UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Wind Power Forecasting using Hybridised Artificial Neural Network Approaches

Liaan van Heerden
21590923

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Prof H. J. Vermeulen

November 2021

Acknowledgements

I would like to extend my gratitude to Prof HJ Vermeulen as his support and insights throughout the semester were paramount to implementing this project successfully. I thoroughly enjoyed working with Prof Vermeulen and I wish to one day be as competent and professional as he is. I would also like to extend my gratitude for access to the dataset utilised in this project.

Society as a whole including myself owes a great debt to every contributor to the open source TensorFlow library. I would like to sincerely thank the TensorFlow authors for enabling the proliferation of easily realisable deep learning models. Without this library and other implementations like it, this project would have been extremely difficult.

I would also like to thank my family namely my parents, Eliana and Schalk van Heerden, and my brother, GC van Heerden, for your unwavering support during my undergraduate degree, both financially and emotionally. Thank you for all the great times in Bloemfontein during the holidays and thank you for all the long phone calls and interesting conversations. Above all, thank you for fostering a happy, loving, and nurturing family environment.

Finally, I would like to thank my friends from Stellenbosch University for keeping me company and entertained throughout my time there and for making this whole journey worth while.

UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

21590923 Studentenommer / *Student number*	[illegible] Handtekening / *Signature*
L. van Heerden Voorletters en van / *Initials and surname*	13 November 2021 Datum / *Date*

Abstract

English

With the ever higher penetration of variable renewable energy (VRE) sources in the grid, short term generation forecasting (STGF) of VRE sources is becoming an important mechanism to protect the grid's security as well as to inform energy trading. This project proposes a variety of artificial neural network (ANN) based forecasting models to forecast power production for a utility-scale wind farm for a range of time horizons. This project utilises long short term memory (LSTM) as well as gated recurrent unit (GRU) based ANN model topologies to produce these forecasts. The effect of hybridising these models with empirical mode decomposition (EMD) was also investigated. A nuanced approach to model the wind turbine power curve with ANN topologies was also explored.

Afrikaans

Met die groeiende hoeveelheid veranderlike hernubare energie bronne in die kragnetwerk word die behoefte aan kort-termyn krag voorsienings vooruitskattings al hoe belangriker om die sekuriteit van die krag netwerk te handhaaf en om energie handel in te lig. Hierdie projek stel 'n verskeidenheid artifisiële neurale netwerk gebaseerde modelle voor om windkrag produksie vooruit te skat vir 'n grootskaalse windplaas oor 'n reeks tye. Die projek benut lang kort termyn geheue asook gehekte herhalende eenheid gebaseerde artifisiële neurale netwerke. Die neurale netwerke is gehibridiseer met empiriese wyse dekomposisie en daar is ook gepoog om die wind turbine se krag kurwe te modeleer met die neurale netwerke se topologie.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

t	Timestep.
$\phi()$	Activation Function.
$\boldsymbol{H}$	Hidden state matrix.
$\boldsymbol{X}$	Input matrix.
$\boldsymbol{W}$	Weight matrix.
$\boldsymbol{b}$	Bias vector.
$\boldsymbol{O}$	Output matrix.
$tanh()$	Tanh activation function.
$\sigma()$	Sigmoid activation function.
x	Refers to the input.
r	Refers to the reset gate.
h	Refers to the hidden state.
z	Refers to the update gate.
$\tilde{\boldsymbol{H}}$	Candidate hidden state matrix.
$\odot$	Elementwise multiplication operator (Hadamard product).
$\boldsymbol{I}$	Input gate matrix.
$\boldsymbol{F}$	Forget gate matrix.
$\boldsymbol{O}$	Output gate matrix.
$\tilde{\boldsymbol{C}}$	Candidate memory cell matrix.
$\boldsymbol{C}$	Memory cell matrix.
i	Refers to input gate.
f	Refers to forget gate.
o	Refers to output gate..
c	Refers to candidate memory gate.
MAE	Mean absolute error.
$MAPE$	Mean absolute percentage error.
MSE	Mean squared error.

$RMSE$	Root mean squared error.
$\Sigma_{i=1}^{n}$	Sigma operator.
y_i	Observed value.
x_i	Actual value.
W	Watt.
G	Giga.
T	Tera.
%	Percentage.
$mins$	Minutes.
m/s	Meters per second.
$x(t)$	Denotes a signal which undergoes empirical mode decomposition.
$e_{min}(t)$	Denotes a cubic spline of local maxima.
$e_{max}(t)$	Denotes a cubic spline of local minima.
$m(t)$	Denotes the mean of a signal.
$d(t)$	Denotes a detailed signal.

Acronyms and abbreviations

ANN	Artificial neural network
EMD	Empirical mode decomposition
LSTM	Long short term memory
GRU	Gated recurrent unit
VRE	Variable renewable energy
OCGT	Open cycle gas turbine
CCGT	Combined cycle gas turbine
CSP	Concentrated solar power
RNN	Recurrent neural network
IMF	Intrinsic mode function
MAE	Mean absolute error
MAPE	Mean absolute percentage error
MSE	Mean squared error
RMSE	Root mean sqquared error
CPU	Computer processing unit
GPU	Graphics processing unit

Chapter 1

Introduction

1.1. Project Background

Wind power generation is the conversion of kinetic energy in the wind to electricity, usually facilitated by wind turbines [1]. This is a renewable form of energy generation characterized by high capacity factors, cheap levelized cost, the ability to generate power at all times of the day, and the ability to generate power in bad weather conditions [2]. Wind energy is generally touted as one of the keystone technologies to facilitate a decarbonised grid. Wind energy, however, does not come without drawbacks. This power source is highly intermittent and a high penetration of wind generation in a power system threatens the power system's security [3].

Electricity generation by wind has grown appreciably in modern times. The total wind generation capacity in the world has risen from 24 to 743 GW from 2001 to 2020. Globally, wind energy accounts for 1412.4 TWh of energy annually [4]. In South Africa, wind energy generation accounts for 2.5 GW of nominal capacity or about 4.85 % of the total nominal capacity for public power generation. For the year 2020, the monthly energy generated by wind power in South Africa varied between 404 to 678 GWh throughout the year [5].

While wind energy generation is a promising source of renewable energy, the intermittency associated with wind-based energy can cause a cascade of problems in power systems. These problems include more complex power system planning, erratic generation dispatching, price fluctuations, negative pricing in deregulated energy markets, the heightened need for utility-scale energy storage as well as the complication of maintenance in the power system [6], [7]. Power generation from wind energy sources is also highly stochastic in nature [8]. This behaviour complicates some of these problems even further.

Forecasting of wind power output is a standard industry practise. Wind power forecasts are used to inform energy trading, to optimally schedule maintenance as well as to balance energy storage in the grid. This addresses some of the problems associated with the intermittency of wind-based power generation. Accurate forecasting can greatly assist in upholding the security of the power system, which in turn will prevent the need for protective measures in the power system.

1.2. Project Motivation

The main motivation of this project is to explore the effect of hybridised artificial neural network (ANN) based approaches on the forecasting of wind power output for a utility-scale wind farm. The hybridisation process will make use of empirical mode decomposition (EMD) applied to the historical power generation data of a wind farm. The idea is that features are captured in the historical power generation data which can be extracted by the decomposition process. The ANN models will then have their topologies augmented to simulate the power curve of a wind turbine. This will hopefully lead to more accurate forecasting of wind power generation.

The forecasting will be done using various models which forecast the following 30 mins to 24 hours of power generation. The scope of this project will be limited to developing forecasting models to alleviate pressures experienced by balancing the grid, planning maintenance activities, and energy trading. This is mainly due to constraints regarding the dataset.

This project will cover various hybridised and non-hybridised ANN-based approaches. This structure will allow for a comparative analysis on different approaches and will also enable a contextual understanding of the ANN-based machine learning models.

1.3. Project Objectives

This project has various objectives and deliverables, all are aimed at contextualizing and comparing the ANN forecasting techniques. The objectives of this project are:

- Implementing a basic (long short term memory) LSTM-based ANN without hybridisation to establish a machine learning baseline.
- Optimising various hyperparameters of the basic LSTM-based ANN without hybridisation.
- Applying hybridisation to the basic LSTM model to explore the influence of the hybridisation process as well as optimising the hyperparameters of this model.
- Augmenting the topology of the hybridised LSTM model in order to better simulate the wind turbine power curve and optimising the hyperparameters of this model.

- Implementing a hybridised (gated recurrent unit) GRU model and optimising the hyperparameters of this model.
- Augmenting the topology of the hybridised GRU model to better simulate the wind turbine power curve and optimising the hyperparameters of this model.
- Comparing and analysing the outputs from the abovementioned models and recommending future work.

1.4. Report Outline

Chapter 1 in this report will cover the background on the problems addressed in the project as well as the motivation for the project and the project objectives.

Chapter 2 will detail a review of the literature relevant to the project. This review is split up into a review on wind power technology and activities and a review on ANN concepts.

Chapter 3 details the methodology followed to implement various models in the execution of this project. This chapter is mainly a retrospective review on the algorithms used to implement this project. This chapter is written very generally, with the goal of informing readers to such a degree that the project can also be implemented by a competent programmer.

Chapter 4 exposes the project's results for different models. This chapter summarises the models implemented, the ANN topology, and the results obtained for various forecasting horizons with different ANN topologies.

Chapter 5 delivers a discussion on the results obtained in Chapter 4, discusses the shortcomings of the techniques used and features recommendations for future work.

Chapter 2

Literature Review

2.1. Wind Power Generation

To fully appreciate the need for wind power forecasting models, a good understanding of wind power generation and the typical wind power generation behaviour must be acquired. An overview of the influences of wind power on the grid will also be given. It should be noted that the data utilized in this project is from a utility-scale wind farm operating the typical three bladed, horizontal axis, wind-facing turbine which proliferates in industry. This type of turbine is most commonly used to generate large amounts of energy in practise [1].

2.1.1. Wind Power Curves

Wind turbines typically have a power curve associated with every specific model of turbine. This curve describes the power generation of the wind turbine with different wind speeds [9]. A typical wind power curve is illustrated in Figure 2.1:

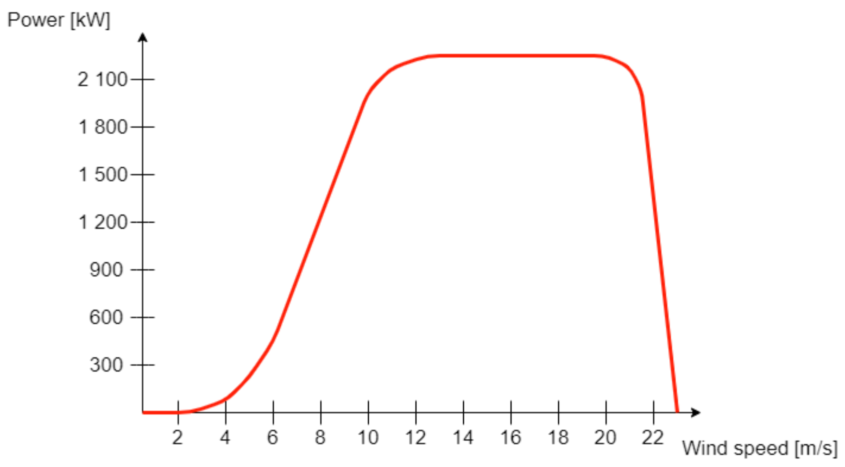


Figure 2.1: Illustration of a typical wind turbine power curve

Salient aspects of the curve includes the cut-in wind speed (2.3 m/s), the non-rated region of the curve (2.3 to 12 m/s), the rated region (12 to 20 m/s) and the cut-out wind speed. The wind turbine will start to produce power at the cut-in wind speed. This power will increase in accordance to the power curve in the non-rated region as the wind speed increases and will settle at the rated power output in the rated region of the power curve. The turbine will go into high speed shutdown when the cut-out speed is exceeded [10].

The implication is that the wind power curve governs the behaviour of the entire wind farm as the turbines are homogeneous. This means that the power output data will have a certain maximum bound (the rated power of the wind farm) and a minimum bound of zero.

2.1.2. Intermittency of Wind Power Generation

Wind power generation is a highly intermittent form of renewable energy and is grouped under a classification commonly referred to as ”variable renewable energy” (VRE) [6]. Some effects of this intermittency on the power system as well as solutions to the problem will be discussed further in the sections following.

2.1.3. Balancing Power in the Grid

Power system balancing in this contexts refers to the activity which utility operators partake in to ensure that there is adequate energy generated in the grid to supply the demanded load.

2.1.3.1. Power station dispatching

In a traditional power system, stations are grouped into baseload, and peaking stations. These groupings indicate the priority in which the stations will be dispatched. The dispatching process is informed by the levelized cost of electricity produced by the stations and the speed which the station can ramp up power production. Baseload stations usually consists of coal or nuclear energy. Peaking stations consists of open cycle gas turbines (OCGT), combined cycle gas turbines (CCGT), and hydro power [11].

With higher penetration of VRE in the grid, the traditional notion of power station dispatching has changed. It is now necessary to have flexibility in the power system to accommodate for intermittency in the power supply and to keep the power system in balance. A versatile power system dispatching routine with low ramp up times are thus of imperative importance in a grid with high VRE penetration. Unfortunately, most modern grids run a mixture of VRE and traditional thermal power generation. The grid operator thus always runs the risk of having the power system becoming unbalanced. This bolsters the need for forecasting power production from VRE sources [7], [12].

2.1.3.2. Energy storage

As a direct consequence to the intermittency of VRE sources and their increasing penetration in the modern grid, utility-scale energy storage systems have become more popular. The specifics of these systems vary but some examples could be: large battery installations, pumped hydro power, molten salt storage in concentrated solar power (CSP) plants, thermal batteries and compressed air energy storage to name a few [13].

The basic idea is that energy storage mechanisms capture energy produced from VRE sources in times of excess and delivers energy in times of low production from VRE sources. To adequately inform the energy storage process, some energy generation forecasting model must be developed for this activity.

2.1.4. Planning of Maintenance

The grid is a complex system and as such, there is a lot of maintenance to be done on the various subcomponents. In a future decarbonised grid with high VRE penetration, planned maintenance activities could be scheduled around the power generation forecast of the near future. The maintenance activities could then be successfully implemented with minimum disruption to the power supply as certain maintenance activities like servicing of wind turbines could be scheduled on days where there would be low energy output.

