Artificial Neural Network Used For Short Term Wind Speed Prediction

By: Alexandra Lodge

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

December 2013
# TABLE OF CONTENTS

Table of Figures .................................................................................................................. 3
Table of Tables ...................................................................................................................... 4
Acknowledgements............................................................................................................... 5
Abstract ................................................................................................................................. 6

Chapter I: Introduction ........................................................................................................ 7
Chapter II: Literature Review .............................................................................................. 9
Chapter III: Neural Network Background .......................................................................... 11
Chapter IV: Test Plans and Results .................................................................................... 15
Chapter V: Conclusion and Future Works ......................................................................... 31
References ............................................................................................................................ 34
Appendices ............................................................................................................................ 35
  Appendix A: Analysis of Senior Project Design ............................................................... 35
  Appendix B: Specifications and Requirements ................................................................. 40
  Appendix C: Parts Lists and Costs .................................................................................. 41
  Appendix D: Schedule – Time Estimates ......................................................................... 42
  Appendix E: Project Code ................................................................................................. 44
LIST OF FIGURES

Figure 1: Renewable Energy as Part of Total Primary Energy Consumption, USA 2010 ([2]) ...........................................7
Figure 2: Block Diagram of Supervised Artificial Neural Network using Back-Propagation ...........................................8
Figure 3: Shape of Hyperbolic Tangent Function .............................................................................................................11
Figure 4: Steepest Descent Tangent Function ....................................................................................................................13
Figure 5: Interconnectivity of Weights ..............................................................................................................................13
Figure 6: Data Collector for National Wind Technology Center ............................................................................................15
Figure 7: Selecting Variables to Plot Out of 66 Variables ....................................................................................................16
Figure 8: Determining Output Type to be Downloaded to the Computer ...........................................................................16
Figure 9: Signal Flow Diagram ...........................................................................................................................................17
Figure 10: Neural Network With No Historic Input ...........................................................................................................18
Figure 11: Neural Network with Historic Input ..................................................................................................................19
Figure 12: Denormalized RMSE of Training Results as a Function of the Number of Iterations .....................................20
Figure 13: Denormalized Training Results ..........................................................................................................................20
Figure 14: Denormalized Testing Results ............................................................................................................................21
Figure 15: Denormalized RMSE of Training Results as a Function of the Number of Iterations .....................................22
Figure 16: Denormalized Training Results for Wind ..............................................................................................................23
Figure 17: Denormalized Testing Results for Wind ..............................................................................................................25
Figure 18: Denormalized RMSE of Training Results as a Function of the Number of Iterations .....................................26
Figure 19: Denormalized Training Results for Wind with Historic Input .............................................................................27
Figure 20: Denormalized Testing Results for Wind Data with Historic Input ..............................................................29
Figure 21: Illustration of Sinc Interpolation to Understand Wavelets ([11]) .......................................................................32
Figure 22: Mother Wavelet Shape ([7]) .............................................................................................................................32
Figure 23: Break-Even Analysis ...........................................................................................................................................37
Figure 24: EE 460 Gantt Chart .................................................................................................................................................42
Figure 25: EE 463 Gantt Chart .................................................................................................................................................42
Figure 26: EE 464 Gantt Chart .................................................................................................................................................43
LIST OF TABLES

Table 1: Conditions for XOR Test .......................................................... 19
Table 2: Performance Metrics Determined From the XOR Gate .................. 21
Table 3: Comparison of Expected Values and Network Output Values ............ 21
Table 4: Conditions for Wind Test .......................................................... 22
Table 5: Performance Metrics Determined From the Wind Data ................... 26
Table 6: Conditions for Historic Wind Test ................................................. 26
Table 7: Performance Metrics Determined From Historic Wind Data .............. 30
Table 8: Comparison Of Wind Tests ......................................................... 31
Table 9: Artificial Neural Network Used for the Prediction of Wind Requirements and Specifications .................. 40
Table 10: Parts Cost Table ...................................................................... 41
Table 11: Total Cost Table ...................................................................... 41
Table 12: Tabulated Gantt Chart Estimates for EE 460 .................................. 42
Table 13: Tabulated Gantt Chart Estimates for EE 463 .................................. 42
Table 14: Tabulated Gantt Chart Estimates for EE 464 .................................. 43
ACKNOWLEDGEMENTS

I would like to thank Dr. Helen Yu for supporting me in my desire to learn about Artificial Neural Networks. Throughout my four years at Cal Poly, I have come to her office many times for her help in my Digital Signal Processing Classes. Her patience and willingness to help me with all my questions pushed me to strive for deeper understandings. Most importantly, she introduced me to the idea of being ‘multidisciplinary’ within EE. She encouraged me to pursue both of my interests- renewable energy and DSP in the creation of this senior project.
ABSTRACT

Currently, wind energy is trivial when compared to other forms of energy. This project attempts to increase the value of wind energy by allowing wind speed prediction to inform energy providers the basis for calculating how much power they are able to generate that day. In this project, wind speed prediction is achieved via the implementation of an Artificial Neural Network. This network is trained using the Back-Propagation Method, with weights updated via the Steepest Descent Method. The performance of the Artificial Neural Network is first tested with the XOR function, then tested with wind data available online. According to the results of the XOR test, the network was successfully able to achieve pattern recognition. Two Neural Network models are compared: one without historic data and another with historic data. Computer simulation results show that the Neural Network model with historic data can predict wind speed more accurately.
CHAPTER I: INTRODUCTION

1.1 PROBLEM STATEMENT:

Increased concentrations of greenhouse gasses cause the Earth’s atmosphere to heat up [1]. Different greenhouse gasses trap heat, thus increasing the Earth’s temperature. The most abundant greenhouse gas — carbon dioxide, is human emitted and created primarily through the burning of fossil fuels via factories, car movement, and electricity production [1]. Currently, there are several renewable forms of energy generation which do not hurt the environment. These include wind power, solar power, geothermal electricity production, and hydroelectric power. This project focuses on wind energy.

Energy is considered renewable if its source is something that can be replenished. Wind is considered a renewable energy because solar heat creates wind, thus wind can replenish itself via the sun. Figure 1 below illustrates the current triviality of renewable energy compared to standard forms of energy. Natural gas, oil, and coal remain the most widely used fuels to generate energy. These resources are considered reliable because they burn quickly allowing them to produce energy faster and satisfy the consumer’s demand for energy.

![Percentage of renewable energy](image)

**FIGURE 1: RENEWABLE ENERGY AS PART OF TOTAL PRIMARY ENERGY CONSUMPTION, USA 2010 ([2])**

Wind’s variability limits its usage as a mainstream source of energy. Accurate predictions of wind speed would grant a utility company the ability to determine how much energy it is capable of generating. Knowing how much energy the turbine would generate increases the reliability of wind energy. If wind energy were reliable, it would become a primary energy source.
1.2 Solution:

An Artificial Neural Network was chosen to predict wind speeds because of its ability to learn adaptively. Learning adaptively enables the network to learn unknown systems and to perform as expected, even with fault tolerances.

The project utilizes a supervised system, meaning that pairs of inputs and outputs are provided for the training process. Specifically, the Back-Propagation Method was used to train the network because it is based on modifying the network’s synaptic weights through an error signal. Each neuron propagates information and is interconnected via a system of weights. The back-propagation method works by sending both a backward signal and a forward signal. The forward signal propagates through the output of each neuron to the output of the network. The output of the network is then compared to the desired output, and the error is calculated. The backward signal propagates the error of the network back to the input and the weights are changed accordingly. This is why the system is supervised; the desired output is necessary to train the weights via an error signal. The weights themselves are updated by applying the Steepest Descent Method to minimize the mean square error.

The fact that the network is multilayered is what enables it to learn nonlinear inputs. The hidden neurons within the hidden layers map the input data to a new feature space, where they then become linearly separable and can thus be classified easier, leading to prediction. Figure 2 below exhibits a block diagram of this system. Note that the dashed line through the black box diagram of the Artificial Neural Network indicates that it changes over time.

![Block Diagram of Supervised Artificial Neural Network Using Back-Propagation](image)

FIGURE 2: BLOCK DIAGRAM OF SUPERVISED ARTIFICIAL NEURAL NETWORK USING BACK-PROPAGATION

The weights inside the neural network allow the system to learn. They were updated via a hyperbolic tangent activation function. A thorough discussion of the weights and how the network is updated can be found in Chapter III: Neural Network Background.
CHAPTER II: LITERATURE REVIEW

This section summarizes the literature used to design the Neural Network. Different papers were read, confirming that an Artificial Neural Network is the best method for simplistic short term wind speed prediction. Models analyzed were Persistence Models, Numeric Weather Prediction Models, models based on meteorological data, Time Series Models, Fuzzy Logic models and Artificial Neural Networks.

The article by Wu and Hong ([3]) analyzes the financial benefits of good forecasting, and states that an advanced forecasting technique is required. It also claims that accurate and reliable forecasting systems for wind power would increase the usage of wind energy. The article states that methods based on numeric weather prediction, AI technologies, and hybrid models should be considered. Also, characteristics of the local wind profile, climate condition, and terrain type should be taken into account as wind speed varies depending on its location.

