An artificial neural network model for predicting deep-level mine refrigeration plant performance

EW Pretorius

orcid.org/ 0000-0002-3028-6797

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Mechanical Engineering* at the North-West University

Supervisor: Dr JH Marais

Co-supervisor: Dr JH van Laar

Graduation: June 2025

ACKNOWLEDGEMENTS

Many individuals played a part in the completion of this study. Their contributions and support are much appreciated, and I would like to acknowledge and express my gratitude.

- First and foremost, to the Heavenly Father, who gave me the ability and opportunity to complete this study.
- My supervisor, Dr Johan Marais and co-supervisor, Dr Jean van Laar, for their insights and guidance during the course of this study.
- ETA Operations (Pty) Ltd, for their financial-, time-, and technical resources without which this study could not have been completed.
- My parents, David and Wilma, for their continuous support, interest, and encouragement throughout the duration of this study. My brothers, Janlu and Schalk, for their technical insights and support.
- My colleagues, Julian Pretorius, Sean Louw, and Zerwick Victor for their feedback and insights throughout the duration of this study.

The information given in this dissertation was presented with acknowledgement of the source and reference to published works.

ABSTRACT

Title: An artificial neural network model for predicting the performance of a deep-level mine refrigeration plant

Author: Evan Pretorius

Supervisor: Dr Johan Marais

Co-supervisor: Dr Jean van Laar

Keywords: Artificial neural network, modelling, prediction, deep-level mine, refrigeration, artificial intelligence, optimisation

Deep-level mining is common in South Africa due to the depletion of shallow resource deposits. To maintain worker productivity in hot underground conditions, mining at depth demands the use of cooling systems that are reliant on the production of chilled water at centralised refrigeration plants. These plants are large consumers of energy and, coupled with persistent and acute increases in electricity prices, place a heavy financial burden on the mining industry.

Improved performance of the refrigeration plant leads to improved energy efficiency. To achieve this this, the creation of a model to establish the relationship between the operational parameters of a refrigeration system and its performance is needed. Artificial neural networks (ANNs) can establish such relationships. The multi-layer perceptron (MLP) is a type of ANN that exhibits superior performance when working with noisy data typical of energy systems and is used to avoid the difficulty of using traditional methods on an ageing system where design specifications are outdated. Despite this, an MLP has not yet been used to model the performance of a deep-level mine refrigeration plant. An MLP was therefore chosen to develop the model.

The aim of this study is to develop a new method that uses multi-layer perceptron theory to create a model of a vapour-compression refrigeration plant on a South African deep-level mine that accounts for changing operational conditions. The input parameters for the model were chosen based on successful implementations of ANNs in studies on refrigeration systems and the availability of sensor data at the case study plant. The raw data was collected, filtered, and randomly distributed into four independent datasets for training, validation, testing, and verification purposes, respectively.

Models with varying architectures and training algorithms were trained where it was found that the best-performing network consisted of two hidden layers containing 37 hidden neurons in its first hidden layer and 21 hidden neurons in its second hidden layer. It was found that the Levenberg-Marquardt algorithm outperformed the scaled conjugate gradient (SCG) algorithm in convergence speed and accuracy. Coefficients of determination (R^2) for the training (0.9999), validation (0.9998), and test (0.9998) subsets show that the model possesses good generalisation capabilities and can accurately predict the COP of the case study refrigeration plant.

The relationship between the input parameters and COP were extracted from the ANN using the fundamental input-output equations of the ANN. Evaluation of these relationships showed that higher relative water flow rates to the evaporator and condenser led to an average increase in COP of 0.52. The improved system performance resulted in a reduction in compressor power of 130 kW, which amounts to R1.8 million in annual savings.

To verify the model, it was implemented on the test subset to evaluate its prediction accuracy on an independent and unseen data set. The model achieved an R^2 of 0.9983, RMSE of 0.0098, and a MAPE of 2.55% which reaffirms is robustness and accuracy in predicting the performance of a deep-level mine refrigeration plant.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Note that bold face usage of a symbol denotes its vector or matrix form.

Symbol	Description
m	Meter
R	South African Rand
°C	Degrees Celsius
$\dot{Q}_L$	Rate of heat removal
$\dot{W}_{net,in}$	Rate of electrical work input
x	Model input
w	Weight value between two neurons
b	Bias to a neuron
y	Model output/target output
u	Net input value of a neuron
e	Error value at an output neuron
$\hat{y}$	Model predicted output
ε	Instantaneous error energy (for online learning)
ε_{av}	Average error over an entire training subset (for batch learning)
λ	Scaling factor/learning rate
$\boldsymbol{g}$	Gradient vector
$\boldsymbol{J}$	Jacobian matrix
$\boldsymbol{r}$	Residual/vector of errors
$\boldsymbol{H}$	Hessian matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{d}$	Search direction for the scaled conjugate gradient algorithm
α	Step size in the search direction of the conjugate gradient algorithm
W	Number of weights in the weight vector
β	Step size direction of the scaled conjugate gradient algorithm
$\|\|d\|\|$	Norm of the search direction
r_{in}	Number of weights going into a neuron
r_{out}	Number of weights going out of a neuron
P_C	Compressor power
y_F	Filtered value (exponentially weighted moving average filter)
y_m	Measured value (exponentially weighted moving average filter)
Δy	Maximum allowable change (exponentially weighted moving average)
τ	Time constant used to adjust smoothing in the EWMA
Δt	Time difference between two samples
$\hat{x}$	Normalised parameter

LIST OF ABBREVIATIONS

Abbreviation	Expansion
R134	Refrigerant 134a
R404	Refrigerant 404
R407	Refrigerant 407
R717	Refrigerant 717
ACU	Artificial cooling unit
AI	Artificial intelligence
ANFIS	Adaptive neuro-fuzzy inference system
ANN	Artificial neural network
BAC	Bulk air cooler
COP	Coefficient of performance
EWMA	Exponentially weighted moving average
GDP	Gross domestic product
HL	Hidden layer
HN	Hidden neuron
HVAC	Heating, ventilation, and air conditioning
IEEE	Institute of electrical and electronics engineers
LM	Levenberg-Marquardt
LPG	Liquid petroleum gas
MAE	Mean absolute error
MAPE	Mean absolute percentage error
MCU	Mobile cooling units
MLP	Multi-layer perceptron
MRE	Mean relative error
MSE	Mean squared error
MSLE	Mean square logarithmic error
TiO_2	Titanium dioxide
PLC	Programmable logic controller
PCT	Pre-cooling tower
R	Correlation coefficient
R^2	Coefficient of determination
RBFN	Radial basis function network
ReLu	Rectified linear unit
RMSE	Root mean square error
RNN	Recurrent neural network
SCG	Scaled conjugate gradient
TCI	Training, calibration, and implementation

CHAPTER 1 INTRODUCTION

1.1 Preamble

This chapter introduces concepts relevant to the study conducted for this paper. Background information is given to provide context for the problem this study aims to address. Refrigeration optimisation is discussed with particular emphasis on the models used to facilitate the optimisation. Artificial neural networks (ANNs) are then discussed, followed by a review of the literature surrounding the use of ANNs in the modelling of refrigeration systems. The information gathered from the literature review is presented in the state-of-the-art section. The objectives of this study are then conveyed, and the chapter concludes with an outline of the document.

1.2 Background

Since the start of the industrial revolution, the world has experienced unprecedented economic growth and technological development [1]. Improvements in energy availability has played a key role in facilitating the transition from the pre-industrial world to the world as it is today. For modern economies to sustain this level of growth, their expanding energy needs will continually have to be met. This has raised questions about sustainability and the environmental impact of increasing energy output [1].

1.2.1 Electricity in South Africa

Eskom is South Africa's primary supplier of electricity. The state-owned utility is responsible for generating and transmitting 95% of electricity consumed in the country [2]. Consequently, the country's industries are heavily dependent on Eskom's ability to provide electricity cheaply and effectively. Since the electricity crisis of 2008, however, Eskom has struggled to generate enough electricity to meet its customers' demand [3].

The power demand from South African consumers managed to outstrip the generating capabilities of the utility for the first time in 2008, resulting in rolling blackouts [3]. In response, Eskom rushed to build more coal-fired power stations to increase supply [4]. These projects faced much difficulty due to poor planning, mismanagement, and contracting irregularities, which plunged the utility into debt [4].

To finance this debt, income had to be increased through drastic hikes in electricity tariffs [4]. Figure 1 illustrates the increases by comparing average electricity prices and inflation from 1988

to 2024 with a projection into 2026. Notably, from 2007 to 2024, electricity prices rose by 937% compared to inflation of 155% over the same period [5].

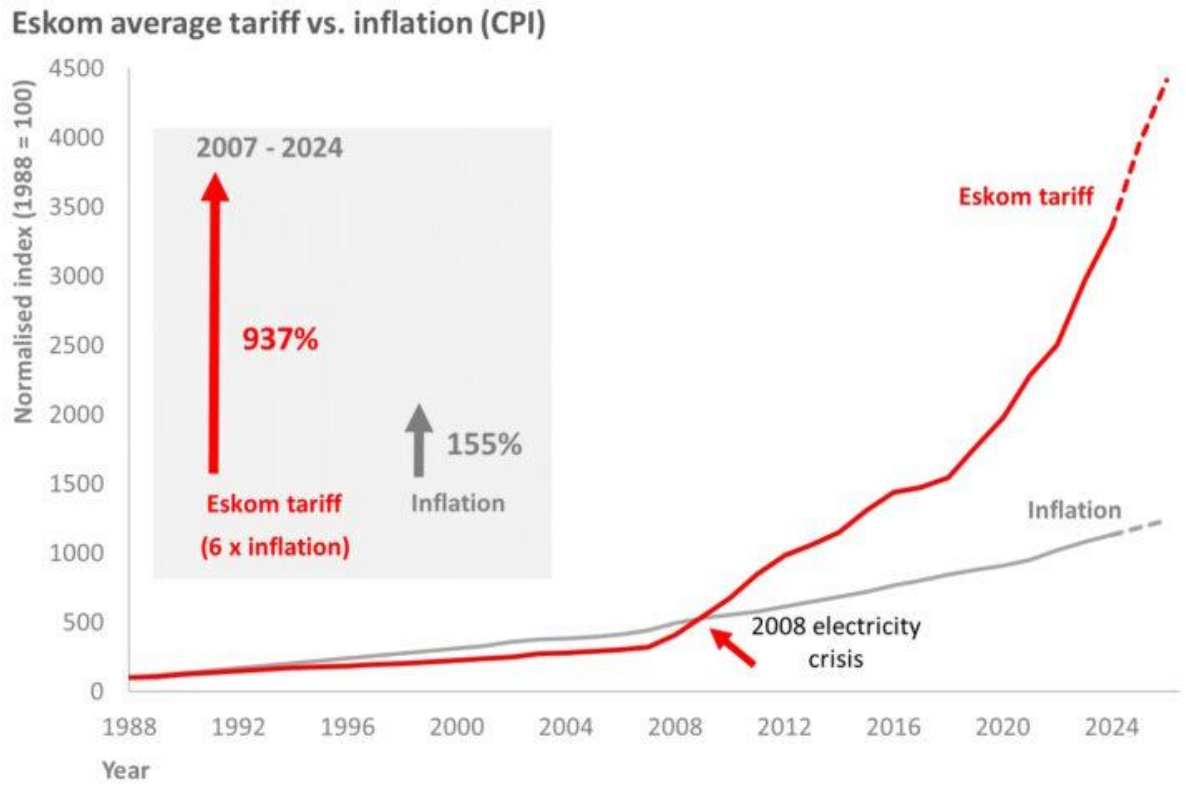


Figure 1: Historical Eskom tariffs between 1988 and 2024 compared to inflation over the same period [5]

This dramatic increase in electricity prices has an acute effect on energy-intensive industries [6]. Increased spending on electricity leads to higher operational costs which results in lower profit margins. This, in turn, negatively impacts their ability to expand and to remain competitive in the global market [6].

This pattern persists on a macro scale where it is has been observed that deficiencies in energy systems slow economic growth [7]. It has been estimated that weak power infrastructure has held back gross domestic product (GDP) growth of sub-Saharan Africa by as much as 2% [8].

1.2.2 South African deep-level mines as energy users

South Africa's mining sector is no longer the sector that contributed 21% of GDP to the country's economy as it did in 1980 [9]. Nevertheless, recent figures show that mining was still directly

contributing 4.3% of GDP in 2023 [10]. The sector also remains important as a job creator where it currently employs 4.8% of the country's work force or 477 000 workers [11]. Despite its contributions to GDP and the labour force, mining demands a disproportionally high fraction of the energy supplied to the industrial sector of the economy. This is highlighted in Figure 2, where it is shown that mining and quarrying consumes 10% of the energy supplied to industry [12].

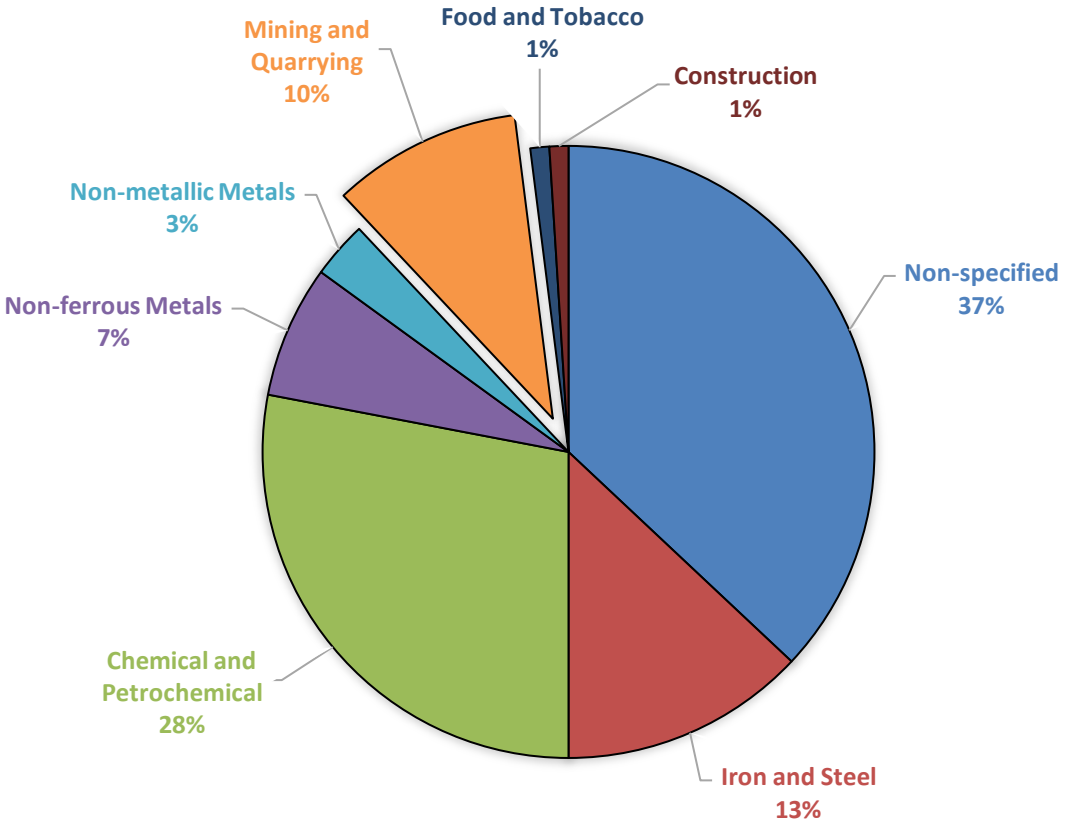


Figure 2: Energy demand of industrial sub-sectors as a proportion of total energy supplied to South African industries in 2018 [12]

South Africa is rich in natural resources. The exploitation of these resources transformed the country's economy and catalysed its transition into modernity [9]. Although the country still has plentiful reserves of minerals and precious metals, shallow deposits have largely been depleted [13]. Metal mines in the country, for example, has an average depth of 2 000m, which presents significant challenges to their profitability because of the associated cost of extracting ore from these depths [14].

Expenditure on labour and electricity costs are the main drivers of the inflated operating costs of deep-level mines [13]. Electricity specifically, often accounts for up to 20% of a mine's total

operational costs [15]. To effectively mine at depth, mines rely on several energy systems that demand a continuous supply of large amounts of power. These systems are electric motor-based systems and are typically grouped into the following categories [16]:

Compressed air

Compressed air is used in conventional mining to power tools such as drills used for mining and to provide fresh air to underground safe areas called refuge bays. These users receive pressurised air that is generated by large compressors and is transported through an underground piping network [16].

Hoisting

Hoisting encompasses all the energy users responsible for the transport of people, equipment, and ore in and out of a mine shaft [17]. In deep-level mines, this typically refers to winders powered by electrical motors on the surface. In especially deep mines, it is common for an additional shaft to be sunk underground. In this case, additional underground winders will be used for hoisting in the sub-shaft [17].

Dewatering

Dewatering pumps are necessary to remove water from the underground environment. They are large consumers of energy and operate 24 hours of the day to prevent the mine from flooding. The energy used for dewatering is dependent on the depth of the mine and the amount of water that must be pumped out to maintain a water balance [18].

Ventilation

Ventilation fans induce air flow through the underground environment of a mine [19]. As air flows through the mine, it absorbs heat from the surrounding areas and ultimately rejects it to the atmosphere. This process is referred to as the natural cooling power of air and is considered the primary method of cooling a mine. The system consists of large centrifugal surface extraction fans and, in some cases, underground axial booster fans. These fans provide a constant supply of oxygen and cooling to underground areas [19].

Refrigeration

Artificial cooling becomes necessary when mining activities exceed depths of 1 400m [20]. It usually comes in the form of chilled water produced at a refrigeration plant. The chilled water is

then used in water-to-air heat exchangers such as bulk air coolers (BACs) or localised mobile cooling units (MCUs) [20]. This mechanical cooling process is the secondary method of cooling the underground environment of a mine [19].

All mines have unique engineering challenges that rely on a combination of the abovementioned energy systems to be solved. This means that no two mines will have a similar distribution of power to these systems. A commonality between deep-level mines, however, is that the cooling systems (refrigeration and ventilation) account for a large proportion of the energy consumed [18]. An example of how a Free State province deep-level gold mine's electricity bill is split between energy systems is given in Figure 3.

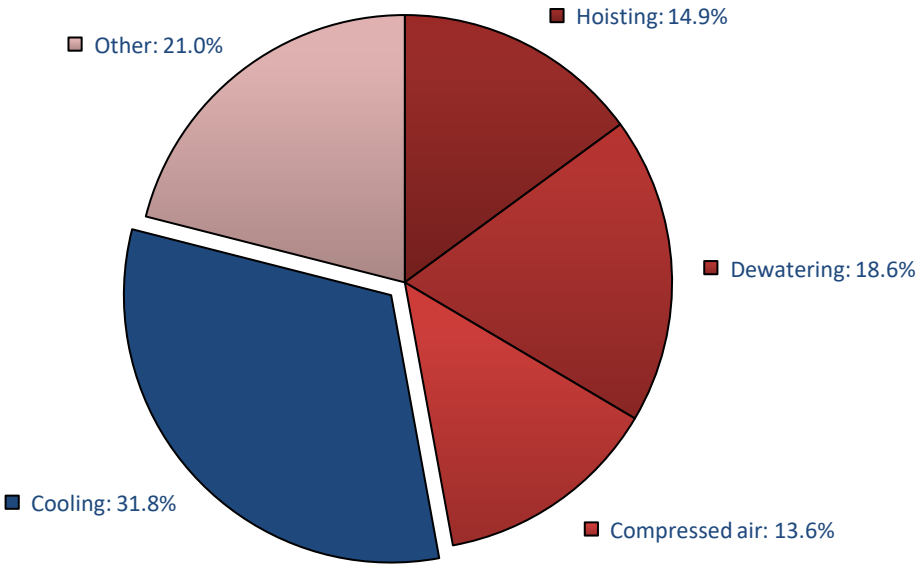


Figure 3: Electricity cost distribution between energy systems at a South African deep-level gold mine in the Free State province

Notably, cooling is responsible for nearly a third of the mine's electricity bill. Ventilation and refrigeration are typically seen as an integrated system because the main purpose of the production of chilled water is to cool the air that is distributed by fans [18].

1.2.3 Cooling systems on deep-level mines

Deep-level mining is inherently challenging for a variety of reasons. Chief among them is the high geothermal gradient which can cause strata temperatures, better known as virgin rock temperatures, to rise beyond 50 °C [21]. This can threaten resource production since a loss of worker productivity has been observed when wet-bulb temperature exceeds 28 °C [22]. To offset the heat loads associated with depth, mine cooling systems are used.

Mine cooling systems can have many different configurations depending on the age, depth, and type of mine [23]. The chosen configuration, however, typically consists of a combination of a group of applications, which are listed in Table 1 [23], [24].

Table 1: Typical cooling applications on a deep-level mine [23], [24]

Cooling application	Description
Ventilation system	The use of surface extraction fans to induce air flow through a mine.
Conventional surface BAC	Air is drawn through a cooling tower of the BAC where chilled water is sprayed over the air, acting as an evaporator. These systems utilise an intake shaft where mine personnel are transported which limits their cooling capacities.
Ultra-cold surface BAC	Ultra-cold surface BACs are not limited in its cooling capacity like its conventional counterpart since it uses a dedicated air intake shaft. This type of BAC consequently possesses superior cooling capabilities.
Cold water from surface	Surface refrigeration plants produce chilled water which is sent to surface cooling applications and underground via insulated pipes to be used in various underground cooling applications.
Underground air-cooling systems	In some cases, BACs can be placed deeper into a mine or mining level to recirculate and re-cool air in a more localised region. For even more localised cooling, mobile cooling units (MCUs) are used. These variants are called secondary and tertiary underground cooling based on the size of the area cooled and the amount of cooling provided.
Underground refrigeration plant	In very deep mines, underground refrigeration is more efficient because of its proximity to the area to be cooled. However, these plants have suppressed heat rejection capabilities in underground environments, resulting in a lower coefficient of performance (COP).

Ice from surface	This form of cooling is reserved for exceptional cases in deep mines as a last resort. Ice is produced on the surface and experiences little to no temperature change when sent underground since the potential energy it picks up is converted into latent energy and spent on melting the ice.

The choice of which cooling applications to use is largely dependent on the depth of the mine it serves [24]. This is expressed in Figure 4, where a hierarchy of cooling applications is presented [23]. As mines get deeper, the need for cooling increases and the introduction of more cooling applications become necessary. The hierarchy serves as a basic structure for determining the phases in which new cooling applications should be implemented.

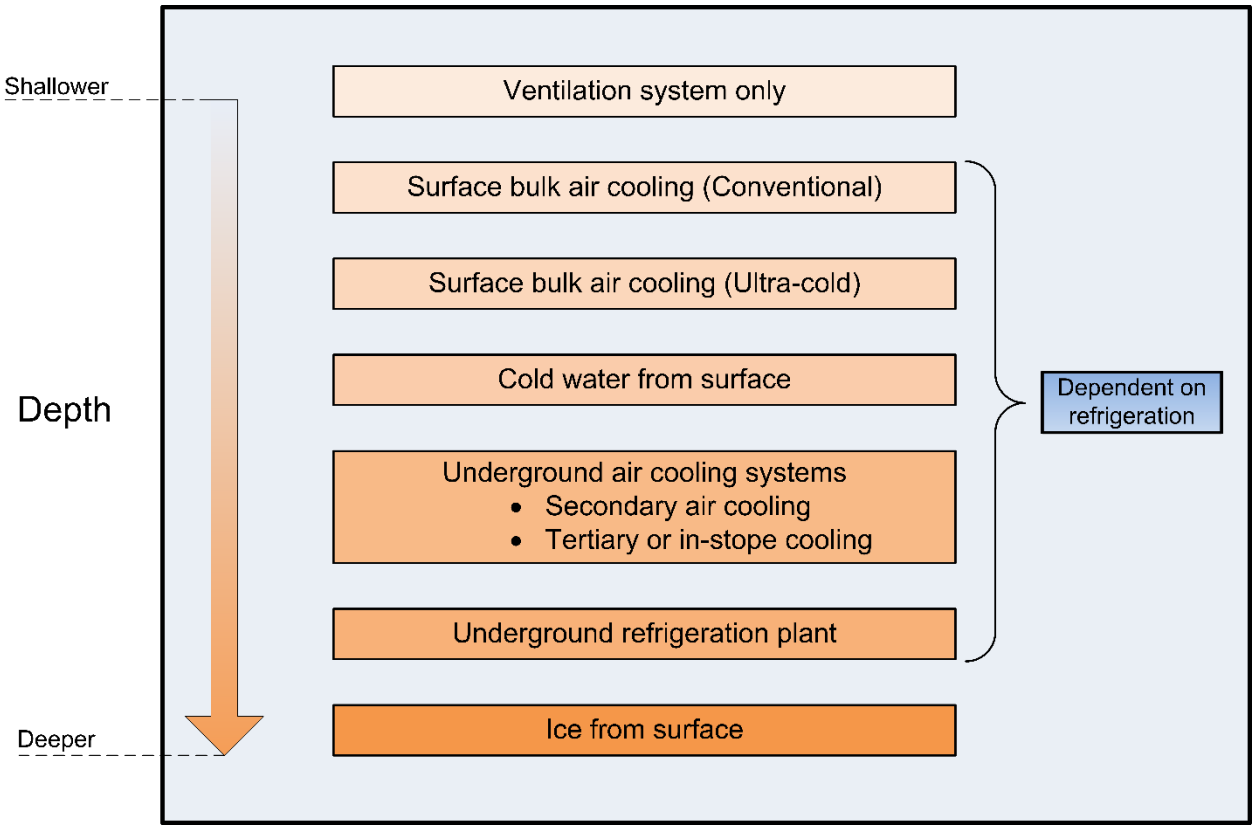


Figure 4: Generic hierarchy of the application of mine cooling at depth (adapted from [23], [24])

It is indicated in Figure 4 that all but two of the cooling applications listed are dependent on the chilled water produced at a refrigeration plant to perform their cooling function. Due to this dependence, one of the most common configurations of mine cooling systems is the centralised

surface mine cooling system [25]. This type of system utilises the superior heat rejection capacity of a surface refrigeration plant compared to its underground variants [25]. To illustrate how this system works, a simplified schematic layout of a typical centralised surface mine cooling system is presented in Figure 5.

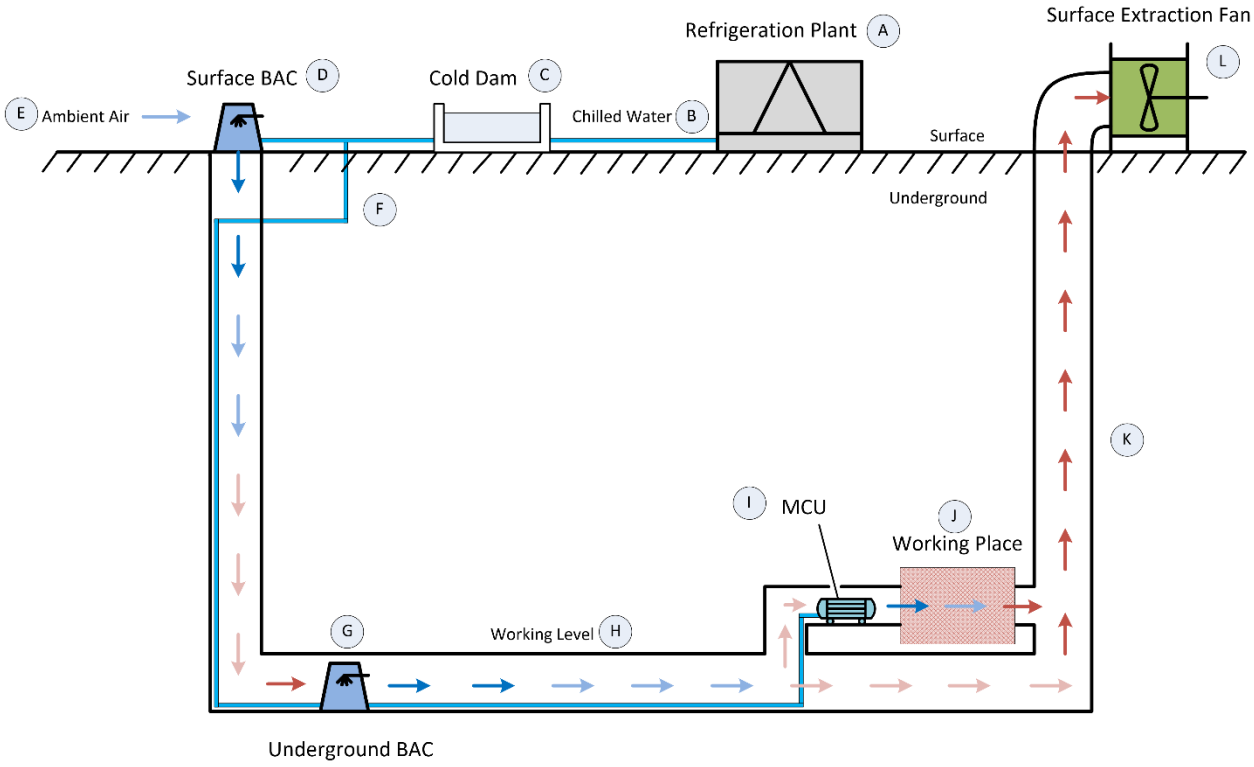


Figure 5: Diagram representing a centralised mine cooling system and its constituent parts

Figure 5 shows that surface extraction fans (L) induce the flow of air through the mine, which pulls ambient air into the mine through the shaft and surface BAC (D). Meanwhile, the refrigeration plant (A) produces chilled water (B) that is pumped to an intermediate storage dam or cold dam (C). Here, some of chilled water is pumped to the surface BAC (D) where it is distributed by sprayers to absorb heat from the ambient air (E) as it moves into the shaft.

The rest of the chilled water (F) is fed underground via an insulated pipe network to support the underground cooling process. As the air flows down the shaft, it is heated by the underground environment as well as the air auto-compression phenomenon [19]. To reject some of this heat, it is directed through an underground BAC (G).

The air then flows through the working level (H) where it absorbs more heat. Once it reaches the active working areas (J), it must be cooled again. This time, cooling is achieved by a mobile

cooling unit (I), which produces cold air for a localised region. Finally, the hot air escapes the underground environment via the return air way (K).

This process illustrates the overarching goal of mine cooling systems, which is to act as heat sinks to capture and remove heat generated in the underground environment. The surface refrigeration plant is also shown to be the core component of the cooling process. The chilled water produced at the plant services the surface BAC, underground BAC, and the MCU.

1.2.4 Vapour-compression refrigeration system performance

Most refrigeration systems on deep-level mines are based on the vapour-compression refrigeration cycle [26]. For industrial-sized refrigeration systems such as mine refrigeration plants, ammonia is typically used as a refrigerant because of its superior coefficient of performance (COP) compared to other refrigerants, which leads to reduced energy cost [27].

The goal of any refrigeration system is to move heat from an area where it is unwanted to an area where it can then be dissipated [27]. The more heat the refrigeration system can remove from the refrigerated space and dissipate into another, the more effective the system becomes. The measure of a refrigeration system's efficiency is its coefficient of performance (COP) and is given by Equation (1 [27]. It is defined as the ratio of the cooling capacity ($\dot{Q}_L$), which is the rate at which heat is transferred away from the refrigerated space, and the required input ($\dot{W}_{net,in}$), which is the rate of electrical energy input to the compressor.

$$COP = \frac{\dot{Q}_L}{\dot{W}_{net,in}} \tag{1}$$

1.3 Problem Statement

Rising operating costs challenge the profitability of deep-level mines. Electricity costs account for as much as 20% of the operational expenses of a typical deep-level mine. Since 2007, electricity prices have increased significantly above inflation and are projected to continue rising rapidly. Deep-level mine cooling systems, which rely on refrigeration systems for their cooling function, are expensive to operate. These cooling systems can be responsible for up to 32% of the electricity cost of deep-level mines. Inefficient operation of refrigeration systems further reduces the profitability of these mines, especially since it can be difficult to determine the optimal control parameters to operate the system efficiently.

1.4 Refrigeration system optimisation and modelling

Refrigeration plants are critical to ensuring safe working conditions, but are energy intensive. The rise in electricity prices has made them even more expensive to operate, which introduces the need to optimise their performance [28].

Understanding the interactions between the mechanical components, thermodynamic characteristics, and external environment with respect to the target variable(s) of a refrigeration system forms the basis of the optimisation process [29]. The target variable represents the variable to be optimised. In the case of refrigeration systems, the most used target variables are the COP, total cost, energy consumption, exergetic efficiency, operational expenditure, and capital expenditure [29]. A control system is then programmed to facilitate the favourable conditions that result in the optimisation of the target variable [30].

The optimisation process, therefore, relies on an accurate model of the component interactions to find these favourable conditions [30]. There are many approaches that are suitable for modelling refrigeration systems with their own advantages and disadvantages. Afroz *et al.* [30] and Estrada-Flores *et al*. [31] reviewed the use of models in refrigeration-related applications and divided the approaches into three categories, namely white-box, grey-box, and black-box models.

The white-box model refers to the physics-based first-principles approach to modelling by using universal mathematical equations stemming from the conservation of energy, mass, and momentum [31]. They have the benefit of being easy to analyse and generalise well [30]. However, they are complex to develop due to the high initial knowledge input and the requirement of physical properties. These models typically have poor accuracy due to the reliance on physical properties that can change over time [30].

Black-box approaches are empirical models that do not require any theoretical considerations and can adequately establish relationships between dependent and independent variables. The advantages of these models are their relative simplicity, high prediction accuracy, and easy integration with real-time control systems. The disadvantages are that they require significant amounts of training data and do not extrapolate well outside of the operational range defined by the data used to train the model [30].

The grey-box model is a hybrid between white-box and black-box approaches [31]. It uses universal mathematical equations to model the well-understood interactions in a system while using empirical methods to model lesser-understood interactions within the system [31]. Grey-box models are very accurate, generalise well, and they can extrapolate better than black-box

models. However, more effort is required to develop these models, and their accuracy is highly dependent on the richness of the data used to train the model [30].

1.4.1 Simulation in mining refrigeration

Simulation models of refrigeration systems have been proven to be effective in energy management applications [25]. This is the predominant method used in the mining industry to model refrigeration systems. Simulation is accurate and allow for high-level system inefficiencies to be identified and addressed [18]. Simulation models serve secondary purposes such as being a form of documentation of the system processes and layout [32].

The modeller is, however, forced to understand the underlying process and interactions being modelled, which is beneficial for the developer but makes the development process more difficult [32]. These models also suffer from deteriorating performance over time as operational and component conditions change [18]. Furthermore, they are modelled after a specific system and must therefore be built from scratch for each system [18]. The process of building, calibrating, and running simulations on these models can also be very difficult and time consuming due to the number complex interactions that have to be simulated [29], [33].

1.4.2 Artificial intelligence in refrigeration system modelling

Artificial intelligence (AI) techniques are the most popular non-classical method used to optimise refrigeration systems [29]. Among the AI techniques used for modelling, the artificial neural network (ANN) variants are the most popular [29]. This is because they incorporate the strengths and negate the weaknesses of many modelling approaches [34]. Specifically, the feedforward variants, including the radial basis function network (RBFN), the adaptive neuro-fuzzy inference system (ANFIS), the recurrent neural network (RNN), and the multi-layer perceptron (MLP), are the most commonly used AI techniques to model refrigeration systems [35].

Radial basis function networks

RBFNs are also a feedforward variation of ANN that have been successfully used to model refrigeration systems with a high degree of accuracy [35]. The technique consists of a single hidden layer containing radial basis transfer functions in the layer [36]. These networks are suitable for non-linear regression, classification, time series, and pattern recognition [35].

Adaptive neuro-fuzzy inference systems

An ANFIS is another variation of feedforward ANN which combines the learning capabilities of ANNs with the reasoning capabilities of fuzzy logic [37]. This hybrid technique is considered the future of expert controllers for refrigeration systems [38]. Since characteristic models and optimisation techniques are typically implemented separately, an ANFIS presents a neat package for combining their functionalities into one. The limitation of this technique is the near expert-level knowledge requirement in the design phase [38].

Recurrent neural networks

RNNs are a type of ANN that are designed to process temporal, or time-sequence data. They "remember" past information to make predictions about the future behaviour of the underlying system [34]. RNNs are not feedforward networks since they make backward connections. This backward connection is what enables an RNN to store information about previous inputs and its predictions therefore are a function of the current and previous inputs. These networks are best employed when working with time-series data and forecasting [35].

Multi-layer perceptrons

Multi-layer perceptrons are feedforward ANNs, meaning signals flow unidirectionally from input to output [35]. These ANNs are becoming an increasingly popular technique in the modelling of refrigeration systems with the goal of saving energy [35]. MLPs model the relationships between a set of input and output parameters [30]. The technique does not require a deep understanding of the system physics, making it easier to develop and update when conditions change [30].

This type of model accounts for deviations in assumptions and design specifications since it is developed using historical data [35]. MLPs are remarkably noise-immune and fault-tolerant, making it a suitable candidate for modelling a system which inherently contains noisy data [39]. It has also been shown that they can predict refrigeration system performance more consistently than simulation and regression models [30].

1.5 Artificial neural networks

An artificial neural network is a machine learning technique loosely based on the learning mechanism of living organisms [40]. An ANN is considered a universal function approximator, which means that, given enough data, an ANN is theoretically capable of modelling any

system [41]. It can *learn* the relationship between input variables and target variables through a training process, enabling it to make predictions based on previous examples.

This section explores how feedforward ANNs work by first considering the perceptron which comprises a single computational layer and is the most basic form of ANN. Multi-layer perceptrons are discussed to examine the interactions between the fundamental building blocks of the network and how they contribute to more complex modelling capabilities. Closer consideration is then given to how an ANN *learns* the relationships between variables during the training process and how it is able to make predictions based on this process. Lastly, consideration is given on how architectural setup and training influences the network.

1.5.1 The perceptron

The perceptron is the artificial neural network with the most basic architecture [42]. It has an input layer and a single computational layer which is also the output layer. Each layer consists of processing elements that are referred to as a neuron or node which produces the output of that layer [43].

A perceptron consists of input(s) (x_i), their corresponding weight (w_{ij}), and a bias (b_i) which is transmitted to the neuron in the computational layer [44]. To produce an output (y_k), two computations are made at this neuron. In order, these are: summation (Σ) and activation (f) [40]. Figure 6 shows a schematic representation of the perceptron's architecture and the fundamental building blocks which define its structure [45].

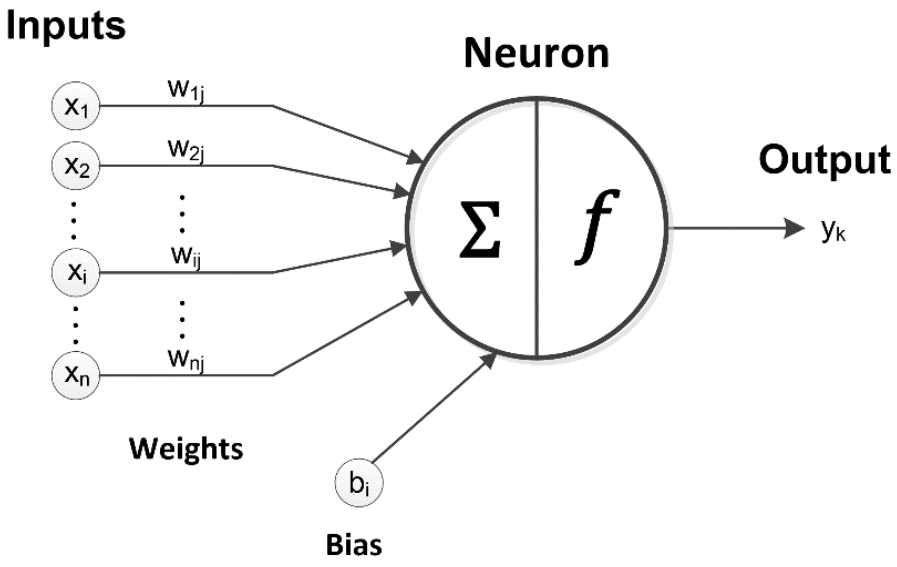


Figure 6: The architecture of the perceptron (adapted from [45])

The number of input nodes (n) directly corresponds to the number of input variables chosen for the perceptron [46]. The input layer is also not a computational layer, meaning the input nodes merely pass the input values along, whereafter they are multiplied by a weight. Weights are scaling coefficients that either amplify or minimise the effect of a certain input before it is transmitted to the neuron(s) in the next layer [46]. Additionally, a bias is added to account for invariance in parts of the output, which can emerge when input data is not mean centred [40].

At the neuron, the sum is taken of all the weighted inputs as well as the bias to produce the net input value (u_i). The mathematical equation to obtain the net input value is given by Equation (2) [46]. A schematic representation of the process to obtain the net input value is given in Figure 7 [47].

$$u_i = \sum_{i=1}^{n} w_{ij} x_i + b_i \tag{2}$$

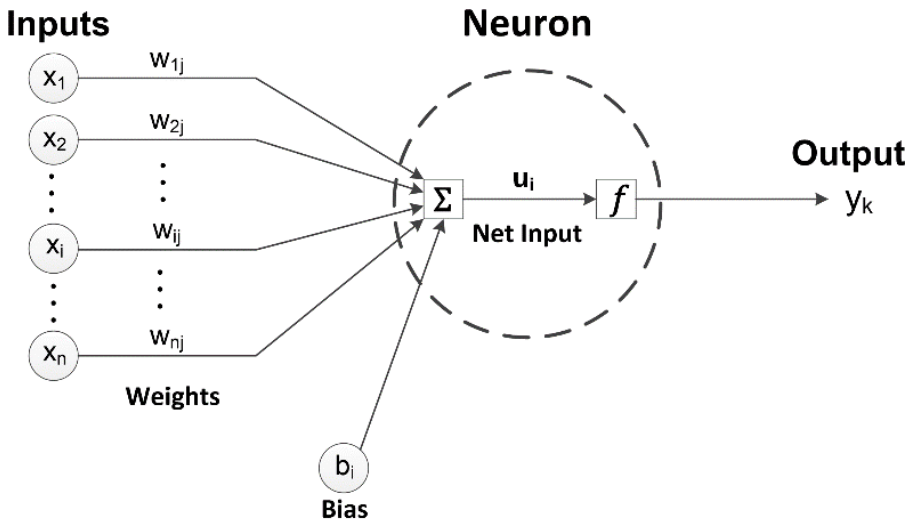


Figure 7: Schematic representation of the process to obtain the net input value of a perceptron (adapted from [47])

The net input value is also referred to as the pre-activation value. This is because after summation, the pre-activation value is transformed into the output or post-activation value (y_k) by the activation function (f) [40]. The activation function bounds and re-scales the pre-activation

value, which can be in an arbitrarily large range, within a range defined by the selected function [48]. The equation for the post-activation value is given by Equation (3) [49]. This value is the output of the perceptron.

$$y_k = f(u_i) \tag{3}$$

A multitude of activation functions have been researched in the literature. The classical activation functions have been well studied and are known to produce excellent results for most regression problems [50]. Table 2 contains a subset of common classical activation functions along with their type, rescaling ranges, and mathematical equations [45], [51]. The graphical shapes of the activation functions listed in Table 2 are given in Figure 8 [45], [51].

Table 2: Classical activation functions and their properties (adapted from [45], [51])

Name	Type	Range	Equation	Equation No.	Figure
Identity	Linear	(-∞, ∞)	$f(u) = u$	(4)	a
Sign	Nonlinear	[-1,1]	$f(u) = sign(u)$	(5)	b
Sigmoid	Nonlinear	[0,1]	$f(u) = \frac{1}{1+e^{-u}}$	(6)	c
Bipolar Sigmoid	Nonlinear	[-1,1]	$f(u) = \frac{1-e^{-u}}{1+e^{-u}}$	(7)	d
Hyperbolic Tangent	Nonlinear	(-1,1)	$f(u) = \tanh(u)$	(8)	e
Rectified Linear Unit (ReLu)	Nonlinear	[0, ∞)	$f(u) = \max(0, u)$	(9)	f

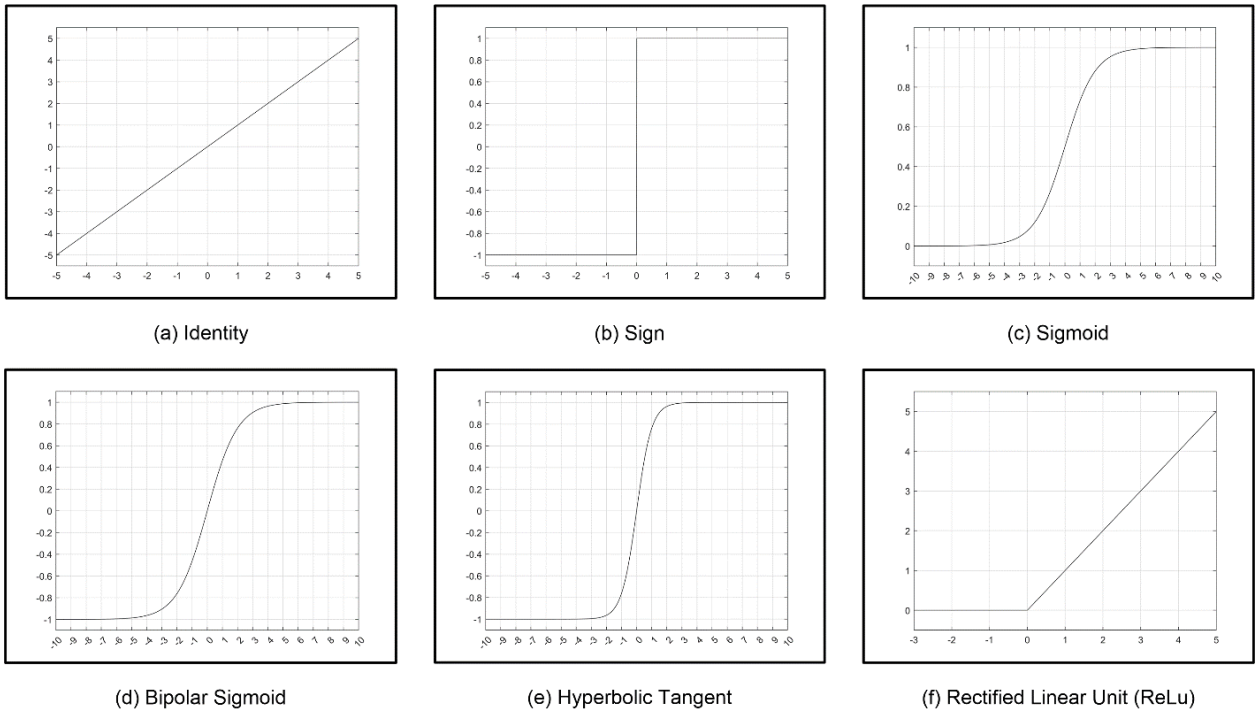


Figure 8: The graphical shapes of the common activation functions (a) Identity, (b) Sign, (c) Sigmoid, (d) Bipolar Sigmoid, (e) Hyperbolic Tangent, (f) Rectified Linear Unit (adapted from [45], [51])

The addition of activation functions has significant implications for the capabilities of an ANN, especially when working with multi-layer architectures and during training. Sections 1.5.2 and 1.5.3 will deal with these implications in more detail.

With the activation function incorporated into the architecture, the fundamental input-output equation of the perceptron can be constructed by substituting the chosen activation function Equations (5) to (9) into Equation (3. Table 3 shows the fundamental input-output equation of the perceptron when using three of the most widely used activation functions [45].

Table 3: Fundamental input-output equation of the perceptron using common activation functions (adapted from [45])

Activation Function	Input-output Equation	Equation No.
Sigmoid	$y_k = \frac{1}{1 + \exp\left(-\sum_{i=1}^{n} w_{ij}x_i + b_i\right)}$	(10)
Hyperbolic Tangent	$y_k = \tanh(\sum_{i=1}^{n} w_{ij}x_i + b_i)$	(11)
Rectified Linear Unit (ReLu)	$y_k = \max(0, \sum_{i=1}^{n} w_{ij}x_i + b_i)$	(12)

These equations are examples of the culmination of all the components of the perceptron and how they interact. The inputs are given a weight and the resulting value is trasmitted to the neuron. The summation function takes the sum of the weighted inputs as well as a bias to produce the pre-activation value which is then mapped onto a range defined by the activation function. The resulting post-activation value is the output of the perceptron. A schematic representation of this process with ReLu as the activation function is given in Figure 9 [45].

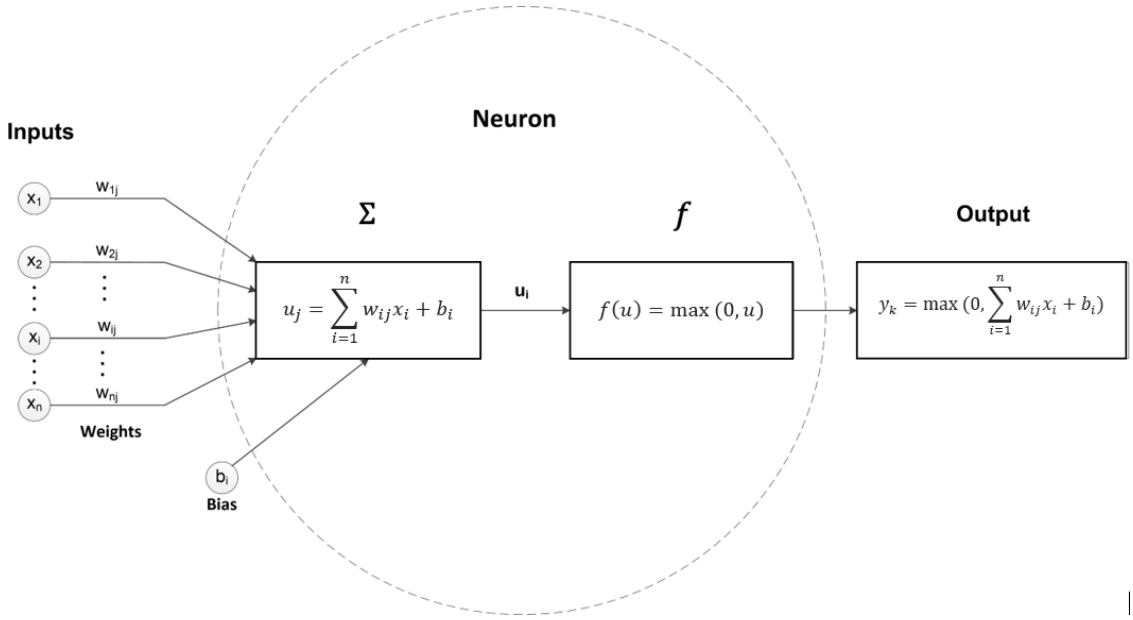


Figure 9: A schematic representation of the data processing procedure in the perceptron (adapted from [45])

1.5.2 The multi-layer perceptron

Multi-layer neural networks consist of more than one computational layer and can have an arbitrary number of neurons in those layers [40]. Any layer between the input and output layers are called hidden layers because their outputs are not directly visible to the user [40].

The multi-layer perceptron is a special case in the broader class of multi-layer neural networks where layers are fully connected [52]. This means that each output in a layer is connected to all neuron inputs in the subsequent layer [53]. As in the case of the perceptron, the multi-layer perceptron is a feedforward network, meaning that information flows exclusively from the input layer in the direction of the output layer [54]. A schematic representation of a typical multi-layer perceptron with one hidden layer is given in Figure 10 [34].

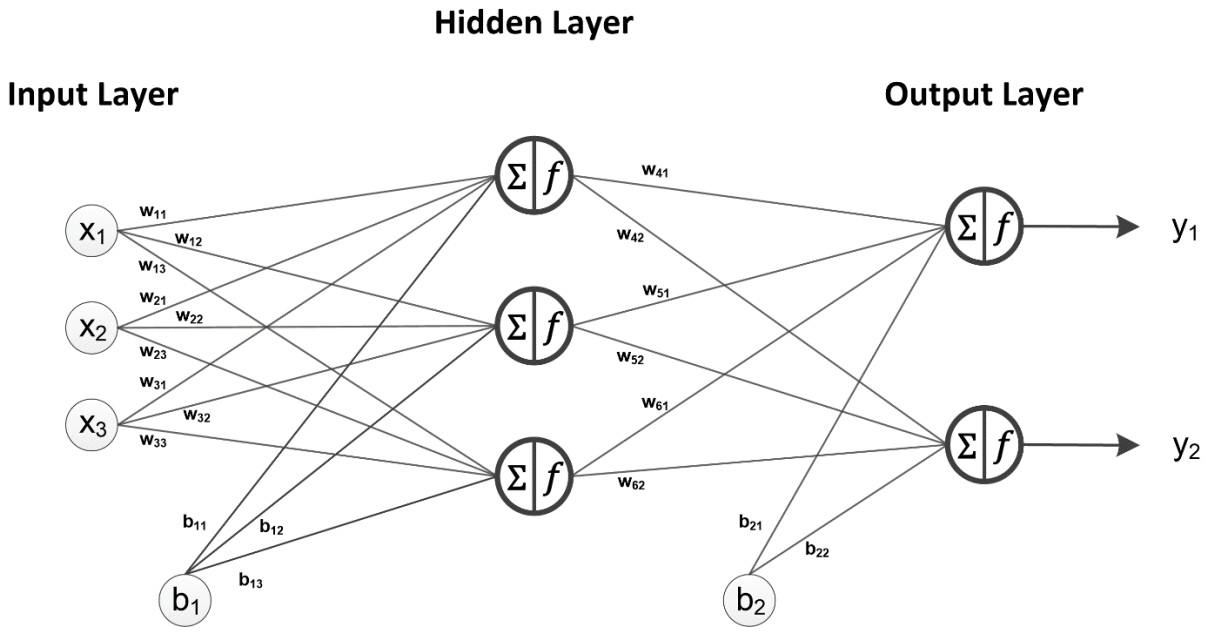


Figure 10: Schematic representation of a generic multi-layer perceptron (adapted from [34])

The input layer of a multi-layer perceptron transmits values to the first hidden layer. Neurons in the hidden layer(s) are processing elements that act in parallel and transform vector inputs to scalar outputs [55]. The neurons in the output layer are also processing elements and, like the input layer, contain the same number of neurons as the number of target variables selected for solving the chosen problem [56].

