Wind Turbine Power Estimation by Neural Networks With Kalman Filter Training on a SIMD Parallel Machine

Shuhui Li Donald C. Wunsch Edgar O'Hair Michael G. Giesselmann

Department of Electrical Engineering
Texas Tech University
Lubbock, TX 79409

Abstract

We use a multi-layer perceptron (MLP) network to estimate wind turbine power generation. Wind power can be influenced by many factors such as wind speeds, wind directions, terrain, air density, vertical wind profile, time of a day, and seasons of a year. It is usually important to train a neural network with multiple influence factors and big training data set. We have parallelized the Extended Kalman Filter (EKF) training algorithm, which can provide fast training even for large training data sets. The MLP network is then trained with the consideration of various possible factors, which can cause influence on turbine power production. The performance of the trained network is studied from the point of view of information presented to the network through network inputs regarding to different affecting factors and large training data set covering all the seasons of a year.

I. Introduction

Wind energy conversion systems appear as an attractive alternative for electricity generation. To maximize the use of wind generated electricity when connected to the electric grid, it is important to estimate and predict power produced by wind farms. However, wind power can be affected by many factors and usually fluctuates rapidly, imposing considerable difficulties on the management of combined electric power systems. Our data is from the Central and South West wind farm near Fort Davis, Texas.

The remainder of the paper proceeds as follows. Section II gives a brief description about the wind farm and the MLP network for wind power estimation. Section III introduces EKF training. Section VI proposes multi-stream decoupled EKF (DEKF) training for the purpose of parallel processing. Section V presents the multi-stream DEKF training on a SIMD parallel machine. Section VI investigates network training corresponding to various influence factors and big training data set. Finally, Section VII offers conclusions.

II. The Wind Farm and the MLP Network for Wind Power Estimation

The Fort Davis wind farm consists of twelve 500kw turbines and two meteorological towers that measure wind velocity and direction at three elevations (10, 30, and 40m). The hub height of the turbine is 40m above the ground. A more complete description of the Fort Davis wind farm is given in [1]. The industry standard is to relate the power to the hub height wind velocity. Such a relationship implies that the velocity at the hub height is known and that the velocity profile [2] (the difference in velocity as a function of height from the bottom to the top of the turbine blade) is known and does not change. Due to the variable terrain and the locations of the turbines and the meteorological towers at Fort Davis, the actual wind velocity and profile for each turbine is unknown.

Both the turbines and the meteorological towers record numerous data sets, including power produced by each turbine, 10, 30 and 40m wind velocities and wind directions, RES (resultant vector average) wind speeds, and STD (standard deviation) wind directions from each meteorological tower, and air pressure and temperature from one meteorological tower. The measurements used in the paper are 10 minute averages. All the measurements from the two meteorological towers are assumed to have some unknown relationship with turbine power production. Millions of recordings can usually be obtained each year from the twelve turbines and two meteorological towers.

The neural network for wind turbine power estimation is a MLP network for each turbine (see detail in [1]). In addition to 40m wind speeds and directions in [1], the raw input patterns could include more information such as 10

0-7803-5529-6/99/$10.00 ©1999 IEEE

or 30m wind speeds and wind directions, RES wind speeds, STD wind directions, vertical wind profile, time of a day, and air density. The network output is power generated corresponding to each different input pattern.

It is often necessary to investigate the network training with dozens of network inputs (or influence factors) and big training data set. In order to deal with the advanced training problem, we us Extended Kalman Filter training algorithm, realized on a SIMD parallel machine, to train the network.

III. Extended Kalman Filter Training

The extended Kalman filter has served as the basis for many recent neural network training algorithm studies because of its strong learning capability and good convergence properties. Singhal and Wu [3] demonstrated that the extended Kalman filter could be used to train MLP networks by treating the weights of the network as an unforced nonlinear dynamical system with stationary states. The network weights are the states the Kalman filter attempts to estimate, and the output of the network is the measurement used by the Kalman filter as shown in the equations below [3, 4]

$$\mathbf{w}[n+1]=\mathbf{w}[n]+\mathbf{q}[n] \quad (1)$$

$$\mathbf{d}[n]=h_n(\mathbf{w}[n], \mathbf{x}[n])+\mathbf{r}[n] \quad (2)$$

where at time n, $\mathbf{w}[n]$ is a vector consisting of all the weights in the network, $\mathbf{x}[n]$ is the network input vector from the training set, $\mathbf{d}[n]$ is the corresponding desired output vector, $h_n(\bullet)$ defines the nonlinear relationship between the inputs, weights and outputs of the network, $\mathbf{q}[n]$ is the driving noise in the system model, and $\mathbf{r}[n]$ is the measurement noise.