The article also discusses the advantages and limitations of Persistence Models, Numeric Weather Prediction models, and Artificial Neural Networks in predicting wind speed. Persistence models are the simplest way to forecast wind. This method uses the assumption that the wind speed at time t+x is the same as it was at time t. Analogously, there is a high correlation between the present and future wind values. The problem however, is that the accuracy degrades with increasing the look ahead hour. The next model that the article reviews is the Numeric Weather Prediction Model. These models provide wind speed forecasts for a grid of surrounding points around the wind generators, thus the forecasts are given with spatial resolution. These advanced models have the potential to improve the modeling of wind flow, however they require complex math and are typically run on super computers. Meteorological models with high resolution are accurate but require high computation time to produce forecasts and thus do not update their outputs frequently. Next, the article discusses neural network models. It states that these models are based on historical values and are easy to model while still capable of providing timely predictions. The advantage of Artificial Neural Networks is that they learn the relationship between the inputs and outputs by a non-statistical approach; they do not require any predefined mathematical models. However, these models can be limited to short term prediction.

Out of Persistence Models, Numeric Weather Prediction Models, and Artificial Neural Networks, Artificial Neural Networks are superior due to their simplicity and acceptable short term predictions. ([3])

Next, the paper by Sreelakshmi and Ramakanthkumar ([4]) discusses the limitations of several models used for short term wind speed prediction such as: models based on meteorological data, Time Series Models, and Fuzzy Logic Models. Models based on meteorological data are limited in their fast acquisition of data and also that they can’t change adaptively. These models depend on a meteorologist and rely on interpolating data from various sources. Time series models are based on historical wind data and other statistical methods. The Auto-Regressive Moving Average model is a benchmark of this type of model. The problem with time series models is that the model requires equations that depend on lots of information. A fuzzy model, a type of probabilistic logic model, was trained using genetic algorithms and determined to be more efficient than ARMA models, however less efficient than Artificial Neural Networks. It was determined that Artificial Neural Networks are the best for wind speed prediction because they can handle large data sets, utilize nonlinear models, and have the ability to adapt. The article also suggests several variables to use, and provides data from a similar experiment. The result claims that Artificial Neural Networks are the best method of predicting wind speed because of their simplicity and adaptability ([4]).

The steps in implementing their experiment of short term wind speed prediction were described: data acquisition, data conversion and normalization, statistical analysis, designing the neural network, training, and testing. The statistical analysis step determines the dependency between each of the values via a ranking system. They designed their neural network with steps such as: weight initialization, propagating inputs forward, propagating error backward, and reaching a terminating condition ([4]).
This article by Bhaskar and Singh ([5]) discusses the importance of accurately predicting wind speed as it allows power companies to compensate for operational problems such as maintaining system frequency, power balance, voltage support and quality factor. It states that the current methods of wind forecasting employ numerical weather predictions. These predictions use meteorological information to predict future data at given locations and altitudes. This model is too complex to implement, takes more time for execution, and is site dependent. Also, the error increases as farther future predictions are made. The article tests how using a two stage network affects the look-ahead hour and error. The two stages of the system are: 1. An adaptive wavelet neural network which is used to regress each decomposed signal 2. A feed-forward neural network used for nonlinear mapping between wind speed and wind power output. This conversion transforms the forecasted wind speed into a wind power prediction. In comparing how the two stages can individually predict wind speed, it was determined that the wavelet model has better approximation and faster training ability compared to the feed forward neural network. Comparing the two stages together confirms that the two stage approach in the article is more effective than current methods of wind forecasting.

Although designing a Neural Network is a suitable approach to predicting wind speeds, a better method is to use wavelets. However, better than both individual methods is using both a wavelet neural network cascaded with a feed forward neural network. This information should be considered when determining future work for this project.
CHAPTER III: NEURAL NETWORK BACKGROUND

This chapter summarizes the theory that goes behind all of the design choices used in implementing this network.

When designing an Artificial Neural Network, the number of neurons in the input layer and output layer depend on the number of inputs and outputs the system receives. The number of hidden layers, and neurons within each hidden layer, are determined via trial and error. Hidden layers are necessary to realize nonlinear patterns. A network that is too large can spend unnecessary time computing, while a network that is too small may never converge to an answer.

The number of weights in the system depends on the number of input neurons, hidden layers, and output neurons. More neurons in the system allow for a more accurate prediction but require increased prediction time as more weights are needed to update each iteration. As the number of neurons increase, the time needed for prediction increases exponentially due to the weights.

Once the number of neurons, and thus weights, are determined, back propagation is used to update the weights. The first stage in the back-propagation method is the forward signal. The weights within the network are initialized to random numbers between -1 and +1. Each neuron propagates forward the previous neuron’s output with a corresponding weight value. The inputs and weights are summed, input into a nonlinear activation function, and then output to the next neuron. The inputs (X) and weights (W) are summed using the equation:

\[ V = W^T X. \] (1)

An activation function then assigns each neuron’s output in the range of -1 to 1. The hyperbolic tangent activation function is used, which has the form:

\[ y(n) = a \frac{e^{bv} - e^{-bv}}{e^{bv} + e^{-bv}} \] (2)

Where a represents the output range, b represents the steepness of the curve, and V represents the input to the neuron. In this function, a and b are set to 1, as is standard. The shape of this function is illustrated in Figure 3 below.

![FIGURE 3: SHAPE OF HYPERBOLIC TANGENT FUNCTION](image)
Increasing $b$ makes the hyperbolic tangent function behave more like the hard limiting function by producing either a -1 or 1 instead of producing a range of values between -1 and 1. Decreasing $b$ makes the hyperbolic tangent function behave linearly, mitigating the purpose of multiple layers. The hyperbolic tangent function was chosen because there is a readily available function in Matlab which increases the design simplicity. This function also exhibits a balance between its nonlinear parts and linear parts, and is easily differentiable, implying its continuity.

Before the network is tested, it must be trained to determine the best set of weights. For each input/output pair, the neurons feed forward the information, and the error signal is fed back which is used to update weights. The weights deliver the information from each neuron and are strengthened depending on the number of iterations they remain active. Each iteration $n$, the weights change by:

$$w(n+1) = w(n) + \gamma \Delta w_{ji}$$  \hspace{1cm} (3)

$$\Delta w_{ji} = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$$  \hspace{1cm} (4)

Here, the weight is represented as being connected from neuron $i$ to $j$, $\eta$ represents the learning rate and $\gamma$ represents the momentum term. If $\eta$ is too small, the weights will yield accurate results, however the time required to train the network is long. This means that the weights will change by too small of an amount each iteration. If $\eta$ is too large, the solution may converge quickly but oscillate. Note that the momentum term is not a part of the standard back-propagation algorithm. It is a modification which influences the convergence speed and resists the network from optimizing to a local minimum rather than a global minimum. A term that is too large however, can cause instability. A small learning rate was chosen to prevent optimizing toward the local minimum rather than the global minimum. For this project, $\eta$ and $\gamma$ were arbitrarily chosen to be 0.1 and 0.02 respectively.

In this project, the weights are updated using the Steepest Descent Method. This is a simple optimization technique that functions by minimizing an objective function, or adjusting unknown parameters in the direction of steepest descent. In this case, the unknown parameters are the values of the weights. The objective function must be continuously differentiable with unknown parameters. In this case, the objective function is the square error function, which can be seen in Equations (5) and (6) below.

$$error = d - y$$  \hspace{1cm} (5)

$$\mathcal{E} = \frac{1}{2} error^2$$  \hspace{1cm} (6)

In Equation (5), $d$ represents the desired value while $y$ represents the output of the neuron. Minimizing this objective function represents minimizing the instantaneous error of each neuron. This type of training is called on-line or incremental training. Here, the weights are updated each time there is a new input/output pair. In a separate type of training called batch training, the weights are updated after all of the training samples have been seen by the network. The advantages of online training are that the weights update faster and they can track small changes in the inputs. One limitation however, is that a noisy signal could cause the weights to change too much. The advantages of batch training include a tolerance to noise and faster altogether convergence. In most cases however, online learning is used because it is simpler to implement and can provide effective solutions for pattern classification problems ([6]).
The negative sign in Equation (4) guarantees that the objective function always decreases. Figure 4 below demonstrates how the Steepest Descent Method would find the minimum value of weight.

![Figure 4: Steepest Descent Diagram](image)

If the weights are on the left side of the optimal middle value, the change in weights will increase as the slope is negative and the sign of \( \Delta w \) is negative. If the weights are on the right, the weights will decrease as the slope is positive and the sign of \( \Delta w \) is negative. After each iteration, the weights will train themselves to get closer to their optimal value. Depending on the neuron’s location within the network, a different formula is used to determine the weights with respect to the objective function. A demonstration of weight interconnectivity can be seen in Figure 5 below.

