# Power Output Prediction For Wind Farms With Machine Learning

Nkosinathi Hlophe (nkosinathi@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Dr. Vukosi Marivate
CSIR, South Africa
Co-Supervised by: Dr. Bubacarr Bah
AIMS, South Africa

24 May 2018
*Submitted in partial fulfillment of a structured masters degree at AIMS South Africa*

# Abstract

Wind power generated by wind farms depends on meteorological conditions such as wind speed, temperature, wind direction and many more. Wind power generated is proportional to the cube of the wind speed up to a certain level that is why wind speed prediction is important. Support vector regression and Long Short term Memory (LSTM) are two models that were developed to accurately predict wind speed at 3, 6, 12 and 24 hours ahead. The data used was transformed using z-score standardization and all the computation are based on the transformed data. In this work we show the performance of the two models comparing them to a persistence model which is used as a benchmarking model. The LSTM on overage performs better than the other two models with a root mean square error of about 0.6 for 24 hour forecasting.

*keywords* – wind speed forecasting, Long Short Term Memory, Support Vector Regression.

## Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Nkosinathi Hlophe, 24 May 2018

# Contents

**Abstract** i

**1 Introduction** 1
- 1.1 Wind energy . . . 1
- 1.2 Existing wind speed prediction models . . . 2
- 1.3 Challenges in wind energy production . . . 2
- 1.4 Organization of the project . . . 2

**2 Related work** 3
- 2.1 Statistical methods . . . 3
- 2.2 Machine learning . . . 3
- 2.3 Optimization algorithms . . . 11

**3 Prediction methods** 15
- 3.1 Problem formulation . . . 15
- 3.2 Methods . . . 16
- 3.3 Performance measures . . . 17

**4 Results and discussion** 19
- 4.1 Data collection . . . 19
- 4.2 Data preprocessing . . . 19
- 4.3 Results . . . 20
- 4.4 Discussion . . . 26

**5 Conclusion** 29

**References** 33

# 1. Introduction

The use of fossil fuel in energy generation is course for concern due to the waste products and the availability of the resources. The demand for electrical energy keeps on increasing meaning the supply has to increase. Wind energy is a free-fuel energy source that produces energy without too much pollutants. Wind energy offers an alternative to energy production that will reduce the over-dependence on fossils fuels for energy. To harvest wind energy and be able to use it, we need good wind speed predicting methods, since wind speed plays a key role in wind power due to the relationship between wind energy from turbines and wind speed (Botha and van der Walt, 2017).

## 1.1 Wind energy

Most textbooks define energy as the ability to do work and it appears in different forms. One of the various energy forms is electrical energy which has become an integral part of human life, providing around $18000$ tera watt-hour of energy per year which is around $40\%$ of humans' total energy use. The production of this high energy come at a cost as tonnes of carbon dioxide is produced (Schiermeier et al., 2008).

Renewable Energy Sources (RES), is seen as an alternative to producing clean energy as opposed to the burning of fossil fuels. Renewable energy, is defined as naturally occurring clean sources of energy which minimize environmental impacts e.g. hydro-power, solar, wind, geothermal, biomass and marine energies (Panwar et al., 2011).

Current electrical energy production methods, like the burning of fossil fuels have big consequences on the environment and have negative effects on human health (Panwar et al., 2011). The over-dependence on fossil fuels for energy production is a concern, since the reserves of the resources are getting smaller, and we are facing danger of running out of them (Wang et al., 2015).

**1.1.1 Future of wind energy.** Countries are trying to reduce the cost of energy production and environmental impacts by changing their energy supplies to RES (Zhu and Genton, 2012). Countries including USA, China, Germany have set energy renewable Portfolio standard and they are leading in the installation of wind power plants. The power out of wind turbine is proportional to the cube of wind speed (Pinson, 2006) and also depends on the characteristics of the wind turbine. South Africa is also among the countries who planning are to increase wind energy production to $20\%$ of the national demand by $2030$ (van der Walt and Botha, 2016).

Wind energy is the fastest growing energy resource around the world. Approximately 10 million mega-watts-hour of wind energy is being installed every year and by $2020$, $12\%$ of the world's power will be coming from wind energy (Wang et al., 2015). RES is expected to provide $15\%$ to $25\%$ of the global electricity by $2050$. The worlds total energy wind power capacity doubles every year since $2010$. It was $197$ Giga Watts (GW) in $2010$ and $369$ GW in $2014$ (Ghaderi et al., 2017). In South Africa and in other parts of the world, the price of wind energy production is becoming more comparable to the fossil fuel power generated (van der Walt and Botha, 2016).

The nature of wind makes it difficult to achieve balance in the power needed to feed into the grid. The increase in the wind power production comes with many problems that need to be fixed, such as balancing the power produced and consumed, interconnection standards and power system stability and reliability (Wang et al., 2015). Wind power generated is hard to predict and store due the high variability of wind speed. Wind power should should be used with other power generation methods to

ensure continuity and minimize the variability (Panwar et al., 2011).

## 1.2 Existing wind speed prediction models

In this paper we focus on wind speed forecasting using Machine Learning (DL) techniques. Support Vector Regression (SVR) has been used in short term forecasting ($1$ to $24$ hours) with an RBF (Gaussian) kernel function (Botha and van der Walt, 2017). Their results shows and improved forecast error up to 11.12 $\%$ by using feature selection on meteorological data from Alexander Bay, South Africa between 2011–2013. It has been shown by other studies that SVR outperforms other regression models even without incorporating feature selection (van der Walt and Botha, 2016).

Among different DL algorithms, Recurrent Neural Network (RNN) has been commonly used in forecasting. Long Short Term Memory (LSTM) has been used in text generation and prediction (Ghaderi et al., 2017). LSTM are good at predictions correlation between sequential data (van der Walt and Botha, 2016). Three models ordinary least squares (OLS), Bayesian ridge regression (BRR) and SVR were used to compare their performance and persistence forecast was used as the benchmark.

Other neural network (NN) methods that have been used, which perform better than the linear methods in time series forecasting are multilayer perceptron (MLP) which uses feed forward neural network (FFNN). Another NN methods that has been used is the time lagged neural network (TLNN), which uses a time series as input in particular lags as input $(x_{t-1}, x_{t-2}, \ldots, x_{t-p})$. An example of TLNN is using seasonal periods like 12 months for a year or 24 hours like in our case. Lastly, there is seasonal artificial neural network (SANN) which performs better on seasonal times series data. This model learn the seasonal patterns in the dataset and that eliminates the preprocessing step (Adhikari and Agrawal, 2013).

## 1.3 Challenges in wind energy production

Since wind is highly variable, it is not easy to accurately predict the wind power that will be generated from the wind turbines. There is a high correlation between wind speed and wind power generated. The other challenges with wind power is that we cannot increase the power output to meet demand since it only depends on the wind speed which we have no control over.

Wind speed which is the main resource for wind power production cannot be stored for future use unlike fossil fuels. Predicting wind speed is better than predicting power output generated. If two or more turbines experiences the same wind speed we can combine the power output they generate.

## 1.4 Organization of the project

This project consists of five chapters which are organized as follows. Chapter 1, presents the introductory part which consists of the background of wind energy, motivation of wind energy production. Chapter 2, presents the related works which include statistical methods used in time series forecasting. A detailed background of machine learning and optimization algorithms is also discussed. Chapter 3, consists of a detailed description of the three predictive models used namely; persistence model, support vector regression model and Long and Short Term Memory model. Chapter 4, contains results and discussions. Data collection is also discussed in this chapter. Finally, Chapter 5 presents the conclusion and future work.

# 2. Related work

In this chapter, we discuss briefly related work to time series forecasting including statistical methods that have been used before. A detailed background of machine learning and optimisation algorithms will also be presented.

## 2.1 Statistical methods

Autoregressive Integrated Moving Average (ARIMA) model is one of the methods used to solve time series problems. This model is derived from three basic linear components which are the autoregressive (AR) model, integration (I) and moving average (MA). All these models assume a linear correlation between variables of the time series which is not true in most real world situations (Díaz-Robles et al., 2008). The ARIMA model is univariate and the mathematical formulation is:

$$y_t = \theta_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t - (\theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}), \tag{2.1.1}$$

where $y_t$ and $\epsilon_t$ are the target value and random error at time $t$ respectively. $\theta_j (j = 0, 1, \cdots, q)$ and $\phi_i (i = 1, 2, \cdots, p)$ are the model parameters with order $q$ and $p$ respectively. When $q = 0$, Equation 2.1.1 represent an AR model of order $p$, and when $p = 0$ the model represent a MA model with order $q$ (Zhang, 2003).

A modified version of ARIMA model is used when we are dealing with seasonal time series, and this model is called seasonal autoregressive integration moving average. Hybrid methods were proposed in literature that combine linear models such as ARIMA and non linear models such as artificial neural networks. These hybrid models have shown an improvement in the accuracy of time series forecasting as compared to traditional statistical methods (Zhang, 2003).

There was a need to use non-linear forecasting models especially in volatility finance time series and economics, which linear models are unable to forecast. Some of the non-linear models that were proposed in literature are Autoregressive Conditional Heteroskedasticity (ARCH), Threshold Autoregressive (TAR), Non-linear Moving Average (NMA), Non-linear Autoregressive (NAR) and many more (Adhikari and Agrawal, 2013)

## 2.2 Machine learning

Machine learning (ML) is the ability of a machine to learn from data. It is described by (Carbonell et al., 1983) as the field concerned with how to make computer programs that improve on their performance with experience. The computer program is assigned a task and to complete that task it has to be given data to learn from.

**2.2.1 Machine learning algorithms.** There are four types of tasks that a ML algorithm can do namely; supervised, unsupervised, reinforcement learning and semi-supervised learning.

**Supervised learning** is a ML algorithms that uses a set of input data ($X$) to predict given output ($Y$).

$$f : X \to Y$$

In supervised learning, we are look for a function such that the error between the function and $y_i$, $\forall x_i \in X$ and $y_i \in Y$ is small. If $Y$ is continuous we will use regression or forecasting, and if $Y$ is discrete, classification algorithm are used (Nielsen, 2017).

There are several types of supervised learning algorithms, namely Support Vector Machine (SVM), Artificial Neural Network (ANN), K-Nearest Neighbors algorithm (KNN), Decision Tree, Bayesian Classification etc. For our study we will only concentrate on SVM and ANN.

Support vector regression (SVR) is an extension of SVM that is used for solving regression problems. It is a supervised ML algorithm used to find the optimal hyperplanes, such that all the points lie within an epsilon distance away from it. If the data is non-linear a kernel is used to transform the input data to higher dimension where the data can be linearly separated.

**Unsupervised Learning** is a ML technique that tries to find similar characteristics from unlabelled data by representing it in smaller clusters. Unsupervised learning can be used for feature extraction (Krenker et al., 2011).

**Reinforcement Learning** is a ML technique that learn based on environmental interaction. Usually the data is not provided, but it is generated during the interaction with the environment. It deals with how the parameters of the ANN should be changed in order to maximize the gain base on a reward if the respond to the environment was positive and a punishment if the environmental response was negative (Krenker et al., 2011). The results is not imminent i.e. these type of ANN are useful for long term gains by using the short term experience to learn. An example of reinforcement learning is a cleaning robot.

**Semi-supervised learning** uses both supervised and unsupervised learning algorithms, by trying to estimate labels of unlabelled data using labelled data. Unlabelled data with similar characteristic will be in the same class (Goodfellow et al., 2016). This type of ML algorithm is mostly used if we do not have enough labelled data or generating of new data from existing data.

