UNIVERSITY OF CAPE TOWN

MEC5000W

DEPARTMENT OF MECHANICAL ENGINEERING

SUPERVISOR: PROF. RYNO LAUBSCHER
COSUPERVISOR PROF. PIETER ROUSSEAU

Application of probabilistic deep learning models to simulate thermal power plant processes

---

Prepared by:
Renita Anand Raidoo

Student Number:
RDXREN002

June 26, 2022

Submitted to the Department of Mechanical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Masters of Science degree in Mechanical Engineering

Abstract

Deep learning has gained traction in thermal engineering due to its applications to process simulations, the deeper insights it can provide and its abilities to circumvent the shortcomings of classic thermodynamic simulation approaches by capturing complex inter-dependencies. This works sets out to apply probabilistic deep learning to power plant operations using historic plant data. The first study presented, entails the development of a steady-state mixture density network (MDN) capable of predicting effective heat transfer coefficients (HTC) for the various heat exchanger components inside a utility scale boiler. Selected directly controllable input features, including the excess air ratio, steam temperatures, flow rates and pressures are used to predict the HTCs. In the second case study, an encoder-decoder mixture-density network (MDN) is developed using recurrent neural networks (RNN) for the prediction of utility-scale air-cooled condenser (ACC) backpressure. The effects of ambient conditions and plant operating parameters, such as extraction flow rate, on ACC performance is investigated. In both case studies, hyperparameter searches are done to determine the best performing architectures for these models. Comparisons are drawn between the MDN model versus standard model architecture in both case studies. The HTC predictor model achieved 90% accuracy which equates to an average error of 4.89 $\frac{W}{m^2K}$ across all heat exchangers. The resultant time-series ACC model achieved an average error of 3.14 kPa, which translate into a model accuracy of 82%.

Keywords— Air-cooled condensers, Natural convection boilers Time-series prediction, Deep learning, Mixture density networks, Recurrent neural networks,

Declaration

I, Renita Raidoo, hereby declare the work contained in this dissertation to be my own. All information which has been gained from various journal articles, text books or other sources has been referenced accordingly. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof. I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed by candidate

Renita Anand Raidoo

28-06-2022

Date

Acknowledgements

First and foremost, I would like to extend my deepest gratitude to my supervisor, Dr Ryno Laubscher for his expert knowledge, advice and continuous guidance throughout this project. Dr Laubscher's patience, motivation and enthusiasm during this study and research was vital to the success of this project. Secondly, I would like to thank my funders, the Eskom Power Plant Engineering Institute (EPPEI) for their support and collaboration during my research. I would also like to thank the South African Weather Service (SAWS) for their data contributions to this research.

Finally, I would like to thank my family, friends and the ATProM team for their encouragement and support throughout this project.

Contents

List of Figures

List of Tables

Nomenclature

α	Excess air ratio
$\bar{a}$	Activation
$\bar{b}$	Bias parameter
$\bar{w}$	Weight parameter
$\bar{X}$	Vector of inputs
$\bar{Y}$	Target value vector
$\bar{Z}$	Linear equation output
β_1	Momentum exponential decay rate
β_2	RMSProp exponential decay rate
$\dot{m}$	Mass flow rate
$\dot{Q}$	Heat transfer rate
ϵ	Epsilon non-zero coefficient (1e-08)
Γ	GRU gate
$\hat{Y}$	Predicted value vector
μ	Mean prediction
$\overrightarrow{CV}$	Context vector
$\overrightarrow{c}$	Memory cell
π	Probabilities / mixing coefficients
σ	Standard deviation
θ	Heat transfer coefficient
A	Output of activation function
$d_{targets}$	Number of predicted variables
g	Activation function
h	Enthalpy
J	Cost function

l	A given layer
M	Molar mass
m	Mass
n	Number of neurons
P	Pressure
Q	Heat load
req	”Required” abbreviation
S_{db}	Second moment vector for bias parameter
S_{dW}	Second moment vector for weight parameter
T	Temperature
V_{db}	First moment vector for bias parameter
V_{dW}	First moment vector for weight parameter
W	Trainable parameter weight matrix
X	Input data
Y	Output data
Y_{Ash}	Ash component of gas composition
Y_C	Carbon component of gas composition
Y_{fg}	Flue gas composition
Y_H	Hydrogen component of gas composition
Y_{N_2}	Nitrogen component of gas composition
Y_{O_2}	Oxygen component of gas composition
Y_S	Sulphur component of gas composition
z	Latent variable

List of Acronyms

ACC Air cooled condenser

Adam Adaptive moment estimation

ANN Artificial neural network

ATT Attemperator

CFD Computational fluid dynamics

CNN Convolutional neural network

FC Fully connected layer

GRU Gated recurrent unit

HPT High pressure turbine

HTC Heat transfer coefficient

LPT Low pressure turbine

LSTM long-short term memory

MAE Mean absolute error

MAPE Mean absolute percentage error

MDN Mixture density network

MLP Multi-layer perceptron

MSE Mean squared error

RH Reheater

RNN Recurrent neural network

SGD Stochastic gradient descent

SH Superheater

SI Sensitivity index

SVM Support vector machine

1 Introduction

1.1 Background and motivation

Recently there has been a strong movement away from fossil fuel based energy sources, with green energy technology gaining momentum in the past decade. The combustion of fossil fuels are one of the greatest contributors to accelerated global warming [6], and many countries are making concerted efforts to employ greener power generation to supply national grids. However, coal-fired power plants are still dominant in countries without the resources for these new technologies [38]. Since conventional hydrocarbon-based fuels are still the primary source of thermal power generation, efforts should be made to improve their efficiency. This is especially true during low-load and flexible operation as is required by electrical grids with a sizeable portion of intermittent green energy being included.

There currently exists the need to bridge flexible plant operation of national grids as more renewable energy solutions permeate the market with efforts to reduce rising carbon emissions [9]. Flexible operation in the context of conventional coal-fired power plants requires the ability to predict and forecast the performance of the plant operations. This allows for operators to make informed decisions to ensure the plants are running optimally, such that the grid experiences seamless transitions in power sources [9]. The ability to forecast plant performance is also an important tool in countries experiencing load deficits, since it would allow for accurate capacity budgeting and pre-planning for potential shortfalls. To achieve this, advanced analytic capabilities are required to predict the current state of indirect plant operations and to forecast the running state of the plant into the future. Machine learning is one such method of advanced analytics that lends itself to fast, computationally inexpensive forecasting and is the main focal point of this research.

Artificial neural networks in machine learning are systems of software that mimic the operation of human brain neurons [8]. They are capable of learning complex pattern recognition, signal processing and even data synthesis. The algorithms being explored in the present study are used for regression tasks, where the desired outputs are continuous numeric values which correlate to some predictable feature. The deep learning approaches applied in the present work are multi-layer perceptrons (MLP), mixture-density networks (MDN) and recurrent neural networks (RNN). MLPs are the archetypal form of machine learning network which comprise of stacked, fully connected layers. RNNs are a family of neural networks for processing sequential data developed by Rumelhart et al. [17]. These algorithms are used for time-series forecasting since their architecture allows them to capture the temporal behaviour of data by understanding the context of the data in time. The present work uses the MDN algorithm as the framework for modelling conditional probability distributions. For any given value of x, the mixture model provides a general formalism for modelling an arbitrary conditional density function [8]. The reason for using these MDN models in the present work is their ability to capture the multi-modality of power plant operations and provide a measure of uncertainty based on modelling multiple Gaussian distributions for a single output signal. These models can be used as fast and effective alternatives to traditional, theoretical models since they are solely based on the data and can capture plant operation that conventional approaches overlook.

In recent years, machine learning and deep learning have been used extensively in areas of scientific study such as neuroscience [7], geosciences [48] and agricultural sciences [22]. Within this, they have been used for a wide range of applications, such as pattern classification, pattern recognition, optimisation, prediction and automatic control. More specifically to power plants, machine learning has been used for system enhancement and optimisation [40], health monitoring [21] and complex thermal management [32].

The current work aims to demonstrate the efficacy and performance of deep learning methodologies applied to power plant operating data. The author proposes two models to showcase the applications to both stead-state and dynamic plant data. The research is presented over two case studies; the first being static heat transfer predictions for an actual coal-fired boiler and the second being time-series forecasting of air-cooled condenser back pressure.

The first case study proposes a data-driven method using a MDN approach to predict effective heat transfer coefficients of the flue gas-to-water heat exchangers for a coal-fired power plant with minimal plant data. Boiler heat exchangers are subjected to extreme conditions which can lead to problems such as slagging, fouling, caustic embrittlement, fire-side corrosion and thermal fatigue failure with the latter being one of the main reasons for unwanted plant downtime due to failures [42]. Heat exchangers can also experience rating problems when the intended design heating duty is not met by the actual heat absorption rate of the tube banks. The effects of ash deposition and non-uniform flue gas flow can reduce the heat transfer from the gas to the heating surfaces. This can result in higher ash deposition rates on downstream heat exchangers, lower boiler thermal efficiency and higher fuel demand [3]. For these reasons, the thermal performance of boiler heat exchangers should be monitored since estimating the residual life of boiler tubes and heat exchanger effectiveness is crucial to safe and efficient operation. Managing these failures can also help reduce unplanned outages and therefore improve plant reliability and availability [42].

This can be done by estimating heat exchanger effectiveness and tube metal temperatures. Deep learning models trained on actual plant measurement data are a fast and direct method to achieve this, since conventional empirical process models struggle to capture the various operational effects in the boiler such as non-uniform flow fields and deposition layers. A model capable of forecasting thermal performance can enable the study of historic metal temperatures in the boiler sections, which can be used to estimate residual life and the effects of controllable parameters on heat exchanger performance. This can be achieved by using the predicted heat exchanger heat transfer coefficients along with process models to estimate heat exchanger characteristics.

In the proposed methodology, heat transfer coefficients are predicted using data from an actual subcritical coal-fired boiler in this case study. The data from the plant, which is located in Southern Africa, spans a period of 336 days. The data in this study utilises 70 inputs for the calculations used to estimate the heat transfer coefficients. Combustion calculations are done initially to determine the furnace and flame temperatures and the mass flow rates of the fuel and air. Mass and energy balances are done to determine the various temperatures and enthalpies at each of the boiler stages. The required heat transfer rates are determined, and subsequently the heat transfer coefficients at the various heat exchangers of the boiler system. A MLP-MDN model is then developed using only 13 input features to predict the HTCs for the respective heat exchangers.

Case study two addresses the issue of time-series forecasting applied to air-cooled condensers (ACC) used in thermal power plants. Many thermal power plants utilise dry-cooling technologies which can often be a major contributor to load deficits due to increased steam pressure at the low pressure turbine exit (called turbine backpressure) [30]. This is caused directly by the inability of the ACC to reject heat due to ambient and operating conditions. The main factors influencing the performance of ACCs are ambient temperature, air relative humidity, wind speed, wind direction, steam flow rate and steam quality existing the turbines [31]. Current methods of determining future ACC performance, such as traditional mathematical process models and CFD (computational fluid dynamics), are impractical since they require the solving of hundreds to millions of non-linear equations with simplifying assumptions that lead to unquantifiable uncertainty and irregularities [21]. These models are also very time-consuming and computationally expensive and cannot be applied to near real-time forecasting. ACC performance can alternatively be predicted using machine learning techniques, which circumvents the need to make simplifying assumptions and explicitly formulate the various physical relationships [13].

In this study, an ensemble time-series deep learning model is developed which is capable of predicting future ACC backpressure using meteorological and operational data. The model uses recurrent neural networks as well as the MDN approach to gain insights into the uncertainty of predictions. The backpressure of an actual 657 MWe power plant ACC is forecast 4 hours into the future at 15 minute increments. The dataset includes weather data from the South African Weather Service and plant operating data for 164 days. Various models with increasing complexity are developed to investigate the effects of 3 datasets and two model architectures. MDN-RNN models are compared to standard RNN models to explore the prevalence of multi-modality in power plant operational domains. Ensemble methods are also applied to find the best combination of model architectures.

1.2 Problem statement and plan of development

To achieve the goal of developing a steady-state MLP-MDN model capable of predicting heat transfer coefficients for boiler heat exchangers, the following was done:

- Heat transfer coefficients for each boiler segment were determined using mass and energy balance calculations based on plant data.
- A MLP-MDN model architecture was defined and optimised using hyperparameter tuning.
- The model was trained using a training set and then validated using a test set.
- A sensitivity study was conducted to gain insight into the individual effects of the feature parameters on model performance.

To achieve the goal of developing a time-series RNN-MDN model capable of forecasting ACC backpressure, the following was done:

- Data pre-processing was performed on the raw dataset to improve its quality.

- Standard RNN models were developed for each of the dataset sizes and optimised using hyperparameter tuning.
- The models were adapted to include a MDN and further tuning was conducted to improve the model's accuracy.
- The models were trained using a training set and then validated using a test set.

This thesis details the development and testing of the two aforementioned machine learning models. This thesis is divided into five chapters: Ch. 2 describes the materials and methods used, Ch. 3 and 4 detail the two proposed case study models and Ch. 5 outlines the conclusions drawn.

2 Literature review

Machine learning and deep learning have gained traction in the field of engineering research in recent years. More specifically, it has shown increasing successes in applications applied to energy systems [13], [40]. Many recent studies in the field of thermal engineering and plant performance monitoring have incorporated the use of machine learning as an alternative to classic methods such as analytical and numerical methods.

A study from Herawan et al. [20] shows the success of using an ANN to predict the power generation of a waste heat recovery system. Their work highlights the method's ability to perform in highly fluctuating environments with multiple inputs affecting the output, as is common with thermal combustion systems. Other uses of machine learning in power plants include system enhancement and optimisation [40], health monitoring [21] and thermal management [32]. Wu et al. [49] used generative adversarial networks coupled with convolutional neural networks to predict pressure profiles on air-foil surfaces for different, fixed, Mach and Reynolds numbers. They demonstrate how data-driven, machine learning approaches can be used to model physical systems within fluid dynamics. Using wind tunnel data, they were able to visualise how close the model predictions were, compared to simulated real life flow profiles. Similarly to the present work, they explore the effects of model architecture on predictive ability and show that hyperparameter tuning is crucial to good model performance. By comparing the model's outputs to CFD generated images, which are inherently resource intensive, they clearly illustrate the benefits of machine learning in terms of computational expense.

Laubscher and Rousseau [27], [29] used variational autoencoders and deep neural networks to predict temperatures, species concentrations and gas velocities inside a methane combustor using CFD simulation data. This application also explored the use of MLPs and convolutional networks to extract features from high-dimensional spaces such as high-definition contours with large cell counts. However this application incorporated CNN layers into a variational autoencoder which allowed them to extract lower latent spaces capable of encoding the contours for use in generative models. These applications of convolutional neural networks to CFD analysis are not directly applicable to the present work, however they do highlight significant advancements in the thermal energy and deep learning field. Closer to the time-series nature of the present work, Laubscher [25] used recurrent neural networks to predict reheater metal temperatures of a coal-fired boiler and showed that the model could accurately predict temperatures at multiple locations 5 minutes into the future. This paper evaluates both gated recurrent units and long-short-term memory units, both capable of capturing the time context within a dataset. The application of this type of neural network to thermal signals is noteworthy since it gives insights into the temporal relationship of power plant operations. Laubscher found that GRUs outperformed LSTMS since these models had consistently lower MAPEs. The research done also provides a compelling intuition into the effects of forecast period length, which is related to the present research in a sense that there are trade-offs between early anomaly notice versus model accuracy as a function of time.

Similarly to Laubscher, Hu et al. [1] made use of a short-term memory-based autoencoder network for early detection of anomalies. The study showed excellent results, a MAPE of 0.027%, which indicate that the framework can be used as a legitimate tool for plant performance monitoring and

anomaly detection. Tan et al. [45] also used long short-term memory neural networks to study plant operations. Their work aimed to predict NOx emissions for a 660MW thermal power plant. An important point the authors raise is that when modelling complex thermal operations, such as the combustion process in their research, it is important to ensure the train and test data-sets both contain a sufficient variety of operating profiles to ensure the model gains good generalizability. As a comparison, they evaluated the LSTM's performance against a support vector machine (SVM) which is a widely used regression approach in machine learning. They did however determine that the LSTM far out-performed the SVM in terms of overall accuracy.

Lv et al. [33] used support vector regression and transient operational data of a circulating fluidized bed combustion system to develop a model capable of predicting the bed temperatures. In their work, they evaluate the benefits of past values of bed temperatures being fed into the model's inputs as feedback. It was determined that model accuracy is significantly improved when context is provided in terms of historical and dynamic sequences, since the steady-state model was insufficient in describing the dynamic characteristics of the bed temperature of the CFB boiler.

Warey et al. [17] used high fidelity CFD simulation data of thermal comfort in an automotive vehicle to develop a machine learning model capable of predicting the equivalent homogeneous temperature for each passenger and the volume-averaged cabin temperature. This study explored the more commonly used machine learning architectures such as linear regression with stochastic gradient descent, random forests and artificial neural networks. These approaches do not consider the temporal behaviours of their predictions, however it is noteworthy that even these simpler architectures are still capable of determining non-linear relationships, especially since this study involves highly non-uniform thermal environments and inconsistent inputs such as human body heat loss factors. Sun et al. [18] developed a model using genetic algorithms and neural networks to predict the heat transfer behaviour of supercritical-CO2 inside a tube. Their study highlights how notable changes in physical operations can have significant changes on thermal heat-transfer processes in energy systems. It is overtly complex to describe the effects environmental mechanisms have on thermal properties and transfer rates which is why machine learning approaches are being adopted and tested as methods of predicting them. The research by Sun et al. further highlights how even basic methods of machine learning, when coupled with good optimisation algorithms, can be used to determine heat-transfer regimes and thermodynamic performances within an acceptable error range as considered by the engineering field.

Machine learning techniques have also been applied to predict the performance of ACCs. Traditionally, numerical simulations, such as CFD, are used to study the effects of ambient conditions on large scale condenser units. Wind effects have proven to significantly impact the performance of condensers that rely solely on forced convection as a means of heat rejection. J. A. van Rooyen [47] developed a detailed fan unit numerical model to study the effects of wind angle on the ACC unit's cooling abilities. The model was complex in nature and could only be used on a few fans at a time to reduce the computational load. Although the models were able to determine complex volumetric effectiveness, it was only suitable for static research and cannot be used to study wind effects in real-time.

A study by Liu et al. [32] explored the effects of hot-air-recirculation at a large scale power

plant. The experimental scale model used for comparison could not imitate the real world case since it does not consider surrounding buildings. Complex wind flow patterns, coupled with the structural complexity of the ACC had vast effects on the flow-field and subsequently the heat-transfer rates. This type of model proved hugely beneficial to understanding how hot-recirculation rates can affect ACC performance and their research offers insightful recommendations to mitigate this issue, however to generate these results in real-time with the intention of proactively altering the plant's operation is unfeasible. To offer operators a practical solution, Du et al. [12] developed a standard neural network capable of predicting ACC backpressure. The developed model did not consider any temporal effects and only used the current state of the ACC and weather to predict the current backpressure. Their study focused heavily on the weather element of the feature space and considered very few plant operating parameters. Wind speed and direction measurements were included from various recording sites on the ACC.

Haffejee and Laubscher [18] used simulation data generated using a detailed 1D process model, to develop an ACC condition monitoring platform capable of predicting backpressures using deep neural networks. Similar to the work of the authors above, the model was based on steady-state data, making it unsuitable for long-term forecasting of ACC performance, seeing as the model is incapable of predicting upcoming weather changes and the effect it has on the ACC. Although this model has limited predictive capabilities, a valuable recommendation by the authors was to use a wider scope of input parameters which will be considered in the present work. As mentioned in the paper by Tan et al. [45], good generalisation can be achieved by including all operating ranges in the datasets.

Machine learning is also being used as an alternative to complex heat transfer calculations to study heat transfer characteristics of combined mode heat exchangers. Taler et al. [44] developed a simplified quasi-2D model of a radiant superheater to monitor heat exchanger performance which was validated using a CFD model. The proposed method showed good results for determining flue gas, steam and tube wall temperatures using methods that are computationally inexpensive, however the model is heavily reliant on extensive measurement inputs which are not always readily available from plants. Haibo et al. [19] conducted a study on a 300 MWe CFB boiler, using measurements from thermocouples in the boiler system and energy balance calculations to determine HTCs. Their work shows how deviations from the design specifications of coal can have significant impacts on the heat transfer rates within the various boiler components. However, applying this research to real-world problems is complicated, as it is unfeasible to test batches of coal in real-time to study the heat transfer rates in the hot circuit of the boiler. It is also extremely challenging to develop valid process models for industrial applications that can accurately capture the mathematical relationships between the many operating parameters according to the authors' conclusions. The applications of machine learning to utility scale boilers have been explored as alternatives to these traditional methods. Khalid et al. [4]. propose a machine learning-based model integrated with an optimal sensor selection scheme to analyse boiler waterwall tube leakage. The second part of their research is of value to the present work since they validate a supervised learning model using a real power plant leak scenario. They also conducted a study into feature space reduction using a correlation analysis. Similarly in the present work, an analysis is conducted into the effects of input feature size on model performance.

As with any scientific analysis, there exists a degree of uncertainty. Traditional deep learning approaches are unable to account for the multi-modality and uncertainty in the datasets which arises due to measurement noise, random uncertainty, data which has a non-singular Gaussian distribution and fluctuations in unmeasured parameters that influences the system [37]. To predict the performance of systems that are strongly influenced by stochastic signals (such as weather), uncertainty quantification is a major theme [34]. Gaussian processes regression (GPR) models [23] infer a probability distribution over all values, using Bayes' rule, rather than learning exact values for predictions. However recent studies have required access to these multiple output distributions for analysis and GPR models are not suited to deal with multi-modal distributions [11]. They also tend to lose efficiency in high-dimensional spaces which means they are not easily compatible with deep learning time-series modelling frameworks such as sequential RNNs.

