
# WIND FARM POWER OUTPUT PREDICTION BASED ON MACHINE LEARNING RECURRENT NEURAL NETWORKS

**A Thesis Submitted for the Degree of**

**Doctor of Philosophy**

**By**

**Ethelbert Chinedu Eze**

**University of Sussex**

**Department of Engineering & Informatics**

**October 2019**

**Dedicated to the Almighty God and to my Parents**

## DECLARATION

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signature

## ABSTRACT

Scientists, investors and policy makers have become aware of the importance of providing near accurate prediction of renewable energy. This is why current studies show improvements in methodologies to provide more precise energy predictions. Wind energy is tied to variabilities of weather patterns, especially wind speeds, which are irregular in climates with erratic weather conditions. To predict wind power output, model technologies like autoregressive integrated moving average (ARIMA), variants of ARIMA, hybrid models involving ARIMA and artificial neural networks (ANN), Kalman filters and support vector regressions (SVR) have been applied for wind speed involving short, ultra-short, medium and long terms kind of predictions. ARIMA ensemble with ANN has shown better performance for short and ultra-short terms of two to three hours ahead. On the other hand, SVR, Kalman filters and ensemble of both has recorded good performance for medium-term kinds of wind speed predictions. Recently, neural networks in particular recurrent neural networks (RNN) have reported immense achievement in time series predictions particularly for medium and long-term. This is largely due to its retentive memory-mapping capabilities in fitting sequence in series. These capabilities are short-lived; when the sequence grows over time, the RNN tend to lose correlated information on back-propagation operations. This can lead to errors in the predicted potentials. Therefore, RNNs are exploited for enhanced wind-farm power output prediction. The main contribution of this research is the study of a model involving a combination of RNN regularisation methods using dropout and long short-term memory (LSTM) for wind-power output predictions. In this research, the regularisation method modifies and adapts to the stochastic nature of the wind and is optimised for the wind-farm power output (WFPO) prediction for up to 12-hours ahead – 72-timesteps. This algorithm implements a dropout method to suit the non-deterministic wind speed by applying LSTM to prevent RNN from overfitting. A demonstration for accuracy using the proposed method is performed on a 14-turbine wind farm with up to ten thousand wind samples for model training and five hundred for model validation and testing. The model out performs the ARIMA model with up to 90% accuracy and is expected to be applied to erratic weather condition, especially those observed in an off-shore wind farms.

## PUBLICATIONS

Some ideas and figures in this thesis have appeared in the following publications:

I. E. C. Eze, C.R Chatwin (2019). Enhanced Recurrent Neural Network for Short-term Wind Farm Power Output Prediction. IJRDO – Journal of Applied Science. ISSN: 2455-6653.

II. E. C. Eze[*1], E.S. Ibrahim[2], C.R Chatwin[1], T. C. Yang[1] (in print 2019) Prediction of Wind Farm Power Output Based on an Enhanced Recurrent Neural Network. Springer – Journal of Energy Systems. Manuscript Number: ENSY-D-19-00020

## ACKNOWLEDGEMENT

First, I will like to thank my main supervisor, Dr Tai Yang for his guidance through the rocky waters of modern research. I would also like to thank my co-supervisor Professor Chris Chatwin for his helpful encouragement, advice and support. In addition, give great thanks to Dr Ejike Igwebuike for his generosity and assistance in the simulations.

I wish to thank all the project students I have worked with in the course of this journey, notably Xing Gao, Dr. Fatai Afolabi, Dr. Ini, Dr. Audey, Esther Shupel and Mr. Alhamdu Hamdo Bello for their support and understanding.

I would like to thank my mentors – Hon. Paul Eze, Dr Eguaroje, Prof. Paulinus Ugwoke and Prof. Seidu Mohamed for their moral supports and words of encouragements.

Special Thanks goes to Petroleum Development Trust Fund (PTDF) of Nigeria for partially funding my PhD programme. I give great thanks to the management of National centre for Remote Sensing for their invaluable support throughout the programme.

Finally, and most importantly, I would especially like to thank my wife and siblings for their love, support and continued encouragement.

# Table of Contents

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ACRONYMS

| | |
|---|---|
| AC | Alternating Current. |
| ACF | Autocorrelation Function. |
| ADAGRAD | Adaptive Gradient Decent. |
| AEP | Annual Energy Production. |
| ANFIS | Adaptive Fuzzy Inference System. |
| ANN | Artificial Neural Network. |
| ARIMA | Auto Regressive Integrated Moving Average. |
| ARMA | Auto Regressive Moving Average. |
| CDBN | Convolution Deep Belief Network. |
| CDF | Cumulative Density Function. |
| CER | Character Error Rate. |
| CGBP | Conjugate Gradient Back Propagation. |
| CNN | Convolution Neural Network. |
| DBN | Deep Belief Network. |
| DC | Direct Current. |
| DFT | Dickey-Fuller Test. |
| ECMWF | European Centre for Medium-range Weather Forecast. |
| eLSTM | enhance LSTM. |
| EMD | Empirical Mode Decomposition. |
| FFNN | Feed Forward Neural Network. |
| GFS | Global Forecast System. |
| GPRS | General Packet Radio Service. |
| GRU | Gated Recurrent Units. |
| GW | Giga watts |
| HMM | Hidden Markov Models. |
| KNN | K-Nearest Neighbourhood. |
| LMA | Levermberge-Marquardt Algorithm. |
| LMBP | Levember-Marquardt back-Propagation. |
| LPTN | Lumped-Parameter Thermal Networks. |
| LSTM | Long Short-Term Memory. |
| LVQ | Learning Vector Quantization. |
| MAE | Mean Absolute Error. |
| MAPE | Mean Absolute Percentage Error. |
| MDLSTM | Multi-Dimensional Long Short-Term Memory. |

| | |
|---|---|
| ML | Machine Learning. |
| MSE | Mean Square Error. |
| NWP | Numerical Weather Prediction. |
| OLS | Ordinary Least Squares. |
| PACF | Partial Autocorrelation Function. |
| PAR | Polynomial Autoregressive. |
| PCA | Principal Component Analysis. |
| PDF | Probability Density Function. |
| PHM | Prognostic and Health Management. |
| RMSE | Root Mean Square Error. |
| RMSprop | Root Mean square Propagation. |
| RNN | Recurrent Neural Network. |
| RNLM | Recurrent Neural Learning Model. |
| SCADA | Supervisory Control and Data Acquisition. |
| SQL | Sequential Query Language. |
| SVM | Support Vector Machines. |
| SVR | Support Vector Regression. |
| VLRBP | Variable Learning Rate Back Propagation. |
| WD | Weibull Distribution. |
| WER | Word Error Rate. |
| WF | Wind Farm. |
| WFPO | Wind Farm Power Output. |
| WP | Wind Power. |
| WPD | Wind Power Density. |

NOMENCLATURE

| | |
|---|---|
| $\Theta\phi$ | Recurrent neuron state. |
| $\Theta$ | element-wise multiplication parameter. |
| $\omega_j$ | Fraction of time the jth unit works |
| $\Gamma$ | Gamma function estimator |
| $\eta_j$ | Improvement factor in the jth unit after an individual scheduled wind |
| $\mathcal{L}_T(w)$ | Loss function associated with weight matrix. |
| $\bar{\mu}$ | Mean of Normal distribution |
| $\bar{u}(t)$ | Mean wind speed |
| $\theta_R$ | Parameter depicting weight matrix. |
| $\theta^T$ | Parameter showing derivative of $h_t$ with respect to $h_{t-1}$ |
| $\boldsymbol{\lambda}$ | Regulariser that controls trade-off between $\mathcal{L}_D(w)$ $and$ $\mathcal{L}_w(w)$ |
| $\theta_j$ | Scale parameter of the jth unit for Weibull distribution |
| $\mathcal{Y}_j$ | Shape parameter of the jth unit for Weibull distribution |
| $\sigma_T$ | Standard deviation |
| $\mathcal{L}_D(w)$ | Sum of square error between $y^{(i)}$ $and$ $h(x^{(i)})$ |
| $g$ | Activation function |
| $b_i$ | Bias term for the i$^{th}$ function |
| $C_t$ | Cell state of the RNN |
| $a_1, a_2$ | Coefficients of chance |
| $c_j$ | Constant factor for the jth unit, |
| $F(t)$ | Cumulative distribution function |
| $H_{t-1}, H_{t-2,}$ | Depiction of history of events. |
| $\mathcal{N}_{t,w,t,h}^{nod}, \mathcal{N}_s^{nod}$ | Different Turbine nodes at different heights and time (hours) |
| $p_i, q_i$ | Dropout parameters. |

| | |
|---|---|
| $Z_t$ | Gate with 0 or 1 element representation. |
| Z | height above earth surface |
| $h_t, h_{t-1}$ | Hidden states of a recurrent neural network. |
| *u'(t)* | Instantaneous speed fluctuation. |
| $D_n$ | Kolmorogrov-Smirnorf test statistic for the lognormal distribution |
| $f(x)$ | Likelihood function of the probability obtained in observed samples. |
| $i_t$, $o_t$, $f_t$ | LSTM input, output and forget gate components. |
| $M$ | Mann's test statistic for the Weibull Distribution |
| $y_{max}, y_{min}$ | Maximum and minimum value of the sample data. |
| $t_{med}$ | Median of lognormal distribution |
| $pn$ | Normalized value of a data value $p$. |
| $n$ | Number of observations (sample size) |
| $f(t)$ | Probability density function |
| $Z_0$ | reference height at which wind speed is known |
| $V_{\alpha,y}(\phi, \delta)$ | represents aerodynamic interactions between turbines. |
| $C_p$ | Rotor coefficient of efficiency or capacity factor. |
| $S$ | Shape parameter of lognormal distribution |
| a | shear coefficient. |
| $z_i$ | Standardized normal probabilities |
| β | swept rotor area ($m^2$) |
| $N$ | Total number of observed wind speed parameters. |
| $X$ | Variable representing the number of trials |
| $S^2$ | Variance |
| $w_1, w_2, w_3, w_4$ | Various weights of the parameters $y(t), g(t)$for a given network |
| *u(t)* | wind speed at any instant of time t |

# Chapter 1.

## 1.1. Introduction.

The intermittency of wind speed introduces challenges to the prediction of wind power operation during energy integration. This result in challenges associated to planning and regulation capabilities associated with sudden wind speed variations, which impacts on the reliability of power system predictions. Wind-power generation and reliability planning relies on fast and strong wind speed prediction and response to system dynamics for better wind power prediction [1]. The global energy report shows that power generation from the wind rose to 54.6 gigawatts (GW) of installed capacity in 2018. China and the USA are leading with installed capacity. Countries like Germany and India are showing a strong appetite for wind energy generation due to effective wind speed prediction capability [2].

Recent literature shows a large variety of time series forecasting methodologies introduced for effective prediction of wind-speed in a time series and sequenced format. Wind data is stochastic; it is a very complex task to forecast the velocity of wind using linear approaches [3]. In addition, the length of the forecasting horizon has a correlation with the accuracy of forecasting methods. This correlation is negative with respect to the forecasting horizon. These horizons are of the ultra-short-term, short-term, medium and long-term time scale. Ultra-short-term wind forecasting refers to wind data prediction in the range of a few minutes to one hour ahead [4]. Prediction techniques are mainly utilised during electricity market clearing, regulation actions, and real-time grid operations. The short-term prediction horizon of wind speed are for a period starting from one hour to several hours ahead. This is generally for unit commitment and operational security in the electricity market. Medium-term and long-term forecasting refers to longer time horizons [5]. Prediction of wind depends from

several atmospheric factors like direction of the wind, temperature, humidity, turbulence, wind shear, and so on. In addition, these atmospheric factors affect wind energy penetration. Thus, effective prediction of wind energy affects not only the wind energy penetration but also the real power balance – load balancing and load demand matching. In wind renewable energy however, the stochastic nature of wind speed affects the excess stored energy process and dissipation during high demand periods.

The method of predicting real power balance from wind energy are classified into four main categories in the technical literature: a) Persistence model, which has a naïve smoothness assumption on the target function. In this approach, the future wind speed is equivalent to the wind speed in the forecasting time [6]. This method is the most economical and the simplest wind forecasting approach and is therefore widely employed by electrical utilities. The drawback however is the rapid degradation of performance model on an extended forecasting time horizon; hence, it is only reliable for ultra-short-term purposes. b) Physical methods. This approach relies on numerical weather prediction (NWP), which considers other complex atmospheric parameters such as temperature, pressure and turbulence wind shear for prediction of wind speed [7, 8]. NWP outputs accurate estimations for long-term predictions mainly utilized for large-scale areas. The major drawback of numerical weather prediction models is the memory demands and high time consumption in producing results; hence, it is not reliable for short forecasting horizons. c) Statistical methods find the mathematical relationship between wind-speed time series data. Statistical models include auto regressive (AR), auto regressive integrated moving average (ARIMA), and Bayesian approaches. Reference [9] studied a simple statistical model approach for wind speed prediction using a K-nearest neighbour (KNN) regression model for short-term wind speed forecasting. This approach has high computational complexity and can suffer dimensionality problem, as the number of parameters grow exponentially with the growth in input size. It is important to note that [10]

presents a hybrid ARIMA-Kalman filter to predict wind speed. Although this model applied a statistical linear model for multi-step ahead prediction, it cannot give accurate estimations for longer time horizons due to the nonlinear assumptions in wind data patterns. Other methods seen in [11] introduces Bayesian forecasting based on a truncated model approach, which can incorporate domain knowledge about wind data. The model is applied for ultra-short-term wind speed prediction. The linear characteristics of the presented structural break method restricts the ability of this model to address more challenging prediction problems with longer forecasting time horizons. d) Artificial intelligence (AI) techniques including artificial neural networks (ANNs) [5, 12-16], support vector regression (SVR) [17, 18], and recurrent neural methods [6, 19-21], which led to novel methodologies for wind prediction. ANNs can capture the relationships between the input data and the predicted wind speed values, hence, it is utilized for time series prediction of different weather variables on various time scales and yields satisfactory predicted results. ANNs have various structural configurations. The Feed-forward ANN [6, 10], recurrent ANN [20, 21], radial basis function (RBF) ANN [14] and adaptive wavelet ANN [22] were proposed recently for wind power and wind speed prediction. RNN-based approaches have been widely applied in the time-series prediction domain due to their capability to co-adapt complex non-linear relationships between the input and output time-sequence variables. Moreover, RNN implicitly learns features in a high dimensional space applying its popular cell state strategy; hence, it suffers from vanishing gradient problems. This problem is the inability of RNN to learn long-range dependencies, the interaction between wind speed sequences at different horizons and time steps apart – the long-term horizon. To tackle this drawback, the presented model [23] extracts error prone-engineered features caused by the vanishing gradient problem to control gradient growth in a type of RNN ANN called long short-term memory (LSTM). Although LSTM [24, 25] controls these growths by mitigating vanishing gradients, LSTM suffers overfitting (perfect learning) problems especially in time

series models, hence, it requires further modifications. The approach in the literature is to use regularisation.

Regularisation is a method of controlling model complexities and numerical stability in neural network model systems [23]. To obtain regularisation in a neural network, an additive penalty term is introduced to the cost function in the form of fake noise. These fake noises can be in the form of dropout, L1, L2 and L1L2 and so on; to favour simpler models over complex ones as [26]. From the literature, during application of regularisation, L1 sums the weight coefficients while L2 sums the squared weight coefficients. Hence, [27] demonstrates the economic approximation of applying both L1L2 by combining large numbers of neural subnetworks and further comparing both results with other methods like the dropout. The author further reports that dropout is an efficient data driven regulariser with weight decay effects on outgoing weights, meaning dropout is a better model for regularisation than L1, L2 and L1L2.

In terms of network structure and issues of backpropagation, [28] reports that the structural implementation of LSTM threatens memorisation ability, which results to poor performance in comparison to network models that applied dropout on LSTM. However, [29] described dropout implementation on predicting a time-series protein sequence as having better prediction, confirming that [30] experienced shorter training time due to structural implementations associated with dropout on LSTM.

In terms of network architecture and assembling of LSTM with other regularisation methods in conjunction with the nature of time series data, in LSTM and dropout network implementation for time series predictions, the structural arrays expects input sequences to be of *samples, time steps and features*. This allows a smooth implementation of dropout regularisation on LSTM, hence, hybrid regularisation on RNN or eLSTM. Therefore,

configuring dropout and L1L2 can be either in the input or in hidden layers. In the research by [32], however, dropout did not only improve memorisation ability and reduce training time but enhanced predictive performance on a long sequence of about four hours ahead.

This research is therefore inspired [28, 31-33], where the idea of sequential modelling is introduced for time series sequences applying dropout on LSTM to forecast sequence generation over speech recognition, handwriting recognition and machine translation. Thus, the concept of leveraging mid-level RNN representation in LSTM [27] inferred in image label annotation is also exploited. Predicting energy consumption and wind power for households using LSTM as reported by [23, 34] was investigated. Reference [23] recorded no improvement on LSTM, however, effective learning of measured energy consumption profile was observed. In a typical time series scenario similar to wind power prediction as described by [29], dropout was implemented to improve LSTM to forecast the risk of a student leaving an online course platform. In view of the above, the main contribution and thesis motivation are as described in the section below.

## 1.2. Research Motivation

In this research, an integration involving combinations of regularisation methods on RNN for wind speed prediction is proposed. This new regularisation involves long short-term memory (LSTM) and a dropout regularisation (LSTM-Dropout) model for learning nonlinear temporal features from the time series wind data in order to address the stipulated issues. Our LSTM-Dropout model is proposed [28, 33] to capture interval-unsupervised features from the underlying input time series. The cell state in RNN learns the decreasing energy function while increasing the learning pattern in the observed input vectors of wind series dataset. The method suffers from the vanishing gradient problem as RNN maps input and output wind data. This mapping results in a huge influence to a given input of a hidden layer. As the wind time steps

increase relative to wind power, the network connection grows resulting in connection decay, exponential bursts and a sharp diminishing weight coefficient gradient. Conversely, due to the nature of the time-series data size, LSTM suffers overfitting of the unsupervised features, as it requires a huge training set unlike what is obtainable in time series systems. This overfitting is because of poor generalisation of the unsupervised features during model testing. In order to tune the parameter, a dropout method involving conditional probability of the visible and hidden LSTM layers results in accurate control of overfitting. These layers can easily decompose to simple factors to learn the recurrent parameters in an LSTM-Dropout fashion. The proposed deep learning research has contributed the following to knowledge:

- A new recurrent network-learning model (RNLM), presented based on hybrid regularisation of long short-term memory and dropout architectures for the robust supervised feature extraction of wind time series. The proposed RNLM is an energy-based generative method proved to capture the co-adaptation of input variables of wind speed. Moreover, the inference and learning algorithms of the devised undirected graphical model are presented. To the best of our knowledge, RNLM is the first recurrent deep learning model capable of capturing interval knowledge from wind data.
- The approach can extract meaningful features from wind speed data input in an unsupervised manner. Thus, unlike other artificial intelligence methodologies including ANNs [5, 12-16], RNN [20, 21] , and dropout regularisation systems [27, 29], which are based on the supervised regression methods, no prior knowledge about the wind data is needed for the feature extraction.
- In contrast to previous deep learning research including [35] and [25] that implement Auto-encoding and classical DBN, in this research, real-valued input units are implemented as designed for the wind domain. The classic deep approaches applied in the domain of time series

prediction assume a probability distribution on the input variables, which is not suitable for the real-world applications that work with real input tensors.

The contributions of the proposed thesis research above is sub-divided into two areas: *a) Machine Learning:* The development of an integral long short-term supervised learning system with the incorporation of the dropout tuning regularisation model to extract robust highly nonlinear features from the wind speed input data. *b) Wind farm Power output Prediction:* The application of an unsupervised feature extraction model (rather than the superficial features applied in previous methodologies), in nonlinear manifold learning from windfarm data for supervised target function of future wind values prediction.

## 1.3. Machine learning for Wind Speed Prediction.

Electricity generation relies on the curve of power production over time to show imbalance of time between peak renewable energy production and demand. In the wind energy market, peak demand occurs after wind troughs and sunset – when low or no wind is experienced. Machine learning techniques coupled with microelectronic sensor devices in the wind can help flatten the curve to prevent the generator fluctuation to maintain the voltage profile instead of relying on batteries or flywheels, which are cost effective.

Machine learning from [36] is defined as " a computer program that is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at task in T as measured by P improves experience E". Author [23] defines machine learning (ML) as a field of study that gives computers the ability to learn without being programmed explicitly. The definition from [8] made machine learning easy to understand. Each of the facets E, T, P pose different challenges to different disciplines with different kinds of dataset, although the link in all of the components is the approach or algorithm implemented for specific applications.

Algorithms like ARIMA, SVR, RNN, CNN, etc. have seen frequent implementation in sequence predictive modelling, such as wind power balance. Hence, from our area of interest – time series (sequence) prediction of wind speed ensures the performance improvement is key to effective predictions. Hence, machine learning (ML) is subdivided into three major sections to address time series wind horizon. These sections are:

- Supervised learning.
- Unsupervised learning
- Reinforcement learning.

Supervised learning: This is a machine-learning program where the algorithm is provided with input features processed to have highly correlated relationship with the target or label features to map underlying data patterns for prediction and other purposes. As depicted in Figure 5.24 of section 5.24, here a value in the time domain is equivalent to a predicted target value of wind velocity, in the frequency domain. This algorithm can be of linear and non-linear types. Linear algorithms sequentially searches or checks for a target value within a pool of data. Examples of these algorithms are the ordinary least squares (OLS), Linear regression etc. Non-linear algorithms are the focus of this research. These are the recurrent neural networks, support vector machines (regressions), Kalman filters, Autoregressive Integrated Moving Averages (ARIMA) models, etc that can be used in the search for a more complex sequential and correlated searches. In literal terms, supervised learning algorithms learn the associations between the inputs and outputs having shown the list of datasets. This type of learning are of the classification and regression problems. In regression, the variable to be predicted is in the continuous valued domain unlike the classification, which is in discrete-valued domain. This research however, is of a typical supervised regression machine learning. Furthermore, the question that leads research to other types of machine learning is ‘what if the data lies in an infinite space?’ this is where the unsupervised learning plays a role.

In this type of learning, the why and how the ML works is discussed. The application of which type of algorithm to apply to which type of dataset is discussed as elaborated in section 4.1. in addition, the data collection approaches – whether research requires more samples and so on, the selection of training data samples, the validation and testing samples are also discussed in learning models.

Unsupervised learning on the other hand, ensures the effective predictable outputs; hence, the right answer to a given problem without being provided with prior knowledge of the input samples. This type of learning however is not the discussion of this thesis. The unsupervised type of learning does more by determining the structural pattern of the data [37] by either partitioning the data into classes in the form of clustering or understanding by grouping data structure in the form of segmentation. For example in image processing, computer vision where pixel values are determined and grouped to build models in 3-dimension, 4-diemension, etc. Other areas of application of unsupervised learning are, astronomical data analysis to understand galaxy formations, social (media) network analysis, segmentations, and so on.

Reinforcement learning: in this type of learning, one-time decision-making is not achievable unlike the supervised and unsupervised type of learning; hence not used in this thesis. Here, algorithms make sequence of decisions over time, which may have a direct implication to the subject of study. Self-driving vehicles uses this type of learning. In addition, what to learn is paramount to the learning outcome. In view of the above, the rest of the thesis is organised as shown below.

## 1.4. Organisation of Thesis

Chapter 2 presents and discusses the theoretical and the physical systems used in this research. The discussion however is in terms of the process of wind data acquisition, processing and storage from anemometer sensors, modelling methods and the supervisory control and data

acquisition (SCADA) systems seen in a typical wind farm. In addition, the basic wind turbine operational structures and controls.

Chapter 3 provides literature review in terms of wind speed predictions, the modelling method associated to the regularisation of long short-term memory type of recurrent neural network and dropout method. In addition, the fundamental principles and characteristics required for wind speed data generation, performance measurement while applying and training wind models in time series schemes. Furthermore, the chapter illustrates the fundamental theory required to understand and make full use of recurrent neural networks within the proposed prediction horizons, full description and theoretical analysis of many existing schemes. Finally, a brief description of some different regularisation procedures and the performance measures applied to measure the accuracy of these models.

In chapter 4, the method described in the literature concerning wind speed prediction and data analysis of a wind farm as implemented in this research is presented. The presentation relates to Weibull distributions, derivations of cummulative and probability density functions for Weibull distributions, wind power density and power curve description. In the second part of the chapter, the theoretical concept of then machine-learning regularisation method involving LSTM and dropout on RNN is discussed. The basic steps seen in a practical achievable training, validation and testing ML models, using the RMSE, MSE and MAPE performance measure considerations. This leads to statements about the problems associated with vanishing gradients on RNN. To solve this problem, a method of regularisation for the univariate time series called enhanced LSTM (eLSTM) is derived and a simple robust method applied for obtaining a multivariate set of predicted data structures, which present regularisation along with their corresponding co-adaptation system structures. In the final part, some further topics related to data analysis and the concept of data merging are discussed.

A case study based on wind-farm power output generation from a 14-turbine unit within the farm is presented in chapter 5. The chapter is divided into two parts. The first part presents simulation results for the proposed power output models. Simulation of the power generation output from the farm, described in chapter 4, is performed; discussion of the results are presented as they are generated. Secondly, simulated real-time training, testing and validation with RMSE, MSE and MAPE results comparing various regularisation methods as discussed in chapter 4, are presented for different prediction horizons. This section discusses the comparison of simulated results generated by comparing the novel approach with other regularisation schemes. In addition, there is a comparison with other sophisticated predictive algorithms in the area of research: Autoregressive Moving Average (ARIMA) in particular.

Finally, a general discussion, future work and conclusions are presented in chapter 6. This chapter reviews the main achievement reported in the thesis, makes some suggestions for further research to extend the proposed methods and introduces a completly new logical and mathematically rich approach. While the main results are presented in the body of the thesis, the appendices give important supporting material, derivations and extensive proofs. These include appendix A and B.

# Chapter 2

## Components for Wind Measurements.

The observation and recording of wind information are classified into two aspects, the wind direction and wind speed. The wind direction indicates the direction in which wind blows and flows. This flow is usually dependent on location, hence, the cardinal points – east (E), west (W), north (N) and south (S) or a combination of the two as shown in Figure 2.1. From the figure however, the dominant direction of wind is in the SW direction. The velocity and force over a unit area of a location in which this wind flows is referred to as wind speed. The wind speed and direction are measured in meters per seconds (m/s). Therefore, the wind rose of Figure 2.1 has the highest wind speed of around 18 – 20 m/s.

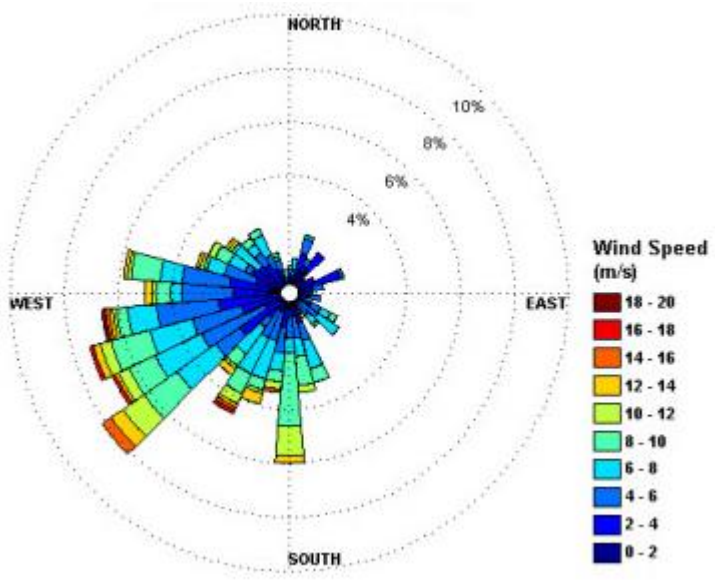



Figure 2. 1: Wind Rose Diagram of 0.125$^{o}$ at 3-hour Interval
Source: *European Centre for Medium-Range Weather Forecasts* https://www.ecmwf.int/ [1]

To obtain accurate measurement, the Met Office of the United Kingdom, the United States of America and other research institutes suggests that where possible the measuring device is to

[1] Website accessed 4th October, 2018

be situated (mounted) on a tower in a large open area to avoid possible interference such as trees, and buildings [102] especially for offshore wind measurements. In addition, measuring sensors, such as anemometers and wind vanes are expected[2] to be 33 feet (10 meters) above the ground. However, for better accuracy, the sensors are to be at least ten times the height of any obstruction.

## 2.1. Wind Measurement and Anemometers Sensor Technology.

In meteorology and wind science, measuring wind speed is achieved with a sensor called an anemometer. The wind speed sensor has many types and design specifications, the design by Dr. John Robinsons' has been used to measure wind speed since 1846. John Patterson, the standard for wind resource assessment, in 1926, developed other sensors such as the 3-cup anemometer as shown in Figure 2.2. The anemometers have a linear measurement range of 0.3 to 75 m/s with a measurement uncertainty of less than 1%. The 1940 physicist Leon Battista Alberti was the first to describe this device. Over the years, this device has evolved through different types with similar underlying principles, to measure force over a certain area of wind in a given location. This device however has many types, namely: the Hot-wire, Laser, Doppler, Ultrasonic, Acoustic resonance, Plate, Tube, Cup anemometers and so on [103].

[2] The renewable research community: https://www.metoffice.gov.uk/guide/weather/observations-guide/how-we-measure-wind, http://www.noaa.gov/weather, https://www.weather.gov/, etc.
Date accessed 15th October, 2018

Figure 2. 2: Typical 3-cup Anemometer.

Cup anemometers appears to be the simplest to understand and the most widely used for wind speed measurements and hence are used for this research.

2.1.1. Working Principle of Cup Anemometer.

Cup anemometers records wind speed from wind in a given location using the revolution or the number of times it spins in a given period. This period is recorded every hour, minutes or seconds depending on the design. The diameter of this instrument as shown in Figure 2.3 is used to calculate the circumference or area around the circle of the cup anemometer in Eq. (2.1);

$$C = d*\pi \qquad (2.1)$$

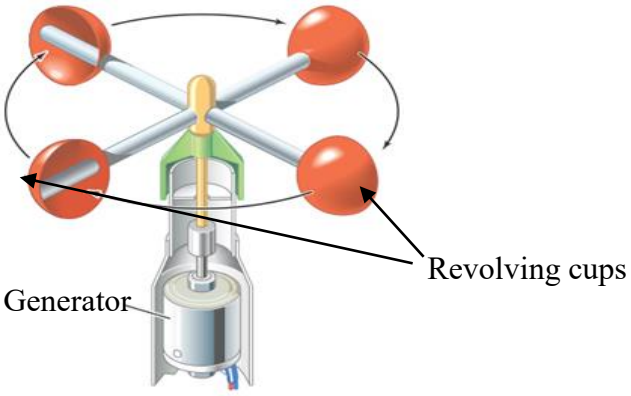



Figure 2. 3: Cup Anemometer for Wind Speed Measurement.

From the figure, assuming the diameter is 7.4 inches, to calculate the circumference, Eq. (2.2) shows how far wind travels every time it spins

$$\frac{23.24}{1} * \frac{1 \text{ foot}}{12 \text{ inches}} = 1.94\text{feet} \quad (2.2)$$

To model the distance it is traveling, we used an arbitrary number of counts in revolutions, which is equivalent to 179, hence,

$$\frac{1.94}{1} * \frac{179 \text{ revolutions}}{1} = 346.60\text{feet/minutes} \quad (2.3)$$

To achieve the value per hour,

$$\frac{346.60}{1 \text{ minute}} * \frac{60 \text{ minutes}}{1 \text{ hour}} = 20{,}796 \text{ feet/hour} \quad (2.4)$$

Achieving this value in terms of miles per hour, we have

$$\frac{20796 \text{ feet}}{1 \text{ hour}} * \frac{1 \text{ mile}}{5280 \text{ feet}} = 3.9\text{miles/hour} \quad (2.5)$$

This is the working principle behind wind speed measurements. As discussed in chapter 3, the data we are using is recorded in 10 minutes intervals. Errors such as inertia and rust experienced in cup anemometers [104] are not discussed in this thesis. However, the process of acquiring, processing and storing of wind data from a typical wind farm for prediction of power output is described.

## 2.2 Data Acquisition System.

In the recent systems, acquisition of wind data for wind power prediction are generally categorized into;

- Wind mast. This records the measurement of wind speed, wind direction, temperature, humidity and pressure. In addition, the mast temporarily logs these generated data into loggers, which is periodically sent to analyzers through GPRS enabled connections.

- The SCADA systems. This system enables the operators to export data for processing. Further data files, for example turbine status, wind speed and other meteorological measurements are logged to the system as shown in Figure 2.4.
- The global forecast systems (GFS) and European center for medium-range weather forecast (ECMWF) systems are downloaded to the SCADA periodically. Other models such as the numerical weather prediction models are downloaded to the center in an automated manner [105].

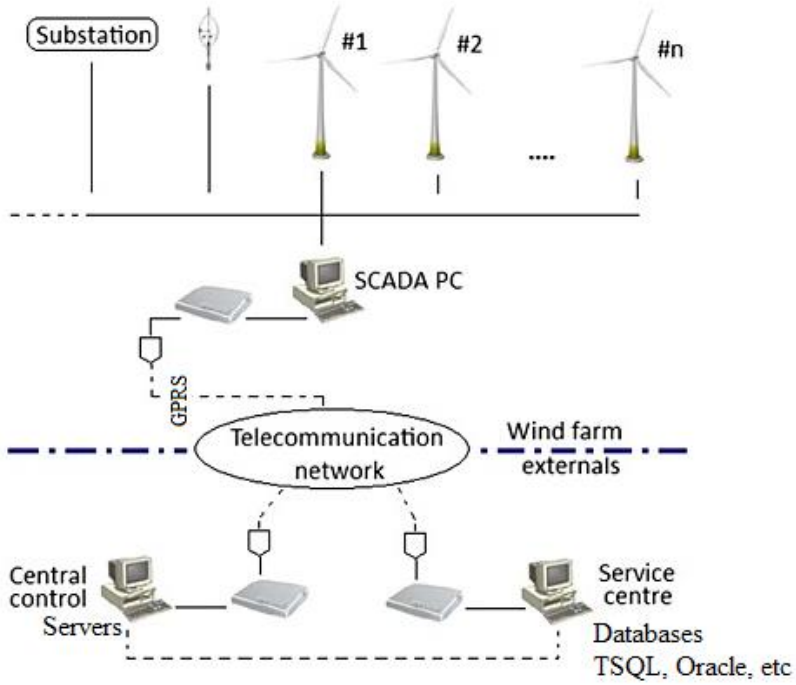



Figure 2. 4: A Simple SCADA Architecture for WFPO Prediction.
Source: Researchgate [3]

2.2.1 Data Storage and Processing.

Wind power prediction and monitoring center comprises of servers as shown in the Figure 2.4 above, which satisfy the overall storage and processing requirements systems. These servers comprise of database servers, for example, Oracle, MySQL, Microsoft-SQL, PostgreSQL, and

[3] https://lh3.googleusercontent.com/
Date Accessed: 15th October, 2018

so on, installed to create database instances. Projects and application web pages are hosted in a Web Server. The data processing server hosts the data modelling and processing – normalization, up sampling, down sampling, conversions, model performance, and so on are also hosted in the prediction application server for periodic runs.

## 2.3. Wind Turbine Operations

The thing with wind turbines is that structurally, it is a strange structure installed by human beings. From the technical literature, wind turbine operation relies on site selection, and elevation level for its performance. However, wind turbine function is better in a low elevation range of hills surrounded by higher mountains. At this elevation, the flow of wind is facing incoming winds thereby squeezing air downwards for an increased wind speed, which in turn increases wind energy yield.

### 2.3.1. Wind turbine Component Functionalities.

Wind turbines converts mechanical energy gained from the rotation of the rotor through the blades to electrical energy by spinning round a set of coils to create electricity. From Figure 2.5, the electricity is brought from the wind turbine into the ground through cables, from which, it goes into a transformer and is finally sent over a sub-station or switchyard to serve consumers.

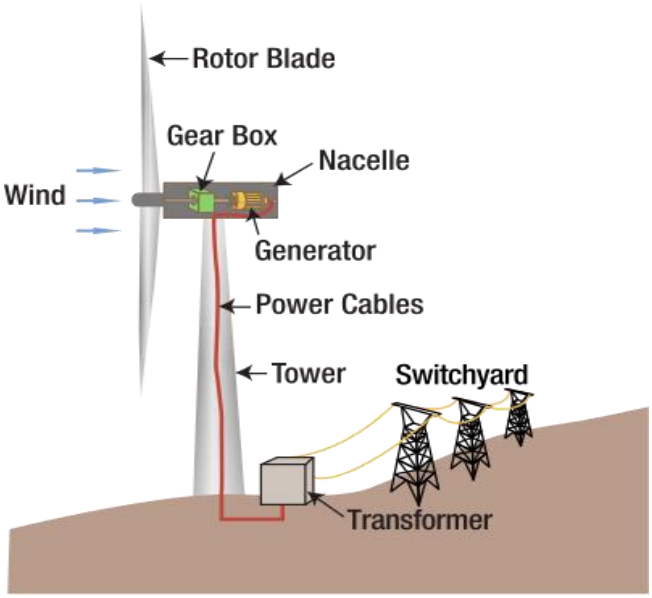



Figure 2. 5: Typical Wind Turbine operation.
Source: Researchgate[4]

Wind turbines are of different sizes in terms of electricity generation. A one megawatt turbine is assumed to produce up to a megawatt of electricity, which typically powers about 750 homes [106]. In the past wind turbines produces alternating currents (AC) which is directly as used in homes. Because of this direct home usage, the turbine is expected to spin at a reasonably precise speed, which is narrow in terms of power productions such that wind could synchronise the electricity frequency to that used in homes. However nowadays, with the use of micro-electronic control systems, wind energy is produced as direct currents (DC), which allows the turbines to spin at a wider range of wind speeds to pull more energy out of the air for all weather conditions.

Wind turbine designs are based on frequency ratings; older wind turbines produced around the 1970s are rated from about 60 kilowatts to roughly 34 Megawatts. Their height ranges from about 30 feet to about 150 or 200 feet. These turbines operates to about 14 – 18% efficiency and are not very efficient compared to what is in existence now. Recent technology has helped

[4] https://ars.els-cdn.com/content/image.jpg
Date accessed. 4th June 2017

the repowering of wind turbines such that they are not only more efficient in terms of electrical power output and therefore economics but better in terms of environmental impact due to reduced maintenance requirements.

#### 2.3.1.1. Basic Components of Wind Turbines.

- The rotor, which is approximately 20% of the wind turbine cost, includes the blades for converting wind energy to low speed rotational energy.
- The generator, which is approximately 34% of the wind turbine cost, includes the electrical generator, the control electronics, and the gearbox. The gearbox is planetary with an adjustable-speed drive or continuously variable transmission component for converting the low-speed incoming rotation to high-speed rotation suitable for generating electricity.
- The surrounding structure, which is approximately 15% of the wind turbine cost, includes the tower and rotor yaw mechanism.

A typical 1.5 MW wind turbine with an 80 meter tower has the rotor assembly as follows: blades and hub weight of 22,000 kilograms. The nacelle, which contains the generator, weighs 52,000 kilograms. The concrete base for the tower is constructed using 26,000 kilograms of reinforcing steel and contains 190 cubic meters of concrete. The base is 15 meters in diameter and 2.4 meters thick near the centre. The interconnection of several wind turbines for a unified power output forms a wind farm. This interconnection has many variants, which are not discussed in this thesis. Although chapter 4 highlights the architecture for power output predictions.

## 2.4. Types of Wind Farm.

Wind farm is of two types, the on-land and offshore wind farms. The offshore wind farms are those sited on the bodies of water usually the ocean to extract wind energy for electricity generation.

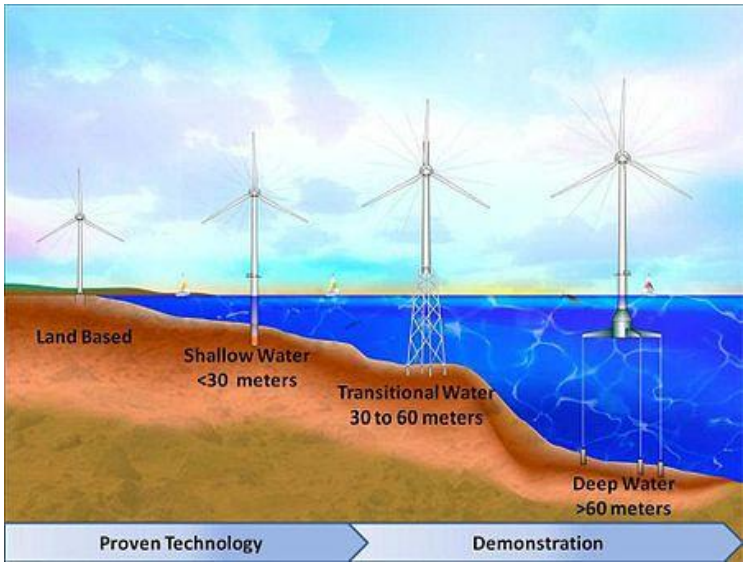



Figure 2. 6: A Typical Offshore Wind Farm
Source: Wikipeadia[5]

These wind farms generate more wind energy per amount of capacity installed since high wind speeds are available offshore compared to on-land, as shown in Figure 2.6. The on-land wind sites as depicted in Figure 2.7 are the wind farms sites on the earth's surface. They have greater hub-heights due to high wind speed in higher atmospheric boundary layers.

Figure 2. 7: A Typical On-land Wind Farm.
Source: Mudgeon Files[6]

[5] https://upload.wikimedia.org/wikipedia/
Date accessed 10th August 2018.
[6] https://thenoisecurmudgeon.files.wordpress.com/2013/09/turbine2.jpg
Date accessed 10th August 2018.

This is because the closer they are to the ground, the higher the roughness is increased as shown in Figure 2.8. For sites with higher terrain roughness, higher–towered turbines appear to be ideal to combat energy loss as turbulence occurs below the blades and does not affect yield.

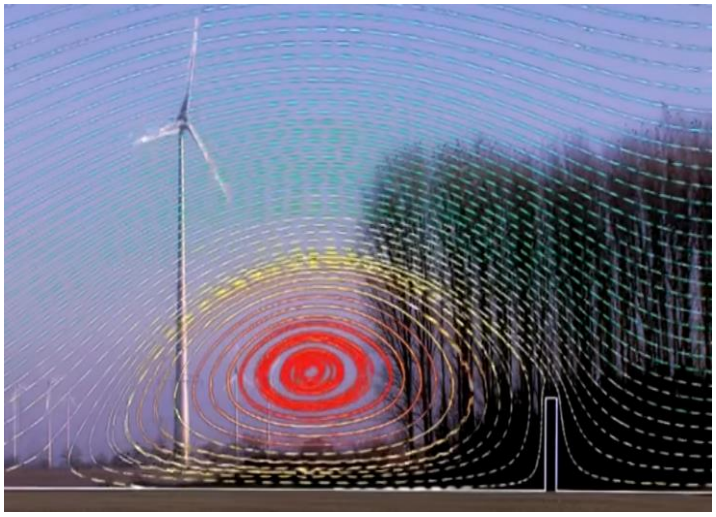


Figure 2. 8**:** Turbulence seen in on-land turbines.
Source: Mudgeon[7]

Wind turbine maintenance is a big topic in the renewable energy industry, although not discussed in this thesis. Friction reduces energy flow as the blades rotate. Uncontrollable friction results in damage to the turbine. The lesser the bearing points, the less likely to experience turbine disintegration. Reduction in mechanical stress increases service life of the turbine, which in turn increases turbine efficiency. This led to the installation of the microelectronics devices in the cabin such that individual use cases are designed in a modular manner for specific project applications.

[7] https://thenoisecurmudgeon.files.wordpress.com/turbine.jpg
Date accessed 16th June 2018.

# Chapter 3.

## Literature Review

A broad area of work on the review and theoretical design of wind power prediction is discussed in this chapter; particularly by addressing the wind speed prediction survey and characteristics and; the use of long short-term memory of recurrent dropout regularisation strategy to combat overfitting. In addition, the detection of error dynamics using performance measures are reviewed, which in turn depicts the knowledge acquired in modelling predictive schemes – resulting in model improvements and accuracies.

To use the dropout method, it is necessary to understand the nature of these systems, at a basic level and successfully apply them for wind renewable energy system predictions. In section 3.1, renewable energy wind data, which is associated with modelling, is discussed with respect to the algorithms used in modelling various wind systems. Wind power models are discussed in section 3.3. These models are the recurrent neural network with application to wind power modelling. The section however further discusses neural network training in relation to recurrent neural network (RNN) and long short-term memory (LSTM); demonstrating certain training case studies. Section 3.4 studies the improvements in recurrent neural networks in the form of regularisation methods applied to improve RNN's gradient vanishing problems. Model performance results are used for acceptance of a given algorithm. Furthermore, performance methods used in time series sequence systems associated with RNN and the LSTM are as discussed in section 3.5. This section however studied the basic theory of performance methods used in the research. Some necessary properties and basic ideas of recurrent neural networks and long short-term memory are introduced in section (3.2) alongside a brief description of their underlying properties.

A simple system, which is supposed to be used to simulate these schemes as suggested in chapter 4, is also introduced in this section. However, the required properties and nature of alternative systems are also discussed.

## 3.1. Renewables and Wind Data.

Wind is the flow and movement of gases and air molecules in the atmosphere. This movement or flow exerts a certain amount of force through the collective weight of gas molecules acting on a specified area; this is typically described as air pressure. This pressure, in turn, varies from location to location, time of the day, weather, landforms and height above land surfaces. Understanding wind characteristics help research in optimizing wind turbine design, wind site selection, measuring techniques and wind power generation from various interconnected turbines within a wind farm.