**2.2.2 Deep Learning (DL) Architectures.** . An artificial neural network with multiple hidden layers between the input and the output layers is called deep learning (Patterson and Gibson, 2017). There are three DL classifiers that we will discuss namely; Artificial Neural Network (ANN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Although the discussion of a perceptron will follow later, a combination of 1 or more of them is call artificial neural network.

(a) **Artificial Neural Network.** ANN is a mathematical model that tries to simulate the structure and function of biological neuron network. All artificial neuron are built from two basic rules of weighted sum of inputs and activations. The output of neuron is the weighted sum plus the bias through an activation called the transfer function. The artificial neuron follows simple rules, but their usefulness comes when we interconnect them and these interconnected neurons are called ANN shown in Figure 2.1.

To fully harvest the mathematical benefits of ANN they are not connect randomly. Several standardized ANN topographies have been proposed from previous researches (Krenker et al., 2011). There are two main types of ANN connections, where the information flow from input to output in one direction and these are called feed forward ANN. The second one, the information flows from the input to the output and uses the output as part of the input, these are called recurrent neural networks. The type of ANN topography to use depends on the type of problem we want to solve (Krenker et al., 2011). There are 4 processes of ANN namely feed-forward propagation (FF), cost function (CF), backward propagation (BP) and updates of weights and bias.

(b) **Convolutional Neural Network (CNN).** Their goal is to learn higher order features in data using complex structure. CNN work best in object recognition and has been successfully used in image

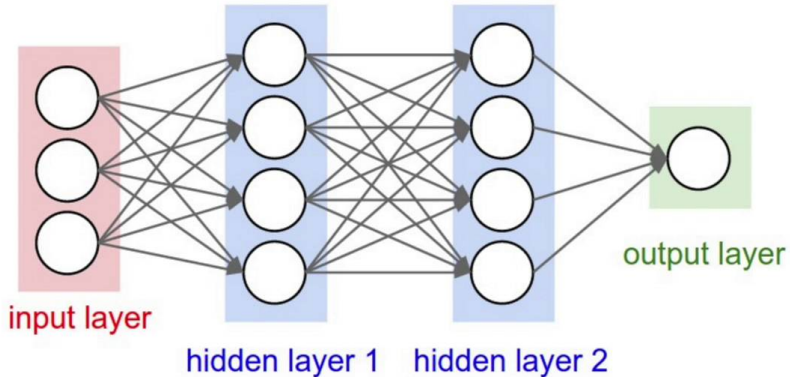


Figure 2.1: A three layer artificial neural network (Karpathy, Accessed May 2018).

classification. They can identify faces, streets signs, and can also be used in text analysis by recognizing the characters in the text. Recent research advances in CNN has lead to improvements in technologies like self-driving cars , robotics, treatment in visual impaired and many more.

(c) **Recurrent Neural Network.**

According to (Chung et al., 2014) RNN is an extension of a normal feed forward neural network which is able to handle variable-length sequence input. RNN uses information from previous output with current input to give an output. RNN have a memory that stores the output from the previous time steps. It has been observed that gradient descent in RNN vanishes due to the long-term dependence on the previous output according to (Bengio et al., 1994).

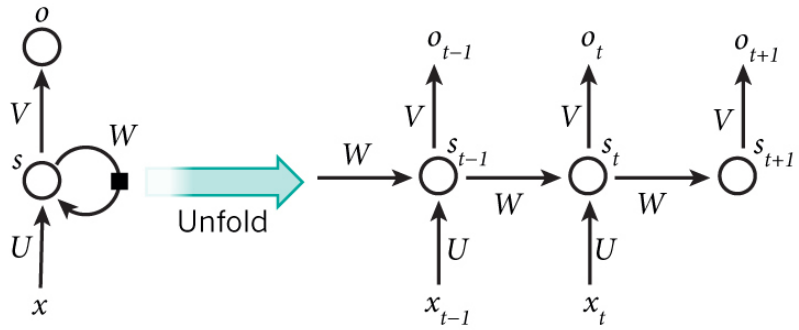


Figure 2.2: A Folded and unfolded RNN architecture with with two inputs (Goodfellow et al., 2016).

Alternative methods to try and solve this problem lead to the development of long short term memory (LSTM). This Neural network architecture was proposed by Hochreiter and Schmidhuber (1997), which include a complicated NN which has a memory cell that is able to remember useful events from the past and forget less useful information.

**RNN Architecture**

The architecture of the RNN is discussed below. Figure 2.2 Shows the folded and unfolded RNN architecture, where the parameters are given in Table 2.1.

Table 2.1: RNN parameter descriptions and their dimensions

| Parameter name | definition | dimensions |
|---|---|---|
| $x_t$ | input vector at time $t$ | (data-points,1) |
| $W$ | weight matrix in the hidden state | (hidden layer,hidden layer) |
| $U$ | weight matrix in the input layer | (hidden layer,data-points) |
| $V$ | weight matrix in the output layer | (output,hidden layer) |
| $s_t$ | hidden state or state transition | (hidden layer,1) |
| $o_t$ | out-put layer | (output,1) |

The state transition is given by

$$s_t = f_w(Ux_t + Ws_{t-1}), \tag{2.2.1}$$

where $s_{t-1}$ and $f_w$ are the hidden state from the previous time step and activation function respectively. The output $(o_t)$;

$$o_t = g_w(V \cdot s_t), \tag{2.2.2}$$

where $g_w$ is the activation functions usually relu function, which we will discuss in Section 2.2.4.

For the network in Figure 2.2 it use data from two previous time steps (window size) for the predictions. To update the weights we use back-propagation through time (BPTT), i.e we start by updating our weights at time $t$ back to time $t = 1$. This makes the calculating of the error a bit easier since the weights are shared throughout the network.

Traditional RNNs have loops which use the output as part of the input. This chain like structure makes them relevant in the use of sequential data. Instead of having a single neural network, LSTM

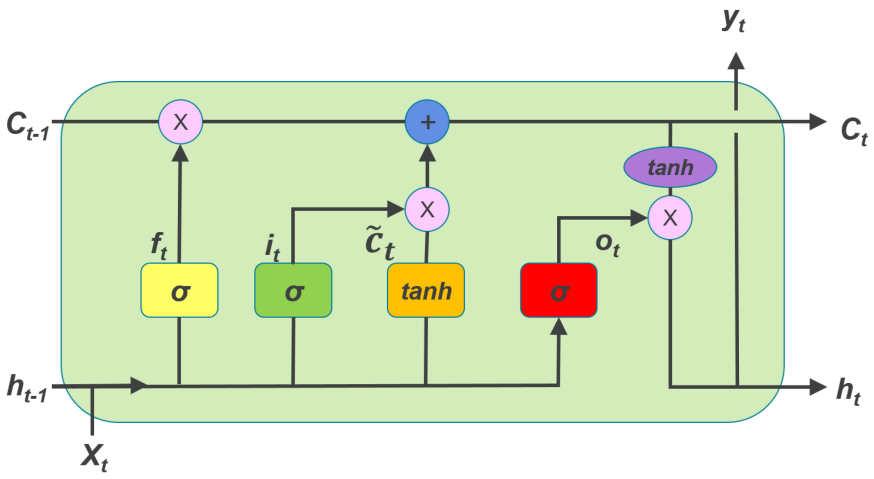


Figure 2.3: A LSTM cell with 4 gated artificial neurons (Olah, Accessed April 2018).

has four artificial neurons which interact in a special way which will be discussed below. The most important part of LSTM is the cell state $(c_t)$ which is the horizontal line running through the top in Figure 2.3.This is the memory that carry important information from previous time steps. The

LSTM has the ability to add and remove information from the cell state, using regulators called gates. Gates have point-wise multiplication operation and a sigmoid NN layer. The sigmoid ($\sigma$) output act as a filter of how much information should be throw away described as a forget gate: $f_t = \sigma(W_f[h_{t-1} + b_f])$, where $W_f$, $h_{t-1}$ and $b_f$ are the forget weights, output from previous time and the bias for the forget gate respectively (Olah, Accessed April 2018).

Deciding which information to be added to the cell state, we use the input gate; $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$, where $W_i$, $x_t$ and $b_i$ are the input weights, the input variable at time $t$ and the input bias. The input gate layer works with tanh function that create candidates vector of information to add, $\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$, To update the cell state,

$$C_t = C_{t-1} \bigotimes f_t \bigoplus i_t \bigotimes \tilde{C}$$

we use the previous cell state and add information from the current output, where $\bigotimes$ and $\bigoplus$ are the point-wise multiplication and point-wise addition respectively. Although we use the sigmoid on $x_t$ and $h_{t-1}$ to filter how much information the output should have, the output depends on the cell state described by (2.2.3),

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \bigotimes \tanh(C_t), \end{aligned} \tag{2.2.3}$$

LSTM have been used in numerous studies and it has been shown that it performs better than an traditional RNN on sequential data (Chung et al., 2014). LSTM will be used in this study as the main forecasting model.

**2.2.3 Perceptron.** A perceptron copy how a biological neuron work. It receive $m$ input, with each input given a weight which determine the importance of the input to the final output. It is a mathematical function that takes in a weighted sum of its inputs and pass it through an activation to a single value output that depends on some threshold ($\theta$). This function will output a real value (0 or 1) depending on the input (Patterson and Gibson, 2017). It is defined by (Bennamoun, May, 2016) as single feed forward neural network which takes in $m$ input and map it into a single output depending on a threshold $\theta \in \mathbb{R}$ using a weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_m)^T \in \mathbb{R}^m$ and a bias as seen in Figure 2.5. The output of the perceptron is:

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} < \theta \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq \theta \end{cases}. \tag{2.2.4}$$

A perceptron with $m$ input described by a weight vector is described in (2.2.4). For convenience the threshold is replaced with a bias ($b$) term and the weighted sum is written as $z = \sum_{i=1}^{m}(w_i x_i + b)$, then (2.2.4) becomes.

$$\hat{y} = \phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}, \tag{2.2.5}$$

where $\phi$ is an activation function. The $m$ input values are points in an m-dimensional hyper plane and this hyperplane separates the input space into two halves, and this is utilized in supervised learning under binary classification.

**2.2.4 Activation functions.** Without activation function, the output of artificial neural networks will simple be linearly connected to the input. This becomes a linear regression model which will be difficult to use since most of the underlying functions are non-linear and the data is non-linear separable, like image classification, text prediction, time series forecasting, and e.t.c (Goodfellow et al., 2016). An activation function gives a single output from a set input. Mostly, in ANN a continuous and differentiable function is chosen, this enables us to use gradient descent in trying to minimize the error during back-propagation.

$$\begin{aligned} y = & \text{ activation } \left( \sum \text{weights} \cdot \text{input} + \text{bias} \right) \\ & = \phi(\mathbf{w}^T \mathbf{x} + b). \end{aligned} \tag{2.2.6}$$

(a) Sigmoid function or logistic activation. It takes a real-valued number in the range $(-\infty, \infty)$ and force it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1 as seen Figure 2.4. The mathematical formula for sigmoid is:

$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{2.2.7}$$

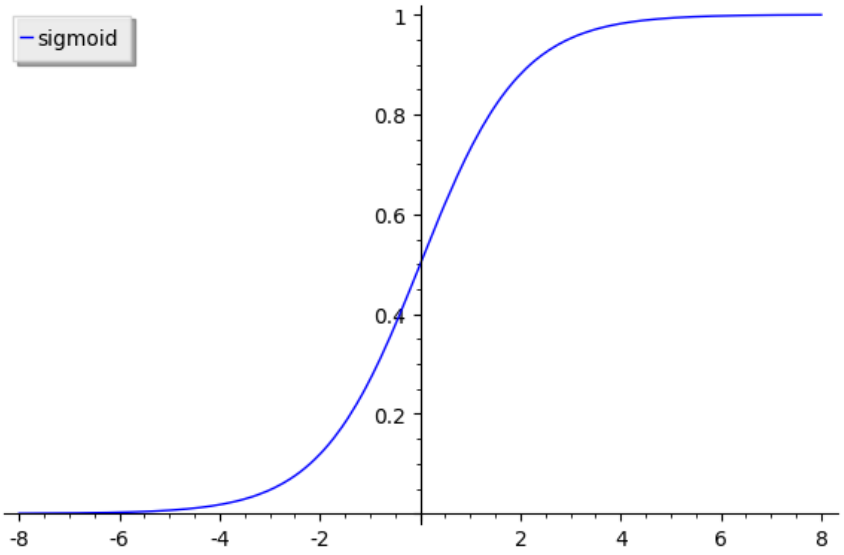


Figure 2.4: A sigmoid activation function.