Mixture density networks (MDNs) are proposed in the present work as an alternative to this problem. MDNs can predict target values along with their associated uncertainties and account for multi-modality in the data [8]. MDNs have been noticed in recent years as a powerful probabilistic machine learning tool for real-world applications of inverse problems. Felder et al. [14] and Zhang et al. [51] both used mixture density networks and recurrent neural networks (MDN-RNN) to predict expected power generation from wind turbines along with the associated uncertainty of the predictions. Felder et al. [14] compared the MDN model to base line predictors and a MLP. This study highlights the benefits of adding multiple modes when forecasting using the MDN. Zhang et al. compared three probabilistic models, Gaussian mixture model, MDN and a relevance vector machine model. This study found that the GMM performed the best when targeted values are required from a single distribution. The added benefits of the MDN in this case are therefore its ability to sample off various distributions and output the value with the highest probability at any given timestep rather than choosing the best performing distribution. Qian et al. [36] used convolutional MDN-RNNs to predict smartphone user locations using time-series data of WIFI fingerprints. The mentioned research proves how stochastic problems can sometimes have more than one possible outcome for a given input, which further highlights the benefits of the MDN architecture outputting all distributions rather than a single optimised distribution.

Another technique being applied in the present work which has been found in literature is ensemble modelling. Literature shows that authors have applied ensemble learning to MDN models with positive results. Men et al. [34] used an ensemble of MDNs to predict wind turbine power generation, showing that the resultant model outperforms various other time-series machine learning approaches and a single trained MDN model. They noted that the trade-off between the number of chosen modes, K, and the overall neural network structure can have significant impacts on a model's ability to generalise. It was also noted that training time can have significant impacts on overall accuracy and the generation of uncertainty parameters, so an ensemble approach has been considered.

Laptev et al. [24] developed an end-to-end MDN-RNN model using ensemble learning to accurately predict Uber requests during normal and high variance days (holidays). The objective of this study is to demonstrate the efficacy of the MDN framework when applied to time-series forecasting. The benefits of ensembling here showed substantial improvements to predictions over

anomalous periods, like holidays for example. The combined bootstrap model performed well under normal operating conditions and was also able to predict 'out-of-the-ordinary' behaviour due to the combination of two models with these respective strengths. Relevant to the present study; Haffejee and Laubscher [18] and Felder et al. [14] developed an ensemble probabilistic model with the aim of showing its ability to outperform a standard neural network using the same dataset. Their research showed how constituent learning models can be combined to improve predictions on outlying operating conditions in complex systems like power plants.

3 Materials and methods: Machine learning

In this chapter, the details of the machine learning, deep learning and probabilistic extension modules used in the present work will be discussed.

3.1 Multilayer perceptron neural network

A neural network can be described as a directed graph with nodes corresponding to neurons with fully linked connections. Each neuron receives as input, a weighted sum of the outputs from the neurons connected to its incoming edges [41]. The preface to the more complicated deep learning architectures is the multilayer perceptron (MLP) model. MLPs are networks which consist of multiple connected stacked neurons forming layers which in turn are connected to up and down stream layers. MLPs are the classical form of artificial neural networks [17] and are typically applied in supervised learning problems where input features are mapped to labelled output features. The model takes input data $\bar{X}$ and target data $\bar{Y}$ and learns a set of parameters in order to make predictions, $\hat{Y}$. The objective of the MLP is to optimise weight and bias parameters, $\bar{W}$ and $\bar{b}$ according to some cost function to make this prediction as accurate as possible. The trainable parameters have the shapes $\bar{W} \in \mathbb{R}^{n_l \times n_{l+1}}$ and $\bar{b} \in \mathbb{R}^{n_l \times 1}$. To calculate the output, $\hat{Y}$, the forward propagation algorithm is applied.

The forward propagation algorithm is shown in equation 1. Forward propagation involves two steps, determining the summed signal denoted by Z, and the activation signal denoted by A. In the first layer, the activations are taken as the input data such that $A^0 = \bar{X}$. The summed signal is then calculated by multiplying the activation by some weight matrix, $\bar{W}$, and adding a bias factor, $\bar{b}$. This summed signal is then passed through an activation function, which can be any easily computable and differentiable function [41]. Various activation functions exist such as linear, ReLu, leaky-ReLu, hyperbolic-tangent and Elu [16]. Typically, sigmoid functions are used for classification problems and hyperbolic tangent functions for regression problems. Recently however, the ReLU function is the most widely used function due to its resistance to vanishing gradients and dead neurons [35]. For regression models, the final layer activation function is typically selected to be linear. Xu et al. [50] showed that leaky variants (leaky-ReLu/Elu) of the ReLu activation function tend to out perform the standard ReLu function. These activations are more balanced, quicker to converge and tend to learn faster and will be used in this work.

$$\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \\ A^{[L]} &= \hat{Y} \end{aligned} \tag{1}$$

In equation 1, the activation from the previous layer, $[l-1]$, is used to determine $Z^{[l]}$ in the current layer, l. The activation function, g, is applied so that the output of the node is a non-linear transformation of the inputs. This process propagates through the entire network to the final layer, L, where the output activation is the predicted value $\hat{Y}$.

As mentioned, the weights and biases of the MLP are optimised using a cost function. Typically the mean-squared-error (MSE) is selected for regression problems [37]. Equation 2 shows the standard MSE cost function for a neural network [17]. Once a complete forward propagation cycle has finished, the model uses the predicted output to estimate the cost function value which indicates the Euclidean distance between the actual and predicted targets.

$$J_{MSE}(\bar{y}^n, \hat{y}^n) = \frac{1}{d_{target}} \sum_{j=1}^{d_{target}} \frac{1}{2}(\bar{y}^n - \hat{y}^n)^2 \tag{2}$$

In the equation above, d_{target} is the output data dimensionality, $\bar{y}^n$ is the n^{th} observation in the output dataset and $\hat{y}^n$ is the predicted MLP output. Once the forward propagation iteration is completed, the network weights are updated to minimise the cost function using the backwards-propagation algorithm [39] which calculates the gradients of the cost function with respect to the weights and biases in each layer. Backward propagation uses the automatic differentiation algorithm to calculate the gradients of the weights and biases in a neural network graph structure with respect to the cost function value. Training is performed by the process of forward- and backward-propagation which aims to adjust the trainable parameters, W, b.

In the current work, the proposed models are optimised using the Adam first-order optimiser algorithm shown in equations 3 and 5 [17]. Adam is a preferred alternative to the commonly used stochastic gradient descent (SGD) algorithm. SGD is slower to converge and tends to be impractical for larger datasets. Adam combines the advantages of two SGD extensions — Root Mean Square Propagation (RMSProp) and Adaptive Gradient Algorithm (AdaGrad). It is known that the default configuration parameters, β_1 and β_2 do well on most problems and were therefore used in the present work.

$$\begin{aligned} V_{dW} &= \beta_1 V_{dW} + (1-\beta_1) dW \quad & V_{db} &= \beta_1 V_{db} + (1-\beta_1) db \\ S_{dW} &= \beta_2 S_{dW} + (1-\beta_2) dW^2 \quad & S_{db} &= \beta_2 S_{db} + (1-\beta_2) db^2 \end{aligned} \tag{3}$$

$$\begin{aligned} \hat{V}_{dW} &= \frac{V_{dW}}{1-\beta_1^t} \quad & \hat{V}_{db} &= \frac{V_{db}}{1-\beta_1^t} \\ \hat{S}_{dW} &= \frac{V_{dW}}{1-\beta_1^t} \quad & \hat{S}_{db} &= \frac{S_{db}}{1-\beta_1^t} \end{aligned} \tag{4}$$

$$W = W - \alpha \frac{\hat{V}_{dW}}{\sqrt{\hat{S}_{dW}} + \varepsilon} \quad b = b - \alpha \frac{\hat{V}_{db}}{\sqrt{\hat{S}_{db}} + \varepsilon} \tag{5}$$

V_{dW} is defined as the first moment vector, treated as in Momentum. It is found using the derivative of the weights parameter. S_{dW} is the second moment vector, treated as in RMSProp. It is updated using the square of the derivatives as seen in equation 3. In the same sense, V_{db} and S_{db} are found using the derivative of the bias parameter. The β factors used are the exponential decay rates for

the moment estimates. V_{dW}, V_{db}, S_{dW} and S_{db} are initialised at 0 at time $t = 0$. Bias correction is then implemented on both V and S as seen in equation 4. The Adam algorithm aims to iteratively adjust the weights and biases as seen in equation 5 such that the cost function is minimised till a global optimum is found for all parameters. The parameter α is the learning rate which governs the extent to which the parameters are altered per iteration. ε is a constant set to 10^{-8}.

3.2 Mixture density networks

Accessing the uncertainty associated with deep learning model predictions can have multiple uses in practical applications. The general idea of the MDN is that it enables deep learning frameworks to output multiple Gaussian distributions per continuous output variable. Underlying data can be multimodal in nature, so to have a model that outputs single values can be limiting. Bishop [8] first proposed this multimodal approach in 1994.

The dataset used in the present work contains random and instrument uncertainty which results in significant scatter in data for given operational states. This is typical of most engineering datasets. Furthermore, unmeasured phenomena which influences system performance and randomness in weather conditions result in stochastic input and target data [37]. In reality, this means that the same set of input operating features can have multiple different, valid output readings. For this reason, probabilistic machine learning allows us to model the real relationships between input data and target features. Traditional neural network frameworks minimise a single cost function which results in a deterministic model, that is unable to estimate the uncertainty associated with each output prediction [37]. MDNs provide a formal framework for modelling conditional probability distributions. For any given value of x, the mixture model provides a general formalism for modelling an arbitrary conditional density function $p(t|x)$ [8]. The conditional probability function generated by training an MDN is shown in equation 6. The mixing coefficient matrix, $\bar{\pi}$, predicted by the neural network contains the discrete probabilities of an output belonging to each of the K normal distributions for all observations in the training batch [37]. In equation 6, K is the number of selected normal distributions, μ_k is the predicted mean and σ_k^2 is the variance per feature of the distribution k given the input data X which is the entire dataset of shape $N \times d_{inputs}$. Although this work explicitly considers Gaussian distributions, other distributions such as the Bernoulli distribution can alternatively be used.

$$P(\bar{Y}|\bar{X}) = \sum_{k=1}^{k} \pi_k(\bar{X}) \mathcal{N}(\bar{Y}|\bar{\mu_k}(\bar{X}), \bar{\sigma_k}^2(\bar{X})) \tag{6}$$

The MDNs developed in the present study are a probabilistic extension module which outputs the mixture parameters, μ, π, σ for each component k. The resulting MDN output therefore, has shape $K \times d_{targets} \times 3$, where $d_{targets}$ are the number of predicted variables and the 3 represents the mixture parameters. The schematic below in figure 1 shows the general architecture of the MDN model. An arbitrary number of input nodes and hidden layers are shown.

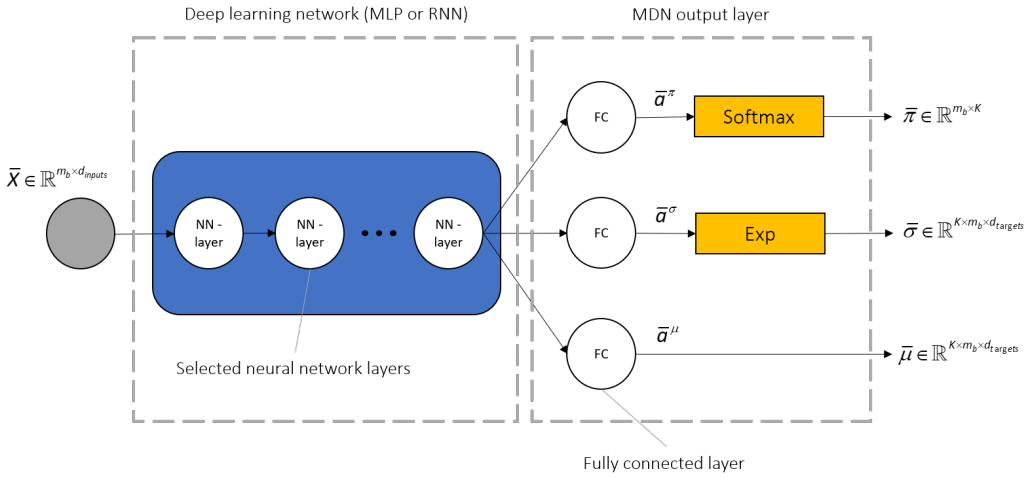


Figure 1: Schematic of simple MDN network

Modifications to the output of the deep learning model are made to enable the model to learn the framework for the conditional probability, $P(\bar{Y}|\bar{X})$. To achieve this, the output of the deep learning network must be split into three parts which are passed into parallel, fully connected layers. This technique enables the components of the MDN to be determined using trainable weight and bias parameters. The mixing coefficients, denoted by π, must satisfy the constraints: $\sum_{k=1}^{k} \pi_k(x) = 1$ and $0 \leq \pi_k(x) \leq 1$ [8]. This is achieved by passing the activation from the fully connected layer through a softmax function as seen in equation 7.

$$\pi_k(x) = \frac{exp(a_k^{\pi,n})}{\sum_{l=1}^{k} exp(a_k^{\pi,n})} \tag{7}$$

The output deviation matrix $\bar{\sigma}$ is used to establish the standard uncertainty for each output feature per observation and distribution k. The constraint on this variable is $\sigma^2 \geq 0$ since standard deviations can only be positive real values. It is therefore represented in terms of the exponential of the corresponding activation shown in equation 8.

$$\sigma_k(x) = exp(a_k^{\sigma}) \tag{8}$$

Since the mean output, $\bar{\mu}$, of each of the $d_{targets}$ features per observation and k distribution are real values, the output of this fully connected layer is simply taken as $\bar{a}^{\mu} = \bar{\mu}$.

The adaptive parameters of the MDN, along with the upstream MLP, comprise of weights and biases which can be optimised by minimising the negative log-likelihood for all observations in the batch as seen in equation 9 below [8]. Negative log-likelihood is a suitable error function for two reasons, the negative multiplier and the log function itself. The standard machine learning optimisation aims to minimise the error, however the MDN aims to maximise the probability of

the predictions. Therefore using a negative function allows the error to be minimised, while maximising the desired output. Logarithmic functions are useful for deep learning optimisation since they enable computers to calculate a large number of products without losing precision if numbers are too small or too large. By minimising the negative log-likelihood, the posterior probability is maximised which results in parameterised neural network model fitting the target distributions by using the known target values for given inputs.

$$J(\bar{Y}, \bar{\pi}, \bar{\sigma}, \bar{\mu}) = -\sum_{n=1}^{d_{target}} ln \left\{ \sum_{k=1}^{k} \pi_k(\bar{x}^n, \bar{W}) \cdot N(\bar{y}^n | \bar{\mu}_k(\bar{x}^n, \bar{W}), \bar{\sigma}_k(\bar{x}^n, \bar{W})) \right\} \tag{9}$$

In order to optimise this function, the derivatives of the error function, $E(W) = J(\bar{Y}, \bar{\pi}, \bar{\sigma}, \bar{\mu})$, need to be calculated with respect to the components of $\bar{W}$. These can be evaluated by using the standard back propagation procedure, provided we obtain suitable expressions for the derivatives of the error with respect to the output unit activations [8].

In order to derive this cost function, it is necessary to show the expansion of the conditional probability function in equation 6. The latent variable, z, is introduced, which is the encoding for the k mode states. Formally, $p(z_k) = \pi_k$. This defines π_k as the probability of z being in a component k_i, and thus the probabilities of the k mixture components. The Gaussian distribution can then be seen as $p(y|x, z)$, where the prediction y is dependent on the input x and the latent variable z. A joint distribution is shown in equation 10.

$$\begin{aligned} p(y|x) &= \int p(y, z|x) dz \\ &= \sum_{k} p(y, z|x) \end{aligned} \tag{10}$$

The conditional distribution $p(y|x)$ can then be updated to explicitly include w.

$$p(y|x, w) = \int p(y, z|x, w) dz \tag{11}$$

Bayes' Theorem is then used to introduce the probabilistic entities shown in equation 12

$$\begin{aligned} p(w|Y, X) &= \frac{p(Y|X, w)p(w)}{p(Y)} \\ p(w|Y, X) &\propto p(Y|X, w)p(w) \\ posterior &\propto likelihood \times prior \end{aligned} \tag{12}$$

The aim is to maximise the posterior with respect to w, and thus define the error function in terms of the likelihood and prior terms. Equation 12 shows the derivation using equations 6, 10, 11 and 12.

$$\begin{aligned} E(w) &= -log(likelihood \times prior) \\ &= -log(p(Y|X,w)p(w)) \\ &= -\frac{1}{N}\sum_{n=1}^{N}(log(\int p(y,z|x,w)dz)) + log(p(w)) \\ &= -\frac{1}{N}\sum_{n=1}^{N}(log(\sum_{k}\pi_k(x^n,w)\cdot N(y^n|\mu_k(x^n,w),\sigma_k(x^n,w)))) + log(p(w)) \end{aligned} \tag{13}$$

A non-informative prior of $p(w) = 1$ is assumed [8]. This simplifies the error function to equation 9 shown above.

3.3 Recurrent neural networks

This section pertains to the second case study only, where time-series forecasting was explored.

Recurrent neural networks (RNNs), are a form of deep learning architecture where the units within the model form connections along a temporal sequence. RNNs typically are used for applications such as language translation, music generation, DNA analysis and activity recognition. The work presented in the second case study requires a model capable of forecasting sequences of backpressures, hence, an encoder-decoder RNN architecture is utilised which enables the use of variable input and output sequence lengths [43]. These models have four parts, an encoder, a context vector, a decoder and a fully-connected output layer. The encoder and decoder layers consist of RNN units. Two of the most popular types of RNN layers are the long short-term memory (LSTM) [36] and gated-recurrent unit (GRU) [37] layers. LSTM's have been used in other power plant applications such as condenser vacuum degree [6] and transformer running state predictions [7]. The present study uses GRU as an alternative to the more widely used LSTMs since their performance is on par however they are computationally more efficient due to the fact that they have fewer parameters to train. [37].

RNN units modify the way in which the activation of a node is determined. The activation function is applied to the parameter w, which is multiplied by the previous activation as well as the current x input, plus some bias variable b. GRUs introduce a new variable C, called the memory cell (not to be confused with the context vector). For this situation, this memory cell is equal to the output activation of the node as seen in equation 18. At each timestep, we consider replacing the current memory cell, $c^{<t>}$ with a new candidate value, $\tilde{c}^{<t>}$, as seen in equation 14. To compute the new candidate, a gate, Γ_r, is used to inform the new value of the memory cell of the relevance of the previous memory cell. This gate value is determined using equation 15 using a parameter matrix, W_r and a bias value b_r. GRU layers utilize another single gate to simultaneously control the

information persistence through time and the memory cell state updating [37]. The update gate, Γ_u can have a value between 0 and 1, and is computed using a sigmoid function and a parameter matrix, W_u and a bias value, b_u as seen in equation 16. In the complete implementation, $c^{<t>}$ is determined using equation 17. Based on the value of the gate, the value of $c^{<t>}$ will either be updated by the new candidate, $\tilde{c}^{<t>}$ or the old value, $c^{<t-1>}$.

$$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \tag{14}$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r) \tag{15}$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u) \tag{16}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \tag{17}$$

$$a^{<t>} = c^{<t>} \tag{18}$$

The internal structure of a GRU can be seen in figure 2 which represents equations 14-18.

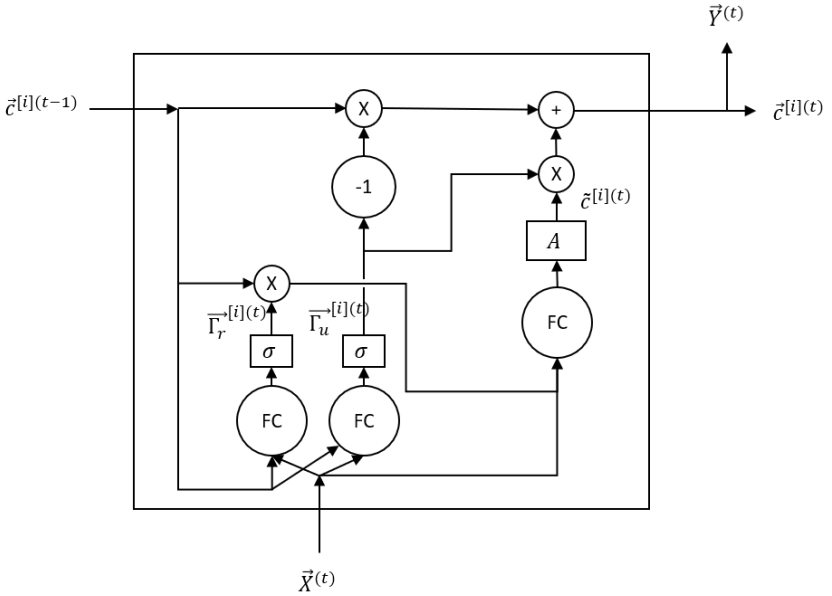


Figure 2: Gated recurrent unit

The weights and biases of RNN layers are optimized using a backwards-propagation algorithm which propagates through the internal time-unfolding of the RNN layers in reverse order. This is called the backwards-propagation through time (BPTT) algorithm [35].

As mentioned before, the encoder and decoder layers and made up of RNN layers. The RNN layers utilize parameter sharing between hidden layer outputs per time step, which makes these special type of neural network layers appropriate for time sequence data learning [37]. Figure 3 shows how parameters are shared since the output from a layer is fed back into that layer. The input data matrix has the shape $\bar{X} \in \mathbb{R}^{m_b \times Tx \times d_{inputs}}$ where m_b is the batch size corresponding to the selected number of observations, Tx is the selected number of time steps for the input data and d_{inputs} is the number of input features. Similarly, for the output data $\bar{Y} \in \mathbb{R}^{m_b \times Ty \times d_{targets}}$, Ty is the number of time steps predicted for the output data and $d_{targets}$ is the output dataset number of features [37].

The encoder and decoder layers comprise of multiple RNN units. A model can then have multiple encoder and decoder layers compiled to improve the prediction capability of the model, by increasing the number of optimised weight parameters. For every entry in a batch of size, m_b the encoder receives an input sequence $\bar{X}^1 \dots \bar{X}^{T_x}$ up to the time step T_x. The encoder outputs a single vector called the context vector $\vec{CV} = \vec{c}_{E,L}^{<T>_x}$, which is a reduced dimensionality representation of the input data and the shared parameters. This vector is then fed into the decoder section, which has a mirrored architecture to the encoder. The decoder section then generates an output sequence from the final RNN layer $\bar{Y}^1 \dots \bar{Y}^{T_y}$ up to the time step T_y where T_y is the final time step-index for the predicted output variable. This process is performed for every batch, m_b.