2.1.5. Energy Trading

Energy trading and more specifically electricity trading is the trading of units of electricity between suppliers and utility operators before the electricity itself is generated. Energy trading in deregulated electricity markets like the European Union and the United States forms a large part of the mechanism in which power is sold on to consumers. This form of trade handles electricity like a future [14]. Accurate pricing of the traded electricity is thus important to ensure profitability on the part of the producer of electricity. Pricing VRE sources like wind power is notoriously difficult due to its variable nature. The higher penetration of VRE sources in modern grids can also cause price fluctuations due to market forces and in some cases electricity has even been traded at negative prices - in essence becoming a liability for the energy producer [15].

Accurate forecasting of power generation from VRE sources would inform energy trading and thus ensure that electricity is sold on at a profitable price. Integrating energy storage with VRE sources with the express goal of energy trading is also becoming a feasible option for energy suppliers improve the profitability of electricity generated.

2.1.6. Wind Power Forecasting

According to [16], 94 percent of grid operators are of the opinion that the forecasting of wind power is a necessity to facilitate higher integration of wind power in the power system. Wind power forecasting can be facilitated by a variety of techniques, these include persistence methods (naive models), numerical weather prediction based models, statistical models, machine learning models, and hybrid models incorporating any mixture of the aforementioned and even introducing mathematical concepts into these models [17], [18].

Comparing previous research with the results obtained in this project will be difficult as the performance metrics used for these models vary greatly and are not necessarily comparable to each other (see Section 2.3.5). Furthermore, to find research focusing on the exact time horizons for forecasting addressed in this project will necessitate a complete meta analysis of research on wind power forecasting. As this field is widely researched with varying success, this task was deemed to be out of the scope of this project. This project consequently focuses on providing enough varied models to facilitate a comparative analysis on recurrent neural network (RNN) based wind power forecasting.

2.2. Data Exploration and Preprocessing

This chapter details the literature relevant to handling and preprocessing data in the machine learning context where this project takes place.

2.2.1. Data Exploration and Error Handling

Data exploration and error handling refers to the activity of validating given data to ensure that the data is fit for processes regarding data analysis.

For the time-series data which is processed in this project, the errors which can occur in this data are mainly limited to measurement errors (outliers and faulty measurements) and missing values. Measurement errors can be imputed or truncated while missing values are commonly imputed. Data imputation is a wide field with various acceptable techniques [19], [20].

The data utilised in this project is however of extremely high quality with basically no errors save for a measurement error at the very end of the time-series data. Hence, the error handling in this project will be limited to truncating this error.

2.2.2. Empirical Mode Decomposition

Empirical mode decomposition (EMD) is a time-domain decomposition used to disassemble signals (time-series data) into their substituent components. This method is very comparable to mathematical transforms like the Fourier transfrom and the wavelet transform. The result of EMD is a variety of signals called intrinsic mode functions (IMFs) which can be used to reconstitute the original signal [21]. The process to extract IMFs is called sifting.

The sifting process for a signal $x(t)$ is defined by the following algorithm:

1. Find the local maximums and minimums in signal $x(t)$.
2. Envelope all the local maximums and minimums and fit cubic splines to these points. These two splines are defined as $e_{min}(t)$ and $e_{max}(t)$.
3. Calculate the mean signal between the two splines as $m(t) = \frac{e_{max}(t)-e_{min}(t)}{2}$.
4. Extract the detail from signal $x(t)$ with the signal $m(t)$. This is done with: $d(t) = x(t) - m(t)$.
5. Repeat step 1. to 4. on the signal $m(t)$.

This process is repeated for the desired amount of IMFs or until no more detail can be extracted.

In practise, step 1-4 is refined until the signal $d(t)$ is zero-meaned. This satisfies the stop criterion for EMD and the signal is then classified as an IMF of $x(t)$. Only once as ignal is classified as an IMF can step 5 commence.

2.2.3. Data Normalisation

Some machine learning models including the recurrent neural networks (RNNs) utilised in this project are extremely sensitive to the normalisation scheme employed on its input data. The difference between an excellent model and a lacklustre one may lie in the normalization scheme and implementation [22].

Traditionally, ANN-based models have their features normalised to either a scheme of [0;1] or [-1; 1]. These schemes seem to work the best with most activation functions in these models [20].

2.3. Artificial Neural Networks

This section details some theory on the ANNs and ANN cells which are implemented in the execution of this project. This section is by no means a complete review of all the literature regarding these ANNs as there is an immense knowledge base on this topic. This section attempts to shed enough light on this topic to understand the fundamentals of the models implemented in Chapter 4. Note that unless otherwise stated, the information in this section was based on [23].

2.3.1. Recurrent Neural Networks

Recurrent neural networks are a subset of neural networks with what is in effect a feedback loop in their structure. RNNs are used to implement persistence in a artificial neural network. This artificial neural network structure is illustrated in Figure 2.2. A common way to represent these ANNs are in an unrolled form (seen in Figure 2.3).

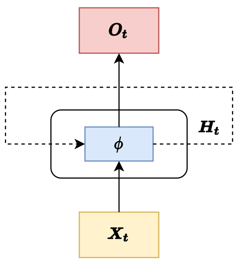


Figure 2.2: Basic RNN cell

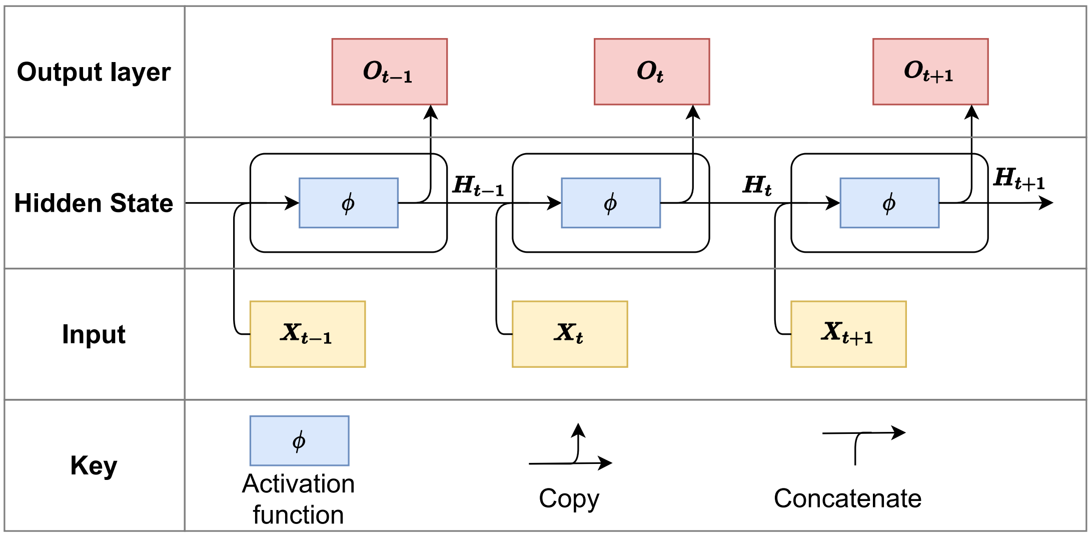


Figure 2.3: Unrolled basic RNN

The hidden state $\boldsymbol{H}_t$ and the output $\boldsymbol{O}_t$ for timestep t are defined as:

$$\boldsymbol{H_t} = \phi(\boldsymbol{X_t W_{xh}} + \boldsymbol{H_{t-1} W_{hh}} + \boldsymbol{b_h}) \tag{2.1}$$

$$\boldsymbol{O_t} = \boldsymbol{H_t W_{hq}} + \boldsymbol{b_q} \tag{2.2}$$

Note that the hidden state and output for the timestep $t + 1$ can be found in a similar fashion. Equation 2.1 and 2.2 shows mathematically how persistence (or memory) is implemented in ANNs.

2.3.2. Backpropagation Through Time and the Vanishing and Exploding Gradient Problem in RNNs

Since RNNs are optimized with gradient descent, the need arises to compute the gradients associated with each timestep in a sequence model based on RNNs.

Backpropagation through time is the mathematical mechanism which is implemented in order to calculate the weight values in a RNN. This technique requires the repetitive application of partial differentiation in order to compute gradients associated with a current sequence step. The problem with this technique is that the repeated calculation tends to become highly dependant on model variables and parameters with each sequence step added. This has a negative effect in the sense that the effect of some dependancies disappears while others contribute to the gradient calculation to an excessive degree. We thus run into the problem of vanishing and exploding gradients.

2.3.3. GRU-based RNNs

The difference between a gated recurrent unit (GRU) RNN and a basic RNN is that the GRU RNN can ”gate” the hidden state. This means that this RNN can selectively update and reset a specific hidden state. The GRU unit has 2 gates, the update gate to capture long-term dependencies, and the reset gate to capture short-term dependencies [24]. This hidden state setup was developed to solve gradient related problems in RNNs.

Figure 2.4 illustrates a GRU RNN cell hidden state.

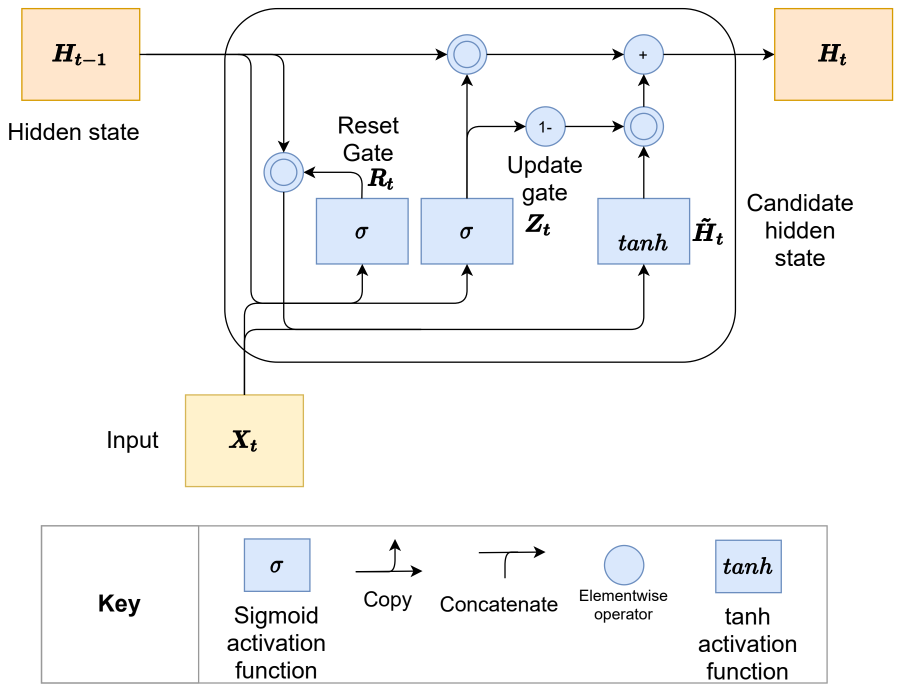


Figure 2.4: GRU hidden state

The reset gate ($\boldsymbol{R_t}$) and the update gate ($\boldsymbol{Z_t}$) are defined by Equations 2.3 and 2.4:

$$\boldsymbol{R_t} = \sigma(\boldsymbol{X_t W_{xr}} + \boldsymbol{H_{t-1} W_{hr}} + \boldsymbol{b_r}) \tag{2.3}$$

$$\boldsymbol{Z_t} = \sigma(\boldsymbol{X_t W_{xz}} + \boldsymbol{H_{t-1} W_{hz}} + \boldsymbol{b_z}) \tag{2.4}$$

The candidate hidden state ($\boldsymbol{\tilde{H}_t}$) is given by Equation 2.5:

$$\boldsymbol{\tilde{H}_t} = tanh(\boldsymbol{X_t W_{xh}} + (\boldsymbol{R_t} \odot \boldsymbol{H_{t-1}})\boldsymbol{W_{hh}} + \boldsymbol{b_h}) \tag{2.5}$$

Finally, the hidden state ($\boldsymbol{H_t}$) is given by Equation 2.6:

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \tag{2.6}$$

2.3.4. LSTM-based RNNs

The LSTM cell, first proposed in 1997 by Hochreiter and Schmidhuber [25], also poses to solve gradient problems associated with RNNs. The LSTM introduces a memory cell with the same shape as the hidden state. Some people consider the LSTM cell to be a specialised form of the hidden state in RNNs.

To control the memory cell, a set of gates are introduced. These gates are:

- The output gate - to read entries from the cell.

- The input gate - to decide when to read entries into the cell.
- The forget gate - to reset the contents of the cell.

Figure 2.5 ilustrates the LSTM hidden state.