![Figure 5: Interconnectivity of Weights](image)

Where \( w_{ij} \) represents a connection from i to j. Equations (5) - (9) are the equations necessary to understand the output of the neural network in terms of the weight parameters:

\[
e = d - y
\]

\[
\varepsilon = \frac{1}{2} e^2
\]

\[
v_j = \sum_{i=1}^{n} w_{ij} y_i
\]

\[
\Phi(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}
\]

\[
\Phi' = \frac{4}{(e^v + e^{-v})^2} = (1 + y)(1 - y)
\]

Equation (9) illustrates the derivative of the activation function in terms of the output of the neural network: \( y \).
Below is an example of the weight derivation using the steepest descent method and the chain rule. For convenience, one example of chain rule is shown:

\[
\frac{\partial E}{\partial w_{11}^{(2)}} = \frac{\partial E}{\partial e} \ast \frac{\partial e}{\partial y_1^{(2)}} \ast \frac{\partial y_1^{(2)}}{\partial y_1^{(1)}} \ast \frac{\partial y_1^{(1)}}{\partial w_{11}^{(2)}}
\]

\[
\Delta w_{11}^{(2)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) y_1^{(1)}
\]

The chain rule was used to derive Equations (11) – (15) below.

\[
\frac{\partial E}{\partial w_{12}^{(2)}} = -e y_1^{(2)} [1 - y_1^{(2)}] y_2^{(1)}
\]

\[
\Delta w_{12}^{(2)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) y_2^{(1)}
\]

\[
\frac{\partial E}{\partial w_{11}^{(1)}} = -e y_1^{(2)} [1 - y_1^{(2)}] w_{11}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_1
\]

\[
\Delta w_{11}^{(1)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) w_{11}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_1
\]

\[
\frac{\partial E}{\partial w_{12}^{(1)}} = -e y_1^{(2)} [1 - y_1^{(2)}] w_{12}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_2
\]

\[
\Delta w_{12}^{(1)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) w_{12}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_2
\]

\[
\frac{\partial E}{\partial w_{21}^{(1)}} = -e y_1^{(2)} [1 - y_1^{(2)}] w_{21}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_1
\]

\[
\Delta w_{21}^{(1)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) w_{21}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_1
\]

\[
\frac{\partial E}{\partial w_{22}^{(1)}} = -e y_1^{(2)} [1 - y_1^{(2)}] w_{22}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_2
\]

\[
\Delta w_{22}^{(1)} = \eta (d - y_1^{(2)}) y_1^{(2)} (1 - y_1^{(2)}) w_{22}^{(2)} y_1^{(1)} [1 - y_1^{(1)}] x_2
\]

The weights are updated during the training phase of the algorithm, and fixed for use during the testing phase. Network performance is then analyzed using the testing results.
CHAPTER IV: TEST PLANS AND RESULTS

4.1 TEST PLANS

Once the neural network is designed, it will be tested to assess its performance. A classic benchmark test for networks is realizing the XOR gate. Once the network successfully achieves pattern recognition and functions within an appropriate error range, the network will be used to predict wind speed using wind data available online.

The network will be trained with wind data from the National Wind Technology Center. This is a renewable energy laboratory in Colorado which records data every minute and archives it in a public location on their website. The location of the data collector is seen in Figure 6 below.

![National Wind Technology Center](image)

FIGURE 6: DATA COLLECTOR FOR NATIONAL WIND TECHNOLOGY CENTER

Once on the website, there is a list of 66 different variables. Variables can be selected using the website’s interface and the output type labeled “Selected 1-Min Data (Zip Compressed)” was chosen to allow the selected variables to be downloaded as an excel file. This can be seen in Figures 7 and 8 on the next page.
Download the zip file to open raw data via excel. Matlab can easily read excel files, meaning the data can be fed into Matlab as inputs for the Neural Network.

Data was taken from October 27th 2013 at sensors 80 meters above the ground. Specifically, the data used in the network was recorded in minute intervals from approximately 12am-5pm. Wind at the highest level, 80m, was chosen because it is the fastest and most sporadic. This is also the typical height at which wind turbines are operational. The paper by Sreelakshmi and Ramakanthkumar ([4]) suggested using the variables: mean temperature (°C), humidity (g/Kg), wind gust (m/s), wind direction (deg), and barometric pressure (mbar) to determine the current wind speed (m/s). Training was comprised of 800 input/output pairs, while testing the network utilized 200 different input/output pairs.

This project can be implemented in 4 main steps. These steps are labeled in the code, located in Appendix E, for convenience.
A signal flow diagram is provided in Figure 9 below to help in understanding the code.

Before using the Artificial Neural Network, constants must be defined at the top of the code. These constants include the learning rate, the maximum number of iterations, the momentum term, and the stopping condition.

In the first step, the input values are presented and processed. Each time information is given to the network, the network assess the input/output size to determine the number of neurons within the input and output layers. The input data is also used to resolve the minimum and maximum values for both the training and testing data via a built-in Matlab function. Lastly, the data is normalized to increase the efficiency of the network. Working with a small range of values is computationally easier than working with a large range of values. The equations used for normalization and denormalization are Equations (16) and (17) below.

\[
X_N = \frac{X_D - X_{\min}}{X_{\max} - X_{\min}}
\]

\[
X_D = X_N \times (X_{\max} - X_{\min}) + X_{\min}
\]

In the equations above, \(X_N\) represents the normalized value and \(X_D\) represents the denormalized value.

In Step 2, the arrays in the code are initialized. The number of layers in the network depends on the number of hidden layers desired. In this case, one hidden layer was used for simplicity. The weight arrays are filled with randomized numbers in the range of -1 to 1. There is one weight matrix that interconnects each neuron pair. Thus, for a three layered network, there will be two weight matrices. The input of each neuron, called “net” in my code, is
initialized with ones. There is one “net” for each hidden layer. The activation function array, which is used to represent the output of each neuron, is set depending on the number of hidden layers as well.

Step 3 involves the back propagation method, which is repeated until the number of iterations reaches the maximum number of iterations, or until the stop criterion is met. In this case, the mean squared error is the stop criterion and is defined at the top of the code before Step 1. Note that only one pattern is trained at a time during the back-propagation phase. During the feed forward phase, the nets (or inputs to each neuron) are modified by a weight value, with initialized random values. These nets are subject to the activation function, which produces the neuron output value and is multiplied by a new weight value. Once the information has traveled through all of the neurons and reaches the end of the network, the data is compared to the desired output. The difference is calculated and becomes an error signal. This begins the back propagation phase. The weights are then adjusted using the error signal via the method of Steepest Gradient Descent. Reference Chapter III: Neural Network Background for details on this method. Then, the number of iterations increases by one and the mean squared error is calculated using the error signal to determine if the network can stop training. If the stop criterion hasn’t been met, the feed forward and back propagation phase are repeated.

Once the mean square error reaches a sufficient value or the maximum number of iterations has been met, the weights are fixed and used for testing data. Step 4 includes applying the fixed weights from training to the weights for testing, and then testing the new data. The test data is a smaller set of completely new data compared to the training data.

Two different topologies of neural networks are then compared. The first model describes the Neural Network with no historic data. A block diagram of this can be seen in Figure 10 below.
The second model utilizes a topology that includes historic data, meaning that the network now achieves time series predictions. The delayed output feedback is what makes the network able to perform time series predictions.

Applying the historic input should generally improve the results.

4.2 RESULTS

This section provides the results of the two tests.

XOR Data

First an XOR function was trained and tested as XOR is a simple problem to assess network performance. The constants in Table 1 below demonstrate the conditions for this test.

<table>
<thead>
<tr>
<th>Max Iterations</th>
<th>Momentum Term</th>
<th>Learning Rate</th>
<th>MSE Wanted</th>
<th>RMSE Wanted</th>
<th>Hidden Layer 1 Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.02</td>
<td>0.1</td>
<td>6.25x10^-4</td>
<td>0.025</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 12 below demonstrates how the root mean square error improved as the number of iterations increased. This demonstrates the network is learning. The graph displays the error between the denormalized network output value and the denormalized desired value.
Figures 13 and 14 below demonstrate the network’s training and testing results.

**FIGURE 12: DENORMALIZED RMSE OF TRAINING RESULTS AS A FUNCTION OF THE NUMBER OF ITERATIONS**

**FIGURE 13: DENORMALIZED TRAINING RESULTS**
The results of this run are shown in Table 2 below.

### TABLE 2: PERFORMANCE METRICS DETERMINED FROM THE XOR GATE

<table>
<thead>
<tr>
<th>Iterations</th>
<th>MSE Trained</th>
<th>MSE Tested</th>
<th>RMSE Trained</th>
<th>RMSE Tested</th>
<th>Time Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>789</td>
<td>6.24x10⁻⁴</td>
<td>6.24x10⁻⁴</td>
<td>0.0250</td>
<td>0.0250</td>
<td>0.2656 seconds</td>
</tr>
</tbody>
</table>

Table 3 below compares the data from the test run to the expected data in a numerical form for convenience.