Training the encoder section of model is no different to training any other deep learning model. Its parameters are updated using back propagation over time. This is because the key role of the encoder is to learn some reduced version of the input features such that the variance of the data is maintained as much as possible. In this sense, the context vector is the penalised trainable parameters since it must represent the input data and the context of each sequence accurately. The decoder, however, is slightly different. The decoder samples over the context vector and outputs a predicted sequence. This sequence is validated using gradient descent and the error is propagated back through the model to update the parameters. Essentially, the decoder has no direct intuition about the input features. It is only penalised on its ability to output correct target sequences.

3.4 Ensemble modelling

This section pertains to the second case study only. Ensemble modelling is used as a method to combine multiple deep learning algorithms to achieve better predictive capabilities than the individual models could achieve alone. The main techniques considered for this study were bagging, boosting and stacking. Bagging and boosting methods are reported as the most popular technique to build ensembles [5]. Bagging is based on a bootstrapping sampling technique. In this method, data-subsets are created with replacement, and multiple models are trained in these subsets in parallel. The outputs of these models is then aggregated to form the final prediction. Boosting, and

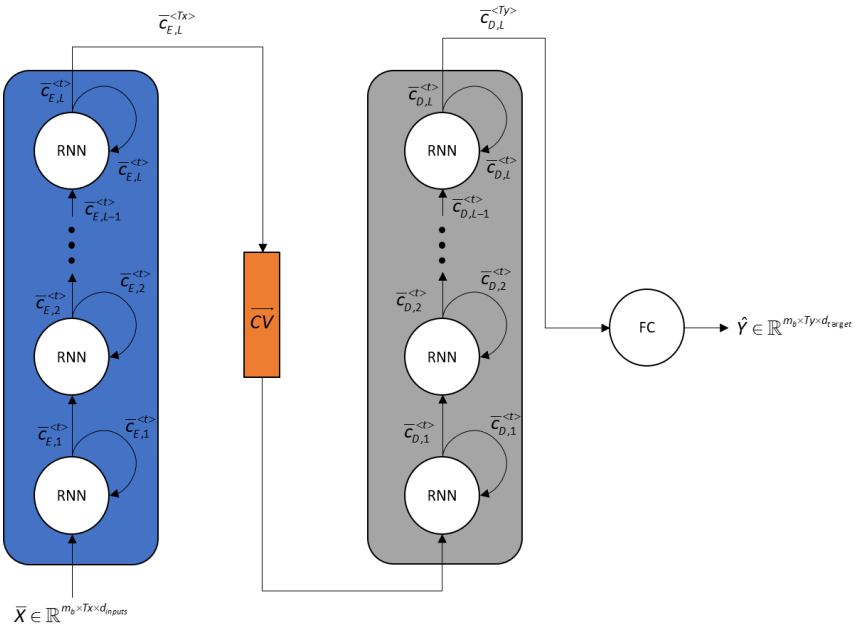


Figure 3: Encoder-decoder RNN configuration

more specifically adaptive boosting (AdaBoost) is a meta-model algorithm formulated by Yoav Freund and Robert Schapire [15]. This algorithm trains multiple models in a sequential architecture where the predictions and errors from the previous meta-learner feed into the contiguous model until a final prediction is made. These algorithms can be used to create strong predictors, however they require an end-to-end modelling and training pipeline whereby the models outputs are aggregated using either voting for classification or weighted averages for regression problems. In the present work, multiple models were generated and were found to have varying weaknesses and shortfalls. Namely the models were either susceptible to overfitting or underfitting.

To create a stand-alone robust model, a stacking approach has been used which combines separate weak learners. The principal concept with this technique is that the outputs from the weak learners are combined to form a new set of training features which is then used to train a new model [46]. For this supervised approach, the target data does not change between models, since the goal is still to make accurate predictions on this desired feature.

The stacked generalisation approach can be seen as follows. Given a dataset $D = \{(y_n, x_n), n = 1...N\}$, where y_n is the target feature space and x_n is the input feature space for N observed instances. Given m level-0 base learner models, we can generate $m \times \hat{y}$ predictions. A second tier model, M can then be trained using a combined dataset, $D' = \{(y_n, x'_n), n = 1...N\}$, where $x'_n = m \times \hat{y}$. Note that these base learner models are tuned separately to achieve local optimal hyperparameters and can be trained for varying epochs, learning rates and batch sizes to achieve different model properties. For example, one model might be trained to learn the global features of a sample space whereas another might be trained to learn the local contexts of the same feature

space. Both models make predictions on the same dataset, however the weights learned by the models will differ based on the models forecasting agenda. An example of this method is seen in the paper by Laptev et al. [24], where one base learner is designed to predict standard every-day activity, and another is designed to predict anomalous Uber activity. The ensemble combination of these two models produced a meta-learner which was able to generalise for normal app activity as well as abnormal behaviour such as public holidays.

4 Part 1: Quasi-steady state prediction of utility scale boiler heat exchanger heat transfer coefficients

4.1 Case study: Boiler heat transfer coefficients

This section details the development of the deep learning model capable of predicting quasi-steady state heat transfer coefficients for a utility scale boiler, using various plant measurements such as the O_2 mole fraction in flue gas at the economiser outlet and the thermal power generated by the unit (simple energy balance using water inlet and outlet conditions).

The boiler in the present study is a 620 MWe two-pass subcritical water-tube boiler. The boiler consists of 7 heat exchangers, namely the furnace evaporator, platen superheater, final superheater, secondary reheater, primary superheater, primary reheater and economiser (additionally the boiler has an air-heater but these do not form part of the water heating loop and was thus not considered. For this project, 336 days worth of data at 5 minute increments have been extracted from the plant for the purpose of this research.

As mentioned before, coal-fired steam boilers are susceptible to slagging, fouling, caustic embrittlement, fire-side corrosion and thermal fatigue which all stem from the heat transfer effectiveness of the heat exchangers. The proposed deep learning model being developed can aid with the management of these failure methods, by providing a quick, direct method of estimating heat exchanger effectiveness. Overall heat transfer coefficients are the targeted output for this model. Heat transfer coefficients represent the proportionality between the heat flux and the temperature gradient between two fluid control volumes. Using the heat transfer rate and the temperature gradients between the fluids, one can estimate the heat transfer coefficients using measured state variables such as excess air ratio, fuel flow rate and generated power.

To calculate the heat transfer coefficients of the 7 heat exchangers in the boiler, 70 inputs are required. The heat absorbed by the different heat exchangers are derived from the change in enthalpy of the water/steam circuit along the flue gas path. Global energy balance and combustion calculations are performed to estimate the flue gas mass flow rate given a certain fuel composition and energy content. Using the combustion calculation results, one can estimate the adiabatic flame temperature and subsequently the furnace exit temperature given the amount of steam generated in the boiler furnace. The heat transfer coefficients per heat exchanger can then be sequentially calculated using estimated flue gas temperatures and mass flow rate along with the measured water side temperatures and pressures.

The ANN is developed and coupled with a probabilistic model to create the MDN which is trained using only 13 input parameters. These inputs are selected on the criteria that they are controlled input parameters. The features include steam flow rate demand, ambient temperature, excess air ratio and final steam temperature and pressure. The model outputs the standard deviations, mixing coefficients and predicted means (the predicted heat coefficient value) for each boiler heat exchanger.

Figure 4 shows the flow path of flue gas and steam through the boiler system. At full load, the

water is fed via the economiser into the steam drum and then fed into the furnace, where steam is generated at saturation conditions dictated by the steam drum internal pressure. The generated steam/water mixture is then fed back into the steam drum where a baffle separates the dry steam from the liquid water. The dry steam is fed into the primary superheater via the roof tubes of the boiler. Attemperation water is added between the primary and platen superheaters and between the platen and final superheaters to regulate the temperature of the steam flowing out of these heat exchangers. Attemperation plays a significant role in the thermal performance of the boiler system, since it has a direct effect on the thermal gradients between the steam side and gas side control volume in each heat exchanger. At full load, the secondary superheater heats the steam from averages of 400°C to 480°C. It is then fed into the final superheater and heated to 530°C and then extracted to the high pressure turbine (HPT). Bleed steam is extracted from the HPT and fed into the primary reheater at an average temperature of 320°C. Attemperation is added and then it is passed into the secondary reheater. Steam from the secondary reheater is extracted to the low pressure turbine (LPT) at 520°C. It is critical that the steam being extracted to the turbines is at superheated temperatures such that no liquid particles enter the turbine drive system and cause pitting on the blades. The flue gas path is a direct flow as seen in figure 4.

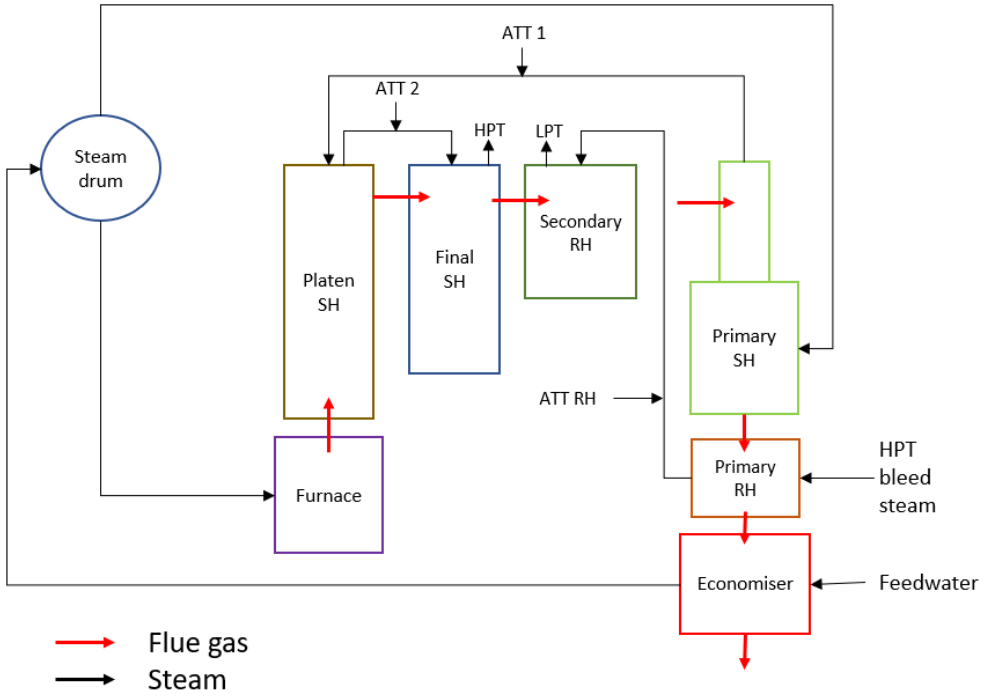


Figure 4: Diagram of water-side and gas-side flows

4.2 Model development

4.2.1 Boiler calculations

In order to determine the heat transfer coefficient θ_i, for the *ith* heat exchanger, both measured and derived data is used for mass and energy balance calculations. To calculate the heat transfer coefficient, the mean temperature difference between hot and cold fluids ΔT_i, as well as the heat transfer rate $\dot{Q}_i$ are required. In equation 19, A_i is the total heat transfer area for the i^{th} heat exchanger, which was taken from the boiler performance schedule documentation. The heat exchanger areas are found using the tube dimensions stated in the C Schedule for the power plant. Table 1 shows the various heat exchanger areas used.

$$\theta_i = \frac{\dot{Q}_i}{A_i \Delta T_i} \tag{19}$$

Heat exchanger	Area (m^2)
Furnace	4281
Platen SH	3156
Final SH	3660
Secondary RH	8135
Primary SH	11040
Primary RH	17667
Economiser	14718

Table 1: Heat exchanger areas

The heat transfer rate per heat exchanger is calculated using the difference between the derived enthalpies using measured water inlet and outlet temperatures and pressures, as seen in equation 20. For the below heat transfer calculations a quasi-steady flow assumption was made, this was deemed acceptable seeing as it is a well know fact that boiler have a slow thermal inertia [2]. In equation 20, $m_{s,i}$ is the mass flow rate of water into the i^{th} heat exchanger and $h_{s,out,i}, h_{s,in,i}$ the outlet and inlet water enthalpies respectively.

$$\dot{Q}_{HX,i} = \dot{m}_{s,i}(h_{s,out,i} - h_{s,in,i}) \tag{20}$$

The enthalpies are evaluated at the inlet and outlet positions of the heat exchangers using the measured temperatures and pressures in the water circuit from the plant data. The mass flow of steam is evaluated at each stage of the boiler since there are added flows between the superheaters and the reheaters due to attemperation. The furnace heat transfer calculation differs slightly for the subcritical boiler. The furnace heat load is determined using the feedwater flow into the steam drum and the latent heat of evaporation of water evaluated at the steam drum pressure. The total water heat load per measured observation is calculated by summing the heat transfer rates of each boiler heat exchanger using $\dot{Q}_{steam} = \Sigma \dot{Q}_{HX,i}$.

The average gas- and water-side temperatures are required to determine the temperature gradient between the gas and water control volumes, as shown in equations 21 and 22. In these equations, $\bar{T}_{s,i}, \bar{T}_{fg,i}$ are the average water (steam) and flue gas temperatures, $T_{s,in,i}, T_{s,out,i}$ the inlet and outlet water temperatures and $T_{fg,in,i}, T_{fg,out,i}$ the inlet and outlet flue gas temperatures. The heat loads of each heat exchanger are found by subtracting the heat load per heat exchanger along the gas path from the energy flux emanating from the combustion zone in the furnace. It is also known that the heat load of each heat exchanger is the difference in inlet and outlet enthalpies. This means that the enthalpy for each input and output stream of gas is found using the previous stages enthalpies and the required heat load for each heat exchanger. Once the average temperatures are found, the heat transfer coefficient can be determined using the effective heat transfer area of the heat exchanger as seen in equation 19.

$$\bar{T}_{s,i} = \frac{T_{s,in,i} + T_{s,out,i}}{2} \quad \bar{T}_{fg,i} = \frac{T_{fg,in,i} + T_{fg,out,i}}{2} \tag{21}$$

$$\Delta T_i = \bar{T}_{fg,i} - \bar{T}_{s,i} \tag{22}$$

Using the known heat transfer rate for a given heat exchanger, the outlet gas enthalpy can be found as by $h_{fg,out}(T_{fg,out}) = h_{fg,in}(T_{fg,in}) - \dot{Q}_i/\dot{m}_{fg}$, where the enthalpy h is a function of gas temperature and composition. The outlet enthalpy of the gas is then used to determine the outlet temperature of the flow leaving the heat exchanger which will be the inlet temperature to next heat exchanger. To calculate the various inlet/outlet gas temperatures, thus, requires the mass flow rate and the composition of the gas.

The gas composition of the flue gas flowing through the boiler is determined using the combustion products from a global-infinitely-fast-complete combustion calculation which takes the fuel ultimate analysis and the excess air ratio α as inputs. The ultimate analysis of the fuel is taken from the design data sheet, where the mass fractions of each constituent are $Y_C = 0.4156$, $Y_H = 0.0222$, $Y_N = 0.0097$, $Y_O = 0.079$, $Y_S = 0.0094$, $Y_{H_2O} = 0.055$ and $Y_{Ash} = 0.409$ [26]. The combustion reactions in the present work are simplified to equation 23, where each reaction is assumed to occur instantaneously and complete in the furnace.

$$\begin{aligned} C + O_2 &\rightarrow CO_2 \\ H_2 + \frac{1}{2}O_2 &\rightarrow H_2O \\ S + O_2 &\rightarrow SO_2 \end{aligned} \tag{23}$$

In addition to the ultimate analysis mass fractions, the design excess air ratio should be selected to complete the combustion calculations and estimate the gas composition and mass flow rate. To determine the required air for complete combustion, the oxygen required for each reaction per kilogram fuel (in equation 23) should be determined by performing a simple species mass balance as shown in equation 24. Once the required oxygen per reaction is known, the stoichiometric oxygen mass per kilogram fuel m_{req,O_2} and in turn required combustion dry air $m_{req,combair}$ can be

found. In the equation below, $Y_{O_2,dryair} = 0.2188$ is the dry air mass fraction of oxygen and $m_{O_2,i}$ is the mass of oxygen required for the i^{th} combustion product, e.g. $i = CO_2$. Once the required air mass is known the total dry air supplied to the combustion chamber per kilogram of fuel is simply calculated as $m_{dryair} = m_{req,combair} \cdot \alpha$.

$$\begin{aligned} m_{O_2,CO_2} &= \frac{M_{O_2}}{M_C} \cdot Y_C \\ m_{O_2,H_2O} &= \frac{2 \cdot M_{O_2}}{M_{H_2}} \cdot Y_H \\ m_{O_2,SO_2} &= \frac{M_{O_2}}{M_S} \cdot Y_S \\ m_{req,O_2} &= m_{O_2,CO_2} + m_{O_2,H_2O} + m_{O_2,SO_2} \\ m_{req,combair} &= \frac{m_{req,O_2} - Y_O}{Y_{O_2,dryair}} \end{aligned} \tag{24}$$

Next, the mass of combustion products, (m_{CO_2}, m_{H_2O}, m_{SO_2}, m_{O_2} and m_{N_2}) per kg of fuel is found. The mass of water in the flue gas is made up of the moisture generated by the combustion reactions, moisture in the fuel and moisture in the supplied humid air. The mass of CO_2, H_2O and SO_2 produced per kilogram fuel, by the chemical reactions are calculated using equation 25.

$$\begin{aligned} m_{CO_2} &= \frac{M_{CO2}}{M_C} \cdot Y_C \\ m_{H_2O} &= \frac{M_{H_2O}}{M_{H_2}} \cdot Y_H \\ m_{SO_2} &= \frac{M_{SO2}}{M_S} \cdot Y_S \end{aligned} \tag{25}$$

Using the mass of products generated during combustion and the amount of moisture in fuel and humid air along with the mass of nitrogen in humid air, the total mass of wet products generated by burning 1 kg of fuel is estimated as shown in equation 26. In the equation below, $m_{H_2O,air}$ is the absolute humidity of the ambient air and $Y_{N_2,dryair}$ is the mass fraction of nitrogen in dry air.

$$\begin{aligned} m_{tot} = m_{CO_2} + (m_{H_2O} + Y_{H_2O} + m_{H_2O,air}) + m_{SO_2} + \\ \left(Y_N + Y_{N_2,dryair} \cdot m_{dryair}\right) + Y_{O_2,dryair} \cdot \left(m_{dryair} - m_{req,combair}\right) \end{aligned} \tag{26}$$

To calculate the flue gas composition in the boiler up to the airheater, equation 27 is used.

$$
\begin{aligned}
Y_{fg,CO_2} &= \frac{m_{CO_2}}{m_{tot}} \\
Y_{fg,H_2O} &= \frac{m_{H_2O} + Y_{H_2O} + m_{H_2O,air}}{m_{tot}} \\
Y_{fg,SO_2} &= \frac{m_{SO_2}}{m_{tot}} \\
Y_{fg,N_2} &= \frac{Y_N + Y_{N_2,dryair} \cdot m_{dryair}}{m_{tot}} \\
Y_{fg,H_2O} &= \frac{Y_{O_2,dryair} \cdot \left(m_{dryair} - m_{req,combair}\right)}{m_{tot}}
\end{aligned}
\tag{27}
$$

Due to holes in ducting and the operation of the rotary airheaters used at the case study boiler, air leaks into the flue gas stream. The air leakage results in a larger mass flow rate of the flue gas leaving the boiler compared to the mass flow rate of gas leaving the combustion chamber. To calculate the mass flow rate of flue gas leaving the boiler and the resulting gas composition an air ingress excess air ratio is defined $\alpha_{ingress}$. The composition and total mass of flue gas per kilogram of fuel under ingress conditions are calculated exactly the same as discussed above but rather using $\alpha_{ingress}$ than α. Therefore, two compositions are calculated namely, Y_{fg} and $Y_{fg,exit}$ with the latter accounting for the ingress air. The ingress air factor, $f_{ingress} = \alpha_{ingress} - \alpha$ is found for the 100% load case and added as a fixed value [28]. For all calculations the ingress air ratio is simply calculated by using the specific calculation step α and adding $f_{ingress}$.

During operation of the boiler the α constantly fluctuates due to various factors such as load changes and ambient condition variations. Therefore, α should be found for each given calculation time step. The O_2 mass fraction is measured at the airheater outlet, at the exit of the boiler, therefore using $f_{ingress}$ and the measured oxygen content in the flue gas one can infer the α in the boiler furnace. A Newton-Raphson algorithm, shown in figure 5 is used along with the above discussed combustion calculations to solve for α at each timestamp such that the O_2 contained in the $Y_{fg,exit}$ matches the measured value.