Each computational layer in the multi-layer perceptron can be seen as a feature extractor. In the case of the multi-layer perceptron shown in Figure 10, the hidden layer extracts features from the

input layer and the output layer extracts features from the hidden layer. Adding more hidden layers will enable more complex features to be extracted from the input data. [57].

The multi-layer perceptron gains superiority over its single-layer counterpart only when non-linear activation functions are used. Using linear activation functions in the multi-layer perceptron limits the network's capabilities to linear transformation operations and the network would behave like a linear regression model [58]. This makes sense when looking at the architecture of the multi-layer perceptron. Each layer would perform a linear transformation on its inputs and a linear function of a linear function is also linear [59].

Conversely, using non-linear activation functions makes the ANN capable of more complex tasks such as non-linear regression [53]. When activating data with these activation functions, non-linear mappings between input and target variables are achievable [50]. This makes the multi-layer ANNs powerful modelling tools because of the complex, multivariable, and often non-linear relationships found in real-world systems.

1.5.3 Multi-layer perceptron training

The learning process can be classified as supervised or unsupervised [60]. In supervised learning, the training dataset consists of feature vectors (which are the input vectors to the ANN) and labels (which are the target vectors of the ANN). The goal of a supervised training algorithm is to produce a model that can predict the target for a given feature vector [53]. This section explores techniques that fall under the supervised learning umbrella.

The training process of an ANN involves continually adjusting weights between neurons to produce an output value that increasingly approaches the target value [61]. In feedforward architectures, training algorithms using gradient-based optimisation such as the well-known backpropagation algorithm and its derivatives have been widely used due to their success in real-world applications [62].

Cost functions

The goal of training algorithms using gradient-based optimisation is to minimise the error between the desired output (y_i) and the output of the model ($\hat{y}_i$) [62]. This error is calculated using a cost function such as the mean squared error (MSE) function, as given by Equation (13) [63].

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N} \varepsilon(n) \quad (13)$$

The MSE cost function is often referred to as the sum-of-squares error and calculates the average error (ε_{av}) over all outputs of the ANN for all input patterns ($n = 1\ to\ n = N$) and is commonly used for regression models that require a real-valued output [64].

The full derivation of the MSE cost function is given in Appendix A, along with some of the most common cost functions that were not covered here. Their strengths and weaknesses are discussed as well as their suitability for solving certain problems.

Backpropagation algorithm

Equation (13 is a function of the ANN output ($\hat{y}_i$) which, in turn, is a function of the weights and biases contained in the network architecture [64]. Therefore, by using cost functions as the basis for determining the accuracy of an ANN's predictions, the training process becomes a problem of adjusting the weights and biases such that the cost function error is minimised [46].

The weights between the final hidden layer and the output layer can be optimised in a direct manner. This is because the predictions in the output layer can be compared directly to their target value (y_i) in the cost function [65]. Weights between other layers are more difficult to optimise since they are composition functions of the weights in previous layers [40].

The backpropagation algorithm can address this problem, given that both the activation and cost functions are continuous and differentiable [62]. If this is the case, the post-activation variable of the output layer is a differentiable function of the weights, biases, and inputs variables of the network. The cost function, in turn, is a differentiable function of the post-activation variable of the output layer [64].

By the chain rule of partial derivatives, the cost function is then a differentiable function of the network weights, and its derivative with respect to the weights can be evaluated. The evaluated derivative can then be used to compute the adjustments that need to be made to the weights [63]. Many algorithms can be used to adjust the weights in a time-efficient manner and are discussed in more detail in Section 1.5.4.

This procedure is a two-stage process. In the forward pass, the network computes the output with the current weight values using the summation and activation functions and its error using the cost function. In the backward pass, the cost function's derivative with respect to the weights is evaluated and adjustments to the weights are made in sequence. This process repeats until an accuracy requirement is satisfied [66]. The full backpropagation procedure is given in Figure 56 in Appendix A [67].

The user has the choice of allowing the algorithm to update weights after each training example, called on-line learning, or after being presented with all training examples, called batch learning [59].

1.5.4 Parameter optimisation algorithms

In the previous section, the backpropagation algorithm was introduced. It was established that the algorithm can efficiently backpropagate the error of the cost function using its derivative with respect to the network weights. This makes it possible to adjust weights throughout the entire network based on these derivatives and reduces the problem of training an ANN to the optimisation of weights such that the cost function error is minimised [46]. This section explores some of the popular parameter optimisation algorithms used to develop ANN models of refrigeration systems that are based on backpropagation's use of the derivative information.

Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm is a powerful iterative minimisation algorithm that combines the strengths of gradient descent and the Gauss-Newton method to solve nonlinear least squares problems [68]. This is useful when dealing with fitting problems in parameterised mathematical models that aim to minimise a multivariate cost function, which is the case in multi-layer perceptrons [69]. The Levenberg-Marquardt algorithm is an intrinsically batch learning technique and relies on the underlying cost function being in the general form of the sum-of-squares equation as given by Equation (13) [64].

The update rule for gradient descent applied to the sum-of-squares cost function of a generalised multi-layer perceptron is given by Equation (14).

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \lambda \boldsymbol{J}^T \boldsymbol{r} \tag{14}$$

The network's weight values are stored in a weight vector $\boldsymbol{w}$ and, using Equation (15), the updated weight values ($\boldsymbol{w}_{i+1}$) are determined by taking the current weight values ($\boldsymbol{w}_i$) and subtracting the product of a scaling factor (λ) and a gradient vector ($\boldsymbol{J}^T\boldsymbol{r}$) [40]. The gradient vector consists of the first order gradients of the error function, which can be represented using the transposed Jacobian matrix ($\boldsymbol{J}$) and the vector of errors or residuals ($\boldsymbol{r}$) [70].

The update rule of gradient descent uses gradient information ($\boldsymbol{J}^T\boldsymbol{r}$) of the error surface to update the parameters in the direction of steepest descent with the goal of finding the minimum of the cost function. An assumption of local linearity is therefore made, which simplifies the optimisation process, but often does not hold over the entire error surface [69]. The scaling factor is often referred to as the learning rate and determines the step size of each iteration. A small learning rate provides smooth convergence but increases the time it takes to find the minimum, while a large learning rate can converge faster but is susceptible to overshooting the optimal point [68].

Gradient descent is versatile and very efficient even when dealing with functions with many parameters, which is often the case in large ANNs [69]. However, it suffers from various convergence problems. When large gradients are present on the error surface, the step size tends to be large and when small gradients are present, the step size tends to be small. This can lead to the algorithm getting stuck in saddle points and overshooting the optimal point. This issue can be improved if both gradient and curvature information from the second order derivative was used to determine the step size [68]. One algorithm which exploits this is the Gauss-Newton method.

The matrix form of the update rule for the Gauss-Newton method, applied to the sum-of-squares cost function of a generalised multi-layer perceptron, is given by Equation (15) [68]:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T\boldsymbol{r} \tag{15}$$

Equation (15) resembles gradient descent with some key differences. The learning rate has been replaced because the step size is now determined by the curvature of the cost function. The curvature information $(\boldsymbol{J}^T\boldsymbol{J})^{-1}$ in Equation (15 approximates the Hessian ($\boldsymbol{H}$) using the Jacobian and consequently assumes that the error surface is quadratic around the initial values of the weight vector $(\boldsymbol{w}_0)$. This assumption comes with the advantage of rapid convergence but is sensitive to initial conditions, specifically the linearity of the surface around $\boldsymbol{\varepsilon}(\boldsymbol{w}_0)$ [64]. Levenberg [71] and Marquardt [72] capitalised on the complimentary nature of gradient descent and the Gauss-Newton method to create the Levenberg-Marquardt update rule, given by Equation (16). The Levenberg-Marquardt update rule is designed to promote the advantages of

either gradient descent or the Gauss-Newton method and diminish the other using the adaptive learning rate (λ) and the identity matrix ($\boldsymbol{I}$) as the regularisation term [68].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \boldsymbol{J}^T \boldsymbol{r} \tag{16}$$

Initially, the algorithm is executed without bias (which favours the Gauss-Newton method) and updates the weights of the next iteration. If the error reduces, the implication is that the quadratic assumption is yielding good results and should be reinforced. This is done by decreasing the learning rate by a large factor (usually a factor of ten), which favours the Gauss-Newton method. Conversely, if the error increases, the linear assumption should be reinforced by increasing the learning rate by a similar factor. This means that, when approaching a solution, Gauss-Newton is favoured to avoid gradient descent's overshoot issues and when far from a solution, gradient descent is favoured to speed up convergence [64].

Scaled conjugate gradient algorithm

The conjugate gradient algorithm and its evolution, the scaled conjugate gradient (SCG) algorithm, are optimisation algorithms based on the observation that the use of successive gradient vectors as the search direction is a suboptimal solution [64]. Instead, at each iteration, the gradient information can be used to determine a better search direction ($\boldsymbol{d}$). The (scaled) conjugate gradient algorithm achieves this by choosing the search direction so that the component of the gradient vector ($\boldsymbol{g}_{i+1}$) that is parallel to the previous search direction ($\boldsymbol{d}_i$) is nullified. This concept is represented mathematically by Equation (17) and graphically by Figure 11 [64].

$$\boldsymbol{g}_{i+1}^T \boldsymbol{d}_i = 0 \tag{17}$$

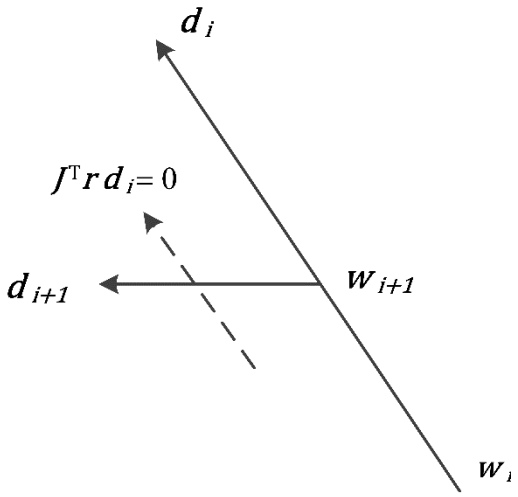


Figure 11: Visual representation of conjugate directions in two dimensions (adapted from [64])

By doing this, the new search direction will be biased away from the previous search direction. If the error surface is assumed quadratic, two arbitrary search directions ($\boldsymbol{d}_x$, $\boldsymbol{d}_y$) constructed with this principle and that satisfy the condition given by Equation (18) are conjugate (with respect to the Hessian) [64].

$$\boldsymbol{d}_x^T \boldsymbol{H} \boldsymbol{d}_y = 0, \;\; x \neq y \tag{18}$$

An arbitrary update rule for the weight vector can be constructed using a similarly arbitrary search direction ($\boldsymbol{d}_i$) and step size (α_i), given by Equation (19) [73].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_i \boldsymbol{d}_i \tag{19}$$

Now, if a set of search directions can be found which are mutually conjugate, the step size (α_i) can be derived as shown in Equation (20) [73].

$$\alpha_i = -\frac{\boldsymbol{d}_i^T \boldsymbol{g}_i}{\boldsymbol{g}_i^T \boldsymbol{H} \boldsymbol{d}_i} \tag{20}$$

Using this step size in the update rule ensures that each gradient vector is orthogonal to all previous conjugate directions. In the case of a quadratic error surface, this is significant because,

after W iterations (where W represents the number of weights in the weight vector), the gradient vector will have been nullified in each direction, meaning that the minimum has been reached [74].

This property only holds if a set of mutually conjugate search directions are used. This can be done by selecting the first search direction as the negative of the gradient ($\boldsymbol{d}_1 = -\boldsymbol{g}_1$) and each successive direction as a linear combination of the current gradient and the previous search direction. Equation (21) shows the update rule for mutually conjugate search directions [75].

$$\boldsymbol{d}_{i+1} = -\boldsymbol{g}_{i+1} + \beta_i \boldsymbol{d}_i \tag{21}$$

The formula for coefficient β_i is given by Equation (18) and is derived by imposing the conjugacy condition given in Equation (22) [75].

$$\beta_i = \frac{\boldsymbol{g}_{i+1}^T \boldsymbol{d}_i}{\boldsymbol{d}_i^T \boldsymbol{H} \boldsymbol{d}_i} \tag{22}$$

It should be noted that determining the Hessian is necessary for each iteration, but is a computationally expensive process. For this reason, alternative methods for finding α_i and β_i coefficients have been developed involving the line search procedure. Line search itself, however, can become a computationally expensive procedure since, for every line search, the cost function must be evaluated multiple times [64]. Møller [74] modified the conjugate gradient algorithm to avoid the line search procedure problem, called the scaled conjugate gradient algorithm. The SCG step size is given by Equation (23).

$$\alpha_i = -\frac{\boldsymbol{d}_i^T \boldsymbol{g}_i}{\boldsymbol{d}_i^T \boldsymbol{H}_i \boldsymbol{d}_i + \lambda_i \|\boldsymbol{d}_i\|^2} \tag{23}$$

The SCG algorithm improves on the conjugate gradient algorithm by adjusting the size of the region in which the quadratic assumption is made, improving its accuracy [76]. The size of the region is governed by a scalar constant (λ). A larger value of λ leads to a smaller trust region and, conversely, a smaller value of λ leads to a larger trust region [64]. The SCG algorithm shows a significant improvement in convergence speed compared to the base conjugate gradient algorithm [64]. A detailed description of the conjugate gradient- and SCG algorithms, as well as the procedure for choosing the value of λ, is given in Appendix A.

1.5.5 Architecture, data, and training considerations

This section explores general considerations to be taken when choosing the data to be used in the model as well as dealing with the architectural setup of a multi-layer perceptron. Special attention is given to training with the goal of reaching good generalisation.

Data considerations

Independent-sample testing

Before training, a data population is split into three subsets using a process called independent-sample testing. The three subsets are called the training-, validation-, and test subsets and are constructed by randomly sampling the population. This process is schematically represented by Figure 12 [67].

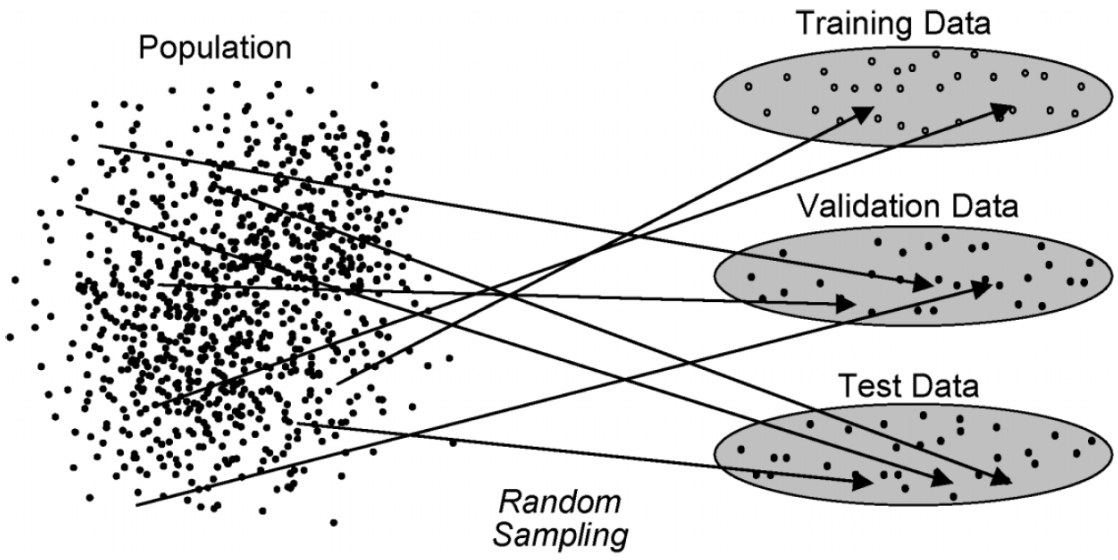


Figure 12: Schematic representation of independent-sample testing used to construct training-, validation-, and test subsets [67]

Training subset

The training subset is used to train the model. Predictions made by the model are compared to the target values and weight adjustments are made based on these results [67].

Validation subset

Samples in the validation subset are used during training to find the best configuration of the neural network and its parameters. The validation set error is commonly used to determine the optimal number of training epochs (called early stopping) and the optimal number of hidden neurons [77]. The predictive capability of trained networks can also be compared using the validation subset error [78].

Test subset

The test subset is used to determine the generalisation capabilities of a neural network [67]. Consequently, it is uninvolved in the training and validation process to ensure it remains independent and without bias. The neural network will be biased toward the training- and validation subsets because of their involvement during training and can therefore not be used to validate its predictive capabilities on unseen data [67].

Data population size

There are no generally accepted standards for determining the optimal amount of data in a population since it depends on the nature of the problem the neural network needs to solve. However, a larger population is considered better since it leads to improved generalisation capabilities and reduced overfitting [79]. Furthermore, when noise is present in the data, a larger dataset is beneficial because the general trend is easier to identify through the noise [67]. As a minimum requirement, there should also be at least more training patterns than there are connections (weights) in the network [62].

Scope of training data

The scope or boundaries of the training data limits of the predictive capabilities of an ANN. When unseen data outside of this scope is presented to the trained model, it is forced to extrapolate. ANNs are powerful interpolators and function approximators but are not proficient in extrapolating beyond its training data. The more an ANN is forced to extrapolate, the higher the prediction error will be [80]. The scope of the training data must therefore be representative of the entire operational range of the system it aims to model to avoid situations where extrapolation is required. Figure 13 shows how the prediction error of a typical neural network increases the further it extends from the scope of the training data [67].

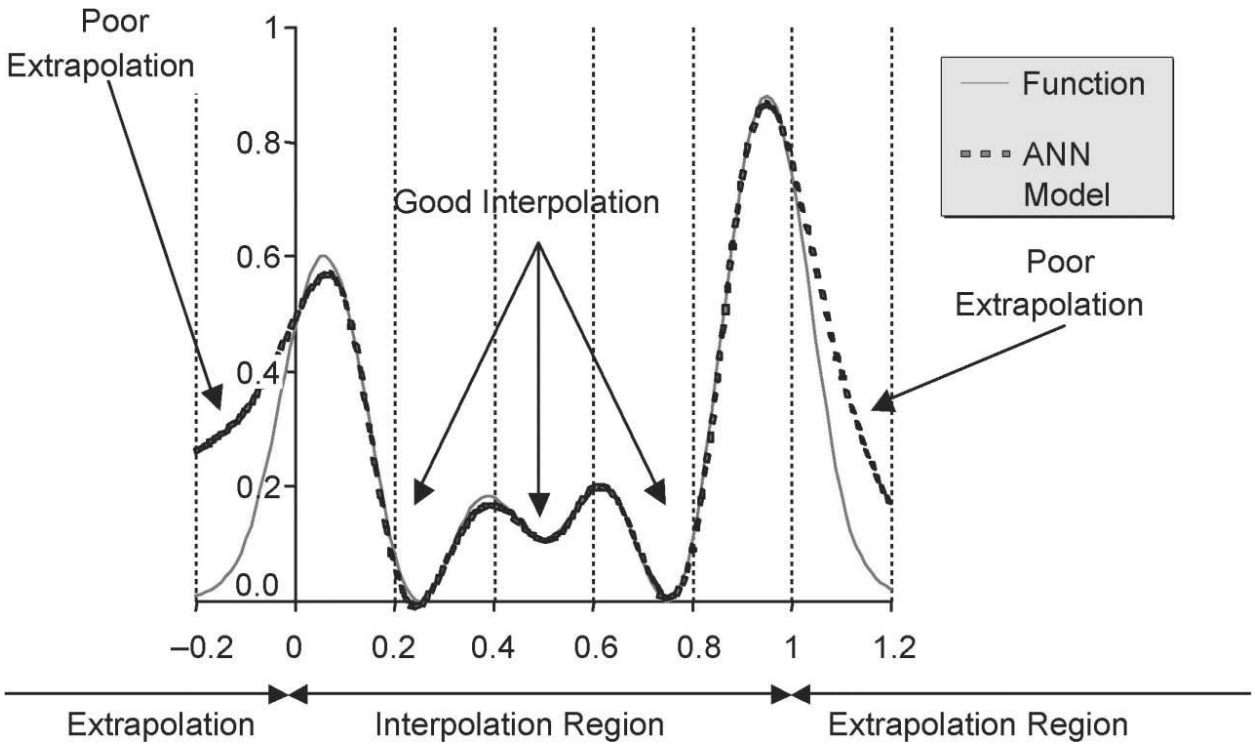


Figure 13: Graphical representation of the performance of a neural network under interpolation and extrapolation situations [67]

Training considerations

Generalisation

The generalisation capabilities of a model refer to how well it performs on new, unseen data which was not used to train the model [79]. A good model is judged on its ability to make accurate predictions on unseen data, not just its training dataset [79]. The test subset is used to determine an ANN's generalisation capabilities.

Over- and underfitting

When a model is overfit, the model has so much information processing capacity that it starts to describe random errors and noise, instead of the underlying trend in the data [77]. Conversely, underfitting is when the model is under-powered and cannot capture the complexity present in the data [81]. Both problems result in large errors between the predicted and desired value produced by the model [79]. The ideal fit represents the model with the smallest error and therefore the best generalisation capabilities. Figure 14 is a graphical representation of the generalisation capabilities of a model, showing the ideal fit as well as the problems of underfitting and overfitting [67].

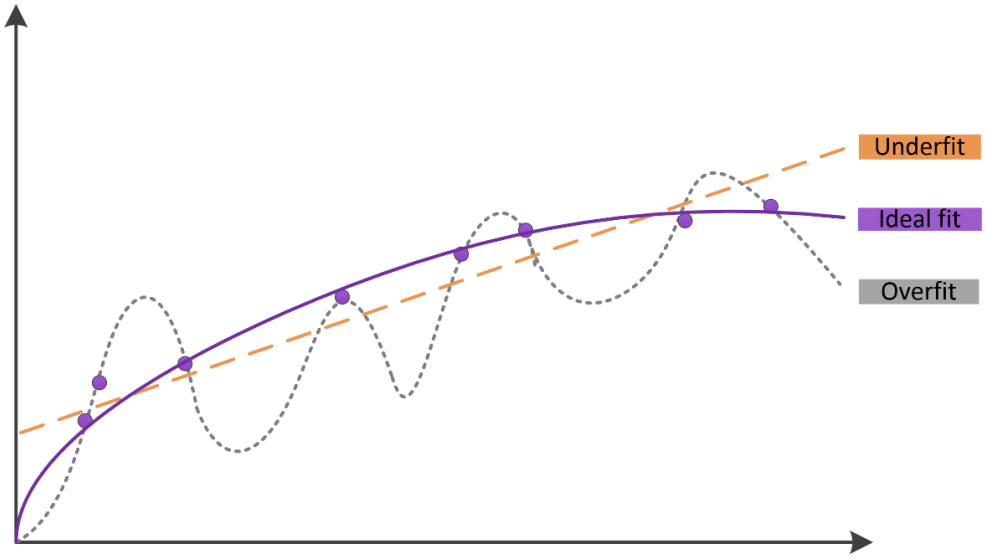


Figure 14: Graphical representation of a model's generalisation capabilities showing underfitting, overfitting and an ideal fit to a dataset (adapted from [67])

Early stopping

During training, the error of the training subset decreases monotonically, typically with diminishing returns with each epoch. At some point, the ANN will start to overfit the training data and its generalisation capabilities will degrade. This point can be found using the validation subset error. This is because the validation subset error will monotonically decrease alongside the training subset error but will start to increase once the ANN starts to overfit the training data.

Early stopping involves monitoring the validation subset error and stopping training after it increases for several successive epochs [77]. This is a popular method used to obtain the model with the best generalisation and to reduce training time. Early stopping using the training- and validation subset errors is represented graphically by Figure 15 [67].

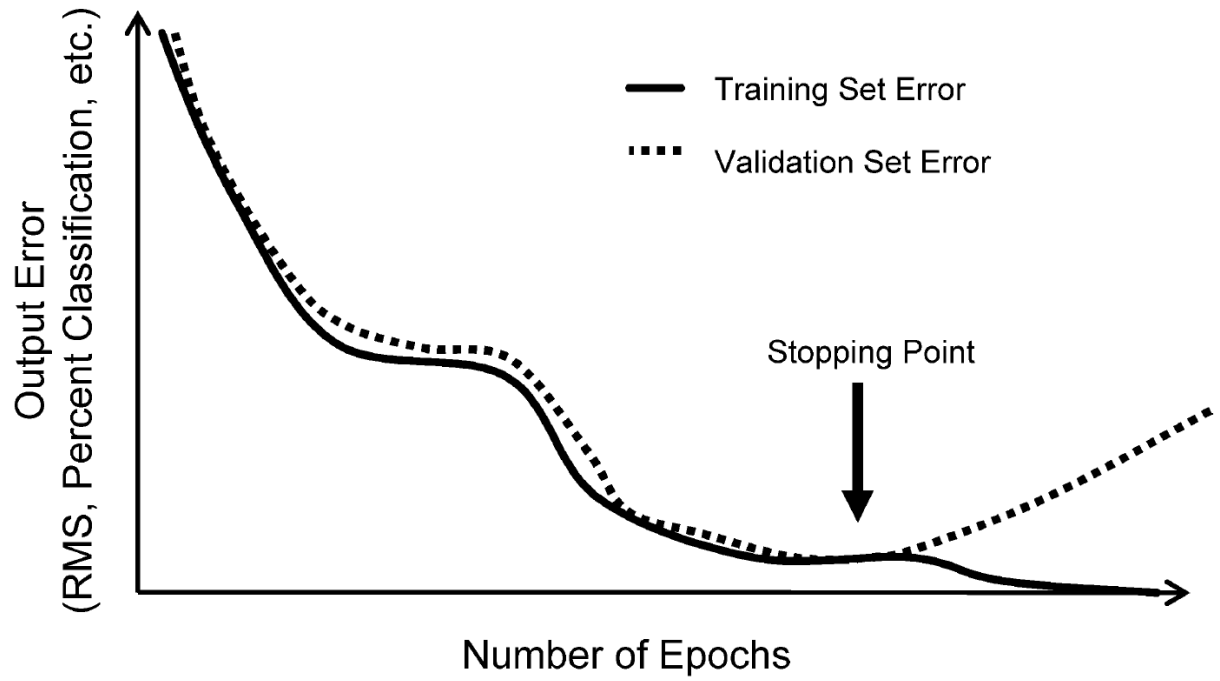


Figure 15: A graph representing the training- and validation subset errors for every epoch during training with the early stopping point indicated [67]

Weight initialisation

ANNs trained using gradient-based optimisation techniques often encounter stability problems due to either the degradation or amplification of successive gradients in hidden layers, especially in deeper network architectures (commonly referred to as the exploding- and vanishing gradient problem). These phenomena can be suppressed using good initialisation practices [82].

One method is to randomise weight values such that it has a distribution with a mean of zero and a standard deviation of one [83]. More sophisticated methods such as *Xavier* initialisation and *He* initialisation use a Gaussian distribution with a standard deviation of $\sqrt{2/(r_{in} + r_{out})}$, where r is the number of weights connected to a neuron [83]. These methods result in smaller changes in neuron outputs, avoiding the exploding and vanishing gradient problems [40].

Architectural setup considerations

The design of a multi-layer perceptron's architecture consists of determining the number of hidden layers, the number of hidden neurons in those layers, and the choice of activation and cost

functions. Each of these considerations are discussed here with reference to the literature surrounding multi-layer perceptron neural networks.

Number of hidden layers

Cybenko [84] and Hornik *et al*. [85] independently proved that a single hidden layer is sufficient to approximate any continuous function to an arbitrary accuracy. For highly non-linear or discontinuous functions, two hidden layers have been shown to be beneficial in modelling the complexities found in these functions [81]. Additionally, two hidden layers have been used in practice to accelerate convergence [67]. Multi-layer perceptrons with more than two hidden layers have rarely been found to improve the performance of the network and can exacerbate the vanishing and exploding gradient problems [86].

Number of hidden neurons

The generalisation capabilities of a multi-layer perceptron can vary significantly depending on the number of neurons in the hidden layer(s). Many theories have been proposed to solve the problem of finding the optimal number of hidden neurons in a multi-layer perceptron. Pruning methods such as the optimal brain damage [87] and optimal brain surgeon [88] algorithms have been proposed to prune the number of weights in a network based on second order derivative information.

Information criteria such as the Akaike information criterion have been used to compare models' "goodness" with the goal of reducing network complexity, with a good measure of success [81]. Some researchers address the problem using empirical methods to provide an estimate of the number of hidden neurons to employ such as the well-known "2/3 rule" and rules based on the size of the input and output layers [81].

No consensus has been reached about which theory is the best for practical use [89]. A robust method authors often employ, however, is the validation subset error stopping condition, which is an empirical method used to determine the network configuration with the best generalisation capabilities. In this method, models are trained with a varying number of hidden neurons. The error for both the training- and validation subsets are calculated and the configuration with the lowest validation error is selected as the network. During training, both the training- and validation errors will increase. However, after a certain number of epochs, the validation error will start to increase while the training error continues to decrease. At this point, the network starts overfitting the training subset, and its generalisation capabilities decline [67].

Training is stopped after the validation error has successively increased for a certain number of epochs, indicating a trend. The best network is selected at the minimum of the validation error. Figure 16 shows the training- and validation subset errors as a function of the number of hidden neurons during training. The optimum number of hidden neurons is indicated as the point where the validation subset error is at a minimum [67].

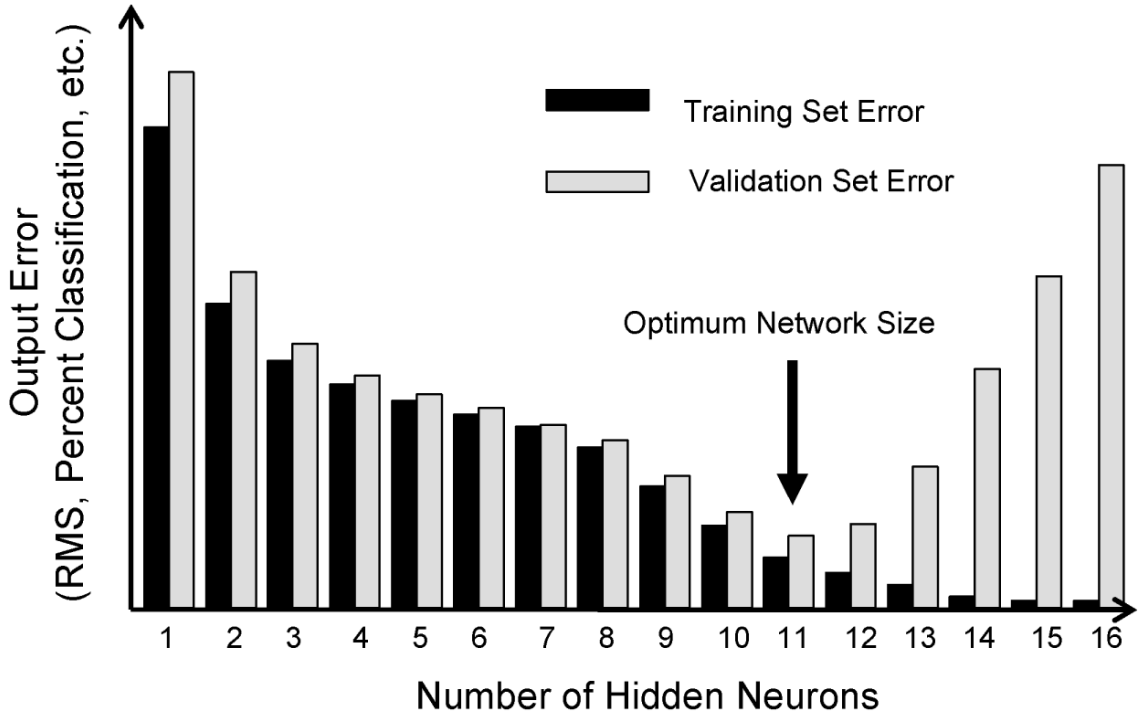


Figure 16: Graphical representation of the validation- and training subset error as a function of the number of hidden neurons [67]

1.6 Literature Review: Modelling refrigeration system performance with ANNs

This section provides a literature review focused on the application of artificial neural networks as a modelling tool for refrigeration systems. From this review, key criteria were identified that a refrigeration optimisation model would need to comply with to ensure improved operations, while catering for the harsh underground environment.

1.6.1 Search strategy

The review was conducted using a systematic keyword-based approach to find studies where the design, operating, and performance characteristics of a refrigeration system are used as parameters in an ANN with the goal of predicting a performance measurement.

The studies were found on a variety of academic platforms which included, but were not limited to, Google Scholar, ScienceDirect, Springer, the North-West University electronic library, and IEEE Explore.

The main keywords used for database searches were variations of *deep-level mining, refrigeration, optimisation, and artificial neural network*. The keywords were then repeated with pre- and post-affixes such as *underground, cooling, modelling,* and *performance*.

This was done to diversify the search space and to create a comprehensive list of the body of information relevant to this study. A total of 26 studies were found that used ANNs as the modelling technique for predicting a performance measurement of a refrigeration system with the goal of optimising its energy usage. This literature review considers a narrowed down list of the 12 highest quality studies (discussed later in Table 4). These studies were chosen based on the quality of their method and, specifically, the determination of the architectural and training components.

1.6.2 Critical literature review

A critical review of the three highest quality studies is given in this section. These studies represent the best descriptions of the design process of the ANN model to predict the performance of a refrigeration system. The following aspects of each study are highlighted because of their relevance to the design and accuracy of the model.

- System parameters
    - Type of refrigeration system
    - Inputs and outputs of model
    - Scope of the available data
    - Data allocation
- Development of the model
    - Determination of the number of hidden layers
    - Determination of optimal number of hidden neurons
    - Activation functions used
    - Training algorithms used
    - Performance indicators used to measure prediction accuracy
- Verification of the model
- Contributions to this study
- Shortcomings

Study A: Artificial neural network analysis of a refrigeration system with an evaporative condenser [90]

System:

Ertunc and Hosoz [90] developed an ANN to model a vapour compression refrigeration system employing an evaporative condenser with R134a refrigerant. A multi-layer perceptron model of the refrigeration system was developed. The model was trained to predict the condenser heat rejection rate, refrigerant mass flow rate, compressor power, electric power input to the compressor motor, and coefficient of performance. The inputs to the ANN were evaporator load, air mass flow rate over the condenser, water mass flow rate, air dry bulb and wet bulb temperatures at the condenser inlet.

Development of the model:

Ertunc and Hosoz [90] trained multiple multi-layer perceptrons. The best-performing network was determined using a trial-and-error method by varying the number of hidden neurons and hidden layers. This network is shown in Figure 17 [90]. Sixty data patterns were collected by conducting steady-state experiments on the refrigeration system. Forty-two patterns (70%) were used to train the model and 18 (30%) were used for testing purposes. The activation function used in the hidden layer was the tangent sigmoid function. The Levenberg-Marquardt algorithm was used to train the network. The performance indicators used to assess the accuracy of the model were the root mean square error (RMSE), coefficient of determination (R^2), and mean relative error (MRE).

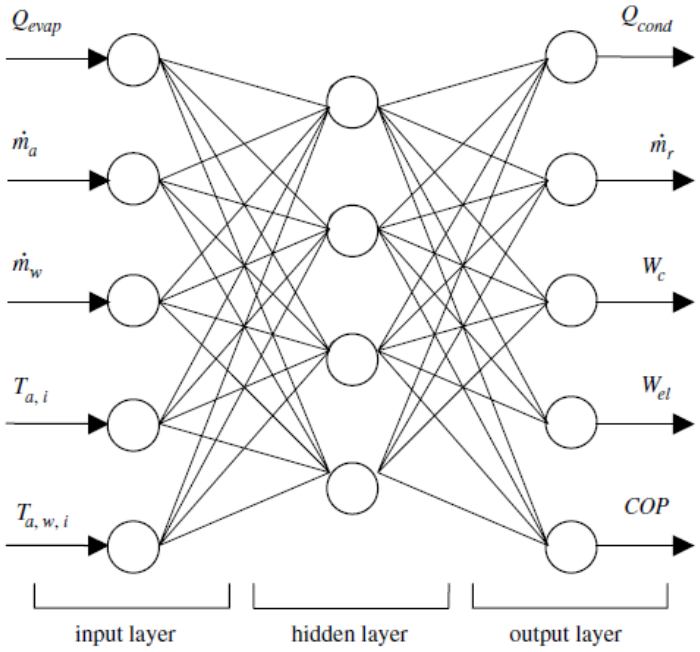


Figure 17: ANN architecture developed to predict the outputs of the refrigeration system with an evaporative condenser (Study A) [90]

Verification of the model:

The best-performing network for predicting the outputs was found to contain four hidden neurons in the hidden layer, trained with the Levenberg-Marquardt algorithm. The model was verified using the test subset. This subset was not involved in the training process to maintain its independence. The model's predictions for each of the outputs were plotted against the unseen outputs. The results are given in Figure 18 [90]. The model was able to generalise well by accurately predicting the outputs of the experiment that are in the independent subset. It can also be seen that the model performed better when predicting the condenser heat rejection rate, refrigerant mass flow rate, and compressor power than it did when predicting the electric power input to the compressor motor and the coefficient of performance.

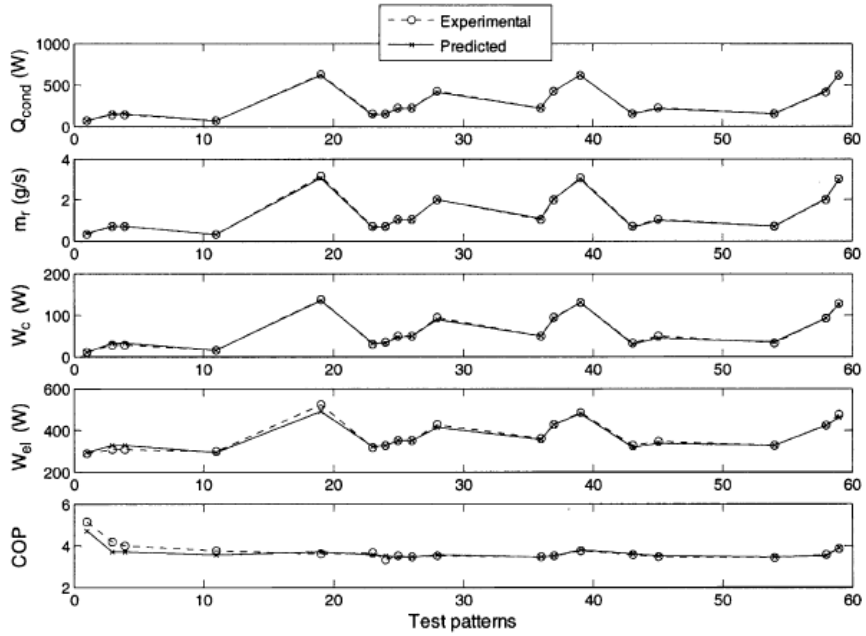


Figure 18: Predictions compared to experimental values of the outputs for the best-performing ANN model of a refrigeration system with an evaporative condenser (Study A) [90]

Contributions to this study:

Study A trained a multi-layer perceptron model of a vapour-compression refrigeration system to predict its performance with a high degree of accuracy. A trial-and-error method was used to determine the optimal configuration of hidden layers and hidden neurons, which broadens the search space for a better model. The Levenberg-Marquardt algorithm was used to train the ANN, which is a training algorithm known for producing accurate models of refrigeration systems generally. A classical activation function was used which has been known to enhance ANN prediction accuracy.

Shortcomings:

Ertunc and Hosoz [90] modelled a refrigeration system with an evaporative condenser using R134a refrigerant. The model was built using a relatively small amount of data obtained from experiments conducted under steady state conditions. It is therefore limited in its scope since the amount of data does not represent the full operational range of that system. A refrigeration plant in operation at a deep-level mine would have to be characterised using a much larger dataset to capture the full operational range. Their study also committed to using only one training algorithm for parameter optimisation instead of testing multiple algorithms to broaden the search for the best-performing ANN model.

Study B: Thermodynamic analysis of variable speed refrigeration system using ANNs [91]

System:

Kizilkan [91] developed an alternative approach to determine the performance of an experimental vapour-compression refrigeration system using R404a as refrigerant. To achieve this, an ANN model was trained to predict the compressor power consumption, refrigerant mass flow rate, experimental COP, theoretical COP, exergetic efficiency, and irreversibility of the system. To predict these outputs, the model received six inputs, namely compressor frequency, cooling load, condenser temperature, evaporator temperature, condenser pressure, and evaporator pressure.

Development of the model:

A multi-layer perceptron neural network was developed to predict the desired outputs. The best-performing network was determined by trial and error. This involved varying the number of hidden neurons between four and eight and using both the Levenberg-Marquardt and SCG training algorithms. The best-performing network is given in Figure 19 [91]. Experiments were conducted on the refrigeration system to collect 80 data patterns; 64 (80%) of the patterns were used to train the model and 16 (20%) were used for testing purposes.

Both the hidden and output layers used the logistic sigmoid activation function. The performance indicators used to assess the best-performing network were the coefficient of determination (R^2), root mean square error (RMSE), mean absolute percentage error (MAPE), and covariance.

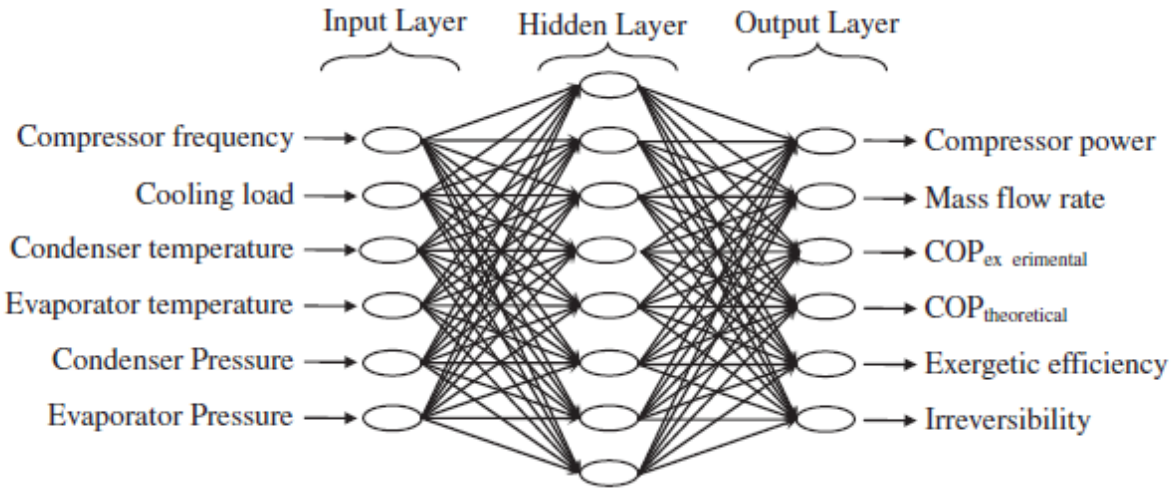


Figure 19: ANN architecture developed to predict the outputs of the refrigeration system with a variable speed compressor (Study B) [91]

Verification of the model:

The topology that yielded the best results for predicting the outputs of the model was a multi-layer perceptron with a single hidden layer containing eight hidden neurons, trained with the Levenberg-Marquardt algorithm. Similarly to Study A, the model was verified using the test subset, which was not involved in the training process to maintain its independence. The model's predictions were plotted against each of the outputs and the correlation coefficient (R) was used to assess the model's accuracy. The results are given in Figure 20 [91]. The model predicted the outputs with a high degree of accuracy and was shown to generalise well across all outputs.

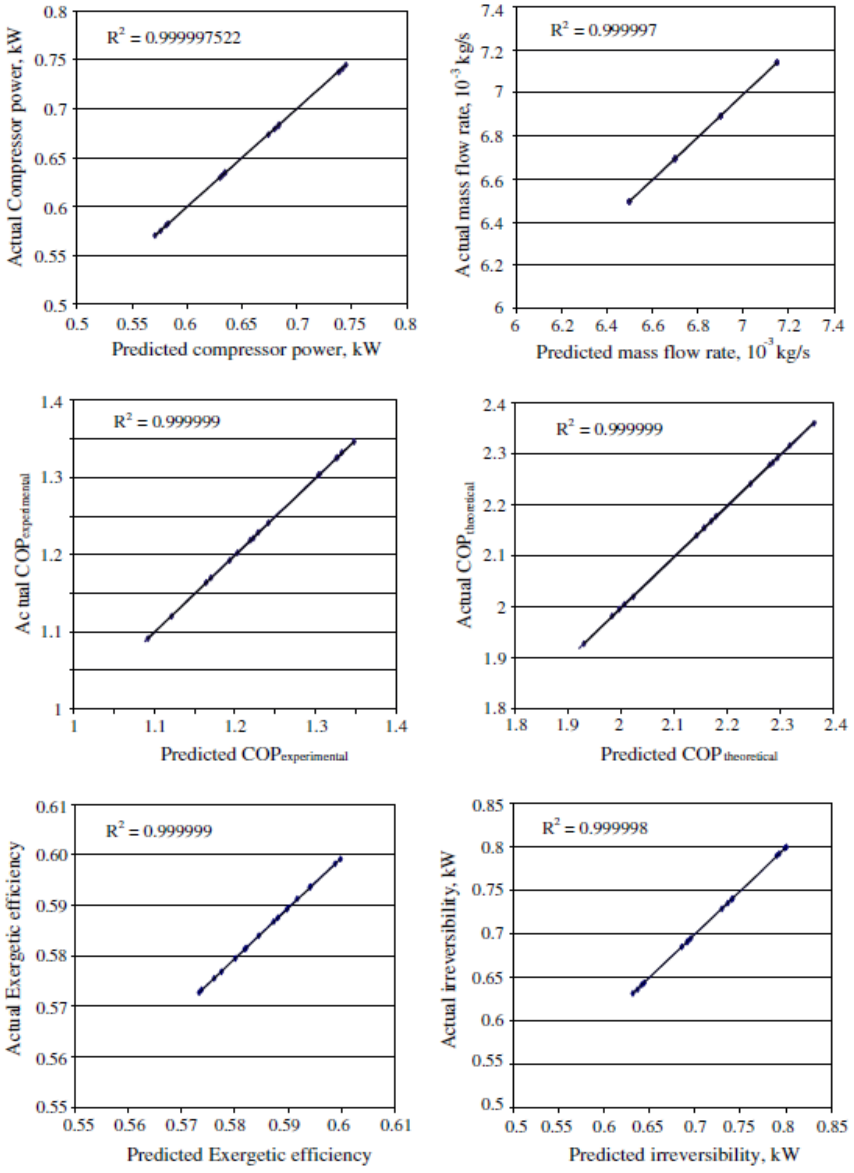


Figure 20: Predictions compared to actual values of the outputs for the best-performing ANN model of a refrigeration system with a variable speed compressor (Study B) [91]

Contributions to this study:

The study by Kizilkan [91] used a multi-layer perceptron ANN to create a model to predict the performance of a vapour-compression refrigeration system with a variable speed compressor. A trial-and-error method was employed to find the best number of hidden layers and hidden neurons. The logistic sigmoid function in the hidden layer is also a classical activation function known for its use in accurate ANN models. Unlike Study A, both the Levenberg-Marquardt and SCG algorithms were used to train the model in order to find the best accuracy.

Shortcomings:

Kizilkan [91], like study A, modelled an experimental refrigeration system and collected relatively small amounts of data from this system. This, again, limits the scope of the model predictions to a small operational range and would not be adequate for a practical system implemented on a deep-level mine. The study also did not consider varying the number of hidden layers. Hidden neurons were varied between four and eight, but this is a small search space, meaning the model has likely not reached its global minimum error.

Study C: Modelling of a cascade refrigeration system using artificial neural network [92]

System:

Hosoz and Ertunc [92] investigated the applicability of predicting performance parameters of a cascade vapour-compression refrigeration system using R134a refrigerant in both the upper and lower circuits. The outputs chosen to characterise the performance of the system were evaporating temperature of the lower circuit, COP of the upper and lower circuits, and compressor power for both circuits. The inputs to the system were evaporator load and water mass flow rate.

Development of the model:

Steady-state experiments were conducted on the cascade refrigeration system to collect a total of 24 input-output pairs. These were split into a subset of 17 patterns (70%) for training and 7 (30%) for validation. A multi-layer perceptron was developed to model the system. The best-performing model was determined by varying the number of hidden layers and neurons in a trial-and-error procedure. The Levenberg-Marquardt algorithm was used to train the neural network, and the tangent sigmoid activation function was used in the hidden layer of the network. The performance indicators used to assess the accuracy of the model predictions were the

RMSE, R^2, and MRE. The architecture of the best-performing network to predict the outputs of the model is shown in Figure 21 [92].

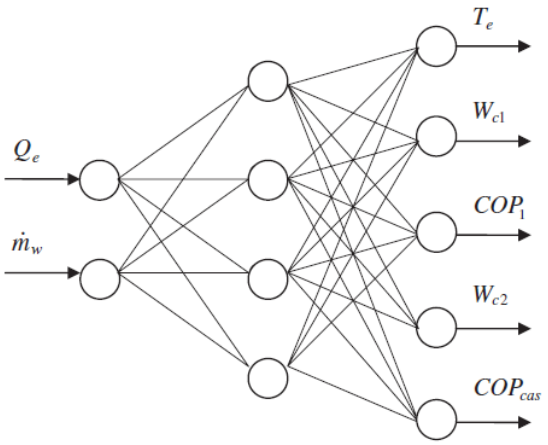


Figure 21: ANN architecture developed to predict the outputs of the refrigeration system with a variable speed compressor (Study C) [92]

Verification of the model:

Using the trial-and-error method, the best network configuration was found to consist of one hidden layer containing four hidden neurons and was trained using the Levenberg-Marquardt training algorithm. The model was verified using the independent dataset to predict the outputs and the coefficient of determination was used to assess the model's performance on the unseen dataset. The predictions were plotted against the subset outputs and are given in Figure **22** [92]. This model, again, generalised well across all outputs but performed slightly worse in predicting the lower circuit COP and compressor power.

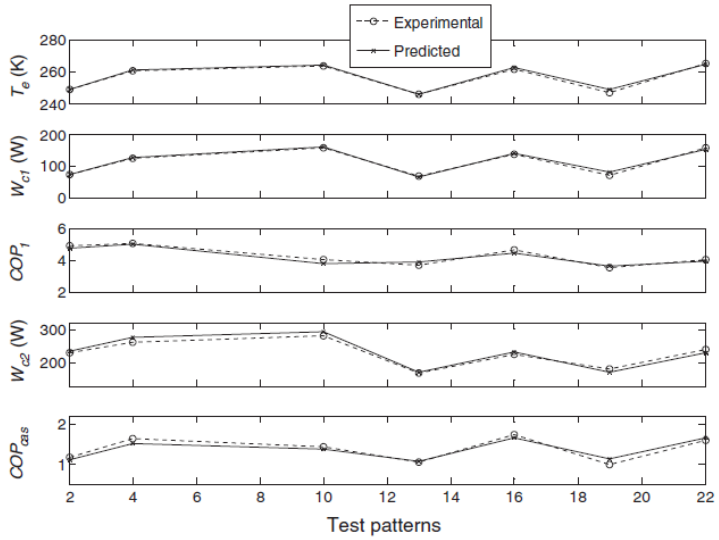


Figure 22: Predictions compared to experimental values of the outputs for the best-performing ANN model of a cascade refrigeration system (Study C) [92]

Contributions to this study:

The best ANN architecture was determined by trial and error by varying the number of hidden layers and hidden neurons. Hosoz *et al*. [92] used the Levenberg-Marquardt training algorithm to train the ANN model. The tangent sigmoid activation function was used in the hidden layer neurons which, again, is a classical activation function known to produce highly accurate ANN models.

Shortcomings:

The model created by Hosoz and Ertunc [92] was trained with small amounts of data gathered from conducting experiments on a cascade refrigeration system using R134a as refrigerant. Similarly to Studies A and B, this limits the model's ability to predict the performance of the system beyond this narrow range. Few inputs were used to characterise the model, meaning that a limited number of options are available if the model is used as the basis for optimisation.

1.6.3 Extended literature review

The literature review is now extended to include the studies that developed ANN models to predict a performance parameter of a refrigeration system but do not provide a detailed description of the method used to find the optimal model architecture. This was done to include alternative approaches to ANN modelling that are not necessarily multi-layer perceptrons.

Şencan [93] developed a model to conduct a performance analysis on an ammonia-water absorption refrigeration system. Ammonia (R717) was the refrigerant and water the absorbent. The goal of the model was to predict the coefficient of performance and circulation ratio. The model took six variables as inputs: generator temperature, evaporator temperature, condenser temperature, absorber temperature, ammonia-poor solution concentration, and ammonia-rich solution concentration.

To predict the COP and circulation ratio, Şencan [93] used a feedforward multi-layer perceptron with a single hidden layer. Seventy-five data patterns were collected and normalised. Sixty of the patterns were allocated for training (80%) and 15 were used for testing (20%). The SCG and Levenberg-Marquardt training algorithms were used, and the number of hidden neurons were varied between 3 and 13 to find the best-performing network architecture. The logistic sigmoid activation function was used in the hidden and output layers of the network. Each model's performance was quantified using the root mean square error (RMSE), correlation coefficient (R^2), and covariance (*cov*).

