LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Faculty of Technology
Environmental Energy Technology

Nicolus Kibet Rotich Nicholas

FORECASTING OF WIND SPEEDS AND DIRECTIONS WITH ARTIFICIAL NEURAL NETWORKS

Examiners: Prof. Lassi Linnanen, PhD (Economics)
Prof. Jari Backman, PhD (Technology)
Supervisor: Daniil Perfiliev, Msc (Technology)

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology
Environmental Energy Technology

Nicolus Kibet Rotich Nicholas

Forecasting of wind speeds and directions with artificial neural networks

Master's Thesis

2012

69 pages, 26 figures, 9 tables, 4 annexes

Examiners: Prof. Lassi Linnanen, PhD (Economics)
Prof. Jari Backman, PhD (Technology)

Supervisor: Daniil Perfiliev, Msc (Technology)

Keywords: Neural Networks, Modeling, Wind Speeds, Directions, Forecasting

In this master's thesis, wind speeds and directions were modelled with the aim of developing suitable models for hourly, daily, weekly and monthly forecasting. Artificial Neural Networks implemented in MATLAB software were used to perform the forecasts. Three main types of artificial neural network were built, namely: Feed forward neural networks, Jordan Elman neural networks and Cascade forward neural networks. Four sub models of each of these neural networks were also built, corresponding to the four forecast horizons, for both wind speeds and directions. A single neural network topology was used for each of the forecast horizons, regardless of the model type. All the models were then trained with real data of wind speeds and directions collected over a period of two years in the municipal region of Puumala in Finland. Only 70% of the data was used for training, validation and testing of the models, while the second last 15% of the data was presented to the trained models for verification. The model outputs were then compared to the last 15% of the original data, by measuring the mean square errors and sum square errors between them. Based on the results, the feed forward networks returned the lowest generalization errors for hourly, weekly and monthly forecasts of wind speeds; Jordan Elman networks returned the lowest errors when used for forecasting of daily wind speeds. Cascade forward networks gave the lowest errors when used for forecasting daily, weekly and monthly wind directions; Jordan Elman networks returned the lowest errors when used for hourly forecasting. The errors were relatively low during training of the models, but shot up upon simulation with new inputs. In addition, a combination of hyperbolic tangent transfer functions for both hidden and output layers returned better results compared to other combinations of transfer functions. In general, wind speeds were more predictable as compared to wind directions, opening up opportunities for further research into building better models for wind direction forecasting.

ACKNOWLEDGEMENTS

I would like to pass my sincere gratitude to my supervising professors. First, Prof. Lassi Linnanen, who played a key role in preparing me for my final thesis, by providing me with individual project that boosted my writing skills and for advising throughout the entire degree program. Secondly, is to Prof. Jari Backman, for his enduring technical as well as logistical support for conducting this master’s thesis. My acknowledgments also go to MSc. Daniil Perfiliev, for closely observing my work as well as offering guidance and direction. I specifically appreciate Daniil’s availability and his invaluable time, spent towards improving my work. Most of the meetings we had were conducted without the need for formal appointments, notwithstanding his busy schedule working on PhD dissertation. Last but certainly not least, is to D.Sc. Matylda Jabłońska, whom we had quality talks about my thesis, sometimes even on Skype. I also wish to thank all my family members and friends for their encouragement and spiritual support throughout the entire study period. The moment I understood artificial neural networks, I now can see everything in a wholly different perspective. I will forever be indebted to all of you and may the almighty God bless you abundantly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYBOLS AND ABBREVIATIONS

q_m	The Mass flow of air/wind on the wind turbine blades, *kg/s*
$\boldsymbol{\rho}$	Air density, *kg/m*^{3}
$\boldsymbol{v}$	Wind speed/velocity, *m/s*
$\boldsymbol{A}$	The projected area of the wind turbine blades, *m*^{2}
q_v	The volumetric flow of air/wind on the wind turbine, *m*^{3}*/s*
$\boldsymbol{l_e}$	The electrical loses of the wind turbine generator, *dimensionless fraction*
$\boldsymbol{P_{th}}$	Total power generated by a wind turbine, *watt*
$\boldsymbol{C_p}$	Power coefficient of the wind turbine, *dimensionless fraction*
$\boldsymbol{\mu}$	Neural network momentum coefficient, *dimensionless constant*
η	Neural network learning rate, *dimensionless constant*
∞	Infinity
$\boldsymbol{H}$	Number of hidden layers in a neural network
ζ	Neural network transfer function
$\boldsymbol{N}$	The number of samples of data in error measurements
$\boldsymbol{R}$	Coefficient of Correlation, *dimensionless fraction*
$\boldsymbol{T}$	Threshold constant
ABL	Atmospheric Boundary Layer
ADALINE	Adaptive Linear Element
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARX	Autoregressive eXogeneous
BBP	Batch Back Propagation
BR	Bayesian regulation/regularization
CDM	Clean Development Mechanism
CER	Carbon Emission Reduction
CFNN	Cascade Forward Neural Networks
CNN	Cable News Network
DOE	Designated Operational Entity
FFNN	Feed Forward Neural Network
GA	Genetic Algorithm
GUI	Graphical User Interface
IBP	Incremental Back Propagation
JENN	Jordan Elman Neural Network
LM	Levenberg-Marquardt
LTG	Linear Threshold Gate
LUT	Lappeenranta University of Technology
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MATLAB	Matrix Laboratory
MLFFNN	Multi-Linear Feed Forward Neural Network
MM5	Fifth-generation Meso-scale Models
MPC	Model Predictive Control
MSE	Mean Square Error
MSEt:	Mean Square Error upon training
MSEv:	Mean Square Error upon verification

NWP	Numerical Weather Prediction
PBL	Planetary Boundary Layer
PS	Premature Saturation
PTG	Polynomial Threshold Gate
QP	Quick Propagation
QTG	Quadratic Threshold Gate
RBF	Radial Bases Functions
RMSE	Root Mean Square Error
RSM	Regional Spectra Model
SNNS	Stuttgart Neural Network Simulator
SSE	Sum Square Error
SSEt:	Sum Square Error upon training
SSEv:	Sum Square Error upon verification
USA	United States of America
VB	Visual Basic
WPPT	Wind Power Prediction Tool
WRF	Weather Research and Forecast
WSJ	Wall Street Journal

1. INTRODUCTION

1.1 The history of Wind Energy

Wind power has been in use for a number of centuries now. The earliest wind machine recorded in English is dated 1911; and in Holland the first grain grinding wind turbine was build in 1439 (Johnson, 2001, 1-2). Wind machines played an important role continuously throughout the pre and post industrial revolution period (1750s-1850s). The main focus in the early times however was the mechanical energy, which was needed for grinding of grains and pumping of fluids (mainly water). With the advent of industrialization a midst the twentieth century, wind machines were slowly replaced by fossil fuels and electrical grids due to the inconsistency and unreliability of the wind power. (Ackermann and Söder, 2002, 69.)

In 1970s wind energy technology, now with a focus of electricity generation became a vast developing field. This was driven by the need for back-up systems and accelerated by environmental lobby groups for a paradigm shift from fossil fuel use to cleaner and renewable sources of energies that are deemed environmentally friendly. Denmark and USA became the first countries to generate electricity from wind. Wind is arguably one of the most cost effective sources of renewable energy, which does not pollute nor get depleted faster than is generated. (Grogg, 2005, 1). While the field is continuing to attract immense attention globally, more research is needed in order to understand the future dynamism of opportunities as well as the challenges associated and how well to address them in advance.

The total energy that reaches the earth surface from the sun is about 1.74×10^{17} watts power. This is approximately equal to 160 times the global fossil fuel reservoirs. A small portion (about 1-2%) of the sun's energy goes to the formation of wind. Wind formation phenomenon is understood as caused by uneven heating/warming of atmospheric air by the sun, forming a 'void' that creates a pressure drop. Direct sun rays reaching the equatorial region make it hotter, causing the hot air to rise (Brownian motion), move and settle in the cooler regions in the northern and southern hemispheres, while the cold air moves beneath it to occupy the void left. Wind formation has also been attributed to the 'Coriolis' effect caused by the rotation of the earth, shifting space objects to the right in the northern hemisphere and to the left in the southern hemisphere. (Boyle, 1996, 29.)

In tapping energy from the wind, long term knowledge of local weather conditions is a key component. Specific wind parameters e.g. speed and direction; density and atmospheric temperatures are some of the important measurements, whose future must be estimated by forecasting from current and past data.

Wind speed and direction forecasting is of interest in many industries at the moment. Marine engineers need forecasting for many operational and construction oriented activities, necessary for planning and scheduling of available resource as well as determining which, when and how much more should be acquired.

It is also of interest in the aviation industry as wind speeds affect to a larger extent aircraft safety during landings and take offs. This prompts for forecasting or rather 'nowcasting' and communicating of the results to the flight crew from a range of a couple of minutes to a number of days into the future.

In wind engineering, wind electricity generation companies need forecasting for optimum site selection during initial visibility studies of wind farm projects. Perhaps to make it more relevant to the current state of energy and environmental technology advancement, is that wind power forecasting is necessary for emission trading experts to estimate the planned carbon offsets from a wind turbine, for a given future period. This forms the basis for project approval, validation by relevant designated operational entities (DOE) and allows efficient monitoring of the emission reduction, verification and possible issuance of the product i.e. carbon emission reductions (CER), for Clean Development Mechanism (CDM) projects.

Electricity trading companies require electricity load forecasting in order to assure their customers of availability and reliability of the commodity, apart from other operational tasks e.g. load switching and infrastructural development (Alfares, 2002, 23). Theoretically speaking, generated wind power is proportional to the square of the rotor diameter and the cube of wind speed (Eq. 1). This implies that twice the wind speed yield an eightfold of expected power output, or twice the rotor diameter increases expected power output by a fourfold and vice versa (Boyle, 1996, 275; Houtzager, 2011, 5.)

Wind speed and direction forecast models are important in control engineering as they are used in data driven control and also in building linear/nonlinear model predictive controls (MPC), for optimization of wind energy generation. Information obtained from these systems forms a critical phase in designing and optimizing rotor control hardware e.g. yaw mechanism. (Houtzager, 2011, 15-18.)

Wind turbine extracts energy from a moving mass of air (wind) and converts its kinetic energy into mechanical energy from which the generator/alternator transforms it into the electrical energy we know. On the other hand, in order for the wind turbine to rotate and generate the mechanical energy, there must be a pressure drop between the windward and leeward side of the wind turbine blades. Putting the above descriptions together, it can then be interpreted that wind power generation is an optimization problem in which the objective is to compromise between the two among other constraints; the air mass flow allowed to pass through to the leeward side of the wind turbine blades and the amount converted into mechanical energy.

The theoretical energy carried by the wind just before coming in contact with the wind turbine blades is the kinetic energy expressed as;

$$P_{th} = \frac{1}{2} q_m v^2 \tag{1}$$

Where q_m is the air mass flow of the wind and v is the velocity of the wind mass.

$$q_m = \rho q_v \tag{2}$$

where ρ is the air density and q_v is the volumetric flow of air that comes in contact or passes through the wind turbine projected area.

$$q_v = vA \tag{3}$$

A is the projected area covered by the wind turbine blades during rotation. We can therefore write.

$$P_{th} = \frac{1}{2} \rho A v^3 \tag{4}$$

Each wind turbine has its own electrical loses, and an overall efficiency referred to as the power coefficient C_p, which determines the maximum amount of power that can be generated from a specific wind turbine. Therefore;

$$P_{th} = \frac{1}{2} C_p \rho l_e A v^3 \tag{5}$$

where C_p is the power coefficient and l_e is the electrical loss factor ranging from 0.9-0.95. According to the German physicist Albert Betz, the maximum C_p is 59.3%. As seen from above basic wind energy calculation equations, accurate wind speeds forecasting is as good as accurate wind power forecasting, when all other factors are kept constant i.e. for identical wind turbines of equal coefficient of power, projected area and constant air density. This fact calls for accurate methods for wind speed forecasting, if the results are to be reliable for all the relevant parties described above. Erroneous wind speed forecasting would lead to largely propagated errors in the expected power output.

Like many real life problems, wind speed patterns are highly dynamic and non-linear and thus cannot be accurately forecast using conventional linear regression models. However, this is possible to some extent with non-linear mathematical modeling techniques which are rather complex. This has been made possible by computational tools that are available in the modern day industry. The main challenge is that enough real past data is required to build and train the models before application in the forecasting exercises.

Data measurement over a long period of time is both tedious and uneconomical, especially when considering the urgency for wind projects development timeliness. As a result, one of the best practices in the industry has been to take accurate measurement over a shorter time and extrapolate over the required time period e.g. daily measurements can be used to forecast weekly data, weekly measurements used to forecast monthly data and monthly measurements to estimate annual wind data, etc. (More and Deo, 2003, 35.)

1.2 Objectives and scope of the study

The scope of this master's thesis is confined to the statistical approach of forecasting of wind speeds and directions using real experimental data collected over a period of two years. Specific wind data from the municipal region of Puumala, in Finland was used to obtain both one-step-ahead forecasts of wind speeds and directions, as well as a one point per sliding window for the entire data length. The tools used for forecasting are artificial neural networks (ANN), nonlinear models built with MATLAB computer software. The results were then compared by statistical analysis, presented, interpreted, and discussed. The rationale behind the preference of the above concept is based on past researches that show the ability of ANN to return better results compared to conventional time series forecasting, (Aladag et *al*., 2009, 1467; Panteri and Papathanassiou, 2008, 8 ; Zhang et *al*., 1998, 35.)

The univariate vectors of wind speeds and directions were separately used to build a time-lagged groups of inputs, which technically involves shifting the time base back by a given number of observations. These observations can be classified as hours, half-days, days, weeks, months, half-years or couple of years depending on the required forecast horizons and the data available. This operation is also helpful in varying the number of inputs/outputs during model construction. The model is then built, trained, validated and tested before application. It should however be understandable that the art and science of forecasting is not meant to give exact results, but to provide estimates that can be used to guide decision making processes, deemed necessary than not forecasting at all.

1.3 The structure of the thesis

The thesis is organized into six main parts, which are further reorganized into relevant subdivisions. The first chapter reviews the historical use of wind power, dating back to the pre industrial revolution era, to the most recent advancements in wind power engineering. The mechanism in which wind energy is tapped and transformed into usable electrical energy is also highlighted with a focus of the effect of wind speeds on the expected power output. Within the first chapter also, a few industries in which wind speed and direction are of interest are listed, pointing out some important issues on each. However, more attention is directed to the relevance of the study in the field of energy and environmental technology. Finally the objective and scope of the study is set, outlining the justification for the preferred computational tools chosen for the study.

Secondly, the historical development of artificial neural networks (ANN), also referred to as neurocomputing, is reviewed and their classifications, specifically as per architectures and functionalities are outlined. Some of the current known real world applications of ANNs are then highlighted with a specific focus on feed forward neural networks (FFNN), which have been applied across most scientific and engineering disciplines.

The third chapter involves selection of appropriate computer application or software, taking into account the mass of the data involved in the computation and data pre-processing. In this section also the methodology of ‘black box’ modeling of wind speeds and directions is discussed, first by selecting the simulation parameters then construction of the models, validation, testing and using them for forecasting. The last three sections entail presentation, discussion and interpretation of the results obtained from each of the model type i.e. (wind speeds and directions), before making conclusion and recommendations for further research work.

1.4 Data acquisition

The data used herein was provided by Lappeenranta University of Technology (LUT) and granted the author permission to use as part of the master’s thesis. Measurements of wind speeds and directions were collected over a period of approximately 2 years from 1.11.2009 up until 30.10.2011. The data sampling intervals was 10 minutes and were taken at a height of 60 meters from the ground in the municipal region of Puumala, Finland.

1.5 Background to wind Power Prediction

Several wind power prediction models have been developed in the recent past. However, different models are suitable for various types of situations, depending on the nature of the required forecast. Some models are better suited for long-term forecasting while others are better for short term forecasting. In this study, ‘short term’ forecasting is used to underscore forecasts of one step ahead i.e. one hour, day, week or a month ahead, while ‘long term’ means forecasting of wind speeds/directions on a ‘one point per step’, for a maximum period equivalent to the entire length of the data used for the study, e.g. one day measurement every week for the entire 2 years. The suitability of a model can be assessed by the number of time steps into the future, the model can be used while still retaining its robustness on the predicted outputs, without losing its generalization ability. Generalization of a model is the ability to produce accurate results even for input data set that the model has not ‘seen’ i.e. not used in the training of the model (Kavzoglu, 1999). In general, three approaches of wind forecasting methods have been well documented so far (2012); the numerical weather prediction models (NWP), physical systems approach and the statistical approaches.

