Prediction model for the performance of a methane-fuelled spark-ignition engine at a deep-level mine

J Pretorius

 **orcid.org/0000-0002-0855-6113**

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

Supervisor:	Dr FG Jansen van Rensburg
Co-supervisor:	Dr JH Marais

Graduation:	July 2022
Student number:	27515796

ABSTRACT

Title:	Prediction model for the performance of a methane-fuelled spark-ignition engine at a deep-level mine
Author:	Julian Pretorius
Supervisor:	Dr FG Jansen van Rensburg
Co-supervisor:	Dr JH Marais
Keywords:	Prediction model, engine performance, deep-level mine, spark-ignition engine, artificial neural network

Electrical energy supply in South Africa is unreliable and the cost of electricity has been increasing significantly annually. This negatively impacts high energy consumers, especially deep-level mines. Therefore, reducing energy consumption and costs has become increasingly essential to the life-of-mine of deep-level mines.

Deep-level mines emit gases, primarily methane, from deep-seated underground geological sources that are intersected during normal mining activity. Specific deep-level mines have implemented spark-ignition (SI) engines to generate electricity using the emitted gases. This has the benefit of reducing the mine's energy costs and reliance on the unstable power grid.

Currently, no model exists that relates the operational parameters of a methane-fuelled SI engine at a deep-level mine to the power generated by the engine. Such a model would prove useful to perform economic optimisation on the system, especially as the amount of power generated by the engine is highly dependent on the varying operational parameters.

From the different modelling approaches, an artificial neural network (ANN) model was selected as most appropriate to model the SI engine given the available data. ANNs learn the relationship between input and output data sets and are capable of quantifying complex and non-linear functions.

The aim of this study is to develop a model that predicts the power generated by a methane-fuelled SI engine – using its operational parameters as inputs – by implementing ANNs. A case

study of a methane-fuelled SI engine at a deep-level mine was selected and investigated. The SI engine generates an average of 280 MWh per month.

The raw data from the case study was filtered and processed. For the model's development, the best network architecture was determined to comprise two hidden layers, with 14 hidden neurons in the first layer and 11 hidden neurons in the second layer. The Levenberg Marquardt (LM) algorithm attained the lowest prediction error and converged to a solution faster than the Scaled Conjugate Gradient (SCG) algorithm.

The developed model was implemented and its prediction accuracy evaluated. The coefficient of determination (R^2) for the training, validation and test subsets achieved 0.9865, 0.9850 and 0.9803, respectively. The overall performance of the data set attained 0.9855. From the results, it can be concluded that the model adequately and successfully predicted the power generated by the engine, achieving a high degree of accuracy.

Mathematical equations were developed for the best performing network to evaluate and investigate the relationships between the operational parameters and power generated. From the evaluation of the mathematical equations, it was determined that the best engine performance is attained at the highest feed flow rate and methane concentration. It was further determined that lower coolant temperatures and higher coolant pressure favour the engine with respect to the amount of power it generates.

From the evaluation, the best operational parameters were implemented using the developed model to investigate to what extent the power generated by the engine can be improved. According to the model, it was found that an additional 5.9 MWh can be generated per day by operating at higher feed flow rates, lower coolant temperatures and higher coolant pressures. This equates to an annual energy cost saving of R 2 860 000.

Lastly, the best performing model was implemented on an independent data set to test its robustness and accuracy on new unseen data for verification purposes. The model performed accurately on the verification data set and attained a R^2 value of 0.9873. This reaffirms that the model has been implemented correctly, is robust and can accurately predict the power generated by the methane-fuelled SI engine.

ACKNOWLEDGEMENTS

As the author, I would like to express my sincere gratitude to the following contributors that were critical to the success of this study:

- ETA Operations (Pty) Ltd for funding the project and supplying the data to complete this study.
- Dr Franco Jansen van Rensburg and Dr Johan Marais, my supervisors, for their inputs and guidance in completing this dissertation.
- Dr Jean van Laar for his assistance with this study.
- My parents, Chris and Zelda Pretorius, for their continuous support and encouragement.
- To the Heavenly Father for giving me the talents and knowledge to complete this study.

All information portrayed in this dissertation was done acknowledging sources and referencing published work.

TABLE OF CONTENTS

NOMENCLATURE

Symbol	Unit	Description
β	-	Steepest descent gradient factor
$\mathcal{E}_T$	-	Error function
η	-	Neural network
λ_k	-	Scaling factor
μ	-	Training control parameter
σ_k	-	Scaling factor
τ_F	-	Time period constant
b_i	-	Network bias
F_F	m^3/hr	Feed flow rate to flare
F_G	m^3/hr	Feed flow rate to generator
g	-	Gradient descent matrix
g_0	-	Steepest descent gradient direction
H	-	Hessian matrix
J	-	Jacobian matrix
k	-	Sampling instant
p_0	-	Search direction
P_C	kPa	Engine coolant pressure
R^2	-	Coefficient of determination
T_C	°C	Engine coolant temperature
T_{ex}	°C	Exhaust gas temperature
u_i	-	Pre-activation value
W_G	kW	Power generated by engine
w_j	-	Network weight
x_j	-	Network input value
X_m	%	Feed methane concentration
X_O	%	Feed oxygen concentration
y_i	-	Network output (prediction)

ABBREVIATIONS

AFR	Air-fuel ratio
ANN	Artificial neural network
BFGS	Broyden-Fletcher-Goldfard-Shanno
BP	Backpropagation
BSFC	Brake specific fuel consumption
CFPS	Coal-fired power station
CG	Conjugate Gradient
EGT	Exhaust gas temperature
EWMA	Exponentially weighted moving average
HC	Unburned hydrocarbon
HL	Hidden layer
HN	Hidden neuron
LM	Levenberg Marquardt
LOM	Life-of-mine
MAPE	Mean absolute percentage error
MSE	Mean square error
NS	Noise-spike
PE	Processing element
ReLu	Rectified Linear Unit
RMSE	Root mean square error
RP	Resilient Backpropagation
SCG	Scaled Conjugate Gradient
SI	Spark-ignition
TOU	Time-of-use

Chapter 1: Introduction

Figure 1: Typical South African mine[1]

"When we are no longer able to change a situation, we are challenged to change ourselves."

— Viktor E. Frankl, Man's Search for Meaning

[1] J Pretorius, Personal photograph, Welkom, 2020

1. INTRODUCTION

1.1. Preamble

The background and motivation for this study are presented, followed by an introduction into artificial neural networks (ANNs). An account of the literature review of similar studies, where the contribution and shortcomings of each study were analysed, is given. A state-of-the-art table is compiled and the need for the study identified. The problem statement, aim and objectives of this study are discussed. The chapter concludes with the outline of this document.

1.2. Background

Energy is an essential part of civilisation. With the exponential population growth experienced during the 21st century, the supply of energy to match the energy demand has become increasingly cardinal to the industrial success of both developing and developed countries. Technology has improved the standard of living and increasingly automated industries, especially in developing countries, resulting in an increase in energy demand [1].

1.2.1. Electrical energy in South Africa

The power utility company, Eskom, dominates South Africa's supply of electrical energy, generating approximately 95 % of the county's total electrical energy supply [2]. The demand for electricity in South Africa exceeded Eskom's supply capabilities, resulting in the implementation of load shedding and load curtailment numerous times since 2008 to reduce the country's energy demand [3]. The supply of electrical energy in South Africa remains unreliable and prone to outages for various reasons, including mismanagement, lack of planning and inadequate maintenance at its power stations [4].

In addition to Eskom's unreliable supply, its electricity tariffs are of the highest in the world [5]. Eskom has accumulated substantial debt due to high operating expenditure and outstanding revenues [6]. To continue supplying electricity, Eskom has been significantly increasing its electricity tariffs annually. Eskom's electricity tariffs have increased by 753 %,

versus an inflation increase of 134 % from 2007 to 2021, as demonstrated by Figure 2 [7]. The unreliable supply of power by the utility company as well as the high tariffs severely impact the industrial and mining sectors that form the basis of South Africa's economy [8].

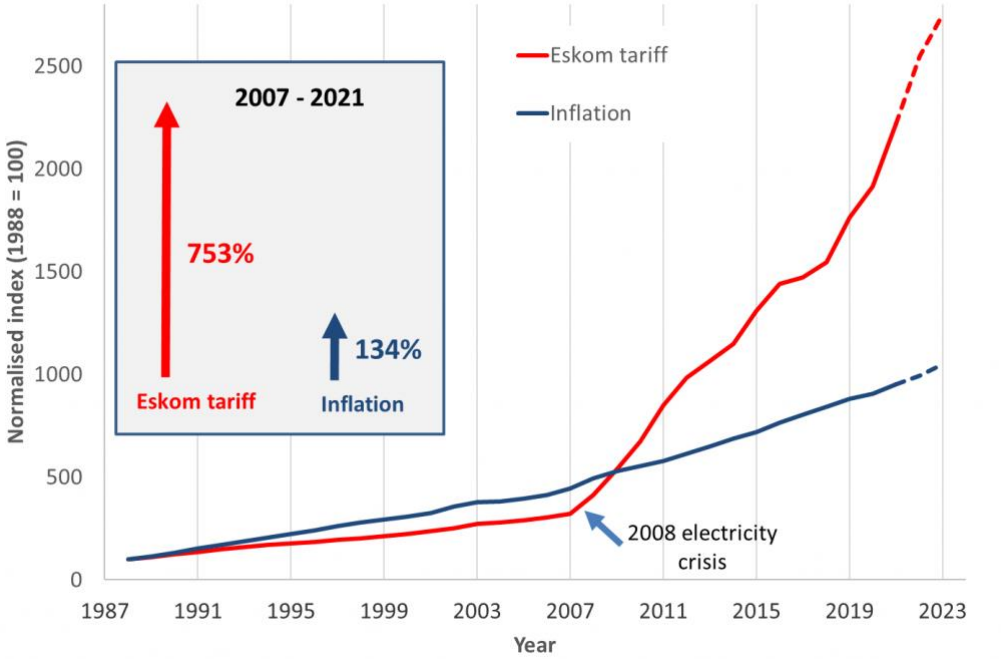


Figure 2: Comparison between the historical and projected normalised electricity tariffs and inflation from 1987 to 2023 [7]

From Eskom's annual integrated report [9], 91 % of the power it generated in 2021 was supplied by conventional coal-fired power stations (CFPS). By burning coal, not only large amounts of carbon dioxide is released, but also sulphur dioxide, nitrous oxides and large particulate matter. These have a severe negative impact on the environment and human health. These emissions contribute to global warming, that has a further detrimental environmental, social and economic impact [10].

1.2.2. Deep-level mines

South Africa has some of the deepest underground gold and platinum mines in the world. Mining is an essential and important sector of the South African economy. Mines are high energy consumers, especially deep-level mines, consuming 14 % of the total electricity generated in South Africa, as given by Figure 3 [9].

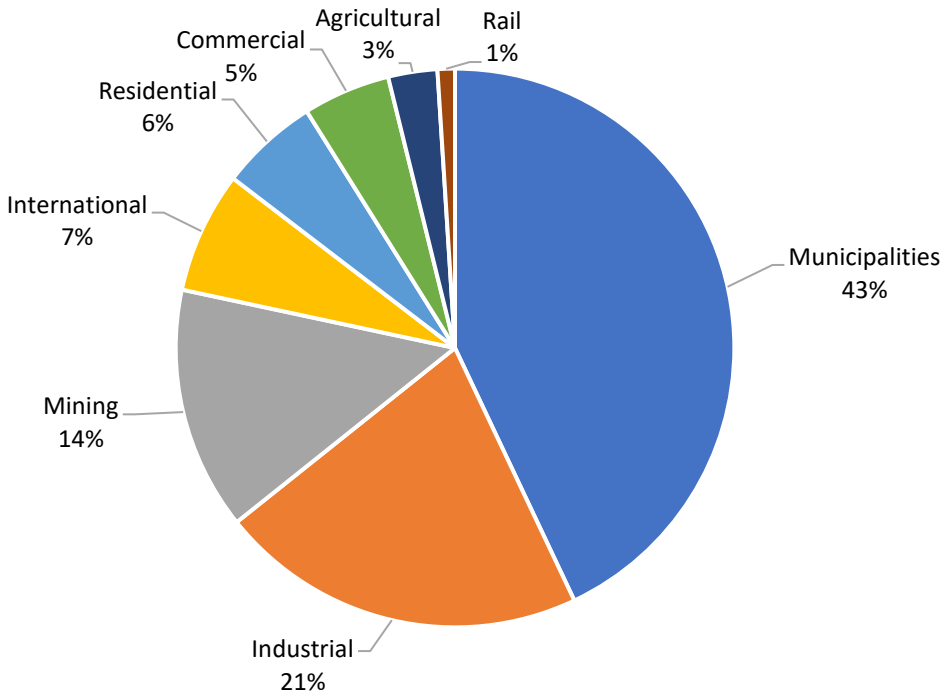


Figure 3: Consumption of energy by different sectors in South Africa [9]

Mines are, however, experiencing a considerable decrease in profitability for a variety of reasons [11]. Energy expenditure accounts for a significant portion of a mine's monthly expenditure. It has therefore become increasingly important for deep-level mines to reduce their energy costs to favour their life-of-mine (LOM). Energy-related projects to decrease their energy consumption are becoming essential to the LOM of deep-level mines [12].

1.2.3. Gas emissions from mining activity

Some deep-level mines emit gases, primarily methane, from deep-seated underground geological sources that are intersected during normal mining activity. These geological sources include faults, dykes and fissures [13]. An analysis of the gases emitted from such a geological source is given by Figure 4. The gases are released into the mining atmosphere and ventilated to the surface. The gases are also released from old exploratory and geological boreholes [14].

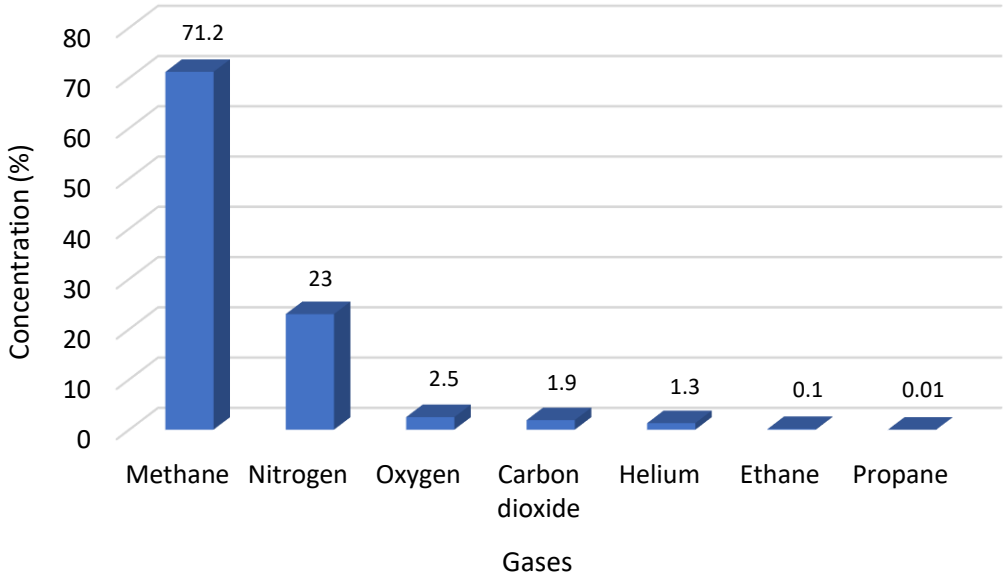


Figure 4: Analysis of the gases emitted from an underground geological source at a deep-level mine

Methane is a colourless, odourless and potent greenhouse gas. Methane is a serious safety risk to underground mines, as it is highly explosive [15]. Methane further poses serious health risks, as it displaces oxygen that can result in suffocation and lead to coma or death [16]. Furthermore, methane is a potent greenhouse gas that contributes to climate change and global warming at a rate 23 times greater than carbon dioxide [17].

1.2.4. Generating electricity from gas emissions

The case study deep-level mine has implemented a spark-ignition (SI) engine to generate electricity using the emitted methane and other combustible gases. This has the benefit of reducing the mine's energy costs, reliance on the unstable Eskom power grid and the mine's environmental impact as less energy is used that is primarily generated by CFPS [14]. Furthermore, because the mine decreases their use of CFPS, they can attain carbon credits, which are reductions to the carbon tax they pay [18].

By flaring or using the emitted gas to generate electricity, the mine reduces its carbon footprint by converting methane to primarily carbon dioxide [13]. The utilisation of methane to generate electricity at this case study mine is unique among deep-level mines in South Africa.

1.2.5. Spark-ignition engines

The SI engines implemented at deep-level mines are generally stand-alone installations that convert the chemical energy of the primarily methane fuel into mechanical energy to generate electricity [14]. An SI engine implements spark-plugs, differentiating it from other internal combustion engines, to initiate the combustion process. The spark-plug comprises two electrodes that discharges a high-voltage electrical spark to ignite the air-fuel mixture present in the combustion chamber [19].

The engine has many operational parameters, such as feed flow rate, fuel, temperature and pressures. These operational parameters influence the performance of the engine [20]. For the purposes of this study, the performance of SI engines is quantified by the power that it generates.

1.2.6. Need for model

By optimising the methane-fuelled SI engine's performance at a deep-level mine, more power can be generated, resulting in greater energy savings, carbon credits and less negative environmental impact. There is thus a need to optimise the engine. However, to optimise the engine, a model that characterises the relationship between the input (operational parameters) and output (power generated) variables must be developed. Currently, no model that relates the operational parameters and performance of a methane-fuelled SI engine at a deep-level mine exists.

1.2.7. Modelling approaches

Different methods can be considered to develop a model for the methane-fuelled SI engine. A first-principles model that implements fundamental principles and derivations to develop mathematical equations would be effective if the engine was new and operated at its manufactured efficiency [21]. However, the specific case study engine has been operating for more than 12 years and has accrued declining efficiencies from its continual operation. A first-principles approach would therefore prove tedious, require multiple operational assumptions and additional investigations to develop a model that might not attain satisfactory accuracy and robustness.

Modelling the engine is a complex problem. Nonetheless, sufficient operational and performance-related data for the case study is available. A data-driven approach would thus prove applicable in modelling the engine. However, basic data-based statistical methods would be inadequate due to the complexity of the system. Henceforth, an ANN-based model is considered. An ANN is a statistical tool that is capable of quantifying complex and non-linear relationships between the variables of a given data set representing the system [22].

1.3. Artificial neural networks

An ANN *learns* the relationships between the variables of a system and is developed by first *training* the network on input and target data sets and is thereafter implemented to predict the outputs for given input data [23]. ANNs can be developed for any system and are thus considered to be universal function approximators, provided sufficient data is available. The variables of the data set should be selected to represent the interactions between the input and output variables [24].

To understand the functioning of ANNs, their architecture and corresponding fundamental equations are presented in this section. The process and algorithms whereby ANNs are trained are examined. Thereafter, the architecture and training rules of thumb relevant to the development and implementation of the ANN model for this study are discussed.

1.3.1. Architecture

The purpose of discussing the perceptron first is to introduce the architecture and fundamental equations of an ANN, which form the basis for understanding multi-layered networks and their connection formulae that differentiate the various types of networks.

Perceptron

A perceptron is the simplest form of a neural network and comprises only one neuron, also known as a node or processing element (PE) [25]. A perceptron consists of one neuron, weighted input(s), a bias (b_i) and an output (y_i). Each input (x_j) has a corresponding weight (w_{ij}) that characterises the strength of its connection to the neuron [26]. Two computations

take place at the neuron, namely summation ($\sum$) and activation (f). The basic structure of a perceptron is given by Figure 5 [27].

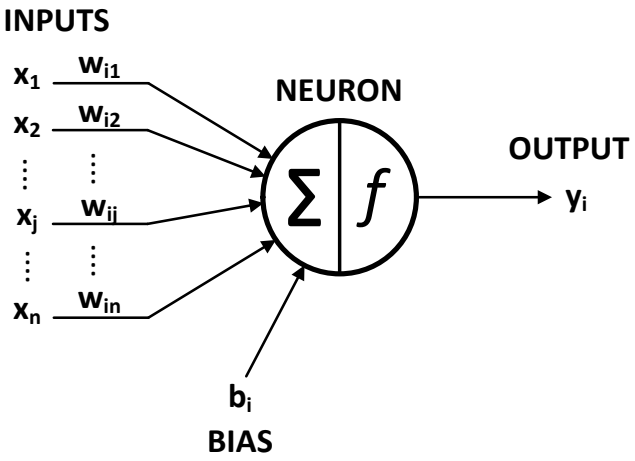


Figure 5: Basic structure of a perceptron (adapted from [27])

Each input is multiplied by its weight and summed to obtain the pre-activation value (u_i). An additional value, the bias (b_i), is added to the pre-activation value to account for data that is not mean-centred [28]. The bias is a non-zero constant, initialised with a random value at the start of the training process [29]. The mathematical equation representing the summation function is given by Equation 1 [30]. The basic structure of a perceptron with its pre-activation value is given by Figure 6.

$$u_i = \sum_{j=1}^{d} w_{ij} x_j + b_i \qquad 1$$

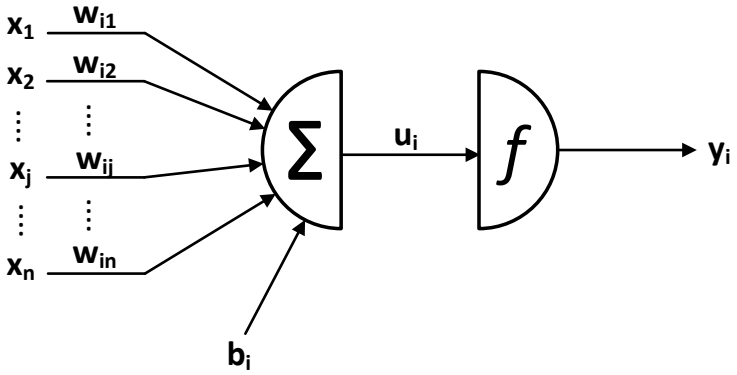


Figure 6: Basic structure of a perceptron indicating the pre-activation value (u_i) (adapted from [28])

After summation, an activation or transfer function (f) is implemented to transform the pre-activation value (u_i) into the post-activation or output (y_i) value [28]. The purpose of the transfer function is to re-scale the data to account for non-linearity in the model, hereby ensuring the perceptron is more powerful than linear transformation functions [31]. The mathematical equation representing the activation function is given by Equation 2. A representation of the mathematical equations of summation and activation for a perceptron is given by Figure 7 [32].

$$y_i = f(u_i) \qquad 2$$

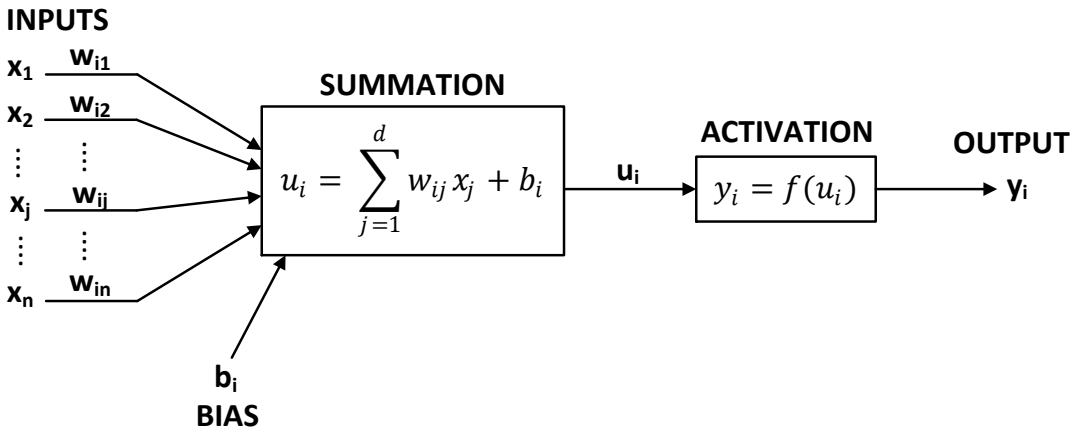


Figure 7: Representation of the mathematical functions of a perceptron (adapted from [32])

The type of transfer function selected for a particular neural network depends on the network type, output and application. Classical non-linear activation functions are the sign, logistic sigmoid and tangent sigmoid functions [33]. In recent years, piecewise linear activation functions, such as the Rectified Linear Unit (ReLU) and hard tangent sigmoid functions, have become more popular [34]. The equations, type and rescaled ranges for the most-employed transfer functions are listed in Table 1. Graphical representations of these transfer functions are given by Figure 8 [28].

Table 1: Equations of commonly used transfer functions (adapted from [28])

Function	Type	Range	Equation	Graph	No.
Identity	Linear	-	$f(u) = u$	a	3
Sign	Non-linear	$[-1, 1]$	$f(u) = sign(u)$	b	4
Logistic Sigmoid	Non-linear	[0, 1]	$f(u) = \frac{1}{1 + e^{-u}}$	c	5
Tangent Sigmoid (tanh)	Non-linear	$[-1, 1]$	$f(u) = \frac{2}{1 + e^{-2u}} - 1$	d	6
ReLU (Rectified Linear Unit)	Linear	[0, 1]	$f(u) = \max\{0, \mathrm{u}\}$	e	7
Hard Tangent Sigmoid (tanh)	Linear	$[-1, 1]$	$f(u) = \max\{-1, \min[1, u]\}$	f	8

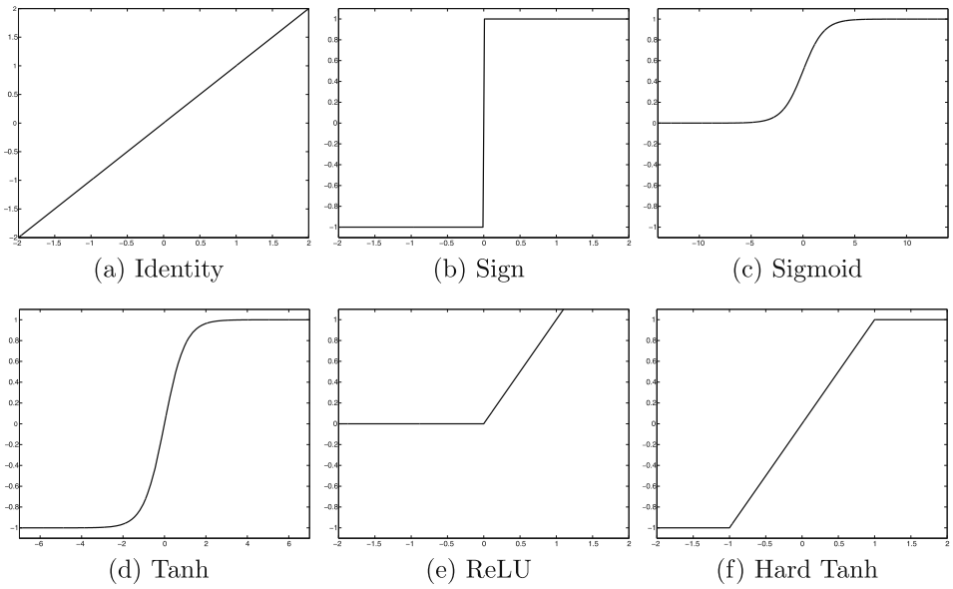


Figure 8: Graphical representations of commonly used transfer functions: (a) identity, (b) sign, (c) logistic sigmoid, (d) tangent sigmoid, (e) ReLu and (f) hard tangent sigmoid [28]

The full input-output equation for a neural network is obtained by substituting Equation 1 into Equation 2. The full input-output equations for the logistic sigmoid and tangent sigmoid transfer are listed in Table 2 [35].

Table 2: Full input-output equations for the logistic and tangent sigmoid transfer functions

Function	Equation	No.
Logistic sigmoid	$y_i = \frac{1}{1 + exp(-\sum_{j=1}^{n} w_{ij} x_j + b_i)}$	9
Tangent sigmoid	$y_i = \frac{2}{1 + \exp\left(-\sum_{j=1}^{n} w_{ij} x_j + b_i\right)} - 1$	10

Multilayer neural network

A multilayer neural network comprises multiple perceptrons (neurons), linked with one another and arranged in three layers, namely input, intermediate and output layers [36]. There is only one input and one output layer, conversely, there may be more than one intermediate layer. The intermediate layer is called the hidden layer, as its computed values cannot be observed directly by the user, unlike that of the input and output layers [37]. The basic architecture of a multilayer neural network, indicating the layers, weights and biases, is given by Figure 9 [28].

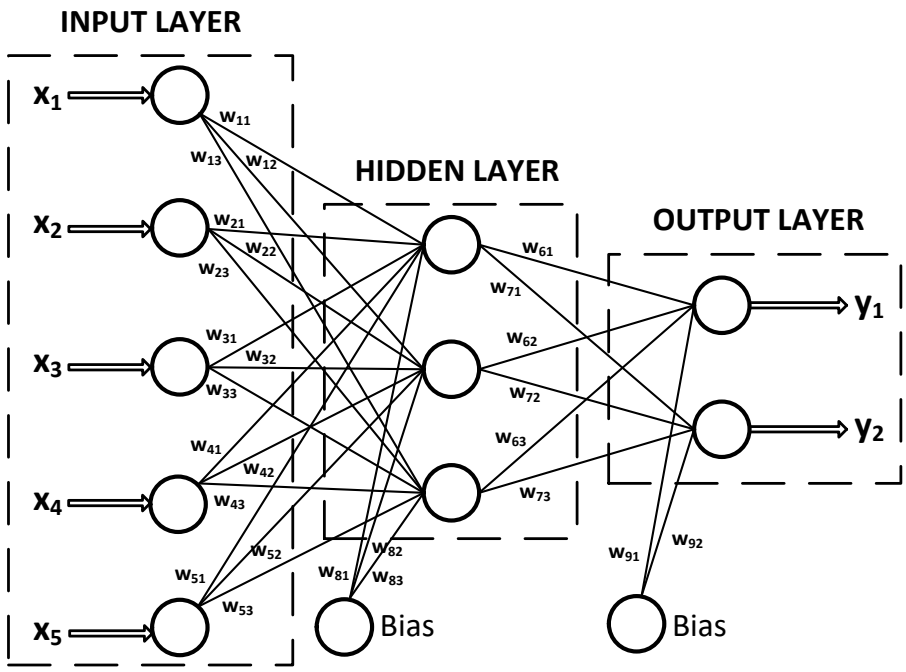


Figure 9: Basic architecture of a multilayer neural network (adapted from [28])

Neurons in the input layer are only responsible for transferring (distributing) the input variables' data to the subsequent hidden layer [33]. Both the neurons in the hidden and output layers are responsible for the computations (processing) discussed in the *Perceptron* section. The architecture of a multilayer neural network that indicates the functions of the different neurons in a multilayer network is given by Figure 10 [28].

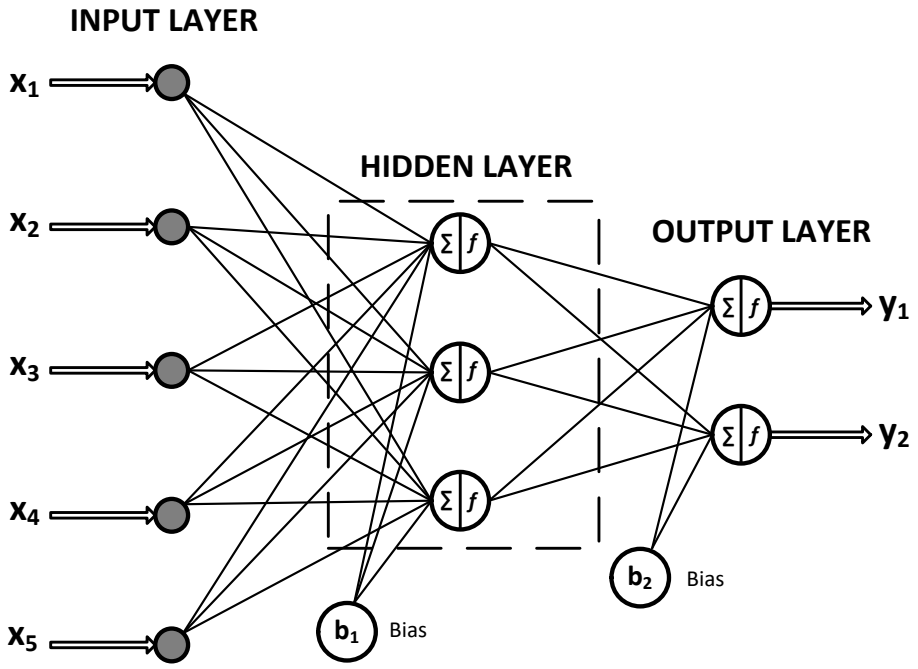


Figure 10: Architecture of a multilayer neural network, indicating the functions of the different neurons (adapted from [28])

Connection formula

The connection formula of a network specifies the configuration of the network, such as the number of inputs, outputs, hidden neurons and hidden layers, as well as the direction of neurons' signals. Multiple different network configurations exist, for example feedforward neural networks, recurrent (feedback) neural networks and autoencoders [38].

The type of networks illustrated and discussed thus far are feedforward neural networks. The layers of a feedforward neural network feeds to the succeeding layer only, in the direction of input to output layers. Recurrent neural networks are time dependent. These networks feed

data both forwards and simultaneously allow, for each hidden neuron, the data of the previous time stamp to directly interact with the computation of the current time stamp [39].

1.3.2. Training

The aim of training is to develop a neural network η with weights $\overline{w}$ whose output y_i (prediction) is sufficiently *close* enough to the desired output t_i (target) for the given set of input variables x_i, as represented by Equation 11 [31].

$$y_i = \eta(\bar{x}_i, \overline{w}) \qquad 11$$

Independent sample testing

Before commencing with training, the data is first randomly sampled (allocated) into three independent data subsets, namely training, validation and testing. This process is called independent sample testing and is schematically represented by Figure 11 [33].

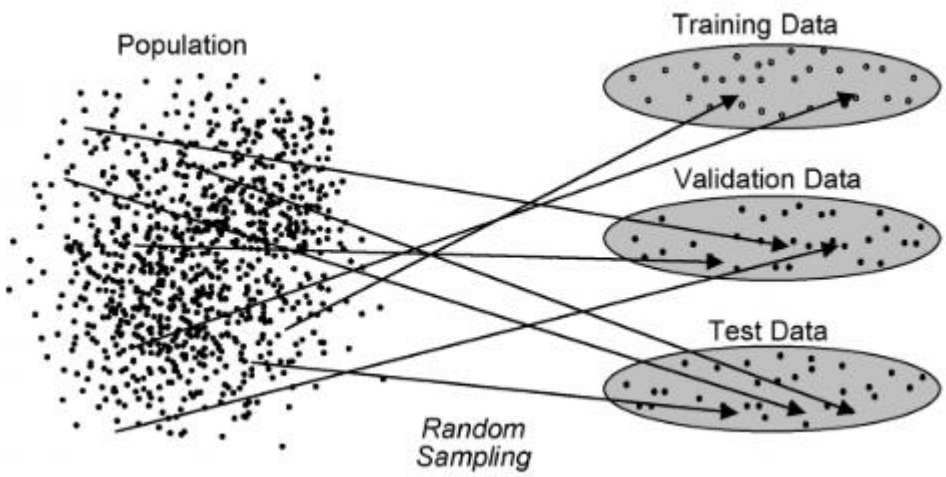


Figure 11: Independent sample testing of a population randomly divided into three groups: training, validation and testing [33]

Training subset

The training subset is used during the training process to adjust the weights of the hidden and output neurons [33].

Validation subset

The validation subset is used to find the best network configuration to prevent underfitting and overfitting by determining the optimum number of epochs (training iterations). The validation subset's prediction error can also be used to determine the optimal number of hidden neurons in the network [40].

Testing subset

The testing subset is an unbiased set used to quantify the generalisation ability of the trained network [33]. The testing subset should not be used to differentiate between networks with different configurations [41].