The network with the best performance for predicting both the COP and circulation ratio of the refrigeration systems contained five hidden neurons and was trained using the Levenberg-Marquardt algorithm [93].

Furumele [94] developed a long short-term memory recurrent neural network to perform time-series forecasting on a deep-level mine cooling system. Five models were created to forecast 30-minutes, 1-hour, 2-hour, 12-hour, and 24-hour windows of the surface and underground temperatures. The models were used as the basis for a dynamic control strategy for BACs and underground fans. The inputs to the model were surface and underground ambient wet- and dry bulb temperature, day of the month, hour of the day, and month of the year.

Since data was collected from a practical system, Furumele [94] applied data pre-processing in the form of outlier detection and removal. Since the model did time-series forecasting, traditional consolidation of data was not possible. The K-nearest neighbour method was thus used to impute the missing datapoints. A sliding window was used to group training examples, normalised, and then data was partitioned into training (80%), validation (5%), and testing (15%) subsets. The number of hidden layers were increased until the model started overfitting, a method of validation subset error tracking. The number of hidden neurons were based on the number of inputs.

The models were evaluated using the RMSE and MAPE performance indicators. It was found that the 12-hour and 24-hour forecast models did not achieve the < 5% threshold for RMSE and the

< 10% threshold for MAPE, indicating alternative methods need to be developed for these time horizons [94].

Belman-Flores *et al.* [95] developed different ANN models to predict COP, compressor power, cooling capacity, water temperature at the condenser outlet, and water-propylene glycol temperature at the evaporator outlet of an experimental vapour-compression refrigeration system. The inputs to these models were compressor rotation speed, volumetric flow rate of water-propylene glycol as well as water, and secondary fluid temperature. 38 071 data points were collected and split between training and validation subsets. The number of hidden neurons were varied between 0 and 15. Early stopping based on the validation subset error was used to determine the number of hidden neurons resulting in the best generalisation capabilities.

The models were pre-trained using simulated annealing and a linear cooling schedule. The Levenberg-Marquardt algorithm was used as the training algorithm. The ANN predicting COP experienced an increase in the validation subset error at 15 hidden neurons, so 14 hidden neurons were selected in the best-performing network. The MSE for the training and validation subsets were 0.0048 and 0.0051, respectively [95].

Şahin [96] modelled a single-stage vapour-compression refrigeration system with three different refrigerants (R134a, R404a, and R407c), using an ANN and a hybrid model of an ANN and fuzzy logic, called an adaptive neuro-fuzzy inference system (ANFIS). Both models had the same inputs, namely cooling capacity, condenser temperature, evaporator temperature, superheating temperature, and subcooling temperature. The output for each was the system COP. The number of hidden neurons was varied between 3 and 13 and the logistic sigmoid activation function was used in the hidden and output layers. The best ANN architecture consisted of seven hidden neurons trained by Levenberg-Marquardt algorithm and outperformed the ANFIS system. The R^2, RMSE and *cov* for R134a was 1.000, 0.010, and 0.002, respectively. For R404a, it was 0.999, 0.046, and 0.009, and for R407c it was 0.999, 0.031, and 0.003.

Navaro-Esbri *et al.* [36] developed a low data requirement model of a refrigeration system with a variable speed compressor. A radial basis function ANN was trained using chilled water- and condensing water inlet temperatures, refrigerant outlet temperature, and compressor rotation speed as inputs. The model predicted cooling capacity, electrical power consumption, and chilled water outlet temperature. Data was collected by conducting eight steady state experiments and collecting 500 measurements from each. Twenty data patterns were sampled randomly for training. Of the data, 75% was allocated to the training subset and 25% to the test subset. The

RMSE for cooling capacity, electrical power consumption, and chilled water outlet temperature for the test subsets were 0.0884, 0.1337, and 0.0557, respectively.

Gill *et al.* [97] developed an ANN model to conduct energy analysis on a domestic refrigeration system. The goal of the analysis was to determine the energy efficiency of liquid petroleum gas (LPG) refrigerants used in conjunction with TiO_2-based nanoparticle lubricants compared to the traditional R134a refrigerant. Three multi-layer perceptrons were trained to predict the compressor power, cooling capacity, and COP, respectively. The inputs to the models were the condenser and evaporator temperatures, amount of LPG charge, and nanoparticle concentration. The best-performing network had nine hidden neurons. The networks were trained with a hybrid of the conjugate gradient algorithm and simulated annealing. The RMSE, MAPE, and R^2-ranges for the outputs were 0.111–2.317, 0.865–3.148%, and 0.914–0.970, respectively.

Reddy *et al*. [98] conducted a comparative study of multiple regression and ANN techniques for a domestic refrigeration system using hydrocarbon refrigerant mixtures. The models took as inputs the refrigerant charge mass, evaporator temperature, and the capillary tube length. Each model had to predict the refrigeration effect, power consumption of the system, and COP. Data was collected from 27 experimental runs. Nineteen (70%) of the data patterns were used for training and the remaining eight (30%) were used for validation. The ANN model was trained using the Levenberg-Marquardt algorithm and used the logistic sigmoid activation function in its hidden neurons. The hidden layer contained 22 neurons; a decision based on literature. The ANN model achieved R^2-values of 0.998, 0.993, and 0.991 for the predictions of refrigeration effect, power consumption, and COP, respectively.

Gill *et al.* [99] created an ANN and adaptive neuro-fuzzy inference system (ANFIS) model to conduct irreversibility analysis on a vapour compression refrigeration system using a R134a/LPG blend as a replacement for the R134a refrigerant. The inputs used were condenser temperature and evaporator temperature and separate models were trained to predict the total irreversibility and second law efficiency of the system, respectively. Experiments were performed to collect training data for the models. Leave-one-out cross validation was used to train the models because of a lack of data to split into the traditional training and validation subsets. The ANN model was trained using a hybrid of simulated annealing and the conjugate gradient algorithm. The best network for each output was found to contain 10 hidden neurons. The ANN achieved accuracy ranges for the RMSE of 0.164 – 0.234, 0.159 – 0.572% for MAPE, and R^2-values of 0.980 – 0.994 for the outputs.

Hosoz *et al.* [37] developed an ANFIS model to predict the performance of a vapour compression refrigeration system employing a counter-flow cooling tower. The model had to predict eight outputs, including coefficient of performance. To do this, five inputs were used: evaporator load, dry bulb temperature, relative humidity, air mass flow rate through the cooling tower, and water mass flow rate. A total of 64 steady-state experiments were run on the system to collect data for the model. Of these, 48 (75%) data patterns were allocated to train the model and 16 (25%) were used for validation purposes. The model was trained using a hybrid of the least squares method and backpropagation gradient descent. The model's predictions for COP scored a RMSE of 0.21, MRE of 4.94%, and R^2 of 0.9958.

1.7 State-of-the-art summary

This section presents the state-of-the-art summary surrounding the topic of this study. Table **4** summarises the research conducted concerning the use of ANNs to model refrigeration system performance. Each column of the table represents commonalities between the reviewed studies relevant to the development of the model that are required to solve the problem. Each row represents a research paper discussed in the previous section. Green indicates that the research paper covered the aspect, while red indicates a gap in the research. The aspects of the table act as evaluation criteria and are used as points of comparison between studies. The evaluation criteria are as follows:

- **Application** – **Experimental, Practical**: Was the refrigeration system part of an experiment or practically applied in industry? This is important because the collection and handling of data in a practical system has a significant effect on the ability of an ANN model to predict the performance of the system in various operational conditions.
- **Sensitivity analysis** – **Number of hidden layers, hidden neurons, training algorithm**: Was sensitivity analysis employed to determine the optimal neural network architecture? This is evaluated in terms of the number of hidden neurons, hidden layers, and training algorithms. This is important since these are the essential factors to test for when developing an accurate model.
- **Data pre-processing** – **Cross-validation, filtering**: Was the data put through pre-processing to ensure data quality and valid determination of the model accuracy? This is important, especially when working with practically implemented systems, since poor data quality is expected. Cross-validation is important to ensure the accuracy of the model is validated by a separate dataset than the one used for training.

- **Verification** – **Independent dataset**: Was an independent dataset used to verify the model results? This is important because the generalisation capabilities of the model cannot be assessed accurately without testing it on unseen, independent data.

Table 4: State-of-the-art literature summary for the development of ANN models of refrigeration systems

Article	Application		Sensitivity analysis			Data pre-processing		Verification	Comments
	Experimental	Practical	Number of hidden layers	Number of hidden neurons	Training algorithm	Cross-validation	Filtering	Independent dataset	
[93]									Single-stage vapour-compression refrigeration plant.
[87]									Refrigeration system with an evaporative condenser.
[88]									Variable speed refrigeration system.
[90]									Ammonia-water absorption refrigeration system.
[92]									Vapour compression system with internal heat exchanger.
[33]									Variable speed vapour compression system.
[34]									Refrigeration system with a counter-flow cooling tower.
[96]									Domestic refrigerator with alternative refrigerants.
[94]									Vapour-compression refrigeration system.
[95]									Domestic vapour-compression refrigeration system.
[89]									Cascade refrigeration system.
[91]									Cooling system auxilliaries

1.8 Need, aim and objectives

1.8.1 Need for the study

Simulation is the predominant method used in the mining industry to model refrigeration systems, but can be difficult and time-consuming to develop. ANNs are the most common non-traditional technique used to model refrigeration systems and the multi-layer perceptron specifically shows potential due to its robustness. From the state-of-the-art summary given in Table **4** and the preceding literature review, it can be seen that a need exists to develop a method to accurately model a practically implemented refrigeration system using a multi-layer perceptron. No model currently exists that relates the operational parameters of a deep-level mine to its performance using multi-layer perceptron theory. To find the optimal model, the optimal configuration of the hidden layers, hidden neurons, and training algorithm must be found based on the accuracy of its predictions.

1.8.2 Aim

The aim of this study was to develop a new method that uses the multi-layer perceptron theory to create a model of a vapour-compression refrigeration plant on a South African deep-level mine that accounts for changing operational conditions.

1.8.3 Objectives

To achieve the aim, the objectives of this study were:

1. Select appropriate input and output parameters for a refrigeration system model.
2. Apply pre-processing to these parameters to improve data quality.
3. Design and evaluate the architecture of an artificial neural network to optimise the:
    - Number of hidden layers,
    - Number of hidden neurons, and
    - Training algorithm.
4. Describe and implement the best-performing model by:
    - Deriving mathematical equations that describe the relationships between the input and output variables, and
    - Using the model's predictions to improve performance of the target parameter,
    - Evaluate the potential financial benefit of using the improved operating parameters
5. Verifying the model output by implementing it on unseen data to evaluate its generalisation capabilities

1.9 Summary

The cost of electricity has increased drastically over the last 16 years in South Africa. This has had a negative impact on the profitability of energy-intensive industries in the country, including the mining sector. Mining operations have largely focused on improving the energy efficiency of their largest energy systems to improve worker productivity and to reduce wasteful expenditure. Cooling systems are responsible for a large proportion of the energy consumption of deep-level mines and are heavily reliant on the chilled water produced at a central refrigeration plant to perform their cooling functions.

An improvement in the performance of a central refrigeration plant leads to improved cooling throughout the mine and reduces electricity cost. A characteristic model of a refrigeration system is required to accurately represent its response under various conditions over the entirety of its operational range. An MLP was determined to be the most adequate technique for developing such a model for a practically implemented refrigeration system.

An overview of MLP modelling was provided before a critical literature review of three studies was conducted. This review affirmed ANN's use as a modelling tool for refrigeration systems. It was found that MLP models can accurately predict the performance of these systems. It was also found that the prediction accuracy of the model varies depending on the number of hidden neurons, the training algorithm used, and the number of hidden layers employed.

To complete the state-of-the-art review surrounding the modelling of refrigeration systems using ANNs, a thorough literature review of nine most relevant studies was conducted. This review showed that a need exists to develop a method to find the optimal configuration of hidden layers, hidden neurons, and training algorithm of an ANN model of a deep-level mine refrigeration system based on the accuracy of its predictions. Following this, the aim and objectives of the study were formulated.

1.10 Research methodology

This study was conducted using a hybrid approach between the case study methodology [100], [101], and quantitative modelling [101].

The quantitative aspect was used as the basis for constructing the method to develop, implement, and validate the model. The modelling process itself followed this methodology with respect to the definition of the model (in terms of the relationship between inputs and outputs), data

collection, development of the model, model validation, and the interpretation of the model predictions.

The case study methodology influences the method used to develop the model in the sense that the method must consider the real-world context within which the refrigeration system exists. The method must be able to compensate for the complexities due to this context and the interpretation of the model prediction must take these factors into consideration as well. Figure 23 shows the steps of the methodology followed in this study.

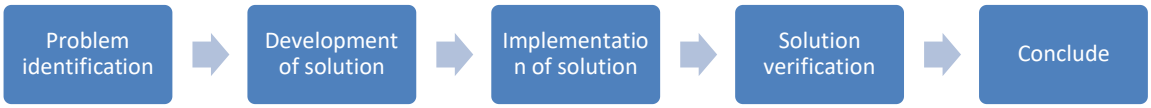


Figure 23: Flow diagram of the methodology followed in this study (adapted from [100], [101])

1.11 Outline of the document

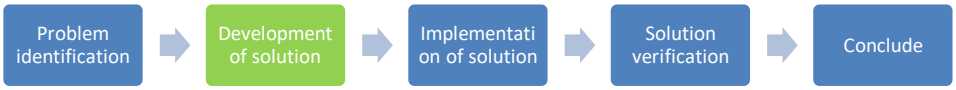
Chapter 1: Introduction – This chapter provides background information relevant to the problem this study aims to address, specifically energy usage and refrigeration systems on deep-level mines. Following this, an introduction to artificial neural networks is given. A critical literature review was conducted, followed by an extended literature review to complete the state-of-the-art summary surrounding ANN modelling of refrigeration systems. Finally, the problem statement, aim, and objectives of the study are given.

Chapter 2: Method – This chapter describes the method which can be used to model a deep-level mine refrigeration system using an MLP. This includes the collection and handling of data, development and implementation of the model, and finally the verification of the model.

Chapter 3: Implementation and Results – This chapter implements the method presented in Chapter 2. The result of each step is presented and discussed. The determination of the best-performing network and its architectural detail is discussed. The model arising from the best-performing network was implemented and evaluated to determine the optimal operating conditions for providing the best performance of the case study refrigeration system. Finally, the model was verified by applying it to a set of unseen data to determine its generalisation.

Chapter 4: Conclusion – This chapter concludes the study by summarising its findings. The shortcomings of the study are communicated along with recommendations for further work.

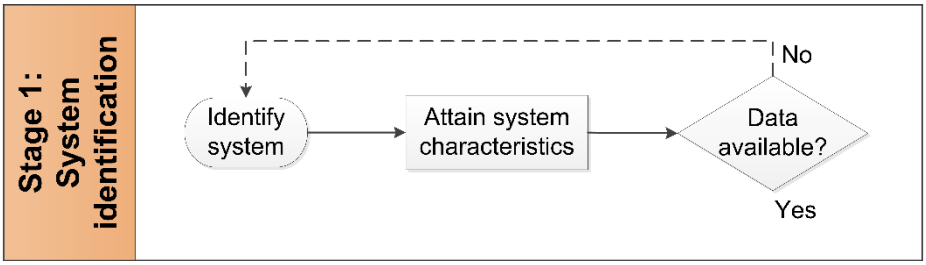
CHAPTER 2 METHOD



2.1 Preamble

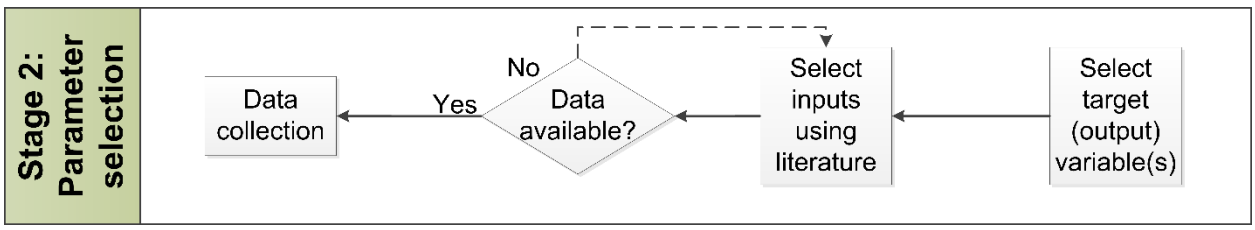
This chapter presents the proposed method to create an artificial neural network model of a deep-level mine refrigeration system to predict its performance. The method consists of five stages, namely: system identification, parameter selection, data pre-processing, model development, and model evaluation and verification.

2.2 Stage 1: System identification



To accurately predict the performance of a refrigeration system, a model must be created that accurately represents a deep-level mine refrigeration system. The model should also be able to serve as the foundation for any optimisation technique applied to it. This requires an understanding of the thermodynamic characteristics, mechanical components, and the external environment [29]. The system investigation should therefore reveal the parameters available to the model.

2.3 Stage 2: Parameter selection



To develop an effective model, the right parameters must be selected to adequately represent the system. The outputs of the model must provide a suitable measure of the holistic performance of the refrigeration system, as discussed in Section 1.3. The inputs must, in turn, have a causal effect on the performance measures captured with the chosen output(s). For the results of the

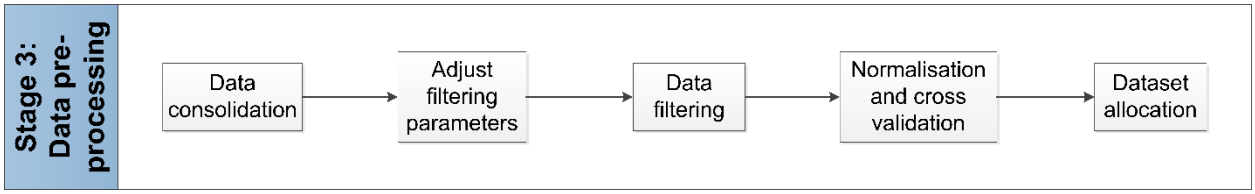
model to have practical applicability on the refrigeration system, some of the inputs must be controllable [30].

The decision of which parameters to select are determined by two factors, namely:

- Literature: Does literature provide insight into which parameters have contributed to successful artificial neural network models for similar systems?
- Availability and reliability: Is there sufficient reliable data available at the refrigeration plant to use the parameter as input?

As discussed in Section 1.5.5, larger datasets are considered better when developing ANN models since they lead to better generalisation capabilities and reduced overfitting [79]. Moreover, data collected from practically implemented systems are expected to contain large amounts of noise. For this reason, it is beneficial to have a larger dataset to identify the general trend more easily [67]. It is also important to ensure that the entirety of the operational range of the refrigeration system is contained within the data collected to improve the robustness of the model.

2.4 Stage 3: Data pre-processing



Data from practically implemented systems are inherently noisy [39]. This may be due to periods of power outages, maintenance, and sensor failures. The raw data must consequently undergo pre-processing before it can be used to train the ANN model.

The data must be consolidated to ensure causality between input and output data pairs and filtered to remove outliers, noise, and periods where the refrigeration plant was inactive. Thereafter, the data must be normalised to account for variation in the orders of magnitude of different parameters. Lastly, cross validation must be done to allocate data to training, validation, and test subsets.

2.4.1 Consolidation

The first step in the data pre-processing procedure is to perform data consolidation. The objective of this step is to create continuous input-output vector pairs. Raw data contains periods of data loss. If this data was used to train an ANN, the network would learn non-causal relationships

between input and outputs parameters. It is therefore important to ensure that the operational conditions represented by each input vector has a causal relationship with the operational conditions represented by its output vector pair. Consolidation is performed by finding the datetime ranges where data loss took place for any parameter. These ranges represent data that cannot be used to develop the model. Data contained within these ranges are then removed from the dataset.

2.4.2 Filtering

Raw data contains noise, outliers, and transient effects, which represent inaccuracies stemming mostly from instrumentation errors. Applying inequality, noise spike, and exponentially weighted moving average filters help to smoothen data. This is a necessary step for the model to accurately represent the operation of the refrigeration system without inadvertently learning these inaccuracies. The combination of these filters has been shown to be effective for filtering data from energy systems used in the development of ANNs [45].

Inequality filter

Refrigeration plants are sometimes switched off for maintenance or when cooling demand is low. During these periods, data is still sampled and is therefore contained within the dataset. These periods are not applicable and negatively impact the model's ability to quantify the interactions between inputs and outputs and consequently have to be removed. Non-operational data can be filtered out by removing all data for when the compressor is off. This can be achieved using an inequality filter on the compressor power value (P_C), governed by Equation (24):

$$P_C \leq 0\ kW \tag{24}$$

Noise spike filter

Noisy datasets contain anomalous fluctuations. These fluctuations are characterised by sudden changes in the value of the data and are caused by faulty electrical signals in a sensor. This phenomenon is called a noise spike and is shown graphically in Figure 24 [102].

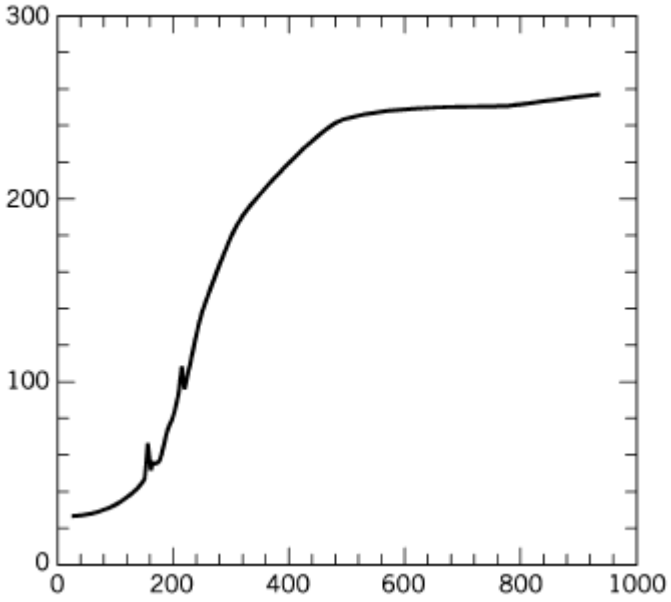


Figure 24: Graphical example of the noise spike phenomenon on a sequential dataset [102]

Noise spikes are inaccurate representations of the underlying measurement and are undesirable. To lessen the effect of noise spikes, the noise spike filter, as presented by Seborg *et al.* [102] can be used. This method limits the amplitude of fluctuations based on the values preceding and succeeding a data sample. The expression governing the noise spike filter is given by:

$$y_F(k) = \begin{cases} y_m(k), & if\ [y_m(k) - y_F(k-1)] \leq \Delta y \\ y_F(k-1) - \Delta y, & if\ [y_F(k-1) - y_m(k)] > \Delta y \\ y_F(k-1) + \Delta y, & if\ [y_m(k) - y_F(k-1)] > \Delta y \end{cases} \quad (25)$$

The filtered value (y_F) for a given instance k is used to replace the measured value (y_m) if the maximum allowable change (Δy) is exceeded. If the measured value adheres to the maximum allowable change, no change is made. If the maximum allowable change is exceeded, the filtered value is used and is equal to the previous filtered value plus (or minus, depending on the direction of the spike) the maximum allowable change. The *medfilt1* function in MATLAB can be used to implement the noise spike filter.

Exponentially weighted moving average

The exponentially weighted moving average (EWMA) filter smoothens data based on a weighted average of a determined number of preceding samples. The number of samples that contribute

to the EWMA is called the memory. The older the sample is, the less weight is given to it. The EWMA filter presented by Hunter [103] is used and is given by:

$$y_F(k) = \lambda y_m(k) + (1 - \lambda) y_F(k - 1) \tag{26}$$

The depth of memory is determined by the λ-parameter and is given by:

$$\lambda = 1 - \frac{\tau_F}{\tau_F + \Delta t} \tag{27}$$

Specifically, the λ-parameter sets the rate of decay of the weights. Therefore, when $\lambda \to 1$, the filtered value approaches the measured value. Δt represents the time difference between two sampling instances, while τ_F is a time constant that can be adjusted to find the desired degree of smoothing.

Normalisation

Each parameter of the dataset describes a physical property of the refrigeration system, including temperatures, flow rates, and power. These parameter values often differ by orders of magnitude, which negatively impacts the model's prediction accuracy and increases training time [104]. To account for this, normalisation is used. The *min-max* normalisation method was chosen based on its successful implementation on a similar ANN developed by Kizilkan [91]. The equation used for *min-max* normalisation is given by:

$$\widehat{x_i} = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \tag{28}$$

where x_i is the sample, $\hat{x}_i$ is the normalised value of the sample, x_{min} is the global minimum value of the variable and x_{max} is the global maximum value of the variable. This method normalises values in the range [0,1].

2.4.3 Cross-validation

Cross-validation is a method to randomly distribute a dataset into two separate classes: one for training, calibration, and implementation (TCI), and the other for verification [105]. The purpose of this is to evaluate and verify a model's predictions on datasets which are independent of the

data used to train the model. Evaluation and verification on these datasets yield a more realistic view of the model's predictive capabilities [105]. In the context of ANNs, cross-validation is used to evaluate the generalisation capabilities of the model.

For the development of this model, 90% of data was allocated to the TCI class. The remaining 10% was allocated to the verification class. The distribution ratio used was based on studies concerned with the development of a predictive ANN model [45], [106].

2.4.4 Dataset allocation

The TCI class data is further distributed into a training, validation, and testing subset as discussed in Section 1.5.5. The data in the verification class is allocated entirely to a verification subset. The proportion of data allocated to the training-, validation-, testing-, and verification subsets are 63%, 18%, 9%, and 10%, with respect to the main data set, respectively. The verification subset data is left out of training, calibration, and implementation to maintain its independence so that it can be used during the verification process. A schematic representation of the entire data pre-processing procedure is given in Figure 25.

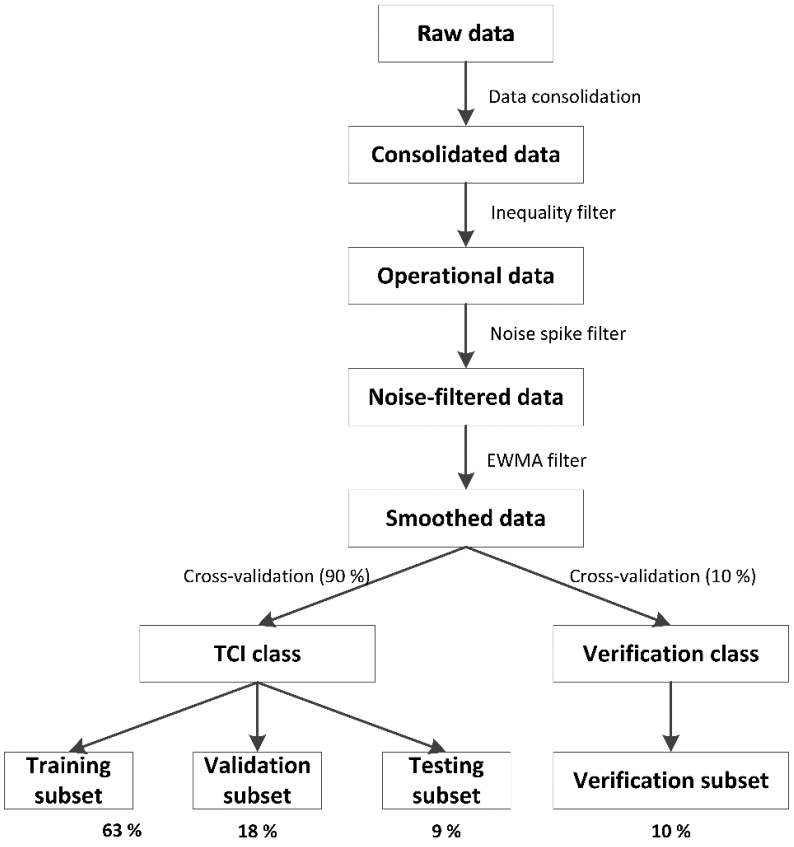
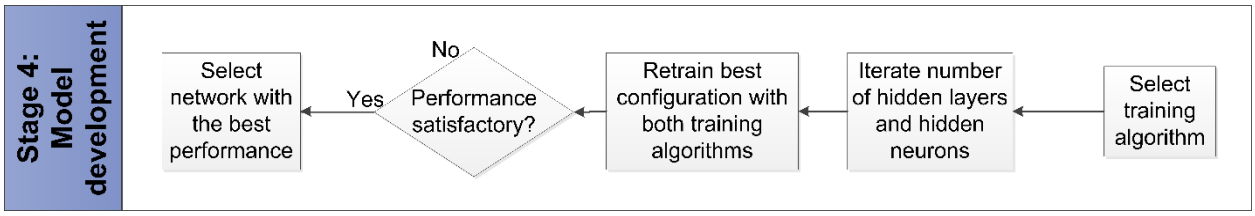


Figure 25: Schematic representation of the flow of data during pre-processing

2.5 Stage 4: Model development



This section details the development of the model to predict the performance of a deep-level mine refrigeration plant. A special case of feedforward ANNs, the multi-layer perceptron, was used to develop the model. A schematic representation of the architecture of a generic multi-layer perceptron with one hidden layer is given in Figure 26.

The method used to develop the model is referred to as sensitivity analysis or the exhaustive search method. It aims to find the best-performing model architecture by training and subsequently comparing multiple ANNs with different configurations. ANNs with up to two hidden layers are considered. The number of hidden neurons in each layer is varied and different training algorithms are used. The best-performing model is then determined by evaluating its generalisation capabilities using performance indicators commonly used in literature.

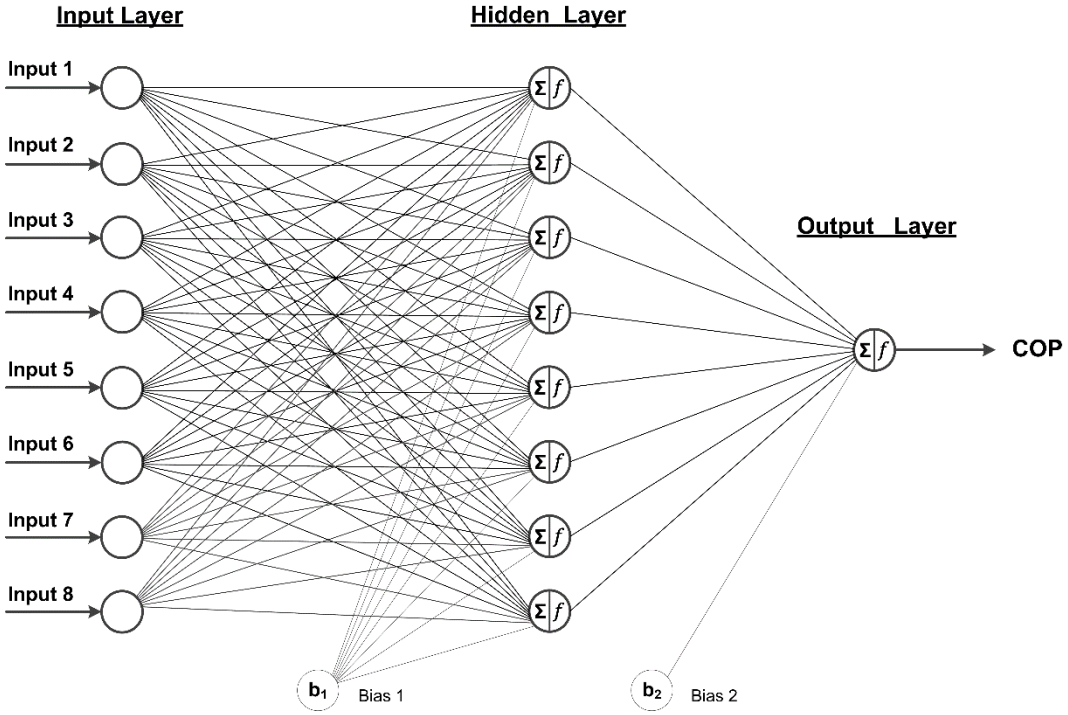


Figure 26: Schematic representation of a generic ANN topology

2.5.1 Hidden neurons

The purpose of determining the optimal number of hidden neurons is to avoid the problems of over- and underfitting to determine the network configuration with the best generalisation capabilities. In the *Architectural setup considerations: Number of hidden neurons* section (Page 32), several theories were mentioned that consider the problem of finding the optimal number of hidden neurons in the hidden layer(s) of an ANN.

To determine the optimal number of hidden neurons for this model, an empirical method was used called the validation subset error stopping condition. This method involves training multiple networks with a varying number of hidden neurons and selecting the network with the best generalisation capabilities based on the minimum error achieved on the validation subset as illustrated in Figure 16 (Page 32).

The performance of each configuration was determined by two performance indicators selected from a similar study conducted by Kizilkan [91] which was covered in the *Critical literature review* section (Page 38). These are the coefficient of determination (R^2), the root mean square error (RMSE), and the mean absolute percentage error (MAPE). The values of R^2, RMSE, and MAPE are given by the following equations:

$$R^2 = 1 - \left(\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(\hat{y}_i)^2}\right), \tag{29}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{30}$$

$$MAPE = \frac{1}{N}\sum_{n=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i} \times 100, \tag{31}$$

where y_i is the target output and $\hat{y}_i$ is the predicted output of the model. These performance indicators are applied to ANNs trained using batch learning techniques; consequently, their values are calculated over the entire training subset from datapoint $i = 1$ to $i = N$.

The R^2-value falls in the range of 0 and 1, it indicates the proportion of the variance in the dependent variable that can be explained by the independent variables. A higher value indicates that the model is a better fit and is therefore desired [98]. The RMSE value measures the error of the predicted output relative to its target [98]. A lower RMSE value is therefore desired.

Using the RMSE and R^2 performance indicators, the optimal number of hidden neurons can be determined. In the case of a single hidden layer, ANNs are trained where the number of hidden neurons is varied between 1 and 200. This number is chosen for the sake of rigour and to allow for the emergence of a trend of weaker generalisation to better identify the optimal point. The optimal point can be identified as the point of minimum error (RMSE) and maximum correlation (R^2) on the validation subset. An illustration of the expected process is given in Figure 27 [45].

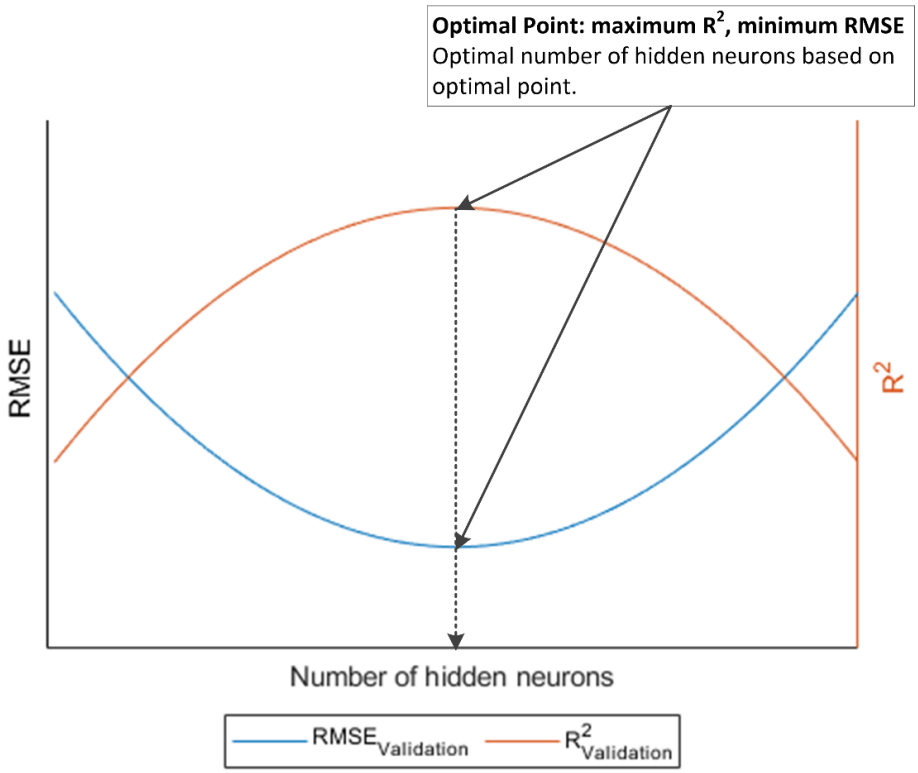


Figure 27: Graphical illustration of the plot used to determine the optimal number of hidden neurons in a single hidden layer ANN (adapted from [45])

For two hidden layers, the RMSE and R^2 performance indicators are again used to determine the optimal number of hidden neurons in each layer. For two hidden layers, the number of hidden neurons in both hidden layers are varied between 1 and 40. This is a practical consideration to avoid excessive training times and computational cost. The optimal number of hidden neurons in each hidden layer is again determined by finding the minimum of the RMSE and maximum R^2 based on the validation subset.

To illustrate this for two hidden layers, separate plots were used for the RMSE and R^2 indicators for the sake of clarity. On each plot, the number of hidden neurons (HNs) in the first hidden layer

(HL) is plotted on the x-axis and the number of hidden neurons in the second hidden layer is plotted on the y-axis. The performance indicators are then plotted on the z-axis. This process is illustrated in Figure 28 and Figure 29.

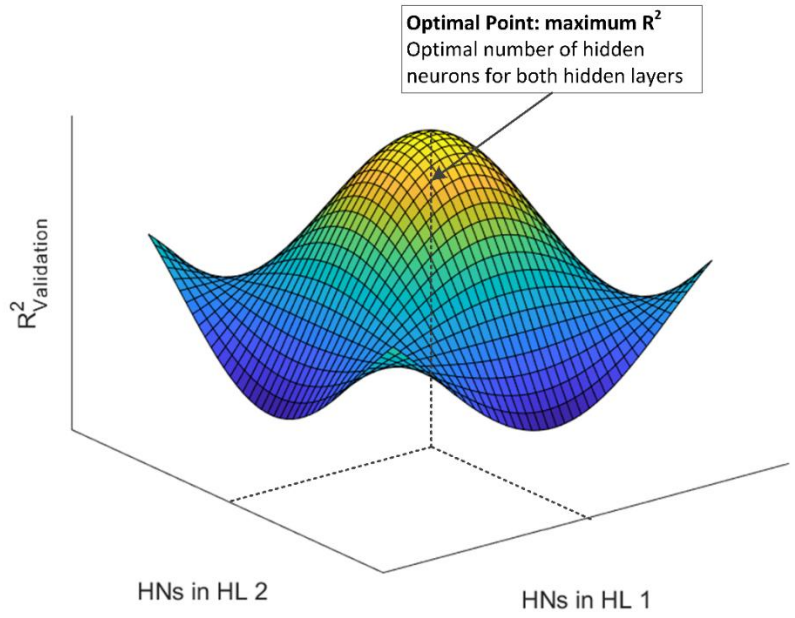


Figure 28: Example graph of the process used to determine the optimal number of hidden neurons in a two hidden layer network based on the R^2-value

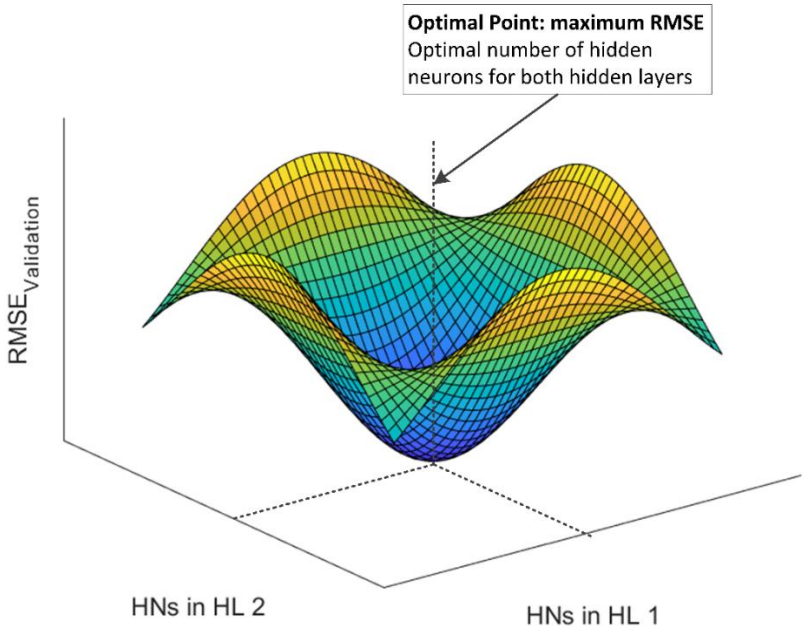


Figure 29: Example graph of the process used to determine the optimal number of hidden neurons in a two hidden layer network based on the RMSE value

2.5.2 Training algorithm

To develop the model, two training algorithms are considered: the Levenberg-Marquardt algorithm and the scaled conjugate gradient algorithm. These algorithms are discussed in detail in Section *1.5.4 Parameter optimisation algorithms*. The use of the LM and SCG algorithms was chosen based on their popularity among researchers developing models of refrigeration systems, as shown in the literature review conducted in Section 1.6.2.

To find the best suited training algorithm for ANNs with both one and two hidden layers containing varying numbers of hidden neurons, every iteration of ANN is trained using the LM and SCG algorithms to compare their accuracy. To avoid stability and convergence issues, discussed in the *Training considerations – Weight initialisation* section (Page 31), weights are randomised at initialisation. Additionally, the best ANNs for one- and to hidden layer configurations are retrained multiple times to account for any biases that randomisation of weights may introduce.

Each algorithm employs a distinct method to adjust the weights and biases of the network to reduce the error of a cost function. Differences therefore emerge in the accuracy of their predictions and the speed of convergence. To compare their performance, early stopping can be used as presented in the *Training considerations – Early stopping* section (Page 31). The mean squared error (MSE) is used as the performance indicator as given by the following equation:

$$MSE = (RMSE)^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{32}$$

where y_i is the target output and $\hat{y}_i$ is the predicted output for data points i to N. After each training repetition of the best networks for one- and two hidden layer networks, the networks' MSE is calculated and plotted for the training, validation, and testing subsets. At the minimum MSE of the validation subset, the combination of weights and biases determined by the training algorithm represents the best generalised performance of the network. Before reaching this ideal point, the network underfits the validation subset data. Conversely, after this point the network starts overfitting the validation subset data. This process is illustrated in Figure 30 [67].

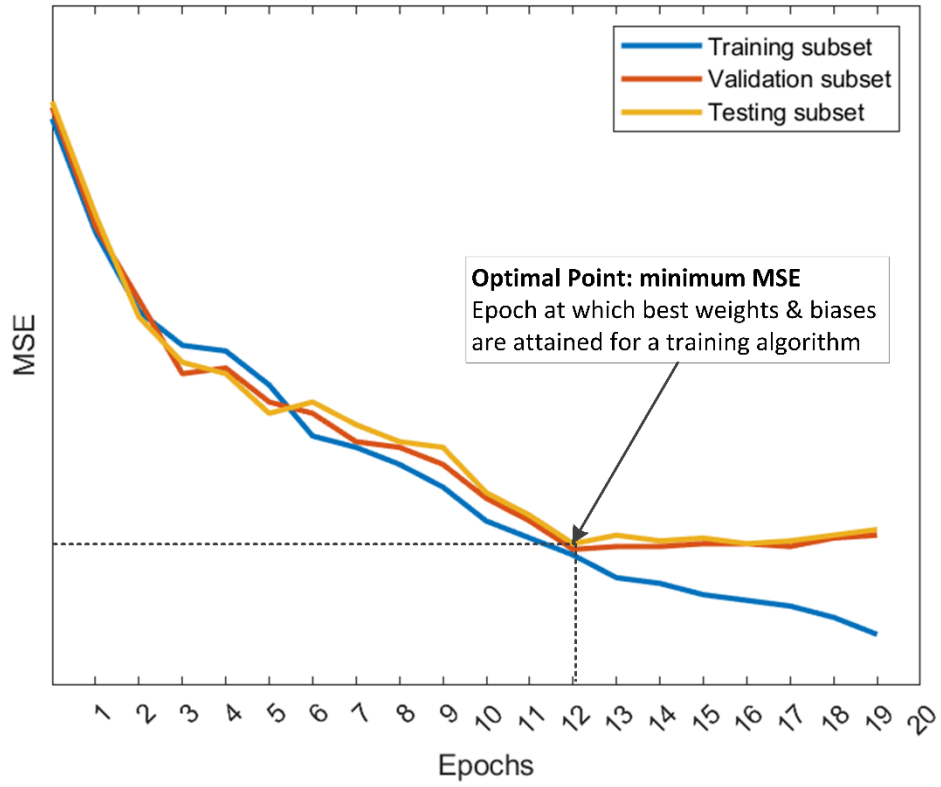


Figure 30: Illustration of the process used to determine the point where a training algorithm provides the best combination of weights and biases in the ANN (adapted from [67])

The global minima of the one- and two hidden layer ANN configurations are determined by choosing the lowest MSE over the set of local minima calculated over all repetitions. The best networks for each configuration are repetitively trained to ensure that the best combination of training algorithm, number of hidden layers, and number of hidden neurons is found.

2.5.3 Hidden layers

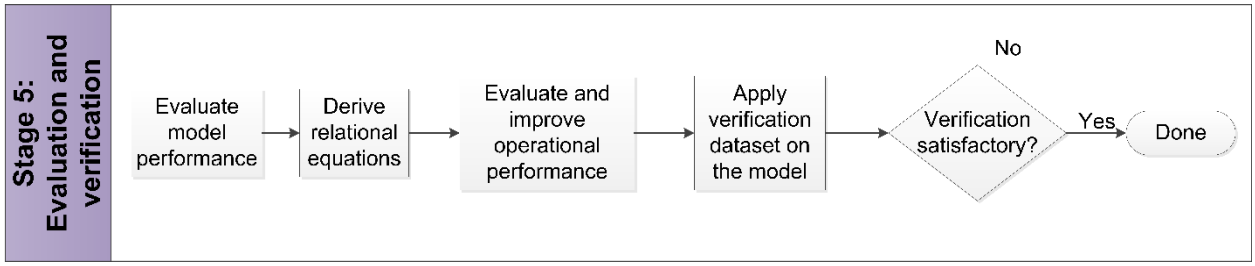
As discussed in the *Architectural setup considerations: Number of hidden layers* section (Page 36), the effect of the number of hidden layers on multi-layer perceptron performance is well-understood. A single hidden layer is sufficient to approximate any continuous functions to an arbitrary accuracy [84], [85]. Two hidden layers have been shown to be beneficial when modelling highly non-linear or discontinuous functions [81]. Although a single hidden layer is theoretically sufficient to model any continuous function, in practice, two hidden layers have been used to speed up convergence [67].

Rarely has it been found that performance is improved when using more than two hidden layers and can, in fact, exacerbate the vanishing- and exploding gradient problems which negatively impact ANN prediction accuracy [86]. ANNs were consequently limited to two hidden layers during the development of the model.

An important consideration regarding hidden layers is the choice of activation function for its hidden neurons. As mentioned in Section 1.5.2: *The multi-layer perceptron* (Page 18), the advantage of adding hidden layers is only realised if a nonlinear activation function is used. The logistic sigmoid activation function was chosen based on this criterion and its popularity in literature [45], [91]. Note that the tangent sigmoid activation function could also be used and is a popular choice amongst developers of ANNs.

During the development of the model, the best-performing network can be determined based on the prediction accuracy of the network. By varying the number of hidden layers, the number of hidden neurons in those layers, and then repeatedly training the best version of each variation with the two training algorithms, the optimal ANN can be chosen. If the accuracy of the networks was unsatisfactory, filtering parameters were incrementally changed, and the steps discussed in this section repeated until deemed satisfactory.

2.6 Stage 5: Model implementation



The best-performing model was trained and evaluated on data from the training and validation subsets. To implement the model to achieve the objectives of this study, its performance was characterised using these subsets as well as the testing subset. Mathematical equations were derived to describe the fundamental input-output relationship between the chosen variables. Finally, the model was used to improve the performance of the refrigeration system by identifying the optimal operating conditions of the refrigeration system based on its effect on the output variable.

2.6.1 Prediction evaluation of the model

To evaluate the prediction capabilities of the model, appropriate performance metrics must be chosen. From the literature review conducted in Sections 1.6.2 and 1.6.3 it was found that the R^2, RMSE, and the mean absolute percentage error (MAPE) were the most used performance indicators for refrigeration system models. These metrics are also generally considered the most popular indicators of ANN performance [45]. Henceforth, these metrics were used to evaluate the performance of the best-performing model on each of the subsets of the TCI class.

The RMSE emphasises the impact of larger errors while MAPE is a more balanced measure, placing no emphasis on larger errors [107]. The R^2-value, as mentioned in Section 2.5.1, indicates how much of the variance in the target variable can be explained by the inputs, given as a value between 0 and 1. A regression plot was used to indicate the correlation between the model's predictions compared to its targets as shown in Figure 31. The predictions made by the model, given by the outlined dots, was plotted relative to the ideal fit given by the dashed line. The blue line represents the equation of best fit of the model predictions. This plot was used to give a visual indication of the model's prediction accuracy over the entire operational range.

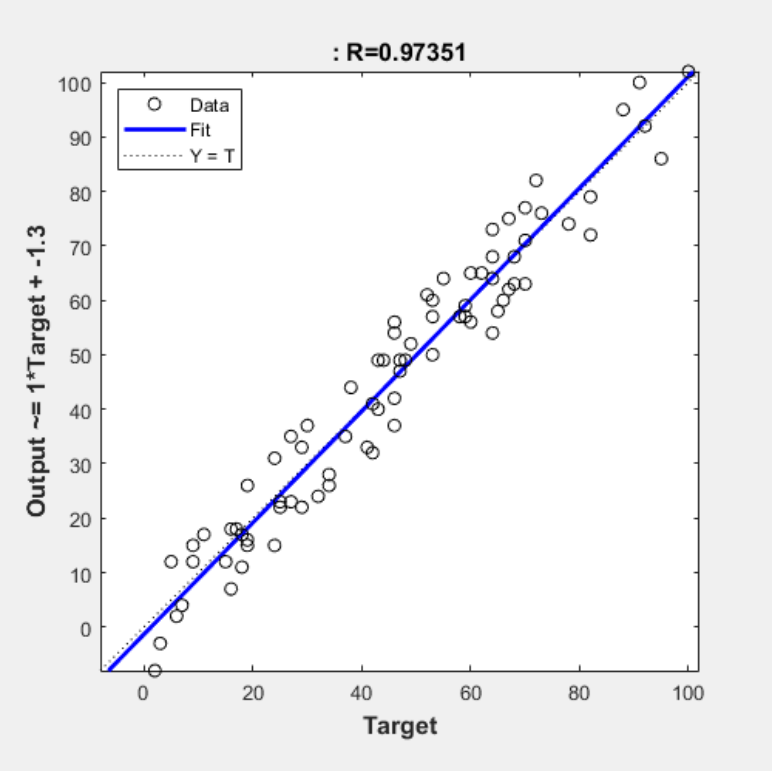


Figure 31: Example of a regression plot used to evaluate the model performance

2.6.2 Equation derivation

The best-performing model can be represented mathematically by using the fundamental input-output equation. This equation is based on the underlying mathematical relationships underpinning the multi-layer perceptron which was discussed in Section 1.5.1: *The multi-layer perceptron*. The summation- and activation function used in this process are given by Equation (2) and Equation (6), respectively.

The form of the equation is dependent on the activation functions of the hidden- and output layers. In this case, the logistic sigmoid activation function was used in the hidden layer(s) and the output layer. The fundamental input-output equations for each of the respective layers of the network were consequently constructed in the form of Equation (10).

The weights between each of the layers and the biases for each of the computational layers can be extracted as matrices from the best-performing ANN to be used in the equation. A set of equations is then obtained that govern the value of the output of each layer. Since the output of a layer is the input of the subsequent layer, these equations can be simplified to a single equation that governs the behaviour of the entire network.

2.6.3 Refrigeration system performance evaluation

Using the best-performing model, the performance of the refrigeration system can be evaluated. To achieve this, the relationships between the input parameters and the output have to be established. This is done by keeping all input values constant except those being investigated.

The constants for each parameter are chosen based on its mean value in the TCI class. This value is chosen since it represents a neutral or typical state of the system. The variable parameters have values ranging from its minimum to its maximum value, which is also taken from the TCI class. This range represents the operational range of the system with respect to that parameter.

By establishing the relationships between the input parameters and the COP, the optimal operating point of the refrigeration system can be determined for each input parameter.

2.6.4 Refrigeration system performance optimisation

To optimise the refrigeration system performance, the operating conditions that favour the best performance of the target variable should be maintained. These operating conditions are identified using the method presented in the previous section. The model can therefore receive

as inputs the optimal values of each parameter to predict the optimal performance. This value represents a theoretical limit where operational conditions can be fine-tuned in a controlled environment.

Practically, however, more complex conditions prevail. A refrigeration system performs radically different depending on ambient conditions. This manifests in improved system performance during colder months because of lower ambient temperatures. This is favourable since it leads to superior heat rejection rates at the condenser and consequently cooling capacity on the evaporator side.

To account for this, daily average values can be determined of the ambient conditions and parameters that are affected by these conditions. This can be calculated over all the available data and used as inputs to the model. These values represent the uncontrollable aspects that influence system performance and describe the typical conditions experienced by the refrigeration system.

Conversely, the rest of the model inputs represent the aspects of the refrigeration system that are adjustable and which, if implemented at the most favourable values, constitutes the practically achievable optimisation of the system.

The improvement in performance can then be determined by comparing the performance of the refrigeration system under favourable conditions to their baselines, determined by historical data.

2.7 Model verification

To meet the objectives of this study, the model developed using the method presented in this chapter must be verified. This is done to determine whether the results obtained from its implementation are correct and, consequently, whether the interpretation of those results can be trusted.

Verification can be done on the independent verification subset. As mentioned in Section 2.4.4, this subset remains uninvolved in the development of the model to ensure that the model is not biased toward the data in this subset during training.