1.6 Numerical Weather Prediction Systems

The numerical weather prediction (NWP) system simulates the atmosphere by numerically integrating the equations of motion starting from the current atmospheric states. This is done by mapping the real world on to a discrete 3-D computational grid that divides the globe into numerous polygonal patterns of certain dimensions e.g. 60 by 60 square kilometres. NWP models are complex and expensive due to its data collection requirement intensity, they are thus operated by national authorities’ weather services. With these models, the resolutions can be localized with a smaller domain to correspond with the home country e.g. it can be reduced to a finite resolution of 7 by 7 sq. km or less. However, this means that variables calculated at each grid point is an average of the grid cell and thus the prediction is obviously not optimally time-averaged for all the grid points within the cell. (Lange, 2003, 7.)

1.7 Physical Systems

Physical systems, model the dynamics of the atmosphere by parametrization of the planetary boundary layer (PBL) concept, also known as the atmospheric boundary layer (ABL). ABL is the lowest part of the atmosphere that is in continuous contact with the surface of the earth. Here, the physical quantities e.g. velocity, temperature and moisture (of the wind/air) are turbulent and vertical mixing is stronger. In principle there are two basic forms of physical prediction systems. Those based on the operational fluid dynamical simulations similar to those of NWP systems, and diagnostic models which works by parametrization of the PBL. The ABL concept has also attracted a lot of research interest in the recent past. Physical systems are further broken down into two, the numerical simulations and diagnostic models.

Numerical simulations can vaguely be referred to as an extension of the NWP systems. As discussed above, NWP cannot explicitly predict the wind speed at a point in space but gives a generalized solution for a grid value. Numerical simulations vividly model the atmospheric phenomena ranging from classes similar to those of NWP system (1000-10 km) down through meso-scale (10-1 km) weather systems to micro-scale (100 m - 0.01 m) levels. (Lange, 2003, 18.)

Some of the numerical models that have been developed based on parametrization of the planetary boundary layer are; Fifth-generation Mesoscale Model (MM5), Weather Research and Forecasting (WRF) model and Regional Spectra Model (RSM), discussed by (Kwun et *al.*, 2007).

Diagnostic models, like numerical models, are also based on parametrization of the planetary boundary layer flow, but without further dynamical situations. This is done by refinement of the results obtained from NWP system. The output of the NWP is adapted to the local conditions e.g. surface roughness, orography of surface terrain, obstacles and thermal gradient of the atmosphere, which has proved to produce better forecast results which are non-persistent, i.e. present value, is not necessarily dependent on the past. One such system has been developed by Landberg at the National Laboratory in Risǿ, Denmark in 1993. The system has since been commercialized under the name *Prediktor.* (Lange, 2003, 8.)

Previento, an application based on the same principle as Prediktor has been developed by the University of Oldenburg, Germany. Previento models the boundary layer with regard to roughness, orography, and wake effects. It also considers the thermal stratification which affects the logarithmic profile of wind speeds at hub heights and most of all, it has the ability to undertake regional forecast that is necessary in aggregating the total wind power output from a location with several wind farms. This has been tested for predictions of 30 sites in the Northern part of Germany. (Focken et *al.*, 2001.)

1.8 Statistical systems approach

Statistical systems are implemented based on training of the models with a sample of real data specific to that a location, taken over a number of discrete periodic cycles. The difference between the predicted output and the required output (error) is minimized by fine-tuning it to a level which can be used for nowcasting and/or forecasting. Statistical approaches are further divided into three subdivisions. Wind Power Prediction Tool (WPPT), is a statistical tool developed and operated by the Danish national laboratories for weather forecasting. The WPPT is based on an autoregressive eXogeneous (ARX) input type model, where wind speed and therefore power is described as a non-linear, non-stationary and time-varying stochastic process representing the dynamics of the atmosphere. The second statistical approach is that which treat future wind speeds as vague or indistinct and thus tries to solve by reasonable approximation with fuzzy logic concept. Such system has been developed and is currently operated for short term predictions by Ecole des Mines de Paris, France. Artificial Neural Networks (ANN) is the third statistical approach which is one of the most recently developed methods for accurate forecasting. The artificial neural networks are the subject of the current thesis and are dealt with in detail on the following section. Figure1. Shows a summary of wind power prediction methods discussed above and the focus of the thesis.

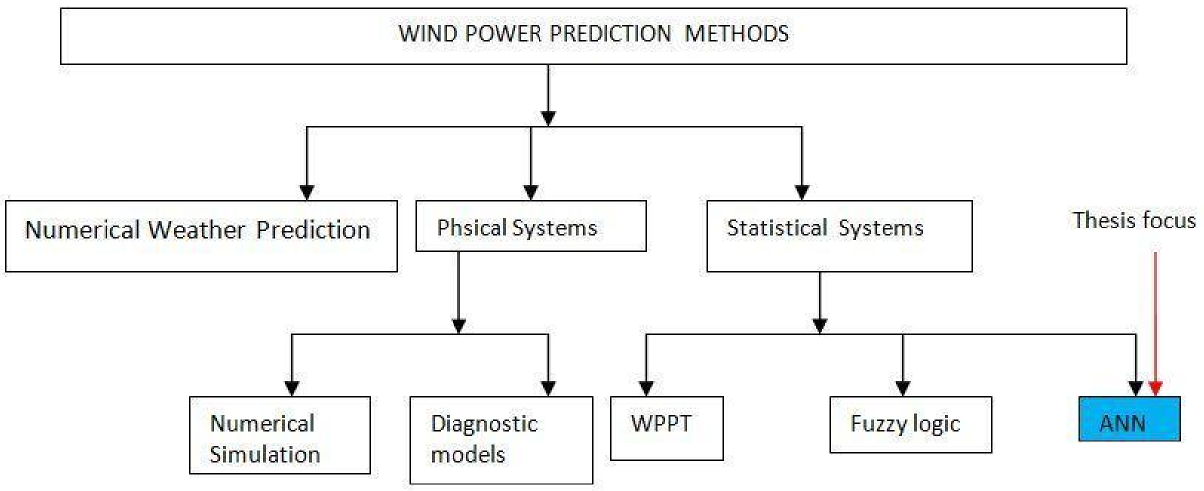


Figure 1: Summary of wind power prediction systems, based on (Lange, 2003)

2. ARTIFICIAL NEURAL NETWORKS

2.1 Evolution of neurocomputing

By definition, an artificial neural network (ANN) is a structure comprised of densely interconnected adaptive simple processing elements that are capable of performing massively parallel computations for data processing and knowledge representation (Basheer and Hajmeer, 2000, 3). The very first artificial neural networks were inspired by the biological neuron from which its structure and functioning have been mimicked extensively in modern computing. A biological neuron consists of the cell body part that acts as the central command point, dendrites that act as transmitters and axon that connects the body part to the synapses. The human brain is composed of numerous interconnected nodes of the neurons. Each node receives input signals from external environment or from neighbouring neurons and processes locally (independently). If the processed signal is strong enough, it causes 'activation' to produces an output which is passed on to the next 'layer' of nodes or to external outputs (effectors) to trigger response. (Zhang, 1998, 37.)

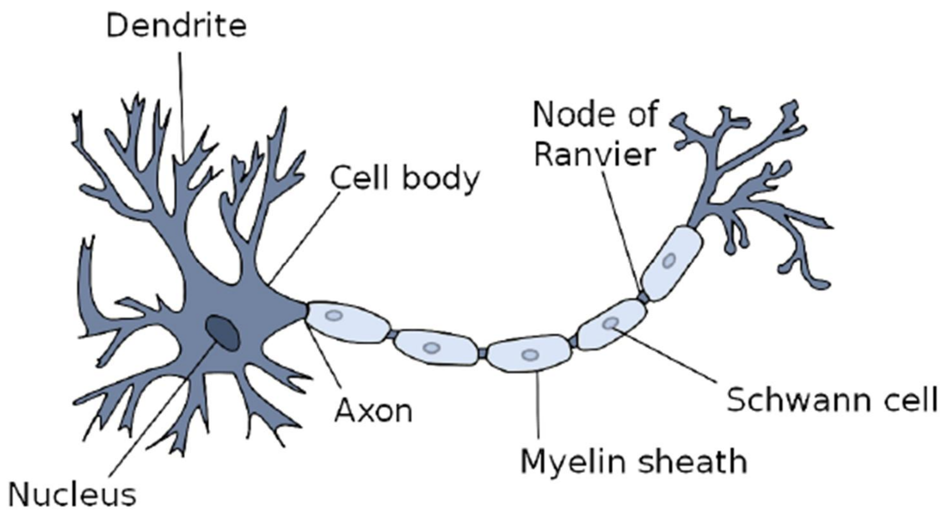


Figure 2: Simplified structure of a biological neuron (Kriesel, 2005)

An artificial neuron dates back to 1943, when psychiatrist Warren McCulloch and Mathematician Walter Pitts introduced a simple neuron. Upon further studies they discovered that biological neurons could be represented as conceptual circuit components that can be used to perform several computational tasks. (Kröse and Smagt, 1996, 13.)

A typical ANN consists of an input layer of neurons, hidden layer and the output layer, all interconnected with weights of different strengths that can be excitatory or inhibitory. Due to these interconnections, the ANN therefore possesses a powerful computational power to learn from examples and generalize the solution to a wide range of problems. The input layer does not perform any computation and therefore in a network only neurons in the hidden and the output layers are counted. The arrangement of neurons in a layer and layers in a network is called neural network topology or architecture. This is important as it defines the structure of ANNs in general and determines their applicability. It is believed that in any study pertaining ANNs, choice of the topology and simulation, parameters forms a greater part of the work, with the commonly applied being the typical feed forward neural networks (FFNN). The fig. 3 below shows a FFNN with three neurons in the input layer, three neurons in the hidden layer and three in the output layer. It is important to mention also herein that, the number of neurons in the input layer corresponds to the number of input parameters and the same case applies to the output layer (fig. 3). Various types of neural networks are presented in the next section.

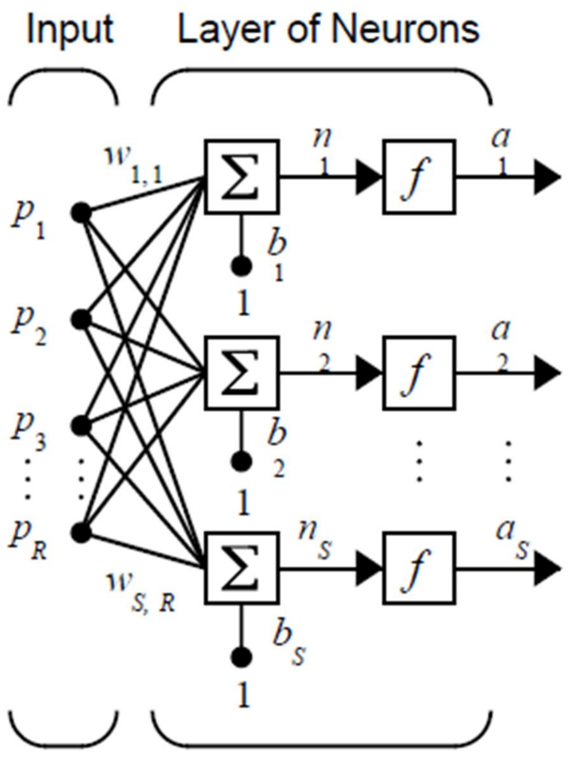
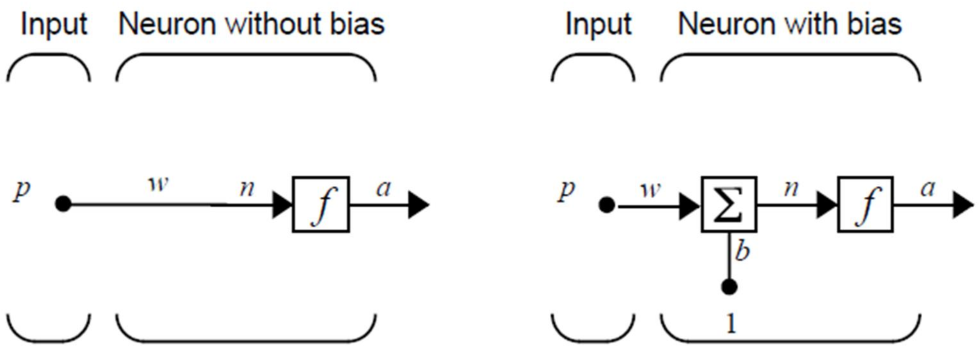


Figure 3: A simple feed forward neural network with R inputs and S neurons in the hidden layer and S outputs (Hagan et *al.* 1996)

2.2 Classifications of neural networks

ANN architecture constitutes the inputs input layer, hidden layer and the output layer together with their summing points and transfer or activation functions. Sometimes more often than not, a *bias* is added to the total weighted sum of a neural network layer. Neural networks can be categorized by virtue of a number of properties, described in (Basheer and Hajmeer, 2000, 12). In the present study, neural networks are broadly classified into two main categories for simplicity; by architecture and functionality.

a. **Architecture**: The architecture of a neural network also seen as the arrangement of nodes or neurons in a layer and layers in the entire network can be used as a network's distinction characteristic. A neuron model consists of a scalar or vector inputs, and may or may not have a bias, and that makes the whole difference in the network architecture, see fig. 4 (i) and (ii) below.



i. Simple neuron without bias ii) Simple neuron with bias

Figure 4: Illustration of a simple neuron with and without bias (Hagan *et al.*, 1996)

As the problem in question becomes more complex, a more complex neural network is desired for faster and efficient computation. This necessitates the use of networks with more than one layer in what will now be referred to as a multiple layer neural network (Hagan *et al.*, 1996). Neural networks can therefore be classified as ones with a single layer of neurons or multiple layers of neurons. Figure 5 below shows a typical multi-layer feed forward neural networks (MLFFNN).

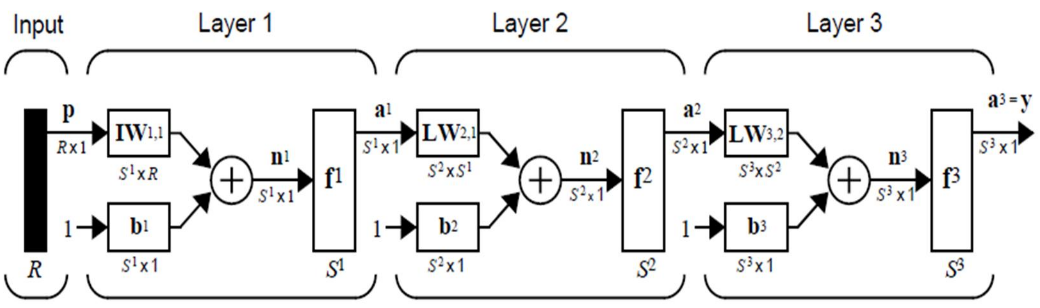


Figure 5: A three-layer neural network of R inputs, S1, S2 and S3 neurons in each layer and y outputs (Hagan *et al*., 1996)

Neural networks can also be classified as recurrent or non-recurrent. A recurrent network means some of the outputs can be connected to the input neurons as feed-backs (fig.5). These set of neural networks are computationally more powerful than simple feed forward networks earlier seen (Lawrence *et al.,* 2000, 1).

b. **Functionality**: ANNs are designed to solve a wide range of problems e.g. associative memory, generalization, optimization, data reduction, prediction and control, and pattern recognition. To achieve these specific functions, several types of network architectures have been designed, which can be listed as follows; Adaptive linear element (ADALINE), Hemming network, Hopfield network, Kohonen network, Boltzmann machine, multi-layer feed forward neural network learned by Back propagation algorithm etc. (Lek & Guégan, 1999, 67.)

2.3 Applications of Artificial Neural Networks

In the present day neural networks cover a wide area of applications ranging from business, engineering, research and development as well as financial applications. They have been used extensively in the business and insurance sectors for planning, operations and product optimization and for insurance policy application and evaluation respectively. Credit facility institutions use neural networks to spot unusual credit card activities that may be associated to lose of the credit cards as well as in many other forensics. They have also used them to predict the risks of bankruptcy, stock markets etc. (Fadlalla & Lin, 2001, 113.)

In engineering, neural networks have been used in the aerospace industry for fault diagnosis, autopilot enhancement, flight path simulation etc. The military uses neural networks for weapon steering and target tracking, object discrimination and classification, facial recognition etc.