Once the flue gas composition is known, the fuel flow rate and flame temperature can be solved for. The quasi-steady fuel flow rate per measurement instance is found by balancing the total boiler energy inputs and energy outflows, as seen in equation 28. For the energy inputs; $h_{fuel,sensible}$ denotes the sensible enthalpy of the fuel and HHV denotes the higher heating value of the fuel. Furthermore, $\dot{m}_{fuel}$ is the mass flow rate of fuel, $m_{dry,air}$ is the dry air supplied per kilogram of fuel, $m_{moist,air}$ is the moisture entering the boiler along with the air per kilogram fuel and $h_{air,ambient}$ is the ambient enthalpy of the air. For the energy outflows, the heat transfer plus all the losses in the system need to considered. These losses include the total heat loss due to the unburnt carbon, radiation, bottom ash, unaccounted losses and losses caused by ash at the exit. $Y_{ar,C,loss}$ is the as-received fraction of unburnt carbon per kilogram of fuel, $f_{rad,loss}$ is the percentage of chemical energy lost to the environment, $Y_{ar,ash}$ is the as-received ash mass fraction, $h_{bottomash} = c_{P,ash}\,(1073K - 298K)$ is the bottom ash heat loss, $C_{unaccounted}$ is the unaccounted heat loss, $\dot{m}_{fg} = \dot{m}_{fuel} \cdot \left(1 - Y_{ar,ash} + m_{dry,air} + m_{moist,air}\right)$ is the flue gas mass flow rate and $h_{fg,out}$ is the boiler exit

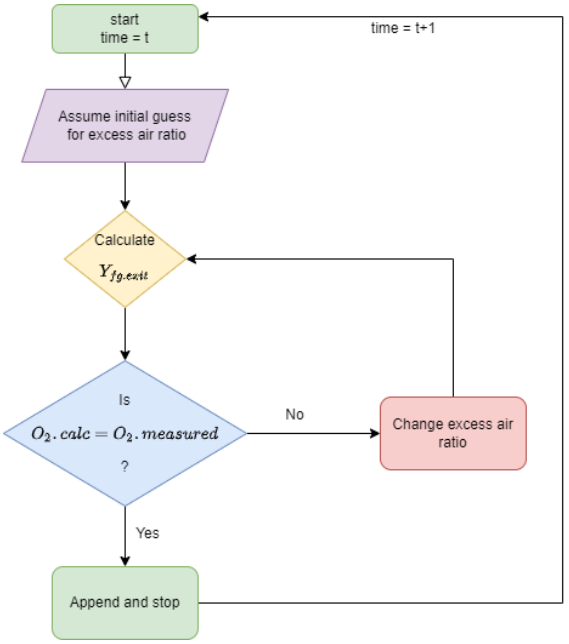


Figure 5: Schematic of Newton-Raphson solver

flue gas enthalpy calculated using a real gas mixture property database.

$$\begin{aligned}\dot{Q}_{sensible,fuel} + \dot{Q}_{combustion,fuel} + \dot{Q}_{combustion,air} = \dot{Q}_{steam} + \dot{Q}_{carbonloss} + \dot{Q}_{radiationloss} + \\ \dot{Q}_{bottomashloss} + \dot{Q}_{unaccountedloss} + \dot{m}_{fg,out}h_{fg,out} + \dot{Q}_{sensibleloss}\end{aligned} \tag{28}$$

$$\begin{aligned}\dot{Q}_{sensible,fuel} &= \dot{m}_{fuel}h_{fuel,sensible} \\ \dot{Q}_{combustion,fuel} &= \dot{m}_{fuel}HHV_{fuel} \\ \dot{Q}_{combustion,air} &= \dot{m}_{fuel}(m_{dry,air} + m_{moist,air})h_{air,ambient} \\ \dot{Q}_{carbonloss} &= \dot{m}_{fuel}Y_{ar,C,loss}HHV_{carbon} \\ \dot{Q}_{radiationloss} &= \dot{m}_{fuel}HHV_{fuel}f_{rad,loss} \\ \dot{Q}_{bottomashloss} &= \dot{m}_{fuel}Y_{ar,ash}(1 - Y_{flyash})h_{bottomash} \\ \dot{Q}_{unaccountedloss} &= \dot{m}_{fuel}HHV_{fuel}C_{unaccountedloss} \\ \dot{Q}_{sensibleloss} &= \dot{m}_{fuel}Y_{ar,ash}Y_{flyash}h_{flyash,exit}\end{aligned}$$

The flame temperature (furnace inlet gas temperature) can be found using an adiabatic combustion control volume calculation. The enthalpy of the flame is evaluated using the equation 29. $h_{air.AH.out}$ is defined as the enthalpy of the flue gas at the air-heater exit temperature using Y_{air}. The energy balance of the furnace states that $f(T_{fg.flame}) = h_{fg.flame} - h_{fg}(T_{fg.flame}, Y_{fg.furnace}) = 0$. A Newton-Raphson solver is also used for $T_{fg.flame}$ which is a function of the flue gas enthalpy.

$$h_{fg.flame} = \frac{m_{fuel} \cdot HHV_{fuel} + h_{fuel.in} \cdot m_{fuel} + m_{tot.air.mass} \cdot m_{fuel} \cdot h_{air.AH.out}}{m_{fg}} \tag{29}$$

The flame temperature and furnace heat transfer rate are then used to determine the furnace exit gas temperature. Radiation leaving the furnace exit plane is assumed to be negligible and is ignored. Using the inlet gas temperatures and calculated heat transfer rates per heat exchanger, as for the furnace, the exit gas temperatures for each heat exchanger can be found using an energy balance calculation which requires the enthalpies at the inlet and outlet of each boiler stage.

4.2.2 Data pre-processing

It is known that the targeted $\bar{Y}$ feature is the heat transfer coefficients for the 7 heat exchangers as seen in figure 4. Using the discussed, combustion calculations (equations 23 $\rightarrow$ 29) and energy balance calculations (equations 19 $\rightarrow$ 22) these heat transfer coefficients can be calculated per measurement instance.

The decrease in heat transfer coefficients along the flue gas path can be seen in figure 6, which shows the coefficients of the boiler components at various boiler loads, represented by $\dot{Q}_{steam}$. The ranges of the bands show how the heat transfer rate from the flue gas to the steam decreases as the temperature difference decreases from the stage closest to the adiabatic flame, the furnace, to the furthest stage at the economiser. The clustering around each of the bands is indicative of the multi-modality of the data which suggests external forces have significant effects on the distribution of the coefficients. The heat transfer to the furnace walls, platen superheater and final superheater for the case study boiler is mainly due to radiation from the hot combustion gases [26]. The unsteadiness of the combustion processes leads to a large variation in estimated heat transfer coefficients per given load for these radiant heat exchangers. It is seen as the heat exchanger location moves further away from the furnace the band of data scatter narrows.

The input data, $\bar{X}$, does not include the entire dataset used to manually calculate the heat transfer coefficients for each heat exchanger. Instead, it has only 13 features which are direct inputs that influence the boiler's operation. The input data to the machine learning model is taken from pressure, temperature and mass flow rate sensors on the plant and are listed in table 2. The inputs include steam flow rate demand, ambient temperature, excess air ratio (α, not shown in the table) and final steam thermal conditions among other integral pressure and temperature readings. These inputs are user controlled and are not products of the boiler's operation, essentially these parameters are operator controlled settings. The main steam temperature, which is the inlet steam temperature to the high pressure turbine, as well as the reheater 2 outlet temperature are in effect controlled parameters since they are directly affected by the amount of attemperation per boiler load. Reheater components have been abbreviated to RH. By limiting this feature space, it is possible to get an accurate understanding of which controllable features have strong influences on the rate of heat transfer through the boiler system which is useful in real world deployments of such models.

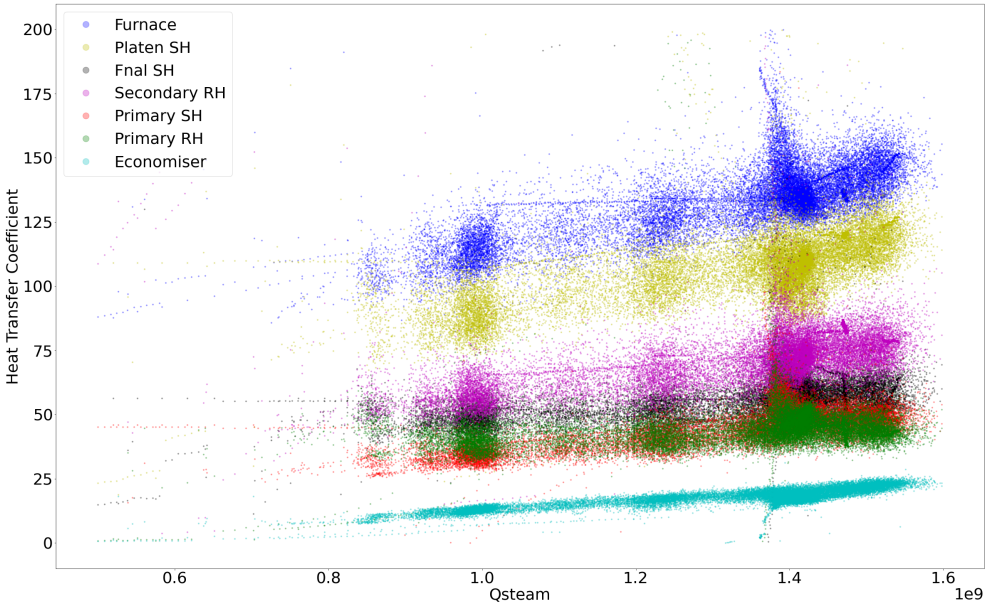


Figure 6: Heat exchanger θ at various boiler loads

Table 2: Measured inputs to the ANN

	Feedwater	Steam drum	Main steam	RH 1 inlet	RH 2 outlet	Attemperator 1	Attemperator 2	RH attemperator
Temperature	x		x	x	x			
Pressure	x	x			x			
Mass flow rate	x			x		x	x	x

4.2.3 Hyperparameter tuning

In the present section, the selected network configuration, data preparation and hyperparameter search will be discussed. The Python 3.8.3 programming language was used in the present work along with Keras machine learning libraries [10]. The development of this model involved only 1 dataset. A standard MLP model was built and optimal hyperparameters found. An MDN probabilistic extension was added to the model and further training and testing was done to find a robust architecture.

To find the best combination of hyperparameters, a course grid search was completed. The motivation for using this tuning approach, rather than a Gaussian, evolutionary or Bayesian optimisation algorithm is to generate results with which one can examine the effects of various hyperparameters on the overall model performance [37]. The downside of this approach is that the global optimal is not necessarily found since that may lie within finer ranges of the course broad search space.

The model was tested using 1 and 2 layers. For each number of layers, the model was tested using 10, 15 and 25 neurons. In traditional machine learning, it is a rule of thumb that the number of neurons chosen should range between the input and output feature sizes, however models tested with less than 10 neurons showed underfitting of the data, hence larger values were chosen. The

complete dataset is split into a training set (85%) and development set (15%). The mean average error (MAE) is used to evaluate each model's performance for the hyperparameter tuning. The best model is selected using the explained variance, which is the absolute difference between the training and development MAE for a single network architecture. A fixed learning rate of 0.0001 and a batch size of 128 was used. This combination ensured smooth training (no oscillations in the gradient descent) and convergence (no overt overfitting or underfitting). Overfitting is the phenomena that occurs when a model is trained too heavily on the training set such that it is unable to generalise when presented with never-before-seen data. Underfitting occurs when a model is not trained enough, or the architecture is overly simplified such that it is unable to make accurate predictions on both the training and the development data. The results of this are detailed in the next section.

Once the best standard model architecture has been found, the MDN component of the model is built onto this architecture. For this project, 3 values for the component k are chosen, namely 2, 3 and 5 to represent the possible distributions in the data. It is initially unknown how many Gaussian distributions are contained in the data so this step of the tuning is to estimate the best performing setting. The MDN model was tested using batch sizes of 128 and 64, and it was trained for 1000 and 1500 epochs respectively, the learning rate was kept at 0.0001 to ensure convergence. These configurations were also evaluated using the mean absolute percentage error to give further intuition on the HTC error. A final, stable model was then developed using a batch size of 256 and trained for 7500 epochs.

A section of this code can be found in Appendix A.1.

4.3 Results and discussion

The models are validated using a variety of methods such as MAE, MAPE and statistical inference methods, namely prediction intervals and confidence intervals. A sensitivity analysis was also done as is detailed in this section.

4.3.1 Hyperparameter search results

Table 3: Training set MAE

	No. neurons		
No. layers	25	15	10
1	21.86	9.51	9.95
2	11.53	3.44	20.02

Table 4: Development set MAE

	No. neurons		
No. layers	25	15	10
1	21.96	9.83	10.16
2	10.86	3.22	20.77

Tables 3 and 4 show the results of the hyperparameter grid search on both the training dataset and the development dataset. Overall, it can be seen that the model architecture with 2 layers and 15 neurons has the lowest MAE ($\frac{W}{m^2K}$).

From table 5 it can be seen that the model with 1 layer and 25 neurons has the lowest explained variance. It is however not a good choice for architecture. From tables 3 and 4 it can be seen

Table 5: Explained variance

	No. neurons		
No. layers	25	15	10
1	0.1	0.32	0.21
2	0.67	0.22	0.75

that these models have a relatively high MAE, compared to the other network architectures. This indicates that the model trained to a stable local optimal, meaning it performs almost equally well on never-before-seen data as it does on data used for training. However, the actual error on both datasets shows that this model does not in fact perform well overall at this optimal. The network architecture with 2 layers and 15 neurons is therefore chosen as the best performing model.

Table 6: MDN K component selection

MDN	MAE train	MAE dev	MAPE train	MAPE dev
2	3.527	3.213	8.4%	7.2%
3	3.598	3.252	8.5%	7.45%
5	3.604	3.241	8.56%	7.19%

For the MDN hyperparameter K, the best model was tested using 2, 3 and 5 distribution components. Considering the computing power required for more components to be predicted, there was no significant increase in prediction accuracy as seen in table 6. Therefore it was decided that 2 components captured the modes in the data sufficiently well.

Table 7: MDN Hyperparameter tuning results

Batch size	MAE train	MAE dev	MAPE train	MAPE dev
256*	5.48	4.88	11%	10%
128	3.9	3.58	9.07%	7.92%
64	3.35	3.11	7.78%	7%

Batch sizes of 64 and 128 were initially tested using a learning rate of 0.0001. It can be seen in table 7 that using a batch size of 64 yielded the best results. The model was trained for 1500 epochs, this was determined using diagnostic training plots which indicate how a model is training. It must be noted however that this configuration was prone to vanishing gradients. In order for the model to be robust and train efficiently, the batch size and learn rate were manually refined to 256 and 0.000005 respectively. The model was trained for 7500 epochs with early stopping with a patience of 500, however the model trained for the full epochs. Batch normalisation was also used as regularisation between the fully connected layers to prevent vanishing gradients.

4.3.2 Final model performance

In this section, the results of the best performing model architecture will be discussed.

The output of the MLP model is shown in figure 7. The prediction and observed HTC time-series is plotted for each heat exchanger in order of the flue gas path. Specific points are indistinguishable (detailed samples are shown below in figure 10) however it can be seen that the model does capture the general trend of the data as seen by the strong overlap of the observations and predictions signals.

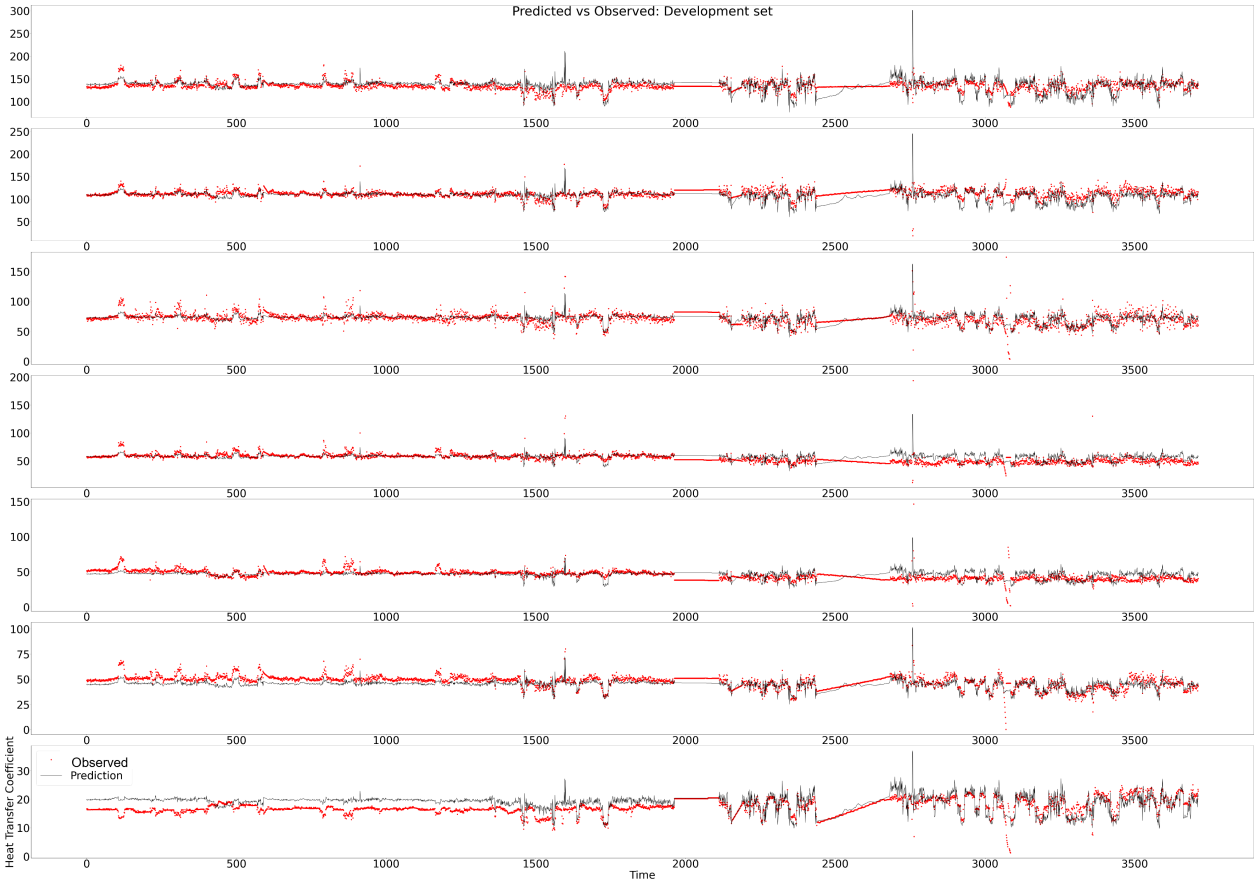


Figure 7: Heat transfer coefficient predictions for all heat exchangers in the boiler system

Figure 8 shows the predictions made by the MLP-MDN model. Predictions and observations are shown as well as the 95% confidence interval bounds where $CI_{95\%} = \pm 1.96\sigma$. The model performance results are shown in table 8.

Table 8: Model performance results

	Training	Development
MAE $[\frac{W}{m^2K}]$	5.48	4.89
MAPE	11%	10%

In this case study the development set error tended to be lower than the training set error. This

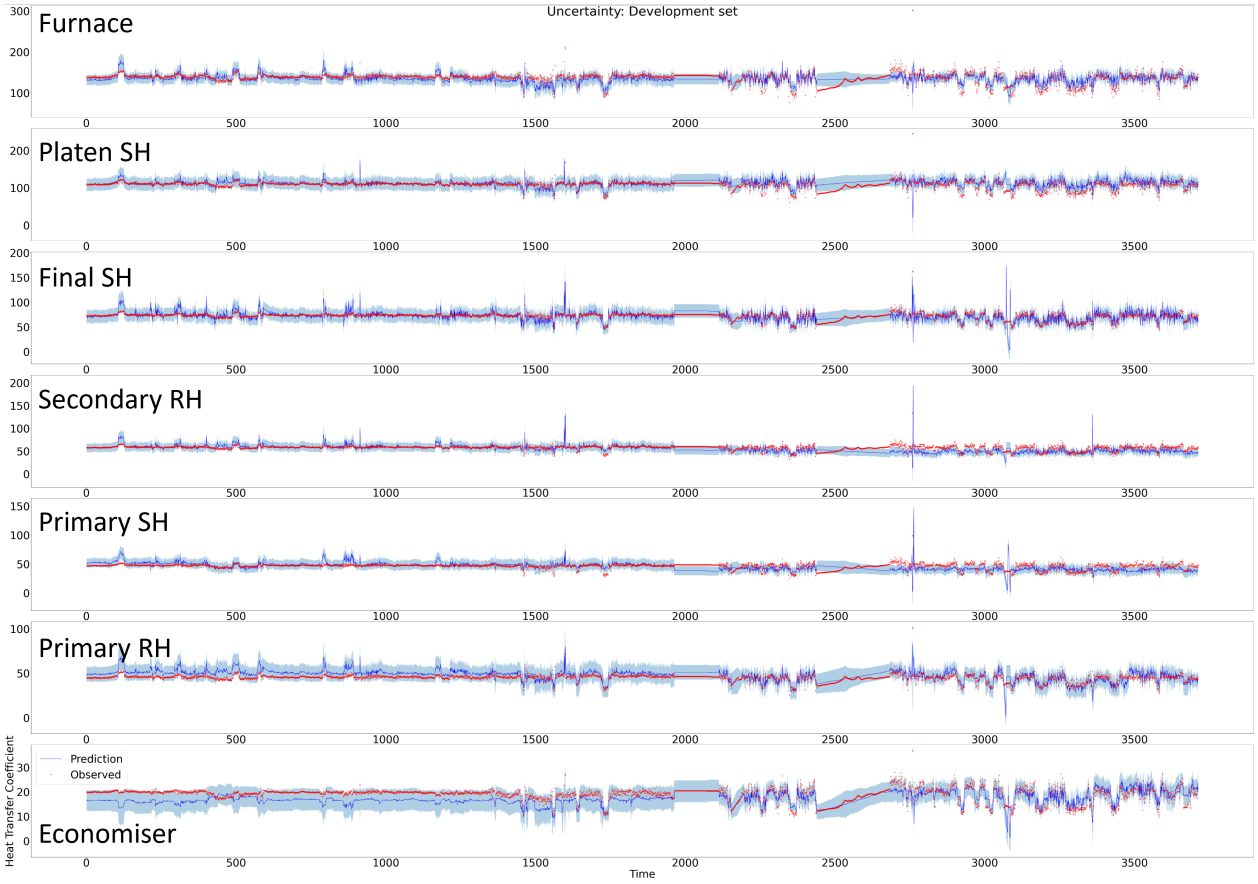
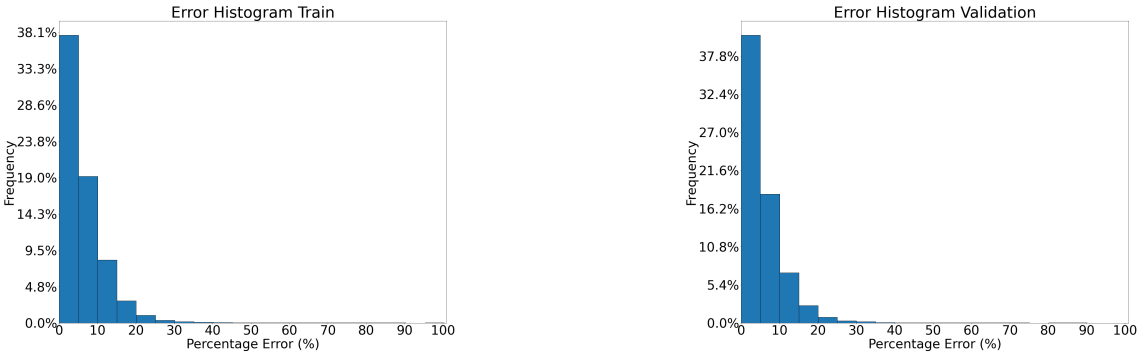


Figure 8: Heat transfer coefficient predictions for all heat exchangers in the boiler system with uncertainty

is due to the development dataset being closer to the training dataset mean and having a lower variance, which leads to the better model performance.