The accuracy and robustness of the model can be evaluated using various methods. The accuracy of its predictions can be visualised with a plot showing the predicted and value against the target value. Regression analysis was used to evaluate the model's fit compared to the ideal fit of Y = T. The R^2-value was also calculated to get an objective measure of the relative variance

in the performance that can be explained by the input variables. The RMSE and MAPE of the verification subset was calculated to get a wholistic view of the prediction capabilities of the model on unseen data.

2.8 Method Verification

The method developed in this section was designed to meet the need of the study identified through the state-of-the-art analysis. To verify that the method satisfies the need, a comparison was drawn between the stages of the method and the state-of-the-art criteria. This comparison is shown in Table 5.

Table 5: Method verification: comparison of developed method to state-of-the-art criteria

	Application	Sensitivity analysis			Data pre-processing		Verification
	Practical	Number of hidden layers	Number of hidden neurons	Training algorithm	Cross-validation	Filtering	Independent dataset
Stage 1: system identification							
Stage 2: Parameter selection							
Stage 3: Data pre-processing							
Stage 4: Model development							
Stage 5: Evaluation and verification							

2.9 Summary

This chapter presents the proposed method to address the need and meet the objectives of this study. The first stage of the proposed method was to conduct a system investigation to ascertain the system interactions and the availability of data. It was then described how ideal input and output variables can be selected, filtered, and allocated to datasets to be ready for training the model. The method of developing the model was presented, which included the determination of the best number of hidden layers, number hidden neurons, and training algorithm.

A description followed of how the model can be implemented to determine the relationship between the inputs and outputs and how the fundamental input-output equation can be derived. A method was then given of how the model predictions can be used to improve the refrigeration system performance. Finally, a description was given of how the model can be verified to evaluate its accuracy and robustness. A summary of the proposed method is given in Figure 32.

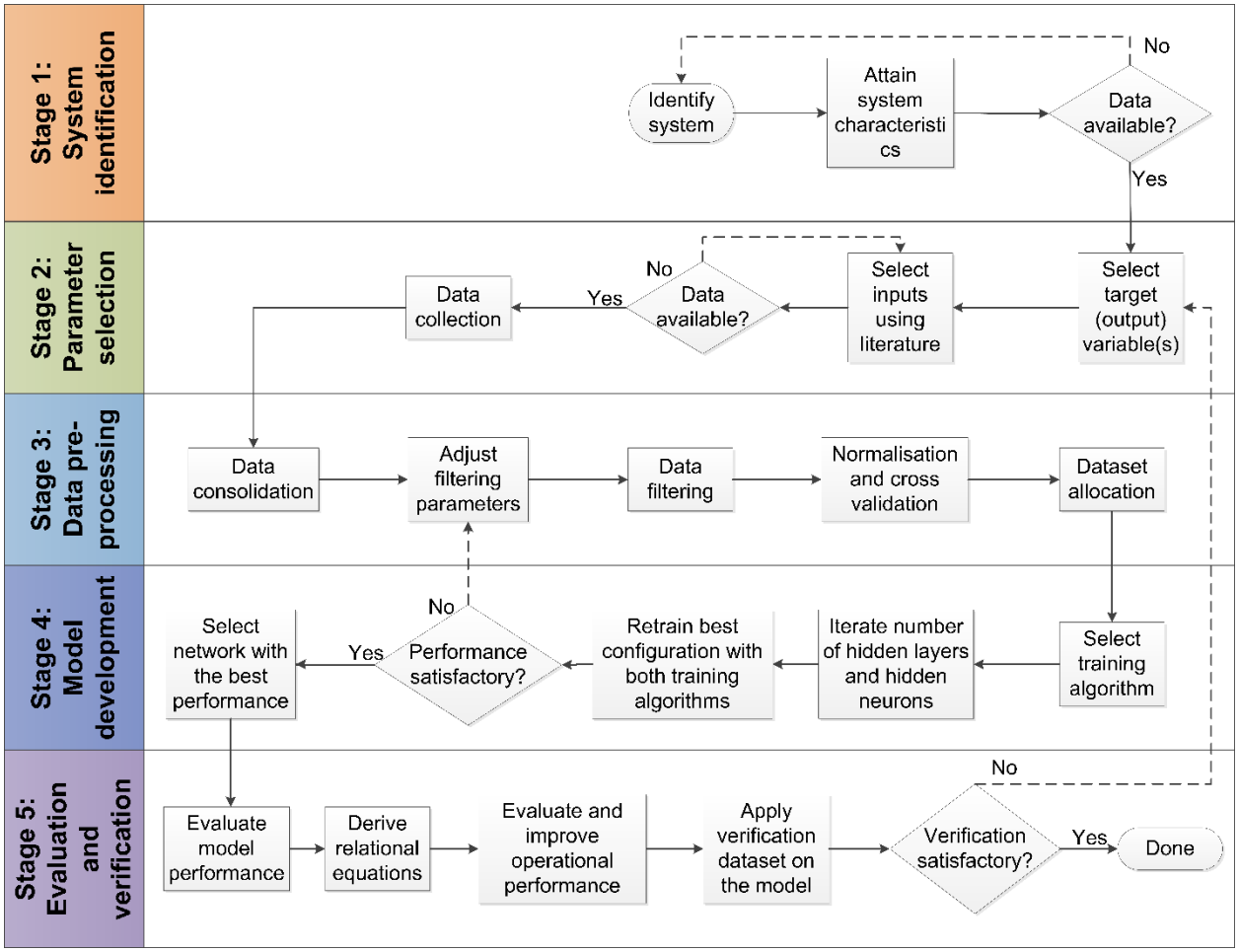
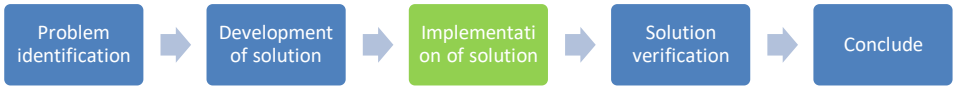


Figure 32: Chart representing the proposed method to meet the objectives of the study

CHAPTER 3 IMPLEMENTATION AND RESULTS

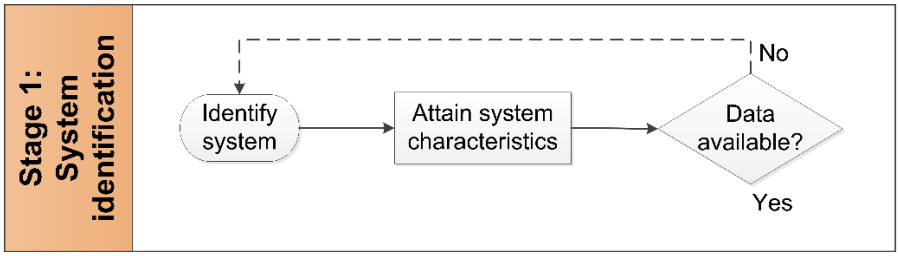


3.1 Preamble

This chapter presents and discusses the results obtained from implementing the method presented in Chapter 2. The system was investigated to describe its components, their interactions, and the availability of data. The results are given for the selection of variables, pre-processing of the data, and data allocation. Emphasis is placed on the development of the model and its best-performing architecture in terms of prediction accuracy. This involves the determination of the optimal number of hidden layers, number of hidden neurons in those layers, and the best training algorithm.

The best-performing model was assessed based on performance indicators. The fundamental input-output equation was derived, and the model was used to establish the relationships between each input and the output of the model. Using this information, the refrigeration system was optimised and, finally, the model was verified.

3.2 Stage 1: System identification



3.2.1 Case study

The case study refrigeration is situated at a deep-level gold mine in South Africa. The mine is 2 500m deep at its deepest and therefore requires artificial cooling in the form of chilled water. Chilled water is produced at a surface refrigeration plant where it can be cooled according to demand. The water is used in both surface and underground cooling applications such as bulk air coolers (BACs) and artificial cooling units (ACUs).

Before entering the refrigeration circuit, water is sent through a pre-cooling system. Water is pumped out of the mine at temperatures of up to 38 °C into a hot dam using dewatering pumps.

The water is sent to pre-cooling towers (PCTs) that reduce the water temperature to between 15 and 24 °C. From here the water flows to an intermediate sump from which gravity feeds water into a warm well. The refrigeration circuit starts at the warm well. A schematic of the refrigeration circuit is given in Figure 33.

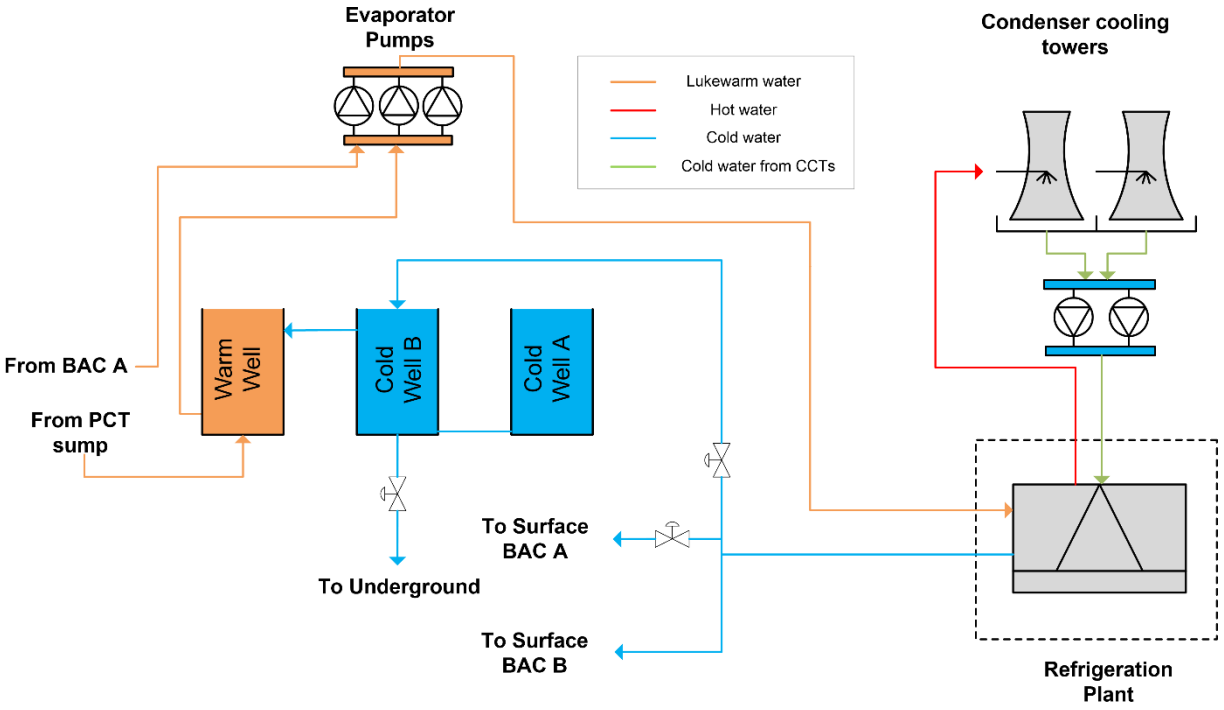


Figure 33: Schematic representation of the refrigeration circuit at Mine A

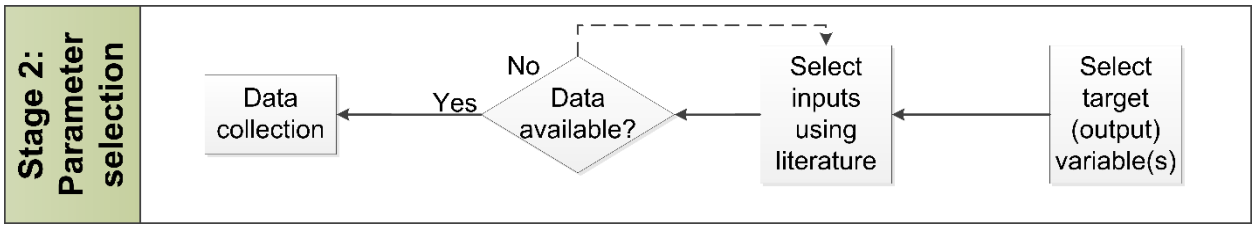
Water is pumped from the warm well to the refrigeration plant using evaporator pumps. The chilled water produced at the refrigeration plant is distributed to two surface BACs and two intermediate storage dams (Cold well A & B) at temperatures between 0.3 and 5 °C. From Cold well B, water is sent underground to be used in an underground BAC and several MCUs. Excess water in Cold well B flows to the warm well.

The refrigeration plant has a cooling duty of 2.25MW and consists of a single-stage vapour compression refrigeration system with ammonia (R717) as refrigerant. On average, the plant receives water at a rate of 320 L/s and is powered by an 1 850-kW centrifugal compressor. To improve the heat rejection rate of the system, condenser cooling towers are employed. Heat is rejected to water through the condensers which is then pumped to the towers to be air-cooled and are recirculated to each module.

3.2.2 Data Availability

The case study refrigeration plant has 18 sensors installed to measure temperature, flow rate, power, and pressure at various points of the system and on various components. Through analytical methods, parameters such as cooling capacity, heat rejection rate, and relative humidity can be calculated. Since enough data is available to characterise the system effectively, this option was not explored to keep the method applicable to refrigeration systems generally. The case study refrigeration plant is monitored by sensors from which data is extracted and transmitted to programmable logic controllers (PLCs). This data is stored on the case study mine's Scada system and can be accessed through their historian database. Data has been stored on this database since October 2020.

3.3 Stage 2: Variable selection



The coefficient of performance was chosen as the model output since it is a holistic measurement of the performance of a refrigeration plant. As discussed in Section 2.3, the inputs were chosen based on their use in literature and the availability and reliability of the data at the case study refrigeration plant. Table 6 shows the list of parameters at the case study refrigeration plant that were also used as input parameters from studies that form part of the state-of-the-art summary.

Using these criteria, the inputs to the model were chosen to be: water temperature at the evaporator inlet (T_{EI}) and outlet (T_{EO}), water flow rate at the evaporator ($\dot{V}_E$), cooling water flow rate at the condenser ($\dot{V}_C$), refrigerant temperature at the evaporator inlet (T_{RI}) and outlet (T_{RO}), ambient wet-bulb temperature (T_{WB}), and compressor power (P_C).

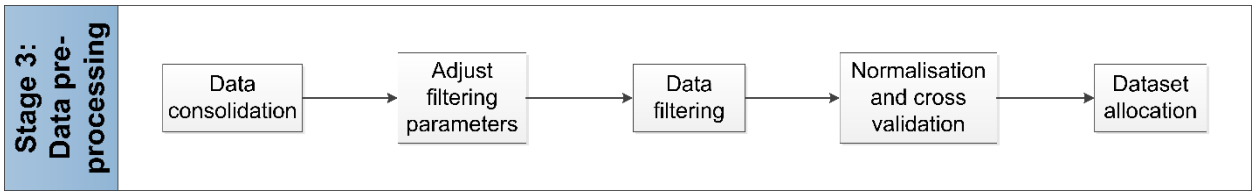
Table 6: List of the most used inputs in literature and the available parameters on the case study mine's database

Inputs from literature	Available	Reliable
Evaporator inlet water temperature	✓	✓
Evaporator outlet water temperature	✓	✓
Evaporator refrigerant inlet temperature	✓	✓
Evaporator refrigerant outlet temperature	✓	✓
Evaporator water volumetric flow rate	✓	✓
Condenser inlet water temperature	✓	X
Condenser outlet water temperature	✓	X
Condenser water volumetric flow rate	✓	✓
Dry-bulb temperature	X	X
Wet-bulb temperature	✓	✓
Relative humidity	X	X
Condenser inlet refrigerant temperature	✓	X
Evaporator temperature	X	X
Evaporator load	X	X
Evaporator pressure	X	X
Condenser temperature	X	X
Condenser pressure	X	X
Compressor rotation speed	X	X
Compressor frequency	X	X
Generator temperature	X	X
Cooling capacity	✓	X
Compressor Power	✓	✓

Data collection

Raw data was extracted from the mine's internal data storage system in 30-minute intervals. A total of 242 963 data points were gathered from 9 parameters (8 inputs and 1 output) from the 1st of October 2020 to the 30th of October 2023. The data was collected over several years to contain a wide range of operational conditions from which the ANN could learn. This process satisfies the objective of choosing appropriate input and output parameters of the model.

3.4 Stage 3: Data pre-processing



This section shows the results of implementing the data pre-processing discussed in Section 2.4. The result of consolidation is given, showing how causality between all input- and output pairs was achieved. The effect of applying the inequality-, noise spike-, and exponentially weighted moving average filters are shown and finally the result for cross validation is given.

3.4.1 Data consolidation

Data was consolidated by removing an entire data pattern at a timestep where no data was recorded. Figure 34 illustrates a simplified version of this process for visual purposes using the evaporator temperatures. The two regions highlighted in red indicate periods where one or more parameters had missing data. The values for all parameters during this period were removed.

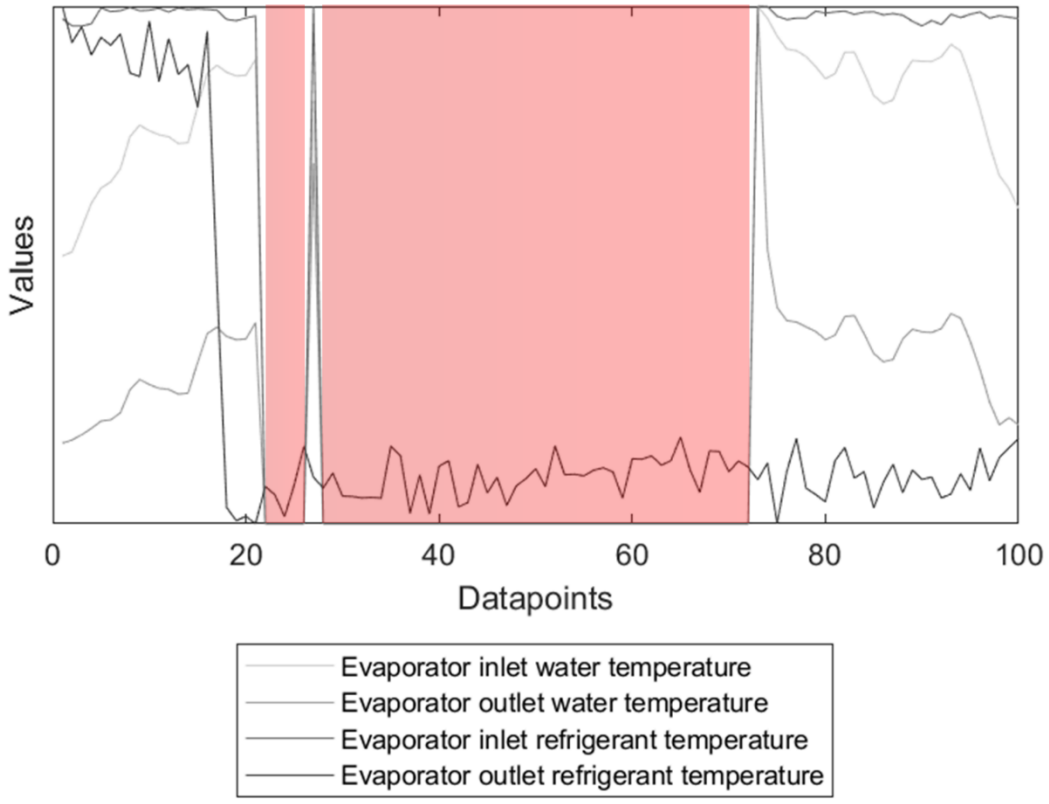


Figure 34: Illustration of the data consolidation process used to ensure input-output causality

3.4.2 Filtering

The inequality, noise spike and EWMA filters were implemented on the raw data. The inequality filter successfully removed data from periods of non-operation. The noise spike filter was applied to each parameter. For each parameter, the maximum allowable change (Δy) was individually tuned to achieve satisfactory noise dampening. The EWMA filter was applied to each parameter where the λ-parameter was adjusted until the transient effects were satisfactorily reduced. Figure **35** shows the raw COP data compared to the data after filters have been applied. The effect of applying filters on the parameters are given in Appendix B.

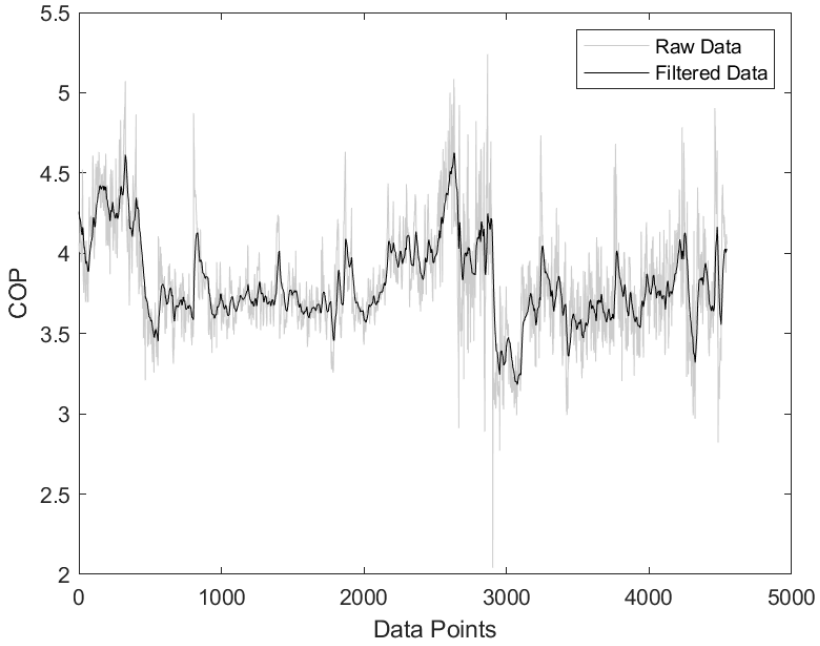
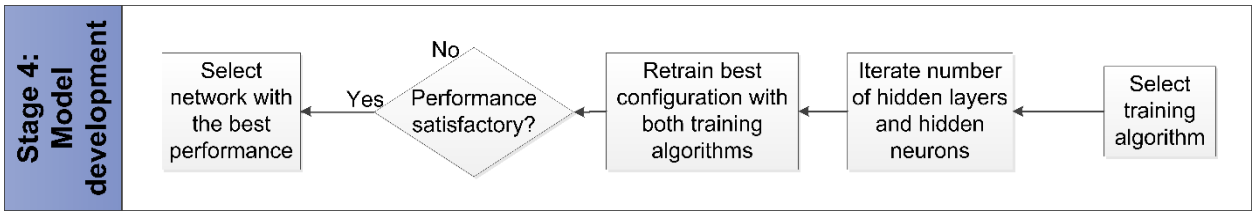


Figure 35: Raw data compared to filtered data for the refrigeration plant COP

The data pre-processing process reduced the total number of data points from 242 963 to 40 905. The number of usable data patterns in the dataset improved from 18.7% to 100%. In total, 4 545 data patterns were usable to develop the model. This process satisfies the objective of using pre-processing to improve the data quality for the purpose of developing an effective model.

3.5 Stage 4: Model development



To develop the best-performing model to predict the refrigeration plant performance, the sensitivity analysis technique was used. The network with the best prediction capabilities were determined in terms of the number of hidden neurons for one- and two hidden layer neural networks, respectively. Each configuration of the network was trained by both the LM and SCG training algorithms.

The best-performing architectures for one- and two hidden layer networks were then chosen based on their validation subset's performance indicators (R^2, RMSE, and MAPE) and subsequently trained repetitively with the LM and SCG training algorithms to determine the best-performing training algorithm. The training algorithm performance was measured with the MSE performance indicator. The network architecture (number of hidden neurons, hidden layers, and training algorithm) with the best performance was then selected for implementation.

3.5.1 Hidden neurons

The first characteristic of the model that was determined was the optimal number of hidden neurons for one- and two hidden layer ANN configurations. The result of this process is detailed in this section.

One hidden layer networks

For ANNs with one hidden layer, it was decided to vary the number of hidden neurons between 1 and 200. As mentioned in Section 2.5.1, this improves the rigour of the process and allows a trend of weaker generalisation to emerge, which assists in determining the optimal points.

Levenberg-Marquardt training algorithm

The RMSE and R^2-values for the training-, validation-, and testing subsets for ANNs trained with the LM algorithm were plotted against the number of hidden neurons (Figure 36). The RMSE values for all three subsets are indicated in blue, the R^2 values for all three subsets are indicated in orange.

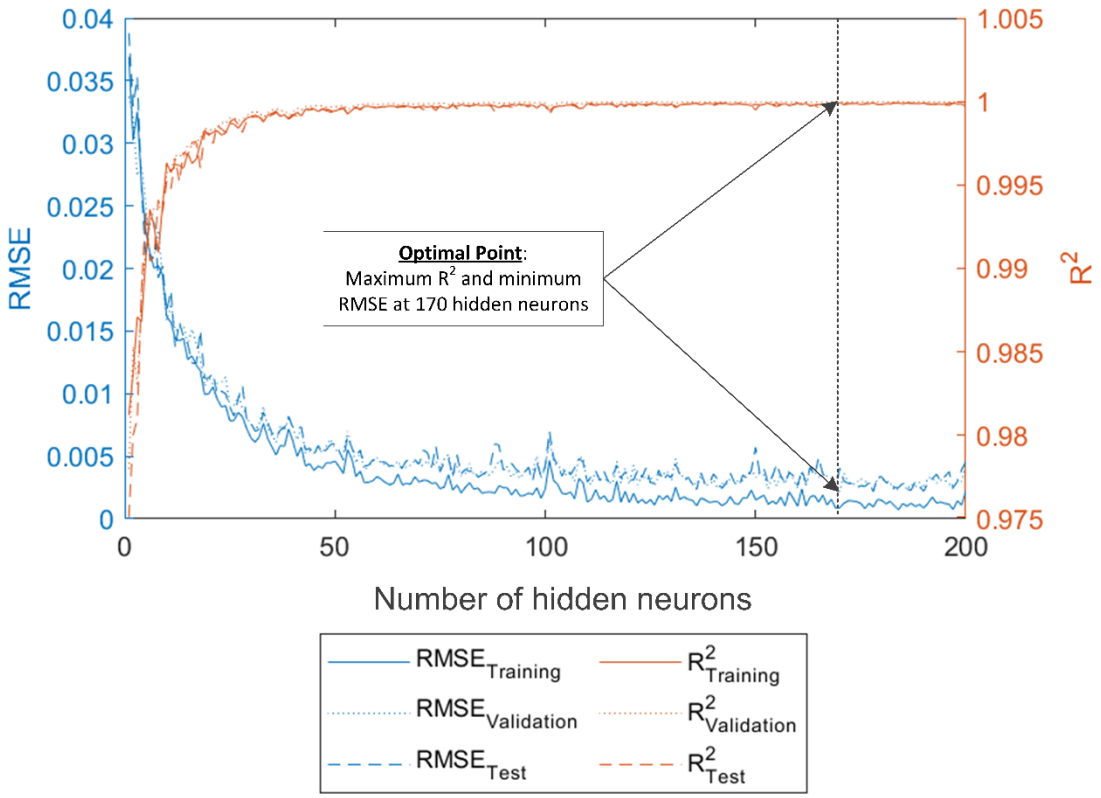


Figure 36: Prediction accuracy of a single hidden layer ANN trained with the LM algorithm

To find the optimal point as indicated in the figure, the validation subset performance indicators were used. The RMSE steadily decreased until it reached a minimum at 170 hidden neurons. Conversely, the R^2 values increased steadily until a maximum was reached (also at 170 hidden neurons). The best-performing configuration was therefore found at 170 hidden neurons. At the optimal point, the best validation subset RMSE, R^2, and MAPE values of 0.0022, 0.9993, and 0.45% were achieved, respectively. The total training time was 391 minutes and 26 seconds.

Scaled conjugate gradient training algorithm

The RMSE and R^2-values for the training-, validation-, and testing subsets for ANNs trained with the SCG algorithm were plotted against the number of hidden neurons (Figure 37). Similarly to the LM algorithm, the RMSE values on all subsets trended downward until a minimum value was reached at 189 hidden neurons. Conversely, the R^2-values for all subsets trended upwards until a maximum value was reached also at 189 hidden neurons. The process was significantly more unstable than its counterpart. At 195 hidden neurons, the RMSE and R^2-values suddenly inverted

the trend, which signals that the search on the error surface might have got stuck at a local minimum.

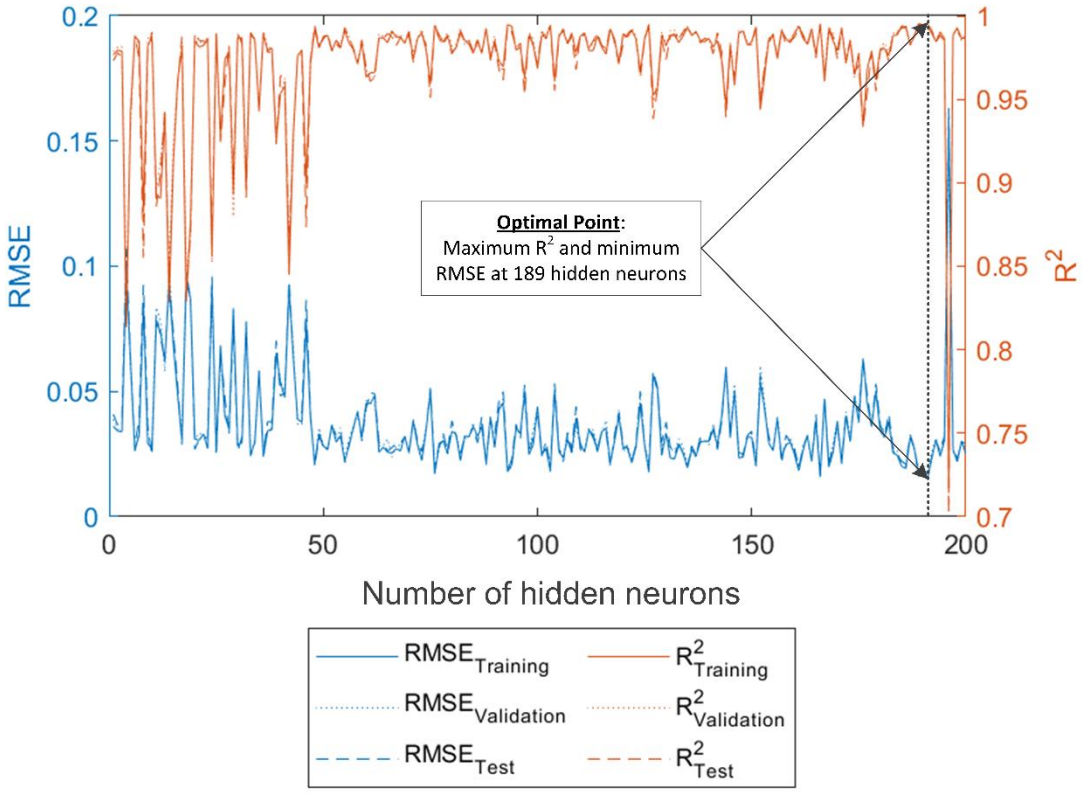


Figure 37: Prediction accuracy of a single hidden layer ANN trained with the SCG algorithm

The minimum RMSE and MAPE on the validation subset for all iterations were 0.016 and 3.99%, respectively. The R^2-value on the validation subset reached a maximum of 0.995. The total training time was 11 minutes and 46 seconds.

Two hidden layer networks

For ANNs with two hidden layers, it was decided to vary the number of hidden neurons between 1 and 40 for both layers. For brevity, only the validation subset performance indicators are visually represented.

Levenberg-Marquardt training algorithm

The R^2 and RMSE values for the validation subset of a two hidden layer ANN trained by the LM algorithm versus the number of hidden neurons in each hidden layer are shown in Figure 38 and Figure 39, respectively. The hidden neurons in the first hidden layer are given on the x-axis and

the hidden neurons in the second hidden layer are given on the y-axis. The RMSE values in Figure 38 and the R^2 values in Figure 39 are given on their respective z-axes. The lowest RMSE achieved was 0.0021 with 37 neurons in the first hidden layer and 34 neurons in the second hidden layer. The highest R^2 value of 0.9993 was achieved with 37 neurons in the first hidden layer and 34 neurons in the second hidden layer. The network MAPE was 0.42%. Similarly to the single hidden layer architecture trained with the LM, a high degree of accuracy was quickly achieved with diminishing returns over time. The total training time was 1 067 minutes and 32 seconds.

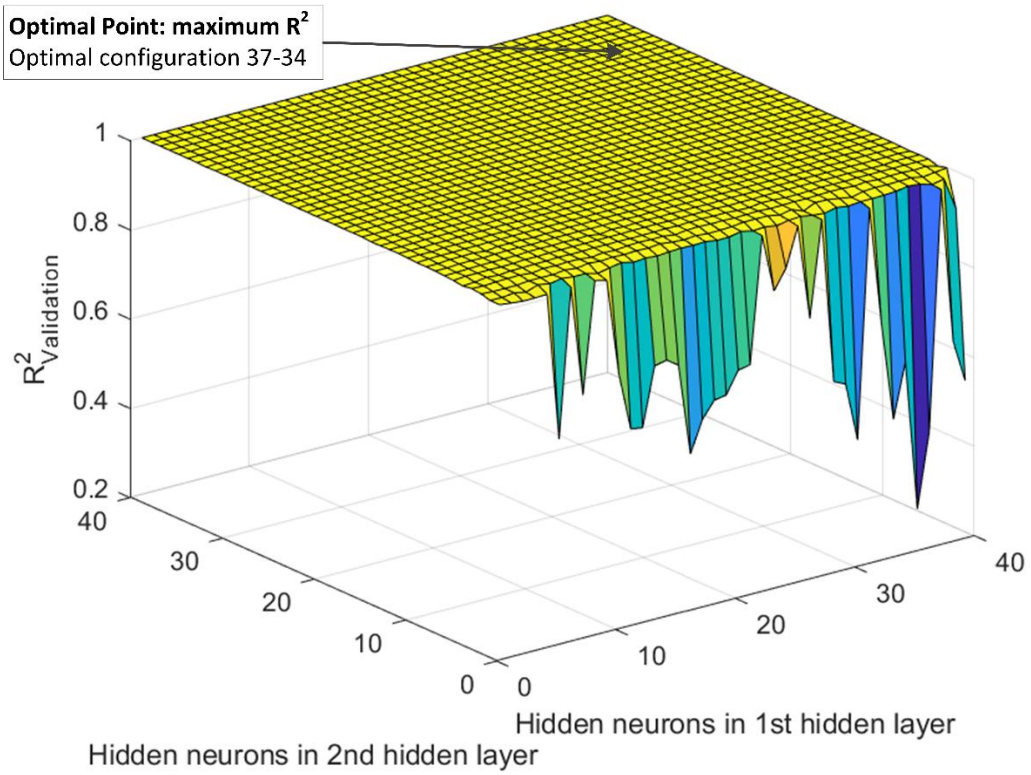


Figure 38: Validation subset R^2 values for a two hidden layer ANN trained with the LM algorithm

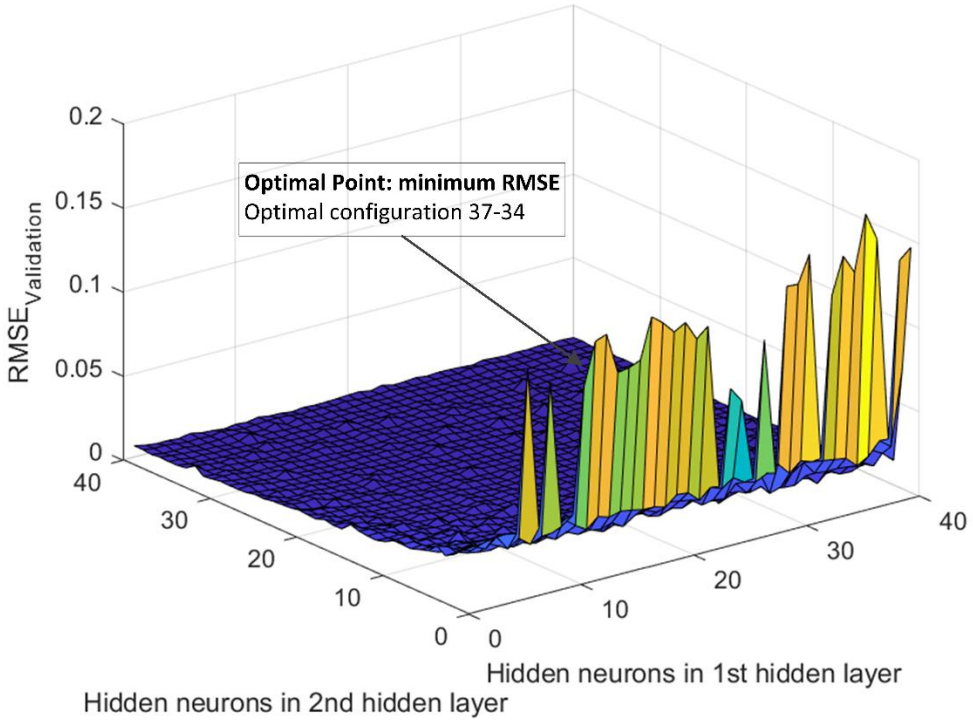


Figure 39: Validation subset RMSE for a two hidden layer ANN trained with the LM algorithm

Scaled conjugate gradient training algorithm

The RMSE and R^2 values for the validation subset of a two hidden layer ANN trained by the SCG algorithm versus the number of hidden neurons in each hidden layer are shown in Figure 40 and Figure 41, respectively. The hidden neurons in the first hidden layer are given on the x-axis and the hidden neurons in the second hidden layer are given on the y-axis in both figures. The performance indicators are given on the z-axes.

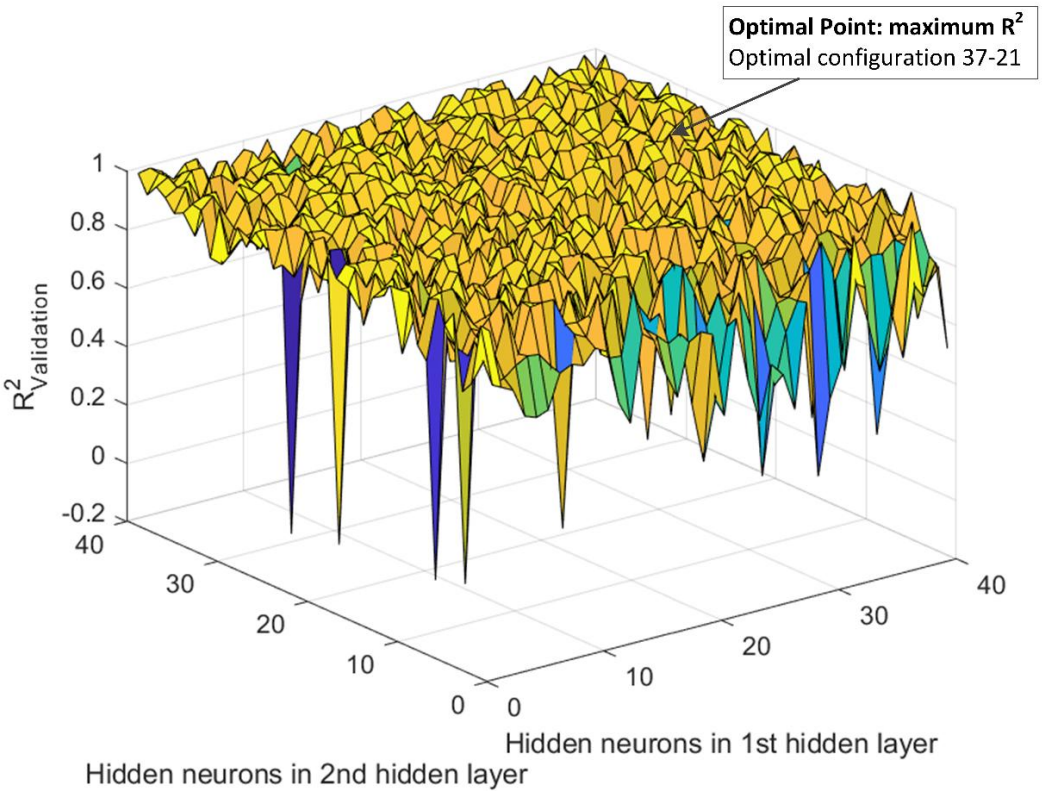


Figure 40: Validation subset R^2 values for a two hidden layer ANN trained with the SCG algorithm

The optimal points for the R^2 and RMSE values were reached at 37 hidden neurons in the first hidden layer and 21 hidden neurons in the second hidden layer. The R^2 value at this point was 0.995 and the best RMSE was 0.017. The network MAPE was 4.03% and the total training time was 31 minutes and 30 seconds.

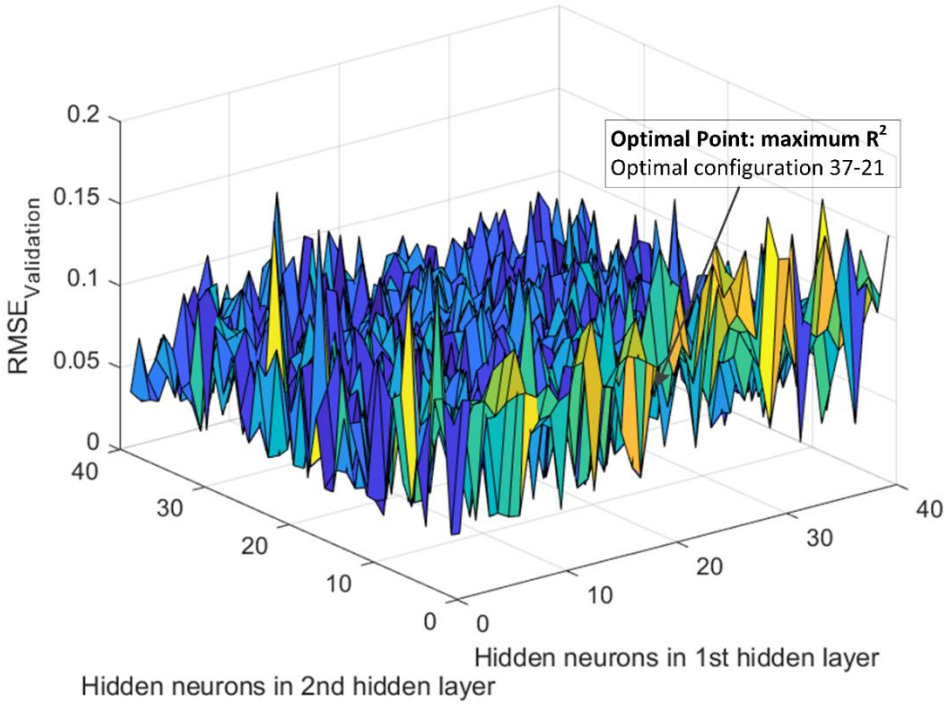


Figure 41: Validation subset RMSE values for a two hidden layer ANN trained with the SCG algorithm

Overview of architectures

One and two hidden layer ANNs were trained with the LM and SCG training algorithms with varying numbers of hidden neurons in their hidden layers. For each configuration, the RMSE, R^2, and MAPE values were used to indicate its performance. The best-performing architectures for each configuration was chosen based on the performance indicators. The performance of each configuration is given in Table 7. The number of hidden neurons that yielded the best performance of each configuration is shown next to the training algorithm.

Table 7: The best-performing architectures for each configuration and training algorithm

Performance indicator	One hidden layer		Two hidden layers	
	LM (170)	SCG (189)	LM (37-34)	SCG (37-21)
RMSE	0.0022	0.0157	0.0021	0.017
R^2	0.9993	0.9950	0.9993	0.9950
MAPE (%)	0.45	3.99	0.42	4.03
Training time (hh:mm:ss)	06:31:26	00:11:46	17:47:32	00:31:30

The results of this section layer provide some insights into the ANN architecture with respect to the network configuration and training algorithms used:

1. The LM algorithm is considerably more stable than the SCG algorithm with respect to the performance indicators.
2. The SCG algorithm trains the networks considerably faster than the LM algorithm.
3. For both algorithms, a high degree of accuracy is achieved relatively quickly. This indicates that the algorithms are extremely effective at finding near-optimal network weights and biases.
4. The SCG algorithm is more likely to converge to a local minimum on the error surface, while the LM algorithm is more likely to converge to a global minimum on the error surface.
5. A clear rise in the RMSE and a decline in R^2 of the validation subset was not observed on any of the tests. This suggests that the network configurations considered did not overfit data as quickly as expected.

3.5.2 Training algorithm

To find the best suited training algorithm for predicting the COP of the case study refrigeration plant, each of the best-performing architectures shown in Table 7 were retrained 100 times with the LM and SCG algorithms. Since weights were randomly initialised, the repetitive training accounts for any bias that may exist in a single iteration and allows the training algorithms to search for the optimal solution over a large portion of the error function surface.

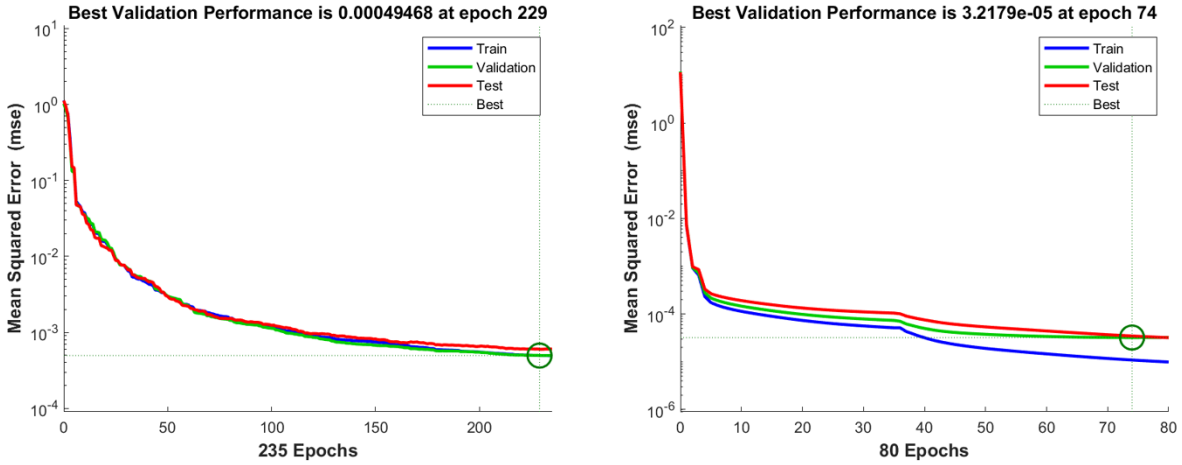
The MSE values on the training, validation, and test subsets were calculated after each training epoch for each algorithm and configuration. Early stopping was employed to avoid over- or underfitting and allow the best generalisation of the network. The early stopping point represents a local minimum on the error surface, measured by the MSE. The best network was found by finding the lowest MSE over all repetitions, which represents the global minimum for a specific configuration and training algorithm.

One hidden layer configurations

The one hidden layer ANN configuration that yielded the best results with the SCG training algorithm consisted of 189 hidden neurons. For the LM algorithm, the best configuration contained 170 hidden neurons in its hidden layer.

Configuration 1: 189 hidden neurons

After retraining the first configuration, minimum MSE values of 4.95×10^{-4} and 3.22×10^{-5} were achieved for the SCG and LM algorithms, respectively. The MSE values for the training-, validation-, and test subsets at each epoch for the SCG algorithm and the LM algorithm are given in Figure 42(a) and Figure 42(b), respectively. The minimum MSE value for the configuration trained by the SCG algorithm was achieved at 229 epochs while the minimum MSE for the LM algorithm variation was achieved at 74 epochs.



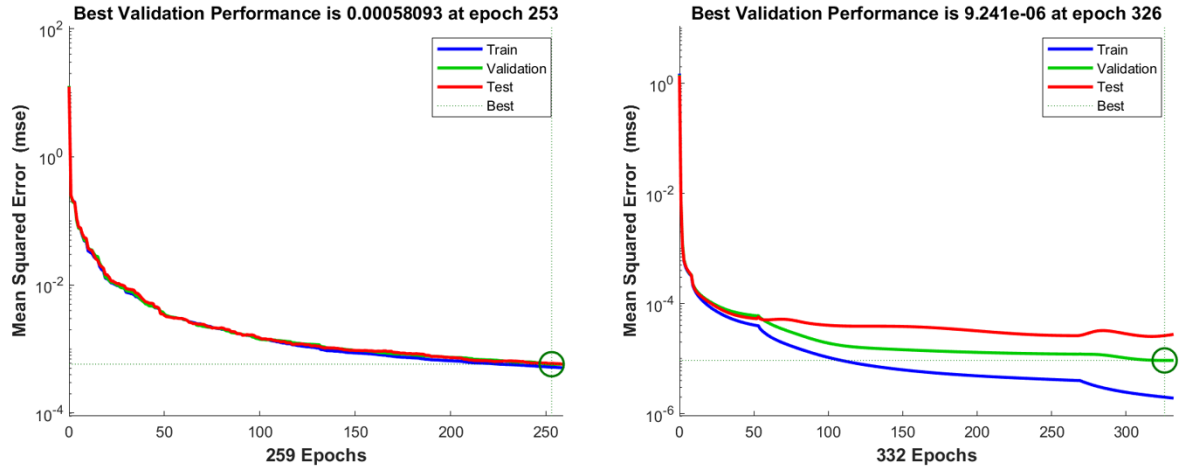
(a) Scaled conjugate gradient (b) Levenberg-Marquardt

Figure 42: Graphs showing the MSE of a single hidden layer ANN with 189 hidden neurons at each training epoch for (a) the SCG algorithm and (b) the LM algorithm

For this ANN configuration, therefore, the LM algorithm yielded a lower MSE and converged faster than the SCG variation.

Configuration 2: 170 hidden neurons

The second configuration yielded minimum MSE values of 5.81×10^{-4} and 9.24×10^{-6} for the SCG and LM algorithms, respectively. The MSE values of the training-, validation-, and test subsets at each epoch for the 170 hidden neuron ANN trained by the SCG, and LM algorithms are given in Figure **43**(a) and Figure 43(b), respectively. The minimum MSE for the ANN trained by the SCG algorithm was reached at Epoch 253. Its LM-trained counterpart reached its minimum MSE at Epoch 326.



(a) Scaled conjugate gradient

(b) Levenberg-Marquardt

Figure 43: Graphs showing the MSE of a single hidden layer ANN with 170 hidden neurons at each training epoch for (a) the SCG algorithm and (b) the LM algorithm

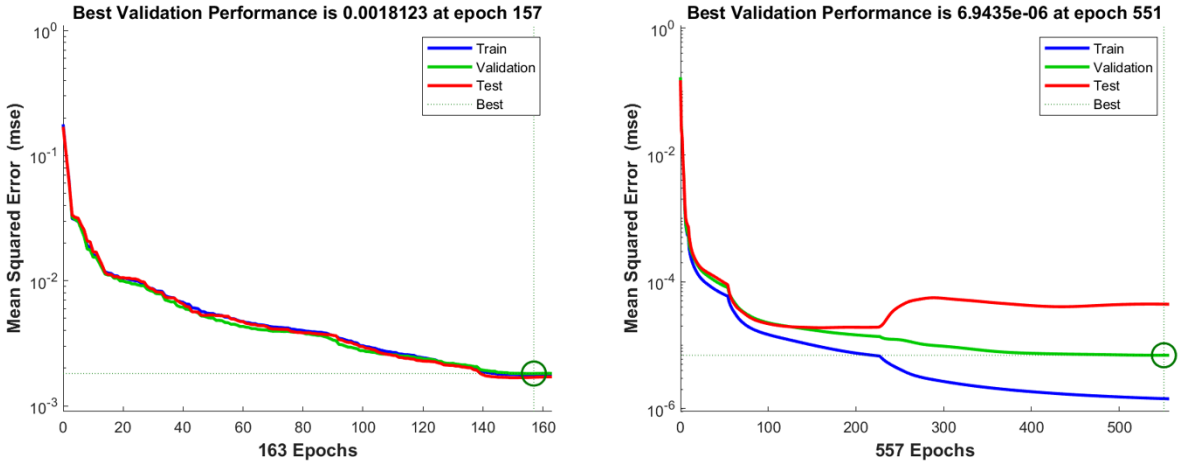
For this configuration, the LM-trained variation again achieved the lowest MSE value, but converged to that value slower than the SCG-trained variation. For all single hidden layer variations, the 170 hidden neuron ANN trained by the LM algorithm performed the best in terms of the MSE.

Two hidden layer configurations

The SCG-trained two hidden layer configuration that performed the best consisted of 37 hidden neurons in its first hidden layer and 21 in its second hidden layer. The best-performing LM-trained configuration consisted of 37 hidden neurons in its first hidden layer and 34 in its second hidden layer.

Configuration 3: (37-21 hidden neurons)

Configuration three, trained by the SCG algorithm achieved a minimum MSE of 1.81×10^{-3} at Epoch 157. Its counterpart, trained by the LM algorithm, achieved a minimum MSE of 6.94×10^{-6} at Epoch 551. The MSE values of the training-, validation-, and test subsets at each epoch for this ANN trained by the SCG and LM algorithms are given in Figure 44(a) and Figure 44(b), respectively. For this configuration, the LM algorithm performed better by achieving a lower MSE; however, its SCG-trained counterpart converged faster.