### TABLE 3: COMPARISON OF EXPECTED VALUES AND NETWORK OUTPUT VALUES

<table>
<thead>
<tr>
<th></th>
<th>EXPECTED OUTPUT</th>
<th>NETWORK OUTPUT (TESTING)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.9787</td>
<td>0.9787</td>
</tr>
<tr>
<td>1</td>
<td>0.9655</td>
<td>0.9655</td>
</tr>
<tr>
<td>0</td>
<td>0.0292</td>
<td>0.0292</td>
</tr>
</tbody>
</table>

From the results in Table 3 above, it is clear that the network performed well. The optimum RMSE was set to be 0.025, which the network achieved in under 1000 iterations. This demonstrates the design of this network is effective in producing accurate results.
Wind Data- No Historic Input

After it was established that the network could achieve pattern classification within an appropriate error range, wind data was presented to the network to be trained and tested. The conditions for the training and testing can be seen in Table 4 below.

<table>
<thead>
<tr>
<th>TABLE 4: CONDITIONS FOR WIND TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Iterations</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>1000</td>
</tr>
</tbody>
</table>

Figure 15 below presents the RMSE per iteration plot for training. Although the error oscillates, the general trend implies that the error decreases which indicates learning.

FIGURE 15: DENORMALIZED RMSE OF TRAINING RESULTS AS A FUNCTION OF THE NUMBER OF ITERATIONS
Figure 16 below portrays all 800 training results.

**FIGURE 16: DENORMALIZED TRAINING RESULTS FOR WIND**

Figure 16 is broken down into segments of 200 samples in Figures 16(A)-16(D) below for convenience.

**FIGURE 16(A): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 0-200**
FIGURE 16(B): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 200–400

FIGURE 16(C): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 400–600
FIGURE 16(D): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 600-800

Testing results can be seen in Figure 17 below.

FIGURE 17: DENORMALIZED TESTING RESULTS FOR WIND
The results from the run can be seen in Table 5 below.

TABLE 5: PERFORMANCE METRICS DETERMINED FROM THE WIND DATA

<table>
<thead>
<tr>
<th>Iterations</th>
<th>MSE Trained</th>
<th>MSE Tested</th>
<th>RMSE Trained</th>
<th>RMSE Tested</th>
<th>Time Elapsed (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.3342</td>
<td>0.7913</td>
<td>0.5781</td>
<td>0.8895</td>
<td>83.89</td>
</tr>
</tbody>
</table>

These results should be compared to the network with the historic inputs to analyze how well the network performed. It can be observed however, that the training results are much better than the testing results.

**Wind Data- Historic Input**

For comparison, wind data with a historic input was applied to the network. The conditions for the training and testing can be seen in Table 6 below.

TABLE 6: CONDITIONS FOR HISTORIC WIND TEST

<table>
<thead>
<tr>
<th>Max Iterations</th>
<th>Momentum Term</th>
<th>Learning Rate</th>
<th>MSE Wanted</th>
<th>RMSE Wanted</th>
<th>Hidden Layer 1 Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.02</td>
<td>0.1</td>
<td>6.25x10^-4</td>
<td>0.025</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 18 below demonstrates the network’s learning as the number of iterations increases. It is apparent that the network makes mistakes in learning, apparent by the spikes in RMSE, but seems to resolve the error and gradually decrease.

![FIGURE 18: DENORMALIZED RMSE OF TRAINING RESULTS AS A FUNCTION OF THE NUMBER OF ITERATIONS](image-url)
Figure 19 below portrays all 800 training results.

**FIGURE 19**: DENORMALIZED TRAINING RESULTS FOR WIND WITH HISTORIC INPUT

For convenience, the 800 samples were broken down into groups of 200 samples, shown in Figures 19(A)-19(D) below.

**FIGURE 19(A)**: DENORMALIZED TRAINING RESULTS BRAKDOWN FROM 0-200
FIGURE 19(B): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 200-400

FIGURE 19(C): DENORMALIZED TRAINING RESULTS BREAKDOWN FROM 400-600
The testing results can be seen in Figure 20 below.

FIGURE 20: DENORMALIZED TESTING RESULTS FOR WIND DATA WITH HISTORIC INPUT
The results from this run are shown in Table 7.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>MSE Trained</th>
<th>MSE Tested</th>
<th>RMSE Trained</th>
<th>RMSE Tested</th>
<th>Time Elapsed (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.3771</td>
<td>0.3054</td>
<td>0.6140</td>
<td>0.5526</td>
<td>87.56</td>
</tr>
</tbody>
</table>

Observe that the testing data is better than the training data.
CHAPTER V: CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

It was determined from the XOR results that the Neural Network performance is acceptable under simple conditions. Next, the wind data with no historic input was applied to the network and compared to the wind data with historic input. This comparison can be seen in Table 8 below.

<table>
<thead>
<tr>
<th>Wind-No Historic Data</th>
<th>Iterations</th>
<th>MSE Trained</th>
<th>MSE Tested</th>
<th>RMSE Trained</th>
<th>RMSE Tested</th>
<th>Time Elapsed (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.3342</td>
<td>0.7913</td>
<td>0.5781</td>
<td>0.8895</td>
<td></td>
<td>83.89</td>
</tr>
</tbody>
</table>

| Wind-Historic Data    | 1000       | 0.3771      | 0.3054     | 0.6140       | 0.5526      | 87.56                  |

It is clear from these runs that the testing results of the Neural Network trained with the historic data outperformed the testing results of the Neural Network trained with no historic data. This is expected as generally a network with historic data should improve the results.

What should be noted is that the training results of the Neural Network trained with no historic data is better than the Neural Network trained with historic data. This could be an indication of overtraining. This means that the training results are good because the network is given the input and output data, and can change the weights accordingly. When provided with testing data, the network is unable to interpolate between the data points, providing a poor RMSE for testing.

Theory dictates that historic inputs should reduce the error of the network. According to this small subset of data, this theory hold true. To test this accurately, more data would be required and statistics would need to be used to illustrate whether or not this holds true for larger sets of data run more often.

5.2 FUTURE WORKS

In the future, more concrete conclusions can be drawn by testing the network’s performance when subject to a larger data set. A set of 80,000 samples could be trained, and 20,000 samples could be tested. This set of data would allow more information in determining why the Neural Network trained with no historic inputs had a better training RMSE than the Neural Network with historic inputs. Analyzing the results from this data could indicate whether overtraining occurred.

Next, the data should be run 100 times, each with newly initialized weight values and the same data set, to determine the Neural Network’s consistency in producing accurate results. The minimum, maximum, mode, and standard deviation of the denormalized testing RMSE could be extracted and compared. Comparing the data from the numerous runs would stand as a better tool in determining whether Neural Networks with historic inputs provide more accurate results than Neural Networks trained with no historic inputs.

The next potential improvement would be to test the effect of the activation function. According to the article written by Yu and Pongponsri, ([7]) high frequency noise can be removed from signals by using a wavelet activation function. The traditional approach in removing high frequency noise is to implement a low pass filter. A
common problem with this method is determining the cut-off frequency. In this article, a wavelet activation function was used to predict the ECG signal and to reduce the noise. The network was trained using a hybrid algorithm with Adaptive Diversity Learning Particle Swarm Optimization and gradient descent optimization. It was determined that the approach was promising for ECG signal modeling and noise reduction. Implementing a wavelet activation function into my network trained via the back propagation method is a potential method of improving the prediction error by eliminating noise. As wind is highly variable with lots of noise, using a wavelet activation function to average out the signal should be implemented in future works.

Wavelet activation functions are specifically used for time series predictions, function approximations and fault diagnosis. Through a series of superimposed wavelet functions, the inherent noise within the data is removed so it can be generalized easier. Relating the concept of wavelets to topics covered in DSP classes, a wavelet behaves similarly to a sinc function. Shifting, scaling, and superimposing sinc functions allows for various signals to be recovered. Figure 21 below illustrates how a sinc function can be manipulated to match an unknown signal.

Using this theory and the concept of resonance, wavelets are used to generalize unknown patterns. According to ([7]), the mother wavelet to be used is:

\[ \Psi(x) = -x e^{-\frac{1}{2}x^2} \]  

Whose shape can be recognized in Figure 22 below:

It is expected that the results will be improved by switching the activation function. The manner in which the weights are updated is critical to the network’s performance. As seen in Chapter III: Neural Network Background,
the weights are updated depending on the output produced by the activation functions. The weights make a neural network adaptive, thus the weights in the neural network define the system’s performance.
REFERENCES


APPENDICES

APPENDIX A: ANALYSIS OF SENIOR PROJECT DESIGN

Project Title: Artificial Neural Network Used for the Prediction of Wind

Student’s Name: Aly Lodge
Student’s Signature: Aly Lodge

Advisors Name: Dr. Helen Yu
Advisor’s Initials:

1. SUMMARY OF FUNCTIONAL REQUIREMENTS

Upon the project’s completion, the project will receive inputs and outputs and autonomously create a model that determines appropriate weights. Project completeness is determined when the model can accept nonlinear inputs and predict an output with acceptable error.

