Proceedings of the 10th International Conference of Control Systems, and Robotics (CDSR'23)
Ottawa, Canada – June 01 - 03, 2023
Paper No. 207
DOI: 10.11159/cdsr23.207

A Radial Basis Function Neural Network Approach to Filtering Stochastic Wind Speed Data

Jiten Parmar[1], Jeff K Pieper[2]
[1]Department of Mechanical and Manufacturing Engineering, University of Calgary
40 Research Pl NW, Calgary, Alberta, Canada T2L 1Y6
Jiten.parmar@ucalgary.ca; pieper@ucalgary.ca
[2]Department of Mechanical and Manufacturing Engineering, University of Calgary
40 Research Pl NW, Calgary, Alberta, Canada T2L 1Y6

Abstract **-** Various types of control methods are utilized in wind turbines to obtain the optimal amount of power from wind. The dynamics of the turbines are required in said methods, and the speed of the wind is a critical component of the analysis. However, the stochastic nature of wind means that wind speed sensor signals are noisy. This paper proposes the utilization of a radial basis function neural network (RBFNN) based filter to process the signal, by training the network with a simulated wind signal. The results showed that the proposed scheme has versatility in terms of noise removal and signal smoothing, and if required, can viably match performance with a Butterworth filter. Three "modes" of processing the signal are determined based on choosing certain ranges of values for parameters which comprise the RBFNN (number of neurons used and learning rate), and the control designer can choose which one to implement based on performance requirements.

Keywords**:** radial basis function networks; wind turbine; wind speed; butterworth

1. Introduction

A neural network is a generalized term for a configuration where units are interconnected and are able to communicate with each other to perform some task. In a contemporary sense, the term refers specifically to the design of artificial systems that are imitating the framework of biological neural structures. Such systems are mathematical models that utilize nodes, generally referred to as "neurons", that take one piece of information, "activate" it, and pass the information to the next neuron in the chain. At the beginning is a set of input(s), and the end a set of output(s). In the middle are layers of said nodes that transform the input(s) to output(s). The nodes themselves have fixed "activation functions". However, the connections between the nodes can have variable weights. It is the nature of the activation functions, the weights, and the number of hidden layers (layers between the input and the output) that dictates the type of neural network it is.

Here, the interest lies in using radial basis functions as the activation functions. Such neural networks are appropriately named Radial Basis Function Neural Networks (RBFNN). Radial basis functions use exponentials and are inherently non-linear. As such, the RBFNN is sometimes referred to as a universal approximator as it can replicate any inputs with arbitrary precision. For functions with highly non-linear dynamics, input-output mapping algorithms that use RBFNNs can be developed that can accurately reproduce said dynamics. Unlike other types of neural network which have multiple hidden layers (Multilayer Perception Networks), a RBFNN only has one hidden layer composed of nodes that utilize the radial basis activation function.

This paper uses an RBFNN to estimate wind speed in real-time based on simulated sensor data to bypass modeling any wind turbine dynamics. This is done by utilizing an RBFNN as a filter on the sensor data. The flexibility of the parameters of the RBFNN allow the filtered signal to have desired characteristics. The method of development is as follows, based on [1].

2. Methods

2.1. Literature Review

Stemming from the work of Powell [2], Broomhead et al [3] introduced the idea of utilizing radial basis functions for machine learning purposes. Much of the development of this method has remained more or less the same at a basic level and the descriptions given in the following sessions are almost identical to those given in [2]. When it comes to wind turbines,

[4][5][6] use an RBFNN for approximating highly non-linear dynamics of a turbine itself. [4] uses input-output mapping algorithms to train a RBFNN to obtain the wind speed by approximating the inverse of the function

$$P_m = \frac{1}{2}\rho A_r v_w^3 C_P(\lambda, \beta) \quad (1)$$

where P_m is the power extracted from the wind, ρ is density, A_r is the area swept by the rotor blades, v_w is the wind speed, C_P is the power coefficient, λ is the tip-speed-ratio, and β is the blade pitch angle. In [4], the errors for the estimated wind speed based on turbine dynamics are within $0.2\frac{m}{s}$. [5] utilizes the same approach and covers the entire operating range of the turbine and obtains results with almost negligible error. [6] has identical development and error as [4].