Modern day media and entertainment industry employs holographic neural networks for discretization of continuous functions or digitalization of images and stimulus symmetrization to achieve the desired special effects as in motion pictures (Manger, 98, 124). Manger concludes by inferring that, holographic neural networks are more suitable for prediction problems as they use less memory, easy to use and converges quickly during training.

The latest applications close to hologram technology were those reportedly used by the renowned cable news network (CNN) correspondents, during the 2008 USA presidential elections, (Poniewozik, 2008). Most recently the Wall Street Journal (WSJ) also reported a hologram-like image of deceased American rap artist, Tupac Shakur performing at the Coachella Valley Music and Arts festival on 15th April, 2012 in Indio, California (Smith, 2012). These represent an area that may be of interest in future scientific research, since many still have doubts about the credibility of media reports.

In manufacturing, neural networks have been used to solve a wide range of problems including manufacturing process control, quality control and measurements and many dynamic modeling of otherwise virtually understood problems. Specific examples for applications of neural networks in the manufacturing industry are the part family formation problems, where manufacturing information e.g. operation sequences, lot sizes and multiple process plans are solved. Clustering method for the part-machine grouping problems have also been developed using neural network algorithms based on similarity coefficients. (Rajagopalan and Rajagopalan, 1996, 450.)

Artificial neural networks are currently applied across most branches of medical sciences, especially in computer aided diagnosis and mammography. Most clinical medicine applications of neural networks are classification problems in nature, (Al-Shayea, 2011, 150). Wu and team investigated the potential of using neural networks as one of the decision-making tools in the analysis of mammographic data to distinguish between benign and malignant lesions. Using a three-layer feed forward neural network with a Back propagation algorithm, they trained the network with 43 selected image features extracted from mammograms by experienced radiologists. The study yielded positive results suggesting that neural networks can form a basis for decision making in distinguishing between malignant and benign lesions. Wu et *al.*, (1993, 81.)

In mathematics, neural networks are used to accurately approximate various multi-dimensional linear and non linear functions, which could otherwise consume much computation time. Neural networks are universal aproximators that can easily be trained to map multi-dimensional non linear equations. This is attributed to their parallel architecture. (Ferrari and Stengel, 2005, 24). Multilayer feed forward networks with a non-polynomial activation function can approximate any function (Leshno et *al.*, 1993, 861). Coincidentally, all the computations done under the current master's thesis were performed on a massively parallel connected digital network of computers that trained the neural network and returned the results in about eight minutes instead of a couple of hours or days if it were to be performed on a standalone unit. In the worst cases, a standalone computer runs out of memory and cease to return the neural network model results.

2.4 The working of an artificial neuron

Basically, a neural network is a technique used to map a random input vector into a corresponding random output vector without assuming that there is any persistent relationship between the two sets. A typical neural network has three layers, the input layer, hidden layer and the output layer. The number of neurons corresponds to the size of the input and output layers, while the hidden layer can be manipulated to suit the level of the desired output. The mapping process is achieved by first assigning each individual input with connection weights, which transmit the information to the next neuron or junction. The weights vector is first assigned randomly and subsequently fixed by 'training' the network (More and Deo, 2003, 37). Training aims at achieving an optimal set of weights that minimizes the error, usually by gradient descent learning (Sagar et *al.*, 2011, 334; Zhang et *al.*, 1998a, 38.)

The network iteratively takes the dot product of the vector of connection weights and the input, and sum the total at a summing junction before sending it to the activation function of the network. It is also important to note that each neuron functions independently as the whole network. Several activation functions can be used, depending on the nature of the inputs and the corresponding desired outputs. Karlik and Olgac listed the commonly used activation functions as sigmoid functions (i.e. bi-polar and uni-polar), conic section, hyperbolic tangent and radial bases functions (RBF). It has been found out that hyperbolic tangent function performs better compared to the rest. In their paper, Karlik and Olgac further found out that a combination of hyperbolic tangent functions for both the hidden and output layer produce good recognition results (Karlik and Olgac, 2010, 121.)

2.5 Rationale for choice artificial neural networks in the study

Neural networks in general are able to learn and generalize situations to produce meaningful solutions. It handles all kinds of data be it experimental, empirical or theoretical. They can cope with even situations in which the data is fuzzy and barely understood by humans and are able to adapt the solutions to even the most dynamic circumstances. (Rafiq et *al*., 2001.)

Back propagation trained feed neural networks are most common in science and engineering applications, they process records by 'learning' each at a time and comparing with the output with the actual result. The next layers are fully connected to the preceding layers and thus an error obtained in one successful prediction is used as an input with an objective to minimize it in the next level (gradient descent), hence the name 'Back propagation' algorithm.

Feed forward neural networks specifically, have the following specific benefits that are owed to its wide applications: Since FFNN is data driven; it can learn and map the inputs to the output without making any assumption during model formulation. This makes the model more accurate as wrong models would always lead to inaccuracy. Secondly, they are universal aproximators of most dynamic and non linear models, which are good for situations which are virtually or not understood at all (grey and black box modeling). FFNNs are formed by multiple connections in which information (weights) is transmitted from one node (neurons) to the other. For this reasons some distortions that may occur do not make much difference, which is advantageous for these types of networks. In addition, FFNNs are easy to implement and are flexible in case of need to extend them to other types of networks e.g. recurrent networks i.e. Jordan Elman or cascaded feed forward neural networks, which are formed only by backward looping of some of the outputs and forward casting of some of the inputs respectively. They have also been proved to be better performers in prediction and are faster, therefore cutting down on the computational cost. (Tsai and Lee, 2005, 1656.)

A standard practice for prediction with neural networks involves the use of lagged variables as inputs and lead variables (negative lags), as the expected network output (fig. 6).

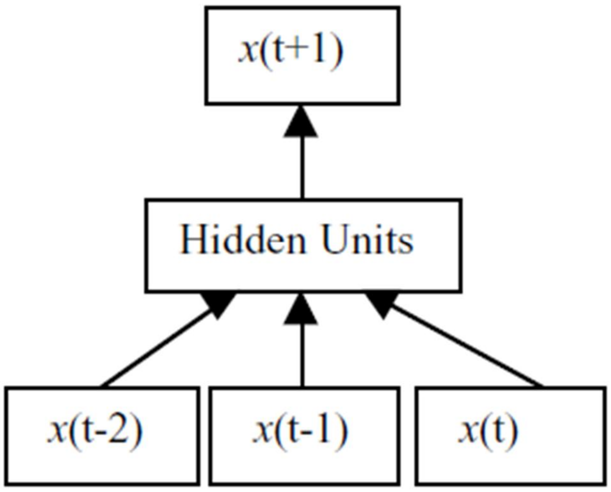


Figure 6: The standard practice for time series forecasting with feed forward ANNs

Where x(t-2) and x(t-1) are the previous two measurements, x(t) is the current input variable and x(t+1) is the predicted output. The difference between predicted and forecast variables should be noted. In modeling, predicted variable usually refers to the output of data used for training while forecasting is the expected results into the future from a predictive model.

2.6 Selection of application software and data processing

For the present master's thesis work, matrix laboratory; MATLAB version 7.14 (MathWorks, R2012a) software was used extensively for all the modeling work. Software selection was based on availability, the mass of data involved and technical considerations e.g. computer memory intensiveness. Other software tried were *Alyuda* family of neural software i.e. forecaster, neurointelligence, neurosignal and forecaster XL. University of Stuttgart Neural Network Simulator (SNNS) was also tried; it is available and seemed well elaborate in the manuals but still presented technical issues with coding. Alyuda group produced good results for the initial phase of pre-processing and excellent results in determining the best network topology especially by iterating with evolved genetic algorithm. However, post-processing did not come out well with these group of software due excess memory consumption and therefore, the present work used MATLAB.

MATLAB became the best option as it is one of the universally accepted platforms for data processing (Gupta, 2010, 1120). It not only act as a programming language but also as a programming environment for C, Java and Visual Basic (VB) with built-in graphical user interfaces (GUI) e.g. the system identification and neural network toolboxes among others, which can be used for quick modeling when few details are of essence.

In neural networks, it is a best practice to pre-process input data before use. Data pre-processing makes the training of the network faster, memory efficient and yield accurate forecast results. Neural networks only work with data usually between a specified range e.g. -1 to 1 or 0 to 1, it makes it necessary then that data is scaled down and normalized. Scaling can be as simple as taking the ratios (reciprocal normalization), computing the differences (range normalization), and multiplicative normalization or Z-axis normalization. Normalization ensures that data is roughly uniformly distributed between the network inputs and the outputs. (Mendelssohn, 1993.)

Post-processing of data involves de-normalizing or reversing the normalization. In some cases both pre-processing and post-processing are built-in to the working environment of the software e.g. *Alyuda* group of neural network software and MATLAB internally carryout these tasks.

2.7 The mathematics of artificial neural networks

As seen before a neural network is composed of a number of elements namely; input layer, hidden layer, output layer, the summing junctions or the threshold gates, and the activation functions. The above elements can be expressed mathematically as the outputs being a function of the inputs supplied to the neural network so that

$$y=f(x1,x2,x3...xp), \qquad (6)$$

Where y is the dependent variable representing the output and x is a vector of p independent variables representing the inputs, while f is the threshold gate. Neural networks used in forecasting typically use lagged variables of the current measurements i.e. wind speeds and/or directions so that

$$y(t+1)= f(xt-1,xt-2,xt-3...xt-p), \qquad (7)$$

Where $y(t+1)$ is a vector of predictions while $x(t-p)$ is the lagged variable of previous wind speeds and/or directions used for the ‘learning’ exercise. (Zhang et *al.*, 1998, 38.)

In the present case therefore, neural networks are used to build forecasting models in which the user has data available but barely understands the internal functioning or dynamics of a system; the atmosphere. This type of modeling is referred to as 'black box modeling'.

The inputs are mapped to the outputs by a set of dynamic parameters called 'weights' that are adjusted iteratively through a 'learning by example' strategy rather than traditional 'programming'. The same elements can also be represented diagrammatically as shown below;

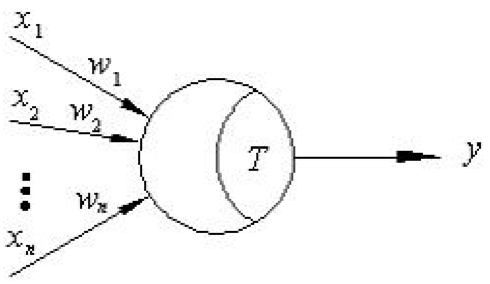


Figure 7: Diagrammatic representation of the inputs, weights, and a linear threshold gate (LTG).

The mathematical representation of the operation carried out by a neural network can be as simple as a linear threshold gate (LTG) operation to quadratic threshold gates (QTG) or even to polynomial thresholds (PTG), see equations 6-8 below. (Hassoun, 1995.)

$$y = \begin{cases} 1 & \text{if} \quad \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} w_{ij} x_i x_j \geq T \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

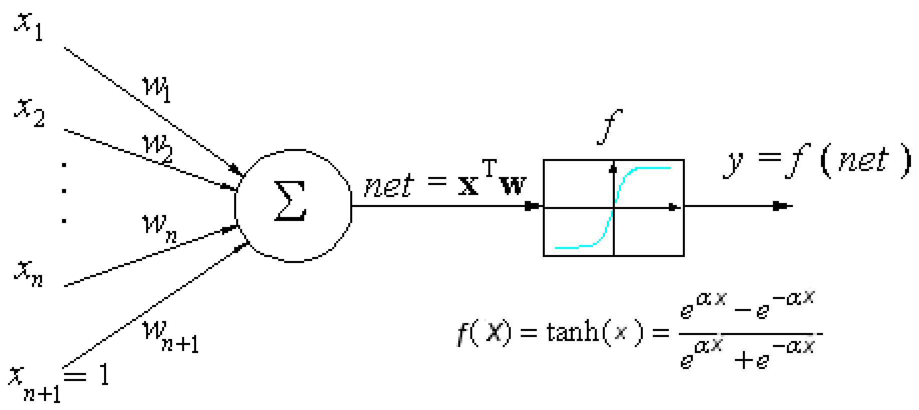
$$y = \begin{cases} 1 & \text{if} \quad \sum_{i_1=1}^{n} w_{i_1} x_{i_1} + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} w_{i_1 i_2} x_{i_1} x_{i_2} + \cdots + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} \cdots \sum_{i_r=i_{r-1}}^{n} w_{i_1 i_2 \ldots i_r} x_{i_1} x_{i_2} \cdots x_{i_r} \geq T \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

So how does the network determine whether the threshold constant, T has been superseded? In the biological motor neuron, the interface between the neuron and the muscular tissue is a synapse referred to as neuro-muscular junction. When the neuron is adequately excited, it releases neuro-transmitters from a nerve terminal by fusion of synaptic vesicle full of transmitter with pre-synaptic plasma membrane, forming a fusion pore through which the transmitter can penetrate the synapses interface to the synaptic cleft and causing post-synaptic receptors to trigger response (Thomson, 2003, 159). This brings about the concept of activation functions. The sum of all activations ($\boldsymbol{wx}$, in eq. 6 above) expresses the idea that summation occurs along the length of the dendrite to produce activation at the cell body (refer to fig. 2), whence the activation is converted into firing, expressed mathematically as;

$$\boldsymbol{y}=\mathrm{f}(wx) \qquad (11)$$

The function f in eq. (9) above is the activation function and can be translated that the firing rate is a function of the post-synaptic activation. Plotting the output firing versus the activation, produces the activation function plots of varying orders. The simplest activation function is a linear function in which the firing rate is directly proportional to the post-synaptic activation. (Rolls and Treves, 1997.)

In the artificial neuron the connection weights represent the synapses of the biological neuron while the threshold function is the last element of a neural network system, which approximates the activity in the soma whence the outputs are received (see fig. 8 below). In general there are three types of activation functions; the linear activation function and piecewise linear activation function, threshold activation function and sigmoid activation functions. Sigmoid family of functions is composed of logistic function, hyperbolic tangent and algebraic sigmoid function. Besides above most common and generally known activation functions, there exists a possibility that there are new activation functions that aims at improving the outputs of neural networks e.g. those referred in (Babel, 1999, 1-3). Fig 8,9,10 and 11 show some of the common activation functions.



x_1
x_2
x_n
$x_{n+1} = 1$
w_1
w_2
w_n
w_{n+1}
Σ
$net = \mathbf{x}^T\mathbf{w}$
f
$y = f(net)$
$f(X) = \tanh(x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}}$

Figure 8: The hyperbolic tangent activation function

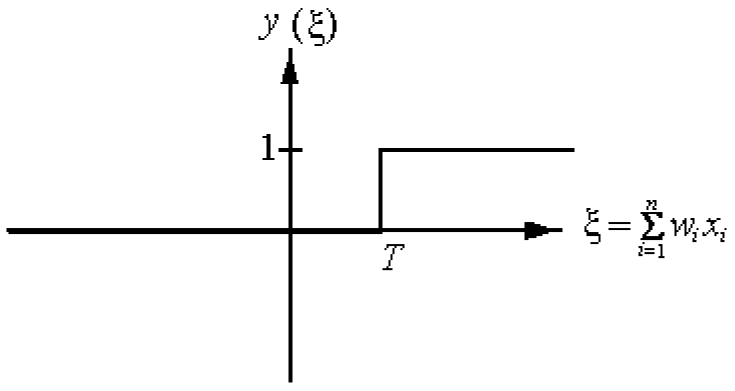


Figure 9: Linear threshold activation function

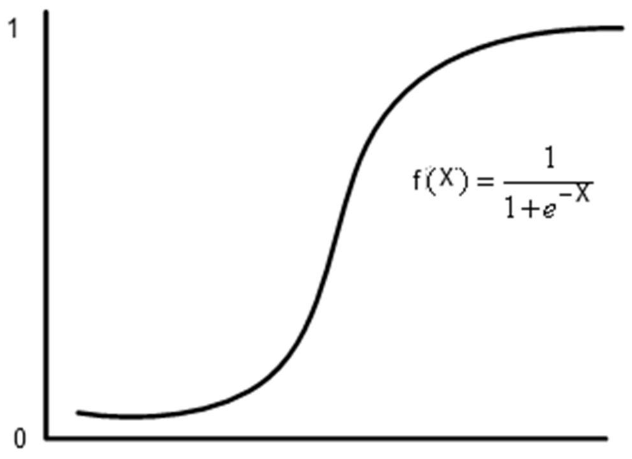


Figure 10: Logistic/sigmoid activation function

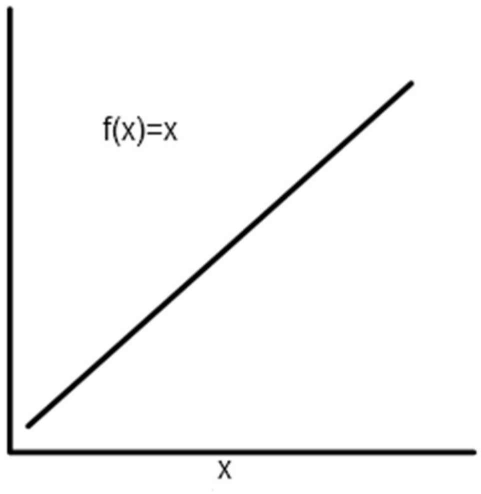


Figure 11: Linear/Identity activation function

3. METHODOLOGY

3.1 An Artificial Neural Network Project Cycle

A successful artificial neural network project (ANN), like project cycles in other disciplines, constitute a number of phases, namely; problem definition and formulation, system design, realization, verification, implementation, and system maintenance phase. The last two phases (system implementation and maintenance) involves embedding the obtained networks in an appropriate working system e.g. hardware or a packaged program that can be installed to run in a computer. This master's thesis is only confined to the first four steps of the project cycle. Fig 12 below shows various stages of an ANN project cycle and the thesis scope.

