Chapter 2

Comparison Study of AI-based Methods in Wind Energy

Ping Jiang, Feng Liu and Yiliao Song

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/63716

Abstract

Wind energy forecasting is particularly important for wind farms due to cost-related issues, dispatch planning, and energy market operations. Thus, improving forecasting accuracy becomes an urgent task for researchers in the field of wind energy. However, there is limited research to discuss an overall comparison among various forecasting types, which is a foundation for future works with respect to wind energy prediction because this comparison may reveal whether there is a best model for a specific forecasting type or specific data in this field. For the purpose of laying a strong foundation for wind energy research, this chapter introduces five basic forecasting models, which are Autoregressive Moving Average Model (ARMA), Back-Propagation Neuron Network (BPNN), Support Vector Regression (SVR), Extreme Learning Machine (ELM), and Adaptive Network-Based Fuzzy Inference System (ANFIS) with implement codes before comparing the forecasting effectiveness of five different models in three wind farms based on five forecasting types. Comparison results indicate that each model has great divergent forecasting results in different wind farms and every forecasting type has its own "best model."

Keywords: wind power, AI-based models, forecasting, comparison, Matlab codes

1. Introduction

The important environmental advantages of renewable energy sources have been significantly noticed, which results in most industrialized countries committed to developing the installation of wind power plants [1]. The share of total installed capacity of wind power in China has reached approximately 27% in the global capacity, which is 96.37 GW, due to the historically high installation of new wind power capacity.[1] Moreover, a renewable-energy-

[1] N.E. Administration, http://www.nea.gov.cn/2015-02/12/c_133989991.htm (2015).

oriented power system was proposed as the fundamental aim of China's energy transformation during the Asia-Pacific Economic Cooperation (APEC) Conferences.[2] Advanced prediction techniques are urgently needed to integrate wind energy into the electrical power grid in a manner that benefits both Transmission System Operators (TSOs) and Independent Power Producers (IPPs) [2,3].

Since wind energy has the inherently intermittent nature and stochastic nonstationarity, it brings significant levels of uncertainty to system operators [4]. Thus, accurate wind forecasts are of primary importance to solve operational, planning, and economic problems in the growing wind power scenario [4,5]. Current wind power forecasting research has been divided into point forecasts (also called deterministic predictions) [6–8] and uncertainty forecasts [9,10]. Deterministic forecasts deliver specific amounts of wind power and focus on reducing the forecasting error [11]. By contrast, it is essential to decision-making processes and electricity market trading strategies [12] that uncertainty forecasts provide uncertainty information for system operators to manage the wind power generation of wind farms [11,13].

The existing approaches published in the literature with respect to wind energy prediction can be divided into three categories—artificial intelligent model, physical model, and statistical model—and sometimes a hybrid one, which integrates advantages of different categories, is involved. Researchers often utilize them to forecast various types of wind speed and wind power by various types including the multi-step forecasting, long-term wind speed (power) forecasting, and so on. However, few literature exist to discuss an overall comparison among various forecasting types, which is a foundation for future works of wind energy prediction because this comparison may reveal whether there is a best model for a specific forecasting type or specific data in the field of wind energy.

The following parts in this paper are demonstrated by four sections: methodology briefly introduces forecasting approaches that this paper adopts; data collection specifically illustrates three types of wind data; simulation and result displays and evaluates the final results of forecasting effectiveness of each model; and conclusion draws the main results that this paper investigates.

2. Methodology

In this section, we will introduce five basic models including Autoregressive Moving Average Model (ARMA), Back-Propagation Neuron Network (BPNN), Support Vector Regression (SVR), Extreme Learning Machine (ELM), and Adaptive Network-Based Fuzzy Inference System (ANFIS) as forecasting methods in this paper to predict wind speed. For the purpose of clearly demonstrating procedures of forecasting wind speed using these methods, this chapter will attach the Matlab code after introduction of each model and the wind speed series is defined as follows:

[2] N.E. Administration, http://www.nea.gov.cn/2015-06/08/c_134305870.htm (2015).

$$Ws = \left(ws_1, \quad ws_2, \quad \cdots \quad , ws_N \right) \tag{1}$$

where N is a positive integer and $ws_n \in (0, \quad +\infty) \subset \mathbb{R}$ represents the wind speed of time n.

2.1. Autoregressive moving average model

2.1.1. Brief introduction of ARMA

This section displays ARMA model and the ARMA(p, q) model can be expressed as follows [14]:

$$y_t = \delta + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{j=1}^{q} \phi_j e_{t-j} + e_t \tag{2}$$

Where δ is the constant term of the ARMA model, φ_i is the ith autoregressive coefficient, ϕ_j is the jth moving average coefficient, e_t is the error term at time period t, and represents the value of wind speed observed or forecasted at time period t. Thus, the first step in applying ARMA model to forecast wind speed is to identify p and q, which is related to the stationarity of the time series, which means that the stationarity assumption of observed series should be checked first. For this purpose, inspection of the run plots and Auto Correlation Function (ACF) plots can be used for deciding on the order of differencing.

2.1.2. Implement for wind speed forecasting

Since the Matlab has the ARMA package, we will implement this model to forecast wind speed using five steps as follows:

1. Identify the domain of p and q. In this chapter, we set $p \in \{1, 2, 3, 4, 5\}$ and $q \in \{1, 2, 3, 4, 5\}$.
2. For each pair (p, q), calculate corresponding model's AIC value.
3. Select the pair (p, q), which makes corresponding model's AIC value to be minimum one among all of pairs of (p, q) as the best (p, q) and denote this pair as (p_test, q_test).
4. Apply (p_test, q_test) to establish ARMA model.
5. Forecast m steps using the model established in step (4).

The detailed Matlab codes are listed in Code 1.

Code 1. Matlab Codes of ARMA.

```
function [y_output,p_test,q_test] = arma_dynmc(Ws,m)

z = Ws; step = m;

test = [];

for p = 1:5

  for q = 1:5

   m = armax(z,[p,q]);

   AIC = aic(m);

   test = [test;p q AIC];

  end

end

[min_aic,min_i] = min(test(:,3));

p_test = test(min_i,1);q_test = test(min_i,2);

m = armax(z,[p_test,q_test]);

P = predict(m,z,step);

y_output = P(end-step+1:end,1)';
```

2.2. Back-propagation neuron network

2.2.1. Brief introduction of BPNN

Mccelland and Rumelhart developed the BP neural network model in 1985. There are three layers in a particular network: the input, hidden, and output layers. Each layer has designed nodes, whose functions aim to calculate the inner product of the input vector and weight vector by the transfer function [15]. The process of BPNN is demonstrated by the following steps:

1. Initialize. Assume that the input layer has n nodes, and hidden layer has l nodes, and output layer has m nodes. Let the network distribute values randomly for each threshold value θ_j, γ_t and the connection weight w_{ij}, v_{jt}, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, l$, and $t = 1, 2, \ldots, m$.