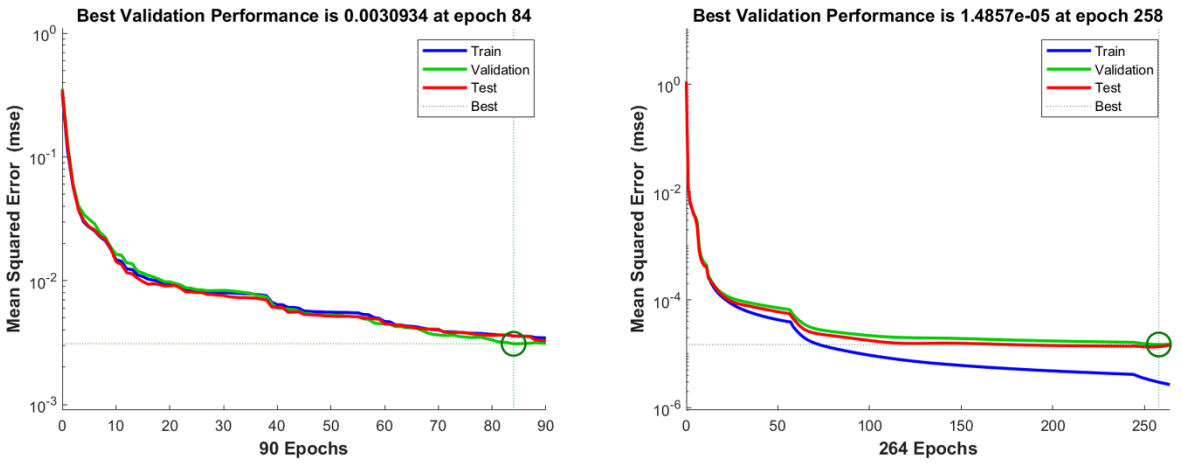


(a) Scaled conjugate gradient (b) Levenberg-Marquardt

Figure 44: Graphs showing the MSE of an ANN with 37 neurons in hidden layer 1 and 21 in hidden layer 2 at each training epoch for (a) the SCG algorithm and (b) the LM algorithm

Configuration 4: (37-34 hidden neurons)

The final configuration tested achieved a minimum MSE of 3.09×10^{-3} at an epoch of 84 when trained by the SCG algorithm. When trained by the LM algorithm, a minimum MSE of 1.48×10^{-5} was achieved at Epoch 258. The MSE values per epoch for the SCG-trained variation of Configuration 4 is given by Figure 45(a). The MSE values per epoch for the LM-trained variation of Configuration 4 is given by Figure 45(b).



(a) Scaled conjugate gradient (b) Levenberg-Marquardt

Figure 45: Graphs showing the MSE of an ANN with 37 neurons in hidden layer 1 and 34 in hidden layer 2 at each training epoch for (a) the SCG algorithm and (b) the LM algorithm

For this configuration, therefore, the LM-trained ANN achieved a lower MSE but converged slower than its SCG-trained counterpart.

Overview of training algorithms

For most configurations retrained, the LM algorithm resulted in slower convergence but higher accuracy. Since the error surfaces of each configuration were identical, it can be inferred that the SCG algorithm has superior convergence speeds and, based on the information presented in Section 1.5.4, this is likely due to the use of conjugate search directions. Conversely, it can be inferred that the LM algorithm's alternating use of gradient decent and the Gauss-Newton method allows it to escape local minima and continue searching for the global minimum. This can be seen in the general shape of the MSE per epoch of the LM algorithm. The MSE dropped fast initially, then settled into a local minimum before "escaping" and moving on to the next minimum. A summary of the training algorithm accuracy and convergence speeds is given in Table 8.

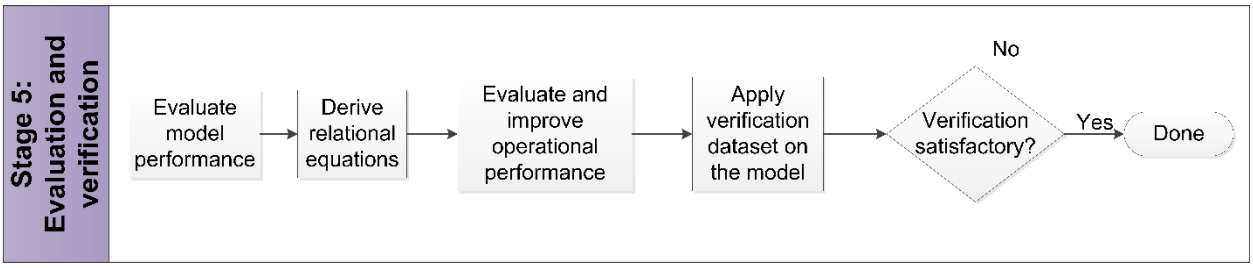
Table 8: Summary of the training algorithm's performance for all configurations

Configuration	Training algorithm	Performance indicator	Convergence speed
		MSE	Epochs
Single HL 189 HNs	SCG	4.95×10^{-4}	229
	LM	3.22×10^{-5}	74
Single HL 170 HNs	SCG	5.81×10^{-4}	253
	LM	9.24×10^{-6}	326
Two HLs 37-21	SCG	1.81×10^{-3}	157
	LM	6.94×10^{-6}	551
Two HLs 37-34	SCG	3.09×10^{-3}	84
	LM	1.48×10^{-5}	258

3.5.3 Hidden layers

Over all the configurations and training algorithms tested, it was found that the best-performing ANN consisted of two hidden layers with a 37-21 configuration. It was therefore decided that the ANN which best represented the case study refrigeration plant would be a two hidden layer ANN with 37 hidden neurons in its first hidden layer and 21 in its second hidden layer, trained by the Levenberg-Marquardt training algorithm.

3.6 Stage 5: Model implementation



The best-performing ANN determined in the previous section represents the best model for predicting the refrigeration plant COP. The ANN consisted of two hidden layers, with a 37-21 hidden neuron configuration, trained by the LM algorithm. In this section, the model's predictive capabilities were evaluated, followed by the derivation of the fundamental input-output equation of the model. The performance of the refrigeration system was then evaluated by using the model to establish the relationship between the input variables and the COP using the model. These relationships were then used to optimise the refrigeration plant performance.

3.6.1 Prediction evaluation of the model

The R^2, RMSE, and MAPE performance indicators were used to evaluate the predictive capabilities of the model. These performance indicators assess the predictive capabilities in distinct ways, which allowed for a broader perspective on the prediction accuracy of the model.

The R^2 value indicates the proportion of variation in the COP that the model inputs can explain. The RMSE provides insight into the prediction accuracy itself, emphasising larger errors, while the MAPE is a more balanced measurement of prediction accuracy since it places no emphasis on larger errors.

The correlation between the predicted- and the target outputs for the training-, validation-, and test subsets are given in Figure 46 (a) – (c). The correlation between the predicted and target outputs of the entire dataset (TCI class) are given in Figure 46 (d). The best fit line of the model is given by the solid line, the predicted values for each respective dataset are given by the black dots, and the ideal fit is given by the dashed line.

The model achieved R^2 scores of 0.99997 on the training subset, 0.99989 on the validation subset, 0.99989 on the test subset and 0.99995 on the overall dataset. The prediction accuracy of the model for all datasets with respect to the RMSE, R^2, and MAPE performance indicators is given in Table 9.

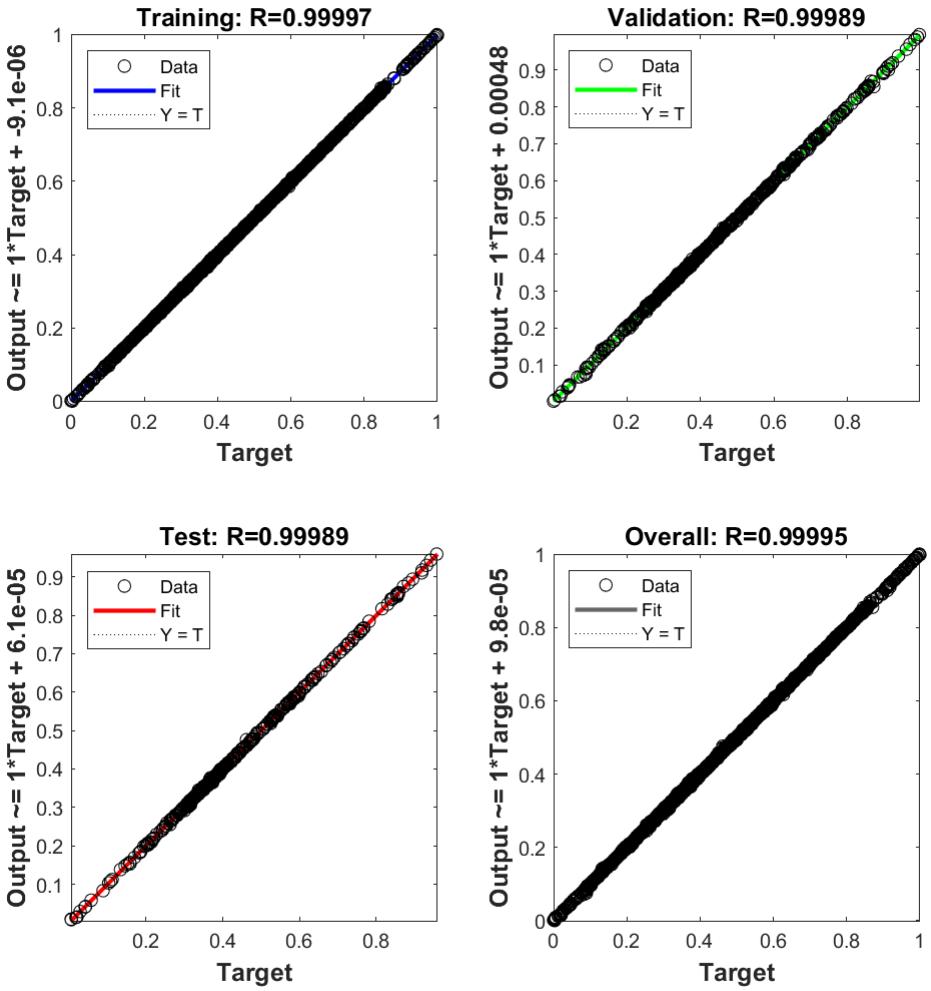


Figure 46: Regression plots indicating the correlation between the predicted outputs and the target outputs of the model for (a) the training subset, (b) the validation subset, (c) the test subset, and (d) the TCI dataset (overall)

Table 9: Performance of the model with respect to the RMSE, R2, and MAPE performance indicators for all datasets

Subset	RMSE	R^2	MAPE
Training	0.0011998	0.99997	0.26229
Validation	0.0026351	0.99989	1.7908
Test	0.0066614	0.99989	0.78889
Overall	0.0026139	0.99995	0.62065

These results indicate that the model was successful in predicting the refrigeration plant COP with a high degree of accuracy and can accurately represent the refrigeration plant performance over its operational range.

3.6.2 Equation derivation

The relationship between the inputs and outputs of the model established during the training process in terms of weights, biases, and activation functions can be represented by the fundamental input-output equation of the ANN. This equation governs the relationship between the input variables and the COP prediction and is derived in this section.

The normalised inputs to the model are the water temperatures at the evaporator inlet (T_{EI}) and outlet (T_{EO}), water flow rate at the evaporator ($\dot{V}_E$), cooling water flow rate at the condenser ($\dot{V}_C$), refrigerant temperatures at the evaporator inlet (T_{RI}) and outlet (T_{RO}), ambient wet-bulb temperature (T_{WB}), and compressor power (P_C). Each input is sent to each neuron in the first hidden layer where a weight is applied, and a bias added. The weight (w_{ij}) and bias (b_j) is denoted by i and j subscripts which indicate the input and hidden neuron, respectively.

Each neuron in the first hidden layer then performs two computations: summation and activation to produce the post-activation values of the first hidden layer ($y_{h1,j}$) which is given by Equation ((50). The weights and biases used in Equation (50 are given in Appendix C Table 17.

$$y_{h1,j} = \frac{1}{1 + e^{(w_{1,j} \times T_{EI} + w_{2,j} \times T_{EO} + w_{3,j} \times \dot{V}_E + w_{4,j} \times \dot{V}_C + w_{5,j} \times T_{RI} + w_{6,j} \times T_{RO} + w_{7,j} \times T_{WB} + w_{8,j} \times P_C + b_j)}} \quad (50)$$

The post-activation values from the first hidden layer are then received as inputs to the second hidden layer, where the same process is followed. The weights (w_{jk}) and biases (b_k) between the hidden layers' neurons are denoted by j and k subscripts which indicate the neuron in the first hidden layer from which the input is received and the neuron in the second hidden layer that receives it, respectively. The post activation values of the second hidden layer ($y_{h2,k}$) is calculated using Equation ((51). The weights and biases used in the equation are given in Appendix C Table 18 to Table 21.

$$y_{h2,k} = \frac{1}{1 + e^{(w_{1,k} \times y_{h1,1} + w_{2,k} \times y_{h1,2} + w_{3,k} \times y_{h1,3} + \ldots + w_{j,k} \times y_{h1,j} + \ldots + w_{37,k} \times y_{h1,37} + b_k)}} \quad (51)$$

Finally, the post-activation values of the second hidden layer are sent to the output layer with the hidden-to-output layer weights ($w_{k,o}$) and bias (b_o) applied. The activation function of the output neuron is the identity function, so the output of the model is the simply the sum of the post-activation values of the second hidden. The output of the model (y) is given by Equation ((52). The weights and bias of the output layer are given in Appendix C Table 22 and Table 23.

$$y = w_{1,o} \times y_{h2,1} + w_{2,o} \times y_{h2,2} + w_{3,o} \times y_{h2,3} + w_{k,o} \times y_{h2,k} + \ \ldots + w_{21,o} \times y_{h2,21} + b_o \tag{52}$$

3.6.3 Refrigeration system performance evaluation

To evaluate the performance of the refrigeration plant, the model was implemented to establish the relationships between the inputs and the COP. First, the relationship between the evaporator inlet water temperature, evaporator outlet water temperature and the COP of the refrigeration plant is given in Figure 47.

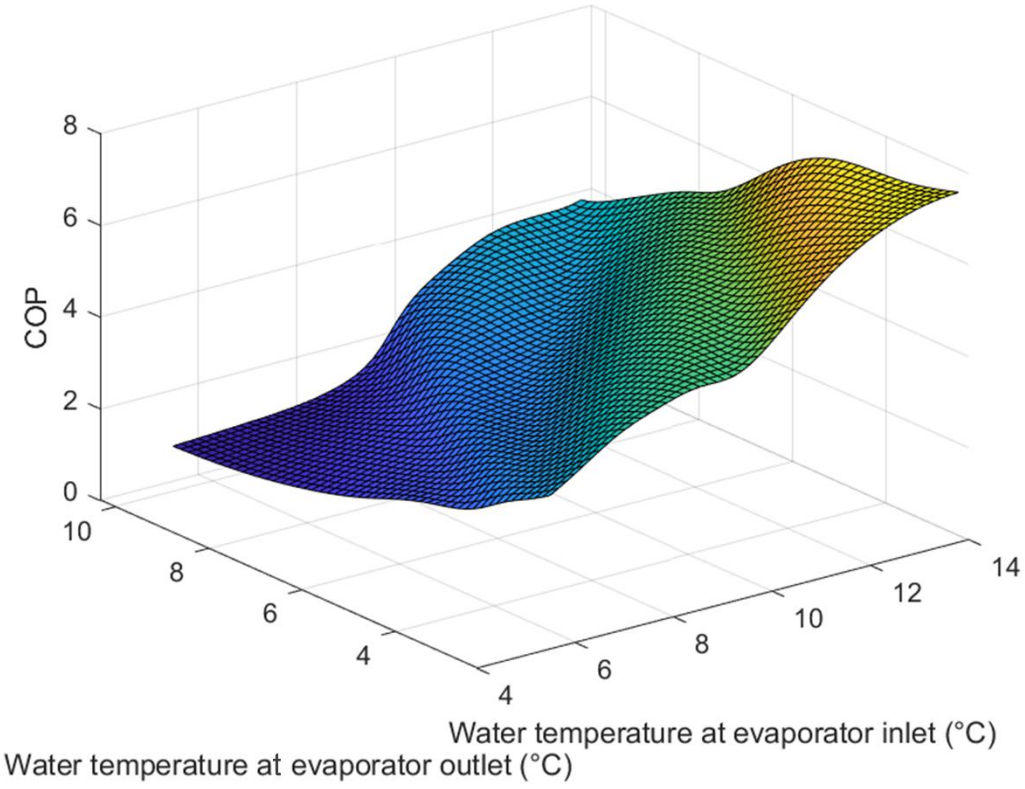


Figure 47: The effect of water temperature at the evaporator inlet and outlet on the refrigeration system's COP

The model predicted that a higher COP can be obtained if water enters the evaporator at a relatively high temperature and leaves at a relatively low temperature. This result agrees with a simple analysis of a refrigeration system's performance since the relationship between these variables indicate that a higher cooling capacity at a constant compressor power demand leads to higher COPs.

The relationship between the ammonia temperature at the evaporator inlet, the ammonia temperature at the evaporator outlet, and the COP is given in Figure 48.

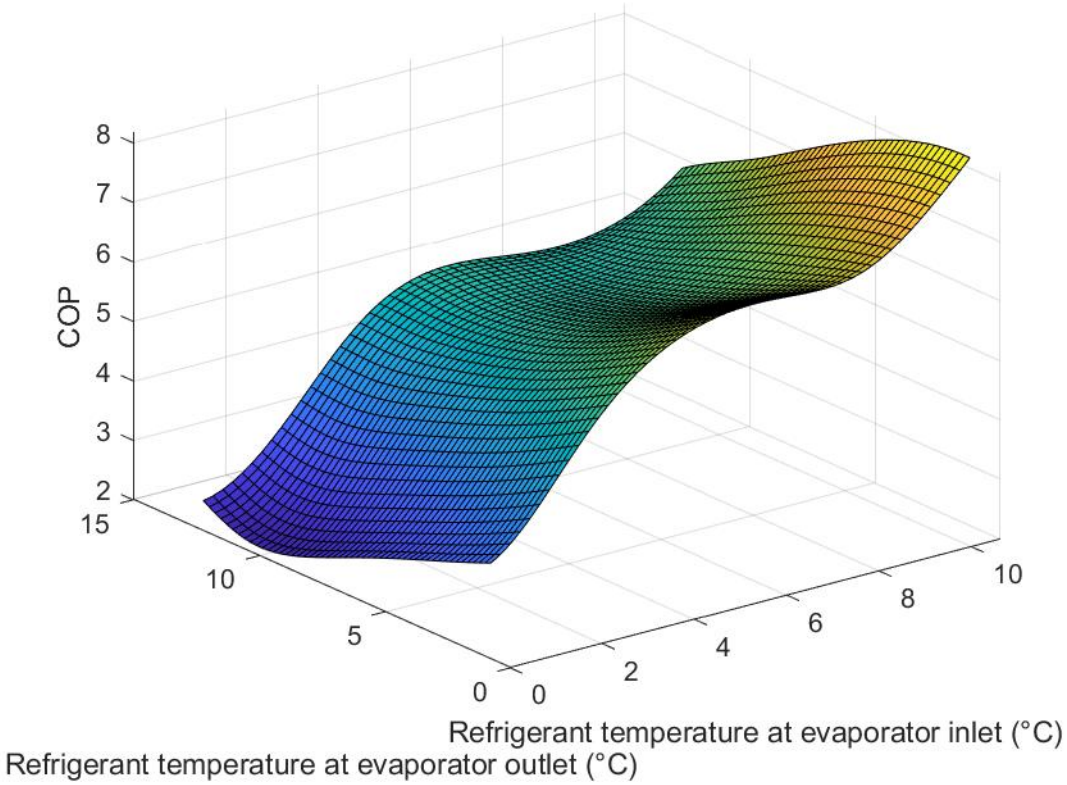


Figure 48: The effect of the temperature of ammonia at the evaporator inlet and outlet on the refrigeration system's COP

Similarly to the evaporator water temperature, the model predicts that higher COPs can be achieved if the refrigerant enters the evaporator at a low temperature and exits at a higher temperature. This prediction indicates that the model implicitly *learned* that the COP of the refrigeration system is relatively higher if the refrigerant has the capacity to allow more heat absorption.

The relationship between the water flow rate through the evaporator, the cooling water flow rate at the condenser, and COP is given in Figure 49. The model predicted that higher flow rates of water through the evaporator and cooling water at the condenser generally result in higher COPs. However, the model predicts a more complex relationship between the respective flow rates and COP for different combinations.

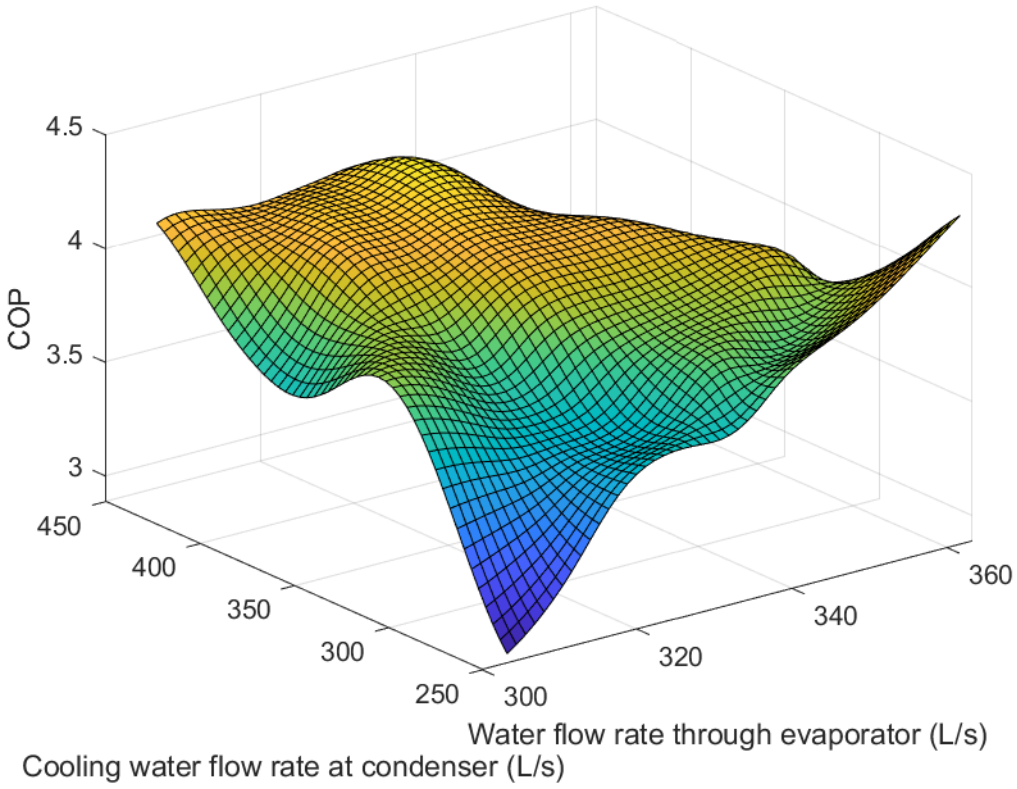


Figure 49: Relationship between the flow rate of water through the evaporator, the flow rate of cooling water at the condenser, and the COP of the refrigeration system

The relationship between the ambient wet-bulb temperature of the air around the refrigeration plant and its COP is given in Figure 50. In colder conditions, the heat rejection rate at the condenser improves, which means the refrigerant enters the evaporator at a lower temperature, resulting in better heat absorption at the evaporator and a higher COP as a consequence. This is reflected in Figure 50.

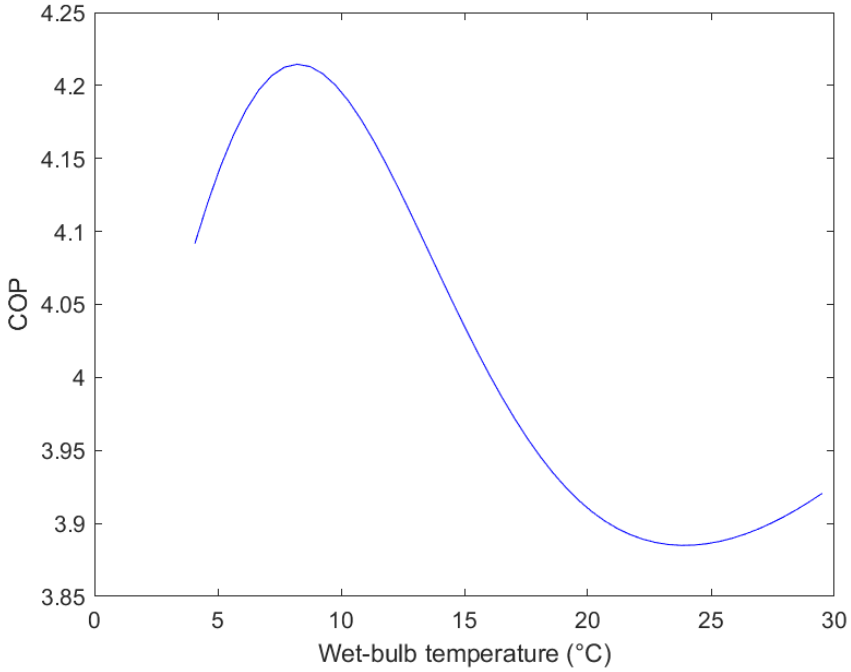


Figure 50: Relationship between the ambient wet-bulb temperature and the COP of the refrigeration system

Finally, the relationship between the compressor power and the COP is given in Figure 51. The power demand is linearly related to the pressure to which the refrigerant is compressed at the condenser. This results in a higher temperature differential between the refrigerant and, consequently, better heat transfer to the colder medium, eventually leading to higher cooling capacities.

From Figure 51, however, it is apparent that the system reaches a point of diminishing returns if compressor demand exceeds 1.27MW. At this point, the increase in cooling capacity is no longer justified by the amount of power needed to produce it.

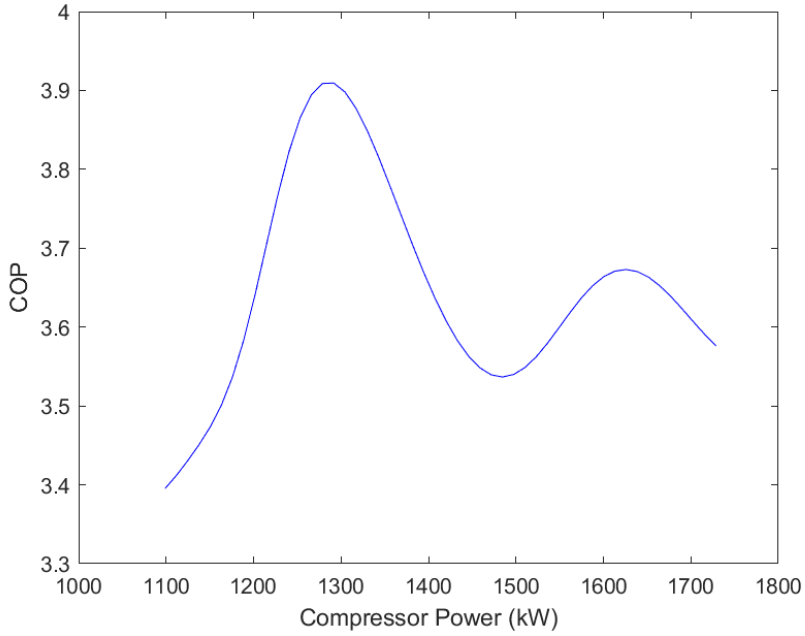


Figure 51: Relationship between the compressor power and the COP of the refrigeration system

3.6.4 Refrigeration system performance optimisation

By establishing the relationships between the input variables and the COP, it was possible to evaluate the predicted performance of the refrigeration plant under various circumstances. A baseline of the refrigeration system was created using data from the TCI class. This baseline represents the average value of each input for a given time of day over the entire dataset and was used as a reference to assess the improvement in performance. The actual baseline of the refrigeration system's COP compared to the predicted baseline COP is given in Figure 52.

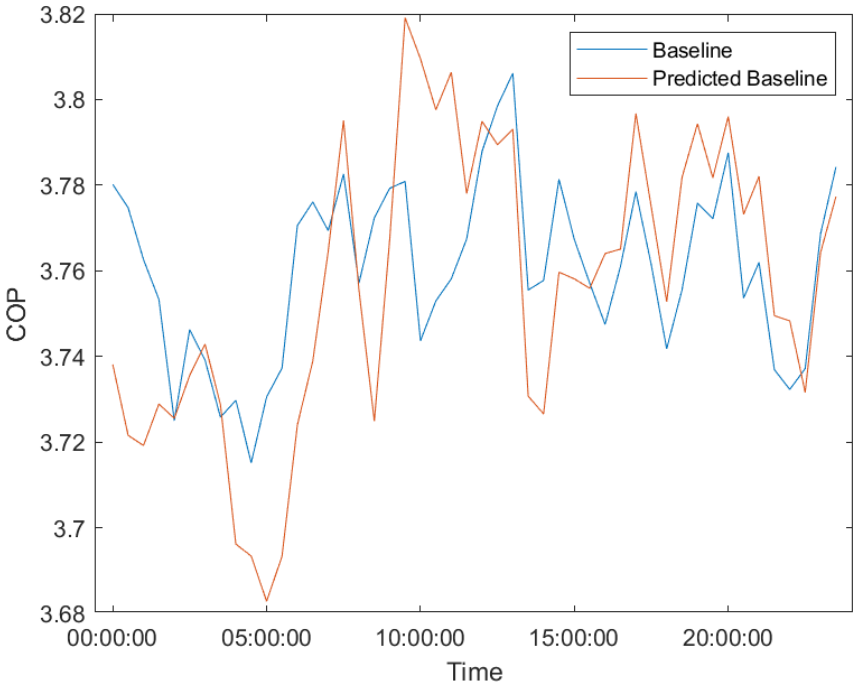


Figure 52: Baseline performance of the refrigeration system compared to the model's predicted baseline

The optimal COP was determined by providing the model with the optimal inputs as determined in the previous section. This value represents the theoretical best performance of the refrigeration system under ideal, controlled circumstances where each of the inputs can be fine-tuned. Using this method, it was found that an optimal COP of 6.58 can be achieved.

In practice, it is not possible to establish favourable conditions for all inputs. Inputs like wet-bulb temperature and the inlet temperatures of the various fluids of the system are dependent on environmental factors. The practically achievable optimal COP was therefore determined by providing as inputs to the model those variables that are controllable. These were: compressor power, water flow rate through the evaporator, and cooling water flow rate at the condenser.

The improved performance profile compared to the baseline performance profile of the refrigeration system over a typical day is given in Figure 53.

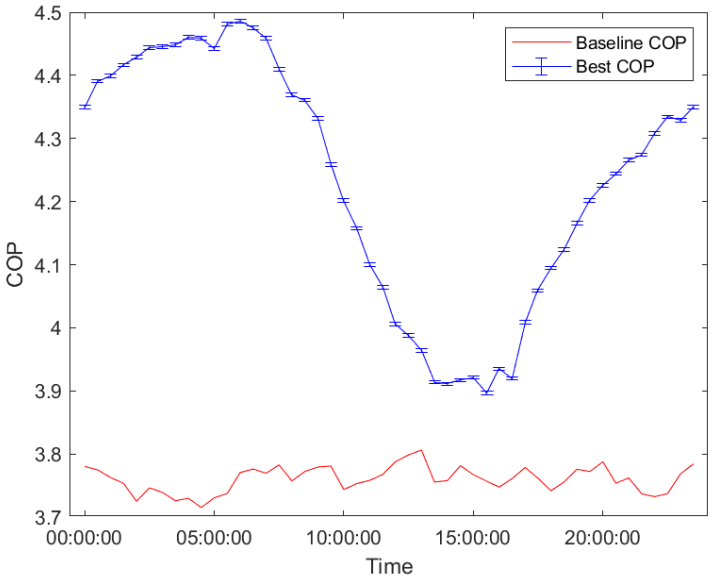


Figure 53: The baseline performance of the refrigeration plant compared to the improved performance profile

The model predicted an average improvement in COP of 0.52 compared to the baseline. The decrease in compressor power compared to the baseline was 130kW on average, which is equivalent to a reduction in energy consumption of 1.14GWh per annum. Using the 2024 Eskom tariff structure for the case study mine, this will result in savings of R1.8 million per annum.

Since the performance of the refrigeration plant is heavily influenced by ambient conditions, the optimal configurational of the parameters differ depending on the season. The process of finding the optimal point for each parameter was therefore repeated for each season and then implemented to predict the COP. Table 10 shows the optimal point for each of the controllable parameters, the average COP, and the predicted improvement in COP for each season.

Table 10: A summary of the seasonal performance improvement using optimal inputs

Season	Optimal flow at the evaporator (L/s)	Optimal flow at the condenser (L/s)	Optimal compressor power (MW)	Average COP	Predicted COP
Summer	342	339	1.27	3.72	4.15
Autumn	355	346	1.31	4.28	4.84
Winter	331	362	1.28	4.39	4.98
Spring	336	343	1.22	3.79	4.29

Lastly, the optimisation process was validated by searching the TCI class for periods of operation similar to those identified as optimal in this section. Few instances exist where the exact optimal configuration of parameters was present simultaneously, so the TCI class was filtered to allow an optimal range of 2% around each optimal point. Table 11 summarises the optimisation process. The range of optimal flow and compressor power is given. The predicted and actual COPs for these conditions are given as well as the MAPE of the predicted performance compared to the actual performance.

Table 11: Summary of the validation of the optimisation process

Season	Range of optimal flow at the evaporator (L/s)	Range of optimal flow at the condenser (L/s)	Range of optimal compressor power (MW)	Actual COP	Predicted COP	MAPE (%)
Summer	335 – 348	332 – 345	1.24 – 1.30	4.06	4.15	2.17
Autumn	347 – 362	349 – 352	1.28 – 1.33	4.61	4.84	4.75
Winter	324 – 337	354 – 369	1.25 – 1.31	4.74	4.98	4.82
Spring	329 – 342	336 – 349	1.20 – 1.24	4.11	4.29	4.20

These MAPE of the predicted values compared to the actual values range between 2% and 5%. The actual improvement in the COP was 0.19 compared to the predicted 0.52. The error margin shows that the model was able to accurately identify the optimal operating range of the refrigeration plant given the controllable parameters. A larger error is observed compared to the model which is likely due to the use of the optimal window which allowed more suboptimal conditions to prevail for the sake of a larger dataset for validation purposes.

3.7 Model Verification

To verify the model, the verification class was used. This class was left out of the training process to maintain its independence. This dataset represents unseen operational conditions of the refrigeration plant. The model was tested on this dataset to verify that it was correctly implemented and to verify its generalisation capabilities. The predicted outputs of the model using verification class inputs compared to its targets for the first 100 data patterns are given in Figure 54.

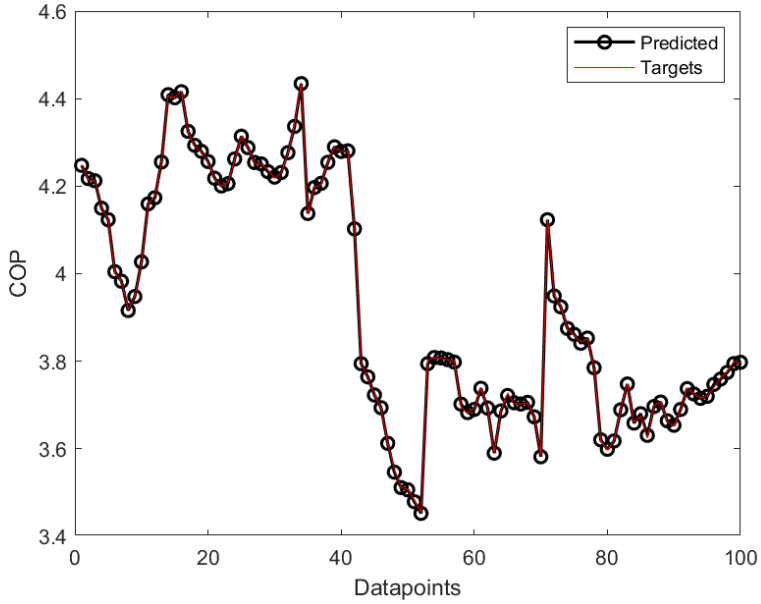


Figure 54: Model predictions compared to its targets for the first 100 data patterns of the verification class

The model accuracy with respect to the verification class was also tested with a regression analysis, given in Figure 55. The figure shows the target values against the model output values (predictions). The best fit line is shown in blue and can be compared to the ideal fit where Y (output) = T (target).

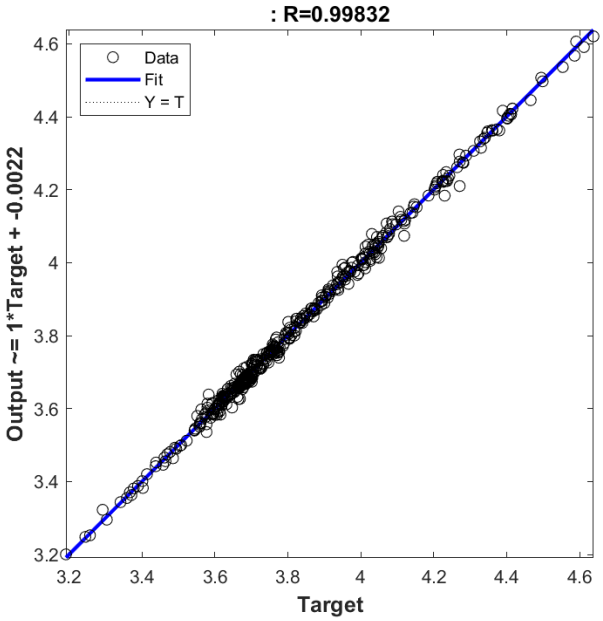
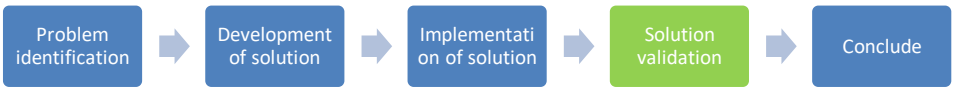


Figure 55: Regression plot showing correlation between the predicted outputs compared to its targets with respect to the verification class

The regression plot indicates a well-rounded model that generalises well across the entire spectrum of operational conditions. Some variation is observed near the middle of the spectrum, which suggests some aspect of the refrigeration system performance which is not accounted for by the model in this region. It can also be seen that, near the boundaries of the operational range, where, by nature of the data distribution, there exists less representation of the system performance from which the ANN could learn, the model can still adequately predict the refrigeration plan performance.

The coefficient of determination was used to determine how much variance in the model's prediction can be explained by the model inputs. An R^2-value of 0.998 was achieved, which suggests that the model can sufficiently explain the refrigeration system performance with the chosen inputs. Furthermore, the model achieved a MAPE of 2.55% and an RMSE of 0.0098.

3.8 Study Validation



The results discussed in this chapter were obtained from implementing the method described in Chapter 2. The method, in turn, was designed to meet the need of the study through the achievement of the study objectives. To validate the study, the results were compared to the objectives to see if the identified problem was solved as discussed in Table 12.

Table 12: Summary of the study objectives and how they were addressed

Objective	Section	Comments
1. Select appropriate input and output parameters	3.3	Input and output parameters were selected based on literature and the availability and reliability of data at the case study refrigeration plant. A total of 242 963 data points were gathered from 9 parameters (8 inputs and 1 output) from 1 October 2020 to 30 October 2023.

2. Apply pre-processing to improve data quality	3.4	The inequality-, noise spike-, and EWMA filters were implemented, which improved the data quality. The number of usable data patterns in the dataset improved from 18.7% to 100%.
3.1 Optimise the number of hidden layers	3.5.2 3.5.3	The optimal number of hidden layers was determined using R^2 and RMSE performance indicators. Through trial and error, the optimal network was found to consist of two hidden layers.
3.2 Optimise the number of hidden neurons	3.5.1	The R^2 and RMSE were used to find the optimal number of hidden neurons. Through trial-and-error, the optimal number of hidden neurons was 37 in the first hidden layer and 21 in the second hidden layer.
3.3 Optimise the training algorithm	3.5.2	The best training algorithm was determined using early stopping and the MSE performance indicator. The LM algorithm performed the best in terms of accuracy and convergence speed overall, achieving an MSE of 6.94×10^{-6} and converging in 551 epochs.
4.1 Derive fundamental input-output equations	3.6.2	The model was described using the weights and biases of the optimal network applied to the fundamental input-output equation given by Equation (52).
4.2 Use the model predictions to improve performance of the target parameter	3.6.3 3.6.4	The model was used to find the optimal operating point for each input parameter. Under these optimal conditions, the COP was improved by 0.52 on average and resulted in a cost reduction of R1.8 million using the 2024/2025 Eskom Megaflex tariffs.
5. Verify the model output by implementing it on unseen data	3.7	The model was verified by comparing its predictions to the actual outputs of an unseen-, independent data set. R^2-, RMSE, and MAPE values of 0.998, 0.0098, and 2.55% were achieved, respectively.

This study aimed to solve the problem of modelling a refrigeration system in a deep-level mine with a high degree of accuracy while accounting for deviations in design specifications, operational conditions, and low data quality. The result of the proposed method shows that this can be achieved using a multi-layer perceptron ANN. The model accurately describes the refrigeration plant performance over its entire operational range and generalises well to unseen data. The model can be easily updated with new training data since the optimal network architecture is already known.

The case study refrigeration plant is an example of a typical deep-level mine refrigeration system. This refers to its configuration, data availability, and data quality. The proposed method will therefore comfortably be able to account for differences in operational conditions of other systems.

3.9 Summary

This section presented the results of developing an ANN model for predicting the performance of a deep-level mine refrigeration system. The results of data pre-processing were given. This included the data consolidation and application of the inequality-, noise spike-, and exponentially weighted moving average filters on all parameters.

The results for the development of the ANN model were then given. The RMSE and R^2 performance indicators were used to determine the optimal number of hidden neurons for one- and two hidden layer architectures, trained with the LM and SCG training algorithms. The optimal architectures for each configuration were then retrained repetitively to find the best-performing training network with respect to the MSE performance indicator. It was found that the best-performing model was a two hidden layer ANN with 37 hidden neurons in its first hidden layer and 21 in its second hidden layer, trained by the LM training algorithm.

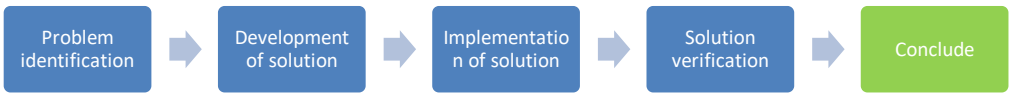
The prediction accuracy of the best-performing model was presented in terms of the RMSE-, R^2, and MAPE performance indicators on its TCI class subsets where values of 0.0026, 0.9999, and 0.6207 were achieved, respectively. The fundamental input-output equation of the model was derived using the optimal network weights and biases. The model was then implemented to determine the relationship between its inputs and the refrigeration system COP.

Using these results, the optimal conditions could be determined and used to predict that a theoretical optimal COP of 6.58 can be achieved in a controlled, ideal environment. The practically achievable optimal COP profile could also be determined by isolating the inputs that can be controlled practically. From this, it was found that an average increase in COP of the refrigeration

plant of 0.52 can be achieved. These conditions allow a reduction in compressor power of 130kW on average, resulting in annual energy savings of 1.14GWh and cost savings of R1.8 million per annum.

Finally, the model was verified by implementing it on the independent verification class. The model predictions were compared to their target values and regression analysis was done to determine the correlation of the datapoints. An R^2-value of 0.9983, and RMSE of 0.0098, and a MAPE of 2.55% was achieved, indicating that the model generalises well on unseen data and is an adequate tool to model the deep-level mine refrigeration system.

CHAPTER 4 CONCLUSION



4.1 Preamble

This chapter presents a summary of the background, objectives, method, and results of this study. The formulation of the problem and the solution to that problem is summarised as well as the results of implementing that solution with respect to the objective. The shortcomings of this study are given, and recommendations are made for further work that can be done. The study is then concluded with final remarks.

4.2 Summary

Electricity prices in South Africa have risen drastically over the last two decades. This has had major effects on energy-intensive industries such as mining, which consumes around 10% of the energy supply in the country. The largest energy consumers on typical deep-level gold mines are the energy systems responsible for cooling the mine. Most of these cooling systems are dependent on chilled water produced at a refrigeration plant to perform their cooling functions. Deep-level mines consequently prioritise optimising the performance of the refrigeration plant to improve the overall cooling capacity of the cooling system and to reduce energy costs.

To improve the performance of a refrigeration plant, it is beneficial to characterise the interactions between its component parts by creating a model of the system. This model can then be used to predict the response of the system to changes in its operational state to evaluate whether a proposed change would be beneficial. Techniques used to model deep-level mine refrigeration systems include first principal models and, more recently, simulation software. Both techniques have yielded good results but rely on a complete description of interactions between all components of the refrigeration system, which is a difficult and time-intensive process that often relies on the underlying system's design specifications which degrade over time and negatively impacts the accuracy of such models.

Empirical models such as artificial neural networks have been used to model refrigeration systems with a high degree of accuracy. These models are not reliant on complete descriptions of the underlying components of a refrigeration system to accurately predict its performance and can be

significantly less time consuming and easier to implement. Furthermore, artificial neural networks learn the relationships between parameters in their current state, so components deviating from their design specifications do not negatively impact the model's accuracy.

The aim of this study was to develop an artificial neural network to relate the operational parameters of a deep-level gold mine refrigeration plant to its coefficient of performance with a high degree of accuracy. Inputs to the model were chosen based on the availability of data at the case study refrigeration plant and their effective use in previous instances of refrigeration system modelling. The data was consolidated to ensure causality between input-output pairs. These parameters were then filtered to remove data from periods of non-operation and instrumentation error. Noise spike and exponentially weighted moving average filters were then applied to smoothen noisy data.

The predictive capability of an ANN is highly dependent on its architecture, which includes the number of hidden layers, number of hidden neurons, and the training algorithm. The optimal architecture for the ANN model was therefore determined using an iterative sensitivity analysis method. First, the optimal number of hidden neurons for one hidden layer networks were determined by varying the number of neurons between 1 and 200 and finding the configuration with the highest coefficient of determination and lowest root mean square error. Each configuration was trained using the Levenberg-Marquardt and scaled conjugate gradient training algorithms. A similar method was used to determine the optimal number of hidden neurons in the hidden layers of a two hidden layer ANN.

The configurations that yielded the best results were retrained 100 times with both training algorithms to determine the optimal training algorithm based on the lowest mean square error of the algorithm. The optimal network was subsequently found to be a two hidden layer ANN with 37 neurons in its first hidden layer and 21 hidden neurons in its second hidden layer, trained by the Levenberg-Marquardt algorithm. This network achieved coefficient of determination (R^2) values of 0.9999 on its training subset, 0.9998 on its validation subset, 0.9998 on its test subset, and 0.9999 overall. Furthermore, a minimum RMSE of 0.0026 and a minimum mean absolute percentage error of 0.6207 were achieved overall.

The fundamental input-output equation of the ANN was derived using the weights and biases of the optimal ANN. The model was implemented to establish the relationships between its inputs and output, the COP of the refrigeration system. The results of this process were then used to determine that a theoretical optimal COP of 6.58 can be achieved if the refrigeration system was placed in a controlled environment where the optimal conditions can be enforced. The practically

achievable optimal COP was also determined, which represents the best performance of the refrigeration system if the practically controllable parameters were maintained at their optimal points. Compared to the refrigeration plant's baseline performance, an average increase in COP of 0.52 was observed and a decrease in power demand of 130 kW. This is equivalent to 1.14GWh per annum or, using the 2024 Eskom energy tariff structure, R1.8 million per annum.

Finally, the model was verified by implementing it on the independent verification dataset. The model predictions were compared to their target values and regression analysis was done to determine the correlation of the datapoints. An R^2-value of 0.9983, an RMSE of 0.0098, and a MAPE value of 2.55% was achieved, indicating that the model generalised well on unseen data and is an adequate tool to model the case study deep-level mine refrigeration system.

4.3 Achieving the objectives

The proposed method was able to meet the study objectives and solve the identified problem. The results show that the refrigeration system can be modelled with a high degree of accuracy while accounting for the problems found in typical refrigeration systems with respect to operational range and data quality. The model was able to improve the COP of the refrigeration system and reduce the compressor energy consumption.

The accuracy of the model suggests that there is more potential for understanding the broader cooling system performance by introducing more outputs to the model. This would allow a better understanding of the effect of the system interactions on, for example, chilled water temperature, energy consumption, or auxiliary cooling system performance, which gives more scope for optimisation of these systems and a more involved control philosophy.

Additionally, the ANN models quickly understood the refrigeration system interactions, indicating that its computational power would be better challenged by attempting to model a more complex, involved system such as the aforementioned auxiliary cooling systems. More target variables can be added to the model to explore the effects of, for example, energy consumption of the compressors, energy cost, and outlet water temperature.

4.4 Limitations of the study

The limitations of this study were:

- The model was limited to the case study mine and the data stored on its internal database, meaning that the choice of parameters to train the model was limited. The model and its

implementation can be improved by including more parameters as inputs and outputs to gain a greater understanding of their effect on the refrigeration plant's performance.

- Refrigeration systems are large, integrated systems that rely on auxiliary systems such as the condenser cooling system and the dynamics of intermixing at intermediate dams in the evaporator circuit. Including parameters from these two auxiliary systems, for example, would allow a deeper understanding of how these processes influence and can be adapted to the benefit of the refrigeration plant's COP.
- Due to computation constraints, the training process was limited to varying the number of neurons between 1 and 200 for one hidden layer variations, and between 1 and 40 for the neurons in two hidden layer networks. These ANNs did not display the inflection point (that indicates that overfitting has started) in their performance indicators , meaning that the optimal architecture for this problem could lie outside of this range and an even more accurate model can be developed, although a point of diminishing returns would quickly be reached.

4.5 Recommendations

Artificial neural network modelling in the mining industry has rarely been utilised. As a result, there is a lot of potential for exploiting this technique with the goal of improving mine energy systems, which has not yet been attempted. The following recommendations are directed at addressing this gap in research:

- Artificial intelligence optimisation techniques have been used in related work on refrigeration systems on deep-level mines with the goal of improving their performance. The characteristic models of these system were, however, developed from first principles and are relatively inaccurate and time-consuming to develop. ANN models, once developed, can be adapted to different systems relatively quickly to high degrees of accuracy and can improve the effectiveness of the optimisation techniques that rely on these models.
- ANNs are a versatile technique which can be applied to a variety of other applications on deep-level mines. If enough data is available, ANN models can be created of various other components in the cooling system to characterise the interactions between these respective components, which can, again, be combined with optimisation techniques to improve the overall cooling system performance.

- ANN models of energy systems such as refrigeration plants and compressors can be used in conjunction with simulation models to improve computational speed and enhance accuracy.
- More advanced AI techniques such as ANFIS are showing promise for combining the accuracy of ANN modelling and fuzzy logic for optimisation purposes and is worth further exploration for application in the mining industry.

LIST OF REFERENCES

[1] D. I. Stern, “The role of energy in economic growth”, *Annals of the New York Academy of Science*, vol. 1219, no. 1, pp. 26–51, 2011

[2] L. Flepisi and C. Mlambo, “Factors influencing the late delivery of projects in state-owned enterprises: The case of Eskom”, *South African Journal of Industrial Engineering*, vol. 32, no. 4, pp. 57–66, 2021

[3] N. Marta and T. Agnieszka, “Load shedding and the energy security of Republic of South Africa”, *Journal of Polish Safety and Reliability Association Summer Safety and Reliability Seminars*, vol. 6, no. 3, pp. 99–108, 2015.

[4] A. Bowman, “Parastatals and economic transformation in South Africa: The political economy of the Eskom crisis”, *African Affairs (London)*, vol. 119, no. 476, pp. 395–431, 2021

[5] S. Moolman, “2024 update: Eskom tariff increases vs inflation since 1988 (with projections to 2026).” Accessed: Aug. 13, 2024. [Online]. Available: https://poweroptimal.com/2024-update-eskom-tariff-increases-vs-inflation-since-1988-with-projections-to-2026/

[6] K. Walsh, R. Theron, and C. Reeders, “Estimating the Economic Cost of Load Shedding in South Africa”, Stellenbosch, 2021.

[7] A.-M. Bercu, G. Paraschiv, and D. Lupu, “Investigating the Energy–Economic Growth–Governance Nexus: Evidence from Central and Eastern European Countries”, *Sustainability*, vol. 11, no. 12, pp. 3355–3376, Jun. 2019

[8] B. G. Pollet, I. Staffell, and K. A. Adamson, “Current energy landscape in the Republic of South Africa”, *International Journal of Hydrogen Energy*, vol. 40, no. 46, pp. 1–17, Dec. 2015

[9] I. Watson and M. Olalde, “The state of mine closure in South Africa-what the numbers say”, *The Journal of the Southern African Institute of Mining and Metallurgy*, vol. 119, pp. 639–645, Jul. 2019

[10] Statistics SA, “Statistical Release P0441: Gross domestic product - First quarter 2023”, Pretoria, Sep. 2023. [Online]. Available: www.statssa.gov.za,info@statssa.gov.za,Tel+27123108911

[11] Statistics South Africa, “Statistical Release P0277: Quarterly employment statistics - March 2023”, Pretoria, Sep. 2023. [Online]. Available: www.statssa.gov.za,info@statssa.gov.za,Tel+27123108911

[12] K. Ratshomo, “The South African Energy Sector Report”, Pretoria, 2022.

[13] C. Musingwini, “Introduction of specialisation in mine planning and optimisation”, *The Southern African Institute of Mining and Metallurgy*, pp. 23–27, 2014.

[14] P. G. Ranjith, J. Zhao, M. Ju, R. V. S. De Silva, T. D. Rathnaweera, and A. K. M. S. Bandara, “Opportunities and Challenges in Deep Mining: A Brief Review”, *Engineering*, vol. 3, no. 4, pp. 546–551, Aug. 2017

[15] J. H. Marais, “Evaluating the impact of energy management on deep-level mines during medium-term production stoppages”, North-West University, Potchefstroom, 2021.