Error function

The closeness of the model's prediction (y_i) to the target (t_i) is quantified by the error function $\mathcal{E}_T$. The function $\mathcal{E}_T$ is defined as the sum-of-squares error in the form [27]:

$$\mathcal{E}_T = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2 \qquad 12$$

where the error e_i is defined as:

$$e_i = (y_i - t_i) \qquad 13$$

Therefore, the error function becomes:

$$\mathcal{E}_T = \frac{1}{2}\sum_{i=1}^{n} e_i^2 \qquad 14$$

The error vector $\bar{e}$ is of dimension n x 1, as defined by:

$$\bar{e} = [e_i] \qquad 15$$

The error $\mathcal{E}_T$ is a function of $\overline{w}$, as the network's output y_i is dependent on the weights $\overline{w}$ of the net η. Therefore:

$$\mathcal{E}_T = \mathcal{E}_T(\overline{w}) \quad (16)$$

The objective of training is to minimise $\mathcal{E}_T(\overline{w})$ by adjusting $\overline{w} \in \mathcal{W} \subset \mathbb{R}^p$. The weights space is represented by $\mathcal{W}$ [27].

Training algorithms

Many different training algorithms exist to adjust a network's weights and minimise the error between its predictions (outputs) and targets. The performance of a training algorithm is dependent on primarily three factors. Firstly, the initial weights of the network are randomly generated at the initialisation of each training instance [42]. Secondly, the weights will exhibit different iterative adjustments and error convergence rates as the algorithms' minimisation equations differ [43]. Thirdly, an algorithm's convergence rate tends to stall at local minima errors due to the inherent characteristics of the algorithm, hence algorithms approach but do not necessarily attain the global minimum error for a specific training instance [44]. The combination of these three phenomena will ensure that different algorithms and training instances result in different final weights and consequently different performance errors.

The training algorithms applicable to this study are discussed in this section:

Backpropagation algorithm

One of the most widely used training algorithms for feedforward multilayer neural networks is the backpropagation (BP) algorithm [45]. During training, the inputs to the neurons (data) are propagated forwards to each subsequent computational neuron, whilst the errors between the network's outputs and targets are propagated backward through the network, as the network's weights are adjusted [46].

The BP algorithm trains the network by executing two steps, iteratively and in sequence, to adjust the weights of the neural network. Firstly, the derivative of the error function $\mathcal{E}_T$ with respect to the network's weights $\overline{w}$ is evaluated. Secondly, the adjustments to be made to the weights are computed using the evaluated derivative [47].

To compute the derivative, the activation and error functions must be differentiable. For this reason, the sum-of-squares error is selected as the error function ($\mathcal{E}_T$). The derivative can be

calculated using methods such as steepest descent (as for the BP algorithm) or more powerful algorithms, such as the Levenberg Marquardt (LM), Conjugate Gradient (CG) and Scaled Conjugate Gradient (SCG) algorithms [27].

Levenberg-Marquardt algorithm

The Levenberg-Marquardt (LM) algorithm – by Levenberg [48] and Marquardt [49] – is a non-linear optimisation algorithm commonly used to train feedforward networks. The algorithm is a compromise between the steepest gradient descent process (first order derivatives as used in backpropagation) and the Quasi-Newton method (second order derivatives), of which the extent of each is determined by a control parameter (μ) [50].

The algorithm makes the assumption that the underlying function to be modelled is linear, such that the minimum error can be computed in one iteration. The algorithm then calculates the error and subsequent weight adjustment for this iteration. If the error decreases, the weights adjustment is accepted and the control parameter is decreased to reinforce the linear assumption. If the error increases, the weight adjustments are rejected and the control parameter is increased to reduce the linear assumption [33].

By reinforcing the linear assumption, the LM algorithm favours the Quasi-Newton computation that is able to rapidly find the minimum error. In contrast, by reducing the linear assumption, the LM algorithm incorporates gradient descent, allowing it to also converge to a minimum error. The process of adjusting the weights is repeated until the maximum number of iterations or the desired error is attained [33].

The LM algorithm and other second order gradient techniques use the Hessian to adapt the weights in the optimal direction. The Hessian ($\mathbf{H}$) comprises the second-order partial derivative of the error function ($\mathcal{E}_T$) with respect to the weights (w), as given by:

$$\mathbf{H} = [H_{ij}] \tag{17}$$

$$H_{ij} = \frac{\partial^2 \mathcal{E}_T}{\partial w_i \partial w_j} \tag{18}$$

To simplify the derivative calculation, the Hessian is approximated by multiplying the Jacobian matrix with its transpose [51], according to:

$$\mathbf{H} \approx \mathbf{J}^T\mathbf{J} \quad (19)$$

The Jacobian ($\mathbf{J}$) is defined as the first-order partial derivatives of the error function (e_j) with respect to the network's weights, as follows:

$$\mathbf{J} = [\mathrm{J}_{ij}] \quad (20)$$

$$J_{ij} = \frac{\partial e_j}{\partial w_i} \quad (21)$$

The gradient ($\mathbf{g}$) is calculated as the product between the transposed Jacobian (of first-order partial derivatives) and the error vector $\bar{e}$ [52], as given by:

$$\mathbf{g} = \mathbf{J}^T\bar{e} \quad (22)$$

The weight update formula for each training iteration k, is given by:

$$\begin{aligned} \bar{w}_{k+1} &= \bar{w}_k - (\mathbf{H} + \mu\,\mathbf{I})^{-1}\mathbf{g} \\ &= \bar{w}_k - (\mathbf{J}^T\mathbf{J} + \mu\,\mathbf{I})^{-1}\mathbf{J}^T\bar{e} \end{aligned} \quad (23)$$

where μ is the control parameter and $\mathbf{I}$ the identity matrix [53].

As can be seen from Equation 23, if the control parameter approaches zero, the Quasi-Newton method, with an approximated Hessian, is dominant. Conversely, the larger the value of the control parameter, the more the gradient-descent method becomes dominant [33], [54]. The procedure for the LM algorithm is given by Table 14 in *Appendix A1: LM algorithm* (page 95).

Scaled Conjugate Gradient algorithm

The steepest descent direction is used by the BR algorithm to adjust the weights of the network, as mentioned previously. The steepest descent is the most negative gradient direction, whereby the error function decreases most rapidly. Nonetheless, the steepest descent does not necessarily result in the fastest convergence to the global minimum error [55].

The Conjugate Gradient (CG) algorithm searches the error function's surface to determine the direction that converges faster than the steepest descent direction, namely the conjugate gradient direction [56]. The error that has been minimised in the previous steps remains minimised during the conjugate gradient direction search. The direction along the line of the conjugate gradient is searched to determine the step size [57].

The line search technique approximates the step size α_k (at each iteration k) and hereby prevents the need of calculating the Hessian matrix. The step size represents the optimal distance or magnitude the previous iteration's weights $\bar{w}_k$ need to be adjusted – along the search direction $\bar{p}_k$ – to compute the current iteration's weights $\bar{w}_{k+1}$. This is mathematically represented by the equation [58]:

$$\bar{w}_{k+1} = \bar{w}_k + \alpha_k \bar{p}_k \qquad 24$$

The steepest descent direction is determined during the first iteration ($k = 0$) of the CG algorithm, according to:

$$p_0 = -g_0 \qquad 25$$

where p_0 represents the search direction and g_0 the steepest gradient descent direction.

The next search direction is computed such that it is conjugate to the previous search direction. This is achieved by combining the new steepest descent gradient direction $\bar{g}_k$ with the product of the previous search direction $\bar{p}_{k-1}$ and factor β_k, using [55]:

$$\bar{p}_k = -\bar{g}_k + \beta_k \bar{p}_{k-1} \qquad 26$$

There are multiple different versions of the CG algorithm, each distinguished by the method of calculating the factor β_k [56].

The Scaled Conjugate Gradient (SCG) algorithm was developed by Møller [59], [60] as an adaption to the Conjugate Gradient (CG) algorithm. The SCG algorithm replaces the line search technique, used to approximate the step size, with the *model-trust region* from the LM algorithm. The SCG algorithm is thus a combination of the CG and LM algorithms [61].

The SCG algorithm is less computationally expensive than the CG algorithm, although it generally requires more iterations to converge to a global minimum error [62]. Furthermore, the CG algorithm requires problem-dependent and user-defined parameters to determine how many iterations are required before its termination, whereas the SCG algorithm inherently performs the scaling of these parameters independent of the user [63].

The step size s_k is approximated using the model trust region, as given by:

$$s_k = \frac{\mathcal{E}_T{}'(w_k + \sigma_k p_k) - \mathcal{E}_T{}'(w_k)}{\sigma_k} + \lambda_k p_k \qquad 27$$

where λ_k and σ_k are scalar scaling factors and $\mathcal{E}_T{}'$ is the first derivative of the total error function $\mathcal{E}_T$. Equation 27 is thus an approximation of the Hessian matrix. The scaling factors are initialised by the user at the beginning of training such that $0 < \lambda_k < 10^{-6}$ and $0 < \sigma_k < 10^{-4}$ [59].

The search direction p_k is computed using the same procedure as for the CG algorithm, using Equation 26. The factor β_k for the SCG algorithm is calculated using the equation [64]:

$$\beta = \frac{|g_{k+1}^T|^2 - g_{k+1}^T g_k}{g_k^T\, g_k} \qquad 28$$

The procedure for the SCG algorithm is given by Table 15 in *Appendix A2: SCG algorithm* (page 96).

Generalisation

Overfitting & underfitting

A network's generalisation represents its ability to make adequate predictions when using independent (new unseen) data and can either be underfitted, have a good (robust) fit or be overfitted. Independent data refers to data that was not used to train the network. The differences between the generalisation abilities of a network are illustrated by Figure 12. The ideal is to obtain a robust fit such that the neural network can make adequate predictions for independent data [65].

Figure 12: Differences between an underfitted, robust fit and overfitted neural network [65]

Using the validation error to stop training

The validation error is used to stop training to obtain the best generalisation and prevent under- and overfitting. If a neural network is trained using a gradient descent or Newton method (e.g. LM and SCG), the training subset's error will monotonically decrease. The error of the validation subset will decrease during the initial training period when the network is underfit, however, the error will start to increase once the network overfits the function [33].

As the validation error increases, training will continue for an appreciable number of epochs to ensure that the error's trend continues to increase. Then the weights that were attained at the minimum validation error (stopping point) are used as the final weights for the network. This process is illustrated by Figure 13, showing the output error for the training and validation subsets versus the number of epochs the network has been trained [33].

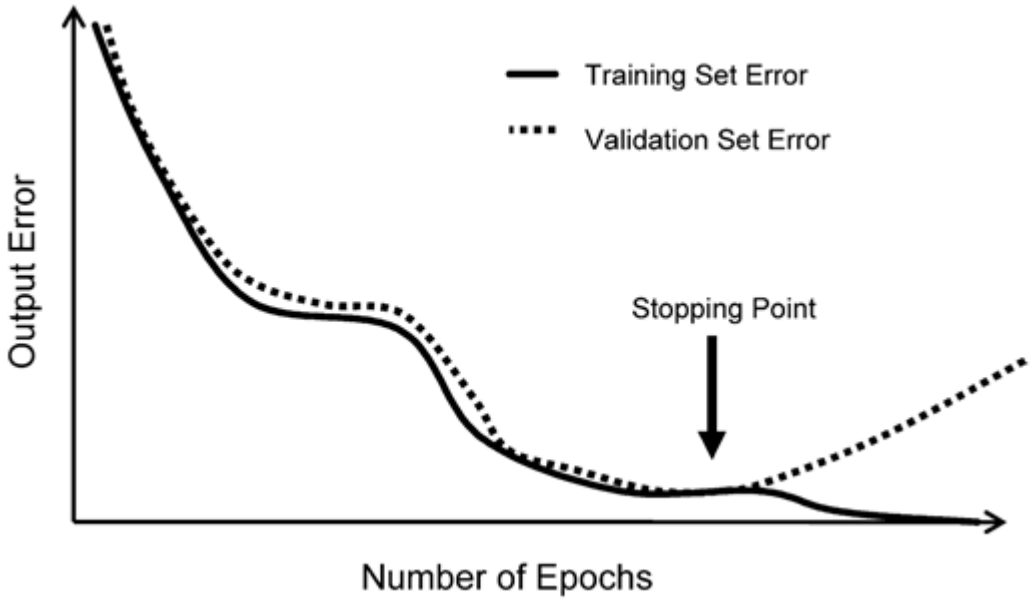


Figure 13: Output error versus epochs during the training process indicating the stopping point where the best generalisation of the network is attained and weights selected [33]

1.3.3. Rules of thumb

The rules of thumb for the number of hidden layers, number of hidden neurons, data population size, distribution of the training subset's data and transfer function selection are discussed in this section.

Number of hidden layers

Cybenko [66] established that one hidden layer is sufficient for the mapping of any given function, provided enough hidden neurons are employed. It has been demonstrated that networks with two hidden layers have the ability to converge faster to a minimum error than those with single hidden layers [33].

There is a discrepancy in literature regarding the use of more than two hidden layers. Some studies suggest using networks with three hidden layers when extreme accuracy is required [67], [68]. However, other sources state that using more than two hidden layers is unnecessary, as networks with a single layer or two hidden layers are already able to solve most problems. These sources state that networks with simpler architecture are preferred as

their mapping abilities tend to perform better than networks with more complex architectures [41].

Number of hidden neurons

The number of hidden neurons employed in the network must correlate with the complexity of the system (or function) to be approximated. Too few hidden neurons will underfit the system and result in a simple lower order function (for example a straight line). Too many hidden neurons will overfit the system, resulting in a more complex higher order function [69]. Both underfitting and overfitting will exhibit a high prediction error (poor performance) on the validation subset. A generic example of a network's approximation performance that has too few (underfit), optimal (ideal fit) and too many (overfit) hidden neurons, with respect to the complexity of the function to be approximated, is given by Figure 14 [33].

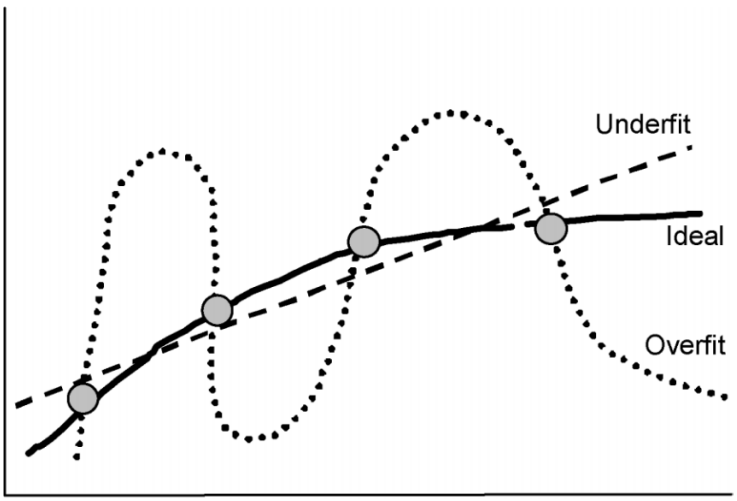


Figure 14: Generic illustration of the impact of the number of hidden neurons on the generalisation ability of approximating a function, showing underfit, ideal fit and overfit scenarios [33]

Using the validation error to select number of hidden neurons

There are multiple different techniques that can be used to determine the optimal network size (number of hidden neurons). One such technique involves using the validation subset's error to select the optimal number of hidden neurons [33]. This technique is also known as sensitivity analysis and is similar to the exhaustive search method [70].

As the number of hidden neurons increases, the training subset's error will continue to decrease. However, the validation error will initially decrease, reach a minimum and increase again as the network's size increases. The optimal number of neurons for the network is found at the minimum validation error, as shown by the graph in Figure 15 [33].

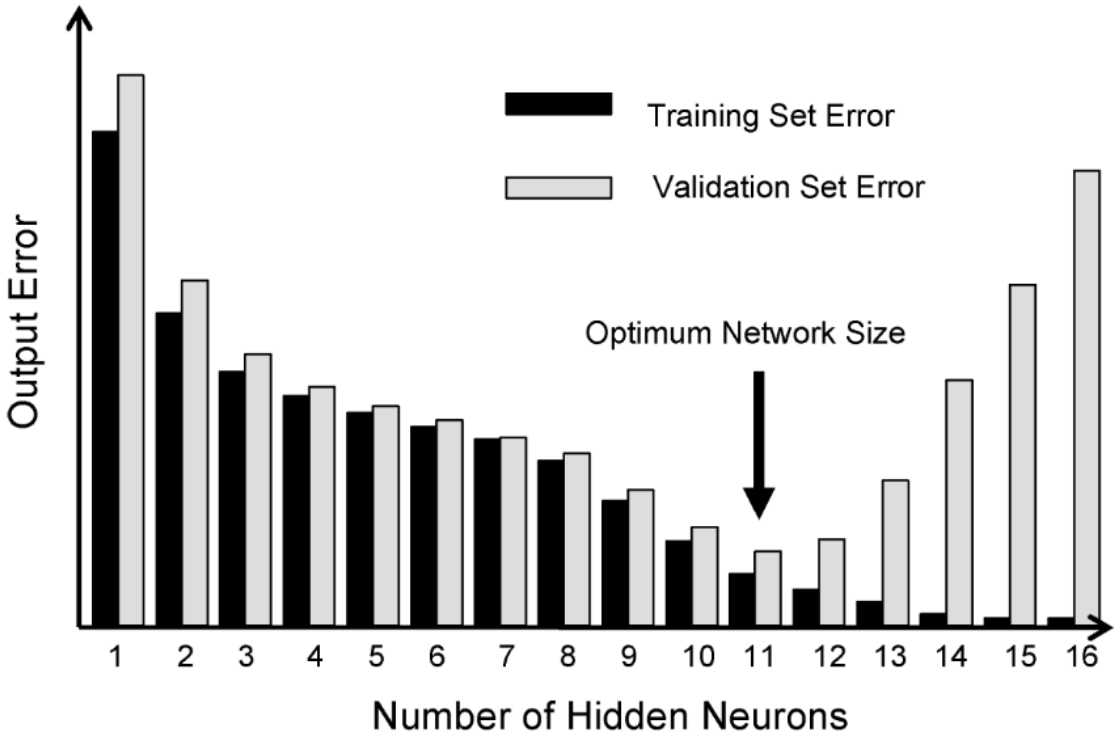


Figure 15: Output training and validation subset errors as a function of the number of hidden neurons, indicating the optimal network size at the minimum validation error [33]

Data population size

Larger data sets are more beneficial for the training process of an ANN, as these data sets provide a better overall representation of the function that is trained. A better general representation ensures more accurate predictions. Furthermore, larger data sets are better able to avoid noise in data sets [71].

Distribution of the training subset data

Feedforward neural networks are particularly effective in approximation and interpolation of functions (systems). Interpolation pertains to making predictions with a set of inputs that are within the upper and lower bounds of the data set used to train the network. Feedforward

neural networks are not able to extrapolate outside these bounds without incurring large errors. An illustration of the interpolation and extrapolation regions are given by Figure 16. It is therefore of importance to ensure that the network's predictions remain within the boundaries of the data used to train the network [33].

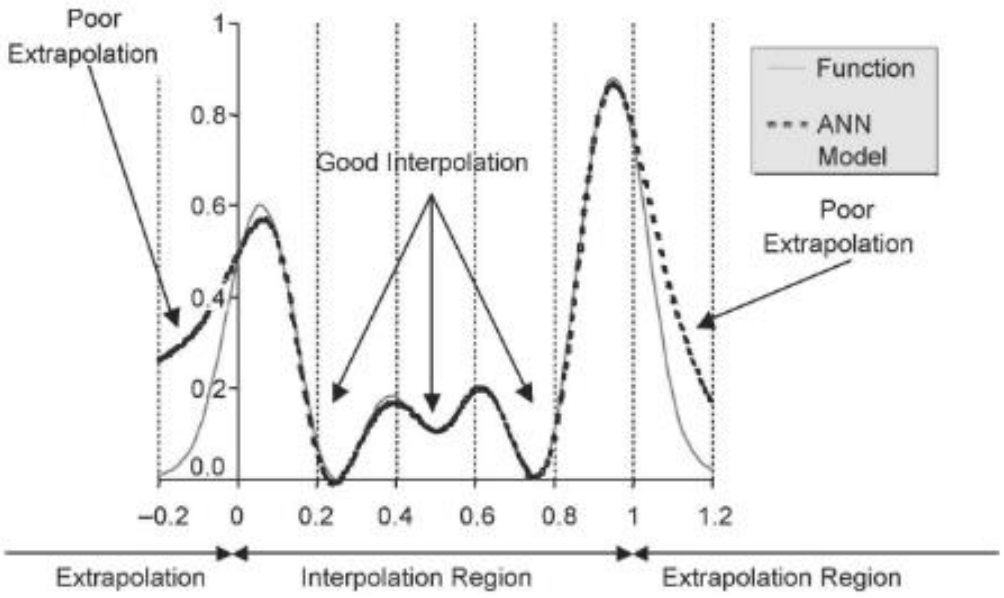


Figure 16: Illustration of the interpolation and extrapolation abilities of a neural network, with the output error on the vertical axis and normalised inputs on the horizontal axis [33]

Transfer functions

The hidden and output layers of a feedforward network can have any one of the transfer functions given in the *Perceptron* section (starting on page 7). For BP, LM, CG and SCG algorithms, the transfer function must be differentiable. Both the non-linear tangent sigmoid and logistic sigmoid functions are suitable as they can be differentiated [33], [72].

1.4. Modelling engine performance using ANNs

The literature review focuses specifically on studies where the operating, design and performance parameters of engines are modelled using artificial neural networks. A critical review is performed on three studies that are most similar to this study. Thereafter, the literature review is extended to other similar studies that are briefly discussed.

The review features studies for both standard and alternative fuel types. Standard fuels comprise petrol and diesel, whilst alternative fuels include natural gas (methane), methanol, ethanol, biodiesel and hydrogen. A detailed state-of-the-art review for the implementation of ANNs in modelling spark-ignition and compression-ignition engines, for both fuel types, has been compiled by Bhatt *et al.* [73].

1.4.1. Critical review of previous studies

The critical review focuses on the following parameters:

- Objective:
    - Type of engine
    - Type of fuel
    - Input parameters (feed flow rate, feed concentration, cooling fluid temperature)
    - Output parameter (power)
- Method:
    - Training algorithm
    - Range of hidden neurons varied to determine best performing network
    - Number of hidden layers
    - Transfer function
- Results & conclusion:
    - Best performing neural network architecture
    - Performance parameters of the best performing network
- Contribution to this study
- Shortcomings

Study A [30]

Title:

Prediction of engine performance for an alternative fuel using artificial neural networks

Objective:

In the study by Çay *et al*. [30], an ANN model was developed to predict the effective power (P_E), mean effective pressure (AP_E), brake specific fuel consumption (BSFC), and exhaust gas temperature (T_{ex}) of an SI engine. The engine is a 4-cylinder, 4-stroke 1.3 L vehicle engine. Methanol was used as fuel. Torque, engine speed, fuel feed flow rate, air feed temperature and cooling water feed temperature are used as inputs to the model.

Method:

Experiments were performed on the engine to obtain data to train the model. An ANN model for effective power, as shown by Figure 17, was developed using the experimental data for each output parameter. The training algorithm and number of hidden neurons in a single hidden layer were varied to obtain the best performing network. The LM and SCG training algorithms were implemented, and the number of hidden neurons varied between 3 and 15. The logistic sigmoid and pure linear functions were used as transfer functions for the hidden neurons and output neurons, respectively.

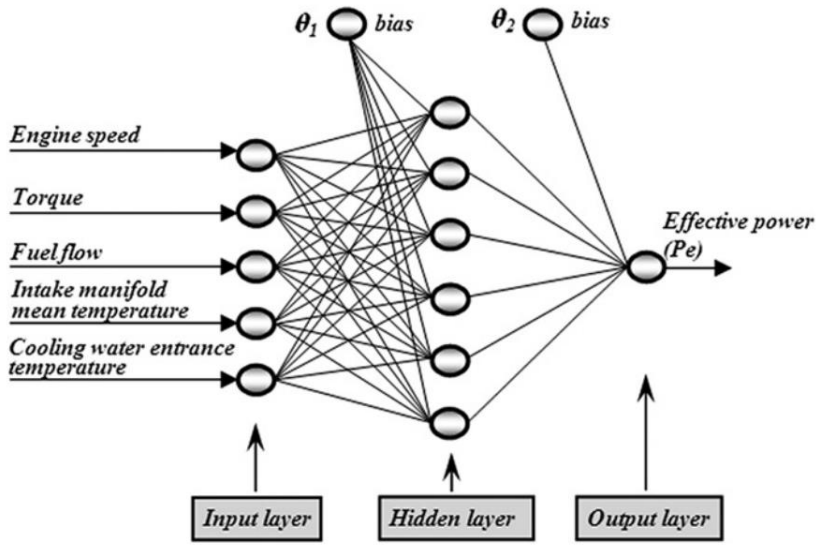


Figure 17: Architecture of the ANN to predict the effective power of the experimental methanol SI engine

Results:

The best ANN model was determined to have 6 hidden neurons with one hidden layer, trained using the LM algorithm. The prediction accuracy of the output parameters with their targets is given in Figure 18. The performance parameters used to evaluate the network performance are the RMSE, MAPE and R^2. The results of these parameters, as well as the number of neurons and training algorithm employed for each parameter, are summarised in Table 3.

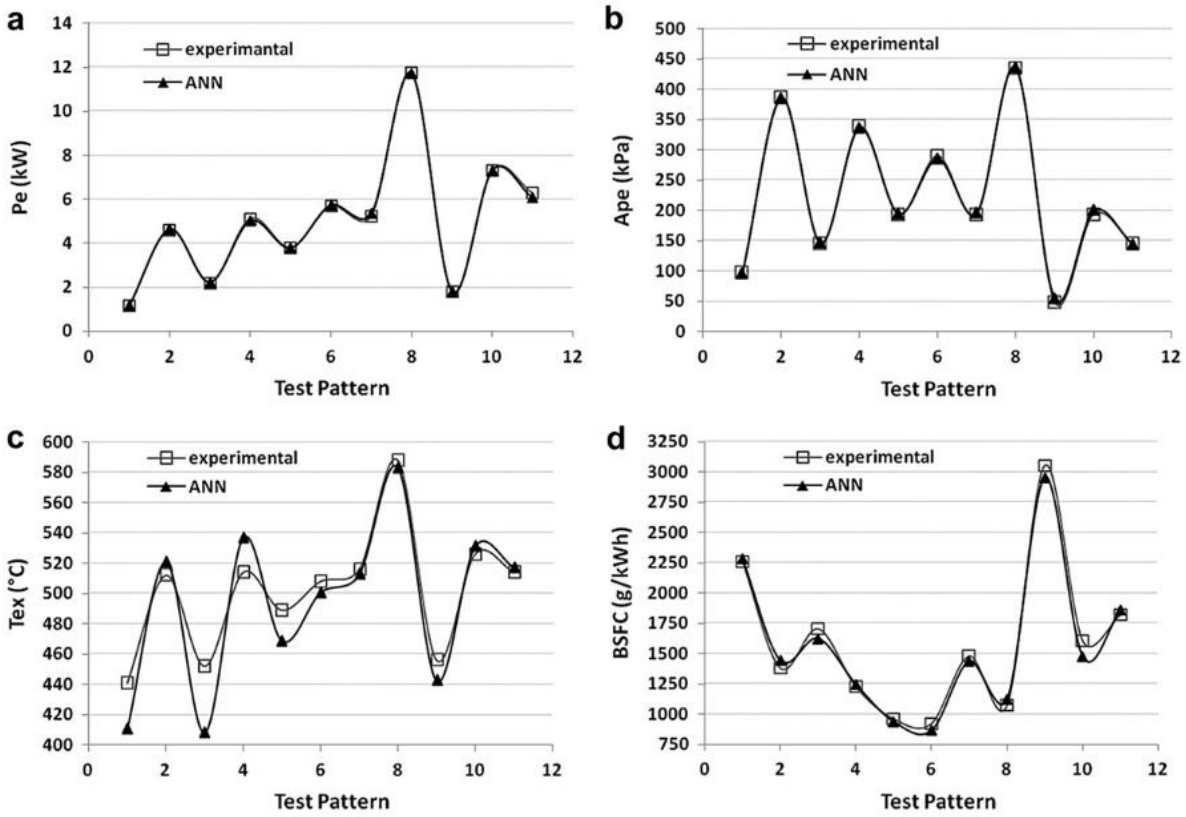


Figure 18: Accuracy of the ANN model's predictions versus its targets for the output parameters (a) effective power P_E, (b) mean effective pressure AP_E, (c) exhaust gas temperature T_{ex} and (d) brake specific fuel consumption (BSFC)

Table 3: Results for the best performing ANNs for the experimental methanol SI engine

Goal	Training algorithm	Number of neurons	Training data			Verification data		
			RMSE	R^2	MAPE	RMSE	R^2	MAPE
P_E	LM	5-6-1	0.0039	0.9999	0.8236	0.0064	0.9998	1.1438
AP_E	SCG	5-7-1	0.0048	0.9999	1.5283	0.0068	0.9998	2.0743
T_{ex}	LM	5-7-1	0.0044	0.9999	0.7953	0.0135	0.9993	3.1408
BSFC	LM	5-5-1	0.0047	0.9998	1.2348	0.0119	0.9985	3.7617

Contribution to this study:

There are multiple similarities and differences between this study and that of Çay *et al.* Both studies implement the fuel flow rate and water (coolant) temperature as input parameters, whilst having the power generated by the engine as an output parameter. This study also implements the LM and SCG algorithms to train the ANN models and uses the logistic sigmoid and linear transfer functions for the hidden neurons and output neurons, respectively.

Shortcomings:

Although both studies investigated the performance of an SI engine, this study's engine is fuelled by methane and that of Çay *et al*. is fuelled by methanol. Furthermore, the engines differ in size. The data collected by Çay *et al*. was obtained from experiments whereas the data collected in this study was obtained from the operation of the engine at a deep-level mine.

Study B [74]

Title:

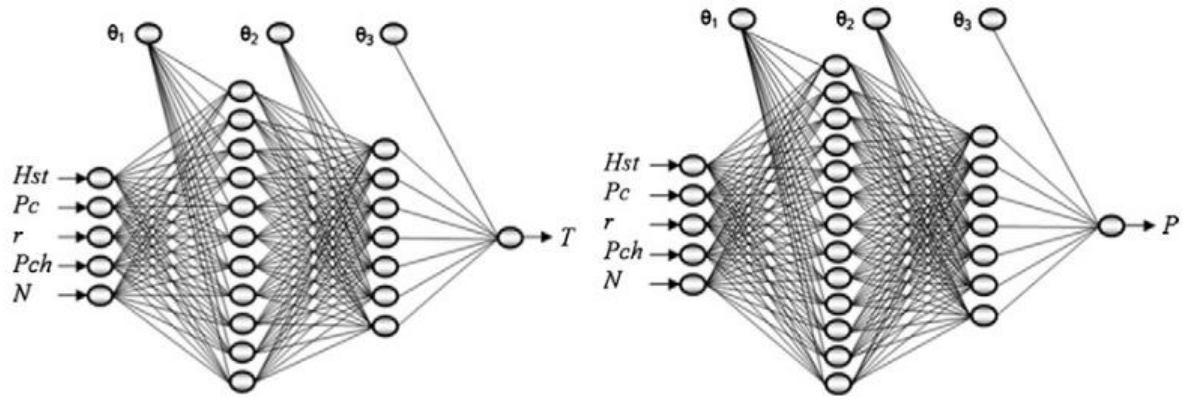
Predictive modelling of performance of a helium charged Stirling engine using an artificial neural network

Objective:

Özgören *et al.* [74] developed an ANN model to predict the power and torque of an experimental Stirling engine, using the engine's heat source temperature, compression ratio, charge pressure, engine speed and type of piston coating as inputs. The working fluid of the engine is helium.

Method:

Multiple neural networks were trained using the LM and SCG algorithms. Two hidden layers were employed and the number of hidden neurons varied between 7 and 15 in each layer to determine the best performing neural network. The developed model is given by Figure 19 for torque (T) and power (P). The logistic sigmoid function was used as transfer function for the hidden neurons and linear transfer function for the output neurons.

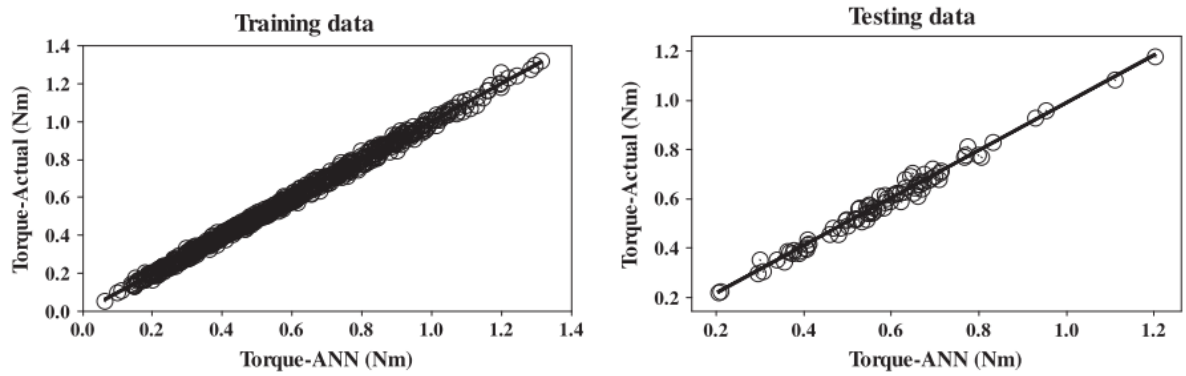


(a) (b)

Figure 19: Architecture of the ANN models employed for the helium charged sterling engine to predict (a) torque and (b) power

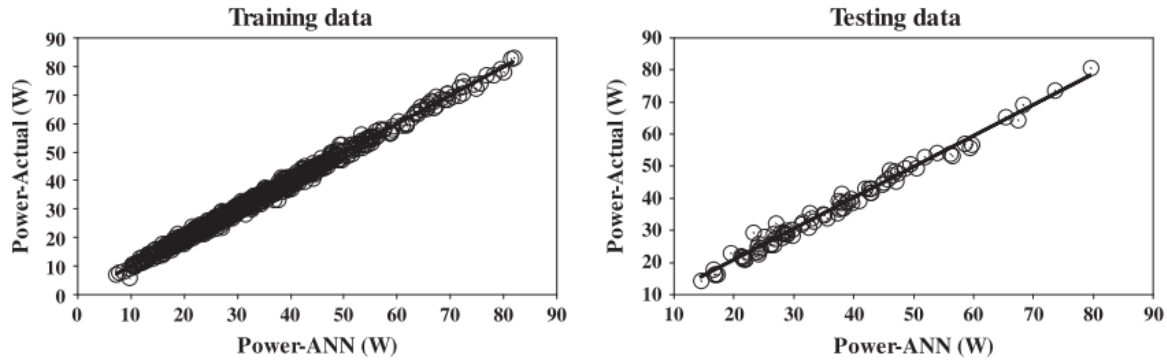
Results:

The best performing model architecture for torque (T) and engine power (P) was determined to have architectures of 5-11-7-1 and 5-13-7-1, respectively. The LM algorithm provided better results than the SCG algorithm. The resulting regression for the training and verification data subsets predicted by the best performing models is given by Figure 20 for torque and by Figure 21 for engine power. The resulting performance results of the best neural network architectures are summarised in Table 3.



(a) (b)

Figure 20: Regression performance of the best performing ANN model to predict torque for the (a) training data subset and (b) verification data subset



(c) (d)

Figure 21: Regression performance of the best performing ANN model to predict power for the (a) training data subset and (b) verification data subset

Table 4: Results for the best performing ANNs for the experimental helium charged sterling engine

Goal	Training algorithm	Number of neurons	Training data			Verification data		
			RMS	R^2	MAPE	RMS	R^2	MAPE
Power (P)	LM	5-11-7-1	0.0117	0.9993	2.4111	0.016	0.9987	2.9397
Torque (T)	LM	5-13-7-1	0.0118	0.9991	2.8680	0.0181	0.9982	3.4985

Contribution to this study:

Both this study and the one by Özgören *et al.* have power as an output parameter to the ANN. The LM and SCG algorithms are implemented in both studies to determine the best trained ANNs. Two hidden layers were employed and the number of hidden neurons is varied in both studies to determine the best performing ANN architecture. Both studies have allocated a division of experimental data in the ratio of 90 % for training and 10 % for verification. Furthermore, in both studies, the logistic sigmoid and linear functions are used as transfer functions for the hidden and output neurons, respectively.