2. Calculate the output of hidden layer. Assume $\mathbf{X} \subset [-1, 1]^n$ is the input space and $\mathbf{Y} \subset [-1, 1]^m$ is the output space, which means $\mathbf{X}$ is an n-dimensional space and $\mathbf{Y}$

is an m-dimensional space. We denote each element in as x_i and each element in **Y** is y_t. Thus, the output of hidden layer can be calculated using the following function:

$$H_j = f\left(\sum_{i=1}^{n} w_{ij}x_i - \theta_j\right),\ j = 1, 2, \ldots, l \tag{3}$$

$f(\bullet)$ is the transfer function and is expressed as follows:

$$f(z) = \frac{1}{1 + e^{-z}}$$

3. Calculate the output of the network. Using H_j, v_{jk}, and γ_t, the output of the network can be calculated as follows:

$$O_t = \sum_{j=1}^{l} v_{jk} H_j - \gamma_t,\ t = 1, 2, \ldots, m \tag{4}$$

4. Error calculation. The error between y_t and O_t can be expressed as $e_t = O_t - y_t$.

5. Weights updating. According to back-propagation algorithm, the weights w_{ij}, v_{jt} can be updated by using the following functions:

$$w_{ij} = w_{ij} + \eta H_j \left(1 - H_j\right) x_i \sum_{t=1}^{m} w_{jt} e_t \tag{5}$$

$$v_{jt} = v_{jt} + \eta H_j e_t \tag{6}$$

where η is the learning rate and $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, l$, and $t = 1, 2, \ldots, m$.

6. Biases updating. Similarly, the biases θ_j, γ_t can be updated by using the following functions:

$$\theta_j = \theta_j + \eta H_j \left(1 - H_j\right) \sum_{t=1}^{m} w_{jt} e_t \tag{7}$$

$$\gamma_t = \gamma_t + e_t \tag{8}$$

7. Iteration. If the error is more than the expected value, then execute steps (2)–(6). Otherwise, denote θ_j, γ_t, w_{ij}, and v_{jt} as the optimized parameter of this network with respect to **X** and **Y**.

2.2.2. Implement for wind speed forecasting

In this part, how to apply BPNN to forecast wind speed will be introduced in detail and the MATLAB code for predicting wind speed will be attached. There are six steps to forecast wind speed applying BPNN as follows:

1. Design forecasting mode. In this chapter, AI-based model will apply the following mode to forecast wind speed of different wind databases

$$\left(ws_{t-1}, ws_{t-2}\right)\xrightarrow{forecast} ws_t \tag{9}$$

Eq. (8) indicates that the input space **X** for BPNN model is a two-dimensional space and the output space **Y** is a one-dimensional space, meaning that $n = 2$ and $m = 1$ as described in Section 2.2.1.

2. Assign training samples. Based on step (1), the input and output of training samples are expressed as follows:

$$Train\_input = \begin{pmatrix} ws_1 & ws_2 & \cdots & ws_{N-2} \\ ws_2 & ws_3 & \cdots & ws_{N-1} \end{pmatrix} \tag{10}$$

$$Train\_output = \left(ws_3 \quad ws_4 \quad \cdots \quad ws_N\right) \tag{11}$$

The Matlab codes of this step are listed in Code 2.

Code 2. Matlab codes to construct training samples.

```
n = length(Ws);

for i = 1:2

  Train_input(i,:) = Ws(i:n+i-3);

End

Train_output = Ws(3:n)
```

3. Normalize *Train_input* and *Train_output*. Since $\mathbf{X}\subset[-1,\ 1]^n$ and $\mathbf{Y}\subset[-1,\ 1]^m$, *Train_input* and *Train_output* need to be adjusted to satisfy this condition. First, the maximum and minimum values of each row in *Train_input* need to be calculated using the following functions:

$$Min_{input}^{(i)} = min\left\{i^{th} \text{ row in } Train\_input\right\} \tag{12}$$

$$Max_{input}^{(i)} = max\left\{i^{th} \text{ row in } Train\_input\right\} \tag{13}$$

Let $D_{input}^{(i)} = Max_{input}^{(i)} - Min_{input}^{(i)}$ and normalized *Train_input* can be obtained by the following equation (i = 1, 2):

$$Train\_inputm = \begin{pmatrix} \frac{ws_1 - Min_{input}^{(1)}}{D_{input}^{(1)}} & \frac{ws_2 - Min_{input}^{(1)}}{D_{input}^{(1)}} & \cdots & \frac{ws_{N-2} - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \frac{ws_2 - Min_{input}^{(2)}}{D_{input}^{(2)}} & \frac{ws_3 - Min_{input}^{(2)}}{D_{input}^{(2)}} & \cdots & \frac{ws_{N-1} - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{14}$$

Similarly, normalized *Train_output* can be obtained by the following equations:

$$Min_{output}^{(i)} = min\left\{i^{th} \text{ row in } Train\_output\right\}$$

$$Max_{output}^{(i)} = max\left\{i^{th} \text{ row in } Train\_output\right\}$$

$$D_{output}^{(i)} = Max_{output}^{(i)} - Min_{output}^{(i)}$$

$$Train\_outputm = \begin{pmatrix} \frac{ws_3 - Min_{output}^{(1)}}{D_{output}^{(1)}} & \frac{ws_4 - Min_{output}^{(1)}}{D_{output}^{(1)}} & \cdots & \frac{ws_N - Min_{output}^{(1)}}{D_{output}^{(1)}} \end{pmatrix} \tag{15}$$

The Matlab codes of this step are listed in Code 3.

Code 3. Matlab codes to normalize training samples.

```
[Train_inputm, inputs] = mapminmax(Train_input);

[Train_outputm, outputs] = mapminmax(Train_output);
```

4. Assign parameters and train the network. We assign that the network has three layers and the input layer has two nodes, the hidden layer has two nodes and the output layer has one node. Thus, according to Section 2.2.1, the optimized weights and biases of this network can be calculated and denoted by θ_j, γ_t, w_{ij}, v_{jt}, $i = 1, 2$, $j = 1, 2$, and $t = 1$. The Matlab codes of this step are listed in Code 4.

Code 4. Matlab codes to build and train BPNN.

```
Net = newff(Train_input, Train_output, 2);

Net = train(Net, Train_input, Train_output);
```

Net is the trained network and can be applied to forecast wind speed in the following steps:

5. Forecast ws_{N+1}. Based on the forecasting mode in step (1), we need to apply ws_{N-1} and ws_N to forecast ws_{N+1}, which means that the *Test_input* is $(ws_{N-1}, ws_N)^T$. Then, $D^{(i)}_{input}$, $Max^{(i)}_{input}$ and $Min^{(i)}_{input}$ obtained in step (3) will be applied to normalize *Test_input* and the normalized *Test_input* can be expressed using *Test_inputm* as follows:

$$Test\_inputm = \begin{pmatrix} \dfrac{ws_{N-1} - Min^{(1)}_{input}}{D^{(1)}_{input}} \\ \dfrac{ws_N - Min^{(2)}_{input}}{D^{(2)}_{input}} \end{pmatrix} \tag{16}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O \in \mathbb{R}$. Thus, the forecasting value of ws_{N+1} is *Ws_forecast* = $O \times D^{(1)}_{output} + Min^{(1)}_{output}$.

The Matlab codes of this step are listed in Code 5.