Refer to Appendix B for a complete description of requirements and specifications.

2. PRIMARY CONSTRAINTS

Describe Significant Challenges or Difficulties Associated with your Project or Implementation.

The main problem with this project was insuring its correct functioning. During 1 quarter of design, I assumed the code was working correctly. I was providing 80 training points and 20 test points to analyze my network’s performance. Unfortunately, when I increased the amount of data I provided to my network, the output of the network flat-lined at the maximum/minimum value of the data. For one entire quarter I thought the network was functioning and was well designed. However, when I changed the length of the data set I realized I had designed it poorly and it would only function for a small data set. This is a common problem when coding. I did not properly test my network properly during my design stage.

What Made your Project Difficult? What Parameters or Specifications Limited your Options or Directed the Approach?

The main problem with this project was determining what output from my neural network was acceptable. Each time the code ran, the weights were initialized to random values. Thus, some runs of the network yielded ‘poor’ predictions, and some runs yielded ‘good’ predictions. Determining the acceptable predictions was challenging.

3. ECONOMIC

What Economic Impacts Result?

In terms of financial capital, the user of the program will need to purchase Matlab. Matlab requires a 64 or 32-bit Windows operating system, Linux 64-bit operating system, or Mac OSX 64-bit operating system [8]. With respect to manufactured or real capital, if the user does not have this operating system capability, he or she will need to upgrade their system. Lastly, in terms of natural capital, the software uses weather information to predict wind speed. The weather variables: mean temperature, humidity, wind gust, wind direction, barometric pressure, and current wind speed are all directly related to wind speed prediction.
When and Where Do Costs and Benefits Accrue Throughout the Product’s Lifecycle?
The lifecycle for this product, in terms of the designer, starts with a one-time fee for Matlab. The other charges would accrue through electricity consumption from the computer usage.

What Inputs does the Experiment Require? How much does the project cost?
The only input that the program requires is weather information. Raw weather data is available from the National Wind Technology Center for free. This facility compiled and archived data every 60 seconds for 66 different variables starting from 1966 and continues to this day [9]. The Total Cost Table from Appendix C is re-created below for convenience.

<table>
<thead>
<tr>
<th>Utility</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab (Student Edition)</td>
<td>$100.00</td>
</tr>
<tr>
<td>National Wind Technology Center Data</td>
<td>Free</td>
</tr>
<tr>
<td>Labor (@rate of $20.00/hr, estimated)</td>
<td>$4,000.00</td>
</tr>
<tr>
<td>Total</td>
<td>$4,100.00</td>
</tr>
</tbody>
</table>

Project Timing
Please refer to the Gantt Chart in Appendix D: Figures 24-26 and Tables 12-14 for an estimated overview of project timing.

4. Manufacturing on a Commercial Basis
This portion of the senior project analysis assumes that the code will be sold. Potential buyers of this software would include utility companies which could use predicted wind speed to learn their maximum energy generation, or private wind turbine owners who generate their own power and thereby benefit from learning the predicted wind speed as well.

Key Terms:
Variable costs=n x cost/unit
Fixed costs=manufacturing line
Total costs=fixed + variable costs
Revenue=n x sales price/unit

This project requires a one-time student fee of $100.00 for Matlab. To distribute the product, a monthly flat fee for maintaining a website would be necessary. Typical web hosting costs $10.00. With the software available online, the cost/unit (variable cost) is negligible since one fee is required for the entire month, no matter how many downloads have occurred. Figure 23 below illustrates a break-even analysis.
The break-even point for this project is when 21 units have been sold. At 21 units, the revenue would be $4200.00, and the labor costs would remain $4100.00 according to Table 11 in Appendix C. It is clear from Figure 23 that any unit sold after the 21st unit will be a profit, if the product sells for $200.00.

5. ENVIRONMENTAL

Environmental Impacts with Manufacturing or Use
The environmental impacts regarding this software are positive. The problem that this code attempts to solve is making wind energy more reliable. Wind energy generates inexpensive electricity, approximately $0.05/kWh according to Wind Energy America, and in an economic competition with fossil fuel generated electricity, it would win [10]. This means fossil fuel emissions would decrease, thus, the carbon footprint would be reduced. The reduction of these two facets would help the environment. Note, though, that continuously running a software program requires the computer to stay on and thus consume electricity, however minimal.

Natural Resources or Ecosystems the Project uses Indirectly and Directly
As mentioned in the above paragraph, the continuous use of a computer requires electric consumption. Hopefully, a wind turbine company generating electricity would run the plant on wind generated electricity. This would mean that the electricity necessary to predicting wind speed would harm the environment compared to electricity generated by other means.

As the purpose of the project is to predict wind speed, wind is directly related to the project. The project merely observes the behavior of the wind; it does not alter it in any way.

Natural Resources or Ecosystems the Project Harms or Improves
If wind energy becomes one of the primary forms of electricity generation, then more turbines would need to be built. If a multitude of wind turbines were added to an area, the noise pollution in that area would increase [11]. This could cause disturbances for the people in the surrounding area as well as agitation of natural habitats. It should be noted that most of the noises coming from the turbines are not from the rotors spinning, but from the transformers humming. Another concern that some people have regarding wind turbines is that they may induce seizures [11]. Some people fear that the shadow the turbine casts over a house could rotate at such a speed to create a slow strobe effect – a strobe which could elicit a seizure. Keep in mind that the project theoretically decreases greenhouse gasses and reduces the effect of global warming; an important improvement to current environmental conditions.
**How does the Project Impact Other Species?**

One of the main complaints about wind turbines is how they impact birds. Note that the number of birds hit by wind turbine rotors is less than the number of birds killed by overhead high voltage power lines and vehicles [10]. As birds fly, they do not notice that the rotor is above them and can get hit. Some birds have been seen to perch atop the turbine since it is so high. These birds sit high and use the tower to watch over other prey, or they build nests atop the machine. According to Wind Energy America, turbines can be painted with UV reflection paint to make the birds more aware of the turbines [10].

**6. MANUFACTURABILITY**

If this project were to be sold as a product, due to the fact that it is software, the product could be easily distributed to other people. Whether the product is manufactured once or several hundred times makes no difference. The product could easily be circuitied by uploading it to a website, which easily allows users to download the project, or it could be easily distributed via disks.

**7. SUSTAINABILITY**

**Describe Issues or Challenges Associated with Maintaining the Program**

The nature of artificial intelligence means the program can correct itself in many different situations. Maintaining accuracy in many locations shouldn’t be a problem. Once the product works, it should be able to work in most situations because it will adapt to those weather conditions and remain self-sustaining.

**Describe How the Project Impacts the Sustainable Use of Resources**

The goal of the project is to promote the use of renewable energy. Wind energy is created due to uneven solar heating hitting the earth’s surface. This means once the sun’s radiation stops hitting the earth, wind power won’t be considered a renewable energy alternative. It is predicted however, that the sun’s energy will hit the earth for a very long time, and once solar heat stops hitting the earth, there will be larger problems other than worrying about the loss of wind energy. As of today, people burn fossil fuel to generate electricity. It is a known fact that the supply of fossil fuels is rapidly decreasing. It is clear that promoting wind energy is much more sustainable than relying on oil, the predominant generator of electricity.

**Describe Upgrades that Could Improve the Project**

Upgrades could be released to improve system efficiency in terms of speed or to increase the look ahead hour. The look ahead hour refers to how long into the future the network can predict. Also, the time the network takes to compute a prediction can be decreased, or the accuracy of the wind speed itself can be increased. The program’s performance can be increased by pruning the number of hidden neurons, and incorporating a wavelet activation function to reduce the incoming noise. Another potential upgrade could be to utilize different weather variables.

**Describe any Issues with Upgrading the Design**

There are many different upgrades that can be applied to the neural network. The only problem lies in determining which upgrades will be most helpful in determining wind speed. An upgrade to software requires only changing the lines of code, so the upgrade itself can be easily implemented.

**8. ETHICAL**

According to the IEEE Code of Ethics, I believe this project fulfills plank 1. Plank 1 states that one must “accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or environment.” By undertaking a senior project that focuses on promoting wind energy instead of other forms of energy, I accept responsibility and acknowledge the current problem with electricity generation. Current electricity generation increases the effects of global warming, which causes harm to different species and ecosystems alike. It is imperative to reduce greenhouse emissions, and
strengthening the appeal of wind energy is one such solution. Focusing more on wind energy helps both humans and the environment. This type of ideology mirrors that of utilitarianism. Utilitarianism suggests that the proper course of action is one that creates more happiness than suffering. In terms of economics, promoting renewable energy decreases the cost of electricity. In some places, people can’t afford to pay for heating or cooling during extreme weather conditions. Lowering the cost of electricity in most areas will empower people, while decreasing fossil fuel emissions benefits life on Earth.