There is also application of RBFNN to predict wind speeds for use in weather models, as reported in [7][8][9]. The centers of the hidden layers in the RBFNN are determined by K-means clustering with addition of Recursive Least Squares in [7] to obtain relatively acceptable errors. [8] uses an RBFNN and compares it to existing algorithms, with a reported improvement range of approximately 50%. [9] combines a back-propagation neural network and an RBFNN to improve results compared to each alone in wind speed prediction.

2.2 Background

2.2.1 Radial Basis function

A radial basis function is a function whose output is based solely on the distance of the input from a particular point. While there are many choices for radial basis functions, a Gaussian is suitable for the application of wind speed estimation due to its ease of implementation. The Gaussian function has a peak at x=0 and is an exponential decrease to 0 everywhere else. Thus any input value has a result based on how far it is from some particular point in the output space (this point can be decided to be the origin of the function with no functional drawbacks.)

The Gaussian function itself is the following:

$$y(x) = \phi(x) = e^{\frac{-||x-\mu||^2}{2\sigma^2}} \quad (2)$$

Where μ is the center of the function and σ is the standard deviation. Utilizing the l_2 norm on the x - μ term generalizes its application to higher dimensions.

2.2.2 Radial Basis Function Neural Network

Neural networks are characterized by the fact that they require training to perform their function. Generally, the more training data there is, the better the network performs, although over-training can become an issue wherein the unwanted frequencies are introduced back into the output signal. Regardless, the output is inherently dependent on the training data. Thus, this data must be incorporated in the network design. The input layer has n-dimensional input data. Each of these is sent to each node in the hidden layer. Assuming just one output, it is the weighted sum of all of the outputs of the hidden layer. Each of the ϕ in the hidden layer is the activation function. As stated previously, this is the Gaussian function. According to its definition, two properties are still required before it can be activated, μ and σ.

For wind speed estimation, the training data is one dimensional and the sample rate is uniform. As such, the centers of the Gaussians can be selected randomly. Each node thus gets one randomly center from the vector x_1 (simply known as x henceforth). As such, μ is now obtained. The standard deviation can be obtained with this process: Obtain the l_2 norms of every centre with respect to each other. Select the maximum value. And the standard deviation is [10]:

$$\sigma = \frac{5\rho_{max}}{\sqrt{2H}} \quad (3)$$

Where ρ_{max} is the maximum value obtained and H is the number of neurons used. This one singular value can be used in every activation function. Finally, the output weights must be determined. The output of the neural network can be denoted to be the following equation:

$$z(x) = \sum_{j=1}^{H} w_j \phi(x, \mu_j) \tag{4}$$

With each element in the x vector propagating through out the entire network one at a time, the equation above can be rearranged to a matrix form as shown below, assuming c elements in the x vector. However, obtaining the weights means that the network output must be as close as possible to the training output:

$$\begin{bmatrix} \phi(x_1, \mu_1) & \dots & \phi(x_1, \mu_H) \\ \dots & \dots & \dots \\ \phi(x_c, \mu_1) & \dots & \phi(c, \mu_H) \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_H \end{bmatrix} = \begin{bmatrix} y_1 \\ \dots \\ y_H \end{bmatrix} \tag{5}$$

As stated before, the left-most matrix here is known as the interpolation matrix. Reducing it to a compact form:

$$\phi w = y \tag{6}$$

Obtaining the weights then must be a simple matter of inverting the interpolation matrix and multiplying it by the output vector on the right hand side. However, unless c=H, as is almost always not the case, inversion is not directly possible as the matrix is not square. As such, a pseudoinverse can be utilized:

$$w = (\phi^T \phi)^{-1} \phi^T y \tag{7}$$

This allows a reasonably reliable way to calculate the weights in a batch formulation. There are still limitations to this method, namely the size of the training data and invertibility of $\phi^T \phi$. Using MATLAB, there are many instances in which the inversion fails because the condition number of that matrix is practically null. With the same training data, the difference in each run is the selection of the data centers. Solution then naturally seems to be to keep the centers identical. However, a more robust way to obtain the weights is desired.