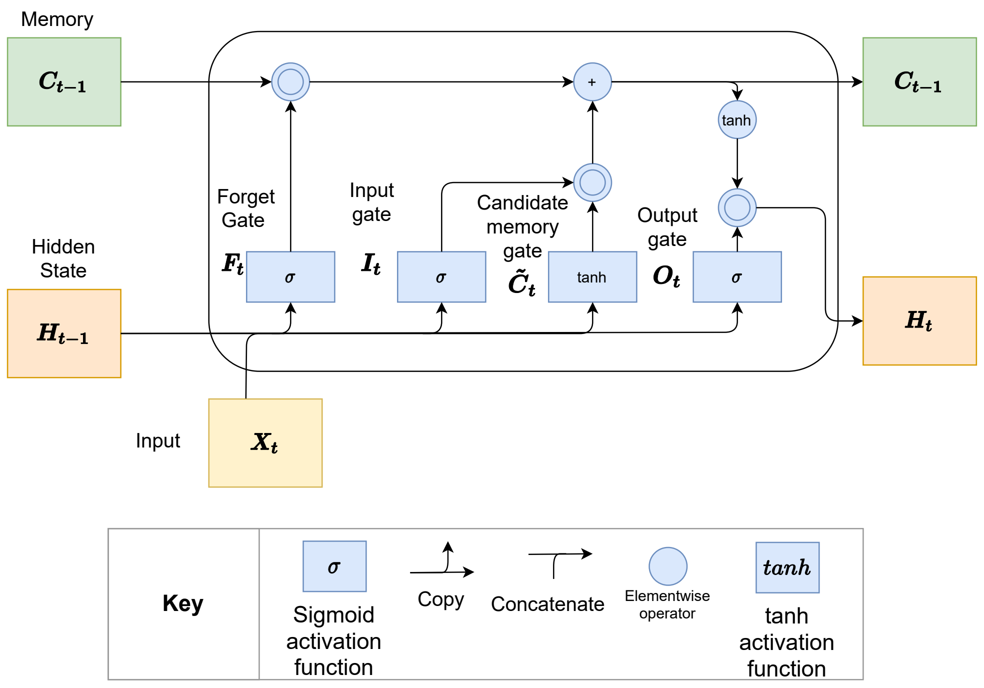


Figure 2.5: LSTM hidden state

The input gate ($\mathbf{I}_t$), forget gate ($\mathbf{F}_t$) and output gate($\mathbf{O}_t$) are defined by Equations 2.7, 2.8, and 2.9:

$$\mathbf{I}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i\right) \tag{2.7}$$

$$\mathbf{F}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right) \tag{2.8}$$

$$\mathbf{O}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o\right) \tag{2.9}$$

The candidate memory cell ($\tilde{\mathbf{C}}_\mathbf{t}$) is defined by Equation 2.10:

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c) \tag{2.10}$$

The memory cell ($\mathbf{C}_\mathbf{t}$) is defined by Equation 2.11:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \tag{2.11}$$

Finally, the hidden state ($\mathbf{H}_\mathbf{t}$) is defined by Equation 2.12:

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh\left(\mathbf{C}_t\right) \tag{2.12}$$

2.3.5. Performance Metrics

Forecasting error metrics are ways to quantify how good a forecasting model performs. There is variety of forcasting error metrics to choose from. The most popular metrics are mean absolute error (MAE), mean absolute percentage error (MAPE), mean squared error (MSE) and root mean squared error (RMSE) [26]. These metrics are defined by Equations 2.13, 2.14, 2.15 and 2.16.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{2.13}$$

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{x_i - y_i}{x_i} \right| \tag{2.14}$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{2.15}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - y_i)^2}{n}} \tag{2.16}$$

This project mainly uses MAE as an error metric.

Chapter 3

Methodology

This section outlines the methodology followed in executing the project. This project is mainly software-based and as such this section will detail the software environment used in the project, the principle methods and objects used in different aspects of the algorithm as well as the flow of the algorithm.

3.1. Software Environment

With the growth of machine learning and more specifically the deep learning subfield within machine learning, a plethora of software options has become available for the processing of tensor-based operations generally applicable to neural networks. It was decided to limit the scope of software environments to Python-based environments, since this programming language was the most familiar.

There are mainly two options to implement Python-based neural networks for this project. One being PyTorch and the other being TensorFlow. The debate on which library is the best is a contentious one with both being suitable for the needs of this project. A final decision was made on using TensorFlow since Keras (of which TensorFlow is a dependancy) is distributed with TensorFlow. This decision was made mainly due to Keras' ease of use and rapid prototyping. It should be noted that Keras implementations do perform worse than PyTorch implementations on large datasets. This should not be a problem for this project, since the dataset is relatively small.

3.2. Python Libraries

There are some pertinent Python libraries which were leveraged in the execution of the project. Table 3.1 gives a summary of the specific libraries utilised as well as a short description as to their use.

Table 3.1: Python libraries utilised in project execution.

Library	Usage
Matplotlib	General graphing of data and results
NumPy	Algorithmic manipulation of arrays
Pandas	Data Manipulation through dataframe functionality
Scikitlearn	Data normalisation
TensorFlow	Tensor operations and Keras dependancies
Keras	Implementing, training and optimizing neural networks
Emd	Used for EMD hybridisation steps
kers_tuner	Used to optimise ANN model hyperparameters

Whole Python libraries are imported with the `import` statement. For example the EMD library is imported with `import emd`. When there are more granular subcomponents of specific Python libraries, such as ”preprocessing” within scikitlearn, the statement `from scikitlearn import preprocessing` was used. Python also has the functionality to rename the import of a library. For example `import matplotlib as mpl` will change the module reference from `matplotlib` to `mpl`.

3.3. Data Exploration and Preprocessing

Data exploration and preprocessing are of paramount importance in this project as the neural networks implemented are very sensitive to this step. There is a lot of improvement in testing metrics to be found with the proper execution of this step. Furthermore, it is essential to prevent lookahead in this step, especially when considering the EMD hybridisation process.

The data acquired for this project took the form of a CSV file containing historical power generation data for a utility-scale wind farm in 30 minute intervals.

The flowdiagrams in Figure C.1 and C.2 in Appendix C summarises the processes described in this section.

3.3.1. Initial Data Exploration

Note that the principle functions for data exploration and descriptions of their use are given in Table 3.2 below.

The data is initially loaded into an instance of a Pandas DataFrame object with the `pandas.read_csv()` function. The data is then plotted with `matplotlib.plot()`

in conjunction with `pandas.to_datetime()` and `pandas.dataframe.astype()`. An extract of the data is shown in Figure 3.1.

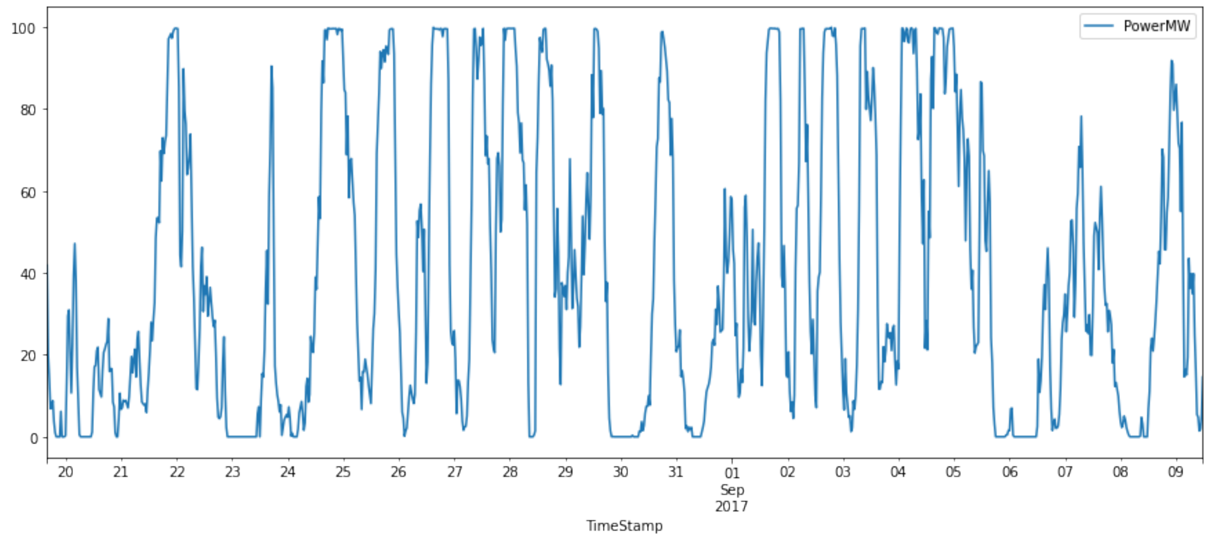


Figure 3.1: Extract from the historical wind power dataset.

The behaviour as governed by the wind power curve described in Figure 2.1 is clearly visible in Figure 3.1. Furthermore, the variability of steady-state power production as well as the stochasticity of the power output can appreciated by consulting this figure. There are some sudden drops in the output power. These drops could simply be timed with wind speeds below the cut-in speed of the turbine model. It could also represent high wind speed shutdown events or maintenance on the wind farm.

The data was then inspected with `pandas.describe()` and no obvious discrepancies like outliers or missing values were found. For prudence, it was decided to iterate through the data. This yielded a measurement error at the very end of the data. This error can be seen in Figure 3.2.

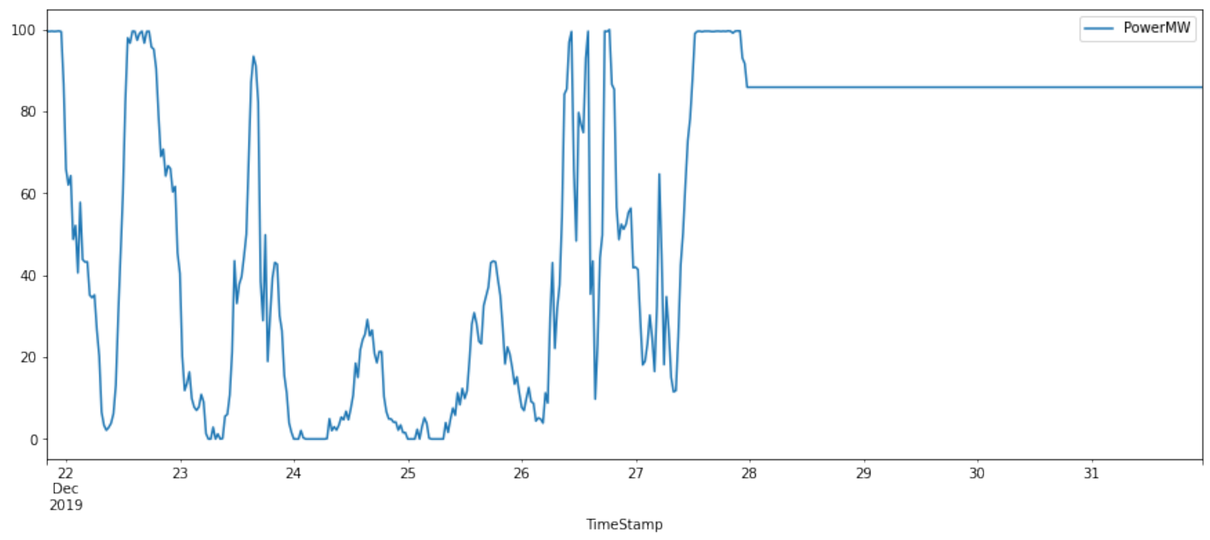


Figure 3.2: Faulty measurements in data.

The measurement error was subsequently truncated using the `pandas.dataframe.drop()` function.

After applying the aforementioned process, the data can confidently be progressed to the preprocessing phase.

3.3.2. Principle Methods and Objects used for Data Exploration

Table 3.2 below summarises the principle methods and objects used in data exploration.

Table 3.2: Principle methods and objects used for data exploration.

Function	Usage
pandas.read_csv()	Imports CSV data into a Pandas DataFrame.
pandas.to_datetime()	Converts timestamp of data to DateTime object for plotting.
pandas.dataframe.astype()	Converts values in Pandas DataFrame to float values.
pandas.dataframe.drop()	Truncates the specified amount of rows from the end of the DataFrame.
matplotlib.plot()	Plotting the data with the designated timestamp.
pandas.describe()	Describes the features in a DataFrame to indicate errors like outliers and missing values.

3.3.3. Data Preprocessing

This subsection details the process followed to preprocess the historical wind power data for use in the forecasting models. The principle functions used in this subsection are given in Table 3.3.

This subsection describes the general methodology followed with regards to preprocessing the data. All processes are not applicable to every model implemented.

3.3.3.1. Splitting Data into Training, Testing and Validation Datasets

Splitting the Pandas DataFrame into training, validation, and testing datasets was done with a 70:20:10 split. This was done with the following code which utilises some of Pandas' built-in functionality like slicing notation.

```
n = len(df)
train_df = df[0:int(n*0.7)]
val_df = df[int(n*0.7):int(n*0.9)]
test_df = df[int(n*0.9):]
```

Note that the data was not randomised before splitting into training, validation and testing sets as is commonly done with non time-series data.

3.3.3.2. Empirical Mode Decomposition

Empirical mode decomposition (EMD) of the historical power data was facilitated with PyPI's `emd` library. The empirical mode decomposition is a time-domain transform which separates a time-series signal into components called intrinsic mode functions (IMFs). Section 2.2.2 can be consulted for background on EMD.

Where applicable, the IMFs are then added to the Pandas DataFrame containing the historical power production to produce a dataset consisting of historical power production, and the first 7 IMFs produced with EMD. Note that the EMD is applied separately to the training, validation and testing datasets. This is to prevent lookahead (information leakage) in the time-series dataset.

This process was implemented by converting the historical power output for the respective dataset to a numpy array with `pandas.dataframe.to_numpy()`. The NumPy array is then processed with the `emd` library by leveraging the `emd.sift.sift()` function with the argument `max_imfs = 7`. This implements the standard sifting mask for EMD and produces a maximum of 7 IMFs. The result of the decomposition process is visualised in Figure 3.3.

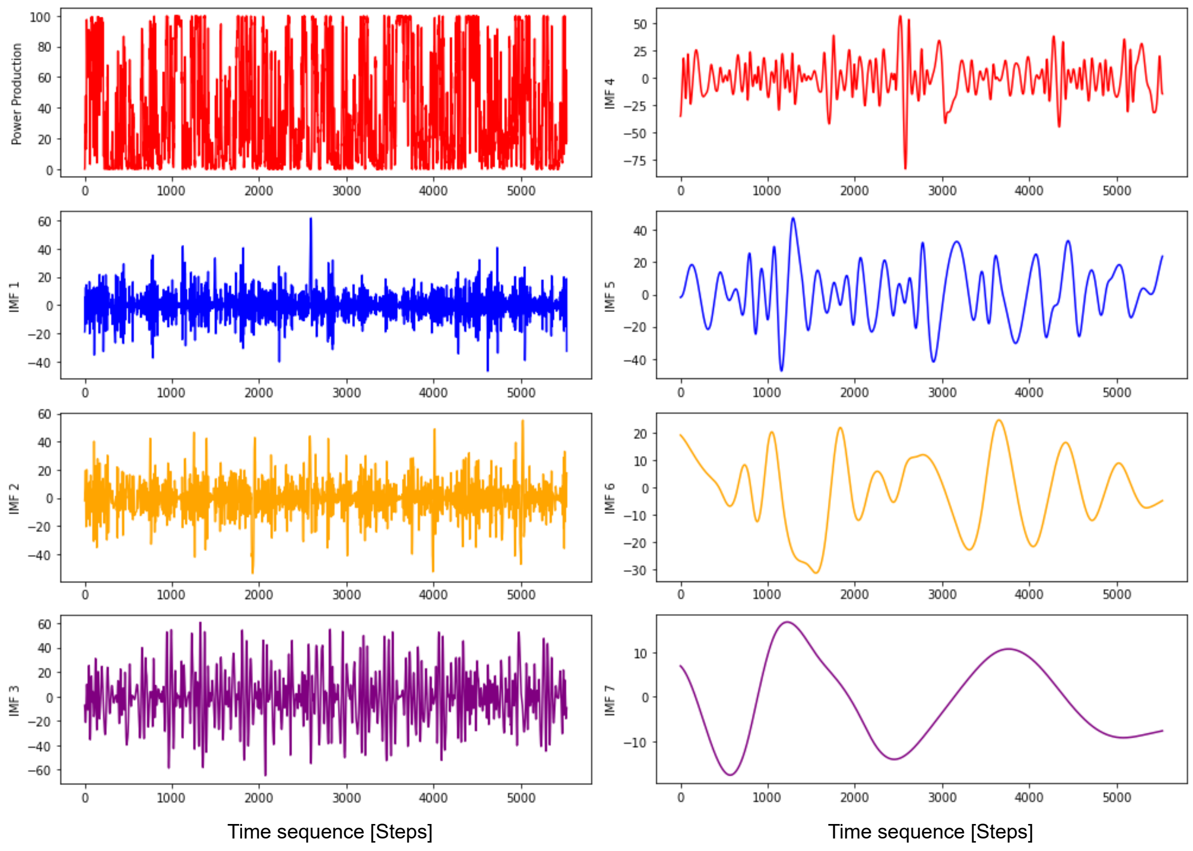


Figure 3.3: IMFs obtained from EMD on historical power output.

