xie et al. "Fan bearing fault diagnosis based on continuous wavelet transform and autocorrelation". In: *Proceedings of IEEE 2012 Prognostics and System Health Management Conference, PHM-2012*. 2012. DOI: 10.1109/PHM.2012.6228837.

[15] Suchitra Venkatesan et al. "Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin". In: *IET Electric Power Applications* 13.9 (Sept. 2019), pp. 1328–1335. ISSN: 17518679. DOI: 10.1049/iet-epa.2018.5732.

[16] Hua Su and Kil To Chong. "Induction machine condition monitoring using neural network modeling". In: *IEEE Transactions on Industrial Electronics* 54.1 (Feb. 2007), pp. 241–249. ISSN: 02780046. DOI: 10.1109/TIE.2006.888786.

[17] Yunus a. Cengel. *Heat and Mass Transfer: A Practical Approach*. 2006.

[18] *Predictive Maintenance Toolbox Documentation - MathWorks Nordic*. URL: https://se.mathworks.com/help/predmaint/index.html?s%7B%5C_%7Dtid=CRUX%7B%5C_%7Dlftnav (visited on 05/19/2020).

[19] Prasanta Chandra Mahalanobis. "On the generalised distance in statistics". In: *Proceedings of the National Institute of Sciences of India* 2.1 (1936), pp. 49–55. URL: http://insa.nic.in/writereaddata/UpLoadedFiles/PINSA/Vol02%7B%5C_%7D1936%7B%5C_%7D1%7B%5C_%7DArt05.pdf.

[20] Oludare Isaac Abiodun et al. *State-of-the-art in artificial neural network applications: A survey*. Nov. 2018. DOI: 10.1016/j.heliyon.2018.e00938.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90020-8.

[22] Jürgen Schmidhuber. *Deep Learning in neural networks: An overview*. 2015. DOI: 10.1016/j.neunet.2014.09.003. arXiv: 1404.7828.

[23] Moshe Leshno et al. *Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function*. Tech. rep. 1993, pp. 861–867.

[24] Chigozie Enyinna Nwankpa et al. *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*. Tech. rep. 2018. arXiv: 1811.03378v1.

[25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0.

[26] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. Vol. 521. 7553. 2017, p. 785. ISBN: 3540620583, 9783540620587. DOI: 10.1016/B978-0-12-391420-0.09987-X. arXiv: arXiv:1011.1669v3.

[27] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2015. arXiv: 1412.6980.

[28] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. Tech. rep. 8. 1997, pp. 1735–1780. URL: http://www7.informatik.tu-muenchen.de/%7B~%7Dhochreithttp://www.idsia.ch/%7B~%7Djuergen.

[29] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. Association for Computational Linguistics (ACL), June 2014, pp. 1724–1734. ISBN: 9781937284961. DOI: 10.3115/v1/d14-1179. arXiv: 1406.1078.

[30] *Requests: HTTP for Humans™ — Requests 2.23.0 documentation*. URL: https://requests.readthedocs.io/en/master/ (visited on 05/23/2020).

[31] *json — JSON encoder and decoder — Python 3.8.3 documentation*. URL: https://docs.python.org/3/library/json.html (visited on 05/23/2020).

[32] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference* 1697900.Scipy (2010), pp. 51–56. ISSN: 0440877763224. DOI: 10.25080/Majora-92bf1922-00a. URL: https://conference.scipy.org/proceedings/scipy2010/mckinney.html.

[33] *Physical Modeling - MATLAB & Simulink - MathWorks Nordic*. URL: https://se.mathworks.com/help/physmod/simscape/physical-modeling.html (visited on 05/25/2020).