Code 5. Matlab codes to forecast using established BPNN (single step ahead)

```
Test_inputm = mapminmax('apply', Test_input, inputs);

O=sim(Net, Test_inputm);

Ws_forecast= mapminmax('reverse', O, outputs);
```

6. Forecast ws_{N+2}. Based on the forecasting mode in step (1), we need to apply ws_N and forecasting value of ws_{N+1}, *Ws_forecast*, to forecast ws_{N+2}, which means that the *Test_input*2 is $(ws_N, Ws\_forecast)^T$. Then, $D^{(i)}_{input}$, $Max^{(i)}_{input}$ and $Min^{(i)}_{input}$ obtained in step (3) will be applied to normalize *Test_input*2 and the normalized *Test_input*2 can be expressed using *Test_inputm*2 as follows:

$$Test\_inputm2 = \begin{pmatrix} \dfrac{ws_N - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{Ws\_forecast - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{17}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O_2 \in \mathbb{R}$. Therefore, the forecasting value of ws_{N+2} is *Ws_forecast*2 = $O_2 \times D_{output}^{(1)} + Min_{output}^{(1)}$.

The Matlab codes of this step are listed in Code 6.

Code 6. Matlab codes to forecast using established BPNN (two-step ahead).

```
Test_inputm2 = mapminmax('apply', Test_input2, inputs);

O2=sim(Net, Test_inputm2);

Ws_forecast2= mapminmax('reverse', O, outputs);
```

2.3. Support vector regression

2.3.1. Brief introduction of SVR

SVR is the most common application of support vector machines (SVMs) and constructs a hyperplane that separates examples with maximum margin, to categorize or forecast series [16]. Given the training data $\{(x_1, y_1), \ldots, (x_l, y_l)\} \subset \mathbf{X} \times \mathbb{R}$, where $\mathbf{X} = \mathbb{R}^d$ denotes the space of input patterns, then the goal of SVR is to find a function $f(x)$ that is as flat as possible with at most ε deviation from the actually obtained targets y_i for all the training data. The simplest, linear formula for the output of a linear SVR is defined as [17]

$$f(x) = \langle w, x \rangle + b, w \in X, b \in \mathbb{R} \tag{18}$$

where $\langle w, x \rangle$ denotes the dot product in X. One way to get the largest flatness is to minimize the w in Eq. (18) and this problem can be written as Eq. (19) by introducing slack variables ξ_i, ξ_i^* into a convex optimization problem function:

$$\begin{aligned} &\mathit{minimize}\ \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\left(\xi_i + \xi_i^*\right) \\ &\mathit{subject\ to} \begin{cases} y_i - \langle w, x_i\rangle - b \le \varepsilon \\ \langle w, x_i\rangle + b - y_i \le \varepsilon \\ \xi_i, \xi_i^* \qquad\qquad \ge 0 \end{cases} \end{aligned} \tag{19}$$

The constant $C>0$ determines the trade-off between the flatness of f and the amount up to which deviations larger than ε are tolerated.

To figure out the support vector regression function, the dual problem of Eq. (19) is defined in Eq. (20) by constructing a Lagrange function from the objective function:

$$\begin{aligned} &\mathit{maximize} \begin{cases} \frac{1}{2}\sum_{i,j=1}^{l}\left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right)\langle x_i, x_j\rangle \\ -\varepsilon\sum_{i=1}^{l}\left(\alpha_i + \alpha_i^*\right) + \sum_{i=1}^{l} y_i\left(\alpha_i - \alpha_i^*\right) \end{cases} \\ &\mathit{subject\ to}\ \ \sum_{i=1}^{l}\left(\alpha_i - \alpha_i^*\right) = 0 \ \ \mathit{and}\ \ \alpha_i, \alpha_i^* \in [0, C] \end{aligned} \tag{20}$$

With the solution of $(\alpha,\ \alpha^*)$ from Eq. (20), the support vector regression function can be written as follows:

$$f(x) = \sum_{i=1}^{l}\left(\alpha_i - \alpha^*\right)\langle x_i, x\rangle + b \tag{21}$$

When it comes to the nonlinear regression problem, the training patterns x_i can be mapped into a high-dimensional space, where the nonlinear regression problem is transformed into a linear one. The expansion of Eq. (21) is defined as Eq. (22):

$$f(x) = \sum_{i=1}^{N}\left(\alpha_i^* - \alpha_i\right)k\left(x_i, x; g\right) + b \tag{22}$$

where α_i^* and α_i are Lagrange multipliers and $k(x_i,\ x; g)$ is the kernel function, in which g is a parameter and generally set as $1/d$.

2.3.2. Implement for wind speed forecasting

We use the libsvm package (Version 3.17) to implement forecasting task of wind speed using SVR. There are six steps and steps (1)–(3) are same as in steps (1)–(3) in Section 2.2.2. Thus, we start to introduce from step (4).

(4) Assign parameters and train the SVR model. We assign that C is 4 and g is 0.5. Thus, according to Section 2.3.1, the optimized solution $(\alpha,\ \alpha^*)$ can be calculated. The Matlab codes of this step are listed in Code 7.

Code 7. Matlab codes to establish and train SVR.

```
Model = svmtrain(Train_output', Train_input', '-C 4 –g 0.5');
```

Model is the trained SVR model and can be applied to forecast wind speed by the following steps:

(5) Forecast ws_{N+1}. Based on the forecasting mode in step (1), we need to apply ws_{N-1} and ws_N to forecast ws_{N+1}, which means that the *Test_input* is $(ws_{N-1},\ ws_N)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input* and the normalized *Test_input* can be expressed using *Test_inputm* as follows:

$$Test\_inputm = \begin{pmatrix} \dfrac{ws_{N-1} - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{ws_N - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{23}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O \in \mathbb{R}$. Thus, the forecasting value of ws_{N+1} is $Ws\_forecast = O \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 8.

Code 8. Matlab codes to forecast using established SVR (single step ahead).

```
Test_inputm = mapminmax('apply', Test_input, inputs);

O=svmpredict(0, Test_inputm',Model);

Ws_forecast= mapminmax('reverse', O, outputs);
```

(6) Forecast ws_{N+2}. Based on the forecasting mode in step (1), we need to apply ws_N and the forecasting value of ws_{N+1}, *Ws_forecast*, to forecast ws_{N+2}, which means that the *Test_input*2 is $(ws_N,\ Ws\_forecast)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input*2 and the normalized *Test_input*2 can be expressed using *Test_inputm*2 as follows:

$$Test\_inputm2 = \begin{pmatrix} \dfrac{ws_N - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{Ws\_forecast - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{24}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O_2 \in \mathbb{R}$. Therefore, the forecasting value of ws_{N+2} is $Ws\_forecast2 = O_2 \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 9.