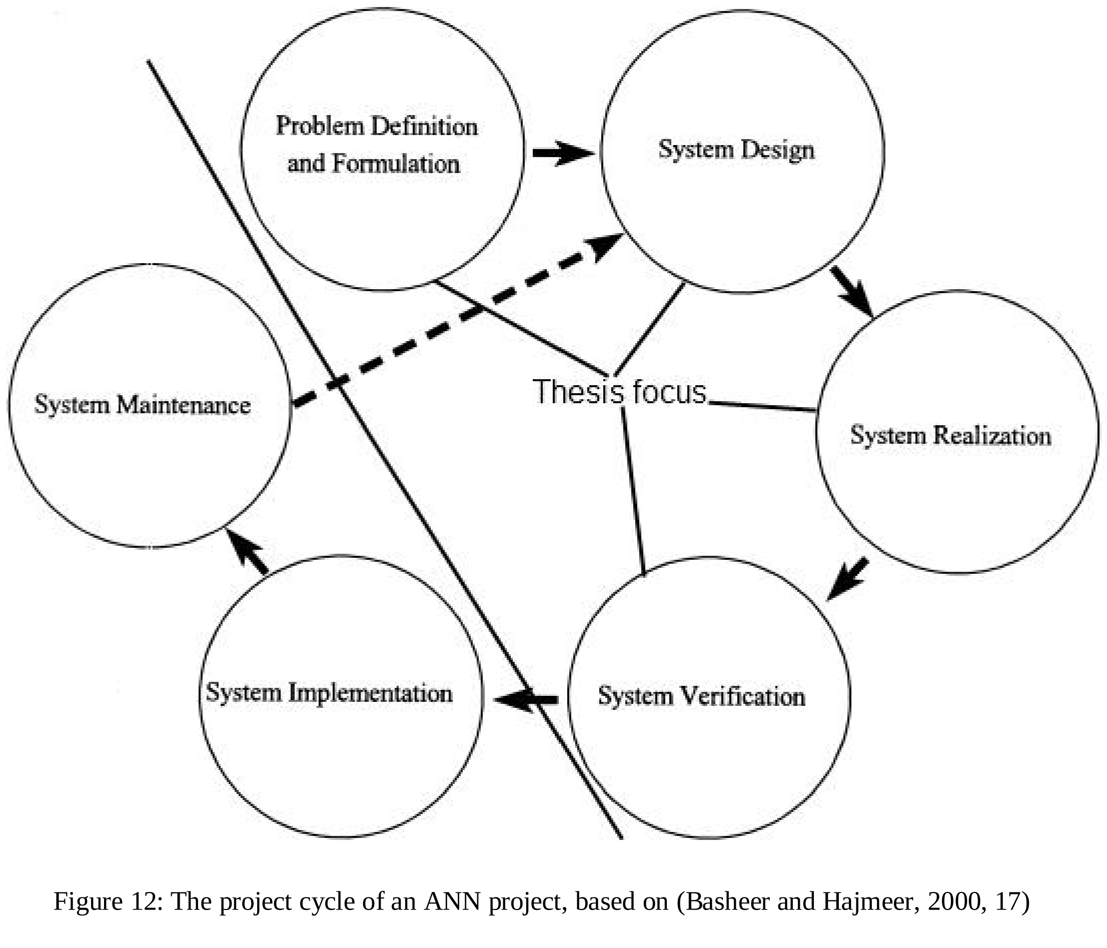


Figure 12: The project cycle of an ANN project, based on (Basheer and Hajmeer, 2000, 17)

3.2 Problem definition and formulation

The overall view of this phase, together with the rationale has been partially covered in the first two chapters. The outstanding part is specific problem definition and formulation which entails explaining the kind of data available and what was required out of it.

The problem involved two non-linear, non-stationery, univariate vectors of wind speeds and directions collected over a period of 2 years (from 1.11.2009 up until 30.10.2011). The data sampling intervals is 10 minutes. It was taken at a height of 60 meters from the ground in the municipal region of Puumala, Finland. The fundamental end results of the project were to construct the three common types of ANNs namely; feed forward, cascade feed forward and Jordan Elman neural networks; and to test the networks by comparing and assessing their mean square error (MSE) and sum squared error (SSE) as the convergence criteria, during training and upon forecasting. Procedurally, the models were used in making a one step ahead hourly forecasts with 10 minute intervals, daily forecasts with hourly averages, weekly forecasts with half-daily averages and monthly forecasts with daily averages and the convergence criteria also measured for this forecasting step and the results presented and discussed.

3.3 System design

System design phase usually starts with data collection, and pre-processing, which can be done within or outside the computation environment. Selection of simulation parameters is the second process before model construction begins. The data used herein was provided by Lappeenranta University of Technology (LUT), and granted the author with permission to use as part of master’s thesis. System design therefore began from data pre-processing i.e. data averaging, subdivision of data into training, validation and testing sets, normalization (scaling) and backward/forward shifting in time into various lagged variables, in a process often referred to ‘sliding window technique’ used as inputs/outputs of the networks.

3.3.1 Data Pre-processing

Wind speed and direction vectors of length (104,043) were periodically averaged into the required time periods. To get hourly data, 6-ten minute measurements were averaged. Similarly to obtain daily means of wind speeds and directions, 24-hourly averages were taken. To accomplish this task, a short MATLAB was written for averaging. (See the code in appendix 1).

Averaging is followed by normalization of the vector. There are a number of ways to normalize data as discussed in the previous chapter, normalization used herein is the reciprocal which scales the data to a range of 0 to 1, before subdividing into three parts; 70% for training, 15% for validation and 15% for system testing as shown in fig. 13 and 14.

Lagged variables (sliding windows) were then created conforming to the desired inputs and outputs; for hourly forecasts, six 10-minute interval outputs were required, for daily forecasts 24 outputs of hourly intervals, weekly interval required 7 outputs of daily averaged values, and monthly interval 30 outputs of daily averages.

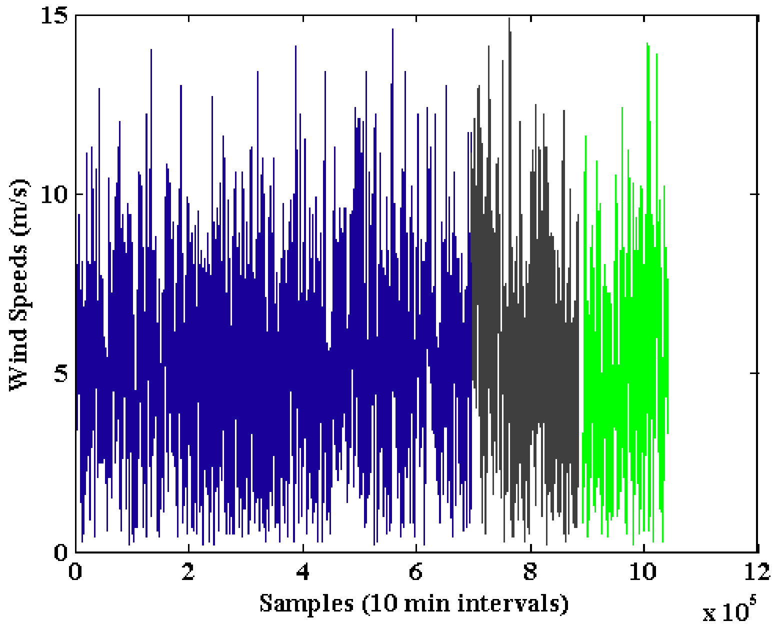


Figure 13: Wind speed data subdivided into training, validation and testing respectively

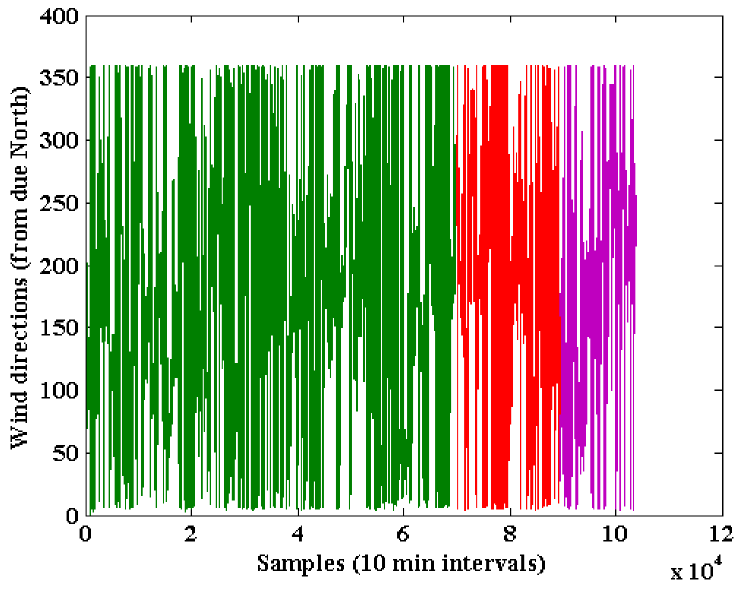


Figure 14: Wind directions data subdivided into training, validation and testing respectively

3.3.2 Model construction

In general three classes of models were constructed; the feed forward neural networks (FFNN), Jordan Elman neural networks (JENN), and Cascaded feed forward neural networks (CFNN), as shown in fig. 15, 16 and 17 below. For each class of models above, lagged variables of wind speeds and directions were separately used as inputs to the networks. Four sub models were then constructed corresponding to the forecast horizons (hourly, daily, weekly and monthly) as described in section 3.2.1 above, making a total of 24 models built.

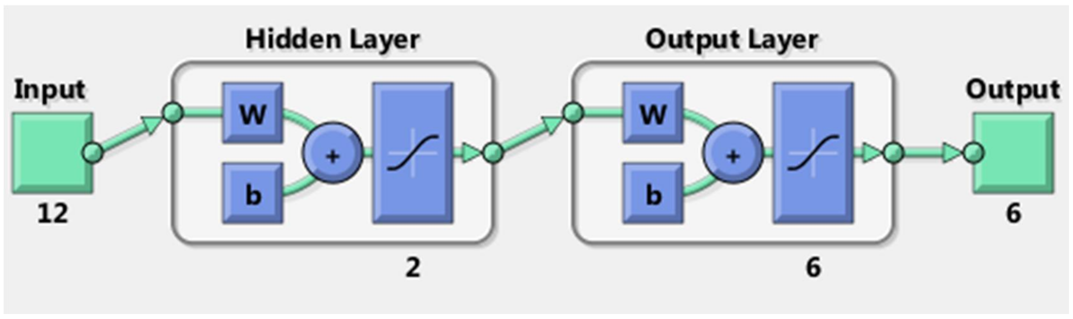


Figure 15: FFNN with 12 inputs, 1 hidden layer with 2 neurons and 6 outputs, six measurements every hour. (Image generated with MATLAB).

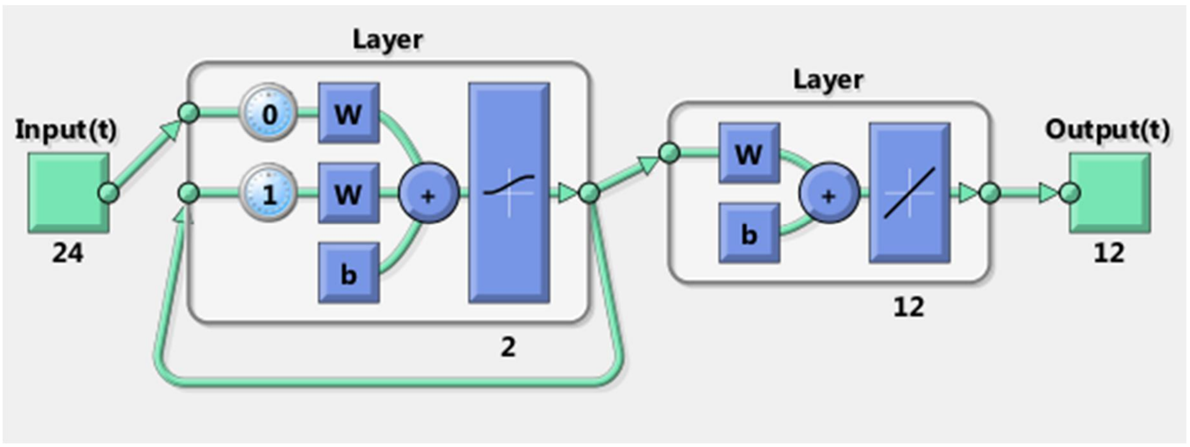


Figure 16: JENN with 24 inputs, 1 hidden layer with 2 neurons, 12 outputs, used for half-day forecasting. (Image generated with MATLAB).

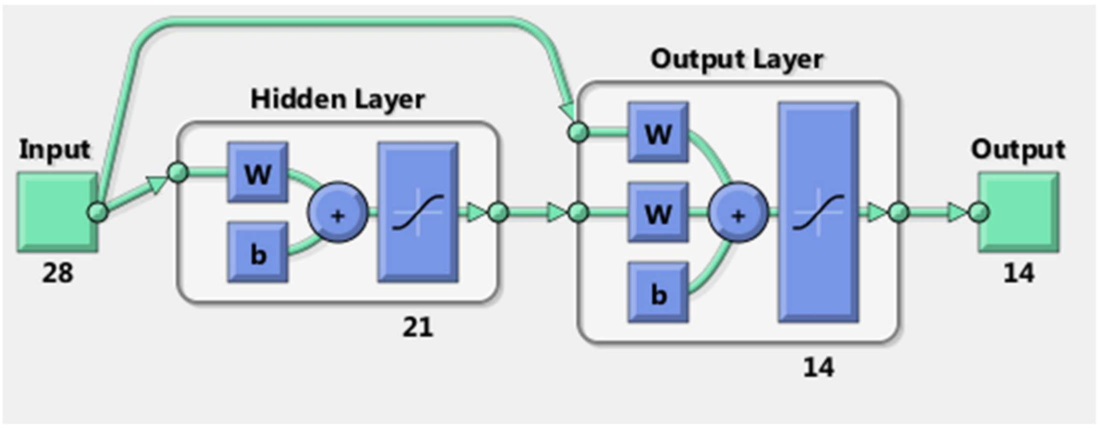


Figure 17: Cascade feed forward neural network with 28 inputs, 1 hidden layer with 21 neurons, 14 outputs, used for weekly forecasting to obtain 2 measurements daily (Image generated with MATLAB).

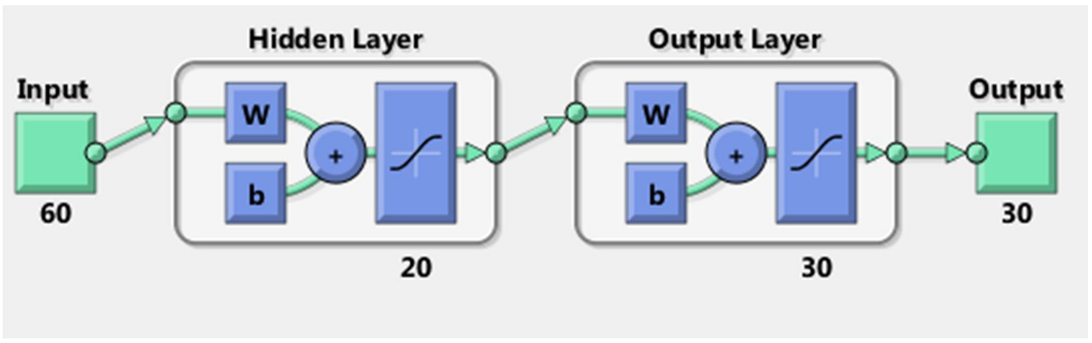


Figure 18: Cascade feed forward neural network with 60 inputs, 1 hidden layer with 20 neurons, 30 outputs, used for monthly forecasting (Image generated with MATLAB).