The gradient of a sigmoid function goes to zero as $z \to \pm\infty$ and this cause slower convergence of the function. Another problem is vanishing gradient, which can occur if we multiply weights which are much smaller than 1. The output is bounded in the range $(0, 1)$. Figure 2.4 show an activation outside the range $(-2, 2)$ will be push towards either $0$ or $1$ which makes clear separations, thus making it a good activation for classification (Avinash, Accessed May 2018). The sigmoid activation function is differentiable and monotonic (if $x \geq y$, then $\phi(x) \geq \phi(y)$) which make it easy to calculate the gradient of the functions and is used in feed forward neural networks (FFNN).

(b) Tanh function. It is a scaled sigmoid function, so it has the same properties as a sigmoid function except that it is bounded in the range (-1,1),

$$\begin{aligned}\phi(z) = \tanh(z) &= \frac{e^{2z}-1}{e^{2z}+1} \\ &= 2\ \text{sigmoid}(2z) - 1.\end{aligned} \tag{2.2.8}$$

The gradient of a $\tanh$ function is much larger than that of a sigmoid function (Avinash, Accessed May 2018). It is mainly used in binary classification and can also be used in FFNN.

(c) Rectified Linear Unit (Relu) is non-linear activation function which output 0 if the output is negative and $x$ if the output is positive. Unlike sigmoid and $\tanh$, relu does not have a problem with vanishing gradient (Patterson and Gibson, 2017). It can make activation to have big values and makes all the neurons to be active. The neuron can die, when the output is negative and the activation gives zero, meaning neuron will no longer updates weights during backpropagation.

$$\phi(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}. \tag{2.2.9}$$

(d) Softmax activation function is a type of sigmoid function used to handle logistic regression with classes ($k$) greater than 2. Each of the classes is shown as a probability which shows how much a certain output is likely to be true, and the sum of all the probabilities equal to 1 (Goodfellow et al., 2016).

$$\phi(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_k}} \qquad \text{for } j = 1, 2, \ldots, k. \tag{2.2.10}$$

The choice of activation to use depends on both the task we are doing and the properties of the activation functions. A sigmoid function works better in classification than the other activation functions. Sigmoid and $\tanh$ both have vanishing gradient near the boundaries, they are both bounded. The relu is the most used activation function in general, especially in the hidden layer, because learning is faster if the input is positive, unlike logistic function. It also solve the problem of vanishing gradient.

**2.2.5 Cost function.** In machine learning we are concerned about how good an algorithm is at predicting the output of a given a input. A cost function is a measure of how good a predictive model is, i.e. minimizing the error $E(w)$,

TO minimize the error we need to minimize the cost function. One of the methods for error minimization methods is called gradient descent. Gradient descent takes the negative of the partial derivatives of the cost function with respect to the weights (Nedrich, 2014).

All ANN algorithms follow the same principles, they use the input data to feed to a neural network (NN), get the predicted values using the feed forward , once we have the predicted output $\hat{y}$ we compare it with the actual value of the output $y$.One way of representing the difference between the predicted and the true values is with a quadratic loss function, given by:

$$E(\text{w,b}) = \frac{1}{N}\sum_{i=1}^{N} ||\hat{y}_i - y_i||^2. \tag{2.2.11}$$

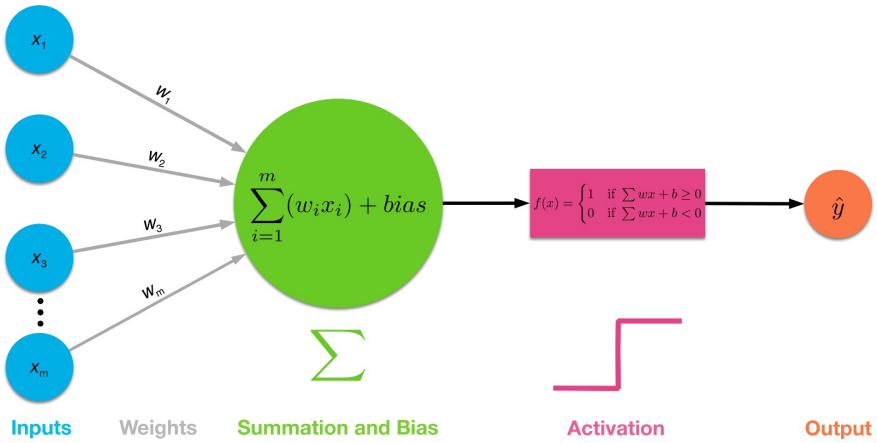


Figure 2.5: A perceptron with $m$ input and one output (Gupta, Accessed April 2018)

There are other cost functions that can be used for calculating the error. The choice of cost function to use depends on the kind of data and problem we want to solve.

Another popular cost function is the **cross-entropy** given by (2.2.12)

$$H(p,q) \;=\; -\sum_{i} p_i \log q_i \;, \tag{2.2.12}$$

where $p$ is the actual probability of a class and $q$ is the predicted probability of the same class. The cross-entropy is better than the quadratic cost function when dealing with classification problems and the quadratic loss function work better in regression problems.

**2.2.6 Sequential data.** Feed-forward Neural Network (FFNN) are not good at predicting sequential data, because their output depends on the current input and the weights and bias, thus a need to use RNN, in which information is looped back. When RNN makes a decision, it considers the current input and also used what it learned before to get the next output. This kind of NN is very useful in predicting data that is time dependant.

The collection of organized data from observations is called time series. (Aghabozorgi et al., 2015), defined it as a sequence of continuous real valued elements. Time series $T$ can be mathematical defined as ordered sequence of $n$ real-variable, $T = (t_1, t_2, \ldots, t_n), t_i \in \mathbb{R}$ (Esling and Agon, 2012).

Time series data mining unveil many sides of the data's complexity that we would have never thought off. This is some times difficult to measure using human perception, that is why the advancement in high speed computing come in to help uncover some of the hidden patterns in the dataset. Time series data is of interest in many sectors including science, engineering, finance, health-sector to name a few. Data analysis help in uncovering interesting patterns and can help in making good predictions and recommendations (Aghabozorgi et al., 2015).

**2.2.7 Back Propagation and Optimization function.** Before neural networks can predict an output it needs to be trained on some data, usually the data is labelled for predictive models, i.e. it consist of a pair of input and output or target. The training usually starts with randomly initialized weights for all the neurons in all the layers and activations are calculated in each neuron from the input layer to the output layer using one of the activation functions and this process is called **forward pass**.

Ideally we want the predicted output to be the same as the target, but in most cases this does not

happen due to randomly initialized weights. To try and get the ideal case where the NN can accurately predict the output, optimization of the weights is needed and this processes called backpropagation.

$E(w, b)$ in (2.2.11) parametrize the weights and the bias to minimize the error. One of the functions used to calculate how weights should be changed in order to minimize the error is called gradient descent. Gradient descent work by taking the partial derivative of the loss function with respect to the weights, and update the previous weights with the new weights. $\eta$ is the learning rate, which simply is the measure of how much the weights and bias should be updated (Nielsen, 2017).

$$
\begin{aligned}
w^{k+1} &= w^k - \eta \frac{\partial E(w,b)}{\partial w^k} \\
b^{k+1} &= b^k - \eta \frac{\partial E(w,b)}{\partial b^k}.
\end{aligned} \tag{2.2.13}
$$

**2.2.8 Back Propagation Through Time (BPTT).** To calculate the gradient descent of each weight we makes use of chain rule since the error does not explicitly depend on the weights. Let $o_j^l$ be the output of the $j^{th}$ neuron in layer $l$ in a NN be defined as:

$$
o_j^l = \phi(\sum_k w_{jk}^l o_k^{l-1} + b_j), \tag{2.2.14}
$$

where $k$ is the number of weights connecting to $j^{th}$ neuron in each layer (Nielsen, 2017).

BPTT is used to learn in RNN. Unlike FFNN, RNN are able to encode longer past information, thus making them suitable for sequential models. BPTT is just an extension of the ordinary back propagation. In RNN we need to learn 3 weight Matrices U, V and W as explained in Table 2.1. Since RNN depends on time, the cost function used in BPTT to learn U, V, and W depends on current input at time $t$ and input from previous time step $(t-1, t-2, \dots)$ making it different from the cost function in (2.2.11) The complexity comes from the use of non linear activation functions and the cost function implicit dependence on the weight. The gradient is calculated using the chain rule as follows (Guo, 2013)

$$
\begin{aligned}
\frac{\partial E}{\partial w_{jk}} &= \frac{\partial E}{\partial o_j^l} \frac{\partial o_j^l}{\partial \phi_j} \frac{\partial \phi_j}{\partial w_{jk}} \\
\frac{\partial E}{\partial b_j} &= \frac{\partial E}{\partial o_j^l} \frac{\partial o_j^l}{\partial \phi_j} \frac{\partial \phi_j}{\partial b_j}.
\end{aligned} \tag{2.2.15}
$$

Since $o^l$ depends on $o_k^{l-1}, o_k^{l-1} \dots, o_k^1$, this means that all output units contribute to the error. Weights are shared over the network, this makes the computations simpler, and faster by reducing the parameters that need to be learned.

## 2.3 Optimization algorithms

Most cases when choosing an activation function, we choose a continuous, differentiable function and sometimes bounded function. A function with these properties enables us to use gradient descent to calculate the change in error due to the weights during backpropagation.

It has been observed that in RNN, gradient tends to vanish in most cases or explodes dues to its long dependence on the time lags, so new alternative methods had to be found. One of the alternative methods proposed was clipping gradient or using of different activation function which is a more complicated task and the LSTM matches this description (Bengio et al., 1994).

For a model to accurately predict output from given input after training optimization algorithms plays a major role. Optimization Algorithms (OA) help in minimization or maximization of the objective function in our case the Error function $(E(X))$. The weights $(w)$ and bias $(b)$ are called the internal learn-able parameters which are used to compute the output of a model. Choosing correctly $W$ and $b$ is important in minimizing the loss function effectively and efficiently.

One of the OA is **gradient descent** (GD) in minimizing the quadratic loss function in (2.2.11). GD has many variants depending on how much data is available, the time at our disposal and what accuracy are we willing to trade off. Below we discuss the different types of GD.