### 3.1.1. Air pressure, Temperature and Wind Speed Data.

One of the most critical characteristics of wind renewable power generation is wind speed, measured in meters per second. Wind speed changes dynamically in both space and time, and is determined by many factors such as weather and geographical conditions. Measuring wind speed is one of the complex aspects of power generation from wind, although statistical methods help in realizing a given wind speed for use in renewable energy generation. In addition, wind speed measurement is described by its diurnal variations, which in turn is theoretically synthesized by sine waves.

From a global perspective, the variations of wind have been analysed in conjunction with temperature of a given location, sun intensity during the day or a given period and relative pattern within a given geographical region. For example, the description by [38] analysed wind speed data over a period between 1970 – 2003 using data from 65 onshore sites across the

United Kingdom to conclude that the monthly average wind speed is inversely proportional to the average monthly temperature. This is because; from the research, wind speed is lower in the summer and higher in the winter, which results in minimum wind speed reports in August and maximum in January.

Describing month-to-month wind speed variations over a fourteen year period, between 1970 – 1984, [39] reported that in Saudi Arabia, at Dhahran, yearly low-temperature variations in a wavy pattern shows no-clear connection between temperatures and wind speed. The wavy pattern at the same location, however, was further reported to experience higher wind speed at daytime at around 3 PM, maximum wind speed is seen in Dhahran, indicating that sunlight is proportional to daytime wind speed. On the other hand, [40] demonstrated that diurnal wind pattern at five different locations in Texas – USA follows a pattern similar to the ones reported in Jos, Nigeria [41] wind speed appears to be constant in the night time while having a curvilinear pattern during the daytime.

Yearly variations of mean wind speed across several locations as described by [42] shows there is no common location in terms of predictive abilities. This is because, in the research by [28, 42-44], similar attributes are reported at which annual wind speed decreases exponentially with time within a thirteen-year, 1970 - 1983 period at Dhahram. In the UK [38] reported a more variable display with the similar year but a longer period of thirty-three years, 1970 – 2003. However, the *European Union Energy Association* [45] reported similar variation significance of an annual wind speed over 20 years having maximum and minimum wind values ranging from 9.2m/s to 7.8m/s respectively.

In the same review, [46] reported long-term wind data in a 29-year period, 1978 to 2007 obtained from an automated synoptic observatory. Studying their results, it is worthy of note that wind speed within the location (Dhahram and UK) experiences slightly low fluctuation

around *Jeju Island* having other sites with randomly depicted trends. In addition, the yearly wind speed variations require statistical analysis for variation decomposition.

Understanding wind speed across these locations is typical of its distribution derivatives, hence, this helps energy and wind power derivations from wind. This is due to waveform sampling derived over time, thus the illustration of the wave-like structure as seen in the literature for wind data series. These distributive derivatives results in effective predictive model building which requires in-depth knowledge.

3.1.2. Modelling Wind Speed.

From the survey, wind speed variations at any location are best described using various statistics. The most used, Weibull probability distribution [47] depicts the illustration of the probability at variable mean wind speed, reported to have occured at different time periods Variable wind speeds are recorded from a given sensor within a certain frequency of occurrence and resolution of time either every 30 minutes, hourly, daily, or weekly. In this research however, the wind distribution is reported every 10 minutes. One of the most popular statistical distributions, Rayleigh in [48] used for the probability density function for wind speed description is described in section 4.7.

3.1.2.1. Wind turbulence.

Theoretically, turbulence is the fluctuation of wind speed over certain time scale usually for the horizontal velocity component. From Eq. (3.1), the wind speed u(t) at any instant of time t is considered as having two components; the instantaneous speed fluctuation u'(t) and mean wind speed ($\bar{u}$).

$$u(t) = u'(t) + \bar{u}(t) \tag{3.1}$$

The power output of a wind turbine depends strongly on wind turbulence. This results in dynamic fatigue loads in the turbine blades for heavy turbulence, which in turn reduces turbine lifetime or results in failure. In addition, wind farm selection requires the knowledge of wind turbulence intensity as described in [38, 42, 45] for optimum energy generation.

3.1.2.2 Wind Direction.

This is one of the major characteristics of wind speed, although mainly required for wind farm selection at a location within a specific time (day, week, month, year, season, etc.) To analyse and understand winds in terms of its direction, the wind rose diagram is one of the most useful tools used. The wind direction of the field data used in this research is depicted in Figure 3.1. In the figure, there are sixteen radial lines which are $22.5^0$ apart from each other. The length of each one is proportional to wind magnitude for that direction.

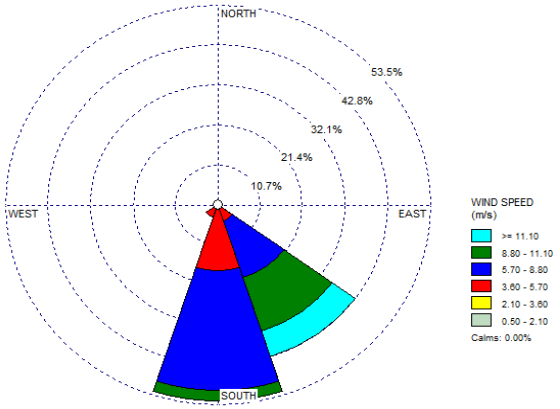



Figure 3. 1**:** Direction of flow – blowing from South.

However, from our dataset, the direction of flow shows that wind is flowing from Southern Texas to the North as shown in Figure 3.2. Near calm or calm air is described by the frequency of a given number in the cycle. Information of wind speeds is contained in the wind rose tool, the figure describes wind direction as used in the thesis.

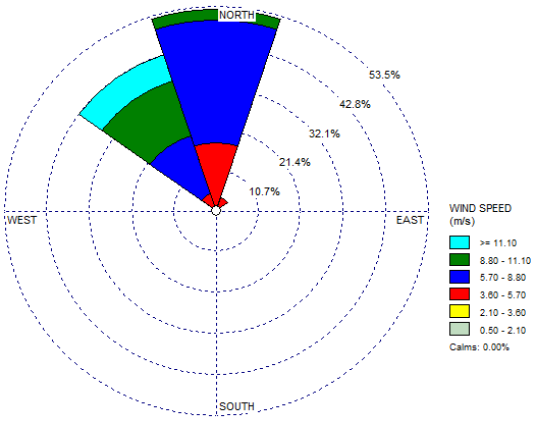



Figure 3. 2**:** Direction of flow – blowing to North.

3.1.2.3. Wind Shear.

This meteorological phenomenon describes the increase in speed as a function of height above the earth's surface and roughness. In addition, the effect on wind speed, which is estimated using traditional equations popularly known as the Hellmann power equations (see chapter 4, section 4.2). From the equation, $Z_0$ is the reference height at which wind speed is known while Z is the height above earth surface and *a* is the shear coefficient.

Other types of wind characteristics not discussed in the thesis are the wind gust, seasonal, and annual patterns. Although wind gust like wind intensity and wind speed changes in cases of turbulent blasts, the wind gust ensures the maintenance of power output from turbines and reduces rotor imbalances.

Building a predictive model for wind power, is the aim of the research and it involves historical recorded data for wind speed, temperature, air pressure, precipitation, and so on since they are the major characteristics required for wind energy predictions. This, however, requires modification of the RNN architecture, the long short-term memory RNN, statistical models like least square regressions, etc. for wind speed predictions.

## 3.2. Survey on Wind Speed and Power Predictions.

Comparing the amount of research work and number of research publications related to wind speed, power prediction in general, and artificial neural network. The research effort focused on the development of a hybrid algorithm to enhance existing algorithms related to recurrent neural networks – long short-term memory and dropout regularisations. The most relevant work that could be cited is those related to sequence prediction by [25, 31] and time series by [42, 48]. These researchers rely on features commonly applied in tasks similar to this work. Different research groups use different features hence; the underlying characteristics are usually the same. The sequence behavioural pattern that exists in a time series system keeps track of *samples, time steps and features* for LSTM implementations. Therefore, inspirations on this research are routed in research carried out by [28, 31-33] where the idea of employing sequential modelling is introduced for time series sequence applying dropout on LSTM to forecast sequence generation for speech recognition, handwriting recognition and machine translation. The concept of leveraging mid-level RNN representation in LSTM [27] described in image label annotation is also exploited. Although time series wind data are of sequence-generated data, the characteristics in terms of size are usually not as described above. Hence, have different features and require modifications prior to model applications.

Predicting energy consumption and wind power for households using LSTM as reported [23, 34] and further investigated. Although in [23], no improvement on LSTM is experienced, hence, effective learning of measured energy consumption profile was reported. However, in a typical time series scenario similar to wind speed prediction [29], use a dropout implementation to improve LSTM to forecast times series sequence events of possible churn for an online course platform. Author recognized that time series sequential models require a data model implementing statistical analysis for effective prediction.

Multi-step forecasting of wind speed was conducted by [4] using an ensemble or combination of two models – the empirical mode decomposition (EMD) and feedforward neural network (FFNN). For each of the models, the nonlinear wind speed is decomposed into small chunks. The residual series of the counterpart EMD enhanced insights on the data structures involving monthly mean wind speed data over three years. In order to measure the performance of these models, MSE, MAPE and MAE metrics measured several trials independently. In addition, different experimental sets were conducted with mostly good predictors where the resultant error signature was then used to train two supervised learning models. The regression performance of the trained model when presented in testing data found an improved 12% reduced error over training FFNN on the average of the metrics.

The author in [34] investigated the hidden features (rules) of wind speed pattern based on a deep belief neural network (DBN) having just three hidden units. In the proposed system, the transient wind speed samples on independent layers reported as an error window, which is as shown in the three hidden layers. With a five-second resolution, up-sampled data points of up to 150 points where 90 points were inputs to the training model while the rest are used for testing the supervised deep network. The MSE and MAE metrics reported model performance. The derived results from this work show that a regression performance of about 11% predictive error was reported as an improvement over other compared neural network models.

In a bid to demonstrate the need for regularizing the recurrent neural network such as LSTM, [23] developed a probabilistic approach of a second order system in training a wind model recursively[8]. Authors' implemented the Levermberge-Marquardt algorithm (LMA) to update the weight of the network during training at 1000 time-steps-long samples and modelled

[8] Model training in a repeated fashion.
In this case, model training was repeated 15 times and compared side-by-side among the performance metrics.

Bayesian RNN as a regularisation method, adding white-noise of 0.05 standard deviation. During model verifications, 250 samples were used for model testing. Normalized mean squared error (nMSE) metrics measured the performance model as they further compared their model with training a Kalman filter, feedforward multilayer perceptron and LMA (MLP-LMA) and support vector regression (SVR). To make sure random weight adaptation takes place during training, 8 days of data were fed into the RNN[9]. The nMSE result demonstrated that the Bayesian RNN has up to 7.5% improved performance over other algorithms including an SVR.

The previous work cited above is related to this research in a predictive sense, having regularisation with the dropout method. However, the proposed scheme of wind speed prediction in this thesis relies upon pattern identification of wind signatures in a wind data by a supervised neural network-based decision module. It is important to review some of the previous work related to ANN based wind speed prediction using machine learning and statistical data modelling.

To compare and implement a simpler non-linear model over complex ones like ANN and adaptive fuzzy inference system (ANFIS), [49] applied a polynomial autoregressive (PAR) model over ANFIS and an ANN[10]. A 2-month recorded hourly wind speed data, fed into the model using 80% of the samples for training and 20% for testing. The author reported however that PAR recorded better performance over ANN and ANFIS due to its linear-in-the-parameter property[11]. The simulation used normalized mean absolute percentage error (nMAPE) alongside normalized root mean squared error (nRMSE) performance metrics to measure the

[9] The process of avoiding weight adaptation is known as priming, it improves regression model performance though it is not trust worthy as it could fail over a long sequence.
[10] The artificial neural network described here is a recursive feed forward neural network designed by the authors.
[11] This parameter depicts a typical linear regression design. Performs better on a small sample data.

performance of all the models. The result shows that on average, PAR performed better than both ANN and ANFIS with up to 9% accuracy over a 24-hour ahead prediction.

Furthermore, in sequence regularisation methods, long short-term memory (LSTM) and its corresponding dropout method, which the thesis relies on has recorded better performance over techniques like hidden Markov models (HMM), learning vector quantization (LVQ), support vector machines (SVM), convolution deep belief network (CDBN), and so on. Hence, in language modelling and handwritten tasks, LSTM comes with different kinds of flavours as described in section 2.4. However, authors in [32] worked on a particular type of LSTM called multi-dimensional long short-term memory (MDLSTM) for handwritten recognition tasks. Their work, however, describes the effect of applying dropout regularisation in both the input, output and hidden LSTM layers. In addition, their results experienced overfitting and poor generalization using lonely LSTM model. To address overfitting on the training data, authors applied the dropout technique involving 50 percent on hidden neurons. They carried out this research based on a well-known IFN/ENIT[12] database. The experiment applied MSE performance metrics and ADAGRAD optimization methods on all the layers. Unlike the traditional time series, the datasets possess similar sequential structures although, the data is presented in a pixel-like manner of up to 100, 200, 300 pixels respectively. An error rate improvement of 8.6% is reported while training MDLSTM over 8.5% of CDBN and others as tested. This result is due to the dropout model application during model training.

To buttress the need to enhance long short-term memory recurrent neural network (LSTM-RNN) for cases applied to this thesis, [50, 51] extracts error prone engineered features to capture the vanishing gradient problem experienced on LSTM for language modelling tasks. Their work, however, describes the effect of applying dropout regularisation in the hidden

[12] A popular public database for training and testing Arabic handwritten text recognition systems.

LSTM layer. The author reports overfitting using only the LSTM model. To address this problem and poor generalization on the training data, authors applied the dropout technique involving 50 percent of hidden neurons. They carried out this research based on a well-known IFN/ENIT[13] database. The experiment applied MSE performance metrics and the RMSprop optimization method on the layers. The datasets again, possess similar sequential structures; the data is presented in a bit-like manner. An error rate improvement of 5.6% is reported while training LSTM over 6.4% of CNN and others as tested. This result is due to the dropout model application during model training.

In order to use weight regularisation for a multistep sequence forecasting, [20] applied dropout on LSTM for time series monitoring and prediction of critical temperature on permanent magnet of a synchronous motor. Authors applied principal component analysis (PCA) for training and testing data, using a 15 Particle Swarm approach for hyper-parameter optimisation. It is deduced that the RNN-LSTM-dropout approach significantly outperformed the traditional lumped-parameter thermal networks (LPTNs) approach.

In order to ascertain the best part of the neural layer to apply dropout in an LSTM architecture, [28] performed a sequence prediction for a handwritten recognition problem. Authors, however, show that better improvement can be reported by implementing dropout differently especially on different layers (units). In addition, MDLSTM-type of LSTM was implemented in this research and further compared with HMM, CDBN[14]. The available data is Rimes[15] training set in French, which is up to 1,500 paragraphs, manually extracted from the images, and an evaluation set of 100 paragraphs. They held out the last 149 paragraphs (approximately 10%) of the training set as a validation set and trained the systems on the remaining 1,391

[13] A popular public database for training and testing Arabic handwritten text recognition systems.
[14] This is synonymous to a typical multivariate sequence prediction. The variables are complex and cost effective in terms of structuring the neurons unlike the univariate systems.
[15] Rimes is a popular data management company responsible for benchmark data services for research and development.

paragraphs. For the recognition dataset, handwritten pages that correspond to English text with 747 images for training and 116 for validation using 336 for evaluation. The last dataset was the handwritten notes from British philosopher Jeremy Bentham that is comprised of 350 pages used for training, 50 images for validation and 33 pages for testing the algorithm. The performance metrics applied were a simple character and word error rate (**CER** and **WER**) baseline model with *no dropout*, traditional LSTM *with dropout* applied *before* and *after* modelling *without dropout*. Authors finally report that dropout is best applied to the inputs and outputs layers of the network respectively.

## 3.3. Description of Typical Models

In engineering system specifications, component selections, modifications and assembly, design and analysis rely on theoretical understanding of best performance. Data science and other related fields, especially predictive analytics follows these trends in ensuring best practice in building predictive models for research implementations. Modern technology over the years has relied on traditional systems like statistics and mathematics using logical algorithm programming to improve understanding of how data informs decisions and how insights are gained from data. This method is described by the term, modelling. This aspect has seen authors of different disciplines define modelling to suit their respective areas of discipline.

### 3.3.1. Recurrent Neural Network and Machine Learning Modeling.

Recurrent neural network (RNN) is a type of neural network used in modelling complex systems like wind speed for forecasting. Wind speed is stochastic in nature with an irregular sequence that requires complex models like autoregressive integrated moving average (ARIMA), support vector regression (SVR), and RNN to model. Due to the vanishing gradients, issues of computational expensiveness and representational power, research has come up with different architectural approaches, namely; LSTM, gated recurrent units (GRUs),

and Stacking[16] as discussed in section 4.3. Although, RNNs are derivations from inabilities of feedforward neural networks as shown Figure 3.3, RNNs perform better in real-life applications.

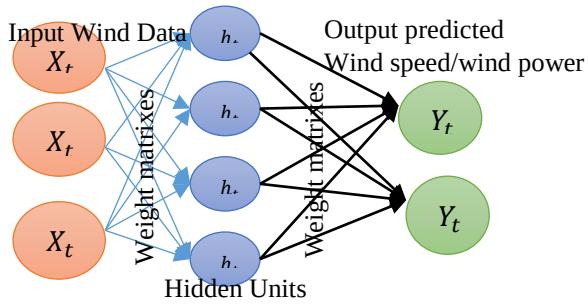



Figure 3. 3: Simple feed-forward neural network.

Discussing RNN requires a review of feedforward neural networks as shown in the figure above. It is important to understand that each unit does relatively straightforward computations using Eq. (3.1). This is done by taking the input $X_j$ and multiplying it by weight $W_{ij}$ and adding bias-term $b_i$ to performs a sum and then pass them through an activation function $g$ to yield the output $Y_t$.

Using vector notation,$W_1$,$W_2$,…,$W_n$ forms a matrix representing the connection between layers, which in turn, yields $Y_k = g(Wy_{k-1} + b)$

$$Y_i = g\left(\sum_j x_j W_{ij} + b_i\right) \quad (3.1)$$

Another interesting aspect of neural networks (NN) is training. Training a NN is a process where a cost function is derived with respect to a derivative of the weights, followed by an application of a mathematical chain rule to move the derivative through the nested layers of

[16] This is a practice in neural network that involves ensemble of LSTMs or algorithms to form single LSTM or algorithm. Usually applied in multivariate or complex systems.

computations. In the form described by Eq. (3.1). Applying the chain rule with respect to Eq. (3.1) forms Eq. (3.2). This systematic unfolding is as shown in Figure 3.4.

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial g} * \frac{\partial g}{\partial a} * \frac{\partial a}{\partial W} \tag{3.2}$$

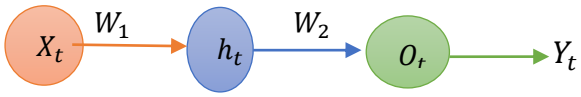



Figure 3. 4**:** Unfolding feed-forward neural network.

On the other hand, the feed-forward neural network (FFNN) [52-54] is robust but has limitations. One of the limitations is the concept of ***fixed length*** where the size of the input layer is fixed. For example, the image size is usually of a fixed length of **32 X 32** pixels whereas, in time series sequence, the length of the input varies from example to example [53, 54] up to an order of magnitude. However, in a more formal manner, given a set of sequences $x = (x_1, x_2, \dots, x_n)$, RNN updates its recurrent hidden state $h_t$ by Eq. (3.3)

$$h_t = \begin{cases} 0, & t = 0 \\ \beta(h_{t-1}, x_t), & \text{otherwise} \end{cases} \tag{3.3}$$

where β is a nonlinear function representing logistic sigmoid of an affine transformation. This recurrent hidden state is updated as implemented in Eq. (3.4) to form Figure 3.5.

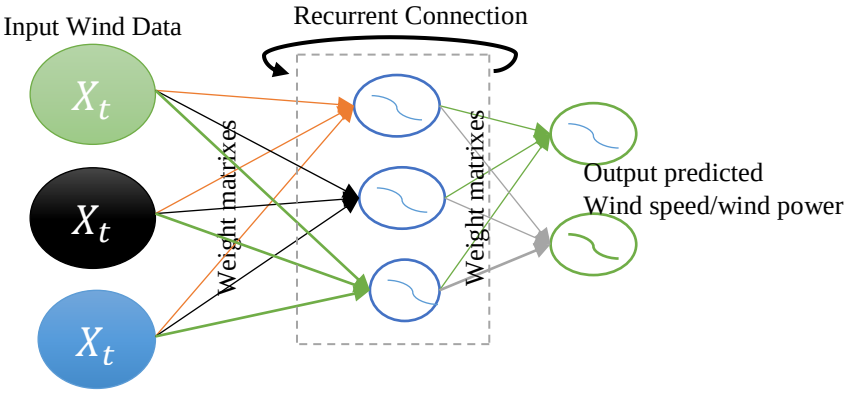



Figure 3. 5: Simple recurrent neural network.

Another reason is the issue of **independence**; this is because different training examples are independent of each other. Hence, other data structures like sentences, voice, which has short and long-term dependencies deal with different training examples in an order of sequence. This, however, led to an RNN that not only learns the short and long-term dependencies but also accommodates input sequences of variable length.

A simple recurrent neuron is as shown in Figure 3.5, which forms Eq. (3.4), from the equation, the difference between Eq. (3.3) and (3.4) is the $\Theta\phi\ (h_{i-1})$ term that depends on the previous time step, multiplied by the weight matrix $Y_t$. This however, informs the basics of recurrent unit as $g_y(W_y h_t + b_y)$ , which in turn forms many of the recurrent units around the study area.

$$h_t = \Theta\phi\ (h_{i-1}) + \theta_x X_t$$
$$y_t = \theta_y\ \phi\ (h_t) \qquad (3.4)$$

3.3.1.1 Basics in Training Neural Networks

The question that usually arises is how such a complicated network is trained. This is done by simply unrolling the complicated network with time to turn the complications into an FFNN form as shown in Figure 3.6. From the figure, the activity of $h_{t+1}$ not only depends on $x_t$ but also the units of activities at the previous time steps as shown in $\theta_i$ of Figure four.

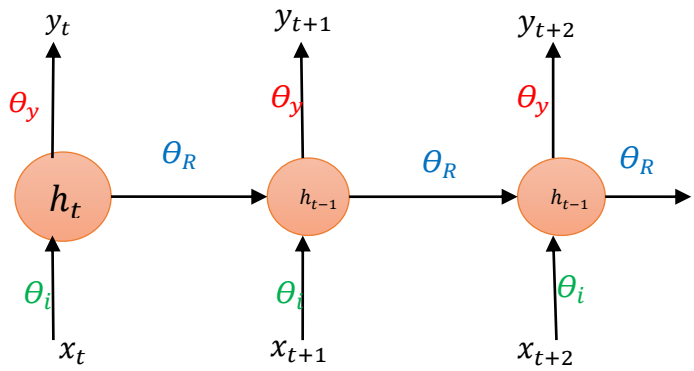



Figure 3. 6**:** Unfolding Simple RNN

The only difference in unfolding RNN and a typical FFNN is that this unit $\theta_i$ , depends on the activity of the previous time step. This is one of the reasons RNN applications perform better in sequential systems since the given inputs are shifted forward in time in a sense using conditional probability models. This probability predicts the sequence of events at time t given a history of activities before t as in $P(H_t|\{H_{t-1}, H_{t-2,}H_{t-3,} \dots, H_0\})$ Conversely, RNN is applied in sentiment analysis to classify an event as positive or negative using the similar method.

The network can be unrolled in time to mimic FFNN as shown in Figure 3.3, training RNN is different [55]. In FFNN, the weight matrix is shared across the network whereas, in RNN, the weight matrix has a comparison in mind while unrolling the network. This comparison is against every time step of the network. Therefore, for ease of interpretation, these comparisons are combined to obtain a gradient update for the weight matrix $\theta_R$, which is derived from Eq. (3.5); this process of computation is known as the chain rule. It is used to back-propagate the nested layer through a set of propagations.

$$\frac{\partial C}{\partial \theta_R} = \sum_t \frac{\partial C_t}{\partial \theta_R}$$

$$\frac{\partial C_2}{\partial \theta_R} = \sum_{k=1}^{t} \frac{\partial C_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial \theta_k}\frac{\partial h_k}{\partial \theta_R} \tag{3.5}$$

However, unlike in Eq. (3.4) where $W_R$ ( $or\ \theta_R$) is commonly shared, the summation is back propagated across each time step as in $a = (W_1\, x_2 + \ W_R\, h_1 + \ \ b_n)$ where $h_1$ depends on $W_R$ over long term dependencies. This dependence introduces an issue addressed as exploding or vanishing gradients of Eq. (3.6), hence, discovered by [56]. Imagine a recurrent neural network is unfolded on a 100 different time steps and expects the derivative to be computed at an initial

state or a 0 time step. To do that, a multiplication or backpropagation must be done all the way back from 100 to 0 as in Eq. (3.6) such that the appropriate weight matrix is captured.

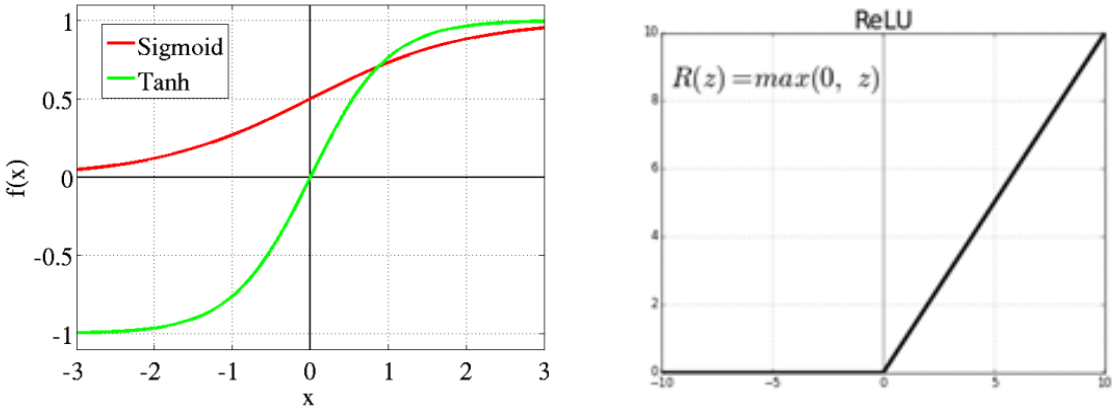
$$\frac{\partial h_t}{\partial \Theta_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial \Theta_{i-1}} = \prod_{i=k+1}^{t} \Theta^T \ \text{diag}[\phi'(h_{i-1})] \tag{3.6}$$

where $\Theta^T$ came from taking the derivative of $h_t$ with respect to $h_{t-1}$

Then the magnitude $\Theta^T$ is scaled across the steps alongside the size of weight matrices, which further compounded in many times, hence, incorrect steps are inevitable. This however is the vanishing gradient issue experienced on training RNN especially on large samples of data. At first, research tried to resolve vanishing gradients with activation functions and realized its effect is minimal especially as data grows to high magnitude.

3.3.1.2 Advantages of Activation Functions.

The importance of activation functions as shown in Figure (3.7a) for tanh and (3.7b) for ReLu detects input from negative infinity to positive infinity and then squashes it from -1 to +1; 0 in the case of sigmoid or 0 to 1 as in ReLu. This process helps in clipping the output to prevent the exploding gradients but those processes do not do much, rather it helps the network to improve the magnitude of weights as in the description seen in [53, 55, 57, 58].




(a) Sigmoid/Tanh transfer function (b) ReLU Transfer Function

Figure 3. 7: Activation functions implemented in RNN

To diagnose and solve these issues especially of exploding gradients, methods by [57] and [59] are proposed. The method uses normalization of the gradient vector and uses an optimization algorithm with a second-order derivative such as ***RMSprop*** to adjust the learning rate such that the cost can be controlled. These methods require gradient clippings at certain threshold prior to explosions. In addition, [19, 58] discovered that the method is predominantly data sample dependent and as such implemented in speech recognition or cases of machine translation of adaptive learning rate algorithm to adjust the vanishing gradients, which proved insufficient on other sequence problems as reported by [60].

Another aspect of explosive or vanishing gradient control requires truncating and back propagating certain chunks of time steps at a given rate. This method appears to be a smart approach but suffers temporal context beyond the level at which the backpropagation is truncated. This process is termed truncated backpropagation through time (BPTT) from where error propagation is recursively computed as in Eq. (3.7) below.

$$\frac{\partial C_t}{\partial \theta_k} \alpha \, |\theta_R|^T \left|\frac{\partial_g}{\partial_a}\right|^T \tag{3.7}$$

Other methods that have proven realistic in clipping RNN gradient is using rectifiers that have no zero gradient units [57]. These rectifiers are for example the rectified linear units (ReLU) of Figure 3.7b, which have a derivative of one on a positive output. This means, no multiplication for activities that are larger or smaller than one. Hence, [25] explore **RMSProp** to control exploding gradients, where diminishing gradients can also adaptively adjust learning rates, which in turn depends on the size of the gradients[17], thereby making the system uncontrollable.

[17] The growth experienced in the gradient curses uncontrollable explosion.

However, for effective control of these vanishing gradients; one of the most reliable approaches results in applying sophisticated architectures, designed specifically to combat vanishing gradient issues in RNN. These architectures lead to the concepts of long short temporal memory (LSTM) and the gated recurrent units (GRU)

3.3.2. Long Short-Term Memory (LSTM).

Reference [61], propose the long short-term memory (LSTM) recurrent neural architecture. This architecture is very simple, at the core with a memory cell Ċ that has a recurrent weight to itself. The architecture when modified solved the issues of vanishing gradients problems especially as the sequence is moved forward in time, the activity of the memory cell inherits the activity of the previous time step – in that case, this unfolds the gradient as in Figure 3.8. It is imperative to note that studies from the noted literature states that the vanishing gradient is not achieved, completely, since the architecture also depends on the gradient data size.

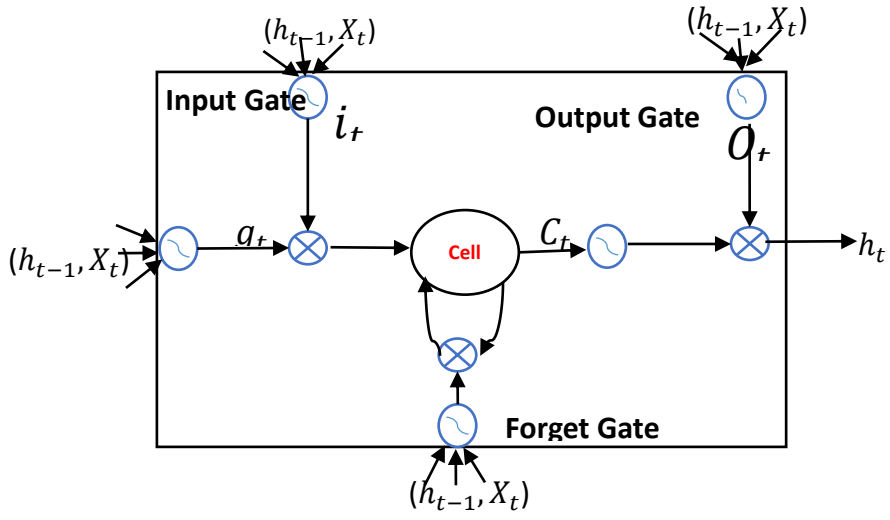



Figure 3. 8: Long Short-Term Memory (LSTM) architecture.

Dealing with this situation involves taking the memory cell and adding operations by simultaneously flushing the memory such that samples are added and retrieved from a state at the same time in a manner like a conveyor belt that keeps the memory intact from one time

step to another [24, 25, 31]. This process is repeated such that the memory learns the process in time, but the issues of gradient explosion remain. To solve this, the idea of gating is introduced (see forget gate in Figure 3.8). Here, the size of the memory cell is modified in order to retrieve the output. From the figure, the transfer function (sigmoid, tanh, ReLU) ensures that the output from forget-gate is between 0 and 1 – leading to gain properties described by [24, 25, 62] that ensure further retrieval from the forget layer. In addition, this process results in Eq. (3.8) where $\Theta$ represents element-wise multiplication and $h_t$ becomes the output from the memory cell.

$$
\begin{aligned}
f_t &= \sigma_g(\Theta_{xf}x_t + \Theta_{hf}h_{t-1} + b_f)\\
i_t &= \sigma_g(\Theta_{xi}x_t + \Theta_{hi} + b_i)\\
o_t &= \sigma_g(\Theta_{xo}x_t + \Theta_{ho}h_{t-1} + b_o)\\
g_t &= \mathrm{Tanh}\,(\Theta_{xg}x_t + \Theta_{hg}h_{t-1} + b_g)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t\\
h_t &= o_t \odot \mathrm{Tanh}\,(c_t)
\end{aligned}
\qquad (3.8)
$$

The retrieval process, however, ensures best state-step movement from the memory cells. In Figure (3.8) the squashing function further ensures a 0 and 1 output of each gate – $f_t$,$i_t$ , $o_t$, and $h_t$ (see the nomenclature page) for the meaning of each gating parameter. It is worthy noting that at this point the process experiences vanishing gradients until a similar approach is repeated for a forgetting process although with a different weight matrix $w_f$, which yields another output between 0 and 1. In the output, the multiplication is repeated on the present memory cell to another until the cell becomes zeros where the memory is flushed completely. Otherwise, the memory cell is retained for another time step $f_t$. The process is incomplete since data is written to the components in a recurrent fashion, which in turn generates a new proposed input into the memory cell as in $g_t$ of Eq. (3.8).

The cell state in the structure modulates the proposed input and then writes it into the memory cell. Studying the final stage of the structure, however it is interesting to note that the generated Eq. (3.8) is equivalent to how much the network intends to forget the proposed input multiplied

by how much that is to accept the new proposed input. This process, however, forms the core of an LSTM network.

It is important to note that the vectors – output gate, input gate, and the forget gate, each share element-wise multiplication between 0 and 1 which makes the manipulation easy to perform a task in a one-hot encoding scenario.

3.3.3. Gated Recurrent Unit (GRU)

This model is relatively popular for its simplicity. However, [60, 63] proposed the gated recurrent unit (GRU). The model tends to deal with the vanishing gradient problem by making each recurrent unit capture adaptive dependencies at different time scales.

In addition, compressing the different gate to an updated gate as depicted in Figure 3.9. From the model, it is seen that the input model $h_t$ proposes with an output gate z to obtain a representation of the next time step. $Z_t$ is a gate with a 0 and 1 element. The remember gate remembers how much the previous time step representation impacts newly proposed input.

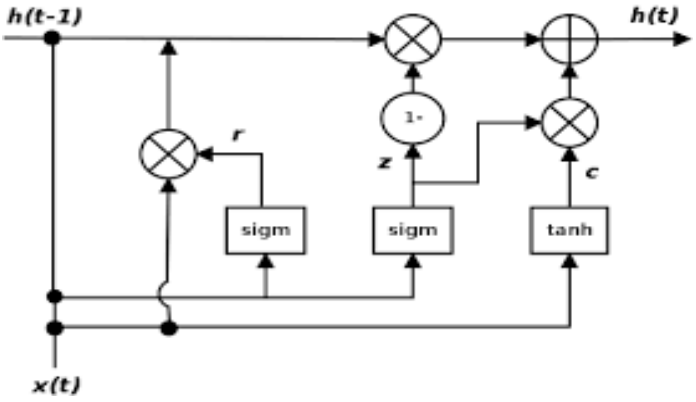



Figure 3. 9: Gated Recurrent Unit
Source: safari online books[18]

In order for $h_t$ to be proposed, the weight matrix is multiplied by the input at $x_t$ remembering the activity at a previous time step $x_{t-1}$, which is further multiplied by how much it is to be

[18] https://www.safaribooksonline.com/library/view/deep-learning
Date Accessed: 10 August 2017

remembered from that time step $r_t$. A new update for the next time step $h_t$ is obtained as $h_{t-1}$ plus the proposed new input $h_t$ modulated by the updated gate $z_t$. Like LSTM but much more simplified. However, in terms of performance, LSTM performs better due to the regularisation acceptance especially over a long sequence and that is why it is considered in this research.

## 3.4. Recurrent Neural Network Feature Description.

The recurrent neural network has experienced major improvements in areas of language modelling, text, speech recognition and sequence-to-sequence or time series modelling. Research work in [24, 27, 31] has the LSTM improvement using the dropout technique to model speech and recorded major improvement of RMSE of 13% over the traditional Hidden Markov models. RNN does this by a simple analogy depicted in Figure 3.10, having a representation, coloured in green, which represents different time steps while the grey counterpart represents different inputs.

Imagine a scenario having alphabets of different letters fed into a network. At first, the letter 'l' is input into the network; the network is required to predict what the following character would be. In our example, the network emits a four-long vector for each element in the alphabet that sums up to one.

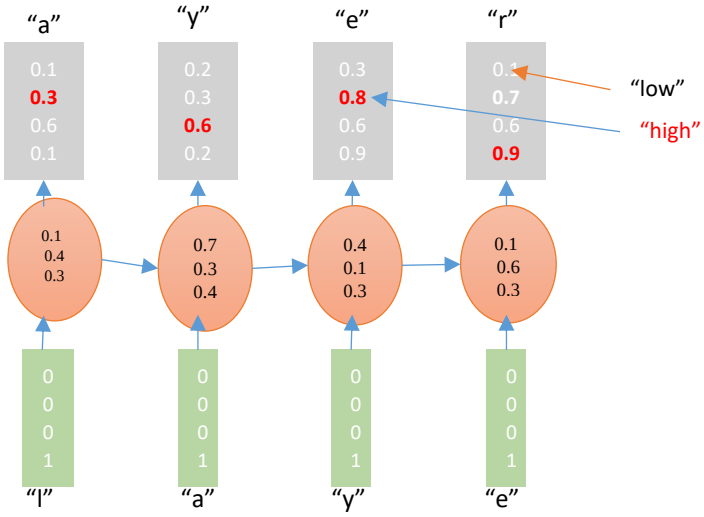



Figure 3. 10**:** Scenario synthesis (case study 1)

A probability of a zero being a certain character given a character it has seen before; the model would output an ‘a’. Inputting more characters in the model obtains the model's prediction as shown in the Figure 3.10. The nature of the process is that characters that are being introduced are the input ‘l, a, y, e, r’ and the model outputs the same input but shifted in time. However, the advantage is that it is shifted forward in time such that at zero time step, it is seen n, expecting the next character ‘a’ given the history of characters that are before it, $as\ in\ P(l_t|\{l_{t-1}, l_{t-2,} \dots, l_0\})$. This example however is a typical language model.

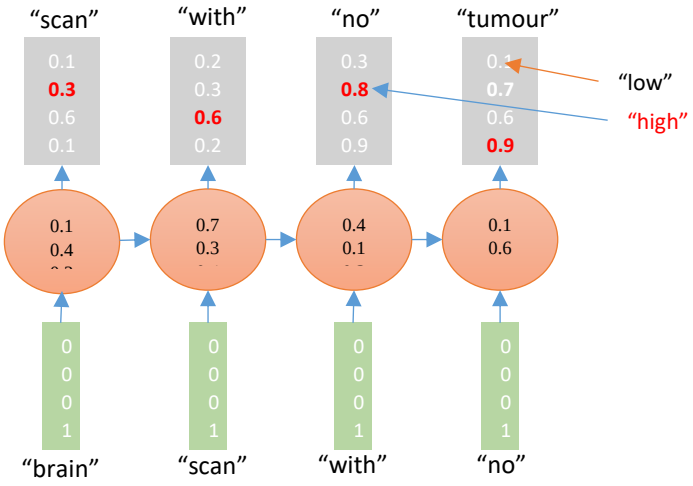



Figure 3. 11**:** Scenario synthesis (case study 2)

The model can be extended to use not just words but models like wind speed, medical data and so on such that a first input of brain, would expect the model to predict the most likely next word scan of Figure 3.11 to have an output – ‘*brain scan with no tumour’.* These words being predicted in Figure 3.10 and 3.11 are typically defined in neural networks as features.

Features commonly applied in tasks similar to this thesis were reviewed in chapter one. It is a common practice that different research groups use different examples to represent features; hence, the underlying characteristic is usually the same. RNN time series models keep track of sequence patterns in the form of *samples, time steps and features* for LSTM hidden layer implementation. The inspiration as discussed in chapter one [28, 31-33] has sequential

modelling introduced to mimic time series sequences applying dropout on LSTM to forecast sequence generation in wind speed, which is similar to speech recognition, handwriting recognition and machine translation. Furthermore, the thesis exploits the concept of leveraging mid-level RNN representation in LSTM [27] inferred in image label annotation.

In a similar vein, the review of wind speed prediction for energy consumption for wind power in the household using LSTM as reported by [23, 34] was further investigated. Although as seen in chapter one [23] with no improvement on LSTM, effective learning of measured energy consumption profile was reported. However, in a typical time series scenario similar to wind speed prediction [29], dropout is implemented to improve LSTM to forecast the risk of student dropout in a massive online course platform. Time series sequential models require effective regularisation for best forecast especially in multivariate cases to combat the collinearity experience, common in multi-level regression problems.

## 3.5. RNN Regularisation Modelling Methods.

This is the process of controlling perfect learning experience observed in long short-term memory architecture for model performance. Regularisation is vital since flexibility makes the LSTM component prone to overfitting. As described in chapter one, LSTM regularisation has seen early stopping using activation functions, the thesis relies on weight noise addition in the form of jitter during training. A different approach that is not discussed in this thesis has a different scenario of application with the underlying similarity in adding jitter once per training sequence. This process reduces the amount of information required to transfer parameters for generalization control. For better understanding about regularisation, let's assume fitting an RNN that overfitting with a cost function J(Θ) as in Eq. 3.9

$$J\left(\omega^{[i]}, b^{[i]}\right) = \frac{1}{m}\sum_{l=m}^{m} \text{loss}\left(\breve{y}^{[i]}, y^{[i]}\right) + \frac{\lambda}{2m}\sum_{l=1}^{l}\left\|\omega^{[l]}\right\|_F^2 \quad (3.9)$$

The extra term $\frac{\lambda}{2m}\sum_{l=1}^{l}\left\|\omega^{[l]}\right\|_F^2$ penalises the weight term $\omega^{[l]}$ from being too large. The $\lambda$ term sets the weight matrices $\omega^{[i]}$ to be close to 0 while $F$ the fabiniouse norm. This scenario makes the neural network more simplified. On the other hand, this method takes overfitting towards the high bias.

Another method of regularisation seen in recurrent neural network (RNN) called gradient clipping, discussed in section 1.2 utilizes clipping of gradient to prevent overfitting as shown in Figure 3.12, described by Equation 3.10.

$$g(z) = \tanh(z) \tag{3.10}$$

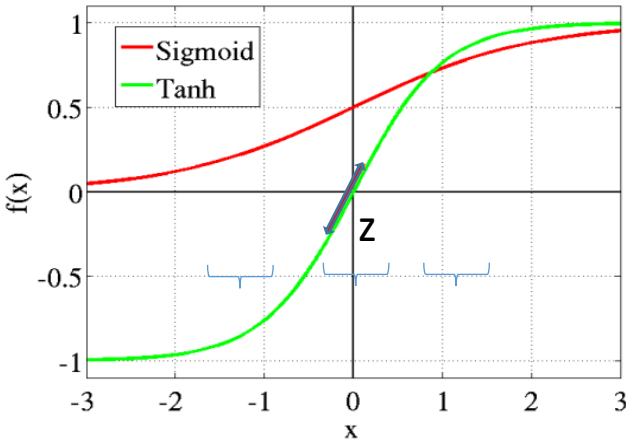



Figure 3.12: Clipping Gradient Regularisation Method

This method states that as far as z is within a small range of parameter as depicted in the Figure…, regularisation model uses a linear regression regime of $\tanh(z)$ to replace $\frac{\lambda}{2m}\sum_{l=1}^{l}\left\|\omega^{[l]}\right\|_F^2$ in Eq. 3.10 therefore, if $\lambda$ is large, the $\omega^{[l]}$ will be relatively small because they are penalised in the J(Θ) looking at Eq. 3.11

$$z^L = \omega^{[L]} * a^{[L-i)} + b^L \tag{3.11}$$

This analogy infers that even a deep network would appear to be linear and overfitting is likely to be prevented since it would form a straight line in the function. The downside of the method is data size.

### 3.5.1. L1L2 Regularisation

In the sequence prediction discussed in the literature, one of the methods used in improving LSTM performance is weight regularisation. This method is simply the application of L1L2 (see section 1.3.2 for L1, L1 explanation) constraints on weights within LSTM nodes – input, hidden and output layer, to reduce overfitting. Research in [24] mathematically resolves the idea in Eq. (3.12 – 3.18) below,

Recall that in logistic regression, the cost function J(Θ) is expected to be minimized as shown in Eq. 3.12

$$J(\omega, b) = \frac{1}{m}\sum_{l=m}^{m} \text{loss}\left(\check{y}^{[i]}, y^{[i]}\right) + \frac{\lambda}{2m}\sum_{l=1}^{l}\|\omega\|_2^2 \tag{3.12}$$

Here, ʎ is the regularization parameter and

$$\|\omega\|_2^2 = \sum_{j=1}^{nx} \omega_j^2 = \omega^T\omega \tag{3.13}$$

where Eq. 3.13 is the Euclidean Norm of the parameter vector and is called the L2 regularisation on logistic regressions.

L1 is similar to L2 but the difference is in $\frac{\lambda}{m}\sum_{l=1}^{nx}|\omega|$ term, which is equal, to $\frac{\lambda}{m}\|\omega\|_1$ which made the $\omega$ in L1 to be sparse, in order words, having more zeros in the model that helps in model compression.