Code 9. Matlab codes to forecast using established SVR (two-step ahead).

```
Test_inputm2 = mapminmax('apply', Test_input2, inputs);

O2=svmpredict(0,Test_inputm2',Model);

Ws_forecast2= mapminmax('reverse', O, outputs);
```

2.4. Extreme learning machine

2.4.1. Brief introduction of ELM

Huang et al. [18] investigated the ELM, which is an effective and efficient learning algorithm. The ELM aims to randomly initialize the weights and biases of SLFN and then to explicitly calculate the hidden layer output matrix and hence the output weights. Because of the nature of non-adjusted weights and bias, the network can be established using a very low computational cost [19]. Then, the ELM with l hidden neurons and transfer function $\varphi(\cdot)$ can approximate the N samples with zero error as

$$\sum_{j=1}^{l} \beta_j \varphi\left(\mathbf{w}_j \mathbf{x}_k + b_j\right) = y_k, \; k = 1, 2, \ldots, N \tag{25}$$

which can be written as ***HB***=**Y**, with

$$H\left(\mathbf{w}_1, \ldots, \mathbf{w}_l, b_1, \ldots, b_l, \mathbf{x}_1, \ldots, \mathbf{x}_N\right) = \begin{pmatrix} \varphi\left(\mathbf{w}_1 \mathbf{x}_1 + b_1\right) & \cdots & \varphi\left(\mathbf{w}_l \mathbf{x}_1 + b_l\right) \\ \vdots & \ddots & \vdots \\ \varphi\left(\mathbf{w}_1 \mathbf{x}_N + b_1\right) & \cdots & \varphi\left(\mathbf{w}_l \mathbf{x}_N + b_l\right) \end{pmatrix} \tag{26}$$

where $\boldsymbol{w}_\mathrm{i}$ is the weight vector connecting the *i*th hidden neuron and the input nodes, β_i is the weight vector connecting the *i*th hidden neuron and the output neurons, and b_i is the threshold of the *i*th hidden neuron, and $B = (\beta_1^T \dots \beta_l^T)^T$, $Y = (y_1^T \dots y_N^T)^T$

Then, the output weights $\boldsymbol{B}$ can be calculated from the hidden layer output matrix $\boldsymbol{H}$ and the target values $\mathbf{Y}$ as $\boldsymbol{B}\hat{} = \boldsymbol{H}\dagger\mathbf{Y}$, where $\boldsymbol{H}\dagger$ is the Moore-Penrose generalized inverse of the matrix $\boldsymbol{H}$.

2.4.2. Implement for wind speed forecasting

We use Matlab to compile ELM model and provide two *.m files, which are Elmtrain.m and Elmpredict.m, in Appendix. Elmtrain function is similar to train function for BPNN and svmtrain function for SVR and Elmpredict function is similar to sim function for BONN and svmpredict function for SVR. In detail, there are six steps for forecasting wind speed using ELM model and steps (1)–(3) are same as in steps (1)(3) in Section 2.2.2. Thus, we start to introduce them from step (4).

(4) Assign parameters and train the ELM model. We design that the input layer of ELM model has two nodes, and the hidden layer of ELM has two nodes, and the output layer of ELM model has one node. Thus, according to Section 2.4.1, the optimized $\boldsymbol{B}$ can be calculated. The Matlab codes of this step are listed in Code 10.

Code 10. Matlab codes to establish and train ELM.

```
[W,b,B,TF,TYPE] = elmtrain(Train_input, Train_output, 2);
```

W is the weight matrix of the input layer and hidden layer, and *b* is the bias vector of input layer and hidden layer, and B is $\boldsymbol{B}\hat{}$ in Section 2.4.1, and TF is the transfer function $\varphi(\cdot)$, and TYPE is model's functions (classification or regression). We select sigmoid function as the transfer function $\varphi(\cdot)$ of ELM, which is same as that of BPNN.

(5) Forecast ws_{N+1}. Based on the forecasting mode in step (1), we need to apply ws_{N-1} and ws_N to forecast ws_{N+1}, which means that the *Test_input* is $(ws_{N-1}, ws_N)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input* and the normalized *Test_input* can be expressed using *Test_inputm* as follows:

$$Test\_inputm = \begin{pmatrix} \dfrac{ws_{N-1} - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{ws_N - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{27}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O \in \mathbb{R}$. Thus, the forecasting value of ws_{N+1} is $Ws\_forecast = O \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 11.

Code 11. Matlab codes to forecast using established ELM (single step ahead).

```
Test_inputm = mapminmax('apply', Test_input, inputs);

O=elmpredict(Test_inputm, W,b,B,TF,TYPE);

Ws_forecast= mapminmax('reverse', O, outputs);
```

(6) Forecast ws_{N+2}. Based on the forecasting mode in step (1), we need to apply ws_N and forecasting value of ws_{N+1}, *Ws_forecast*, to forecast ws_{N+2}, which means that the *Test_input*2 is $(ws_N,\ Ws\_forecast)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input*2 and the normalized *Test_input*2 can be expressed using *Test_inputm*2 as follows:

$$Test\_inputm2 = \begin{pmatrix} \dfrac{ws_N - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{Ws\_forecast - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{28}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O_2 \in \mathbb{R}$. Therefore, the forecasting value of ws_{N+2} is $Ws\_forecast2 = O_2 \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 12.

Code 12. Matlab codes to forecast using established ELM (two-step ahead).

```
Test_inputm2 = mapminmax('apply', Test_input2, inputs);

O2=svmpredict(Test_inputm2, W,b,B,TF,TYPE);

Ws_forecast2= mapminmax('reverse', O, outputs);
```

2.5. Adaptive network-based fuzzy inference system

2.5.1. Brief introduction of ANFIS

ANFIS is introduced to compensate for the disability of conventional mathematical tools to address uncertain systems, such as human knowledge and reasoning processes. There are two contributions of FIFs restructured: proposing a standard method for transforming ill-defined factors into identifiable rules of FIS and using an adaptive network to tune the membership functions. We assume that there are two fuzzy if-then rules contained in the system, two inputs (*x* and *y*) and one output (*z*), and the processes of ANFIS are described in **Figure 1** [20–21].

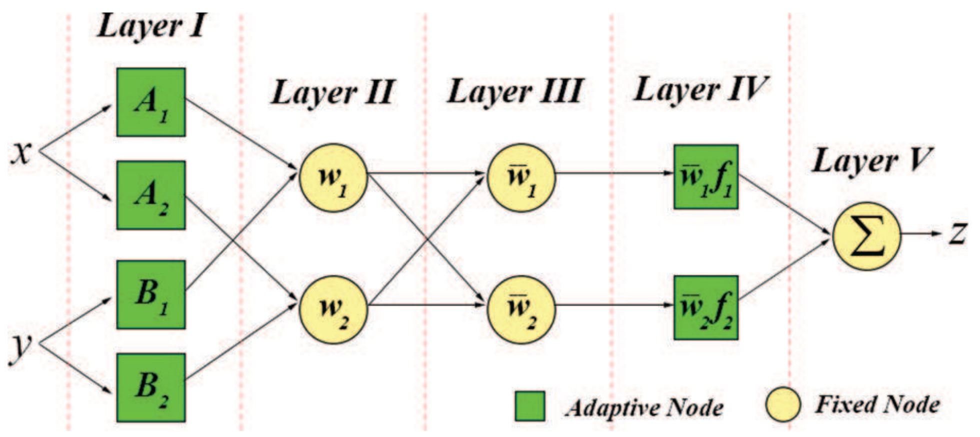

Figure 1. Five processes included in this figure. In an ANFIS architecture, circles are fixed nodes without parameters and squares represent adaptive nodes whose parameters are determined by training data and a gradient-based learning procedure.