[16] C. Cilliers, “Benchmarking electricity use on deep-level mines”, Department of Mechanical Engineering, North-West University, Potchefstroom, 2015.

[17] Z. Szymański, “Intelligent, energy saving power supply and control system of hoisting mine machine with compact and hybrid drive system”, *Archives of Mining Sciences*, vol. 60, no. 1, pp. 239–251, 2015

[18] M. D. Harmse, “Optimising mining refrigeration systems through artificial intelligence”, Potchefstroom, Jun. 2021.

[19] D. Nell, “Practical determination of heat loads for existing deep level gold mines”, Department of Mechanical Engineering, North-West University, Potchefstroom, 2019.

[20] H. J. van Staden, J. F. van Rensburg, and H. J. Groenewald, “Optimal use of mobile cooling units in a deep-level gold mine”, *International Journal of Mining Science and Technology*, vol. 30, no. 4, pp. 547–553, Jul. 2020

[21] J. G. Pretorius, M. J. Mathews, P. Maré, M. Kleingeld, and J. van Rensburg, “Implementing a DIKW model on a deep mine cooling system”, *International Journal of Mining Science and Technology*, vol. 29, no. 2, pp. 319–326, Mar. 2019

[22] J. Van Der Walt and E. M. De Kock, “Developments in the engineering of refrigeration installations for cooling mines”, *International Journal of Refrigeration*, vol. 7, no. 1, pp. 27–40, 1984.

[23] A. Greth and C. Kocsis, “A Review of Cooling System Practices and Their Applicability to Deep and Hot Underground US Mines”, Golden, Colorado, Jun. 2017.

[24] M. Hooman, R. C. W. Webber-Youngman, J. J. L. du Plessis, and W. M. Marx, “A decision analysis guideline for underground bulk air heat exchanger design specifications Engineering and technical requirements”, *The Journal of the Southern African Institute of Mining and Metallurgy*, vol. 115, pp. 125–129, Feb. 2015.

[25] P. Maré, “Novel simulations for energy management of mine cooling systems”, Mechanical, North-West University, Potchefstroom, 2017.

[26] P. F. H. Peach, J. I. G. Bredenkamp, and J. F. van Rensburg, “Dynamic optimisation of deep-level mine refrigeration control”, *South African Journal of Industrial Engineering*, vol. 29, no. 3 Special Edition, pp. 261–270, 2018

[27] Y. A. Cengel and M. A. Boles, *Thermodynamics: An Engineering Approach*, 8th ed. New York: McGraw-Hill Education, 2015.

[28] A. Kamyar, S. Mostafa Aminossadati, C. Leonardi, and A. Sasmito, “Current Developments and Challenges of Underground Mine Ventilation and Cooling Methods”, Wollongong, Feb. 2016

[29] R. Ahmed, S. Mahadzir, N. Erniza B Rozali, K. Biswas, F. Matovu, and K. Ahmed, “Artificial intelligence techniques in refrigeration system modelling and optimization: A multi-disciplinary review”, *Sustainable Energy Technologies and Assessments*, vol. 47, Oct. 2021

[30] Z. Afroz, G. M. Shafiullah, T. Urmee, and G. Higgins, “Modeling techniques used in building HVAC control systems: A review”, *Renewable and Sustainable Energy Reviews*, vol. 83, pp. 64–84, Mar. 2018

[31] S. Estrada-Flores, I. Merts, B. De Ketelaere, and J. Lammertyn, “Development and validation of ‘grey-box’ models for refrigeration applications: A review of key concepts”, *International Journal of Refrigeration*, vol. 29, no. 6, pp. 931–946, Sep. 2006

[32] M. Le Roux, “Identifying water wastage in deep-level mines using novel simulation-based zero-waste baselines”, Stellenbosch University, Stellenbosch, 2024

[33] J. Vermeulen, “Simplified high-level investigation methodology for energy saving initiatives on deep-level mine compressed air systems”, PhD, North-West University, Potchefstroom, 2018.

[34] M. Pérez-Gomariz, A. López-Gómez, and F. Cerdán-Cartagena, “Artificial Neural Networks as Artificial Intelligence Technique for Energy Saving in Refrigeration Systems—A Review”, *Clean Technologies*, vol. 5, no. 1, pp. 116–136, Mar. 2023

[35] D. S. Adelekan, O. S. Ohunakin, and B. S. Paul, “Artificial intelligence models for refrigeration, air conditioning and heat pump systems”, *Energy Reports*, vol. 8, pp. 8451–8466, Nov. 2022

[36] J. Navarro-Esbrí, V. Berbegall, G. Verdu, R. Cabello, and R. Llopis, “A low data requirement model of a variable-speed vapour compression refrigeration system based on neural networks”, *International Journal of Refrigeration*, vol. 30, no. 8, pp. 1452–1459, Dec. 2007

[37] M. Hosoz, H. M. Ertunc, and H. Bulgurcu, “An adaptive neuro-fuzzy inference system model for predicting the performance of a refrigeration system with a cooling tower”, *Expert Systems with Applications*, vol. 38, no. 11, pp. 14148–14155, Oct. 2011

[38] J. M. Belman-Flores, D. A. Rodríguez-Valderrama, S. Ledesma, J. J. García-Pabón, D. Hernández, and D. M. Pardo-Cely, “A Review on Applications of Fuzzy Logic Control for Refrigeration Systems”, Feb. 01, 2022, *MDPI*.

[39] A. Mosavi, M. Salimi, S. F. Ardabili, T. Rabczuk, S. Shamshirband, and A. R. Varkonyi-Koczy, “State of the art of machine learning models in energy systems, a systematic review”, *Energies (Basel)*, vol. 12, no. 7, Apr. 2019

[40] C. C. Aggarwal, *Neural Networks and Deep Learning*, 1st ed. Cham: Springer, 2018.

[41] O. Isaac Abiodun, A. Jantan, A. Esther Omolara, K. Victoria Dada, N. AbdElatif Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey”, *Heliyon*, vol. 4, p. 938, 2018

[42] Surya Engineering College and Institute of Electrical and Electronics Engineers, *Proceedings of the 3rd International Conference on Computing Methodologies and Communication (ICCMC 2019) : 27-29, March 2019*.

[43] D. V Raghunatha Reddy, P. Bhramara, and K. Govindarajulu, “A Comparative Study of Multiple Regression and Artificial Neural Network models for a domestic refrigeration system with a hydrocarbon refrigerant mixtures”, in *Materials Today: Proceedings*, 2020, pp. 1545–1553

[44] M. Popescu, L. Perescu-Popescu, V. Balas, and N. Mastorakis, “Multilayer Perceptron and Neural Networks”, *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, Jul. 2009.

[45] J. Pretorius, “Prediction model for the performance of a methane-fuelled spark-ignition engine at a deep-level mine”, Potchefstroom, 2022.

[46] J. Patterson and A. Gibson, *Deep Learning: A Practitioners Approach*, 1st ed. Sebastopol: O’Reilly Media, Inc, 2017.

[47] N. Gupta, “Network and Complex Systems Artificial Neural Network”, *Network and Complex Systems*, vol. 3, no. 1, pp. 24–28, 2013

[48] P. A. Jansson, “Neural Networks: An Overview”, *Analytical Chemistry Journal*, vol. 63, no. 6, pp. 357–362, Mar. 1991.

[49] J. Singh and R. Banerjee, “A study on single and multi-layer perceptron neural network,” in *Proceedings of the Third International Conference on Computing Technologies and Communication*, 2019, pp. 35–40.

[50] S. Sharma, S. Sharma, and A. Athaiya, “Activation functions in neural networks”, *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310–316, 2020

[51] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, “A survey on modern trainable activation functions”, *Neural Networks*, vol. 138, pp. 14–32, Jun. 2021

[52] A.-N. Sharkawy, “Principle of Neural Network and Its Main Types: Review”, *Journal of Advances in Applied & Computational Mathematics*, vol. 7, pp. 8–19, 2020.

[53] A. Burkov, *The Hundred-Page Machine Learning*. Andriy Burkov, 2019.

[54] A. Joshi, J. Sasumana, N. M. Ray, and V. Kaushik, “Neural Network Analysis”, in *Advances in Bioinformatics*, Springer Nature, 2021, pp. 351–364

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. MIT Press, 2016.

[56] Ö. Kizilkan, “Thermodynamic analysis of variable speed refrigeration system using artificial neural networks”, *Expert Systems with Applications*, vol. 38, no. 9, pp. 11686–11692, 2011

[57] S. Abreu, “Automated Architecture Design for Deep Neural Networks”, Aug. 2019.

[58] A. Jagtap, K. Kawaguchi, and G. Karniadakis, “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks”, *Journal of Computational Physics*, 2019.

[59] A. Joshi, *Machine Learning and Artificial Intelligence*, 1st ed. Cham: Springer, 2020.

[60] A. C. Cinar, “Training Feed-Forward Multi-Layer Perceptron Artificial Neural Networks with a Tree-Seed Algorithm”, *Arabian Journal of Science and Engineering*, vol. 45, no. 12, pp. 10915–10938, Dec. 2020

[61] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou, and M. Ettaouil, “Multilayer Perceptron: Architecture Optimization and Training”, *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 1, pp. 26–30, 2016

[62] V. K. Ojha, A. Abraham, and V. Snášel, “Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research”, *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 97–116, May 2017

[63] M. Sazli, “A brief review of feedforward neural networks”, *Communications Faculty of Science University of Ankara*, vol. 50, no. 1, pp. 11–17, 2006.

[64] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 2005.

[65] J. Bilski, B. Kowalczyk, M. Kisiel-Dorohinicki, A. Siwocha, and J. Zurada, “Towards a Very Fast Feedforward Multilayer Neural Networks Training Algorithm”, *Journal of Artificial Intelligence and Soft Computing Research*, vol. 12, no. 3, pp. 181–195, Jul. 2022

[66] J. C. R. Whittington and R. Bogacz, “Theories of Error Back-Propagation in the Brain”, *Trends in Cognitive Science*, vol. 23, no. 3, pp. 235–250, Mar. 2019

[67] K. L. Priddy and P. E. Keller, *Artificial Neural Networks: An Introduction*, vol. TT68. Bellingham, Washington: The International Society for Optical Engineering, 2005.

[68] A. Ranganathan, “The Levenberg-Marquardt Algorithm”, Jun. 2004.

[69] H. P. Gavin, “The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems”, 2022.

[70] M. I. A Lourakis, “A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar”, Heraklion, Feb. 2005.

[71] M. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares”, Chicago, Feb. 1943.

[72] D. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”, *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, Jun. 1963.

[73] M. F. Moller, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, Computer Science Department Aarhus University, Aarhus, Nov. 1990.

[74] M. F. Moller, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, *Neural Networks*, vol. 6, pp. 525–533, 1993.

[75] A. E. Kostopoulos and T. N. Grapsa, “Self-scaled conjugate gradient training algorithms”, *Neurocomputing*, vol. 72, no. 13–15, pp. 3000–3019, 2009

[76] B. Cetişli and A. Barkana, “Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training”, *Soft computing*, vol. 14, no. 4, pp. 365–378, 2010

[77] H. D. Khalaf Jabbar Rafiqul Zaman Khan, “Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)”, in *Computer Science, Communication & Instrumentation Devices*, J. Stephen, H. Rohil, and S. Vasavi, Eds., 2015, pp. 163–172.

[78] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, “Theory and Methodology Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis”, *European Journal of Operational Research*, vol. 116, pp. 16–32, 1999.

[79] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[80] K. Yotov, E. Hadzhikolev, S. Hadzhikoleva, and S. Cheresharov, “A Method for Extrapolating Continuous Functions by Generating New Training Samples for Feedforward Artificial Neural Networks”, *Axioms*, vol. 12, no. 759, pp. 1–22, Aug. 2023

[81] G. Panchal, A. Ganatra, Y. P. Kosta, and D. Panchal, “Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers”, *International Journal of Computer Theory and Engineering*, vol. 3, no. 2, pp. 332–337, 2011

[82] B. Hanin, “Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?”, in *Neural Information Processing Systems*, 2018, pp. 1–10.

[83] P. Patel, M. Nandu, P. Raut, and A. Professor, “Initialization of Weights in Neural Networks”, *International Journal of Scientific Development and Research*, vol. 3, no. 11, pp. 73–79, Feb. 2019

[84] G. Cybenko, “Approximation by Superpositions of a Sigmoidal Function”, *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 1989.

[85] K. Hornik, “Multilayer feedforward networks are universal approximators”, *Neural Networks*, vol. 2, pp. 359–366, 1989.

[86] B. Macukow, “Neural networks - state of art, brief history, basic models and architecture”, in *IFIP International Conference on Computer Information Systems and Industrial Management*, K. Saeed and W. Homenda, Eds., Springer International Publishing Switzerland, 2016, pp. 3–14

[87] Y. Le Cun, J. S. Denker, and S. A. Sol, “Optimal Brain Damage”, in *Neural Information Processing Systems*, Holmdel, 1989.

[88] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal Brain Surgeon”, in *Neural Information Processing Systems*, Stanford, Nov. 1992.

[89] K. G. Sheela and S. N. Deepa, “Review on methods to fix number of hidden neurons in neural networks”, *Mathematical Problems in Engineering*, vol. 2013, pp. 1–11, Jun. 2013

[90] H. M. Ertunc and M. Hosoz, “Artificial neural network analysis of a refrigeration system with an evaporative condenser”, *Applied Thermal Engineering*, vol. 26, no. 5–6, pp. 627–635, Apr. 2006

[91] Ö. Kizilkan, “Thermodynamic analysis of variable speed refrigeration system using artificial neural networks”, *Expert Systems with Applications*, vol. 38, no. 9, pp. 11686–11692, 2011

[92] M. Hosoz and H. M. Ertunc, “Modelling of a cascade refrigeration system using artificial neural network”, *International Journal of Energy Research*, vol. 30, no. 14, pp. 1200–1215, Nov. 2006

[93] A. Şencan, “Performance of ammonia-water refrigeration systems using artificial neural networks”, *Renewable Energy*, vol. 32, no. 2, pp. 314–328, Feb. 2007

[94] M. C. Furumele, “Dynamic control of deep-level mine cooling systems using artificial intelligence”, Potchefstroom, 2024.

[95] J. M. Belman-Flores, S. E. Ledesma, M. G. Garcia, J. Ruiz, and J. L. Rodríguez-Muñoz, “Analysis of a variable speed vapor compression system using artificial neural networks”, *Expert Systems with Applications*, vol. 40, no. 11, pp. 4362–4369, Sep. 2013

[96] A. Ş. Şahin, “Performance analysis of single-stage refrigeration system with internal heat exchanger using neural network and neuro-fuzzy”, *Renewable Energy*, vol. 36, no. 10, pp. 2747–2752, Oct. 2011

[97] J. Gill, J. Singh, O. S. Ohunakin, and D. S. Adelekan, “Energy analysis of a domestic refrigerator system with ANN using LPG/TiO 2 –lubricant as replacement for R134a”, *Journal of Thermal Analysis and Calorimetry*, vol. 135, no. 1, pp. 475–488, Jan. 2019

[98] D. V Raghunatha Reddy, P. Bhramara, K. Govindarajulu, and R. Reddy, “A Comparative Study of Multiple Regression and Artificial Neural Network models for a domestic refrigeration system with a hydrocarbon refrigerant mixtures”, in *Materials Today: Proceedings 22*, 2020, pp. 1545–1553

[99] J. Gill, J. Singh, O. S. Ohunakin, and D. S. Adelekan, “ANN approach for irreversibility analysis of vapor compression refrigeration system using R134a/LPG blend as replacement of R134a”, *Journal of Thermal Analysis and Calorimetry*, vol. 135, no. 4, pp. 2495–2511, Feb. 2019

[100] P. Baxter and S. Jack, “Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers”, 2008

[101] A. Saltelli, S. Tarantola, and K. P.-S. Chan, “A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output”, *American Statistical Association and the American Society for Quality Technometrics*, vol. 41, no. 1, pp. 39–56, 1999.

[102] S. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle, *Process Dynamics and Control*, 3rd ed. John Wiley & Sons, Inc, 2011.

[103] J. S. Hunter, “The Exponentially Weighted Moving Average”, *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, Oct. 1986

[104] B.-S. Kwon, R.-J. Park, S.-W. Jo, and K.-B. Song, “Analysis of Short-Term Load Forecasting Using Artificial Neural Network Algorithm According to Normalization and Selection of Input Data on Weekdays”, in *2018 IEEE PES Asia-Pacific Power and Energy Engineering Conference*, 2018, pp. 280–283.

[105] M. W. Browne, “Cross-Validation Methods”, *Journal of Mathematical Psychology*, vol. 44, pp. 108–132, 2000

[106] D. J. Swider, M. W. Browne, P. K. Bansal, and V. Kecman, “Modelling of vapour-compression liquid chillers with neural networks”, *Applied Thermal Engineering*, vol. 21, pp. 311–329, 2001

[107] M. A. Shahin, H. R. Maier, and M. B. Jaksa, “Predicting Settlement of Shallow Foundations using Neural Networks”, *Journal of Geotechnical and Geoenvironmental Engineering*, vol. 128, no. 9, pp. 785–793, Sep. 2002

APPENDIX A: ANN TRAINING

This appendix covers the derivation of the MSE cost function and shows some of the most common cost functions and their applicability to specific problems. The full backpropagation procedure is given to supplement the discussion from the main body of the report. Finally, the full derivation of the Levenberg-Marquardt and scaled conjugate gradient algorithms are presented

Appendix A1: The most common cost functions and their applications

The goal of training algorithms using gradient-based optimisation is to minimise the error between the desired output (y_i) and the output of the model ($\hat{y}_i$) [62]. The error (e_i) at the i-th output neuron for any given training example (n) is defined as [63]:

$$e_i(n) = \hat{y}_i(n) - y_i(n) \tag{A1}$$

The instantaneous error energy (ε_i) of the i-th output neuron is then given by:

$$\varepsilon_i(n) = \frac{1}{2} e_i^2(n) \tag{A2}$$

In ANN architectures with k output neurons, the instantaneous error energy for the whole output then becomes:

$$\varepsilon(n) = \frac{1}{2} \sum_{i=1}^{k} e_i^2(n) \tag{A3}$$

Equation (A3) represents the total error of an ANN's output layer for a single training example. In the training subset, there are (N) training examples, so the average error (ε_{av}) over the entire training subset can be calculated as:

$$\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^{N} \varepsilon(n) \tag{A4}$$

Equation (A4) is known as the mean squared error (MSE) cost function and is commonly used for regression models that require a real-valued output [64]. However, there are more cost functions

that can be derived in a similar manner to the MSE cost function. Table 13 shows some of the most used cost functions and their equations [46].

Table 13: Common cost functions and their mathematical formulas (adapted from [46])

Cost function	Equation	No
Mean absolute error (MAE)	$\varepsilon_{av} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{i=1}^{k}\lvert\hat{y}_i - y_i\rvert$	(A5)
Mean squared log error (MSLE)	$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{k}[\log(\hat{y}_i) - \log(y_i)]^2$	(A6)
Mean absolute percentage error (MAPE)	$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{k}\frac{\lvert\hat{y}_i - y_i\rvert \times 100}{y_i}$	(A7)

When the ANN model predicts more than one label, and the predictions have a large variation in range, the MSE and MAE cost functions tend to penalise the error of the larger output. However, as can be seen in Equations (A5) – (A7), the MSLE and MAPE cost functions make the errors relative and therefore are a more balanced choice for ANN models predicting more than one output [46].

Appendix A2: The full backpropagation procedure

The full procedure for the backpropagation algorithm, applied to a feedforward multi-layer perception is given in Figure 56.

Step 1.	Initialize Weights w_{ij}^{ℓ} = Uniformly Random Over $-\varepsilon$ to ε for all i, j, and ℓ.
Step 2.	Pick a labeled pattern (input and target) from the training set and present input pattern to the network. input = x_{pi} target = t_{pj} for $i = 1$ to N^0 (all input nodes) for $j = 1$ to N^L (all output neurons) for $p = 1$ to P (all patterns)
Step 3.	Propagate data forward and generate the output pattern. output = y_{pj} for $j = 1$ to N^L (all output neurons)
Step 4.	Calculate error between target and actual output by using Eq. (A.8).
Step 5.	Propagate error backwards through network and calculate changes to the synaptic weights that will reduce output error by using the weight-update formula, either with or without momentum, as given by Eq. (A.27) or Eq. (A.31).
Step 5a.	If batch mode, do not apply the weight changes until Step 7.
Step 5b.	If incremental mode, apply the weight changes.
Step 6.	If there are more patterns (i.e., $p < P$) in the training set, loop back to Step 2.
Step 7a.	If batch mode, update synaptic-weight values in the network by using the summation of all weight changes from all pattern presentations as given by Eq. (A.28).
Step 8.	If output error is high or the maximum number of iterations has not been met, then loop back to Step 2.

Figure 56: A stepwise procedure for applying the backpropagation algorithm to a feedforward multilayer perceptron [67]

Appendix A3: The Levenberg-Marquardt training algorithm derivation and procedure

The matrix form of the gradient descent update rule applied to the sum-of-squares cost function of a generalised multilayer perceptron is given by Equation (A8) [68].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \lambda \nabla \boldsymbol{\varepsilon}(\boldsymbol{w}) \tag{A8}$$

The network's weight values are stored in a weight vector $\boldsymbol{w}$ and, using Equation (A8), the updated weight values ($\boldsymbol{w}_{i+1}$) are determined by taking the current weight values ($\boldsymbol{w}_i$) and subtracting the product of a scaling factor (λ) and a gradient vector ($\nabla \boldsymbol{\varepsilon}(\boldsymbol{w})$) [40]. The gradient vector consists of the first order gradients of the error function and can be represented using the Jacobian matrix ($\boldsymbol{J}$) and the vector of errors or residuals ($\boldsymbol{r}$) as shown in Equation (A9) [70].

$$\nabla \boldsymbol{\varepsilon}(\boldsymbol{w}) = \boldsymbol{g} = \boldsymbol{J}^T \boldsymbol{r} \tag{A9}$$

The elements of the Jacobian matrix are the first order partial derivatives of the cost function with respect to the weights, given by Equation (A10).

$$\boldsymbol{J} \equiv \frac{\partial \boldsymbol{\varepsilon}}{\partial \boldsymbol{w}} \tag{A10}$$

Using this, the gradient descent update rule can be simplified into Equation (A11).

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \lambda \boldsymbol{J}^T \boldsymbol{r} \tag{A11}$$

The update rule of gradient descent uses gradient information ($\boldsymbol{J}^T \boldsymbol{r}$) of the error surface to update the parameters in the direction of steepest descent with the goal of finding the minimum of the cost function. An assumption of local linearity is therefore made, which simplifies the optimisation process, but often does not hold over the entire error surface [69]. The scaling factor is often referred to as the learning rate and determines the step size of each iteration. A small learning rate provides smooth convergence but increases the time it takes to find the minimum, while a large learning rate can converge faster but is susceptible to overshooting the optimal point [68].

Gradient descent is versatile and very efficient even when dealing with functions with many parameters, which is often the case in large ANNs [69]. However, it suffers from various convergence problems. When large gradients are present on the error surface, the step size tends to be large and when small gradients are present, the step size tends to be small. This can lead to the algorithm getting stuck in saddle points and overshooting the optimal point. This issue can be improved if both gradient and curvature information from the second order derivative is used to determine the step size [68]. One algorithm which exploits this is the Gauss-Newton method.

The matrix form of the update rule for the quasi-Newton method, applied to the sum-of-squares cost function of a generalised multilayer perceptron is given by Equation (A12) [68]:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\nabla^2 \boldsymbol{\varepsilon}(\boldsymbol{w}_i))^{-1} \nabla \boldsymbol{\varepsilon}(\boldsymbol{w}_i) \tag{A12}$$

Equation (A12) resembles gradient descent with some key differences. The learning rate has been replaced because the step size is now determined by the curvature of the cost function. The second order gradient vector ($\nabla^2 \boldsymbol{\varepsilon}(\boldsymbol{w}_i)$) contains the curvature information of the cost function and is often referred to as the Hessian matrix ($\boldsymbol{H}$). The Hessian matrix can be approximated by Equation (A13) through a special property of nonlinear least squares optimisation problems using the Jacobian [68].

$$\mathbf{H} \approx \nabla^2 \boldsymbol{\varepsilon}(\boldsymbol{w}) = \boldsymbol{J}^T \boldsymbol{J} \tag{A13}$$

This approximation assumes that the error surface is quadratic around the initial values of the weight vector $\boldsymbol{w}_0$. This assumption comes with the advantage of rapid convergence, but is sensitive to initial conditions, specifically the linearity of the surface around $\boldsymbol{\varepsilon}(\boldsymbol{w}_0)$ [64]. Equation (A12) can now be simplified by substituting the approximation of the Hessian and the Jacobian to give Equation (A14).

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \boldsymbol{r} \tag{A14}$$

Levenberg [71] capitalised on the complimentary nature of gradient descent and the Gauss-Newton method to create the Levenberg update rule, given by Equation (A15). The Levenberg update rule is designed to promote the advantages of either gradient descent or the Gauss-Newton method and diminish the other using the adaptive learning rate (λ) and the identity matrix ($\boldsymbol{I}$) as the regularisation term [68].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{H} + \lambda \boldsymbol{I})^{-1} \boldsymbol{J}^T \boldsymbol{r} \tag{A15}$$

The value of the adaptive learning rate is determined based on the proximity to the optimal solution. If the error is increasing, the Levenberg approximation favours gradient descent. Conversely, if the error is decreasing, Gauss-Newton is favoured. This process is summarised in Table 14.

Table 14: Summary of the Levenberg algorithm optimisation process

Δ Error	Δ Learning rate	Levenberg approximation	Bias
Decrease	$\lambda_{new} = \frac{1}{10}\lambda_{current}$	$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{H})^{-1}\boldsymbol{J}^T\boldsymbol{r}, \quad \lambda \approx 0$	Gauss-Newton
Increase	$\lambda_{new} = 10\lambda_{current}$	$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - (\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\boldsymbol{J}^T\boldsymbol{r}, \quad \lambda \gg 0$	Gradient descent

Appendix A4: The SCG training algorithm derivation and procedure

An abridged description of the conjugate gradient and SCG algorithms was presented in Section 1.5.4. The full description is given here.

In the context of finding the minimum of a cost function of a multilayer perceptron, most optimisation algorithms generally follow a similar procedure. First, a weight vector is initialised, and its corresponding cost function is calculated. Next, a search direction is chosen along the error surface and the step size in that direction is determined. Finally, the weight vector is updated with new values and the procedure continues until a satisfactory error is obtained [64].

Gradient descent does not employ the best method for choosing the search direction along the error surface nor the size of the step to be taken in that direction [64]. Some parameter optimisation algorithms such as Levenberg-Marquardt address this problem by incorporating curvature information to improve the selection of the step size, but rely on the use of successive gradient vectors to determine the search direction. Other algorithms aim to address both problems.

The conjugate gradient algorithm is one of these algorithms and is based on the observation that the use of successive gradient vectors as the search direction is a suboptimal solution. Instead, at each iteration, the gradient information can be used to determine a better search direction ($\boldsymbol{d}$). The conjugate gradient algorithm achieves this by choosing the search direction so that the component of the gradient vector ($\boldsymbol{g}_{i+1}$) that is parallel to the previous search direction ($\boldsymbol{d}_i$) is nullified. This concept is represented mathematically by Equation A16 and graphically by Figure **57** [64] (repeated here from the main body).

$$\boldsymbol{g}_{i+1}^{T}\boldsymbol{d}_i = 0 \tag{A16}$$

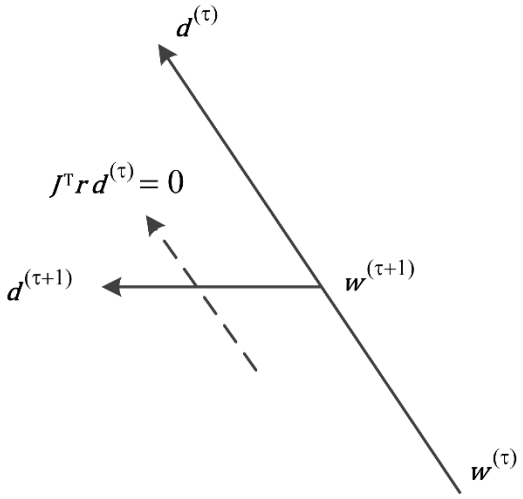


Figure 57: Visual representation of conjugate search directions in two dimensions (adapted from [64])

By doing this, the new search direction will be biased away from the previous search direction. If the error surface is assumed quadratic, two search directions constructed with this principle and that satisfy the condition given by Equation (A17) are conjugate (with respect to the Hessian) [64].

$$\boldsymbol{d}_x^{T}\boldsymbol{H}\boldsymbol{d}_y = 0, \;\; x \neq \tag{A17}$$

An arbitrary update rule for the weight vector can be constructed using a similarly arbitrary search direction ($\boldsymbol{d}_i$) and step size (α_i), given by Equation (A18) [73].

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_i\boldsymbol{d}_i \tag{A18}$$

Now, if a set of search directions can be found which are mutually conjugate, the step size (α_i) can be derived as shown in Equation (A19) [73].

$$\alpha_i = -\frac{\boldsymbol{d}_i^{T}\boldsymbol{g}_i}{\boldsymbol{g}_i^{T}\boldsymbol{H}\boldsymbol{d}_i} \tag{A19}$$

Using this step size in the update rule will ensure that each gradient vector is orthogonal to all previous conjugate directions [find ref]. In the case of a quadratic error surface, this is significant because, after W iterations (where W represents the number of weights in the weight vector), the gradient vector will have been nullified in each direction, meaning that the minimum has been reached [74].

This property only holds if a set of mutually conjugate search directions are used. This can be done by selecting the first search direction as the negative of the gradient ($\boldsymbol{d}_1 = -\boldsymbol{g}_1$) and each successive direction as a linear combination of the current gradient and the previous search direction. Equation (A20) shows the update rule for mutually conjugate search directions [75].

$$\boldsymbol{d}_{i+1} = -\boldsymbol{g}_{i+1} + \beta_i \boldsymbol{d}_i \tag{A20}$$

The formula for coefficient β_i is given by Equation ((A21) and is derived by imposing the conjugacy condition given in Equation (A21) [75].

$$\beta_i = \frac{\boldsymbol{g}_{i+1}^T \boldsymbol{d}_i}{\boldsymbol{d}_i^T \boldsymbol{H} \boldsymbol{d}_i} \tag{A21}$$

It should be noted that determining the Hessian is necessary for each iteration, but is a computationally expensive process. For this reason, alternative methods for finding α_i and β_i coefficients have been developed. α_i can be determined using line minimisation along the search direction ($min\ \varepsilon(\boldsymbol{w}_i + \alpha \boldsymbol{d}_i)$) and formulas for β_i have been derived that eliminate the necessity of calculating the Hessian. The alternative forms of β_i are given in Table 15 [64].

Table 15: Alternative forms of the $\boldsymbol{\beta}$ coefficient without the use of the Hessian (adapted from [64])

Derivation name	Equation	No
Hestenes-Stiefel	$\beta_i = \frac{\boldsymbol{g}_{i+1}^T (\boldsymbol{g}_{i+1} - \boldsymbol{g}_i)}{\boldsymbol{d}_i^T (\boldsymbol{g}_{i+1} - \boldsymbol{g}_i)}$	(A22)

Polak-Ribiere	$\beta_i = \frac{\boldsymbol{g}_{i+1}^T(\boldsymbol{g}_{i+1} - \boldsymbol{g}_i)}{\boldsymbol{g}_i^T \boldsymbol{g}_i}$	(A23)
Fletcher-Reeves	$\beta_i = \frac{\boldsymbol{g}_{i+1}^T \boldsymbol{g}_{i+1}}{\boldsymbol{g}_i^T \boldsymbol{g}_i}$	(A24)

For a quadratic cost function, Equations (A23) - (A24) are equivalent. In the general case, each equation will yield different results. This is because, when a non-quadratic function is used, the conjugacy of search directions starts to deteriorate, and the minimum is often not found in W line searches. In this case, the algorithm can be rebooted by,again setting the search direction equal to the negative gradient [76]. Equation (A23) performs slightly better in this case because, when successive gradients are small, β_i tends to be small as well. Using Equation (A20), it can then be seen that the search direction tends towards the negative gradient, which is equivalent to a reboot of the algorithm [64]. A summary of the conjugate gradient algorithm is given in Table 16.

Table 16: The conjugate gradient algorithm procedure

Step	Action	Calculation
1	Populate initial weight vector with randomised values	$\boldsymbol{w}_1 = rand()$
2	Evaluate the gradient vector and initialise the first search direction to the negative of the gradient	$\boldsymbol{d}_1 = -\boldsymbol{g}_1$
3	Minimise the cost function of the linear combination of the weight vector and the search direction with respect to the step size coefficient (α) to obtain an update that minimises the error	1. $minimise\ \varepsilon(\boldsymbol{w}_i + \alpha \boldsymbol{d}_i)$ 2. $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_{min}\boldsymbol{d}_i$
4	Check if the stopping criteria have been met	$\varepsilon(\boldsymbol{w}_{i+1}) < threshold$
5	Evaluate the new gradient vector	$evaluate\ \boldsymbol{g}_{i+1}$
6	Find the new search direction using the mutual conjugacy update rule using the chosen equation for β	$\boldsymbol{d}_{i+1} = -\boldsymbol{g}_{i+1} + \beta_i \boldsymbol{d}_i$
7	Increment the step number and go back to step 3	$i = i + 1$

Using Equations (A17) - (A21), the values of a weight vector that yields the minimum of a quadratic function can be found by calculating mutually conjugate search directions. For a cost function with a general, non-quadratic form, an arbitrarily large region around the minimum is approximately quadratic, so the conjugate gradient algorithm should still be able to converge [76]. Additionally, it has been shown that the computationally expensive evaluation of the Hessian can be replaced by line search. Line search itself, however, can become a computationally expensive procedure since, for every line search, the cost function must be evaluated multiple times [64].

Møller [74] modified the conjugate gradient algorithm to avoid the line search procedure problem, called the scaled conjugate gradient (SCG) algorithm. The observation was made that, in the formulas for coefficients, the Hessian was always in the form $\boldsymbol{H}\boldsymbol{d}_i$. This is significant because in the special case of the product of the Hessian and an arbitrary vector, the reverse-Hessian vector product technique ($\mathcal{R}\{\cdot\}$-technique) can be used. Calculating the Hessian takes in the order of W^2 steps per training pattern, but the $\mathcal{R}\{\cdot\}$-technique can calculate $\boldsymbol{H}\boldsymbol{d}_i$ in the order of W steps. This allows the reintroduction of the Hessian into the algorithm to improve its performance [64].

However, when a non-quadratic cost function is used, the Hessian is not necessarily positive definite. If the Hessian is not positive definite, the denominator of Equation (A19) can be negative, which leads to an increase in the error [64]. Møller borrows from Levenberg to address this by adding some multiple (λ) of the unit matrix ($\boldsymbol{I}$) to the Hessian to ensure it is positive definite as shown in Equation (A25) [73].

$$\boldsymbol{H} = \boldsymbol{H} + \lambda \boldsymbol{I}, \qquad \lambda \geq 0 \tag{A25}$$

The SCG version of the step size then becomes Equation (A26) [64].

$$\alpha_i = -\frac{\boldsymbol{d}_i^T \boldsymbol{g}_i}{\boldsymbol{d}_i^T \boldsymbol{H}_i \boldsymbol{d}_i + \lambda_i \|\boldsymbol{d}_i\|^2} \tag{A26}$$

As mentioned in the discussion of the conjugate gradient algorithm, an arbitrarily large region around the minimum is approximately quadratic. The SCG algorithm improves on the conjugate gradient algorithm by adjusting the size of the region in which the quadratic assumption is made, improving its accuracy [76]. The SCG algorithm therefore falls under the model trust region

methods. The size of the region is governed by a scalar constant (λ). A larger value of λ leads to a smaller trust region and, conversely, a smaller value of λ leads to a larger trust region [64].

The value of λ is therefore an important part of the performance of the SCG algorithm and must be chosen appropriately. To do this, we note that if $\lambda = 0$, the weight vector will be updated such that the resulting error moves towards the minimum only if [64]:

1. The Hessian is positive definite.
2. The cost function is quadratic.

To ensure that the first condition is met, the denominator of Equation (A26) must be positive. Using this restraint, the modified value of λ can be derived to be Equation (A27) [73].

$$\overline{\lambda_i} = 2(\lambda_i - \frac{\boldsymbol{d}_i^T \boldsymbol{H}_i \boldsymbol{d}_i + \lambda_i \|\boldsymbol{d}_i\|^2}{\|\boldsymbol{d}_i\|^2}) \tag{A27}$$

For the second condition, the quadratic approximation of the cost function must be accurate around a region of the current point in weight space ($\boldsymbol{w}_i$). Since the size of the region is governed by λ, its value can be adjusted to improve the accuracy of the approximation. The basis for this adjustment is a comparison parameter given by Equation (A28) [64].

$$\Delta_i = \frac{\varepsilon(\boldsymbol{w}_i) - \varepsilon(\boldsymbol{w}_i + \alpha_i \boldsymbol{d}_i)}{\varepsilon(\boldsymbol{w}_i) - \varepsilon_Q(\boldsymbol{w}_i + \alpha_i \boldsymbol{d}_i)} = \frac{2\{\varepsilon(\boldsymbol{w}_i) - \varepsilon(\boldsymbol{w}_i + \alpha_i \boldsymbol{d}_i)\}}{\alpha_i \boldsymbol{d}_i^T \boldsymbol{g}_i} \tag{A28}$$

The comparison parameter (Δ_i) compares how well the quadratic approximation of the cost function (ε_Q) correlates with the actual cost function. The λ-parameter is adjusted based on the value of the comparison parameter as follows [64]:

- If $\Delta_i > 0.75$: $\lambda_{i+1} = \frac{\lambda_i}{2}$
- If $\Delta_i < 0.25$: $\lambda_{i+1} = 4\lambda_i$
- Else, set $\lambda_{i+1} = \lambda_i$

By modifying the value of λ to satisfy both conditions after each weight update, the SCG shows a significant improvement in convergence speed compared to the base conjugate gradient algorithm [64].

APPENDIX B: FILTERING

The noise spike filter was applied to parameters involved in the development of the ANN model. Figure 58 to Figure **65** shows the unfiltered data for all parameters plotted against the filtered data.

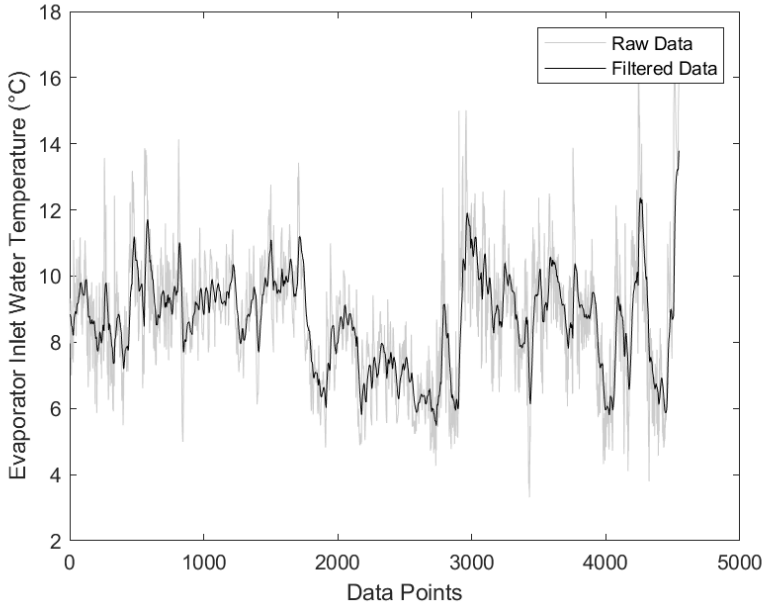


Figure 58: Raw data compared to filtered data for the evaporator inlet water temperature

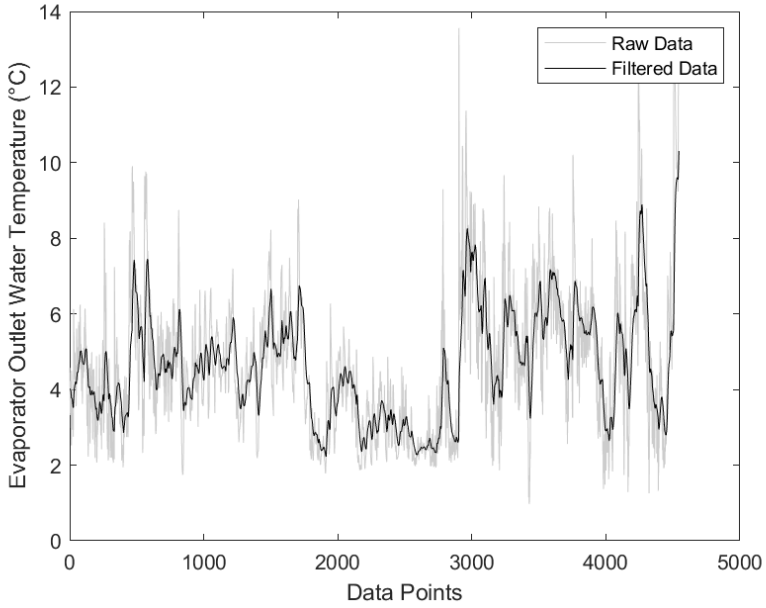


Figure 59: Raw data compared to filtered data for the evaporator outlet water temperature

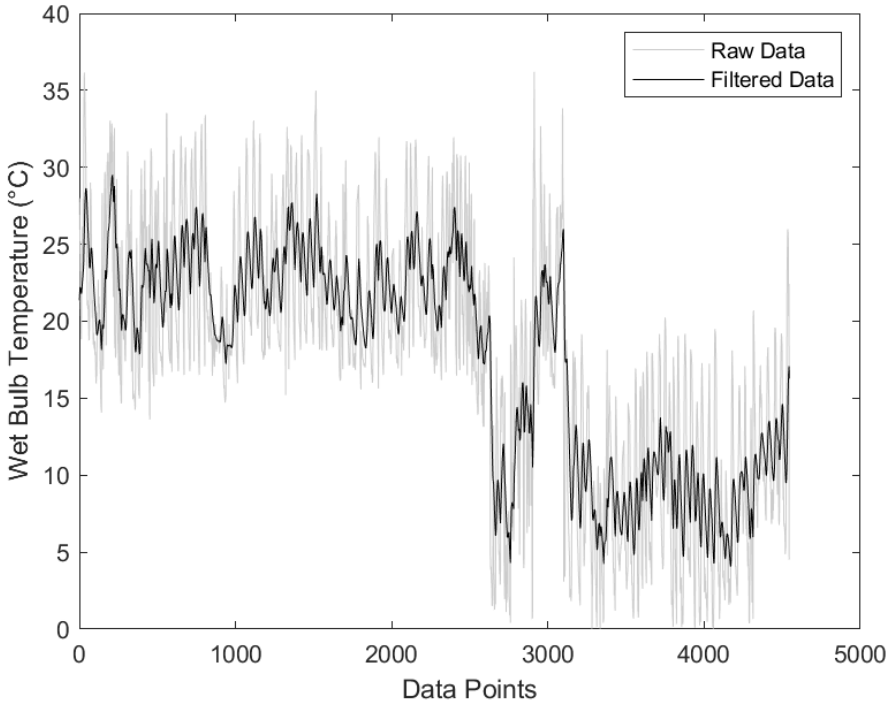


Figure 60: Raw data compared to filtered data for the wet-bulb temperature

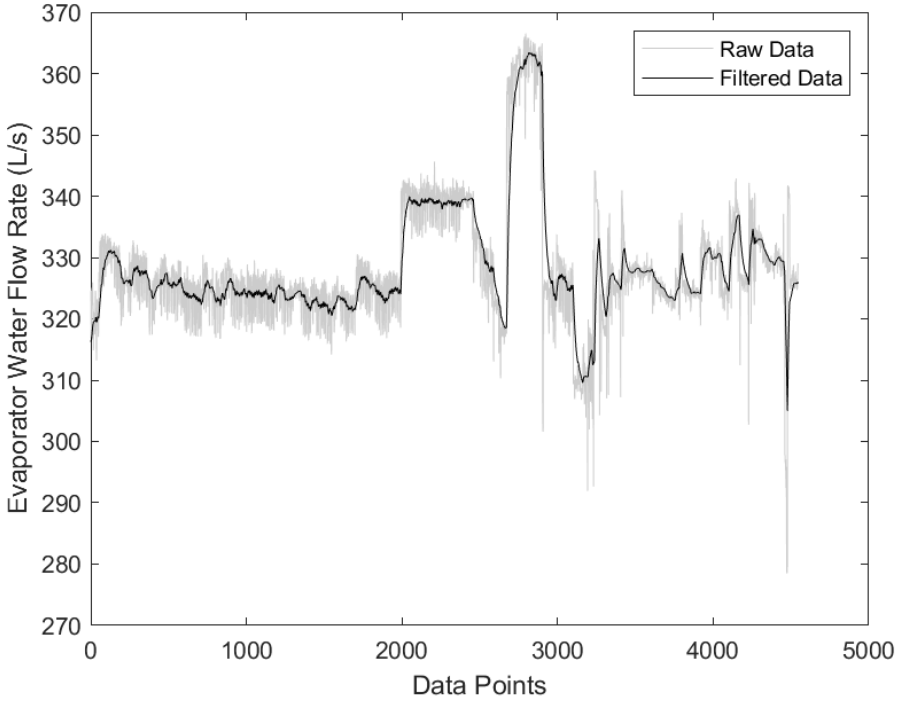


Figure 61: Raw data compared to filtered data for the evaporator water flow rate

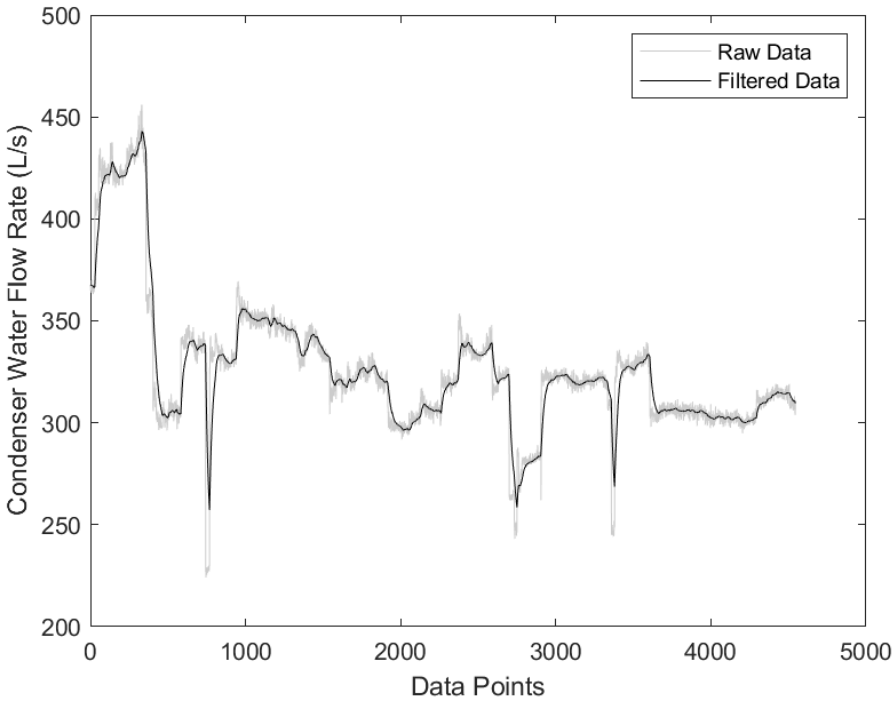


Figure 62: Raw data compared to filtered data for the condenser water flow rate

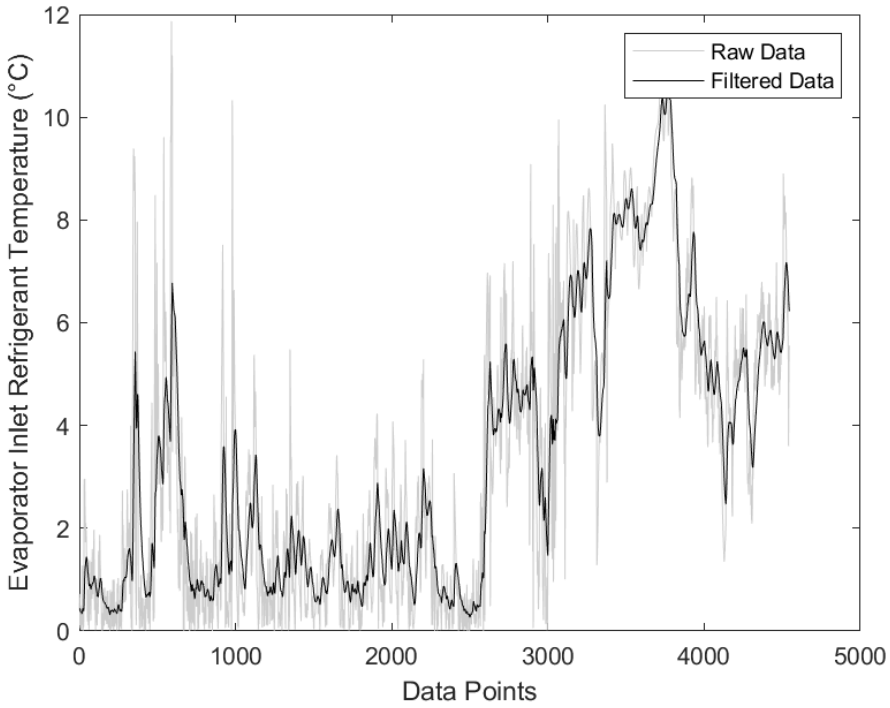


Figure 63: Raw data compared to filtered data for the evaporator inlet refrigerant temperature

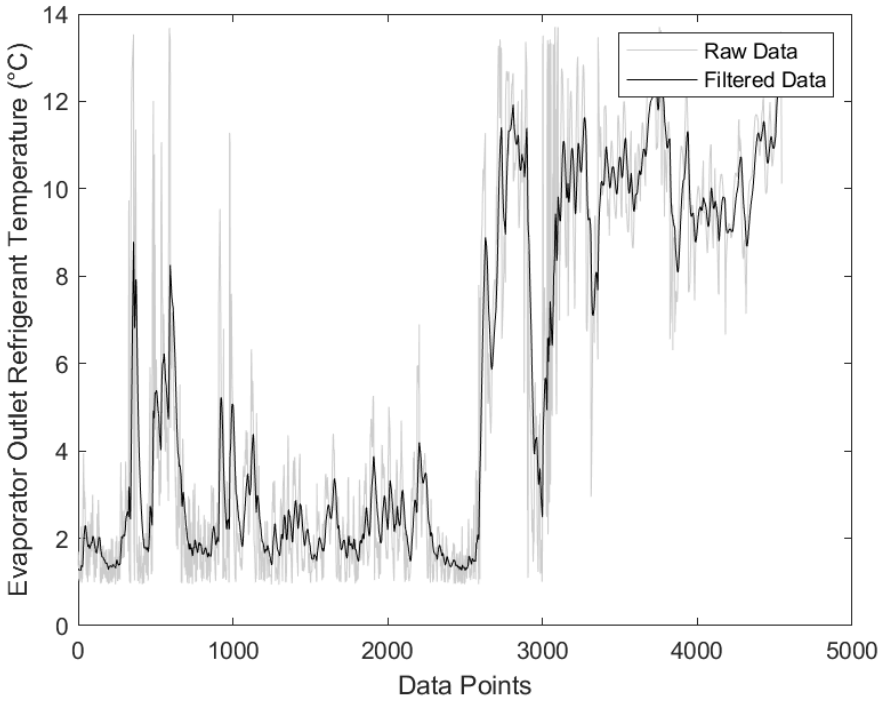


Figure 64: Raw data compared to filtered data for the evaporator outlet refrigerant temperature

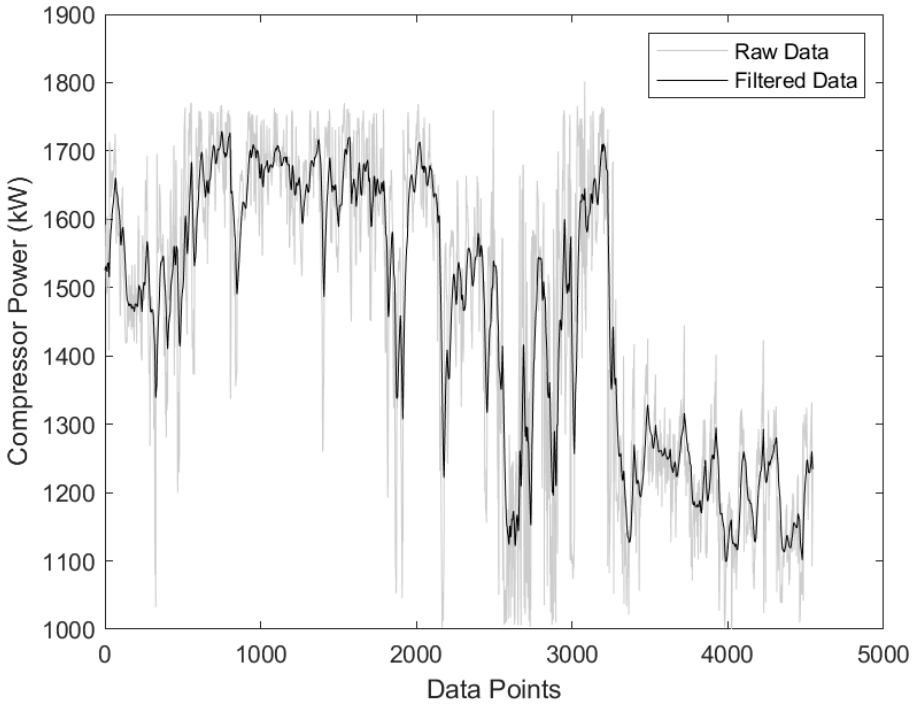


Figure 65: Raw data compared to filtered data for the compressor power

APPENDIX C: WEIGHT AND BIAS MATRICES

This section shows the weights and biases of the best-performing model developed in Chapter 3. Table 17 shows the weights and biases between the input and first hidden layer.