In neural network however, regularisation implementation is different since it considers element-wise multiplication in its activation functions as described in Eq. (3.14).

$$J\left(\omega^{[i]}, b^{[i]}, \ldots, \omega^{[l]}, b^{[l]}\right) = \frac{1}{m}\sum_{i=m}^{m} \text{loss}\left(\check{y}^{[i]}, y^{[i]}\right) + \frac{\lambda}{2m}\sum_{l=1}^{l}\left\|\omega^{[l]}\right\|_F^2 \tag{3.14}$$

were, $\left\|\omega^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[n-1]}}.\sum_{j=1}^{n^{[l]}}(\omega_{ij}^{[l]})^2$ and $l-1\ and\ l$ are the number of hidden units in layer $n-l$. The matric norm is the Frobenius norm of the matrices.

During training of this neural network, on back propagation and to implement optimisation,

$$\partial\omega^{[l]} = [\text{from backprop}] + \frac{\lambda}{\text{m}} + \omega^{[l]} \quad (3.15)$$

Where

$$\omega^{[l]} := \omega^{[l]} - \delta\,\partial\omega^{[l]} \quad (3.16)$$

$\delta$ is the learning rate and Eq. (3.16) is the L2 regularisation to the neural network, which is called weight decay.

Plugging Eq. 3.15 into Eq. (3.16), we have

$$\omega^{[l]} := \omega^{[l]} - \delta[\text{from back prop}] + \frac{\lambda}{\text{m}}\omega^{[l]} \quad (3.17)$$

Which is equivalent to Eq. 3.18

$$\omega^{[l]} := \omega^{[l]} - \delta\frac{\lambda}{\text{m}}\omega^{[l]} - \delta[\text{from back prop}] \quad (3.18)$$

Therefore, the $\omega^{[l]} - \delta\frac{\lambda}{m}\omega^{[l]}$ term shows that whatever $\omega^{[l]}$ is, the regularisation makes the model small since the matric $\omega^{[l]}$ is multiplied by $1 - \delta\frac{\lambda}{m}\omega^{[l]}$ hence reducing model complexity in neural networks.

### 3.5.2 Dropout Regularisation.

The proposed method [64] for correcting weight values due to over-adaptation that causes diminishing accuracy on new samples while training artificial neural networks (ANN) is described. Researchers in various deep learning areas especially image and visual recognitions [65-67] have applied the technique to solve various complex learning challenges in relation to overfitting and under-fitting. To achieve this concept, a mathematical relationship described in Eq. (3.13) [68] applied Bernoulli random variable $\delta_i$ to randomly remove neuron from a neural network using description of Eq. (3.19). Here, the probability $P(\delta_i = 0) = q_i$ is assumed to be independent from each other, however, if $P(\delta i = 1) = 1 - q_i = p_i$ This

however, forms linearity property that is applied to the expectation of the output of the neuron such that, at Eq. (3.19) modified to 14, sums the derivative of Eq. (3.20):

$$E[y^{(i)}] = \sum_{k=1}^{n} w_k \odot x_k^{(i)} E[\delta_k] < bE[\delta_k] \tag{3.19}$$

$$= \sum_{k=1}^{n} w_k \odot x_k^{(i)} p_k < bp_b \tag{3.20}$$

where $w_k$ is the weight-vector and $x_k^{(i)}$ is the neural shape parameter. At IID, the q becomes associated to the random number generator that ensures the shape of the network is kept even at every iteration while p is the probability of keeping a neuron at random. Therefore, during training Eq. (3.13) is applied to train individual nodes of a RNN. However, simplifying Eq. (3.19) results in **dropout** Eq. (3.21). During training, the backpropagation is associated to $p_i$ which is element wise multiplied ($\odot$) by the weight parameters $w_k$ of the reduced node to present a zero-out neuron by reducing co-adaptation among the neurons. This scenario results in an LSTM network that is insensitive to specific neuron weights at the nodes, thereby influencing better generalization with relatively less likelihood for overfitting training data. To address the issues at low testing time, the scale factor is **inverted** in a form as $\frac{1}{1-p} = \frac{1}{q}$, subject to Eq. 3.21.

$$E[y^{(i)}] = \frac{1}{q} \cdot [\sum_{k=1}^{n} w_k \odot x_k^{(i)} q < b] \tag{3.21}$$

The equation above results in the concept of **inverted dropout** that further results in the test time being untouched, while effectively reducing training time but improves generalization irrespective of the LSTM neural configuration. The Python code implementation of inverted dropout is further discussed in the Appendix A (xiii). From Eq. (3.14), $x_k^{(i)}$ will be reduced by 50% (ie if the p = 0.5), meaning 50% of $x_k^{(i)}$ will be zeroed out. However, in order not to reduce the expected value of the network $E[y^{(i)}$, $x_k^{(i)}$ is divide by $q$ such that the remaining $x_k^{(i)}$ would be bumped back up by the required 50% thereby not influencing the generalisation of expected value as in Eq. 3.19.

Another advantage of the inverted dropout is that no matter the value $q$ is set, $x_k^{(i)}$ remains unchanged.

3.6. Performance in Wind Power Predictions.

In regression problems, one of the major processes is a measure of algorithm performance when fitted to a model, in other words, training. This measure is with respect to error reduction during the training process. This implies that the increase in error reduces algorithm adaptability on data samples, which in turn reduces performance on unseen (test) data, after training. In addition, the measure is scale dependent and has the ability to compare forecasting errors of different models over a particular data sample, hence not between datasets.

3.6.1 Root Mean Square Error.

In sequence prediction, the value of the predicted model is measured by how it performs in understanding or training sample data. Research reported by [4, 18, 53, 69, 70] measures the performance of an ARIMA model over several training samples using RMSE as shown in Eq. (3.17); MSE, MAE performance measures metrics for a univariate system. In a bid to compute the performance of different algorithms – SVR, ARIMA, PAR and EDM over a sample set of 650 samples, [53] employed RMSE, MSE, nRMSE to measure the independent performance of the algorithms. On the other hand, [70] presented RMSE as the best metrics and further reported that SVR is the best algorithm that learns the behaviour and pattern of the samples.

$$\text{RMSE} = \sqrt{\frac{\sum_{t-t}^{T}(P_t - y_t)^2}{n}} \tag{3.22}$$

Depending on datasets and problems, RMSE and MSE appear to be the most used metrics in for time series and sequence prediction. This is due to the effect on unseen data as RMSE in proportional to the size of the squared error. This means that larger variations of errors have a disproportionately large effect on the square root of the error. However, the consequence of squaring this error is the sensitivity to outliers.

In a regression problem shown in Figure 3.13 RMSE is calculated as Eq. (3.23), which means that for a predicted value $y_t$ of a time sequence t of dependent regression variable $P_t$ observed over T times and computed in a T-variable.

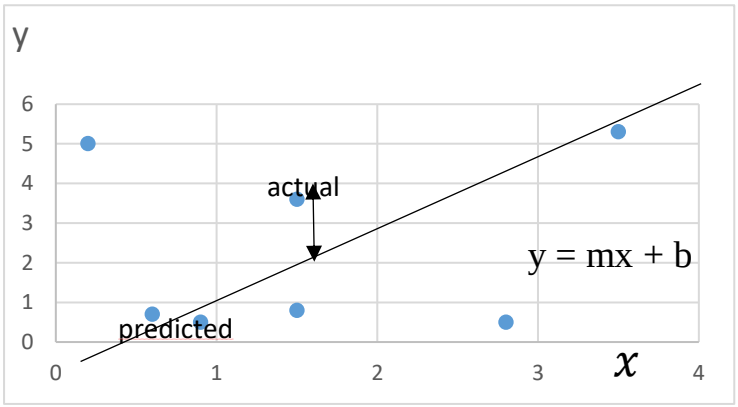



Figure 3. 12**:** Sample of error measurement.

In a cross-sectional sequence of data, *t* is described as *i*. While T is seen as n. RMSE is simply used to compare differences between two variables in some discipline. In an unbiased estimator, the RMSE is also described as the square root of the variance, which in turn is known as the standard deviation. RMSE is of different variants in terms of use. Other performance metrics for time series and sequence forecasting are the nRMSE, MAE, MSE and Mean absolute percentage error (MAPE) as discussed below.

### 3.6.2. Normalized RMSE (nRMSE)

This is applied when data is of different scales, although in the literature, research has not recorded consistent means of normalization. This means that the mean and range of RMSE becomes a common choice for normalizing root mean squared error. The equation of nRMSE shown in Eq. (3.23) is expressed as a percentage metric having a high value that indicates high residual variance. The metrics are dependent on sample size for better comparison.

$$\text{nRMSE} = \frac{RMSE}{(y_{max} - y_{min})} \tag{3.23}$$

where $y_{max}$ and $y_{min}$ are the maximum value and minimum value of the sample data respectively.

### 3.6.3. Mean Absolute Error (MAE)

This is described as the absolute average difference between two variables having the X and Y as in Figure 3.12 fundamentally, MAE is easier to understand than RMSE due to its interpretability. From Figure 3.12, MAE shown in Eq. (3.24) is the average vertical distance between each point, in order words; the average is in the form

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} \tag{3.24}$$

where $y_i$ is the predicted variable and $x_i$ is the actual variable.

### 3.6.4. Mean Absolute Percentage Error (MAPE)

This metric, represented by Eq. (3.25) is similar to RMSE; MAPE has the absolute value summed for every forecasted point in time divided by the number of fitted points, n. the factor multiplied by 100 made it a percentage error.

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - H_t}{H_t}\right| \tag{3.25}$$

From the Eq. (3.20) above, $A_t$ represents the actual value, while $H_t$ is the predicted value. The difference between $A_t$ and $H_t$ is however divided by actual value $H_t$. Research employing MAPE experiences drawbacks; it suffers poor generalisations on zero values or missing values. In addition, MAPE performs poorly on predictions that cannot exceed 100% for forecasts that are too low and fails forecasts, which are too high. In order words, no upper limit to percentage error. Hence, the reasons why MAPE is not statistically accurate when compared to other metrics such as RMSE, MAE, nRMSE, etc.

3.6.5. Mean Square Error (MSE)

This metric measures the average of squares of errors and is often, addressed as a risk function, which corresponds to the expected value of the quadratic loss. The difference occurs because of randomness and is controlled by the square root term as in RMSE. Furthermore, in MSE metrics, values closer to zero are better estimators as shown in Eq. (3.26)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(P_i - y_i)^2 \quad (3.26)$$

# Chapter 4.

## Wind-Farm Power Prediction Methodology

This Chapter presents various models and steps to achieve the research objective. It studies the mathematical literature underlying the principles of wind speed prediction for generated wind power outputs and its logical relationships to artificial neural network (ANN) especially for wind-farm data analysis and prediction. New models formulated where necessary. Data collection and statistical models for wind data analysis, which involves the utilization of empirical correlations and standard probability distributions to estimate the parameters necessary to develop the output wind-power operation process.

One of the major focuses of wind research is to understand the relationship between wind power statistical distributions and atmospheric variables. These variables are turbulence, wind speed (see section 3.1.2) that is dependent on a broad range of temporal and spatial scales in a geographic area as shown in Figure 4.1. The distributions on the other hand are the Weibull, Rayleigh, and the wind power density. Verification of numerical models by fieldwork and statistical analysis enhance understanding of atmospheric variables both in horizontal and vertical landmasses above earth or sea surfaces. Operators and wind developers however rely on high-resolution remote sensing computer simulation to provide useful prediction, which in turn enables them to select and operate, wind farm sites efficiently. The high resolution computer simulation are built on sophisticated models such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), hidden Markov models, (HMM) [71-73] and so on, to understand wind complexities and atmospheric instabilities. Therefore, understanding wind dynamics by improving accuracy of wind prediction is critical to grid operators to balance

power generation by decreasing or increasing production from other sources – biomass, geothermal, hydroelectricity, natural gases (coal), etc.

4.1. Wind Farm Power Output Prediction.

The atmospheric instability especially on wind speed and turbulence affects the amount of power extraction in wind turbines and turbines' lifespan components. Remote sensing instruments such as Lidar and sonar has been used to provide vertical wind profiles – wind speed, wind direction and turbulence in the lower layer of the atmosphere. These data are collected in the location as described by [74]. Sonia Wharton, an atmospheric scientist explained in Figure 4.1 that wind turbines operates in the first 150 meters of the 1-kilometer-high atmospheric boundary layer which is adjacent to ground surfaces [75].

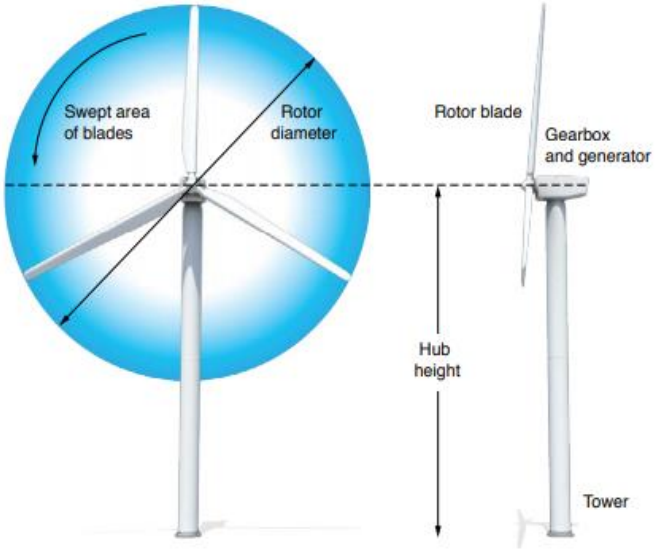



Figure 4. 1**:** Wind-power generation dynamics
Source: Lawrence Livermore National Laboratory.[19]

These layers in addition experiences significant heat exchange during daytime (see section 3.1.2) between the atmosphere and the surface, which in turn induces turbulence. Turbulence

[19] Accessed 26 February 2018

in turn induces friction from moving wind through trees, hills and buildings. Wind farms over the years has improved with trade-off in turbine hub height – distance from the ground to blade rotor, blade diameter and power generating capacity.

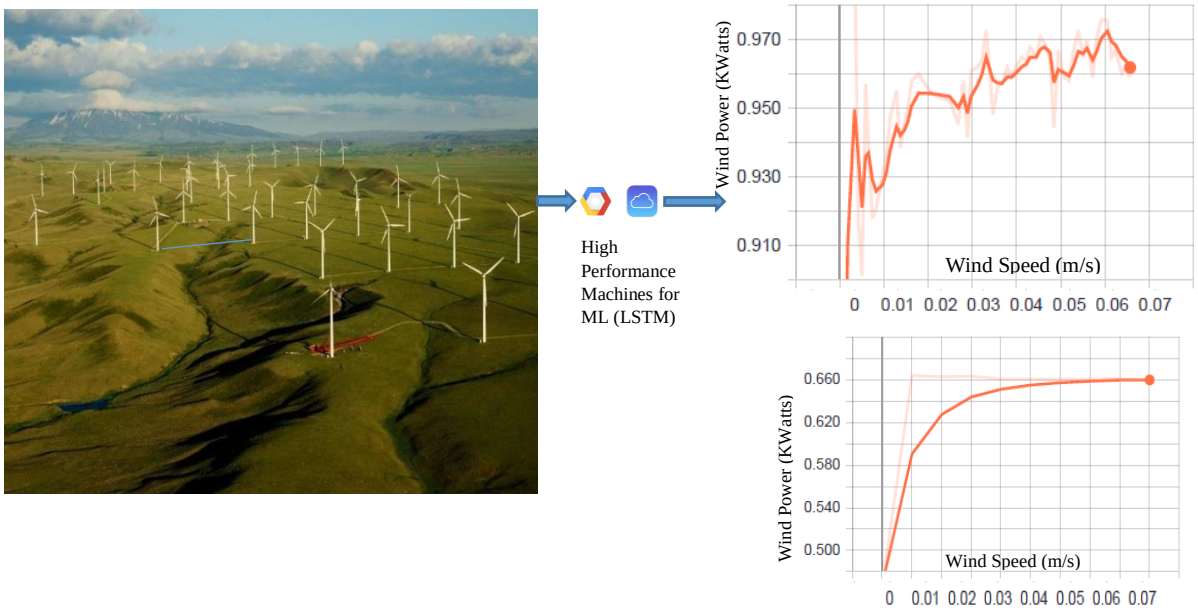



Figure 4. 2**:** Typical Wind Farm. Power Output

Wharton's findings indicates that taller turbines encounters high wind speeds to generate a greater amount of energy, with trade-off on complex airflows, which is driven by turbulent mixing that affects turbine components. Power output, depicted in Figure 4.2 on the other hand depend on average wind speed and blade swept area. Conversely, wind power from Eq. (4.1) is proportional to the cube of wind speed. Therefore effective prediction of wind speed, enhance better power output estimation from a wind farm.

4.2. Wind-farm Power Output Predictions Parameters.

Wind power modelling over the years has relied on the traditional power curves of Figure 4.3 to model wind power. These curves models wind power as a function of wind speed at hub height of various turbines, hence, adjusted for air density. In the literature, the intuitive

algorithm used for calculating the induction factors, which can be used to obtain the load on the turbine, which are the force $dF_N$ and torque $dQ$ using Eq. (4.1) and (4.2) respectively [76] used to define wind power as the conversion of atmospheric forecasts into power output from many turbines or a single turbine.

$$dF_N = B * \frac{1}{2} * \rho U_{rel}^2 cdr(C_l cos\theta + C_d sin\theta) \quad (4.1)$$

$$dQ = BrdF_T = B * \frac{1}{2} * \rho U_{rel}^2 crdr(C_l cos\theta - C_d sin\theta) \quad (4.2)$$

From the equation, $U$ is the relative wind and its angle, $\theta$ as it approaches the turbine blade. Once torque is computed, the power generation from the wind becomes the torque multiplied by angular velocity. The rotor disk generates power by the summation of each wind in the blades.

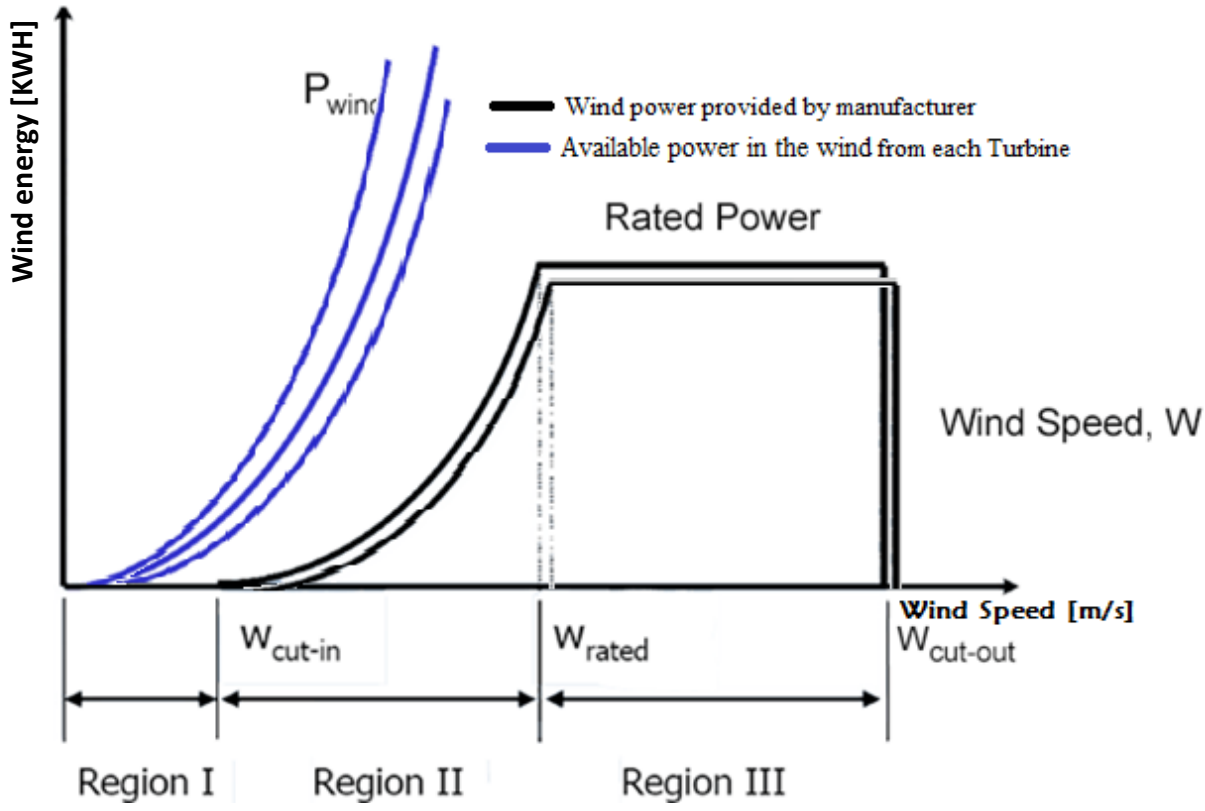



Figure 4. 3: Wind Turbine Power Curve.

Therefore, in any given conditions, the power output from the rotor, is further computed by plotting the generated power output from each turbine in the farm as a function of free-stream wind – $P_{wind}$ whereas turbine manufacturers issue wind power, $P_T$. It is different from the computed wind power from a given site, extracted using the BEM theory. The difference in both wind powers can be small in both region I and III. In region I, the turbine is not generating any power hence the cut in wind speed. Here, either the blade is not rotating or it is rotating without producing any power, estimated to be around 3 – 4 meter per second (m/s) for a conventional large turbine. The rated wind speed ($W_{Rated}$) is when the power output of the generator in the turbine reaches its maximum capacity. At this point, the generated power does not increase with wind speed anymore; this is mainly to protect the turbine and estimated to be about 11 – 15 (m/s). At the cut-out wind speed, which is around 25 m/s, for a typical large turbine, the entire turbine system is shut down to protect the mechanical structure of the turbine. This is because; at such high wind speed, the mechanical stress on the turbine is high, and could damage the turbine.

4.2.1. Power Curve and Wind Speed Histogram

The power curve is purely a characteristic of the wind turbine; it cannot show energy generation from the turbines per year, known as annual energy production (AEP). To compute AEP, the characteristics of the wind is required. These approaches are used to obtain the characteristics of wind in a wind farm and predict power output from the farm. First is using wind speed data collected at the farm, secondly; by using statistical estimation – predefined probability distributions and thirdly, by using velocity duration curve. In the first, anemometer sensors collect wind speed (see section 5.1, Figure 5.2 through 5.3). The Figure depicts the variation of wind as a function of time. Each data point is the wind speed averaged within certain time interval – minutes, hours, days, weekly, etc. The wind speed is however, further subdivided into bins such that the data point that fall within each bin is counted to form Figure 4.4 called

the wind speed histogram. In the histogram, each data point is associated to an average wind speed within a certain time interval to obtain the total amount of time during which wind is blowing at a speed associated to the bin. Using this approach, the AEP or wind speed over a certain period can be obtained for prediction of wind power.

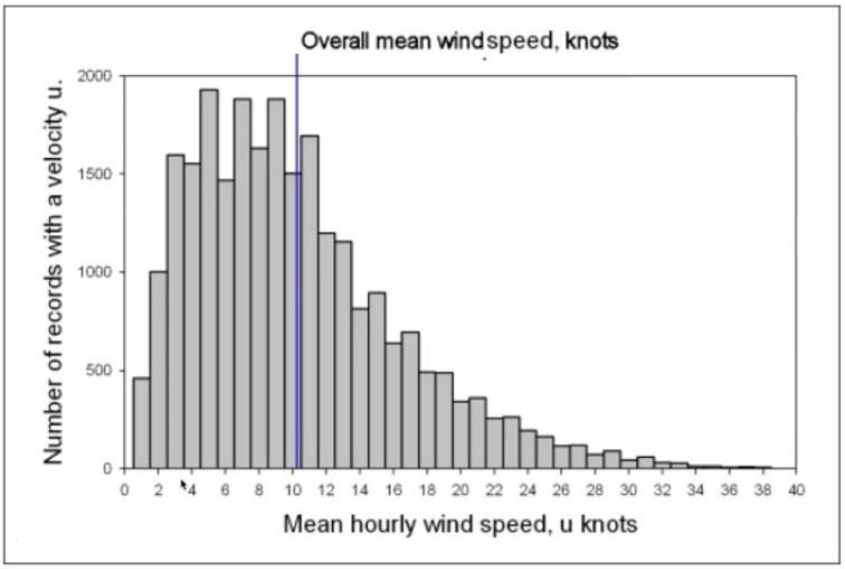



Figure 4. 4: Typical Wind Speed histogram
Source: The Met Office, UK[20]

The second approach, which requires a statistical method, shows that the Weibull distribution can provide a good fit to the wind speed histogram. It has been widely used to approximate wind speed histograms [47, 77]. The probability density function (PDF) has two parameters that allow users to adjust the shape of wind power within the wind farm, as shown in Eq. (4.5). From the equation, c is the scale factor while k is the shape factor. The shape factor controls the location and peak of the distribution, while the width of the function is controlled by the scale factor c, often selected at average wind speed. In addition, the shape factor can be obtained from a nearby wind farm or the scientific community. Once these factors are known the wind speed histogram can be approximated.

[20] Date Accessed: 20th May 2018

4.2.2. Average Energy Production (AEP).

To estimate the AEP using the statistical approach, Eq. (4.3) and (4.4) are used

$$\text{AEP} = 8760 \int_0^{\infty} \mathrm{P_w(U)p_{PDF}(u)dU} \tag{4.3}$$

$$= 8760 \int_0^{\infty} \mathrm{P_w(U)dF_{CDF}(u)dU} \tag{4.4}$$

where $p_{PDF}(u)$ is the PDF that gives the proportion of velocities that occur, that is the number of times per year, 8760 is the number of hours per year and $p_{PDF}(u) = dF_{CDF}(u)$ . Using the integral to replace the summation over the number of bins (NB), the Weibull cumulative distribution is rearranged to obtain AEP using Eq. (3.5)

$$\text{AEP} = \sum_{\mathrm{i=1}}^{\mathrm{NB}} \left\{ \exp[-(\tfrac{\mathrm{U_{i-1}}}{\mathrm{c}})^{\mathrm{k}}] - \exp[-\left(\tfrac{\mathrm{U_i}}{\mathrm{c}}\right)^{\mathrm{k}}] \right\} \mathrm{P_w}(\tfrac{\mathrm{U_{i-1}+U_i}}{2}) \tag{4.5}$$

From the equation above, $U_{i-1}$ becomes $F_{CDF}$ at the left boundary of the bin while $U_i$ is $F_{CDF}$ at the right boundary of the bin. $P_w(U)$ is the power of the turbine which can be obtained from the power curve. The quantity $P_w(\frac{U_{i-1}+U_i}{2})$ represents the power curve at the centre of the bin.

The third and a slightly different approach for computing AEP is the velocity duration curve of Figure 4.5. At each point, the duration in terms of hours for the wind to have a speed at or exceeding a velocity at the point is computed using Eq. (4.6).

$$\text{Velocity Duration Curve} = 8760 * [1 - \mathrm{F_{CDF}(U)}] \tag{4.6}$$

Since the power curve of the wind is proportional to the cube of wind speed, the velocity axis can be converted to the power axis of Figure 4.6.

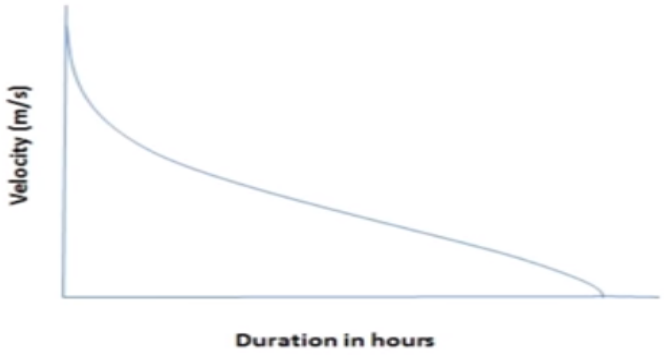



Figure 4. 5: Velocity Duration Curve.
Source: The Met Office, UK[21]

At this point, the power curve of the wind turbine is incorporated such that the velocity in Figure 4.5 is mapped to a power value of individual turbine power curves as depicted in Figure 4.6 where AEP is the area under the power duration curve.

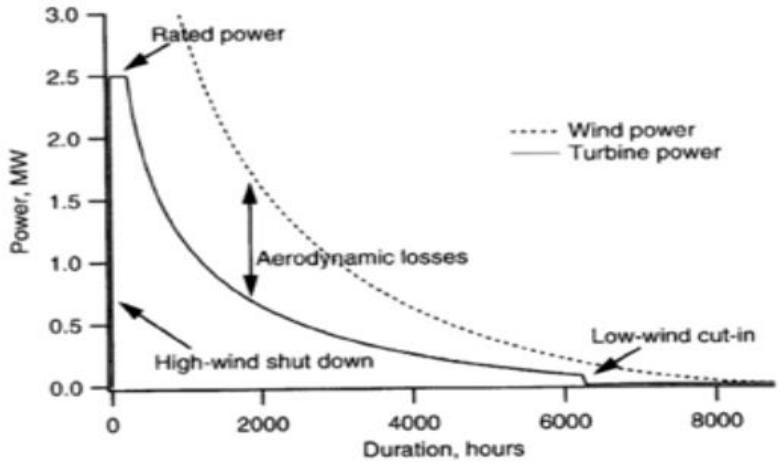



Figure 4. 6: Turbine Power Duration Curve.
Source: The Met Office, UK[22]

4.2.3. Wind Power Density.

This is an indicator, which shows wind resource capacity in a specific site or farm, usually expressed in Watts per square meter ($W/m^2$) and calculated based on available power in the

[21] Date accessed, 15th May 2018.
[22] Date accessed, 17th may 2018.

wind farm applying the Weibull parameters method of Eq. (4.7). This method is adapted from [78]. From the literature, wind power density is generally considered a better indicator of the wind resource than wind speed [47].

$$\frac{P}{A} = \int_0^{\infty} \frac{1}{2}\rho U^3 f(V)dV = \frac{1}{2}\rho c^3 \Gamma\left(\frac{k+3}{k}\right) \quad (4.7)$$

The average wind-power density in terms of wind speed is calculated using Eq. (4.8). This is because wind power is proportional to the cube of wind speed, hence, the root mean cube (rmc) of wind speed, which analytically results in Eq. (4.9)

$$WPD = \frac{\sum_{i=1}^{N} 0.5\rho U_i^3}{N} \quad (4.8)$$

$$U_{rmc} = \sqrt[3]{\frac{1}{N}\sum_{i=1}^{N} U_i^3} \quad (4.9)$$

N is 14, equivalent to the number of Turbines in the Wind Farm.

4.3. Wind Farm Power Output (WFPO) Modelling.

Wind farm power output is not complete without an effective wind speed prediction. The system requires an efficient prediction of wind speed such that the power generation is integrated to the grid for supply and load distributions. Wind farms are generally designed having certain layouts and components in place. These components are the shape, shade, wake, heights and spacing, distance from one turbine to another. The shape determines the structure of the farm. Description of the shapes [79] can be rectangular or triangular and work with the direction of the wind, mainly in a downward or up-ward direction to the turbine. The wake on the other hand describes the wind flow from one turbine to another. Effective wind-farm power output yield is expected to have shade in either the upwind or the downwind direction, wake behind turbines, which is dependent on the height and spacing of turbines within the farm. The

velocity of probability distribution, costs, and revenue generation are also important factors considered in wind farm design.

The aerodynamic interactions within the turbines in a wind farm [80] directly affects wind farm power output prediction. These interactions caused by the wake effect as the incoming wind to the turbine has more energy content than the out-going wind, leading to casting of a wind shadow in downward direction.

To demonstrate wind farm power output prediction from a wind farm, a 14-turbine wind farm is considered in this research, which adapts the design of [81], although it does not consider the wake effect, hence, other configurations are described by [82]. From the layout of Figure 4.7, an **α**, y˝layout of 14 turbines, separated by δ (m) and **ℒ** (m) spacing equivalent to 120 and 300 meters respectively is assumed to provide an Npv of $12.5mil and Ir of 16.7%.

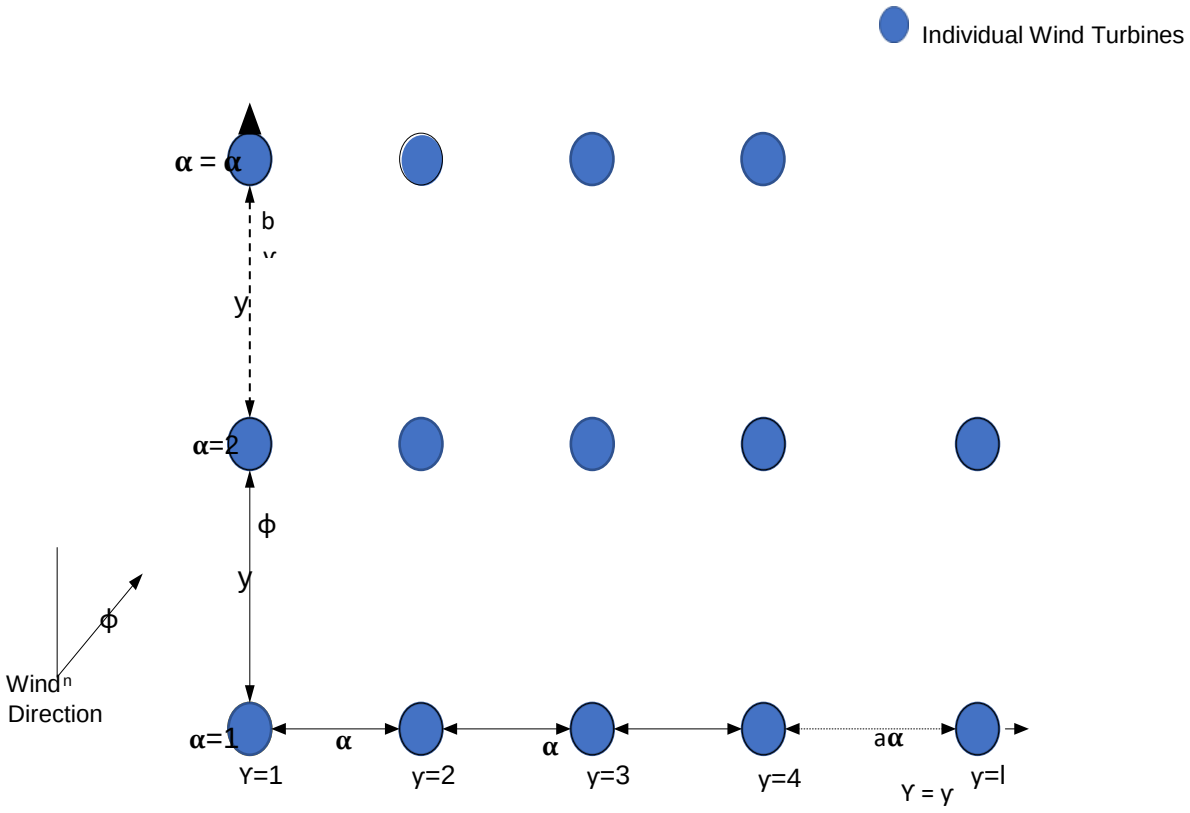



Figure 4. 7**:** Location of turbine **α**, y˝in a rectangular grid layout of **α** X y˝turbines.

From the figure, wind blows in a direction ϕ with a nominal velocity V (ϕ), having effective wind velocity in front of turbine at $\boldsymbol{\alpha}$, y˙position. Here, the aerodynamic interactions between turbines are $V_{\alpha,y}(\phi, \delta)$. Therefore, the electrical power generated by turbine $\boldsymbol{\alpha}$, y˙is as shown in Eq. (4.10), hence,

$$eP_{\alpha,y}(\phi, \delta, \boldsymbol{\mathcal{L}})[\text{Kwh}] = \frac{1}{2}\rho V_{\alpha,y}(\phi, \delta, \mathcal{L})^3 \beta C_p N_m \quad (4.10)$$

where β is the swept rotor area ($m^2$), $\rho$ is the air density, $V_{\alpha,y}$ is the wind velocity of the site, $C_p$ is the rotor coefficient of efficiency or capacity factor, which we assume to be 85%, $N_m$ is the efficiency of converting the rotor mechanical power into electricity; assumed to be 85%. The prediction of wind speed as mentioned above, results in better prediction using either machine learning or statistical approaches.

4.4. Machine Learning Model for WPO Prediction.

The discussion above informs the formulation of correct wind data profiles in a wind farm considering the learning strategy of LSTM to enhance accurate prediction of wind speed. Although LSTM is an improvement to RNN, LSTM from section 5.5 (Figure suffers from overfitting or perfect learning as shown in Figure 5.26 that results in poor generalisation at test time. This effect may slow convergence of the network and increase error. Suggestions from [83] ensures clipping gradients of hidden neurons, which results in local minima problems or instability of the model. To maintain stability and accuracy, dropout is ensemble with LSTM to ensure suitable prediction of WPO. The assembling of these methods is as described in chapter five.

The RMSProp optimisation and mean square error (MSE) performance model is implemented during training the RNN to ensure minimum error is gained from the resulting learning process of wind speed prediction. This criterion is as discussed in Eq. (4.15). However, neural network

training as discussed theoretically in section 2.4, which results in practical modelling use-cases as shown below.

4.4.1 Training and Validation Modelling.

Although section 3.5 discussed NN modelling, one of the purposes of neural network trainings is to minimize the output error. The process of training a neural network involves tuning the values of the weights and biases of the network to optimize network performance, as defined by the network performance function. The default performance function for feed-forward networks is the mean square error (MSE), which typically averaged the squared error between the network outputs and the target outputs.

4.4.2. Network Validation Modelling.

When the training is complete, the network performance is checked to determine if any changes need to be made to the training process, the network architecture or the data sets. The first thing to do is to check the training record. The next step in validating the network is to create a regression plot, which shows the relationship between the outputs of the network and the targets. If the training were perfect, the network outputs and the targets would be exactly equal, but the relationship is rarely perfect in practice.

4.4.3. Training Algorithm Model.

The back-propagation algorithm proved very good for this work, and is used extensively in neural network applications. The network learns a predefined set of input-output sample pairs using a two-phase propagation-adaption cycle as a gradient-based optimization procedure. The training begins with random weights and biases that are adjusted by the chosen algorithm for minimizing errors. Each unit in the hidden layer receives only a portion of the total error signal, based roughly on the relative contribution to the unit made to the original output. This process

repeats layer by layer until each node in the network has received an error signal, which describes its relative contribution to the error. Consequently, each unit causes the network to converge toward a state, which allows all the training set be prearranged before updating connection weights. Different nodes learn how to recognize different features within the input space after training. The usual strategy is to experiment with several algorithms and locate the most suitable one for the given application. Other algorithms are as shown in Table 4.1 below.

Table 4. 1: Neural Network Training Algorithms Table.

| Function | Algorithm |
|---|---|
| Traincgp | Polak-Ribiére Conjugate Gradient |
| Trainoss | One Step Secant |
| Traingdx | Variable Learning Rate Gradient Descent |
| Traingdm | Gradient Descent with Momentum |
| Traingd | Gradient Descent |
| Traincgb | Conjugate Gradient with Powell/BealeRestarts |
| Traincgf | Fletcher-Powell Conjugate Gradient |
| Trainlm | Levenberg-Marquardt |
| Trainbr | Bayesian Regularisation |
| Trainbfg | BFGS Quasi-Newton |
| Trainrp | Resilient Backpropagation |
| Trainscg | Scaled Conjugate Gradient |

**Source: [86]**

4.4.4 Basic Back-Propagation Modelling.

The three-layer network of Figure 4.8 and 4.9 are employed to aid the understanding of the mathematical expressions as described by [84].

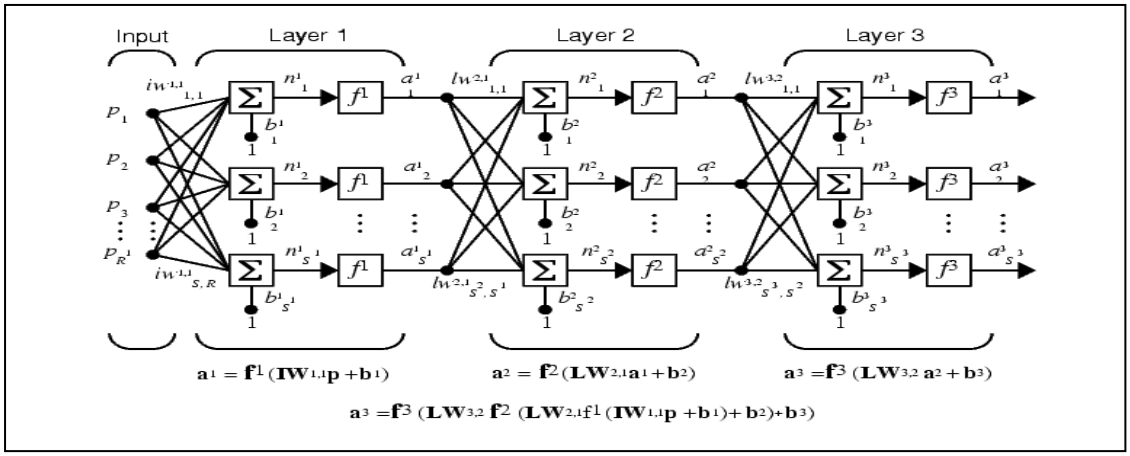



Figure 4. 8**:** Three-Layer Network
Source: [86]

The same three-layer network can be represented using Figure 4.4b

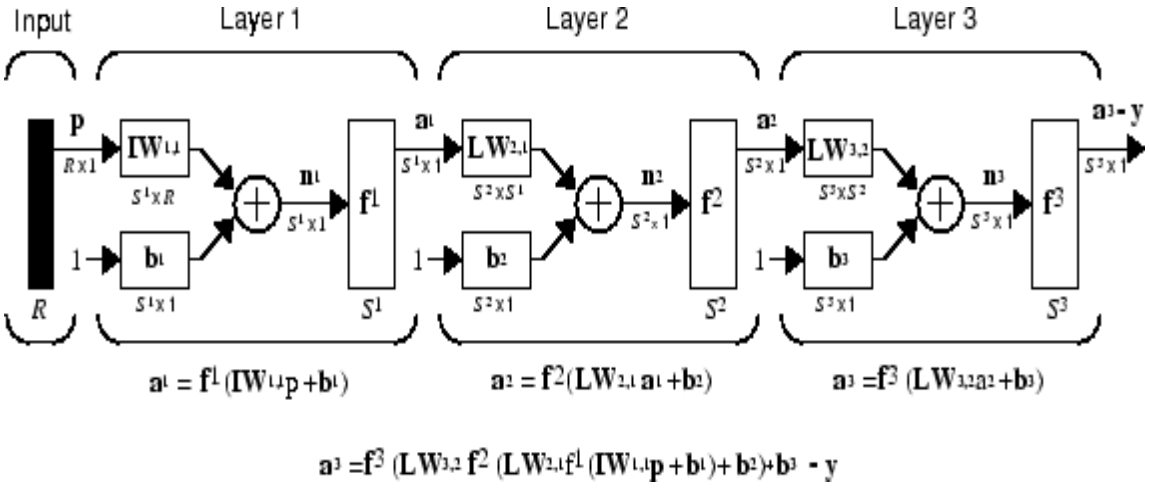



Figure 4. 9**:** Three-Layer Network (Abbreviated Notation)

The models that describe the operation of the multilayer network shown in Figure 4.4 is mathematically depicted in Eq. (4.11)

$$a^{m+1} = f^{m+1}(w^{m+1}a^m + b^{m+1}) \text{ for } m = 0, 1, \cdots, m-1. \quad (4.11)$$

where $m$ is the number of layers in the network. However, the first layer neurons receive external inputs and the outputs of the neurons in the last layer are termed external outputs. This is shown in Eq. (4.12) and (4.13) respectively.

$$a^o = P \tag{4.12}$$

$$a = a^m \tag{4.13}$$

The algorithm is provided with a set of examples of proper network behaviour

$$\{p_1, t_1\}\{p_2, t_2\}, \dots, \{p_q, t_q\} \tag{4.14}$$

where $p_q$ is an input to the network and $t_q$ is the corresponding output. As each input is applied to the network, the network output is compared to the target. The algorithm adjusts the network parameters in order to minimize the performance index, which is the mean square error, defined by Hagan using Eq. (4.15)

$$F(\boldsymbol{X}) = E[e^2] = E[(t-a)^2] \tag{4.15}$$

Here $\boldsymbol{X}$ becomes the vector of network weights and biases. $t$ is the target value and $a$ is the output value and $e$ is the error between the target and the output. Therefore, Eq. (4.16) is applied for multiple layer networks

$$\mathrm{F}(\mathbf{X}) = \mathrm{E}[\mathbf{e}^{\mathbf{T}}\mathbf{e}] = \mathrm{E}[(\mathbf{t}-\mathbf{a})^{\mathrm{T}}(\mathrm{t}-\mathrm{a})] \tag{4.16}$$

4.4.5 Performance Index.

The performance index of a back propagation artificial neural network is the mean square error given by Eq. (4.17):

$$\hat{\mathrm{F}}(\mathbf{X}) = (\mathrm{t}(\mathrm{k}) - \mathrm{a}(\mathrm{k}))^{\mathrm{T}}(\mathrm{t}(\mathrm{k}) - \mathrm{a}(\mathrm{k})) = \mathrm{e}^{\mathrm{T}}(\mathrm{k})\mathrm{e}(\mathrm{k}) \tag{4.17}$$

However, in Eq. (4.18), the sensitivities are propagated backward through the network from the last layer to the first layer in time, that is;

$$S^m \rightarrow S^{m-1} \rightarrow \cdots \rightarrow S^2 \rightarrow S^1 \tag{4.18}$$

where the starting point $S^M$ is obtained at the final layer for network convergence.

4.4.5.1 Convergence

One of the major problems with the back-propagation algorithm has been the long training times. It can take several weeks to train a neural network. This has spurred considerable research on methods to accelerate the convergence of the algorithm. Consequently, some heuristic techniques are now available which include such ideas as varying the learning rate. There are also existing numerical optimization techniques, for example, gradient descent, the conjugate gradient algorithm, the Levenberg-Marquardt (LM) algorithm and RMSprop algorithm, a variation of Newton's method that provides fast convergence in training neural networks such as multilayer recurrent perceptron or RNN. A combination of these algorithms may be necessary to achieve a particular purpose. While the basic concept of some convergence criteria are given in this work, the mathematical expressions underlying each of them can be found in [84]

4.4.5.1.1 Gradient Descent Optimization Description.

Gradient descent is probably the most popular and widely used out of all optimizers. It is a simple and effective method to find the optimum values for the neural network. The objective of all optimizers is to reach the global minima as shown in Figure 4.10 where the cost function attains the least possible value.