Layer I: Mapping a certain input x to a fuzzy set $O_i^{(1)}$ for every node i by the member functions μ_A, which is usually bell-shaped with a parameter set $\{a_i,\ b_i,\ c_i\}$, as is y

$$O_i^{(1)} = \mu_A(x), \text{ where } \mu_A(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)\right]^{b_i}} \text{ or } \mu_A(x) = e^{-\left(\frac{x-c_i}{a_i}\right)^2} \tag{29}$$

Layer II: In this layer, each circle node performs the connection "AND" and multiplies inputs, as well as sends the product out:

$$O_i^{(2)} = \omega_i = \mu_A(x) \times \mu_B(x) \tag{30}$$

Layer III: Every circle node in this layer calculates a normalized firing strength, namely a ratio of the rule's firing strength to the sum of all rules' firing strengths:

$$O_i^{(3)} = \bar{\omega}_i = \frac{\omega_i}{\sum \omega_i} \tag{31}$$

Layer IV: Assume the rules of this system are as follows:

Rule 1: If x is A_1 and y is B_1, then $f_1 = p_1 x + q_1 y + r_1$

Rule 2: If x is A_2 and y is B_2, then $f_2 = p_2 x + q_2 y + r_2$

Then, the outputs of the adaptive nodes in this layer are computed by

$$O_i^{(4)} = \bar{\omega}_i f_i = \bar{\omega}_i \left(p_1 x + q_1 y + r_1 \right) \tag{32}$$

Layer V: The overall output is the weighted average of all incoming signals:

$$O_i^{(5)} = \sum_i \bar{\omega}_i f_i = \frac{\sum_i \omega_i f_i}{\sum_i \omega_i} \tag{33}$$

Particularly, in this case,

$$O_i^{(5)} = \sum_{i=1}^{2} \bar{\omega}_i f_i = \sum_{i=1}^{2} \left(\bar{\omega}_i x \right) p_i + \left(\bar{\omega}_i y \right) q_i + \bar{\omega}_i r_i \tag{34}$$

2.5.2. Implement for wind speed forecasting

We use the genfis3 function and ANFIS function in Matlab to implement forecasting task of wind speed using ANFIS. There are six steps and steps (1)–(3) are same as in steps (1)–(3) in Section 2.2.2. Thus, we start to introduce from step (4).

(4) Assign parameters and train the ANFIS model. We set the number of iteration of ANFIS as 100. Thus, the optimized parameters can be calculated. The Matlab codes of this step are listed in Code 13.

Code 13. Matlab codes to establish and train ANFIS.

```
fismat = genfis3(Train_input', Train_output');

out_fis1 = anfis([Train_input' Train_output'],fismat,100);
```

The out_fis1 is the trained ANFIS and can be applied to forecast wind speed.

(5) Forecast ws_{N+1}. Based on the forecasting mode in step (1), we need to apply ws_{N-1} and ws_N to forecast ws_{N+1}, which means that the *Test_input* is $(ws_{N-1}, ws_N)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input* and the normalized *Test_input* can be expressed using *Test_inputm* as follows:

$$Test\_inputm = \begin{pmatrix} \dfrac{ws_{N-1} - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{ws_N - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{35}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O \in \mathbb{R}$. Thus, the forecasting value of ws_{N+1} is $Ws\_forecast = O \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 14.

Code 14. Matlab codes to forecast using established ANFIS (single step ahead).

```
Test_inputm = mapminmax('apply', Test_input, inputs);

O=evalfis(Test_inputm', out_fis1);

Ws_forecast= mapminmax('reverse', O, outputs);
```

(6) Forecast ws_{N+2}. Based on the forecasting mode in step (1), we need to apply ws_N and forecasting value of ws_{N+1}, *Ws_forecast*, to forecast ws_{N+2}, which means that the *Test_input*2 is $(ws_N, Ws\_forecast)^T$. Then, $D_{input}^{(i)}$, $Max_{input}^{(i)}$ and $Min_{input}^{(i)}$ obtained in step (3) will be applied to normalize *Test_input*2 and the normalized *Test_input*2 can be expressed using *Test_inputm*2 as follows:

$$Test\_inputm2 = \begin{pmatrix} \dfrac{ws_N - Min_{input}^{(1)}}{D_{input}^{(1)}} \\ \dfrac{Ws\_forecast - Min_{input}^{(2)}}{D_{input}^{(2)}} \end{pmatrix} \tag{36}$$

Applying Eqs. (3) and (4) in Section 2.2.1, we can obtain the output of the network and denote it by $O_2 \in \mathbb{R}$. Therefore, the forecasting value of ws_{N+2} is $Ws\_forecast2 = O_2 \times D_{output}^{(1)} + Min_{output}^{(1)}$. The Matlab codes of this step are listed in Code 15.

Code 15. Matlab codes to forecast using established ANFIS (two-step ahead).

```
Test_inputm2 = mapminmax('apply', Test_input2, inputs);

O2=evalfis(Test_inputm2', out_fis1);

Ws_forecast2= mapminmax('reverse', O, outputs);
```

3. Data collection

We select three types of wind data to train and test forecasting models, which are 10-minute wind speed data in wind farm A (this database is denoted by WFD1), 15-minute wind speed data in wind farm B (this database is denoted by WFD2), and 15-minute wind speed data in wind farm C (this database is denoted by WFD3), and wind farms B and C are in the same city in China.

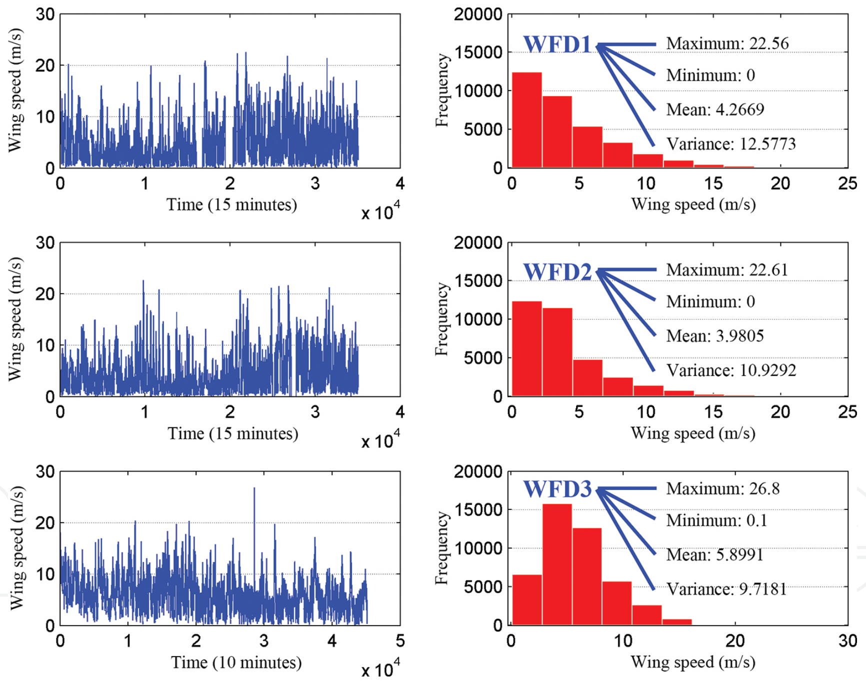

Figure 2. Descriptive information and histograms of three wind farm databases.