(a) **Batch Gradient Descent/ Vanilla Gradient Descent (BGD)** Compute the gradient of the cost function with respect to the weights for the entire training set and update once. It can be very slow and does not allow the model to update online, but converge to the global minimum for a convex error surface and local minimum for a non convex surface is guaranteed.

(b) **Stochastic gradient descent (SGD)** Unlike BGD, SGD compute the gradient of the cost function and performs the weights updates using (2.2.13) for each set $(x^{(i)}, y^{(i)}), \quad i = 1, 2, \cdots N$ where $N$ is the total number of training examples. SGD can be used for online learning and it is usually faster than BGD, but it is highly variant and does not guarantee convergence to global minimum.

(c) **Mini-Batch Gradient Descent (MBGD)** It is like BGD, however instead of computing the error for the whole training set we take a small sample size $n$ of the total training set and performs the weights updates on. MBGD reduces the variance of the parameter update and increase the stability of the convergence. It is also faster than BGD (Ruder, 2016).

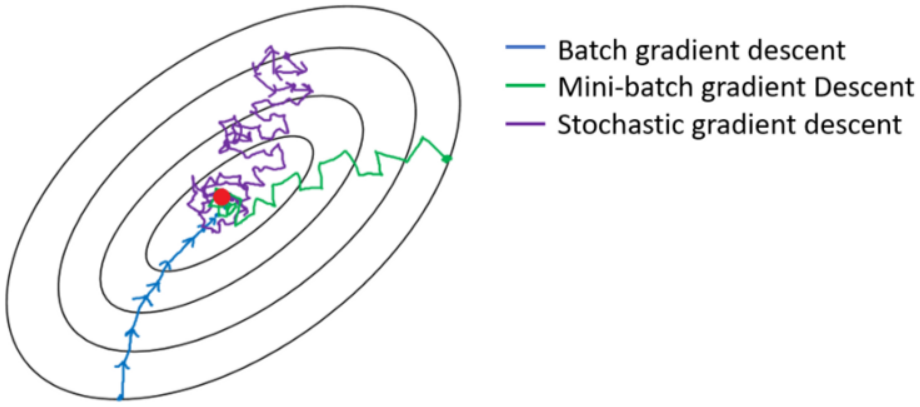


Figure 2.6: The 3 different variants of gradient descent trajectory towards minimum (Imad, Accessed April 2018).

The difference between the three GD variants is summarised in Figure 2.6. We can see that SGD is not smooth and if we have more than one local minima it can converge to it.

For BGD and MBGD, choosing a good learning rate ($\eta$) can be very difficult. If the learning rate is chosen to be very small the cost function converges very slow, while choosing a large learning rate can prevent convergence and cause the loss function to fluctuate around the global minimum or in worse case it can diverges. Another key challenge arises not from the cost function having many local minimum but from saddle points. If we are taking very small steps as you approach a saddle point, the gradient turns towards zero from all direction, this may cause the gradient of the cost function to be equal to zero meaning no updates of the cost function. This results in the cost function getting stuck at a saddle point (Dauphin et al., 2014).

**2.3.1 Gradient Descent Optimization Algorithms (GDOA).** In trying to solve the challenges stated above other OA were developed and we are going to discuss some of the widely used GDOA. SGD does not behaves well when the gradient is large, e.g around local optima. We want a case where gradient descent will take the shortest path toward an optimum, but that does not happen as the gradient fluctuates across the slope while making slow movement towards the bottom of the local optimum. The first GDOA we will look at is momentum.

(a) **Momentum** a method that helps accelerates the SGD in the relative direction. It does this by adding a friction coefficient $\gamma$ of the update from the previous time.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_w E(w) \\ w &:= w - v_t, \end{aligned} \tag{2.3.1}$$

where $\gamma$ is the momentum term which is in the range (0,1). After adding the momentum the function converges faster and reduces the fluctuations (Dauphin et al., 2014).

(b) **Nesterov accelerated gradient (NAG)** is an improvement of the momentum method. The momentum term helps to accelerate the gradient in the right direction but it does so in a blindly way, i.e gradient descent does not follow the optimal path. NAG tries to solve this problem, the loss updates uses the momentum update and the current update term to guess which direction to follow when moving down toward the local optimum. NAG does this by computing the gradient with respect to the future position of our parameter $w$,

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_w E(w - \gamma v_{t-1}) \\ w &:= w - v_t. \end{aligned} \tag{2.3.2}$$

NAG first jump in the direction of the previous gradient, then make a correction by computing the gradient the direction of the anticipated future path (Dauphin et al., 2014). This makes the NAG to slow down as it approach the local optima and it increases performance especially in RNN.

(c) **Adagrad** is one of the adaptive learning rate algorithms. It performs large updates for infrequent parameters and small updates for very noisy (frequent) parameters. For this reason, it is very good for dealing with sparse data. It naturally adjusts the learning rate to meet the demand of different layers (Dean et al., 2012). Unlike the previous discussed GDOA which uses the same learning rate for all parameter ($w_i$) updates, adagrad use different learning rate for all parameter at every time step $t$ and it update the learning rate base on the past gradient that have been computed.

$$\begin{aligned} g_{t,i} &= \nabla_{w_t} E(w_t, i) \\ w_{t+1,i} &= w_{t,i} - \frac{\gamma}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}, \end{aligned} \tag{2.3.3}$$

where $G_t \in \mathbb{R}^{d\times d}$ a diagonal matrix and each $i, i$ is the sum of the squares of the previous calculated gradients $w_i$ up to time $t$, and $\epsilon$ is small term that is used to avoid diving by zero. One of the disadvantages is the accumulation of the squared gradient in the denominator which causes the learning to decrease every time we train meaning the gradients end being updated by very small values (Ruder, 2016).

(d) **Adadelta** and **RMSpro** are methods aimed at correcting the decreasing learning rate by restricting the accumulation of the past squared gradient to a finite windows size, (we are going to discuss more about adadelta). This is done by taking a recursive sum of the gradient as a decaying average using a running average $\mathcal{E}[g^2]_t$ where $g_t$ is calculated the same way as the one in adagrad. The running average only depends on the current and previous gradient thus cutting the long term dependency of all previous terms.

$$\mathcal{E}[g^2]_t = \gamma\mathcal{E}[g^2]_{t-1} + (1-\gamma)g_t^2,$$

from the parameter update vector of adagrad we replace $G_t$ with $\mathcal{E}[g^2]_t$

$$w_{t+1} = w_t - \frac{\gamma}{\sqrt{\mathcal{E}[g^2]_t + \epsilon}} \cdot g_t. \tag{2.3.4}$$

With adadelta setting the default learning rate is not necessary as it will be eliminated from the update rule.

(e) **Adaptive Moment Estimator (Adam)** is another methods that compute adaptive learning rate. It is more like adadelta however it keeps an exponential decaying average of past gradient $m_t$ similar to momentum.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta)g_t^2, \end{aligned} \tag{2.3.5}$$

$m_t$ and $v_t$ are estimates of the mean and the uncentered variance of the gradients respectively and $\beta_1$ and $\beta_2$ are the decay rates between $0$ and $1$.The Adam update parameter equation is given by:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \quad \text{where } \hat{v}_= \frac{v_t}{1-\beta_2^t} \text{ and } \hat{m}_t = \frac{m_t}{1-\beta_1^t}\ . \tag{2.3.6}$$

It has been shown that Adam works well in practise than the other adaptive learning rate algorithms especially on high dimensional parameter space. It is useful when working with large data set since it requires little memory and it is computationally effective (Kingma and Ba, 2014). In this study we are going to use the adam GDOA in our LSTM wind forecasting model.

# 3. Prediction methods

There are different models that can be used to forecast wind speed, and here will discuss the models that will be used in this project. The three models that will be used are, persistence model, SVR model and LSTM model. A mathematical model formulation of the problem is also discussed.

## 3.1 Problem formulation

**3.1.1 Model properties.** We are concerned with a regression task in machine learning that predicts wind speed at different future time using times series data. Two regression models might do the prediction well on a given data. It can be of need to define properties that prediction models should satisfy to ensure good performance, that one can consider in choosing a model. LSTM will be used to find a function that gives good prediction on the time series data. The notations that will be used throughout this work are:

Let

- $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^n$ be the set of input and output data respectively, where $n$ is size of sample and $p$ is dimensionality.
- $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \{\}$, where $X_1,\ X_2$ is the training set and testing set respectively.
- $x_t \in X$, $y_t \in Y$ be the input data and the output data at time $t$ respectively.
- $x_t$, $t = 1, 2, \ldots, T$ be a time series such that $\{x_1, x_2, \ldots, x_T\}$ is a set of repeated observation of the same variable.
- The true underlying function that maps $X$ to $Y$ is

$$f : X \to Y \tag{3.1.1}$$

  which is unknown. Modelling consist of looking for a function $g$,

$$g : X \to Y \tag{3.1.2}$$

  such that

$$E(f, g|X) = \sum_{x_t \in X} ||f(x_t) - g(x_t)||, \tag{3.1.3}$$

  the norm in output space is minimized, which means is less than some threshold $\epsilon$. $f(x)$ is known by its values not its algebraic structure.

Although the study of model properties is out of the scope of this work, the following illustrate such properties.

**Property 1 (Robustness)** Would like the LSTM model to be robust to noise. Let $x_t^* = x_t + \nu$, where $\nu$ denote the noise caused by the instruments used in data collection. We want;

$$g(x_t^*) = f(x_t). \tag{3.1.4}$$

**Property 2 (Optimality)** The optimal function $f(x_t)$ of $g(x_t)$ is unknown. It is not easy to get $g(x_t) = f(x_t)$ since the data available is non linearly separable. So we try to minimize the error between the two functions.

**Property 3 (Convergence)** The regression model $g(x_i)$ has to fit the function $f$ for all sample points such that the error,

$$E(f, g|X) \to 0.$$

**Property 4 (Uniqueness)** The regression function $g(x_t)$ does not have an explicit algebraic expression and is not unique, it depends on the approach used. Different approaches will give different results which depends on that data being used. Let $X$ be as defined above, $g_1(x_t)$ and $g_2(x_t)$ be two regression models. Then;

$$g_1 \text{ is better than } g_2 \text{ in } X \iff E(f, g_1|X) \leq E(f, g_2|X). \tag{3.1.5}$$

## 3.2 Methods

This section discusses how each of the predictive models are built. The structure of the data that each of these models take is also discussed. The methods used when choosing the hyperparameters is also mentioned. Hyperparameters are the non trainable parameters that we choose when building the model e.g. the number of layers in ANN.

**3.2.1 Persistent Forecasting (PM).** Persistent model is the most used benchmarking model in meteorological forecasting in (3.2.1)

$$\hat{p}_{t+k} = p_t, \tag{3.2.1}$$

where $t$ is the time index and $k$ is the look-forward time, $\hat{p}$ is the predicted wind speed and $p$ is the measure values (Nielsen et al., 1998). The persistent model is used because of its simplicity. The model is used with the assumption that the atmosphere is quasi-stationary and it has a time scale of $3$ hours. The forecast window is directly proportional to the error, as the forecast window become greater than the time scale. The persistent error in forecast has $24$ hours cycle, this could be explained by the relationship between the wind speed and time of the day (van der Walt and Botha, 2016).

Due to the slow changes in the atmospheric conditions, RMSE for shorter forecast time is less than RMSE for large forecast time. This means that the present flow does not provide enough information about future flow. A new model is needed for forecasting when the lookahead is large. For $k$ larger than 36, the flow in the atmosphere no longer remain constant and the correlation between $p_{t+k}$ and $p_t$ goes to zero.