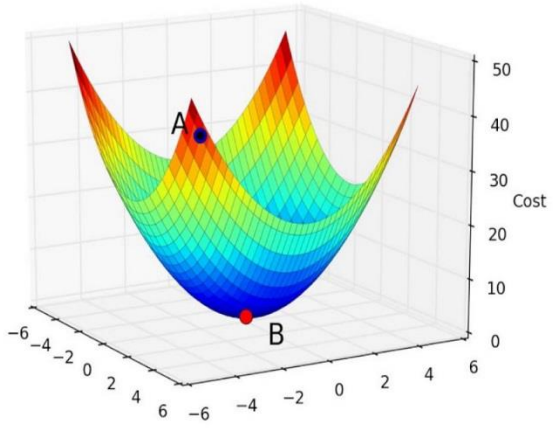



Figure 4.10: Gradient Descent Optimisation Procedures.
Source: Author, Gradient Descent Intuition[23]

Each time we find the gradient and update the values of weights and biases, we move closer to the optimum value. However, prior to the start of neural network training, the cost would be high, which is represented by the point A. Through each iteration of training the neural network – finding gradients and updating the weights and biases, the cost reduces and moves closer to the global minimum value, which is represented, by the point B. The algorithm is as demonstrated below

**Stochastic Gradient Descent - Algorithm**

```
For each example in the data
    - find the value predicted by the recurrent neural network
    - calculate the loss from the loss function
    - find partial derivatives of the loss function, these partial
derivatives produce gradients
    - use the gradients to update the values of weights and biases.
```

4.4.5.1.2 Description of Learning Rate

Learning rate is probably the most important aspect of training neural network. Learning rate restricts oscillation and guides optimizers in understanding weights at every epochs during

[23] https://miro.medium.com/max/
Date accessed, 18th August 2018

neural network training. The analogy depicted in Figure 4.11 explains learning rate. Imagine the cost function as a pit, optimizer will be starting from the top and the objective is to get to the bottom of the pit. Think of learning rate as the steps taken to reach the bottom (global minima) of the pit. If large values were to be chosen, a drastic change to the weights and bias values would result to reaching the bottom. There is also a huge probability of overshooting the global minima (bottom) and end up on the other side of the pit instead of the bottom.

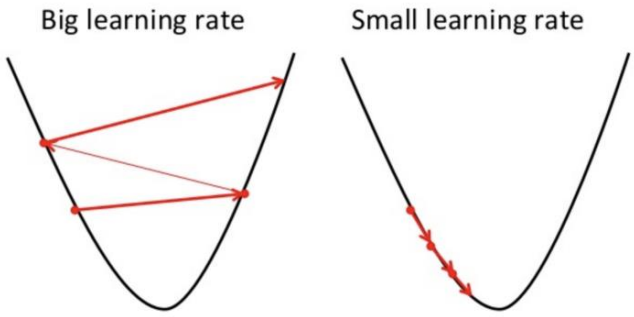



Figure 4.11: Learning Rate Description.
Source: Author, The Gradient Descent Intuision[24]

Hence, with a large learning rate, convergence to the global minima is fast but will always wander around the global minima. On the other hand, if a small value of learning rate is chosen, the optimizer would lose the risk of overshooting the minima but will take longer time to converge, that is, takes shorter steps hence, would have to be trained for a longer period of time. If the cost function is non-convex, the optimizer might easily be trapped in local minima and be unable to get out and converge to the global minima. Therefore, there is no generic right value for learning rate. It comes down to experimentation and intuition.

4.4.5.1.3 Root Mean Square Propagation (RMSprop) Convergence.

[24] https://fromthegenesis.com/wp-content/uploads/2018/06/GDS_3.png
Date accessed, 10th June, 2018

This is one of the robust convergence algorithm used in neural network especially recurrent neural networks. RMSprop follows the central idea behind stochastic gradient descent, which works efficiently on low learning rate by averaging gradients over mini-batches. The convergence algorithm keeps the moving average of the squared gradients for each weight and then divide the gradient by the mean-square square root. This scenario works well on large dataset, hence used in this research. The convergence criteria uses Eq. 4.21.1 for its update rule.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$
$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}}\frac{\delta C}{\delta w} \qquad (4.22.1)$$

From the Eq. 4.21.1, E[g] is the moving average of squared gradients. $\frac{\delta C}{\delta w}$ is the gradient of the cost function with respect to the weights while η is learning rate. β is the moving average parameter.

4.4.5.2 Back-propagation with Momentum (BPM)

The BPM is a modification based on the observation that improves convergence if a low pass filter is used to smooth out the oscillations in the network trajectory. This modification is achieved using Eq. (4.19) and the modified version of Eq. (4.20)

$$\Delta \mathbf{W^m}(\mathrm{k}) = \boldsymbol{\gamma}\Delta \mathbf{W^m}(\mathrm{k}-1) - (1-\boldsymbol{\gamma})\boldsymbol{\alpha}\mathrm{S^m}(\mathbf{a^{m-1}})^{\mathrm{T}} \qquad (4.19)$$

$$\Delta \mathbf{b^m}(\mathrm{k}) = \boldsymbol{\gamma}\Delta \mathbf{b^m}(\mathrm{k}-1) - (1-\boldsymbol{\gamma})\boldsymbol{\alpha}\mathrm{S^m} \qquad (4.20)$$

where

$\mathbf{W}(\mathbf{k})$, is the input to the filter. $\boldsymbol{\gamma}(\mathbf{k})$, is the output of the filter and $\boldsymbol{\gamma}$[25] is the Momentum coefficient that must satisfy $0 \leq \boldsymbol{\gamma} < 1$.

4.4.5.3 Variable Learning Rate Back-Propagation (VLRBP).

In this method, if the squared error over the entire training set increases by more than some set percentage, say $\zeta$, (typically one to five percent) after a weight update as per Eq. (4.20), then the weight update is discarded, the learning rate is multiplied by some factor $P < 1$, and the momentum coefficient $\boldsymbol{\gamma}$ (if it is used) is set to zero.

4.4.5.4 Conjugate Gradient Back-Propagation (CGBP).

The CGBP has the quadratic convergence character. It converges to the minimum of a quadratic function in a minimum number of iterations. It involves interval location and reduction processes. The interval location step helps to find some initial interval that contains a local minimum, while the interval reduction step reduces the size of the initial interval until the minimum is located to the desired error goal – accuracy.

4.4.5.5 Levenberg-Marquardt Back-Propagation (LMBP)

The LMBP algorithm is designed for maximizing functions that are sums of squares of other nonlinear functions. The performance index in neural network training is the mean square error. However, from Eq. (4.21), the mean squared error is proportional to the sum of squared errors over the Q targets in the training set.

$$F(\mathbf{X}) = \sum_{q=1}^{Q} (t_q - a_q)^T (t_q - a_q) \qquad (4.21)$$

The algorithm is assumed to have converged when the sum of the squares have been reduced to some error goal.

[25] Gamma is used here for notation purposes.

### 4.4.2 Stopping Criteria.

The back-propagation algorithm is a first order approximation of the steepest-descent technique in the sense that it depends on the gradient of the instantaneous error surface in weight space [85]. Consequently, weight adjustments terminate under certain circumstances. According to the work of [86], the back-propagation algorithm is deemed to have converged if:

- The Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.
- The absolute rate of change in the average squared error per epoch is sufficiently small.
- The generalization performance is adequate, or when it is apparent that the generalization performance has peaked.

#### 4.4.2.1 General Network Optimization Criterion.

Artificial neural networks are optimized for simulation of the physical behaviour of the system. For a better optimisation, the features of the network requires rigorous manipulation. For instance, selection of training algorithm, number of hidden neurons and weight estimation. An unsatisfactory performance is relative to inadequacy of the selected network configurations. In designing network configuration, the main concern is the number of hidden layers and neurons in each layer. Unfortunately, there is no rule defining this feature and its estimations.

While starting with a small number of neurons and hidden layers, monitoring the performance may help to resolve this problem efficiently. Trial and error procedure is adopted in this research. The optimal training procedure is achieved by using randomly initialized weights and inversion of the training algorithm. This is because many algorithms are subject to trapping in local minima where they are stuck unless certain design criteria are modified in the form of dropout. The existence of local minima is because the error function is the superposition of

nonlinear activation functions that may have minima at various points, which sometimes result in a non-convex error function.

## 4.5 Data Handling

Data preparation is a very important consideration in developing a neural network; it lays the success of any neural network model. Pre-processing steps are considered before feeding a set of raw data to the network to improve the efficiency of the training. This is also useful in analysing the response of the trained network. Thereafter, the data needs to be, divided into subsets – training, testing and validation or training and testing depending on required criterion. Finally, a post-processing step transforms the output of the trained network to its original format to make interpretation of the result possible.

### 4.5.1 Data Pre-Processing (Normalization) and Post-Processing (De-normalization)

It is standard practice to normalize the inputs before applying them to neural network. Several pre-processing routines are available in the literature. For example, [87] presented a normalization model of Eq. (4.22) where a value (p) of the data having the minimum value (p min) and maximum value (p max) is converted into a normalized value ($pn$). The normalized values lie between -1 and +1.

$$pn = \frac{2(p-p_{min})}{p_{max}-p_{min}} - 1 \tag{4.22}$$

Common normalization processes are provided automatically when the network is created and they become part of the network object, so that whenever the network is used, the data coming into the network is pre-processed in the same way [88]. It is easiest to think of the neural network as having a pre-processing block that appears between the input and the first layer of the network and a Post-processing block that appears between the last layer of the network and the output, as shown in Figure 4.12.

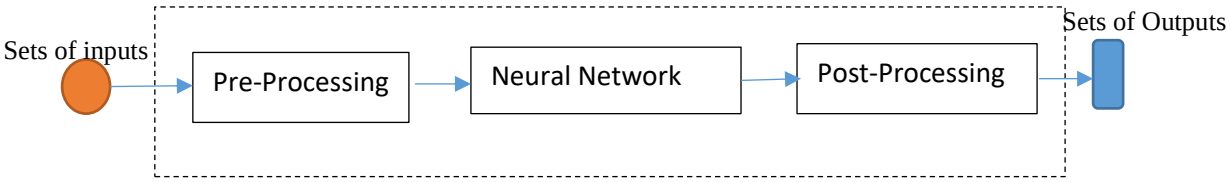



Figure 4. 10: Data Processing Steps.

### 4.5.2 Data Partitioning

Partitioning happens after preparing the data. The general practice is to first, divide the data into three sets: training set – used for computing the gradient and updating the network weights and biases. The validation set used to ensure generalization of the developed network during the training phase and finally the test set used to examine the final performance of the network. The most important consideration here is to ensure that: (1) the training set contains enough data, and suitable data distribution to cover the entire range of data adequately and (2) there is no inadequacy in similarity between data in different data sets.

Various authors have suggested different partitioning ratios. For instance, [89] presented 75 percent of available data to the ANN as training and validation set and 25 percent as test set. [16] suggested a ratio of 4:1:1. [90] divided all of the data into two data groups; the first data group (70% of all the data) was used for training the network and the second data group (remaining data) was utilized for verification of the ANN models. Hence, the partitioning choice is governed by the ratio that yields the best training and test results.

## 4.6. Model Output Variables

This is where the output variables of the wind power model, which include prediction of 6-hour, three-hour or 72-hour ahead as the case may be, are presented for performance measure comparison especially with other algorithms.

### 4.6.1 Descriptive Statistics

To identify or eliminate a candidate distribution, descriptive statistics are applied. For instance, the sample mean and median times will be close for a symmetrical or nearly symmetrical distribution, such as the normal or Weibull with a shape parameter between 3 and 4. If the mean is considerably larger than the median, then the exponential or lognormal distribution will provide a better fit.

The mean, $\mu$ or wind variability, Median, $t_{med}$, Mode,$t_{mode}$, and variance, $\sigma^2$, of the lognormal distribution are discussed in [62], hence, realisation of Eq. (4.23)

$$\mu = t_{med} \exp \frac{1}{2}\sigma^2 \tag{4.23}$$

The mean of the wind speed data over the wind farm $\mu'$ is described in terms of $t_{med}$, and $\sigma$ is given by Eq. (4.24)

$$\mu' = \ln(t_{med}) - \frac{1}{2}\ln\left(\frac{\sigma^2}{t_{med}} + 1\right) \tag{4.24}$$

Furthermore, the median of this distribution is given by Eq. (4.25):

$$t_{med} = e^{\mu'} \tag{4.25}$$

The mode is estimated using Eq. (4.26) while the standard deviation is obtained using Eq. (4.27):

$$t_{mode} = \frac{t_{med}}{\exp(S^2)} \tag{4.26}$$

$$\sigma^2 = t_{med}^2 \exp(S^2)[\exp(S^2) - 1] \tag{4.27}$$

### 4.6.2 Probability Plots

A probability plot is utilized to provide a better visual test of a distribution than comparison of a histogram with a probability density function. Initial estimates of the parameters of the

distribution fitted are made possible with probability plots, such as exponential, normal distributions, and lognormal plots.

4.6.2.1 Exponential Plots

The cumulative distribution function plot for the exponential distribution is expressed by the computation of Eq. (4.28)

$$F(t) = 1 - e^{-\lambda t} \tag{4.28}$$

Taking the natural logarithm of both sides

$$\ln[1 - F(t)] = -\lambda t$$

$$-\ln[1 - F(t)] = \ln\left[\frac{1}{1-F(t)}\right] = \lambda t$$

An accurate fit to the observed times data may be obtained by performing a least-squares fit of Eq. (4.29)

$$\hat{\lambda} = b = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} {x_i}^2} \tag{4.29}$$

where

$y_i = \ln\frac{1}{[1-F(t)]}$ and $x_i = t_i$

The estimate for wind power from adjacent turbines is expressed by $= \frac{1}{b}$

4.6.2.2 Normal Distribution Plots

For the normal distribution, which represents random wind speed variable selection from different turbines is estimated with Eq. (4.30) to form the tradition bell-shaped graph shown in Figure 5.7.

$$F(t) = \phi\left(\frac{t-\mu}{\sigma}\right) = \phi(z) \tag{4.30}$$

The inverse function is rewritten as Eq. (4.31), which is linear in time $t$.

$$z_i = \phi^{-1}[F(t)] = \frac{t_i - \mu}{\sigma} \tag{4.31}$$

The points $(t_i, \widehat{\mathcal{F}}(t_i))$ are plotted with appropriate transformation of the vertical scale. A least squares fit is obtained by setting

$$x_i = t_i \text{ and } y_i = z_i \tag{4.32}$$

The values of $z_i$ are obtained from a Table of standardized normal probabilities, based on the corresponding values of $\widehat{\mathcal{F}}(t_i)$. From the least-squares fit and Eq. (4.33),

$$\hat{\sigma} = \frac{1}{b} \text{ and } \hat{\mu} = -a\hat{\sigma} = -\frac{a}{b} \tag{4.33}$$

4.6.2.3 Lognormal Plots.

The lognormal plot is made with the relationship of the distribution with the normal distribution.

Since $F(t) = \phi\left(\frac{1}{S}\ln\frac{t}{t_{med}}\right) = \phi(z)$ and $Z = \phi^{-1}[F(t)] = \frac{1}{S}\ln t - \frac{1}{S}\ln t_{med}$, The points $(lnt_i, Z_i)$ are plotted. For the least squares fit, Eq. (4.35) is presented, where;

$$x_i = \ln t_i \text{ and } y_i = Z_i \tag{4.35}$$

The shape parameter $S$ is the reciprocal of the slope of the plotted line and $t_{med}$, the median,is obtained from the intercept of the fitted line. That is

$$\hat{S} = \frac{1}{b} \text{ and } t_{med} = e^{-\hat{S}a} \tag{4.36}$$

4.6.2.4 Parameter Estimation

Probability plots and least squares data fitting only provides estimates of the distribution parameters and does not provide best results. For this reason, a maximum likelihood estimator (MLE) $P$ is defined in Eq. (4.37).

$$\widehat{P} = \frac{n}{n+\sum_{i=1}^{n}(x_{i-1})} = \frac{n}{\sum_{i=1}^{n} x_i} \tag{4.37}$$

If the probability of a failure remains a constant $P$ and each trial is independent, then

$$P_r\{X = x\} = f(x) = (1 - P)^{x-1}P \quad x = 1,2, \ldots \tag{4.38}$$

Where $X$ is the variable representing the number of trials necessary to obtain the first failure and $x$ represents the sample size, f(x) is the likelihood function and represents the probability of obtaining the observed sample.

4.6.3 Stationarity Test.

Prior to building predictive models for training time series algorithms, a stationarity test is one of the required steps. The research employed the Dickey-Fuller Test (DFT) to study stationarity considering daily wind variations within five days of historical data. From the test, the series is not stationary as in Figure3. The non-stationarity is attributed to trend and issues of normal distribution.

To obtain stationarity, the first level DFT is computed (d = 1) where d = differencing which is the difference between current series ($\gamma_t$) and previous series ($\gamma_{t-1}$) as in $\Delta\gamma_t = \gamma_t - \gamma_{t-1}$.

From our differenced data, Figure 3.6 is generated. This figure implies that maximum wind speed is experienced from 4AM to noontime leaving the afternoon time with low wind speeds.

The insight gained in the figure led to data split; 06-14-2003 to 06-17-2003, equivalent to 80% of the data for training while the rest in 6-17-2003 to 06-18-2003 equivalent to 20%, within the high wind time is set for testing the regularised models.

The non-linear nature of RNN and improvements in LSTM as demonstrated by [70, 91] shows the dynamic nature of wind in relation to geography, climate, landforms, seasonality can be addressed although in [34], statistical wind power transform appears strenuous.

## 4.7. LSTM and Dropout Learning Rules

This is a procedure for modifying the weights and biases of a network. It is also referred to as the training algorithm. The LSTM hybridization by dropout is as presented below for learning the proposed wind-farm power output prediction based on wind speed.

### 4.7.1. The Hybrid Long Short-Term Memory and Dropout Modelling.

In neural network technology, there is no existing and specific methodology in neural selection especially the hidden neurons although as stated in [75, 83] new methods of designing hidden neurons is based on data structure and the nature of the predictive horizon employed. However, the author in reference [75] tried to fix the hidden neuron selection problem using a mathematical foundation of convergence theorem. Here, during the training process, each criterion that satisfies the convergence is tested optimally for error reductions. Therefore, the flowchart of Figure 4.13 demonstrates the combination or hybridisation of our learning model. Mathematically, however, this model is as described in section 5.3.

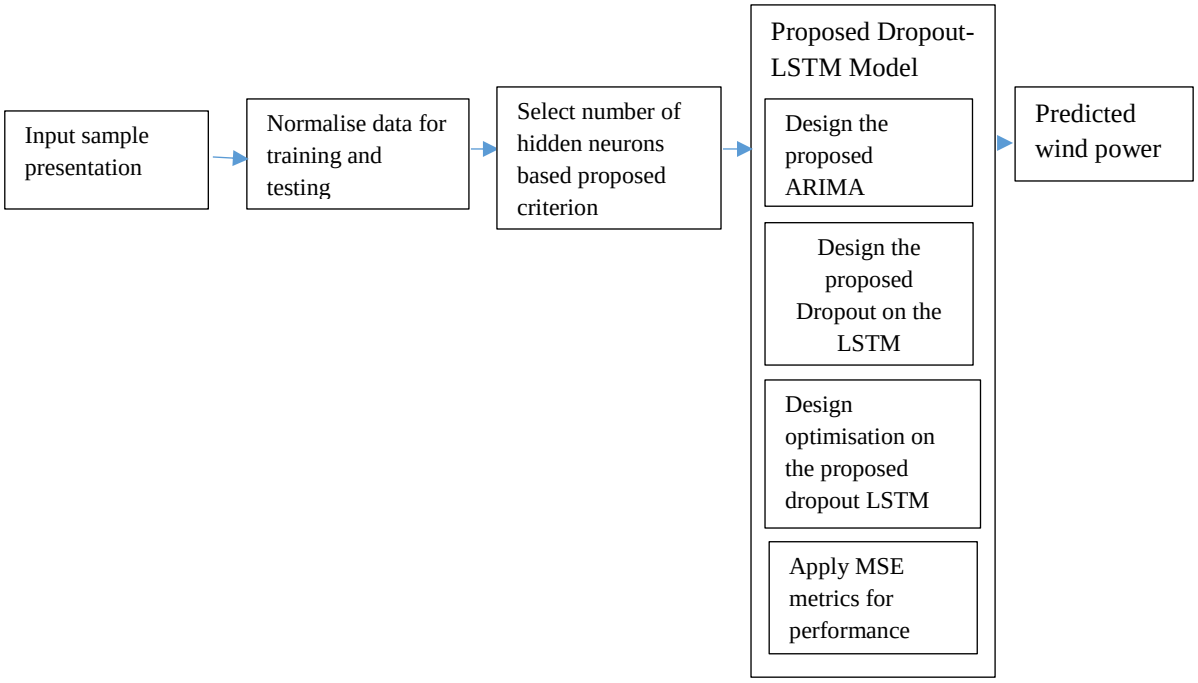



Figure 4. 11**:** Wind Power Simulation Model.

In addition, this research however adopts the method by averaging the output of the independent input layers for wind speed prediction, since the method enhances the stability of the network after many trials. The network is as shown in Figure 4.8, which depicts the LSTM dropout connection of the network. However, the design is categorised into four modules as follows:

#### 4.7.1.1. LSTM and Dropout Modelling Modules

**Module 1**. The Improved long short-term memory (LSTM) is the model with the supervised learning approaches in mind and there exists a nonlinear activation function. The logistic sigmoidal function is used in the input layers while the ReLu is used in the hidden layers of the network. The input Layer has six input layers with sigmoid functions with 20 hidden layers of the network. In addition, this hidden layer neuron is fixed as the proposed criterion. The hidden neurons enable the cell state of the LSTM network to activate for both the input gate, forget gate and the output gate considering highly complex tasks like wind speed forecasting. Each of the LSTM layers is connected as shown in Figure 4.14 (a generic RNN) within them by synaptic weights - $\sigma_{1,1n,2n,3n,etc,}$

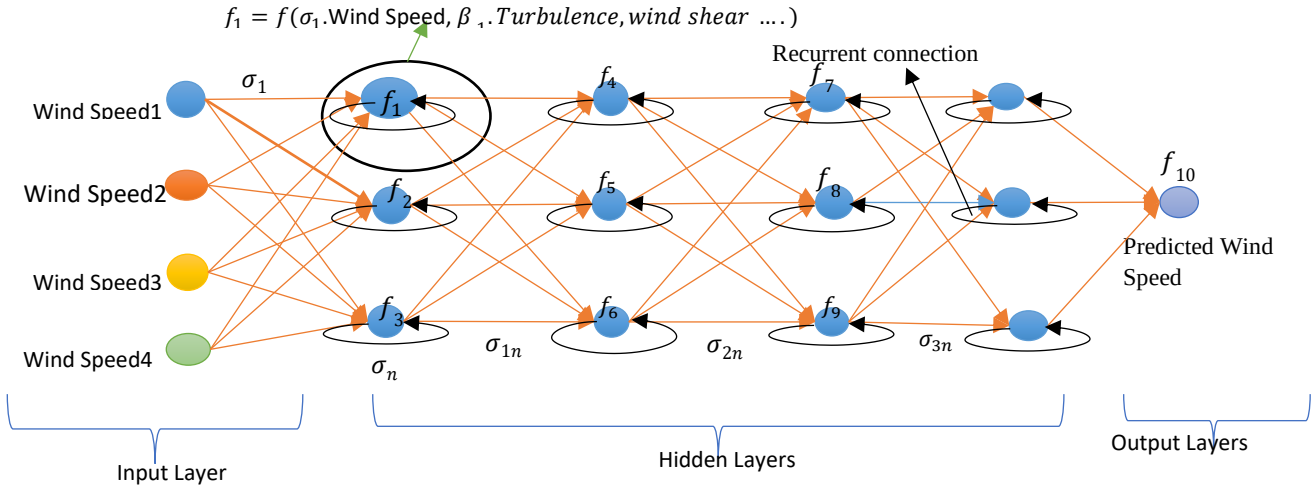



Figure 4. 12**:** Fully Connected LSTM network for wind power prediction[26]

[26] This is not the generic neural network used in the model. The Figure is simply for illustration purposes.

**Module 2**. During the dropout of certain neurons as shown in Figure 4.15, single LSTM combine, so that the output from certain LSTM becomes the input for a few others; then the network becomes multi-layer recurrent linear neuron. In the dropout processing, the initialization experiences first, the weights between the input and the hidden units generating small positive random values. The considered input is as presented in the figure and based on the weighted interconnections; the network input is computed for both the hidden units. To obtain the activation of the outputs, hidden units are then applied. However, with suitable weighted interconnections, the hidden output acts as input to the output layer and the net input and output of the output layer are computed. The element wise comparison ensures that the target and suitable weight updates are performed.

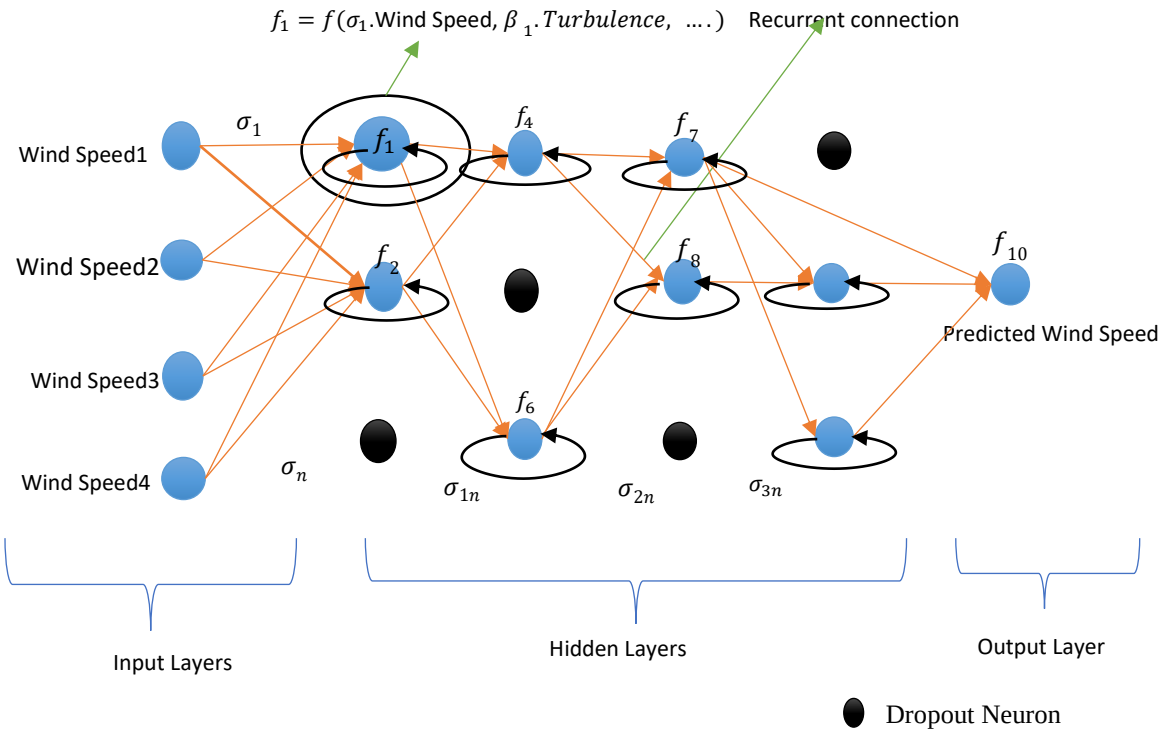



Figure 4. 13**:** Dropout connected LSTM layers[27].

[27] The Figure is simply for illustration purposes.

**Module 3**. During back propagation (BPN) in time, the neural network module as shown in the Figure 4.15 ensures that the input layer connects to the hidden layer and the hidden layer in turn connects to the output layer by means of interconnected weights – in a chain form. Whereas, in the training phase, the signals are sent in the reverse direction. The increase in the number of hidden layers results in the computational complexity of the network and hence a randomly selected hidden layer is dropped-out, in order words 20% of it. The proposed criterion is incorporated into the training algorithm to fix the number of hidden neurons in the single hidden layer. In BPN processing, the bias is provided for both the hidden and the output layer to act upon, hence, the net input to be calculated.

**Module 4**. These set of inputs are multiplied by a set of weights ($w_{\theta i}$), which are further processed by individual deep units, that are of 11-hidden layers, as (see section 4.6). Finally the output Θ unit as in Eq. (4.39) through 4.41.

$$y_{\Theta}(t) = P\left(f(eLSTM_{\Theta}(t), ARIMA_{p,d,q}) \left| \{ws_{t-1}, , \dots\}\right.\right). \quad (4.39)$$

where

$$eLSTM_{\Theta}(t) = \frac{1}{p}.\left[g\left(\sum_{i=1}^{\Theta} w_{\theta i} X_i(t).q + b\right)\right] \quad (4.40)$$

$$ARIMA_{p,d,q} = \sum_{i=1}^{\Theta} X_i(t) \quad (4.41)$$

In our model, t represents a 10-minutes interval of wind data recorded, while our model represents sigmoid function implemented as non-linear output. The $p$ is the probability of keeping an LSTM neuron.

$$f(x) = \frac{1}{1+e^{-x}} \quad (4.42)$$

Modelling 3-hours ahead is as proposed in Eq. (4.43), where N is the number of hours considered in the dataset. However, for each node and hour, our formulation using the

generated data, discussed in chapter 4 for prediction at the succeeding 180 minutes ahead results in the formulation of Eq. (4.43)

$$\mathrm{N}_{\mathrm{d}}^{\mathrm{nod}} = \left\{\mathcal{N}_{\mathrm{s}}^{\mathrm{nod}}[\mathrm{t_h} + \mathrm{min}], \mathcal{N}_{\mathrm{t,w,t,h}}^{\mathrm{nod}}[\mathrm{t_h} + \mathrm{min}]\right) \mid \mathrm{min} = 1, 2, \ldots., 180\} \quad (4.43)$$

Here, $\mathrm{h} = 1,2,3$ and $\mathrm{nod} \in \{\text{position of wind Turbine}\}$, which is not disclosed in our dataset. $\mathcal{N}_{\mathrm{t,w,t,h}}^{\mathrm{nod}}$ and $\mathcal{N}_{\mathrm{s}}^{\mathrm{nod}}$ denotes the turbine's node prediction of wind power respectively at time $\mathrm{t} = \mathrm{t_h} + \mathrm{min}$ given temperature, humidity, wind shear, turbulence.

All the proposed four modules that comprise the ensemble model initiate its training process by learning from the normalized data. The training process is carried out until the error (performance metric) reaches a negligible value. The average value of the dropout LSTM corresponding to the minimal error gives the final output of the proposed neural network. The proposed training algorithm of the neural network is given in nine basic steps as follows:

#### 4.7.1.2. Basic Steps in LSTM Modelling

- Step 1. Initialize the necessary parameters of the individual LSTM, and dropout-LSTM.
- Step 2. Introduce the proposed criterion to fix the number of hidden neurons into each of the training algorithms of individual neural network models.
- Step 3. Present the input and target vector pair to the individual neural network models. The input-target vector pair corresponds to training datasets when the training process is initiated and for testing the trained network, testing datasets are employed.
- Step 4. Compute the net input of the individual networks and obtain their corresponding outputs by applying activation over the calculated network input. The outputs computed for each of the individual networks are given by *Y*LSTM, *Y*Dropout-LSTM, and the individual networks are denoted as *H*LSTM, *H*Dropout-LSTM.

- Step 5. Develop the neural network as the aggregation of the individual neural network models and determine the final predicted wind power:
- Step 6. Train each of the individual ensemble networks and compute the error value
- Step 7. Select the appropriate hidden neurons to be placed in the hidden layer of each of the ensembles based on the minimum error performance.
- Step 8. Output the selected criterion and the minimum MSE value.
- Step 9. Test for stopping condition (the stopping condition is the reaching point of minimum MSE or specified number of epochs).

## 4.8. Modelling ARIMA model.

Machine learning is mushrooming in time series systems due to the integration of traditional statistical approaches. As described in chapter three, unlike the language model, speech recognition and computer vision where models are applied directly to the data [65, 92, 93], time series requires efficient modelling due to outliers inherited from the data.

In section 4.4, data is differenced to improve performance, the core approach of the Box-Jenkins time series model [94] which is the d component of the ARIMA model. The p and q parameters as modelled in [95, 96] can either be generated using machine learning tools[28] or the traditional autocorrelation function (ACF) and partial autocorrelation functions (PACF) as described below.

### 4.8.1 Autocorrelation functions (ACF)

For wind speed and wind power of a given series, selection of the decomposition level remains one of the most important decisions. However, to the best of the author's knowledge; there is

[28] Using the traditional ML technique to deduce the p, d, q components on Sklearn-metrics ARIMA model selection.

no theoretical criterion for the selection of theoretical decomposition level. Scikitlearn packages in python results in the use of machine learning algorithm developed by Python developers to solve issues of decomposition, traditional ACF has over the years, employed decomposition to solve the seasonality of the wind series data. However, the motivation of the ACF usage as described by [97] is the ability to characterize ‘’periodicity or self-similarity’’ which invariably is the scale invariance property of time series systems (see Appendix A (i)).

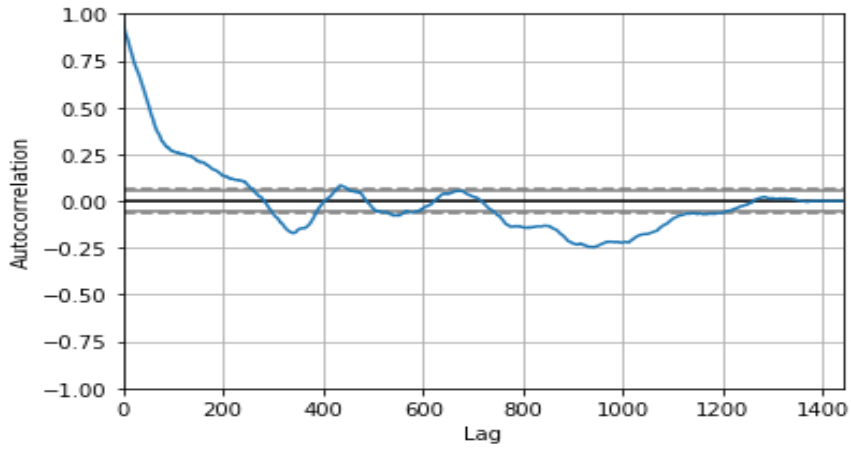



Figure 4. 14**:** Autocorrelation plot of the wind series.

Furthermore, time series components relies on ACF to verify whether observations such as wind speed has trend and or seasonality as observed in Figure 4.14, then report the extent or sufficiency level in relation to the wind decomposition. Conversely, when the trend and seasonality component seen in Figure 4.15 of the wind speed series is similar to the ACF, we can reasonably conclude that the decomposition of moving average (MA) or auto-regression (AR) provides sufficient accuracy.

Autocorrelation values are statistically obtained by the use of [98, 99] for the associations between data in time series separable at different lags of times. In other words, time series y with Z data points at index lag of $\tau$ is estimated with Eq. (4.44) where y is the average of time series, shown in Figure 4.14.

$$\rho(\tau) = \frac{\sum_{z}^{Z-\tau}[y(m+\tau)-\ddot{y}][y(z)-\ddot{y}]}{\sum_{m=1}^{Z-\tau}[y(z)-\ddot{y}]^2} \quad (4.44)$$

4.8.2 Partial autocorrelation function (PACF).

Like the decomposition discussed above, the partial autocorrelation function is used to measures the correlation between observations of the wind series that are separated by t time units – $y_t$and $y_{t-1}$, after adjusting for the presence of all the other terms of shorter lag $y_{t-1}, y_{t-2,}, \ldots, y_{t-t-1}$ as shown in Eq. (4.36) of [100] gave a detailed description of PACF.

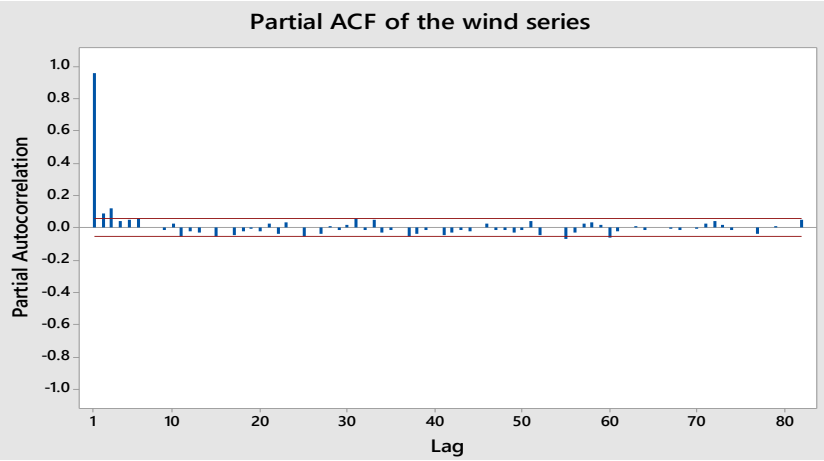



Figure 4. 15: PACF of the wind series.

Furthermore, in terms of the ARIMA model, the relationship with PACF is related to the p component of the ARIMA component. From the test, the component is as shown in Table 4.2 below.

Research over the recent years has developed algorithms and systems that can derive these ARIMA components programmatically without differencing, correlating and partially correlating the signals, this result in the machine learning aspect of component derivation.

4.8.3 Machine learning ARIMA Derivation.

The grid search technique[29] is the machine learning approach to calculate the best ARIMA$_{(p,d,q)}$ components. Applying grid search hyper-parameter for the p, d, q components to the training and test sets is ideal for optimum performance due to the model building across each component.

The approach in this process is iteration through parameter combinations for each possible model combination [101]. The grid search method depends on data size and is expensive computationally, it relies on the performance of the system processor and RAM in use. In addition, grid searching can be used to tune for performance measures, like the AIC, AUC criteria and so on. The model tuning result is a measure of the RMSE statistical quantity for error search accuracy. In our case, about 90% evaluated RMSE error on each of the trials – Table 4.2 were reported meaning that the search has the best (p, d, q) components. However, the code utilised in grid searching the data is presented in Appendix a (ii).

Table 4. 2: Comparing the p, d, q component of ARIMA and ML derivation.

| Sample wind data | ARIMA (P,D,Q) | ML-ARIMA (P,D,Q) |
|---|---|---|
| 720 | (0.1, 1.2, 2.3) | (0,1,2) |
| 1440 | (2.4, 1.2, 2.6) | (2,1,1) |
| 2160 | (1.3, 0.3, 1.6) | (1,0,2) |
| 2880 | (3.3, 0.3, 0.4) | (2,1,2) |
| 3600 | (4.3, 1.2, 3.2) | (1,1,2) |
| 4230 | (0.3, 1.0, 3.2) | (1,2,5) |
| 5040 | (2.3, 1.2, 0.2) | (0,1,4) |
| 5760 | (3.2, 4.2, 3.2) | (2,2,3) |
| 6480 | (0.3, 1.0, 2.2) | (3,1,2) |
| 7200 | (2.3, 4.2, 2.2) | (2,3,2) |

4.8.4 Time Series Residual Component.

This is the difference between an observed value (y) and its corresponding fitted value (ŷ). For example, the scatterplot of Figure 3.12 (chapter 3) that plots men's weight against their height,

[29] Grid searching is the process of scanning data for optimal parameter configurations

the regression line plots the fitted values of weight for each observed value of height. Suppose a man is 6 feet tall and the fitted value of his weight is 190 lbs. If his actual weight is 200, the residual is 10. If his actual weight is 175, the residual is -15. Residual values are especially useful in regression and ANOVA procedures because they indicate the extent to which a model accounts for the variation in the observed data.

# Chapter 5.

## 5.1. Field Data Description.

The wind site data used for this research is from the prognostic and health management (PHM) society. PHM[30] conducts annual Data challenge for conferences and possible journal publications on sensor management. The 2011 Data challenge is for wind farm management based on sensor failure detection. Therefore, this research leveraged on the data to model power-output prediction from the farm. The location of the wind farm is not disclosed due to security reasons, hence, the k factor of the Weibull distribution is assumed for the research purposes. Cup anemometers (see chapter 2) and wind vanes were the major sensors used by PHM to record the wind farm dataset. The accuracy of these sensors is, in part, due to the health of the bearings supporting the cup shaft. The data is captured as depicted in Figures 5.1 and consists of a 5-day measurement period with 420 wind data samples similar to 420 turbines in a wind farm, which includes:

- Mean, standard deviation, minimum and maximum wind speed at height 10, 39, 49 and 59 meters respectively.
- Mean, standard deviation, minimum and maximum wind direction at height 39, 49 and 69 meters respectively.
- Date in Month/Day/Year and time in hours: minutes: seconds.

In this research, data from fourteen turbines are independently considered and no assumptions on topography of the wind farm is made. These are data from wind turbine 8 (WT8), wind turbine 21 (WT21), WT61, WT93, WT120, WT171, WT190, WT208, WT230, WT274, WT263, WT291, WT310 AND WT330 respectively.

[30] The PHM data is selected for this research because they have reliable wind farm data that comprise of wind speed from different turbines.

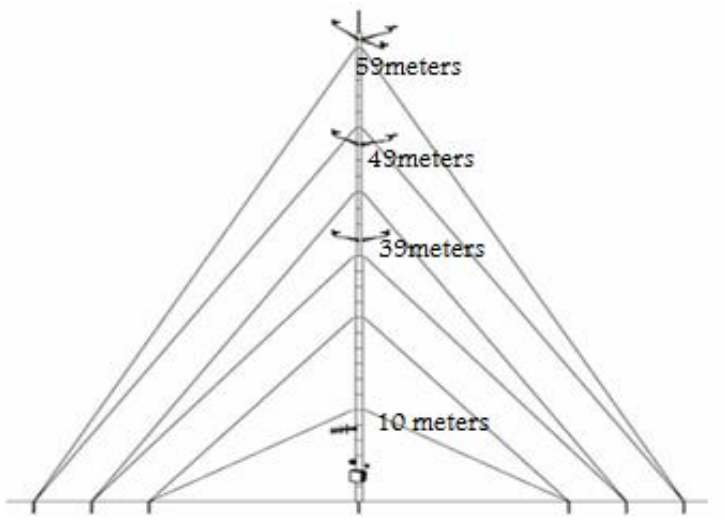



Figure 5. 1: 60m tower with three-sensor attachments.
Source: [28]

The sample dataset are as shown in Table 5.1, 5.2 and 5.3 representing the wind data from WT8, WT21 and WT61 respectively. These data are used for the plots of Figure 5.2, 5.3 and 5.4 respectively to demonstrate the variations within the wind farm.

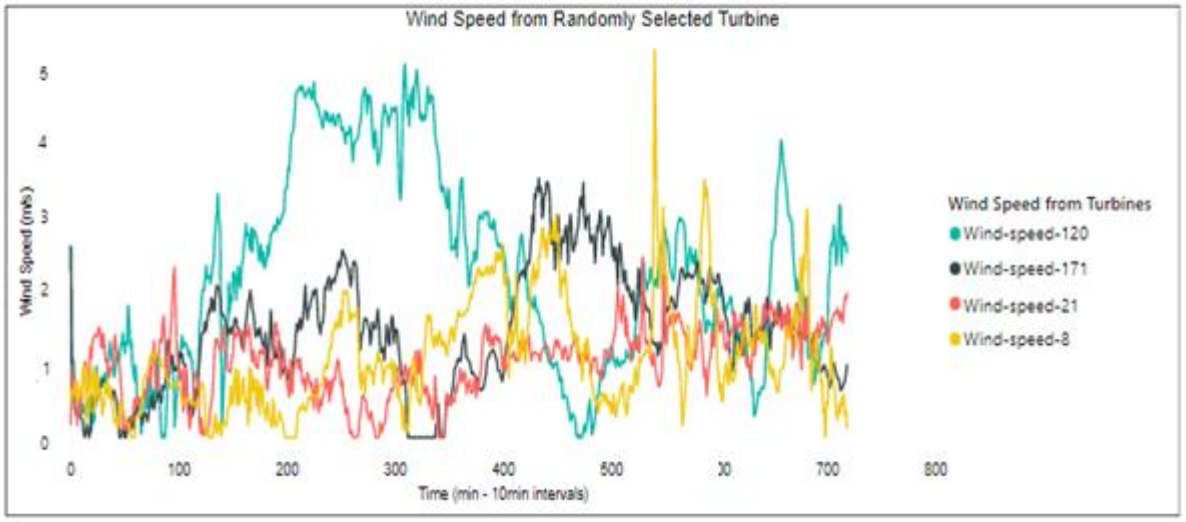



Figure 5. 2: Wind Series from four wind turbines.