To make them comparable, authenticable and more realistic, models of the same network topologies were constructed and used for the same forecast horizon, e.g. for hourly forecasting, a model with 12 inputs, 2 hidden neurons and 6 outputs, (denoted as 12:2:6), was used throughout for all model types (JENN, FFNN and CFNN). For daily forecasting: 24:2:12, weekly forecasting: 28:21:14 and monthly forecasts were performed with the largest model with a topology of 60:20:30.

3.4 System realization

The most interesting, challenging and critical phase of the study is to build the models. Tens of parameters are usually controlled during modeling with neural networks. However, not all of them have significant effects on the network's generalization ability. As a result, a number of modeling parameters are selected depending on the forecast horizon, degree of accuracy required, the speed at which the results are needed, among other factors. In most cases, applications used for modeling have inbuilt default settings e.g. MATLAB has readily available codes for quick modeling. In order to achieve a more meaningful model however, the modeler has to diligently select the parameters and optimize them according to some set rules and/or past experience. Noted parameters that influence network results are; the data size partitioning i.e. into training, validation and testing, type of data normalization used, input/output representation, network weight initialization, the learning rate, momentum coefficient, transfer function, convergence criteria, number of training cycles (epochs), hidden layer sizes, the training algorithm etc. For the current study, the following modeling parameters were considered

3.4.1 Input/output representation

Besides data pre-processing, input/output representation determines the effectiveness of the neural network to a large extent. Data representation can be continuous or discrete or a mixture of the two (Basheer and Hajmeer, 2000, 19). This however depends on the nature of the problem in question, e.g. a classification problem setup to distinguish between two classes of objects would use binary numbers i.e. 0 and 1. As stated in section 2.6, MATLAB has internal (inbuilt) normalization function, In this thesis study, the default normalization function was disabled to give room for custom defined normalization and denormalization; continuous, normalized variables between 0 and 1 were used as inputs and outputs representing wind speeds and directions, before denormalization to their original formats.

3.4.2 Transfer function (ζ)

The total sum of the weighted signals is transformed by means of transfer function which determines the ability of the neuron to fire. Several transfer functions e.g. those listed in section 2.7 above, can be implemented on a neural network model. However, there are no rules that exclusively outline the advantages and/or disadvantages of choosing between a certain transfer function or the other (Hassoun, 1995). The choice is best made rather by trial and error, or by following some custom logic defined by the modeler or by own rules specific to the model proved to improve the network's performance. The transfer functions used for this study were arrived at by trial and error methods starting from the presumption that data was scaled to the range of 0 to 1 and thus a sigmoidal transfer functions which possesses the distinctive properties of continuity and differentiability on the range ($-\infty$, $+\infty$) was necessary, an essential requirement of Back propagation learning (Basheer and Hajmeer, 2000, 21). A prior consideration was also given for the fact that a combination of hyperbolic transfer functions for both the hidden and the output layers yielded better recognition results (Karlik and Olgac, 2010, 121).

3.4.3 Size of the hidden layer (*H*)

The number of inputs and outputs automatically determine the number of neurons in the input/output layers. However, the size of the hidden layer, in particular that yield optimal results is difficult to determine since it is a conjugate result of several factors e.g. learning parameter, number of iterations, transfer functions used and the characteristics of the data (Kavzoglu, 1999).

Determining the size of the hidden layer (*H*) is one of the most difficult tasks in neural network modeling. Here, *H* implies the number of hidden layers together with the number of hidden nodes/neurons associated with a neural network. A small *H*, with few inputs seems unable to capture all the underlying information about the data, ending up only producing a linear estimate of the actual required output. On the other hand a larger H increasingly becomes time consuming to train and tend to produce good results during training and validation but large errors in the testing phase. It therefore results in 'overfitting' and loses of generalization ability. (Basheer and Hajmeer, 2000, 24; Nagendra and Khare, 2006, 102.)

Tang and Fishwick suggested that there is a trend existing between the number of hidden units and the network error. Up to a certain point, the error decreases to its minimum then it starts to increase. They conclude that there are therefore an optimal number of hidden units for different series of data. (Tang and Fishwick, 1993, 380.)

So, how do you determine the optimal size of the hidden layer? The current focus of the research in artificial intelligence (AI), particularly artificial neural networks is on determining the optimal size of neural networks, which improves generalization. Different authors have come up with various rules e.g. the ‘rules of the thumb’ for determining the sizes of the hidden layer (Kiranyaz et *al.*, 2009, 1449). In most cases however, researchers may be forced to try modeling with different sizes of the hidden layers, which do not necessarily conform to any particular rule (trial and error). Nagendra and Khare in their study also suggest that the rules failed to yield the ‘optimal’ size of hidden layer, inferring that the best way to obtaining the required hidden layer size is by iteratively adjusting the size while measuring the error during neural network testing (Nagendra and Khare, 2006, 102).

In this study the neural networks should ideally be able to learn and ‘understand’ the fluid statics/dynamics of the atmosphere e.g. the effects of longitudinal and transverse wind velocity gradients, atmospheric temperature and pressure among other factors and assign appropriate weights to accurately forecast the future values. The final sizes of the hidden layer was arrived at by continuously iterating, while measuring the convergence criteria i.e. sum squared error (SSE) and mean squared error (MSE) during evaluation of the network. SSE and MSE were evaluated for one point per ‘sliding window’ and for ‘one step ahead’ forecasts and compared as shown in [Tables 7, 8 and 9].

3.4.4 The training algorithm

Training algorithms are a means of presenting the training set of data to the neural network. Some important aspects about the choice of the training algorithm are computer usage (memory), time and speed. In general, most neural network training algorithms suffer from slow learning speed. In his empirical study devoted to the speed of back-propagation networks, Prof. Fahlman reported that most neural network learning systems use some form of back-propagation algorithm. Although there seems to be no conclusive method for choosing between various training algorithms, the fastest training algorithm which is used in most cases is back-propagation algorithm that runs faster than any of the earlier methods. (Fahlman, 1988, 1-2).

A number of training algorithms are currently used in building various artificial intelligence systems, some of them are; Levenberg-Marquardt (LM), genetic algorithm (GA), incremental and batch Back propagation (IBP, BBP), and quick propagation (QP), which are the most common ones. Different training algorithms are good for various purposes, the predictive ability (which is the subject of the current thesis), has been tested by Ghaffari and team, who concluded that the order of predictive ability of a network trained using above group of training algorithms is IBP, BBP followed by LM, QP and lastly GA. (Ghaffari et *al*., 2006, 136.)

For this master's thesis, back-propagation (BP) algorithm was used for the training exercise. Lavenberg-Marquardt (LM) was also tried but it proved to take too long training time than expected. Both LM and BP training algorithms are implemented in MATLAB and can be invoked by a single command. Many training algorithms (including BP) also suffer from the problem of overfitting, a phenomenon in ANN, caused by overtraining, resulting in memorization of input/output, rather than basing them on the internal factors determined by the weights generated. This causes the network to respond poorly when presented with new data that was not used during training, thus losing the object orientedness, an important aspect of the network, also referred to as generalization.

Regularization and 'early-stopping' are the two most used strategies to take care of overtraining and consequential overfitting. Regularization tries to prevent the network from modeling the noise in the training data by limiting the complexity of the decision boundaries (Laura, 2002, 30). Early stopping involves monitoring the convergence criteria e.g. MSE or the classification error (for classification problems). When a training cycle (epoch) is completed, the training and validation errors are continuously checked; when the error stops reducing or starts increasing, the training is stopped.

Early-stopping requires some prior experience in modeling, especially to monitor the convergence criteria of the neural network. Some latest neural network applications have in-built mechanisms which act as stopping criteria. MATLAB has this in-built mechanism in form of a BP training algorithm, called Bayesian regulation (regularization). For a complete listing of the training algorithms implemented in MATLAB refer to (Si-Moussa, 2008, 187).

Bayesian regulation (BR) Back propagation algorithm was used for all the models in this master's thesis, as it seemed to train successfully, faster and more accurately. Other researchers who have used this framework and come with similar supporting claims are (Lisbôa, 2005, 3-4; Si-Moussa, 2008, 187.)

3.4.5 Network weight initialization

As noted earlier in section 2.4, the initial weight vector is assigned randomly and subsequently fixed by iterative search through the training. This continuous search for appropriate weight vector comes with several drawbacks, especially when using Back propagation training algorithms with gradient descent. One such problem is the aspect of the outputs of an ANN getting trapped in a high error level (or local minimum) and secondly, slowing down of the training process (Lee et al., 1999, 1738). These two aspects constitute a phenomenon referred to as 'premature saturation' of hidden neurons in an ANN.

According to many researchers (Lee et al., 1999, 1737; Lee et al., 1993, 719; Basheer and Hajmeer, 2000, 19), premature saturation (PS) is one among the important factors affecting error convergence of a neural network. It is caused by improper assignment of the initial weights, with respect to the network parameters (Lee et al., 1993, 719; Goerick and Seleen, 1996). Bi and team attribute it to the update disharmony between weights connected to the input and output layers (Bi et al., 2005, 3645).

Several main techniques are currently used to get rid of PS. Nguyen and Widrow had suggested that initializing adaptive weights over a large number of training problems achieved major improvements in learning efficiency (Nguyen and Widrow, 1990, 26). Moallem and Ayoughi proposed three methods; increasing the number of hidden neurons, Weigend weight regularization and renewing saturated terms by adding anti-saturating terms (Moallem and Ayoughi, 2010, 127). Network weight initialization involves assigning predetermined optimum initial values for the weights to all existing connection links that help the network to converge faster.

For the current thesis, Nguyen and Widrow weight initialization algorithm was used. In this algorithm, weight bias initialization values are picked between the intervals located randomly in the predetermined region i.e. -1 and 1. Nguyen and Widrow suggested that, if H is the number of units in the first layer, *Wbi=0.7H*. *Wi* are chosen between -1 and 1 and the weights, *w* are assigned so that *w=-Wi/Wbi*, simply put as the uniform random values between -1 and 1 and is implemented in MATLAB as a script, initnw.m. (Pavelka and Procházka, 456, 2004.)

3.4.6 Learning rate (η)

Learning rate is basically the quantifier for the acceleration at which the weight vector changes. A small learning rate means that from one epoch to the next, the step change in the weight vector is small and vise versa. A high learning rate is detrimental to the network as it poses a risk of overshooting while a slow learning rate takes too much time for the network to converge. The learning rate can be constant throughout, as was done in this thesis or can be made adaptive i.e. to vary with time, η(t). In the case of adaptive parameter, it can be made high in the beginning of the training or rather when the search is far away from the minimum; and smaller as the search reaches minimum. This parameter rate can be anything between 0 and 10. In all the networks created for this master's thesis, the learning rates ranging from 0.01 to 3 gave satisfactory results. (Basheer and Hajmeer, 2000, 20.)

3.4.7 Momentum coefficient (μ)

Back propagation momentum coefficient is a term applied in weight updating, meaning the speed at which the weight vector changes. It helps the network not to get trapped in local minima (as described in section 3.3.5 above). A high μ is likely to reduce the risk of getting trapped in the local minima. However, it runs the risk of overshooting just as a high learning rate does. This value, just like the learning rate, can be made adaptive, i.e. μ(t). It is set relatively high when the search is far away from the solution and lower as the search approaches the true minimum, depending on the error gradient (Fahlman, 1988). Literary, momentum coefficient is analogous to velocity of a physically moving object, while the learning rate is analogous to its acceleration. Many authors have different suggestions on how to fix the momentum coefficient (Basheer and Hajmeer, 2000, 20). For this project, the momentum coefficient between 0.0 and 1.0, as suggested by (Hassoun, 1995), produced satisfactory results.

3.4.8 Number of training cycles (Epochs)

An epoch is defined as a single presentation of each input/output data on the training set (Fahlman, 1988, 5). Epochs are set as one of the training parameters and are important in gauging the training time taken by a neural network to reach convergence and also to set the goal that determines the extent to which the network should be trained. For this study, the training epochs were set by trial and error, with a range of 100 to 1000. At most 1000 epochs for all the models built produced satisfactory results. The use of Bayesian regulation training algorithm also was used as a tool for setting the stopping time, making epoch setting just a supporting criterion.

3.5 System verification

The ultimate goal of a neural network modeling project is to obtain workable results e.g. if the aim of the project was to predict the future behaviour of a phenomenon, the end result would be able to show how close to reality the modeler attained. Although the neural network model includes testing against the data while training is in progress i.e. using the allocated percentages of data for (training, testing, validation sets), it is a good practice that the final models be examined for its generalization ability. This stage is meant to measure the capability of the ANN model to respond to completely ‘strange’ data, not used during training. This is what is referred here as system verification. This is the stage that this thesis is focused on, as it clearly distinguishes the variation in the original data to the predicted one. It was made part of the modeling stage by supplying the model with the range of original data set, from the 70-85th percentile and comparing the model output to the last 85th to 100 percentile of the same data, (see the code in appendix 2). The convergence criteria were then measured by determining the two statistical properties, i.e. the MSE and the SSE between the forecast and the target results, which were compared and reported for each model built. In the next section the quantitative results of the study were presented graphically and by tabulation.

3.6 Models performance measurement

Neural networks are all about finding the weights that minimize the error between some input and the expected output in a modeling process. A number of error measurement techniques can be applied to assess the relationship between the two sets of data, e.g. the mean square error (MSE), the sum square error (SSE), the root mean square error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), coefficient of correlation square (R^2), or simply taking the coefficient of correlation (R) between the two sets of data. Each of these error measurements has its own advantages and disadvantages, depending on the goals and objectives of the study.

In this study the men square errors and the sum square errors were taken into considerations. MSE has been used for over five decades now, especially in signal processing as a fidelity measure. This is because it is simple to perform, takes less computing memory and provides a clear physical meaning and has been a norm to use for a better understanding of everyone. (Wang and Bovik, 2009, 99-100.)

The use of MSE particularly in neural networks has in the recent past gained popularity as a means of performance measurement, and has been applied widely, (Ferrari and Stengel, 2005; Aladag et *al*., 2009; Al-Shayea, 2011, 150). The mean square error is the average of all the squares of individual errors between the model and the real measurements, and is given by eq. 11:

$$MSE(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{y}_i)^2 \tag{11}$$

where N is the number of samples, x_i and y_i are measured and predicted values respectively.

The sum square error (SSE) is the total summation of the individual squares of errors without averaging, and it gives an indication of the total magnitude of the error between the models and the measured results. SSE is given by eq. 12:

$$SSE(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{N}(\boldsymbol{x}_i - \boldsymbol{y}_i)^2 \tag{12}$$

In addition, MSE and SSE are useful in making comparisons between several models with same sets of data and same observations, N. In the event that more than one models are compared, one important indicator obtained is how better a model is, compared to the others. As seen, both MSE and SSE are dependent on the number of observations and so the quantity (order) of error is only significant, relative to those of other models under the same study. Note that MSE and SSE cannot be run directly, in versions of MATLAB earlier than R2010b.

4. RESULTS

4.1 Models assessment for wind Speeds

4.1.1 Models assessment for long term wind speed forecasting

Table 1 shows the results of the models based on both MSE and SSE upon training and on simulation with totally new inputs, which were not used during training. Here we compare a column on the model outputs, to the corresponding column on the measured data. This is referred to as 1-point per sliding window. A 1- point per sliding window extends for the entire column length. Therefore in this way, the generalization ability of the models in the long-term was assessed.