3.3.3.3. Data Normalisation

As described in Section 2.2, the neural networks utilised in this project perform optimally when the input features of the data is normalised to a scheme of either [-1; 1] or [0; 1]. It was decided that the data will be normalised to [0; 1] since the label data already has a minimum of 0 and a sigmoid nature.

Firstly, the `scikitlearn.preprocessing.MinMaxScaler()` will be instantiated. The instance of this object will then have its `fit()` method called on the training data. The training, validation and testing dataframes are then overwritten with the `transform()` method, called on each corresponding dataframe respectively. The normalisation process can be reversed by calling the `inverse_transform()` method.

Figure 3.4 below shows the normalised historical wind power data from Figure 3.1.

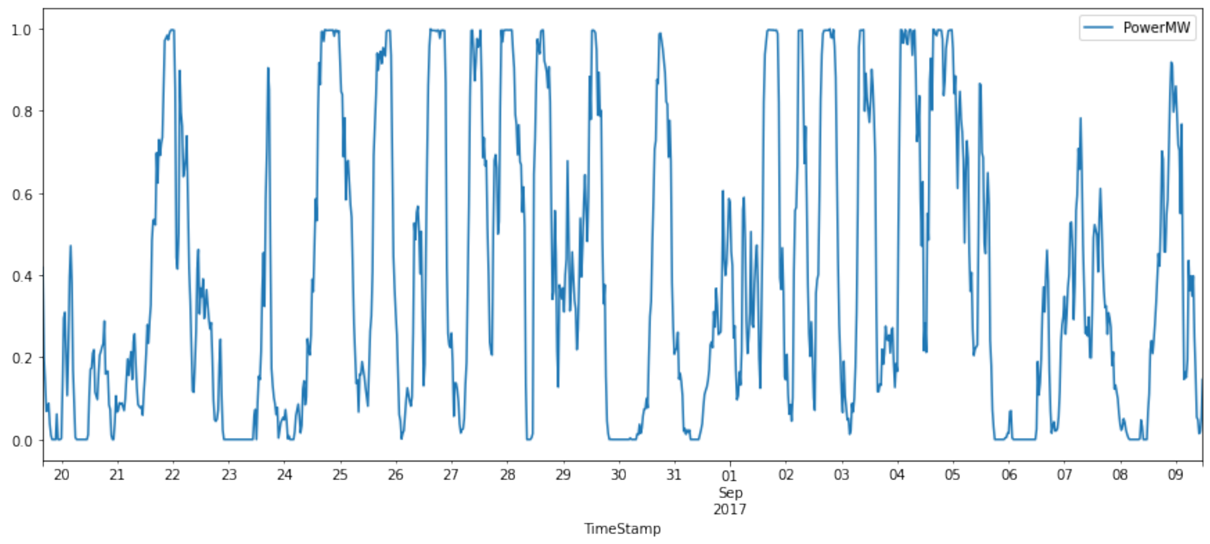


Figure 3.4: Normalised extract from the historical wind power data.

It is clear that the normalisation process does not alter the salient features of the data. For instances of Pandas Dataframe objects with multiple features, this process works similarly for each feature independently. Each feature will thus be normalised to a scheme of [0; 1] without losing salient features of the attributes.

3.3.4. Principle Methods and Objects used for Data Preprocessing

Table 3.3 below indicates the principle methods and objects deployed in preprocessing the data.

Table 3.3: Principle methods and objects used for data preprocessing.

Function	Usage
sklearn.preprocessing.MinMaxScaler()	Instantiates scikitlearn's MinMaxScaler() object.
sklearn.preprocessing.MinMaxScaler.fit()	Fits the MinMaxScaler to training data.
sklearn.preprocessing.MinMaxScaler.transform()	Used to overwrite normalised training, validation and testing data to non-normalised data.
sklearn.preprocessing.MinMaxScaler.inverse_transform()	Reconstructs non-normalised data from normalised.
emd.sift.sift()	Implements standard EMD sifting mask.
pandas.dataframe.to_numpy()	Converts pandas DataFrame to NumPy array.

3.4. Data Windowing and Generation of Datasets

The time-series data must be windowed and separated into feature-label pairs contained in training, validation, and testing datasets in order to properly apply machine learning based forecasting to these datasets. It is also necessary to feed these datasets into the neural networks in batches, as to not overload the memory of the computer. This functionality is implemented by defining a custom window generator class which leverages Python's object orientated nature.

Note that this process is summarised in Figure C.3 in Appendix C.

3.4.1. Data Windowing Process (High Level)

A flexible windowing scheme needs to be implemented for this project as models are produced to forecast various time intervals ranging from 30 minutes to 24 hours (1 label to 48 labels) into the future. Various lengths of historical windowing (inputs) will also be explored for these models, where applicable.

Furthermore, it was decided to integrate an offset for the the windowing class as this would provide for extra functionality and versatility. The offset would allow to predict for example two specific hours in a following day (this functionality was not eventually utilised in this project, but it could help with future work).

Figure 3.5, Figure 3.6 and Figure 3.7 illustrates the versatility needed in the data windowing implementation of this project.

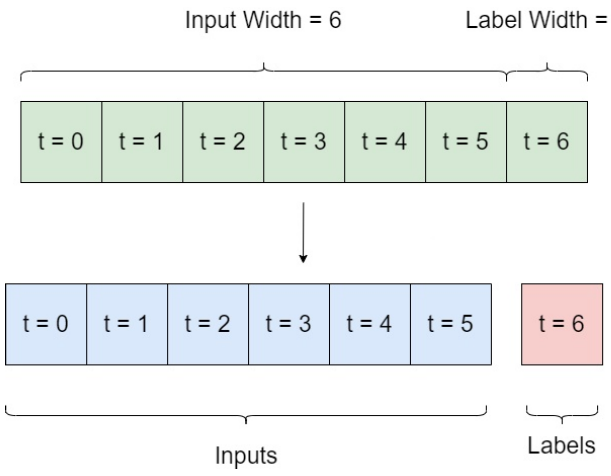


Figure 3.5: Data windowing with 6 inputs, 1 label, and an offset of 0.

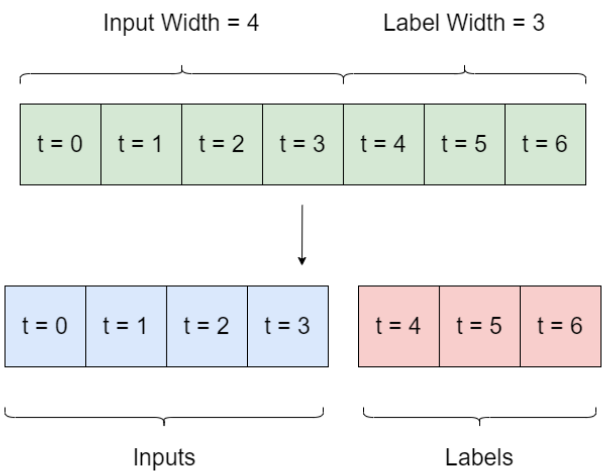


Figure 3.6: Data windowing with 4 inputs, 3 labels, and an offset of 0.

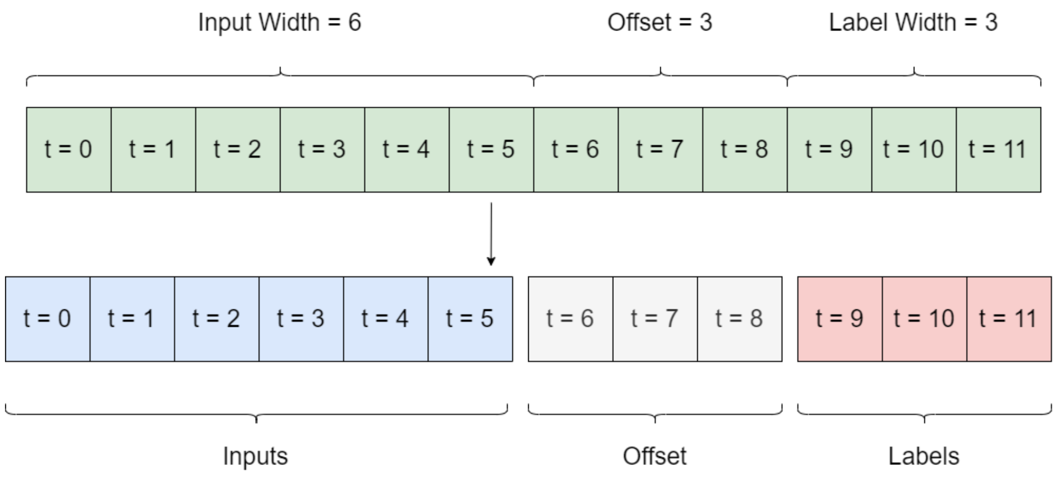


Figure 3.7: Data windowing with 6 inputs, 3 labels, and an offset of 3.

Note that the previous illustrations only illustrates the windowing process for one feature of the dataset. Where more than one feature is windowed, this process extends to all features.

A dataset then needs to be constructed from the windowed time-series in a sliding window fashion. This dataset then needs to be divided into batches. Figure 3.8 illustrates this process for two batches. The batches are generated until all the data is windowed and batched for each dataframe. These batches can then be fed sequentially into the ANN model.

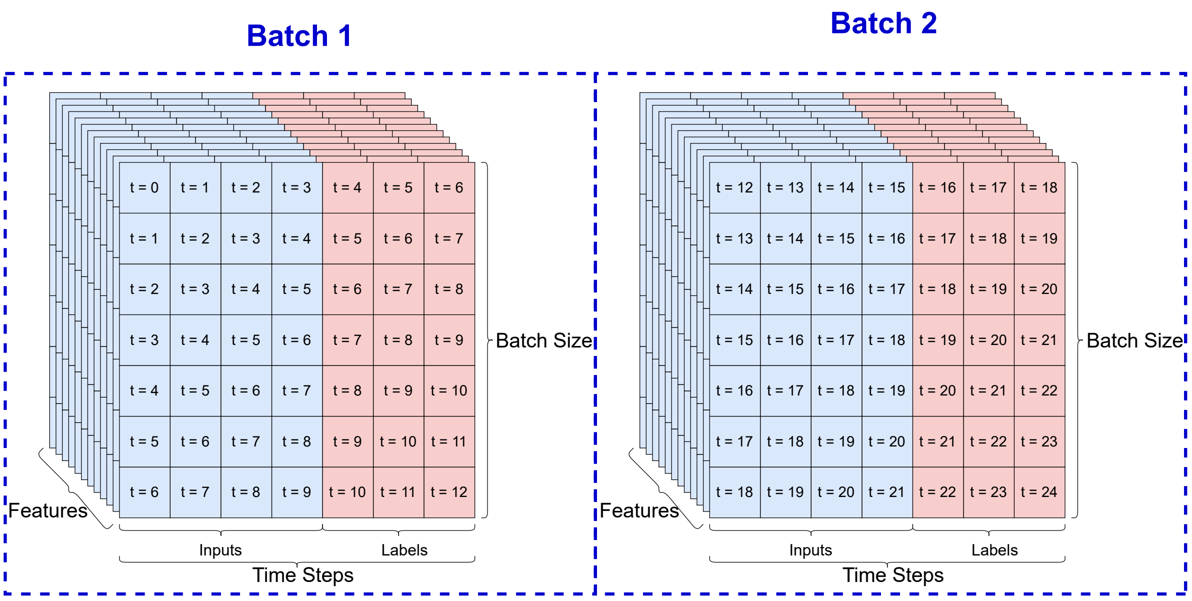


Figure 3.8: Batched and windowed data, illustrating application of WindowGenerator() class.

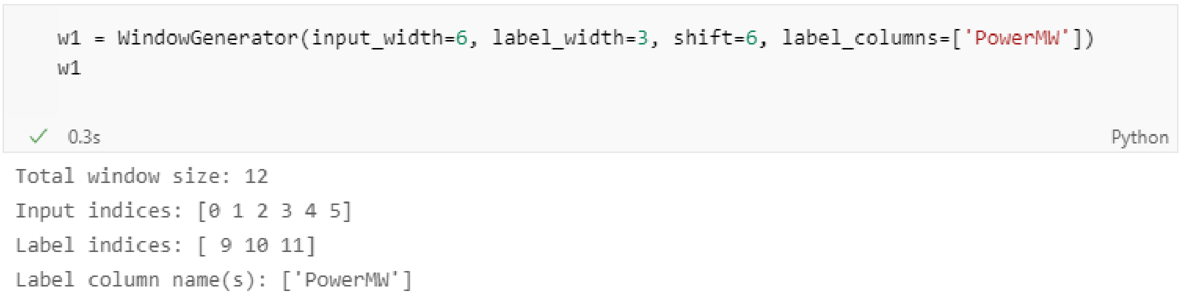
3.4.2. Feature and Label Indices

A custom `WindowGenerator()` class is defined. This class accepts the following class parameters:

- `self` - Allows access to the `__init__` method within the class.
- `input_width` - The input width used for index calculations.
- `label_width` - The label width used for index calculations.
- `offset` - The offset value used for index calculations.
- `train_df = train_df` - The training dataset after all applicable preprocessing steps.
- `val_df = val_df` - The validation dataset after all applicable preprocessing steps.
- `test_df = test_df` - The testing dataset after all applicable preprocessing steps.
- `label_columns` - The feature from which to construct the labels of the windowed dataset.