Table 17: Weights and biases of the input-to-hidden layer of the best-performing model

j	w_{1j}	w_{2j}	w_{3j}	w_{4j}	w_{5j}	w_{6j}	w_{7j}	w_{8j}	b_j
1	0.65	-0.44	1.36	2.27	-4.95	-0.06	0.63	-1.70	3.60
2	0.43	0.42	-3.03	2.43	-3.71	2.90	0.23	-0.18	2.53
3	3.35	-4.31	-0.03	-1.37	-1.75	2.00	-2.00	-2.42	-5.01
4	-0.02	-2.08	-0.82	-4.59	-0.20	-0.41	2.72	1.20	-4.12
5	5.47	-6.00	2.57	5.38	-0.95	0.86	-1.20	-1.85	1.33
6	0.35	5.23	-2.58	-0.24	-1.98	1.13	3.45	2.92	-3.92
7	-4.84	-0.30	-0.08	3.70	1.54	2.83	0.02	5.11	3.54
8	-1.01	7.19	-1.48	0.42	2.73	-2.00	-1.79	1.73	-1.28
9	1.96	1.37	3.03	-2.06	4.50	-1.71	-1.61	-2.00	-0.68
10	6.51	-0.78	0.87	3.64	1.15	-0.50	2.26	-0.51	-1.90
11	-1.20	0.32	-4.65	-3.84	3.82	0.52	-3.80	5.60	3.08
12	-1.47	-1.16	0.55	-5.01	-2.34	-0.93	-2.54	-1.34	-2.75
13	1.23	-5.10	-3.06	3.80	-4.95	4.57	-1.16	3.96	-4.77
14	2.26	3.88	-2.13	-2.00	1.32	0.87	2.74	1.66	-3.30
15	3.71	-0.81	-3.27	5.95	-0.91	1.10	-4.43	-3.20	-0.69
16	1.42	2.95	2.92	0.08	-3.11	0.44	-4.41	-0.68	0.22
17	1.30	0.84	-1.86	1.22	2.60	-0.54	0.12	-1.72	0.13
18	-1.25	-1.97	-0.61	-3.45	-4.39	2.24	-1.87	-1.75	1.22
19	6.01	-3.05	2.50	-6.51	-0.70	0.60	2.95	-1.56	-1.00
20	-2.44	7.04	0.60	6.13	-5.49	3.88	1.17	-1.02	0.99
21	1.22	-2.72	-2.53	-1.86	-4.45	-3.15	-1.05	1.20	-0.02
22	0.93	4.74	7.39	-4.66	-1.71	-2.30	-1.24	1.40	-1.31
23	0.70	2.52	-1.92	-5.71	-2.38	3.10	0.59	2.21	-2.69
24	-3.51	-3.44	-0.42	-0.28	2.98	-3.29	3.08	-6.46	0.21
25	3.48	-0.89	1.04	-2.23	2.27	-0.34	1.95	-3.76	3.47
26	2.88	-3.06	-4.40	0.81	1.25	2.25	1.39	-3.42	-2.05
27	-1.31	-3.67	-0.29	0.44	-0.72	-1.33	2.70	2.32	-3.14
28	2.30	-7.75	4.63	-3.62	-0.17	-2.86	1.44	-0.22	-5.95
29	-0.33	-0.26	8.57	4.05	0.28	-4.83	-1.62	-2.67	-3.52
30	-2.68	1.29	2.48	3.35	0.47	-3.95	-2.93	2.91	3.43

31	3.15	1.00	-2.22	-4.36	-1.14	4.52	5.66	-0.93	5.09
32	1.86	4.55	-1.18	3.67	-3.92	-0.36	-0.52	2.87	3.51
33	6.67	-4.34	0.90	4.83	-3.46	2.02	1.75	-1.34	1.93
34	5.20	-1.16	-0.40	1.45	0.77	2.67	3.74	-1.34	3.93
35	1.29	-2.02	-2.41	1.78	0.80	-0.47	2.33	0.05	3.54
36	-1.57	3.33	4.53	4.67	5.40	-1.22	0.28	2.26	6.32
37	0.02	-2.59	0.10	0.87	2.09	-1.17	-0.32	4.86	-4.37

The weight/bias matrix between the hidden layers of the ANN is a 21x38 matrix. The weights will therefore be split between Table 18, Table 19, Table 20, and Table 21. The biases are shown in Table 22 and Table 23.

Table 18: Hidden-to-hidden layer weights and biases (Part 1)

k	w_{1k}	w_{2k}	w_{3k}	w_{4k}	w_{5k}	w_{6k}	w_{7k}	w_{8k}	w_{9k}
1	0.62	-0.09	-2.42	-0.30	-1.67	-2.06	1.00	2.82	-0.80
2	2.09	1.20	-0.50	0.65	1.03	0.88	-0.19	-1.13	0.84
3	-2.60	0.08	-0.06	3.92	0.30	-3.25	0.55	3.48	2.04
4	2.21	-0.40	-0.43	-2.27	-0.82	-0.28	-0.34	1.53	0.46
5	1.04	2.56	-2.93	0.24	-1.66	0.97	2.02	2.92	0.16
6	-1.21	-2.37	0.89	-2.44	2.28	1.78	-0.50	-0.92	-2.71
7	4.30	-0.63	-1.60	4.61	2.36	1.33	-1.81	3.36	-1.94
8	0.68	1.95	1.17	3.33	-2.61	2.39	-1.16	1.78	-1.19
9	-0.35	1.49	0.49	0.77	-0.30	-1.67	0.76	2.03	0.71
10	2.62	0.32	-1.97	1.74	4.13	3.51	-0.39	-1.44	1.53
11	2.97	-1.91	-1.66	2.72	4.09	4.19	-1.42	1.97	-3.19
12	-1.24	-1.05	-2.22	0.98	-0.25	1.83	-1.22	1.80	2.33
13	-0.20	1.21	0.15	2.49	-4.39	1.57	-0.46	0.35	-1.61
14	-0.11	-1.11	-0.92	3.25	-0.85	-1.02	-0.12	0.83	1.02
15	1.18	-0.35	-1.77	1.76	3.05	0.85	-2.86	-1.71	-2.37
16	0.80	0.07	-0.11	3.05	2.50	-0.27	-2.06	-0.87	-3.60
17	-0.42	-1.30	2.18	-0.81	-1.14	0.13	1.68	-2.62	3.93
18	-1.24	1.81	-0.28	-0.44	-0.83	-1.60	1.55	-2.05	0.81
19	-1.26	2.27	-1.60	3.28	-0.24	-1.16	0.37	-0.82	0.54
20	2.03	2.03	1.05	-1.92	-2.45	2.02	0.99	1.10	0.64
21	-1.12	0.89	-0.47	-1.29	2.93	-0.44	2.10	-0.77	-0.45

Table 19: Hidden-to-hidden layer weights and biases (Part 2)

k	w_{10k}	w_{11k}	w_{12k}	w_{13k}	w_{14k}	w_{15k}	w_{16k}	w_{17k}	w_{18k}
1	1.02	-1.20	1.70	-1.21	1.77	2.23	-2.01	0.72	0.90
2	1.11	1.53	1.50	-0.07	1.84	-0.66	2.94	3.57	0.13
3	0.33	3.27	1.37	0.87	1.66	-5.41	2.94	0.66	-3.15
4	0.51	0.29	-0.69	1.06	-2.02	0.75	-2.58	1.35	1.04
5	-1.77	-2.22	0.02	2.66	-0.94	1.56	0.80	0.93	-1.61
6	-4.09	1.71	2.21	1.45	1.86	0.25	-0.71	0.47	0.99
7	0.01	0.04	2.16	0.53	-1.52	1.58	-2.07	0.11	-2.13
8	-0.91	-0.95	-1.09	1.60	-2.66	-2.53	-2.41	-1.10	-3.17
9	-0.47	1.95	0.34	-2.92	-0.39	-0.72	-2.01	-1.92	1.89
10	-3.26	-1.05	1.09	1.96	-0.31	-0.30	1.68	-1.67	-3.13
11	-0.09	-1.20	-1.69	-0.72	0.29	4.27	-1.77	-0.09	0.17
12	0.71	-0.91	1.13	3.06	-1.51	1.07	-0.22	0.13	0.23
13	3.11	2.69	1.66	2.22	2.29	-2.46	0.55	0.05	0.95
14	0.85	0.00	2.73	0.01	-1.62	-1.94	1.41	-2.64	-0.19
15	-0.73	1.74	0.72	3.79	1.19	0.45	1.45	1.77	-2.02
16	-0.69	0.02	0.51	-4.20	0.58	-1.11	-1.03	-1.40	0.26
17	-3.24	3.91	-0.98	-3.83	0.91	-2.99	0.95	0.89	-1.50
18	-0.35	-0.09	-0.43	-0.02	3.51	2.06	1.17	-1.57	-0.32
19	2.15	1.28	2.04	1.64	2.73	2.29	-0.13	-0.28	1.09
20	2.33	-0.81	-1.12	1.00	-1.53	1.57	-0.73	-0.17	0.89
21	-2.17	0.17	-2.11	1.92	-0.56	0.98	2.84	0.24	-1.16

Table 20: Hidden-to-hidden layer weights and biases (Part 3)

k	w_{19k}	w_{20k}	w_{21k}	w_{22k}	w_{23k}	w_{24k}	w_{25k}	w_{26k}	w_{27k}	w_{28k}
1	-2.43	1.12	1.14	1.67	1.97	0.10	2.29	-1.64	-1.76	-1.04
2	1.38	-3.34	-0.85	-0.08	-1.02	0.25	1.43	0.18	-1.83	3.40
3	0.89	-1.44	0.02	2.43	2.27	0.07	-2.02	-0.42	-2.56	-1.78
4	-0.15	-2.96	1.03	-1.63	-2.48	-2.42	0.61	-3.57	-0.49	0.88
5	-1.49	-1.57	-0.25	-3.10	1.94	-0.37	0.17	1.28	0.12	0.20
6	-1.79	2.12	0.23	-2.31	2.11	-1.87	-1.38	-3.78	3.13	0.07
7	-1.25	-2.73	0.28	-0.49	1.83	-0.08	1.92	1.28	-2.07	0.51
8	-2.25	3.27	-0.41	0.84	1.51	-0.70	0.35	-0.15	-2.16	1.41
9	-4.34	2.36	-0.30	0.08	-0.83	0.43	0.13	-1.82	0.63	-0.66

10	-2.26	1.64	-0.52	1.03	1.36	1.38	1.50	1.09	-1.73	0.44
11	0.21	-1.87	-1.31	-0.32	-0.63	0.31	-1.49	1.67	-2.62	3.18
12	-1.23	1.02	0.31	0.60	-1.80	1.37	-0.01	-1.12	-1.91	3.21
13	-0.66	0.27	0.07	2.50	-0.90	-0.57	-2.17	-2.20	-3.90	-2.63
14	-3.61	1.57	0.22	0.47	1.33	1.47	-4.10	-0.96	-1.02	-2.09
15	1.89	-1.02	-2.02	-1.70	0.31	-0.47	1.05	1.15	-1.82	3.31
16	0.15	1.78	-1.62	2.08	0.64	0.88	-0.84	0.14	-2.82	-1.41
17	-2.39	1.39	3.82	0.87	1.76	-1.16	-5.19	-2.59	0.17	-7.23
18	2.65	-0.17	1.04	0.98	-0.96	0.26	0.25	1.62	1.41	0.30
19	0.79	-0.93	0.95	1.58	-2.60	0.08	-0.21	-0.73	0.63	-0.81
20	1.92	-0.94	-1.98	0.03	-1.09	-0.61	2.13	1.43	-3.52	3.17
21	2.03	0.09	0.55	2.90	1.22	0.06	3.00	0.92	0.78	-0.32

Table 21: Hidden-to-hidden layer weights and biases (Part 4)

w_{29k}	w_{30k}	w_{31k}	w_{32k}	w_{33k}	w_{34k}	w_{35k}	w_{36k}	w_{37k}	b_k
-0.30	1.57	-1.11	-0.94	-0.48	-0.65	0.22	-1.90	1.47	-0.80
2.24	-0.21	-0.52	-1.12	0.85	1.24	1.24	0.94	0.01	-14.21
-1.19	0.15	-0.98	-2.38	3.09	3.74	-0.22	-0.32	-2.91	1.69
1.14	-1.32	2.35	-0.60	2.90	1.03	-1.16	0.77	1.66	0.95
-1.34	-0.73	-1.44	-0.19	1.20	0.29	1.30	-0.01	-0.66	-5.83
1.45	-0.10	3.96	0.29	0.47	1.58	0.23	1.02	-0.74	-1.32
-0.41	0.92	0.17	-0.45	-4.17	-1.70	0.66	0.57	0.59	-4.09
-0.22	-0.95	0.63	1.19	3.22	0.04	0.30	0.12	-1.66	-1.46
-3.67	-2.90	2.02	1.14	-0.33	-2.12	-0.71	0.39	2.64	-3.16
-0.36	-4.04	-2.50	-0.39	-3.08	3.29	0.55	0.24	-1.09	-2.78
1.28	-0.38	1.43	1.73	-2.75	-2.18	-2.04	-1.15	0.77	3.00
0.77	-2.98	2.59	-0.09	-0.10	-0.70	1.35	1.01	1.55	-3.28
-2.13	4.63	0.52	-0.06	0.11	1.48	-1.71	-0.26	-1.66	3.73
0.15	-1.48	3.03	-1.77	-1.17	5.32	0.73	1.56	-2.91	1.44
0.49	-0.64	-1.02	0.35	-1.56	-1.74	-0.17	2.53	-1.65	-0.82
2.94	2.60	2.83	0.16	-0.92	-0.38	-0.43	0.52	1.62	2.24
-0.18	2.24	1.07	-3.35	-1.43	3.94	-1.15	-0.25	-0.87	-2.12
1.61	0.55	1.31	-0.63	-0.92	-0.74	-1.96	0.49	1.19	-3.21
-1.15	-2.18	-0.40	0.14	-2.67	0.82	-4.42	1.30	-0.32	0.56
1.78	-0.04	-2.05	0.18	2.68	1.86	1.87	-1.73	1.60	-5.97
-1.87	0.12	1.06	-1.31	2.63	-4.86	-0.32	0.08	-0.23	-2.94

Table 22: Hidden-to-output layer weights and biases (Part 1)

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}
-2.38	3.11	-1.78	-2.30	3.64	2.30	1.92	-1.43	-3.94	-3.06	-2.85

Table 23: Hidden-to-output layer weights and biases (Part 2)

w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}	w_{21}	b_o
3.30	-1.38	-1.96	-2.66	3.66	1.76	-3.59	3.50	2.57	1.41	-0.15

APPENDIX D: SOURCE CODE

An artificial neural network model for predicting the performance of a deep-level mine refrigeration system: Source Code

```
% Initialise variables
test_ID = "07_21_1818";
HNs_1HL = 200;
Reps_1HL = 1;
HNs_2HL_HL1 = 40;
HNs_2HL_HL2 = 40;
Reps_2HL = 1;
LM = 'trainlm';
SCG = 'trainscg';
retrain_reps = 1;

% Load raw data
Variables_File_Path = 'ANN\Raw_Data.xlsx';
[Raw_EWIT, Raw_EWOT, Raw_EWFR, Raw_CWFR, Raw_ERIT, Raw_EROT, Raw_WBT, Raw_CP, Raw_COP] =
loadrawdata(Variables_File_Path);

% Consolidate raw data
[Unfiltered_Consolidated_Data, Datetime_Array] = consolidate(Raw_EWIT, Raw_EWOT, Raw_EWFR,
Raw_CWFR, Raw_ERIT, Raw_EROT, Raw_WBT, Raw_CP, Raw_COP);

% Filter consolidated data
Filtered_Data = filterdata(Unfiltered_Consolidated_Data, Datetime_Array);

% Normalise and perform cross-validation
Normalised_Data = mapminmax(Filtered_Data(:,1:9)', 0, 1);
[TCI_Class, Verification_Class] = crossvalidation(Normalised_Data');

% Train single hidden layer ANNs with LM and SCG training algorithms
train_ID = "1HL_LM";
best_1HL_LM = singlelayerANN(TCI_Class', HNs_1HL, Reps_1HL, LM, train_ID, test_ID);

train_ID = "1HL_SCG";
best_1HL_SCG = singlelayerANN(TCI_Class', HNs_1HL, Reps_1HL, SCG, train_ID, test_ID);

% Train two hidden layer ANNs with LM and SCG training algorithms
train_ID = "2HL_LM";
best_2HL_LM = twolayerANN(TCI_Class', HNs_2HL_HL1, HNs_2HL_HL2, Reps_2HL, LM, train_ID,
test_ID);

train_ID = "2HL_SCG";
best_2HL_SCG = twolayerANN(TCI_Class', HNs_2HL_HL1, HNs_2HL_HL2, Reps_2HL, SCG, train_ID,
test_ID);
```

```
 % Repetitively retrain best architectures for one hidden layer ANNs
 [Opt_1HL_SCG_Config_1, O1HL_SCG_C1_TR] = retrain1HL_ANN(best_1HL_SCG, SCG, TCI_Class',
retrain_reps, test_ID,"1a");
 [Opt_1HL_SCG_Config_2, O1HL_SCG_C2_TR] = retrain1HL_ANN(best_1HL_LM, SCG, TCI_Class',
retrain_reps, test_ID,"1b");
 [Opt_1HL_LM_Config_1, O1HL_LM_C1_TR] = retrain1HL_ANN(best_1HL_SCG, LM, TCI_Class',
retrain_reps, test_ID,"2a");
 [Opt_1HL_LM_Config_2, O1HL_LM_C2_TR] = retrain1HL_ANN(best_1HL_LM, LM, TCI_Class',
retrain_reps, test_ID,"2b");

 % Repetitively retrain best architectures for two hidden layer ANNs
 [Opt_2HL_SCG_Config_1, O2HL_SCG_C1_TR] = retrain2HL_ANN(best_2HL_SCG(1), best_2HL_SCG(2),
SCG, TCI_Class', retrain_reps, test_ID,"3a");
 [Opt_2HL_SCG_Config_2, O2HL_SCG_C2_TR] = retrain2HL_ANN(best_2HL_LM(1), best_2HL_LM(2), SCG,
TCI_Class', retrain_reps, test_ID,"3b");
 [Opt_2HL_LM_Config_1, O2HL_LM_C1_TR] = retrain2HL_ANN(best_2HL_SCG(1), best_2HL_SCG(2), LM,
TCI_Class', retrain_reps, test_ID,"4a");
 [Opt_2HL_LM_Config_2, O2HL_LM_C2_TR] = retrain2HL_ANN(best_2HL_LM(1), best_2HL_LM(2), LM,
TCI_Class', retrain_reps, test_ID,"4b");

 % Select and evaluate the best performing model overall
 [Best_ANN, Best_R_2] = compare_ANNs(Opt_1HL_SCG_Config_1, Opt_1HL_SCG_Config_2,
Opt_1HL_LM_Config_1, Opt_1HL_LM_Config_2, Opt_2HL_SCG_Config_1, Opt_2HL_SCG_Config_2,
Opt_2HL_LM_Config_1, Opt_2HL_LM_Config_2, O1HL_SCG_C1_TR, O1HL_SCG_C2_TR, O1HL_LM_C1_TR,
O1HL_LM_C2_TR, O2HL_SCG_C1_TR, O2HL_SCG_C2_TR, O2HL_LM_C1_TR, O2HL_LM_C2_TR, TCI_Class',
test_ID);
 % Derive fundamental input-output equation
 if Best_ANN.numLayers == 3
     [weights_IH, weights_HH, weights_HO, biases_h1, biases_h2, bias_O] =
Derive_FIO_Equation_2HL(Best_ANN);
 else
     [weights_IH, weights_HO, biases_H, bias_O] = Derive_FIO_Equation_1HL(Best_ANN);
 end
 % Evaluate the refrigeration plant performance
 [Eval_Results, Input_minmax_range, COP_minmax_range, Corrected_Eval_Inputs] =
Evaluate_FP_Performance(Filtered_Data, TCI_Class, Best_ANN, test_ID);
 % Optimise refrigeration plant performance
 Optimise_FP(Best_ANN, Eval_Results, Corrected_Eval_Inputs, Filtered_Data, Best_R_2, test_ID)

 % Verify the model
 Verify_Model(Best_ANN, Verification_Class, COP_minmax_range, test_ID);
```