Figure 9 shows the error distribution graphs for the training and validation set. It can be seen that 57% of the predictions have less than a 10% error. The validation set has approximately 60% of the predictions within the 10% error margin.

Figure 10 shows a sample of 1000 time steps from the development dataset for all the heat exchangers, with the model interval coverage (prediction interval) and mean prediction interval coverage (confidence interval). The mean prediction interval coverage is the same uncertainty band shown in figure 8, where the σ value is an output of the MDN model. The model interval coverage is determined using the standard deviation obtained using the observed data. Although prediction intervals and confidence intervals have underlying similarities, they represent different properties of the data. The confidence interval is the range which has a 95% chance of containing the true population of the data. It is based off the statistical parameters, standard deviation and mean of



(a) Training set (b) Validation set

Figure 9: Error distributions

the estimated predictions generated by the model. It can be interpreted as the likelihood of the observed data falling within 2 standard deviations of the predicted mean. The predictive interval stipulates what range the predicted mean can fall between based on the range of the true data. This can be interpreted as the estimated range in which a future prediction will fall given the true observations that have already been observed within some level of confidence, also 95% in this case. The percentage of observed points that fall within these bands, as well as the average width of each band is shown in table 9. These values are the average for all time-steps over all heat exchangers.

Table 9: Statistical results

	% In-band	Band Width
Model Interval Coverage	91.98	22.33
Mean Prediction Interval Coverage	76.83	12.49

The furnace is the stage of the boiler that is most susceptible to fluctuations in the boiler's operating parameters. The predicted values are constantly slightly higher than the observed values, except for the peaks where the model makes predictions that are too low. This indicates a weakness in the model predictive capability. This disparity is due to the model over generalising due to harsh regularisation. Another cause could be the exclusion of the direct radiation leaving the plane. This could manifest as varying ratios between heat input and furnace wall absorption. This phenomena at the peaks is experienced by all heat exchangers. For the first four heat exchangers however, the observed peaks lie within the mean prediction interval coverage and for the latter 3 heat exchangers they are encompassed by the wider model interval coverage. The platen superheater, final superheater and secondary reheater are have the most accurate predictions. This is a product of input features, such as the main steam and super heater attemperators, having direct, controllable effects on these heat exchangers. This strong relationship lets the model find well fitted learnable parameter weights which are responsible for controlling the signals used to calculate these predictions. It is encouraging to see that the model is capable of predicting the actual heat transfer coefficients of the superheaters with reasonable accuracy. The boiler tubes which typically

fail due to localised overheating are the superheater elements. The final superheater is therefore the most critical superheater seeing as it has the highest steam temperatures and it can be seen that the mean prediction falls sufficiently within the scatter of the observed data points. The economiser is the final stage in the flue gas path of the boiler. A constant offset is noticed for the economiser heat transfer coefficient predictions, but the magnitude of the error is relatively low, approximately 5 $\frac{W}{m^2K}$. Furthermore, the fluctuations in the observed data of the economiser is significantly lower when compared to the furnace values, meaning the economiser is less susceptible to changes in the boiler operating load and therefore its heat transfer rate is not a direct product of input features.

Although there are some outliers across all the heat exchangers, the confidence interval captures the vast majority of these peaks as seen in table 9. The statistical prediction interval is substantially wider than the model's predicted confidence interval. The width of these bands increases along the flue gas path of the heat exchangers, which suggests that the flue gas temperatures have a direct impact on the uncertainty associated with the predictions.

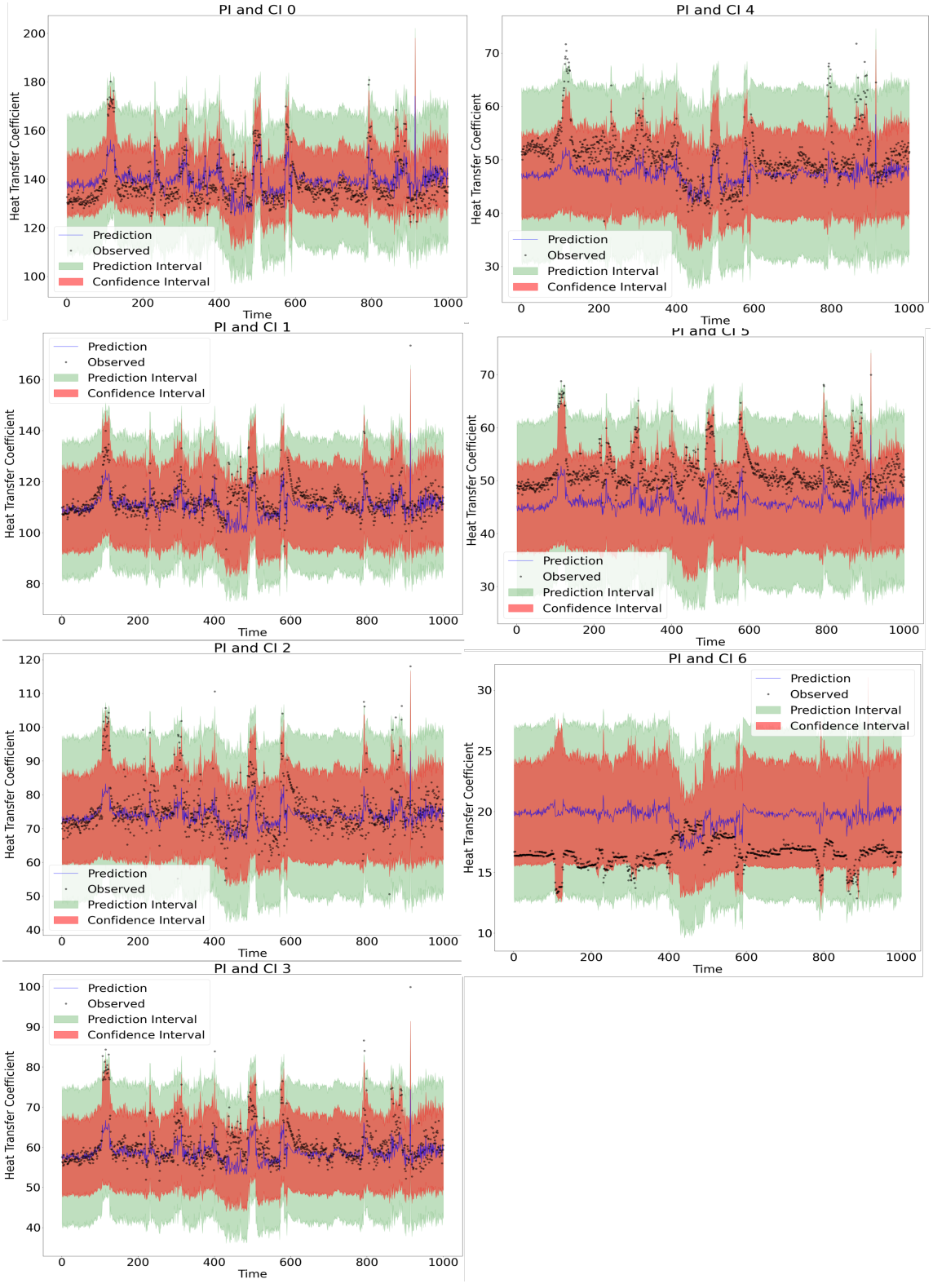


Figure 10: Development prediction with confidence interval and prediction interval for all heat exchangers

4.3.3 Sensitivity analysis

A sensitivity analysis was done to determine how the input features influence the predictions of the output features. For each input feature, an average value of each input feature was determined for three load cases, 60%, 80%, and 100%. Each independent feature is permuted one at a time and the respective outputs calculated using the MLP-MDN model developed in the preceding sections. Permuting is the process of getting a maximum and minimum bound for each variable by adding and subtracting 10% from the average input values. HTC predictions were then made one variable change at a time for the maximum bound and the minimum bound. The sensitivity coefficient is calculated using equation 30.

$$\frac{(\frac{\partial \theta_j}{\partial x_i})^2}{\sum_i (\frac{\partial \theta_j}{\partial x_i})_i^2} \times 100 = C \tag{30}$$

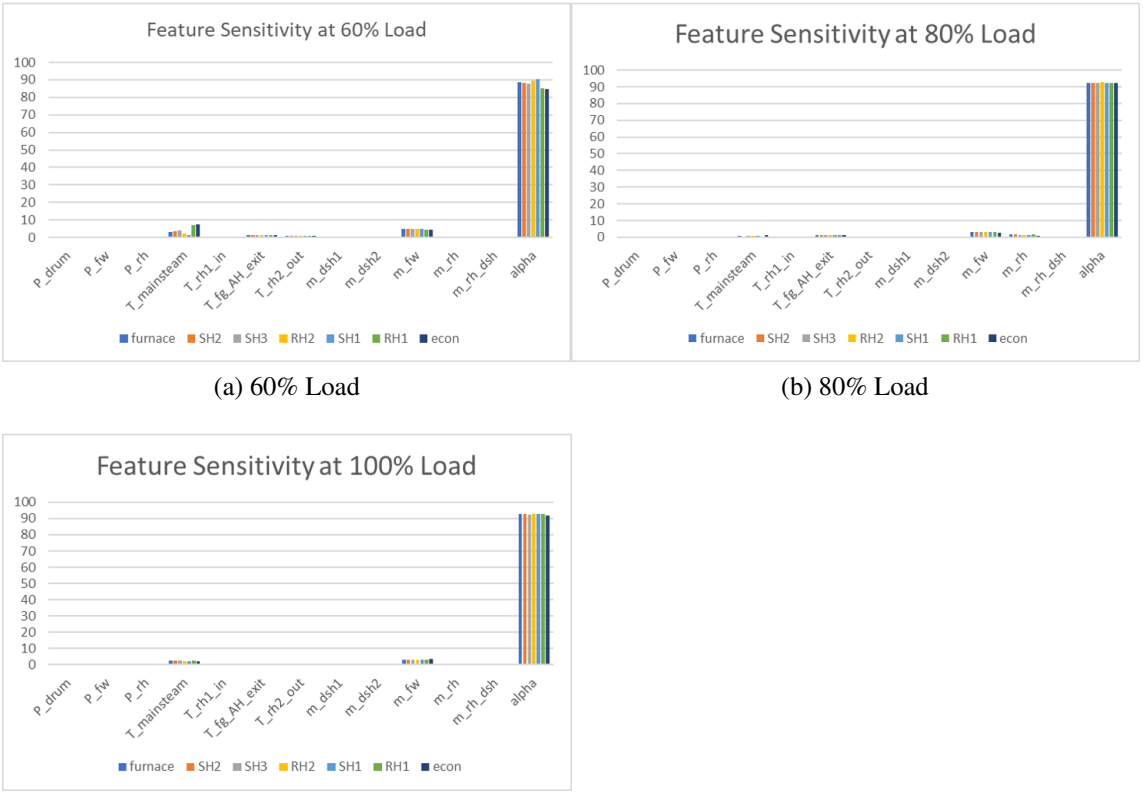
As an example of the above perturbation process, the steam drum pressure will be used. An average drum pressure, denoted by P_{drum}, is found by averaging the pressure over the 100% (±5%) load case data range. Load cases are a function of Q_{steam}. The pressure is permuted to get P^{+}_{drum} and P^{-}_{drum} by adding and subtracting 10% of the average value. Steady-state predictions are made for all heat exchangers, using each of these values respectively, while all other features are kept unchanged. The sensitivity index is then found for P_{drum} for each heat exchanger using equation 31, where θ_j denotes each heat exchanger HTC and x_i denotes the feature variable, x_{Pdrum} in this case.

$$SI = \Big(\frac{\partial \theta_j}{\partial x_i}\Big)^2 = \Big(\frac{\theta_j^{+} - \theta_j^{-}}{x_i^{+} - x_i^{-}}\Big)^2 \tag{31}$$

Considering a single heat exchanger HX_j, a sensitivity index denoted by SI is found for each of the 13 input variables using equation 31. As seen in equation 30, the sensitivity coefficient C is found by dividing the SI by the sum of all variables SI for all heat exchangers, HX_j and converting this ratio to a percentage. Figures 11 and 12 show the results of this analysis.

It can be seen in figure 11 that the excess air ratio, α, has a significant effect compared to the other input parameters. This is because alpha represents the flue gas flow profile as fuel effects are captured by this feature. The excess air ratio varies to ensure total combustion and therefore can be seen as a proxy for fuel quality and flow rate. α has a direct affect on the mass flow and velocity of flue gas, which is the main driving force for heat transfer so it is expected that the excess air ratio has the largest influence on the predictions.

Figure 12 shows the same data without the excess air, α, values to compare how the other parameters performed. For all load cases, the feedwater flow rate, m_{fw} has a strong effect. The flow of feedwater into the system directly controls the water level in the steam drum and therefore is a strong driving force for maintaining a proper flow of steam through the heat exchangers for effective heat absorption. Hence, a strong relationship exists between heat transfer and feedwater



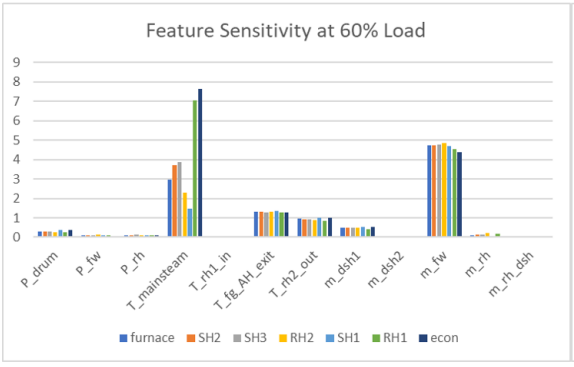
(a) 60% Load

(b) 80% Load

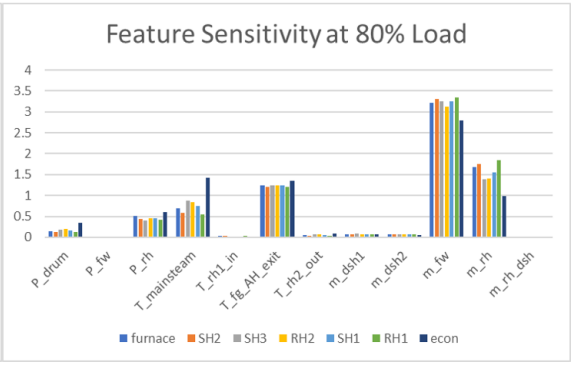
(c) 100% Load

Figure 11: Sensitivity analysis with excess air ratio (alpha) shown

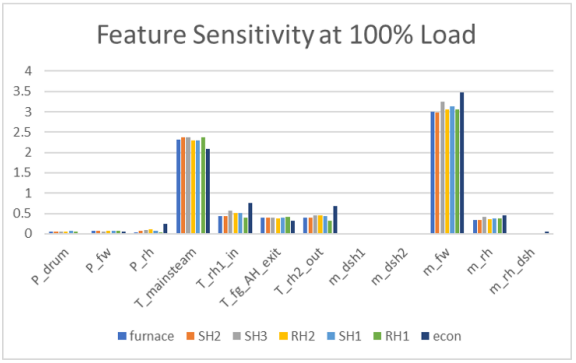
mass flow. At the 60% load case, $T_{mainsteam}$ is more significant in the low heat exchangers. This is because at low loads, there is less fuel flow therefore less heat. It is likely that the performance will drop below the performance rating so this variable needs to be carefully controlled to ensure total combustion. In this sense, the main steam flow is directly proportional to excess air ratio. The flue gas flow drives the rate of heat taken up by the heat exchangers and therefore the flow of steam. At the 80% load case, the reheat pressure and mass flow have an obvious affect. This is due to the increased rate of heat transfer happening at the reheaters. This is also evident by the flue gas exit temperature as seen by the increase in $T_{fg.AH.exit}$. This indicates that at lower loads, fluctuations in the exit temperature are a direct result of the heat not being completely absorbed by the primary, secondary and final superheaters. Flue gas exit temperature is important because it is an indication of the gas energy content. At lower operating bands this is increased because of incomplete heat absorption. At 100%, it is seen that the reheater temperatures are a contributing factor, unlike other load cases. This is due to the increased temperature gradient between the flue gas and the bleed steam extracted from the HPT. More energy is extracted at peak performance, therefore this temperature difference is seen as driving force for heat transfer in the system.



(a) 60% Load



(b) 80% Load



(c) 100% Load

Figure 12: Sensitivity analysis without excess air ratio (alpha) shown

5 Part 2: Data-driven forecasting with model uncertainty of utility-scale air-cooled condenser performance using ensemble encoder-decoder mixture-density recurrent neural networks

5.1 Case Study: Air-cooled condenser backpressure

This section details the development of a RNN-MDN deep learning model capable of predicting ACC backpressure, 4 hours into the future at 15 minute increments.

As mentioned, for many thermal power plants utilising dry-cooling technologies, one of the major contributors to loss of achievable power generation is high steam pressures at the outlet of the low-pressure turbines (also called turbine backpressure) [30]. This pressure can be strongly affected by ambient conditions such as ambient temperature, wind speed and wind direction. The ACC being analysed in the present case study is from an actual 657 MWe coal fired power plant.

ACCs are used to disperse excess heat from working fluid in power cycles such as the Rankine cycle. Steam from the LPT is ducted to the ACC where hundreds to thousands of finned heat exchanger tube bundles release heat into the environment. The heat transfer from the tube bundles to the environment is driven by large axial fans beneath the A-frame of the ACC units. The pressure in the ACC is approximately equal to that of the LPT exit. The ACC inlet water vapour temperature ranges between 32°C and 75°C which results in a temperature difference between it and the atmospheric air [37]. The rate of heat transfer from the steam in the tube bundles to the atmosphere is indirectly proportional to the difference between the steam temperature and the ambient air temperature. The higher the ambient air temperature, the less heat is dissipated and the higher the back pressure. This is also true for the wind speed in a sense that low wind speeds inversely affect the rate of heat transfer from the ACC to the atmosphere. Wind direction is also a vital feature since the wind speed is optimal at certain angles based on the orientation of the condenser housing. As power production increases, the steam flow rate increases and with it, the quality of the ACC inlet steam. This is due to a higher volume and flow rate of steam being required to turn the turbines. This induces a higher water-side heat rejection and therefore a higher ACC backpressure. There is a direct correlation between turbine backpressure and the heat transfer from the water vapour to the air [37]. This effect decreases the enthalpy difference over the turbine system and therefore decreases the amount of mechanical work that can be extracted. The power cycle under consideration consists of a once-through boiler, reheater, HP/IP/LP turbines, ACC, LP feedwater heaters, HP feedwater heaters and associated ancillaries.

The specifications of the ACC under consideration can be seen in table 10 below. The plant data is extracted from the GP Strategies EtaPro® plant condition monitoring server which streams and stores the live plant measurements. The raw input dataset consists of 43 process parameters that affect ACC performance such as the main steam flow rate, final steam pressure and temperature, reheater outlet temperature and pressure, various turbine extraction flow rates, extraction temperatures and pressures, gross power generation and ACC fan power usage. The input dataset also

contains the following meteorological features: wind speed, wind direction and dry-bulb temperature which were supplied by the South African Weather Service. The targeted output for the model is the corresponding ACC backpressures. The multi-modality of the data is evident from figure 13, which shows the gross generation as a function of ACC backpressure with colour being a function of time. The coldest period in July is represented as dark blue which transitions to yellow for the warmest month of February. Between 400 MW and 700 MW band there is a strong concentration of data for the entire range of back pressures and across all seasons of weather. This means that various combinations of operating parameters and weather conditions can lead to the same relationships between plant load and ACC backpressure which is the root cause of this experienced multi-modality.

Table 10: ACC design parameters

Design parameter	Value (units)
Heat rejection rate	Approx. 900 MW
ACC platform area	$84.5 \times 82.5\ m^2$
No of heat exchanger cells/axial flow fans	48
Airflow per Fan	550-600 kg/s
Electrical power requirement per fan	215 kW
Tube bundles per heat exchanger cell	107
Platform height	45 m

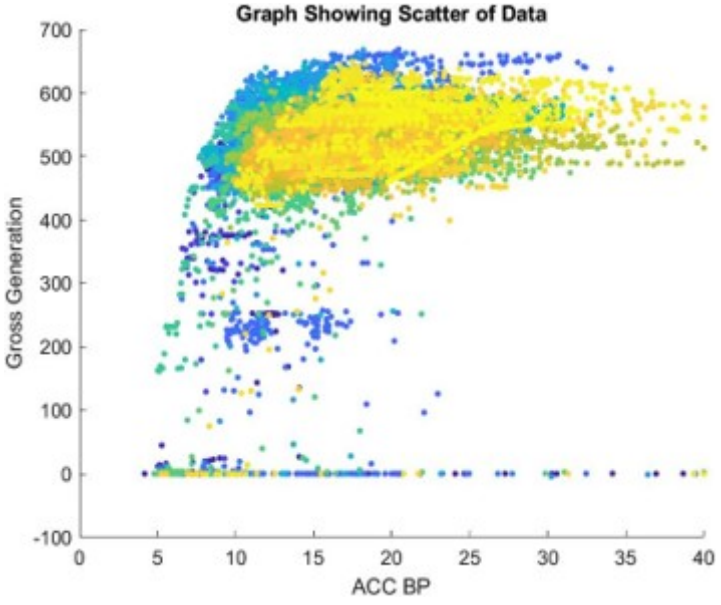


Figure 13: Plot of gross generated load as a function of ACC backpressure and time

5.2 Model development

In the present section, the selected network configuration, data preparation and hyperparameter search will be discussed. This section details the development of two types of deep learning models, namely standard encoder-decoder RNNs (designated network-1) and encoder-decoder MDN-RNNs (designated network-2). Network-1 is used as a benchmark to compare the performance of the RNN-MDN model. The aim here is to highlight the added benefits of predicting the output data using the additional probabilistic mixture distribution model. The Python 3.8.3 programming language was used in the present work along with Tensorflow 2 and Keras [10] machine learning libraries.