In this section, the descriptive information of each database will be provided and we will test the similarity between WFD2 and WFD3 using Friedman test. In the beginning, **Figure 2** shows the three databases and their histograms.

From **Figure 2**, this is apparent that WFD1 and WFD2 have similar maximum and minimum values because they are in the same city. However, WFD1 and WFD2 have different mean and variance values. Thus, we use Friedman test to figure out whether both of two time series are significantly different, which is shown in **Table 1**. In **Table 1**, the *p*-value of Friedman test is very small (2.46e–11), which indicates that WFD1 and WFD2 have a significant difference.

Source	SS	df	MS	Chi-squared	Prob>Chi-squared
Columns	73.3	1	73.2825	44.57	2.46e–11
Interaction	61568.7	17,085	3.6037		
Error	22,649	34,172	0.6628		
Total	84,291	68,343			

Table 1. The result of Friedman test between WFD1 and WFD2.

For WFD3, its mean value is significantly different from others and the histogram of WFD3 is also different from those of other databases. From the description of these three databases, it can be seen that they are very different, which lays a strong foundation for the comparison study.

4. Simulation and results

For these different databases, we will test 1-h-ahead, 4-h-ahead, 1-h average, 4-h average, and 1-day average wind speed forecasting effectiveness by using five methods separately, which is to say that we conduct five experiments of different time scales or time horizons in this comparison study. To test the overall forecasting effectiveness of each model, the testing period is 30 weeks in 2014 and we apply three criteria to evaluate forecasting error, which are the mean absolute percent error (MAPE), root-mean-squared error (RMSE), and mean absolute error (MAE). The formulas of three criteria are expressed as follows:

$$e_t = \hat{y}_t - y_t \tag{37}$$

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\frac{|e_t|}{y_t}\times 100\% \tag{38}$$

$$RMSE = \frac{1}{N}\sqrt{\sum_{t=1}^{N}e_t^2} \tag{39}$$

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|e_t| \tag{40}$$

where t represents the time t, y_t is the actual value in time t, $\hat{y}_t$ is the forecasting value in time t, and N is the length of testing period.

4.1. Experiment I: 1-h-ahead wind speed forecasting

In this experiment, we will test the 1-h-ahead forecasting effectiveness of five models. It should be pointed that 1-h-ahead prediction means a four-step forecast for WFD1 and WFD2 and a means six-step forecast for WFD3 because WFD3 is constructed by 10-min wind speeds. The main results are shown in **Table 2**, which contains MAPE, RMSE, and MAE of each model when forecasting 1-h-ahead wind speed.

Wind farms	Criteria	ANFIS	BPNN	ELM	SVM	ARMA
WFD1	MAE	1.4193	0.8066	0.8335	0.9471	1.4293
	MAPE	43.67%	19.25%	19.51%	30.74%	30.96%
	RMSE	1.5092	0.9083	0.9375	1.0521	1.5574
WFD2	MAE	1.8110	0.7651	0.7649	0.8590	2.9846
	MAPE	49.82%	20.21%	19.76%	28.11%	83.01%
	RMSE	1.8837	0.8611	0.8607	0.9550	3.0937
WFD3	MAE	1.2618	0.9781	0.8571	0.9055	3.0764
	MAPE	25.22%	20.26%	16.29%	19.05%	64.92%
	RMSE	1.3735	1.1019	0.9797	1.0290	3.1724

Table 2. The forecasting results of Experiment I.

From **Table 2**, it is apparent that BPNN has the best performance among these five models in WFD1 and ELM outperforms others in WFD2 and WFD3 by three criteria. By contrast, the forecasting accuracy of ANFSI and ARMA is poor and even cannot be accepted in some wind farms, such as the performance of ARMA in WFD1 and WFD2 and the performance of ANFIS in WFD1 and WFD2. These results show that each wind farm has the corresponding best forecasting model and there is no single model that can perform best in every wind farm by three criteria.

4.2. Experiment II: 4-h-ahead wind speed forecasting

Experiment II shows the 4-h-ahead forecasting effectiveness of wind speed, and the 4-h-ahead forecasting is equal to a 16-step forecast for WFD1 and WFD2 and is a 24-step forecast for WFD3. **Table 3** descripts forecasting results of five models in three wind farms.

In **Table 3**, BPNN is the best model in WFD1 and WFD2 by three criteria and outperforms other models in three databases by MAE and RMSE. ANFIS has the lowest MAPE in WFD3 among these models but has high MAPE, MAE, and RMSE in WFD1 and WFD2. It is similar to Experiment I that ARMA does not have a decent forecasting results in three wind farms.

These results indicate that model's forecasting effectiveness varies a lot when the wind farm changes (such as the effectiveness of ANFIS).

Wind farms	Criteria	ANFIS	BPNN	ELM	SVM	ARMA
WFD1	MAE	1.9441	1.4385	1.4725	1.6147	2.4799
	MAPE	54.41%	35.36%	36.67%	52.28%	55.59%
	RMSE	2.1531	1.6664	1.7062	1.8376	2.7478
WFD2	MAE	1.9937	1.3251	1.3492	1.4246	2.8887
	MAPE	54.96%	34.71%	36.36%	45.85%	78.64%
	RMSE	2.1770	1.5440	1.5647	1.6301	3.1138
WFD3	MAE	1.3250	1.3176	1.3575	1.3951	2.6248
	MAPE	26.04%	26.80%	27.20%	30.12%	53.30%
	RMSE	1.5494	1.5347	1.5924	1.6122	2.8495

Table 3. The forecasting results of Experiment II.

4.3. Experiment III: 1-h average wind speed forecasting

In Experiment III, we will forecast 1-h average wind speed in three wind farms using ANFIS, BPNN, ELM, SVM, and ARMA, and 1-h average wind speed forecasting is a one-step forecast for three wind farms.[3] **Table 4** shows the forecasting effectiveness of each model in this experiment.

Wind farms	Criteria	ANFIS	BPNN	ELM	SVM	ARMA
WFD1	MAE	1.5664	0.8586	0.8315	0.9069	1.2883
	MAPE	44.82%	20.15%	19.33%	24.92%	27.58%
	RMSE	2.0981	1.2376	1.1882	1.2805	1.8759
WFD2	MAE	1.7780	0.7924	0.7607	0.7919	2.5731
	MAPE	47.44%	20.30%	19.35%	21.92%	80.67%
	RMSE	2.3788	1.1838	1.1298	1.1712	3.3596
WFD3	MAE	0.8873	0.7964	0.7943	0.7914	1.1274
	MAPE	16.90%	15.07%	14.95%	14.91%	20.75%
	RMSE	1.1568	1.0568	1.0548	1.0539	1.4833

Table 4. The forecasting results of Experiment III.

[3] It should be pointed that every day has 24 1-h average wind speed.

From **Table 4**, ELM and SVM have decent forecasting accuracy in WFD3, but ELM is better than SVM and other models in WFD1 and WFD2. ANFIS has a good performance in WFD3 and cannot forecast 1-h average wind speed with a reasonable accuracy. Similarly for ARMA, it works well in WFD1 and WFD3 but has a high MAPE in WFD2.