The new proposed reference forecast model using a weighting between the persistence and its mean. This model calculate the mean of $\hat{p}$ for each forecast length $k$ and the correlation coefficient $a_k$ (Nielsen et al., 1998). The persistence model takes in the wind speed at current time as input and use it as the forecast time. There is no hyper parameter tuning since there is no training in this model.

**3.2.2 Support Vector Regression (SVR).** The main idea of support vector machine was binary classification. It was used to find the best hyperplane that separates points into two classes. Further considerations were taken into account where SVM was used in function approximation and regression and this is called support vector regression (Adhikari and Agrawal, 2013).

In SRV there are hyper parameters that have to be tuned. The parameter that tell us by how much you want to avoid misclassification is called the penalty parameter $C$. We can work in higher dimensional space if the data is not linear separable, and to do that we make use of the kernel $(k)$. Examples of kernels are, polynomial kernel $k(\bar{x}_i, \bar{x}_j) = (\bar{x}_i \cdot \bar{x}_j + 1)^d$, where $d$ is the dimension of the polynomial. Secondly we have the radial basis function (rbf) $k(\bar{x}_i, \bar{x}_j) = \exp(-\gamma||\bar{x}_i - \bar{x}_j||^2)$, for $\gamma > 0$. Thirdly is th sigmoid kernel $k(\bar{x}_i, \bar{x}_j) = \tanh(\kappa \bar{x}_i.\bar{x}_j + c)$ for some $\kappa > 0$ and $c < 0$ the kernel is related to the transformation $\varphi(x)$ by $\varphi(\bar{x}_i, \bar{x}_j) = \varphi(x_i).\varphi(x_j)$

the weight is given by $\bar{w} = \sum_i \alpha_i y_i \varphi(\bar{x}_i)$, and the role of SVR is to minimize $\frac{1}{2}||w||^2$

subject to $\begin{cases} y_i - \langle \bar{w}, x_i \rangle - b \leq \epsilon \\ \langle \bar{w}, x_i \rangle + b - y_i \leq \epsilon \end{cases}$,

where $x_i$ is the training sample or input space and $y_i$ is the target values. $< \bar{w}, x_i > +b$ is the predictor and $\epsilon$ is the threshold parameter (Botha and van der Walt, 2017). SVR require the tuning of 3 hyper parameters for optimal performance. The parameters are, epsilon $(\epsilon)$ ranging $10^{-2}$ to $10^2$, the penalty parameter (C) ranging from $10^{-2}$ to $10^4$ and the gamma $(\gamma)$ from $10^{-5}$ to $10^2$ (Botha and van der Walt, 2017).

For our model a RBF kernel function is used and after using a search grid to tune the hyper parameters and the ones that gave the minimum error were: $C = 10^2$, $\epsilon = 10^{-6}$, $\gamma = 10^{-1}$. The SVR model takes in data of 2 dimensional shape, which are; sample size and input features.

**3.2.3 LSTM.** The proposed model is for wind speed forecasting at a height of 60 m, and we will use the model to predict 3 ,6 ,12,and 24 hours ahead using Long Short Term Memory (LSTM). The LSTM model had one output layer, meaning for the different forecast time different models were used. A LSTM model takes in 3 dimensional input and in our model the output will be 1 dimensional. The input dimensional are, sample size, the window size and the number of input features. To improve the performance of the model we need to train the model on the whole training set and this process is called an epoch.

A persistent model and SVR models will be used as benchmarking models. Other features will be added such as wind direction, temperature to see how the model will perform. The RMSE will be used to calculate the error. The windows size will be varied between 2 and 24 hour of the previous data points and observe the effects of the windows size on the wind speed forecasting.

The persistence model does not have hyper parameters that need to be tuned to minimize the error. The tuning of hyper parameters is need to minimize the error and some of the hyper parameters needed to be tuned are the number hidden layer, the number of epochs, the number of neurons in each hidden layer, the optimization function to use.

## 3.3 Performance measures

Once the model is built, the next step is to check how good our forecasting model is, and to do that we need to use performance measures. There are several performance measures proposed in literature which compare the actual values and forecasted values of times series. We will discuss a few of the frequently used performance measures.

Let $E_t$ be the absolute error as defined in Equation 3.1.3. Firstly, we will discuss the mean absolute error (MAE) which measures the absolute deviation of the forecast values from the actual values. It is defined mathematically by,

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} E_t. \tag{3.3.1}$$

It shows the average magnitude of the overall error of the forecasting. MAE is sensitive on data transformation and rescaling, and it does not provide any information about the direction of the error.

The mean square error (MSE) is a measure of the average square deviations of the forecast values from the actual values. The mathematical expression is ;

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} E_t^2. \tag{3.3.2}$$

It gives an overall idea of the error in the forecasting, and extreme forecast error are penalized by contributing more to the over MSE. MSE does not provide an idea about the direction of the error and it also sensitive to data transformation and rescaling like MAE.

Lastly we will discuss the root mean square error (RMSE) which is nothing but the square root of MSE. It is given by,

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} E_t^2}. \tag{3.3.3}$$

Out of all the performance metrics presented above in our model we will use the RMSE because it penalises larger errors than other methods and this is useful in improving the model performance. Also it avoids the use of absolute values which might be difficult to work with in-case of calculating gradient (Chai and Draxler, 2014).

The forecast values will be normalized values. To get the un-normalized RMSE from the root mean square error of the normalized values ($\text{RMSE}_{norm}$) we assume that the mean and standard deviation of the predicted values is the same as that of the target values.

$$\begin{aligned}
\text{RMSE} &= \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}} \\
&= \sqrt{\frac{\sum_{i=1}^{N}((y_{i_{norm}}\sigma_y + \bar{y}_i) - (\hat{y}_{i_{norm}}\hat{\sigma}_y + \bar{\hat{y}_i}))^2}{N}}, \text{ simplify using the assumption} \\
&= |\sigma|\sqrt{\frac{y_{i_{norm}} - \hat{y}_{i_{norm}}}{N}} \\
&= |\sigma|\,\text{RMSE}_{norm},
\end{aligned} \tag{3.3.4}$$

where $\hat{y}_{norm}$ is the normalized predicted target value.

# 4. Results and discussion

This chapter focuses on the results from different wind speed predictive models. It compare the results produced from each model. We will investigate the effects of hyper parameter selection and the effects of different variables on wind speed forecasting.

## 4.1 Data collection

Data used was made available by Wind Atlas for South Africa (WASA). The dataset was obtained from Vredenburg station. It consists of a time series ranging from 1 January 2011 to 31 December 2012 with a 10 minutes resolution and $96336$ data points.

For each parameter the measurements were made 4 times in 10 minutes and the mean, minimum maximum and standard deviation were recorded. The dataset consists of wind speed (WS) measured at 62, 60, 40, 20 and 10 metres above the ground measured in m/s. Other variables such as wind direction (deg) TN, air temperature in °C, temperature gradient between height at 60 and 20 m, barometric pressure (hPa) and relative humidity ($\%$) at 60 m above the ground level were measured. The wind speed mean at 60 m was used as the target variables and all the other parameters were used as predictors.

## 4.2 Data preprocessing

Times series data usually come from observations and sensors which are subject to noise and outliers. Preprocessing is necessary to resolve the issues arising with noisy data, redundancy features and missing values. Data preprocessing lead to high quality and manageable data. One of the data preprocessing techniques is normalization. Normalization deals with linear transformation of time series data by reducing the scaling differences between different data points (Esling and Agon, 2012).

Normalization help in speeding up the mining process and improve accuracy of different models. Normalized data fall within a small specific range which improves the learning rate and help prevent larger ranges of values from overweighting smaller range values (Al Shalabi et al., 2006). It also provides better results for analysis since the data has been rescaled to a specific new range e.g [0,1] (Al Shalabi et al., 2006). There are two widely used data normalization techniques; Mini-max normalization and z-score standardization.

(a) In Mini-max scaling the data is scaled between 0-1. It has a smaller standard deviation thus the data suppresses the effects of outliers. This technique requires us to know the minimum and maximum values in our dataset. The equation is given by;

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{4.2.1}$$

(b) Standardization (z-score) require that we know the mean, which we can easy get from the dataset that we are given, and the equation is given by;

$$X_{norm} = \frac{X - \mu}{\sigma}, \tag{4.2.2}$$

where $\mu$ is the mean, $\sigma$ is the standard deviation, $X$ is the data point from the original data set and $X_{norm}$ is the normalized data point. The z-score normalized data has a mean of 0 and unit standard deviation. This method is useful when the minimum and maximum is unknown, thus it becomes a good normalization method for wind speed forecasting since the maximum and minimum wind speed keeps on changing as we collect more data (Patro and Sahu, 2015).

The dataset is split into two parts; the training set $(X_1)$ which is $70\%$ of the original data set and testing set $(X_2)$ which is $30\%$ of the original data set. The splitting of the data was not random in order to maintain the relationship between the time series. (Dobbin and Simon, 2011) suggested that allocation $2/3$ for training is closer to the optimal for a dataset larger than 100. The optimal proportion of training set has to range between $40\% - 80\%$ for different problems and dataset. Separation of dataset into training and testing dataset help to reduce large bias in estimation. A new data set with a resolution of 1 hour was created from the existing data by taking the values at hourly intervals. The results presented below use the dataset created that has a resolution of 1 hour. The dataset had a few missing value and the missing values were replaced by the mean values of that column. The test data has 4809 data points.

## 4.3 Results

The objective of this work is to predict wind speed using different machine learning algorithms. The different models used for wind speed forecasting were persistence model, SVR and a special kind of RNN called LSTM. All the computation of the SVR and LSTM models were done using resources at Centre for High Performance Computing (CHPC).

The RMSE produced in the results uses normalized values, and as discussed in Section 3.3 that it is variate under transformation and rescaling. To transform the error back to the normal values we can use (3.3.4). The mean and standard deviation of the target values are $6.22$ and $3.487$ respectively. To get an overview of how the data used was correlated, Figure 4.1 shows how different variable the different predictors and the target variable from the dataset is related. The features which are highly corrected to the wind wind speed at 60 m high are shown with green and those with low correlation are shown with red color. The first model to be discussed is the persistent model.

**4.3.1 Persistence model.** The persistence model was used as a benchmarking model. The target values for different forecast times is presented in Figure 4.2. The subplot show wind speed forecast 3, 6, 12, and 24 ahead. The RMSE for the persistent model is show in Table 4.1. The Table shows a summary of the results presented in Figure 4.2

Table 4.1: RMSE for persistent model for the different forecast times

| forecast hours | RMSE |
|---|---|
| 3 | 0.73 |
| 6 | 1.16 |
| 12 | 1.54 |
| 24 | 1.15 |

**4.3.2 SVR model.** The SVR model need some hyper parameter tuning to get the optimal solution. the results for different future time is shown in the Figure 4.3 and 4.4 for window size 2 and 12 respectively. These results were produced using wind speed at 62, 40,20 and 10 metres above the ground. A summary of the results for all the windows sizes considered is shown in Table 4.2. The average loss of each forecast time is presented to make it easy for comparison with other models.