Shortcomings:

The main shortcoming of the study by Özgören *et al.* with respect to this study, is that an experimental sterling helium charged engine is investigated. This study investigates a much larger methane SI engine located at a deep-level mine. Further differences are that this study trains ANNs with plant data that requires noise filtering and removal of non-operational data,

unlike the study of Özgören *et al.* where only experimental data was used. The input parameters to the ANN models of the two studies also differs.

Study C [75]

Title:

Biogas engine performance estimation using ANN

Objective:

ANNs were trained by Kurtgoz *et al*. [75] to predict the volumetric efficiency, thermal efficiency and BSFC of an experimental 4-stroke spark-ignition engine run on biogas generated from the anaerobic fermentation of bovine manure. The main component of biogas in the study was methane. The input parameters to the model were selected as the methane concentration in the feed fuel, engine load, fuel feed temperature (T_{in}), air-fuel ratio (AFR) and maximum cylinder pressure (P_{max}).

Method:

In the study by Kurtgoz *et al*., multiple experiments were performed on the SI engine to obtain data to train the ANN models. The ANN model used by Kurtgoz *et al*. is given by Figure 22. The BP algorithm was used to train the ANN models with varying architectures to determine the best performing model. The number of hidden neurons were varied between 2 and 12. The logistic sigmoid and pure linear functions were used as transfer functions for the hidden neurons and output neurons, respectively.

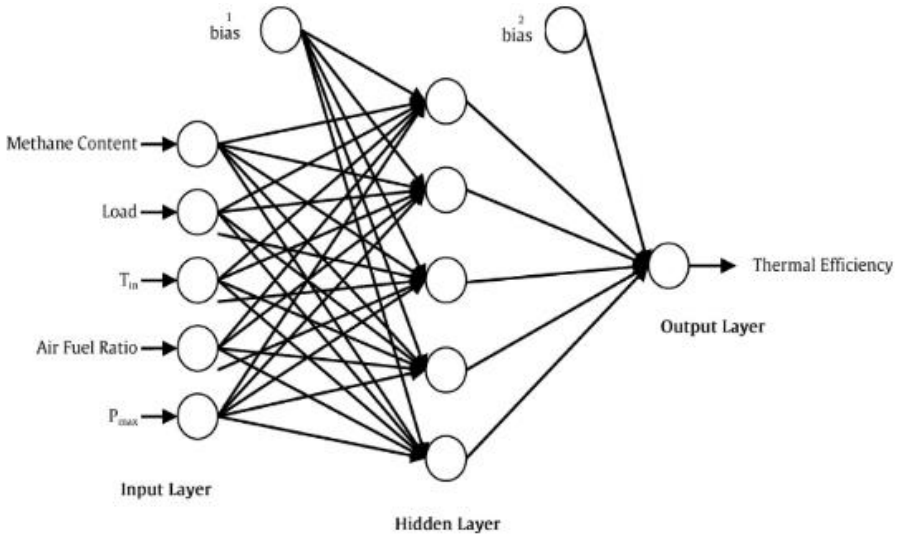


Figure 22: ANN architecture for the biogas-fuelled experimental biogas engine

Results:

After varying the number of neurons, the best architecture was determined to be a single layer, trained with the BP algorithm, with 5, 10 and 6 hidden neurons for thermal efficiency, BSFC and volumetric efficiency output parameters, respectively. The prediction accuracy of the output parameters with respect to their targets is given by Figure 23. The performance parameters used to evaluate the ANN performance are the RMSE, MAPE and R^2. The results of the performance parameters, as well as the best performing architecture, is summarised in Table 5.

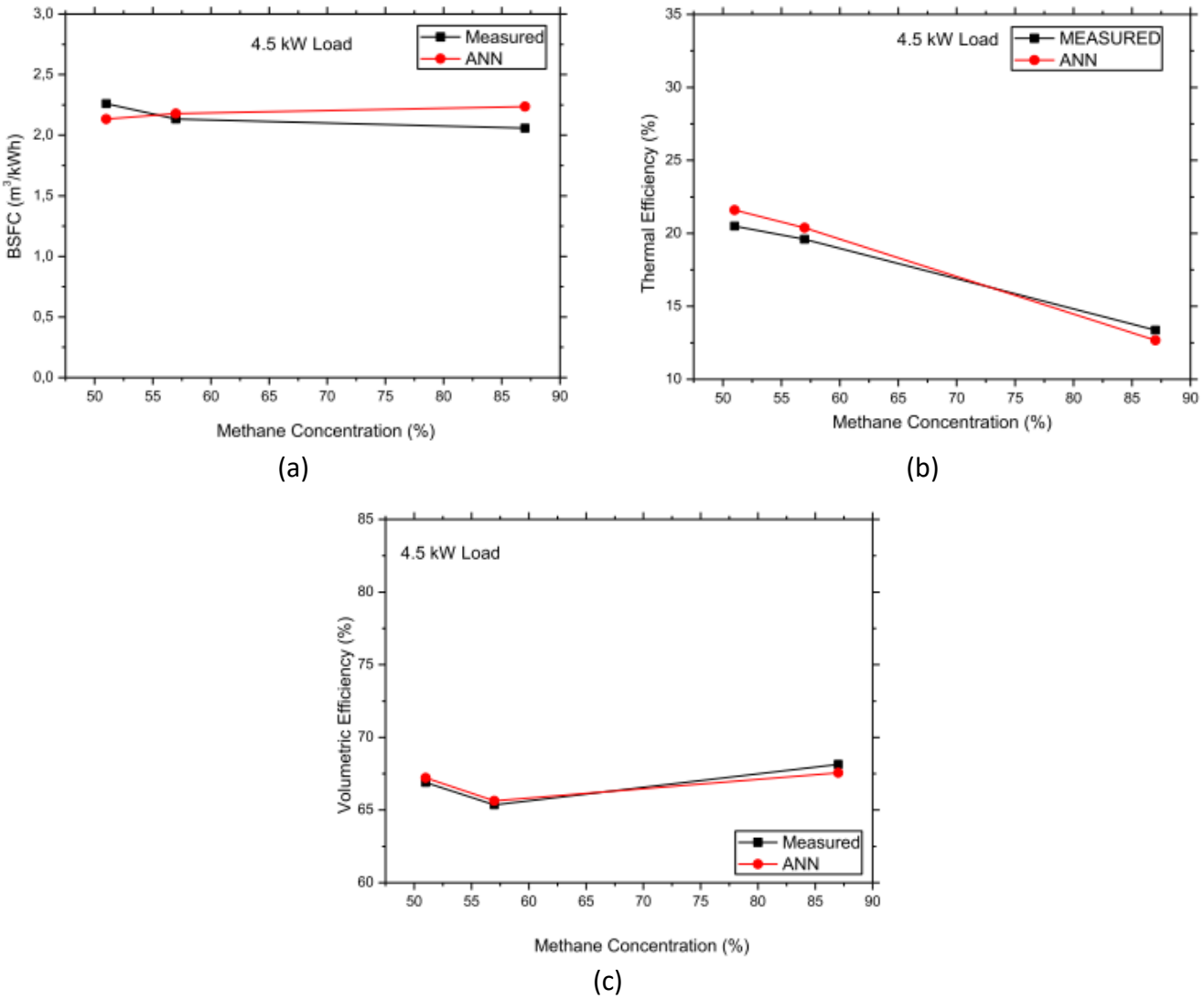


Figure 23: Accuracy of the ANN model's predictions versus its targets over varying methane concentrations for (a) BSFC, (b) thermal efficiency and (c) volumetric efficiency

Table 5: Results for the best performing ANNs for the experimental biogas SI engine

Goal	Number of neurons	Training data			Verification data		
		RMSE	R^2	MAPE	RMSE	R^2	MAPE
BSFC	5-10-1	0.30	0.9992	1.80	0.78	0.9901	3.19
Thermal efficiency	5-5-1	0.04	0.9996	1.36	0.11	0.9594	5.26
Volumetric efficiency	5-6-1	0.28	0.9996	0.29	1.49	0.9608	1.54

Contribution to this study:

Both this study and that of Kurtgoz *et al*. investigated the usage of ANNs to model an SI engine fuelled with biogas (primarily methane). Both studies also used the methane composition of the feed stream to the engine as an input to the ANN.

Shortcomings:

Kurtgoz *et al*. used the BP algorithm to train ANNs, whereas, this study used the LM and SCG algorithms. Furthermore, the input parameters (excluding methane composition) and output parameters of the model developed by Kurtgoz *et al*. differs from those used in this study. The data used in this study is plant data from a methane-fuelled SI engine, and therefore must first be filtered, unlike the experimental data used by Kurtgoz *et al*.

1.4.2. Extended review of previous studies

Togun and Basec [76] developed an artificial neural network model that predicts the BSFC and torque of a spark-ignition engine that uses petrol as fuel. The engine's speed, spark advance and throttle position are used as inputs to the model. Data was obtained from experiments performed on the test engine setup, with 63 and 18 data sets being allocated to training and verification, respectively.

The ANN was trained using the LM backpropagation algorithm and the logistic sigmoid was selected as the transfer function. The best neural network topology for the system was determined to be one hidden layer with 13 hidden neurons. The R^2 values for the BSFC and torque is 0.98331 and 0.9971, respectively, with a MAPE of 1.0186 and 2.7588 for BSFC and torque, respectively.

In a study by Çay [77], power, BSFC and exhaust gas temperature (EGT) were predicted using ANNs from fuel flow rate, engine speed, inlet manifold temperature, cooling water temperature and torque as inputs. Different number of hidden neurons (for a single hidden layer) and training algorithms were used to determine the best performing neural network. The number of hidden neurons were varied between 3 and 15 and the LM and SCG algorithms used to train the ANNs. The best performing neural network was trained using the SCG algorithm with 7 hidden neurons, achieving RMSE, MAPE and R^2 values of 0.02, 0.027 and 0.99, respectively.

Experimental studies were conducted by Sahin [78] to develop models to predict the AFR in the cylinder of a petrol engine by implementing engine speed, ignition timing, engine load and the excess air coefficient as input parameters. Both SCG and LM training algorithms were implemented with varying numbers of hidden neurons, between 5 and 20 in two hidden layers, to determine the best performing ANN. The experimental data was divided into 75 % for training and 25 % for verification purposes. The best neural network was determined to have 9 hidden neurons in both hidden layers, trained using the SCG algorithm. The coefficient of determination for the training and verification sets was 0.99698 and 0.99206, respectively.

An ANN model was developed by Golcu et al. [79] to predict torque and BSFC of an experimental petrol engine using engine speed and valve-timing as inputs. The number of hidden neurons was varied between 15 and 19, whilst, both the SCG and LM training algorithms were implemented to determine the best performing ANN model. The logsig and purelin were implemented as transfer functions. The experimental data was divided into 62 data points for training and 15 data points for verification. The best network was determined to have 15 hidden neurons with one hidden layer, 2-15-2 architecture and trained using the LM algorithm. The RMSE of torque and BSFC was 0.009 and 0.0028, respectively.

The effect of a chromium carbide engine coating on the performance and emissions of a gasoline engine was investigated by Hazar and Gul [32]. An ANN model was trained using different engine speeds for the coated and uncoated engine. The tangent sigmoid transfer function was implemented. An ANN with one hidden layer and 10 hidden neurons was sufficient, with the resulting coefficient of determination approaching 1.

In a another study by Cay *et al*. [80], BSFC, AFR, carbon monoxide (CO) and unburned hydrocarbon (HC) exhaust emissions of an engine fuelled by a mixture of gasoline and methane were predicted with ANNs trained using different training algorithms. The algorithms used were the LM, SCG, Broyden-Fletcher-Goldfard-Shanno (BFGS) and Resilient Backpropagation (RP) algorithms. The number of hidden neurons in a single hidden layer were varied between 5 and 15 and the best performing neural network was determined to have 7, 11, 7 and 14 hidden neurons to predict BSFC, AFR, CO and HC, respectively. The corresponding coefficients of determination were 0.9986, 0.9961, 0.9983 and 0.9777.

Further similar studies are presented by Kapusuz *et al.* [81], Kiani *et al.* [82], Najafi *et al.* [83], Yu and Arcakliog [84] and Sayin *et al.* [85].

1.5. Summary of the state-of-the-art

The state-of-the-art for the prediction of power generated by a methane-fuelled SI engine is given by Table 6. The legend for the state-of-the-art is given by Table 7. Each study was investigated and evaluated according to nine criteria:

1. **Fuel – Methane:** Was the engine that was investigated and modelled by the study fuelled by methane (or biogas)? This is important because the gas emissions from specific deep-level mines comprises primarily methane.
2. **Engine – Spark-ignition:** Did the study model a SI engine? This is needed because the type of engine implemented at specific deep-level case study mine is an SI engine that generates power from the emitted gas.
3. **Input – Fuel flow rate & methane feed composition:** Did the study implement the fuel flow rate and methane feed composition as inputs to the ANN model? This is important because these two input parameters have a direct causal effect on the power generated by the engine.
4. **Output – Power generated:** Was power generated by the engine used as an output to the ANN model? This is needed as the focus of this study is on modelling the performance of an engine that is quantified by the power that it generates.
5. **Training algorithm – LM & SCG:** Was the ANN model trained using the LM and/or SCG algorithms? These two training algorithms are focused on and investigated by this

study as both, according to literature, are the most commonly employed training algorithms for models that predict engine performance. Furthermore, the training algorithm employed affects the ANN model's prediction performance, accuracy and robustness.

6. **Number of hidden layers – One & Two:** Did the study develop an ANN with architecture comprising one or two hidden layers? This criterion is important as the number of hidden layers affects the ANN model's generalisation and learning abilities, which are essential to the model's prediction performance, accuracy and robustness.

Table 6: State-of-the-art for the performance prediction of a methane-fuelled SI engine using ANNs

ARTICLE	Fuel	Engine	Inputs		Output	Training algorithm		Hidden layers		COMMENTS
	Methane	Spark-Ignition	Fuel flow rate	Methane %	Power	LM	SCG	Single	Two	
[7]										Hydrogen & methane SI engine
[14]										Helium sterling engine
[15]										Gasoline SI engine
[16]										Methanol SI engine
[17]										Biogas waste to energy
[18]										Methanol & gasoline SI engine
[19]										Gasoline SI engine
[20]										Gasoline SI engine
[21]										Biogas SI engine
[22]										Hydrogen SI engine
[23]										Biogas SI engine

Table 7: Legend for the state-of-the-art

LEGEND	
	Study considered the criteria
	Study did not consider the criteria

1.6. Problem statement and objectives

This study's problem statement, aim and objectives are formulated, following from the introduction and review of literature presented in this chapter.

1.6.1. Problem statement

From the literature review and state-of-the-art, no model that relates the operational parameters – specifically fuel feed flow rate and methane composition – of a methane-fuelled spark-ignition engine to the power generated by the engine exists. Such a model would prove useful to perform economic optimisation on the system, especially as the amount of power generated is highly dependent on the varying operational parameters.

1.6.2. Aim

The aim of this study is to develop and evaluate a model that predicts the power generated by a methane-fuelled spark-ignition engine from its operational parameters by implementing artificial neural networks.

1.6.3. Objectives

The objectives of this study are:

- Select appropriate input and output parameters
- Process available data by implementing filters, normalisation and data set allocation
- Develop ANN models with the selected input and output parameters by training ANNs with different characteristics:
    - Number of hidden neurons
    - Training algorithms
    - Number of hidden layers
- Select the best performing ANN model
- Implement the best performing ANN model by:
    - Evaluating its prediction accuracy
    - Developing mathematical equations that describes the model

    - Using the equations to evaluate the relationship between the input and output parameters and accordingly determine the best operating conditions (with the highest power output)
    - Applying the model (by evaluating the developed equations) to improve the performance of the engine
- Verify the model by implementing the best performing model on new unseen data and compare the model's resultant predictions to its targets

1.7. Summary

The case study mine has implemented an SI engine that generates electricity from the gases – primarily methane – emitted from underground sources. The power generated by the methane-fuelled SI engine is highly dependent on its operational parameters. A model is required to perform economic and operational optimisation on the system. From the different modelling approaches, an ANN model was selected as most appropriate to model the SI engine from the available data.

A critical literature review of three similar studies was performed. It was found that the power generated by SI engines can be modelled with the engine's feed flow rate, feed composition and coolant temperature as input variables. It was determined that the model can be applied successfully to different fuel types, including methane. It was further found that the number of hidden neurons, training algorithms and number of hidden layers employed must be varied and performances compared to determine the best model for the SI engine.

An extended review was performed on six additional studies. These studies are summarised according to nine criteria into a state-of-the-art table. From the literature review and state-of-the-art, it was found that a model that relates the operational parameters of a methane-fuelled SI engine with the power it generates, specifically at a deep-level mine, does not exist. Thereafter, the problem statement, aim and objectives of this study were developed.

1.8. Outline of document

Chapter 1: Introduction – The chapter presents the background required for this study, followed by an introduction of ANNs. A critical and extended review on literature similar to

this study – that model engine performance with ANNs – was performed. From the background and literature review, the study's problem statement, aim and objectives were formulated.

Chapter 2: Method – The chapter investigates the specific case study of the engine to be modelled using ANNs. The methodology through which the data of the applicable input parameters was processed and how the model was developed, implemented and verified is discussed.

Chapter 3: Implementation and results – In Chapter 3, an ANN model is developed, implemented and verified according to the method presented in the Chapter 2. The results of each step is presented and discussed. The network with the best performing characteristics is identified for the development of the model. The developed model is implemented to determine the operating conditions that favours the performance of the SI engine. The model is verified by applying it to an unseen and independent data set.

Chapter 4: Conclusion – The conclusions and recommendations drawn from the model's development, implementation and verification are discussed.

Chapter 2: Method

Figure 24: Methane-fuelled SI engine at a deep-level mine[2]

"Simplicity is the ultimate sophistication."

– Leonardo da Vinci

[2] Z Lombard, Personal photograph, Welkom, 2021

2. METHOD

2.1. Preamble

Different methods from literature have been integrated and implemented to solve the problem identified by this study. The methodology's steps have been determined to specifically achieve the study's objectives. The method comprises five main stages, namely System Identification, Data Processing, Model Development, Model Implementation and Model Verification. Each stage consists of multiple steps which are discussed in detail in each respective section. The MATLAB source code for each step is given in *Appendix C: Source code* (page 100). An overview of the method followed in this study is given by Figure 25.

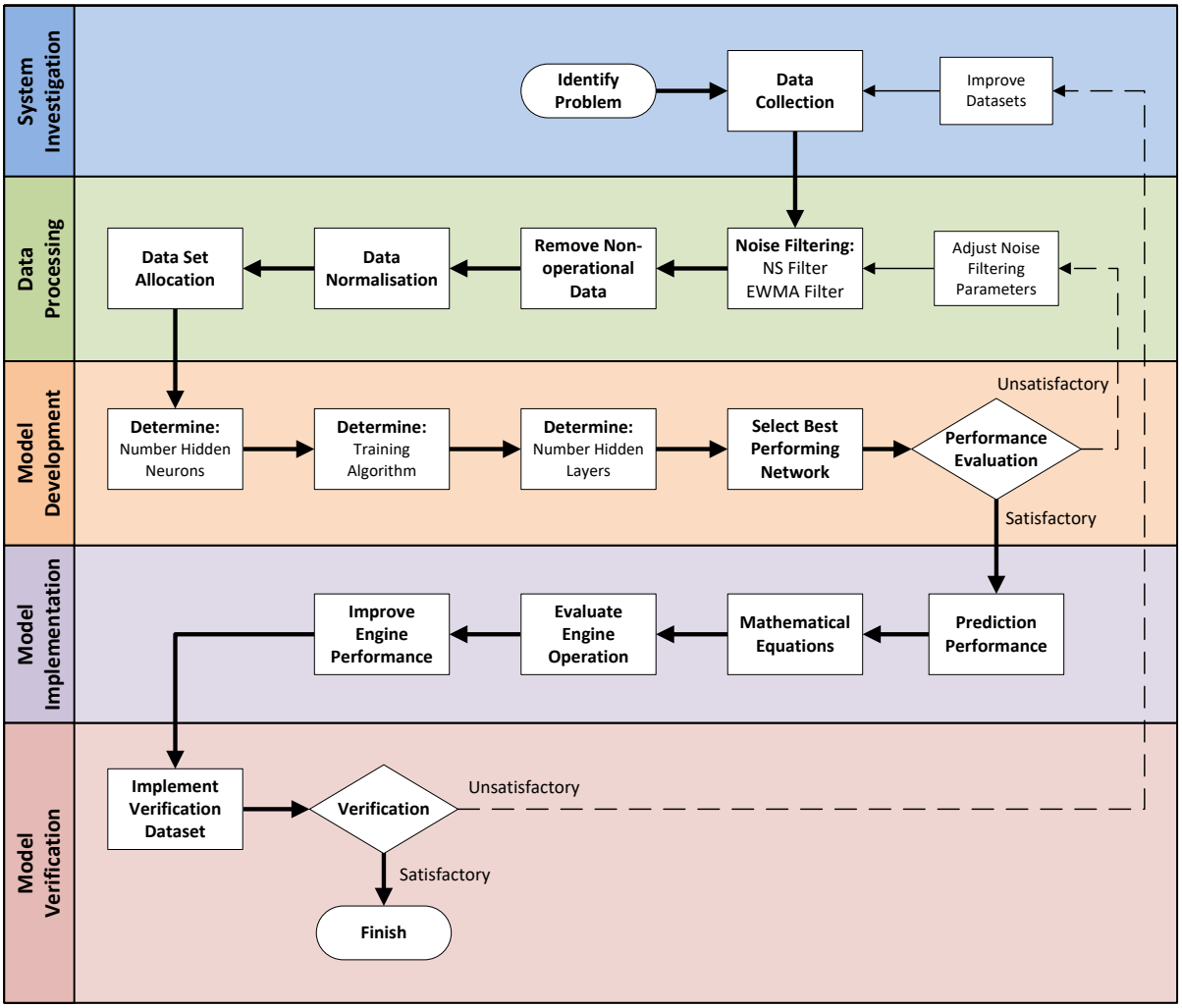


Figure 25: Overview of the method followed in this study

2.2. System investigation

The aim and objectives of the study were formulated to attain a viable solution to the identified problem, as discussed in Chapter 1. To build a model that accurately predicts the power generated by a methane-fuelled SI engine, the system and the interactions between its variables must be adequately investigated and understood. From the investigation and available data, the appropriate input and output parameters were selected and corresponding data collected.

2.2.1. Case study

Mining Complex A consists of three shafts – two active shafts and one inactive shaft. In 2011, a project was implemented at Mining Complex A to capture and flare the methane liberated by normal mining activity. By flaring, the methane is converted primarily to carbon dioxide, henceforth decreasing the mine's contribution to global warming as well as reducing its carbon tax payable to the government.

In 2013, Mining Complex A installed an SI engine as a generator that utilises methane as fuel source to generate electricity that is then returned to the power grid. The SI engine produces an average of 280 MWh per month from a feed flow rate of 400 m^3/hr of methane at a concentration that varies between 60 and 90 %. The generator only operates for 22 hours per day from Monday to Friday. The technical specifications of the engine are summarised in Table 8.

Table 8: Technical specifications of the methane engine

Item description	Specification
Make	Cummins QSK60 Gas G-Drive
Engine type	4 cycle, turbocharged, after-cooled
Cylinder block	16 cylinder
Fuel system	Direct Injection Cummins HPI
Displacement litre	60.2 litre
Power output rating	1100 kW @ 50 Hz 1300 kW @ 60 Hz
Coolant system	2 pumps – 2 loops
Coolant type	50 % ethylene glycol, 50 % water

The layout of the methane collection, cooling, flaring and power generating system is schematically represented by Figure 26. Methane is collected from walled-off inactive areas underground. Blowers on surface are used to create a negative pressure to extract the methane gas from underground to the surface. A heat exchanger cools down the captured gas to below 60°C using chilled water. The methane is then fed to the SI engine to generate electricity, moreover, the methane is flared when the engine is not running.

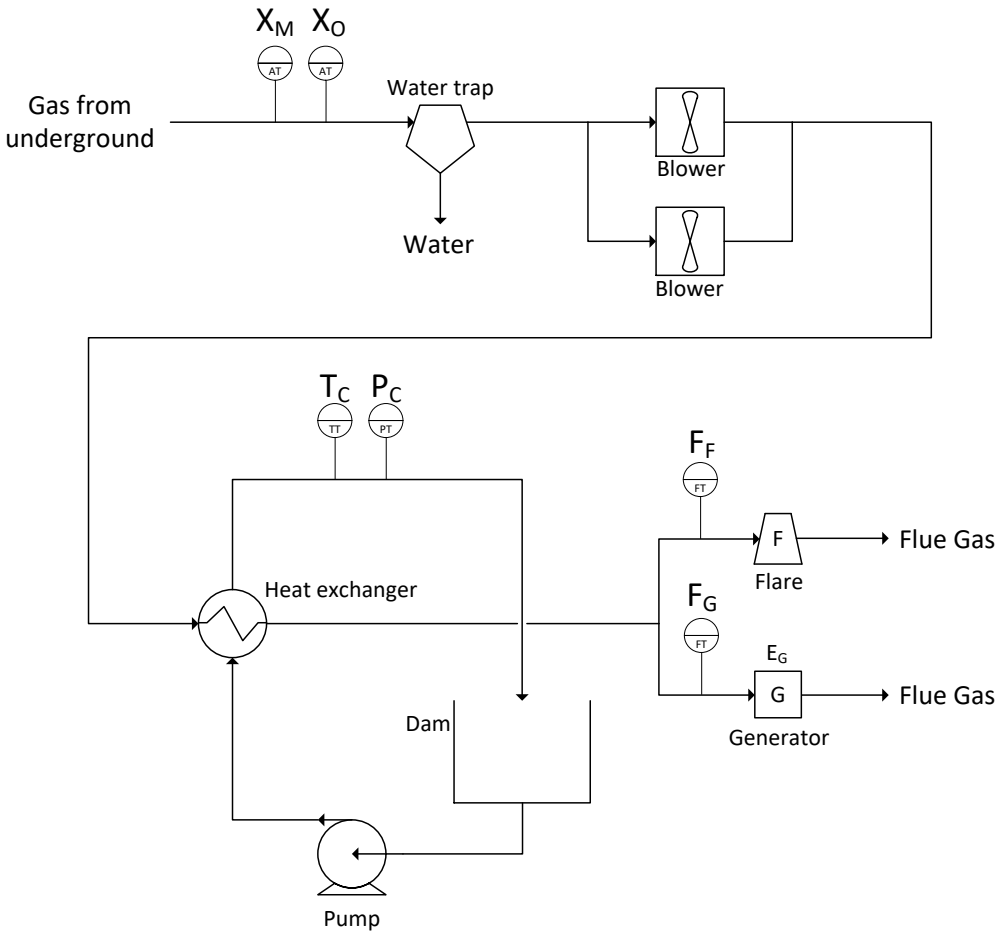


Figure 26: Schematic representation of the methane collection, cooling, flaring and power generation system

The mine is in the process of investigating the economic feasibility of either refurbishing their current generator or purchasing an additional generator. However, these generators are very expensive and thorough investigation is required to ensure a favourable return on investment. The model developed by this study can be used to supplement the economic feasibility study.

2.2.2. Data collection

Raw operational data was obtained from the methane generator's historian in 30-minute resolution from January 2020 to May 2021 (23 856 data points). The input variables to the model were selected as the feed methane concentration (X_M in %), feed oxygen concentration (X_O in %), feed flow rate to generator (F_G in m^3/hr), feed flow rate to flare (F_F in m^3/hr), temperature (T_C in °C) and pressure (P_C in kPa).

These input variables were selected on the basis that they have an influence on the power generated, can be adjusted manually and are recorded by the mine's historian. The target variable was selected as the power generated by the generator (W_G in kW). Power data was obtained from an independent power meter located at the methane generator's feeder power panel.

2.3. Data processing

The data obtained from the historian had to first be processed before it could be used to train neural networks. The operation (inputs) and power generation (target) data were both raw, containing a significant degree of noise as well as outliers. The raw data thus first had to be filtered to remove the noise and outliers as well as remove data for when the engine was not running. Thereafter, the data was normalised and cross-validation applied to allocate data into different data sets for training, validation, testing and verification purposes.

2.3.1. Noise filtering

The noise filters implemented to reduce the noise and remove the outliers in the raw data obtained from the historian were the noise-spike (NS) and exponentially weighted moving average (EWMA) filters. These filtering methods are presented in Seborg *et* al [86]. The combination of these two filters has been shown to be effective in filtering noisy data sets successfully [87].

Noise-spike filter

The noise-spike (NS) filter limits the magnitude a measured value (output) can vary between two sampling instances. If the measured value exceeds the maximum allowable change, the

difference between the measured and previously filtered values are used. Conversely, if the measured value is less than the previously filtered value, the measured value is added to the previously filtered value [86]. The mathematical expression for the NS filter is given by:

$$y_F(k) = \begin{cases} y_m(k), & if\ [y_m(k) - y_F(k-1)] \leq \Delta y \\ y_F(k-1) - \Delta y, & if\ [y_F(k-1) - y_m(k)] > \Delta y \\ y_F(k-1) + \Delta y, & if\ [y_m(k) - y_F(k-1)] > \Delta y \end{cases} \quad 29$$

where y_F and y_m are the filtered and measured values, respectively. The sampling instance is represented by k and the maximum allowable change is denoted by Δy. The function *medfilt1* in MATLAB was used for the NS filter.

Exponentially weighted moving average filter

The exponentially weighted moving average (EWMA) filter accounts for the dynamics of time-series data by, for each data point, assigning different weights of importance to the current and previous data points [88]. A dimensionless parameter, denoted α, quantifies the magnitude of the weight distribution, according to the mathematical expression given by:

$$y_F(k) = \alpha y_m(k) + (1-\alpha)\, y_F(k-1) \quad 30$$

The dimensionless parameter α is defined as follows:

$$\alpha = \frac{\Delta t}{\tau_F + \Delta t} \quad 31$$

where Δt represents the time difference between two sampling intervals and τ_F represents the time period constant. The time period constant is adjusted to control the smoothing of the raw data [86]. The function *dsp.MovingAverage* in MATLAB was used for the EWMA filter.

2.3.2. Removal of non-operation data

The methane generator is set to shut down if the methane concentration drops below 60 %. To ensure that only data applicable to the model was used, data during which the SI engine was not operating (below 60 %) was removed. By removing these non-operational data

points, the model was better able to quantify the interactions between the input and output (target) variables when the engine was operating. The inequality filter applied to the methane concentration variable is given by:

$$X_M \geq 60\ \% \tag{32}$$

2.3.3. Data normalisation

Due to the difference in magnitude between the different input and output variables, data normalisation is necessary to prevent large errors, hereby improving the prediction accuracy and generalisation ability of the trained network [89]. The equation to scale all variables to a uniform range between 0 and 1 is given by:

$$x_i = \frac{d_i - d_{min}}{d_{max} - d_{min}} \tag{33}$$

where x_i represents the normalised and d_i represents the original values for the ith data point. The maximum and minimum values for each variable are represented by d_{max} and d_{min} respectively.

2.3.4. Data set allocation

The method presented by Özgören *et al.* [74] was followed to allocate data to the specific data sets used for training, validating and testing the networks, as well as for the verification of the best performing network.

Firstly, the processed data was divided into two classes using cross-validation in the ratio of 90 % for the main data set and 10 % for the verification data set. The ratio was selected from previous studies [74], [91]. Cross-validation is a method applied to predictive models to resample data into classes to evaluate their generalisation ability [90]. The MATLAB function *cvpartition* with a *Hold Out* was used to perform cross-validation.

Secondly, the main data set was divided randomly into three subsets, namely the training, validation and testing subsets, as discussed in the *Independent sample testing* section (on

page 13). The main data set was divided according to the ratio of 70 %, 20 % and 10 % for the training, validation and testing subsets, respectively, as done by Najafi *et al.* [92].

2.4. Model development

Only feedforward neural networks were developed and implemented in this study. Recurrent networks were not considered because the recordings of the power generated at an instant is allocated directly to the flow meter readings of the same instant, accounting for no lag time. The time it takes the feed gas to flow between the flow meter and the engine is thus regarded as negligible. This is based on the fact that the case study's flow meter is located at the inlet of the SI engine and the data recorded by the historian is only available in 30-minute resolution. The same applies for the composition analysis, temperature and pressure instrumentation.

A generic schematic diagram of the multi-layered feedforward neural network developed in this study, with the six input neurons and one output neuron, is given by Figure 27.

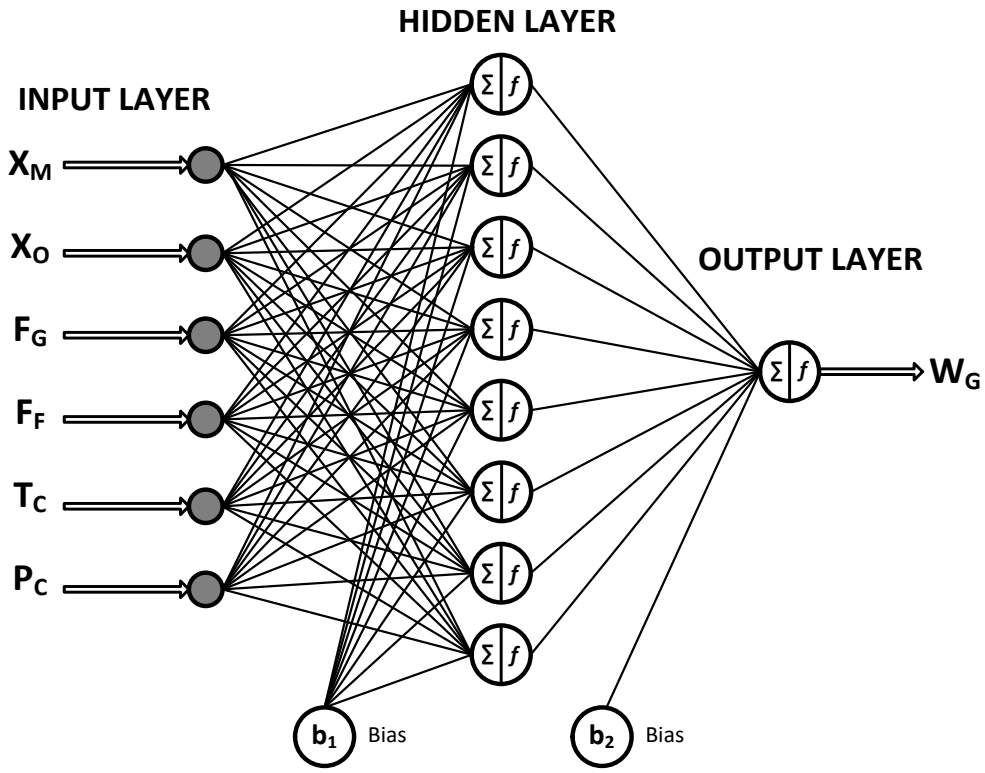


Figure 27: Generic schematic diagram of the neural network with six input neurons and one output neuron

The development of the model comprises determining the best performing architecture and training algorithm. To find the network that exhibits the best performance, a trial-and-error search-based method, also known as a sensitivity analysis, was employed. The sensitivity analysis method presented by Jiang *et al.* was followed [93]. The sensitivity analysis method entails iteratively training neural networks, each iteration with a different combination of characteristics for all the predetermined combinations.

As mentioned previously, networks with different initial weights and biases, albeit same architecture and training algorithm, will produce different results. Thus, each combination of network architecture and training algorithm was trained repeatedly, each repetition with different randomly assigned initial weights and biases. The performance of each combination was saved so that the best performing network could be identified and implemented. Hence, the best global performance of the repetitions could be selected from the repetitions' local performances.

The three neural network model characteristics this study focused on to determine the best performing network was: number of hidden neurons, type of training algorithm employed and number of hidden layers. The method to determine the best performing network for each of these three characteristics follows below.

2.4.1. Number of hidden neurons

The purpose of determining the best performing number of hidden neurons is to prevent a predictive model from overfitting or underfitting the data, as discussed in the *Rules of thumb: Number of hidden layers* section (page 22). From this section, the number of hidden neurons that exhibits a minimum error on the validation subset indicates the best performing respective network, as illustrated by Figure 15 (page 23).