Table 1: The results of the models, assessing the generalization ability when used for long term forecasting of wind speeds (Hourly & Daily)

	HOURLY FORECASTS				**DAILY FORECASTS**			
MODEL	**MSEt**	**SSEt**	**MSEv**	**SSEv**	**MSEt**	**SSEt**	**MSEv**	**SSEv**
JENN	0.3801	27772	4.2E6	6.6E10	0.667	8106	8.689	22695
CFNN	0.3638	26582	1.7E7	2.7E11	0.667	8106	8.452	22076
FFNN	0.3807	27817	3767	5.9E7	0.688	8364	7.899	20632

Table 2: The results of the models, assessing the generalization ability when used for long term forecasting of wind speeds (Weekly & Monthly forecasts)

	WEEKLY FORECASTS				**MONTHLY FORECASTS**			
MODEL	**MSEt**	**SSEt**	**MSEv**	**SSEv**	**MSEt**	**SSEt**	**MSEv**	**SSEv**
JENN	2.104	2047.3	4.9982	1079.6	1.798	751.4	4.122	445.22
CFNN	2.56	2490.9	6.8416	1477.8	2.041	853.2	9.306	1005
FFNN	2.0931	2067	6.3329	1368	1.628	680.6	4.18	451.3

4.1.2 Models assessment for short term wind speed forecasting

The short term usability of the models was assessed by measuring the relative error between the model output rows and the measured data, referred to as a sliding window. A sliding window is simply one set of inputs and outputs to a neural network model, e.g. for hourly forecasting with 10-minute interval data, a row of six model outputs are compared to the corresponding row in the real measured data matrix.

Table 3: The results of the models, assessing their generalization ability when used for short term forecasting of wind speeds

	HOURLY FORECASTS		**DAILY FORECASTS**		**WEEKLY FORECASTS**		**MONTHLY FORECASTS**	
MODEL	**MSEv**	**SSEv**	**MSEv**	**SSEv**	**MSEv**	**SSEv**	**MSEv**	**SSEv**
JENN	16.4211	98.266	0.3430	4.1156	2.8959	40.5432	3.1452	94.3548
CFNN	17.3025	103.8152	0.3492	4.1899	1.8416	25.7827	9.1991	275.9728
FFNN	16.4183	98.51	0.5430	6.5154	1.6656	23.3138	2.1257	63.7702

4.2 Models assessment for wind directions

4.2.1 Result for wind directions forecasting on a 1-point per sliding window

Table 4: The results of the models, assessing their generalization ability when used for long term forecasting of wind directions (Hourly & Daily forecasts)

	HOURLY FORECASTS				**DAILY FORECASTS**			
MODEL	**MSEt**	**SSEt**	**MSEv**	**SSEv**	**MSEt**	**SSEt**	**MSEv**	**SSEv**
JENN	6.4E3	4.7E8	8.8E6	1.3E11	4.2E3	5E7	1.3E4	3.4E7
CFNN	1.3E4	9.6E8	1.3E9	2.1E13	4.1E3	5E7	1.3E4	3.5E7
FFNN	1.6E4	1.1E9	1.7E4	2.7E8	3.3E3	4E7	1.3E4	3.5E7

Table 5: The results of the models, assessing their generalization ability when used for long term forecasting of wind directions (Weekly & Monthly forecasts)

	WEEKLY FORECASTS				**MONTHLY FORECASTS**			
MODEL	**MSEt**	**SSEt**	**MSEv**	**SSEv**	**MSEt**	**SSEt**	**MSEv**	**SSEv**
JENN	5.4E3	5.3E6	1.4E4	3.0E6	9.9E3	4.1E6	1.5E4	1.7E6
CFNN	6.6E3	6.4E6	1.3E4	2.8E6	6.2E3	2.6E6	1.2E4	1.2E6
FFNN	6.3E3	6.1E6	1.6E4	3.4E6	1.1E4	4.6E6	1.3E4	1.4E6

4.2.2 Result for wind direction forecasting on a sample sliding window

Table 6: The results of the models, assessing their generalization ability when used for short term forecasting of wind directions

	HOURLY FORECASTS		**DAILY FORECASTS**		**WEEKLY FORECASTS**		**MONTHLY FORECASTS**	
MODEL	**MSEv**	**SSEv**	**MSEv**	**SSEv**	**MSEv**	**SSEv**	**MSEv**	**SSEv**
JENN	9.6E3	5.8E4	4E3	4.7E4	2.8E4	4.0E5	1.5E4	4.4E5
CFNN	1.0E4	6.0E4	3.6E3	4.3E4	6.1E3	8.6E4	1.2E4	3.6E5
FFNN	3.0E4	1.8E5	3.7E3	4.4E4	2.5E4	3.4E5	1.8E4	5.3E5

Tables Chart

JENN:	Jordan Elman Neural Networks
CFNN:	Cascade Forward Neural Networks
FFNN	Feed Forward Neural Networks
MSEt:	Mean Square Error upon training
SSEt:	Sum Square Error upon training
MSEv:	Mean Square Error upon verification
SSEv:	Sum Square Error upon verification

4.3 Developing the criteria for choosing between different forecasting models

The core needs determines the criteria applied by the modeler in choosing between various types of models. A number of criteria used in this study to assist in making that choice are identified as the degree of accuracy needed, the forecast horizon for which the model is designed, and whether the model is usable for long-term or short-term forecasting. In this case, long-term forecasting can be hourly forecasts for a relatively long period of time e.g. several months ahead.

With the kind of results presented in section 4.2 therefore, one can tell which model type has the lowest statistical error compared to other models, during training and upon verification i.e. with new inputs. It is also possible to tell which model is best suited for which forecast horizon, and which one is good enough for long/short term forecasting for both wind speeds and directions. Table 5 and 6 summarizes the obtained results, specifically answering the above important questions regarding the models.

4.3.1 Choice of wind speeds forecasting models based on generalization ability

Table 7: Making a choice between the models for use in forecasting wind speeds

	Generalization Error (MSE and SSE)								
MODEL	**Hourly**		**Daily**		**Weekly**		**Monthly**		**Score**
	Long-term	**Short-term**	**Long-term**	**Short-term**	**Long-term**	**Short-term**	**Long-term**	**Short-term**	
JENN				✓	✓		✓		**3**
CFNN									**0**
FFNN	✓	✓	✓			✓		✓	**5**

▨✓ **Chosen model**

4.3.1 Choice of wind directions forecasting models based on generalization ability

Table 8: Making a choice between the models for use in forecasting wind directions

	Generalization Error (MSE and SSE)								**Score**
MODEL	**Hourly**		**Daily**		**Weekly**		**Monthly**		
	Long-term	**Short-term**	**Long-term**	**Short-term**	**Long-term**	**Short-term**	**Long-term**	**Short-term**	
JENN		✓	✓						**2**
CFNN				✓	✓	✓	✓	✓	**5**
FFNN	✓								**1**

▨✓ **Chosen model**

4.4 Sample forecast results from selected networks

4.4.1 Hourly forecasting of wind speeds with FFNN

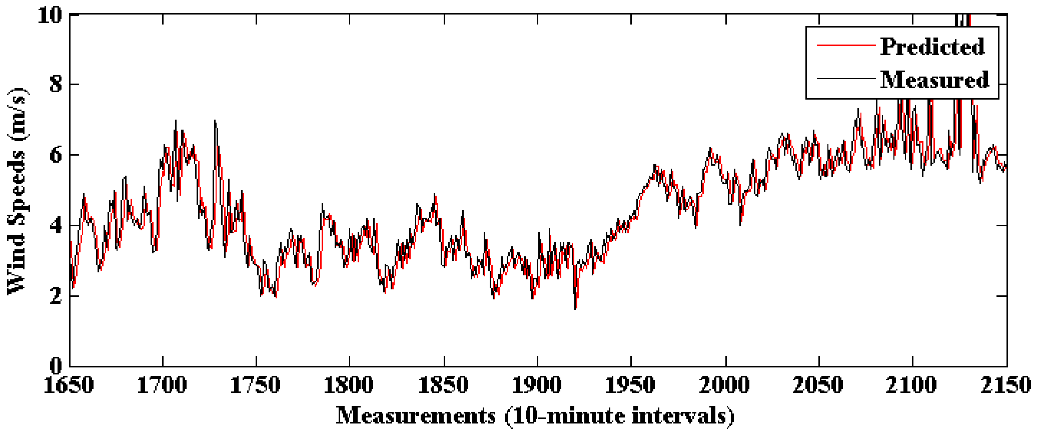


Figure 19: Comparing model outputs and the measured hourly wind speeds upon training

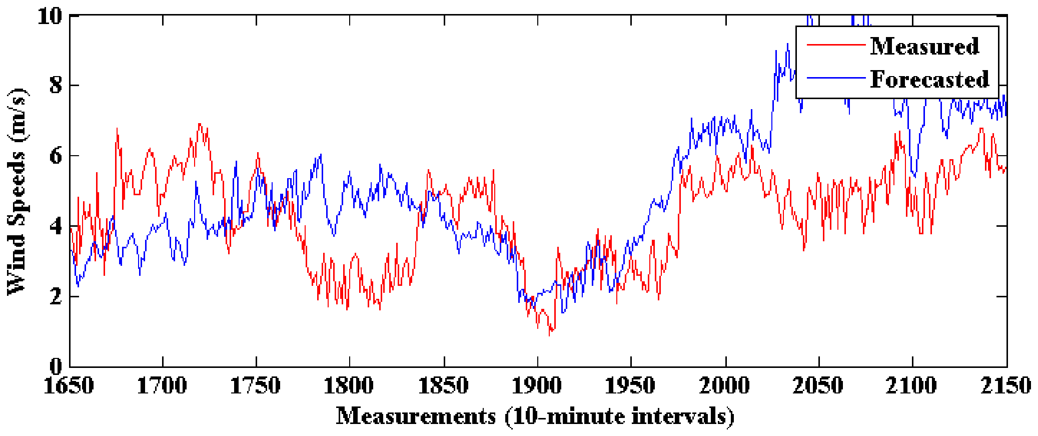


Figure 20: Comparing model outputs and the measured hourly wind speeds upon verification

NB: The measured data on training is a part of the first 70% and on verification is part of the last 15% of the original data; therefore they are not the same set of data.

4.4.2 Weekly forecasting of wind directions with CFNN

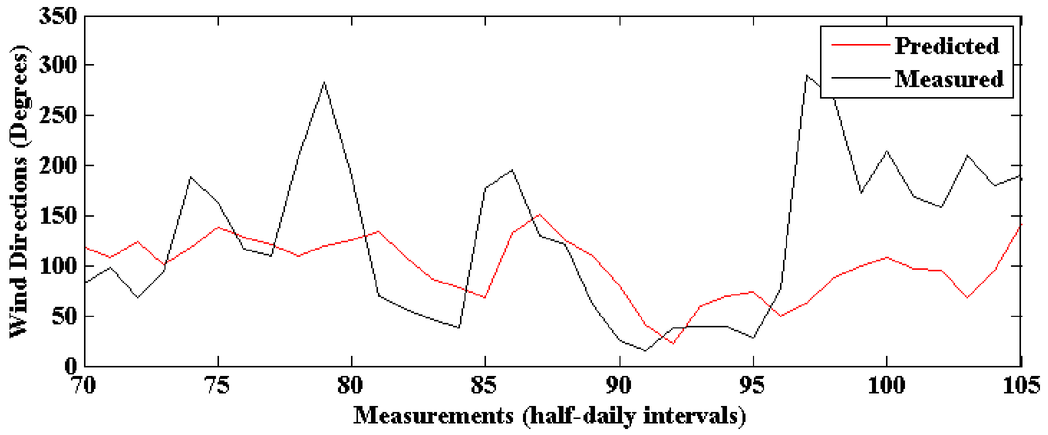


Figure 21: Comparing model outputs and the measured weekly wind directions upon training

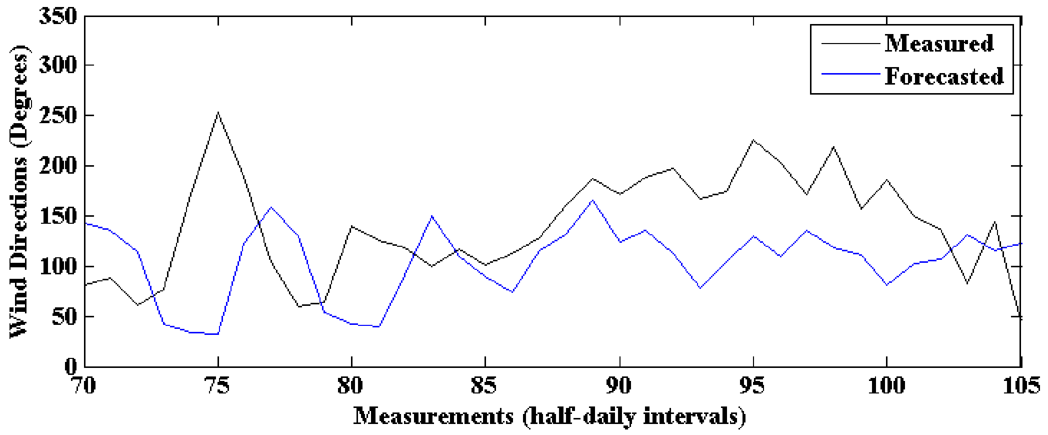


Figure 22: Comparing model outputs and the measured weekly wind directions upon verification

NB: The measured data on training is a part of the first 70% and on verification is part of the last 15% of the original data; therefore they are not the same set of data.

4.4.3 Monthly forecasting of wind directions with CFNN

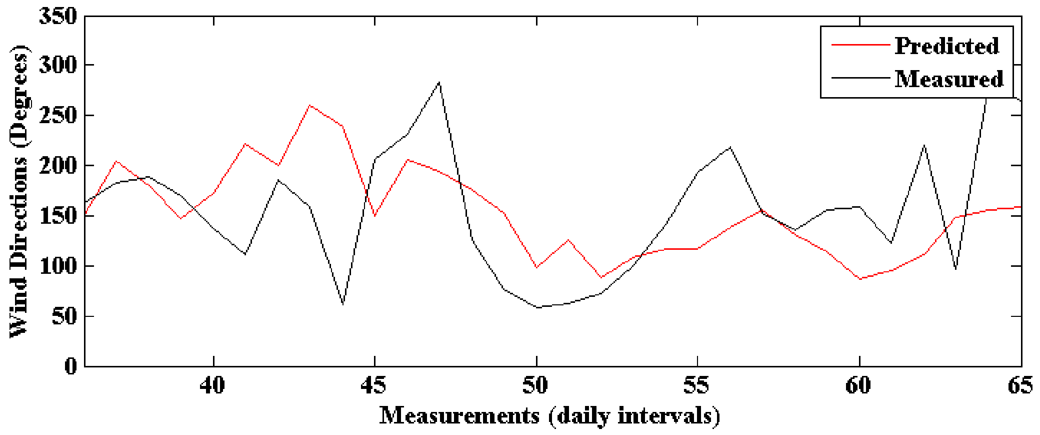


Figure 23: Comparing model outputs and the measured monthly wind directions upon training

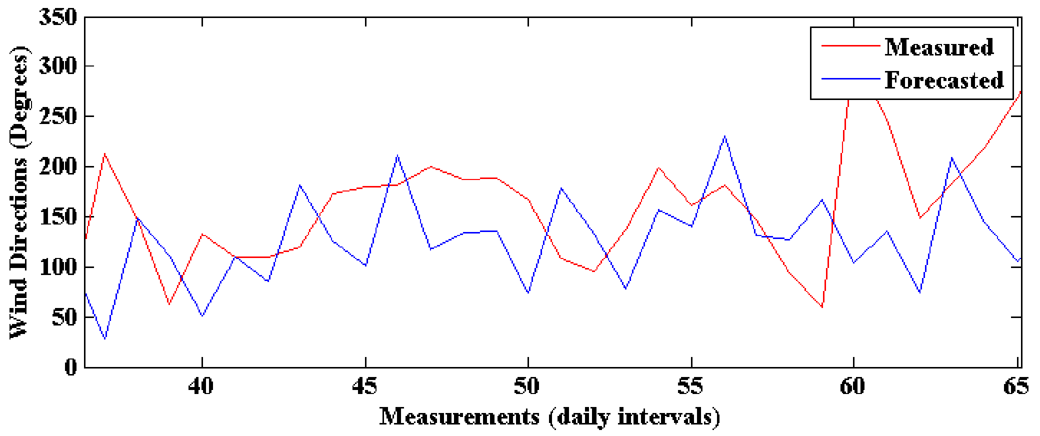


Figure 24: Comparing model outputs and the measured monthly wind directions upon forecasting

NB: The measured data on training is a part of the first 70% and on verification is part of the last 15% of the original data; therefore they are not the same set of data.