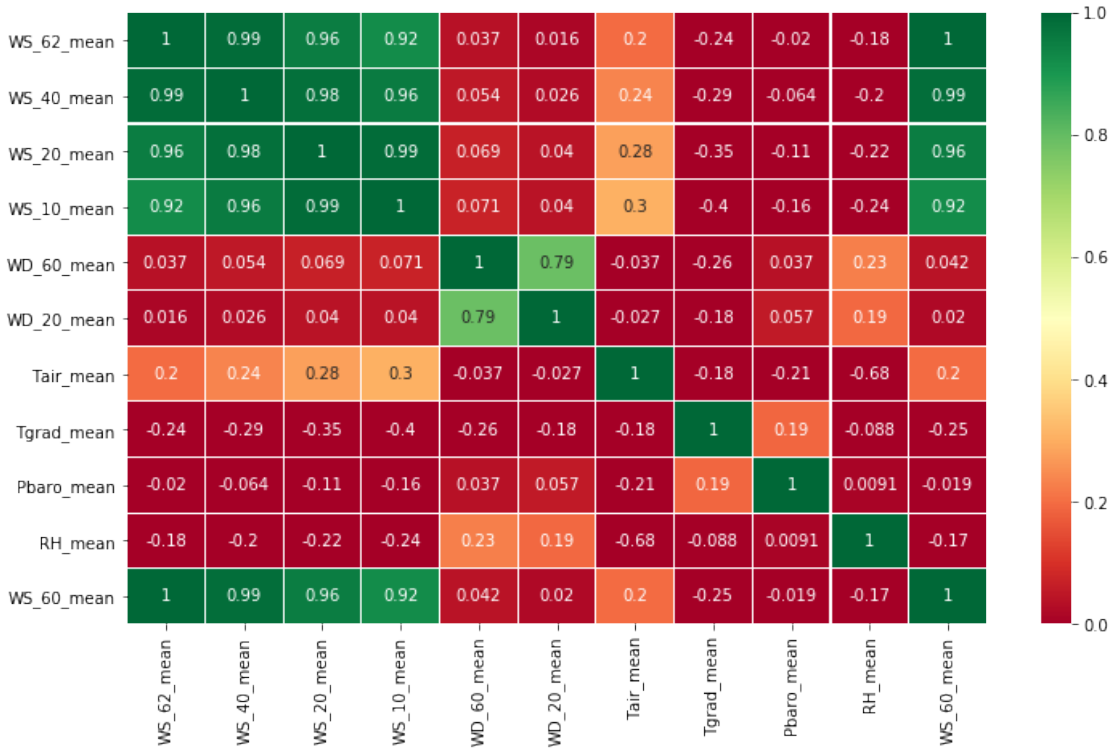


Figure 4.1: Correlation coefficient of the wind data.

Table 4.2: RMSE for SVR model for the different times and window sizes

| | 3 hours ahead | 6 hours ahead | 12 hours ahed | 24 hours ahead |
|---|---|---|---|---|
| window size 2 | 0.964 | 1.001 | 1.049 | 0.503 |
| window size 6 | 1.204 | 1.239 | 1.279 | 0.700 |
| widow size 12 | 1.137 | 1.099 | 1.123 | 0.649 |
| window size 24 | 0.978 | 0.981 | 0.973 | 0.817 |
| average | 1.0708 | 1.08 | 1.106 | 0.667 |

The effect of feature selection is seen clearly from the results in Table 4.3. These results shows clearly the importance of feature section. Table 4.3 shows choosing different window sizes affects the results.

Table 4.3: RMSE for SVR model for the different times and window sizes using all the features

| | 3 hours ahead | 6 hours ahead | 12 hours ahed | 24 hours ahead |
|---|---|---|---|---|
| window size 2 | 0.060 | 0.622 | 0.709 | 0.639 |
| window size 6 | 0.343 | 0.351 | 0.359 | 0.369 |
| widow size 12 | 0.744 | 0.743 | 0.744 | 0.745 |
| window size 24 | 1.031 | 1.030 | 1.030 | 1.029 |

**4.3.3 LSTM model.** The LSTM model used the same predictors as the SVR model. The LSTM model needs hyper parameter tuning in order for the model to get optimum results and these results used 6 neurons in hidden layer and adam as an optimization algorithm. Figure 4.5 show the forecast wind speed with the actual data points at different future times with a window size of 24. For the different windows sizes, the loss is summarized in Table 4.4 which present the data for different window sizes and

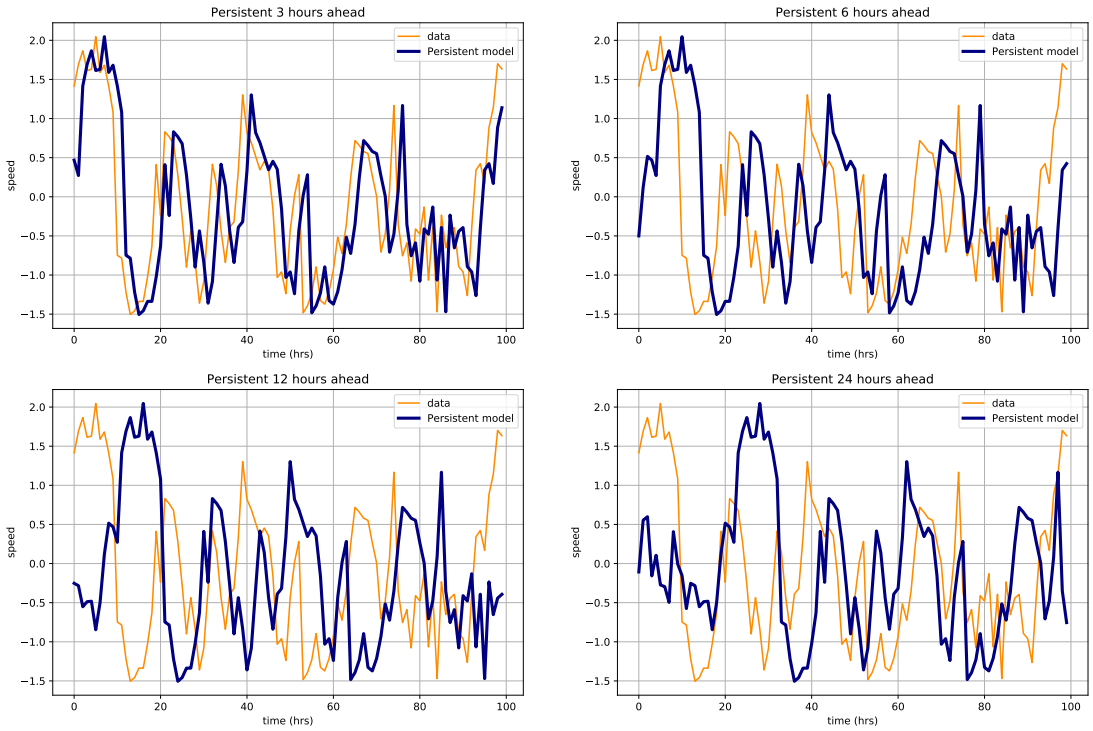


Figure 4.2: Persistent model for 3,6,12,and 24 hours prediction

different future times. The model was run 3 times and the results given in the table is the mean error for all the window sizes and future time steps ahead.

Table 4.4: RMSE for LSTM model for the different times and window sizes with 6 neurons in the hidden layer using wind speeds as features

| | 3 hours ahead | 6 hours ahead | 12 hours ahead | 24 hours ahead |
|---|---|---|---|---|
| window size 2 | 0.934 | 0.975 | 1.021 | 0.494 |
| window size 6 | 0.956 | 0.979 | 1.047 | 0.517 |
| widow size 12 | 0.916 | 0.898 | 0.931 | 0.494 |
| window size 24 | 1.047 | 1.053 | 1.061 | 0.544 |
| average | 0.941 | 0.949 | 0.987 | 0.497 |

The training and validation loss for the LSTM model with 6 neurons are shown in Figure 4.6, this shows how well our model was performing during training stage. The effects of increasing the neurons to in the hidden layer on the training and validation data is seen in Fig 4.8

The summary of the RMSE of the forecast values for a model with 30 neurons in in the hidden layer is shown in Table 4.5.