Within this class' `__init__` method, which is automatically called when an object of this class created, the datasets are migrated to this class. This is done with code similar to `self.train_df = train_df` within the `__init__` method. The rest of the `__init__` method deals with the calculation of the indexes and parameters to slice the time-series dataset. This is facilitated by passing the class parameter to the instance of the class with code like `self.input_width = input_width`, `self.label_width = label_width` and `self.offset = offset` within the `__init__` method.

The `WindowGenerator()` class also has a `__repr__` method setup to return a string representation of the current window object. The behaviour of this function can be seen in Figure 3.9. Figure 3.9 represents the windowing case outlined in Figure 3.7.



```
w1 = WindowGenerator(input_width=6, label_width=3, shift=6, label_columns=['PowerMW'])
w1
```

✓ 0.3s Python

```
Total window size: 12
Input indices: [0 1 2 3 4 5]
Label indices: [ 9 10 11]
Label column name(s): ['PowerMW']
```

Figure 3.9: __repr__ method in action.

3.4.3. Slicing the Window and Generating Datasets

Two methods are defined for this stage of generating datasets to feed into the ANN models. These are `split_window()` and `generate_dataset()`.

The `split_window()` method has the goal of generating a data container (tensor) to hold feature-label pairs as is specified for the class. This method does this by leveraging broadcasting in Numpy. The method creates input and label tensors which are in 3 dimensions to accommodate for the batch, the time interval and the features in the dataset (this dimension is specified by the neural network layers utilised). This method initially generates empty tensors which will be populated by the `generate_dataset()` method. The `split_window()` method is extended to the `WindowGenerator()` class by running `WindowGenerator.split_window = split_window`.

The `generate_dataset()` method generates the sliding window for the different dataframes and batches the data. This method calls the `tf.keras.preprocessing.timeseries_dataset_from_array()` method with the specified arguments of `stride_sequence = 1` and `batch_size = 32`. After execution of this method, the data is windowed and batched in a `tf.data.Dataset` object. This object is then mapped to the data container produced by `split_window()`. Figure 3.8 represents an object with a similar structure. The `generate_dataset()` method is extended to the `WindowGenerator()` by running `WindowGenerator.generate_dataset = generate_dataset`.

3.5. Constructing, Training, and Optimizing ANNs

Training and optimizing the forecasting models which rely on ANN structures was mainly facilitated by the Keras library which is distributed with Tensorflow. This library was excellent to speed up the prototyping phase of this project.

Note that the principle methods applicable to the ANN training, optimizing and performance evaluation are given in Table 3.4.

Figure C.4 in Appendix C summarises the processed followed to construct, optimize and train ANN models.

3.5.1. Historical Data Windowing

A series of experiments were run with a variety of models to ascertain the ideal amount of input timesteps for the windowing scheme. It was determined that most models decay rapidly once the ratio of the inputs to labels approaches 1. Consequently, the length of the historical windowing scheme was chosen to be at least double the required forecasted labels for a given model. For example, Figure 3.10 illustrates this decay in performance for the 10 hour (20 label) ANN model specified in Section 4.2.

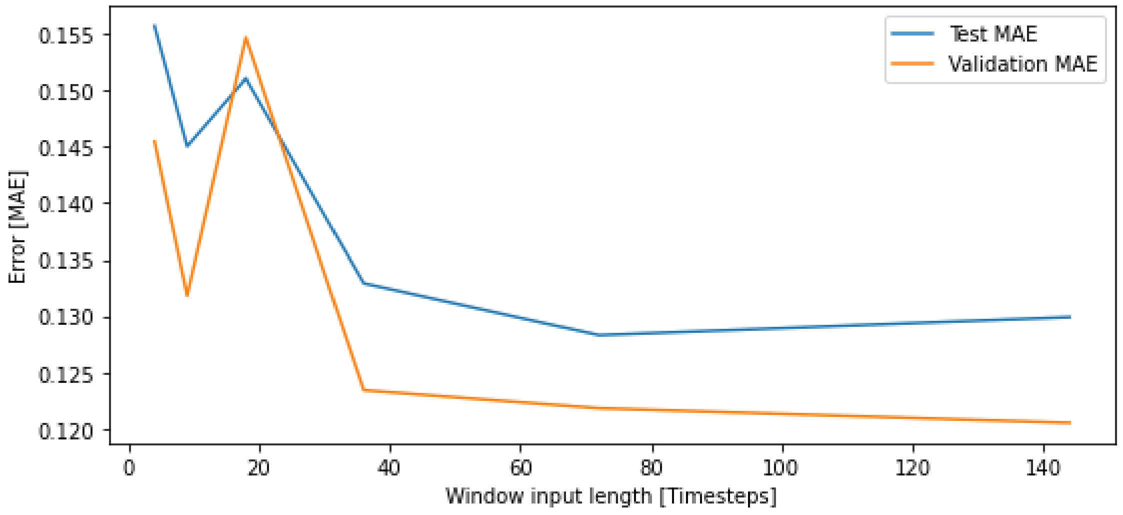


Figure 3.10: Model decay with historical windowing

3.5.2. Compile and Fit Function

Compilation, training, and callbacks related to the ANNs are packaged into a custom function called `compile_and_fit()`. This function takes as parameters a Keras sequential model, a windowed and batched dataset produced as specified in Section 3.4, and the required patience parameter for the callback function.

This function firstly sets up a Keras early stopping callback with `tf.keras.EarlyStopping()`. The early stopping callback monitors the validation loss metric of the ANN model throughout each epoch of training and stops the training process when this metric decays for more than two consecutive epochs (or any other specified `patience`). The callback then reverts the weights of the model back to the weights obtained from the epoch with the best performance. This is done by specifying `restore_best_weights=True` as a parameter. The idea is that this callback makes it very difficult to overfit the model while also returning the best model for the given training cycle and model topology.

The function then sets up the compiler to be used for the training cycle. This is facilitated by running `model.compile()` and defining the model to be a Keras sequential model. The compiler is setup to monitor mean squared error (MSE) as a loss metric with the optimizer set to `tf.optimizers.Adam()` with the optimized learning rate of the model passed to this function with the `learning_rate` parameter. Furthermore, the mean absolute error (MAE) is specified as one of the performance metrics to monitor.

Finally, the function defines the actions of `model.fit()`. This feeds into the method the generated training and validation batches for the given preprocessed dataset. A maximum amount of training epochs and the Keras callbacks to be utilised in the training cycle are also specified in this method.

When a training cycle needs to be initiated, the `compile_and_fit()` function can simply be called with the Keras sequential model and a windowed and batched dataset as parameters.

3.5.3. Defining the Keras Sequential Model

The ANN topology is constructed by instantiating a Keras sequential model with `tf.models.keras.Sequential()`. Various ANN layers can then be added to the sequential model by either leverageing the `model.add()` method or passing the sequential model topology to the model in a list form. This project utilises a variety of the layers available in Keras including `tf.keras.layers.LSTM()`, `tf.keras.layers.Dense()`, and `tf.keras.layers.GRU()`.

3.5.4. Optimizing ANN Hyperparameters

Optimization of ANN model hyperparameters is enabled by the Keras Tuner library.

A `model_builder(hp)` function is defined which takes in its structure the Keras Sequential model to optimise. Within this model, the search space for hyperparameters are defined by code like `hp.Int('units', min_value=5, max_value=150, step=5)`. This code can be assigned to a variable and then passed into the model structure. Various model and compiler hyperparameter spaces can be defined in a similar fashion.

After the search space is defined, `keras_tuner.Hyperband()` is assigned to a variable called `tuner`. This defines the type of search to do on the search space.

Finally, `tuner.search()` is called. This starts the hyperparameter search and returns the optimal model hyperparameters.

3.5.5. Monitoring Performance Metrics

Once the optimized model hyperparameters are plugged back into the Keras sequential model and this model is trained, the performance metrics of the ANN can be accessed by calling `tf.keras.models.evaluate()` and passing either the validation set or the testing set to this method. This method returns a dictionary containing the loss and mean absolute error for the desired set.

3.5.6. Principle Methods and Objects used for ANNs

Table 3.4 summarizes the principle methods and objects used in various aspects concerning ANN models as is described in this section.

Table 3.4: Principle methods and objects used for ANN construction, optimization and training.

Function	Usage
tf.keras.models.Sequential()	Initialises the ANN model.
tf.keras.models.add()	Adds the specified layer to the ANN model.
tf.keras.layers.LSTM()	Initialises the LSTM layer.
tf.keras.layers.Dense()	Initialises the Dense layer.
tf.keras.layers.GRU()	Initialises the GRU layer.
tf.keras.models.EarlyStopping()	Sets up stop conditions for the ANN training cycle.
tf.keras.models.compile()	Sets up the compiler in the ANN training cycle.
tf.keras.models.fit()	Initiates the specified training process.
keras_tuner.HyperParameter.Int()	Adds the specified integer range to the Keras Tuner search space.
keras_tuner.HyperParameters.Choice()	Adds a choice of values to the Keras Tuner search space.
keras_tuner.HyperBand()	Specifies that a Hyperband search must take place.
keras_tuner.search()	Starts the specified search.
tf.keras.models.evaluate()	Evaluates the performance metrics for the specified dataset.

Chapter 4

Implementation and Case Study Results

This section details the implementation and results obtained for various models forecasting wind power output from 30 mins (1 label) to 24 hours (48 labels) into the future. This section aims to compare various ANN-based forecasting models with each other.

Where ANN model topologies are concerned, the key in Figure 4.1 is applicable:

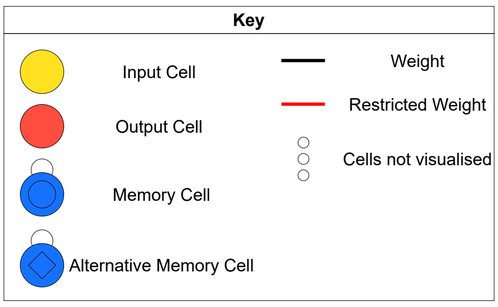


Figure 4.1: Key to disseminate neuron cells

Note that Figure 4.6 in Section 4.6 coalesces the performance for different models into one graph.

4.1. Basic LSTM Model without EMD Hybridisation

The first ANN-based model explored was a single LSTM layer interconnected with a Dense layer set up to perform a regression output. This model utilises only the historical power output data as a feature from the dataset.

This model can be visualised by Figure 4.2 below. Note that the amount of cells in the layer corresponds with the caption above the specific layer.

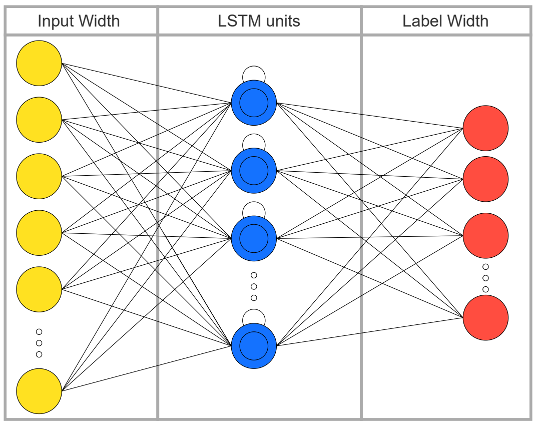


Figure 4.2: Basic LSTM model topology

4.1.1. Model Optimization and Performance for Different Forecasting Intervals

The optimization cycle was run as is described in Section 3.5.4. For this model, the amount of LSTM units and the optimizer's learning rate was optimized.

Table 4.1 below summarizes the results obtained for the different forecasting horizons. This table details the specifics of the hyperparameter optimization as well as the validation and testing MAE for the individual forecasting horizons.

Table 4.1: Results of model optimization on basic LSTM model.

Model	LSTM Units	Optimizer learning rate	Validation MAE	Testing MAE
30 min	85	0.01	0.055487	0.061134
1 hour	95	0.01	0.076908	0.086439
2 hour	40	0.01	0.107128	0.114277
5 hour	95	0.01	0.162327	0.164659
10 hour	55	0.01	0.209001	0.214024
24 hour	105	0.01	0.241469	0.238214

Figure D.1 illustrates this model's performance for different forecasting horizons. Note the similarity between the testing and validation MAE for the different models. This indicates that there is not information leakage or lookahead in the training process and it also shows that the model is not overfitted.

4.2. Basic LSTM Model with EMD Hybridisation

This model utilises the same ANN topology illustrated in Figure 4.2. For this model, the historical power output in conjunction with the first 7 IMFs obtained from the historical power output's EMD is fed into the model as features in the input to forecast power for different time horizons. The model's hyperparamters are then optimized in a similar fashion to the previous model.

The results obtained from this augmentation of the input datsets are summarized in Table 4.2 below.

Table 4.2: Results of model optimization on EMD hybridised LSTM model

Model	LSTM Units	Optimizer learning rate	Validation MAE	Testing MAE
30 min	30	0.01	0.042552	0.048995
1 hour	50	0.01	0.050056	0.054439
2 hour	90	0.01	0.059603	0.065658
5 hour	50	0.01	0.082371	0.092656
10 hour	60	0.01	0.124715	0.138033
24 hour	60	0.01	0.166851	0.177972

Clearly, hybridising the basic LSTM model with EMD does have an appreciable positive effect on the performance of the model.

Figure D.2 indicates the performance of this model.

4.3. LSTM Model with EMD Hybridisation and a Restricted Output

The EMD hybridised LSTM-based ANN model performed appreciably better than the basic LSTM model. This model was however not designed with the wind turbine power curve in mind and consequently, the model can overshoot the prediction ceiling. It was decided to rectify this by adding a relu activation function to the output dense layer and restricting the activation function to a maximum of 1. This model is illustrated by Figure 4.3.