5.2.1 Encoder-decoder RNN and MDN-RNN configuration

The structure and split of the input and output data is important to the training of RNN models due to the dimensionality and shape required. The data is fed into the model in batches. For a batch, m_b of some size, the output data from the model will have the size $m_b \times Ty \times 1$, where Ty is the number of time-steps and the 1 represents the single output variable. This tensor arrangement, and more specifically the time-steps dimension, is determined using a sliding window approach. This divides the time-series data into overlapping sequences which improve the model by exposing the network to the beginning and end of consecutive sequence. This is shown in figure 14. The sequences are stacked into lag periods/ time-steps. The input data has the shape $m_b \times Tx \times d_{inputs}$ where d_{inputs} is the number of input features, this will be discussed further in section 5.2.2. Tx is the input lookback window width, corresponding to Ty.

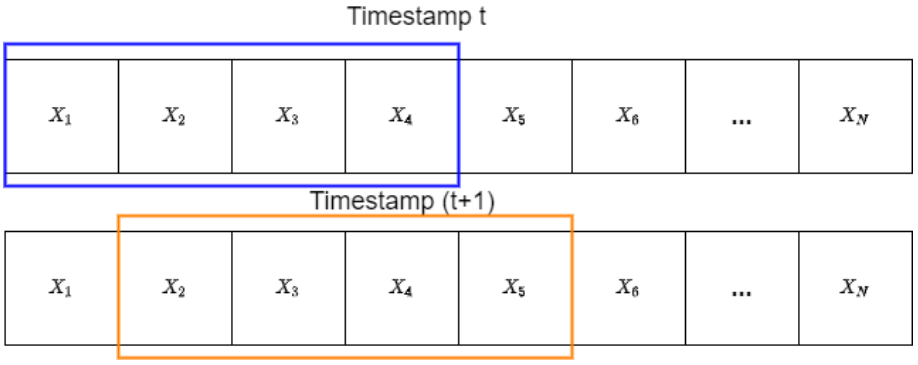


Figure 14: Sliding window configuration

The standard models developed will use this 3-dimensional structure for the input and output. The encoder-decoder RNN-MDN output will have a 4th dimension due to the addition of the probabilistic mixture outputs. This tensor will have 3 parts: the mixing coefficients $\bar{\pi}$ with shape $m_b \times Ty \times K$, standard deviation $\bar{\sigma}$ with shape $m_b \times Ty \times K \times 1$ and predicted means $\bar{\mu}$ with shape $m_b \times Ty \times K \times 1$. Recall that K is the chosen number of Gaussian distributions or discrete modes within the data.

Networks with single and double encoder and decoder sections were tested to find the best performing model architecture. The architecture of network-1 is similar to figure 3. The simplified architecture of network-2 is shown in figure 15. The output shapes of the MDN stage of the model are also shown.

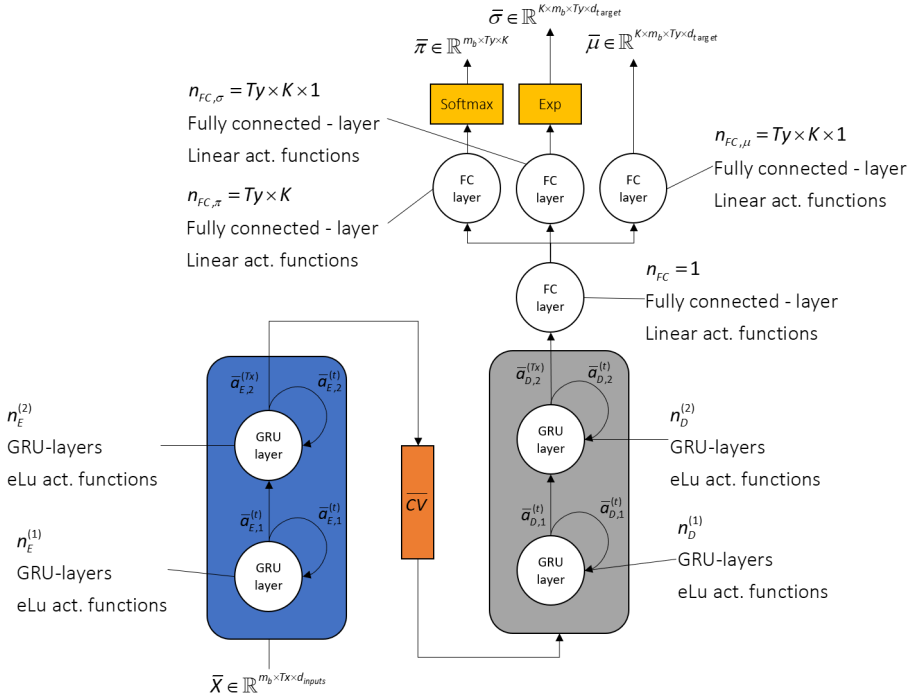


Figure 15: Encoder-decoder MDN RNN (network-2) schematic

The model uses GRU units with ELu activation functions. There is only 1 neuron in the output layer since $d_{targets} = 1$ and uses a linear activation function. The model is trained using the BPTT and Adam algorithms with the standard MSE cost function such that optimal network parameters, $\bar{W}$ and $\bar{b}$ can be found. The next section will discuss the hyperparameter search used to find the best performing architecture. In figure 15, n_E^l is the number of neurons on encoder layer l and n_D^l is the number of neurons on decoder layer l. For the present research, it was set that $n_E^l = n_D^l$. The output of the RNN stage of the model has a single fully-connected output neuron, n_{FC} with a linear activation as mentioned before. The optimal parameters here are found for network-2 by optimising to minimise the negative log-likelihood equation shown in equation 9. The hyperparameter search for this network is also discussed in section 5.2.2.

Both models are trained using early stopping with a patience of 500 epochs. Early stopping is a method of reduce overfitting to improve the generalisation of deep neural networks [17]. It works by monitoring the validation loss and then stopping training when this value starts to increase or doesn't improve for a set number of patience epochs.

A section of this code can be found in Appendix A.2.

5.2.2 Data pre-processing and hyperparameter tuning

Three datasets have been set up to investigate the effects of input features on two network configurations. Table 11 shows the data contained in each of the three datasets. The purpose of this split is to explore the effects that weather and plant conditions (discussed in section 5.1) have on the predictive abilities of models and show how increased data complexity can alter the accuracy of forecasts. Pre-processing was required to ensure no unrealistic output readings were fed into the model for the training process. In these rare occurrences (9 data points), the backpressure had to be clipped at 40kPa when a unit trip or undesirable sensor behaviour caused a value beyond a physical limit to be recorded.

Table 11: Input features used for three datasets

Design parameter	**Value (units)**
Dataset 1: gross gen only	
Gross generation	MW to grid
Dataset 2: gross gen + weather	
Gross generation	MW to grid
Weather conditions	Dry bulb temperature
	Wind direction
	Wind speed
Dataset 3: gross gen + weather + P & T measurements	
Gross generation	MW to grid
Weather conditions	Dry bulb temperature
	Wind direction
	Wind speed
High-pressure turbine measurements	Inlet T/P, exhaust T/P, inlet flow rate, reheat flow rate, shaft power and leak-off flow rate
Intermediate-pressure turbine measurements	Inlet T/P, inlet flow rate, extraction flow rates, extraction T's/P's and IP shaft power output
Low-pressure turbine measurements	Inlet P, exhaust flow rate, LP shaft power output, extraction flow rates and extraction T's/P's
Air-cooled condenser measurements	Make water flow rate, hotwell outlet T/P, hotwell outlet flow rate and total fan motor power usage.

The training/development split used here is 90% training dataset and 10% development dataset. Min-max normalisation was applied to scale the input feature data between 0 and 1. This alters the scale of the data such that no information or difference in range is distorted. It is required since large orders of magnitudes between input features can hinder the training of parameters and lead to sub-optimal solutions or extremely long training times. Hyperparameters are selected for all 3 datasets and for both network configurations such that 6 final model architectures are developed. The hyperparameters for network-1 and network-2, using datasets 1, 2 and 3 were found using a course grid search, similar to the method detailed in section 4.2.3. The tested model architectures and hyperparameter combinations are shown in table 12. The results of this search will be discussed in section 5.3.

Table 12: Hyperparameter search space for encoder-decoder RNN and MDN-variant

Parameter	Search space
Lag period, Tx	4, 8, 12 (hours)
Number of neurons in GRU layers	200, 250, 350, 500
Learning rate	0.01, 0.001, 1E-4
Batch size	128, 256, 512
Number of encoder and decoder layers	1, 2

5.2.3 Ensemble model

Because the models used for the hyperparameter search are subjected to early stopping regularisation (to improve generalisation), they have an impaired ability to fit outliers. However, the ability to predict uncommonly high back pressures accurately is important since this is one of the failure methods experienced by the system. Therefore the regularisation should be relaxed enough to allow the model to fit these extreme values, without altering the generalising ability of the models. To do this an additional network was generated using the overall best performing network-2 configuration with early stopping regularisation deactivated and allowed to train for 3000 epochs. This model will however have poor generalisation. To over come this, it is combined with the network with early-stopping active using a meta-learner approach using ensemble modelling. The meta-learner model is a relatively small encoder-decoder MDN-RNN network with only a single encoder and decoder RNN layer and 50 neurons per layer. The inputs to the meta-learner are the outputs of the two encoder-decoder MDN-RNN networks, π,μ and σ. The meta-learner is required to learn how to combine the results to yield the lowest negative log-likelihood error with respect to the actual output data. Essentially it will learn network parameters that capture the benefits of both models. Figure 16 shows the schematic of the ensemble model. The predictions generated by the selected base-learner MDN-RNNs are used to identify the most probable predictions by selecting the means and variances that correspond to the maximum mixing coefficient values for each network. These two sets of output values ($\bar{\pi}_{k\rightarrow\pi_{max}}$, $\bar{\mu}_{k\rightarrow\pi_{max}}$, $\bar{\sigma}_{k\rightarrow\pi_{max}}$) from each base-learner are then concatenated before being used as input data to train the meta-learner MDN-RNN on. The effect of combining the early-stopped and the unregularised MDN-RNN networks will be discussed further in the next section.

5.3 Results and discussion

The models are validated using a variety of methods such as mean average error and statistical inference methods, namely prediction intervals and confidence intervals.

5.3.1 Hyperparameter search results

The best-performing standard encoder-decoder RNNs and encoder-decoder MDN-RNNs are found for the three prepared datasets as seen in table 11, using the hyperparameter search space shown in table 12. The results of the best-performing network settings for each dataset using the coarse

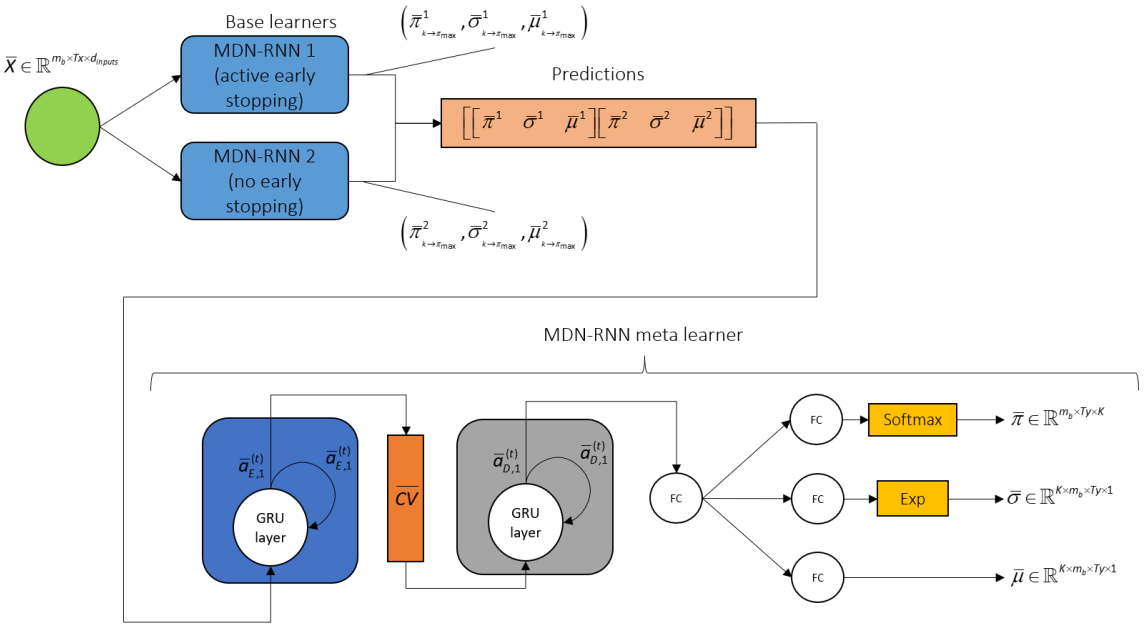


Figure 16: Ensemble model configuration

grid search are shown in table 13.

Table 13: Hyperparameter search results

Model/dataset	Lag period (hrs)	No. neurons	Batch size	No. layers	Train RMSE (kPa)	Dev RMSE (kPa)
Network-1/dataset-1	12	250	128	2	7.097	10.261
Network-2/dataset-1	12	250	128	2	4.997	5.658
Network-1/dataset-2	4	250	128	1	2.798	3.493
Network-2/dataset-2	4	250	128	1	2.47	3.433
Network-1/dataset-3	12	200	128	2	2.877	3.941
Network-2/dataset-3	12	200	128	2	2.208	3.405

The lag period refers to the hours into the past used to make the input tensor. The number of neurons is per layer, and the number of layers refers to the number of encoder/decoder pairs. As mentioned above, the data is split into a training set and development set. The accuracy of each is given by the root mean squared error (RMSE) in kPa. The RMSEs are calculated as the difference between the actual dataset values and predicted values for the network-1 architecture. For the network-2 architecture, the RMSEs are calculated as the difference between the actual values and the most probable mean output values $\bar{\mu}_{max(\pi)}$ predicted by the MDN. The number of mixing components was also tweaked to find the optimal K value. 1, 3 and 7 K components were tested and it was found that $K = 3$ had the lowest RMSE and variance. As was expected, the configuration of network-2 outperforms network-1 for all 3 datasets as seen in table 13. The dataset with the most information, dataset-3 had the lowest overall RMSE for training and development.

5.3.2 Results of networks trained using different datasets

In this section, the performance of the six models will be discussed in detail.

The actual versus predicted backpressures are shown in figure 17, these are the outputs of the standard RNN model for each of the three datasets explained in table 11. The model is incapable of predicting the backpressures effectively when only provided with the gross generation. Although a small correlation is found between these two variables, the majority of the time-series signals do not overlap since there is not enough information in the input to accurately describe the output. The importance of weather conditions on the performance of the ACC is highlighted by the dataset-2 model results, it is seen that the addition of weather data significantly improved the predictive capability of the model. The model predictions cannot reach the outer limits of the time-series signal, showing it is incapable of capturing the high backpressure peaks and the relatively high-frequency fluctuations in the data. It is however capable of capturing the general trend of the peaks and troughs. The cause of this is relatively low-frequency of the weather signals which typically exhibit slow changes in measurements. It is seen from the results of dataset-3, that the addition of high-frequency plant measurement data enables the model to predict higher-frequency changes in backpressure. This enables the model to capture more intricate details that affect backpressure due to the added information from the plant operations.

Figure 18 shows the mean prediction versus observed backpressure results using the network-2 configuration are shown for the prepared development datasets. Additionally, the 95% confidence interval bounds of the predictions are shown, which is calculated as $CI_{95\%} = \mu \pm 1.96\sigma$. Recall that μ is the predicted mean and σ is the predicted standard deviation per timestamp. The model is still unable to capture the backpressure using dataset-1, despite the addition of the MDN. The upper and lower bands of the CI has quite a large range over the predictions. This indicates that there is a large measure of uncertainty in the predictions, which is due to the ineptitude of the model to find a pattern in the input data which describes the systematic and random error as well as the output data. Network-1 was previously unable to capture many of the outlier data points in the model for dataset-2, the results in figure 18 show that the observed points seen in figure 17 now falls within the 95% confidence interval, which shows that the model generalises well. Dataset-3 shows similar results where the CI bands encapsulate more of the actual outlier data points when compared to the results in figure 18. The outlier band is also noticeably more tapered to the signal for dataset-3 compared to dataset-2, which indicates that the power plant data adds more information regarding the measured and random uncertainty contained in the process. This visual narrowing is indicative of a higher expected model certainty in the predicted mean values.

Despite having the added features from the plant operation, both network-1 and network-2 show that they are both unable to capture the extreme peaks above 35 kPa on dataset-3. It is seen that besides the networks still struggling to accurately estimate all the high backpressure peaks for dataset-2 and dataset-3, these peaks are also not contained in the uncertainty ranges. Recall from section 5.2.2 that the extreme outliers due to sensor failure had to be clipped to 40 kPa to remove these unrealistic peaks from the data while still maintaining the integrity of the noise and trends in the data. This limitation of the model regarding high peaks could be caused by discrepancies in the data, rather than the innate inability of the model to fit these values. It is possible that

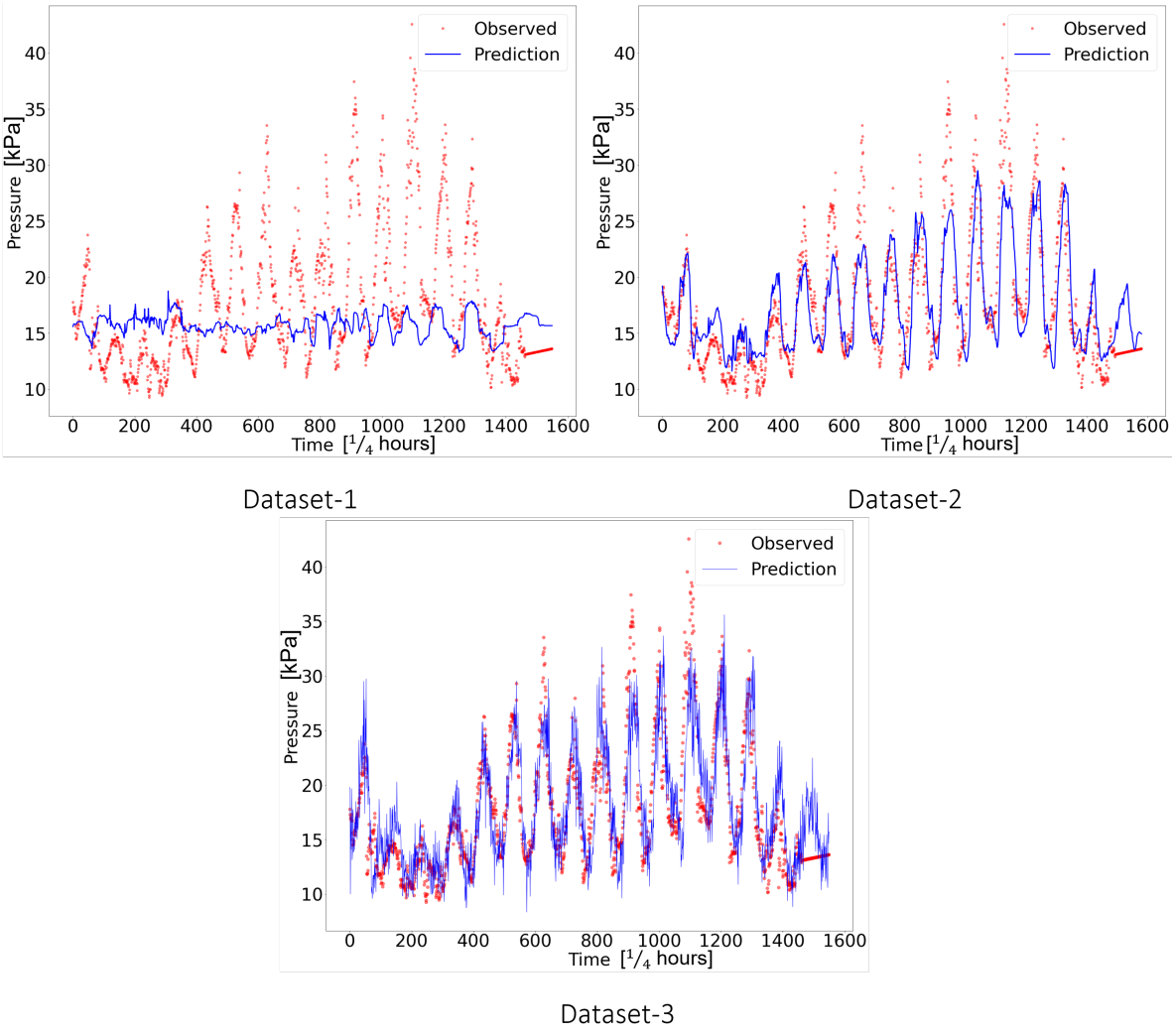


Figure 17: Actual and predicted outputs using standard encoder-decoder RNN (network-1) for the three prepared development (testing) datasets

since a sensor failure is represented as the highest possible calibrated value, some of these peaks do not have corresponding plant data hence no real pattern can be found to describe these peaks. A noteworthy feature of the model is its ability to output substantial increases in the uncertainty bandwidth around these peaks which could be used as a proxy to indicate an upcoming anomaly in the system.