A technique known as back-propagation is used extensively in neural network research to modify weights across multiple layers. For application to RBFNNs, it is straightforward as only one layer of weights must be determined. This involves a gradient descent approach to obtain the weights at each time step. A quadratic cost function can be designed as follows, which outlines the error between the actual output and the network output (denoted as z):

$$J = (y - z)^2 \tag{8}$$

This cost function must be minimized as it is the error between the network and the training output. To obtain the weights, the following update rule is used with the current iteration being the k^{th} iteration:

$$w_{k+1} = w_k - \eta \frac{\partial J}{\partial w} \tag{9}$$

The parameter η is known as the learning rate and is a critical value to be tuned to obtain the correct results. If this value is too large, the weights will increase exponentially and destabilize the network. If it is too small, the network will be untrained. Typically, a value of 0.01 is first used to determine how well the network performs. It became evident that this value was unable to stay effective for a wide range of neurons and sample rates. With an increase in neurons, the training

data was over-fit and noise would effectively be reintroduced. With an increase in the sample rate the frequency of the network response would proportionately increase. A reduction in the learning rate as the neurons increased and an increase as the sample rate increased was shown to be effective at counter-acting unwanted response noise as a result of changing the properties of the network. As such, the following novel equation is proposed:

$$\eta = \frac{\lambda}{aH} \tag{10}$$

Where a is the sample rate, H is the number of neurons, and λ is a correction constant that is most effective when equal to 20 for this particular application. This implies that there is further tuning to be performed, although 20 has demonstrated to be reliable for a wide range of neurons and sample rates.

3. Results

Here are the results of application to simulated wind data. Such data was obtained from TurbSim. A continuous time turbulence simulation was run for 600 seconds in three-dimensional space. For all intents and purposes, the x-direction is sufficient to test the network. The stochastic nature of wind is evident, and the noisy element is rich enough to reflect non-ideal conditions, as shown in Figure 1. However, λ has to be modified to 0.5. This implies that the network is not encountering the same frequency of noise in the simulation as it did with previous tests which utilized the Weibull distribution. As such, a high λ is not required to compensate.

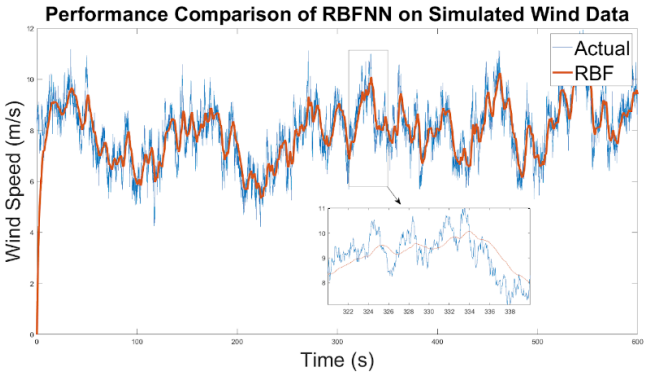

Figure 1: RBFNN Performance on Simulated Wind Speeds

3.1 Butterworth Filter

This section introduces a Butterworth filter to compare performance with the RBFNN. It is a low-pass linear filter which ideally is as flat as possible in the band-pass region. The result should thus be a filter that removes higher frequency noise in the data. The order of the filter is dictated by the slope of the frequency response after the cut-off frequency.
Performance wise, there are a few properties that are immediately noticeable in the Butterworth filter (1st order, cut off frequency of = 0.5 $\frac{rad}{s}$), as shown in Figure 2.

Firstly, the Butterworth filter takes approximately a quarter of the time compared to the neural network to find suitable weights, which is desirable. Secondly, the process delay is about the same as the neural network. Lastly, the noise removal ability is notably superior to the neural network, as the “smoothness” of the filtered data is substantially more pronounced.

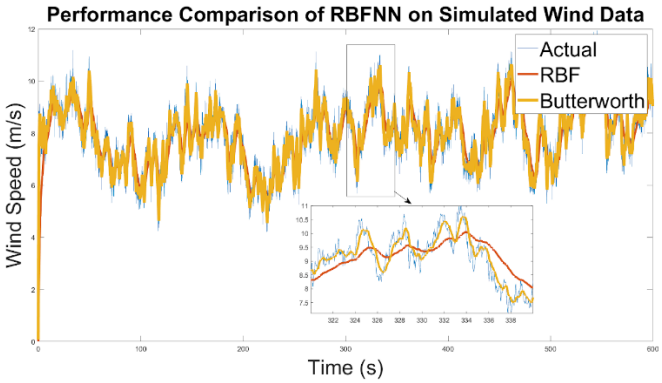

Figure 2: Filtering Performance of the Butterworth Filter

Visually, there is only limited information that can be extracted. Signal analysis is required to numerically compare the performance. Furthermore, the neural network has not been finely tuned (learning rate and number of neurons can be modified) to achieve optimal performance.