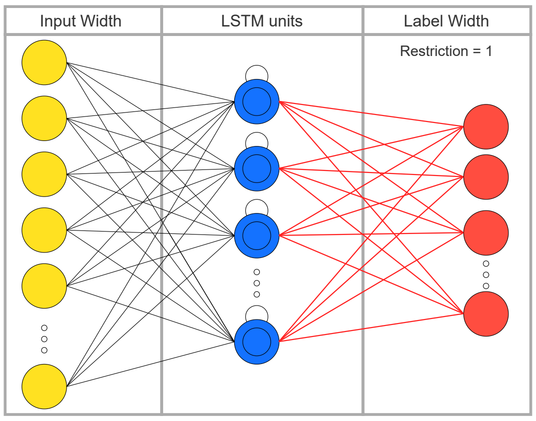


Figure 4.3: Restricted EMD hybridised LSTM-based ANN topology.

The results obtained from the optimization as well as the performance of this model is summarized in Table 4.3

Table 4.3: Results of model optimization on Restricted EMD hybridised LSTM model.

Model	LSTM Units	Optimizer learning rate	Validation MAE	Testing MAE
30 min	25	0.01	0.040251	0.043833
1 hour	140	0.001	0.043814	0.048730
2 hour	50	0.01	0.054261	0.060712
5 hour	55	0.01	0.080256	0.087842
10 hour	15	0.01	0.130043	0.134616
24 hour	40	0.01	0.157992	0.168053

Figure D.3 indicates the results obtained from this model.

4.4. GRU Model with EMD Hybridisation

It was decided to explore a Gated Recurrent Unit (GRU)-based model as some of the literature indicated that this model might perform better for non-language sequence data. The GRU is a simplified version of the LSTM unit which is tailored for more efficient computing. Section 2.3.3 can be consulted for more background on the GRU cell.

Figure 4.4 indicates the structure of this model. Note that the model is almost identical to the model indicated in Figure 4.2. The only difference between the two models is the memory mechanisms which are deployed.

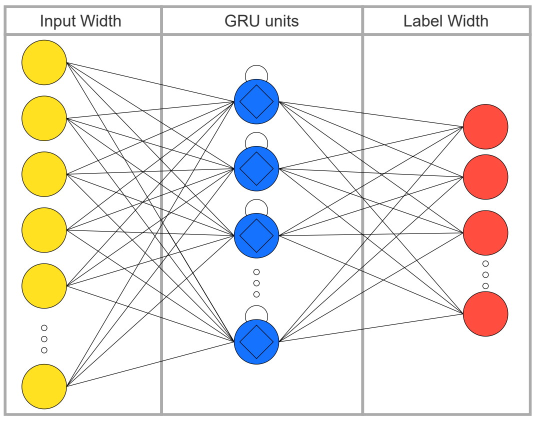


Figure 4.4: EMD hybridised GRU-based model topology

Table 4.4 indicates the results obtained from optimizing the hyperparamters of this model as well as the performance of the model.

Table 4.4: Results of model optimization on EMD hybridised GRU model.

Model	LSTM Units	Optimizer learning rate	Validation MAE	Testing MAE
30 min	110	0.001	0.040790	0.044393
1 hour	110	0.001	0.046235	0.049720
2 hour	70	0.001	0.057454	0.063840
5 hour	95	0.001	0.088476	0.096069
10 hour	105	0.001	0.115942	0.127222
24 hour	115	0.001	0.160847	0.174511

Figure D.4 indicates the results obtained from this model.

4.5. GRU Model with EMD Hybridisation and a Restricted Output

In a similar fashion to Section 4.3, it was decided to restrict the output of the EMD hybridised GRU-based ANN model. The structure of this model is illustrated below in Figure 4.5.

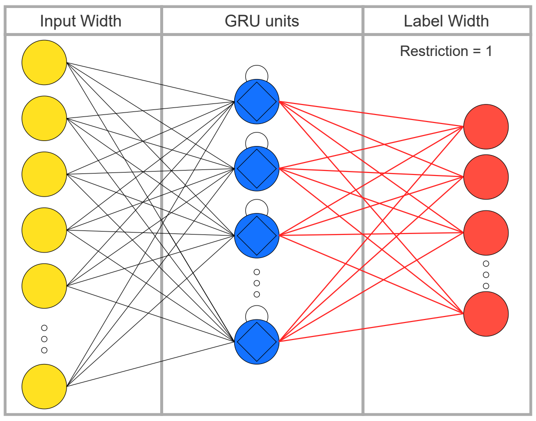


Figure 4.5: Restricted EMD hybridised GRU-based ANN topology

Table 4.5 indicates the results obtained from optimizing the hyperparamters of this model as well as the performance of the model.

Table 4.5: Results of model optimization on Restricted EMD hybridised GRU model.

Model	LSTM Units	Optimizer learning rate	Validation MAE	Testing MAE
30 min	25	0.001	0.041376	0.044850
1 hour	25	0.001	0.046026	0.052333
2 hour	145	0.001	0.054469	0.059455
5 hour	40	0.001	0.078212	0.085923
10 hour	20	0.001	0.118681	0.129681
24 hour	25	0.001	0.157903	0.167178

Figure D.5 indicates the results obtained from this model.

4.6. Chapter Summary

This chapter detailed the results obtained from forecasting a wide range of time horizons with various ANN-based forecasting models of varying topology and input features. Figure 4.6 illustrates the results obtained for the different models.

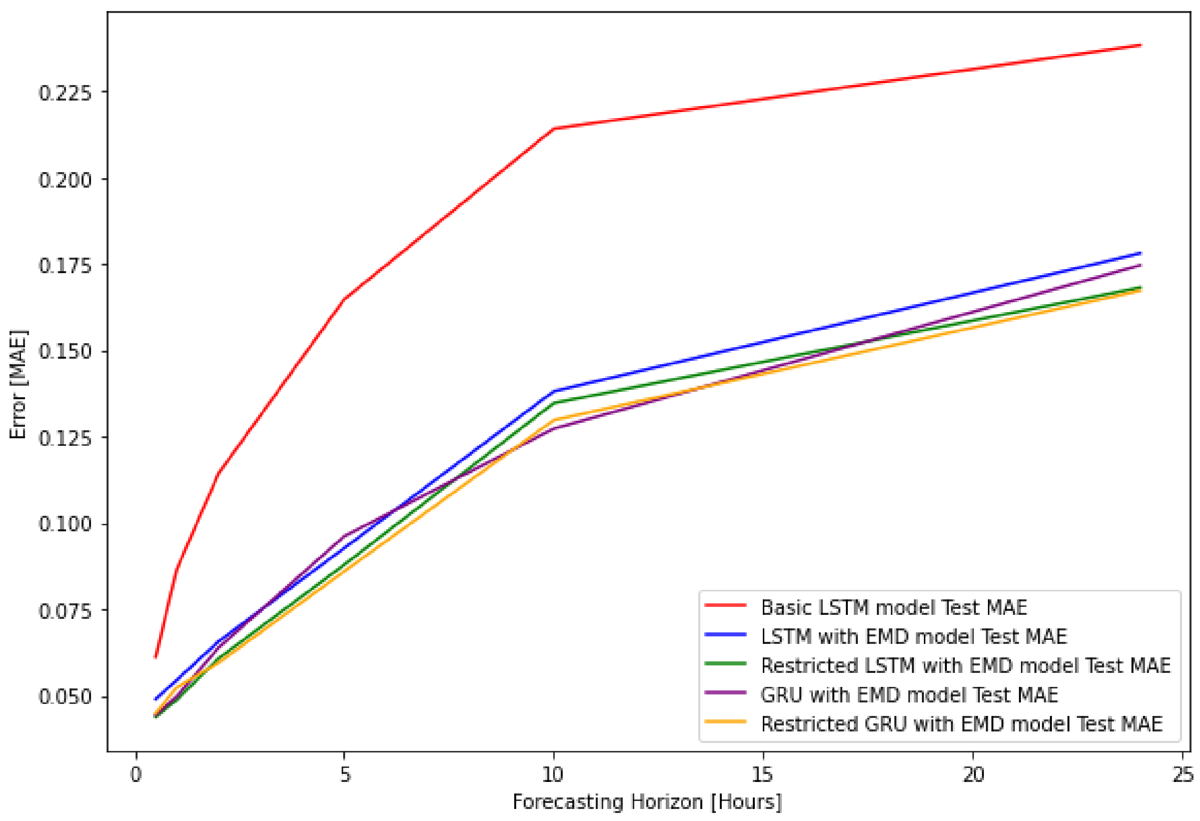


Figure 4.6: Test MAE of all developed ANN-based forecasting models for different time horizons

Chapter 5

Project Conclusion

This chapter aims to provide discussion on the results obtained in Chapter 4, to discuss some shortcomings of this project and to recommend future work.

5.1. Discussion of Results

The results outlined in Chapter 4 adequately proved that recurrent neural network (RNN) based artificial neural networks (ANN) can be applied to wind power forecasting problems. This chapter also illustrated the effect of hybridising RNN models with empirical mode decomposition (EMD). This chapter exposed the effect of modelling the wind turbine power curve with these ANNs.

Mean absolute error was chosen as an error metric for this project. To fully appreciate the results it should be noted that the data was normalised to a scheme of [0;1] in this project.

5.1.1. Modelling Wind Power Output with Basic LSTM-based RNNs

Basic LSTM-based RNNs can be used to model wind power output, but the results obtained from only a basic configuration will not perform as well as for hybridised approaches. The testing MAE for these models varied between 0.061134 to 0.238214 for time horizons ranging from 30 mins to 24 hours. The model's testing MAE showed an inverse logarithmic decay with extension of the forecasting horizon.

5.1.2. The Effect of EMD Hybridisation on Wind Power Output Modelling in RNNs

For the hybridised LSTM-based models, hybrising the features with EMD proved to provide good improvement in the performance metrics of the model. This process yielded testing MAE values of between 0.048995 to 0.177972 for forecasting horizons ranging from 30 mins to 24 hours. The testing MAE also showed inverse logarithmic behaviour with extension of the forecasting horizon.

The hybridised GRU-based models performed marginally better than LSTM-based hybridised models with testing MAEs ranging from 0.044393 to 0.174511. These models also showed similar decay in testing MAE with extension of the forecasting horizon.

5.1.3. The Effect of Capturing the Wind Power Curve into RNN Topologies

Capturing the wind turbine power curve was explored for both EMD hybridised GRU-based and EMD hybridised LSTM-based models. The positive effect of altering the model topologies in this way was marginal with both models showing very slight improvements on their predecessors. This shows that the RNNs sometimes tend to overshoot the forecasting ceiling but that it does not happen to an excessive degree.

5.1.4. LSTM-based Models vs. GRU-based Models

The GRU-based models typically performed better than the LSTM-based models. It might be that the memory mechanism in the GRU-based model is more suitable to forecast the stochastic nature of wind power output.

5.1.5. General Comments on Wind Power Forecasting

Wind power forecasting in general is a daunting activity as the power output from wind farms are highly stochastic. This project proved that wind power forecasting is possible by applying highly complex machine learning techniques. While the models are far from perfect, the fact that these models become more accurate with closer forecasting horizons do give them commercial appeal, especially when considering the 30 mins to 5 hour forecasting horizon.

In general this project has produced models which could viably address issues regarding short-term dispatching of thermal power generation to protect the grid's integrity as well as medium-term energy storage.

5.2. Shortcoming of the Project

This section will discuss some shortcomings of the project outlined in the report.

5.2.1. Historical Windowing

The historical windowing scheme was qualitatively determined instead of applying rigorous quantitative analysis. This was due to the inherent nature of the algorithm developed. This algorithm specified the historical windowing scheme within a custom class. The optimisation procedure was unfortunately incompatible with defining algorithmic hyperparameters specified in this way. This problem could be solved by approaching the algorithm in a more functional programming based manner.

5.2.2. The Application of Dropout

It is acknowledged that recurrent dropout and even dropout within RNN structures are available to improve performance metrics of the models. This was however not explored and the reasons for this are simple. Firstly, this activity would be prohibitively computationally expensive. And secondly, these mechanisms were developed with natural language processing applications in mind and consequently, the positive effects of these mechanisms on this project would be debatable.

5.2.3. Computational Cost associated with Optimising RNNs

The optimisation of RNN models were extremely computationally expensive with most models for a specific forecasting horizon taking in excess of 4.5 hours to optimise with a computer processing unit (CPU) even when using faster optimizer search algorithms. Unfortunately, a graphics processing unit (GPU) was not available to leverage for training and cloud-based solutions were unfeasible as the data utilised in this project is highly proprietary.

5.3. Recommendations for Future Work

There is wide range of future work which can build on this project varying from iterations on this project to fundamental shifts in the approach to the problem.

5.3.1. Iterations on this Project

- Ensemble methods could be explored for the range of models oulined in this project.
- The project can be attempted with more adequate computational hardware.
- A wider array of model hyperparameters like different optimizers and historical windowing can be explored.

- Feature engineering can be applied to the data or a more complete dataset with features like labelling high wind speed shutdown can be sought.
- A more rigorous analysis on the application of EMD IMFs can be done. This review can also feature recommendations from feature engineering activities.
- The effect of recurrent dropout and dropout within RNN models can be expanded upon.

5.3.2. Fundamental Shifts in Problem Solving Approach

- Autoregressive RNN models can be explored for this problem.
- The attention mechanism can be applied to this problem.

Bibliography

[1] M. Balat, "A review of modern wind turbine technology," *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, vol. 31, no. 17, pp. 1561–1572, 2009. [Online]. Available: https://doi.org/10.1080/15567030802094045

[2] U. k. Nath and R. Sen, "A comparative review on renewable energy application, difficulties and future prospect," in *2021 Innovations in Energy Management and Renewable Resources(52042)*, 2021, pp. 1–5.

[3] W. Xiaoming, X. Yuguang, G. Bo, Z. Yuanjie, and C. Fan, "Analysis of factors affecting wind farm output power," in *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, 2018, pp. 1–5.

[4] IEA, "Global energy review 2021," 2021. [Online]. Available: https://www.iea.org/reports/global-energy-review-2021

[5] J. Calitz and J. Wright, "Statistics of utility-scale power generation in south africa in 2020," Council for Scientific and Industrial Research, South Africa, Tech. Rep., 2021. [Online]. Available: http://hdl.handle.net/10204/11865

[6] S. R. Sinsel, R. L. Riemke, and V. H. Hoffmann, "Challenges and solution technologies for the integration of variable renewable energy sources—a review," *Renewable energy*, vol. 145, pp. 2271–2285, 2020.