An overall assessment of network-1 shows the obvious drawback that its inability to capture the non-deterministic nature of the plant behaviour and struggles with high dimensionality of the feature space. Typical shortcomings of the MDN approach, on the other hand, are numerical instability during training, mode distribution mode collapse and sensitivity to network initialisation

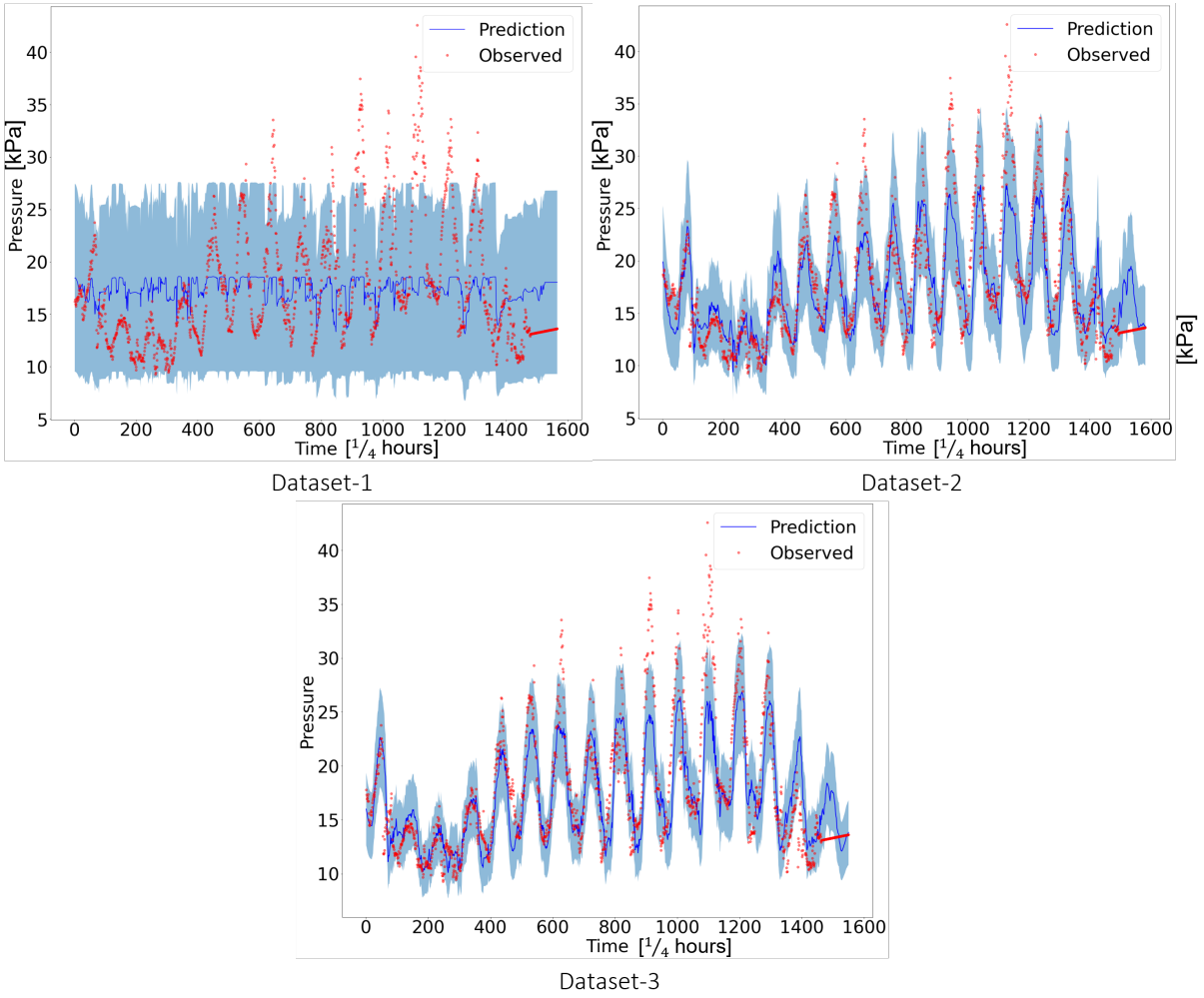


Figure 18: Actual and predicted outputs using standard encoder-decoder RNN (network-1) for the three prepared development (testing) datasets with uncertainty band

[37].

Figure 19 shows the error distribution graphs for the training and development dataset mean predictions using network-1 and network-2. For network-1 using dataset-1, it is seen that approximately 25% of the training dataset predictions have mean absolute percentage errors (MAPEs) below 10%. For the development/testing dataset of the same model and dataset combination, only 22% of the out-of-sample predictions have MAPEs below 10%. For network-1 trained on dataset-2, approximately 50% of the training dataset and 38% of the development dataset predictions have MAPEs below 10%. For network-1 trained on dataset-3, approximately 45% of the training dataset predictions and 36% of the development dataset predictions have MAPEs below 10%. The error distribution data for network-1 variants show that the network trained using dataset-2 therefore has

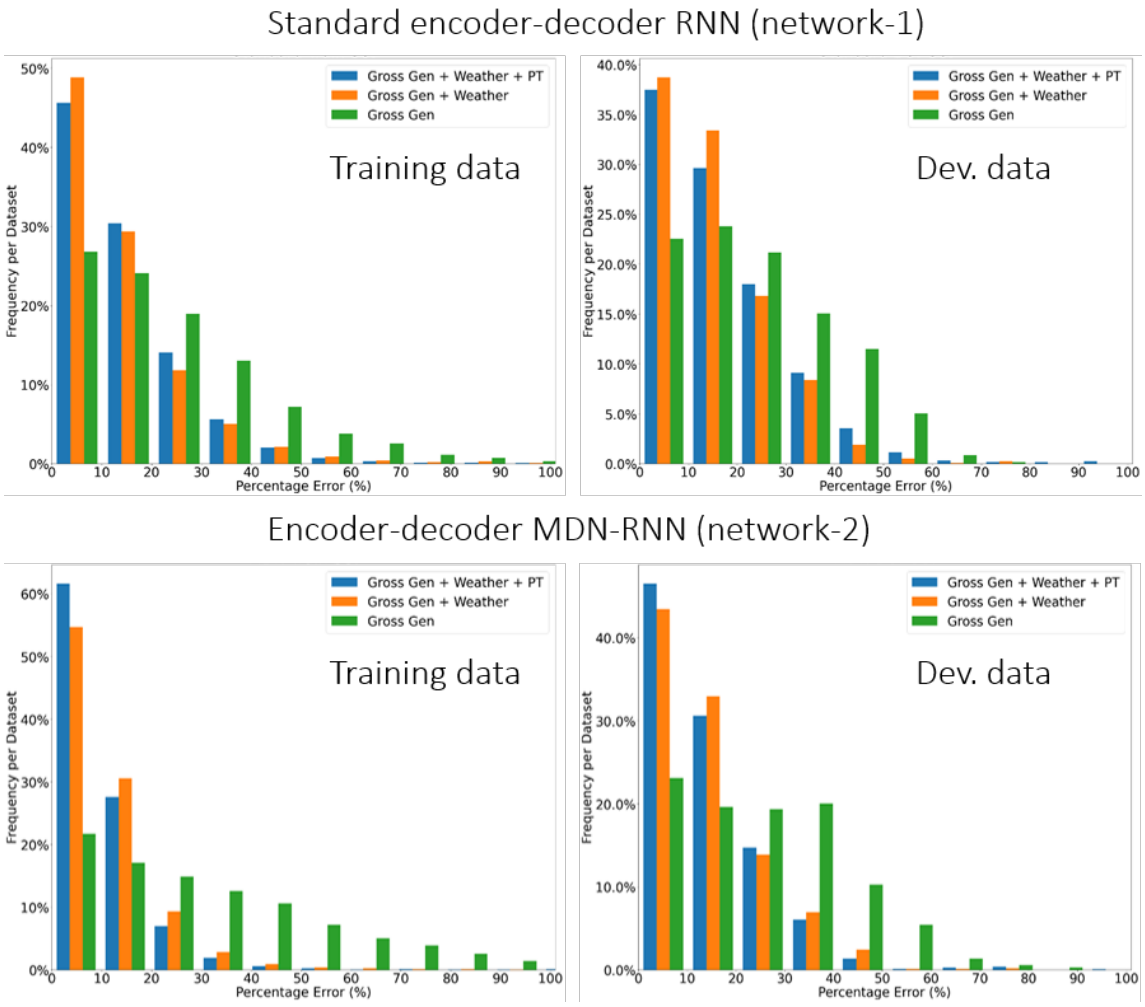


Figure 19: Error distributions for selected network architectures for prepared datasets

the best performance. For the network-2 using dataset-1 results, it is seen that the training dataset and development dataset have respectively, approximately 28% and 22% of their predictions with MAPEs below 10%. For the dataset-2 network-2 combination, 55% of the training dataset and 42% of the development dataset have MAPEs below 10%. The results of the network-2 configuration trained using dataset-3 show that 60% of the training dataset and approximately 46% of the development dataset predictions have MAPEs below 10%. It can be seen from figure 19, that the best performing model overall is network-2 using dataset-3 since the distribution has a strong right skew, meaning that the majority of the data as low percentage errors, with only a few (±1%) data points lying above the 50% error margin.

Figure 20 shows the mean absolute errors (MAE) per output sequence step ($t = 1 \rightarrow Ty$) averaged over all predictions in the validation datasets using the network-2 architecture. The average error

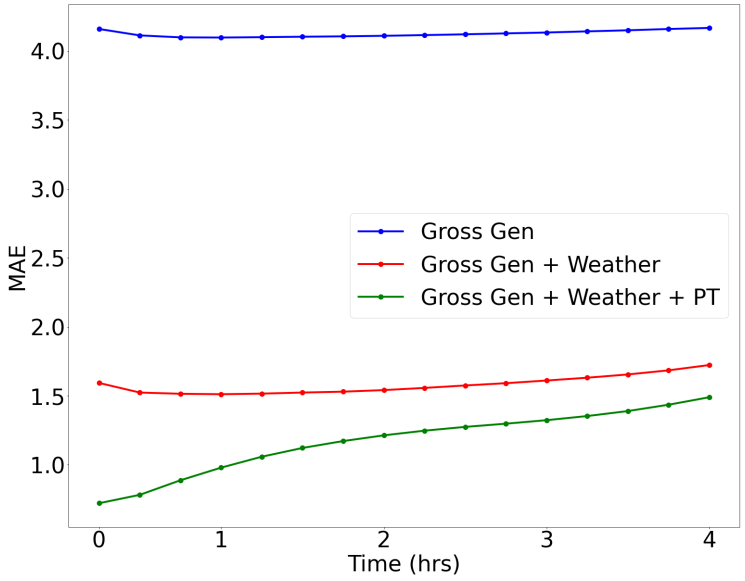


Figure 20: Development dataset MAE per time step

per timestep into the future for dataset-1 is nearly constant and also relatively high compared to the other two datasets. This is due to the models overall inability to predict variations in the output data as well the limited temporal quality of the gross generation data alone. There is a slight increasing trend with dataset-2, and dataset-3 shows seen that the further the predicted time step is into the future, the higher the error. Dataset-3 has the highest error at the largest historic lag period (4 hours into the past) due to the accumulation of the temporal prediction errors. The plot also further reiterates that the model trained using dataset-3 yields the lowest errors consistently when compared to the models using datasets 1 and 2.

5.3.3 Ensemble model results

It is known that the best performing network architecture is network-2 using the features from dataset-3. In the current section, a model with a proclivity to fit outliers will be discussed. The model is trained using this same architecture and combined to the early-stopped model using the stacking ensemble method detailed in section 3.4. This model will be an unregularised version, since it will be trained for the full 3000 epochs without the early-stopping enabled. Figure 21 show the history graphs for the regularised and unregularised models.

The early-stopping aborts the training at approximately 550 epochs, where both the training and validation error have plateaued to an optimally low error. The unregularised model however, has evident overfitting since the training error decreases but the development (validation) error increases after a few hundred epochs. The generalisation error, in turn, increases. This phenomenon is most likely a result of higher noise and variance errors in the predicted conditional probability distribution that indicate a higher tendency of the model to fit outliers which is the objective of the unregularised model [37].

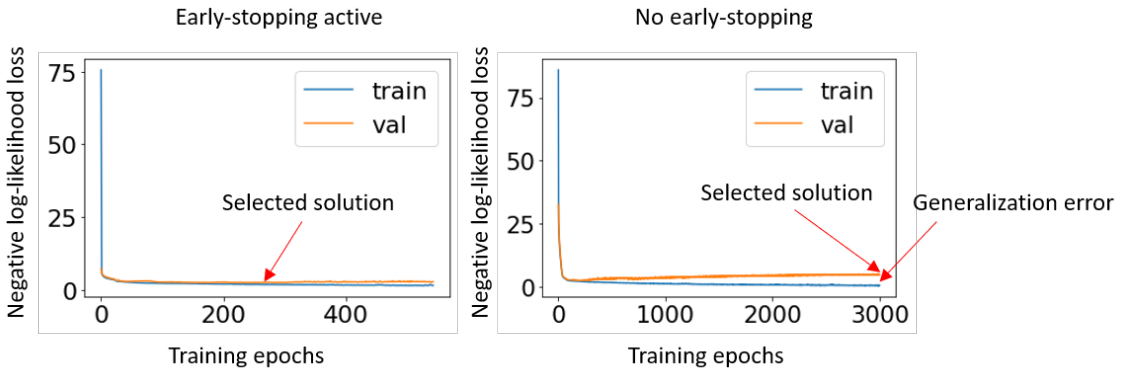


Figure 21: Training history for early-stopped and unregularised encoder-decoder MDN-RNN architectures trained using dataset-3

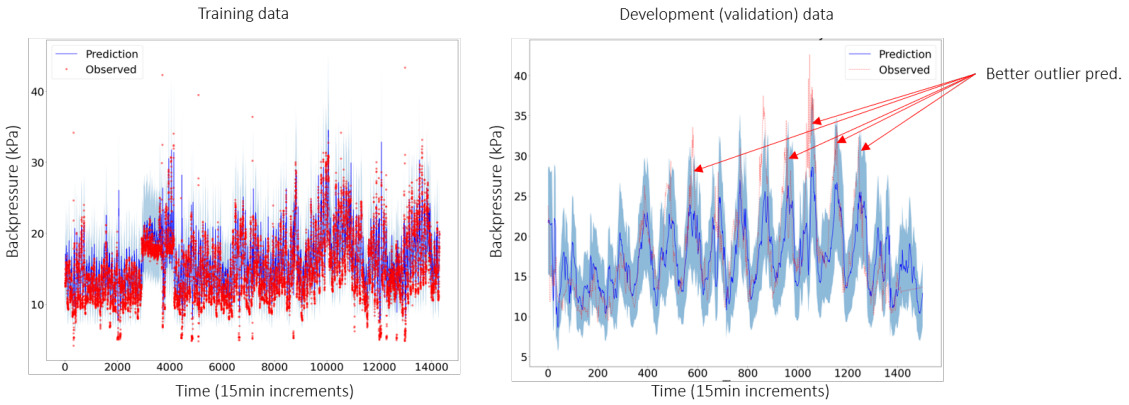


Figure 22: Training and development dataset actual data and predictions along with 95% CI bands

The outputs of these two base models are combined to form the input data to the meta-learner model. Figure 22 shows the predictions of the ensemble MDN-RNN along with the actual backpressure values. Comparing the development time-series plot in figure 22 to figure 15, it is visibly evident that the meta-learner is more capable of outputting a 95% confidence interval that encapsulates the backpressure extreme outliers. However, despite the model's improved performance, it still struggles to capture the most extreme peaks. But as mentioned previously, this is because both models used to develop the meta-model were unable to capture the anomalies in the data which were based off filtered output data with uncorrelated input data.

The overall error metrics for the ensemble model can be seen in table 14. For both the training and development datasets, the average model uncertainty (95% CI) is 1.94 kPa , which given the average backpressure being approximately 17.66 kPa, this 11% uncertainty is acceptable. Furthermore, the results in the table show that the ensemble model development dataset RMSE is 8.4% lower than the network-2 model trained using early-stopping as seen in table 13. The ensemble

Table 14: Error metrics for trained ensemble model.

Error metric	Training data	Development data
Avg. uncertainty 95% CI (kPa)	1.94	1.94
RMSE (kPa)	2.492	3.12
MAPE (%)	18.7	17.5
Actual data points within 95% CI (%)	91.4	85.1

meta-learner model achieved an accuracy of 81.3% and 82.5% on the training dataset and development data respectively, deduced from the MAPEs seen in table 14. It is important to ensure the actual data points within 95% CI is as high as possible, while maintaining a reasonable bandwidth as seen in table 15. A model that makes predictions which encompass a high percentage of the actual backpressure values is able to be deployed usefully since its 95% CI can capture most variations in the back pressure due to exogenous circumstances. In table 14, it is shown that 91.4% of the actual backpressure training data points fall within the predicted CI and 85.1% of the development data points fall within this range, indicating that the model can capture the majority of changes in the system.

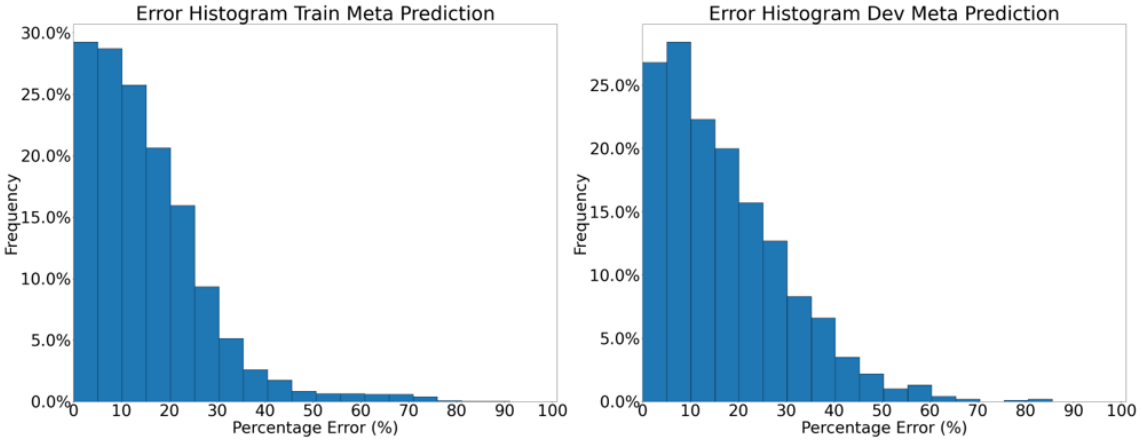


Figure 23: Error distribution graphs for ensemble model predictions using training and development datasets

Table 15: Confidence level search space for encoder-decoder RNN and MDN-variant

Confidence Level	Model interval coverage (%)	Prediction interval coverage (%)	Model prediction width [kPa]	Prediction interval width [kPa]
95%	85.1	93.87	10.55	17.33
80%	70.8	84.6	6.89	11.32
68%	59.2	74.53	5.38	8.84

Figure 23 shows the error distribution graphs for the ensemble model training and development

dataset predictions. It is seen that approximately 57% of the training dataset and 52% of the development dataset predictions have MAPEs below 10%.

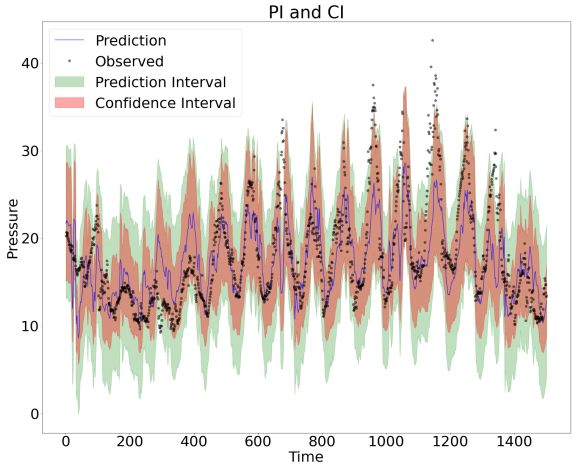


Figure 24: Meta-learner development prediction with confidence interval and prediction interval

The PI was computed and compared to the confidence interval to account for the uncertainty associated with predicting a mean value as well as the random variation of individual values. Table 15 shows the interval coverage and average interval width which are the two factors of comparison. The prediction interval contains 93.87% of the observed data and had an average width of 17.33 kPa which is an acceptable result for the desired 95% confidence interval.

6 Conclusion

This thesis concludes by highlighting the final results from the work and summarising the key points from the study. In addition, opportunities for future work will be outlined.

6.1 Summary of work and final results

For the first case study, the HTC values for seven heat exchangers were found using classic mass and energy calculations. Following this, a standard MLP was developed and optimised, then the MDN probabilistic module was added. In addition, a sensitivity analysis was conducted. Considering the standard model had a development error of $3.22\frac{W}{m^2K}$, the MDN shows an improvement by achieving a reduced error of $3.11\frac{W}{m^2K}$. Inference from this model showed that 57% of the training set had an error of less than 10%, and 60% of the development set had predictions with an error less than 10%. This margin of error is considered more than adequate. It should be noted that further development was done to find the most robust set of hyperparameters which yielded the following results: 76.83% of the observed values fell within the model interval coverage, the development set had an average error of $4.89\ \frac{W}{m^2K}$ which equates to an overall error of 10%. From the results of the sensitivity analysis it was found that the excess air ratio had a dominating effect on the predictions being made, with the main steam temperature and feedwater flow rates having secondary significant effects. These observations confirm that the flue gas flow and steam-side flow are the major driving forces for heat transfer.

In the second case study regarding the ACC backpressure predictor, various models were developed and the standard RNN and RNN-MDN versions were compared. The complexity of the datasets was explored, by increasing the feature space from a simple load case through to the full model which included the load, weather conditions and plant pressure and temperature readings. From the results presented in section 5.3.2, it can be concluded that despite the dataset complexity, the MDN outperforms the standard RNN. Thus further justifying the efficacy of predicting multiple distributions when considering multi-modal thermal systems. It can be seen that although the single load case using dataset-1 was unable to produce useful predictions, the addition of only weather data showed impressive increases in performance by decreasing the average error from 5.658 kPa to 3.433 kPa. The further addition of the plant features improved error from 3.433kPa to 3.405kPa as seen in table 13, which indicates this added information actually was less significant than one would assume. This added information did however provide the model with sufficient information to better infer the uncertainty band. The highlight of this section of research was the development of the ensemble learning approach, which yielded the lowest overall error of 3.12 kPa. Broadly translated, these results show how models can benefit from the combined exploitation of both outfitting and generalised predictors. An important finding was the effect of forecasting period on accuracy, where it can be seen that dataset-2 had greater stability as a function of time compared to dataset-3. The final model had a total of 85.1% of the observed data falling within the predicted model interval coverage with a width of 10.55 kPa, which compared to the much wider prediction interval coverage indicates that the model uncertainty quantification is statistically valid.

In conclusion, the objective to develop probabilistic machine learning models and explore the efficacy of applying such models to thermal engineering has been met since sufficiently accurate results were achieved in both case studies. It was found in both cases that the MDN variation of the models performed better than the standard MLP models as seen by the results. However, mention should be made to the limitations of the MDN. They require large computational power to train and cannot be computed on CPUs alone due to the large number of trainable parameters. They are also very delicate and require rigorous manual hyperparameter tuning beyond the standard grid search to ensure stable training, consistent predictions and reproducible results.