3.2 Performance Analysis

With a built-in signal analysis tool in MATLAB called Signal Analyzer, the power spectrum of the signals can be obtained. In the original signal, there are low-frequency contents in the signal which are most notably present. The slope for this region (up to $10^{-1}\frac{rad}{s}$) is approximately 0. Around $10^{-1}\frac{rad}{s}$ is when there is a notable drop in the slope, after which there is a region with a slope of approximately -20 $\frac{dB}{dec}$. Beyond 10 $\frac{rad}{s}$, there are 4 dips in the power. When the simulated wind signal was designed, these spikes were placed into the signal to give it its stochastic characteristics. Based on the understanding developed in the prior section, it is expected that the Butterworth filter will allow lower frequencies to pass while rejecting more of the high frequency content. This is exactly the result. At higher frequency, the integrity of the signal is preserved while significantly reducing the relative power. Placing the RBFNN into this graphing space shows its capabilities in eliminating higher frequency content even still. Figure 3 shows the comparative spectrums of the signals shown in Figure 2. The RBFNN can be seen as an "averaging" entity for the simulated signal in Figure 2. This is displayed in its sharp rejection of the signals beyond the $10^{-1}\frac{rad}{s}$ frequencies.

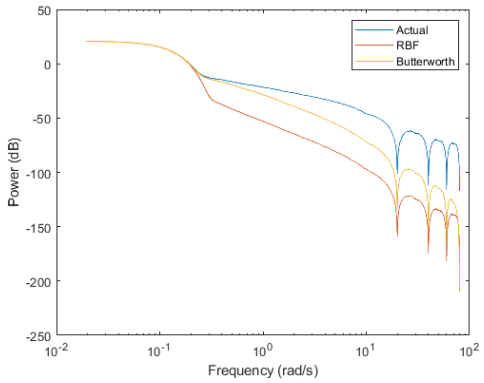

Figure 3: Power spectrum of simulated wind, Butterworth, and the RBFNN filter

Modification of the RBFNN such that it has 10 neutrons with λ=3 yields the following results. The power spectrum is almost identical at low and middle frequency ranges, and reasonably so in the majority of the high frequency range, as noted from Figure 4. Visually, however, there is no practical difference between the outputs in either filter, as seen in Figure 5. In

terms of a control method, it would be inconsequential which filter is utilized. However, it demonstrates that with just a few neurons and minimal tuning, the RBFNN can match the performance of the Butterworth filter. Figure 3 can then be the result is more filtering is desired, and Figure 4 can be the result if the RBFNN is to match the performance of the Butterworth filter.

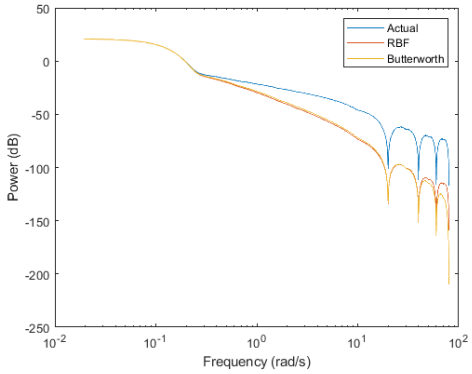

Figure 4: Power spectrum of RBFNN and Butterworth Filters

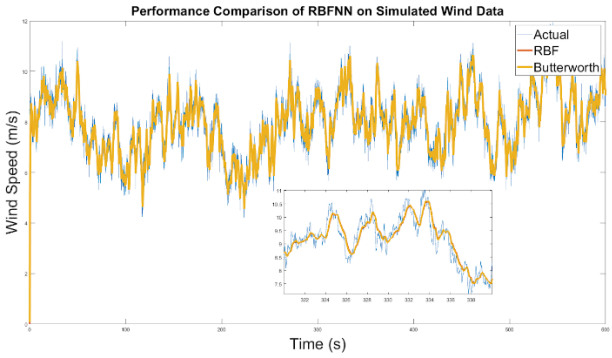

Figure 5: Performance of RBFNN and Butterworth Filters

A simple method to compare the filters with a quantifiable result is to compare the relative power of the original signal with Butterworth and RBFNN processed signals. This can be done by numerically integrating the signals with the following formula:

$$F_p = \sum_{n=1}^{N}(W(n) - BW(n))s \tag{11}$$

Where F_p is a performance factor, N is the total number of samples in the data, W is the wind simulation signal vector, BW is the Butterworth signal vector, and s is the increment size between data points.