The performance indicators selected from literature [70] – used to determine the best performing number of hidden neurons – was the root mean square error (RMSE), mean absolute percentage error (MAPE) and coefficient of determination (R^2). These performance indicators are the most popular measures of prediction error and accuracy for ANNs [94]. The RMSE, MAPE and R^2 values are given by the following equations:

$$RMSE = \sqrt{\frac{1}{N} \cdot \sum_{j=1}^{N} (t_j - o_j)^2} \quad 34$$

$$MAPE = \frac{1}{N} \cdot \sum_{j=1}^{N} \left| \frac{t_j - o_j}{t_j} \cdot 100 \right| \quad 35$$

$$R^2 = 1 - \left(\frac{\sum_{j=1}^{N} (t_j - o_j)^2}{\sum_{j=1}^{N} (o_j)^2} \right) \quad 36$$

where o_j is the predicted output and t_j is the target for each data point j, for all data points from $j = 1$ to N.

The RMSE and MAPE performance indicators quantify the accuracy of the networks' predictions relative to their targets [95]. The RMSE gives more emphasis to large errors, in contrast to the MAPE which does not give any emphasis to large errors [94]. The R^2 value indicates the variance of the network's predictions on a scale from 0 to 1 [96]. The number of hidden neurons with the lowest (minimum) RMSE and MAPE signifies the best performing neural network with the best prediction accuracy, whilst the highest (maximum) R^2 value indicates the best performing network with the least variance between the model's outputs and targets.

To determine the maximum R^2 value and minimum RMSE and MAPE, multiple networks were trained – as mentioned previously – each with a different number of hidden neurons over a pre-selected range. For a one hidden layer network, the number of hidden neurons was varied between 1 and 200, trained with the LM and SCG algorithms. For two hidden layers, due to its computational complexity, the number of hidden neurons was varied between 5 and 40 in both layers, trained using the LM and SCG algorithms.

For a single hidden layer network, the performance indicators (RMSE and R^2 values) were plotted on separate vertical axes versus the number of hidden neurons on the horizontal axis for all the subsets. From the plot, the number of hidden neurons at which the network exhibits

a minimum validation RMSE and maximum validation R^2 value could be identified – indicating the best performing network architecture. This is demonstrated by Figure 28.

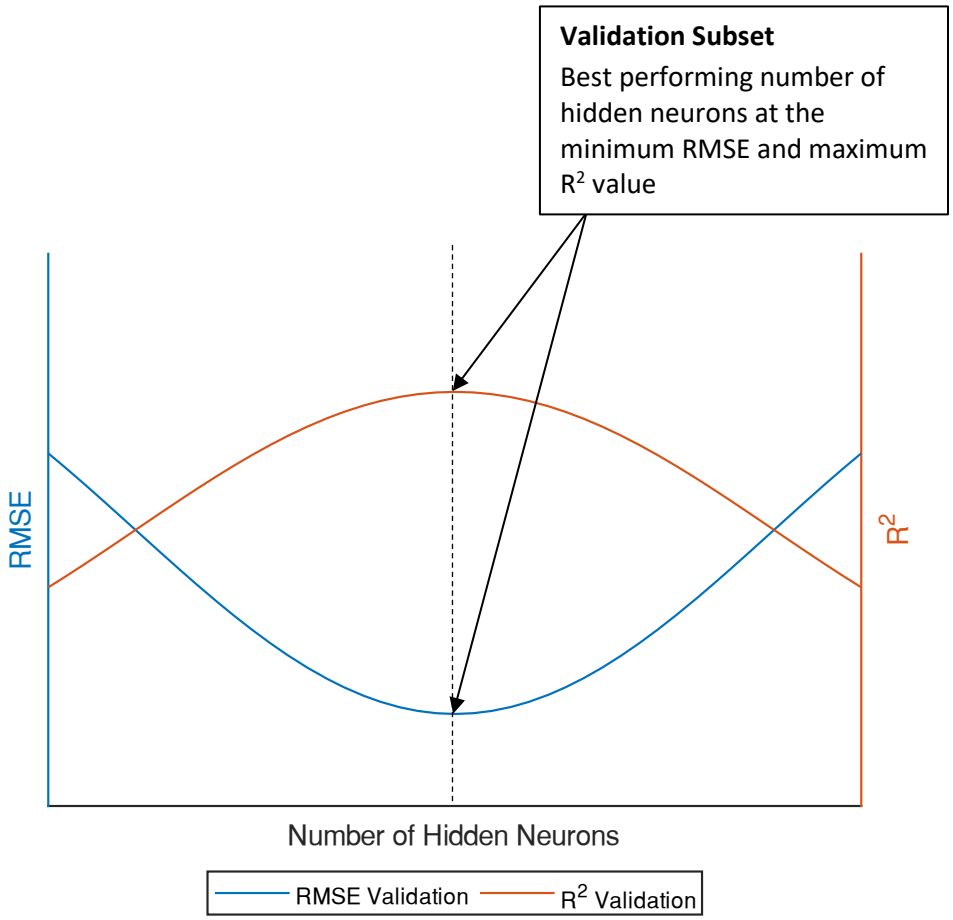


Figure 28: Example of plot used to determine the best performing number of hidden neurons for a single hidden layer using the validation subset's RMSE and R^2 values

For two hidden layers, the performance indicators (RMSE and R^2 values) were plotted separately and only for the validation subset for simplicity purposes. The number of hidden neurons in the first hidden layer was plotted on the x-axis, number of hidden neurons in the second hidden layer on the y-axis and the performance indicator on the z-axis. From the plots, the number of hidden neurons in the first and second hidden layers at which the network exhibits the minimum RMSE and maximum R^2 value for the validation subset could be identified – indicating the best performing network architecture. This is demonstrated by Figure 29 and Figure 30 for the RMSE and R^2 values, respectively. Number of hidden neurons is acronymised as HNs and hidden layer as HL.

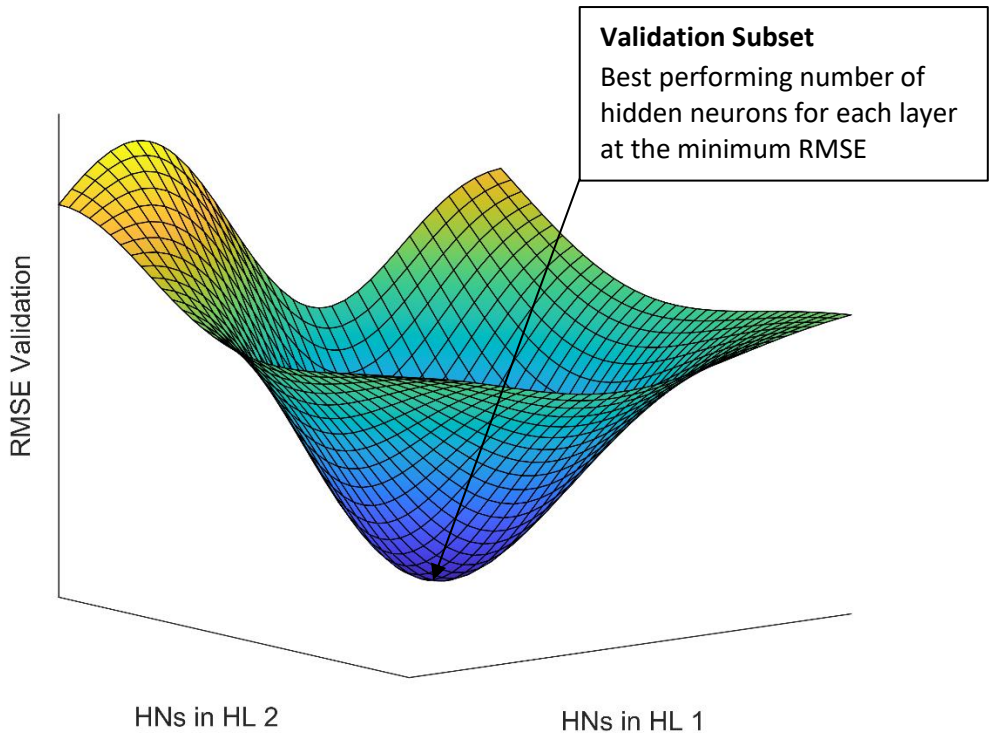


Figure 29: Example of plot used to determine the best performing number of hidden neurons in the first and second hidden layers using the validation subset's RMSE

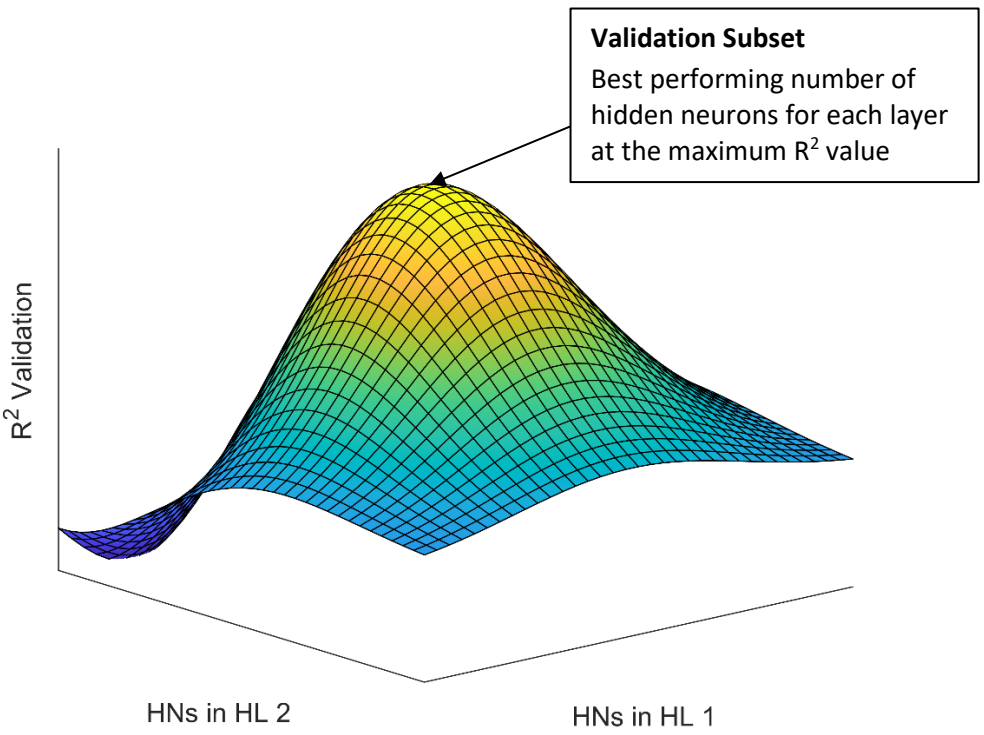


Figure 30: Example of plot used to determine the best performing number of hidden neurons in the first and second hidden layers using the validation subset's R^2 value

2.4.2. Training algorithm

Different training algorithms implement different methods to adjust the weights and biases of a network during the training process, as elaborated upon in the section *Training: Training algorithms* (page 15). As such, the weights and biases that the training algorithm converges to will differ depending on the algorithm. Henceforth, the best performing training algorithm was to be determined for single- and double-layer networks by training the networks with each algorithm and evaluating their performances.

The network was trained repeatedly – as previously discussed – to determine and select the network with the best global performance (minimum error) of all the repetitions from the repetitions' local performances (minimum errors). Furthermore, the results obtained in the previous section, namely the best performing number of hidden neurons, were implemented for both training algorithms.

For this section, the mean square error (MSE) was used to evaluate the networks' performances and determine the best performing network. The MSE was selected as it is the primary parameter used in MATLAB to evaluate network training performances. MSE is given by the following equation:

$$MSE = (RMSE)^2 = \frac{1}{N} \cdot \sum_{j=1}^{N} (t_j - o_j)^2 \qquad 37$$

where o_j is the predicted output and t_j is the target for each data point j, for all data points from $j = 1$ to N.

The MSE of the training, validation and testing subsets were plotted versus the number of epochs required by the LM and SCG algorithms to train the networks. The plots follow the trend given by Figure 13 in the *Training: Generalisation* section (page 20), where the validation error (MSE) attained a minimum which indicates the best performing weights and biases, as demonstrated by Figure 31. Prior to the minimum validation MSE, the network underfits the data. After reaching the minimum validation MSE, the error will increase again as the network overfits the data.

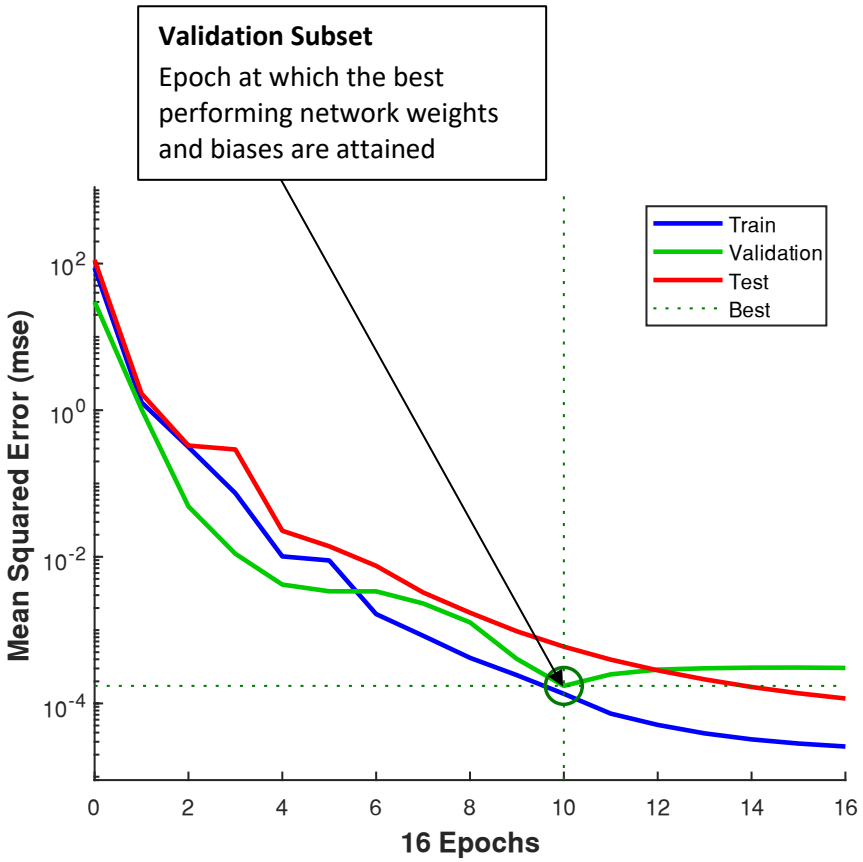


Figure 31: Example of the plot used to determine the network with the best performing weights and biases during the training process

From the minimum MSEs for each training algorithm over all the repetitions, the best performing training algorithm was selected for both single- and double-layered networks. Hence, the best training algorithm for the model could be determined. Furthermore, the number of epochs required to attain the minimum MSE was identified to compare the convergence rates between the different training algorithms.

2.4.3. Number of hidden layers

After selecting the best performing number of hidden neurons and training algorithm, the best number of hidden layers could be determined. From the section *Rules of thumb: Number of hidden layers* (page 21), the number of hidden layers employed may affect the convergence and accuracy of the network. The best number of hidden layers for the network was determined using the results obtained in the previous section, hence the MSEs obtained for

the respective number of hidden layers were evaluated and the best performing number of hidden layers was selected.

From the steps in the model development section, the best performing model could be determined. Thereafter, the model's performance was evaluated and if the performance was satisfactory, the next model implementation steps were followed. However, if the network's performance was unsatisfactory, the noise filtering parameters were re-adjusted and the model development steps were repeated until the performance was satisfactory.

2.5. Model implementation

After developing the model that best represents the SI engine system, the model was implemented to achieve the respective objectives of this study. The implementation of the developed model entailed assessing its prediction performance, developing the corresponding mathematical equations, evaluating these equations' relationships and using the equation's evaluation to improve the performance of the engine.

The method of implementing the developed model is presented below:

2.5.1. Prediction performance

To evaluate the model's prediction performance on the main data set, the model's output (predictions) was plotted versus its targets, as demonstrated by Figure 32. From the plot, the equation of the fit between the models output and targets for each data subset (training, validation and testing) could be determined. For an ideal fit, the model's output (Y) is equal to its target (T) for each data point, therefore the equation of an ideal fit is Y = T. An ideal fit has $R^2 = 1$ and is shown by the dashed line in Figure 32. The prediction performance could thus be evaluated by comparing the model's fit equation with the ideal fit equation.

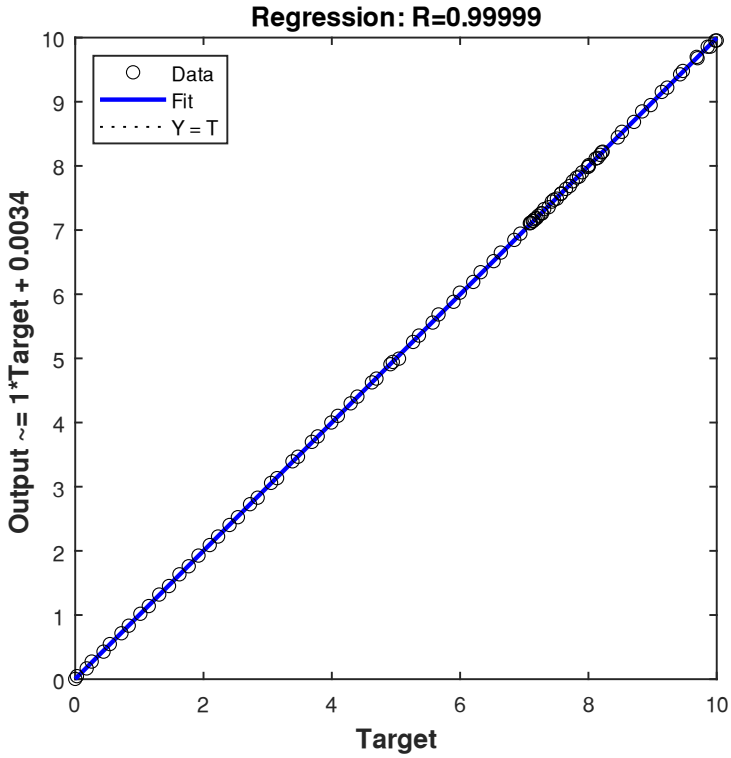


Figure 32: Example of the plot used to evaluate the model's prediction performance

2.5.2. Mathematical equations

The trained network can be represented by the fundamental summation and activation equations on which neural networks are based. These fundamental equations are discussed in the section *Architecture: Perceptron* (page 7) and the summation and activation functions are given by Equation 1 and Equation 2, respectively. The logistic sigmoid function was selected as transfer function for the hidden and output neurons. The combination of Equation 1 and Equation 2 yields the full input-output equation for the logistic sigmoid transfer function, namely Equation 9.

To develop the mathematical equations, each hidden layer and the output layer of the trained network are expressed in the form of Equation 9. The weights and biases determined during the training phase of the network were used. Predictions can be made by implementing these equations using the input parameter values. For the first hidden layer, the input parameters' values needed to be normalised, as discussed in the section *Data processing: Data normalisation* (page 46). Thereafter, equations for each hidden layer and for the output layer could be developed.

2.5.3. Engine operation evaluation

The developed mathematical equations were implemented to investigate the relationship between the operational input parameters and the power generated by the engine by keeping all inputs except one (or two) constant and plotting the effect of that operational parameter on the power generated. From the evaluation of the mathematical equations, the best operating conditions could be determined.

The relationships between the following input parameters and power generated by the engine were investigated:

- Flow rate to the engine
- Methane concentration
- Flow rate to the engine and methane concentration
- Coolant temperature
- Coolant pressure
- Coolant temperature and coolant pressure

2.5.4. Engine performance improvement

The model can be used to determine the power generated by the engine at the most favourable operating input conditions, as determined in the previous section. Methane and oxygen compositions were set to their average daily values to introduce the variability of the methane supply source. The flow rate to the generator, flow rate to the flare, coolant temperature and coolant pressure were set according to the best operating conditions. The engine only operates on weekdays, from 6:00 to 22:00.

A baseline performance (for weekdays) and the model's prediction at the favourable input conditions (as determined previously) were compared by plotting them on the same graph. The power generated was integrated with respect to time to obtain the energy generated (in kWh) by the engine over the period. The MATLAB function *trapz* was used to perform the integration. The energy generated was related to an energy cost value by implementing the time-of-use (TOU) tariffs for 2022. TOU entails having different energy tariffs during different periods of the day. The three TOU periods are the Peak, Standard and Off-peak.

2.6. Model verification

The model was verified to confirm whether it had been correctly implemented in solving the problem identified by this study. The model was verified by evaluating its accuracy and robustness on the independent and unseen verification data set. Model verification is important as it verifies that the results obtained from its development and implementation are correct.

To evaluate the model's accuracy and robustness on the verification data set, the model's output (predictions) was plotted versus its targets, as done in section *Model implementation: Prediction accuracy* (page 70). The R^2 value was also computed and compared to the ideal fit of $R^2 = 1$. From the figure, the model's ability to accurately predict the power generated by the engine for different power outputs could be evaluated.

If the model's prediction accuracy on the verification data set was sufficient, the model was deemed acceptable and thereby confirmed that the objectives of the study have been achieved. Conversely, if the model's prediction accuracy was unacceptable, the main data set was improved and the subsequent steps repeated. In other words, the data processing, model development and model implementation steps were all repeated after improving the main data set.

The main data set could be improved by selecting data over a different time interval, as the operation of the generator varied over time due to environmental conditions, such as wear-and-tear and maintenance. Therefore, the data set could be increased or decreased in size to better account for the methane generator's operation over a specific interval of time. Furthermore, by varying the inequality filters, more variant operational conditions could be removed, improving the data set's quality.

2.7. Summary

The methodology for developing a model that predicts the power generated by a SI engine using neural networks is presented in this chapter. The method comprises the investigation of the engine system, followed by the processing of the collected data and the development of the model. Thereafter, the model was implemented to evaluate its prediction accuracy,

develop mathematical equations, evaluate these equations and use the evaluations to improve the engine's performance. Finally, the model was assessed to verify and confirm that it had been correctly implemented as a solution to the problem posed by this study.

Chapter 3: Implementation and results

Figure 33: Methane flare and heat exchanger piping and instrumentation for an SI engine at a deep-level mine[3]

"We are what we repeatedly do. Excellence, then, is not an act, but a habit."

– William J. Durant, The Story of Philosophy (based on Aristotle)

[3] J Pretorius, Personal photograph, Welkom, 2020

3. IMPLEMENTATION AND RESULTS

3.1. Preamble

The results and discussion of the model's development, implementation and verification are provided in this chapter, according to the methodology presented in the previous chapter. The results from the determination and development of the best performing model of the engine is detailed. The development of the model comprises determining the model characteristics that has the highest prediction accuracy. The characteristics investigated were the number of hidden neurons, number of hidden layers and training algorithm.

After determining the best performing model characteristics, the model was implemented to evaluate its performance. The model's accuracy was assessed by comparing its outputs with its targets. Mathematical equations representing the model were developed and the relationship between the power generated and input parameters was investigated. From the investigation, the best performing operating conditions that result in the best engine performance was identified. Lastly, the model was verified with unseen and independent data to test its robustness and applicability in modelling the engine.

3.2. Data processing

The noise and inequality filters were applied to the raw operational and power data. The noise filters adequately damped the sensor data. After trial and error, the maximum allowable change (Δy) and dimensionless parameter (τ_F) values were determined for each variable. The results of the raw and filtered training data for the feed flow rate to the flare is given by Figure 34. The figure indicates the effect of filtering on the raw data. The filtered outputs for the remaining variables are given by Figure 56 to Figure 61 in Appendix *B1: Noise filtering* (page 97).

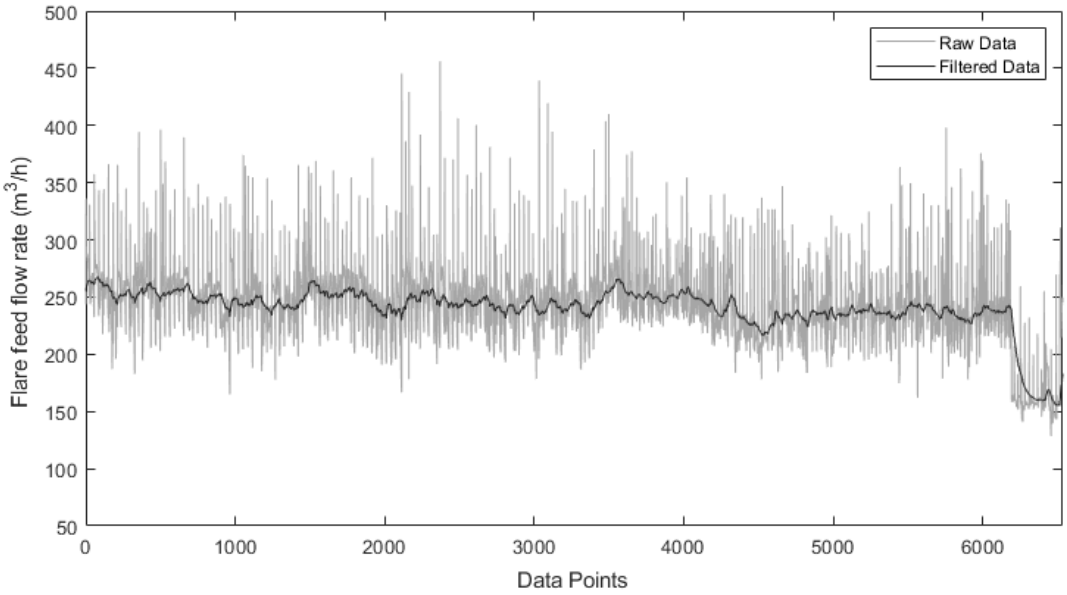


Figure 34: Filtered and raw feed flow rate to the flare

3.3. Model development

The best performing ANN model was determined using the sensitivity analysis technique. From the results, the best performing network architecture and training algorithm could be identified. The sensitivity analysis technique was performed on networks with one and two hidden layers, each trained using either the LM or SCG algorithms. The RMSE and MAPE of the validation set were used to determine the best performing network architecture.

3.3.1. Number of hidden neurons

The first network characteristic that was evaluated was the best performing number of hidden neurons for single- and double-layered networks, trained with the LM and SCG algorithms.

One hidden layer

For one hidden layer, the network was repetitively trained for architectures varying from 1 to 200 hidden neurons.

Levenberg-Marquardt algorithm

The RMSE and R^2 versus number of hidden neurons for a one hidden layer network, trained with the LM algorithm, is given by Figure 35. In the figure, the RMSE (blue) and R^2 value (orange) were plotted versus the number of hidden neurons in the network. As seen from literature, a minimum prediction error (RMSE) was observed on the validation set that indicates the best performing network architecture (number of hidden neurons).

The validation set's RMSE reached a minimum and its R^2 reached a maximum at 20 hidden neurons. This indicates that the best performing network architecture modelling the methane SI engine is 20 hidden neurons for a single layer network trained using the LM algorithm. At 20 hidden neurons, the validation set's RMSE, R^2 and MAPE were 19.86, 0.9798 and 1.571, respectively.

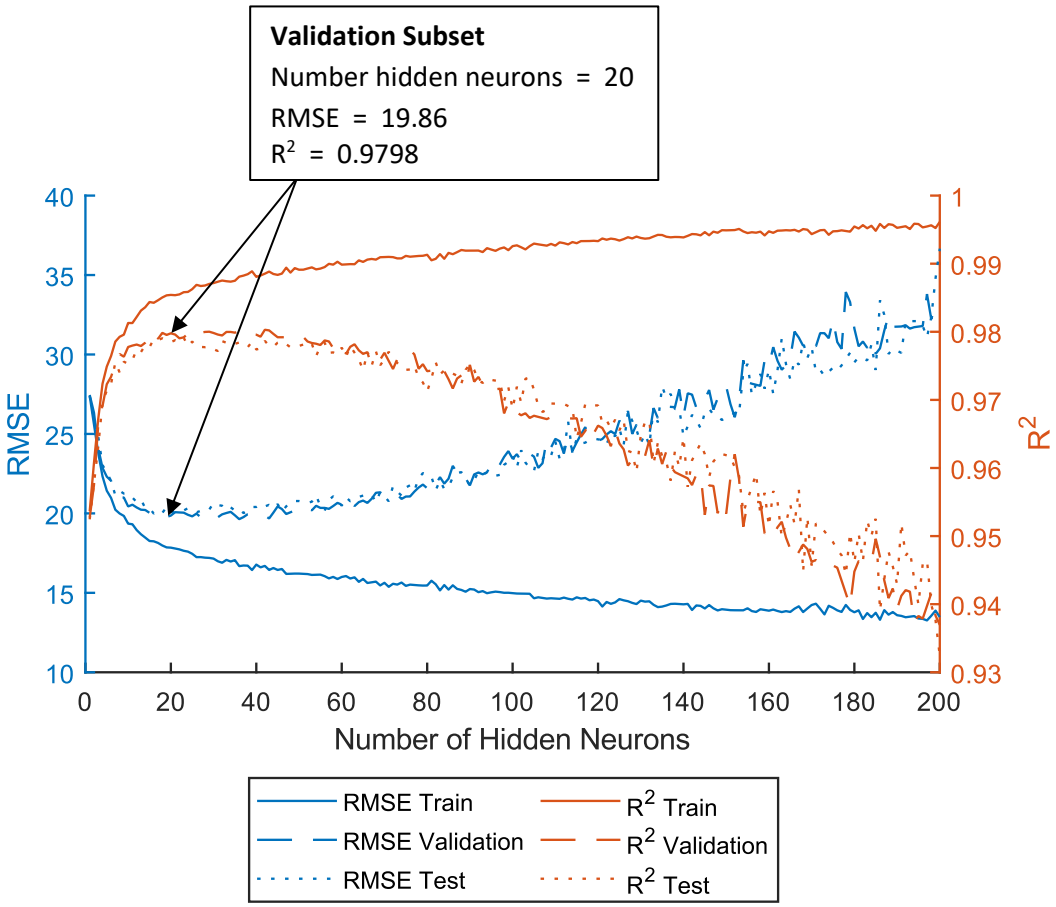


Figure 35: Performance of a one hidden layer network trained with the LM algorithm

Scaled Conjugate Gradient algorithm

The RMSE and R^2 versus number of hidden neurons for a one hidden layer network, trained with the SCG algorithm, is given by Figure 36. In the figure, the RMSE (blue) and R^2 value (orange) were plotted versus the number of hidden neurons in the network. Similar to the previous section, the prediction error (RMSE) on the validation set exhibited a minimum, indicating the best performing network architecture (number of hidden neurons).

The validation set attained a minimum RMSE of 27.45 and maximum R^2 of 0.9443 for an architecture of one hidden layer with 20 hidden neurons. This confirms the results of the previous section, signifying that the best network architecture for the methane SI engine model was 20 hidden neurons, regardless of the training algorithm. The MAPE of the validation set for the best performing network was 2.213.

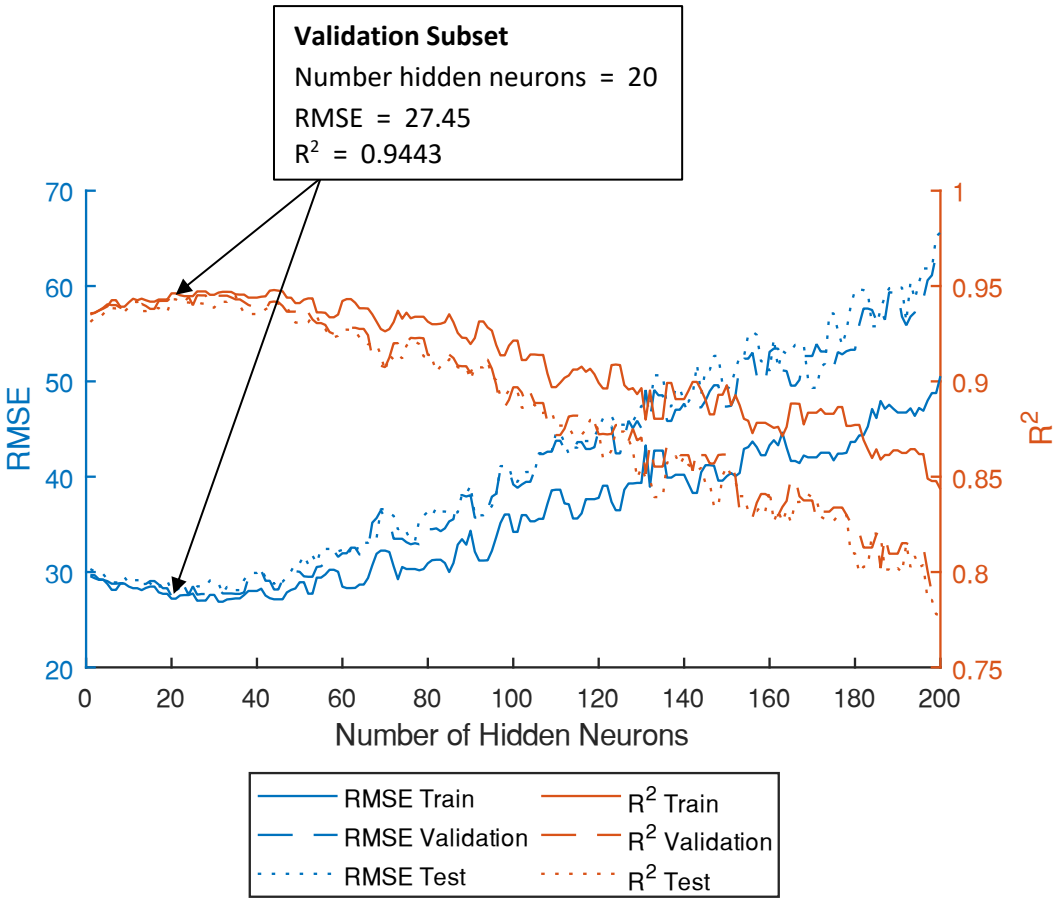


Figure 36: Performance of a one hidden layer network trained using the SCG algorithm

Two hidden layers

For two hidden layers, the network was repetitively trained for architectures varying from 5 to 40 hidden neurons in both hidden layers.

Levenberg-Marquardt algorithm

The RMSE and R^2 values versus the number hidden neurons in hidden layer one and in hidden layer two, trained using the LM algorithm, are given by Figure 37 and Figure 38, respectively. In Figure 37 and Figure 38, the RMSE and R^2 value versus the number of hidden neurons in hidden layer 1 and hidden layer 2 were plotted. A minimum prediction error (RMSE) and maximum R^2 value were obtained for the validation set, indicating the best performing network architecture (number of hidden neurons). The validation set reached a minimum RMSE of 20.27 and maximum R^2 value of 0.9707 at 14 neurons in hidden layer one and 11 neurons in hidden layer two. Hence, the best network architecture for predicting the power of a methane-fuelled SI engine was 14-11, with a MAPE of 1.523 on the validation set.