Functions

```
 function [rEvap_Inlet_Water_Temp, rEvap_Outlet_Water_Temp, rEvap_Water_Flow,
rCond_Water_Flow, rEvap_NH3_Inlet_Temp, rEvap_NH3_Outlet_Temp, rWB_Temp, rComp_Power, rCOP] =
loadrawdata(filepath)
```

```
 % Import the inputs and outputs from their respective sheets in the Excel file
 rEvap_Inlet_Water_Temp =
readtable(filepath,'Sheet',1,TextType='string',VariableNamingRule='preserve');
 rEvap_Outlet_Water_Temp =
readtable(filepath,'Sheet',2,TextType='string',VariableNamingRule='preserve');
 rEvap_Water_Flow =
readtable(filepath,'Sheet',3,TextType='string',VariableNamingRule='preserve');
 rCond_Water_Flow =
readtable(filepath,'Sheet',4,TextType='string',VariableNamingRule='preserve');
 rEvap_NH3_Inlet_Temp =
readtable(filepath,'Sheet',5,TextType='string',VariableNamingRule='preserve');
 rEvap_NH3_Outlet_Temp =
readtable(filepath,'Sheet',6,TextType='string',VariableNamingRule='preserve');
 rWB_Temp = readtable(filepath,'Sheet',7,TextType='string',VariableNamingRule='preserve');
 rComp_Power = readtable(filepath,'Sheet',8,TextType='string',VariableNamingRule='preserve');
 rCOP = readtable(filepath,'Sheet',9,TextType='string',VariableNamingRule='preserve');

 % Convert the variable tables to string arrays
 rEvap_Inlet_Water_Temp = table2array(rEvap_Inlet_Water_Temp(:,[1 2 3 4 5 6 7]));
 rEvap_Outlet_Water_Temp = table2array(rEvap_Outlet_Water_Temp(:,[1 2 3 4 5 6 7]));
 rEvap_Water_Flow = table2array(rEvap_Water_Flow(:,[1 2 3 4 5 6 7]));
 rCond_Water_Flow = table2array(rCond_Water_Flow(:,[1 2 3 4 5 6 7]));
 rEvap_NH3_Inlet_Temp = table2array(rEvap_NH3_Inlet_Temp(:,[1 2 3 4 5 6 7]));
 rEvap_NH3_Outlet_Temp = table2array(rEvap_NH3_Outlet_Temp(:,[1 2 3 4 5 6 7]));
 rWB_Temp = table2array(rWB_Temp(:,[1 2 3 4 5 6 7]));
 rComp_Power = table2array(rComp_Power(:,[1 2 3 4 5 6 7]));
 rCOP = table2array(rCOP(:,[1 2 3 4 5 6 7]));

 % Convert the variable string arrays to double arrays
 rEvap_Inlet_Water_Temp = str2double(rEvap_Inlet_Water_Temp);
 rEvap_Outlet_Water_Temp = str2double(rEvap_Outlet_Water_Temp);
 rEvap_Water_Flow = str2double(rEvap_Water_Flow);
 rCond_Water_Flow = str2double(rCond_Water_Flow);
 rEvap_NH3_Inlet_Temp = str2double(rEvap_NH3_Inlet_Temp);
 rEvap_NH3_Outlet_Temp = str2double(rEvap_NH3_Outlet_Temp);
 rWB_Temp = str2double(rWB_Temp);
 rComp_Power = str2double(rComp_Power);
 rCOP = str2double(rCOP);

 end

 function [Variables_reduced, Variables_reduced_datetimes] =
consolidate(cEvap_Inlet_Water_Temp, cEvap_Outlet_Water_Temp, cEvap_Water_Flow,
cCond_Water_Flow, cEvap_NH3_Inlet_Temp, cEvap_NH3_Outlet_Temp, cWB_Temp, cComp_Power, cCOP)

 % Convert the date and time columns of all variables to datetime format
 datetimes_Evap_Inlet_Water_Temp =
datetime(cEvap_Inlet_Water_Temp(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
```

```
 datetimes_Evap_Outlet_Water_Temp =
datetime(cEvap_Outlet_Water_Temp(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
 datetimes_Evap_Water_Flow = datetime(cEvap_Water_Flow(:,1:6),'InputFormat','yyyy/MM/dd
HH:mm:ss');
 datetimes_Cond_Water_Flow = datetime(cCond_Water_Flow(:,1:6),'InputFormat','yyyy/MM/dd
HH:mm:ss');
 datetimes_Evap_NH3_Inlet_Temp =
datetime(cEvap_NH3_Inlet_Temp(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
 datetimes_Evap_NH3_Outlet_Temp =
datetime(cEvap_NH3_Outlet_Temp(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
 datetimes_WB_Temp = datetime(cWB_Temp(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
 datetimes_Comp_Power = datetime(cComp_Power(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');
 datetimes_COP = datetime(cCOP(:,1:6),'InputFormat','yyyy/MM/dd HH:mm:ss');

 % Find the common time range of the datasets
 tmin = max(min(datetimes_Evap_Inlet_Water_Temp),min(datetimes_Evap_Outlet_Water_Temp));
 if tmin < min(datetimes_Evap_Water_Flow)
     tmin = min(datetimes_Evap_Water_Flow);
 end
 if tmin < min(datetimes_Cond_Water_Flow)
     tmin = min(datetimes_Cond_Water_Flow);
 end
 if tmin < min(datetimes_Evap_NH3_Inlet_Temp)
     tmin = min(datetimes_Evap_NH3_Inlet_Temp);
 end
 if tmin < min(datetimes_Evap_NH3_Outlet_Temp)
     tmin = min(datetimes_Evap_NH3_Outlet_Temp);
 end
 if tmin < min(datetimes_WB_Temp)
     tmin = min(datetimes_WB_Temp);
 end
 if tmin < min(datetimes_Comp_Power)
     tmin = min(datetimes_Comp_Power);
 end
 if tmin < min(datetimes_COP)
     tmin = min(datetimes_COP);
 end

 tmax = min(max(datetimes_Evap_Inlet_Water_Temp),max(datetimes_Evap_Outlet_Water_Temp));
 if tmax > max(datetimes_Evap_Water_Flow)
     tmax = max(datetimes_Evap_Water_Flow);
 end
 if tmax > max(datetimes_Cond_Water_Flow)
     tmax = max(datetimes_Cond_Water_Flow);
 end
 if tmax > max(datetimes_Evap_NH3_Inlet_Temp)
     tmax = max(datetimes_Evap_NH3_Inlet_Temp);
 end
```

```
 if tmax > max(datetimes_Evap_NH3_Outlet_Temp)
     tmax = max(datetimes_Evap_NH3_Outlet_Temp);
 end
 if tmax > max(datetimes_WB_Temp)
     tmax = max(datetimes_WB_Temp);
 end
 if tmax > max(datetimes_Comp_Power)
     tmax = max(datetimes_Comp_Power);
 end
 if tmax > max(datetimes_COP)
     tmax = max(datetimes_COP);
 end

 % Create a vector of 30-minute intervals within the common time range
 tvec = tmin:minutes(30):tmax;
 tvec = tvec';

 % Initialize a new matrix to store the filled data
 Variables_filled = zeros(height(tvec),9);

 % Copy the date and time columns from tvec
 array2timetable(Variables_filled(:,1),"RowTimes",tvec);

 % Find the largest array
 largest_array =
max(height(datetimes_Evap_Inlet_Water_Temp),height(datetimes_Evap_Outlet_Water_Temp));
 if height(datetimes_Evap_Water_Flow)>largest_array
     largest_array = height(datetimes_Evap_Water_Flow);
 end
 if height(datetimes_Cond_Water_Flow)>largest_array
     largest_array = height(datetimes_Cond_Water_Flow);
 end
 if height(datetimes_Evap_NH3_Inlet_Temp)>largest_array
     largest_array = height(datetimes_Evap_NH3_Inlet_Temp);
 end
 if height(datetimes_Evap_NH3_Outlet_Temp)>largest_array
     largest_array = height(datetimes_Evap_NH3_Outlet_Temp);
 end
 if height(datetimes_WB_Temp)>largest_array
     largest_array = height(datetimes_WB_Temp);
 end
 if height(datetimes_Comp_Power)>largest_array
     largest_array = height(datetimes_Comp_Power);
 end
 if height(datetimes_COP)>largest_array
     largest_array = height(datetimes_COP);
 end
```

```
 % Loop over the original data and copy the inputs' values to the corresponding rows
 for row_index = 1:largest_array
     % Find the row index that matches the date and time of the original data
     if row_index <= height(datetimes_Evap_Inlet_Water_Temp)
         Cooling_Capacity_index = tvec == datetimes_Evap_Inlet_Water_Temp(row_index);
         % Copy the cooling capacity value to that row
         Variables_filled(Cooling_Capacity_index,1) = cEvap_Inlet_Water_Temp(row_index,7);
     end

     if row_index <= height(datetimes_Evap_Outlet_Water_Temp)
         Cond_Heat_Rejection_Rate_index = tvec ==
datetimes_Evap_Outlet_Water_Temp(row_index);
         % Copy the evap water flow value to that row
         Variables_filled(Cond_Heat_Rejection_Rate_index,2) =
cEvap_Outlet_Water_Temp(row_index,7);
     end

     if row_index <= height(datetimes_Evap_Water_Flow)
         Evap_H2O_Flow_index = tvec == datetimes_Evap_Water_Flow(row_index);
         % Copy the evap water inlet temp value to that row
         Variables_filled(Evap_H2O_Flow_index,3) = cEvap_Water_Flow(row_index,7);
     end

     if row_index <= height(datetimes_Cond_Water_Flow)
         Evap_H2O_Inlet_Temp_index = tvec == datetimes_Cond_Water_Flow(row_index);
         % Copy the evap outlet water temp value to that row
         Variables_filled(Evap_H2O_Inlet_Temp_index,4) = cCond_Water_Flow(row_index,7);
     end

     if row_index <= height(datetimes_Evap_NH3_Inlet_Temp)
         Evap_H2O_Outlet_Temp_index = tvec == datetimes_Evap_NH3_Inlet_Temp(row_index);
         % Copy the evap outlet water temp value to that row
         Variables_filled(Evap_H2O_Outlet_Temp_index,5) = cEvap_NH3_Inlet_Temp(row_index,7);
     end

     if row_index <= height(datetimes_Evap_NH3_Outlet_Temp)
         WB_Temp_index = tvec == datetimes_Evap_NH3_Outlet_Temp(row_index);
         % Copy the wet bulb temp value to that row
         Variables_filled(WB_Temp_index,6) = cEvap_NH3_Outlet_Temp(row_index,7);
     end

     if row_index <= height(datetimes_WB_Temp)
         Rel_Humidity_index = tvec == datetimes_WB_Temp(row_index);
         % Copy the dry bulb temp value to that row
         Variables_filled(Rel_Humidity_index,7) = cWB_Temp(row_index,7);
     end

     if row_index <= height(datetimes_Comp_Power)
```

```
        COP_index = tvec == datetimes_Comp_Power(row_index);
        % Copy the cond water flow value to that row
        Variables_filled(COP_index,8) = cComp_Power(row_index,7);
    end

    if row_index <= height(datetimes_COP)
        Compressor_Power_index = tvec == datetimes_COP(row_index);
        % Copy the cond inlet water temp value to that row
        Variables_filled(Compressor_Power_index,9) = cCOP(row_index,7);
    end
end

% Create a new array to store the values of all non-zero positive values
Variables_reduced = zeros(height(Variables_filled),size(Variables_filled,2));

% Create a new array to store the indeces of all non-zero positive values
index_nonzero_positive = zeros(height(Variables_reduced),1);
index_nonzero_positive_counter = 1;

% % Loop over the variables and copy all rows that have don't have columns containing a zero
or negative value
for row = 1:size(Variables_reduced,1)
    if Variables_filled(row,:)>0
        Variables_reduced(row,:) = Variables_filled(row,:);
        index_nonzero_positive(index_nonzero_positive_counter,1) = row;
        index_nonzero_positive_counter = index_nonzero_positive_counter + 1;
    end
end

% Remove all zero value rows from the reduced variables array and their indeces' array
index_nonzero_positive(index_nonzero_positive(:,1)==0,:) = [];
Variables_reduced(Variables_reduced(:,1)==0,:) = [];

% Create a new array to store the datetimes of the reduced outputs
Variables_reduced_datetimes = [tvec(index_nonzero_positive)];

end


function [Filtered_data] = filterdata(Unfiltered_data, Unfiltered_data_datetimes)

% Record indices and remove data over a period of instrumentation error
Instr_Error_Indices_1 = Unfiltered_data(:,6) == 13.7030;
Unfiltered_data(Unfiltered_data(:,6)==13.7030,:) = [];
Instr_Error_Indices_2 = Unfiltered_data(:,8)<1000;
Unfiltered_data(Unfiltered_data(:,8)<1000,:) = [];
Instr_Error_Indices_3 = Unfiltered_data(:,4)<220;
Unfiltered_data(Unfiltered_data(:,4)<220,:) = [];
```

```
 % Remove datetimes of instrumentation error
 Unfiltered_data_datetimes(Instr_Error_Indices_1,:) = [];
 Unfiltered_data_datetimes(Instr_Error_Indices_2,:) = [];
 Unfiltered_data_datetimes(Instr_Error_Indices_3,:) = [];

 % Find and replace outliers
 Unfiltered_data = filloutliers(Unfiltered_data,'linear', 'movmedian', 50);

 % Export operational, reliable data for validation purposes
 Operational_Data_Headings = ["Datetime" "Evap_Inlet_Water_Temp" "Evap_Outlet_Water_Temp"
"Evap_Water_Flow" "Cond_Water_Flow" "Evap_NH3_Inlet_Temp" "Evap_NH3_Outlet_Temp" "WB_Temp"
"Comp_Power" "COP"];
 Operational_Data_File_Path = 'Baseline Operation\Operational_Data.xlsx';
 writematrix(Operational_Data_Headings,Operational_Data_File_Path,'Sheet','Operational
Data','Range','A1:J1');
 writematrix(Unfiltered_data,Operational_Data_File_Path,'Sheet','Operational
Data','Range','B2');
 writematrix(Unfiltered_data_datetimes,Operational_Data_File_Path,'Sheet','Operational
Data','Range','A2');

 % Split variables for filtering
 uEvap_Inlet_Water_Temp = Unfiltered_data(:,1);
 uEvap_Outlet_Water_Temp = Unfiltered_data(:,2);
 uEvap_Water_Flow = Unfiltered_data(:,3);
 uCond_Water_Flow = Unfiltered_data(:,4);
 uEvap_NH3_Inlet_Temp = Unfiltered_data(:,5);
 uEvap_NH3_Outlet_Temp = Unfiltered_data(:,6);
 uWB_Temp = Unfiltered_data(:,7);
 uComp_Power = Unfiltered_data(:,8);
 uCOP = Unfiltered_data(:,9);

 % Noise Spike Filter
 EIWT_NS_Strictness = 5;
 Evap_Inlet_Water_Temp_NS_Filtered =
medfilt1(uEvap_Inlet_Water_Temp,EIWT_NS_Strictness,'omitnan','truncate');
 EOWT_NS_Strictness = 5;
 Evap_Outlet_Water_Temp_NS_Filtered =
medfilt1(uEvap_Outlet_Water_Temp,EOWT_NS_Strictness,'omitnan','truncate');
 EWF_NS_Strictness = 5;
 Evap_Water_Flow_NS_Filtered =
medfilt1(uEvap_Water_Flow,EWF_NS_Strictness,'omitnan','truncate');
 CWT_NS_Strictness = 5;
 Cond_Water_Flow_NS_Filtered =
medfilt1(uCond_Water_Flow,CWT_NS_Strictness,'omitnan','truncate');
 ERIT_NS_Strictness = 5;
 Evap_NH3_Inlet_Temp_NS_Filtered =
medfilt1(uEvap_NH3_Inlet_Temp,ERIT_NS_Strictness,'omitnan','truncate');
```

```
 EROT_NS_Strictness = 5;
 Evap_NH3_Outlet_Temp_NS_Filtered =
medfilt1(uEvap_NH3_Outlet_Temp,EROT_NS_Strictness,'omitnan','truncate');
 WB_NS_Strictness = 5;
 WB_Temp_NS_Filtered = medfilt1(uWB_Temp,WB_NS_Strictness,'omitnan','truncate');
 CP_NS_Strictness = 5;
 Comp_Power_NS_Filtered = medfilt1(uComp_Power,CP_NS_Strictness,'omitnan','truncate');
 COP_NS_Strictness = 5;
 COP_NS_Filtered = medfilt1(uCOP,COP_NS_Strictness,'omitnan','truncate');

 % Exponentially Weighted Moving Average Filter
 Forgetting_Factor = 0.95;
 EWMA_Function = dsp.MovingAverage('Method','Exponential
weighting','ForgettingFactor',Forgetting_Factor);

 Evap_Inlet_Water_Temp_Filtered = EWMA_Function(Evap_Inlet_Water_Temp_NS_Filtered);
 Evap_Outlet_Water_Temp_Filtered = EWMA_Function(Evap_Outlet_Water_Temp_NS_Filtered);
 Evap_Water_Flow_Filtered = EWMA_Function(Evap_Water_Flow_NS_Filtered);
 Cond_Water_Flow_Filtered = EWMA_Function(Cond_Water_Flow_NS_Filtered);
 Evap_NH3_Inlet_Temp_Filtered = EWMA_Function(Evap_NH3_Inlet_Temp_NS_Filtered);
 Evap_NH3_Outlet_Temp_Filtered = EWMA_Function(Evap_NH3_Outlet_Temp_NS_Filtered);
 WB_Temp_Filtered = EWMA_Function(WB_Temp_NS_Filtered);
 Comp_Power_Filtered = EWMA_Function(Comp_Power_NS_Filtered);
 COP_Filtered = EWMA_Function(COP_NS_Filtered);

 % Neglect transient filter effects
 start = 196;
 limit = length(COP_Filtered);

 Unfiltered_data = [uEvap_Inlet_Water_Temp(start:limit) uEvap_Outlet_Water_Temp(start:limit)
uEvap_Water_Flow(start:limit) uCond_Water_Flow(start:limit) uEvap_NH3_Inlet_Temp(start:limit)
uEvap_NH3_Outlet_Temp(start:limit) uWB_Temp(start:limit) uComp_Power(start:limit)
uCOP(start:limit)];
 Filtered_data = [Evap_Inlet_Water_Temp_Filtered(start:limit)
Evap_Outlet_Water_Temp_Filtered(start:limit) Evap_Water_Flow_Filtered(start:limit)
Cond_Water_Flow_Filtered(start:limit) Evap_NH3_Inlet_Temp_Filtered(start:limit)
Evap_NH3_Outlet_Temp_Filtered(start:limit) WB_Temp_Filtered(start:limit)
Comp_Power_Filtered(start:limit) COP_Filtered(start:limit)];

 % Export filtered data for validation purposes
 Filtered_Data_Headings = ["Evap_Inlet_Water_Temp" "Evap_Outlet_Water_Temp" "Evap_Water_Flow"
"Cond_Water_Flow" "Evap_NH3_Inlet_Temp" "Evap_NH3_Outlet_Temp" "WB_Temp" "Comp_Power" "COP"];
 Filtered_Data_Workbook = 'Data Filters\Filtered_Data.xlsx';
 writematrix(Filtered_Data_Headings,Filtered_Data_Workbook,'Sheet','Filtered
Data','Range','A1:K1');
 writematrix(Filtered_data,Filtered_Data_Workbook,'Sheet','Filtered Data','Range','A2');

 % Plot data before and after filters are applied
```

```
plotfilters(Filtered_data, Unfiltered_data);

end

function [] = plotfilters(Data_filtered, Data_unfiltered)

% Extract filtered and unfiltered data from their respective arrays
pEvap_Inlet_Water_Temp = Data_unfiltered(:,1);
pEvap_Inlet_Water_Temp_Filtered = Data_filtered(:,1);
pEvap_Outlet_Water_Temp = Data_unfiltered(:,2);
pEvap_Outlet_Water_Temp_Filtered = Data_filtered(:,2);
pEvap_Water_Flow = Data_unfiltered(:,3);
pEvap_Water_Flow_Filtered = Data_filtered(:,3);
pCond_Water_Flow = Data_unfiltered(:,4);
pCond_Water_Flow_Filtered = Data_filtered(:,4);
pEvap_NH3_Inlet_Temp = Data_unfiltered(:,5);
pEvap_NH3_Inlet_Temp_Filtered = Data_filtered(:,5);
pEvap_NH3_Outlet_Temp = Data_unfiltered(:,6);
pEvap_NH3_Outlet_Temp_Filtered = Data_filtered(:,6);
pWB_Temp = Data_unfiltered(:,7);
pWB_Temp_Filtered = Data_filtered(:,7);
pComp_Power = Data_unfiltered(:,8);
pComp_Power_Filtered = Data_filtered(:,8);
pCOP = Data_unfiltered(:,9);
pCOP_Filtered = Data_filtered(:,9);

% Plot evaporator water inlet temperature before and after filtering

fig = figure;
plot(pEvap_Inlet_Water_Temp,'Color', [0.8 0.8 0.8])
hold on
plot(pEvap_Inlet_Water_Temp_Filtered,'k');
xlabel('Data Points');
ylabel('Evaporator Inlet Water Temperature (°C)');
legend('Raw Data','Filtered Data');
hold off
figname = "Evap_Inlet_Water_Temp_Filtered" + ".png";
saveas(fig,figname);

% Plot evaporator water outlet temperature before and after filtering
fig = figure;
plot(pEvap_Outlet_Water_Temp,'Color', [0.8 0.8 0.8]);
hold on
plot(pEvap_Outlet_Water_Temp_Filtered,'k');
xlabel('Data Points');
ylabel('Evaporator Outlet Water Temperature (°C)');
legend('Raw Data','Filtered Data');
hold off
```

```
figname = "Evap_Outlet_Water_Temp_Filtered" + ".png";
saveas(fig,figname);

% Plot evaporator water flow rate before and after filtering
fig = figure;
plot(pEvap_Water_Flow,'Color', [0.8 0.8 0.8]);
hold on
plot(pEvap_Water_Flow_Filtered,'k');
xlabel('Data Points');
ylabel('Evaporator Water Flow Rate (L/s)');
legend('Raw Data','Filtered Data');
hold off
figname = "Evap_Water_Flow_Filtered" + ".png";
saveas(fig,figname);

% Plot condenser water flow rate before and after filtering
fig = figure;
plot(pCond_Water_Flow,'Color', [0.8 0.8 0.8]);
hold on
plot(pCond_Water_Flow_Filtered,'k');
xlabel('Data Points');
ylabel('Condenser Water Flow Rate (L/s)');
legend('Raw Data','Filtered Data');
hold off
figname = "Cond_Water_Flow_Filtered" + ".png";
saveas(fig,figname);

% Plot evaporator refrigerant inlet temperature before and after filtering
fig = figure;
plot(pEvap_NH3_Inlet_Temp,'Color', [0.8 0.8 0.8]);
hold on
plot(pEvap_NH3_Inlet_Temp_Filtered,'k');
xlabel('Data Points');
ylabel('Evaporator Inlet Refrigerant Temperature (°C)');
legend('Raw Data','Filtered Data');
hold off
figname = "Evap_NH3_Inlet_Temp_Filtered" + ".png";
saveas(fig,figname);

% Plot evaporator refrigerant outlet temperature before and after filtering
fig = figure;
plot(pEvap_NH3_Outlet_Temp,'Color', [0.8 0.8 0.8]);
hold on
plot(pEvap_NH3_Outlet_Temp_Filtered,'k');
xlabel('Data Points');
ylabel('Evaporator Outlet Refrigerant Temperature (°C)');
legend('Raw Data','Filtered Data');
hold off
```

```
figname = "Evap_NH3_Outlet_Temp_Filtered" + ".png";
saveas(fig,figname);

% Plot wet-bulb temperature before and after filtering
fig = figure;
plot(pWB_Temp,'Color', [0.8 0.8 0.8]);
hold on
plot(pWB_Temp_Filtered,'k');
xlabel('Data Points');
ylabel('Wet Bulb Temperature (°C)');
legend('Raw Data','Filtered Data');
hold off
figname = "Evap_Outlet_Water_Temp_Filtered" + ".png";
saveas(fig,figname);

% Plot compressor power before and after filtering
fig = figure;
plot(pComp_Power,'Color', [0.8 0.8 0.8]);
hold on
plot(pComp_Power_Filtered,'k');
xlabel('Data Points');
ylabel('Compressor Power (kW)');
legend('Raw Data','Filtered Data');
hold off
figname = "Comp_Power_Filtered" + ".png";
saveas(fig,figname);

% Plot COP before and after filtering
fig = figure;
plot(pCOP,'Color', [0.8 0.8 0.8]);
hold on
plot(pCOP_Filtered,'k');
xlabel('Data Points');
ylabel('COP');
legend('Raw Data','Filtered Data');
hold off
figname = "COP_Filtered" + ".png";
saveas(fig,figname);

end

function [tci_Class, verification_Class] = crossvalidation(main_dataset)

% Create the cross-validation object and perform cross-validation
cross_validation_object = cvpartition(length(main_dataset),'HoldOut',0.1);

% Allocate cross-validated data to TCI- and verification classes
tci_class_indices = cross_validation_object.training;
```

```
verification_class_indices = cross_validation_object.test;
tci_Class = main_dataset(tci_class_indices,:);
verification_Class = main_dataset(verification_class_indices,:);

end

function best_net = singlelayerANN(tci_class, max_HNs, max_reps, algorithm, trainID, testID)

TCI_Inputs_1HL = tci_class(1:8,:);
TCI_Outputs_1HL = tci_class(9,:);

% Initialise training time array
tempTraining_Time = zeros(max_HNs,max_reps);

% Initialise placeholder subset error arrays
tempMSE_Training = zeros(max_HNs,max_reps);
tempRMSE_Training = zeros(max_HNs,max_reps);
tempR_2_Training = zeros(max_HNs,max_reps);
tempMAPE_Training = zeros(max_HNs,max_reps);

tempMSE_Validation = zeros(max_HNs,max_reps);
tempRMSE_Validation = zeros(max_HNs,max_reps);
tempR_2_Validation = zeros(max_HNs,max_reps);
tempMAPE_Validation = zeros(max_HNs,max_reps);

tempMSE_Test = zeros(max_HNs,max_reps);
tempRMSE_Test = zeros(max_HNs,max_reps);
tempR_2_Test = zeros(max_HNs,max_reps);
tempMAPE_Test = zeros(max_HNs,max_reps);

% Initialise cell array to store ANN iterations and best ANN per repetition
ANN_Iteration_Array = cell(max_HNs, max_reps);

for hidden_neurons = 1:max_HNs
    for reps = 1:max_reps

        % Start timer
        Start_Time = tic;

        % Initialise ANN parameters for training and allocate data to subsets
        net = feedforwardnet(hidden_neurons,algorithm);
        net.layers{1}.transferFcn = 'logsig';
        net.layers{end}.transferFcn = 'purelin';
        net = configure(net,TCI_Inputs_1HL,TCI_Outputs_1HL);
        net.divideParam.trainRatio = 0.7;
        net.divideParam.valRatio = 0.2;
        net.divideParam.testRatio = 0.1;
        net.trainParam.showWindow = 0;
```

```
        % Train ANN and store a record of the training time parameters
        [net,training_record] = train(net,TCI_Inputs_1HL,TCI_Outputs_1HL);

        % Make output predictions
        ANN_predictions = net(TCI_Inputs_1HL);

        % Extract the training, validation, and testing subsets
        Training_predictions = ANN_predictions(training_record.trainInd');
        Validation_predictions = ANN_predictions(training_record.valInd');
        Test_predictions = ANN_predictions(training_record.testInd');
        Training_targets = TCI_Outputs_1HL(training_record.trainInd');
        Validation_targets = TCI_Outputs_1HL(training_record.valInd');
        Test_targets = TCI_Outputs_1HL(training_record.testInd');

        % Performance indicators
        tempMSE_Training(hidden_neurons,reps) =
mse(net,Training_targets,Training_predictions);
        tempRMSE_Training(hidden_neurons,reps) =
sqrt(tempMSE_Training(hidden_neurons,reps));
        tempR_2_Training(hidden_neurons,reps) =
regression(Training_targets,Training_predictions);
        tempMAPE_Training(hidden_neurons,reps) =
mape(Training_predictions,Training_targets,"omitzero");

        tempMSE_Validation(hidden_neurons,reps) =
mse(net,Validation_targets,Validation_predictions);
        tempRMSE_Validation(hidden_neurons,reps) =
sqrt(tempMSE_Validation(hidden_neurons,reps));
        tempR_2_Validation(hidden_neurons,reps) =
regression(Validation_targets,Validation_predictions);
        tempMAPE_Validation(hidden_neurons,reps) =
mape(Validation_predictions,Validation_targets,"omitzero");

        tempMSE_Test(hidden_neurons,reps) = mse(net,Test_targets,Test_predictions);
        tempRMSE_Test(hidden_neurons,reps) = sqrt(tempMSE_Test(hidden_neurons,reps));
        tempR_2_Test(hidden_neurons,reps) = regression(Test_targets,Test_predictions);
        tempMAPE_Test(hidden_neurons,reps) = mape(Test_predictions,Test_targets,"omitzero");

        ANN_Iteration_Array{hidden_neurons,reps} = net;

        % Stop timer
        tempTraining_Time(hidden_neurons,reps) = toc(Start_Time);
    end
end

% Find the best ANN based on performance indicators
Best_RMSE = min(tempRMSE_Validation,[],'all');
```

```
 [~, RMSE_best_index] = min(tempRMSE_Validation(:));
 [RMSE_row_best, RMSE_col_best] = ind2sub(size(tempRMSE_Validation), RMSE_best_index);
 Best_R_2 = max(tempR_2_Validation,[],'all');
 [~, R_2_best_index] = max(tempR_2_Validation(:));
 [R_2_row_best, R_2_col_best] = ind2sub(size(tempR_2_Validation), R_2_best_index);
 Best_MAPE = min(tempMAPE_Validation,[],'all');
 [~, MAPE_best_index] = min(tempMAPE_Validation(:));
 [MAPE_row_best, MAPE_col_best] = ind2sub(size(tempMAPE_Validation), MAPE_best_index);

 % Calculate the average of performance indicators per repetition
 RMSE_Training = mean(tempRMSE_Training,2);
 R_2_Training = mean(tempR_2_Training,2);
 % MAPE_Training = mean(tempMAPE_Training,2);

 RMSE_Validation = mean(tempRMSE_Validation,2);
 R_2_Validation = mean(tempR_2_Validation,2);
 % MAPE_Validation = mean(tempMAPE_Validation,2);

 RMSE_Test = mean(tempRMSE_Test,2);
 R_2_Test = mean(tempR_2_Test,2);
 % MAPE_Test = mean(tempMAPE_Test,2);

 Training_Time = mean(tempTraining_Time,2);

 % Return the best ANN number of HNs and lowest R^2
 best_net = R_2_row_best;

 % Calculate total training time
 Total_Training_Time = sum(tempTraining_Time,"all");

 % Display the best performance indicators and training time
 disp("The best ANN contains " + R_2_row_best + " hidden neurons");
 disp("Best RMSE = " + Best_RMSE + " at: " + RMSE_row_best + " hidden neurons and repetition:
" + RMSE_col_best);
 disp("Best R_2 = " + Best_R_2 + " at: " + R_2_row_best + " hidden neurons and repetition: "
+ R_2_col_best);
 disp("Best MAPE = " + Best_MAPE + " at: " + MAPE_row_best + " hidden neurons and repetition:
" + MAPE_col_best);
 disp("Total training time = " + floor(Total_Training_Time/60) + "min " +
round(mod(Total_Training_Time,60),0) + "s");

 % Plot the RMSE and R^2 values versus hidden neurons
 fig = figure;
 yyaxis("left");
 plot(1:max_HNs,RMSE_Training);
 hold on
 plot(1:max_HNs,RMSE_Validation,':');
 plot(1:max_HNs,RMSE_Test,'--');
```

```
 ylabel("RMSE");
 yyaxis("right");
 plot(1:max_HNs,R_2_Validation);
 plot(1:max_HNs,R_2_Training,':');
 plot(1:max_HNs,R_2_Test,'--');
 ylabel("R^2");
 legend("RMSE_T_r_a_i_n_i_n_g","RMSE_V_a_l_i_d_a_t_i_o_n","RMSE_T_e_s_t",
"R^2_T_r_a_i_n_i_n_g","R^2_V_a_l_i_d_a_t_i_o_n","R^2_T_e_s_t",'Location','southoutside','NumC
olumns',2);
 hold off
 figname = "R2_RMSE_vs_HN_" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % replace outliers and apply noise spike filters to smoothen results
 Window_Size = 20;
 Forgetting_Factor = 0.45;
 EWMA_Function = dsp.MovingAverage('Method','Exponential
weighting','ForgettingFactor',Forgetting_Factor);

 RMSE_Training = EWMA_Function(RMSE_Training);
 R_2_Training = EWMA_Function(R_2_Training);
 RMSE_Validation = EWMA_Function(RMSE_Validation);
 R_2_Validation = EWMA_Function(R_2_Validation);
 RMSE_Test = EWMA_Function(RMSE_Test);
 R_2_Test = EWMA_Function(R_2_Test);

 filloutliers(RMSE_Training,'linear','movmedian',Window_Size);
 filloutliers(R_2_Training,'linear','movmedian',Window_Size);
 filloutliers(RMSE_Validation,'linear','movmedian',Window_Size);
 filloutliers(R_2_Validation,'linear','movmedian',Window_Size);
 filloutliers(RMSE_Test,'linear','movmedian',Window_Size);
 filloutliers(R_2_Test,'linear','movmedian',Window_Size);

 % Plot the RMSE and R^2 values versus hidden neurons with filter applied
 fig = figure;
 yyaxis("left");
 plot(1:max_HNs,RMSE_Training);
 hold on
 plot(1:max_HNs,RMSE_Validation,':');
 plot(1:max_HNs,RMSE_Test,'--');
 ylabel("RMSE");
 yyaxis("right");
 plot(1:max_HNs,R_2_Validation);
 plot(1:max_HNs,R_2_Training,':');
 plot(1:max_HNs,R_2_Test,'--');
 ylabel("R^2");
```

```
 legend("RMSE_T_r_a_i_n_i_n_g","RMSE_V_a_l_i_d_a_t_i_o_n","RMSE_T_e_s_t",
"R^2_T_r_a_i_n_i_n_g","R^2_V_a_l_i_d_a_t_i_o_n","R^2_T_e_s_t','Location','southoutside','NumC
olumns',2);
 hold off
 figname = "R2_RMSE_vs_HN_Filtered_" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % Plot the average training times for each ANN configuration
 fig = figure;
 plot(1:max_HNs, Training_Time);
 hold on
 ylabel("Training time (s)");
 xlabel("Number of hidden neurons");
 hold off
 figname = "Training_Time_1HL" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % Save workspace
 workspace_ID = testID + "_" + trainID + "_workspace";
 save(workspace_ID);

 end


 function best_net = twolayerANN(tci_class, max_HNs_HL1, max_HNs_HL2, max_reps, algorithm,
trainID, testID)

 TCI_Inputs = tci_class(1:8,:);
 TCI_Outputs = tci_class(9,:);

 % Initialise training time array
 tempTraining_Time = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);

 % Initialise subset error arrays
 tempMSE_Training = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempRMSE_Training = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempR_2_Training = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempMAPE_Training = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);

 tempMSE_Validation = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempRMSE_Validation = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempR_2_Validation = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempMAPE_Validation = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);

 tempMSE_Test = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempRMSE_Test = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempR_2_Test = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
 tempMAPE_Test = zeros(max_HNs_HL1, max_HNs_HL2, max_reps);
```

```
% Initialise cell array to store ANN iterations and best ANN per repetition
ANN_Iteration_Array = cell(max_HNs_HL1, max_HNs_HL2, max_reps);

for HNs_HL1 = 1:max_HNs_HL1
    for HNs_HL2 = 1:max_HNs_HL2
        for reps = 1:max_reps

                % Start timer
                Start_Time = tic;

                % Initialise ANN parameters for training and allocate data to subsets
                net = feedforwardnet([HNs_HL1 HNs_HL2], algorithm);
                net.layers{1}.transferFcn = 'logsig';
                net.layers{2}.transferFcn = 'logsig';
                net.layers{end}.transferFcn = 'purelin';
                net = configure(net,TCI_Inputs,TCI_Outputs);
                net.divideParam.trainRatio = 0.7;
                net.divideParam.valRatio = 0.2;
                net.divideParam.testRatio = 0.1;
                net.trainParam.showWindow = 0;

                % Train ANN and store a record of the training time parameters
                [net,training_record] = train(net,TCI_Inputs,TCI_Outputs);

                % Make output predictions
                ANN_predictions = net(TCI_Inputs);

                % Extract the training, validation, and testing subsets
                Training_predictions = ANN_predictions(training_record.trainInd');
                Validation_predictions = ANN_predictions(training_record.valInd');
                Test_predictions = ANN_predictions(training_record.testInd');
                Training_targets = TCI_Outputs(training_record.trainInd');
                Validation_targets = TCI_Outputs(training_record.valInd');
                Test_targets = TCI_Outputs(training_record.testInd');

                % Performance indicators
                tempMSE_Training(HNs_HL1,HNs_HL2,reps) =
mse(net,Training_targets,Training_predictions);
                tempRMSE_Training(HNs_HL1,HNs_HL2,reps) =
sqrt(tempMSE_Training(HNs_HL1,HNs_HL2,reps));
                tempR_2_Training(HNs_HL1,HNs_HL2,reps) =
regression(Training_targets,Training_predictions);
                tempMAPE_Training(HNs_HL1,HNs_HL2,reps) =
mape(Training_predictions,Training_targets,"omitzero");

                tempMSE_Validation(HNs_HL1,HNs_HL2,reps) =
mse(net,Validation_targets,Validation_predictions);
```

```
                tempRMSE_Validation(HNs_HL1,HNs_HL2,reps) =
sqrt(tempMSE_Validation(HNs_HL1,HNs_HL2,reps));
                tempR_2_Validation(HNs_HL1,HNs_HL2,reps) =
regression(Validation_targets,Validation_predictions);
                tempMAPE_Validation(HNs_HL1,HNs_HL2,reps) =
mape(Validation_predictions,Validation_targets,"omitzero");

                tempMSE_Test(HNs_HL1,HNs_HL2,reps) = mse(net,Test_targets,Test_predictions);
                tempRMSE_Test(HNs_HL1,HNs_HL2,reps) =
sqrt(tempMSE_Test(HNs_HL1,HNs_HL2,reps));
                tempR_2_Test(HNs_HL1,HNs_HL2,reps) =
regression(Test_targets,Test_predictions);
                tempMAPE_Test(HNs_HL1,HNs_HL2,reps) =
mape(Test_predictions,Test_targets,"omitzero");

                ANN_Iteration_Array{HNs_HL1,HNs_HL2,reps} = net;

                % Stop timer
                tempTraining_Time(HNs_HL1,HNs_HL2,reps) = toc(Start_Time);
        end
    end

 end

 % Find the best ANN based on performance indicators
 [Best_MSE, MSE_best_index] = min(tempMSE_Validation(:));
 [MSE_row_best, MSE_col_best, MSE_page_best] = ind2sub(size(tempMSE_Validation),
MSE_best_index);
 [Best_RMSE, RMSE_best_index] = min(tempRMSE_Validation(:));
 [RMSE_row_best, RMSE_col_best, RMSE_page_best] = ind2sub(size(tempRMSE_Validation),
RMSE_best_index);
 [Best_R_2, R_2_best_index] = max(tempR_2_Validation(:));
 [R_2_row_best, R_2_col_best, R_2_page_best] = ind2sub(size(tempR_2_Validation),
R_2_best_index);
 [Best_MAPE, MAPE_best_index] = min(tempMAPE_Validation(:));
 [MAPE_row_best, MAPE_col_best, MAPE_page_best] = ind2sub(size(tempMAPE_Validation),
MAPE_best_index);

 % Calculate the average of performance indicators per repetition
 RMSE_Training = mean(tempRMSE_Training,3);
 R_2_Training = mean(tempR_2_Training,3);
 MAPE_Training = mean(tempMAPE_Training,3);

 RMSE_Validation = mean(tempRMSE_Validation,3);
 R_2_Validation = mean(tempR_2_Validation,3);
 MAPE_Validation = mean(tempMAPE_Validation,3);

 RMSE_Test = mean(tempRMSE_Test,3);
```

```
 R_2_Test = mean(tempR_2_Test,3);
 MAPE_Test = mean(tempMAPE_Test,3);

 Training_Time = mean(tempTraining_Time,3);
 Total_Training_Time = sum(tempTraining_Time,'all');

 best_net = [R_2_row_best R_2_col_best];

 % Display the best performance indicators and training time
 disp("The best performing architecture: " + R_2_row_best + "-" + R_2_col_best);
 disp("Best RMSE = " + Best_RMSE + " at: " + RMSE_row_best + "-" + RMSE_col_best);
 disp("Best R_2 = " + Best_R_2 + " at: " + R_2_row_best + "-" + R_2_col_best);
 disp("Best MAPE = " + Best_MAPE + " at: " + MAPE_row_best + "-" + MAPE_col_best);
 disp("Total training time = " + floor(Total_Training_Time/60) + "min " +
round(mod(Total_Training_Time,60),0) + "s");

 % Plot the RMSE values versus hidden neurons in the hidden layers
 [X,Y] = meshgrid(1:max_HNs_HL1, 1:max_HNs_HL2);
 fig = figure;
 surf(X,Y,RMSE_Validation);
 xlabel("Hidden neurons in 1st hidden layer");
 ylabel("Hidden neurons in 2nd hidden layer");
 zlabel("RMSE_V_a_l_i_d_a_t_i_o_n");
 figname = "RMSE_vs_HNs_2HL" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % Plot the R^2 values versus hidden neurons in the hidden layers
 fig = figure;
 surf(X,Y,R_2_Validation);
 xlabel("Hidden neurons in 1st hidden layer");
 ylabel("Hidden neurons in 2nd hidden layer");
 zlabel("R^2_V_a_l_i_d_a_t_i_o_n");
 figname = "R^2_vs_HNs_2HL" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % Plot the training times for each ANN configuration
 fig = figure;
 surf(X,Y,Training_Time);
 hold on
 xlabel("Hidden neurons in 1st hidden layer");
 ylabel("Hidden neurons in 2nd hidden layer");
 zlabel("Training time (s)");
 hold off
 figname = "Training_Time_2HL" + trainID + "_" + testID + ".png";
 saveas(fig,figname);

 % Save workspace
 workspace_ID = testID + "_" + trainID + "_workspace";
```

```
 save(workspace_ID);

 end

 function [best_net, best_net_tr] = retrain1HL_ANN(num_HNs, algorithm, tci_class, max_reps,
testID, config)

 % Allocate tci class data to inputs and outputs
 TCI_Inputs = tci_class(1:8,:);
 TCI_Outputs = tci_class(9,:);

 % Initialise training time array
 Training_Time = zeros(max_reps,1);

 % Initialise subset error arrays
 MSE_Training = zeros(max_reps,1);
 RMSE_Training = zeros(max_reps,1);
 R_2_Training = zeros(max_reps,1);
 MAPE_Training = zeros(max_reps,1);

 MSE_Validation = zeros(max_reps,1);
 RMSE_Validation = zeros(max_reps,1);
 R_2_Validation = zeros(max_reps,1);
 MAPE_Validation = zeros(max_reps,1);

 MSE_Test = zeros(max_reps,1);
 RMSE_Test = zeros(max_reps,1);
 R_2_Test = zeros(max_reps,1);
 MAPE_Test = zeros(max_reps,1);

 % Initialise cell array to store ANN iterations
 ANN_Iteration_Array = cell(max_reps,1);
 Training_Record_Array = cell(max_reps,1);

 for reps = 1:max_reps

     % Start timer
     Start_Time = tic;

     % Initialise ANN parameters for training and allocate data to subsets
     net = feedforwardnet(num_HNs,algorithm);
     net.layers{1}.transferFcn = 'logsig';
     net.layers{end}.transferFcn = 'purelin';
     net = configure(net,TCI_Inputs,TCI_Outputs);
     net.divideParam.trainRatio = 0.7;
     net.divideParam.valRatio = 0.2;
     net.divideParam.testRatio = 0.1;
     net.trainParam.showWindow = 0;
```

```
    % Train ANN and store a record of the training time parameters
    [net,training_record] = train(net,TCI_Inputs,TCI_Outputs);

    % Make output predictions
    ANN_predictions = net(TCI_Inputs);

    % Extract the training, validation, and testing subsets
    Training_predictions = ANN_predictions(training_record.trainInd');
    Validation_predictions = ANN_predictions(training_record.valInd');
    Test_predictions = ANN_predictions(training_record.testInd');
    Training_targets = TCI_Outputs(training_record.trainInd');
    Validation_targets = TCI_Outputs(training_record.valInd');
    Test_targets = TCI_Outputs(training_record.testInd');

    % Performance indicators
    MSE_Training(reps) = mse(net,Training_targets,Training_predictions);
    RMSE_Training(reps) = sqrt(MSE_Training(reps));
    R_2_Training(reps) = regression(Training_targets,Training_predictions);
    MAPE_Training(reps) = mape(Training_predictions,Training_targets,"omitzero");

    MSE_Validation(reps) = mse(net,Validation_targets,Validation_predictions);
    RMSE_Validation(reps) = sqrt(MSE_Validation(reps));
    R_2_Validation(reps) = regression(Validation_targets,Validation_predictions);
    MAPE_Validation(reps) = mape(Validation_predictions,Validation_targets,"omitzero");

    MSE_Test(reps) = mse(net,Test_targets,Test_predictions);
    RMSE_Test(reps) = sqrt(MSE_Test(reps));
    R_2_Test(reps) = regression(Test_targets,Test_predictions);
    MAPE_Test(reps) = mape(Test_predictions,Test_targets,"omitzero");

    ANN_Iteration_Array{reps} = net;
    Training_Record_Array{reps} = training_record;

    % Stop timer
    Training_Time(reps) = toc(Start_Time);

end

% Find the best ANN based on performance indicators
[Best_MSE, MSE_best_index] = min(MSE_Validation);
[Best_RMSE, RMSE_best_index] = min(RMSE_Validation);
[Best_R_2, R_2_best_index] = max(R_2_Validation);
[Best_MAPE, MAPE_best_index] = min(MAPE_Validation);

% Calculate total training time
Total_Training_Time = sum(Training_Time);
```

```
 % Display the best performance indicators and training time
 disp("The best RMSE & achieved at: " + RMSE_best_index + ", R^2 at: " + R_2_best_index + ",
and MAPE at: " + MAPE_best_index);
 disp("Best MSE = " + Best_MSE);
 disp("Best RMSE = " + Best_RMSE);
 disp("Best R_2 = " + Best_R_2);
 disp("Best MAPE = " + Best_MAPE);
 disp("Total training time = " + floor(Total_Training_Time/60) + "min " +
round(mod(Total_Training_Time,60),0) + "s");

 % Plot the RMSE and R^2 values versus hidden neurons
 fig = figure;
 yyaxis("left");
 plot(1:max_reps,RMSE_Training);
 hold on
 plot(1:max_reps,RMSE_Validation,':');
 plot(1:max_reps,RMSE_Test,'--');
 ylabel("RMSE");
 yyaxis("right");
 plot(1:max_reps,R_2_Validation);
 plot(1:max_reps,R_2_Training,':');
 plot(1:max_reps,R_2_Test,'--');
 ylabel("R^2");
 legend("RMSE_T_r_a_i_n_i_n_g","RMSE_V_a_l_i_d_a_t_i_o_n","RMSE_T_e_s_t",
"R^2_T_r_a_i_n_i_n_g","R^2_V_a_l_i_d_a_t_i_o_n","R^2_T_e_s_t",'Location','southoutside','NumC
olumns',2);
 hold off
 figname = "R2_RMSE_vs_reps_" + testID + ".png";
 saveas(fig,figname);

 best_net = ANN_Iteration_Array{R_2_best_index};
 best_net_tr = Training_Record_Array{R_2_best_index};

 fig = figure;
 plotperform(best_net_tr);
 figname = "Training_Performance_" + config + "_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plottrainstate(best_net_tr);
 figname = "Training_State_" + config + "_" + testID + ".png";
 saveas(fig,figname);


 end

 function [best_net, best_net_tr] = retrain2HL_ANN(num_HNs_HL1, num_HNs_HL2, algorithm,
tci_class, max_reps, testID, config)
```

```
% Allocate tci class data to inputs and outputs
TCI_Inputs = tci_class(1:8,:);
TCI_Outputs = tci_class(9,:);

% Initialise training time array
Training_Time = zeros(max_reps,1);

% Initialise subset error arrays
MSE_Training = zeros(max_reps,1);
RMSE_Training = zeros(max_reps,1);
R_2_Training = zeros(max_reps,1);
MAPE_Training = zeros(max_reps,1);

MSE_Validation = zeros(max_reps,1);
RMSE_Validation = zeros(max_reps,1);
R_2_Validation = zeros(max_reps,1);
MAPE_Validation = zeros(max_reps,1);

MSE_Test = zeros(max_reps,1);
RMSE_Test = zeros(max_reps,1);
R_2_Test = zeros(max_reps,1);
MAPE_Test = zeros(max_reps,1);

% Initialise cell array to store ANN iterations
ANN_Iteration_Array = cell(max_reps,1);
Training_Record_Array = cell(max_reps,1);

for reps = 1:max_reps

    % Start timer
    Start_Time = tic;

    % Initialise ANN parameters for training and allocate data to subsets
    net = feedforwardnet([num_HNs_HL1, num_HNs_HL2],algorithm);
    net.layers{1}.transferFcn = 'logsig';
    net.layers{2}.transferFcn = 'logsig';
    net.layers{end}.transferFcn = 'purelin';
    net = configure(net,TCI_Inputs,TCI_Outputs);
    net.divideParam.trainRatio = 0.7;
    net.divideParam.valRatio = 0.2;
    net.divideParam.testRatio = 0.1;
    net.trainParam.showWindow = 0;

    % Train ANN and store a record of the training time parameters
    [net,training_record] = train(net,TCI_Inputs,TCI_Outputs);

    % Make output predictions
```

```
    ANN_predictions = net(TCI_Inputs);

    % Extract the training, validation, and testing subsets
    Training_predictions = ANN_predictions(training_record.trainInd');
    Validation_predictions = ANN_predictions(training_record.valInd');
    Test_predictions = ANN_predictions(training_record.testInd');
    Training_targets = TCI_Outputs(training_record.trainInd');
    Validation_targets = TCI_Outputs(training_record.valInd');
    Test_targets = TCI_Outputs(training_record.testInd');

    % Performance indicators
    MSE_Training(reps) = mse(net,Training_targets,Training_predictions);
    RMSE_Training(reps) = sqrt(MSE_Training(reps));
    R_2_Training(reps) = regression(Training_targets,Training_predictions);
    MAPE_Training(reps) = mape(Training_predictions,Training_targets,"omitzero");

    MSE_Validation(reps) = mse(net,Validation_targets,Validation_predictions);
    RMSE_Validation(reps) = sqrt(MSE_Validation(reps));
    R_2_Validation(reps) = regression(Validation_targets,Validation_predictions);
    MAPE_Validation(reps) = mape(Validation_predictions,Validation_targets,"omitzero");

    MSE_Test(reps) = mse(net,Test_targets,Test_predictions);
    RMSE_Test(reps) = sqrt(MSE_Test(reps));
    R_2_Test(reps) = regression(Test_targets,Test_predictions);
    MAPE_Test(reps) = mape(Test_predictions,Test_targets,"omitzero");

    ANN_Iteration_Array{reps} = net;
    Training_Record_Array{reps} = training_record;

    % Stop timer
    Training_Time(reps) = toc(Start_Time);

end

% Find the best ANN based on performance indicators
[Best_MSE, MSE_best_index] = min(MSE_Validation);
[Best_RMSE, RMSE_best_index] = min(RMSE_Validation);
[Best_R_2, R_2_best_index] = max(R_2_Validation);
[Best_MAPE, MAPE_best_index] = min(MAPE_Validation);

% Calculate total training time
Total_Training_Time = sum(Training_Time);

% Display the best performance indicators and training time
disp("The best RMSE & MSE achieved at: " + RMSE_best_index + ", R^2 at: " + R_2_best_index +
", and MAPE at: " + MAPE_best_index);
disp("Best_MSE = " + Best_MSE);
disp("Best RMSE = " + Best_RMSE);
```

```
 disp("Best R_2 = " + Best_R_2);
 disp("Best MAPE = " + Best_MAPE);
 disp("Total training time = " + floor(Total_Training_Time/60) + "min " +
round(mod(Total_Training_Time,60),0) + "s");

 % Plot the RMSE and R^2 values versus hidden neurons
 fig = figure;
 yyaxis("left");
 plot(1:max_reps,RMSE_Training);
 hold on
 plot(1:max_reps,RMSE_Validation,':');
 plot(1:max_reps,RMSE_Test,'--');
 ylabel("RMSE");
 yyaxis("right");
 plot(1:max_reps,R_2_Validation);
 plot(1:max_reps,R_2_Training,':');
 plot(1:max_reps,R_2_Test,'--');
 ylabel("R^2");
 legend("RMSE_T_r_a_i_n_i_n_g","RMSE_V_a_l_i_d_a_t_i_o_n","RMSE_T_e_s_t",
"R^2_T_r_a_i_n_i_n_g","R^2_V_a_l_i_d_a_t_i_o_n","R^2_T_e_s_t",'Location','southoutside','NumC
olumns',2);
 hold off
 figname = "R2_RMSE_vs_reps_2HL_" + testID + ".png";
 saveas(fig,figname);

 best_net = ANN_Iteration_Array{R_2_best_index};
 best_net_tr = Training_Record_Array{R_2_best_index};

 fig = figure;
 plotperform(best_net_tr);
 figname = "Training_Performance_2HL_" + config + "_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plottrainstate(best_net_tr);
 figname = "Training_State_2HL_" + config + "_" + testID + ".png";
 saveas(fig,figname);

 end

 function [best_net, Best_R_2] = compare_ANNs(O1HL_C1_ANN, O1HL_C2_ANN, O1HL_C3_ANN,
O1HL_C4_ANN, O2HL_C1_ANN, O2HL_C2_ANN, O2HL_C3_ANN, O2HL_C4_ANN, O1HL_C1_TR, O1HL_C2_TR,
O1HL_C3_TR, O1HL_C4_TR, O2HL_C1_TR, O2HL_C2_TR, O2HL_C3_TR, O2HL_C4_TR, tci_class, testID)

 % Allocate tci class data to inputs and outputs
 TCI_Inputs = tci_class(1:8,:);
 TCI_Outputs = tci_class(9,:);
```

```
% Make output predictions
O1HL_C1_ANN_predictions = O1HL_C1_ANN(TCI_Inputs);
O1HL_C2_ANN_predictions = O1HL_C2_ANN(TCI_Inputs);
O1HL_C3_ANN_predictions = O1HL_C3_ANN(TCI_Inputs);
O1HL_C4_ANN_predictions = O1HL_C4_ANN(TCI_Inputs);
O2HL_C1_ANN_predictions = O2HL_C1_ANN(TCI_Inputs);
O2HL_C2_ANN_predictions = O2HL_C2_ANN(TCI_Inputs);
O2HL_C3_ANN_predictions = O2HL_C3_ANN(TCI_Inputs);
O2HL_C4_ANN_predictions = O2HL_C4_ANN(TCI_Inputs);

% Extract the validation subset predictions and targets from each ANN
VP_O1HL_C1 = O1HL_C1_ANN_predictions(O1HL_C1_TR.valInd');
VT_O1HL_C1 = TCI_Outputs(O1HL_C1_TR.valInd');

VP_O1HL_C2 = O1HL_C2_ANN_predictions(O1HL_C2_TR.valInd');
VT_O1HL_C2 = TCI_Outputs(O1HL_C2_TR.valInd');

VP_O1HL_C3 = O1HL_C3_ANN_predictions(O1HL_C3_TR.valInd');
VT_O1HL_C3 = TCI_Outputs(O1HL_C3_TR.valInd');

VP_O1HL_C4 = O1HL_C4_ANN_predictions(O1HL_C4_TR.valInd');
VT_O1HL_C4 = TCI_Outputs(O1HL_C4_TR.valInd');

VP_O2HL_C1 = O2HL_C1_ANN_predictions(O2HL_C1_TR.valInd');
VT_O2HL_C1 = TCI_Outputs(O2HL_C1_TR.valInd');

VP_O2HL_C2 = O2HL_C2_ANN_predictions(O2HL_C2_TR.valInd');
VT_O2HL_C2 = TCI_Outputs(O2HL_C2_TR.valInd');

VP_O2HL_C3 = O2HL_C3_ANN_predictions(O2HL_C3_TR.valInd');
VT_O2HL_C3 = TCI_Outputs(O2HL_C3_TR.valInd');

VP_O2HL_C4 = O2HL_C4_ANN_predictions(O2HL_C4_TR.valInd');
VT_O2HL_C4 = TCI_Outputs(O2HL_C4_TR.valInd');

% Predict the validation subset R^2 value for each ANN
R_2_01HL_C1 = regression(VT_O1HL_C1, VP_O1HL_C1);
R_2_01HL_C2 = regression(VT_O1HL_C2, VP_O1HL_C2);
R_2_01HL_C3 = regression(VT_O1HL_C3, VP_O1HL_C3);
R_2_01HL_C4 = regression(VT_O1HL_C4, VP_O1HL_C4);
R_2_02HL_C1 = regression(VT_O2HL_C1, VP_O2HL_C1);
R_2_02HL_C2 = regression(VT_O2HL_C2, VP_O2HL_C2);
R_2_02HL_C3 = regression(VT_O2HL_C3, VP_O2HL_C3);
R_2_02HL_C4 = regression(VT_O2HL_C4, VP_O2HL_C4);

R_2_Array = [R_2_01HL_C1, R_2_01HL_C2, R_2_01HL_C3, R_2_01HL_C4, R_2_02HL_C1, R_2_02HL_C2,
R_2_02HL_C3, R_2_02HL_C4];
```

```
[Best_R_2, R_2_best_index] = max(R_2_Array);

switch R_2_best_index
    case 1
        best_net = O1HL_C1_ANN;
        best_tr = O1HL_C1_TR;
    case 2
        best_net = O1HL_C2_ANN;
        best_tr = O1HL_C2_TR;
    case 3
        best_net = O1HL_C3_ANN;
        best_tr = O1HL_C3_TR;
    case 4
        best_net = O1HL_C4_ANN;
        best_tr = O1HL_C4_TR;
    case 5
        best_net = O2HL_C1_ANN;
        best_tr = O2HL_C1_TR;
    case 6
        best_net = O2HL_C2_ANN;
        best_tr = O2HL_C2_TR;
    case 7
        best_net = O2HL_C3_ANN;
        best_tr = O2HL_C3_TR;
    case 8
        best_net = O2HL_C4_ANN;
        best_tr = O2HL_C4_TR;
end

% Evaluate the performance of the optimal network
best_net_predictions = best_net(TCI_Inputs);
Training_predictions = best_net_predictions(best_tr.trainInd');
Validation_predictions = best_net_predictions(best_tr.valInd');
Test_predictions = best_net_predictions(best_tr.testInd');
Training_targets = TCI_Outputs(best_tr.trainInd');
Validation_targets = TCI_Outputs(best_tr.valInd');
Test_targets = TCI_Outputs(best_tr.testInd');

MSE_Training = mse(best_net,Training_targets,Training_predictions);
RMSE_Training = sqrt(MSE_Training);
R_2_Training = regression(Training_targets,Training_predictions);
MAPE_Training = mape(Training_predictions,Training_targets,"omitzero");

MSE_Validation = mse(best_net,Validation_targets,Validation_predictions);
RMSE_Validation = sqrt(MSE_Validation);
R_2_Validation = regression(Validation_targets,Validation_predictions);
MAPE_Validation = mape(Validation_predictions,Validation_targets,"omitzero");
```

```
 MSE_Test = mse(best_net,Test_targets,Test_predictions);
 RMSE_Test = sqrt(MSE_Test);
 R_2_Test = regression(Test_targets,Test_predictions);
 MAPE_Test = mape(Test_predictions,Test_targets,"omitzero");

 MSE_Overall = mse(best_net,TCI_Outputs,best_net_predictions);
 RMSE_Overall = sqrt(MSE_Overall);
 R_2_Overall = regression(TCI_Outputs,best_net_predictions);
 MAPE_Overall = mape(best_net_predictions,TCI_Outputs,"omitzero");

 % Display the performance indicators
 disp("Best R^2 = " + Best_R_2 + " for ANN configuration " + R_2_best_index);
 disp("Overall RMSE = " + RMSE_Overall + " Training RMSE = " + RMSE_Training + " Validation
RMSE = " + RMSE_Validation + " Test RMSE = " + RMSE_Test);
 disp("Overall R_2 = " + R_2_Overall + " Training R_2 = " + R_2_Training + " Validation R_2 =
" + R_2_Validation + " Test R_2 = " + R_2_Test);
 disp("Overall MAPE = " + MAPE_Overall + " Training MAPE = " + MAPE_Training + " Validation
MAPE = " + MAPE_Validation + " Test MAPE = " + MAPE_Test);

 % Correct the predicted outputs

 % Plot the regression figures for the main set and all subsets
 fig = figure;
 plotregression(Training_predictions,Training_targets,'Training',Validation_predictions,Valid
ation_targets,'Validation',Test_predictions,Test_targets,'Test',best_net_predictions,TCI_Outp
uts,'Overall');
 figname = "Regression_Plot_Best_Net_" + testID + ".png";
 saveas(fig,figname);
 end

 function [w_ih, w_ho, b_h, b_o] = Derive_FIO_Equation_1HL(best_net)

 w_ih = best_net.IW{1};  % Read input-to-hidden layer weights
 b_h = best_net.b{1};     % Read hidden layer biases
 w_ho = best_net.LW{2,1}; % Read hidden-to-output layer weights
 b_o = best_net.b{2};       % Read output layer bias

 end

 function [w_ih, w_hh, w_ho, b_h1, b_h2, b_o] = Derive_FIO_Equation_2HL(best_net)

 w_ih = best_net.IW{1};  % Read input-to-hidden layer 1 weights
 b_h1 = best_net.b{1};     % Read 1st hidden layer biases
 w_hh = best_net.LW{2,1}; % Read hidden-to-hidden layer weights
 b_h2 = best_net.b{2};       % Read 2nd hidden layer biases
 w_ho = best_net.LW{3,2};
 b_o = best_net.b{3};

 end
```

```
 function [Evaluation_Results, Input_minmax_range, range_COP,
Corrected_Varied_Evaluation_Inputs] = Evaluate_FP_Performance(filtered_data, tci_class,
best_net, testID)

 % Model evaluation - impact measurement per input

 % Find the range of each input
 range_Evap_Inlet_Water_Temp = [min(filtered_data(:,1)) max(filtered_data(:,1))];
 range_Evap_Outlet_Water_Temp = [min(filtered_data(:,2)) max(filtered_data(:,2))];
 range_Evap_Water_Flow = [min(filtered_data(:,3)) max(filtered_data(:,3))];
 range_Cond_Water_Flow = [min(filtered_data(:,4)) max(filtered_data(:,4))];
 range_Evap_NH3_Inlet_Temp = [min(filtered_data(:,5)) max(filtered_data(:,5))];
 range_Evap_NH3_Outlet_Temp = [min(filtered_data(:,6)) max(filtered_data(:,6))];
 range_WB_Temp = [min(filtered_data(:,7)) max(filtered_data(:,7))];
 range_Comp_Power = [min(filtered_data(:,8)) max(filtered_data(:,8))];
 range_COP = [min(filtered_data(:,9)) max(filtered_data(:,9))];
 Input_minmax_range = [range_Evap_Inlet_Water_Temp; range_Evap_Outlet_Water_Temp;
range_Evap_Water_Flow; range_Cond_Water_Flow; range_Evap_NH3_Inlet_Temp;
range_Evap_NH3_Outlet_Temp; range_WB_Temp; range_Comp_Power];


 % Find the mean value for all inputs
 Input_modes = mean(tci_class(:,1:8),1);

 % Populate inputs with 50 datapoints of their mode
 input_vectors = 50;
 Evaluation_Inputs_Means = zeros(8,input_vectors);
 for evaluation_row = 1:8
     Evaluation_Inputs_Means(evaluation_row,1:input_vectors) = Input_modes(evaluation_row);
 end

 % Assign each input variable with linearly spaced values within their range
 Vary_Evap_Inlet_Water_Temp = linspace(0, 1, input_vectors);
 Vary_Evap_Outlet_Water_Temp = linspace(0, 1, input_vectors);
 Vary_Evap_Water_Flow = linspace(0, 1, input_vectors);
 Vary_Cond_Water_Flow = linspace(0, 1, input_vectors);
 Vary_Evap_NH3_Inlet_Temp = linspace(0, 1, input_vectors);
 Vary_Evap_NH3_Outlet_Temp = linspace(0, 1, input_vectors);
 Vary_WB_Temp = linspace(0, 1, input_vectors);
 Vary_Comp_Power = linspace(0, 1, input_vectors);
 Varied_Evaluation_Inputs = [Vary_Evap_Inlet_Water_Temp; Vary_Evap_Outlet_Water_Temp;
Vary_Evap_Water_Flow; Vary_Cond_Water_Flow; Vary_Evap_NH3_Inlet_Temp;
Vary_Evap_NH3_Outlet_Temp; Vary_WB_Temp; Vary_Comp_Power];


 % Evaluate the relationships between each input and the outputs of the ANN
 Corrected_Varied_Evaluation_Inputs = zeros(8,input_vectors);
 Evaluation_Results_Raw = zeros(8,input_vectors);
```

```
 Evaluation_Results = zeros(8,input_vectors);
 for input_number = 1:8
     Evaluation_Inputs = Evaluation_Inputs_Means;
     Evaluation_Inputs(input_number,:) = Varied_Evaluation_Inputs(input_number,:);
     Evaluation_Predictions = best_net(Evaluation_Inputs);
     Evaluation_Results_Raw(input_number,:) = Evaluation_Predictions;
     Evaluation_Results(input_number,:) =
Evaluation_Results_Raw(input_number,:).*(range_COP(2) - range_COP(1)) + range_COP(1);
     Corrected_Varied_Evaluation_Inputs(input_number,:) =
Varied_Evaluation_Inputs(input_number,:).*(Input_minmax_range(input_number,2) -
Input_minmax_range(input_number,1)) + Input_minmax_range(input_number,1);
 end

 % Plot results
 fig = figure;
 plot(Corrected_Varied_Evaluation_Inputs(1,:), Evaluation_Results(1,:), 'b');
 hold on
 ylabel('COP');
 xlabel('Evaporator inlet water temperature (°C)');
 figname = "Evap_Inlet_H2O_Temp_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plot(Corrected_Varied_Evaluation_Inputs(2,:), Evaluation_Results(2,:), 'b');
 hold on
 ylabel('COP');
 xlabel('Evaporator outlet water temperature (°C)');
 figname = "Evap_Outlet_H2O_Temp_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plot(Corrected_Varied_Evaluation_Inputs(3,:), Evaluation_Results(3,:), 'b');
 hold on
 ylabel('COP');
 xlabel('Evaporator water flow rate (L/s)');
 figname = "Evap_H2O_Flow_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plot(Corrected_Varied_Evaluation_Inputs(4,:), Evaluation_Results(4,:), 'b');
 hold on
 ylabel('COP');
 xlabel('Condenser cooling water flow rate (L/s)');
 figname = "Cond_H2O_Flow_" + testID + ".png";
 saveas(fig,figname);

 fig = figure;
 plot(Corrected_Varied_Evaluation_Inputs(5,:), Evaluation_Results(5,:), 'b');
```

```
hold on
ylabel('COP');
xlabel('Evaporator inlet NH3 temperature (°C)');
figname = "Evap_NH3_Inlet_Temp_" + testID + ".png";
saveas(fig,figname);

fig = figure;
plot(Corrected_Varied_Evaluation_Inputs(6,:), Evaluation_Results(6,:), 'b');
hold on
ylabel('COP');
xlabel('Evaporator outlet NH3 temperature (°C)');
figname = "Evap_NH3_Outlet_Temp_" + testID + ".png";
saveas(fig,figname);

fig = figure;
plot(Corrected_Varied_Evaluation_Inputs(7,:), Evaluation_Results(7,:), 'b');
hold on
ylabel('COP');
xlabel('Wet-bulb temperature (°C)');
figname = "Wet_Bulb_Temp_" + testID + ".png";
saveas(fig,figname);

fig = figure;
plot(Corrected_Varied_Evaluation_Inputs(8,:), Evaluation_Results(8,:), 'b');
hold on
ylabel('COP');
xlabel('Compressor Power (kW)');
figname = "Comp_Power_" + testID + ".png";
saveas(fig,figname);


% Model evaluation - impact measurement of input pairs
% Evaluate evap water temperatures' impact on COP

Varied_Input_Pairs = [Vary_Evap_Inlet_Water_Temp; Vary_Evap_Outlet_Water_Temp];
Input_Pair_minmax_range  = [range_Evap_Inlet_Water_Temp; range_Evap_Outlet_Water_Temp];

num_rows = 50;
num_cols = 50;
Input_Pair_Results = zeros(1,num_cols);
Z = zeros(num_rows,num_cols);

for surfcol = 1:num_cols
    Evaluation_Input_Pairs = Evaluation_Inputs_Means;
    Evaluation_Input_Pairs(1,:) = Varied_Input_Pairs(1,surfcol);
    for surfrow = 1:num_rows
        Evaluation_Input_Pairs(2,:) = Varied_Input_Pairs(2,surfrow);
        Input_Pair_Predictions = best_net(Evaluation_Input_Pairs);
```

```
        Input_Pair_Results_Raw(1,:) = Input_Pair_Predictions;
        Input_Pair_Results(1,:) = Input_Pair_Results_Raw(1,:).*(range_COP(2) - range_COP(1))
+ range_COP(1);
        Z(surfrow,surfcol) = Input_Pair_Results(1,surfcol);
    end
 end
 Corrected_Varied_Input_Pair(1,:) = Varied_Input_Pairs(1,:).*(Input_Pair_minmax_range(1,2) -
Input_Pair_minmax_range(1,1)) + Input_Pair_minmax_range(1,1);
 Corrected_Varied_Input_Pair(2,:) = Varied_Input_Pairs(2,:).*(Input_Pair_minmax_range(2,2) -
Input_Pair_minmax_range(2,1)) + Input_Pair_minmax_range(2,1);
 [X, Y] = meshgrid(Corrected_Varied_Input_Pair(1,:), Corrected_Varied_Input_Pair(2,:));

 fig = figure;
 surf(X,Y,Z);
 xlabel("Water temperature at evaporator inlet (°C)");
 ylabel("Water temperature ar evaporator outlet (°C)");
 zlabel("COP");
 figname = "Evap_Water_Temps_vs_COP_" + testID + ".png";
 saveas(fig,figname);

 % Evaluate evap water flow rates vs COP

 Varied_Input_Pairs = [Vary_Evap_Water_Flow; Vary_Cond_Water_Flow];
 Input_Pair_minmax_range  = [range_Evap_Water_Flow;range_Cond_Water_Flow];

 num_rows = 50;
 num_cols = 50;
 Input_Pair_Results = zeros(1,num_cols);
 Z = zeros(num_rows,num_cols);

 for surfcol = 1:num_cols
    Evaluation_Input_Pairs = Evaluation_Inputs_Means;
    Evaluation_Input_Pairs(3,:) = Varied_Input_Pairs(1,surfcol);
    for surfrow = 1:num_rows
        Evaluation_Input_Pairs(4,:) = Varied_Input_Pairs(2,surfrow);
        Input_Pair_Predictions = best_net(Evaluation_Input_Pairs);
        Input_Pair_Results_Raw(1,:) = Input_Pair_Predictions(1,:);
        Input_Pair_Results(1,:) = Input_Pair_Results_Raw(1,:).*(range_COP(2) - range_COP(1))
+ range_COP(1);
        Z(surfrow,surfcol) = Input_Pair_Results(1,surfcol);
    end
 end
 Corrected_Varied_Input_Pair(1,:) = Varied_Input_Pairs(1,:).*(Input_Pair_minmax_range(1,2) -
Input_Pair_minmax_range(1,1)) + Input_Pair_minmax_range(1,1);
 Corrected_Varied_Input_Pair(2,:) = Varied_Input_Pairs(2,:).*(Input_Pair_minmax_range(2,2) -
Input_Pair_minmax_range(2,1)) + Input_Pair_minmax_range(2,1);
 [X, Y] = meshgrid(Corrected_Varied_Input_Pair(1,:), Corrected_Varied_Input_Pair(2,:));
```

```
 fig = figure;
 surf(X,Y,Z);
 xlabel("Water flow rate through evaporator (L/s)");
 ylabel("Cooling water flow rate at condenser (L/s)");
 zlabel("COP");
 figname = "Evap_Cond_Water_Flow_vs_COP_" + testID + ".png";
 saveas(fig,figname);

 % Evaluate evap refrigerant temps vs COP

 Varied_Input_Pairs = [Vary_Evap_NH3_Inlet_Temp;Vary_Evap_NH3_Outlet_Temp];
 Input_Pair_minmax_range  = [range_Evap_NH3_Inlet_Temp;range_Evap_NH3_Outlet_Temp];

 num_rows = 50;
 num_cols = 50;
 Input_Pair_Results = zeros(1,num_cols);
 Z = zeros(num_rows,num_cols);

 for surfcol = 1:num_cols
     Evaluation_Input_Pairs = Evaluation_Inputs_Means;
     Evaluation_Input_Pairs(5,:) = Varied_Input_Pairs(1,surfcol);
     for surfrow = 1:num_rows
         Evaluation_Input_Pairs(6,:) = Varied_Input_Pairs(2,surfrow);
         Input_Pair_Predictions = best_net(Evaluation_Input_Pairs);
         Input_Pair_Results_Raw(1,:) = Input_Pair_Predictions;
         Input_Pair_Results(1,:) = Input_Pair_Results_Raw(1,:).*(range_COP(2) - range_COP(1))
+ range_COP(1);
         Z(surfrow,surfcol) = Input_Pair_Results(1,surfcol);
     end
 end
 Corrected_Varied_Input_Pair(1,:) = Varied_Input_Pairs(1,:).*(Input_Pair_minmax_range(1,2) -
Input_Pair_minmax_range(1,1)) + Input_Pair_minmax_range(1,1);
 Corrected_Varied_Input_Pair(2,:) = Varied_Input_Pairs(2,:).*(Input_Pair_minmax_range(2,2) -
Input_Pair_minmax_range(2,1)) + Input_Pair_minmax_range(2,1);
 [X, Y] = meshgrid(Corrected_Varied_Input_Pair(1,:), Corrected_Varied_Input_Pair(2,:));

 fig = figure;
 surf(X,Y,Z);
 xlabel("Refrigerant temperature at evaporator inlet (°C)");
 ylabel("Refrigerant temperature at evaporator outlet (°C)");
 zlabel("COP");
 figname = "Evap_NH3_Temps_vs_COP_" + testID + ".png";
 saveas(fig,figname);

 % Evaluate wet-bulb temperature and compressor power vs COP

 Varied_Input_Pairs = [Vary_WB_Temp; Vary_Comp_Power];
 Input_Pair_minmax_range  = [range_WB_Temp;range_Comp_Power];
```

```
 num_rows = 50;
 num_cols = 50;
 Input_Pair_Results = zeros(1,num_cols);
 Z = zeros(num_rows,num_cols);

 for surfcol = 1:num_cols
     Evaluation_Input_Pairs = Evaluation_Inputs_Means;
     Evaluation_Input_Pairs(7,:) = Varied_Input_Pairs(1,surfcol);
     for surfrow = 1:num_rows
         Evaluation_Input_Pairs(8,:) = Varied_Input_Pairs(2,surfrow);
         Input_Pair_Predictions = best_net(Evaluation_Input_Pairs);
         Input_Pair_Results_Raw(1,:) = Input_Pair_Predictions;
         Input_Pair_Results(1,:) = Input_Pair_Results_Raw(1,:).*(range_COP(2) - range_COP(1))
+ range_COP(1);
         Z(surfrow,surfcol) = Input_Pair_Results(1,surfcol);
     end
 end
 Corrected_Varied_Input_Pair(1,:) = Varied_Input_Pairs(1,:).*(Input_Pair_minmax_range(1,2) -
Input_Pair_minmax_range(1,1)) + Input_Pair_minmax_range(1,1);
 Corrected_Varied_Input_Pair(2,:) = Varied_Input_Pairs(2,:).*(Input_Pair_minmax_range(2,2) -
Input_Pair_minmax_range(2,1)) + Input_Pair_minmax_range(2,1);
 [X, Y] = meshgrid(Corrected_Varied_Input_Pair(1,:), Corrected_Varied_Input_Pair(2,:));

 fig = figure;
 surf(X,Y,Z);
 xlabel("Ambient wet-bulb temperature (°C)");
 ylabel("Compressor power demand (kW)");
 zlabel("COP");
 figname = "WB_Temp_Comp_Power_vs_COP_" + testID + ".png";
 saveas(fig,figname);

 end

 function Optimise_FP(best_net, eval_results, corrected_varied_eval_inputs, filtered_data,
best_R_2, testID)

 range_Evap_Inlet_Water_Temp = [min(filtered_data(:,1)) max(filtered_data(:,1))];
 range_Evap_Outlet_Water_Temp = [min(filtered_data(:,2)) max(filtered_data(:,2))];
 range_Evap_Water_Flow = [min(filtered_data(:,3)) max(filtered_data(:,3))];
 range_Cond_Water_Flow = [min(filtered_data(:,4)) max(filtered_data(:,4))];
 range_Evap_NH3_Inlet_Temp = [min(filtered_data(:,5)) max(filtered_data(:,5))];
 range_Evap_NH3_Outlet_Temp = [min(filtered_data(:,6)) max(filtered_data(:,6))];
 range_WB_Temp = [min(filtered_data(:,7)) max(filtered_data(:,7))];
 range_Comp_Power = [min(filtered_data(:,8)) max(filtered_data(:,8))];
 range_COP = [min(filtered_data(:,9)) max(filtered_data(:,9))];
```

```
 input_minmax_range = [range_Evap_Inlet_Water_Temp; range_Evap_Outlet_Water_Temp;
range_Evap_Water_Flow; range_Cond_Water_Flow; range_Evap_NH3_Inlet_Temp;
range_Evap_NH3_Outlet_Temp; range_WB_Temp; range_Comp_Power];

 % Performance Improvement - Baseline
 % Import the baseline profile
 Baseline_Operation = readtable('Baseline
Operation\Baseline_Operation.xlsx','Sheet',4,TextType='string',VariableNamingRule='preserve')
;
 % Convert the tables to a double array
 Baseline_Operation = table2array(Baseline_Operation(:,[1 2 3 4 5 6 7 8 9]));

 % Extract the baseline inputs and outputs from the double array
 Baseline_Inputs = Baseline_Operation(:,1:8);
 Baseline_Output = Baseline_Operation(:,9);

 % Define the start and end time for a full day in 30min intervals
 start_time = duration(0, 0, 0);
 end_time = duration(23, 30, 0);
 time_interval = minutes(30);
 time_axis = start_time : time_interval : end_time;

 % Compare ANN baseline predictions to actual baseline
 Baseline_Inputs_Transposed = Baseline_Inputs';
 Baseline_Inputs_Normalised = zeros(8,48);
 for num_inputs = 1:8
     for num_timestamps = 1:48
         Baseline_Inputs_Normalised(num_inputs,num_timestamps) =
(Baseline_Inputs_Transposed(num_inputs,num_timestamps)-
input_minmax_range(num_inputs,1))/(input_minmax_range(num_inputs,2)-
input_minmax_range(num_inputs,1));
     end
 end
 Baseline_predictions = best_net(Baseline_Inputs_Normalised);
 Baseline_predictions_corrected = Baseline_predictions*(range_COP(2) - range_COP(1)) +
range_COP(1);

 % Plot ANN baseline predictions to actual baseline
 fig = figure;
 plot(time_axis, Baseline_Output);
 hold on
 plot(time_axis, Baseline_predictions_corrected);
 xlabel('Time');
 ylabel('COP');
 legend("Baseline", "Predicted Baseline")
 hold off
 figname = "Baseline_vs_Predicted_Baseline_" + testID + ".png";
 saveas(fig,figname);
```

```
 % System Improvement - introducing control of inputs

 % Find input values that yield the best COP based on the input-output relationships
established by the ANN
 [Best_COP_per_input, Best_COP_per_input_index] = max(eval_results,[],2);

 % Use the index to get the input values yielding the best COPs
 Best_Input_Values = zeros(8,1);
 for input_singles = 1:8
     Best_Input_Values(input_singles,1) =
corrected_varied_eval_inputs(input_singles,Best_COP_per_input_index(input_singles,1));
 end

 Best_Evap_Inlet_Water_Temp = Best_Input_Values(1,1);
 disp("Best Evap Inlet Water Temp = " + Best_Evap_Inlet_Water_Temp);
 Best_Evap_Outlet_Water_Temp = Best_Input_Values(2,1);
 disp("Best Evap Outlet Water Temp = " + Best_Evap_Outlet_Water_Temp);
 Best_Evap_Water_Flow = Best_Input_Values(3,1);
 disp("Best Evap Water Flow = " + Best_Evap_Water_Flow);
 Best_Cond_Water_Flow = Best_Input_Values(4,1);
 disp("Best Cond Water Flow = " + Best_Cond_Water_Flow);
 Best_Evap_NH3_Inlet_Temp = Best_Input_Values(5,1);
 disp("Best NH3 Inlet Temp = " + Best_Evap_NH3_Inlet_Temp);
 Best_Evap_NH3_Outlet_Temp = Best_Input_Values(6,1);
 disp("Best NH3 Outlet Temp = " + Best_Evap_NH3_Outlet_Temp);
 Best_WB_Temp = Best_Input_Values(7,1);
 disp("Best Wet-bulb Temp = " + Best_WB_Temp);
 Best_Comp_Power = Best_Input_Values(8,1);
 disp("Best Compressor Power = " + Best_Comp_Power);

 % Find the theoretical best performance of the fridge plant
 Input_Ranges = [range_Evap_Inlet_Water_Temp; range_Evap_Outlet_Water_Temp;
range_Evap_Water_Flow; range_Cond_Water_Flow; range_Evap_NH3_Inlet_Temp;
range_Evap_NH3_Outlet_Temp; range_WB_Temp; range_Comp_Power;];
 Best_Input_Values_Normalised = zeros(8,1);
 for input_singles = 1:8
     Best_Input_Values_Normalised(input_singles,1) = (Best_Input_Values(input_singles,1)-
Input_Ranges(input_singles,1))/(Input_Ranges(input_singles,2) -
Input_Ranges(input_singles,1));
 end

 % Predict the fridge plant performance under ideal operational conditions
 Best_Theoretical_Performance = best_net(Best_Input_Values_Normalised);
 % De-normalise to correct the network predictions
 Best_Theoretical_Performance_Corrected = Best_Theoretical_Performance*(range_COP(2) -
range_COP(1)) + range_COP(1);
 disp("Maximum COP theoretically achievable = " + Best_Theoretical_Performance_Corrected)
```

```
 % System Improvement - practically achievable through control

 % Normalise baseline inputs based on test subset values
 Baseline_Inputs_Transposed = Baseline_Inputs';
 Baseline_Inputs_Normalised = zeros(8,48);
 for num_inputs = 1:8
     for num_timestamps = 1:48
         Baseline_Inputs_Normalised(num_inputs,num_timestamps) =
(Baseline_Inputs_Transposed(num_inputs,num_timestamps)-
input_minmax_range(num_inputs,1))/(input_minmax_range(num_inputs,2)-
input_minmax_range(num_inputs,1));
     end
 end

 % Create the practically achievable best baseline performance profile
 Best_Practical_Inputs_Normalised = Baseline_Inputs_Normalised;
 Best_Practical_Inputs_Normalised(3,:) = Best_Input_Values_Normalised(3,:);
 Best_Practical_Inputs_Normalised(4,:) = Best_Input_Values_Normalised(4,:);
 Best_Practical_Inputs_Normalised(8,:) = Best_Input_Values_Normalised(8,:);

 % Predict frige plant performance under best practical inputs
 Best_Practical_Performance = best_net(Best_Practical_Inputs_Normalised);
 % De-normalise to correct the network prediction
 Best_Practical_Performance_Corrected = Best_Practical_Performance*(range_COP(2) -
range_COP(1)) + range_COP(1);

 % Plot the improved baseline performance profile
 Error_Bars = zeros(1,48);
 for predictions = 1:48
     Error_Bars(predictions) = (1-
best_R_2)*Best_Practical_Performance_Corrected(predictions);
 end

 fig = figure;
 plot(time_axis, Baseline_Output,'r');
 hold on
 errorbar(time_axis,Best_Practical_Performance_Corrected,Error_Bars,'b');
 xlabel('Time');
 ylabel('COP');
 legend("Baseline COP", "Best COP");
 hold off
 figname = "Baseline_vs_Improved_Baseline" + testID + ".png";
 saveas(fig,figname);

 end

 function Verify_Model(best_net, verification_class, range_COP, testID)
```

```
 % Model verification
 Verification_Inputs = verification_class(:,1:8)';
 Verification_Outputs = verification_class(:,9)';

 Verification_predictions = best_net(Verification_Inputs);
 Verification_predictions_corrected = Verification_predictions*(range_COP(2) - range_COP(1))
+ range_COP(1);
 Verification_outputs_corrected = Verification_Outputs*(range_COP(2) - range_COP(1)) +
range_COP(1);

 fig = figure;
 plotregression(Verification_predictions_corrected,Verification_outputs_corrected);
 figname = "Verification_Regression_" + testID + ".png";
 saveas(fig,figname);

 % plot predicted outputs vs targets of verification subset
 fig = figure;
 plot(1:100,Verification_predictions_corrected(1:100), '-o','Color', 'k','LineWidth', 1.5);
 hold on
 plot(1:100,Verification_outputs_corrected(1:100),'-r');
 xlabel("Datapoints");
 ylabel("COP");
 legend("Predicted", "Targets");
 hold off
 figname = "Verification_Predictions_vs_Targets" + testID + ".png";
 saveas(fig,figname);


 MSE_Verification = mse(Verification_predictions_corrected, Verification_outputs_corrected);
 RMSE_Verification = sqrt(MSE_Verification);
 R_2_Verification =
regression(Verification_outputs_corrected,Verification_predictions_corrected);
 MAPE_Verification =
mape(Verification_predictions_corrected,Verification_outputs_corrected,"omitzero");

 disp("RMSE = " + RMSE_Verification);
 disp("R_2 = " + R_2_Verification);
 disp("MAPE = " + MAPE_Verification);

 end
```