9. Health and Safety
Although the software itself wouldn’t hurt anyone, some of the effects it yields can be dangerous. With an increase in demand for wind turbines, human injury might occur. Typical turbine heights can range between 164 feet to 328 feet [10]. Building these turbines require a large space and careful attention. With more turbines in operation, the probability of maintenance would increase as well. Sending a person up a ladder hundreds of feet in the air to work on a generator has serious safety risks.

10. Social and Political
Social and Political Issues Associated with Design, Manufacture, and Use
The design of the project itself requires only Matlab software and weather information. If I chose to sell this software, manufacturing the product would be simple. I would just need to upload my software to a website - this allows easy accessibility to utility companies and private investors. When the product is used, theoretically the cost for electricity in the area would decrease because that area is being powered by a cheap alternative fuel. The externality between the utility company and myself, via purchasing the software, would cause residents in an area to pay for reduced electricity costs.

Who Does the Project Impact? Who are the Direct and Indirect Stakeholders?
If my project were to be used for mainstream purposes, the direct stakeholders would be the utility companies. With knowledge of how much energy their plant is capable of producing that day, they can reliably supply power to the grid. The objective of this project is to increase the reliability of wind turbines. The indirect stakeholders for this project are the general public. They would not only receive electricity for lower costs, but the decrease in fossil fuel emissions helps everyone. Stakeholders that would suffer from this project are other utility companies such as oil refineries, coal burning plants and natural gas plants. These utility companies provide the most electricity, according to data in 2010 [2].

11. Development
I wanted my senior project to revolve around some type of renewable energy issue. My first introduction to renewable energy was in the class: Sustainable Electric Energy Conversion. In this class, we spent time understanding how turbines worked and how they generated electricity. This basic understanding of wind turbines allowed me to determine that predicting wind speed could result in their increased usage. Once I knew that I needed to predict wind speed, I decided upon implementing an Artificial Neural Network. The class: Computational Intelligence, provided me with lots of knowledge about Artificial Neural Networks. To implement my network, I had to grow in my Matlab skills so I could succeed in designing it.
APPENDIX B: SPECIFICATIONS AND REQUIREMENTS

A thought experiment was run to determine a potential customer’s needs for this type of project. Table 9 below summarizes the customer’s needs with their respective project requirements and specifications for convenience.

**TABLE 9: ARTIFICIAL NEURAL NETWORK USED FOR THE PREDICTION OF WIND REQUIREMENTS AND SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Marketing Requirements</th>
<th>Engineering Specifications</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Six wind input variables: mean temperature (°C), humidity (g/Kg), wind gust (m/s), wind direction (deg), barometric pressure (mbar), current wind speed (m/s) ([3]) for wind tests</td>
<td>Information provided to the model yields more accurate predictions. Research from “Neural Networks for Short Term Wind Speed Prediction” recommends them ([3]).</td>
</tr>
<tr>
<td>1</td>
<td>Model outputs wind speed in meters/second</td>
<td>The goal of the project is to deliver an accurate wind speed prediction which utility companies can then use to determine the energy a wind turbine can generate.</td>
</tr>
<tr>
<td>2</td>
<td>Implement artificial neural network in Matlab</td>
<td>Matlab’s current math functions handle complex analysis [6]. One-time fee of $100.00 suggests easy availability for most people.</td>
</tr>
<tr>
<td>1, 4</td>
<td>System retrieves information from at least one weather source.</td>
<td>National Wind Technology center is the U.S. Department of Energy’s primary national laboratory for renewable energy and energy efficiency research and development. They compiled an archive of data every 60 seconds of 66 different variables from 1996 [7].</td>
</tr>
<tr>
<td>3</td>
<td>Maximum time per prediction is 5 minutes.</td>
<td>Predictions would be useless if they took too much time to compute. An arbitrary time, 5 minutes, was deemed appropriate for this project. This time parameter can be subject to future changes.</td>
</tr>
</tbody>
</table>

**Marketing Requirements**

1. Accuracy
2. Easy to use software
3. System predicts wind speed in a short time span
4. Weather variables derived from a reliable source

Note: The requirements and specifications table format derives from [12]. Chapter 3.
APPENDIX C: PARTS LISTS AND COSTS

This section provides an analysis for costs associated with parts and labor.

Labor Costs:

Although accurate estimations of cost are based on past experiences, the formula shown below helps estimate the costs and time needed to complete the project [12].

\[ t_e = \frac{t_a + 4t_m + t_p}{6} \]

Where \( t_a \)=most optimistic time, \( t_m \)=most realistic time and \( t_p \)=most pessimistic. In my case, I decided the units would be weeks.

\[ t_e = \frac{7 + 4(10) + 14}{6}, \ t_e \approx 10 \text{ weeks.} \]

Labor costs are tied to the length of the project and are often the most expensive. Using the estimated time derived, and charging $20.00/hour, predicting I’d work 4 hours a day, 5 days a week, the total labor cost becomes $4,000. The charge of $20.00/hour was used because of my knowledge of undergraduate friends working at internships who are getting paid $24.00/hour.

Parts Costs:

The only part I need for this project is Matlab in addition to access to weather conditions. This is tabulated in Table 10 below.

Note: No time needed to select, purchase, and ship parts due to software based project.

<table>
<thead>
<tr>
<th>Software</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab (Student Edition)</td>
<td>$100</td>
</tr>
<tr>
<td>National Wind Technology Center (weather</td>
<td>Free</td>
</tr>
<tr>
<td>conditions)</td>
<td></td>
</tr>
</tbody>
</table>

A total cost table is provided below in Table 11.

<table>
<thead>
<tr>
<th>Utility</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab (Student Edition)</td>
<td>$100.00</td>
</tr>
<tr>
<td>National Wind Technology Center</td>
<td>Free</td>
</tr>
<tr>
<td>Labor (@rate of $20.00/hr, estimated)</td>
<td>$4,000.00</td>
</tr>
<tr>
<td>Total:</td>
<td>$4,100.00</td>
</tr>
</tbody>
</table>
APPENDIX D: SCHEDULE – TIME ESTIMATES

This section provides an overview for the time allotted to completing this project.

Figure 24 portrays the planning stages of the project, achieved in my EE 460 class.

FIGURE 24: EE 460 GANTT CHART

Table 12 below shows Figure 24 in a tabulated manner for easy reference.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Number of Days</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chose Senior Project</td>
<td>11</td>
<td>Tue 1/1/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>Requirements and Specifications</td>
<td>5</td>
<td>Tue 1/15/13</td>
<td>Mon 1/21/13</td>
</tr>
<tr>
<td>Block Diagram</td>
<td>6</td>
<td>Mon 1/21/13</td>
<td>Mon 1/28/13</td>
</tr>
<tr>
<td>Literature Search</td>
<td>6</td>
<td>Mon 1/28/13</td>
<td>Mon 2/4/13</td>
</tr>
<tr>
<td>Abet Sr. Project Analysis</td>
<td>4</td>
<td>Wed 2/13/13</td>
<td>Mon 2/18/13</td>
</tr>
<tr>
<td>Report V1</td>
<td>9</td>
<td>Wed 2/13/13</td>
<td>Mon 2/25/13</td>
</tr>
<tr>
<td>Develop Functions</td>
<td>10</td>
<td>Mon 2/25/13</td>
<td>Fri 3/8/13</td>
</tr>
<tr>
<td>Report V2</td>
<td>16</td>
<td>Mon 2/25/13</td>
<td>Mon 3/18/13</td>
</tr>
</tbody>
</table>

Note: Function include Initializing Matrixes and Normalization

Figure 25 below shows the build process, achieved in my EE 463 class.

FIGURE 25: EE 463 GANTT CHART

Again, a tabulated form of Figure 25 is available as Table 13 below for reference.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Number of Days</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ANN against XOR data</td>
<td>5</td>
<td>Tue 4/2/13</td>
<td>Mon 4/8/13</td>
</tr>
<tr>
<td>Fix Errors</td>
<td>5</td>
<td>Mon 4/8/13</td>
<td>Fri 4/12/13</td>
</tr>
<tr>
<td>Retest against XOR</td>
<td>4</td>
<td>Wed 4/17/13</td>
<td>Mon 4/22/13</td>
</tr>
<tr>
<td>Test ANN with Wind Data</td>
<td>6</td>
<td>Mon 4/22/13</td>
<td>Mon 4/29/13</td>
</tr>
<tr>
<td>Fix Errors</td>
<td>6</td>
<td>Mon 4/29/13</td>
<td>Mon 5/6/13</td>
</tr>
<tr>
<td>Increase Data to System</td>
<td>10</td>
<td>Mon 5/6/13</td>
<td>Fri 5/17/13</td>
</tr>
<tr>
<td>Report V4</td>
<td>44</td>
<td>Tue 4/2/13</td>
<td>Fri 5/31/13</td>
</tr>
</tbody>
</table>

Note: Baseline tests will be run against XOR inputs. XOR is an arbitrary gate used to ensure that the Neural Network functions properly. Also observe that Report V4 will was updated throughout the entire quarter.
Figure 26 below shows the improvements implemented into the code and the final report writing completed in EE 464. Summer break accounts for the gap between work periods.