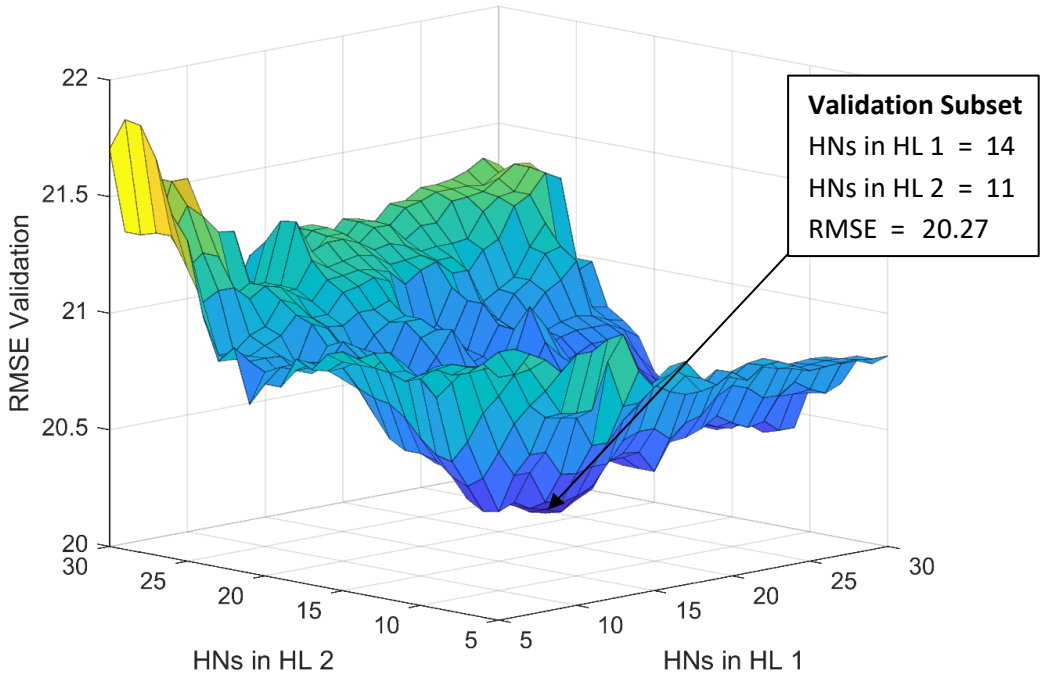


Figure 37: RMSE performance of the validation set for a two hidden layer network trained using the LM algorithm

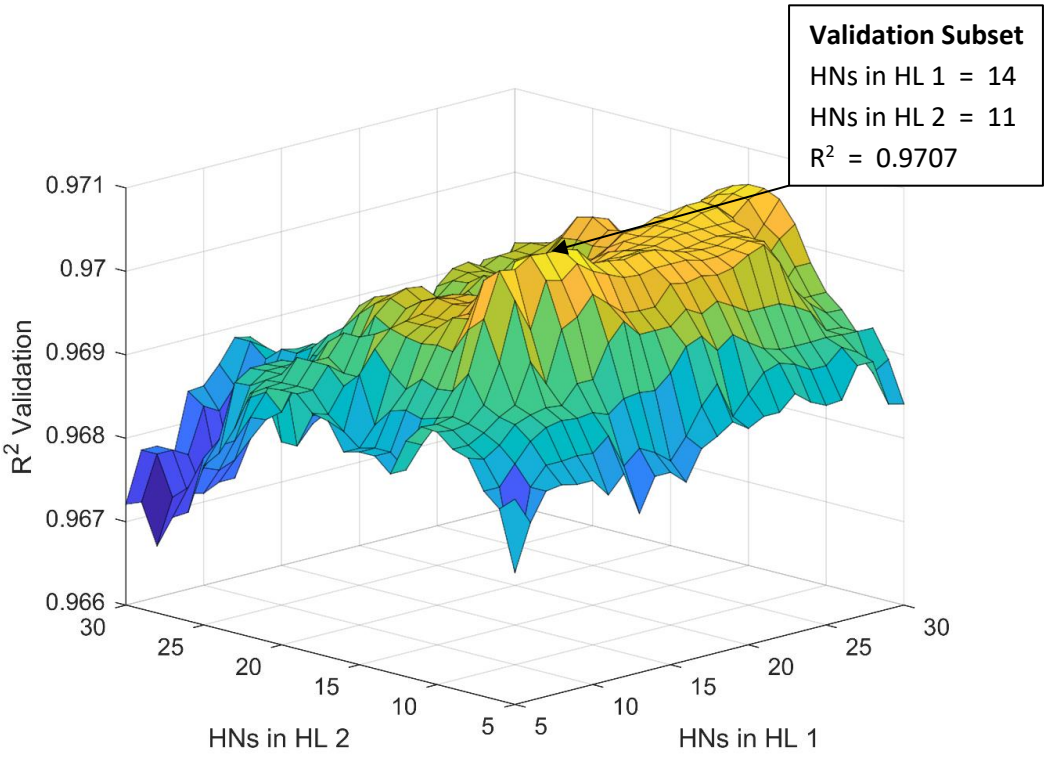


Figure 38: R^2 performance of the validation set for a two hidden layer network trained using the LM algorithm

Scaled Conjugate Gradient algorithm

The RMSE and R^2 values versus the number of hidden neurons in both hidden layers, trained using the SCG algorithm, are given by Figure 39 and Figure 40, respectively. In Figure 39 and Figure 40, the RMSE and R^2 value versus the number of hidden neurons in hidden layer 1 and hidden layer 2 were plotted. Unlike the previously trained networks, the two hidden layer SCG-trained network exhibit neither a minimum error (RMSE) nor a maximum R^2 value. The range of hidden neurons over which the network was trained was extended, however, the results followed the same trend of not exhibiting minimums nor maximums.

The best performing RMSE and R^2 values were then taken as the best performing (minimum RMSE and maximum R^2) parameters within the bounds of between 5 and 30 hidden neurons of which the networks have been trained. The architecture 18-30 was selected as the best performing network, with RMSE, R^2 and MAPE values on the validation set being 26.93, 0.9458 and 2.129, respectively.

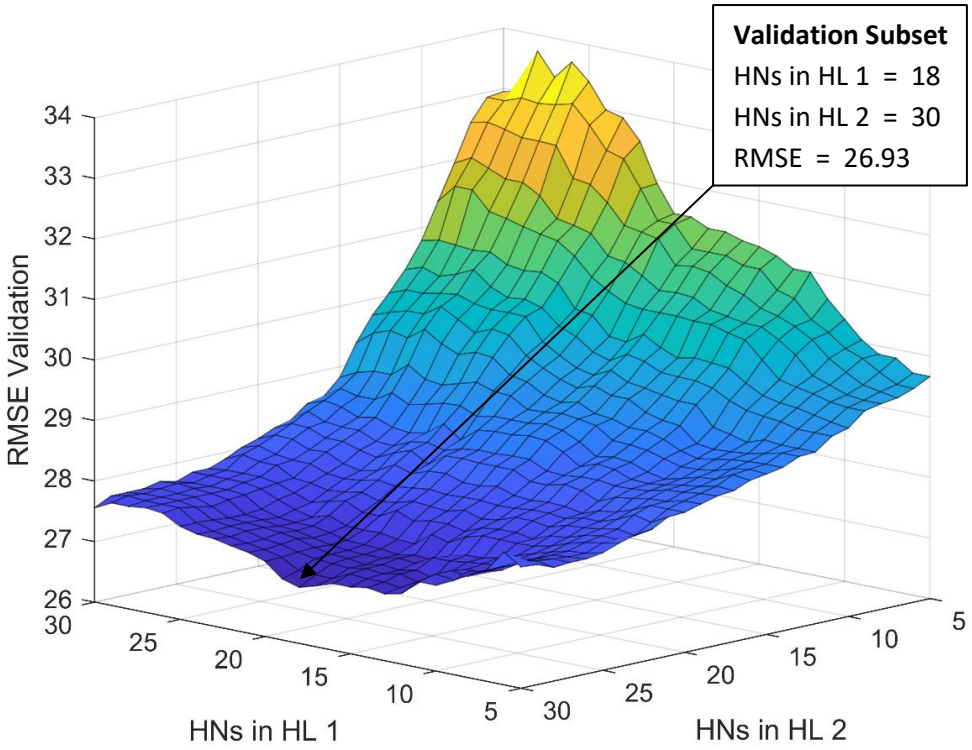


Figure 39: RMSE performance of the validation set for a two hidden layer network trained using the SCG algorithm

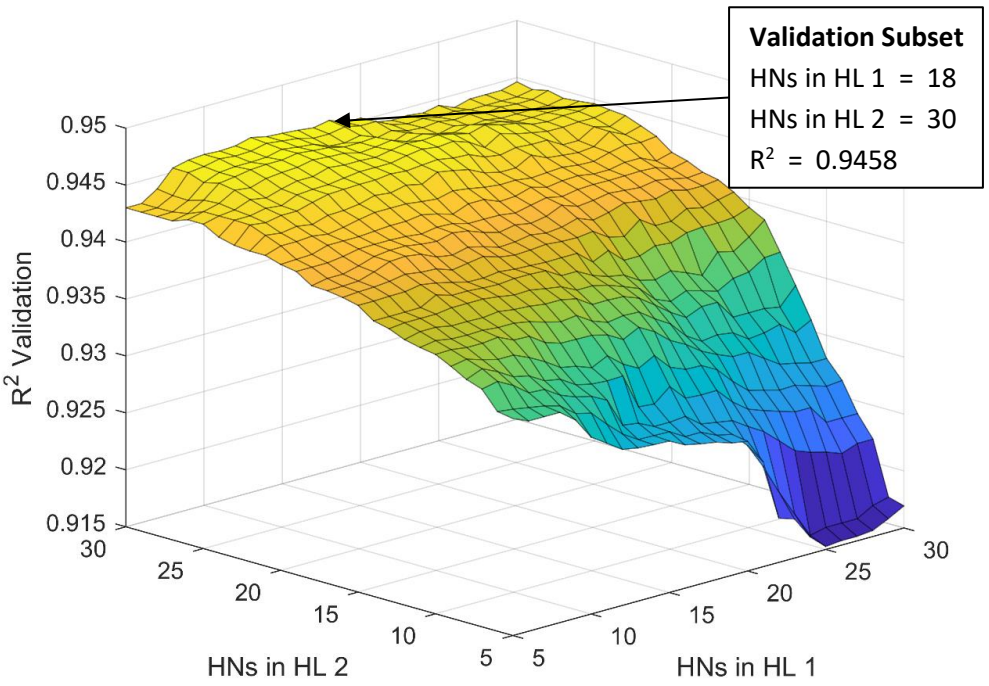


Figure 40: R^2 performance of the validation set for a two hidden layer network trained using the SCG algorithm

Summary

The best performing network for each case is summarised in Table 9. From the results, the best network architecture for a single- and double-layer network was determined to be 20 and 14-11, respectively.

Table 9: Performance parameters of the validation set for each case

	One hidden layer		Two hidden layers	
	LM	SCG	LM	SCG
Architecture	20	20	14-11	18-30
RMSE	19.86	27.45	20.27	26.93
R^2	0.9798	0.9443	0.9707	0.9458
MAPE	1.571	2.213	1.523	2.129

3.3.2. Training algorithm

To determine the best training algorithm for the model, the networks were retrained repetitively with the LM and SCG algorithms and the validation subset's performance was subsequently compared to select the best performing algorithm. For each repetition, the weights and biases were randomised to obtain the global minimum validation error (RMSE and MAPE) from the local minima over the set of repetitions.

The number of hidden neurons that have been determined in the previous section were used, with 20 hidden neurons for the single hidden layer network and 14-11 hidden neurons for the double hidden layer network. The configurations of the networks used in this section, as represented in MATLAB, are given by Figure 41 and Figure 42 for single- and double-layer networks, respectively.

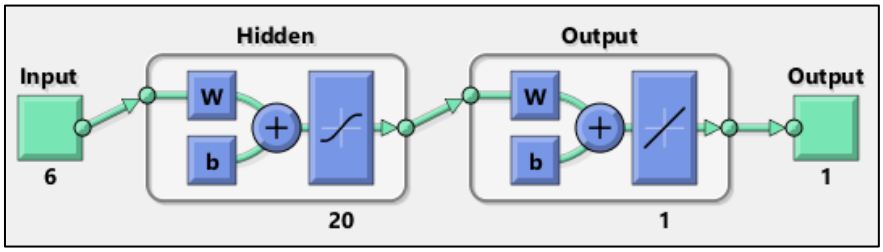


Figure 41: Configuration of an ANN in MATLAB with one hidden layer, comprising 6 input neurons, 20 hidden neurons and 1 output neuron

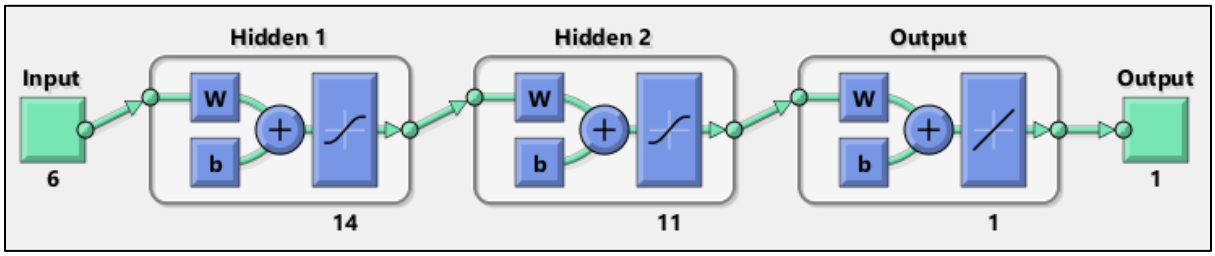
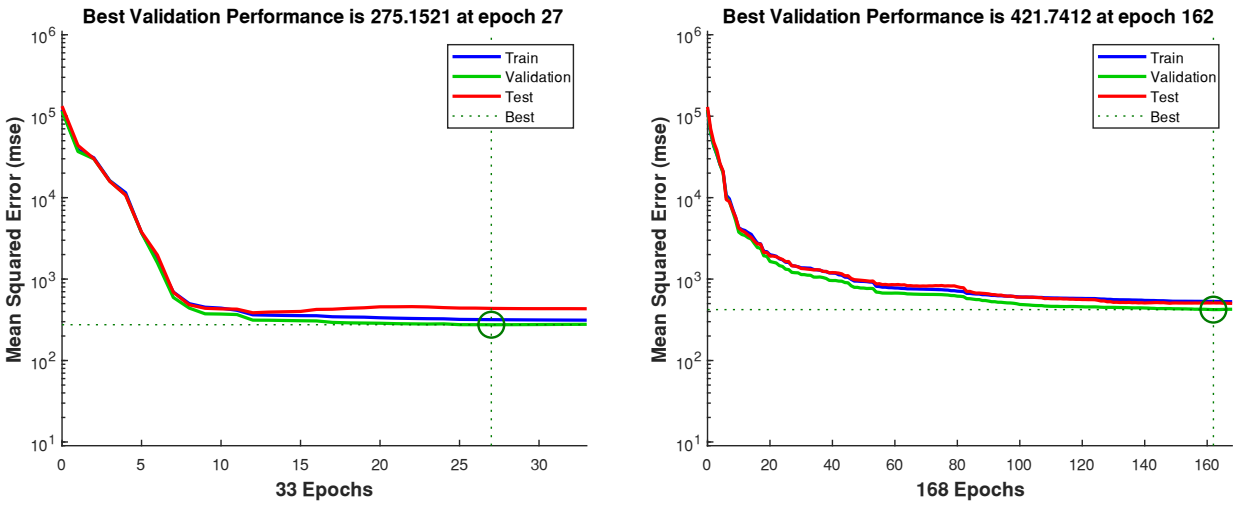


Figure 42: Configuration of an ANN in MATLAB with two hidden layers, comprising 6 input neurons, 14 hidden neurons in the first layer, 11 hidden neurons in the second layer and 1 output neuron

One hidden layer

For one hidden layer and the LM algorithm, the best validation performance (MSE) of 275.2 was achieved at epoch 27, before increasing again. The MSE versus number of epochs is given by Figure 43 (a) for the training, validation and test subsets. For the SCG algorithm, the best validation performance was achieved after 168 epochs, with a minimum MSE of 421.7, as given by Figure 43 (b). Therefore, for one hidden layer, the LM algorithm converged faster and performed better than the SCG algorithm by attaining a lower error (MSE).



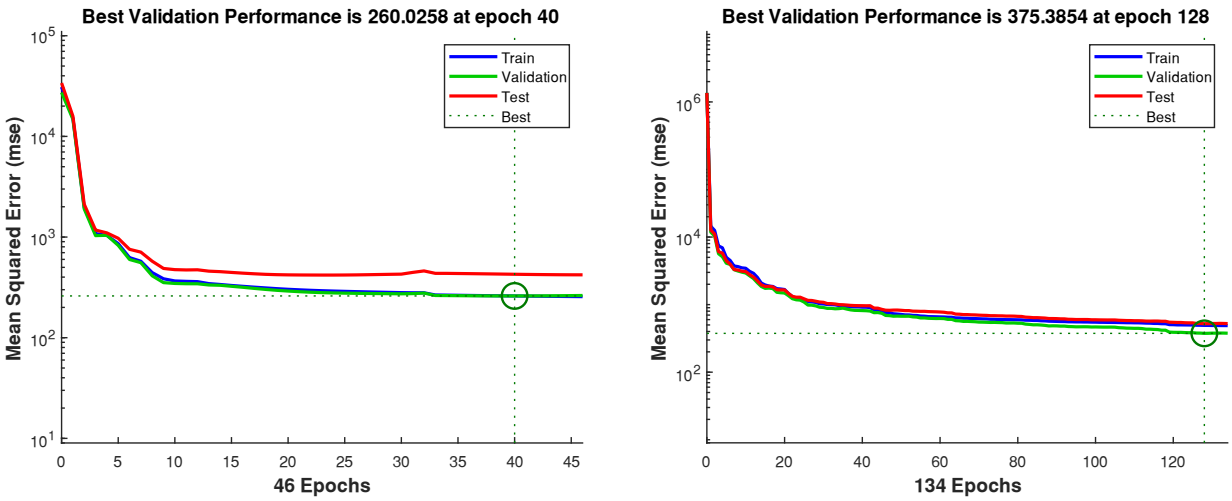
(a) LM algorithm

(b) SCG algorithm

Figure 43: Performance (MSE) of a single hidden layer network for each subset (train, validation and test) versus the number of iterations required to converge to a minimum validation error

Two hidden layers

For two hidden layers, the LM algorithm attained a minimum validation MSE of 260 at epoch 40. The MSE versus the number of epochs is given by Figure 44 (a). The SCG algorithm achieved a minimum validation MSE of 375.4 at epoch 134. Similarly, the performance progress is given by Figure 44 (b). Therefore, the LM algorithm performed better and converged to a lower MSE faster than the SCG algorithm.



(a) LM algorithm

(b) SCG algorithm

Figure 44: Performance (MSE) of a double hidden layer network for each subset (train, validation and test) versus the number of iterations required to converge to a minimum validation error

Summary

For both the single- and double-layer networks, the LM algorithm performed better and converged faster to a minimum MSE. A summary of the results presented in this section is given by Table 10.

Table 10: Performance parameters of the validation set for each case

	One hidden layer		Two hidden layers	
	LM	SCG	LM	SCG
Architecture	20	20	14-11	18-30
MSE Validation	275.2	421.7	260.0	375.4
Epochs	33	168	46	134

3.3.3. Number of hidden layers

From the results of the previous section, two hidden layers performed better than one hidden layer for both training algorithms. Nonetheless, two hidden layers took longer to converge compared to a single hidden layer. Although two hidden layers still converged to a solution in a relatively small number of epochs. For the purpose of determining the best network architecture, two hidden layers were selected.

3.4. Model implementation

The best performing neural network was determined as comprising two hidden layers, with 14 hidden neurons in the first layer and 11 hidden neurons in the second layer, trained using the LM algorithm. This network was then implemented to evaluate its prediction accuracy, develop equivalent mathematical equations, investigate the relationship between the input and output parameters and improve the engine's operation performance.

3.4.1. Prediction accuracy

The coefficient of determination (R^2) was used to evaluate the prediction accuracy of the model by plotting the model's prediction versus its targets. The correlation between the model's predictions (output) versus its targets is given by Figure 45 (a)-(d). The model's fit is given by the equations on the plots' vertical axes. Data points are represented by a circle, the model's fit by the solid line for each respective subset and the ideal fit (Y = T) by the dashed line.

As mentioned previously, a fit of $R^2 = 1$ is ideal. The training, validation and test subsets achieved R^2 values of 0.98645, 0.98498 and 0.98029, respectively. The overall performance of the data set attained 0.98548. From the results, it can be concluded that the model adequately and successfully predicted the power generated by the engine, achieving a high degree of accuracy.

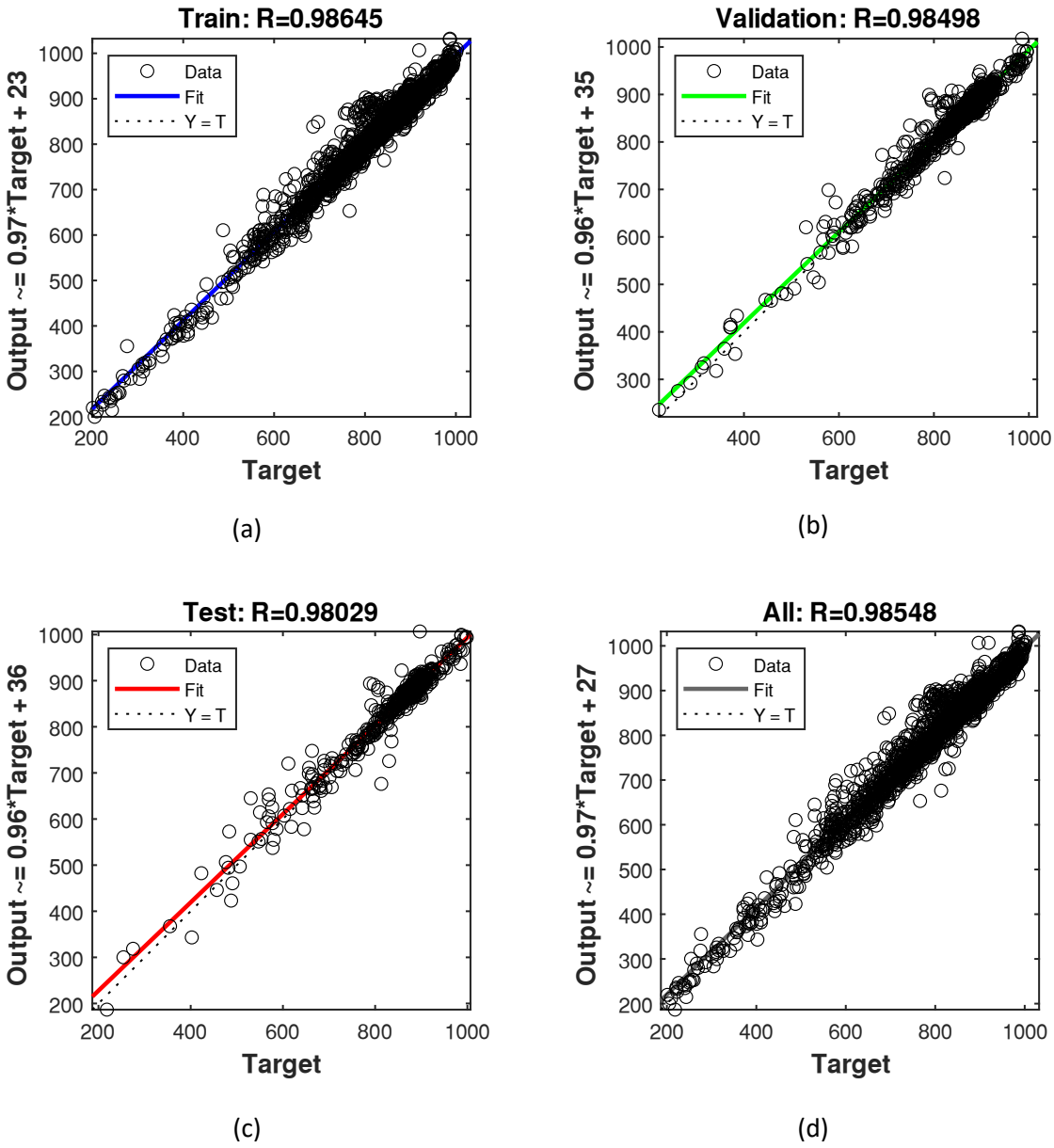


Figure 45: Coefficient of determination (R^2) plots of the model's prediction (output) versus its targets for the (a) training subset, (b) validation subset, (c) test subset and (d) main (all) data set

3.4.2. Mathematical equations

The mathematical equations to accurately predict the power generated by the SI engine, using the best performing neural network model, is detailed in this section. To calculate power, the activation functions of the first hidden layer, second hidden layer and output layers must be computed using the weights and biases obtained by training. The engine's operational parameters are required as inputs, namely: methane content (X_M), oxygen

content (X_O), feed flow to generator (F_G), feed flow to flare (F_F), coolant fluid temperature (T_C) and coolant fluid pressure (P_C).

The first layer's activation functions are represented by N_i and were calculated using Equation 38 with the weights and biases given in Table 11. The second layer's activation functions are represented by F_j and were calculated using Equation 39 with the weights and biases given in Table 12. Finally, power was calculated using Equation 40 using the weights given in Table 13.

$$N_i = \frac{1}{1 + e^{-(w_1 \times (X_M/84.4) + w_2 \times (X_O/3.2) + w_3 \times (F_G/392.6) + w_4 \times (F_F/496.7) + w_5 \times (T_C/49.7) + w_6 \times (P_C/95.6) + b_i)}} \quad 38$$

$$F_j = \frac{1}{1 + e^{-(w_1 \times N_1 + w_2 \times N_2 + w_3 \times N_3 + \cdots + w_j \times N_j + \cdots + w_{14} \times N_{14} + b_j)}} \quad 39$$

$$Power = \frac{1}{1 + e^{-(w_1 \times F_1 + w_2 \times F_2 + w_3 \times F_3 + \cdots + w_k \times F_k + \cdots + w_{11} \times F_{11} + 0.2801)}} \quad 40$$

Table 11: Weights and biases between the input layer and first hidden layer

i	w_1	w_2	w_3	w_4	w_5	w_6	b
1	0.7050	-0.8362	0.1333	-0.1626	-0.0504	2.0771	-2.6775
2	0.0726	0.1232	-0.5782	0.2425	2.0096	-0.9591	1.8157
3	-0.2300	0.5607	0.7298	0.5300	0.5507	0.3358	0.9527
4	-0.1920	-0.5165	0.9796	2.3545	2.0767	-0.1185	-2.7133
5	-0.4285	-0.1275	-1.5354	-0.5531	1.8042	0.2550	0.9128
6	-0.4664	-0.0049	1.0153	-4.0498	0.6855	0.1422	-0.4342
7	0.2953	0.5395	-1.6675	-0.2185	2.3849	0.1274	-0.7764
8	0.5580	-0.9926	-3.2959	-1.7378	-0.3659	0.0503	2.0144
9	-1.9893	-0.2939	-1.2572	-0.9251	-0.8264	-0.2421	-1.2837
10	0.3521	-0.7836	0.2674	-0.5875	0.3901	-0.3988	-0.3751
11	-2.1501	1.2030	1.1975	0.6331	0.2269	-1.3886	-1.6585
12	-0.4658	1.9625	0.6574	-1.5805	0.7712	0.6379	1.5141
13	-2.4836	-1.8363	-1.3277	-0.8634	0.6239	-0.1467	-3.1202
14	-0.0458	-0.2526	-0.7827	0.7682	1.0702	0.9702	2.2780

Table 12: Weights and biases between the first and second hidden layers

j	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	b
1	0.3532	0.7308	-0.7507	1.6936	-0.7965	0.0155	-0.6794	-0.0119	-0.2855	0.0123	0.3139	0.3521	0.4596	1.0405	2.1191
2	1.1529	0.1383	-0.2573	-0.1463	-0.1715	1.7683	0.3318	1.2084	-1.5685	0.2667	0.3340	-0.5828	-2.3901	0.0780	-1.3257
3	-0.2609	0.3747	-0.5339	0.7906	0.2224	-0.4297	-0.4160	-0.4607	0.1407	-0.0601	0.2673	-0.3303	-0.7874	0.1667	-1.8779
4	-0.2193	-0.0287	0.6097	-0.1460	0.0108	-0.4443	-0.2438	-0.0888	0.6592	-0.3270	-0.1557	0.1655	0.7274	-0.1523	0.7205
5	-0.2308	0.0649	1.1183	-0.4652	0.1749	-1.0051	-0.6343	-0.5624	2.0067	0.1592	-0.6715	0.6670	-0.1655	0.0806	-0.4418
6	0.4715	0.7916	-0.9961	-1.3461	-0.1522	1.2160	-0.0849	-0.1602	0.6152	0.6799	-1.2370	1.2217	0.3177	0.5146	0.9324
7	-0.3784	-0.6969	1.2826	-0.4858	-0.1870	0.2451	0.0709	-0.5104	-0.7378	-0.2427	-0.0807	-1.2533	0.9129	0.1902	-0.1218
8	0.4421	0.2706	-0.5555	-1.1515	0.3579	-1.9333	-0.5762	-0.5905	0.3004	-1.2535	-0.3658	0.6710	-1.0573	0.5916	1.1012
9	-0.5461	-0.3159	0.2113	-1.0632	0.4027	0.7729	0.9271	-0.9771	0.3442	-1.1361	0.8809	-1.3213	-0.8718	0.6917	1.4127
10	-0.0736	-0.2705	0.6113	-0.7339	-0.0565	-0.0427	0.1285	-1.0936	0.0533	0.8130	0.4899	0.2873	-0.8439	-0.7971	1.4377
11	0.1765	0.4475	-0.8600	1.6126	-0.7025	-0.1527	-0.4577	-2.9811	-0.0008	0.1538	-0.2025	0.0736	-1.0563	-0.9008	-2.1122

Table 13: Weights between the second hidden layer and output layer

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}
0.5689	-0.2686	-1.1452	-1.3811	0.7117	-0.3558	0.3315	-0.2867	-0.4631	0.6698	0.7288

3.4.3. Engine operation evaluation

From the mathematical equations of the best performing model, the relationship between the power generated by the engine and the input parameters could be evaluated. The power generated versus methane content supplied to the engine is given by Figure 46. The power generated by the engine increased directly proportionally to the methane content between 70 % and 100 %, with an almost linear increase between 70 % and 90 %.

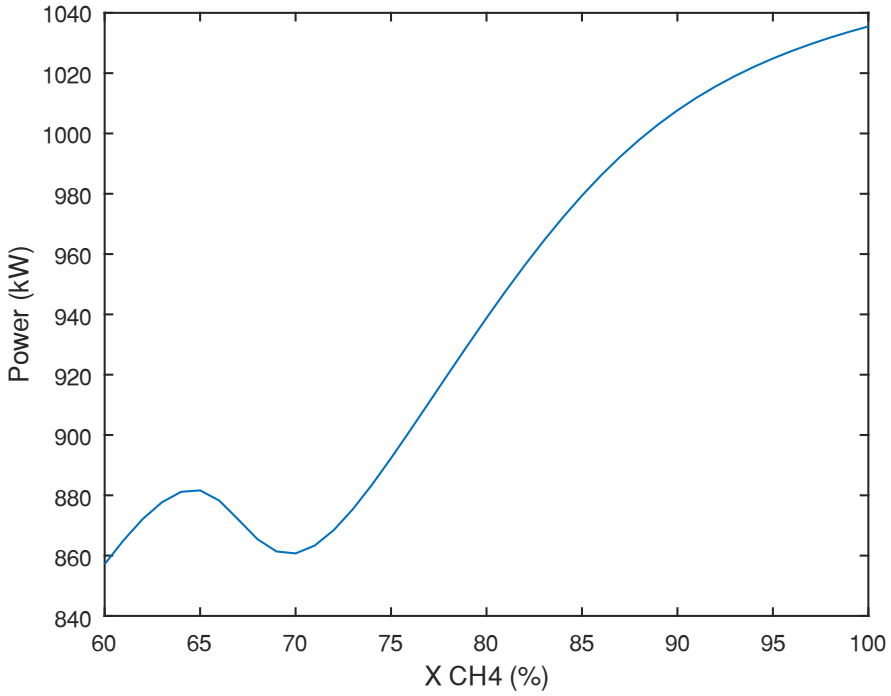


Figure 46: Relationship between the methane composition in the feed stream and power generated by the engine

The relationship between the power generated by the engine and flow rate of the feed stream is given by Figure 47. Power was directly proportional to the feed flow rate to the generator, with a linear relationship between 200 and 425 m^3/hr. The effect of methane content and feed flow rate to the generator on the power generated is given by Figure 48. Higher flow rates and higher methane composition resulted in greater power generation.

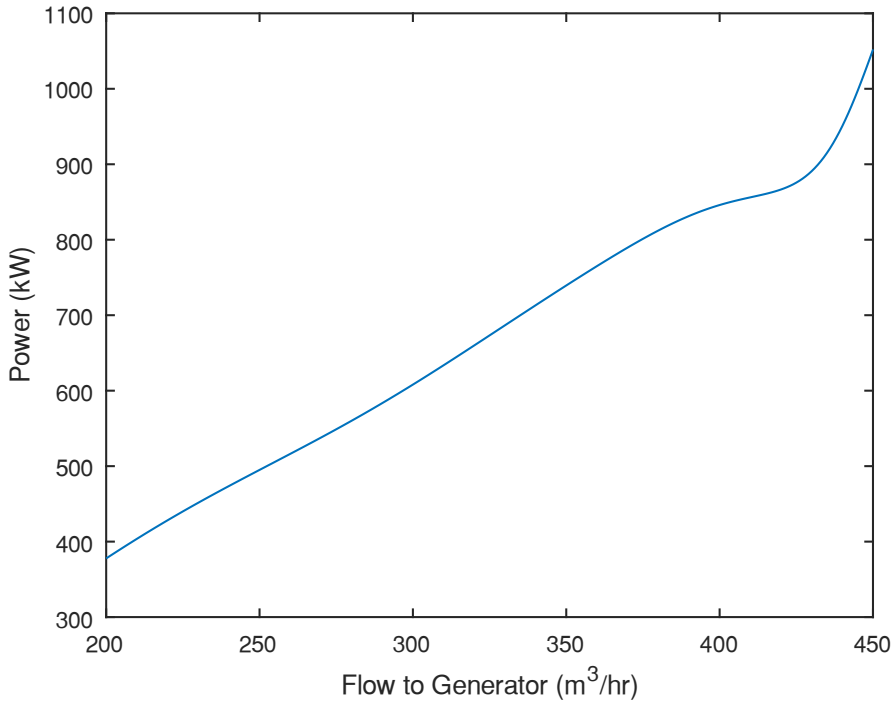


Figure 47: Relationship between the flow rate to the engine and power generated by the engine

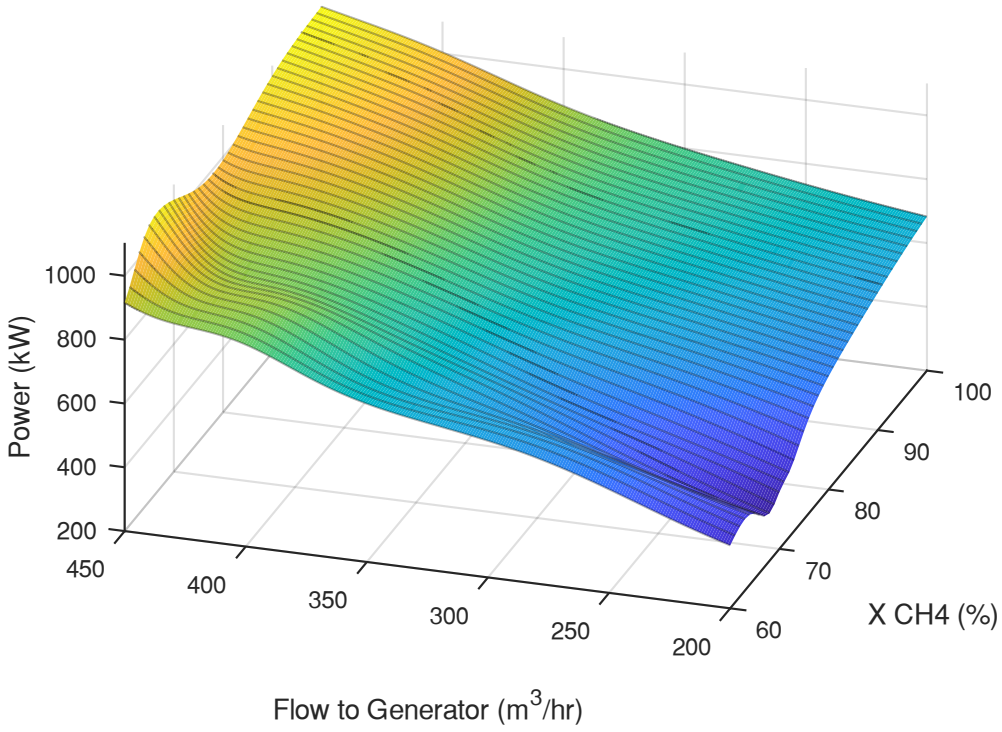


Figure 48: Relationship between the feed flow rate to the engine, methane content of the feed stream and power generated by the engine

Coolant temperature and pressure had a slight impact on the power generated by the engine. The relationship between power generated by the engine and coolant temperature and pressure is given by Figure 49 and Figure 50, respectively. The relationship of coolant temperature to power generated at different pressures is given by Figure 51. Generally, a lower temperature and higher pressure resulted in better performance by the engine, albeit to a lesser extent relative to the effect of feed flow rate and feed methane composition on power generated by the engine.