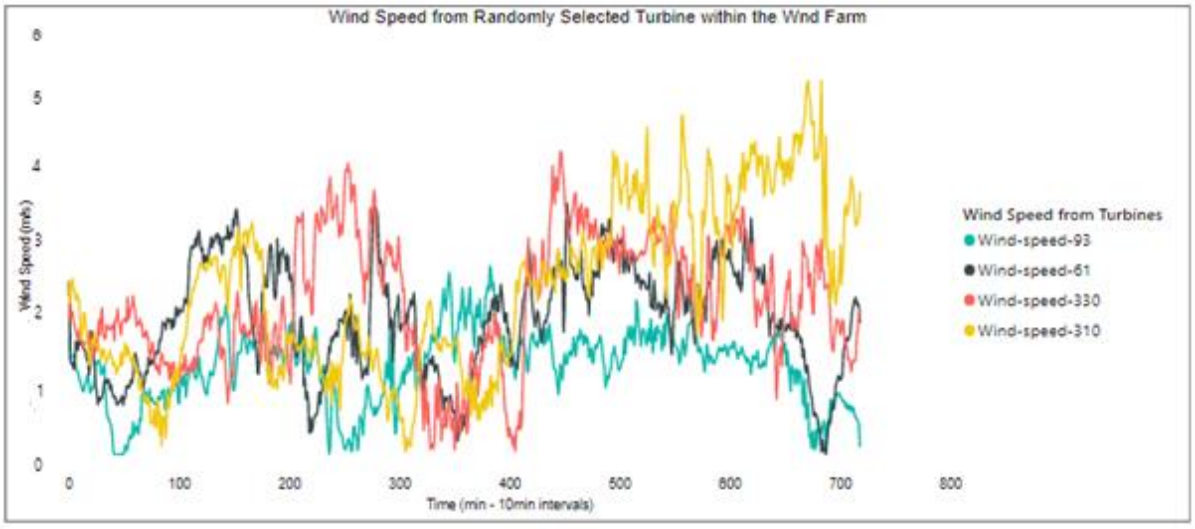



Figure 5. 3**:** Wind Series from four wind turbines.

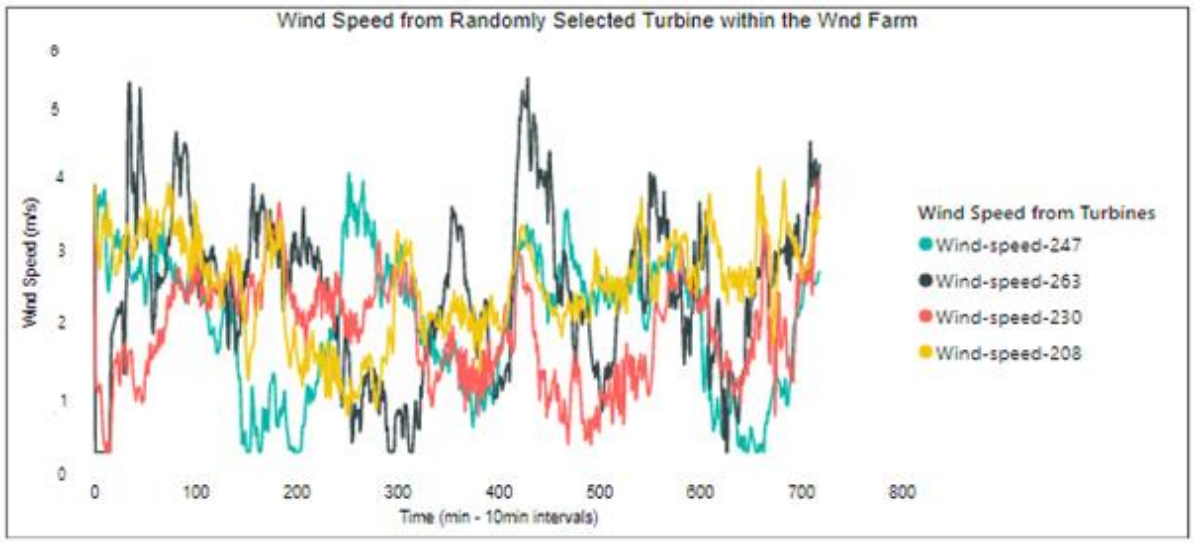



Figure 5. 4: Wind Series from four wind turbines.

Table 5. 1: Sample Wind Data of Wind Turbine 8 (WT8)

| Mean49 | Max49 | Min49 | MeanWD | SDWD | MaxWD | MeanTemp | MaxTemp | MinTemp | Date | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| 6.15 | 7.24 | 4.56 | 163.21 | 7.28 | 175.39 | 11.96 | 12.09 | 11.83 | 10/11/2007 | 00:00:00 |
| 5.69 | 7.61 | 3.8 | 164.51 | 7.28 | 173.97 | 12.23 | 12.36 | 12.09 | 10/11/2007 | 00:10:00 |
| 5.42 | 6.85 | 4.17 | 171.39 | 7.28 | 173.03 | 12.09 | 12.09 | 11.83 | 10/11/2007 | 00:20:00 |
| 5.48 | 6.85 | 3.8 | 174.02 | 7.94 | 184.64 | 11.89 | 12.09 | 11.56 | 10/11/2007 | 00:30:00 |
| 5.13 | 6.85 | 3.8 | 169.7 | 8.66 | 162.14 | 11.83 | 12.09 | 11.56 | 10/11/2007 | 00:40:00 |
| 5.28 | 6.85 | 3.8 | 163.1 | 8.29 | 161.27 | 12.29 | 12.36 | 11.83 | 10/11/2007 | 00:50:00 |
| 6.7 | 8.37 | 5.32 | 171.28 | 9.44 | 195.45 | 12.56 | 12.9 | 12.36 | 10/11/2007 | 01:00:00 |
| 6.5 | 8.77 | 4.94 | 167.58 | 9.86 | 176.34 | 12.56 | 12.9 | 12.36 | 10/11/2007 | 01:10:00 |
| 6.17 | 7.24 | 5.32 | 163.21 | 7.61 | 157.81 | 12.23 | 12.36 | 11.83 | 10/11/2007 | 01:20:00 |
| 6.78 | 7.61 | 6.09 | 168.36 | 6.68 | 172.56 | 12.03 | 12.09 | 11.83 | 10/11/2007 | 01:30:00 |
| 6.51 | 7.24 | 5.71 | 164.84 | 6.97 | 159.53 | 11.7 | 11.83 | 11.3 | 10/11/2007 | 01:40:00 |
| 6.58 | 6.85 | 6.09 | 164.62 | 6.12 | 149.49 | 11.36 | 11.56 | 11.3 | 10/11/2007 | 01:50:00 |
| 6.73 | 7.61 | 6.09 | 158.65 | 3.97 | 155.27 | 11.1 | 11.3 | 10.77 | 10/11/2007 | 02:00:00 |
| 6.33 | 7.24 | 5.32 | 164.51 | 5.86 | 154.43 | 10.9 | 11.03 | 10.77 | 10/11/2007 | 02:10:00 |
| 6.21 | 6.85 | 5.71 | 163.64 | 5.38 | 162.14 | 10.84 | 11.03 | 10.77 | 10/11/2007 | 02:20:00 |
| 6.07 | 6.85 | 5.32 | 163.97 | 5.86 | 168.41 | 10.77 | 11.03 | 10.51 | 10/11/2007 | 02:30:00 |
| 5.64 | 6.47 | 4.17 | 164.08 | 7.61 | 175.39 | 10.44 | 10.77 | 10.25 | 10/11/2007 | 02:40:00 |
| 5.69 | 6.85 | 4.94 | 154.21 | 4.93 | 167.5 | 10.44 | 10.77 | 10.25 | 10/11/2007 | 02:50:00 |
| 4.81 | 6.47 | 4.17 | 166.59 | 7.61 | 167.95 | 10.57 | 10.77 | 10.25 | 10/11/2007 | 03:00:00 |
| 5.38 | 6.85 | 4.56 | 164.84 | 5.15 | 171.16 | 10.38 | 10.51 | 10.25 | 10/11/2007 | 03:10:00 |
| 7.07 | 7.61 | 6.47 | 169.59 | 6.12 | 171.63 | 9.92 | 10.25 | 9.72 | 10/11/2007 | 03:30:00 |
| 6.65 | 7.24 | 5.71 | 173.67 | 6.12 | 171.16 | 9.59 | 9.98 | 9.46 | 10/11/2007 | 03:40:00 |
| 5.25 | 6.47 | 4.17 | 160.12 | 6.97 | 154.85 | 9.46 | 9.72 | 9.2 | 10/11/2007 | 03:50:00 |
| 4.28 | 5.71 | 2.65 | 138.79 | 11.23 | 149.9 | 9.72 | 9.98 | 9.46 | 10/11/2007 | 04:00:00 |
| . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . |
| 720 | 4.56 | 2.26 | 185.55 | 10.3 | 184.15 | 9.46 | 10.25 | 8.95 | 10/12/2007 | 04:40:00 |

Table 5. 2**:** Sample Wind Data of Wind Turbine 21 (WT21)

| Mean49 | Max49 | Min49 | Mean39 | Max39 | MeanWD | SDWD | MaxWD | MeanTemp | MaxTemp | MinTemp | Date | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13.17 | 15.27 | 10.29 | 11.25 | 13.34 | 184.7 | 6.97 | 180.2 | 15.21 | 15.35 | 15.35 | 10/12/2007 | 01:00:00 |
| 13.97 | 15.63 | 11.45 | 12.05 | 14.52 | 183.36 | 7.94 | 195.45 | 15.21 | 15.35 | 15.35 | 10/12/2007 | 01:10:00 |
| 14.11 | 15.63 | 12.2 | 12.15 | 14.14 | 183.11 | 6.97 | 181.18 | 15.14 | 15.35 | 15.07 | 10/12/2007 | 01:20:00 |
| 13.58 | 16.05 | 11.45 | 11.53 | 13.34 | 183.96 | 7.28 | 189.2 | 15.07 | 15.35 | 15.07 | 10/12/2007 | 01:30:00 |
| 12.52 | 14.52 | 10.29 | 10.66 | 13.34 | 184.33 | 7.61 | 182.16 | 14.93 | 15.07 | 14.8 | 10/12/2007 | 01:40:00 |
| 11.49 | 14.14 | 9.54 | 9.87 | 12.2 | 186.29 | 7.28 | 184.15 | 14.66 | 15.07 | 14.8 | 10/12/2007 | 01:50:00 |
| 11.57 | 13.73 | 9.15 | 9.71 | 11.45 | 187.16 | 6.97 | 184.15 | 14.59 | 14.8 | 14.52 | 10/12/2007 | 02:00:00 |
| 11.82 | 14.14 | 9.92 | 9.86 | 11.82 | 187.53 | 6.97 | 182.16 | 14.52 | 14.8 | 14.52 | 10/12/2007 | 02:10:00 |
| 12.13 | 14.14 | 9.92 | 10.29 | 12.2 | 190.03 | 6.12 | 192.82 | 14.52 | 14.8 | 14.52 | 10/12/2007 | 02:20:00 |
| 11.82 | 13.34 | 9.54 | 10.16 | 12.59 | 191.04 | 6.12 | 187.16 | 14.52 | 14.8 | 14.52 | 10/12/2007 | 02:30:00 |
| 11.52 | 13.34 | 8.37 | 9.85 | 11.82 | 191.3 | 6.4 | 195.98 | 14.59 | 14.8 | 14.52 | 10/12/2007 | 02:40:00 |
| 11.54 | 13.34 | 8.77 | 10.27 | 12.59 | 193.98 | 6.12 | 201.36 | 14.8 | 15.07 | 14.8 | 10/12/2007 | 02:50:00 |
| 10.83 | 12.96 | 8.77 | 9.62 | 12.59 | 197.22 | 4.72 | 196.51 | 14.93 | 15.35 | 14.8 | 10/12/2007 | 03:00:00 |
| 9.19 | 12.2 | 6.47 | 8.06 | 10.67 | 202.12 | 3.8 | 202.45 | 14.73 | 15.07 | 14.8 | 10/12/2007 | 03:10:00 |
| 8.82 | 10.67 | 6.09 | 7.64 | 10.29 | 205.64 | 4.93 | 203 | 14.52 | 14.8 | 14.52 | 10/12/2007 | 03:20:00 |
| 8.11 | 9.92 | 6.47 | 7.08 | 9.54 | 210.33 | 4.72 | 213.71 | 14.52 | 14.8 | 14.52 | 10/12/2007 | 03:30:00 |
| 7.36 | 8.77 | 6.09 | 6.58 | 8.37 | 211.59 | 4.93 | 205.77 | 14.59 | 14.8 | 14.52 | 10/12/2007 | 03:40:00 |
| 7.9 | 9.54 | 6.47 | 7.3 | 8.77 | 216 | 3.8 | 218.39 | 14.39 | 14.52 | 14.25 | 10/12/2007 | 03:50:00 |
| 6.84 | 7.99 | 5.71 | 6.37 | 7.61 | 214.28 | 3.64 | 220.18 | 14.25 | 14.52 | 14.25 | 10/12/2007 | 04:00:00 |
| 6.55 | 7.61 | 5.32 | 6.07 | 7.24 | 214.42 | 3.64 | 211.99 | 14.05 | 14.25 | 13.98 | 10/12/2007 | 04:10:00 |
| 6.27 | 7.61 | 4.56 | 5.37 | 6.47 | 212.16 | 4.52 | 209.7 | 13.71 | 13.98 | 13.71 | 10/12/2007 | 04:20:00 |
| 4.92 | 6.09 | 3.41 | 4.39 | 5.71 | 235.03 | 15.88 | 218.39 | 13.5 | 13.71 | 13.44 | 10/12/2007 | 04:30:00 |
| 4.86 | 6.47 | 3.41 | 4.46 | 6.09 | 260.24 | 13.36 | 261.83 | 12.63 | 13.44 | 12.09 | 10/12/2007 | 04:40:00 |
| 6.08 | 7.24 | 4.94 | 5.67 | 6.47 | 253.22 | 11.73 | 252.1 | 12.09 | 12.36 | 11.83 | 10/12/2007 | 04:50:00 |
| 6.85 | 8.37 | 5.71 | 5.78 | 6.47 | 254.41 | 1.34 | 253.46 | 11.89 | 12.09 | 11.56 | 10/12/2007 | 05:00:00 |
| 9.35 | 10.67 | 7.99 | 6.89 | 7.61 | 257.48 | 2.57 | 259.01 | 11.5 | 11.83 | 11.3 | 10/12/2007 | 05:10:00 |
| 9.44 | 10.29 | 8.37 | 7.02 | 9.54 | 262.15 | 3.06 | 262.54 | 11.17 | 11.3 | 11.03 | 10/12/2007 | 05:20:00 |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| 720 | 11.45 | 8.37 | 7.71 | 9.54 | 261.8 | 3.8 | 265.4 | 11.36 | 11.83 | 10.77 | 10/12/2007 | 06:40:00 |

Table 5. 3: Sample Wind Data of Wind Turbine 61 (WT61)

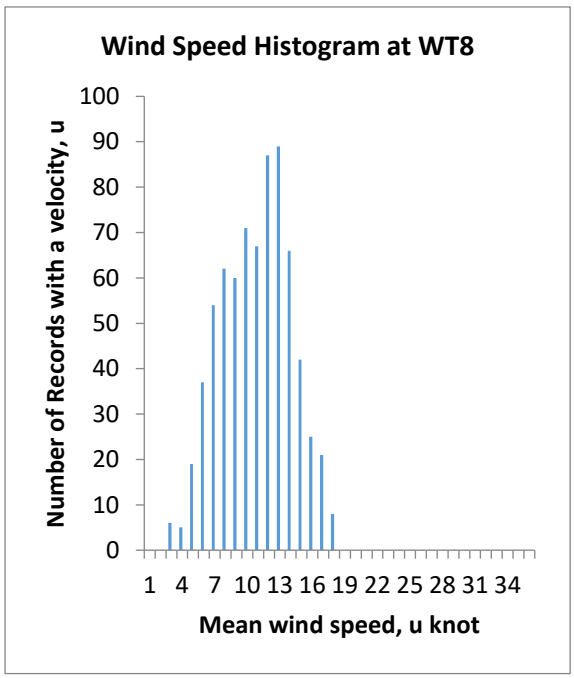
| Mean49 | Max49 | Min49 | Mean39 | Max39 | MeanWD | SDWD | MaxWD | MeanTemp | MaxTemp | MinTemp | Date | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5.34 | 7.61 | 2.26 | 5.61 | 8.37 | 332.94 | 10.76 | 331.38 | 15.62 | 15.62 | 15.35 | 10/15/2007 | 15:20:00 |
| 4.94 | 6.85 | 2.65 | 5.27 | 7.24 | 326.54 | 15.88 | 348.88 | 15.62 | 16.18 | 15.35 | 10/15/2007 | 15:30:00 |
| 5.35 | 6.85 | 3.03 | 5.56 | 7.24 | 330.94 | 9.44 | 346.05 | 15.62 | 16.18 | 15.35 | 10/15/2007 | 15:40:00 |
| 4.88 | 6.85 | 2.26 | 5 | 7.24 | 329.4 | 8.29 | 326.04 | 15.55 | 15.62 | 15.35 | 10/15/2007 | 15:50:00 |
| 4.21 | 6.47 | 1.88 | 4.5 | 6.47 | 320.06 | 10.76 | 313.07 | 15.69 | 16.18 | 15.35 | 10/15/2007 | 16:00:00 |
| 3.74 | 5.71 | 1.88 | 4.11 | 6.47 | 327.42 | 15.88 | 324.28 | 15.48 | 16.18 | 15.35 | 10/15/2007 | 16:10:00 |
| 3.63 | 4.94 | 2.26 | 4 | 5.71 | 333.83 | 19.73 | 325.16 | 15.35 | 15.9 | 15.07 | 10/15/2007 | 16:20:00 |
| 3.08 | 4.56 | 1.88 | 3.41 | 4.94 | 308.72 | 9.86 | 319.06 | 14.93 | 15.35 | 14.8 | 10/15/2007 | 16:30:00 |
| 2.48 | 3.03 | 1.5 | 2.84 | 3.8 | 303 | 6.4 | 304.7 | 14.66 | 15.07 | 14.52 | 10/15/2007 | 16:40:00 |
| 2.69 | 3.41 | 1.88 | 2.88 | 3.41 | 328.08 | 4.52 | 336.81 | 13.98 | 14.52 | 13.71 | 10/15/2007 | 16:50:00 |
| 2.62 | 3.03 | 1.88 | 2.91 | 3.41 | 336.07 | 3.64 | 331.38 | 13.03 | 13.71 | 12.63 | 10/15/2007 | 17:00:00 |
| 2.46 | 3.03 | 1.88 | 2.73 | 3.03 | 345.19 | 0.5 | 347.93 | 12.03 | 12.63 | 11.56 | 10/15/2007 | 17:10:00 |
| 2.16 | 2.65 | 1.5 | 2.59 | 3.03 | 344.72 | 2.36 | 346.05 | 11.43 | 11.56 | 11.3 | 10/15/2007 | 17:20:00 |
| 1.88 | 2.26 | 1.12 | 2.34 | 2.65 | 353.36 | 2.69 | 351.72 | 11.43 | 11.56 | 11.3 | 10/15/2007 | 17:30:00 |
| 1.45 | 1.88 | 0.73 | 1.37 | 2.26 | 0 | 0 | 0 | 11.5 | 11.83 | 11.03 | 10/15/2007 | 17:40:00 |
| 0.51 | 1.12 | 0.35 | 0.4 | 0.73 | 0 | 0 | 0 | 11.5 | 11.83 | 11.3 | 10/15/2007 | 17:50:00 |
| 1.15 | 1.5 | 0.35 | 0.75 | 1.12 | 7.91 | 3.8 | 0 | 11.36 | 11.56 | 11.3 | 10/15/2007 | 18:00:00 |
| 0.54 | 1.12 | 0.35 | 0.42 | 0.73 | 11.5 | 0 | 7.73 | 11.03 | 11.3 | 10.51 | 10/15/2007 | 18:10:00 |
| 0.64 | 1.12 | 0.35 | 0.98 | 1.5 | 128.87 | 29.13 | 120.7 | 9.85 | 10.77 | 8.69 | 10/15/2007 | 18:20:00 |
| 0.35 | 0.35 | 0.35 | 0.35 | 0.35 | 148.52 | 0.06 | 0 | 9.33 | 10.51 | 8.43 | 10/15/2007 | 18:30:00 |
| 1.02 | 1.88 | 0.35 | 1.2 | 2.26 | 149.9 | 3.06 | 151.53 | 10.44 | 10.77 | 9.98 | 10/15/2007 | 18:40:00 |
| 1.66 | 3.41 | 0.73 | 1.91 | 3.41 | 96.89 | 22.46 | 64.23 | 10.05 | 10.51 | 9.98 | 10/15/2007 | 18:50:00 |
| 2.29 | 3.03 | 1.88 | 2.54 | 3.41 | 82.64 | 15.21 | 64.93 | 9.27 | 10.77 | 8.43 | 10/15/2007 | 19:00:00 |
| 2.07 | 2.65 | 1.5 | 2.13 | 2.65 | 105.3 | 1.34 | 105.99 | 8.3 | 8.69 | 7.92 | 10/15/2007 | 19:10:00 |
| 2.66 | 3.41 | 2.26 | 2.7 | 3.41 | 128.11 | 10.3 | 124.69 | 8.11 | 8.43 | 7.92 | 10/15/2007 | 19:20:00 |
| 3.2 | 3.41 | 2.65 | 3 | 3.41 | 128.44 | 3.2 | 124.69 | 8.5 | 8.69 | 8.18 | 10/15/2007 | 19:30:00 |
| 2.9 | 3.41 | 2.65 | 3.23 | 3.8 | 141.93 | 9.44 | 150.3 | 8.56 | 8.95 | 8.18 | 10/15/2007 | 19:40:00 |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . | . | . | . | . |
| 720 | 5.32 | 4.56 | 3.99 | 4.94 | 155.13 | 5.15 | 142.38 | 7.47 | 7.66 | 7.41 | 10/12/2007 | 06:40:00 |

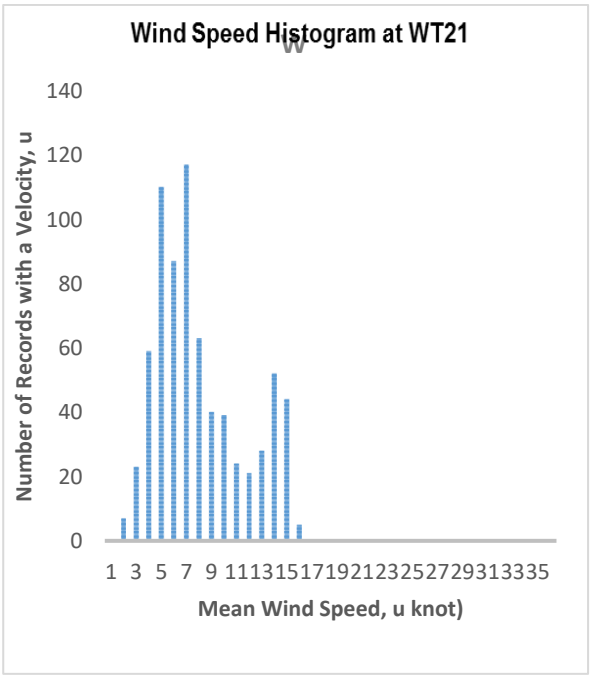
5.1.1. Results and Observations.

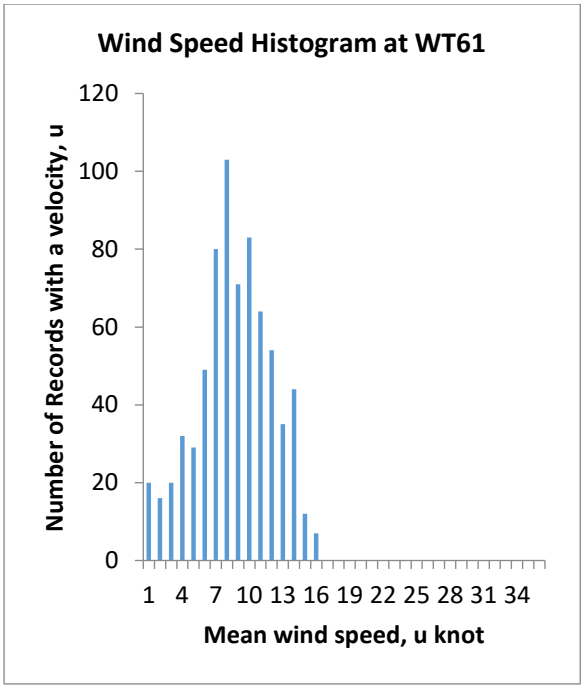
From the plots of Figure 5.2 above, it is clear that these wind turbines experiences wind differently within the Wind Farm. This is because wind is stochastic and experiences problems during predictions, hence, the purpose of the research. The Tables 5.1, 5.2, and 5.3 on the other hand depicts a typical wind data set from a wind farm. The data captures all the wind information from a Turbine as enumerated in Figure 5.1. Understanding the wind speed in the farm is crucial for the research, hence, the wind-speed histogram computations.

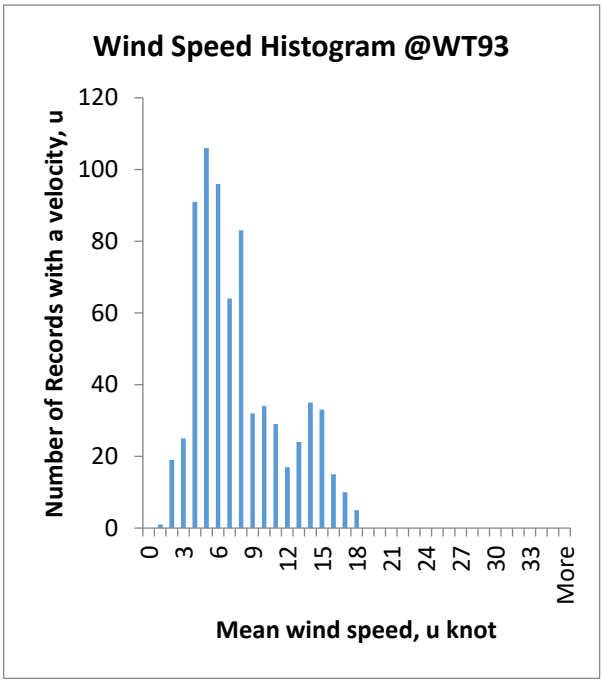
## 5.2. Wind Speed Histogram Computation.

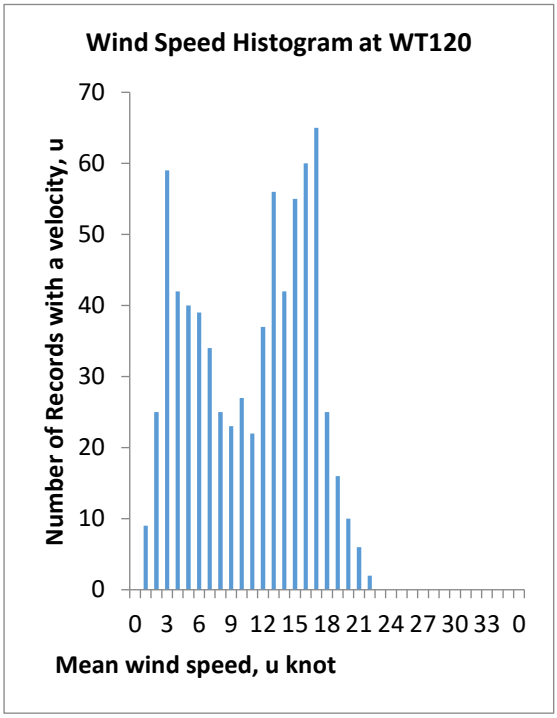
The integral of probability density function (PDF) as discussed in section 4.2, Eq. (4.4) is the cumulative distribution function $F(u)_{CDF}$, which gives the probability of wind speed at or below speed, $U$. From the data, we assumed a k of about 1.798 and obtained the c from the average wind speed of the data. Another assumption made in the research is that the turbine sizes and types are unknown. However, at individual plots of the histogram of Figure 5.5, different turbines within the site experience power differently at their blades.

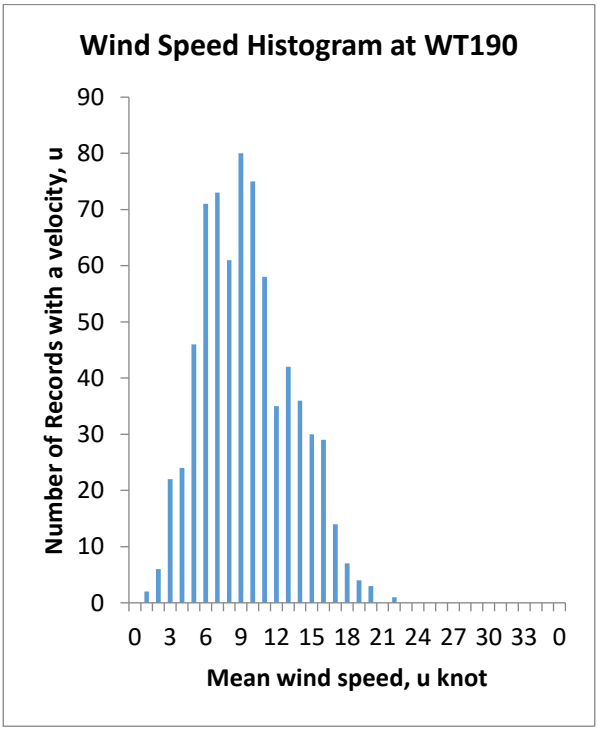


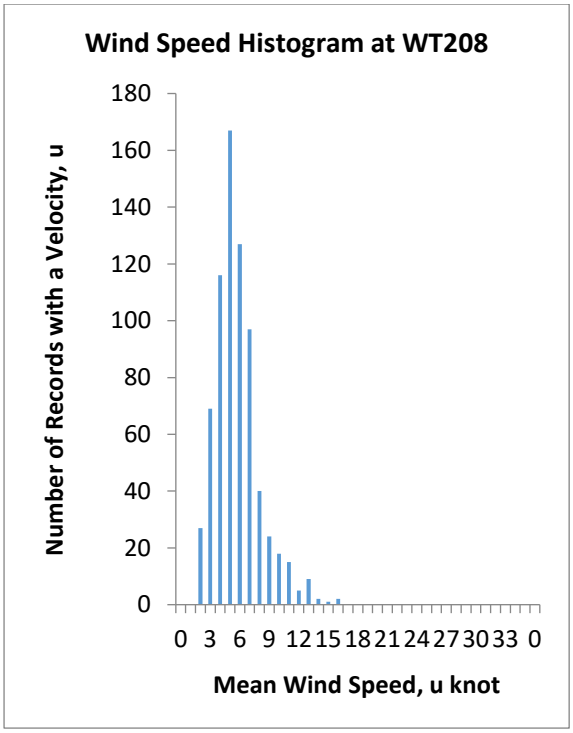



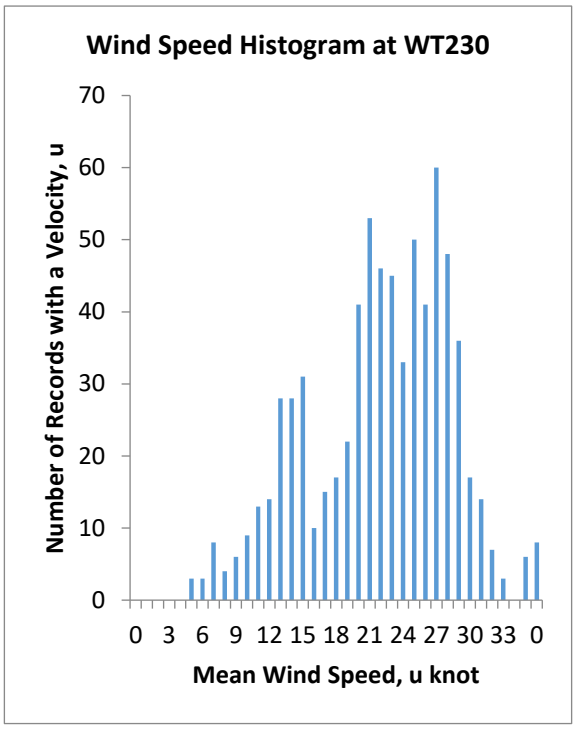

Wind Speed Histogram at WT61
Number of Records with a velocity, u
120
100
80
60
40
20
0
1 4 7 10 13 16 19 22 25 28 31 34
Mean wind speed, u knot

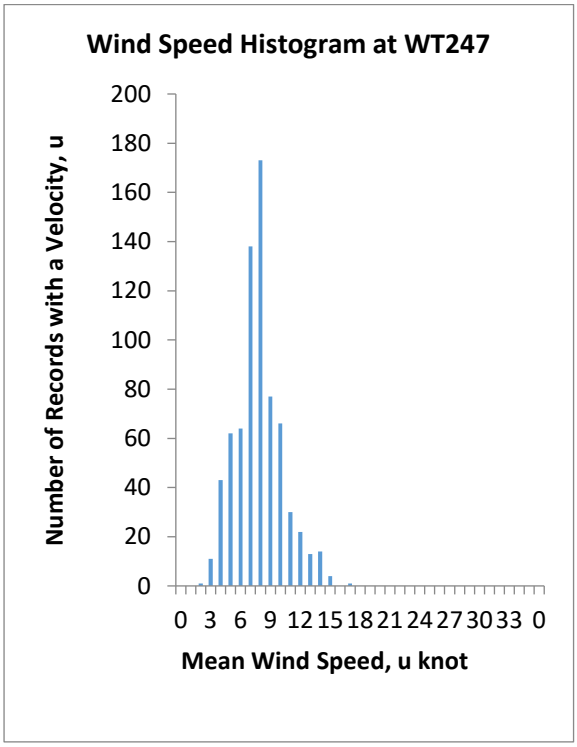

Wind Speed Histogram @WT93
Number of Records with a velocity, u
120
100
80
60
40
20
0
0 3 6 9 12 15 18 21 24 27 30 33 More
Mean wind speed, u knot

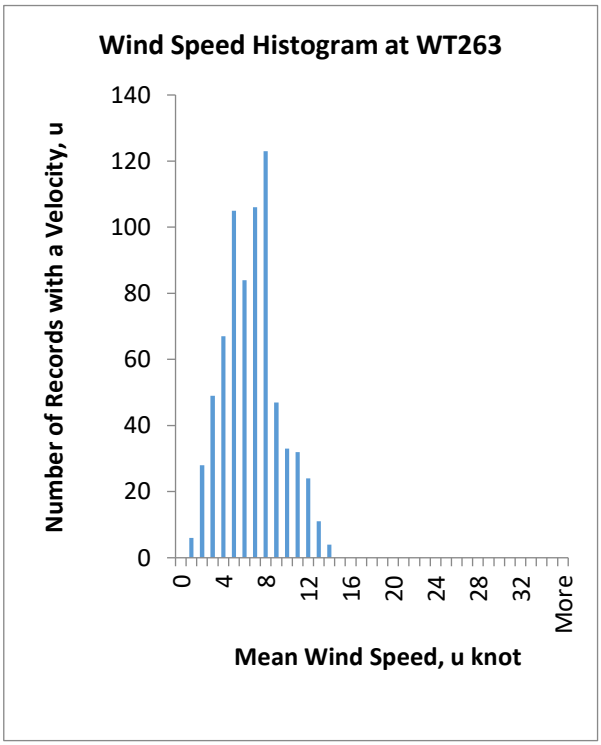

Wind Speed Histogram at WT120
Number of Records with a velocity, u
70
60
50
40
30
20
10
0
0 3 6 9 12 15 18 21 24 27 30 33 0
Mean wind speed, u knot

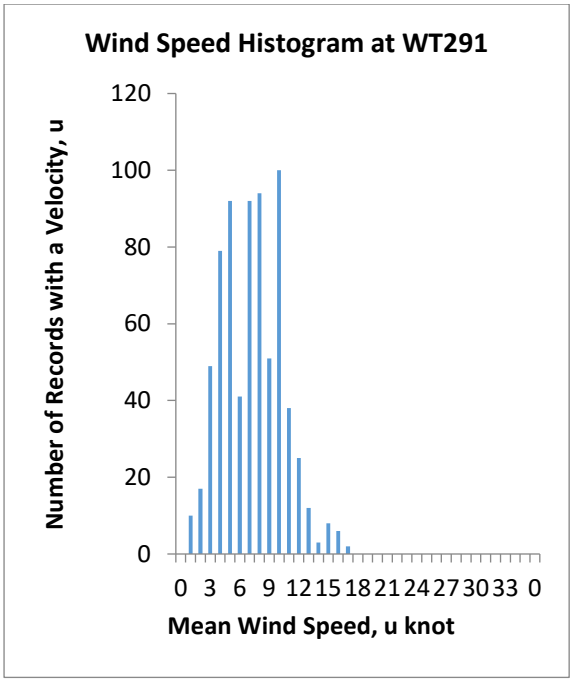

Wind Speed Histogram at WT190
Number of Records with a velocity, u
90
80
70
60
50
40
30
20
10
0
0 3 6 9 12 15 18 21 24 27 30 33 0
Mean wind speed, u knot

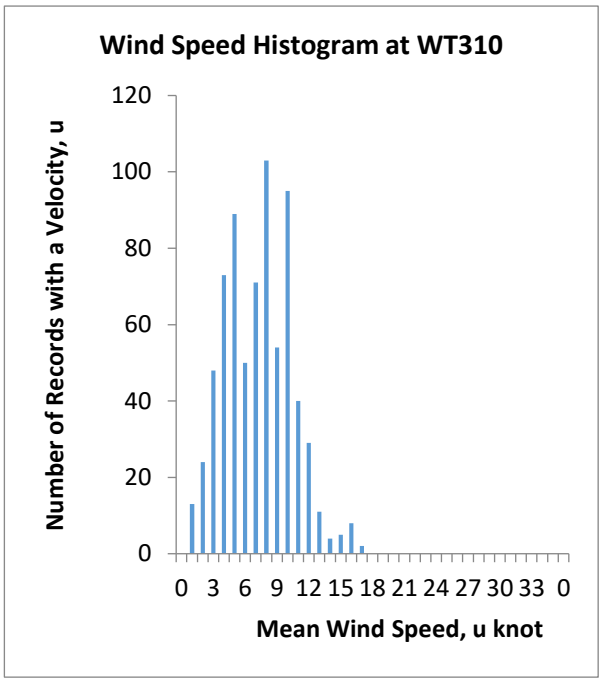

Wind Speed Histogram at WT208
Number of Records with a Velocity, u
180
160
140
120
100
80
60
40
20
0
0 3 6 9 12 15 18 21 24 27 30 33 0
Mean Wind Speed, u knot

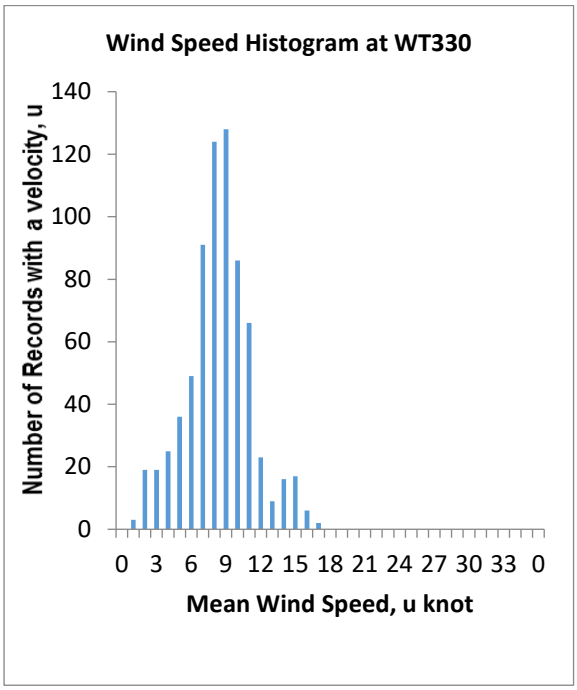

Wind Speed Histogram at WT230
Number of Records with a Velocity, u
70
60
50
40
30
20
10
0
0 3 6 9 12 15 18 21 24 27 30 33 0
Mean Wind Speed, u knot


Wind Speed Histogram at WT247
Number of Records with a Velocity, u
200
180
160
140
120
100
80
60
40
20
0
0 3 6 9 12 15 18 21 24 27 30 33 0
Mean Wind Speed, u knot


Wind Speed Histogram at WT263
Number of Records with a Velocity, u
140
120
100
80
60
40
20
0
0 4 8 12 16 20 24 28 32 More
Mean Wind Speed, u knot



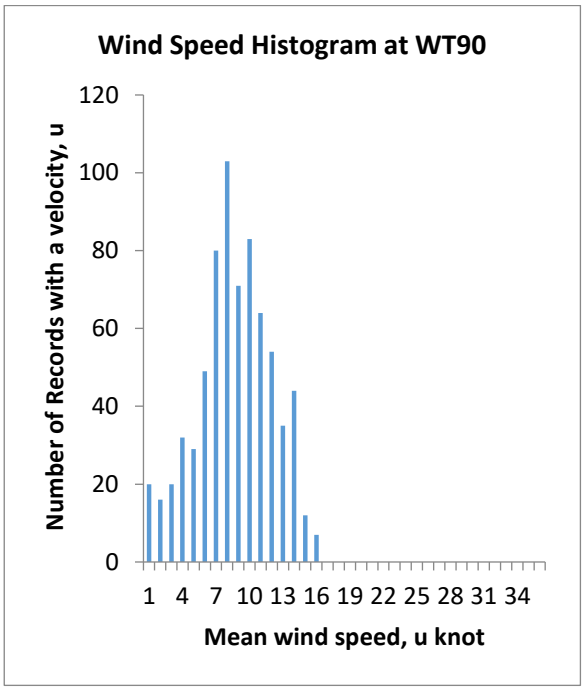




Figure 5. 5: Histogram of Wind Speed from 14 Turbines

Thus, in the absence of wind histogram, wind power can be approximated using the using the cube root of wind speed and Weibull parameters for wind power density (WPDs) estimations.

### 5.2.1. Weibull Distributions from the Individual Turbines.

The statistical distribution of wind power estimation as discussed in section 4.3, applying Weibull distribution of Eq. (4.5) results in Figure 5.6.

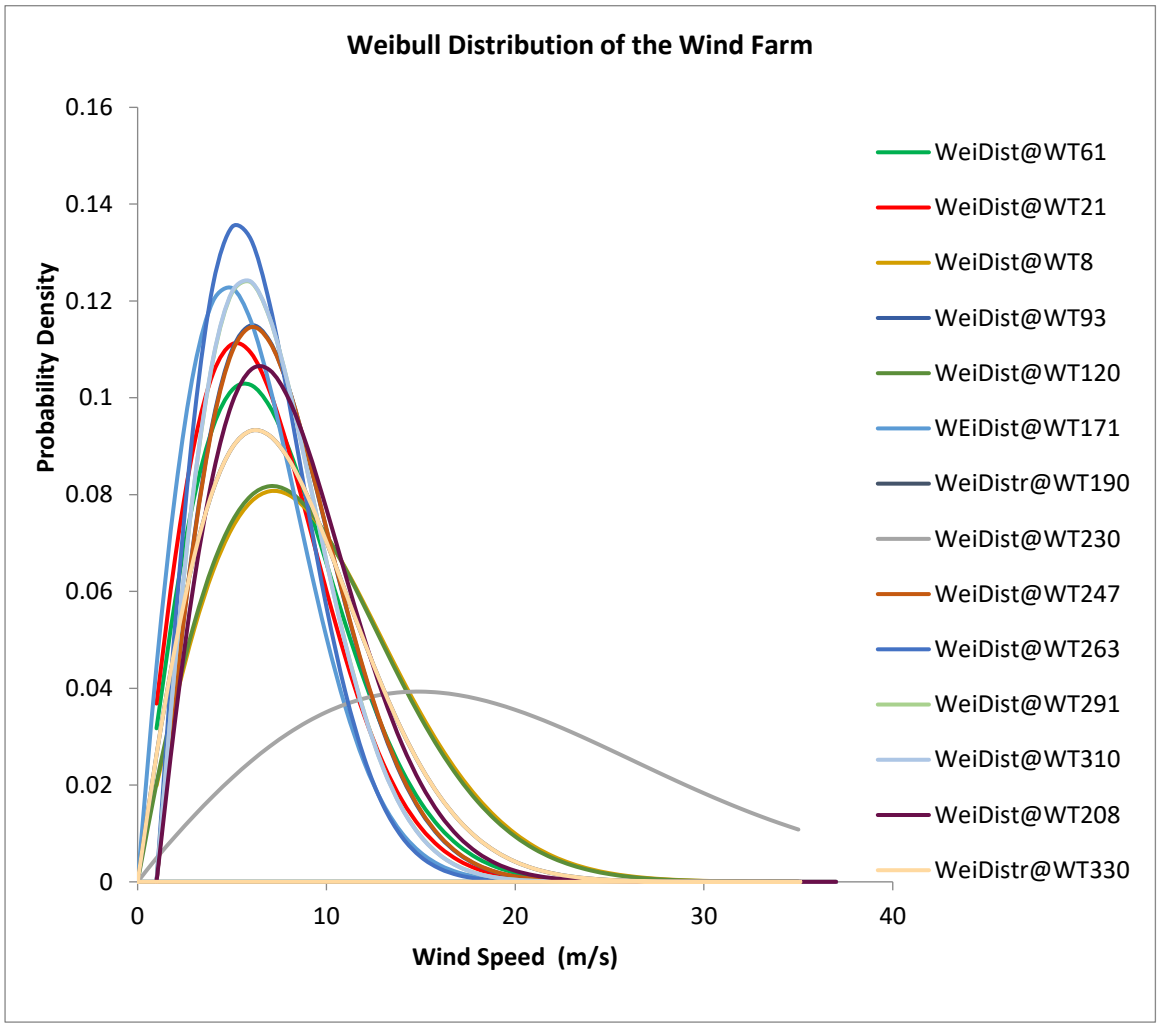



Figure 5. 6: Weibull Distribution from 14-Turbines

From the figure, the research studies the distribution from each turbine to estimate the possible AEP from the farm using wind power density (WPD) estimations of Eq. (4.5 and 4.9) which results to Figure 5.7.

5.2.2. Wind Power Density Estimations within the Farm.

The power in the wind, which is equivalent to the energy potential of the wind, varies as the cube of the wind speed and in proportion to the air density. The distribution of wind energy at different wind speed for different turbines is then computed from the Weibull plot of Figure 5.6 using Eq. (4.7) for WPDs in the wind farm that results in Figure 5.7 below.