**FIGURE 26: EE 464 GANTT CHART**

Table 14 lists the events in Figure 26 for easy reference.

**TABLE 14: TABULATED GANTT CHART ESTIMATES FOR EE 464**

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Number of Days</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduce Historic Data to Network</td>
<td>6</td>
<td>Mon 9/30/13</td>
<td>Mon 10/7/13</td>
</tr>
<tr>
<td>Fix Errors</td>
<td>19</td>
<td>Mon 10/7/13</td>
<td>Thu 10/31/13</td>
</tr>
<tr>
<td>Compare Different Network Configurations</td>
<td>10</td>
<td>Thu 10/31/13</td>
<td>Wed 11/13/13</td>
</tr>
<tr>
<td>Report V5</td>
<td>46</td>
<td>Mon 9/30/13</td>
<td>Mon 12/2/13</td>
</tr>
</tbody>
</table>

Observe that Report V5, the final report, was updated throughout the entire quarter.

The Gantt Charts above show several iterations of the project as well as report.
APPENDIX E: PROJECT CODE

Neural Network Code-XOR Test

clc
clear
Max_iterations=1000;
mseWanted= 0.000625;
n=0.1; % learning rate
gamma=0.05; % momentum term
hidden_neurons=5;
mseHave=1;
iterations=0;
t=cputime;

%%% Step 1: Presenting/Normalizing Inputs and Outputs%%%

Unnorm_input = [0,0,0,1,1,0,1,1];
Unnorm_desired = [0; 1; 1; 0];

Unnorm_Testin=[0,0;0,1;1,0;1,1];
Unnorm_Testdesired=[0; 1; 1; 0];

[inputsPERinput,Num_inputs]=size(Unnorm_input);
[outputsPERoutput,Num_outputs]=size(Unnorm_desired);
[TestinputsPERinput,Num_Testinputs]=size(Unnorm_Testin);
[TestoutputsPERoutput,Num_Testoutputs]=size(Unnorm_Testdesired);

%Find Max/Min to help Normalize
Max_in=max([max(Unnorm_input);max(Unnorm_Testin)]);
Min_in=min([min(Unnorm_input);min(Unnorm_Testin)]);
Max_desired=max([max(Unnorm_desired);max(Unnorm_Testdesired)]);
Min_desired=min([min(Unnorm_desired);min(Unnorm_Testdesired)]);

% Normalize Data
[ Norm_in, Norm_testIn,Norm_desired,Norm_testDesired ] = Normalize2(
inputsPERinput,Num_inputs,outputsPERoutput,Num_outputs,TestinputsPERinput,
TestoutputsPERoutput,Num_inputs,Num_outputs,Num_Testinputs,
Num_Testoutputs,Unnorm_input,Unnorm_Testin,Unnorm_desired,Min_in,Max_in,Min_desired,
Max_desired,Unnorm_Testdesired);

%%% Step 2: Initializing Arrays%%%

pattern=inputsPERinput;

weight_hidden_input = randn(Num_inputs,hidden_neurons);

weight_hidden_output = randn(1,hidden_neurons); % because linear

while iterations < Max_iterations && mseHave >= mseWanted

%%% Train 1 pattern at a time
for j=1:pattern
num=round((rand*pattern)+0.5);
if num>pattern
num=pattern;
elseif num<1
num=1;
end

Train_in=Norm_in(num,:);
% keeps row

% Train output
Train_out=Norm_desired(num,:);
% keeps row

% Calculations
z=Train_in*weight_hidden_input;
net2=z*weight_hidden_output+
gamma*Train_out;
net=net2-n.*norm_grad;
Net=2.*net(:,:,1);
Net2=2.*net(:,:,2);

% Backpropagation
Delta_net=Net.*Net.2.*Norm_in(num,:);
Delta_net2=Net2.*Net2.2.*Norm_in(num,:);

% Update weights
weight_hidden_output=weight_hidden_output+
(n.*Delta_net+gamma*Delta_net2);

% Exploration
weight_hidden_input=weight_hidden_input*(1+n./2)

iterations=iterations+1;

end

t=cputime-t;
time=t

%%% Results%%%

mseHave=mse(Norm_in,Net2,Num_inputs,Num_outputs);
if mseHave >mseWanted
pause
end

end

time
Train_desired=Norm_desired(num,1); %1 number

%%Step 3: Back-Propagation Method%%

%feed forward
net=Train_in * weight_hidden_input;
Actfunc=tanh(net);
output=Actfunc*weight_hidden_output'; %1 number

error=Train_desired-output; %1 number

%Feedback
%output layer--linear
delta_hiddenOutput=n.*error.*Actfunc;

weight_hidden_output=weight_hidden_output+gamma.*delta_hiddenOutput;

%hidden layer
delta_hiddenInput=(error*(1-(Actfunc.^2))*weight_hidden_output')*(1-(Actfunc.^2))'*Train_in;

weight_hidden_input=weight_hidden_input+gamma.*delta_hiddenInput';

end

%Norm_desired
output = weight_hidden_output*tanh(Norm_in*weight_hidden_input)';
Unnorm_trainOutput=(output.*(Max_desired-Min_desired))+Min_desired;

%want denormalized RMSE plots
error=Unnorm_desired-Unnorm_trainOutput';
SumSquareError=sum(error.^2);
rmseHave=SumSquareError/inputsPERinput;
rmse(iterations+1)=rmseHave^0.5;
iterations=iterations+1;

end

%this is the unnormalized output of the TRAINING
errorTrain=Unnorm_desired-Unnorm_trainOutput';
SumSquareErrorTrain=sum(errorTrain.^2);
rmseTrain=rmseTrain^0.5
iterations
mseHave
TimeEllapsed=cputime-t

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Step 4:Use Fixed Weights for Testing Data%%%

%retrain network using old weight values and new testing values
netTest=Norm_testIn * weight_hidden_input;
Actfunc=tanh(net);
output=Actfunc*weight_hidden_output'; %1 number

%using the same weights, the TESTING values were put into the network
Norm_testIn
Unnorm_testOutput=(Norm_testIn.*(Max_desired-Min_desired))+Min_desired
outputTest = (weight_hidden_output*tanh(Unnorm_testOutput*weight_hidden_input)')'
errorTest=Unnorm_Testdesired-outputTest
SumSquareErrorTest = sum(errorTest.^2);
msHaveTest = SumSquareErrorTest / inputsPERinput;
rmseTest = msHaveTest^0.5

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Plotting%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%RMSE vs Iterations Training
figure(1)
start = 1;
stop = iterations;
plot(start:stop, rmse(start:stop))
xlabel('Number of Iterations')
ylabel('Root Mean Square Error of Denormalized Training Results')
title('Training Results')

%De-Normalized Plots (train)
figure(2)
start = 1;
stop = length(Unnorm_trainOutput);
plot(start:stop, Unnorm_trainOutput, '--r') % actual output from network
hold on
plot(start:stop, Unnorm_desired) % expected XOR data
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output', 'Expected Output', 'Location', 'EastOutside')

%De-Normalized Plots (test)
figure(3)
start = 1;
stop = length(outputTest);
plot(start:stop, outputTest, '--r') % actual output from network
hold on
plot(start:stop, Unnorm_Testdesired) % expected XOR
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Testing Results')
legend('Tested Actual Output', 'Expected Output', 'Location', 'EastOutside')

Neural Network Code-Wind

clc
clear
% Wind
Max_iterations = 2000;
k = 0;
msWanted = 0.000625;
n = 0.1; % learning rate
gamma = 0.02; % momentum term
msHave = 1;
iterations = 0;
t = cputime;

%%% Step 1: Presenting/Normalizing Inputs and Outputs %%%
% Unnorm_input = [0,0;0,1;1,0;1,1];
% Unnorm_desired = [0; 1; 1; 0];
% Unnorm Testin=[0,0;0,1;1,0;1,1];
filename='WindDataFall2.xlsx';

Unnorm_input=xlsread(filename,'E3:I802'); %800 train points
Unnorm_desired=xlsread(filename,'D3:D802');

Unnorm_Testin=xlsread(filename,'E803:I1002'); %200 test points
Unnorm_Testdesired=xlsread(filename,'D803:D1002');

[inputsPERinput,Num_inputs]=size(Unnorm_input);
[outputsPERoutput,Num_outputs]=size(Unnorm_desired);
[TestinputsPERinput,Num_Testinputs]=size(Unnorm_Testin);
[TestoutputsPERoutput,Num_Testoutputs]=size(Unnorm_Testdesired);

%Find Max/Min to help Normalize
Max_in=max([max(Unnorm_input);max(Unnorm_Testin)]);
Min_in=min([min(Unnorm_input);min(Unnorm_Testin)]);
Max_desired=max([max(Unnorm_desired);max(Unnorm_Testdesired)]);
Min_desired=min([min(Unnorm_desired);min(Unnorm_Testdesired)]);