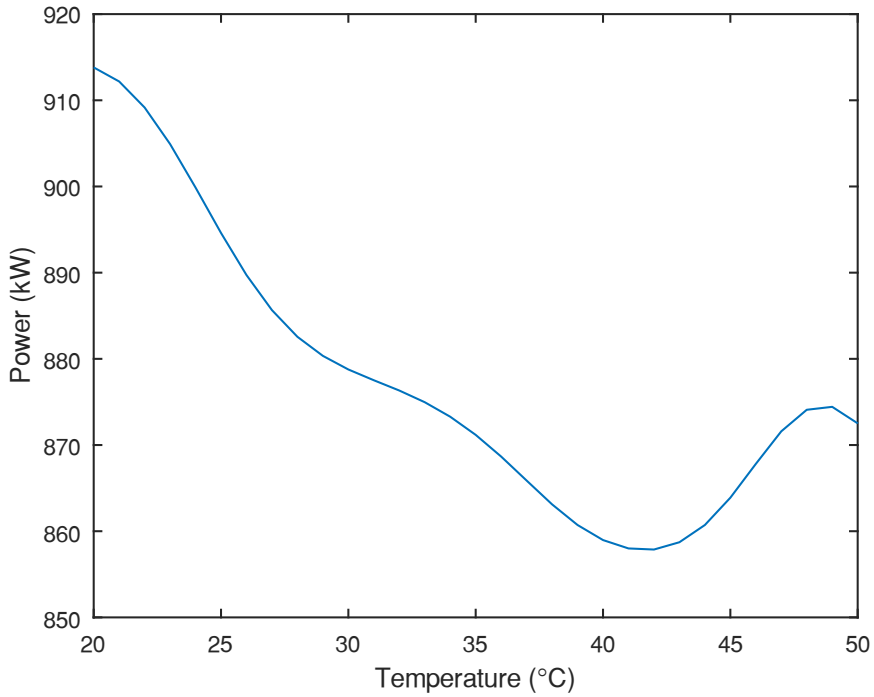


Figure 49: Relationship of coolant temperature on the power generated by the engine

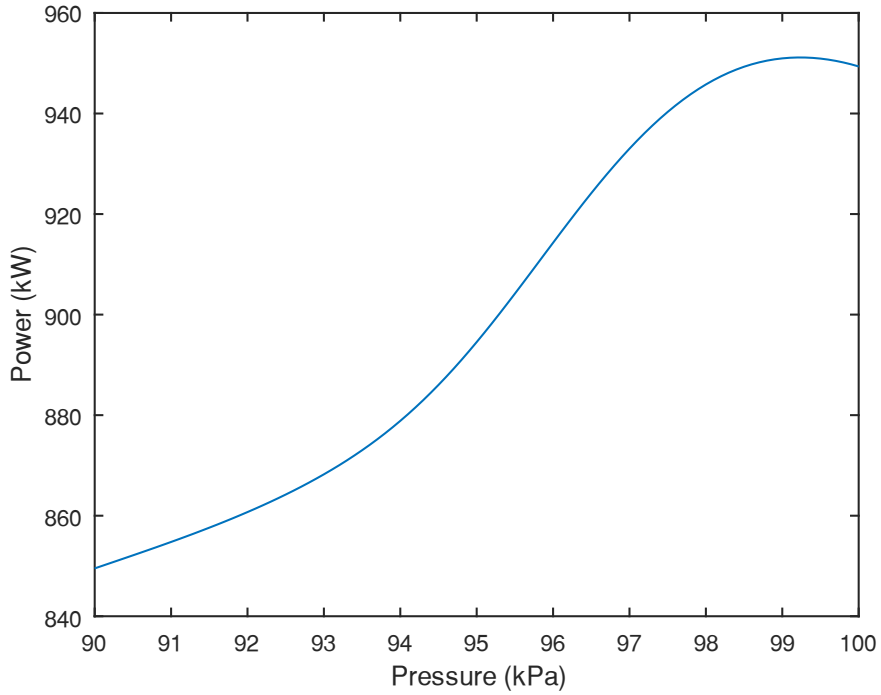


Figure 50: Relationship of coolant pressure on the power generated by the engine

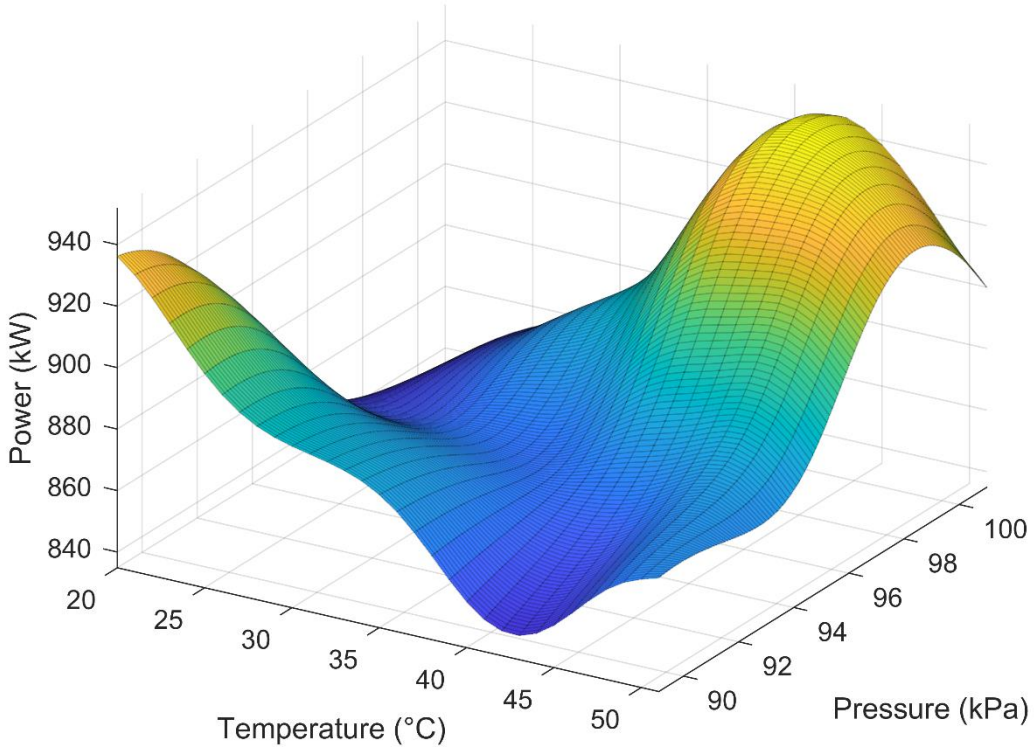


Figure 51: Relationship of coolant temperature and pressure on the power generated by the engine

3.4.4. Engine performance improvement

To investigate the improvement of the engine's performance, the model was used to predict the power generated by the engine at the best operating parameters as determined by the results of the previous section. The power generated by the engine versus time of day is given by Figure 52. The blue line represents the average baseline engine operation, whilst the orange line indicates the improved engine operation. The error bars on the improved engine performance line indicate the accuracy of the model's predictions.

From the figure, it can be concluded that there was a substantial improvement in the methane generator's performance at the determined best operating conditions. The improved operation generated 15 200 kWh per day versus the daily baseline energy generation of 9 300 kWh. This is thus an improvement of 5 900 kWh by implementing the favourable operating input conditions. This equates to an improvement of R 11 000 per day and an annual energy cost saving of R 2 860 000 with the favourable operating input conditions.

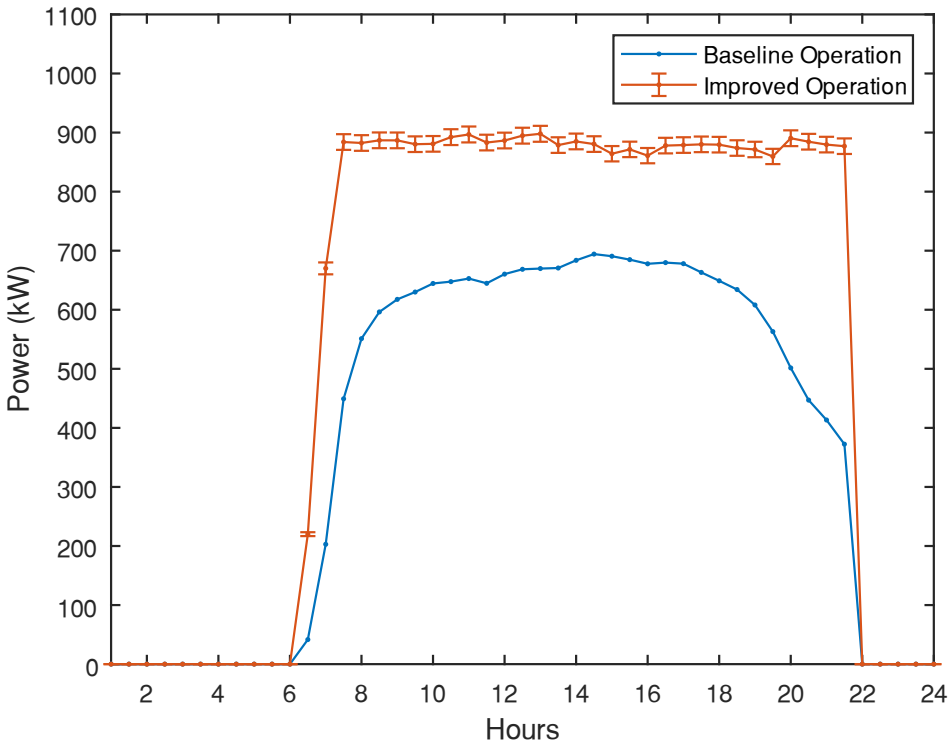


Figure 52: Average baseline operation and improved operation for the power generated by the engine versus the time of day

3.5. Model verification

For verification purposes, the model was tested on the independent verification data set to check and reaffirm that the model was correctly implemented and, furthermore, to determine its robustness as a prediction model for unseen operational conditions. The model's outputs (predictions) and its targets for the first 100 data points is given by Figure 53. The blue and orange lines represent the prediction and targets, respectively. From the figure, the model gave the expected results and accurately predicted the power generated by the engine using its operational parameters as inputs.

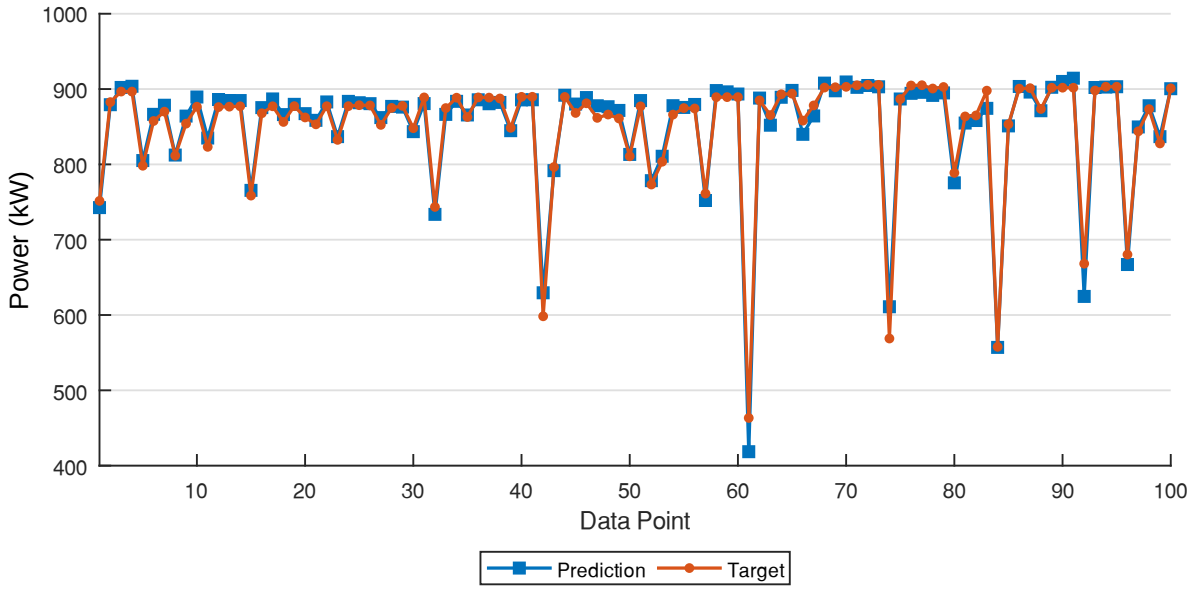


Figure 53: Model's outputs (predictions) versus its targets for the first 100 data points of the verification data set

The coefficient of determination (R^2) was implemented on the entire verification data set to evaluate the model's accuracy and robustness on high and low power outputs. The R^2 plot of the outputs (predictions) versus targets for the verification data set (569 data points) is given by Figure 54. The verification data set achieved a R^2 of 0.9873. The model's fit is given by the equation on the plots' vertical axis. The model performed very accurately between 750 and 900 kW but performed less accurately for lower power output values (especially between 500

and 700 kW). This is most likely caused by less representation of lower power output operating conditions in the network's training data subset.

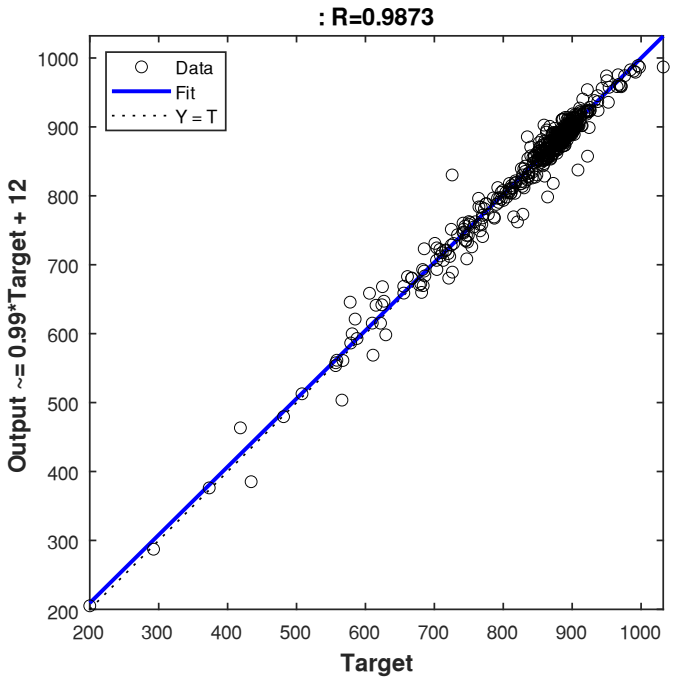


Figure 54: Coefficient of determination (R^2) plots of the model's power prediction (output) versus its targets for the verification data set

3.6. Summary

The results from the data processing, model development and model implementation are presented in this chapter. After processing the data, ANNs were trained with varying numbers of hidden neurons, numbers of hidden layers and training algorithms. The best performing ANN (i.e. lowest RMSE, MAPE and MSE) was determined to comprise two hidden layers, with 11 hidden neurons in the first layer and 14 hidden neurons in the second layer.

The developed model was implemented by evaluating its prediction accuracy. From the results, it can be concluded that the model adequately and successfully predicted the power generated by the engine, achieving a high degree of accuracy. After developing and evaluating the mathematical equations, it was determined that the best engine performance was

attained at the highest feed flow rate, highest methane concentration, lower coolant temperatures and higher coolant pressures.

From the validation of the model, it was found that an additional 5.9 MWh can be generated per day by operating at higher feed flow rates, lower coolant temperatures and higher coolant pressures. This equates to an annual energy cost saving of R 2 860 000. The model was tested on an independent and unseen data set for verification purposes and was determined to be applicable, robust and accurate.

Chapter 4: Conclusion

Figure 55: Typical deep-level mine in South Africa[4]

"It's not the big things that add up in the end; it's the hundreds, thousands, or millions of little things that separate the ordinary from the extraordinary."

— Darren Hardy, The Compound Effect

[4] J Pretorius, Personal photograph, Welkom, 2020

4. CONCLUSION

4.1. Preamble

An overview of the study's background, problem statement, formulated solution and results from implementing the solution is given in this chapter. The objectives with their accompanying results are discussed with respect to the study's aim and success in attaining a viable solution. The shortcomings of the study and recommendations for future work pertaining to this study are also discussed, followed by the concluding remarks.

4.2. Study overview

The supply of electricity in South Africa is not only unreliable, but also charged at high tariffs that increase significantly annually. This negatively impacts high energy consumers, such as deep-level mines. Energy cost saving initiatives have become increasingly important to the LOM of deep-level mines. Mines emit gases – such as methane – from underground sources that are intersected during normal mining activity.

Methane can be used as a source of energy. The case study mine has implemented an SI engine to use the methane for generating electricity. The power generated by the methane-fuelled SI engine is highly dependent on its operational parameters. These operational parameters include the feed flow rate, methane composition of the feed, coolant temperature and coolant pressure.

Currently, no model that relates the operational parameters for a methane-fuelled SI engine at a deep-level mine to the power generated by the engine exists. Such a model would prove useful to perform economic and operational optimisation on the system, especially as the amount of power generated by the engine is highly dependent on the varying operational parameters.

The aim of this study was to develop a model that predicts the power generated by a methane-fuelled SI engine, from its operational parameters, by implementing ANNs. A case study of an SI engine operating on methane at a deep-level mine was selected. The SI engine

generates an average of 280 000 kWh per month. A feed flow rate of 400 m^3/hr of methane at a concentration that varies between 60 and 90 % is fed to the engine.

From the different modelling approaches, an ANN model was selected as most appropriate to model the SI engine given the available data. ANNs learns the relationship between input and output data sets and are capable of quantifying complex and non-linear functions. The prediction performance of an ANN is dependent on its selected architecture and training algorithm. The raw data from the case study was filtered, normalised and split into a main and verification data set. The main data set was further split into three subsets, namely training, validation and testing.

For the model's development, networks were trained repeatedly and prediction performance compared to determine the best performing architecture and training algorithm. The best network architecture was determined to comprise two hidden layers, with 14 hidden neurons in the first layer and 11 hidden neurons in the second layer. The model was repeatedly trained with this architecture using the LM and SCG algorithms. The LM algorithm attained the lowest prediction error (MSE) and converged to a solution faster than the SCG algorithm.

The developed model was implemented by evaluating its prediction accuracy. The coefficient of determination (R^2) was used to evaluate the prediction accuracy of the model. The training, validation and test subsets achieved R^2 values of 0.9865, 0.9850 and 0.9803, respectively. The overall performance of the data set attained 0.9855. From the results, it can be concluded that the model adequately and successfully predicted the power generated by the engine, achieving a high degree of accuracy.

Mathematical equations were developed for the best performing network to evaluate and investigate the relationships between the operational parameters and power generated. From the evaluation of the mathematical equations, it was determined that the best engine performance was attained at the highest feed flow rate and methane concentration. It was further determined that lower coolant temperatures and higher coolant pressure favour the engine with respect to the power that it generates.

From the evaluation, the best operational parameters were implemented using the developed model to investigate to what extent the power generated by the engine could be

improved. According to the model, it was found that an additional 5.9 MWh can be generated per day by operating at higher feed flow rates, lower coolant temperatures and higher coolant pressures. This equates to an annual energy cost saving of R 2 860 000.

Lastly, the best performing model was implemented on an independent data set to test its robustness and accuracy on new unseen data. The model performed accurately on the verification data set and attained an R^2 value of 0.9873. This reaffirmed that the model had been implemented correctly and is robust and can accurately predict the power generated by the methane SI engine.

4.3. Shortcomings

The following are shortcomings in the methodology followed by this study:

- Only one case study was considered for this study. The accuracy and robustness of the model can be further improved by considering more case studies of methane-fuelled SI engines.
- The variables selected for the development and implementation of the model were limited to those available from the mine's historian. Nonetheless, additional variables not included in the model were kept constant over the specific period of which the data has been obtained.

4.4. Recommendations for future work

Recommendations for future work pertaining to this study and field are as follows:

- By implementing more case studies, a more accurate and robust model that is representative of SI engines used in different applications can be developed.
- By applying the method presented in this study to different engine sizes, with the same input and output variables, the predictive model's ability to scale to larger and smaller engines can be investigated and validated.
- The developed model can be used to perform optimisation on the methane SI engine, that is to find the optimal operating conditions using an optimisation algorithm that results in the most power generated by the engine. Furthermore, the reasons for the

resultant causal relationships for these optimal operating conditions can be investigated.

- The methodology presented in this study can be followed to develop and implement a model for any given application, provided sufficient data is available that can be used to quantify the causal relationship between the input and output variables. This is because of the ANN's versatility as a universal function approximator, as discussed in Chapter 1 (page 7). By following this method, a network can be developed with the appropriate architecture that sufficiently fits the complexity of the system to be modelled (preventing under- and overfitting).
- The best performing neural network was selected after repeatedly training the network with the same architecture to obtain the global minimum error from the local minimum errors. However, different results could be obtained by increasing the number of repetitions, although the number of repetitions used in this study was sufficient to obtain the results from which the best performing neural network was selected.

4.5. Concluding remarks

A methane SI engine is an effective method by which the energy costs of a deep-level mine can be offset and its dependence on the national power grid can be decreased, whilst it utilises and converts gases that are otherwise harmful to human health and the environment. In literature, models have been developed to model the power generated by SI engines using operational parameters such as feed flow rate and methane concentration.

There is, however, no model in literature that relates the power generated by a methane-fuelled SI engine at a deep-level mine with its operational parameters. Such a model was developed in this study and employed to assess its accuracy and robustness. The relationship between the parameters were investigated and best operating conditions implemented using the model to attain improved engine performance. The model was successful in modelling the methane-fuelled SI engine and obtained a high degree of accuracy and robustness.

5. LIST OF REFERENCES

[1] J. Wu, O. J. Abban, A. D. Boadi, M. Haris, P. Ocran, and A. A. Addo, "Exploring the relationships among CO2 emissions, urbanization, economic growth, economic structure, energy consumption, and trade along the BRI based on income classification," *Energy, Ecol. Environ.*, vol. 6, no. 3, pp. 213–231, 2021, doi: 10.1007/s40974-020-00176-0.

[2] L. Flepisi and C. Mlambo, "Factors Influencing the Late Delivery of Projects in State-Owned Enterprises: the Case of Eskom," *South African J. Ind. Eng.*, vol. 32, no. 4, pp. 57–66, 2021, doi: 10.7166/32-4-2455.

[3] H. Trollip, A. Butler, J. Burton, T. Caetano, and C. Godinho, "Energy Security in South Africa Energy Security in South Africa," *Mitig. Action Plans Scenar.*, vol. 17, p. 45, 2014.

[4] D. de Vos, "Eskom's dim understanding of the market," *Mail & Gaurdian*, 2013. http://mg.co.za/article/2013-05-03-00-eskoms-dim-understanding-of-the-market (accessed Apr. 18, 2016).

[5] H. N. Matlala, A. Marnewick, and J. H. C. Pretorius, "Energy Efficiency Through the Use of Technology in South African Industry," *IEEE Int. Conf. Power Syst. Technol.*, 2016.

[6] P. Ash, "Seven reasons why Eskom is in so much trouble," *Sunday Times*, 2021. https://www.timeslive.co.za/sunday-times-daily/news/2021-03-16-explainer--seven-reasons-why-eskom-is-in-so-much-trouble/ (accessed Mar. 01, 2022).

[7] S. Moolman, "2021 update: Eskom tariff increases vs inflation since 1988 (with projections to 2023)," 2021. https://www.poweroptimal.com/2021-update-eskom-tariff-increases-vs-inflation-since-1988/ (accessed Nov. 24, 2021).

[8] H. Khobai, G. Mugano, and P. Le Roux, "The impact of electricity price on economic growth in South Africa," *Int. J. Energy Econ. Policy*, vol. 7, no. 1, pp. 108–116, 2017.

[9] Eskom, "Eskom Integrated Report 2021." https://www.eskom.co.za/wp-content/uploads/2021/08/2021IntegratedReport.pdf (accessed Mar. 01, 2021).

[10] S. F. Koch, "Projecting the external health costs of a coal-fired power plant : The case of Kusile," *J. Energy South. Africa*, vol. 23, no. 4, pp. 52–66, 2012.

[11] F. G. Jansen van Rensburg, "Development of an integrated project sustainability model using digital mining solutions," North-West University, 2020.

[12] P. F. H. Peach, M. Kleingeld, and J. I. G. Bredenkamp, "Optimising deep-level mine refrigeration control for sustainable cost savings," *Proc. Conf. Ind. Commer. Use Energy, ICUE*, no. November, 2017, doi: 10.23919/ICUE.2017.8068000.

[13] J. J. L. Du Plessis and D. C. Van Greuning, "Destruction of underground methane at

Beatrix Gold Mine," *J. South. African Inst. Min. Metall.*, vol. 111, no. 12, pp. 887–894, 2011.

[14] J. J. L. Du Plessis and M. van der Bank, "Safe Extraction of Methane and Power Generation in a Gold Mine in South Africa," *Proc. 11th Int. Mine Vent. Congr.*, pp. 347–354, 2019, doi: 10.1007/978-981-13-1420-9_29.

[15] I. Karakurt, G. Aydin, and K. Aydiner, "Mine ventilation air methane as a sustainable energy source," *Renew. Sustain. Energy Rev.*, vol. 15, no. 2, pp. 1042–1049, 2011, doi: 10.1016/j.rser.2010.11.030.

[16] J. R. Gill, S. F. Ely, and Z. Hua, "Environmental Gas Displacement: Three Accidental Deaths in the Workplace," *Am. J. Forensic Med. Pathol.*, vol. 23, no. 1, pp. 26–30, 2002.

[17] G. Narayanan and S. O. B. Shrestha, "The performance of a spark ignition engine fueled with landfill gases," *SAE Tech. Pap.*, no. 724, 2006, doi: 10.4271/2006-01-3428.

[18] J. Wellsted, "Generating Clean Energy: Beatrix Methane Capture & Destruction Project," 2019. https://reports.sibanyestillwater.com/2019/download/SSW-FS19-beatrix-methane-project.pdf (accessed Sep. 09, 2021).

[19] W. W. Pulkrabek, *Engineering Fundamentals of the Internal Combustion Engine*, Second Edi. Essex: Pearson New International Edition, 2014.

[20] Q. Z. Al-Hamdan and M. S. Y. Ebaid, "Modeling and simulation of a gas turbine engine for power generation," *J. Eng. Gas Turbines Power*, vol. 128, no. 2, pp. 302–311, 2006, doi: 10.1115/1.2061287.

[21] P. L. Curto-Risso, A. Medina, and A. Calvo Hernández, "Optimizing the operation of a spark ignition engine: Simulation and theoretical tools," *J. Appl. Phys.*, vol. 105, no. 9, 2009, doi: 10.1063/1.3116560.

[22] R. G. Kamp and H. H. G. Savenije, "Optimising training data for ANNs with genetic algorithms," *Hydrol. Earth Syst. Sci.*, vol. 10, no. 4, pp. 603–608, 2006, doi: 10.5194/hess-10-603-2006.

[23] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018, doi: 10.1016/j.heliyon.2018.e00938.

[24] W. S. Sarle, "Neural Networks and Statistical Learning," in *SAS Users Group International Conference*, 1994, doi: 10.1007/978-1-4471-5571-3.

[25] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *J. Pharm. Biomed. Anal.*, vol. 22, no. 5, pp. 717–727, 2000, doi: 10.1016/S0731-7085(99)00272-1.

[26] O. Eyercioglu, E. Kanca, M. Pala, and E. Ozbay, "Prediction of martensite and austenite start temperatures of the Fe-based shape memory alloys by artificial neural networks,"

J. Mater. Process. Technol., vol. 200, no. 1–3, pp. 146–152, 2008, doi: 10.1016/j.jmatprotec.2007.09.085.

[27] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[28] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. New York: Springer, 2018.

[29] J. B. Caplan, K. Xu, S. Chakravarty, and K. E. Jones, "Adding a bias to vector models of association memory provides item memory for free," *J. Math. Psychol.*, vol. 97, 2020, doi: 10.1016/j.jmp.2020.102358.

[30] Y. Çay, A. Çiçek, F. Kara, and S. Sağiroğlu, "Prediction of engine performance for an alternative fuel using artificial neural network," *Appl. Therm. Eng.*, vol. 37, pp. 217–225, 2012, doi: 10.1016/j.applthermaleng.2011.11.019.

[31] T. L. Fine, *Feedforward Neural Network Methodology*. New York: Springer, 1999.

[32] H. Hazar and H. Gul, "Modeling analysis of chrome carbide (Cr3C2) coating on parts of combustion chamber of a SI engine," *Energy*, vol. 115, pp. 76–87, 2016, doi: 10.1016/j.energy.2016.08.083.

[33] K. L. Priddy and P. E. Keller, *Artificial Neural Networks: An Introduction*. New York: Spie Press, 2005.

[34] G. Montavon, W. Samek, and K. R. Müller, "Methods for interpreting and understanding deep neural networks," *Digit. Signal Process. A Rev. J.*, vol. 73, pp. 1–15, 2018, doi: 10.1016/j.dsp.2017.10.011.

[35] M. Hooshang, R. Askari Moghadam, S. Alizadeh Nia, and M. T. Masouleh, "Optimization of Stirling engine design parameters using neural networks," *Renew. Energy*, vol. 74, pp. 855–866, 2015, doi: 10.1016/j.renene.2014.09.012.

[36] M. Anthony and P. L. Bartlett, *Neural Network Learning: Theoretical Foundations*. Cambridge: Cambridge University Press, 2009.

[37] F. S. Panchal and M. Panchal, "Review on Methods of Selecting Number of Hidden Nodes in Artificial Neural Network," *Int. J. Comput. Sci. Mob. Comput.*, vol. 3, no. 11, pp. 455–464, 2014.

[38] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017, doi: 10.1016/j.neucom.2016.12.038.

[39] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, 2017, doi: 10.1016/j.media.2017.07.005.

[40] B. M. Negash and A. D. Yaw, "Artificial neural network based production forecasting for a hydrocarbon reservoir under water injection," *Pet. Explor. Dev.*, vol. 47, no. 2, pp.

383–392, 2020, doi: 10.1016/S1876-3804(20)60055-6.

[41] B. D. Ripley, *Pattern Recognition via Neural Networks*. Cambridge: Cambridge University Press, 1996.

[42] R. F. MacLin and D. W. Opitz, “An empirical evaluation of bagging and boosting for artificial neural networks,” *IEEE Int. Conf. Neural Networks - Conf. Proc.*, vol. 3, pp. 1401–1405, 1997, doi: 10.1109/ICNN.1997.613999.

[43] Ö. Kişi, “Streamflow Forecasting Using Different Artificial Neural Network Algorithms,” *J. Hydrol. Eng.*, vol. 12, no. 5, pp. 532–539, 2007, doi: 10.1061/(asce)1084-0699(2007)12:5(532).

[44] I. A. Basheer and M. Hajmeer, “Artificial neural networks: fundamentals, computing, design, and application,” *J. Microbiol. Methods*, pp. 3–31, 2000, doi: 10.1016/s0167-7012(00)00201-3.

[45] Y. Chauvin and D. E. Rumelhart, *Backpropagation: Theory, Architectures and Applications*. Hillsdale: Lawrence Erlbaum Associates Inc., 1995.

[46] M. Swain, S. K. Dash, S. Dash, and A. Mohapatra, “An Approach for IRIS Plant Classification Using Neural Network,” *Int. J. Soft Comput.*, vol. 3, no. 1, pp. 79–89, 2012, doi: 10.5121/ijsc.2012.3107.

[47] H. H. Tan and K. H. Lim, “Review of second-order optimization techniques in artificial neural networks backpropagation,” in *IOP Conference Series Materials Science and Engineering*, 2019, vol. 495, doi: 10.1088/1757-899X/495/1/012003.

[48] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares,” *Q. Appl. Math.*, vol. 2, no. 2, pp. 164–168, 1944, doi: 10.1090/qam/10666.

[49] D. W. Marquardt, “An Algorithm for Least-Squares Estimation of Non-linear Parameters,” *Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963, doi: 0.1090/qam/10666.

[50] B. M. Wilamowski, O. Kaynak, S. Iplikci, and M. O. Efe, “An algorithm for fast convergence in training neural networks,” in *International Joint Conference on Neural Networks*, 2001, vol. 3, no. 2, pp. 1778–1782, doi: 10.1109/ijcnn.2001.938431.

[51] M. T. Hagan and M. B. Menhaj, “Training Feedforward Networks with the Marquardt Algorithm,” *IEEE Trans. Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994, doi: 1045-9227/94$04.0.

[52] H. Liu, “On the Levenberg-Marquardt training method for feed-forward neural networks,” in *2010 6th International Conference on Natural Computation, ICNC 2010*, 2010, pp. 456–460, doi: 10.1109/ICNC.2010.5583151.

[53] A. Reynaldi, S. Lukas, and H. Margaretha, “Backpropagation and Levenberg-Marquardt algorithm for training finite element neural network,” in *Proceedings - UKSim-AMSS*

6th European Modelling Symposium, 2012, no. 2, pp. 89–94, doi: 10.1109/EMS.2012.56.

[54] B. M. Wilamowski and H. Yu, "Improved Computation for Levenberg-Marquardt Training," *IEEE Trans. Neural Networks*, vol. 21, no. 6, pp. 930–937, 2010, doi: 10.1109/TNN.2010.2045657.

[55] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural network design*, 2nd ed. 2014.

[56] Ö. Kişi and E. Uncuoğlu, "Comparison of three back-propagation training algorithms for two case studies," *Indian J. Eng. Mater. Sci.*, vol. 12, no. October, pp. 434–442, 2005.

[57] N. Andrei, "Scaled conjugate gradient algorithms for unconstrained optimization," *Comput. Optim. Appl.*, vol. 38, no. 3, pp. 401–416, 2007, doi: 10.1007/s10589-007-9055-7.

[58] H. Chel, A. Majumder, and D. Nandi, "Scaled Conjugate Gradient Algorithm in Neural Network," in *International Conference on Computational Science, Engineering and Information Technology*, 2011, pp. 196–210.

[59] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993, doi: 10.1016/S0893-6080(05)80056-5.

[60] M. F. Møller, "Effcient Training of Feed-Forward Neural Networks," Aarhus University, 1997.

[61] P. M. Williams, "A Marquardt algorithm for choosing step-size in backpropagation learning with conjugate gradients," University of Sussex, 1991.

[62] K. Gopalakrishnan, "Effect of training algorithms on neural networks aided pavement diagnosis," *Int. J. Eng. Sci. Technol.*, vol. 2, no. 2, pp. 83–92, 2010, doi: 10.4314/ijest.v2i2.59147.

[63] O. Baghirli, "Comparison of Levenberg-Marquardt, Scaled Conjugate Gradient And Bayesian Regularization Backpropagation Algorithms for Multistep Ahead Wind Speed Forecasting Using Multilayer Perceptron Feedforward Neural Network," Uppsala University, 2015.

[64] B. Cetişli and A. Barkana, "Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training," *Soft Comput.*, vol. 14, no. 4, pp. 365–378, 2010, doi: 10.1007/s00500-009-0410-8.

[65] L. Liew, "What is Overfitting in Trading?," 2020. https://algotrading101.com/learn/what-is-overfitting-in-trading (accessed Oct. 22, 2021).

[66] G. Cybenko, "Approximation of Superpositions of a Sigmoidal Function," *Math. Control*

Signals Syst., vol. 2, pp. 303–314, 1989, doi: 10.1007/BF02836480.

[67] G. S. Linoff and M. J. A. Berry, *Data Mining Techniques: For Marketing, Sales, and Customer Relationship*, 3rd ed. Indianapolis: Wiley Publishing, Inc., 2011.

[68] Z. Boger and H. Guterman, "Knowledge extraction from artificial neural networks models," *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 4, pp. 3030–3035, 1997.

[69] T. Kavzoglu, "Determining Optimum Structure for Artificial Neural Networks," in *Annual Technical Conference and Exhibition of the Remote Sensing Society*, 1999, pp. 675–682.

[70] A. Azadeh, R. Babazadeh, and S. M. Asadzadeh, "Optimum estimation and forecasting of renewable energy consumption by artificial neural networks," *Renew. Sustain. Energy Rev.*, vol. 27, pp. 605–612, 2013, doi: 10.1016/j.rser.2013.07.007.

[71] A.-N. Sharkawy, "Principle of Neural Network and Its Main Types: Review," *J. Adv. Appl. Comput. Math.*, vol. 7, pp. 8–19, 2020, doi: 10.15377/2409-5761.2020.07.2.

[72] B. Karlik and A. V. Olgac, "Performance Analysis of Various Activation Functions in Artificial Neural Networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2019, doi: 10.1088/1742-6596/1237/2/022030.