The performance of the different predictive methods used is shown below

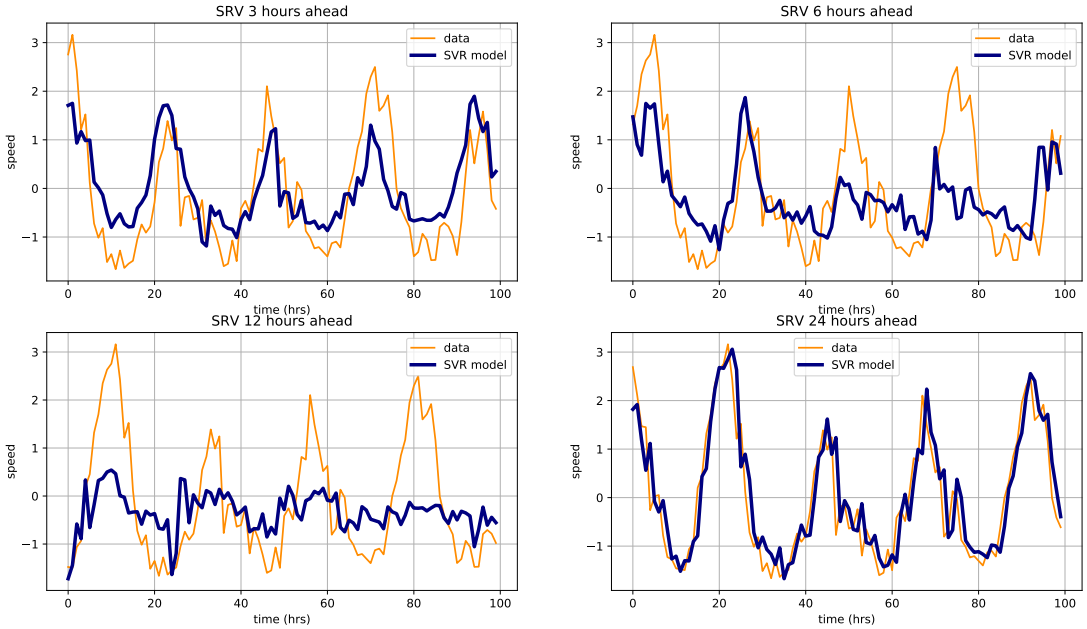


Figure 4.3: SVR model for window size of 2

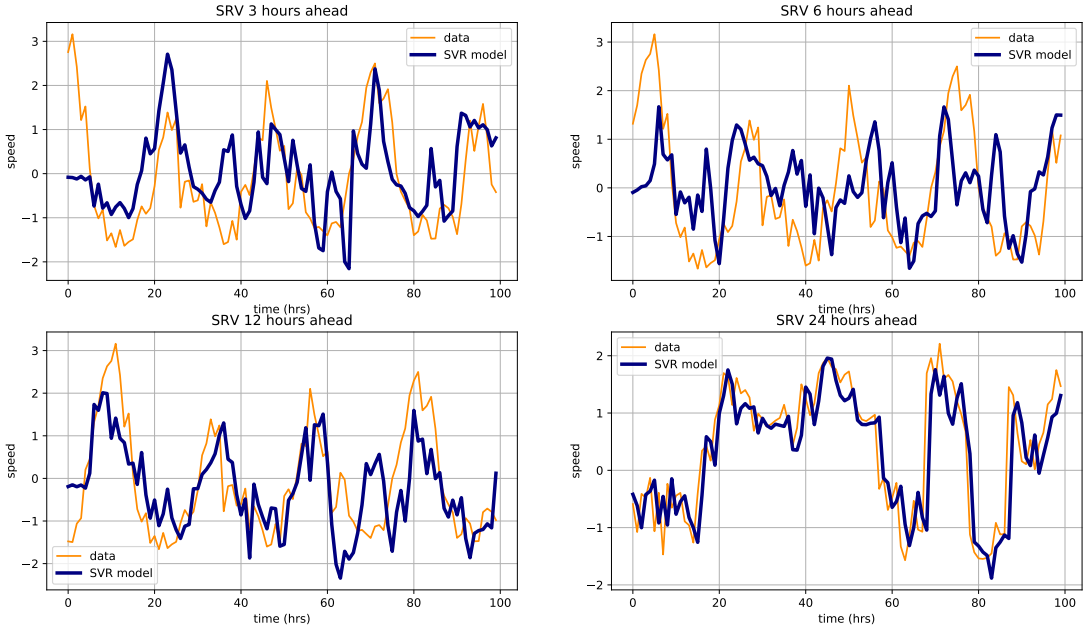


Figure 4.4: SVR model for window size 12

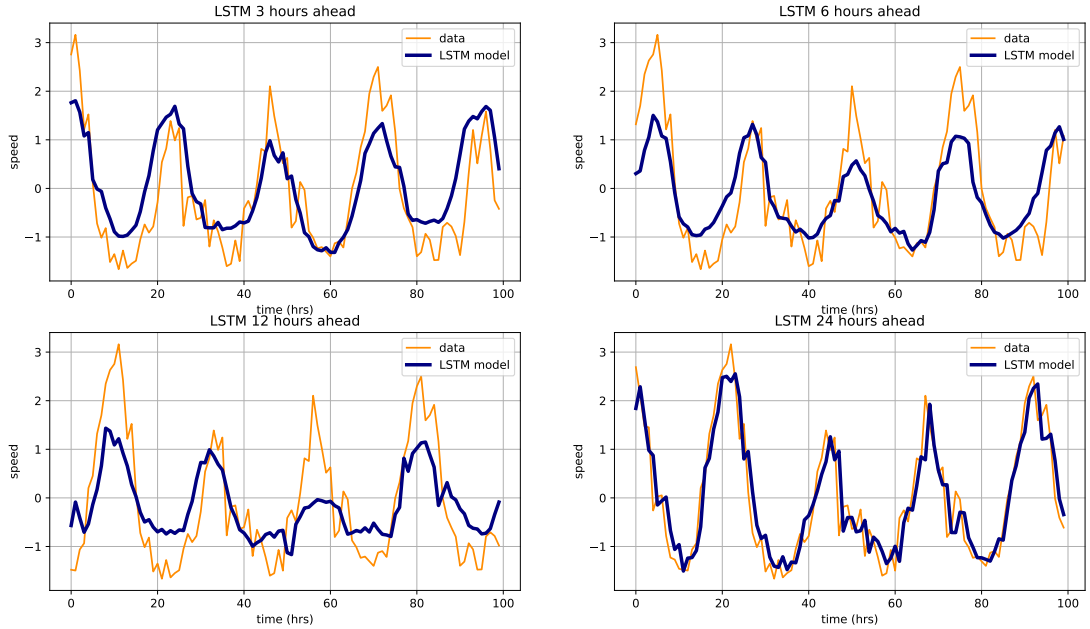


Figure 4.5: LSTM model for window size 24

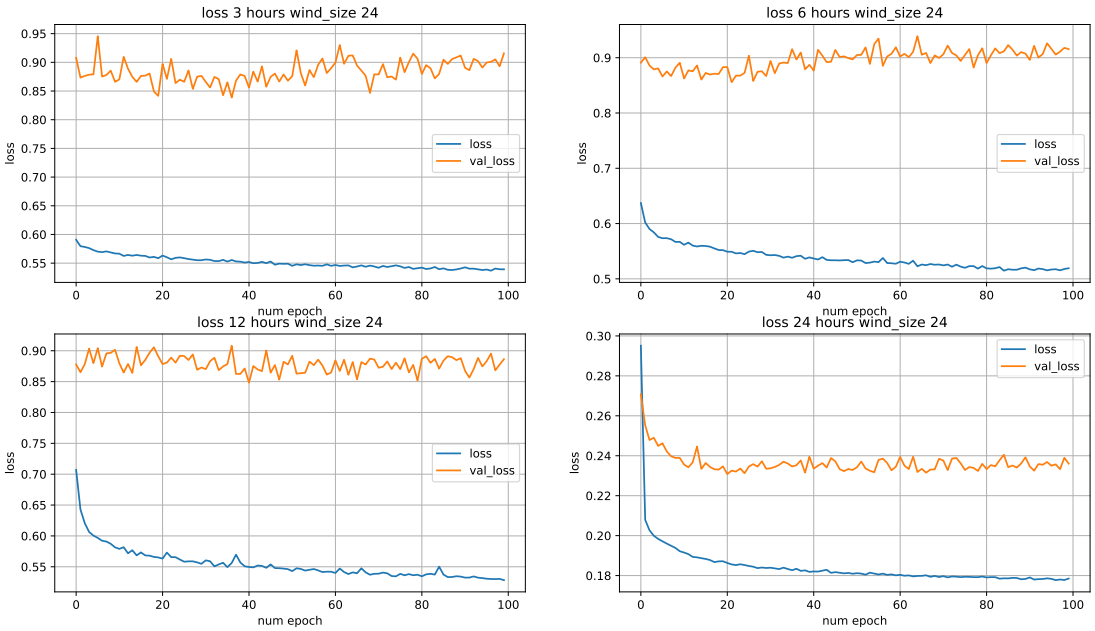


Figure 4.6: Training and validation loss for 6 neurons in 1 hidden layer

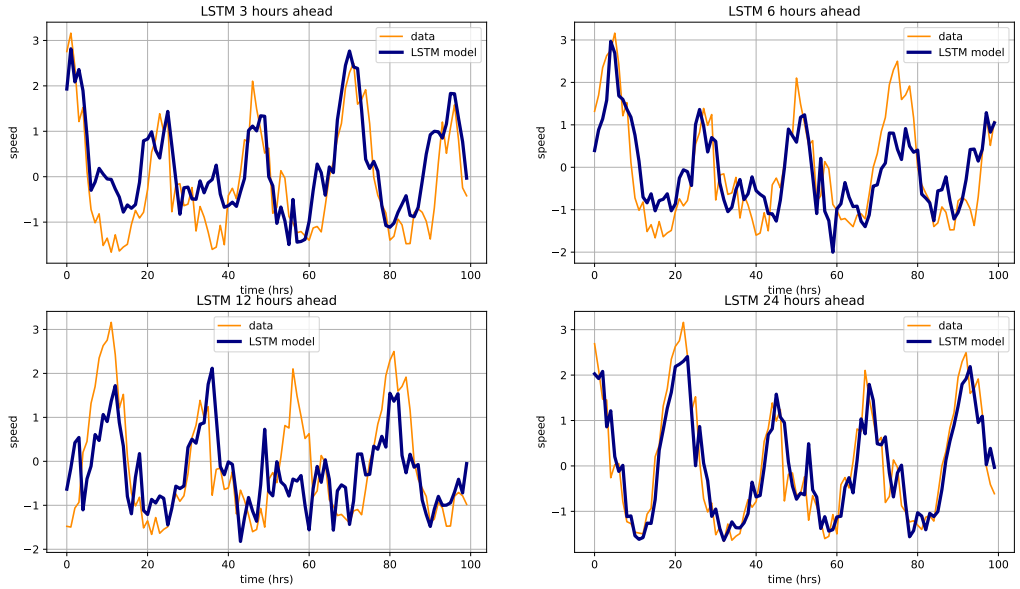


Figure 4.7: LSTM window size 12 for 30 neurons in hidden layer

Table 4.5: RMSE for LSTM model for the different times and window sizes with 30 neurons and 1 hidden layer

| | 3 hours ahead | 6 hours ahead | 12 hours ahead | 24 hours ahead |
|---|---|---|---|---|
| window size 2 | 0.937 | 0.997 | 1.070 | 0.532 |
| window size 6 | 1.067 | 1.075 | 1.163 | 0.561 |
| widow size 12 | 1.070 | 1.076 | 1.099 | 0.561 |
| window size 24 | 1.047 | 1.053 | 1.061 | 0.544 |
| average | 1.030 | 1.050 | 1.0982 | 0.549 |

Table 4.6: RMSE for the different predictive models

| | Persistence model | SVR | LSTM 6 neurons | LSTM 30 neurons |
|---|---|---|---|---|
| 3 hours | 0.73 | 1.071 | 0.941 | 1.03 |
| 6 hours | 1.16 | 1.08 | 0.949 | 1.05 |
| 12 hours | 1.54 | 1.106 | 0.987 | 1.098 |
| 24 hours | 1.15 | 0.667 | 0.497 | 0.549 |

Different set of results were obtained when we used all the features form the predictors given. Table 4.7 provide a summary of all the RMSE for the LSTM model when using all the features. The results will be compared with those in Table 4.4 to see the effects of feature subsection on the model training.

The results when we have used all feature for the LSTM model is show in Table 4.7. These results show how the model performs when feature selection is not done. The results will help in determining the importance of feature selection in LSTM predictive models.

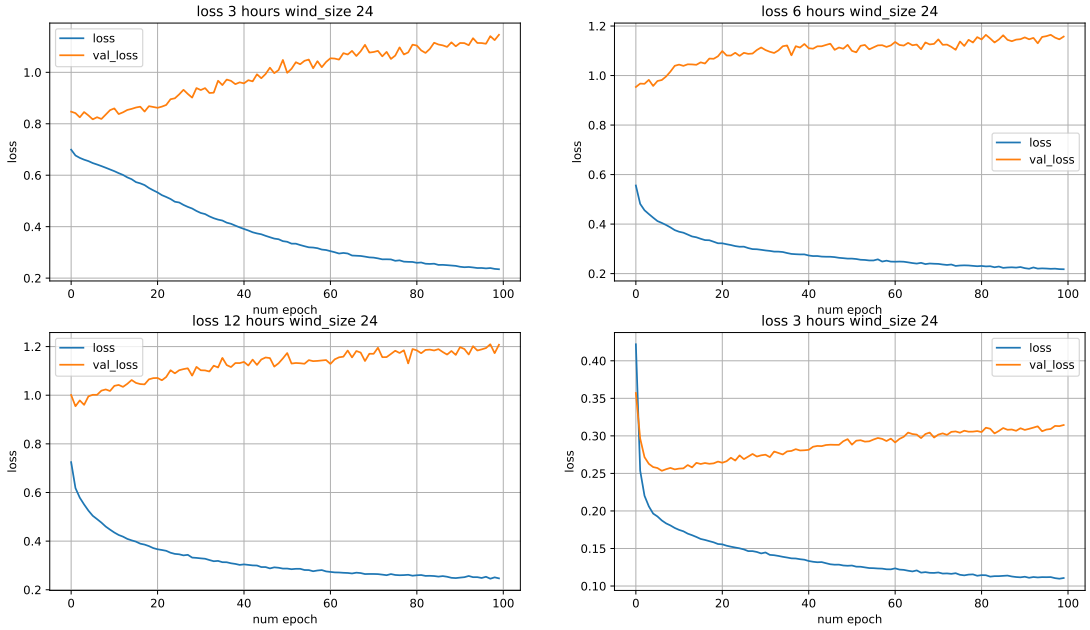


Figure 4.8: LSTM for a windows size 24 and 30 neuron in the hidden layers

Table 4.7: RMSE for LSTM model for all the input features

| | 3 hours ahead | 6 hours ahead | 12 hours ahead | 24 hours ahead |
|---|---|---|---|---|
| window size 2 | 1.205 | 1.297 | 1.293 | 0.683 |
| window size 6 | 1.235 | 1.355 | 1.379 | 0.697 |
| widow size 12 | 1.175 | 1.152 | 1.161 | 0.619 |
| window size 24 | 1.154 | 1.192 | 1.130 | 0.662 |
| average | 1.192 | 1.249 | 1.241 | 0.665 |

## 4.4 Discussion

The wind speed at a height of 60 m has high positive correlated with other wind speed at a height of 62, 40, 20, and 10 m as shown in Figure 4.1. The air temperature also has a positive correlation to the wind speed. Relative humidity has the most negative correlation among with the wind speed.