6.2 Future work

In future work, applications of the ACC model developed in the second case study could be explored. This model architecture could be implemented into a full dynamic deploy which can provide feedback to the power plant system operators, informing them of the future state of the ACC backpressure. Using the predictive inference achieved by the model, it is possible to forecast when the backpressure will be outside the designed operating range. A further application is the use of the network-2/dataset-2 model using forecast weather data. Load demand profiles are usually determined in advance of power generation and an accurate backpressure prediction using only this load profile and weather data can inform the plant of the likelihood and attainability of this load being generated. Future studies based on this work could explore the prospects of integrating physics based equations into the machine learning process for better extrapolation of the predictions.

Further research could be done to develop the boiler HTC model to incorporate the transient behaviour of the boiler system. A potential application of the boiler HTC model might explore the integration of these predicted HTCs into Flownex modelling. This would allow a process model to calculate heat transfer from gas to steam using HTCs inferred from real plant data. Furthermore, corresponding metal temperatures can then be extracted which enables the study of the materials' remnant life to inform preventative maintenance.

References

[1] *Sensors*, (21):1–20.

[2] *Combustion and gasification in fluidized beds*. CRC/Taylor Francis, 2006.

[3] *Theory and Calculation of Heat Transfer in Furnaces*. Elsevier Science and Technology, San Diego, 2016.

[4] Intelligent steam power plant boiler waterwall tube leakage detection via machine learning-based optimal sensor selection. *Sensors*, 20:6356, 2020.

[5] Sina Ardabili, Amir Mosavi, and Annamária R Várkonyi-Kóczy. Advances in machine learning modeling reviewing hybrid and ensemble methods. In *Engineering for Sustainable Future*, Lecture Notes in Networks and Systems, pages 215–227. Springer International Publishing, Cham, 2020.

[6] Edmond Baranes, Julien Jacqmin, and Jean-Christophe Poudou. Non-renewable and intermittent renewable energy sources: Friends and foes? *Energy Policy*, 111:58–67, 2017.

[7] Ismail Bilgen, Goktug Guvercin, and Islem Rekik. Machine learning methods for brain network classification: Application to autism diagnosis using cortical morphological networks. *Journal of Neuroscience Methods*, 343:108799, 2020.

[8] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[9] Anne Sjoerd Brouwer, Machteld van den Broek, Ad Seebregts, and André Faaij. Impacts of large-scale intermittent renewable energy sources on electricity systems, and how these can be modeled. *Renewable and Sustainable Energy Reviews*, 33:443–466, 2014.

[10] Francois Chollet et al. Keras, 2015.

[11] Christopher N. Davis, T. Deirdre Hollingsworth, Quentin Caudron, and Michael A. Irvine. The use of mixture density networks in the emulation of complex epidemiological individual-based models. *PLOS Computational Biology*, 16(3):1–16, 03 2020.

[12] Xiaoze Du, Lihua Liu, Xinming Xi, Lijun Yang, Yongping Yang, Zhuxin Liu, Xuemei Zhang, and Cunxi. Back pressure prediction of the direct air cooled power generating unit using the artificial neural network model. *Applied Thermal Engineering*, 31(14):3009–3014, 2011.

[13] Xiaoze Du, Lihua Liu, Xinming Xi, Lijun Yang, Yongping Yang, Zhuxin Liu, Xuemei Zhang, Cunxi Yu, and Jinkui Du. Back pressure prediction of the direct air cooled power generating unit using the artificial neural network model. *Applied Thermal Engineering*, 31(14):3009–3014, 2011.

[14] Martin Felder, Anton Kaifel, and Alex Graves. Wind power prediction using mixture density recurrent neural networks. In *Poster presentation Gehalten auf der European wind energy conference*, 04 2010.

[15] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[16] Aurélien Géron. Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*, 2017.

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

[18] Rashid A. Haffejee and Ryno Laubscher. Application of machine learning to develop a real-time air-cooled condenser monitoring platform using thermofluid simulation data. *Energy and AI*, 3:100048, 2021.

[19] Qinggang Lu Yunkai Sun Haibo Wu, Man Zhang. The heat transfer coefficients of the heating surface of 300 mwe cfb boiler. *Journal of Thermal Science*, 21:368–372, 2012.

[20] Safarudin Gazali Herawan, Kamarulhelmy Talib, Azma Putra, Ahmad Faris Ismail, Shamsul Anuar Shamsudin, and Mohd Tahir Musthafah. Prediction of generated power from steam turbine waste heat recovery mechanism system on naturally aspirated spark ignition engine using artificial neural network. *Soft Computing*, 22(18):5955–5964, 2018.

[21] Prabhas Hundi and Rouzbeh Shahsavari. Comparative studies among machine learning models for performance estimation and health monitoring of thermal power plants. *Applied Energy*, 265:114775–, 2020.

[22] David Ireri, Eisa Belal, Cedric Okinda, Nelson Makange, and Changying Ji. A computer vision system for defect discrimination and grading in tomatoes using machine learning and image processing. *Artificial Intelligence in Agriculture*, 2:28–37, 2019.

[23] M.J. Kochenderfer and T.A. Wheeler. *Algorithms for Optimization*. The MIT Press. MIT Press, 2019.

[24] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, volume 34, pages 1–5, 2017.

[25] Ryno Laubscher. Time-series forecasting of coal-fired power plant reheater metal temperatures using encoder-decoder recurrent neural networks. *Energy*, 189:116187, 2019.

[26] Ryno Laubscher and Pieter Rousseau. Numerical investigation into the effect of burner swirl direction on furnace and superheater heat absorption for a 620 mwe opposing wall-fired pulverized coal boiler. *International Journal of Heat and Mass Transfer*, 137:506–522, 2019.

[27] Ryno Laubscher and Pieter Rousseau. Application of generative deep learning to predict temperature, flow and species distributions using simulation data of a methane combustor. *International Journal of Heat and Mass Transfer*, 163:120417, 2020.

[28] Ryno Laubscher and Pieter Rousseau. An enhanced model of particle radiation properties in high ash gas-particle dispersion flow through industrial gas-to-steam heat exchangers. *Fuel (Guildford)*, 285:119153–, 2021.

[29] Ryno Laubscher and Pieter Rousseau. An integrated approach to predict scalar fields of a simulated turbulent jet diffusion flame using multiple fully connected variational autoencoders and mlp networks. *Applied Soft Computing*, 101:107074, 2021.

[30] Jian Li, Yan Bai, and Bo Li. Operation of air cooled condensers for optimised back pressure at ambient wind. *Applied Thermal Engineering*, 128:1340–1350, 2018.

[31] Jennifer Lin, Allison J. Mahvi, Taylor S. Kunke, and Srinivas Garimella. Improving air-side heat transfer performance in air-cooled power plant condensers. *Applied Thermal Engineering*, 170:114913, 2020.

[32] Peiqing Liu, Huishen Duan, and Wanli Zhao. Numerical investigation of hot air recirculation of air-cooled condensers at a large power plant. *Applied Thermal Engineering*, 29(10):1927–1934, 2009.

[33] You Lv, Feng Hong, Tingting Yang, Fang Fang, and Jizhen Liu. A dynamic model for the bed temperature prediction of circulating fluidized bed boilers based on least squares support vector machine with real operational data. *Energy*, 124:284–294, 2017.

[34] Zhongxian Men, Eugene Yee, Fue-Sang Lien, Deyong Wen, and Yongsheng Chen. Short-term wind speed and power forecasting using an ensemble of mixture density neural networks. *Renewable Energy*, 87:203–211, 2016.

[35] Chigozie Nwankpa, W. Ijomah, Anthony Gachagan, and Stephen Marshall. In *2nd International Conference on Computational Sciences and Technologies*, 12 2020.

[36] Weizhu Qian, Fabrice Lauri, and Franck Gechter. Supervised and semi-supervised deep probabilistic models for indoor positioning problems. *Neurocomputing*, 435:228–238, 2021.

[37] Renita Raidoo and Ryno Laubscher. Data-driven forecasting with model uncertainty of utility-scale air-cooled condenser performance using ensemble encoder-decoder mixture-density recurrent neural networks. *Energy*, 238, 2022.

[38] Pieter Rousseau and Ryno Laubscher. A thermofluid network-based model for heat transfer in membrane walls of pulverized coal boiler furnaces. *Thermal Science and Engineering Progress*, 18:100547, 2020.

[39] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[40] Hosham Salim, Khalid Faisal, and Raheel Jawad. Enhancement of performance for steam turbine in thermal power plants using artificial neural network and electric circuit design. *Applied Computational Intelligence and Soft Computing*, 2018:1–9, 2018.

[41] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[42] Akash Singh, Vivek Sharma, Siddhant Mittal, Gopesh Pandey, Deepa Mudgal, and Pallav Gupta. An overview of problems and solutions for components subjected to fireside of boilers. *International Journal of Industrial Chemistry*, 9, 12 2017.

[43] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 2014.

[44] Dawid Taler, Marcin Trojan, Piotr Dzierwa, Karol Kaczmarski, and Jan Taler. Numerical simulation of convective superheaters in steam boilers. *International Journal of Thermal Sciences*, 129:320–333, 2018.

[45] Peng Tan, Biao He, Cheng Zhang, Debei Rao, Shengnan Li, Qingyan Fang, and Gang Chen. Dynamic modelling of nox emission in a 660mw coal-fired boiler with long short-term memory. *Energy*, 176:429–436, 2019.

[46] Kai Ting and Ian Witten. Stacked generalization: when does it work? 11 1997.

[47] J. A. van Rooyen. Performance trends of an air-cooled steam condenser under windy conditions. *University of Stellenbosch*.

[48] Alex J.C. Witsil and Jeffrey B. Johnson. Volcano video data characterized and classified using computer vision and machine learning algorithms. *Geoscience Frontiers*, 11(5):1789–1803, 2020.

[49] Haizhou Wu, Xuejun Liu, Wei An, Songcan Chen, and Hongqiang Lyu. A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils. *Computers Fluids*, 198:104393, 2020.

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

[51] Jinhua Zhang, Jie Yan, David Infield, Yongqian Liu, and Fue sang Lien. Short-term forecasting and uncertainty analysis of wind turbine power based on long short-term memory network and gaussian mixture model. *Applied Energy*, 241:229–244, 2019.

Appendix A Machine learning model code

A.1 Part 1: Coal-fired boiler case study

A.1.1 MLP model

The following code was used to develop the final MLP model.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import json
import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import InputLayer
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Split the dataset
X_train, X_dev, X_test, Y_train, Y_dev, Y_test = split_data(inputs, outputs)

# Define model parameters
numHiddenUnits = 10
numFeatures = inputs.shape[1]
numResponse = outputs.shape[1]

# Build sequential model
model = keras.Sequential()
model.add(layers.Dense(numHiddenUnits, activation="relu", input_shape=(13,)))
model.add(
    layers.Dense(
        numHiddenUnits,
        activation="relu",
        activity_regularizer=keras.regularizers.l2(1e-3),
    )
)
model.add(
    layers.Dense(
        numHiddenUnits,
        activation="relu",
        activity_regularizer=keras.regularizers.l2(1e-3),
    )
```

```
)
# model.add(layers.Dense(numHiddenUnits, activation="relu"))
model.add(layers.Dense(7, activation="relu"))

callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=150,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)

# Compile the model
model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.0001), loss="mean_squared_error"
)

# Fit the model and plot the training history
model.summary()
history = model.fit(
    X_train,
    Y_train,
    epochs=1000,
    batch_size=128,
    callbacks=[callback],
    verbose=2,
    shuffle=True,
    validation_data=(X_dev, Y_dev),
)

plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="val")
plt.legend()
plt.show()

# Make predictions
Ypred = model.predict(X_train)
Ydev_pred = model.predict(X_dev)
```

A.1.2 MLP-MDN model

The following code was used to develop the final MLP-MDN model.

```
# Define the MDN model architecture
numHiddenUnits = 15 # number of hidden units
numFeatures = inputs.shape[1] # number of input features
numResponse = outputs.shape[1] # number of output features
num_components = 2 # Number of components K in the mixture
event_shape = 7 # output feature dimension
params_size = tfp.layers.MixtureNormal.params_size(num_components, event_shape) #
    define MDN output dimension

# Build sequential model
model = keras.Sequential()
model.add(layers.Dense(numHiddenUnits, activation="relu", input_shape=(numFeatures,)))
model.add(layers.BatchNormalization())
model.add(
    layers.Dense(
        numHiddenUnits,
        activation="relu",
        activity_regularizer=keras.regularizers.l2(5e-2),
    )
)
model.add(layers.BatchNormalization())
model.add(
    layers.Dense(
        numHiddenUnits,
        activation="relu",
        activity_regularizer=keras.regularizers.l2(5e-2),
    )
)
model.add(layers.BatchNormalization())
model.add(
    layers.Dense(
        numHiddenUnits,
        activation="relu",
        activity_regularizer=keras.regularizers.l2(5e-2),
    )
)
model.add(layers.Dense(7, activation="relu"))

model.add(layers.Dense(params_size, activation=None))
```

```
model.add(tfp.layers.MixtureNormal(num_components, event_shape))

callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=500,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)

# Compile the model
model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.000005),
    loss=lambda Y_train, model: -model.log_prob(Y_train),
)

# Fit the model to the data
model.summary()
history = model.fit(
    X_train,
    Y_train,
    epochs=7500,
    batch_size=256,
    callbacks=[callback],
    verbose=2,
    shuffle=False,
    validation_data=(X_dev, Y_dev),
)

# Plot the training and validation errors
plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="val")
plt.legend()
plt.show()

# Make predictions
Ypred = model.predict(X_train)
Ydev_pred = model.predict(X_dev)


# Function to extract pi, mu and sigma from the multidimensional predictions
```

```
def MDN(X):
    yhat = model(X) # map inputs to outputs using the trained model object
    print("yhat shape:", yhat.shape)
    print(yhat.submodules) # extract prediction submodules
    yhat_means = yhat.submodules[2].mean().numpy() # array of predicted mean output
    print("yhat means:", yhat_means.shape)
    yhat_stddev = yhat.submodules[2].stddev().numpy() # array of standard deviations
    print("yhat std dev:", yhat_stddev.shape)
    yhat_pi = yhat.submodules[1].probs_parameter().numpy() # array of probabilities
    print(
        "yhat prob param:", yhat_pi.shape
    )

    probs = np.amax(yhat_pi, axis=1) # get max probability
    print("Probs", probs.shape)
    print(np.average(probs))

    component = np.argmax(yhat_pi, axis=1) # get component K with max probability
    print("Component vector", component)
    print(np.average(component))


    Y_pred = np.zeros((yhat.shape[0], yhat.shape[1]))
    Y_stddev = np.zeros((yhat.shape[0], yhat.shape[1]))

    # Loop to get corresponding mean and standard deviation for best component K
    for i in range(yhat.shape[0]):
        for j in range(7):
            Y_pred[i, j] = yhat_means[i, component[i], 0]
            Y_stddev[i, j] = yhat_stddev[i, component[i], 0]
    return (Y_pred, Y_stddev) # save these outputs


train_pred, train_stddev = MDN(X_train)
test_pred, test_stddev = MDN(X_dev)
```

A.2 Part 2: ACC case study

A.2.1 RNN model

The following code was used to develop the standard network-2 RNN model for dataset-3.

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
import json
import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Sliding window function
def arrange_data(inputs, outputs, lagwindow=1, leadwindow=1):
    X, Y = [], []
    for t in range(len(inputs) - lagwindow - leadwindow - 1):
        drange_in = inputs[t : (t + lagwindow)]
        drange_out = outputs[(t + lagwindow) : (t + lagwindow + leadwindow)]
        X.append(drange_in)
        Y.append(drange_out)
    X = np.array(X)
    Y = np.array(Y)
    return X, Y


lagwindow = 48 # lag period as 15-minute increments (12 hrs)
leadwindow = 16 # lead period as 15-minute increments (4 hrs)

# Split the data
X_train, X_dev, X_test, Y_train, Y_dev, Y_test = split_data(X, Y)

# Arrange data into 3-D tensors
Xtrain, Ytrain = arrange_data(
    X_train, Y_train, lagwindow=lagwindow, leadwindow=leadwindow
)
Xdev, Ydev = arrange_data(X_dev, Y_dev, lagwindow=lagwindow, leadwindow=leadwindow)

# Define model architecture
numHiddenUnits1 = 200
inputLength = lagwindow
outputLength = leadwindow
numFeatures = 49

# Build sequential model with GRU layers
```

```
model = keras.models.Sequential()
# Encoder
model.add(
    keras.layers.GRU(
        numHiddenUnits1,
         activity_regularizer =keras.regularizers.l2(1e-4),
        input_shape=(inputLength, numFeatures),
        activation ="relu",
         kernel_initializer ="glorot_uniform",
        return_sequences=True,
    )
)
model.add(
    keras.layers.GRU(
        numHiddenUnits1,
         activity_regularizer =keras.regularizers.l2(1e-4),
        input_shape=(inputLength, numFeatures),
        activation ="relu",
         kernel_initializer ="glorot_uniform",
        return_sequences=False,
    )
)
# Context vector
model.add(keras.layers.RepeatVector(Ytrain.shape[1]))
model.add(
    keras.layers.GRU(
        numHiddenUnits1,
         activity_regularizer =keras.regularizers.l2(1e-4),
        activation ="relu",
         kernel_initializer ="glorot_uniform",
        return_sequences=True,
    )
)
# Decoder
model.add(
    keras.layers.GRU(
        numHiddenUnits1,
         activity_regularizer =keras.regularizers.l2(1e-4),
        activation ="relu",
         kernel_initializer ="glorot_uniform",
        return_sequences=True,
    )
)
```

```
# Time-series output layer
model.add(keras.layers.TimeDistributed(keras.layers.Dense(Ytrain.shape[2])))


callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=1000,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)

# Compile the model
model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.0001), loss="mean_squared_error"
)

# Fit the model to the data
history = model.fit(
    Xtrain,
    Ytrain,
    epochs=1000,
    batch_size=512,
    callbacks=[callback],
    verbose=2,
    shuffle=False,
    validation_data=(Xdev, Ydev),
)

# Plot the training and validation errors
plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="val")
plt.legend()
plt.show()

# Make predictions
Ypred = model.predict(Xtrain)
Ydev_pred = model.predict(Xdev)
```

A.2.2 MDN-RNN model

The following code was used to develop the network-2 MDN-RNN model for dataset-3.

```
# MDN parameters
numHiddenUnits1 = 200
inputLength  = lagwindow
outputLength  = leadwindow
numFeatures = 49
numResponse = 1 # Features output shape
num_components = 5 # Number of components in the mixture
event_shape  = 1   # Shape of the target
params_size  = tfp . layers .MixtureNormal.params_size(num_components, event_shape)

# Build MDN-RNN model
model = keras .models. Sequential ()
model.add(
    keras . layers .GRU(
        numHiddenUnits1,
        input_shape =(inputLength, numFeatures),
        activation ="relu",
          kernel_initializer  ="glorot_normal",
          activity_regularizer  =keras. regularizers .l2(1e-4),
        return_sequences =True,
    )
)

model.add(
    keras . layers .GRU(
        numHiddenUnits1,
        input_shape =(inputLength, numFeatures),
        activation ="relu",
          kernel_initializer  ="glorot_normal",
          activity_regularizer  =keras. regularizers .l2(1e-4),
        return_sequences =False,
    )
)
model.add(keras . layers .RepeatVector( Ytrain .shape[1]) )
model.add(
    keras . layers .GRU(
        numHiddenUnits1,
        activation ="relu",
          kernel_initializer  ="glorot_normal",
          activity_regularizer  =keras. regularizers .l2(1e-4),
```

```
            return_sequences =True,
        )
)
model.add(
        keras . layers .GRU(
            numHiddenUnits1,
             activation =”relu”,
              kernel_initializer  =”glorot_normal”,
              activity_regularizer  =keras. regularizers .l2(1e−4),
             return_sequences =True,
        )
)

model.add(keras . layers . TimeDistributed ( keras . layers .Dense(Ytrain.shape [2]) ) )

# Mixture normal layers
model.add(
        keras . layers . TimeDistributed ( keras . layers .Dense(params_size,  activation =None))
)
model.add( tfp . layers .MixtureNormal(num_components, event_shape))

callback  = tf . keras . callbacks .EarlyStopping(
        monitor=” val_loss ”,
        min_delta=0,
        patience =3000,
        verbose=0,
        mode=”auto”,
        baseline =None,
         restore_best_weights  =False,
)

# Define custom loss function
loss  = lambda y, rv: −rv.log_prob(y)

model.compile( optimizer=tf . optimizers .Adam(learning_rate=0.0001), loss=loss)

history  = model. fit (
        Xtrain ,
        Ytrain ,
        epochs=3000,
        batch_size =128,
        callbacks =[ callback ],
        verbose=2,
```

```
    shuffle =False,
    validation_data =(Xdev, Ydev),
)

plt.plot(history.history["loss"], label="train")
plt.plot(history.history["val_loss"], label="val")
plt.legend()
plt.show()


# Sampled prediction
Ypred = model.predict(Xtrain)
Ydev_pred = model.predict(Xdev)

# MDN prediction
def MDN(X):
    yhat = model(X)
    yhat_means = yhat.submodules[2].mean().numpy()
    yhat_stddev = yhat.submodules[2].stddev().numpy()
    yhat_pi = yhat.submodules[1].probs_parameter().numpy()

    probs = np.amax(yhat_pi, axis=2) # Probability parameter pi

    component = np.argmax(yhat_pi, axis=2)

    Y_pred = np.zeros((yhat.shape[0], yhat.shape[1], 1)) # mu
    Y_stddev = np.zeros((yhat.shape[0], yhat.shape[1], 1)) # sigma

    for i in range(yhat.shape[0]):
        for j in range(yhat.shape[1]):
            Y_pred[i, j, 0] = yhat_means[i, j, component[i, j], 0]
            Y_stddev[i, j, 0] = yhat_stddev[i, j, component[i, j], 0]
            # for each time step j use the index stored in components[i,j]
    return (Y_pred, Y_stddev, probs)
```