[73] A. N. Bhatt and N. Shrivastava, "Application of Artificial Neural Network for Internal Combustion Engines: A State of the Art Review," *Arch. Comput. Methods Eng.*, no. 0123456789, 2021, doi: 10.1007/s11831-021-09596-5.

[74] Y. Ö. Özgören, S. Çetinkaya, S. Saridemir, A. Çiçek, and F. Kara, "Predictive modeling of performance of a helium charged Stirling engine using an artificial neural network," *Energy Convers. Manag.*, vol. 67, pp. 357–368, 2013, doi: 10.1016/j.enconman.2012.12.007.

[75] Y. Kurtgoz, M. Karagoz, and E. Deniz, "Biogas engine performance estimation using ANN," *Eng. Sci. Technol. an Int. J.*, vol. 20, no. 6, pp. 1563–1570, 2017, doi: 10.1016/j.jestch.2017.12.010.

[76] N. Kara Togun and S. Baysec, "Prediction of torque and specific fuel consumption of a gasoline engine by using artificial neural networks," *Appl. Energy*, vol. 87, no. 1, pp. 349–355, 2010, doi: 10.1016/j.apenergy.2009.08.016.

[77] Y. Cay, "Prediction of a gasoline engine performance with artificial neural network," *Fuel*, vol. 111, pp. 324–331, 2013, doi: 10.1016/j.fuel.2012.12.040.

[78] F. Şahin, "Effects of engine parameters on ionization current and modeling of excess air coefficient by artificial neural network," *Appl. Therm. Eng.*, vol. 90, pp. 94–101, 2015, doi: 10.1016/j.applthermaleng.2015.06.100.

[79] M. Gölcü, Y. Sekmen, P. Erduranli, and M. S. Salman, "Artificial neural-network based modeling of variable valve-timing in a spark-ignition engine," *Appl. Energy*, vol. 81, no. 2, pp. 187–197, 2005, doi: 10.1016/j.apenergy.2004.07.008.

[80] Y. Çay, I. Korkmaz, A. Çiçek, and F. Kara, “Prediction of engine performance and exhaust emissions for gasoline and methanol using artificial neural network,” *Energy*, vol. 50, no. 11, pp. 177–186, 2013, doi: 10.1016/j.energy.2012.10.052.

[81] M. Kapusuz, H. Ozcan, and J. A. Yamin, “Research of performance on a spark ignition engine fueled by alcohol-gasoline blends using artificial neural networks,” *Appl. Therm. Eng.*, vol. 91, pp. 525–534, 2015, doi: 10.1016/j.applthermaleng.2015.08.058.

[82] M. K. Deh Kiani, B. Ghobadian, T. Tavakoli, A. M. Nikbakht, and G. Najafi, “Application of artificial neural networks for the prediction of performance and exhaust emissions in SI engine using ethanol- gasoline blends,” *Energy*, vol. 35, no. 1, pp. 65–69, 2010, doi: 10.1016/j.energy.2009.08.034.

[83] G. Najafi, B. Ghobadian, T. Tavakoli, D. R. Buttsworth, T. F. Yusaf, and M. Faizollahnejad, “Performance and exhaust emissions of a gasoline engine with ethanol blended gasoline fuels using artificial neural network,” *Appl. Energy*, vol. 86, no. 5, pp. 630–639, 2009, doi: 10.1016/j.apenergy.2008.09.017.

[84] H. S. Yu and E. Arcakliog, “Comparative study of mathematical and experimental analysis of spark ignition engine performance used ethanol – gasoline blend fuel,” vol. 27, pp. 358–368, 2007, doi: 10.1016/j.applthermaleng.2006.07.027.

[85] C. Sayin, H. M. Ertunc, M. Hosoz, I. Kilicaslan, and M. Canakci, “Performance and exhaust emissions of a gasoline engine using artificial neural network,” *Appl. Therm. Eng.*, vol. 27, no. 1, pp. 46–54, 2007, doi: 10.1016/j.applthermaleng.2006.05.016.

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

[87] H. Kaneko and K. Funatsu, “Smoothing-Combined Soft Sensors for Noise Reduction and Improvement of Predictive Ability,” *Ind. Eng. Chem. Res.*, vol. 54, no. 50, 2015, doi: 10.1021/acs.iecr.5b03054.

[88] J. Yu, S. B. Kim, J. Bai, and S. W. Han, “Comparative study on exponentially weighted moving average approaches for the self-starting forecasting,” *Appl. Sci.*, vol. 10, no. 20, pp. 1–18, 2020, doi: 10.3390/app10207351.

[89] C. Sanjay and C. Jyothi, “A study of surface roughness in drilling using mathematical analysis and neural networks,” *Int. J. Adv. Manuf. Technol.*, vol. 29, no. 9–10, pp. 846–852, 2006, doi: 10.1007/s00170-005-2538-8.

[90] M. W. Browne, “Cross-validation methods. Journal of Mathematical Psychology,” *J. Math. Psychol.*, vol. 44, pp. 108–132, 2000, doi: 10.1006/jmps.1999.1279.

[91] A. Mellit and A. M. Pavan, “Performance prediction of 20 kWp grid-connected photovoltaic plant at Trieste (Italy) using artificial neural network,” *Energy Convers. Manag.*, vol. 51, no. 12, pp. 2431–2441, 2010, doi: 10.1016/j.enconman.2010.05.007.

[92] B. Najafi, S. F. Ardabili, A. Mosavi, S. Shamshirband, and T. Rabczuk, “An intelligent

artificial neural network-response surface methodology method for accessing the optimum biodiesel and diesel fuel blending conditions in a diesel engine from the viewpoint of exergy and energy analysis," *Energies*, vol. 11, no. 4, 2018, doi: 10.3390/en11040860.

[93] H. Jiang, Z. Xi, A. A. Rahman, and X. Zhang, "Prediction of output power with artificial neural network using extended datasets for Stirling engines," *Appl. Energy*, vol. 271, 2020, doi: 10.1016/j.apenergy.2020.115123.

[94] M. A. Shahin, H. R. Maier, and M. B. Jaksa, "Predicting Settlement of Shallow Foundations using Neural Networks," *J. Geotech. Geoenvironmental Eng.*, vol. 128, no. 9, pp. 785–793, 2002, doi: 10.1061/(asce)1090-0241(2002)128:9(785).

[95] K. G. Sheela and S. N. Deepa, "Review on methods to fix number of hidden neurons in neural networks," *Math. Probl. Eng.*, 2013, doi: 10.1155/2013/425740.

[96] I. I. Ozigis, R. A. Adeyemi, P. A. Ondachi, and S. O. Oodo, "Performance evaluation of Kainji hydro-electric power plant using artificial neural networks and multiple linear regression," *Int. J. Energy Water Resour.*, 2021, doi: 10.1007/s42108-021-00135-3.

6. APPENDICES

Appendix A: Training algorithms

A1: LM algorithm

The procedure for the LM algorithm is summarised in Table 14.

Table 14: Procedure for the LM training algorithm [33]

Step 1.	Initialize Weights $w_{ij}^{\ell} =$ Uniformly Random Over $-\varepsilon$ to ε for all i, j, and ℓ.
Step 2.	Present each pattern to the input of the network.
Step 3.	Propagate data forward and generate the output pattern. Calculate the error between the target output and the actual output.
Step 4.	If there are more patterns (i.e., $p < P$) in the training set, loop back to Step 2.
Step 5.	Now calculate the error vector, **e**, between target and actual output for all patterns presented by using summed squared error as in Eq. (A.8).
Step 6.	Compute the Jacobian matrix, **J**, from the first-order partial derivatives.
Step 7.	Compute the weight update as given in Eq. (A.36).
Step 8.	Recalculate the sum of squared errors. If the new error is lower, reduce μ by a factor β, update the weights by Δw, and go to Step 9. If the new error is higher, increase μ by β and go back to Step 7.
Step 9.	If the norm of the gradient, **g**, is less than the desired amount, stop; otherwise loop back to Step 1.

A2: SCG algorithm

The procedure for the SCG training algorithm is summarised in Table 15.

Table 15: Procedure for the SCG training algorithm [59]

1. Choose weight vector $\tilde{w}_1$ and scalars $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-6}$, $\bar{\lambda}_1 = 0$.
   Set $\tilde{p}_1 = \tilde{r}_1 = -E'(\tilde{w}_1)$, $k = 1$ and success = true.
2. If success = true, then calculate second order information:
   $\sigma_k = \sigma / |\tilde{p}_k|$,
   $\tilde{s}_k = (E'(\tilde{w}_k + \sigma_k \tilde{p}_k) - E'(\tilde{w}_k))/\sigma_k$,
   $\delta_k = \tilde{p}_k^T \tilde{s}_k$.
3. Scale δ_k: $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|\tilde{p}_k|^2$.
4. If $\delta_k \leq 0$ then make the Hessian matrix positive definite:
   $\bar{\lambda}_k = 2(\lambda_k - \delta_k/|\tilde{p}_k|^2)$,
   $\delta_k = -\delta_k + \lambda_k |\tilde{p}_k|^2$,
   $\lambda_k = \bar{\lambda}_k$.
5. Calculate step size:
   $\mu_k = \tilde{p}_k^T \tilde{r}_k$,
   $\alpha_k = \mu_k / \delta_k$.
6. Calculate the comparison parameter:
   $\Delta_k = 2\delta_k[E(\tilde{w}_k) - E(\tilde{w}_k + \alpha_k \tilde{p}_k)]/\mu_k^2$.
7. If $\Delta_k \geq 0$ then a successful reduction in error can be made:
   $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$,
   $\tilde{r}_{k+1} = -E'(\tilde{w}_{k+1})$,
   $\bar{\lambda}_k = 0$, success = true.
   If k mod $N = 0$ then restart algorithm:
   $\tilde{p}_{k+1} = \tilde{r}_{k+1}$
   else:
   $\beta_k = (|\tilde{r}_{k+1}|^2 - \tilde{r}_{k+1}^T \tilde{r}_k)/\mu_k$,
   $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$.
   If $\Delta_k \geq 0.75$, then reduce the scale parameter:
   $\lambda_k = \frac{1}{4}\lambda_k$.
   else:
   $\bar{\lambda}_k = \lambda_k$,
   success = false.
8. If $\Delta_k < 0.25$, then increase the scale parameter:
   $\lambda_k = \lambda_k + (\delta_k(1 - \Delta_k)/|\tilde{p}_k|^2)$.
9. If the steepest descent direction $\tilde{r}_k \neq \tilde{0}$, then set $k = k + 1$ and go to 2 else terminate and return $\tilde{w}_{k+1}$ as the desired minimum.

Appendix B: Results

B1: Noise filtering

The plots indicating the raw and filtered data for the input operational variables and output power is given in this section by Figure 56 to Figure 61.

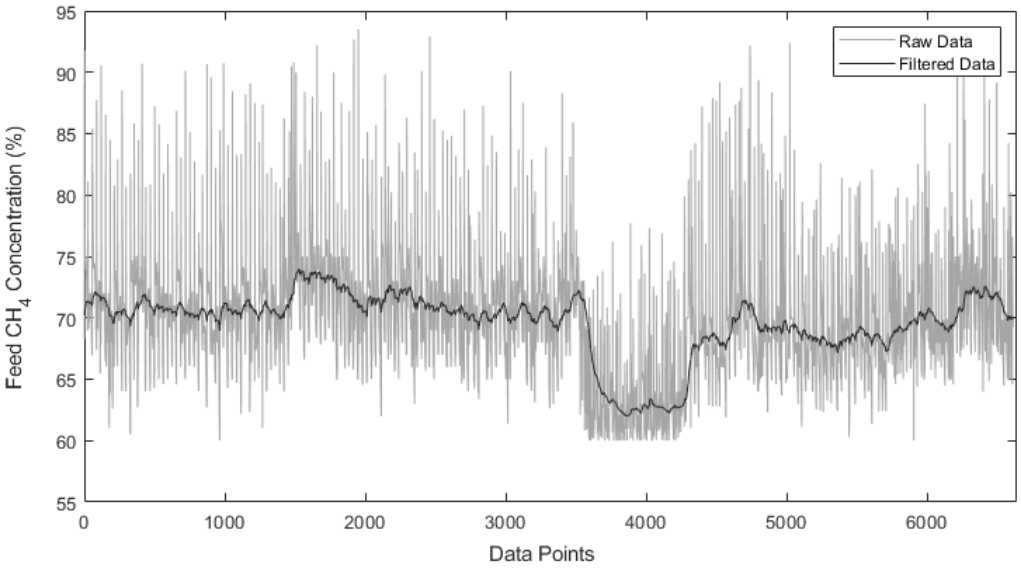


Figure 56: Raw and filtered methane composition data

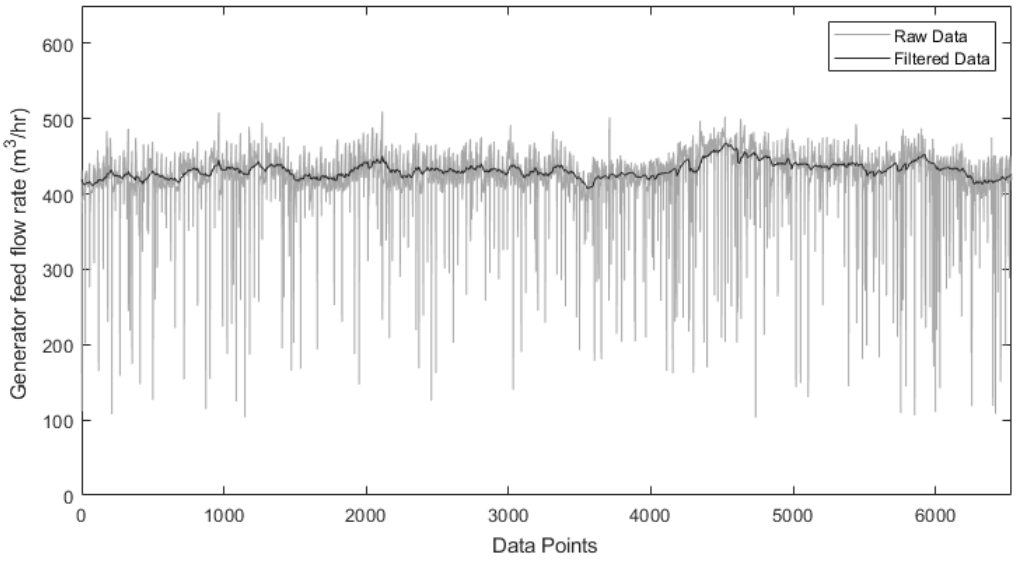


Figure 57: Raw and filtered feed flow rate to generator data

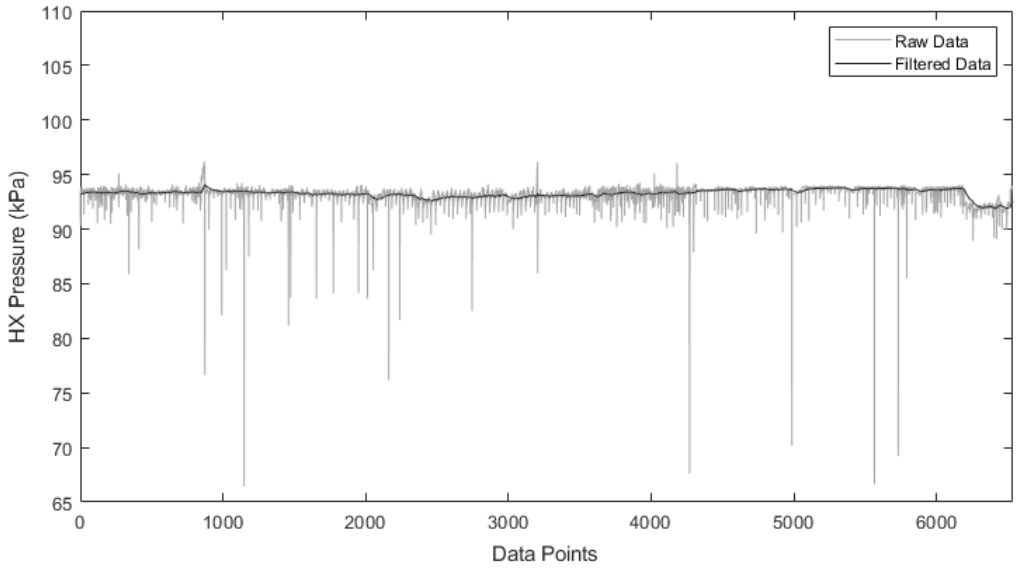


Figure 58: Raw and filtered coolant pressure data

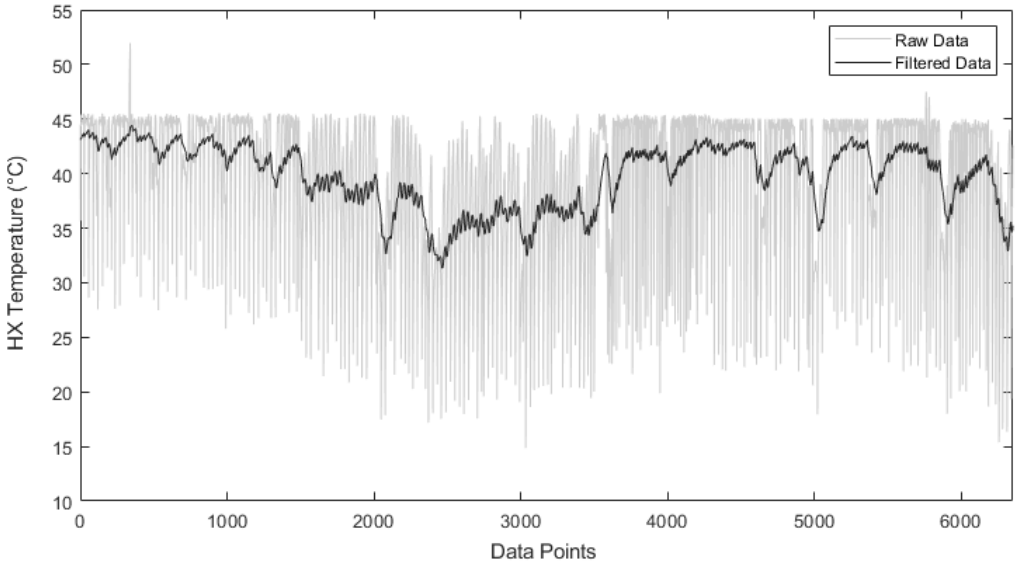


Figure 59: Raw and filtered coolant temperature data

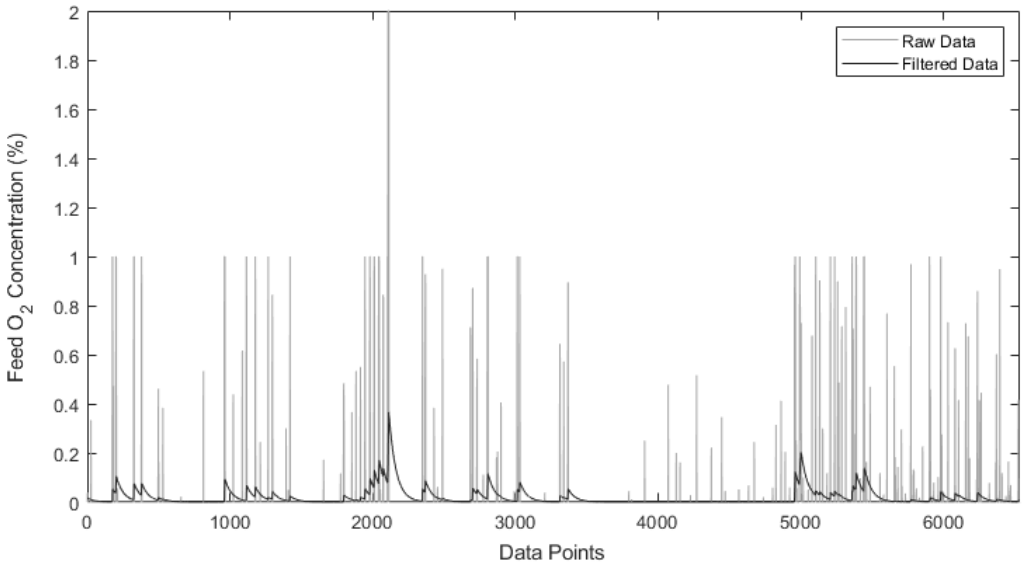


Figure 60: Raw and filtered oxygen composition concentration

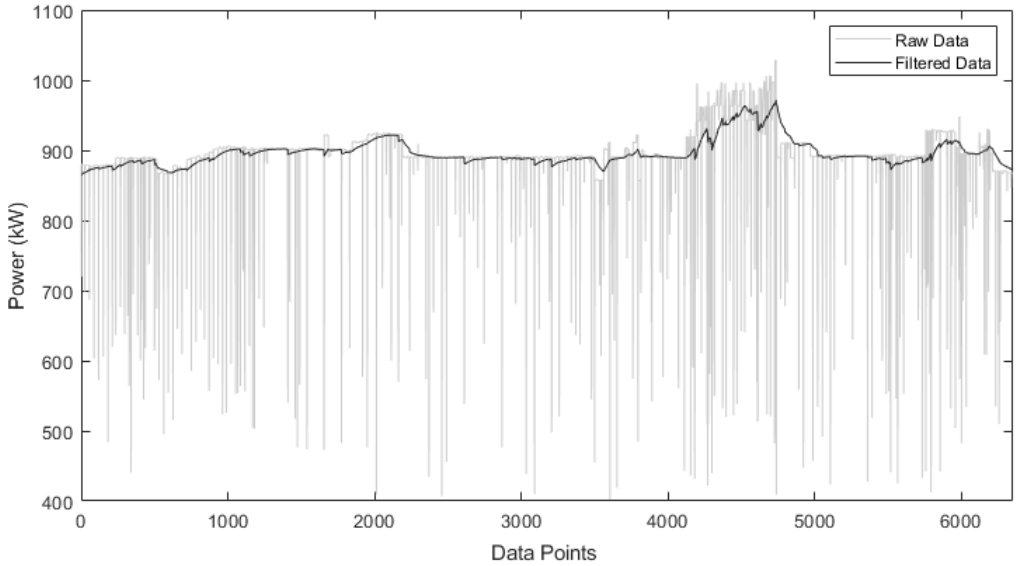


Figure 61: Raw and filtered output power data

Appendix C: Source code

C1: Main call script

```
% Test Identification
testID = "A7_T01";

% Configuration
p = 0.1;
HL1_minH = 1;
HL1_maxH = 200;
HL1_minRep = 1;
HL1_maxRep = 50;
HL2_minH = 5;
HL2_maxH = 30;
HL2_minRep = 1;
HL2_maxRep = 10;
optH_HL1 = 20;
optH1_HL2 = 14;
optH2_HL2 = 11;
optMinRep = 1;
optMaxRep = 1000;

% Load Raw Data
T_Raw = readtable('DataSource_Rev16.xlsx','Sheet','Raw Data');

% Data Filtering
T_NF = F1_A7_N_Filter(T_Raw);
T_IF = F2_A7_I_Filter(T_NF);
P1_A7_Filter(testID,T_Raw,T_NF,T_IF);

% Cross-Validation
cv = cvpartition(size(T_IF,1),'HoldOut',p);
cvIndex = cv.test;
mT = T_IF(~cvIndex,:);
vT = T_IF(cvIndex,:);

% Training - LM Algorithm
algorithm = 'trainlm';
F3_A7_Train_1HL(testID,algorithm,mT,HL1_minH,HL1_maxH,HL1_minRep,HL1_maxRep)
F4_A7_Train_2HL(testID,algorithm,mT,HL2_minH,HL2_maxH,HL2_minRep,HL2_maxRep)

% Training - SCG Algorithm
algorithm = 'trainscg';
F3_A7_Train_1HL(testID,algorithm,mT,HL1_minH,HL1_maxH,HL1_minRep,HL1_maxRep)
F4_A7_Train_2HL(testID,algorithm,mT,HL2_minH,HL2_maxH,HL2_minRep,HL2_maxRep)

% Optimal Network Selection
algorithm = 'trainlm';
F5_A7_Optimal_1HL(testID,algorithm,mT,optH_HL1,optMinRep,optMaxRep)
F6_A7_Optimal_2HL(testID,algorithm,mT,optH1_HL2,optH2_HL2,optMinRep,optMaxRep)

% Model Plotting
P5_A7_ModelPlotting(testID)

% Verification
F7_A7_Verify(testID,vT);

% Save Results
saveString = testID + "_Main_Workspace";
```

C2: Data processing

C2.1 Noise Filter

```
function y = F1_A7_N_Filter(T)

% Variable Allocation
XM = T{:,1};
XO = T{:,2};
FG = T{:,3};
FF = T{:,4};
TC = T{:,5};
PC = T{:,6};
WG = T{:,7};

% Noise Spike Filter (medfilt1)
nsf_XM = medfilt1(XM,3,'omitnan','truncate');
nsf_XO = medfilt1(XO,3,'omitnan','truncate');
nsf_FG = medfilt1(FG,3,'omitnan','truncate');
nsf_FF = medfilt1(FF,3,'omitnan','truncate');
nsf_TC = medfilt1(TC,3,'omitnan','truncate');
nsf_PC = medfilt1(PC,3,'omitnan','truncate');
nsf_WG = medfilt1(WG,3,'omitnan','truncate');

% Exponentially Weighted Moving Average Filter (dsp.MovingAverage)
aXM = 0.9;
aXO = 0.9;
aFG = 0.9;
aFF = 0.9;
aTC = 0.9;
aPC = 0.9;
aWG = 0.9;
ewma_XM = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aXM);
ewma_XO = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aXO);
ewma_FG = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aFG);
ewma_FF = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aFF);
ewma_TC = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aTC);
ewma_PC = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aPC);
ewma_WG = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',aWG);
expMA_XM = ewma_XM(nsf_XM);
expMA_XO = ewma_XO(nsf_XO);
expMA_FG = ewma_FG(nsf_FG);
expMA_FF = ewma_FF(nsf_FF);
expMA_TC = ewma_TC(nsf_TC);
expMA_PC = ewma_PC(nsf_PC);
expMA_WG = ewma_WG(nsf_WG);

% Function Output
y = [expMA_XM expMA_XO expMA_FG expMA_FF expMA_TC expMA_PC expMA_WG];

end
```

C2.2 Inequality filter

```
function y = F2_A7_I_Filter(T)

% Variable Allocation
XM = T(:,1);
XO = T(:,2);
FG = T(:,3);
FF = T(:,4);
TC = T(:,5);
PC = T(:,6);
WG = T(:,7);

% Inequality filter for XM
indexXM = XM > 60 & XM <= 100;
XM = XM(indexXM);
XO = XO(indexXM);
FG = FG(indexXM);
FF = FF(indexXM);
TC = TC(indexXM);
PC = PC(indexXM);
WG = WG(indexXM);

% Inequality filter for XO
indexXO = XO >= 0 & XO < 5;
XM = XM(indexXO);
XO = XO(indexXO);
FG = FG(indexXO);
FF = FF(indexXO);
TC = TC(indexXO);
PC = PC(indexXO);
WG = WG(indexXO);

% Inequality filter for FG
indexFG = FG > 200 & FG <= 450;
XM = XM(indexFG);
XO = XO(indexFG);
FG = FG(indexFG);
FF = FF(indexFG);
TC = TC(indexFG);
PC = PC(indexFG);
WG = WG(indexFG);

% Inequality filter for FF
indexFF = FF > 0 & FF <= 400;
XM = XM(indexFF);
XO = XO(indexFF);
FG = FG(indexFF);
FF = FF(indexFF);
TC = TC(indexFF);
PC = PC(indexFF);
WG = WG(indexFF);

% Inequality filter for TC
indexTC = TC > 20 & TC <= 50;
XM = XM(indexTC);
XO = XO(indexTC);
FG = FG(indexTC);
FF = FF(indexTC);
TC = TC(indexTC);
PC = PC(indexTC);
WG = WG(indexTC);

% Inequality filter for PC
```

```
indexPC = PC > 90 & PC <= 110;
XM = XM(indexPC);
XO = XO(indexPC);
FG = FG(indexPC);
FF = FF(indexPC);
TC = TC(indexPC);
PC = PC(indexPC);
WG = WG(indexPC);

% Inequality filter for WG
indexWG = WG > 200 & WG <= 1100;
XM = XM(indexWG);
XO = XO(indexWG);
FG = FG(indexWG);
FF = FF(indexWG);
TC = TC(indexWG);
PC = PC(indexWG);
WG = WG(indexWG);

% Function Output
y = [XM XO FG FF TC PC WG];

end
```

C2.3: Noise filter plots

```
function P1_A7_Filter(testID,T_Raw,T_NF,T_IF)

% Variable Allocation
rawXM = T_Raw{:,1};
rawXO = T_Raw{:,2};
rawFG = T_Raw{:,3};
rawFF = T_Raw{:,4};
rawTC = T_Raw{:,5};
rawPC = T_Raw{:,6};
rawWG = T_Raw{:,7};
nfXM = T_NF(:,1);
nfXO = T_NF(:,2);
nfFG = T_NF(:,3);
nfFF = T_NF(:,4);
nfTC = T_NF(:,5);
nfPC = T_NF(:,6);
nfWG = T_NF(:,7);
ifXM = T_IF(:,1);
ifXO = T_IF(:,2);
ifFG = T_IF(:,3);
ifFF = T_IF(:,4);
ifTC = T_IF(:,5);
ifPC = T_IF(:,6);
ifWG = T_IF(:,7);

bound = 1000;

% Noise Filter Plots
figure
plot(rawXM(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfXM(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('XM (%)')
```

```
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_XM');

figure
plot(rawXO(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfXO(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('XO (%)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_XO');

figure
plot(rawFG(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfFG(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Flow rate to Generator (m^3/hr)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_FG');

figure
plot(rawFF(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfFF(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Flow rate to Flare (m^3/hr)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_FF');

figure
plot(rawTC(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfTC(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Coolant Temperature (°C)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_TC');

figure
plot(rawPC(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfPC(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Coolant Pressure (kPa)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_PC');

figure
plot(rawWG(1:bound), 'Color', [0.8 0.8 0.8])
hold on
plot(nfWG(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Power (kW)')
legend('Raw','Noise Filter')
savefig('A7_' + testID + '_NFilter_WG');

% Inequality Filter Plots
figure
plot(ifXM(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('XM (%)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_XM');
```

```
figure
plot(ifXO(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('XO (%)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_XO');

figure
plot(ifFG(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Flow rate to Generator (m^3/hr)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_FG');

figure
plot(ifFF(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Flow rate to Flare (m^3/hr)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_FF');

figure
plot(ifTC(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Coolant Temperature (°C)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_TC');

figure
plot(ifPC(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Coolant Pressure (kPa)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_PC');

figure
plot(ifWG(1:bound), 'Color', 'k')
xlabel('Data Points')
ylabel('Power (kW)')
legend('Inequality Filter')
savefig('A7_' + testID + '_IFilter_WG');

close all

end
```

C3: Model development

C3.1: Training neural networks – One hidden layer

```
function F3_A7_Train_1HL(testID,algorithm,mT,minH,maxH,minRep,maxRep)

% Identification
testID = testID + '_' + algorithm + '_1HL';

% Iteration Bounds
diffH = maxH - minH + 1;
diffRep = maxRep - minRep + 1;

% Variable Allocation
```

```
XM = mT(:,1);
XO = mT(:,2);
FG = mT(:,3);
FF = mT(:,4);
TC = mT(:,5);
PC = mT(:,6);
WG = mT(:,7);
input = [XM XO FG FF TC PC]';
target = [WG]';

% Variable Initialisation
netArray = ObjectArray.empty;
trArray = ObjectArray.empty;
outArray = ObjectArray.empty;
perfMSE = zeros(diffH, diffRep);
perfMAE = zeros(diffH, diffRep);
perfSSE = zeros(diffH, diffRep);
perfSAE = zeros(diffH, diffRep);
perfMAPE = zeros(diffH, diffRep);
perfRMSE = zeros(diffH, diffRep);
e = ObjectArray.empty;
eMSE = zeros(diffH, diffRep);
rArray = zeros(diffH, diffRep);
mArray = zeros(diffH, diffRep);
bArray = zeros(diffH, diffRep);
mseTrain = zeros(diffH, diffRep);
mseVal = zeros(diffH, diffRep);
mseTest = zeros(diffH, diffRep);
rTrain = zeros(diffH, diffRep);
rVal = zeros(diffH, diffRep);
rTest = zeros(diffH, diffRep);
rmseTrain = zeros(diffH, diffRep);
rmseVal = zeros(diffH, diffRep);
rmseTest = zeros(diffH, diffRep);
mapeTrain = zeros(diffH, diffRep);
mapeVal = zeros(diffH, diffRep);
mapeTest = zeros(diffH, diffRep);

% Training by varying Number of Hidden Neurons - 1 Hidden Layer
for i = 1:diffH
    for j = 1:diffRep

        ticIT = tic;
        disp(algorithm + "    " + (minH + i - 1) + ":" + (minRep + j - 1))

        % Number of Hidden Neurons
        HNs = minH + i - 1;

        % Network Configuration
        net = feedforwardnet(HNs, algorithm);
        net.divideParam.trainRatio = 0.7;
        net.divideParam.valRatio = 0.2;
        net.divideParam.testRatio = 0.1;
        net.trainParam.showWindow = 0;
        net = configure(net,input,target);

        % Train
        [net, tr] = train(net,input,target);

        % Predict
        out = net(input);

        % Performance
        perfMSE(i,j) = mse(net,target,out);
        perfMAE(i,j) = mae(net,target,out);
```

```
        perfSSE(i,j) = sse(net,target,out);
        perfSAE(i,j) = sae(net,target,out);
        perfMAPE(i,j) = mean((abs(target-out))./target)*100;
        perfRMSE(i,j) = sqrt(perfMSE(i,j));
        e(i,j) = gsubtract(con2seq(out(1,:)'),target(1,:)');
        eMSE(i,j) = immse(target,out);
        [rArray(i,j),mArray(i,j),bArray(i,j)] = regression(target,out);

        % Performance Sub Datasets
        indexOutTrain = out(tr.trainInd');
        indexOutVal = out(tr.valInd');
        indexOutTest = out(tr.testInd');
        indexTargetTrain = target(tr.trainInd');
        indexTargetVal = target(tr.valInd');
        indexTargetTest = target(tr.testInd');
        mseTrain(i,j) = immse(indexTargetTrain,indexOutTrain);
        mseVal(i,j) = immse(indexTargetVal,indexOutVal);
        mseTest(i,j) = immse(indexTargetTest,indexOutTest);
        rTrain(i,j) = regression(indexTargetTrain,indexOutTrain);
        rVal(i,j) = regression(indexTargetVal,indexOutVal);
        rTest(i,j) = regression(indexTargetTest,indexOutTest);
        rmseTrain(i,j) = sqrt(mseTrain(i,j));
        rmseVal(i,j) = sqrt(mseVal(i,j));
        rmseTest(i,j) = sqrt(mseTest(i,j));
        mapeTrain(i,j) = mean((abs(indexTargetTrain -indexOutTrain ))./indexTargetTrain)
        *100;
        mapeVal(i,j) = mean((abs(indexTargetVal -indexOutVal ))./indexTargetVal )*100;
        mapeTest(i,j) = mean((abs(indexTargetTest -indexOutTest ))./indexTargetTest)*100

        % Save to Array
        netArray(i,j) = ObjectArray(net);
        trArray(i,j) = ObjectArray(tr);
        outArray(i,j) = ObjectArray(out);

        % Display Iteration Results
        disp(" Test MSE = " + mseTest(i,j))
        disp(" Test R^2 = " + rTest(i,j))
        disp(" Train elapsed time = " + toc(ticIT) + " s")

    end
end

% Plot - Performance VS Architecture
P2_A7_PerfHN_1HL(testID,minH,maxH,mseTrain,mseVal,mseTest,rTrain,rVal,rTest,rmseTrain,rm
seVal,rmseTest,mapeTrain,mapeVal,mapeTest)

% Save Workspace
save(testID + '_Workspace','-v7.3');

end
```