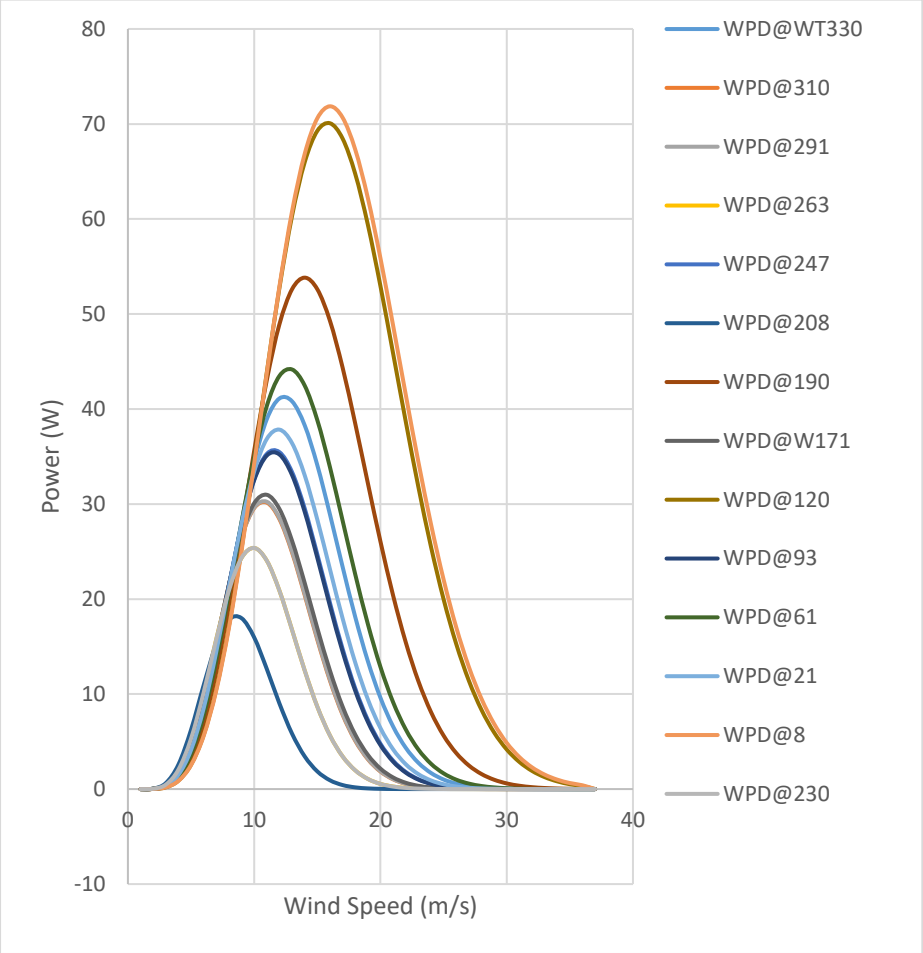



Figure 5. 7: Wind Power Densities for different Turbines.

Other estimations like the wind power capacity, which is WPD *[31] Wind Turbine Surface Area * Efficiency of the Turbine are not considered in this research due to lack of adequate data. However, estimated power generation output from the wind is computed, which is WPD

[31] Multiplication sign (X)

measured with the Weibull parameters is tabulated in Table 5.4 alongside the power from the turbine.

Table 5. 4: Wind-Farm Power Generation

| Wind Turbines | AVGWPD(m/s^2 | Lambda@AVGws | u_bar | Power@Wind (W/m^3) | WPD@Wi |
|---|---|---|---|---|---|
| WT8 | 895.8627779 | 10.44830556 | 9.264612 | 487.0659179 | 953.9141 |
| WT21 | 457.9821697 | 7.582375 | 6.723364 | 186.1512363 | 364.8042 |
| WT61 | 500.343325 | 8.196708333 | 7.268099 | 235.1627833 | 460.8524 |
| WT93 | 468.792354 | 7.340097222 | 6.508534 | 168.8711977 | 330.9402 |
| WT120 | 1219.255404 | 10.32066667 | 9.151433 | 469.4327678 | 919.4984 |
| WT171 | 362.7787935 | 6.862263889 | 6.084834 | 137.9915177 | 270.4247 |
| WT190 | 706.4223795 | 9.042652778 | 8.018206 | 315.745894 | 618.7567 |
| WT208 | 148.2032846 | 5.261458333 | 4.665385 | 62.19687333 | 121.8885 |
| WT230 | 7589.375903 | 21.47227778 | 19.03967 | 4227.508574 | 5083.089 |
| WT247 | 323.789077 | 7.362625 | 6.528509 | 170.4308422 | 333.9967 |
| WT263 | 228.6459775 | 6.21 | 5.506466 | 102.2645113 | 200.4098 |
| WT291 | 316.9938616 | 6.786930556 | 6.018036 | 133.4966486 | 261.616 |
| WT310 | 319.3036394 | 6.7775 | 6.009673 | 132.9409339 | 260.527 |
| WT330 | 423.2603788 | 7.91925 | 7.022074 | 212.0812656 | 415.6196 |
| Total | 13961.00933 | 121.5831111 | 107.8089 | 7041.340963 | 10596.34 |

From the table, results of average wind speed and average velocity (u_bar) are estimated. The u_bar is a function of the Weibull parameters from the individual turbines. It demonstrates the actual converted wind energy reaching the blades of the turbines into electricity; hence, it is compared to wind speed as shown in Figure 5.8.

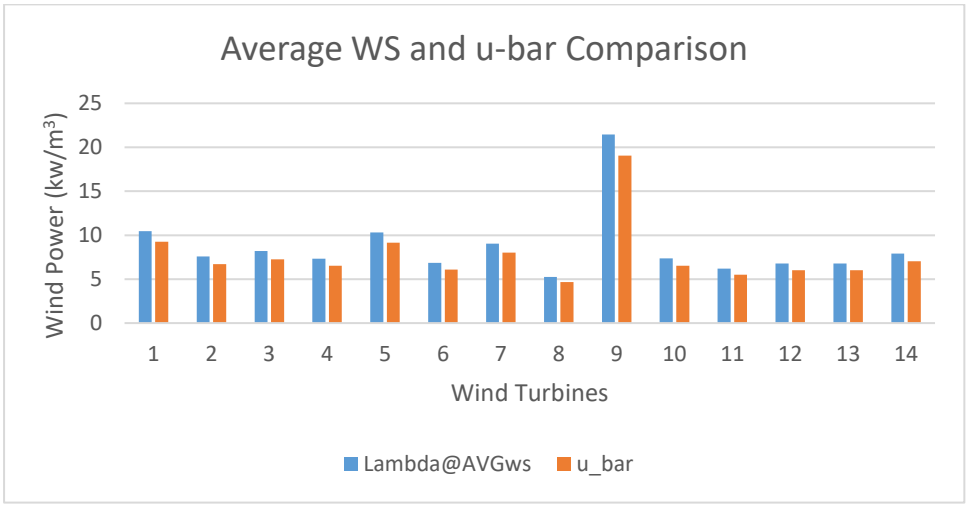



Figure 5. 8: Average Wind Speed and Velocity Comparison.

To understand the average WPD at each wind turbine and the percentage of time at each wind velocity (bin), Figure 5.9 is used

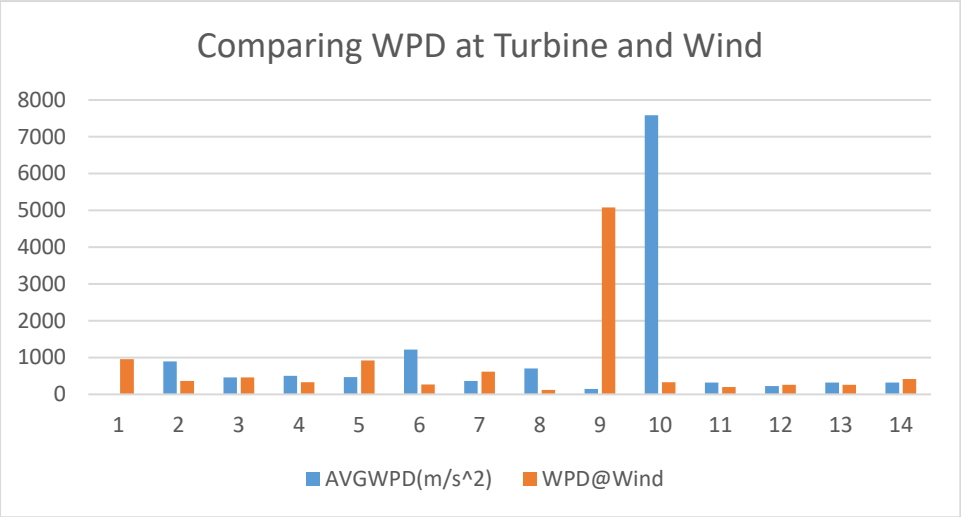



Figure 5. 9: WPD Comparison.

The power delivered to each of the wind turbines, computed from Table 5.2 is shown in Figure 5.10.

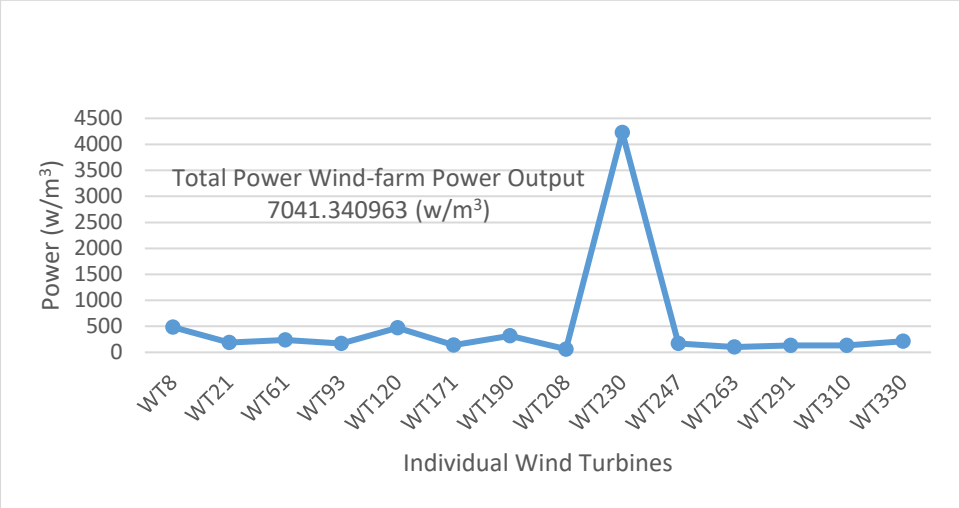



Figure 5. 10: Power Generated by the Wind farm.

5.2.3. Wind Power Output Estimations.

The area under each curve of Figure 4.9 informs the amount of electrical wind power that can be converted theoretically to mechanical power according to Betz law, which is 16/27 of total power in the wind at individual turbines. The power curve generated from the computed wind power density estimated from the Weibull parameters is as shown in Figure 5.11. From the figure, individual power densities from each turbine are combined to show the plot of the wind power curve.

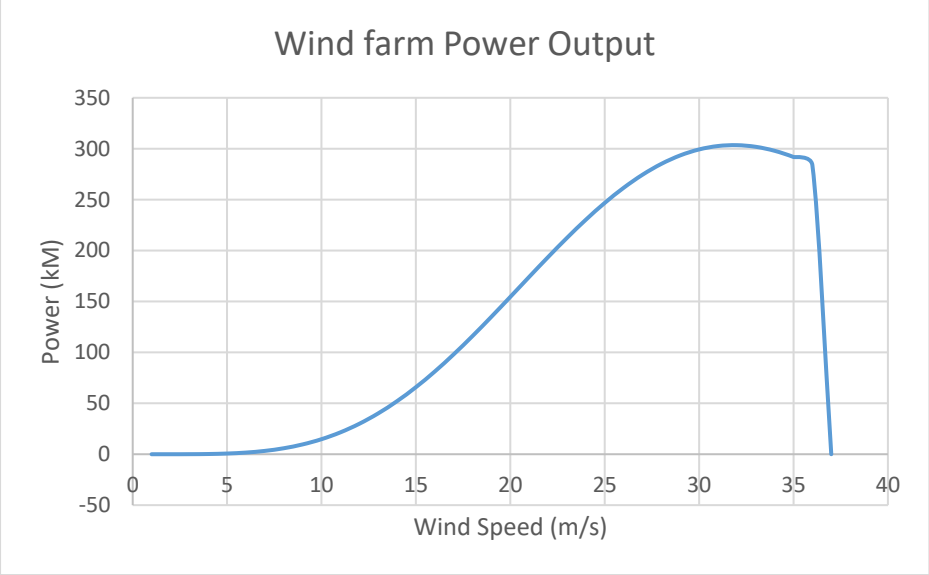



Figure 5. 11: Generated WPD from the Wind farm.[32]

Assumptions are made in generation of Figure 5.11 above. These assumptions are (i) the air density from the wind farm is $1kg/m^3$. (ii) Fourteen standard 1.2WM Turbines with blade swept areas of $1m^2$ each are applied to the power curve of Eq. 4.10. (iii) Average wind speed

[32] This figure is generated based on assumptions, simply to depict typical wind farm power output.

of 5m/s. Comparing Figure 5.11 and Figure 4.3, the later considers wind turbines of different sizes with different blades swept areas in wind farm while the former is as assumed.

5.2.4. Results.

It is observed from the individual plots that WT230 experiences outliers in the generated data from the sensor, hence, the abnormalities shown in Figure 5.10. However, the actual wind reaching the turbine blades from Figure 5.9 is as expected in the sense that the literature deduces low tip wind speed as seen in Figure 4.10 where the actual wind density is higher than computed WPD from Weibull parameters. The typical power output from the wind is as shown in Figure 5.10. Understanding the power output of the wind farm under study using Figure 5.11 is crucial. However, in the farm, the average cut in wind speed is around 8.5 m/s while the rated power is around 25 m/s. at 35 m/s, the wind farm is expected to shut down to avoid damaging the wind farm. The farm loses power at this area.

To understand the data from the farm at individual turbines, descriptive statistics demonstrates pattern relationships within the data and measures of wind speed variability within the farm.

## 5.3. Wind Farm Data Descriptive Statistics

The descriptive statistics are determined using Microsoft Excel Software (MExS). Similar values are as obtained with Eq. (4.13) through (4.17). The results are as presented in Table 5.5.

Table 5. 5: Descriptive Statistics on the Individual Turbine data

| | WT330 | WT310 | WT291 | WT263 | WT247 | WT230 | WT208 | WT190 | WT171 | WT120 | WT93 | WT61 | WT21 | WT8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mean | 0.534 | 0.356 | 0.283 | 0.145 | 0.497 | 0.590 | 0.301 | 0.204 | 0.362 | 0.386 | 0.128 | 0.936 | 0.610 | 0.550 |
| Standard Error | 0.055 | 0.039 | 0.033 | 0.038 | 0.008 | 0.006 | 0.027 | 0.019 | 0.018 | 0.054 | 0.020 | 0.054 | 0.004 | 0.014 |
| Median | 0.008 | 0.102 | 0.116 | 0.161 | 0.527 | 0.633 | 0.200 | 0.336 | 0.536 | 0.252 | 0.211 | 0.534 | 0.639 | 0.678 |
| Mode | 0.496 | 0.320 | 0.230 | 0.725 | 0.725 | 0.714 | 0.392 | 0.112 | 0.725 | 0.725 | 0.725 | 2.348 | 0.725 | 0.725 |
| Standard Deviation | 1.485 | 1.046 | 0.898 | 1.021 | 0.220 | 0.173 | 0.718 | 0.517 | 0.470 | 1.451 | 0.544 | 1.461 | 0.116 | 0.385 |
| Sample Variance | 2.204 | 1.094 | 0.807 | 1.041 | 0.049 | 0.030 | 0.515 | 0.267 | 0.221 | 2.105 | 0.296 | 2.136 | 0.013 | 0.148 |
| Kurtosis | 2.390 | 1.231 | 3.559 | 5.276 | 1.476 | 131.772 | 0.232 | 2.127 | 4.770 | 0.675 | 5.013 | 0.158 | 14.299 | 112.417 |
| Skewness | 1.589 | 1.243 | 1.623 | 2.121 | 1.178 | 8.248 | 0.739 | 1.454 | 2.122 | 1.414 | 1.798 | 0.942 | 2.602 | 8.160 |
| Range | 7.984 | 5.428 | 5.097 | 5.985 | 1.188 | 3.173 | 3.829 | 2.877 | 2.545 | 5.869 | 3.653 | 6.710 | 1.178 | 6.552 |
| Minimum | 0.725 | 0.724 | 0.725 | 0.725 | -0.725 | 0.725 | 0.710 | 0.724 | 0.725 | 0.725 | -0.725 | -0.725 | 0.725 | 0.725 |
| Maximum | 7.259 | 4.704 | 4.372 | 5.261 | 0.464 | 2.449 | 3.119 | 2.153 | 1.820 | 5.144 | 2.928 | 5.986 | 0.453 | 5.827 |
| Sum | 384.37 | 256.46 | 203.63 | 104.10 | 357.52 | 425.04 | 216.67 | 146.63 | 260.99 | 278.02 | 92.22 | 673.65 | -439.36 | 395.15 |
| Count | 720.000 | 720.000 | 720 | 720 | 720 | 720 | 720 | 720 | 720 | 720 | 720 | 720 | 720 | 720 |
| Confidence Level(9 | 0.109 | 0.077 | 0.066 | 0.075 | 0.016 | 0.013 | 0.053 | 0.038 | 0.034 | 0.106 | 0.040 | 0.107 | 0.008 | 0.028 |

From the Table above, the sum and kurtosis of the wind speed data demonstrates the sum of individual wind turbines within the site assuming a flat surface.

5.3.1. Observations on Other Generated Wind Data.

The generated wind power has wind speed as the major variable contributor. Although individual data samples – temperature, wind shear and turbulence contributes to the power generation, the blade and generator efficiencies contribute significantly to the wind power generation. The sample mean and median of the wind speed are not close. The mean is considerably larger than the median, making Lognormal or Weibull distribution the best candidate distribution. The mean and standard deviation are close. This suggests that the prediction process could be exponential.

Furthermore, there is a close relationship of the Lognormal, Exponential and Weibull distributions from the generated wind power data. This necessitates taking a step further in pinpointing the distribution. The least square fit to the distributions estimates a high index of fit. In addition, the relationship within the generated turbulence and wind shear on the wind speed data as observed conforms to the literature on the farm. Hence, shown in Figure 5.12 and 5.13 respectively.

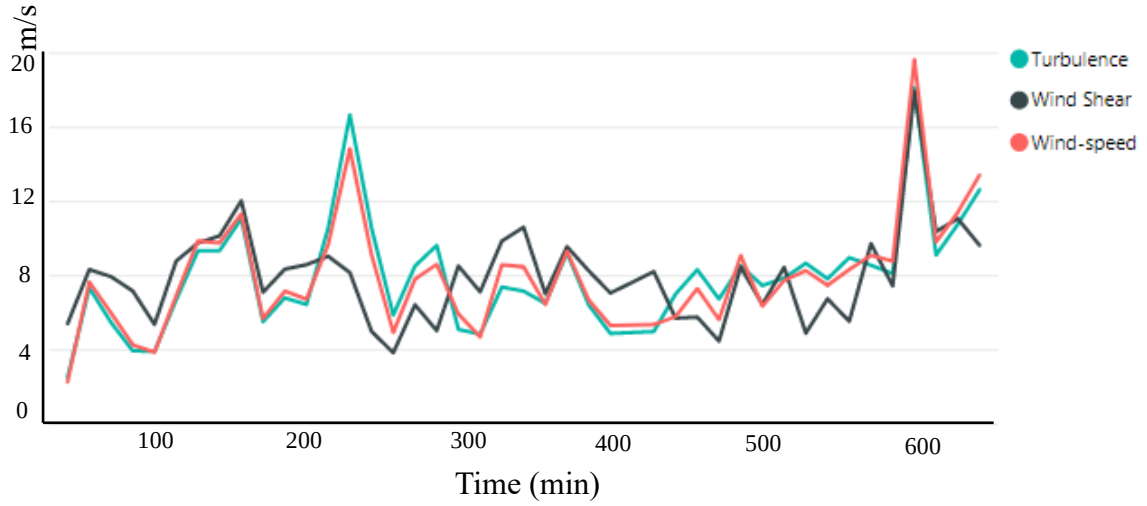



Figure 5. 12: Turbulence and Wind shear effect on wind speed

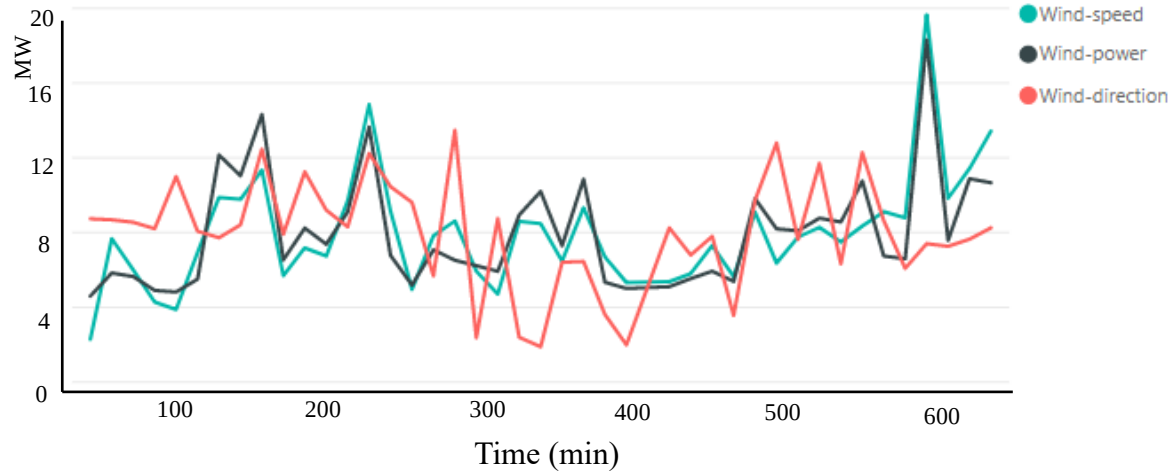



Figure 5. 13**:** Wind power, wind speed and direction relationship.

5.3.2. Least Squares Fitting.

A better accurate fit to the data is obtained by performing a least squares fit of the tested distributions using Eq. (4.18) through (4.28) for the weibull distribution and Eq. (4.19) for the exponential distribution. Eq. (4.22) and (4.24) demonstrates the normal and lognormal

distributions respectively. Table 5.6 shows the plotting positions for the distributions. However, Figure 5.14 through 5.17 shows the least square plots for the various distributions considering a sample of the maximum generated wind power within the farm.

Table 5. 6**:** Least Square Plotting Positions of Various Distributions of wind speed

| *i* | **Wind speed (WS)** | **LnWS** | **F(t)= i-0.3/n+0.4** | **1/[1-F(t)]** | **Ln[1/1-F(t)]** | **Lnln[1/1-F(t)]** | **Zi** |
|---|---|---|---|---|---|---|---|
| 1 | 2.800 | 1.030 | 0.023026 | 1.023569 | 0.023296 | -3.75949 | 0.509185 |
| 2 | 2.290 | 0.829 | 0.055921 | 1.059233 | 0.057545 | -2.85518 | 0.522298 |
| 3 | 2.640 | 0.971 | 0.088816 | 1.097473 | 0.09301 | -2.37505 | 0.535386 |
| 4 | 2.710 | 0.997 | 0.121711 | 1.138577 | 0.129779 | -2.04192 | 0.548436 |
| 5 | 3.140 | 1.144 | 0.154605 | 1.182879 | 0.167952 | -1.78408 | 0.561434 |
| 6 | 1.980 | 0.683 | 0.1875 | 1.230769 | 0.207639 | -1.57195 | 0.574366 |
| 7 | 3.170 | 1.154 | 0.220395 | 1.2827 | 0.248968 | -1.39043 | 0.587218 |
| 8 | 2.330 | 0.846 | 0.253289 | 1.339207 | 0.292078 | -1.23074 | 0.599978 |
| 9 | 2.370 | 0.863 | 0.286184 | 1.400922 | 0.33713 | -1.08729 | 0.612631 |
| 10 | 2.800 | 1.030 | 0.319079 | 1.468599 | 0.384309 | -0.95631 | 0.625167 |
| 11 | 2.610 | 0.959 | 0.351974 | 1.543147 | 0.433824 | -0.83512 | 0.637571 |
| 12 | 3.810 | 1.338 | 0.384868 | 1.625668 | 0.485919 | -0.72171 | 0.649833 |
| 13 | 3.950 | 1.374 | 0.417763 | 1.717514 | 0.540878 | -0.61456 | 0.66194 |
| 14 | 3.270 | 1.185 | 0.450658 | 1.820359 | 0.599034 | -0.51244 | 1.0 |

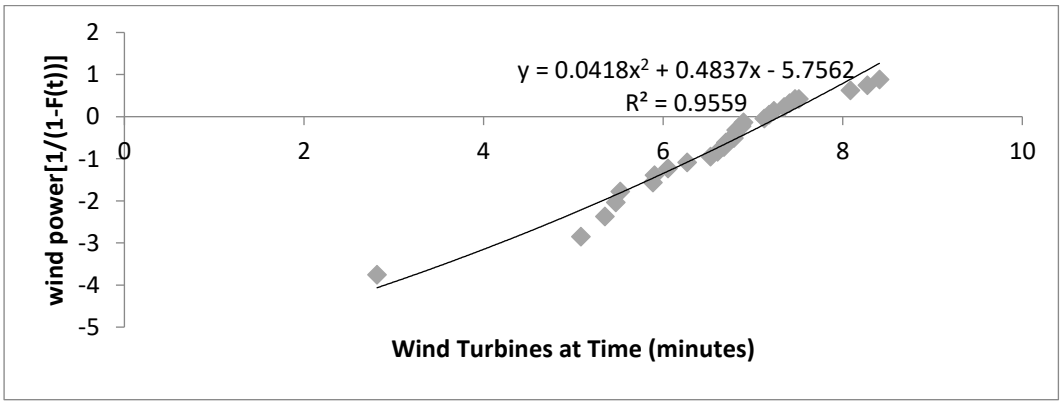



Figure 5. 14**:** Weibull Least Square Plot of wind speed

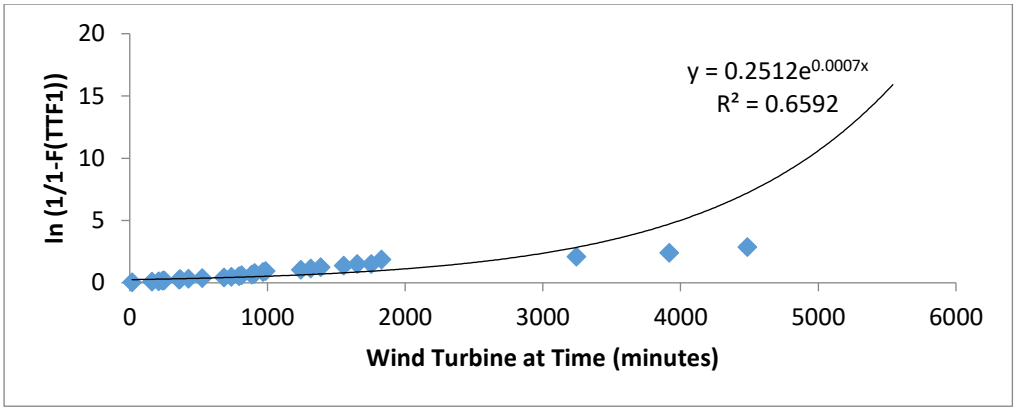



Figure 5. 15**:** Exponential Least Square Plot of wind speed

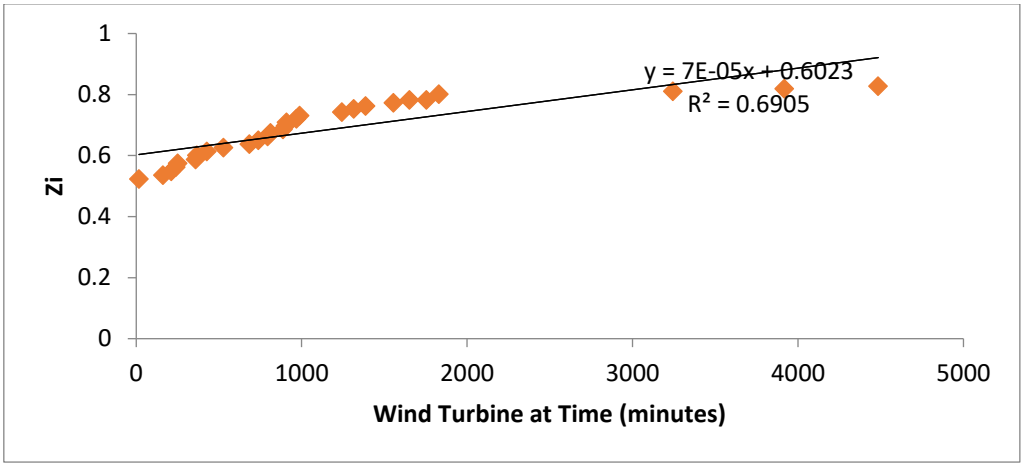



Figure 5. 16**:** Normal Least Square Plot of wind speed

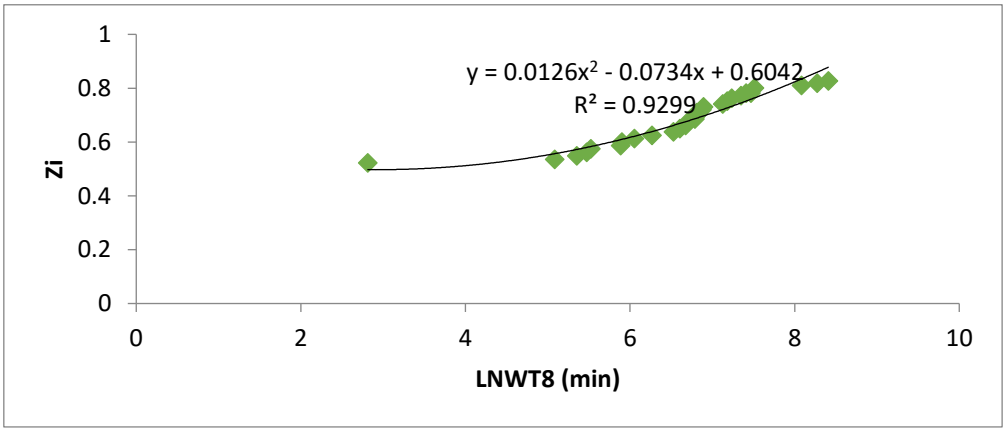



Figure 5. 17**:** Lognormal Least Square Plot of wind speed

5.3.3 Coefficient of Determination.

The coefficient of determination (R-Squared), compares estimated and actual y-values, and ranges in value from 0 to 1. If it is 1, there is a perfect correlation depicted in Table 5.7 of the generated wind sample — there is no difference between the estimated y-value and the actual y-value. At the other extreme, if the coefficient of determination is 0, the regression becomes unhelpful in predicting a y-value.

**Table 5.7:** Correlation of Wind Data Samples.

| Attributes | Relative-humitiy | Temperature | Time | Turbulence | Wind-direction | Wind-power | Wind Shear | Wind-speed |
|---|---|---|---|---|---|---|---|---|
| Relative-humitiy | 1 | 0.204 | -0.050 | -0.201 | -0.079 | -0.070 | 0.325 | -0.136 |
| Temperature | 0.204 | 1 | -0.480 | -0.397 | -0.234 | -0.261 | 0.092 | -0.321 |
| Time | -0.050 | -0.480 | 1 | 0.226 | -0.099 | 0.149 | 0.087 | 0.241 |
| Turbulence | -0.201 | -0.397 | 0.226 | 1 | 0.171 | 0.804 | 0.323 | 0.953 |
| Wind-direction | -0.079 | -0.234 | -0.099 | 0.171 | 1 | 0.142 | -0.006 | 0.124 |
| Wind-power | -0.070 | -0.261 | 0.149 | 0.804 | 0.142 | 1 | 0.443 | 0.853 |
| Wind Shear | 0.325 | 0.092 | 0.087 | 0.323 | -0.006 | 0.443 | 1 | 0.456 |
| Wind-speed | -0.136 | -0.321 | 0.241 | 0.953 | 0.124 | 0.853 | 0.456 | 1 |

The resultstant R-squared statistics of each distribution is shown in Table 5.8.

**Table 5.8:** Least-Square Fitting Results for Wind Speed

| S/N | Distribution | R-Square value |
|---|---|---|
| 1 | Weibull | 0.9459 |
| 2 | Exponential | 0.9395 |
| 3 | Lognormal | 0.8379 |
| 4 | Normal | 0.6905 |

5.3.4. Maximum Likelihood Estimation

The MLE for the parameters, $\beta$ and $\theta$ are determined using Eq. (4.29) through (4.31). Where $t_s = 1, n = r = 14$. Applying Newton Raphson's numerical method, which requires solving for $\mathcal{Y}$ iteratively, gives:

$$\hat{\mathcal{Y}}_{j+1} = \hat{\mathcal{Y}}_j - \frac{g(\mathcal{Y}_j)}{g'(\mathcal{Y}_j)} \tag{5.3}$$

where

$$g'(\hat{\mathcal{Y}}) = \frac{d}{d\hat{\beta}}\left[\frac{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s}{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}}}\right] - \frac{d}{d\hat{\beta}}\left(\frac{1}{\hat{\beta}}\right) - \frac{d}{d\hat{\beta}}\left[\frac{1}{r}\sum_{i=1}^{r} lnt_i\right]$$

$$= \frac{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\mathcal{Y}} t_s\right]\frac{d}{d\hat{\beta}}\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s\right] - \left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s\right]\frac{d}{d\hat{\beta}}\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}} t_s\right]}{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}}\right]^2} +$$

$$\frac{1}{\hat{\mathcal{Y}}^2} = 0$$

$$g'(\hat{\mathcal{Y}}) =$$

$$\frac{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}} t_s\right]\left\{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} (lnt_i)^2 + (n-r)t_s^{\hat{\mathcal{Y}}} (lnt_s)^2\right\} - \left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s\right]\left\{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s\right\}}{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}}\right]^2}$$

$$+\frac{1}{\hat{\mathcal{Y}}^2} = 0$$

$$g'(\hat{\mathcal{Y}}) = \frac{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}} t_s\right]\left\{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} (lnt_i)^2 + (n-r)t_s^{\hat{\mathcal{Y}}} (lnt_s)^2\right\} - \left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i + (n-r)t_s^{\hat{\mathcal{Y}}} lnt_s\right]^2}{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} + (n-r)t_s^{\hat{\mathcal{Y}}}\right]^2} + \frac{1}{\hat{\beta}^2} = 0 \tag{5.4}$$

For incomplete or censored data, Eq. (5.4) holds. But for complete data like the data used in this study, $n = r$. Hence, Eq. (5.4) reduces to:

$$g'(\hat{\mathcal{Y}}) = \frac{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}}\right]\left\{\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} (lnt_i)^2\right\} - \left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}} lnt_i\right]^2}{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}}\right]^2} + \frac{1}{\hat{\mathcal{Y}}^2} = 0 \tag{5.4a}$$

$$g'(\hat{\mathcal{Y}}) = \frac{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}}\right]^2 (lnt_i)^2 - \left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}}\right]^2 (lnt_i)^2}{\left[\sum_{i=1}^{r} t_i^{\hat{\mathcal{Y}}}\right]^2} + \frac{1}{\hat{\mathcal{Y}}^2} = 0 \tag{5.5}$$

$$g'(\hat{y}) = \frac{1}{\hat{\beta}^2} \tag{5.5a}$$

The initial estimate for $\hat{y}$ (0.9786) was obtained from least squares fit. See Table 5.8 for an improved value of $\hat{y}$ obtained through Newton Raphson numerical method.

$\hat{y} = 0.9459$ Remembering that $n = r$, Eq. (4.30) reduces to

$$\hat{\theta} = \left\{\frac{1}{r}\left[\sum_{i=1}^{r} t_i^{\hat{y}}\right]\right\}^{\frac{1}{\hat{y}}} \tag{5.6}$$

Hence,

$\hat{\theta} = 1229.298$

5.3.5. MANN'S Test.

The confirmation of Weibull distributions for ease of predictions results in the use of Eq. (4.43) through (4.44) as the final test that the data of wind farm (WF) came from the Weibull distribution and is merge-able for wind speed prediction. The value of α is set at 0.05. The null hypothesis is:

$H_0$: wind series are Weibull with $y = 0.92459$ and $\theta = 1229.298$

$H_1$: wind series are not Weibull with $y = 0.92459$ and $\theta = 1229.298$

Table 5.9 provides the computed parameters of Eq. (4.43) and (4.44), from the table,

$$M = \frac{K_1 \sum_{i=16}^{29}[(\boldsymbol{lnWindFarm_{i+1} - lnWindFarm_i})/\boldsymbol{M_i}]}{K_2 \sum_{i=1}^{15}[(\boldsymbol{lnWindFarm_{i+1} - lnWindFarm_i})/\boldsymbol{M_i}]} \tag{5.6a}$$

where

$$K_1 = \left\lfloor\frac{r}{2}\right\rfloor = 14, \quad K_2 = \left\lfloor\frac{r-1}{2}\right\rfloor = 14, \tag{5.6b}$$

Numerator $= -428.11$, Denominator $= -236.28$, `$M = 1.579$ with 28 degrees of freedom for both the numerator and the denominator. Since $M = 1.579 < F_{crit,0.05,28,28} = 1.868$, $H_0$

is accepted. Therefore $\mathcal{Y} = 0.924$ and $\theta = 122.20$. Values of F critical are obtained from critical values for the F-distribution.

Table 5. 7**:** Determination of Mann's Test for Wind Power (WP) data

| $i$ | $WT1$ | $lnWT$ | $\frac{i-0.5}{n+0.25}$ | $Z_i$ | $M_i$ | $lnWT_{i+1} - lnWT_i$=H | $\frac{H}{M_i}$ |
|---|---|---|---|---|---|---|---|
| 1 | 2.800 | 1.030 | 0.016529 | -4.09432 | 1.115612 | 1.034469 | 0.927266 |
| 2 | 2.290 | 0.829 | 0.049587 | -2.97871 | 0.528324 | 0.819956 | 1.551993 |
| 3 | 2.640 | 0.971 | 0.082645 | -2.45039 | 0.354503 | -1.31733 | -3.716 |
| 4 | 2.710 | 0.997 | 0.115702 | -2.09588 | 0.269914 | -2.10815 | -7.81046 |
| 5 | 3.140 | 1.144 | 0.14876 | -1.82597 | 0.219879 | -2.99936 | -13.641 |
| 6 | 1.980 | 0.683 | 0.181818 | -1.60609 | 0.186916 | -1.01729 | -5.44249 |
| 7 | 3.170 | 1.154 | 0.214876 | -1.41917 | 0.163666 | -4.04191 | -24.696 |
| 8 | 2.330 | 0.846 | 0.247934 | -1.25551 | 0.146489 | -1.88158 | -12.8445 |
| 9 | 2.370 | 0.863 | 0.280992 | -1.10902 | 0.133376 | -1.54695 | -11.5984 |
| 10 | 2.800 | 1.030 | 0.31405 | -0.97564 | 0.123131 | -1.34726 | -10.9417 |
| 11 | 2.610 | 0.959 | 0.347107 | -0.85251 | 0.115 | -2.55855 | -22.2483 |
| 12 | 3.810 | 1.338 | 0.380165 | -0.73751 | 0.108486 | -2.62843 | -24.2283 |
| 13 | 3.950 | 1.374 | 0.413223 | -0.62903 | 0.103253 | -3.84004 | -37.1907 |
| 14 | 3.270 | 1.185 | 0.446281 | -0.52577 | 0.099068 | -2.42461 | -24.4741 |

5.3.6. Confidence Intervals.

In order to ascertain that the result obtained from the research maybe be ascribed to chance, a confidence interval is applied. More than 90 percent confidence intervals for the estimated parameters are computed directly from Eq. (4.32) and (4.31) to result in Eq. (5.6c) and (5.6d)

$$0.92459\exp\left(\frac{-0.78Z_{\alpha/2}}{\sqrt{30}}\right) \leq \beta \leq 0.92459\exp\left(\frac{0.78Z_{\alpha/2}}{\sqrt{30}}\right) \quad (5.6c)$$

$$0.73149 \leq \boldsymbol{y} \leq 1.16866$$

Also,

$$1229.298\exp\left(\frac{-1.05Z_{\alpha/2}}{0.92459\sqrt{30}}\right) \leq \theta \leq 1229.298\exp\left(\frac{1.05Z_{\alpha/2}}{0.92459\sqrt{30}}\right) \quad (5.6d)$$

$$874.041 \leq \boldsymbol{\theta} \leq 1728.938$$

Where

$Z_{0.1/2} = 1.645$ is the standardized normal deviation obtained from statistical Table of critical t values with v degrees of freedom.

5.3.7. Stationary Modeling of Wind Farm Data.

The statistical method described in section 4.4 is used in the research. It ensures that wind speed from all the 14 turbines are combined to form Figure 5.18. Then the data as modelled for prediction using the proposed LSTM, eLSTM and ARIMA model.

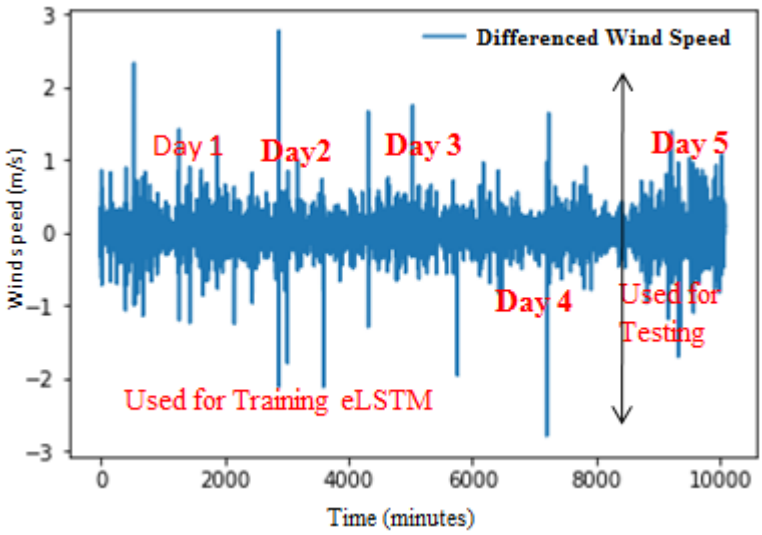



Figure 5. 18**:** Standardised data of a wind speed data

The normal distribution of the wind farm data, computed using Eq. (4.31) is as depicted as shown in Figure 5.19.

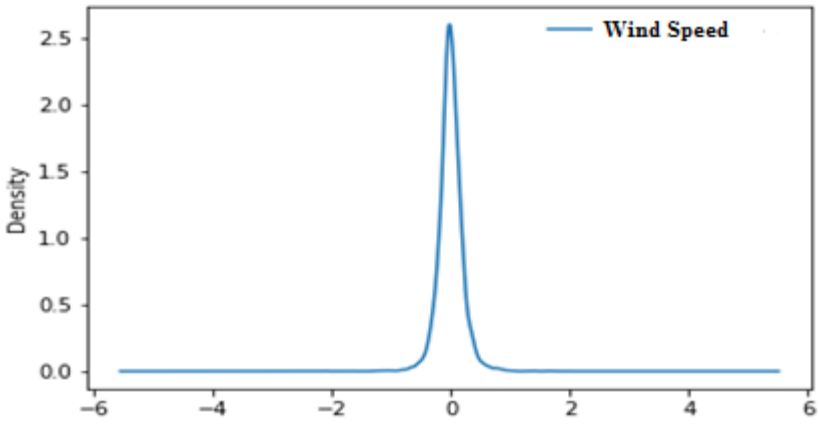



Figure 5. 19**:** Normal distribution of the Wind Farm.

5.3.8 Results.

The maximum likelihood estimation and MANNs test suggested that the individually generated wind speed data is suitable for merging and combined for power output predictions. The CDF and PDF on the other hand ensures the continuousness of the merged data, hence it is available for normalisation and model fitting. Looking at the wind literature, the power output

estimations from the wind farm are estimated using Figure 5.19. This Figure is derived from the cumulative Weibull distribution of Eq. (4.5) of section 4.2.1.

## 5.4. Data Preparation for machine Learning Model.

After due statistical processes to confirm the emergence of the wind data from different turbines using the Mann Test, PDF and CDF on Weibull distribution, it is observed that the data is set for merging such that algorithm models can be applied for predictions. Conversely, the wind data is transformed such that our model can fit to it. Hence, the research employed these three basic steps to achieve this:

- Transform series data to be stationary. In order words, have a lag = 1 using the Dickey Fuller statistical method – Figure 4 was realised. We computed the first level (d = 1) differencing using the difference between current series ($\gamma_t$) and previous series ($\gamma_{t-1}$) as in $\Delta\gamma_t = \gamma_t - \gamma_{t-1}$.
- Transform our series into a supervised learning problem. Here the author specifically used feature engineering to have data in an input/output pattern such that at prior steps, observations are used as input to predict observation at current time step. The window method is applied.
- Transform our observation to have a specific scale – that between -1 and 1. These transforms were then reverted after the prediction to return them into their original scale before errors are calculated and scored.