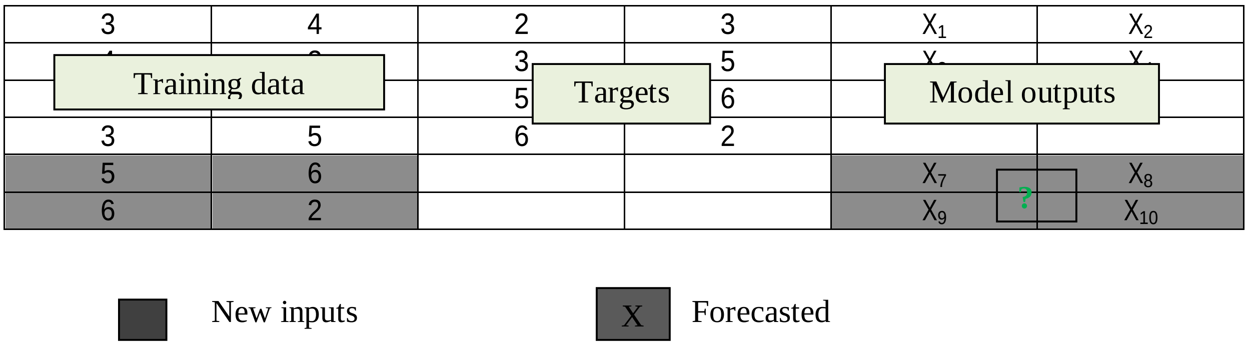
5. RESULTS INTERPRETATION AND DISCUSSION

The results were obtained by taking 70% of the data, further divided into a second set of 70, 15 and 15%, and used for training, validation and testing respectively. The last 30% of the data was used for verification, i.e. by presenting the model with the second last 15% which was not used for training and assessing how the output from the models compares with the last 15% of the original data, as explained in section 3.5. The results from each of the models were organized and assessed in terms of the magnitude of the statistical error between the measured result and the real data. This was achieved by measuring the average of the squares of errors (MSE) and the total sum of the squared errors (SSE), for each model used for forecasting of both wind speeds and directions. The procedure was repeated for the two stages of data analysis, during training and upon verification, for **one-point per sliding window** and for a sample of **whole sliding window** and the overall error magnitude, as shown in [tables 1-6]. The sliding window concept is explained;

When data is converted into lagged variables, they form sliding windows of different sizes depending on the required inputs and outputs of the model. The example below illustrates the sliding window concept, as used in neural networks. A univariate vector e.g. A, below can be converted into lagged variables of the same vector and therefore six sliding windows are created, corresponding to the number of rows. Table 1 shows a skeleton summary of sliding window used for neural network forecasting modeling.

A= [3 4 2 3 5 6 2 3 4];

Table 9: A skeleton summary of sliding windows used for neural network forecasting model


Training data
Targets
Model outputs
?

3	4	2	3	X_1	X_2
4	2	3	5	[illegible]	[illegible]
[illegible]	[illegible]	5	6	[illegible]	[illegible]
3	5	6	2		
5	6			X_7	X_8
6	2			X_9	X_{10}

▩	New inputs	X	Forecasted

The sliding windows formed are used to control the number of inputs/outputs of the networks, e.g. we can form a model with 2 inputs and 2 outputs. This will ‘instruct’ the neural network for instance, that at a particular time when the two previous wind speeds were 3 and 4, for one reason or another, the next two became 2 and 3, from first row in [table 9]. The network then tries to ‘explain’ and generalize the underlying factors for that, and applies the rule to new inputs not used in training it. Therefore the first two columns [table 9], forms the inputs and the third and fourth forms the targets while the fifth and sixth represent the outputs from the model. Lag matrix code is included in appendix. For further references, see appendix 4.

To conduct ‘mass’ forecasting, the new inputs to the network must be in the form of the training inputs (same column size). In the same way, the outputs from the model have the same column size as the target matrix/vector. The success of the models was realized by measuring the relationship (MSE & SSE) between the expected versus the model outputs.

Plotting and comparing the rows cutting across the model output matrix to those of the target matrix is what was referred to, as sample whole sliding window. This measures the generalization ability of the model on a short term basis, also commonly referred to as one-step-ahead forecasting, e.g. that presented in [tables 3, 6]. On the other hand comparing a column on the target matrix versus a corresponding column on the model output matrix is the one point per sliding window forecasts. This measures the generalization ability of the model with increasing forecast horizon [Tables 1, 2, 4, 5].

In general, for each of the three types of models (Feed Forward, Jordan Elman and Cascade forward): 4 similar models (topologically) were built, corresponding to four forecast horizons: hourly, daily, weekly and monthly forecasting, of both wind speeds and directions. As an overall observation, the mean square error and the sum square errors, which were used as the convergence criteria were relatively lower during training, but shot up steadily upon simulation with new inputs.

The wind speed forecasting models seemed to produce relatively more consistent results compared to those of wind directions. This can partly be attributed to the nature of data e.g. distribution, probable existence of anomalies in the original data, and in the data collection process. The wind directions were relatively uniformly distributed, with a range between 0 and 360^o measured from due north, while wind speeds ranged from 0 to 14.9 m/s, with a seemingly normal (Gaussian) distribution, with 5 m/s as a persistent speed. (See appendix 3).

Even though normalization would have reduced the range of the two sets of data, there is still a larger range between direction measurements, compared to those of speeds, even after normalization. It can be seen therefore, it is more difficult for the neural networks to train the sets of data with a large range in between, compared to training one with relatively small range. As a result, none of the models built can vividly be said to possess the ability to forecast wind directions, and thus opening up an opportunity for further research in this context. Nevertheless, FFNNs returned the lowest generalization errors for hourly, weekly and monthly forecasts; while JENNs returned the lowest errors when used for forecasting of daily wind speeds. CFFNs gave the lowest errors when used for forecasting daily, weekly and monthly wind directions; while JENNs proved to be the best when used for hourly forecasting of wind directions [tables 3, 6]. In addition, a combination of hyperbolic tangent transfer functions for both hidden and output layer returned better results for most of the models that were used for forecasting in this study.

All data were normalized to a range between 0 and 1; a logistic transfer function would have been expected to have a better performance on the data. On the contrary however, from the tests, a combination of hyperbolic tangent transfer functions for both hidden and output layer returned a relatively low error for most of the models. (See Fig. 15-18).

6. SUMMARY AND CONCLUSION

Even though wind power has been in use for several centuries now, its application for a long time was limited to mechanical processes e.g. pumping of fluids and grinding of grains. However, during the last few decades there has been a rapid development in the field especially focusing on electricity generation from wind power. This paradigm shift has partly been fuelled by the need for cheaper, renewable and also environmentally friendly sources of energies.

Like many new concepts, wind power electricity generation requires incremental studies to enhance its applicability and better understand its future dynamism, challenges as well as the opportunities. It has become a necessity in the recent past, that future wind speeds and directions are estimate or rather forecasted. This allows for planning and scheduling purposes, optimal wind farms sitting, approximation of total future energies to be generated from new wind projects, electricity loads forecasting once the projects are operational and many more benefits.

Artificial neural networks forms an important modern component of statistical modeling of real world problems, wind speeds and directions included. Computers have now evolved to an extent that they are able to learn from experience and experimental data without assuming pre-existing, persistent relationship between the training data and the expected outputs. Methods now exist, whereby computers are exposed to a situation by providing past data of inputs and outputs. They then generate internal rules and/or functions not necessarily visible to the modeler (‘black box’ modeling), generalize and apply when queried with new inputs, not used during the training process. They work much in the same way as linear regression but are much more powerful to an extent that they can handle highly dynamic and non-linear processes. Neural networks have been applied widely in science, engineering, and finance, mathematics, as well as artistic works, such as audiovisual entertainment industry.

Various types of artificial neural networks used for different purposes, have been developed over time and at the moment they have been implemented in a number of mathematical computer programs which makes them easy to construct and use, in a customized manner. In this study three types of neural network models namely: Feed forward, Jordan Elman and Cascade forward neural networks, were built in MATLAB computer software, and used for training and forecasting of wind speeds and directions. The training data was real data taken at a height of 60 meters from the ground in Puumala municipal region, in Finland. It was provided through courtesy of Lappeenranta University of Technology (LUT), for use in this master's thesis project.

Quantitatively, based on the models' generalization ability, considering long-term and short-term forecasting, and by using both MSE and SSE as the convergence criteria, the feed forward neural networks (FFNN) emerged as the most preferred type of models that may be used both for short-term and long-term wind speed forecasting, amongst other types of models tested. FFNN returned the lowest generalization error for 5 out of the 8 models built for wind speeds forecasting. On the other hand, cascaded feed forward neural networks (CFNN) proved to be a better choice among the rest when applied for wind direction forecasting. CFNN returned the lowest generalization error in 5 out of the 8 models built for wind directions forecasting.

Qualitatively, hourly forecasting of wind speeds with FFNNs consistently returned the lowest generalization error both in the short term and in the long run. This adds up to the conclusions made by various researchers in the past. However, for wind directions CFNNs, which has less often been used compared to FFNN, returned the lowest generalization error when used both for weekly and monthly forecasting of wind directions. On a per-forecast-horizon basis, FFNNs returned the lowest generalization errors for hourly, weekly and monthly forecasts; while JENNs returned the lowest errors when used for forecasting of daily wind speeds. CFFNs gave the lowest errors when used for forecasting daily, weekly and monthly wind directions; while JENNs proved to be the best when used for hourly forecasting of wind directions. In addition, a combination of hyperbolic tangent transfer functions for both hidden and output layer returned better results for most of the models that were used for forecasting in this study.

To obtain good results with neural networks, data quantity is as important as data quality. A large amount of data is needed for training of the models. For this study two years data, seemed to limit the possibility of the models to adapt well and to develop accurate rules for generalization. It is possible that the relatively low quality results from both wind speed and direction models were as a result of the limitation of the data quantity.

As an overall observation, even though neural networks may be used to forecast natural phenomena e.g. wind speeds and directions, their ‘intelligence’ is limited to a relatively progressive change in the unique factors/rules developed and used by the networks during training. For instance, the training data of wind speeds and directions collected over a period of say 5 years can only be used for forecasting as long as the human, physical and environmental factors e.g. surrounding forests, buildings, terrain, etc., remain as is, or with minimal and gradual changes. This limits the applicability of the neural network in many areas, with ecological modeling included, and thus further research is called for in this area of study.

7. REFERENCES

Ackermann, T & Söder, L. 2002. An overview of wind energy-status 2002. *Renewable and Sustainable Energy Reviews*, vol. 6, pp. 67-128.

Aladag, C.H., Egrioglu, E. & Kadilar, C. 2009. Forecasting nonlinear time series with a hybrid methodology. *Applied Mathematics Letters*, Vol. 22, pp. 1467-1470.

Alfares, H. K. & Nazeeruddin, M. 2002. Electric load forecasting: literature survey and classification of methods. *International Journal of Systems Science*, Vol. 33 (1), pp. 23-34.

Al-Shayea, Q. K. 2011. Artificial Neural Networks in Medical Diagnosis. *International Journal of Computer Science Issues*, Vol. 8 (2), pp. 150-154.

Babel, J. & Kotillová, A. 2009. New types of activation functions for RBF neural networks.

Basheer, I.A. & Hajmeer, M. 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, Vol. 43, pp. 3-31.

Bi, W., Wang, X., Tang, Z. & Tamura, H. 2005. Avoiding the Local Minima Problem in Back propagation Algorithm with Modified Error Function. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*. Vol. E88-A (12), 3642-3653.

Boyle, G. 1996. Renewable Energy: Power for Sustainable Future. Oxford: Oxford University Press.

Fadlalla, A. & Lin, C. 2001. An analysis of the applications of neural networks in finance. *Interfaces*, Vol. 31 (4), pp. 112–122.

Fahlman, S. E. 1988. An empirical study of learning speed in back-propagation networks. Technical report, CMU-CS-88- 162. Pittsburgh: Carnegie Mellon University.

Ferrari, S. & Stengel, R. 2005. Smooth Function Approximation Using Neural Networks. *IEEE Transactions on Neural Networks*, Vol. 16, pp. 24-38.

Focken, U., Lange, M. & Waldl, H. P. 2001. Previento - A Wind Power Prediction System with an Innovative Up-scaling Algorithm. European Wind Energy Conference (EWEC), July 2 - 6, Copenhagen, Denmark.

Ghaffari, A. Abdollahi, H., Khoshayand, M. R. & Bozchalooi, I. S. 2006. Performance comparison of neural network training algorithms in modeling of bimodal drug delivery. *International Journal of Pharmaceutics*, Vol. 327, pp. 126-138.

Goerick, C. & Von Seleen, W. 1996. On Unlearn-able problems or a model for premature saturation in Back propagation learning. Proceedings of the European Symposium on Artificial Neural Networks (ESANN), April 24 – 26, Bruges, Belgium.

Grogg, K. 2005. Harvesting the Wind: The physics of wind turbines [online document]. [Accessed 31.7.2012]. Available at https://dspace.lasrworks.org/handle/10349/145

Gupta, R., Bera, J.N. & Mitra M. 2010. Development of an embedded system and MATLAB-based GUI for online acquisition and analysis of ECG signal. *Measurement*, Vol. 43, pp. 1119-1126.

Hagan, M.T, Demuth, B.H & Beale M.H. 1996. *Neural Network Design*. Boston: PWS Publishing Company.

Hassoun, M. H. 1995. *Fundamentals of Artificial Neural Networks*. Cambridge: The MIT press

Houtzager. I. 2011. Towards data-driven control for modern wind turbines. Doctoral dissertation. Delft: Dutch institute of systems and control. Delft centre for systems and control.

Johnson, G. 2001. *Wind Energy Systems*. Harlow: Prentice-Hall

Karlik, B. & Olgac, A. 2010. Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. *International Journal of Artificial Intelligence and Expert Systems*, Vol. 1 (4), pp. 111-122.

Kavzoglu, T. 1999. Determining Optimum Structure for Artificial Neural Networks. 25th Annual Technical Conference and Exhibition of the Remote Sensing Society (RSS), September 8 - September 10, Cardiff, UK.

Kim, Y.K., Seo, J. W. & Jeong, H. E. 2007. Sensitivity experiments for winds prediction with planetary boundary layer parameterization. Jcomm Scientific and Technical Symposium on Storm Surges, October 2 - 6, Seoul, Korea.

Kiranyaz, S., Ince, T., Yildirim, A. & Gabbouj, M. 2009. Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. *Neural networks,* Vol. 22, pp. 1448-1462.

Kriesel, D. 2005. A brief Introduction to Neural Networks [online document]. [Accessed 10.4.2012]. Available at http://www.dkriesel.com/en/science/neural_networks

Kröse, B. & Smagt, P. 1993. *An Introduction to Artificial Neural Networks*. 8th Ed. Amsterdam. University of Amsterdam.

Lange. M. 2003. Analysis of the uncertainty of wind power predictions. Doctoral dissertation. University of Oldenburg: Department of Mathematics and Natural Sciences.

Laura. M. 2002. Signal Processing Methods for Non-Invasive Respiration Monitoring. Doctoral dissertation. University of Oxford: Department of Biomedical Signal Processing & e-health.

Lawrence, S., Giles, C.L. & Fong, S. 2000. Natural Language Grammatical Inference with Recurrent Neural Networks. *IEEE Transactions on Knowledge and Data Engineering,* Vol. 12, pp. 126–140.

Lee, H.-M.,Huang, T.-C. & Chen, C.-M. 1999. Learning Efficiency Improvement of Back propagation Algorithm by Error Saturation Prevention Method. *International Joint Conference on Neural Networks*. 41 (3), pp. 1737-1742.

Lee, Y., Oh, S.H., & Kim, M. 1993. An analysis of premature saturation in Back propagation learning. *Neural networks,* Vol. 6, pp. 719-728.

Lek, S. & Guégan, J.F. 1999. Artificial neural networks as a tool in ecological modelling, an introduction. *Ecological Modelling*. Vol. 120, pp. 65-73.

Leshno, M., Lin, V., Pinkus M. & Schocken, S. 1993. Multilayer feed forward networks with a non-polynomial activation function can approximate any function. *Neural Networks,* Vol. 6, pp. 861-867.

Lisbôa, A., Pimentel, W. & Martignoni W. 2005. Modeling fluid catalytic cracking with multivariate Statistics and artificial neural networks. 4th Mercosur Congress on Process Systems Engineering, August 14-18, Rio de Janeiro, Brazil.

MathWorks, 2012, version 7.14.0.739 (R2012a), The MathWorks Inc., Natick, Massachusetts, USA.

Mendelssohn, L. 1993. Preprocessing Data for Neural Networks [online document]. [Accessed 20.5.2012]. Available at http://vp.tradertech.com/lbm_library/neural_networks/

Moallem, P. & Ayoughi, S. A. 2010. A Complementary Method for Preventing Hidden Neurons' Saturation in Feed Forward Neural Networks Training. *Iranian Journal of Electrical and Computer Engineering*, Vol. 9 (2), pp. 127-133.