**4.4.1 Persistence model.** The persistent model which is used for benchmarking has results for 3, 6, 12 and 24 hours ahead. The behaviour of the forecast values for the different future times are shown in Figure 4.2. This model only uses the output vector since there is no training that happens. The persistent model give better results when the forecast time is small. At around 12 hours ahead we get the worst results and as the forecast window goes to 24 hours the error become better. This could be explained by the strong relationship between wind speed and time of the day. Persistence forecasting model is cyclic over a 24 hour period (van der Walt and Botha, 2016).

**4.4.2 SVR model.** From Figure 4.1 we can observe that the wind speed at a height of 60 m is highly correlated with the wind speeds at height of 62 m, 40 m, 20 m, and 10 m. Results from for the SVR model were produced with sub-plots of 3, 6, 12 and 24 hours forecast time. For all the plots we chose

the first 100 data points to show in each of the plots, this enables us to visualize perfectly the treads followed by our forecast results. The results were produced for four different windows sizes, which are 2, 6, 12, and 24.

The results shown in Figure 4.3 and 4.4 are for window size 2 and 12 respectively. The plot in Figure 4.3 show poor results when the forecast time is 12 hours. For 3, 6, and 12 hours ahead the extreme values were not learned properly by the model. Using a windows size of 12 we can observe in Figure 4.4 that the model learn better the trends followed by the winds speeds at different future time values. The model is unable to learn well the behaviour of the wind speeds profiles when we are forecasting 12 hours ahead. The model is able to learn properly if we are predicting 24 hour ahead as compare to the other forecast time. The results for the error of the SVR is summarized in Table 4.2. For 24 windows size 12 hours ahead has a minimum error.

Table 4.3 shows results from the models when we have used all the features. Comparing Table 4.3 and 4.2 show how important choosing the correct features to use when training the model. The windows size plays a important role in the RMSE that we get from the models. A windows size of 6 on average performs better for all the forecast time. For SVR a windows size of 2 hours ahead gives the best result.

Weather seem to follow a 24 hour pattern. the error start decreasing as the the forecast time ahead get closer to 24. A window size of 24 has a small average error across all the forecast future time, this could be caused by the correlation of of how the weather changes daily. The error has its lowest value when the window size is 2 and we forecasting 24 hours ahead. A window size of 12 give better graphical results for 12 hour predictions.

**4.4.3 LSTM.** The standard deviations of the error from the mean of the model with 6 neurons are $0.023,\ 0.022,\ 0.021$ and $0.001$ for 3 hours, 6 hours, 12 hour and 24 hours ahead respectively. This shows the model's consistence, and the results are reproducible.

These results show that the model does not learn well when we predict the near future such as 3 hours ahead. A graphical representation of window size of 24 are is shown. The model get worse as the forecast time increase towards 12 hours this follows the same behaviour as the one observed in the SVR model. Table 4.4 show that the model performs better when the window size is 2 and 24.

This model again show a smaller error when we predict 24 hours ahead as compared to the other future time. When forecasting 24 hour ahead the windows size does not have a big effect on how the model is learning. For a small window size the LSTM model does not learn properly the extreme values.

The results shows that the LSTM model with 6 neurons performs better than all the other models for 6, 12 and 24 hours ahead. The persistent model performs better than all the other models when we are forecast time is small. Table 4.6 shows that with the correct hyper parameter tuning the LSTM model when dealing with time series data can outperform all the other models in this for this dataset.

The model start to learn as time continues. The model fit well for 6 and 24 hours ahead. The model does not perform well when we are forecasting 12 hours ahead, this can be seen in the results in Table 4.5

Increasing the number of LSTM neurons ($>$6) the model training loss decreases smoothly but the validation loss which is expected to decrease start to increase, as shown in Figure 4.8. This is is a result of the model over fitting on training. For 24 hours forecasting the validation loss increase until around 80 epochs and then it start to stabilise as the number of epochs approach 100.

The structure of the neural network and the choice of the hyper parameters are very important in getting the optimal solution. Wrong choice affect the convergence or divergence of the model during training.

Figure 4.6 shows the training and validation loss during the model training. The training loss decrease as the number of epochs increase. The validation loss for 3, 6, 12 forecast does not decrease significantly but it fluctuate between 0.8 and 1. This could be interpreted as the model's training does not depend too much on the number of epoch, meaning reducing the number of epochs does not affect performance of the model. The training loss decreases slowly after 25 epochs, meaning it is not necessary to continues with training.

The results shown in Table 4.7 show the importance of feature selection during the model training. When we have used all the features the forecast time 3, 6 and 12 hours the model does not perform well. Feature sub-selection help learning algorithms to operate faster and more effective (Kotsiantis et al., 2006).

# 5. Conclusion

This project aimed to use machine learning approaches to predict power output from wind farms. Our concentration was more on wind speed forecasting. Due to the relationship between the wind speed and wind power, forecasting one can allow interpretation of the other. In this work, we forecast the wind speed.

To address the forecasting problem, we considered the persistence model as benchmarking model. Other models used were SVR and LSTM. The LSTM is the model we designed to observe how it performs on time series wind speed forecasting. The persistence model outperforms other models when the forecast time is small. The error increases as the forecast time increases.

On average the LSTM outperforms other models. In the LSTM models, the forecast error is smaller when we only use the features that are highly correlated to our target than when we use all the features. The SVR model instead performs better when the window size is 24 and we forecast $24$ hours ahead.

The results from the LSTM model are comparable with the results produced by (Botha and van der Walt, 2017) using the SVR model. Although the performance of the LSTM model is better. It is much more computationally expensive during training than the other SVR and persistence models. In addition, the choosing of hyperparameters correctly affect the performance of the LSTM model during training. Increasing the complexity of the LSTM model causes the model to over-fit on the training.

**Future work** The work we might consider later are as follows: Combining of data from different stations and having a model that will give output of the different stations simultaneously. Creating a model that has a 24 hour output which will give the forecast values of the 24 hours simultaneously. Since the three models works best on different set of forecast times, creating a hybrid model that will combine the three models can be done.

# Acknowledgements

I would like to thank AIMS for giving me an opportunity to study and do this project. I would also like to express my gratitude to my supervisors, Dr. Vukosi Marivate and Dr. Bubacarr Bah for their support, guidance, valuable input and care throughout my work. Furthermore, I extend my gratitude to my tutor Mr Felicien Jordan Masakuna for tutoring me during the course of this project. Next I would like to thank Dr Nicolene Botha from CSIR for her valuable contributions in this project. Not forgetting to thank the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources to this research project. A special thanks goes my colleagues Juliana, Lambert, Rojo and Yasser for the encouragement, the team work and the struggles we had together. I would also like to thank my family for the everything. Above all I would thank the Almighty God who protected me and grant me the strength to do this project.

# References


Adhikari, R. and Agrawal, R. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*, 2013.

Aghabozorgi, S., Shirkhorshidi, A. S., and Wah, T. Y. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

Al Shalabi, L., Shaaban, Z., and Kasasbeh, B. Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739, 2006.

Avinash, S. Understanding activation functions in neural networks. BLog, https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, Accessed May 2018.

Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Bennamoun, P. M. Artificial neural network lect4 : Single layer perceptron classifiers, May, 2016. Unpublished manuscript.

Botha, N. and van der Walt, C. M. Forecasting wind speed using support vector regression and feature selection. In *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), 2017*, pages 181–186. IEEE, 2017.

Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. An overview of machine learning. In *Machine Learning, Volume I*, pages 3–23. Elsevier, 1983.

Chai, T. and Draxler, R. R. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, pages 1–8, 2014.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

Díaz-Robles, L. A., Ortega, J. C., Fu, J. S., Reed, G. D., Chow, J. C., Watson, J. G., and Moncada-Herrera, J. A. A hybrid arima and artificial neural networks model to forecast particulate matter in urban areas: The case of temuco, chile. *Atmospheric Environment*, 42(35):8331–8340, 2008.

Dobbin, K. K. and Simon, R. M. Optimally splitting cases for training and testing high dimensional classifiers. *BMC medical genomics*, 4(1):31, 2011.

Esling, P. and Agon, C. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):12, 2012.

Ghaderi, A., Sanandaji, B. M., and Ghaderi, F. Deep forecast: Deep learning-based spatio-temporal forecasting. *arXiv preprint arXiv:1707.08110*, 2017.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Guo, J. Backpropagation through time. *Unpubl. ms., Harbin Institute of Technology*, 2013.

Gupta, S. Neural networks theory. Blog, https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f/, Accessed April 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Imad, D. Gradient descent algorithm and its variants. Blog, https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3/, Accessed April 2018.

Karpathy, A. Stanford university cs231n: Convolutional neural networks for visual recognition. Stanford University, https://cs231n.github.io/neural-networks-1/, Accessed May 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

Krenker, A., Bester, J., and Kos, A. Introduction to the artificial neural networks. In *Artificial neural networks-methodological advances and biomedical applications*. InTech, 2011.

Nedrich, M. An introduction to gradient descent and linear regression. BLog, https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/, 2014.

Nielsen, M. *Neural Networks and Deep Learning*. MIT Press, 2017. http://neuralnetworksanddeeplearning.com.

Nielsen, T. S., Joensen, A., Madsen, H., Landberg, L., and Giebel, G. A new reference for wind power forecasting. *Wind energy*, 1(1):29–34, 1998.

Olah, C. Understanding lstm networks. Blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Accessed April 2018.

Panwar, N., Kaushik, S., and Kothari, S. Role of renewable energy sources in environmental protection: a review. *Renewable and Sustainable Energy Reviews*, 15(3):1513–1524, 2011.

Patro, S. and Sahu, K. K. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

Patterson, J. and Gibson, A. *Deep Learning: A Practitioner's Approach*. " O'Reilly Media, Inc.", 2017.

Pinson, P. *Estimation of the uncertainty in wind power forecasting*. PhD thesis, École Nationale Supérieure des Mines de Paris, 2006.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Schiermeier, Q., Tollefson, J., Scully, T., Witze, A., and Morton, O. Energy alternatives: Electricity without carbon. *Nature News*, 454(7206):816–823, 2008.

Vahey, S. P. and Wakerly, L. Moving towards probability forecasting. 2013.

van der Walt, C. M. and Botha, N. A comparison of regression algorithms for wind speed forecasting at alexander bay. In *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016*, pages 1–5. IEEE, 2016.

Wang, J., Zhou, Q., Jiang, H., and Hou, R. Short-term wind speed forecasting using support vector regression optimized by cuckoo optimization algorithm. *Mathematical Problems in Engineering*, 2015, 2015.

Zhang, G. P. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

Zhu, X. and Genton, M. G. Short-term wind speed forecasting for power system operations. *International Statistical Review*, 80(1):2–23, 2012.