[7] J. Cochran, M. Miller, O. Zinaman, M. Milligan, D. Arent, B. Palmintier, M. O'Malley, S. Mueller, E. Lannoye, A. Tuohy, B. Kujala, M. Sommer, H. Holttinen, J. Kiviluoma, and S. K. Soonee, "Flexibility in 21st century power systems." [Online]. Available: https://www.osti.gov/biblio/1130630

[8] A. Alabdulwahab, A. Abusorrah, X. Zhang, and M. Shahidehpour, "Coordination of interdependent natural gas and electricity infrastructures for firming the variability of wind energy in stochastic day-ahead scheduling," *IEEE Transactions on Sustainable Energy*, vol. 6, no. 2, pp. 606–615, 2015.

[9] E. T. Renani, M. F. M. Elias, and N. A. Rahim, "Wind power prediction using enhanced parametric wind power curve modeling," in *4th IET Clean Energy and Technology Conference (CEAT 2016)*, 2016, pp. 1–5.

[10] E. Dupont, R. Koppelaar, and H. Jeanmart, "Global available wind energy with physical and energy return on investment constraints," *Applied Energy*, vol. 209, 10 2017.

[11] G. Masters, *Renewable and Efficient Electric Power Systems*, 2005.

[12] Q. Zhang, M. Wangg, X. Wang, and S. Tian, "Mid-long term optimal dispatching method of power system with large-scale wind-photovoltaic-hydro power generation," in *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, 2017, pp. 1–6.

[13] L. Ma, H. Li, Y. Ge, J. Shi, G. Liu, and B. Li, "Energy storage in high penetration of renewable energy power system: Technologies, application and supporting policies," in *2019 IEEE Sustainable Power and Energy Conference (iSPEC)*, 2019, pp. 1428–1433.

[14] J. Smith, S. Beuning, H. Durrwachter, E. Ela, D. Hawkins, B. Kirby, W. Lasher, J. Lowell, K. Porter, K. Schuyler, and P. Sotkiewicz, "Impact of variable renewable energy on us electricity markets," in *IEEE PES General Meeting*, 2010, pp. 1–12.

[15] J. C. R. Filho, A. Tiwari, and C. Dwivedi, "Understanding the drivers of negative electricity price using decision tree," in *2017 Ninth Annual IEEE Green Technologies Conference (GreenTech)*, 2017, pp. 151–156.

[16] K. D. Orwig, M. L. Ahlstrom, V. Banunarayanan, J. Sharp, J. M. Wilczak, J. Freedman, S. E. Haupt, J. Cline, O. Bartholomy, H. F. Hamann, B.-M. Hodge, C. Finley, D. Nakafuji, J. L. Peterson, D. Maggio, and M. Marquis, "Recent trends in variable generation forecasting and its value to the power system," *IEEE Transactions on Sustainable Energy*, vol. 6, no. 3, pp. 924–933, 2015.

[17] P. Agarwal, P. Shukla, and K. B. Sahay, "A review on different methods of wind power forecasting," in *2018 International Electrical Engineering Congress (iEECON)*, 2018, pp. 1–4.

[18] M. S. H. Lipu, M. S. Miah, M. A. Hannan, A. Hussain, M. R. Sarker, A. Ayob, M. H. M. Saad, and M. S. Mahmud, "Artificial intelligence based hybrid forecasting approaches for wind power generation: Progress, challenges and prospects," *IEEE Access*, vol. 9, pp. 102 460–102 489, 2021.

[19] X. Wang and C. Wang, "Time series data cleaning: A survey," *IEEE Access*, vol. 8, pp. 1866–1881, 2020.

[20] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python.* Machine Learning Mastery, 2020. [Online]. Available: https://books.google.co.za/books?id=uAPuDwAAQBAJ

[21] G. Rilling, P. Flandrin, P. Goncalves *et al.*, "On empirical mode decomposition and its algorithms," in *IEEE-EURASIP workshop on nonlinear signal and image processing*, vol. 3, no. 3. Citeseer, 2003, pp. 8–11.

[22] N. Passalis and A. Tefas, "Global adaptive input normalization for short-term electric load forecasting," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 1–8.

[23] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

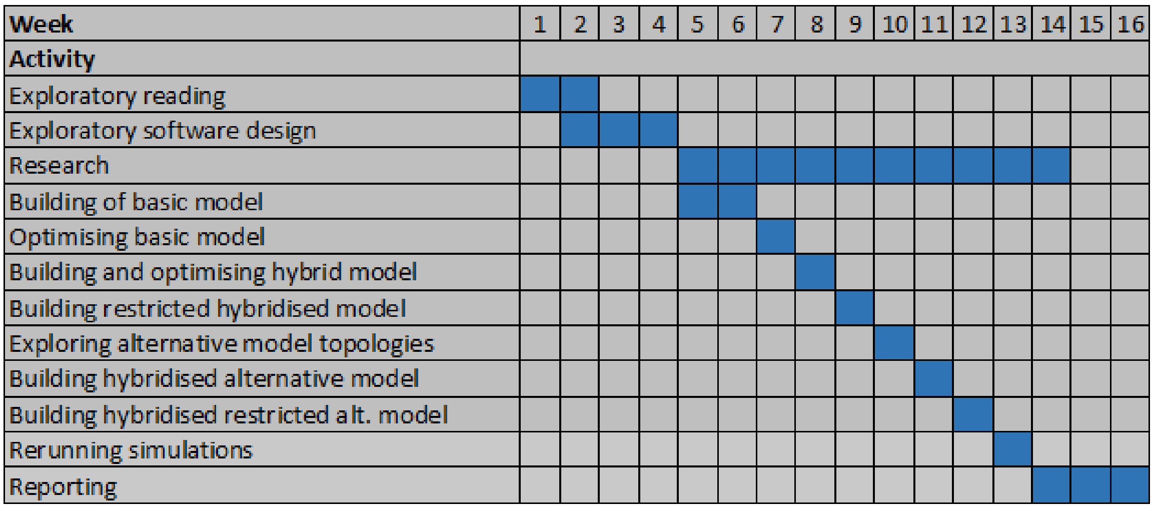
[24] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Australia: OTexts, 2018.

Appendix A

Project Planning Schedule



Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Activity																
Exploratory reading	🟦	🟦														
Exploratory software design		🟦	🟦	🟦												
Research					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
Building of basic model					🟦	🟦										
Optimising basic model							🟦									
Building and optimising hybrid model								🟦								
Building restricted hybridised model									🟦							
Exploring alternative model topologies										🟦						
Building hybridised alternative model											🟦					
Building hybridised restricted alt. model												🟦				
Rerunning simulations													🟦			
Reporting														🟦	🟦	🟦

Figure A.1: Project timeline

Appendix B

Outcomes Compliance

B.1. ELO 1: Problem Solving

Chapter 1 identifies and formulates a complex engineering problem. This is done by exposing background to the problem, and stating the motivation behind the problem solution as well as identifying deliverables to solve the problem. This section narrows down the identified problem to clearly defined goals.
Chapter 2 provides in-depth analysis on this problem. **Chapter 2** concerns a review on current solutions to the problem as well as a review on the constructs used to solve the problem in this project.
Chapter 3 and Chapter 4 solves the identified problem creatively and innovatively.
Chapter 3 details a highly complex software design utilised in the project execution.
Chapter 4 details the results obtained from applying this solution.

B.2. ELO 2: Application of Scientific and Engineering Knowledge

Throughout the project a scientific and engineering oriented approach was taken.
Chapter 2 coalesces engineering and scientific knowledge into one source. **Chapter 2** undertook a wide review of accredited literature in a variety of fields in order to solve this engineering problem.
Chapter 3 utilises engineering knowledge to solve a complex engineering problem.
Chapter 3 applied scientific and engineering knowledge from a wide variety of fields including a highly specialised field.
Chapter 4 applies scientific experimentation and data analysis. This chapter is highly experimentally based and it was focused on applying scientific technique to this section.

B.3. ELO 3: Engineering Design

Chapter 3 mainly deals with the engineering design aspect of this project. **Chapter 3** details a complex software design. This design was done over many hours of work and

required working and reworking parts of the algorithm iteratively whilst applying engineering principles.
Chapter 4 also applied engineering design to the experiments to a degree. The experiments run were highly computationally expensive and engineering knowledge was leveraged in order to successfully complete the experiments.

B.4. ELO 4: Investigations, Experiments and Data analysis

Chapter 4 is mainly concerned with investigation, experiments and data analysis. This chapter aims to compare different ANN structures in a scientific way. This was done by running various experiments as well as designing experiments to be comparable in the first place. This chapter also analyses the data from experiments in **Chapter 4** to a degree.
Chapter 5 deals with data analysis and interpretation to a degree. This chapter features interpretations of the results obtained in **Chapter 4**.

B.5. ELO 5: Engineering Methods, Skills and Tools, including Information Technology

Chapter 3 is mainly concerned with the application of engineering methods and tools. Software packages were extensively used for modelling and information handling. This chapter also deals with techniques applied to successfully run these software packages in a sustainable manner. This chapter is also highly focused on detailing discipline specific procedures and processes applied in this project.

B.6. ELO 6: Professional and Technical Communication

The entirety of this report is written with a professional and technical audience in mind. This report was written with the express goal of effectively communicating highly specialised and technical concept in a manner which is comprehensible to the audience.

B.7. ELO 8: Individual Work

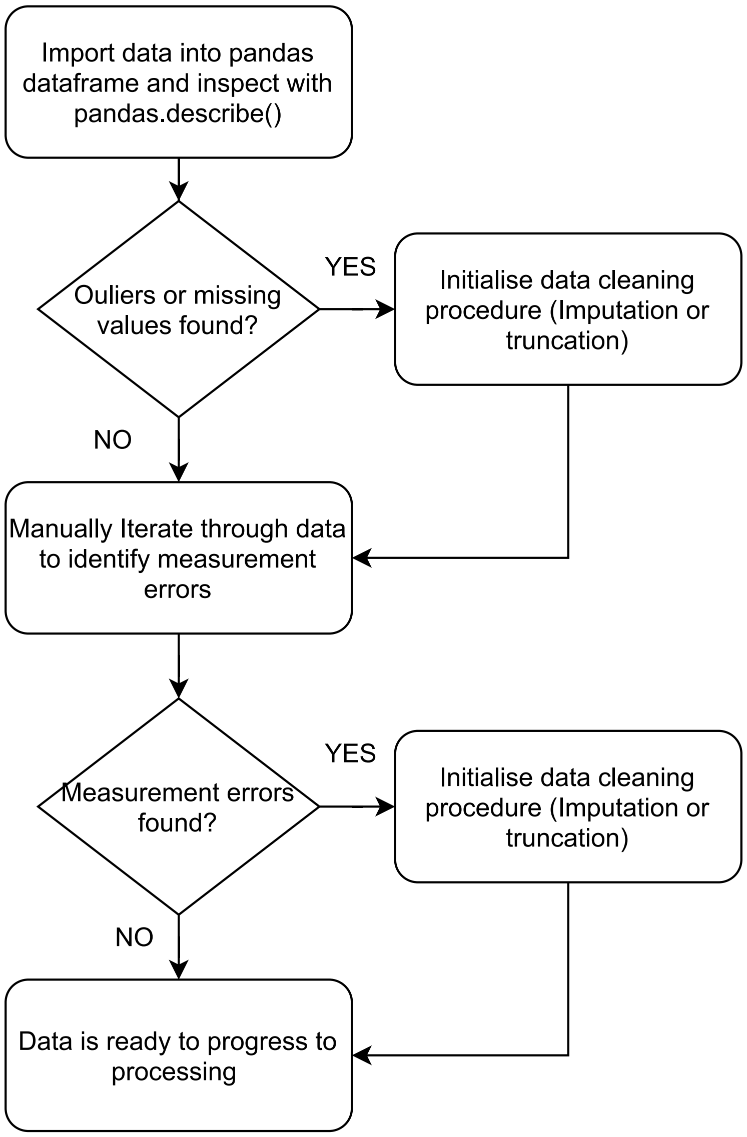
This project is the culmination of work from a single student. The outcomes of the project were achieved with minimal consultation to external parties. All activities relating to the exit level outcomes were done individually.

B.8. ELO 9: Independent learning ability

This project was a large exercise in independent unstructured learning. The solutions applied in this project was extensively researched which in turn required large amounts of independent learning. Furthermore, few of the software libraries and application thereof were familiar before the commencement of this project. This demonstrated a high degree of independent learning.

Appendix C

Flow diagrams



Import data into pandas dataframe and inspect with pandas.describe()
Ouliers or missing values found?
YES
Initialise data cleaning procedure (Imputation or truncation)
NO
Manually Iterate through data to identify measurement errors
Measurement errors found?
YES
Initialise data cleaning procedure (Imputation or truncation)
NO
Data is ready to progress to processing

Figure C.1: Flow diagram illustrating the data exploration and cleaning process.

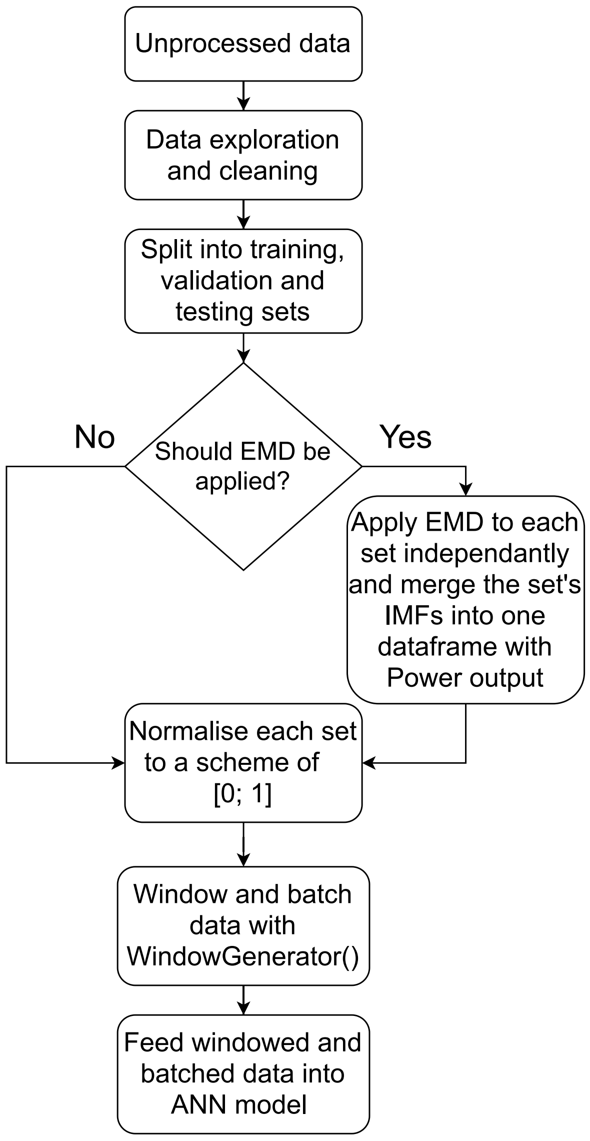

Unprocessed data
Data exploration and cleaning
Split into training, validation and testing sets
Should EMD be applied?
No
Yes
Apply EMD to each set independantly and merge the set's IMFs into one dataframe with Power output
Normalise each set to a scheme of [0; 1]
Window and batch data with WindowGenerator()
Feed windowed and batched data into ANN model

Figure C.2: Flow diagram illustrating the data pipeline for ANN-based models.