More, A. & Deo, M.C. 2003. Forecasting wind with neural networks. *Marine Structures*, Vol. 16, pp. 35-49.

Nagendra, S. M & Khare, M. 2006. Artificial neural network approach for modelling nitrogen dioxide dispersion from vehicular exhaust emissions. *Ecological Modelling*, Vol. 190, pp. 99-115.

Nguyen, D. & Widrow, B. 1990. Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights. Proceedings of International Joint Conference on Neural Networks (IJCNN), June 14 - 17, San Diego, California.

Panteri, E. & Papathanassiou, S. 2008. Evaluation of two simple wind power forecasting models. European Wind Energy Conference, March 31 – April 3, Brussels, Belgium.

Pavelka, A. & Procházka, A. 2004. Algorithms for Initialization of Neural Network Weights. In: Sborník príspevku 12. rocníku konference MATLAB, Vol. 2, pp. 453-459.

Poniewozik, J. 2008. Election Night: Whiteboards Out, Holograms In [online document]. [Accessed 8.5.2012]. Available at www.time.com/time/magazine/article/0,9171l

Proceedings of the 17th Annual Conference in Technical Computing, November 19, Prague, Czech Republic.

Rafiq, M., Bugmann, M. & Easterbrook. 2001. Neural network design for engineering applications. *Computers and structures*, Vol. 79, pp. 1541-1552.

Rajagopalan, R. & Rajagopalan, P. 1996. Applications of neural networks in manufacturing. 29th Hawaii International Conference on System Sciences (HICSS) Vol. 2: Decision Support and Knowledge-Based Systems, January 3 - 6, Hawaii, USA.

Rolls, E. & Treves A. 1997. *Neural Networks and Brain Function*. Oxford: Oxford University Press.

Sagar, G., Chalam, S. & Singh, M. 2011. Evolutionary Algorithm for Optimal Connection Weights in Artificial Neural Networks. *International Journal of Engineering*, Vol. 5 (5), pp. 333-340.

Si-Moussa, C., Hanini, S., Derriche, R., Bouhedda, M. & Bouzidi, A. 2008. Prediction of High-Pressure Vapour Liquid Equilibrium of Six Binary Systems, Carbon Dioxide With Six Esters, Using an Artificial Neural Network Model. Brazilian Journal of Chemical Engineering, Vol. 25 (1), pp. 183-199.

Smith, E. 2012. Rapper's De-Light: Tupac 'Hologram' May Go on Tour [online document]. [Accessed 26.6.2012]. Available at http://online.wsj.com/article/SB10001424052702304818

Tang, Z. & Fishwick, P.A. (1993). Feed-forward neural nets as models for time series forecasting. *ORSA Journal on computing*. vol. 5 (4), 374-385.

Thomson, A. 2003. Presynaptic Frequency- and Pattern-Dependent Filtering. *Journal of Computational Neuroscience*, Vol. 15, pp. 159-202.

Tsai, T.-H. & Lee, C.-K. 2005. Design of Dynamic Neural Networks to Forecast Short-term Railway Passenger Demand. *Journal of the Eastern Asia Society for Transportation Studies*, vol. 6, pp. 1651-1666.

Wang, Z. & Bovik, A.C. 2009. Mean square error: Love it or leave it? *Signal Processing Magazine*, January, 99-117.

Wu, Y., Giger, M.L. & Doi, K. 1993. Artificial Neural Networks in Mammography: Applications to decision making in the diagnosis of breast cancer, *Radiology*, Vol. 187, pp. 81-87.

Zhang, G., Patuwo, B. E. & Hu, M.Y. 1998. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, Vol. 14, pp. 35-62.

APPENDICES

Appendix 1: MATLAB averaging code

```
J=1;

for i=1:measurements:length(data);

        data_av(j)=mean(data(i:i+measurements-1));

        J=J+1;

end;
```

where *J* is a temporary counter, *measurement* is the number of measurements in one period e.g. 6 in one hour for measurements taken every ten minutes, *data* is the vector containing speed/direction variables and *data_av* is the averaged vector.

Appendix 2: MATLAB code used to create the networks

```
% This MATLAB code was used to pre-process data, create the networks and
% forecast
clc
clear all
close all

load puumala_data.mat

spd=data(:,1); % wind speeds taken every ten minutes

spd(isnan(spd)) = 0;

period=6; % periodically averaging them into hourly measurements
%
j=1;
for i=1:period:length(spd);
    spd_av(j)=mean(spd(i:i+period-1));
    j=j+1;
end


%% Turn off default normalization

nn.inputs{1}.processFcns={};
nn.outputs{1}.processFcns={};
% custom normalization
```

```
spd_av=1./spd_av;

%% 70% of the data is used for network training leaving out 30% for testing

tra1=spd_av(1:round(0.7*length(spd_av)));

%% creating a lagged matrix of the current and two previous to forecasted

n1=23;
n2=12;

n=[n1:-1:0];
m=[1:n2];

Lagtra1=lagmatrix(tra1,n); % lag matrix of X(t), X(t-1),X(t-2),X(t-3, etc.
Leadtra1=lagmatrix(tra1,-m); % lead matrix of X(t+1) and X(t+2)....X(t+6)
Lagtra1(isnan(Lagtra1)) = 0;% replacing NaNs with zero
Leadtra1(isnan(Leadtra1)) = 0;
% Lagtra2=lagmatrix(tra2,n);
% neural network construction

P=Lagtra1(n1+1:end-n2,:);%[Lagtra2(2:19,:)];%(5:228,:);
T=Leadtra1(n1+1:end-n2,:);%(5:228); % targets
net=newelm(P',T',[2],{'tansig','tansig'},'trainbr');

net.trainParam.goal=0; % error goal
net.trainParam.epochs=500; % maximum iterations
net.trainParam.show=25; % showing intervals
net.trainParam.max_fail=50;
net.trainParam.lr=0.01;
net.trainParam.mc=0.9;
%% Network initialization
net.initFcn='initlay';
net.layers{1}.initFcn='initnw';
%net.layers{2}.initFcn='initnw';
%net.layers{3}.initFcn='initnw';
net=init(net);% initialize the net (weights and biases initialized)

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

%% % training the net

[net,tr]=train(net,P',T');
view(net)
% simulating the network with the training inputs
y=sim(net,P');

y=1./y;

T=1./T';
plot(y(1,:),'-r'); hold on
plot(T(1,:),'-k');
```

```
perf_train_inputs=mse(net,T(1,:),y(1,:))

%% simulating the network with new inputs

n3=round((length(spd_av)-length(tra1))/2-1);
n4=2*n3;

a=spd_av(length(tra1)+1:length(tra1)+n3);


c=1./spd_av(length(tra1)+n3+1:length(tra1)+n4);


aLag=lagmatrix(a,n);% lagging in to the correct format as training inputs
aLag(isnan(aLag)) = 0;

cLag=lagmatrix(c,-m);


p=aLag;
q=cLag';
c_sim=sim(net,p'); % simulated a

c_sim=1./c_sim;% denormalization

%% comparison of real (a) with simulated outputs, one point/sliding window

plot(q(1,:),'-r'); hold on % real a, spd X(t+1) 2580-2605
% plot(error,'-k'); hold on
plot(c_sim(1,:),'-b'); % sim a, spd X(t+1) 2580-2605

perf_newinputs=mse(net,q(1,:),c_sim(1,:))

% plot(z_real(:,1),'--k'); hold on % real z, spd X(t)
% plot(z_sim(1,:),'--p');  % simulated z, spd X(t)

% end
display 'sample predicted versus real data'

[c_sim(:,252),q(:,252)]
%% performance on the prediction horizon
mse_wholewindow_forecast=mse(net,c_sim(:,252),q(:,252))
sse_wholewindow_forecast=sse(net,c_sim(:,252),q(:,252))
% end
```

Notes:

Alter *period* to change averaging period
n1, *n2*; to change the inputs/outputs
n3, *n4*; to change the training and verification sets

Appendix 3: Histograms showing wind speeds and directions distribution

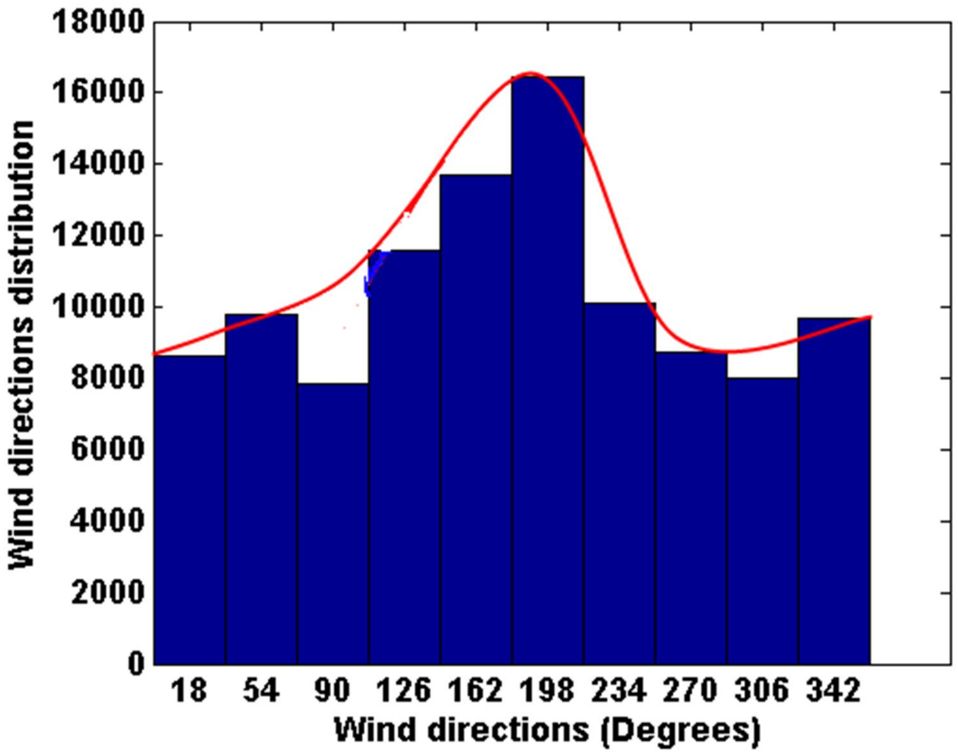


Figure 25: A histogram of wind directions distribution

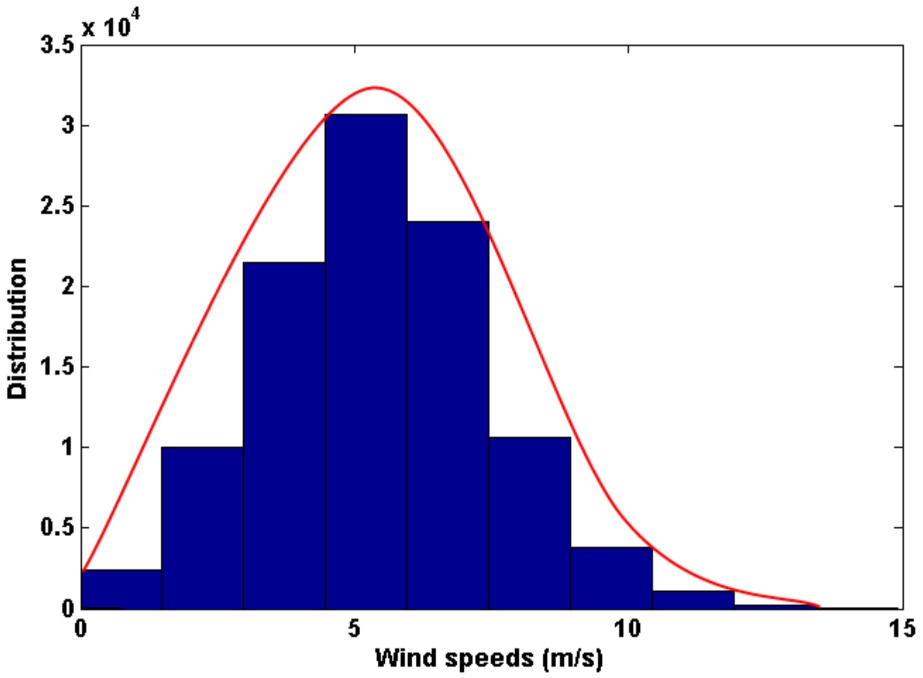


Figure 26: A histogram of wind speeds distribution

Appendix 4: MATLAB code for converting matrices in to lag variables

```
function xLag = lagmatrix(x , lags)
%LAGMATRIX Create a lagged time series matrix.
%   Create a lagged (i.e., shifted) version of a time series matrix. Positive
%   lags correspond to delays, while negative lags correspond to leads. This
%   function is useful for creating a regression matrix of explanatory
%   variables for fitting the conditional mean of a return series.
%
%   XLAG = lagmatrix(X , Lags)
%
% Inputs:
%   X - Time series of explanatory data. X may be a vector or a matrix. As
%     a vector (row or column), X represents a univariate time series whose
%     first element contains the oldest observation and whose last element
%     contains the most recent observation. As a matrix, X represents a
%     multivariate time series whose rows correspond to time indices in which
%     the first row contains the oldest observations and the last row contains
%     the most recent observations. For a matrix X, observations across any
%     given row are assumed to occur at the same time for all columns, and each
%     column is an individual time series.
%
%   Lags - Vector of integer lags applied to each time series in X. All lags are
%     applied to each time series in X, one lag at a time. To include a time
%     series as is, include a zero lag. Positive lags correspond to delays, and
%     shift a series back in time; negative lags correspond to leads, and shift
%     a series forward in time. Lags must be a vector integers.
%
% Output:
%   XLAG - Lagged transform of the time series X. Each time series in X is
%     shifted by each lag in Lags, one lag at a time for each successive time
%     series. Since XLAG is intended to represent an explanatory regression
%     matrix, XLAG is returned in column-order format, such that each column
%     is an individual time series. XLAG will have the same number of rows as
%     observations in X, but with column dimension equal to the product of the
%     number of time series in X and the number of lags applied to each time
%     series. Missing values, indicated by 'NaN' (Not-a-Number), are used to
%     pad undefined observations of XLAG.
%
% Example:
%   The following example creates a simple bi-variate time series matrix X with
%   5 observations each, then creates a lagged matrix XLAG composed of X and
%   the first 2 lags of X. XLAG will be a 5-by-6 matrix.
%
%      X = [1 -1; 2 -2 ;3 -3 ;4 -4 ;5 -5]  % Create a simple bi-variate series.
%   XLAG = lagmatrix(X , [0 1 2])          % Create lagged matrix.
%
% See also NaN, ISNAN, FILTER.
% Don’t forget to reference MathWorks, see how to do it in my bibliography.

if nargin ~= 2
    error('Error: Inputs "X" and "Lags" are both required.');
end

%
% If time series X is a vector (row or column), then assume
% it's a univariate series and ensure a column vector.
%

if prod(size(x)) == length(x)            % check for a vector.
   x  =  x(:);
end

%
% Ensure LAGS is a vector of integers.
%

if prod(size(lags)) ~= length(lags)       % check for a non-vector.
   error(' "Lags" must be a vector.');
end

lags  =  lags(:);                         % Ensure a column vector.

if any(round(lags) - lags)
   error(' All elements of "Lags" must be integers.')
end

missingValue  =  0;  % Assign default missing value. modified Aug/7/2002

%
% Cycle through the LAGS vector and shift the input time series. Positive
% lags are delays, and can be processed directly by FILTER. Negative lags
% are leads, and are first flipped (reflected in time), run through FILTER,
% then flipped again. Zero lags are simply copied.
%

nLags =  length(lags);  % # of lags to apply to each time series.

[nSamples , nTimeSeries] = size(x);

xLag  =  zeros(nSamples , nTimeSeries * nLags);

for c = 1:nLags

    columns  =  (nTimeSeries*(c - 1) + 1):c*nTimeSeries;   % Columns to fill

    if lags(c) > 0     % Time delays.
```

```
        xLag(:,columns) = filter([zeros(1,lags(c)) 1] , 1 , x , ...
            missingValue(ones(1,lags(c))));

    elseif lags(c) < 0 % Time leads.

        xLag(:,columns) = flipud(filter([zeros(1,abs(lags(c))) 1] , 1 ,...
            flipud(x) , missingValue(ones(1,abs(lags(c))))));

    else                % No shifts.

        xLag(:,columns) = x;

    end

end
```