Likewise, the formula can be adjusted for the RBFNN as

$$F_p = \sum_{n=1}^{N}(W(n) - NN(n))s \tag{12}$$

Where NN is the RBFNN signal vector. The Butterworth and the wind signals stay constant and therefore are reasonable metrics to compare against the RBFNN. With the equations stated above, the integral is as follows:

$$F_p = 3200 \tag{13}$$

The results for the RBFNN can be normalized about the performance factor stated above. It is expected that the network will shift about the graphs shown in Figures 3 and 4 as the number of neurons and λ are modified, and this will be reflected in its normalized performance factor. The resultant data can be combined into a surface plot shown in Figure 6. Since the data is normalized, a semi-transparent plane showing the base value of 1 is shown for clarity. Across the entire range of the neurons tested, there are values of λ that can be selected such that the performance of the network is practically identical to the Butterworth filter. This range is approximately $2 \leq \lambda \leq 6$ and increases roughly linearly with the number of neurons. The normalized performance factor (NPF) dictates the noise removing element of the network. As such, a higher factor would translate to a less noisy signal. However, the cost becomes signal integrity with respect to the original. Conversely, a lower factor would translate to more noise, while increasingly preserving the signal integrity. The conclusion being that at a certain point, the network would recreate the original unfiltered signal.

It is also evident that modification of λ has a significantly higher impact than modification of the number of neurons, especially at lower λ ranges as shown in Figure 7. The displayed curve can be approximated by some arbitrary exponential decay function. As such, the performance factor has an increasing slope as the λ keeps increasing indicating that there is a steady state value. Based on how the performance factor is calculated, this steady state value is 0, which pertains to the unfiltered signal as mentioned before. When it comes to the number of neurons, the impact on the performance factor is not nearly as drastic as λ, as seen in Figure 8. Here, the displayed curve can be approximated by some arbitrary bounded exponential growth function. Similar to λ, the highest impact is in the lower range of neurons utilized, after which there are diminishing returns.

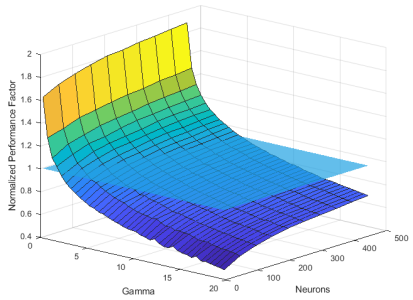

Figure 6: NPF vs. λ and neurons

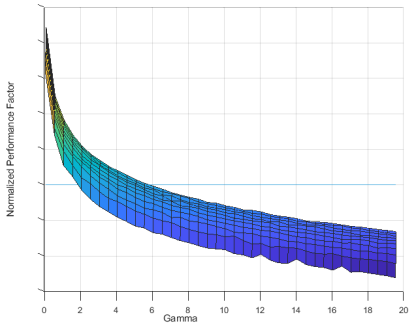

Figure 7: Impact of λ on the NPF

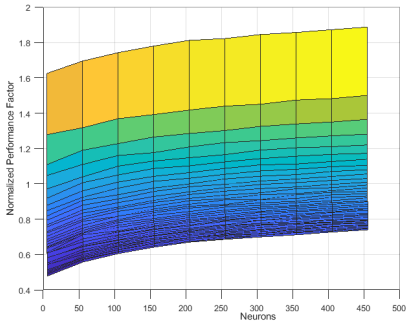

Figure 8: Impact of neurons on the NPF

The inquiry then becomes the following: in this interplay of computational cost, signal integrity, comparison to an existing filter, and noise removal, what are the ideal parameters to choose for the RBFNN? It comes down to the requirements of the output signal. The control designer must choose the appropriate parameters for the application that is utilizing the output signal. There may be instances where high frequency noise must be eliminated, or the middle range be kept. Or a signal delay might be critical or irrelevant to the performance of the whole system. These are a few examples of the factors that must be considered.

Tabulated below are the tested λ and neuron ranges. A total of 9 combinations were tested.