C3.2: Plot neural network results – One hidden layer

```
function P2_A7_PerfHN_1HL(testID,minH,maxH,mseTrainRaw,mseValRaw,mseTestRaw,rTrainRaw,
rValRaw,rTestRaw,rmseTrainRaw,rmseValRaw,rmseTestRaw,mapeTrainRaw,mapeValRaw,
mapeTestRaw)

% Initialisation
axisH = minH:maxH;
```

```
% Find & Replace Outliers
mseTrain = filloutliers(mseTrainRaw,'linear','movmedian',5);
mseVal = filloutliers(mseValRaw,'linear','movmedian',5);
mseTest = filloutliers(mseTestRaw,'linear','movmedian',5);
rTrain = filloutliers(rTrainRaw,'linear','movmedian',5);
rVal = filloutliers(rValRaw,'linear','movmedian',5);
rTest = filloutliers(rTestRaw,'linear','movmedian',5);
rmseTrain = filloutliers(rmseTrainRaw,'linear','movmedian',5);
rmseVal = filloutliers(rmseValRaw,'linear','movmedian',5);
rmseTest = filloutliers(rmseTestRaw,'linear','movmedian',5);
mapeTrain = filloutliers(mapeTrainRaw,'linear','movmedian',5);
mapeVal = filloutliers(mapeValRaw,'linear','movmedian',5);
mapeTest = filloutliers(mapeTestRaw,'linear','movmedian',5);

% Calculate Performance Mean over All Repetitions
mseMeanTrain = mean(mseTrain,2);
mseMeanVal = mean(mseVal,2);
mseMeanTest = mean(mseTest,2);
rMeanTrain = mean(rTrain,2);
rMeanVal = mean(rVal,2);
rMeanTest = mean(rTest,2);
rmseMeanTrain = mean(rmseTrain,2);
rmseMeanVal = mean(rmseVal,2);
rmseMeanTest = mean(rmseTest,2);
mapeMeanTrain = mean(mapeTrain,2);
mapeMeanVal = mean(mapeVal,2);
mapeMeanTest = mean(mapeTest,2);

% Filter Mean Results with 1D Median Filter
mseMeanTrain = medfilt1(mseMeanTrain,3,'omitnan','truncate');
mseMeanVal = medfilt1(mseMeanVal,3,'omitnan','truncate');
mseMeanTest = medfilt1(mseMeanTest,3,'omitnan','truncate');
rMeanTrain = medfilt1(rMeanTrain,3,'omitnan','truncate');
rMeanVal = medfilt1(rMeanVal,3,'omitnan','truncate');
rMeanTest = medfilt1(rMeanTest,3,'omitnan','truncate');
rmseMeanTrain = medfilt1(rmseMeanTrain,3,'omitnan','truncate');
rmseMeanVal = medfilt1(rmseMeanVal,3,'omitnan','truncate');
rmseMeanTest = medfilt1(rmseMeanTest,3,'omitnan','truncate');
mapeMeanTrain = medfilt1(mapeMeanTrain,3,'omitnan','truncate');
mapeMeanVal = medfilt1(mapeMeanVal,3,'omitnan','truncate');
mapeMeanTest = medfilt1(mapeMeanTest,3,'omitnan','truncate');

% Main Plot (Train)
figure
hold on
yyaxis left
plot(axisH,mseMeanTrain)
ylabel('MSE')
yyaxis right
plot(axisH,rMeanTrain)
ylabel('R^2')
hold off
legend('MSE Train', 'R^2 Train')
legend('Location','southoutside','Orientation','horizontal')
nameFigure = testID + '_Figure_P2_Train';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (Val)
figure
hold on
yyaxis left
plot(axisH,mseMeanVal)
ylabel('MSE')
yyaxis right
```

```
plot(axisH,rMeanVal)
ylabel('R^2')
hold off
legend('MSE Validation', 'R^2 Validation')
legend('Location','southoutside','Orientation','horizontal')
nameFigure = testID + '_Figure_P2_Val';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (Test)
figure
hold on
yyaxis left
plot(axisH,mseMeanTest)
ylabel('MSE')
yyaxis right
plot(axisH,rMeanTest)
ylabel('R^2')
hold off
legend('MSE Test', 'R^2 Test')
legend('Location','southoutside','Orientation','horizontal')
nameFigure = testID + '_Figure_P2_Test';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main MSE Plot
figure
hold on
yyaxis left
plot(axisH,mseMeanTrain)
plot(axisH,mseMeanVal)
plot(axisH,mseMeanTest)
ylabel('MSE')
yyaxis right
plot(axisH,rMeanTrain);
plot(axisH,rMeanVal);
plot(axisH,rMeanTest);
hold off
ylabel('R^2')
xlabel('Number of Hidden Neurons')
legend('MSE Train', 'MSE Validation','MSE Test', 'R^2 Train', 'R^2 Validation', 'R^2
Test')
legend('Location','southoutside','NumColumns',2)
nameFigure = testID + '_Figure_P2_MSEPlot';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main RMSE Plot
figure
hold on
yyaxis left
plot(axisH,rmseMeanTrain)
plot(axisH,rmseMeanVal)
plot(axisH,rmseMeanTest)
ylabel('RMSE')
yyaxis right
plot(axisH,rMeanTrain)
plot(axisH,rMeanVal)
plot(axisH,rMeanTest)
hold off
ylabel('R^2')
xlabel('Number of Hidden Neurons')
legend('RMSE Train', 'RMSE Validation','RMSE Test', 'R^2 Train', 'R^2 Validation', 'R^2
Test')
legend('Location','southoutside','NumColumns',2)
```

```
nameFigure = testID + '_Figure_P2_RMSEPlot';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main MAPE Plot
figure
hold on
yyaxis left
plot(axisH,mapeMeanTrain)
plot(axisH,mapeMeanVal)
plot(axisH,mapeMeanTest)
ylabel('MAPE')
yyaxis right
plot(axisH,rMeanTrain)
plot(axisH,rMeanVal)
plot(axisH,rMeanTest)
hold off
ylabel('Regression (R^2)')
xlabel('Number of Hidden Neurons')
legend('MAPE Train', 'MAPE Validation','MAPE Test', 'R^2 Train', 'R^2 Validation', 'R^2
Test')
legend('Location','southoutside','NumColumns',2)
nameFigure = testID + '_Figure_P2_MAPEPlot';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Save Workspace
save(testID + '_Workspace_P2','-v7.3');

end
```

C3.3: Training neural networks – Two hidden layers

```
function F4_A7_Train_2HL(testID,algorithm,mT,minH,maxH,minRep,maxRep)

% Identification
testID = testID + '_' + algorithm + '_2HL';

% Iteration Bounds
diffH = maxH - minH + 1;
diffRep = maxRep - minRep + 1;

% Variable Allocation
XM = mT(:,1);
XO = mT(:,2);
FG = mT(:,3);
FF = mT(:,4);
TC = mT(:,5);
PC = mT(:,6);
WG = mT(:,7);
input = [XM XO FG FF TC PC]';
target = [WG]';

% Variable Initialisation
netArray = ObjectArray.empty;
trArray = ObjectArray.empty;
outArray = ObjectArray.empty;
perfMSE = zeros(diffH, diffH, diffRep);
perfMAE = zeros(diffH, diffH, diffRep);
perfSSE = zeros(diffH, diffH, diffRep);
```

```
perfSAE = zeros(diffH, diffH, diffRep);
perfMAPE = zeros(diffH, diffH, diffRep);
perfRMSE = zeros(diffH, diffH, diffRep);
e = ObjectArray.empty;
eMSE = zeros(diffH, diffH, diffRep);
rArray = zeros(diffH, diffH, diffRep);
mArray = zeros(diffH, diffH, diffRep);
bArray = zeros(diffH, diffH, diffRep);
mseTrain = zeros(diffH, diffH, diffRep);
mseVal = zeros(diffH, diffH, diffRep);
mseTest = zeros(diffH, diffH, diffRep);
rTrain = zeros(diffH, diffH, diffRep);
rVal = zeros(diffH, diffH, diffRep);
rTest = zeros(diffH, diffH, diffRep);
rmseTrain = zeros(diffH, diffH, diffRep);
rmseVal = zeros(diffH, diffH, diffRep);
rmseTest = zeros(diffH, diffH, diffRep);
mapeTrain = zeros(diffH, diffH, diffRep);
mapeVal = zeros(diffH, diffH, diffRep);
mapeTest = zeros(diffH, diffH, diffRep);

% Training by varying Number of Hidden Neurons - Two Hidden Layers
for i = 1:diffH
    for j = 1:diffH
        for k = 1:diffRep

            disp(algorithm + "       " + (minH + i - 1) + "-" + (minH + j - 1) + ":" +
            (minRep + k - 1))
            ticIT = tic;

            % Number of Hidden Neurons
            HNs_HL1 = minH + i - 1;
            HNs_HL2 = minH + j - 1;

            % Network Configuration
            net = feedforwardnet([HNs_HL1 HNs_HL2], algorithm);
            net.divideParam.trainRatio = 0.7;
            net.divideParam.valRatio = 0.2;
            net.divideParam.testRatio = 0.1;
            net.trainParam.showWindow = 0;
            net = configure(net,input,target);

            % Train
            [net, tr] = train(net,input,target);

            % Predict
            out = net(input);

            % Performance
            perfMSE(i,j,k) = mse(net,target,out);
            perfMAE(i,j,k) = mae(net,target,out);
            perfSSE(i,j,k) = sse(net,target,out);
            perfSAE(i,j,k) = sae(net,target,out);
            perfMAPE(i,j,k) = mean((abs(target-out))./target)*100;
            perfRMSE(i,j,k) = sqrt(perfMSE(i,j,k));
            e(i,j,k) = gsubtract(con2seq(out(1,:)'),target(1,:)');
            eMSE(i,j,k) = immse(target,out);
            [rArray(i,j,k),mArray(i,j,k),bArray(i,j,k)] = regression(target,out);

            % Performance Sub Datasets
            indexOutTrain = out(tr.trainInd');
            indexOutVal = out(tr.valInd');
            indexOutTest = out(tr.testInd');
            indexTargetTrain = target(tr.trainInd');
            indexTargetVal = target(tr.valInd');
```

```
            indexTargetTest = target(tr.testInd');
            mseTrain(i,j,k) = immse(indexTargetTrain,indexOutTrain);
            mseVal(i,j,k) = immse(indexTargetVal,indexOutVal);
            mseTest(i,j,k) = immse(indexTargetTest,indexOutTest);
            rTrain(i,j,k) = regression(indexTargetTrain,indexOutTrain);
            rVal(i,j,k) = regression(indexTargetVal,indexOutVal);
            rTest(i,j,k) = regression(indexTargetTest,indexOutTest);
            rmseTrain(i,j,k) = sqrt(mseTrain(i,j,k));
            rmseVal(i,j,k) = sqrt(mseVal(i,j,k));
            rmseTest(i,j,k) = sqrt(mseTest(i,j,k));
            mapeTrain(i,j,k) = mean((abs(indexTargetTrain -indexOutTrain))
            indexTargetTrain )*100;
            mapeVal(i,j,k) = mean((abs(indexTargetVal -indexOutVal))./indexTargetVal)
            *100;
            mapeTest(i,j,k) = mean((abs(indexTargetTest -indexOutTest))
            ./indexTargetTest))*100;

            % Save to Array
            netArray(i,j,k) = ObjectArray(net);
            trArray(i,j,k) = ObjectArray(tr);
            outArray(i,j,k) = ObjectArray(out);

            % Display Iteration Results
            disp("Test MSE = " + mseTest(i,j,k))
            disp("Test R^2 = " + rTest(i,j,k))
            disp("Train elapsed time = " + toc(ticIT) + " s")

        end
    end
end

% Plot - Performance VS Architecture
P3_A7_PerfHN_2HL(testID,minH,maxH,mseTrain,mseVal,mseTest,rTrain,rVal,rTest,rmseTrain,rm
seVal,rmseTest,mapeTrain,mapeVal,mapeTest)

% Save Workspace
save(testID + '_Workspace','-v7.3');

end
```

C3.4: Plot neural network results – Two hidden layers

```
function P3_A7_PerfHN_2HL(testID,minH,maxH,mseTrainRaw,mseValRaw,mseTestRaw,rTrainRaw,
rValRaw,rTestRaw,rmseTrainRaw,rmseValRaw,rmseTestRaw,mapeTrainRaw,mapeValRaw,
mapeTestRaw)

% Initialisation
axisH = minH:maxH;
faceAlpha = 0.5;

% Find & Replace Outliers
mseTrain = filloutliers(mseTrainRaw,'linear','movmedian',5);
mseVal = filloutliers(mseValRaw,'linear','movmedian',5);
mseTest = filloutliers(mseTestRaw,'linear','movmedian',5);
rTrain = filloutliers(rTrainRaw,'linear','movmedian',5);
rVal = filloutliers(rValRaw,'linear','movmedian',5);
rTest = filloutliers(rTestRaw,'linear','movmedian',5);
rmseTrain = filloutliers(rmseTrainRaw,'linear','movmedian',5);
rmseVal = filloutliers(rmseValRaw,'linear','movmedian',5);
rmseTest = filloutliers(rmseTestRaw,'linear','movmedian',5);
```

```
mapeTrain = filloutliers(mapeTrainRaw,'linear','movmedian',5);
mapeVal = filloutliers(mapeValRaw,'linear','movmedian',5);
mapeTest = filloutliers(mapeTestRaw,'linear','movmedian',5);

% Calculate Performance Mean over All Repetitions
mseMeanTrain = mean(mseTrain,3);
mseMeanVal = mean(mseVal,3);
mseMeanTest = mean(mseTest,3);
rMeanTrain = mean(rTrain,3);
rMeanVal = mean(rVal,3);
rMeanTest = mean(rTest,3);
rmseMeanTrain = mean(rmseTrain,3);
rmseMeanVal = mean(rmseVal,3);
rmseMeanTest = mean(rmseTest,3);
mapeMeanTrain = mean(mapeTrain,3);
mapeMeanVal = mean(mapeVal,3);
mapeMeanTest = mean(mapeTest,3);

% Filter Mean Results with 2D Median Filter
mseMeanTrain = medfilt2(mseMeanTrain,[8 8],'symmetric');
mseMeanVal = medfilt2(mseMeanVal,[8 8],'symmetric');
mseMeanTest = medfilt2(mseMeanTest,[8 8],'symmetric');
rMeanTrain = medfilt2(rMeanTrain,[8 8],'symmetric');
rMeanVal = medfilt2(rMeanVal,[8 8],'symmetric');
rMeanTest = medfilt2(rMeanTest,[8 8],'symmetric');
rmseMeanTrain = medfilt2(rmseMeanTrain,[8 8],'symmetric');
rmseMeanVal = medfilt2(rmseMeanVal,[8 8],'symmetric');
rmseMeanTest = medfilt2(rmseMeanTest,[8 8],'symmetric');
mapeMeanTrain = medfilt2(mapeMeanTrain,[8 8],'symmetric');
mapeMeanVal = medfilt2(mapeMeanVal,[8 8],'symmetric');
mapeMeanTest = medfilt2(mapeMeanTest,[8 8],'symmetric');

% Main Plot (MSE Train)
figure, surf(axisH,axisH,mseMeanTrain,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MSE Train')
%zlim(zLimit1)
nameFigure = testID + '_Figure_P3_MSETrain';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (MSE Val)
figure, surf(axisH,axisH,mseMeanVal,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MSE Validation')
nameFigure = testID + '_Figure_P3_MSEVal';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (MSE Test)
figure, surf(axisH,axisH,mseMeanTest,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MSE Test')
nameFigure = testID + '_Figure_P3_MSETest';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (R^2 Train)
figure, surf(axisH,axisH,rMeanTrain,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('R^2 Train')
nameFigure = testID + '_Figure_P3_R2Train';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (R^2 Val)
figure, surf(axisH,axisH,rMeanVal,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('R^2 Validation')
nameFigure = testID + '_Figure_P3_R2Val';
```

```
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (R^2 Test)
figure, surf(axisH,axisH,rMeanTest,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('R^2 Test')
nameFigure = testID + '_Figure_P3_R2Test';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (RMSE Train)
figure, surf(axisH,axisH,rmseMeanTrain,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('RMSE Train')
nameFigure = testID + '_Figure_P3_RMSETrain';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (RMSE Val)
figure, surf(axisH,axisH,rmseMeanVal,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('RMSE Validation')
nameFigure = testID + '_Figure_P3_RMSEVal';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (RMSE Test)
figure, surf(axisH,axisH,rmseMeanTest,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('RMSE Test')
nameFigure = testID + '_Figure_P3_RMSETest';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (MAPE Train)
figure, surf(axisH,axisH,mapeMeanTrain,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MAPE Train')
nameFigure = testID + '_Figure_P3_MAPETrain';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (MAPE Val)
figure, surf(axisH,axisH,mapeMeanVal,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MAPE Validation')
nameFigure = testID + '_Figure_P3_MAPEVal';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Main Plot (MAPE Test)
figure, surf(axisH,axisH,mapeMeanTest,'FaceAlpha',faceAlpha)
xlabel('HNs in HL 1'), ylabel('HNs in HL 2'), zlabel('MAPE Test')
nameFigure = testID + '_Figure_P3_MAPETest';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Save Workspace
save(testID + '_Workspace_P3','-v7.3');

end
```

C3.5: Training best performing neural network – One hidden layer

```
function F5_A7_Optimal_1HL(testID,algorithm,mT,optH,minRep,maxRep)

% Identification
testID = testID + '_Optimal_' + algorithm + '_' + '1HL';

% Iteration Bounds
diffRep = maxRep - minRep + 1;

% Variable Allocation
XM = mT(:,1);
XO = mT(:,2);
FG = mT(:,3);
FF = mT(:,4);
TC = mT(:,5);
PC = mT(:,6);
WG = mT(:,7);
input = [XM XO FG FF TC PC]';
target = [WG]';

% Network Configuration
netArray = ObjectArray.empty;
trArray = ObjectArray.empty;
outArray = ObjectArray.empty;
perfMSE = zeros(diffRep,1);
perfMAE = zeros(diffRep,1);
perfSSE = zeros(diffRep,1);
perfSAE = zeros(diffRep,1);
perfMAPE = zeros(diffRep,1);
perfRMSE = zeros(diffRep,1);
e = ObjectArray.empty;
eMSE = zeros(diffRep,1);
rArray = zeros(diffRep,1);
mArray = zeros(diffRep,1);
bArray = zeros(diffRep,1);
mseTrain = zeros(diffRep,1);
mseVal = zeros(diffRep,1);
mseTest = zeros(diffRep,1);
rTrain = zeros(diffRep,1);
rVal = zeros(diffRep,1);
rTest = zeros(diffRep,1);
rmseTrain = zeros(diffRep,1);
rmseVal = zeros(diffRep,1);
rmseTest = zeros(diffRep,1);
mapeTrain = zeros(diffRep,1);
mapeVal = zeros(diffRep,1);
mapeTest = zeros(diffRep,1);

% Training Repetitively - 1 Hidden Layer
for i = 1:diffRep

    ticIT = tic;
    disp(algorithm + "-" + i)

    % Network Configuration
    net = feedforwardnet(optH,algorithm);
    net.divideParam.trainRatio = 0.7;
    net.divideParam.valRatio = 0.2;
    net.divideParam.testRatio = 0.1;
    net.trainParam.showWindow = 0;
    net = configure(net,input,target);
```

```
    % Train
    [net, tr] = train(net,input,target);

    % Predict
    out = net(input);

    % Performance
    perfMSE(i) = mse(net,target,out);
    perfMAE(i) = mae(net,target,out);
    perfSSE(i) = sse(net,target,out);
    perfSAE(i) = sae(net,target,out);
    perfMAPE(i) = mean((abs(target-out))./target)*100;
    perfRMSE(i) = sqrt(perfMSE(i));
    e(i) = gsubtract(con2seq(out(1,:)'),target(1,:)');
    eMSE(i) = immse(target,out);
    [rArray(i),mArray(i),bArray(i)] = regression(target,out);

    % Performance Sub Datasets
    indexOutTrain = out(tr.trainInd');
    indexOutVal = out(tr.valInd');
    indexOutTest = out(tr.testInd');
    indexTargetTrain = target(tr.trainInd');
    indexTargetVal = target(tr.valInd');
    indexTargetTest = target(tr.testInd');
    mseTrain(i) = immse(indexTargetTrain,indexOutTrain);
    mseVal(i) = immse(indexTargetVal,indexOutVal);
    mseTest(i) = immse(indexTargetTest,indexOutTest);
    rTrain(i) = regression(indexTargetTrain,indexOutTrain);
    rVal(i) = regression(indexTargetVal,indexOutVal);
    rTest(i) = regression(indexTargetTest,indexOutTest);
    rmseTrain(i) = sqrt(mseTrain(i));
    rmseVal(i) = sqrt(mseVal(i));
    rmseTest(i) = sqrt(mseTest(i));
    mapeTrain(i) = mean((abs(indexTargetTrain -indexOutTrain))./indexTargetTrain )*100;
    mapeVal(i) = mean((abs(indexTargetVal -indexOutVal))./indexTargetVal )*100;
    mapeTest(i) = mean((abs(indexTargetTest -indexOutTest))./indexTargetTest )*100;

    % Save to Array
    netArray(i) = ObjectArray(net);
    trArray(i) = ObjectArray(tr);
    outArray(i) = ObjectArray(out);

    % Display Iteration Results
    disp("Test MSE = " + mseTest(i))
    disp("Test R^2 = " + rTest(i))
    disp("Train elapsed time = " + toc(ticIT) + " s")

end

% Select Best Performing Network
optRep = find(mseVal == min(mseVal));
optNet = netArray(optRep).Value;
optTr = trArray(optRep).Value;
optOut = outArray(optRep).Value;

% Plot - Performance VS Architecture
P4_A7_Optimal(testID,optNet,optTr,input,target,optOut)

% Save Workspace
save(testID + '_Workspace','-v7.3');

end
```

C3.6: Training best performing neural network – Two hidden layers

```
function F6_A7_Optimal_2HL(testID,algorithm,mT,optH1,optH2,minRep,maxRep)

% Identification
testID = testID + '_Optimal_' + algorithm + '_' + '2HL';

% Iteration Bounds
diffRep = maxRep - minRep + 1;

% Variable Allocation
XM = mT(:,1);
XO = mT(:,2);
FG = mT(:,3);
FF = mT(:,4);
TC = mT(:,5);
PC = mT(:,6);
WG = mT(:,7);
input = [XM XO FG FF TC PC]';
target = [WG]';

% Network Configuration
netArray = ObjectArray.empty;
trArray = ObjectArray.empty;
outArray = ObjectArray.empty;
perfMSE = zeros(diffRep,1);
perfMAE = zeros(diffRep,1);
perfSSE = zeros(diffRep,1);
perfSAE = zeros(diffRep,1);
perfMAPE = zeros(diffRep,1);
perfRMSE = zeros(diffRep,1);
e = ObjectArray.empty;
eMSE = zeros(diffRep,1);
rArray = zeros(diffRep,1);
mArray = zeros(diffRep,1);
bArray = zeros(diffRep,1);
mseTrain = zeros(diffRep,1);
mseVal = zeros(diffRep,1);
mseTest = zeros(diffRep,1);
rTrain = zeros(diffRep,1);
rVal = zeros(diffRep,1);
rTest = zeros(diffRep,1);
rmseTrain = zeros(diffRep,1);
rmseVal = zeros(diffRep,1);
rmseTest = zeros(diffRep,1);
mapeTrain = zeros(diffRep,1);
mapeVal = zeros(diffRep,1);
mapeTest = zeros(diffRep,1);

% Training Repetitively - 2 Hidden Layers
for i = 1:diffRep

    ticIT = tic;
    disp(algorithm + "-" + i)

    % Network Configuration
    net = feedforwardnet([optH1 optH2],algorithm);
    net.divideParam.trainRatio = 0.7;
    net.divideParam.valRatio = 0.2;
    net.divideParam.testRatio = 0.1;
    net.trainParam.showWindow = 0;
    net = configure(net,input,target);

    % Train
```

```
    [net, tr] = train(net,input,target);

    % Predict
    out = net(input);

    % Performance
    perfMSE(i) = mse(net,target,out);
    perfMAE(i) = mae(net,target,out);
    perfSSE(i) = sse(net,target,out);
    perfSAE(i) = sae(net,target,out);
    perfMAPE(i) = mean((abs(target-out))./target)*100;
    perfRMSE(i) = sqrt(perfMSE(i));
    e(i) = gsubtract(con2seq(out(1,:)'),target(1,:)');
    eMSE(i) = immse(target,out);
    [rArray(i),mArray(i),bArray(i)] = regression(target,out);

    % Performance Sub Datasets
    indexOutTrain = out(tr.trainInd');
    indexOutVal = out(tr.valInd');
    indexOutTest = out(tr.testInd');
    indexTargetTrain = target(tr.trainInd');
    indexTargetVal = target(tr.valInd');
    indexTargetTest = target(tr.testInd');
    mseTrain(i) = immse(indexTargetTrain,indexOutTrain);
    mseVal(i) = immse(indexTargetVal,indexOutVal);
    mseTest(i) = immse(indexTargetTest,indexOutTest);
    rTrain(i) = regression(indexTargetTrain,indexOutTrain);
    rVal(i) = regression(indexTargetVal,indexOutVal);
    rTest(i) = regression(indexTargetTest,indexOutTest);
    rmseTrain(i) = sqrt(mseTrain(i));
    rmseVal(i) = sqrt(mseVal(i));
    rmseTest(i) = sqrt(mseTest(i));
    mapeTrain(i) = mean((abs(indexTargetTrain -indexOutTrain ))./indexTargetTrain )*100;
    mapeVal(i) = mean((abs(indexTargetVal -indexOutVal ))./indexTargetVal )*100;
    mapeTest(i) = mean((abs(indexTargetTest -indexOutTest ))./indexTargetTest )*100;

    % Save to Array
    netArray(i) = ObjectArray(net);
    trArray(i) = ObjectArray(tr);
    outArray(i) = ObjectArray(out);

    % Display Iteration Results
    disp("Test MSE = " + mseTest(i))
    disp("Test R^2 = " + rTest(i))
    disp("Train elapsed time = " + toc(ticIT) + " s")

end

% Select Best Performing Network
optRep = find(mseVal == min(mseVal));
optNet = netArray(optRep).Value;
optTr = trArray(optRep).Value;
optOut = outArray(optRep).Value;

% Plot - Performance VS Architecture
P4_A7_Optimal(testID,optNet,optTr,input,target,optOut)

% Save Workspace
save(testID + '_Workspace','-v7.3');

end
```

C4: Model implementation

C4.1: Network accuracy evaluation

```
function P4_A7_Optimal(testID,net,tr,inputs,targets,outputs)

% Dataset Allocation
setInTrain = inputs(tr.trainInd');
setInVal = inputs(tr.valInd');
setInTest = inputs(tr.testInd');
setOutTrain = outputs(tr.trainInd');
setOutVal = outputs(tr.valInd');
setOutTest = outputs(tr.testInd');
setTargetTrain = targets(tr.trainInd');
setTargetVal = targets(tr.valInd');
setTargetTest = targets(tr.testInd');

% Plot Performance
figure
plotperform(tr)
nameFigure = testID + '_Figure_P4_Perform';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Plot Training State
figure
plottrainstate(tr)
nameFigure = testID + '_Figure_P4_TrainState';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Plot Regression
figure
plotregression(setTargetTrain,setOutTrain,'Train',setTargetVal,setOutVal,'Validation',se
tTargetTest,setOutTest,'Test',targets,outputs,'All')
nameFigure = testID + '_Figure_P4_Reg';
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Save Workspace
save(testID + '_Workspace_P4','-v7.3');

end
```

C4.2: Plots for engine operation evaluation

```
function P5_A7_ModelPlotting(testID)

% Identification
testID = testID + '_ModelPlot';

% Load Optimal Network
load('A7_T08_Optimal_trainlm_2HL_Workspace.mat','optNet');

% Variable Bounds
minXM = 59;
maxXM = 100;
sXM = 1;
```

```
dXM = (maxXM - minXM)/sXM;
minFG = 199;
maxFG = 451;
sFG = 1;
dFG = (maxFG - minFG)/sFG;
minTC = 19;
maxTC = 51;
sTC = 1;
dTC = (maxTC - minTC)/sTC;
minPC = 89;
maxPC = 101;
sPC = 0.1;
dPC = (maxPC - minPC)/sPC;

% Variables Default
defXM = 70;
defXO = 0;
defFG = 415;
defFF = 260;
defTC = 44;
defPC = 92;

% Plot WG vs XM
XM = zeros(dXM,1);
XO = zeros(dXM,1);
FG = zeros(dXM,1);
FF = zeros(dXM,1);
TC = zeros(dXM,1);
PC = zeros(dXM,1);
WG = zeros(dXM,1);
for i = 1:dXM
    disp("i = " + i)
    XM(i) = minXM + sXM*i;
    XO(i) = defXO;
    FG(i) = defFG;
    FF(i) = defFF;
    TC(i) = defTC;
    PC(i) = defPC;
    input = [XM(i) XO(i) FG(i) FF(i) TC(i) PC(i)]';
    WG(i) = optNet(input);
    if WG(i) < 0
        WG(i) = 0.0001;
    end
end
figure
plot(XM,WG')
xlabel('X CH4 (%)'), ylabel('Power (kW)')
nameFigure = testID + "_Figure_ModelPlot_XM";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

% Plot WG vs FG
XM = zeros(dFG,1);
XO = zeros(dFG,1);
FG = zeros(dFG,1);
FF = zeros(dFG,1);
TC = zeros(dFG,1);
PC = zeros(dFG,1);
WG = zeros(dFG,1);
for i = 1:dFG
    disp("i = " + i)
    XM(i) = defXM;
    XO(i) = defXO;
    FG(i) = minFG + sFG*i;
    FF(i) = defFF;
```

```
    TC(i) = defTC;
    PC(i) = defPC;
    input = [XM(i) XO(i) FG(i) FF(i) TC(i) PC(i)]';
    WG(i) = optNet(input);
    if WG(i) < 0
        WG(i) = 0.0001;
    end
end
figure
plot(FG,WG')
xlabel('Flow to Generator (m^3/hr)'), ylabel('Power (kW)')
nameFigure = testID + "_Figure_ModelPlot_FG";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

% Plot WG vs XM & FG
XM = zeros(dXM,1);
XO = zeros(dXM,dFG);
FG = zeros(dFG,1);
FF = zeros(dXM,dFG);
TC = zeros(dXM,dFG);
PC = zeros(dXM,dFG);
WG = zeros(dXM,dFG);
for i = 1:dXM
    for j=1:dFG
        disp("i = " + i + " -- j = " + j)
        XM(i) = minXM + sXM*i;
        XO(i,j) = defXO;
        FG(j) = minFG + sFG*j;
        FF(i,j) = defFF;
        TC(i,j) = defTC;
        PC(i,j) = defPC;
        input = [XM(i) XO(i,j) FG(j) FF(i,j) TC(i,j) PC(i,j)]';
        WG(i,j) = optNet(input);
        if WG(i,j) < 0
            WG(i,j) = 0.0001;
        end
    end
end
figure, surf(XM,FG,WG')
xlabel('X CH4 (%)'), ylabel('Flow to Generator (m3/hr)'), zlabel('Power (kW)')
nameFigure = testID + "_Figure_ModelPlot_XMFG";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

% Plot WG vs TC
XM = zeros(dTC,1);
XO = zeros(dTC,1);
FG = zeros(dTC,1);
FF = zeros(dTC,1);
TC = zeros(dTC,1);
PC = zeros(dTC,1);
WG = zeros(dTC,1);
for i = 1:dTC
    disp("i = " + i)
    XM(i) = defXM;
    XO(i) = defXO;
    FG(i) = defFG;
    FF(i) = defFF;
    TC(i) = minTC + sTC*i;
    PC(i) = defPC;
    input = [XM(i) XO(i) FG(i) FF(i) TC(i) PC(i)]';
    WG(i) = optNet(input);
    if WG(i) < 0
        WG(i) = 0.0001;
```

```
    end
end
figure
plot(TC,WG')
xlabel('Temperature (°C)'), ylabel('Power (kW)')
nameFigure = testID + "_Figure_ModelPlot_TC";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

% Plot WG vs PC
XM = zeros(dPC,1);
XO = zeros(dPC,1);
FG = zeros(dPC,1);
FF = zeros(dPC,1);
TC = zeros(dPC,1);
PC = zeros(dPC,1);
WG = zeros(dPC,1);
for i = 1:dPC
    disp("i = " + i)
    XM(i) = defXM;
    XO(i) = defXO;
    FG(i) = defFG;
    FF(i) = defFF;
    TC(i) = defTC;
    PC(i) = minPC + sPC*i;
    input = [XM(i) XO(i) FG(i) FF(i) TC(i) PC(i)]';
    WG(i) = optNet(input);
    if WG(i) < 0
        WG(i) = 0.0001;
    end
end
figure
plot(PC,WG')
xlabel('Pressure (kPa)'), ylabel('Power (kW)')
nameFigure = testID + "_Figure_ModelPlot_PC";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

% Plot WG vs TC & PC
XM = zeros(dTC,dPC);
XO = zeros(dTC,dPC);
FG = zeros(dTC,dPC);
FF = zeros(dTC,dPC);
TC = zeros(dTC,1);
PC = zeros(dPC,1);
WG = zeros(dTC,dPC);
for i = 1:dTC
    for j=1:dPC
        disp("i = " + i + " -- j = " + j)
        XM(i,j) = defXM;
        XO(i,j) = defXO;
        FG(i,j) = defFG;
        FF(i,j) = defFF;
        TC(i) = minTC + sTC*i;
        PC(j) = minPC + sPC*j;
        input = [XM(i,j) XO(i,j) FG(i,j) FF(i,j) TC(i) PC(j)]';
        WG(i,j) = optNet(input);
        if WG(i,j) < 0
            WG(i,j) = 0.0001;
        end
    end
end
figure
surf(TC,PC,WG')
xlabel('Temperature (°C'), ylabel('Pressure (kPa)'), zlabel('Power (kW)')
```

```
nameFigure = testID + "_Figure_ModelPlot_TCPC";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

end
```

C4.3: Engine performance improvement

```
% Identification
testID = testID + '_DailyProfile';

% Load Optimal Network
load('A7_T08_Optimal_trainlm_2HL_Workspace.mat','optNet');

% Variable Allocation
XM = data(:,1);
XO = data(:,2);
FG = data(:,3);
FF = data(:,4);
TC = data(:,5);
PC = data(:,6);
tWG = data(:,7);
input = [XM XO FG FF TC PC]';

% Improved Performance Prediction
for i=1:48
    oWG(i) = optNet(input(:,i));
    if tWG(i) < 30
        oWG(i) = 0;
    end
end

% Error Bar Calculation
err = (1-0.98498)*oWG;

% Plot
figure
plot(0.5:0.5:24,tWG,'.-')
hold on
errorbar(0.5:0.5:24,oWG,err,'.-')
xlim([1 24])
ylim([0 1100])
xlabel('Hours'), ylabel('Power (kW)')
legend('Normal Operation','Improved Operation')

% Integration to Determine kWh
tTrapz = trapz(tWG)/2;
oTrapz = trapz(oWG)/2;
```

C5: Model verification

```
function F7_A7_Verify(testID,vT)

% Identification
testID = testID + '_Verification';

% Load Optimal Network
load('A7_T08_Optimal_trainlm_2HL_Workspace.mat','optNet');

% Variable Allocation
XM = vT(:,1);
XO = vT(:,2);
FG = vT(:,3);
FF = vT(:,4);
TC = vT(:,5);
PC = vT(:,6);
WG = vT(:,7);
input = [XM XO FG FF TC PC]';
target = [WG]';

% Predict
out = optNet(input);

% Performance
perfMSE = mse(optNet,target,out);
perfMAE = mae(optNet,target,out);
perfSSE = sse(optNet,target,out);
perfSAE = sae(optNet,target,out);
perfMAPE = mean((abs(target-out))./target)*100;
perfRMSE = sqrt(perfMSE);
e = gsubtract(con2seq(out(1,:)'),target(1,:)');
eMSE = immse(target,out);
[rArray,mArray,bArray] = regression(target,out);

% Plot Verification
scale = 100;
points = 1:size(target,2);
figure;
hold on
plot(points(1:scale),out(1:scale))
plot(points(1:scale),target(1:scale))
xlabel('Data Point')
ylabel('Energy (kW)')
legend('Prediction','Target')
hold off
nameFigure = testID + "_Figure_VPlot";
saveas(gcf,nameFigure,'png')
savefig(nameFigure)

% Plot Verification Regression - Optimal Network
figure
plotregression(out,target)
nameFigure = testID + "_Figure_VReg";
saveas(gcf, nameFigure, 'png')
savefig(nameFigure)

end
```