4.4. Experiment IV: 4-h average wind speed forecasting

It is similar to Experiment III, but this experiment shows the one-step-ahead wind speed forecasting effectiveness of each model. **Table 5** demonstrates the results of five models obtained in three wind farms.

Wind farms	Criteria	ANFIS	BPNN	ELM	SVM	ARMA
WFD1	MAE	2.0516	1.6435	1.6209	1.5747	2.1725
	MAPE	55.75%	41.64%	38.76%	36.76%	48.81%
	RMSE	2.6411	2.1411	2.1478	2.1241	2.8977
WFD2	MAE	2.1538	1.4282	1.3489	1.3422	2.4222
	MAPE	55.53%	36.02%	34.27%	30.86%	74.71%
	RMSE	2.8318	1.9498	1.8597	1.8933	3.1035
WFD3	MAE	1.2564	1.2015	1.1483	1.1689	1.6348
	MAPE	23.37%	22.33%	21.13%	21.19%	29.41%
	RMSE	1.5951	1.5249	1.4856	1.5074	2.0953

Table 5. The forecasting results of Experiment IV.

The analyzing results in **Table 5** are quite different from that in **Table 4**. Specifically, SVM model has better performance in WFD1 and WFD2 than other models, and ELM is the best forecasting model in WFD3. For each model, wind speed in WFD3 can be forecasted more accurately than that in WFD1 and WFD2, and AI-based models (ANFIS, BPNN, ELM, and SVM) outperform ARMA in WFD3.

4.5. Experiment V: 1-day average wind speed forecasting

In this experiment, we will test the one-step-ahead forecasting effectiveness based on 1-day average wind speed database. Similar to other experiments, **Table 6** demonstrates each model's forecasting effectiveness.

Table 6 shows that SVM has the best forecasting performance on three criteria in three wind farms. By contrast, ARMA and ANFIS have the higher MAPE, MAE, and RMSE than other models.

Wind farms	Criteria	ANFIS	BPNN	ELM	SVM	ARMA
WFD1	MAE	2.1001	1.9069	1.8377	1.7675	2.5753
	MAPE	52.78%	42.99%	45.70%	36.81%	60.59%
	RMSE	2.5729	2.3784	2.3053	2.2692	3.1655
WFD2	MAE	1.9384	1.9710	1.9038	1.7653	1.9271
	MAPE	51.81%	47.82%	48.19%	38.04%	52.61%
	RMSE	2.4685	2.4758	2.3848	2.3573	2.4679
WFD3	MAE	1.9288	1.8385	1.5886	1.5487	1.8763
	MAPE	32.30%	30.40%	26.39%	24.79%	30.47%
	RMSE	2.4229	2.3061	1.9152	1.9159	2.2568

Table 6. The forecasting results of Experiment V.

5. Conclusion

The inherently intermittent nature and stochastic nonstationarity of wind sources bring great levels of uncertainty to system operators and are an urgent problem in the wind-forecasting field. This chapter introduces some basic forecasting approaches and corresponding procedures to forecast wind speed (detailed codes are attached). After simulating these methods (the testing period is 30 weeks) in MATLAB based on three databases, conclusions can be drawn as follows:

1. None of ANFIS, BPNN, ELM, SVM, and ARMA has the best forecasting performance in all experiments.
2. Based on the experimental results of different forecasting types (1-h-ahead forecasting, 4-h-ahead forecasting, 1-h-average-ahead forecasting, 4-h-average-ahead forecasting, and 1-day-average-ahead forecasting), the best model varies in time scales and time horizons.
3. The forecasting effectiveness differs a lot from database to database (in Experiment II, ANFIS is the best model of which MAPE is 26.04% in WFD3 but has a high MAPE, 54.41%, in WFD1).
4. AI-based models are more suitable for wind speed forecast than ARMA.

According to these conclusions, it is obvious that wind speed forecasting models or systems need to be built up under specific conditions in wind farms and model selection in wind speed forecasting plays a significant role to improve the accuracy.

Acknowledgements

The work was supported by the National Natural Science Foundation of China (grant no. 71573034).

Appendices

In the appendix, we will provide two functions which are coded by Matlab to implement training process (elmtrain.m) and predicting process (elmpredict.m) of ELM. The elmtrain.m is expressed as follows:

Code A1. Matlab function file to establish and train ELM.

```
function [IW,B,LW,TF,TYPE] = elmtrain(P,T,N,TF,TYPE)

% ELMTRAIN Create and Train a Extreme Learning Machine

% Syntax

% [IW,B,LW,TF,TYPE] = elmtrain(P,T,N,TF,TYPE)

% Description

% Input

% P—Input Matrix of Training Set (R*Q)

% T—Output Matrix of Training Set (S*Q)

% N—Number of Hidden Neurons (default = Q)

% TF—Transfer Function:

% 'sig' for Sigmoidal function (default)

% 'sin' for Sine function

% 'hardlim' for Hardlim function

% TYPE—Regression (0,default) or Classification (1)

% Output

% IW—Input Weight Matrix (N*R)

% B—Bias Matrix (N*1)

% LW—Layer Weight Matrix (N*S)

% Example

% Regression:
```

```
% [IW,B,LW,TF,TYPE] = elmtrain(P,T,20,‘sig’,0)

% Y = elmtrain(P,IW,B,LW,TF,TYPE)

% Classification

% [IW,B,LW,TF,TYPE] = elmtrain(P,T,20,‘sig’,1)

% Y = elmtrain(P,IW,B,LW,TF,TYPE)

% See also ELMPREDICT

% Yu Lei,11-7-2010

% Copyright www.matlabsky.com

% $Revision:1.0 $

if nargin < 2

   error(‘ELM:Arguments’,‘Not enough input arguments.’);

end

if nargin < 3

   N = size(P,1);

end

if nargin < 4

   TF = ‘sig’;

end

if nargin < 5

   TYPE = 0;

end

if size(P,2) ~= size(T,2)

   error(‘ELM:Arguments’,‘The columns of P and T must be same.’);

end
```

```
[R,Q] = size(P);
if TYPE == 1
   T = ind2vec(T);
end
[S,Q] = size(T);
% Randomly Generate the Input Weight Matrix
IW = rand(N,R) * 2–1;
% Randomly Generate the Bias Matrix
B = rand(N,1);
BiasMatrix = repmat(B,1,Q);
% Calculate the Layer Output Matrix H
tempH = IW * P + BiasMatrix;
switch TF
   case ‘sig’
     H = 1 ./(1 + exp(-tempH));
   case ‘sin’
     H = sin(tempH);
   case ‘hardlim’
     H = hardlim(tempH);
end
% Calculate the Output Weight Matrix
% find(isnan(T) == 1)
LW = pinv(H') * T';
```

The elmpredict.m is expressed as follows:

Code A2. Matlab function file to forecast using establish ELM.

```
function Y = elmpredict(P,IW,B,LW,TF,TYPE)

% ELMPREDICT Simulate an Extreme Learning Machine

% Syntax

% Y = elmtrain(P,IW,B,LW,TF,TYPE)

% Description

% Input

% P–Input Matrix of Training Set (R*Q)

% IW—Input Weight Matrix (N*R)

% B—Bias Matrix (N*1)

% LW—Layer Weight Matrix (N*S)

% TF—Transfer Function:

% 'sig' for Sigmoidal function (default)

% 'sin' for Sine function

% 'hardlim' for Hardlim function

% TYPE—Regression (0,default) or Classification (1)

% Output

% Y—Simulate Output Matrix (S*Q)

% Example

% Regression:

% [IW,B,LW,TF,TYPE] = elmtrain(P,T,20,'sig',0)

% Y = elmtrain(P,IW,B,LW,TF,TYPE)
% Classification

% [IW,B,LW,TF,TYPE] = elmtrain(P,T,20,⊚sig’,1)

% Y = elmtrain(P,IW,B,LW,TF,TYPE)

% See also ELMTRAIN

% Yu Lei,11-7-2010

% Copyright www.matlabsky.com

% $Revision:1.0 $

if nargin < 6

  error(‘ELM:Arguments’,‘Not enough input arguments.’);

end

% Calculate the Layer Output Matrix H

Q = size(P,2);

BiasMatrix = repmat(B,1,Q);

tempH = IW * P + BiasMatrix;

switch TF

  case ‘sig’

    H = 1 ./(1 + exp(-tempH));

  case ‘sin’

    H = sin(tempH);

  case ‘hardlim’

    H = hardlim(tempH);

end

% Calculate the Simulate Output

Y = (H' * LW)';
```

```
if TYPE == 1

  temp_Y = zeros(size(Y));

  for i = 1:size(Y,2)

    [max_Y,index] = max(Y(:,i));

    temp_Y(index,i) = 1;

  end

  Y = vec2ind(temp_Y);

end
```

Author details

Ping Jiang, Feng Liu[*] and Yiliao Song

*Address all correspondence to: liuf13@lzu.edu.cn or feng.liu.1990@ieee.org

School of Statistics, Dongbei University of Finance and Economics, Dalian, Liaoning, China.

References

[1] I.J. Ramirez-Rosado, L.A. Fernandez-Jimenez, C. Monteiro, J. Sousa, and R. Bessa. Comparison of two new short-term wind-power forecasting systems. Renewable Energy. 2009;34(7):1848–1854. DOI: 10.1016/j.renene.2008.11.014

[2] G. Sideratos, N.D. Hatziargyriou. Wind power forecasting focused on extreme power system events. IEEE Transactions on Sustainable Energy. 2012;3(3):445–454. DOI: 10.1109/TSTE.2012.2189442

[3] P. Zhao, J. Wang, J. Xia, Y. Dai, Y. Sheng, J. Yue. Performance evaluation and accuracy enhancement of a day ahead wind power forecasting system in China. Renewable Energy. 2012;43(1):234–241. DOI: 10.1016/j.renene.2011.11.051

[4] K. Bhaskar, S.N. Singh. AWNN-assisted wind power forecasting using feed-forward neural network. IEEE Transactions on Sustainable Energy. 2012;3(2):306–315. DOI: 10.1109/TSTE.2011.2182215

[5] L. Han, C.E. Romero, Z. Yao. Wind power forecasting based on principle component phase space reconstruction. Renewable Energy. 2015;81(1):737–744. DOI: 10.1016/j.renene.2015.03.037

[6] L. Silva. A feature engineering approach to wind power forecasting: GEFCom 2012. International Journal of Forecasting. 2014;30(2):395–401. DOI: 10.1016/j.ijforecast.2013.07.007

[7] W.P. Mahoney, et al. A wind power forecasting system to optimize grid integration. IEEE Transactions on Sustainable Energy. 2012;3(4):670–682. DOI: 10.1109/TSTE.2012.2201758

[8] C. Croonenbroeck, C.M. Dahl. Accurate medium-term wind power forecasting in a censored classification framework. Energy. 2014;73(1):221–232. DOI: 10.1016/j.energy.2014.06.013

[9] N.B. Karayiannis, M.M. Randolph-Gips. On the construction and training of reformulated radial basis function neural networks. IEEE Transactions on Neural Networks. 2003;14(4):835–846. DOI: 10.1109/TNN.2003.813841

[10] P. Kou, F. Gao, X. Guan. Sparse online warped Gaussian process for wind power probabilistic forecasting. Applied Energy. 2013;108(1):410–428. DOI: 10.1016/j.apenergy.2013.03.038

[11] R.J. Bessa, V. Miranda, A. Botterud, J. Wang, E.M. Constantinescu. Time adaptive conditional kernel density estimation for wind power forecasting. IEEE Transactions on Sustainable Energy. 2012;3(4):660–669. DOI: 10.1109/TSTE.2012.2200302

[12] A. Carpinone, M. Giorgio, R. Langella, A. Testa. Markov chain modeling for very short term wind power forecasting. Electric Power Systems Research. 2015;122(1):152–158. DOI: 10.1016/j.epsr.2014.12.025

[13] P. Pinson. Very short term probabilistic forecasting of wind power with generalized logit–normal distributions. Journal of the Royal Statistical Society: Series C (Applied Statistics). 2012;61(4):555–576. DOI: 10.1111/j.1467-9876.2011.01026.x

[14] E. Erdem, J. Shi. ARMA based approaches for forecasting the tuple of wind speed and direction. Applied Energy. 2011;88:1045–1044. DOI: 10.1016/j.apenergy.2010.10.031

[15] L. Xiao, J. Wang, R. Hou, J. Wu. A combined model based on data pre-analysis and weight coefficients optimization for electrical load forecasting. Energy. 2015;82:524–549. DOI: 10.1016/j.energy.2015.01.063

[16] Z. Guo, J. Zhao, W. Zhang, J. Wang. A corrected hybrid approach for wind speed prediction in Hexi Corridor of China. Energy. 2011;36(3):1668–1679. DOI: 10.1016/j.energy.2010.12.063

[17] J. Wang, J. Hu. A robust combination approach for short-term wind speed forecasting and analysis – combination of the ARMA (Autoregressive Integrated Moving Average), ELM (Extreme Learning Machine), SVM (Support Vector Machine) and LSSVM (Least

Square SVM) forecasts using a GPR (Gaussian Process Regression) model. Energy. 2015;93(1):41–56. DOI: 10.1016/j.energy.2015.08.045

[18] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew. Extreme learning machine: theory and applications. Neurocomputing. 2006;70:489–501. DOI: 10.1016/j.neucom.2005.12.126

[19] J. Zhao, J. Wang, F. Liu. Multistep forecasting for short-term wind speed using an optimized extreme learning machine network with decomposition-based signal filtering. Journal of Energy Engineering. Article ID: 04015036, 2015. DOI: 10.1061/(ASCE)EY.1943-7897.0000291.

[20] Z. Zhang, Y. Song, F. Liu, J. Liu. Daily average wind power interval forecasts based on an optimal adaptive-network-based fuzzy inference system and singular spectrum analysis. Sustainability. 2016;8(2): 125. DOI: 10.3390/su8020125.

[21] S.J.R. Jiang. ANFIS: adaptive-network-based fuzzy inference system. IEEE Transactions on Systems, Man and Cybernetics. 1993;23(3):665–685. DOI: 10.1109/21.256541