### 5.4.1. Preparation (Normalization) of Data of Input Parameters

The PHM society's data, the wind-power-generated data of Eq. (4.1) cannot be used in its present form to develop an RNN-LSTM model. In accordance with sections 4.2 and 4.2.1, the input data of the wind farm is rank-ordered and normalized using Eq. (4.12). The results of the

normalized data as presented in Table 5.8 are used to develop the RNN-LSTM model for the research.

$$pn = \frac{2(p-p_{min})}{p_{max}-p_{min}} - 1 \quad (5.7)$$

The output data of the wind power data (table 5.14c) is rank-ordered and normalized using Eq. (4.12). The results presented in Table 5.8 (normalized data) were used to develop the RNN-LSTM model for the unit. For example, from Table 4.14c, $R(t_1 = p = 0.780375)$ was normalized to -0.70752 in Table 5.8 using Eq. (4.12) as follows:

$$pn = \left[\frac{2(0.780375-0.044982)}{0.867-0.044982}\right] - 1 = -0.70752 \quad (5.8)$$

5.4.2. Network Architecture.

Generally, ANN architectures are decided based on trial and error, depending on the one that provides the fastest convergence for the given problem. The long short-term memory back-propagation neural networks were configured for the research after several trials of some architectures. The output was calculated directly from the input through recurrent to feed-forward connections. The Python platform is employed for network training, validating and testing all the trained RNN-LSTM developed in this research. Each set of data of the wind farm model contains eight input parameters and one system output parameter as shown in Table 5.10.

5.4.3. Data Partitioning.

The choice of a partitioning ratio is governed by the ratio that yields the best training and testing results. Following the ratio of 3:1:1, which apart from giving the best results, also provided enough data to validate the results, all the available data sets for the unit were divided into

three- 60 per cent of the data was used for training, 20 per cent for testing and the other 20 per cent for validation.

5.4.4. Training and Testing on LSTM and eLSTM Network.

The Python programming language is used in carrying out all the RNN-LSTM training and validation exercises in this research. The default performance function for feed-forward Back Propagation networks used in the work is the mean square error (MSE) - the average squared error between the network outputs and the target outputs. Table 5.10 shows the architecture of the neural network for the model. The architecture was developed by first using eight neurons in the input layer, thirty neurons in the hidden layer, and one neuron in the output layer. The mean square error during training, validation and testing were $7.35 \times 10^{-4}$, $4.13 \times 10^{-3}$ and $7.90 \times 10^{-4}$ respectively. However, at these points, the coefficients of regression were 0.9351, 0.9031 and 0.9014 respectively. Several other configurations were tried some of which are shown in Table 5.10a. Conversely, a similar model was produced using a 20% dropout method and further 50% as shown in Table 5.10.

The best configuration is represented by the sixth trial in Table 5.10. The R- value between the target (T) and the actual output (A) of the LSTM neural network was 0.996144 at the sixth trial, that is; when trained with eight neurons in the input layer, forty neurons in the hidden layer (twenty neurons apiece in the two hidden layers) and one neuron in the output layer Figure 5.12. The corresponding values during validation and testing with six sets of data were 0.898120 and 0.892096 respectively. The MSE at these points were $6.441 \times 10^{-3}$, $3.528 \times 10^{-3}$and $8.684 \times 10^{-3}$ respectively, for training, validation and testing. This implies excellent generalization of Figure 5.20. The root mean square error between the desired and actual output of the network during validation is $1.876 \times 10^{-8}$ while a zero error goal is as shown in Figure 5.21.

The training algorithm for the Long short-term memory Back-Propagation (LSTM-BP) and the transfer function are namely hyperbolic tangent function (Tansig) for neurons in the input and hidden layers and the rectified linear unit function (RELu) for the output layer leads to quicker convergence during training and validation. Convergence, achieved at the one hundredth epoch. Despite the trial and error procedure used for selecting the training algorithm and the transfer functions. Although as discussed above, [61] had established that the above combination of transfer functions approximates any given function arbitrarily well.

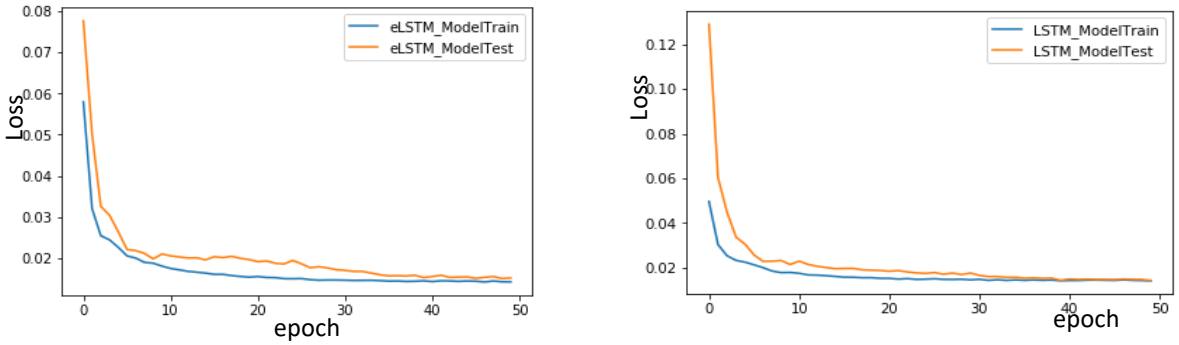



Figure 5. 20**:** eLSTM (50% dropout) and LSTM Model at 18 timestep

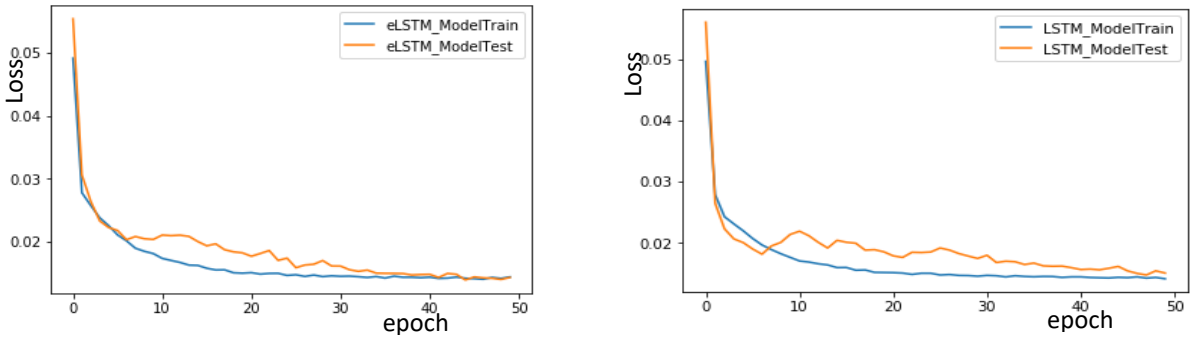



Figure 5. 21**:** eLSTM (20% dropout) and LSTM Model at 12 timestep

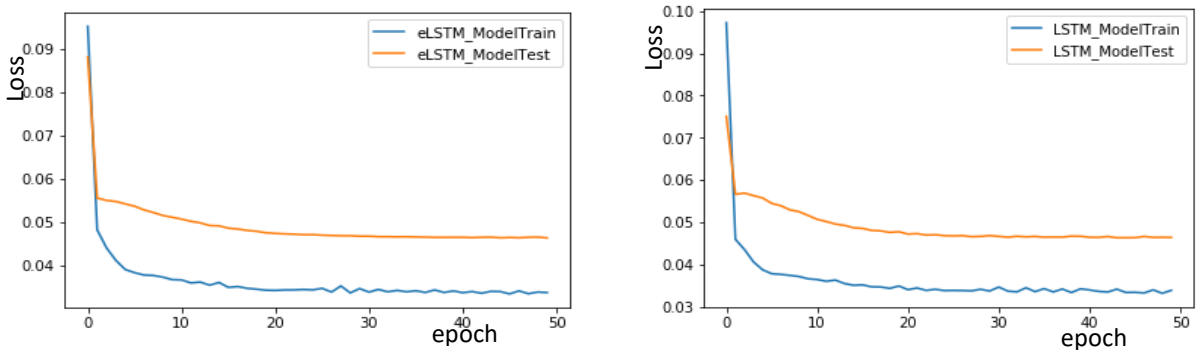



Figure 5. 22**:** eLSTM (20% dropout) and LSTM Model at 6 timesteps.

In this study, a regression plot was created at the end of the network training. This shows the relationship between the outputs of the network and the targets. If the training were perfect, for example, the network output and the target would be equal. However, the deterioration observed in Figure 5.22 in increasing time steps from 6 to 12 steps is associated to model generalisation as the model fails to capture underlying data pattern. This can be associated to model overfitting or data size. This in turn is as discussed in the literature in section 1.1. Overfitting is resolved in Figure 5.20 where eLSTM resolves the over-fitted model by increasing Dropout from 20% to 50%. After training, the neural network is tested on the six sets of validation data (see Appendix (iv and vi)). Data size on the other hand can be increased. In this research, it was not. This is because the model trains better as the time step increases.

Other trials carried-out are by varying the number of neurons in the hidden layers as shown in Table 5.11. The sixth trial gave the best values of MSE and regression as further trials did not improve the results further. Thus, the arrangement in the sixth trial is selected as the best RNN architecture for wind speed prediction, which is; eight neurons in the input layer, thirty neurons in the hidden layer and one neuron in the output layer.

Table 5. 8: eLSTM/LSTM Training and Validation on Different Network Configurations.

| S/N | Architecture | Training | | Validation | | Testing | |
|---|---|---|---|---|---|---|---|
| | | MSE | Regression | MSE | Regression | MSE | Regression |
| 1 | 8 – 10-15 (0.2) – 1 | $8.35612 \times 10^{-4}$ | 0.935187 | $4.13226 \times 10^{-3}$ | 0.903126 | $7.90146 \times 10^{-4}$ | 0.901432 |
| 2 | 8 – 15-8 (0.2) - 1 | $4.30941 \times 10^{-4}$ | 0.942896 | $4.64213 \times 10^{-3}$ | 0.921358 | $6.12147 \times 10^{-4}$ | 0.941082 |
| 2 | 8 – 10-15 (0.2) - 1 | $7.73952 \times 10^{-3}$ | 0.933449 | $3.74323 \times 10^{-4}$ | 0.949325 | $7.24583 \times 10^{-3}$ | 0.941233 |
| 4 | 8 –20-8 - 1 | $3.01395 \times 10^{-4}$ | 0.982135 | $3.47316 \times 10^{-3}$ | 0.934773 | $8.67277 \times 10^{-3}$ | 0.952375 |
| 5 | 8 –10-20 - 1 | $5.74986 \times 10^{-3}$ | 0.91876 | $3.55236 \times 10^{-4}$ | 0.956124 | $3.76315 \times 10^{-3}$ | 0.988641 |
| 6 | 8-15-15 (0.2) -1 | $6.44124 \times 10^{-3}$ | 0.996144 | $3.52837 \times 10^{-3}$ | 0.998120 | $8.68488 \times 10^{-3}$ | 0.992096 |
| 7 | 8 – 20-15 (0.2) -1 | $4.31996 \times 10^{-3}$ | 0.971895 | $3.79497 \times 10^{-3}$ | 0.921242 | $6.87332 \times 10^{-4}$ | 0.950123 |

Table 5. 9**:**RMSE for Different Trials.

| | eLSTM MSE (%) | Dropout MSE (%) | L1L2 MSE (%) | LSTM MSE (%) |
|---|---|---|---|---|
| **Exp. 2** | 73.20 | 72.40 | 68.90 | 71.02 |
| **Exp. 4** | 73.12 | 72.31 | 69.01 | 71.06 |
| **Exp. 6** | 73.08 | 72.24 | 69.00 | 70.12 |
| **Exp. 8** | 73.03 | 72.05 | 70.01 | 70.10 |
| **Exp. 10** | 73.02 | 72.01 | 70.40 | 70.06 |

5.4.5. ARIMA Model Configurations.

In modelling ARIMA for the wind speed prediction and for comparison with eLSTM and LSTM RNN. The following p, d, q parameters equivalent to 0, 1, 1 and 0, 0, 0 respectively are the best grid searches obtained. In the model, $x_t$ is a linear function of the values of $x$ at the previous time steps. However, the concept of psi-weights where the model can be converted to

a finite order of moving averages. The root mean square error estimation of these algorithm are as shown in Table 5.12 below.

Table 5. 10: ARIMA and eLSTM RMSE comparison.

| | **RMSE-ARIMA (%)** | **RMSE-eLSTM (%)** |
|---|---|---|
| **@ 20%** | 73.417 | 78.283 |
| **@30%** | 74.315 | 78.952 |

5.4.6. Overall Model Evaluation.

The research employed the rolling forecast method; meaning, each test dataset will be walked a step at a time after which our model ARIMA/Dropout is used to make a forecast for six time steps.

- After which, the expected value from the test set is made available to the model for the next step forecast.
- We further collect all the test dataset and calculate error scores to check our model skills.
- Because RMSE punishes large errors and results in a score that is the same as the forecast data, we used it to check how our model has performed.

Furthermore, to understand model performance, on the training, test and validation sets, the algorithms are compared in terms of their mean deviations during model evaluations as shown in Figure 5.23. The result show the median range of each model before performance is measured by MSE.

The generated wind speed data on a single turbine and the overall fourteen turbines is used to demonstrate visually what happens during training, testing and validation as shown in Figure 5.24 and 5.25. From the figures, a good model is expected to follow the data pattern during

training and testing such that during prediction, the algorithm would be able to deliver a good judgment, following the data pattern – the concept of a rolling forecast.

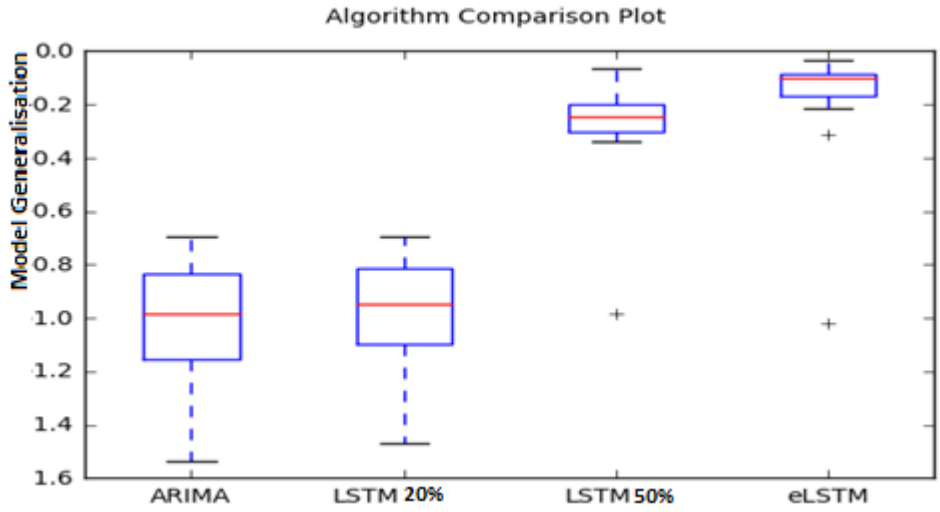



Figure 5. 23**:** Algorithm comparison.

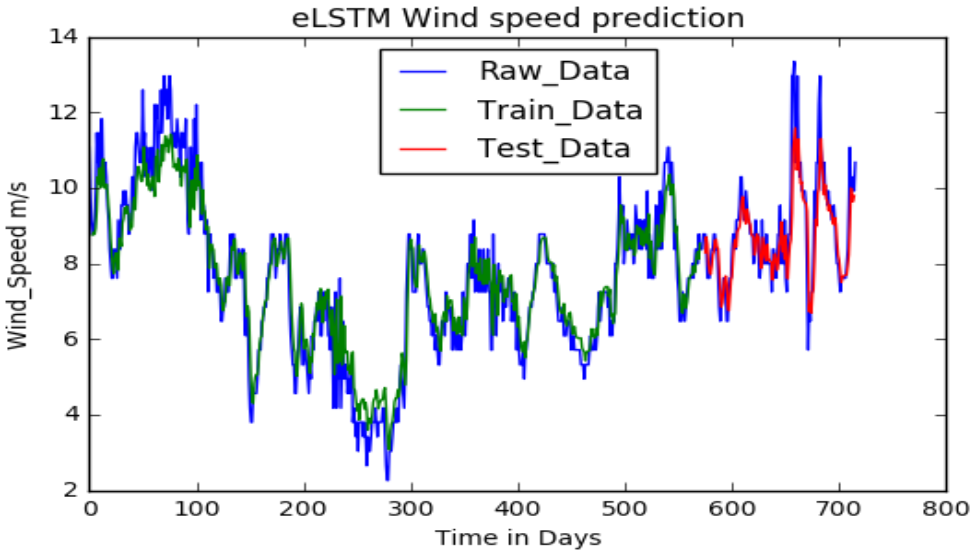



Figure 5. 24**:** Training-test pattern on raw data from a wind turbine.

For further discussions, the codes used in studying Figure 5.24 is explained in Appendix A (x)

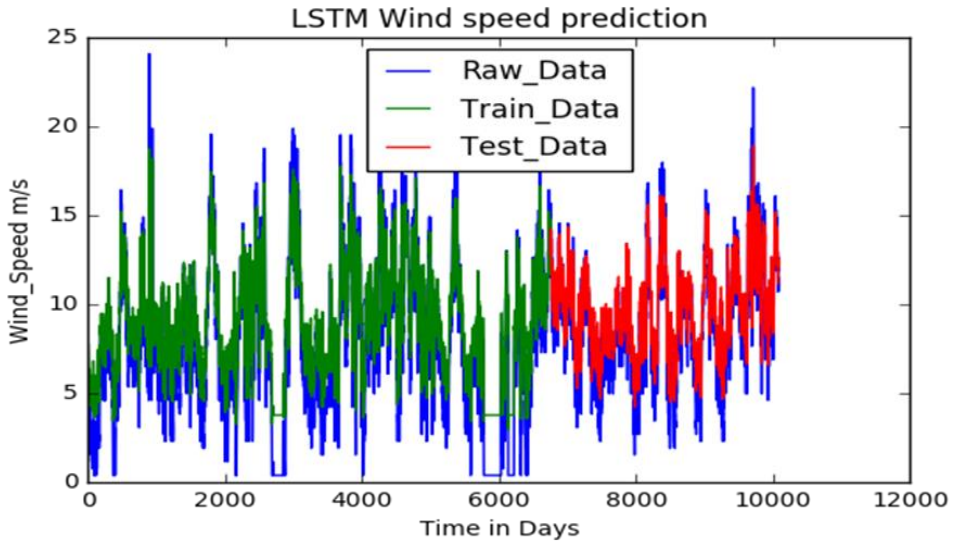



Figure 5. 25**:** Training-test pattern on raw data from a wind farm.

See Appendix A (viii)) for codes used in generating Figure 5.25 and subsequent discussions.

## 5.5. RNN Overfitting Demonstration.

As discussed in section 1.3, overfitting is the major problem with a long short-term memory type of RNN. To demonstrate this setback, a sample of data from the wind farm as modelled with LSTM – Figure 5.25 and further passed through dropout shown in Figure 5.26. The summary statistics are as shown in Table 5.13.

Table 5. 11: Applied Regularisation Methods on the Wind Farm Data

| | No Dropout | Dropout (20%) | Dropout (50%) | L1L2Regularisation |
|---|---|---|---|---|
| count | 5.0000 | 5.0000 | 5.0000 | 5.0000 |
| mean | 1.3648 | 1.3577 | 1.3546 | 1.3443 |
| std | 0.0180 | 0.0228 | 0.0243 | 0.0251 |
| min | 1.3409 | 1.3303 | 1.3210 | 1.3232 |
| 25% | 1.3574 | 1.3375 | 1.3361 | 1.3267 |
| 50% | 1.3605 | 1.3665 | 1.3368 | 1.3355 |
| 75% | 1.3796 | 1.3703 | 1.3695 | 1.3512 |
| max | 1.3858 | 1.3841 | 1.3779 | 1.3849 |

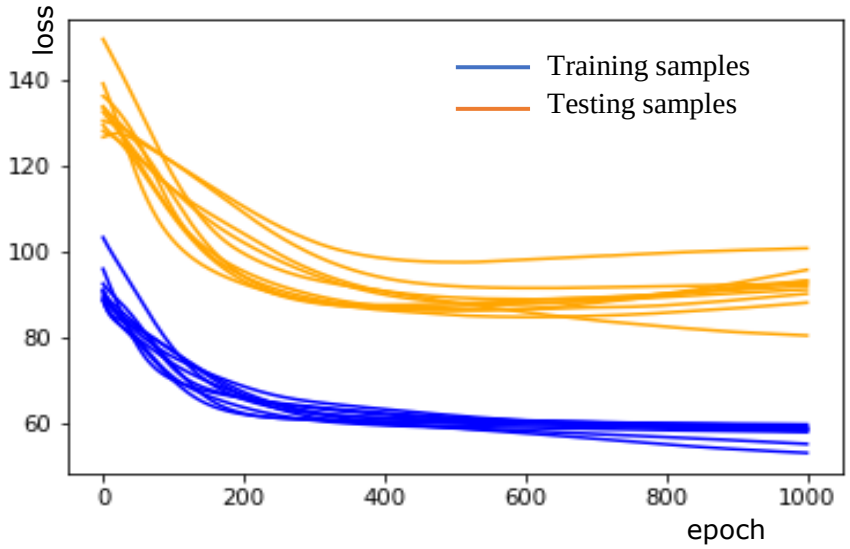



Figure 5. 26: Training sample without overfitting.

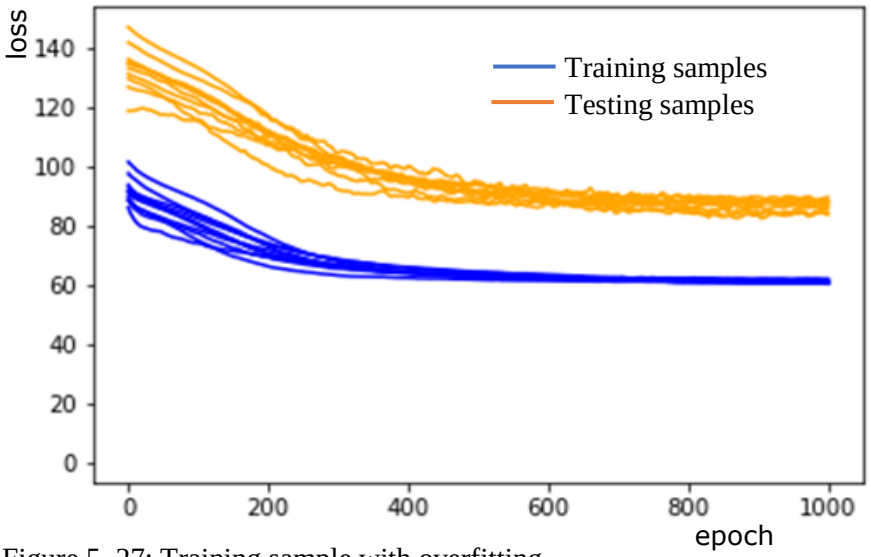



Figure 5. 27: Training sample with overfitting.

In the simulations, the research compared no-dropout (traditional LSTM), dropout at a rate of 20%, dropout at same rate of 50% and L1L2 regularisation of the data on a baseline model of Figure 5.27 and experience some closeness in the training data. This closeness however, infers overfitting of the sample data while training. In addition, overfitting is seen by clear addition of bumps to the train and test RMSE traces – more pronounced on the test RMSE scores of

Figure 5.26. To illustrate the research point, we show a boxplot of Figure 5.27 that compared the distribution of results for each configuration.

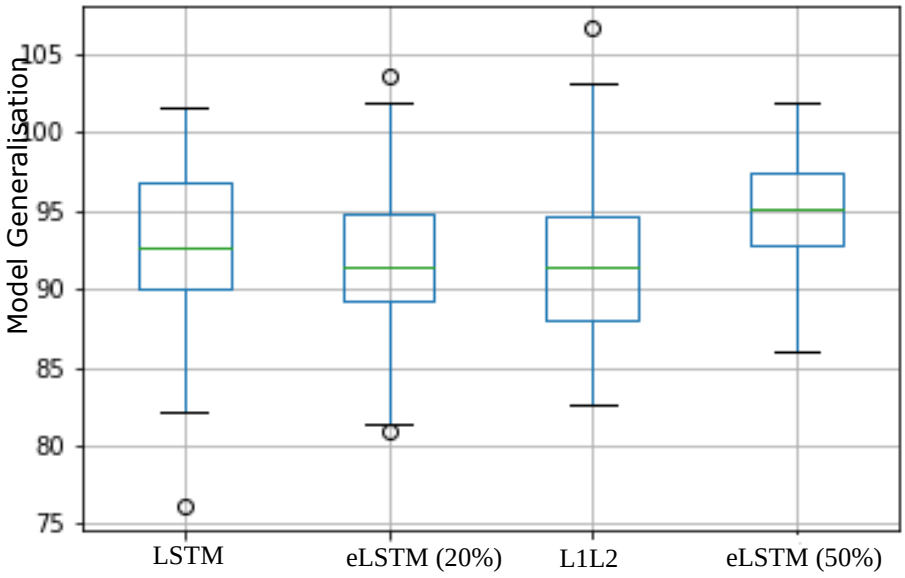



Figure 5. 28: Sample Plot of Over-fitted Data

### 5.5.1 Result.

The data preparation shows how the wind farm data is prepared prior for the machine-learning model fitting. The research tried to study the need for combining regularisation for RNN by training the models, LSTM and LSTM/dropout (eLSTM) at different time steps. It is evident that eLSTM experiences better generalization from Figure 5.22 through 5.20 especially for a long prediction horizon. However, in order to ensure the need for LSTM, overfitting is tested and the sluggish nature of over-fitted data is evident. The ARIMA model comparison shows that the performance of a complicated network that is simplified due to dropout being applied demonstrated that eLSTM is better at predicting wind sequences.

## 5.6. Predicted Results.

After the algorithm configurations, the model fit results are obtained for different ARIMA and eLSTM configurations. In this research, the prediction and confidence intervals on the wind

speed is as shown in Figure 5.28 through 5.31. The eSLTM configuration is at 50% dropout for Figure 5.30 and 20% for Figure 5.31. ARIMA models on the other hand are configured with p, d, q of 0, 1, 1 and 1, 1, 0 respectively as shown in Figure 5.28 and 5.30. the code used in getting these results are found in Appendix B.

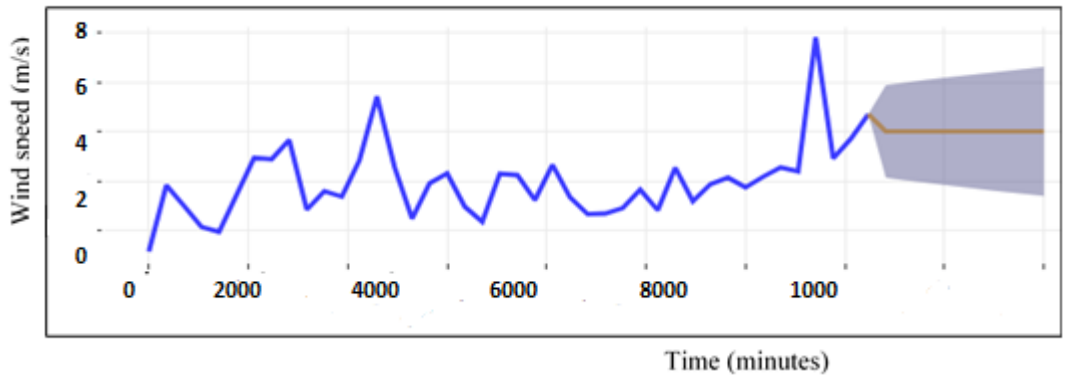



Figure 5. 29: ARIMA (0, 1, 1) Wind Speed Prediction.

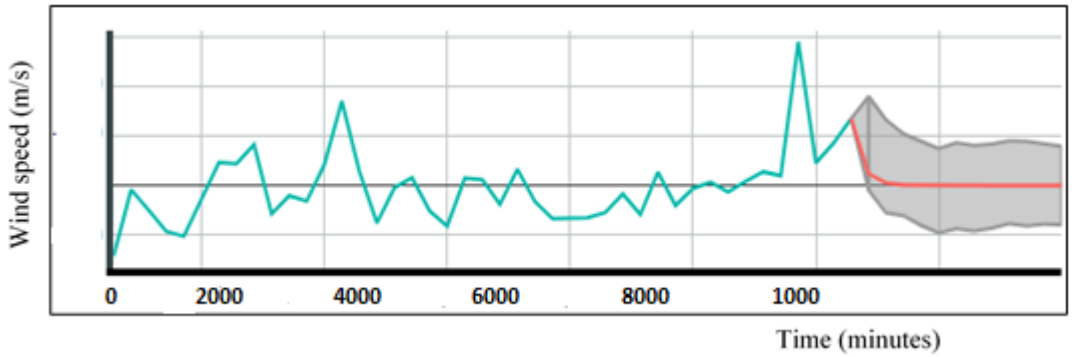



Figure 5. 30: eLSTM at 50% Dropout for Wind Speed Prediction.

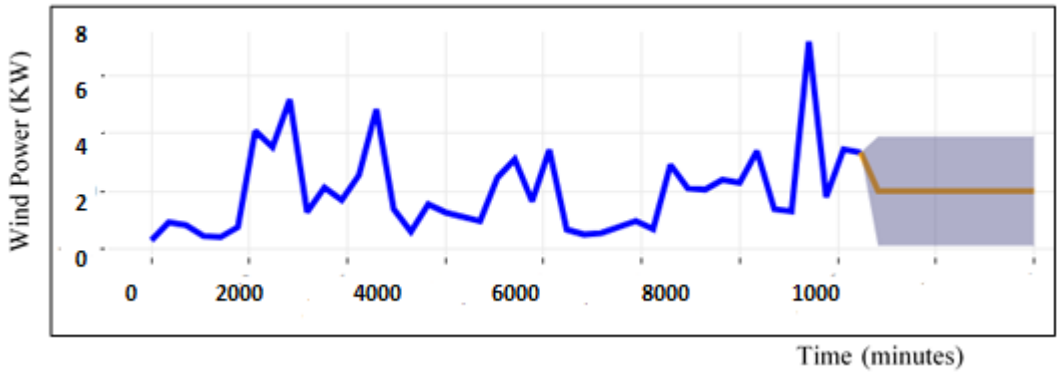



Figure 5. 31: ARIMA (1, 1, 0) Wind Speed Prediction.

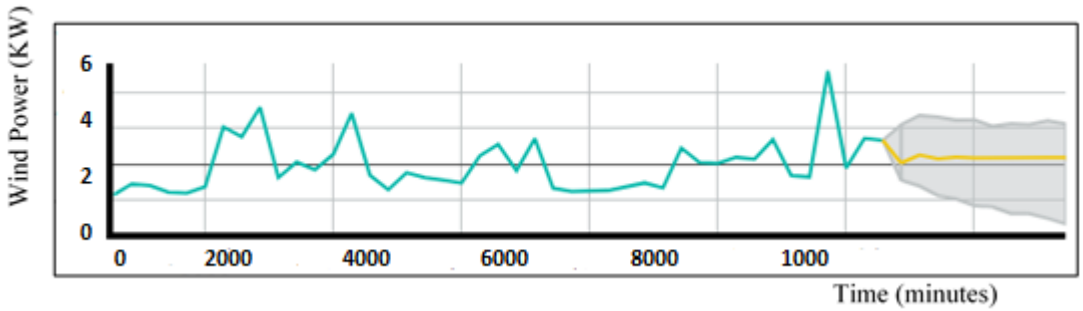



Figure 5. 32**:** eLSTM at 20% Wind Speed Prediction.

Furthermore, in Figure 5.33, the research tends to compare the models – ARIMA, LSTM and eLSTM on typical wind farm data. The results appears blurred.

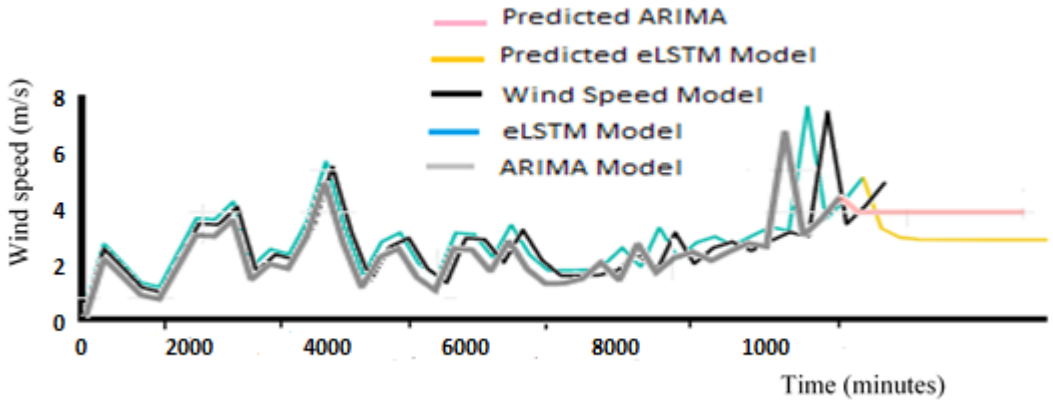



Figure 5. 33: Wind Speed Prediction Plots

The confidence interval observed in Figure 5.29 to 5.32 describes the true mean of the wind speed within the predicted wind speed. It also describes the stability of eLSTM model as shown in yellow line of Figure 5.32.

# Chapter 6

## 6.1. General Discussions and Future Work.

Renewable energy systems such as wind speed predictions are usually very complex in nature. This is due to the uncertainties in the wind. In addition, with the growth of experimental access, the common problem of the nature of time series data appear a concern in terms of near accurate predicted results. The rapid development of novel predictive methods further aggravates this problem. New logical, statistical and experimental methods are therefore required to make sense of the data and by doing so reproduce an improved understanding of wind interactions for predictions.

In a wind farm, wind sensors and recording methods require adequate sequential recording of wind series data from different sources and ranges for less processing time and more reliable output power prediction from the farm. In this context, revealing the most appropriate and reliable mapping method for wind energy is a challenge. Applications such as statistical models, recurrent neural networks (RNN) and the likes have been very useful, successful and are increasingly common. However, as emphasized in the thesis, application of RNN may have resulted in misleading causal network structures due to the influence of exogenous inputs and latent variables. In this research, however, we have confronted the important problem by introducing a new definition of the regularisation method, which constitutes a hybrid long short-term memory (LSTM) and dropout method, which is robust against various wind perturbations, associated with data and common input sizes.

The hybrid regularisation method is inspired by the definition of long short-term memory but the analogy is not exact. LSTM is capable of full reconnection of a previous sequence, in other words, learning long-term dependencies. Due to the nature of time series systems in general and the idea of restructuring for model fitting, LSTM faces the problem that hidden neurons

are not entirely useful and must be estimated in a fashion that requires probability drop-off of certain applied neurons. For this reason, LSTM alone cannot eliminate the influence of exogenous input series and latent variables in all cases. Full generalisation especially on time series is only possible if the hidden neurons have equal influence on the random variables. However, our theoretical analysis and numerical results shows that in a variety of cases, hybrid LSTM or eLSTM outperforms the conventional LSTMs as demonstrated in Figure 5.20 through 5.21. Importantly, this includes cases in which physical methods such as ARIMA in Figure 5.29 and eLSTM of Figure 5.28 has a differing influence on the predicted variables. These findings support the notion that eLSTM is of substantial practical value for attempt to identify a causal network from time series data as demonstrated in Figure 5.32.

In this research, eLSTM is considered mainly in the time domain. Research involving spectral decomposition [50, 51] expanded LSTM to the frequency domain and spectral process. This decomposition leads to a set of causality measures, which are of particular relevance to the spectral characteristic of wind farm models. In the future, the exploration of the spectral version of eLSTM is relevant. Fitting the data with an eLSTM model in the time domain, hence, accessing the statistical relationship between the input (X) and output (Y) variables becomes ideal for time series estimation especially in maintenance of wind farm equipment. Conversely, the advantage of eLSTM over the LSTM is in the application of quantitative measurements.

Finally, another popular approach for regularisation of RNN elements is the use of the Bayesian method. However, a major difference between Bayesian and eLSTM on RNN is the difficulty of the incorporation of backpropagation interactions in the Bayesian approach, which in some cases is a limiting factor as demonstrated in [109].

## 6.2. Future Work.

This research has demonstrated recurrent neural network (RNN) can be regularised using a hybrid approach to improve model performance especially in complex time series type of models due to the stochastic nature of wind. Wind turbine components on the other hand requires reliable estimations for optimum preventive maintenance. Therefore, regularising RNN considering the approaches in this research and other methods such as GRU can be applied to improve wind farm power output for a better return on investment (ROI). Hence, in the future, research will consider the following;

1. In terms of improving recurrent neural network, further research would benefit from leveraging a combination or hybridisation of other regularisation methods such as L1, L1 and L1L2 and further compare results with results obtained from LSTM, dropout for possible improvement in performance measure.
2. In the future, research could leverage eLSTM to estimate the likelihood of bearing failures in the wind turbines to reduce gearbox damage as frequently reported in the wind renewable energy industry.
3. With the adaptive nature of eLSTM in terms of its mapping – explanatory-to-response variable capabilities, winding damages experienced due to wind over-speed, which causes mechanical breakdown in the wind turbine can be controlled with an effective winding speed prediction system.
4. Stress related failures such as Nacelle damages, axial stress and grid failures could be improved by building reliable predictive maintenance systems, considering variants of RNN to control things like yaw motor operation and other management events. This is due to the robust nature of RNN in learning and improving complex patterns.

## 6.2. Conclusions.

In this thesis, we proposed a novel regularisation method, called enhanced long short-term memory (eLSTM) to determine the causal and mapping relationship between reconstructed input wind speeds and the output predicted wind speed for output wind-power prediction from a wind farm.

Looking at the effect of dropout rate as shown in Figure 5.27, which is the probability of dropping a neuron between 0 and 1, it is evident to understand that regularizing recurrent neural networks (RNN) especially with the long short-term memory (LSTM) makes wind series prediction better.

In the machine learning and statistical literature, bias variance remains an issue for predictive error minimization. Models with low bias in parameter estimation has higher variance of the parameter estimate across samples and vice versa. Conversely, models with high bias and low variance pays little attention to training data and causes under-fitting – models are unable to capture underlying data pattern, example Naïve Bayes. In this research, we have demonstrated and corrected overfitting as shown in Figure 5.23 and 5.24 where LSTM is used in modelling wind series shows low bias and high variance and captures noise along with underlying wind data patterns. This is however, corrected by introduction of dropout to LSTM (eLSTM) to maintain a good balance, which is low bias and low variance on the underlying data pattern and hence, improved predictive performance. Another topic of importance in wind power prediction is the issue of removing correlated data as shown in Table 5.7 from the historical wind-farm data model.

Correlation affects model performance, however, it depends on the level of correlated data or the number of variables affected because of issues associated in merging the data from different turbines, in our case. Apart from making the learning algorithm faster by possible reduction of

dimensionality, it decreases harmful bias in the model. It is important to note that not all models are affected by correlation. Models like Naïve Bayes, ARIMA and its variants benefit from positively correlated data while models like Random Forest struggles with correlations, whether positively or negatively. On the other hand, recurrent neural network (RNN) models are affected by correlation since it relies on better dimensional features for model improvement in terms of speed. These issues of correlation goes hand-in-hand with data size and stationarity in feature engineering processes as discussed in chapter 4. Furthermore, another topic of discussion this research has been able to benefit from is the idea behind improved architectural design.

In Figure 5.28 through to Figure 5.32, it is pertinent to observe model improvement based on architectural design by considering best approach as studied from Table 5.10. This study infers the relationship between batch size, weight, optimizer and epoch as seen in the literature and how they are related in the research. Weight, which defines the amount of neuron contribution are changed using an optimizer – in our case, RMSprop. Optimizer on the other hand reduces loss while a change in weight is known as epoch. In Figure 5.23 and 5.24, choosing RMSprop improves model performance by over 90% in terms of learning the stochastic wind data. Hence, the knowledge gained from these figures led to model building of Figure 5.29 and 5.31 respectively. In terms of loss reduction, Figure 5.20 through 5.22 revealed that increase in predicted horizon increases loss. This observation in turn relates significantly to the nature of batch size applied in model configuration, 12 in our case. Hence, overfitting is addressed by imposing dropout on LSTM.

Using the results obtained from the probability density function estimation, which is directly associated to the Weibull parameter estimation, prediction is affirmed by statistical means and hence, eLSTM is capable of eliminating the influence of uncorrelated data and under-fitting in LSTM and therefore ensures prediction of wind speed for wind-farm power output within a

wind farm. This demonstrated that eLSTM is more robust than LSTM and ARIMA especially on predictions involving a long sequence of predicted horizons – multiple steps ahead such as 6-hours, which is equivalent to 72-time-steps ahead. In addition, eLSTM is found to be more robust in deriving causal relationships underlying a complex stochastic system compared to the traditional LSTM. In chapter 4, the performance of eLSTM is tested using RMSE on wind speed data. In the future research, we plan to apply the method on an operational wind farm to predict likely failure and improvements in rotor dynamics of a wind turbine.

# Reference


[1] A. Iessa, N. I. A. Wahab, N. Mariun, and H. Hizam, "Method of estimating the maximum penetration level of wind power using transient frequency deviation index based on COI frequency," in *2016 IEEE International Conference on Power and Energy (PECon)*, 2016, pp. 274-279.

[2] CNBC, "China and US lead way with wind power installations, says global energy report.," 05 May, 2017 13 Feb 2017

[3] C. Liang, P. Wang, X. Han, W. Qin, Y. Jia, and T. Yuan, "Battery Energy Storage Selection Based on a Novel Intermittent Wind Speed Model for Improving Power System Dynamic Reliability," *IEEE Transactions on Smart Grid*, 2017.

[4] Z. Guo, W. Zhao, H. Lu, and J. Wang, *Multi-step forecasting for wind speed using a modified EMD-based artificial neural network model*. 2012, pp. 241–249.

[5] Z. Guo, W. Zhao, H. Lu, and J. Wang, "Multi-step forecasting for wind speed using a modified EMD-based artificial neural network model," *Renewable Energy*, vol. 37, no. 1, pp. 241-249, 2012/01/01/ 2012.

[6] S. Balluff, J. Bendfeld, and S. Krauter, "Short term wind and energy prediction for offshore wind farms using neural networks," in *2015 International Conference on Renewable Energy Research and Applications (ICRERA)*, 2015, pp. 379-382.

[7] T. Wardah, A. A. Kamil, A. B. S. Hamid, and W. W. I. Maisarah, "Statistical verification of numerical weather prediction models for quantitative precipitation forecast," in *2011 IEEE Colloquium on Humanities, Science and Engineering*, 2011, pp. 88-92.

[8] N. Chen, Z. Qian, I. T. Nabney, and X. Meng, "Wind Power Forecasts Using Gaussian Processes and Numerical Weather Prediction," *IEEE Transactions on Power Systems*, vol. 29, no. 2, pp. 656-665, 2014.

[9] M. Yesilbudak, S. Sagiroglu, and I. Colak, "A new approach to very short term wind speed prediction using k-nearest neighbor classification," *Energy Conversion and Management*, vol. 69, pp. 77-86, 2013/05/01/ 2013.

[10] H. Liu, H.-q. Tian, and Y.-f. Li, "Comparison of two new ARIMA-ANN and ARIMA-Kalman hybrid methods for wind speed prediction," *Applied Energy*, vol. 98, pp. 415-424, 2012/10/01/ 2012.

[11] S. Baran, "Probabilistic wind speed forecasting using Bayesian model averaging with truncated normal components," *Computational Statistics & Data Analysis*, vol. 75, pp. 227-238, 2014/07/01/ 2014.

[12] D. Lee and R. Baldick, "Short-Term Wind Power Ensemble Prediction Based on Gaussian Processes and Neural Networks," *IEEE Trans. Smart Grid*, vol. 5, no. 1, pp. 501-510, 2014.

[13] H.-z. Li, S. Guo, C.-j. Li, and J.-q. Sun, "A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm," *Knowledge-Based Systems*, vol. 37, pp. 378-387, 2013/01/01/ 2013.

[14] W. Zhang, J. Wang, J. Wang, Z. Zhao, and M. Tian, "Short-term wind speed forecasting based on a hybrid model," *Applied Soft Computing*, vol. 13, no. 7, pp. 3225-3233, 2013/07/01/ 2013.

[15] J. Shi, Z. Ding, W.-J. Lee, Y. Yang, Y. Liu, and M. Zhang, "Hybrid forecasting model for very-short term wind power forecasting based on grey relational analysis and wind speed distribution features," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 521-526, 2014.

[16] S. Haykin, *Kalman filtering and neural networks*. John Wiley & Sons, 2004.

[17] S. Salcedo-Sanz, E. G. Ortiz-Garcı´a, Á. M. Pérez-Bellido, A. Portilla-Figueras, and L. Prieto, "Short term wind speed prediction based on evolutionary support vector regression algorithms," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4052-4057, 2011/04/01/ 2011.

[18] J. Liu and E. Zio, "SVM hyperparameters tuning for recursive multi-step-ahead prediction," (in English), *Neural Computing & Applications*, Article vol. 28, no. 12, pp. 3749-3763, Dec 2017.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[20] O. Wallscheid, W. Kirchgässner, and J. Böcker, "Investigation of long short-term memory networks to temperature prediction for permanent magnet synchronous motors," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1940-1947.

[21] W. Yao, P. Huang, and Z. Jia, "Multidimensional LSTM Networks to Predict Wind Speed," in *2018 37th Chinese Control Conference (CCC)*, 2018, pp. 7493-7497.

[22] B. Kanna and S. N. Singh, "Long term wind power forecast using adaptive wavelet neural network," in *2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)*, 2016, pp. 671-676.

[23] D. T. Mirikitani and N. Nikolaev, "Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling," *IEEE Transactions on Neural Networks*, vol. 21, no. 2, pp. 262-274, 2010.