[Norm_in,Norm_testIn,Norm_desired,Norm_testDesired]=Normalize2( inputsPERinput,outputsPERoutput,TestinputsPERinput, TestoutputsPERoutput,Num_inputs,Num_outputs,Num_Testinputs, Num_Testoutputs,Unnorm_input,Unnorm_Testin,Unnorm_desired,Min_in,Max_in,Min_desired,Max_desired,Unnorm_Testdesired);

%historic = ones(inputsPERinput,1); %uncomment for no historic

%%Step 2: Initializing Arrays%%

[Actfunc,net,layers,Num_layers,weights] = Setting_matrices10(Num_inputs,Num_outputs);
pattern=size(Norm_in,1);

while iterations < Max_iterations && mseHave >=mseWanted

%%Train 1 pattern at a time
for j=1:pattern
    num=round((rand*pattern)+0.5);
    if num>pattern
        num=pattern;
    elseif num<1
        num=1;
    end
    Norm_inPattern=Norm_in(num,:);
    %keeps row
    Train_in= [Norm_inPattern historic];  %uncomment for historic
    Train_desired=Norm_desired(num,1);  %number
    historic=Train_desired;  %uncomment for historic

    %%Step 3: Back-Propagation Method%%
    %feed forward
    net{1}=Train_in*weights{1};
    Actfunc{1}=tanh(net{1});
    %For each layer, feed forward the new net (which is the output of last activation
function *
    %weight function, then recalc activation for next result

    %start at output and move back. start at situation 1 in notes
output=Actfunc{end}*weights{end}'; %1 number
error=Train_desired-output; %1 number

delta=n.*error.*(1-(Actfunc{end}.^2)).*Actfunc{end};
weights{2}=weights{2} + gamma.*delta;

delta=((error*(1-(Actfunc{1}.^2))*weights{2}')*(1-(Actfunc{1}.^2))'\cdot\text{Train}_in)';
weights{1}=weights{1} + gamma.*delta;
end

%Norm_desired
outputfinal=zeros(outputsPERoutput,1);
outputfinal=(tanh([Norm_in ones(inputsPERinput,1)])\cdot\text{weights}_1'){\cdot}\text{weights}_2';
%take out ones(inputsPERinput,1) for no historic
%the weights are all trained for a Norm_in with an extra input (the
%historic input) so when you multiply the weights again, you need to
give Norm_in another column

Unnorm_trainOutput=(outputfinal.*(Max_desired-Min_desired))+Min_desired;
error=Unnorm_desired-Unnorm_trainOutput;
SumSquareError=sum(error.^2);
rmse(iterations+1)=mseHave^0.5;
iterations=iterations+1;
end

mseHave

%this is the denormalized output
iterations
errorTrain=Unnorm_desired-Unnorm_trainOutput;
SumSquareErrorTrain=sum(errorTrain.^2);
mseHaveTrained=SumSquareErrorTrain/outputsPERoutput
rmseTrain=mseHaveTrained^0.5

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Step 4:Use Fixed Weights for Testing Data%%%

%retrain network using old weight values and new testing values
%provide Norm_testIn with bias since dont want to give it historic
%inputs and need it to be the same dimensions for weights
Norm_testIn=[Norm_testIn ones(TestinputsPERinput, 1)];
netTest=Norm_testIn {\cdot} \text{weights}_1';
output=tanh(netTest)*weights{2}';

%using the same weights, the TESTING values were put into the network
Unnorm_testOutput=(output.*(Max_desired-Min_desired))+Min_desired;
```matlab
errorTest=Unnorm_Testdesired-Unnorm_testOutput;
SumSquareErrorTest=sum(errorTest.^2);
mseHaveTest=SumSquareErrorTest/TestoutputsPERoutput
rmseTest=mseHaveTest^0.5

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Plotting%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%RMSE vs Iterations Training
figure(1)
start=1;
stop=iterations;
plot(start:stop,rmse(start:stop))
xlabel('Number of Iterations')
ylabel('Root Mean Square Error of Denormalized Training Results')
title('Training Results')

%De-Normalized Plots (train)
figure(2)
start=1;
stop=length(Unnorm_trainOutput);
plot(start:stop,Unnorm_trainOutput, '--r') %actual output from network
hold on
plot(start:stop,Unnorm_desired)
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output','Expected Output', 'Location', 'EastOutside')

%break down big plot
%0-200
figure(3)
start=1;
stop=length(Unnorm_trainOutput)/4;
plot(start:stop,Unnorm_trainOutput(1:200), '--r') %actual output from network
hold on
plot(start:stop,Unnorm_desired(1:200))
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output','Expected Output', 'Location', 'EastOutside')

%200-400
figure(4)
start=length(Unnorm_trainOutput)/4;
stop=length(Unnorm_trainOutput)/2;
plot(start:stop,Unnorm_trainOutput(200:400), '--r') %actual output from network
hold on
plot(start:stop,Unnorm_desired(200:400))
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output','Expected Output', 'Location', 'EastOutside')

%400-600
figure(5)
start=length(Unnorm_trainOutput)/2;
stop=length(Unnorm_trainOutput)/(4/3);
plot(start:stop,Unnorm_trainOutput(400:600), '--r') %actual output from network
```

hold on
plot(start:stop,Unnorm_desired(400:600))
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output','Expected Output', 'Location', 'EastOutside')

%600-800
figure(6)
start=length(Unnorm_trainOutput)/(4/3);
stop=length(Unnorm_trainOutput);
plot(start:stop,Unnorm_trainOutput(400:600), '--r') %actual output from network
hold on
plot(start:stop,Unnorm_desired(400:600))
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Training Results')
legend('Trained Actual Output','Expected Output', 'Location', 'EastOutside')

%De-Normalized Plots (test)
figure(7)
start=1;
stop=length(Unnorm_testOutput);
plot(start:stop,Unnorm_testOutput, '--r') %actual output from network
hold on
plot(start:stop,Unnorm_Testdesired) 
hold off
xlabel('Samples')
ylabel('Magnitude')
title('Denormalized Testing Results')
legend('Tested Actual Output','Expected Output', 'Location', 'EastOutside')

TimeEllapsed=cputime - t

Normalization Function

function [ Norm_in, Norm_testIn,Norm_desired,Norm_testDesired ] = Normalize2(inputsPERinput,outputsPERoutput, TestinputsPERinput, TestoutputsPERoutput,Num_inputs,Num_outputs,Num_Testinputs, Num_Testoutputs,Unnorm_input,Unnorm_Testin,Unnorm_desired,Min_in,Max_in,Min_desired,Max_desired,Unnorm_Testdesired)
%Normalized number=(number_to_normalize - min_set)/(max_set-min_set)
%need to normalize all values within a column of data

for i=1:Num_inputs
    for j=1:inputsPERinput
        Norm_in(j,i)=(Unnorm_input(j,i)-Min_in(i))./(Max_in(i)-Min_in(i));
    end
end

for i=1:Num_Testinputs
    for j=1:TestinputsPERinput
        Norm_testIn(j,i)=(Unnorm_Testin(j,i)-Min_in(i))./(Max_in(i)-Min_in(i));
    end
end

for i=1:Num_outputs
    for j=1:outputsPERoutput
\[
\text{Norm\_desired}(j,i) = \frac{\text{Unnorm\_desired}(j,i) - \text{Min\_desired}}{\text{Max\_desired} - \text{Min\_desired}}; \\
\text{end}
\]

\[
\text{for } i=1:\text{Num\_Testoutputs} \\
\text{for } j=1:\text{Testoutputs}\_\text{PERoutput} \\
\text{Norm\_testDesired}(j,i) = \frac{\text{Unnorm\_Testdesired}(j,i) - \text{Min\_desired}}{\text{Max\_desired} - \text{Min\_desired}}; \\
\text{end}
\text{end}
\]

\textbf{Initializing Arrays}

\textit{Function [Actfunc,net,layers,Num\_layers,weights] = Setting\_matrices10(Num\_inputs,Num\_outputs)}

\text{Num\_inputs}=\text{Num\_inputs}+1; \\
% need to account for historic input \\

%%% Setting Layers %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% \\
% Vector specifying number of nodes at each later \\
layers=[\text{Num\_inputs};25;\text{Num\_outputs}]; \\
Num\_layers=length(layers); \\

%%% Setting Weights %%%%%%%%%%%%%%%%%%%%%%%%%%%%% \\
% Need one weight matrix to connect each layer \\
weights=cell(Num\_layers-1,1); \\
% this creates multiple blank arrays. \\
weights{1}=2.*rand(layers(1),layers(2))-1; \\
weights{2}=randn(layers(3),layers(2)); \\

%%% Setting up network %%%%%%%%%%%%%%%%%%%%%%%%%%%%% \\
% matrix holding input to each node summing junction \\
net=cell(Num\_layers-2,1); \\
% 1 matrix for each hidden layer \\
net{1}=ones(1,layers(2)); \\

%%% Setting up Activation (activation output matrix) % %%%%%%% \\
Actfunc=cell(Num\_layers-2,1); \\
% 1 activation matrix for each hidden layer \\
Actfunc{1}=ones(1,layers(2)); \\
end