Table 1: Ranges tested for λ and neurons

λ	0.1-1	1-3	3-10
Neurons	0-50	50-100	100-150

Every additional neuron added is an additional 6 calculations at every time step. Depending on the sample size, this can become unfavorable very quickly at high numbers of neurons. For application to control design, and control design for wind turbines in particular, it is critical to minimize active delay in the output since wind is extremely stochastic. At the same time, wear and tear of moving parts must be taken into account as the controller attempts to overcome the rapid changes in the disturbances. As such, there are several modes that can be deduced from the results that a control designer might utilize:

Table 2: Modes for ranges of λ and Neurons

Mode	**Parameters**		**Notes**
	Neurons	λ	
Durability	50-100	0.1 - 1	When fatigue on the electromechanical parts must be minimized, this mode can be used. It ensures an extremely smooth output while rejecting noise that would cause any sudden output changes.

Speed	0-50	3-10	This mode ensures that the appropriate noise is removed while maintaining adequate response times. For applications in noisy environments and those where maximum power-point tracking is desired, these parameters ensure that the output will have enough integrity to accurately reflect the real conditions.
Robustness	100-150	1-3	When rapid changes are to be expected in the disturbances, this mode ensures a balance between speed and accuracy of the output. A higher number of neurons create smoothness in the data while the range can compensate for slower speeds with its ability to overcome the largest expected changes in the original signal.

5. Conclusions

This paper proposes a RBFNN based filtering technique for wind speed estimation for application to wind turbine control methods. Simulated wind data was input and the resulting signal was analyzed. The parameters of the RBFNN can be manipulated as per the tabulated results to obtain the desired characteristics in the filtered signal. The modes that can be demonstrated are for wind turbine durability, control method response speed, and control method robustness. The results also show that the RBFNN can be on par with a Butterworth filter if the design parameters are particularly set.

References

[1] J. Liu, "Radial basis function (RBF) neural network control for mechanical systems," SpringerLink, 2015. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-34816-7. [Accessed: 30-Dec-2022].

[2] M. Powell, "Radial basis functions for multivariable interpolation: A Review," Semantic Scholar, 01-Jan-1987. [Online]. Available: https://www.semanticscholar.org/paper/Radial-basis-functions-for-multivariable-a-review-Powell/c71ca26b183025b9f39f940f5e730f2c9a64e414. [Accessed: 30-Dec-2022].

[3] D. Broomhead and D. Lowe, "[pdf] multivariable functional interpolation and adaptive networks: Semantic scholar," Complex Syst., 01-Jan-1988. [Online]. Available: https://www.semanticscholar.org/paper/Multivariable-Functional-Interpolation-and-Adaptive-Broomhead-Lowe/b08ba914037af6d88d16e2657a65cd9dc5cf5da1. [Accessed: 30-Dec-2022].

[4] Wei Qiao, Wei Zhou, J. M. Aller, and R. G. Harley, "Wind speed estimation based sensorless output maximization control for a wind turbine driving a DFIG," IEEE Transactions on Power Electronics, vol. 23, no. 3, pp. 1156–1169, 2008.

[5] K. Kaur, T. K. Saha, S. N. Mahato, and S. Banerjee, "Wind Speed Estimation based control of stand-alone Doig for Wind Energy Conversion System," 2014 IEEE International Conference on Industrial Technology (ICIT), 2014.

[6] L. Tian, Q. Lu, and W.-zhuo Wang, "A Gaussian RBF Network Based Wind Speed Estimation Algorithm for maximum power point tracking," Energy Procedia, vol. 12, pp. 828–836, 2011.

[7] Silva, Gonçalo & Fonte, Pedro & Quadrado, Jose. (2006). Radial Basis Function Networks for Wind Speed Prediction.

[8] G. Sideratos and N. D. Hatziargyriou, "Application of radial basis function networks for wind power forecasting," Artificial Neural Networks – ICANN 2006, pp. 726–735, 2006.

[9] G. Chunlin, M. Zhou, P. Xin, and Y. Xiaoyan, "Research on the combined forecasting method of wind speed," 2017 China International Electrical and Energy Conference (CIEEC), 2017.

[10] C. C. Aggarwal, "Training The Hidden Layer," in Neural networks and deep learning: A textbook, New York: Springer, 2019, pp. 221–221.

[11] S. W. Smith, "The z-Transform," in The scientist and engineer's Guide to Digital Signal Processing, San Diego, CA: California Technical Pub., 1997, pp. 610–613.