[24] J. Brownlee, "How to Use Weight Regularisation with LSTM Networks for Time Series Forecasting," p. 1, May 5, 2017 2017.
[25] M. Coto-Jimenez and J. Goddard-Close, "LSTM Deep Neural Networks Postfiltering for Enhancing Synthetic Voices," (in English), *International Journal of Pattern Recognition and Artificial Intelligence*, Article; Proceedings Paper vol. 32, no. 1, p. 24, Jan 2018, Art. no. 1860008.
[26] D. Q. Wang, Y. L. Gao, J. X. Liu, C. H. Zheng, and X. Z. Kong, "Identifying drug-pathway association pairs based on L1L2,1-integrative penalized matrix decomposition," (in English), *Oncotarget*, Article vol. 8, no. 29, pp. 48075-48085, Jul 2017.
[27] V. Pham, C. Kermorvant, and J. Louradour, *Dropout Improves Recurrent Neural Networks for Handwriting Recognition*. 2013.
[28] T. Bluche, C. Kermorvant, and J. Louradour, "Where to apply dropout in recurrent neural networks for handwriting recognition?," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, 2015, pp. 681-685.
[29] M. Fei and D. Y. Yeung, "Temporal Models for Predicting Student Dropout in Massive Open Online Courses," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, pp. 256-263.
[30] S. Hochreiter, M. Heusel, and K. Obermayer, "Fast model-based protein homology detection without alignment," *Bioinformatics*, vol. 23, no. 14, pp. 1728-1736, 2007.
[31] R. Maalej and M. Kherallah, "Improving MDLSTM for Offline Arabic Handwriting Recognition Using Dropout at Different Positions," in *Artificial Neural Networks and Machine Learning – ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II*, A. E. P. Villa, P. Masulli, and A. J. Pons Rivero, Eds. Cham: Springer International Publishing, 2016, pp. 431-438.
[32] R. Maalej, N. Tagougui, and M. Kherallah, "Recognition of Handwritten Arabic Words with Dropout Applied in MDLSTM," in *Image Analysis and Recognition: 13th International Conference, ICIAR 2016, in Memory of Mohamed Kamel, Póvoa de Varzim, Portugal, July 13-15, 2016, Proceedings*, A. Campilho and F. Karray, Eds. Cham: Springer International Publishing, 2016, pp. 746-752.
[33] T. Moon, H. Choi, H. Lee, and I. Song, *RnnDrop: a novel dropout for RNNs in ASR*. 2015, pp. 65-70.
[34] Y. Tao, H. Chen, and C. Qiu, "Wind power prediction and pattern feature based on deep learning method," in *2014 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2014, pp. 1-4.
[35] Q. Hu, R. Zhang, and Y. Zhou, "Transfer learning for short-term wind speed prediction with deep neural networks," *Renewable Energy*, vol. 85, pp. 83-95, 2016.
[36] C. Y. Quek and T. Mitchell, "Classification of world wide web documents," *Master's thesis, School of Computer Science Carnegie Mellon University*, 1997.
[37] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31-44, 1996.
[38] G. Sinden, "Characteristics of the UK wind resource: Long-term patterns and relationship to electricity demand," *Energy Policy*, vol. 35, no. 1, pp. 112-127, 2007/01/01/ 2007.
[39] A. Y. Al-Hassan and D. R. Hill, "Islamic technology; an illustrated history," 1986.
[40] S. R. Waichler and M. S. Wigmosta, "Development of Hourly Meteorological Values From Daily Data and Significance to Hydrological Modeling at H. J. Andrews Experimental Forest," *Journal of Hydrometeorology*, vol. 4, no. 2, pp. 251-263, 2003.
[41] M. Adaramola and O. M. Oyewola, *Wind speed distribution and characteristics in Nigeria*. 2011, pp. 82-86.
[42] S. Mukhopadhyay and P. K. Panigrahi, "Wind speed data analysis for various seasons during a decade by wavelet and S transform," *arXiv preprint arXiv:1308.2773*, 2013.
[43] S. Mukhopadhyay, N. K. Das, R. Kumar, D. Dash, A. Mitra, and P. K. Panigrahi, "Study of the dynamics of wind data fluctuations: a wavelet and MFDFA based novel method," *Elsevier Procedia Technology*, 2014.
[44] S. Mukhopadhyay, S. Mandal, P. Panigrahi, and A. Mitra, "Heated wind particle's behavioral study by the continuous wavelet transform as well as the fractal analysis," *Computer Science & Information Technology*, pp. 169-174, 2013.
[45] E. W. E. Association, *The economics of wind energy*. EWEA, 2009.
[46] K. Ko, K. Kim, and J. Huh, *Characteristics of wind variations on Jeju Island, Korea*. 2010, pp. 36-45.
[47] W. Weibull, "A statistical distribution function of wide applicability," *Journal of applied mechanics*, vol. 18, no. 3, pp. 293-297, 1951.
[48] M. Davidian, *Nonlinear models for repeated measurement data*. Routledge, 2017.

[49] O. Karakuş, E. E. Kuruoğlu, and M. A. Altınkaya, "One-day ahead wind speed/power prediction based on polynomial autoregressive model," *IET Renewable Power Generation,* vol. 11, no. 11, pp. 1430-1439, 2017.
[50] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.
[51] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.
[52] K. Khan and A. Sahai, "A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context," *International Journal of Intelligent Systems and Applications,* vol. 4, no. 7, p. 23, 2012.
[53] D. Davidian, "Feed-forward neural network," ed: Google Patents, 1995.
[54] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems,* vol. 39, no. 1, pp. 43-62, 1997.
[55] S. Singhal and L. Wu, "Training feed-forward networks with the extended Kalman algorithm," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, 1989, pp. 1187-1190: IEEE.
[56] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks,* vol. 5, no. 2, pp. 157-166, 1994.
[57] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026,* 2013.
[58] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104-3112.
[59] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1033-1040: Citeseer.
[60] Y.-w. Jun *et al.*, "Nanoscale size effect of magnetic nanocrystals and their utilization for cancer diagnosis via magnetic resonance imaging," *Journal of the American Chemical Society,* vol. 127, no. 16, pp. 5732-5733, 2005.
[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.
[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," (in eng), *Neural Comput,* vol. 9, no. 8, pp. 1735-80, Nov 15 1997.
[63] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259,* 2014.
[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research,* vol. 15, no. 1, pp. 1929-1958, 2014.
[65] G. Vaca-Castano, S. Das, J. P. Sousa, N. D. Lobo, and M. Shah, "Improved scene identification and object detection on egocentric vision of daily activities," *Computer Vision and Image Understanding,* vol. 156, pp. 92-103, 3// 2017.
[66] S. V. Ravuri and A. Stolcke, "Recurrent neural network and LSTM models for lexical utterance classification," in *INTERSPEECH*, 2015, pp. 135-139.
[67] T. N. Sainath *et al.*, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks,* vol. 64, pp. 39-48, 2015.
[68] E. Phaisangittisagul, "An Analysis of the Regularisation between L2 and Dropout in Single Hidden Layer Neural Network," *IEEE Computer society,* Conference Paper vol. DOI 10.1109/ISMS.2016.14, no. 2166-0670/16, p. 6, 2016 2016.
[69] J. M. Hu, J. Z. Wang, and L. Q. Xiao, "A hybrid approach based on the Gaussian process with t-observation model for short-term wind speed forecasts," *Renewable Energy,* vol. 114, pp. 670-685, Dec 2017.
[70] I. Tanaka and H. Ohmori, "Method Evaluation for Short-Term Wind Speed Prediction Considering Multi Regions in Japan," (in English), *Journal of Robotics and Mechatronics,* Article vol. 28, no. 5, pp. 681-686, Oct 2016.
[71] A. Shamshad, M. Bawadi, W. W. Hussin, T. Majid, and S. Sanusi, "First and second order Markov chain models for synthetic generation of wind speed time series," *Energy,* vol. 30, no. 5, pp. 693-708, 2005.
[72] P. Pinson and H. Madsen, "Adaptive modelling and forecasting of offshore wind power fluctuations with Markov-switching autoregressive models," *Journal of forecasting,* vol. 31, no. 4, pp. 281-313, 2012.
[73] J. Garcia-Gonzalez, R. M. R. de la Muela, L. M. Santos, and A. M. Gonzalez, "Stochastic joint optimization of wind generation and pumped-storage units in an electricity market," *IEEE Transactions on Power Systems,* vol. 23, no. 2, pp. 460-468, 2008.

[74] P. Society, "phm11 data challenge - condition monitoring of anemometers," *https://www.phmsociety.org/competition/pmh/11/problems*. , Case study 10/11/2014 2011.
[75] V. Ranganayaki and S. Deepa, "An Intelligent Ensemble Neural Network Model for Wind Speed Prediction in Renewable Energy Systems," *The Scientific World Journal*, vol. 2016, pp. 9293529-9293529, 2016.
[76] S. W. Vera Bulaevskaya, Andy Clifton, Wayne Miller, "statistical analysis and modeling for wind power forecasting," *Article*, Conference Preceedings 10/06/2014 10/06/2014.
[77] S. H. Pishgar-Komleh, A. Keyhani, and P. Sefeedpari, "Wind speed and power density analysis based on Weibull and Rayleigh distributions (a case study: Firouzkooh county of Iran)," *Renewable and Sustainable Energy Reviews*, vol. 42, pp. 313-322, 2015/02/01/ 2015.
[78] J. Jeon and J. W. Taylor, "Using conditional kernel density estimation for wind power density forecasting," *Journal of the American Statistical Association*, vol. 107, no. 497, pp. 66-79, 2012.
[79] A. Wędzik, "The Optimization of Cable Layout Design in Wind Farm Internal Networks," *Acta Energetica*, 2014.
[80] K. Attias and S. P. Ladany, "Optimal Econoimc Layout of Turbines on Windfarms," *Wind Engineering*, vol. 30, no. 2, pp. 141-151, 2006.
[81] K. Attias and S. P. Ladany, "Optimal layout for wind turbine farms," in *World Renewable Energy Congress-Sweden; 8-13 May; 2011; Linköping; Sweden*, 2011, no. 57, pp. 4153-4160: Linköping University Electronic Press.
[82] X. Gao, H. Yang, and L. Lu, "Investigation into the optimal wind turbine layout patterns for a Hong Kong offshore wind farm," *Energy*, vol. 73, pp. 430-442, 2014/08/14/ 2014.
[83] H. Liangyou, J. Dongxiang, H. Qian, and D. Yongshan, "Wind speed forecasting using fully recurrent neural network in wind power plants."
[84] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural network design*. Pws Pub. Boston, 1996.
[85] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994, p. 768.
[86] A. H. Kramer and A. Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks," in *Advances in neural information processing systems 1*, S. T. David, Ed.: Morgan Kaufmann Publishers Inc., 1989, pp. 40-48.
[87] D. Schneidman-Duhovny *et al.*, "A method for integrative structure determination of protein-protein complexes," *Bioinformatics*, vol. 28, no. 24, pp. 3282-3289, 2012.
[88] C. Farrington, N. J. Andrews, A. Beale, and M. Catchpole, "A statistical algorithm for the early detection of outbreaks of infectious disease," *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, pp. 547-563, 1996.
[89] S. Teki, M. Grube, S. Kumar, and T. D. Griffiths, "Distinct neural substrates of duration-based and beat-based auditory timing," *Journal of Neuroscience*, vol. 31, no. 10, pp. 3805-3812, 2011.
[90] M. J. Nasr, A. H. Khalaj, and S. Mozaffari, "Modeling of heat transfer enhancement by wire coil inserts using artificial neural network analysis," *Applied Thermal Engineering*, vol. 30, no. 2-3, pp. 143-151, 2010.
[91] P. Chatziagorakis *et al.*, "Enhancement of hybrid renewable energy systems control with neural networks applied to weather forecasting: the case of Olvio," (in English), *Neural Computing & Applications*, Article; Proceedings Paper vol. 27, no. 5, pp. 1093-1118, Jul 2016.
[92] S. Ryu, S. Kim, J. Choi, H. Yu, and G. G. Lee, "Neural sentence embedding using only in-domain sentences for out-of-domain sentence detection in dialog systems," *Pattern Recognition Letters*, vol. 88, pp. 26-32, 2017.
[93] U. Güçlü and M. A. J. van Gerven, "Modeling the Dynamics of Human Brain Activity with Recurrent Neural Networks," (in English), *Frontiers in Computational Neuroscience*, Original Research vol. 11, no. 7, 2017-February-09 2017.
[94] G. Box, "Box and Jenkins: Time Series Analysis, Forecasting and Control," in *A Very British Affair: Six Britons and the Development of Time Series Analysis During the 20th Century*London: Palgrave Macmillan UK, 2013, pp. 161-215.
[95] L. Hui, T. Hong-qi, and L. Yan-fei, "Comparison of two new ARIMA-ANN and ARIMA-Kalman hybrid methods for wind speed prediction.," *Applied Energy*, vol. 98, pp. 415-424, 2012.
[96] R. G. Kavasseri and K. Seetharaman, "Day-ahead wind speed forecasting using f-ARIMA models," *Renewable Energy*, vol. 34, no. 5, pp. 1388-1393, 5/5/2009 2009.
[97] X. Jiang and H. Adeli, "Wavelet packet-autocorrelation function method for traffic flow pattern analysis," *Computer-Aided Civil and Infrastructure Engineering*, vol. 19, no. 5, pp. 324-337, 2004.
[98] C. M. Mastrangelo and D. R. Forrest, "Multivariate autocorrelated processes: Data and shift generation," *Journal of Quality Technology*, vol. 34, no. 2, pp. 216-220, 2002.

[99] F. X. Diebold and R. S. Mariano, "Comparing predictive accuracy," *Journal of Business & economic statistics*, vol. 20, no. 1, pp. 134-144, 2002.
[100] M. Applications, "How-to-model-partial-autocorrelation-function-pacf," Blog (online publication) 22/08/2018 2018.
[101] E. Lutins, "Grid Searching in Machine Learning: Quick Explanation and Python Implementation," (in English), 5/9/2017 2017.
[102] M. Office, "Weather and Climate Measurements " website 19/09/2018 2018.
[103] Y.-R. Chen, "Characterization of Cup Anemometer Dynamics and Calculation of the Acoustic Noise Produced by a NREL Phase VI Wind Turbine Blade," Case Western Reserve University, 2016.
[104] G. Fortin, J. Perron, and A. Ilinca, "Behaviour and modeling of cup anemometers under Icing conditions," 2005.
[105] E. Terciyanli *et al.*, "The architecture of a large-scale wind power monitoring and forecast system," in *4th International Conference on Power Engineering, Energy and Electrical Drives*, 2013, pp. 1162-1167.
[106] U. D. o. Energy, "How a Wind Turbine Works," (in English), Blog 2014.
[107] I. Anaconda, "Anaconda Installation on Linux.," (in English.), p. web page, 2016.
[108] F. a. V. Pedregosa, G. and Gramfort, A. and Michel, V., B. a. G. and Thirion, O. and Blondel, M. and Prettenhofer, P., R. a. D. and Weiss, V. and Vanderplas, J. and Passos, A. and, and D. a. B. Cournapeau, M. and Perrot, M. and Duchesnay, E., "Scikit-learn: Machine Learning in {P}ython}," (in English), *Journal of Machine Learning Research*, vol. 12, pp. 2825--2830, 2011.
[109] L. Fahrmeir, T. Kneib, and S. Konrath, "Bayesian regularisation in structured additive regression: a unifying perspective on shrinkage, smoothing and predictor selection," *Statistics and Computing*, vol. 20, no. 2, pp. 203-219, 2010.
[110] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994, p. 768.
[111] A. H. Kramer and A. Sangiovanni-Vincentelli, "Efficient parallel learning algorithms for neural networks," in *Advances in neural information processing systems 1*, S. T. David, Ed.: Morgan Kaufmann Publishers Inc., 1989, pp. 40-48.
[112] S. Teki, M. Grube, S. Kumar, and T. D. Griffiths, "Distinct neural substrates of duration-based and beat-based auditory timing," *Journal of Neuroscience*, vol. 31, no. 10, pp. 3805-3812, 2011.
[113] D. Kececioglu, *Reliability engineering handbook*. DEStech Publications, Inc, 2002.

# Appendix A

### (i) Autocorreltion Code

```
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
from pandas.tools.plotting import autocorrelation_plot
#Autocorrelation of the wind series
series = read_csv('wind_data.csv', usecols=[2], engine='python', skipfooter=3)
series = series.values
autocorrelation_plot(series)
pyplot.show()
```

### (ii) Wind Speed ARIMA Code

```
from pandas import read_csv
from pandas import datetime
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot
series = read_csv('wind_data.csv', usecols=[2], engine='python', skipfooter=3)
series = series.values
# fit model
model = ARIMA(series, order=(1,1,2))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())
```

**(iii) ARIMA Model Results**

```
==============================================================================
Dep. Variable:                     y   No. Observations:                 1437
Model:                 ARMA(6, 0)   Log Likelihood               -1785.164
Method:                   css-mle   S.D. of innovations              0.837
Date:                Sun, 20 May 2018   AIC                           3586.329
Time:                        22:09:21   BIC                           3628.491
Sample:                             0   HQIC                          3602.070
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.0417      0.941      5.359      0.000       3.198       6.886
ar.L1.y        0.7313      0.026     27.844      0.000       0.680       0.783
ar.L2.y        0.0860      0.033      2.637      0.008       0.022       0.150
ar.L3.y        0.0695      0.033      2.127      0.034       0.005       0.133
ar.L4.y       -0.0129      0.033     -0.396      0.692      -0.077       0.051
ar.L5.y        0.0142      0.033      0.436      0.663      -0.050       0.078
ar.L6.y        0.0897      0.026      3.411      0.001       0.038       0.141
                                    Roots
==============================================================================
                  Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1            1.0132           -0.0000j            1.0132           -0.0000
AR.2            1.0120           -1.1623j            1.5411           -0.1360
AR.3            1.0120           +1.1623j            1.5411            0.1360
AR.4           -0.7109           -1.4519j            1.6166           -0.3225
AR.5           -0.7109           +1.4519j            1.6166            0.3225
AR.6           -1.7736           -0.0000j            1.7736           -0.5000
------------------------------------------------------------------------------
```

**(iv) eLSTM Model Comparison Code of Figure 5.20, 5.21 and 5.22**

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, Dropout
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg

# load dataset
dataset = read_csv('wind_data.csv', header=0, index_col=0)
values = dataset.values
# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
print(reframed.head())

# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
```

```
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='rmsprop')
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='eLSTM_ModelTrain')
pyplot.plot(history.history['val_loss'], label='eLSTM_ModelTest')
pyplot.legend()
pyplot.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

**(v) Result Generated from (iv)**

| | var1(t-1) | var2(t-1) | var3(t-1) | var4(t-1) | var5(t-1) | var6(t-1) | \ |
|---|---|---|---|---|---|---|---|
| 1 | 0.13 | 0.35 | 0.25 | 0.53 | 0.67 | 2.29e-03 | |
| 2 | 0.15 | 0.37 | 0.25 | 0.53 | 0.67 | 3.81e-03 | |
| 3 | 0.16 | 0.43 | 0.23 | 0.55 | 0.67 | 5.33e-03 | |
| 4 | 0.18 | 0.49 | 0.23 | 0.56 | 0.67 | 8.39e-03 | |
| 5 | 0.14 | 0.49 | 0.23 | 0.56 | 0.67 | 9.91e-03 | |

```
   var7(t-1)  var8(t-1)  var1(t)
1       0.00        0.0     0.15
2       0.00        0.0     0.16
3       0.00        0.0     0.18
4       0.04        0.0     0.14
5       0.07        0.0     0.11
(8760, 1, 8) (8760,) (35039, 1, 8) (35039,)
Train on 8760 samples, validate on 35039 samples
Epoch 1/50
 - 3s - loss: 0.0567 - val_loss: 0.0526
Epoch 2/50
 - 2s - loss: 0.0412 - val_loss: 0.0528
Epoch 3/50
 - 2s - loss: 0.0254 - val_loss: 0.0434
Epoch 4/50
 - 1s - loss: 0.0172 - val_loss: 0.0365
Epoch 5/50
 - 2s - loss: 0.0156 - val_loss: 0.0236
Epoch 6/50
 - 2s - loss: 0.0149 - val_loss: 0.0180
Epoch 7/50
 - 2s - loss: 0.0147 - val_loss: 0.0166
Epoch 8/50
 - 2s - loss: 0.0146 - val_loss: 0.0158
Epoch 9/50
 - 2s - loss: 0.0146 - val_loss: 0.0150
Epoch 10/50
 - 2s - loss: 0.0146 - val_loss: 0.0144
Epoch 11/50
 - 2s - loss: 0.0145 - val_loss: 0.0142
Epoch 12/50
 - 2s - loss: 0.0146 - val_loss: 0.0142
Epoch 13/50
 - 1s - loss: 0.0145 - val_loss: 0.0140
Epoch 14/50
 - 2s - loss: 0.0145 - val_loss: 0.0139
Epoch 15/50
 - 2s - loss: 0.0145 - val_loss: 0.0139
Epoch 16/50
 - 2s - loss: 0.0145 - val_loss: 0.0138
Epoch 17/50
 - 2s - loss: 0.0145 - val_loss: 0.0138
Epoch 18/50
 - 2s - loss: 0.0145 - val_loss: 0.0139
Epoch 19/50
 - 2s - loss: 0.0145 - val_loss: 0.0140
Epoch 20/50
 - 2s - loss: 0.0144 - val_loss: 0.0139
Epoch 21/50
 - 2s - loss: 0.0145 - val_loss: 0.0139
```

```
Epoch 22/50
 - 2s - loss: 0.0144 - val_loss: 0.0137
Epoch 23/50
 - 2s - loss: 0.0144 - val_loss: 0.0138
Epoch 24/50
 - 2s - loss: 0.0145 - val_loss: 0.0139
Epoch 25/50
 - 2s - loss: 0.0145 - val_loss: 0.0138
Epoch 26/50
 - 2s - loss: 0.0144 - val_loss: 0.0139
Epoch 27/50
 - 1s - loss: 0.0144 - val_loss: 0.0140
Epoch 28/50
 - 2s - loss: 0.0144 - val_loss: 0.0139
Epoch 29/50
 - 2s - loss: 0.0144 - val_loss: 0.0138
Epoch 30/50
 - 2s - loss: 0.0144 - val_loss: 0.0136
Epoch 31/50
 - 2s - loss: 0.0145 - val_loss: 0.0142
Epoch 32/50
 - 2s - loss: 0.0145 - val_loss: 0.0138
Epoch 33/50
 - 2s - loss: 0.0144 - val_loss: 0.0136
Epoch 34/50
 - 2s - loss: 0.0144 - val_loss: 0.0139
Epoch 35/50
 - 1s - loss: 0.0145 - val_loss: 0.0137
Epoch 36/50
 - 2s - loss: 0.0144 - val_loss: 0.0136
Epoch 37/50
 - 2s - loss: 0.0144 - val_loss: 0.0135
Epoch 38/50
 - 2s - loss: 0.0144 - val_loss: 0.0135
Epoch 39/50
 - 2s - loss: 0.0144 - val_loss: 0.0134
Epoch 40/50
 - 1s - loss: 0.0143 - val_loss: 0.0138
Epoch 41/50
 - 2s - loss: 0.0145 - val_loss: 0.0135
Epoch 42/50
 - 2s - loss: 0.0144 - val_loss: 0.0134
Epoch 43/50
 - 2s - loss: 0.0144 - val_loss: 0.0134
Epoch 44/50
 - 1s - loss: 0.0144 - val_loss: 0.0134
Epoch 45/50
 - 2s - loss: 0.0143 - val_loss: 0.0134
Epoch 46/50
 - 2s - loss: 0.0143 - val_loss: 0.0135
```

```
Epoch 47/50
 - 2s - loss: 0.0144 - val_loss: 0.0134
Epoch 48/50
 - 2s - loss: 0.0143 - val_loss: 0.0134
Epoch 49/50
 - 2s - loss: 0.0143 - val_loss: 0.0134
Epoch 50/50
 - 2s - loss: 0.0143 - val_loss: 0.0134
```

**(vi) LSTM Model Comparison Code of Figure 5.20, 5.21 and 5.22.**

```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
	# put it all together
	agg = concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg
```

```
# load dataset
dataset = read_csv('wind_data.csv', header=0, index_col=0)
values = dataset.values
# integer encode direction
encoder = LabelEncoder()
values[:,1] = encoder.fit_transform(values[:,1])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# specify the number of lag hours
n_hours = 3
n_features = 5
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)
print(reframed.shape)

# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print(train_X.shape, len(train_X), train_y.shape)
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)

# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='rmsprop')
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test_X,
test_y), verbose=2, shuffle=False)
# plot history
pyplot.plot(history.history['loss'], label='LSTM_ModelTrain')
pyplot.plot(history.history['val_loss'], label='LSTM_ModelTest')
pyplot.legend()
pyplot.show()

# make a prediction
```

```
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -4:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -4:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

**(vii) Result Generated from (vi)**

```
(10076, 20)
(8760, 15) 8760 (8760,)
(8760, 3, 5) (8760,) (1316, 3, 5) (1316,)
Train on 8760 samples, validate on 1316 samples
Epoch 1/50
 - 3s - loss: 0.0953 - val_loss: 0.0882
Epoch 2/50
 - 1s - loss: 0.0482 - val_loss: 0.0555
Epoch 3/50
 - 1s - loss: 0.0440 - val_loss: 0.0550
Epoch 4/50
 - 1s - loss: 0.0411 - val_loss: 0.0547
Epoch 5/50
 - 1s - loss: 0.0390 - val_loss: 0.0542
Epoch 6/50
 - 1s - loss: 0.0382 - val_loss: 0.0536
Epoch 7/50
 - 2s - loss: 0.0377 - val_loss: 0.0528
Epoch 8/50
 - 2s - loss: 0.0376 - val_loss: 0.0522
Epoch 9/50
 - 1s - loss: 0.0372 - val_loss: 0.0515
Epoch 10/50
 - 1s - loss: 0.0366 - val_loss: 0.0511
Epoch 11/50
 - 1s - loss: 0.0366 - val_loss: 0.0507
Epoch 12/50
 - 1s - loss: 0.0359 - val_loss: 0.0501
Epoch 13/50
 - 1s - loss: 0.0361 - val_loss: 0.0498
Epoch 14/50
```

- 1s - loss: 0.0353 - val_loss: 0.0492
Epoch 15/50
- 1s - loss: 0.0360 - val_loss: 0.0491
Epoch 16/50
- 1s - loss: 0.0348 - val_loss: 0.0486
Epoch 17/50
- 1s - loss: 0.0350 - val_loss: 0.0484
Epoch 18/50
- 1s - loss: 0.0346 - val_loss: 0.0480
Epoch 19/50
- 2s - loss: 0.0344 - val_loss: 0.0478
Epoch 20/50
- 1s - loss: 0.0342 - val_loss: 0.0475
Epoch 21/50
- 1s - loss: 0.0341 - val_loss: 0.0473
Epoch 22/50
- 1s - loss: 0.0342 - val_loss: 0.0472
Epoch 23/50
- 1s - loss: 0.0343 - val_loss: 0.0472
Epoch 24/50
- 1s - loss: 0.0343 - val_loss: 0.0470
Epoch 25/50
- 1s - loss: 0.0343 - val_loss: 0.0471
Epoch 26/50
- 1s - loss: 0.0346 - val_loss: 0.0469
Epoch 27/50
- 2s - loss: 0.0337 - val_loss: 0.0469
Epoch 28/50
- 1s - loss: 0.0352 - val_loss: 0.0468
Epoch 29/50
- 1s - loss: 0.0336 - val_loss: 0.0468
Epoch 30/50
- 1s - loss: 0.0346 - val_loss: 0.0467
Epoch 31/50
- 1s - loss: 0.0337 - val_loss: 0.0467
Epoch 32/50
- 1s - loss: 0.0344 - val_loss: 0.0466
Epoch 33/50
- 1s - loss: 0.0338 - val_loss: 0.0466
Epoch 34/50
- 2s - loss: 0.0341 - val_loss: 0.0465
Epoch 35/50
- 1s - loss: 0.0338 - val_loss: 0.0466
Epoch 36/50
- 1s - loss: 0.0340 - val_loss: 0.0465
Epoch 37/50
- 2s - loss: 0.0337 - val_loss: 0.0465
Epoch 38/50
- 1s - loss: 0.0342 - val_loss: 0.0464
Epoch 39/50

```
 - 1s - loss: 0.0337 - val_loss: 0.0464
Epoch 40/50
 - 2s - loss: 0.0340 - val_loss: 0.0464
Epoch 41/50
 - 1s - loss: 0.0336 - val_loss: 0.0464
Epoch 42/50
 - 1s - loss: 0.0339 - val_loss: 0.0464
Epoch 43/50
 - 1s - loss: 0.0335 - val_loss: 0.0464
Epoch 44/50
 - 1s - loss: 0.0339 - val_loss: 0.0465
Epoch 45/50
 - 1s - loss: 0.0339 - val_loss: 0.0463
Epoch 46/50
 - 1s - loss: 0.0333 - val_loss: 0.0464
Epoch 47/50
 - 1s - loss: 0.0340 - val_loss: 0.0463
Epoch 48/50
 - 1s - loss: 0.0334 - val_loss: 0.0464
Epoch 49/50
 - 1s - loss: 0.0338 - val_loss: 0.0465
Epoch 50/50
 - 1s - loss: 0.0336 - val_loss: 0.0463
```

**(viii) Codes in Generating Figure 5.25**

```
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
	dataX, dataY = [], []
	for i in range(len(dataset)-look_back-1):
		a = dataset[i:(i+look_back), 0]
		dataX.append(a)
		dataY.append(dataset[i + look_back, 0])
	return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
#dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python',
skipfooter=3)
```

```
dataframe = read_csv('wind_data.csv', usecols=[2], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 6
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(20, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
model.summary()
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

**(ix) Results generated from the Code**

```
Epoch 1/100
 - 7s - loss: 0.0081
Epoch 2/100
 - 6s - loss: 0.0052
Epoch 3/100
 - 7s - loss: 0.0047
Epoch 4/100
 - 7s - loss: 0.0044
Epoch 5/100
 - 8s - loss: 0.0042
Epoch 6/100
 - 6s - loss: 0.0042
Epoch 7/100
 - 8s - loss: 0.0042
Epoch 8/100
 - 6s - loss: 0.0042
Epoch 9/100
 - 6s - loss: 0.0041
Epoch 10/100
 - 6s - loss: 0.0041
Epoch 11/100
 - 5s - loss: 0.0041
Epoch 12/100
 - 5s - loss: 0.0041
Epoch 13/100
 - 5s - loss: 0.0041
Epoch 14/100
 - 5s - loss: 0.0041
Epoch 15/100
 - 6s - loss: 0.0040
Epoch 16/100
 - 6s - loss: 0.0041
Epoch 17/100
 - 6s - loss: 0.0041
Epoch 18/100
 - 6s - loss: 0.0041
Epoch 19/100
 - 6s - loss: 0.0041
Epoch 20/100
 - 5s - loss: 0.0040
Epoch 21/100
 - 5s - loss: 0.0041
Epoch 22/100
 - 5s - loss: 0.0041
Epoch 23/100
```

- 6s - loss: 0.0040
Epoch 24/100
- 5s - loss: 0.0041
Epoch 25/100
- 5s - loss: 0.0040
Epoch 26/100
- 6s - loss: 0.0040
Epoch 27/100
- 6s - loss: 0.0040
Epoch 28/100
- 6s - loss: 0.0040
Epoch 29/100
- 6s - loss: 0.0040
Epoch 30/100
- 6s - loss: 0.0040
Epoch 31/100
- 6s - loss: 0.0040
Epoch 32/100
- 7s - loss: 0.0040
Epoch 33/100
- 6s - loss: 0.0040
Epoch 34/100
- 6s - loss: 0.0040
Epoch 35/100
- 5s - loss: 0.0041
Epoch 36/100
- 6s - loss: 0.0040
Epoch 37/100
- 6s - loss: 0.0040
Epoch 38/100
- 7s - loss: 0.0039
Epoch 39/100
- 7s - loss: 0.0040
Epoch 40/100
- 6s - loss: 0.0040
Epoch 41/100
- 6s - loss: 0.0040
Epoch 42/100
- 7s - loss: 0.0040
Epoch 43/100
- 6s - loss: 0.0040
Epoch 44/100
- 6s - loss: 0.0040
Epoch 45/100
- 6s - loss: 0.0040
Epoch 46/100
- 6s - loss: 0.0040
Epoch 47/100
- 7s - loss: 0.0040
Epoch 48/100

- 6s - loss: 0.0039
Epoch 49/100
- 6s - loss: 0.0040
Epoch 50/100
- 6s - loss: 0.0040
Epoch 51/100
- 6s - loss: 0.0040
Epoch 52/100
- 7s - loss: 0.0040
Epoch 53/100
- 7s - loss: 0.0040
Epoch 54/100
- 7s - loss: 0.0039
Epoch 55/100
- 6s - loss: 0.0040
Epoch 56/100
- 6s - loss: 0.0039
Epoch 57/100
- 5s - loss: 0.0040
Epoch 58/100
- 6s - loss: 0.0040
Epoch 59/100
- 6s - loss: 0.0040
Epoch 60/100
- 5s - loss: 0.0039
Epoch 61/100
- 6s - loss: 0.0040
Epoch 62/100
- 5s - loss: 0.0040
Epoch 63/100
- 6s - loss: 0.0040
Epoch 64/100
- 5s - loss: 0.0040
Epoch 65/100
- 5s - loss: 0.0040
Epoch 66/100
- 5s - loss: 0.0040
Epoch 67/100
- 5s - loss: 0.0040
Epoch 68/100
- 6s - loss: 0.0040
Epoch 69/100
- 6s - loss: 0.0040
Epoch 70/100
- 6s - loss: 0.0040
Epoch 71/100
- 5s - loss: 0.0040
Epoch 72/100
- 5s - loss: 0.0040
Epoch 73/100

- 6s - loss: 0.0039
Epoch 74/100
- 6s - loss: 0.0040
Epoch 75/100
- 6s - loss: 0.0040
Epoch 76/100
- 6s - loss: 0.0040
Epoch 77/100
- 5s - loss: 0.0040
Epoch 78/100
- 6s - loss: 0.0040
Epoch 79/100
- 6s - loss: 0.0040
Epoch 80/100
- 5s - loss: 0.0040
Epoch 81/100
- 6s - loss: 0.0040
Epoch 82/100
- 6s - loss: 0.0040
Epoch 83/100
- 6s - loss: 0.0039
Epoch 84/100
- 5s - loss: 0.0039
Epoch 85/100
- 6s - loss: 0.0040
Epoch 86/100
- 6s - loss: 0.0040
Epoch 87/100
- 6s - loss: 0.0040
Epoch 88/100
- 6s - loss: 0.0040
Epoch 89/100
- 6s - loss: 0.0040
Epoch 90/100
- 5s - loss: 0.0040
Epoch 91/100
- 5s - loss: 0.0040
Epoch 92/100
- 5s - loss: 0.0040
Epoch 93/100
- 6s - loss: 0.0040
Epoch 94/100
- 5s - loss: 0.0040
Epoch 95/100
- 6s - loss: 0.0040
Epoch 96/100
- 6s - loss: 0.0040
Epoch 97/100
- 6s - loss: 0.0039
Epoch 98/100

```
 - 6s - loss: 0.0039
Epoch 99/100
 - 6s - loss: 0.0040
Epoch 100/100
 - 5s - loss: 0.0040
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_22 (LSTM)               (None, 20)                2160
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 21
=================================================================
Total params: 2,181
Trainable params: 2,181
Non-trainable params: 0
_________________________________________________________________
Train Score: 0.99 RMSE
Test Score: 0.76 RMSE
```

**(x) Codes in generating Figure 5.24, Applying Dropout**

```
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.constraints import maxnorm
from keras.layers import Dropout
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras import backend as K
import numpy as np
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
	dataX, dataY = [], []
	for i in range(len(dataset)-look_back-1):
		a = dataset[i:(i+look_back), 0]
		dataX.append(a)
		dataY.append(dataset[i + look_back, 0])
	return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('wind_data.csv', usecols=[2], engine='python', skipfooter=3)
dataset = dataframe.values
```

```
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
model = Sequential()
model.add(LSTM(32, input_shape=(1, look_back)))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
# ####################################
# layer_name = 'my_layer'
# get_layer_output = K.function([model.layers[0].input], [model.layers[3].output])
# layer_output = get_layer_output([x][0])
# ####################################
model.fit(trainX, trainY, epochs=100, batch_size=32, verbose=2)
model.summary()
#make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
print(dataset)
plt.plot(scaler.inverse_transform(dataset))
```

```
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

### (xi) Results generated from the Code, Applying Dropout

```
Epoch 1/100
 - 1s - loss: 0.1429
Epoch 2/100
 - 0s - loss: 0.0751
Epoch 3/100
 - 0s - loss: 0.0412
Epoch 4/100
 - 0s - loss: 0.0305
Epoch 5/100
 - 0s - loss: 0.0293
Epoch 6/100
 - 0s - loss: 0.0256
Epoch 7/100
 - 0s - loss: 0.0224
Epoch 8/100
 - 0s - loss: 0.0198
Epoch 9/100
 - 0s - loss: 0.0197
Epoch 10/100
 - 0s - loss: 0.0157
Epoch 11/100
 - 0s - loss: 0.0125
Epoch 12/100
 - 0s - loss: 0.0120
Epoch 13/100
 - 0s - loss: 0.0114
Epoch 14/100
 - 0s - loss: 0.0096
Epoch 15/100
 - 0s - loss: 0.0103
Epoch 16/100
 - 0s - loss: 0.0107
Epoch 17/100
 - 0s - loss: 0.0096
Epoch 18/100
 - 0s - loss: 0.0088
Epoch 19/100
 - 0s - loss: 0.0099
Epoch 20/100
 - 0s - loss: 0.0095
Epoch 21/100
 - 0s - loss: 0.0090
```

```
Epoch 22/100
 - 0s - loss: 0.0086
Epoch 23/100
 - 0s - loss: 0.0089
Epoch 24/100
 - 0s - loss: 0.0079
Epoch 25/100
 - 0s - loss: 0.0080
Epoch 26/100
 - 0s - loss: 0.0089
Epoch 27/100
 - 0s - loss: 0.0080
Epoch 28/100
 - 0s - loss: 0.0089
Epoch 29/100
 - 0s - loss: 0.0080
Epoch 30/100
 - 0s - loss: 0.0079
Epoch 31/100
 - 0s - loss: 0.0081
Epoch 32/100
 - 0s - loss: 0.0080
Epoch 33/100
 - 0s - loss: 0.0080
Epoch 34/100
 - 0s - loss: 0.0082
Epoch 35/100
 - 0s - loss: 0.0074
Epoch 36/100
 - 0s - loss: 0.0077
Epoch 37/100
 - 0s - loss: 0.0078
Epoch 38/100
 - 0s - loss: 0.0074
Epoch 39/100
 - 0s - loss: 0.0075
Epoch 40/100
 - 0s - loss: 0.0075
Epoch 41/100
 - 0s - loss: 0.0075
Epoch 42/100
 - 0s - loss: 0.0073
Epoch 43/100
 - 0s - loss: 0.0070
Epoch 44/100
 - 0s - loss: 0.0073
Epoch 45/100
 - 0s - loss: 0.0075
Epoch 46/100
 - 0s - loss: 0.0070
```

```
Epoch 47/100
 - 0s - loss: 0.0067
Epoch 48/100
 - 0s - loss: 0.0066
Epoch 49/100
 - 0s - loss: 0.0067
Epoch 50/100
 - 0s - loss: 0.0067
Epoch 51/100
 - 0s - loss: 0.0069
Epoch 52/100
 - 0s - loss: 0.0069
Epoch 53/100
 - 0s - loss: 0.0067
Epoch 54/100
 - 0s - loss: 0.0069
Epoch 55/100
 - 0s - loss: 0.0069
Epoch 56/100
 - 0s - loss: 0.0066
Epoch 57/100
 - 0s - loss: 0.0064
Epoch 58/100
 - 0s - loss: 0.0067
Epoch 59/100
 - 0s - loss: 0.0067
Epoch 60/100
 - 0s - loss: 0.0061
Epoch 61/100
 - 0s - loss: 0.0067
Epoch 62/100
 - 0s - loss: 0.0062
Epoch 63/100
 - 0s - loss: 0.0064
Epoch 64/100
 - 0s - loss: 0.0064
Epoch 65/100
 - 0s - loss: 0.0066
Epoch 66/100
 - 0s - loss: 0.0063
Epoch 67/100
 - 0s - loss: 0.0061
Epoch 68/100
 - 0s - loss: 0.0061
Epoch 69/100
 - 0s - loss: 0.0062
Epoch 70/100
 - 0s - loss: 0.0060
Epoch 71/100
 - 0s - loss: 0.0063
```

```
Epoch 72/100
 - 0s - loss: 0.0057
Epoch 73/100
 - 0s - loss: 0.0058
Epoch 74/100
 - 0s - loss: 0.0057
Epoch 75/100
 - 0s - loss: 0.0059
Epoch 76/100
 - 0s - loss: 0.0060
Epoch 77/100
 - 0s - loss: 0.0061
Epoch 78/100
 - 0s - loss: 0.0057
Epoch 79/100
 - 0s - loss: 0.0057
Epoch 80/100
 - 0s - loss: 0.0058
Epoch 81/100
 - 0s - loss: 0.0059
Epoch 82/100
 - 0s - loss: 0.0061
Epoch 83/100
 - 0s - loss: 0.0061
Epoch 84/100
 - 0s - loss: 0.0062
Epoch 85/100
 - 0s - loss: 0.0061
Epoch 86/100
 - 0s - loss: 0.0055
Epoch 87/100
 - 0s - loss: 0.0058
Epoch 88/100
 - 0s - loss: 0.0059
Epoch 89/100
 - 0s - loss: 0.0057
Epoch 90/100
 - 0s - loss: 0.0059
Epoch 91/100
 - 0s - loss: 0.0055
Epoch 92/100
 - 0s - loss: 0.0057
Epoch 93/100
 - 0s - loss: 0.0059
Epoch 94/100
 - 0s - loss: 0.0061
Epoch 95/100
 - 0s - loss: 0.0060
Epoch 96/100
 - 0s - loss: 0.0056
```

```
Epoch 97/100
 - 0s - loss: 0.0055
Epoch 98/100
 - 0s - loss: 0.0058
Epoch 99/100
 - 0s - loss: 0.0056
Epoch 100/100
 - 0s - loss: 0.0053
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 32)                4352
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
Train Score: 0.91 RMSE
Test Score: 0.74 RMSE
[[ 0.40508118]
 [ 0.45871559]
 [ 0.40508118]
 ...,
 [ 0.40508118]
 [ 0.43189836]
 [ 0.45871559]]
```

**(xii) Sample Codes for creating lag features**

```
# create lag features
from pandas import Series
from pandas import DataFrame
from pandas import concat
series = Series.from_csv('wind_data.csv' , header=0)
ws = DataFrame(series.values)
dataframe = concat([ws.shift(5), ws.shift(4), ws.shift(3), ws.shift(2), ws.shift(1), ws], axis=1)
dataframe.columns = ['t-4', 't-3','t-2', 't-1', 't', 't+1']
print(dataframe.head(10))
```

(xiii) Sample Results from (xii)

```
   t-4   t-3   t-2   t-1     t   t+1
0   NaN   NaN   NaN   NaN   NaN  6.09
1   NaN   NaN   NaN   NaN  6.09  6.85
2   NaN   NaN   NaN  6.09  6.85  6.09
```

3 NaN NaN 6.09 6.85 6.09 5.71
4 NaN 6.09 6.85 6.09 5.71 4.56
5 6.09 6.85 6.09 5.71 4.56 3.80
6 6.85 6.09 5.71 4.56 3.80 4.17
7 6.09 5.71 4.56 3.80 4.17 3.80
8 5.71 4.56 3.80 4.17 3.80 3.80
9 4.56 3.80 4.17 3.80 3.80 4.94

### (xiii) Python Implementation of Inverted Dropout method.

```
p = 0.5 # probability of keeping a unit active. higher = less
dropout

def train_step(X):
      # forward pass for example 3-layer neural network
     H1 = np.maximum (0, np.dot(W1, X) + 11
     U1 = (np.random.rand*H1. shape) <p) #/p first dropout mask.
Notice /p!
     H1 *= U1 # drop!
     H2 = np.maximum , np.dot (W2, H1) + 52)
     U2 = (np.random.rand (*H2. shape) <p) #/p second dropout mask.
Notice /p.
     H2 *= U2 # drop!
     out = np.dot (W3, H2) + b3

# backward pass: compute gradients... (not shown)
# perform parameter update... (not shown)
```

Test time is unchanged

```
def predict(X):
         # ensembled forward pass
         H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
         H2 = np.maximum(, np.dot (W2, H1) + b2)
         out = np.dot (W3, H2) + b3
```

From the code, p = 0.5 which is the probability that a given hidden node would be kept. This means that there is a 50% chance of eliminating any hidden unit. The procedure generate a random metrics ***U1***, meaning there is a 0.5 chance the corresponding ***U1*** is 1 and 50% chance of being 0. At ***H2***, every element that =0, has a 50% chance of being 0, thereby zeroing out the corresponding element ***U2***. ***H1*** will finally be divided up by 0.5 ie ***U2*** /= 0.5 (p)

The final step out is a 10 X 1 dimensional array. Therefore, if with 50% of keeping and 50% elimination, on average, there is an m unit shut-off.

Note: in Python, ***U2*** is a Boolean array of T or F rather than 0 or 1.

# Appendix B

## (i) Comparison Code of the Model

```
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from pandas import read_csv
from pandas import datetime
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
	dataX, dataY = [], []
	for i in range(len(dataset)-look_back-1):
		a = dataset[i:(i+look_back), 0]
		dataX.append(a)
		dataY.append(dataset[i + look_back, 0])
	return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = read_csv('wind_data.csv', usecols=[2], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
#Create and fit ARIMA Model
# fit model
model = ARIMA(series, order=(1,1,2))
```

```
model_fit = model.fit(disp=0)
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(50, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# create and fit the eLSTM network
model = Sequential()
model.add(LSTM(50, input_shape=(1, look_back)))
model.add(Dropout(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
#Piping the Algorithm
for models in models = []:
   models.append(('ARIMA', ARIMARegression()))
   models.append(('LSTM', LSTMRegression()))
   models.append(('eLSTM', eLSTM()))
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
print(model_fit.summary())
# plot residual errors
residuals = DataFrame(model_fit.resid)
residuals.plot()
plt.show()
```