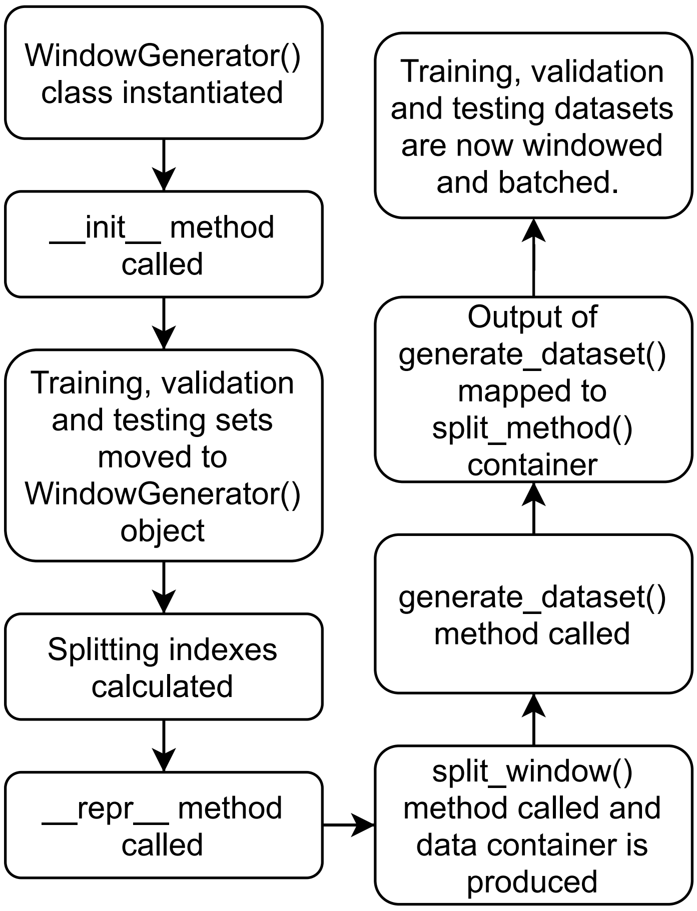

WindowGenerator() class instantiated
__init__ method called
Training, validation and testing sets moved to WindowGenerator() object
Splitting indexes calculated
__repr__ method called
split_window() method called and data container is produced
generate_dataset() method called
Output of generate_dataset() mapped to split_method() container
Training, validation and testing datasets are now windowed and batched.

Figure C.3: Flow diagram illustrating proccesses followed in the WindowGenerator() class.

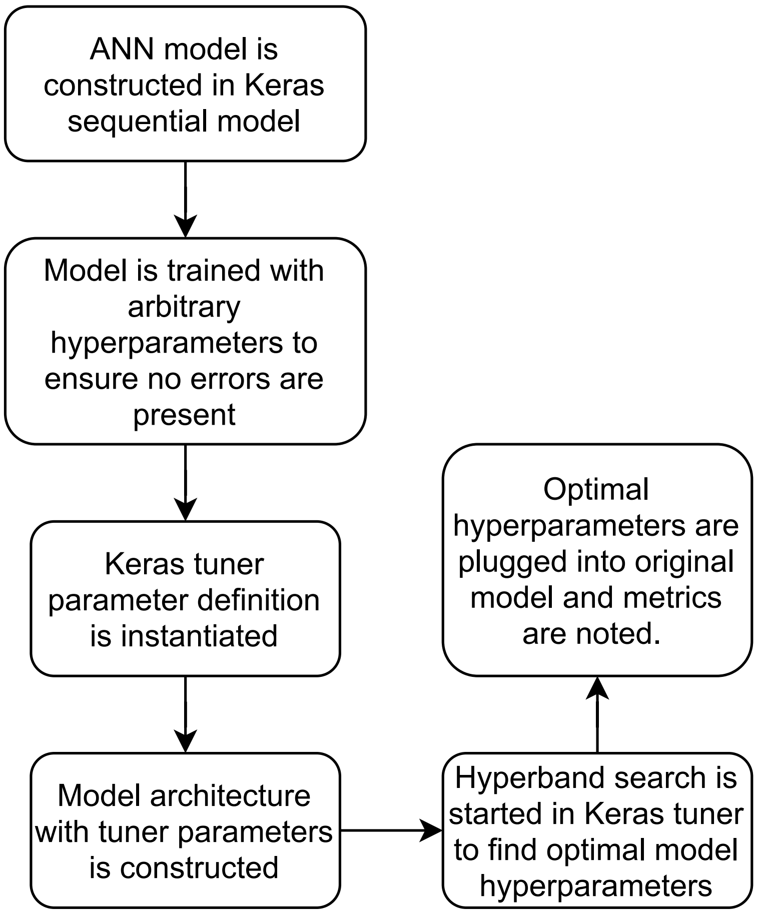

ANN model is constructed in Keras sequential model
Model is trained with arbitrary hyperparameters to ensure no errors are present
Keras tuner parameter definition is instantiated
Model architecture with tuner parameters is constructed
Hyperband search is started in Keras tuner to find optimal model hyperparameters
Optimal hyperparameters are plugged into original model and metrics are noted.

Figure C.4: Flow diagram illustrating proccesses followed to construct, optimize and train ANNs.

Appendix D

ANN-based model results

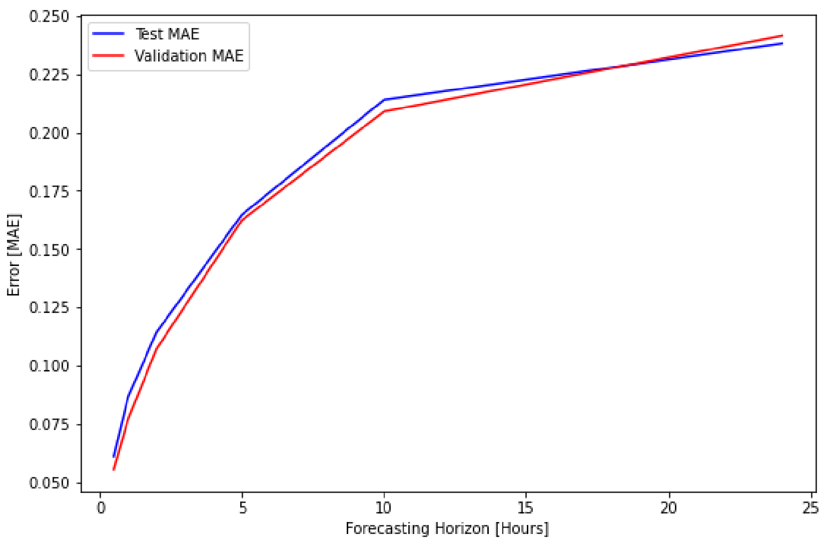


Figure D.1: Basic LSTM model MAE for validation and testing sets on different forecasting horizons.

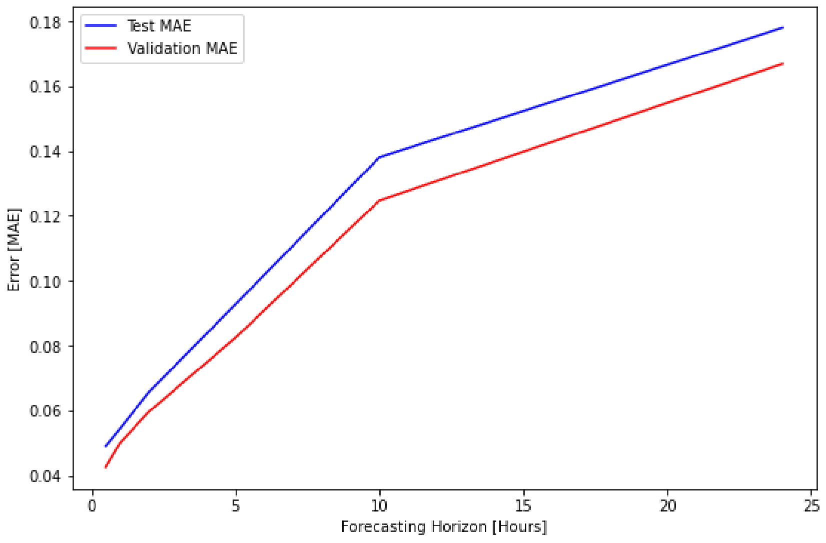


Figure D.2: EMD hybdridised LSTM model MAE for validation and testing sets on different forecasting horizons.

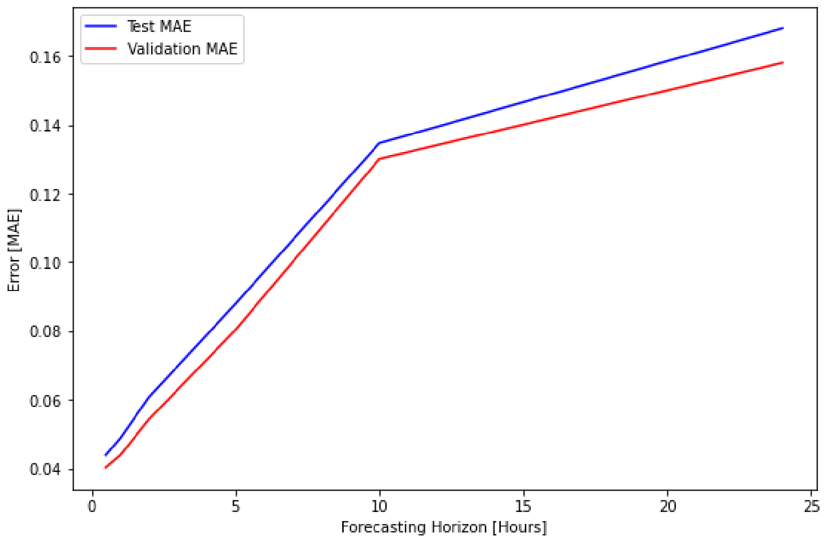


Figure D.3: Restricted EMD hybdridised LSTM model MAE for validation and testing sets on different forecasting horizons.

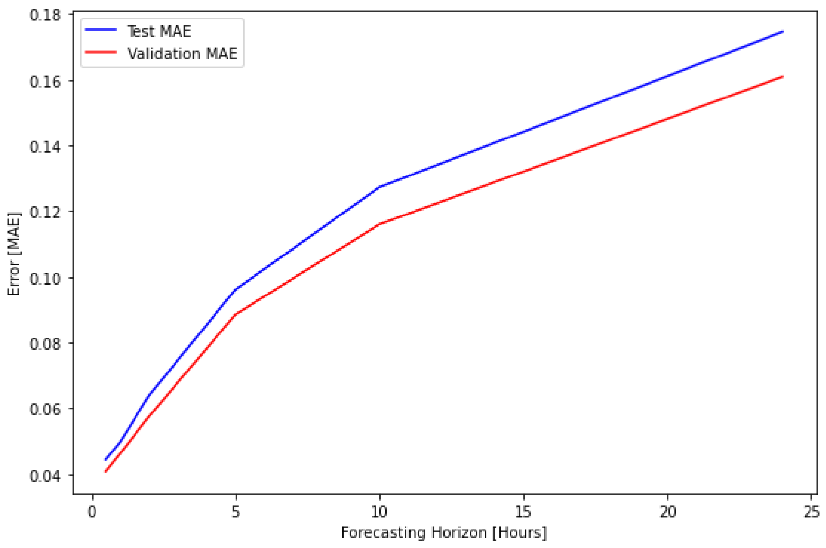


Figure D.4: EMD hybridised GRU model MAE for validation and testing sets on different forecasting horizons.

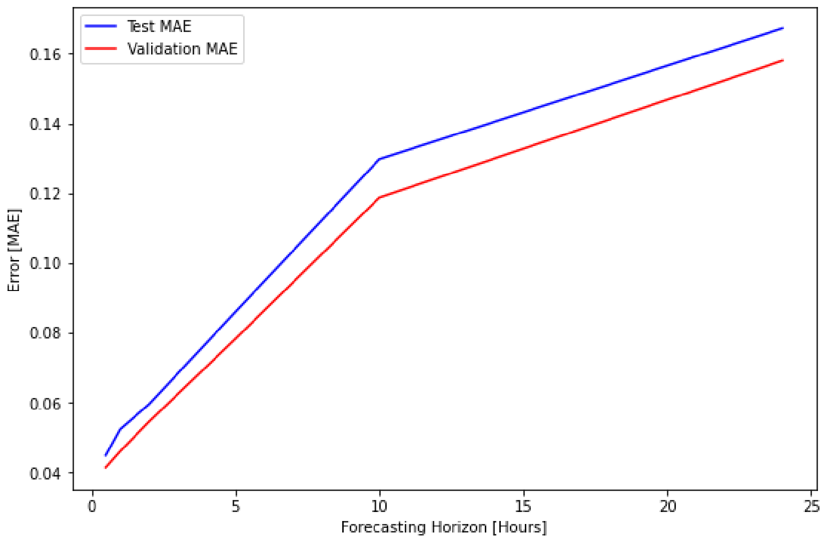


Figure D.5: Restricted EMD hybridised GRU model MAE for validation and testing sets on different forecasting horizons.