For a nonlinear MLP network, the usual Kalman filtering algorithm can be used only if the model of the system is linearized, resulting in the extended Kalman filter (EKF) [4]. The EKF training algorithm is usually realized with the node decoupled EKF (DEKF) algorithm [5] which ignores the interaction between certain weight groups so that the error covariance matrix can be approximated into block-diagonal form. The update equations of DEKF training algorithm are (3)~(6) [6].

$$\mathbf{A}(n)=[\eta(n)^{-1}\mathbf{I}+\sum_{j=1}^{g}\mathbf{H}_j(n)^T\mathbf{P}_j(n)\mathbf{H}_j(n)]^{-1} \quad (3)$$

$$\mathbf{K}_i(n)=\mathbf{P}_i(n)\mathbf{H}_i(n)\mathbf{A}(n) \quad (4)$$

$$\mathbf{w}_i(n+1)=\mathbf{w}_i(n)-\mathbf{K}_i(n)\xi(n) \quad (5)$$

$$\mathbf{P}_i(n+1)=\mathbf{P}_i(n)-\mathbf{K}_i(n)\mathbf{H}_i(n)^T\mathbf{P}_i(n)+\mathbf{Q}_i(n) \quad (6)$$

The sizes of the above matrices and vectors depend on two factors: the number of nodes at network output layer (L) and the number of weights in each weight group. When L=1, $\mathbf{A}(n)$ and $\xi(n)$ become scalars, $\mathbf{H}_i(n)$ and $\mathbf{K}_i(n)$ reduce to vectors, but $\mathbf{P}_i(n)$ and $\mathbf{w}_i(n)$ are unchanged.

IV. Multi-Stream EKF Training

The EKF training is usually realized with instance-by-instance updates rather than performing updates at the end of a batch of training instances. In order for the EKF algorithm to be realized on a parallel machine more efficiently, we hope to compute a single update on the basis of a batch of training examples. But this is different from the gradient averaging in standard BP batch mode training [3, 6]. In the case of EKF training, for a batch size of T patterns and a neural network with L outputs, it is treated as that of training a single shared-weight network with L×T outputs [7] (Figure 1), i.e., T data streams which feed T networks constrained to have identical weights are formed from the training set. At each training cycle, one pattern from each stream is presented to its respective network and L outputs for each stream are computed. A single weight update is then calculated and applied equally to each stream's network.

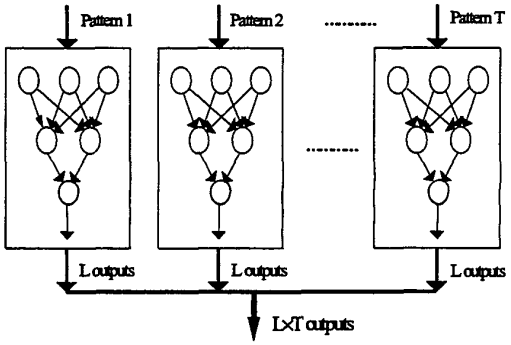

Fig. 1 Multi-Stream network structure

The update equations for multi-stream DEKF are basically the same as Eqs. (3)~(6). For the case of multi-stream network, we have T identical networks for T data stream so that the number of weight variables is unchanged compared to the single stream network. This makes the sizes of the approximate error covariance matrices $\mathbf{P}_i(n)$ as well as the weight vectors $\mathbf{w}_i(n)$ remain unchanged. However, the overall network outputs or measurements will increase from L to L×T so that the columns of the derivative matrices $\mathbf{H}_i(n)$ and Kalman gain matrices $\mathbf{K}_i(n)$ will increase from L to L×T.

The major additional computational burden for the multi-stream DEKF algorithm is the inversion required to obtain the matrix $\mathbf{A}$, whose dimension is T times larger for the multi-stream network model than that of a single-stream

network model. Because $\mathbf{A}$ requires a matrix inversion, it is often not practical to perform a larger batch update on a sequential computer.

V. Multi-Stream EKF Training on MasPar Parallel Machine

The MasPar parallel machine is a massively parallel SIMD supercomputer system which supports the data-parallel programming model [8]. The MasPar parallel machine at Texas Tech contains 2048 (32×64) processing elements (PEs), each of which has private (unshared) memory and operates on a distinct data element, i.e., different from those on other PEs. Data elements may be modified on all PEs in a single operation. All PEs can send or fetch messages concurrently to or from other PEs (Figure 2).

There are three major elements to the MasPar programming model (Figure 3): the Front End workstation, the Data-Parallel Unit (DPU) and communication both within the DPU and between the Front End and DPU. The Front End is usually used for control operations such as the user interface, initial data set-up, and final collection. The DPU handles most of the computations and can be looked as a massively parallel machine, the PE array, with an added scalar processor, the Array Control Units (ACU). The ACU broadcasts instructions to all the PEs in the PE array. The data in PE memory space is distributed, and each instruction from the ACU operates on all PEs simultaneously (Figure 2).

The training of the neural network on the parallel machine begins with the initial data to be read in or set up at the Front End workstation. The raw data such as wind speeds, wind directions, air density, vertical wind profile, and turbine power are read into the Front End, divided into several batches (2048 sets of data per batch), and then mapped to the 2048 processors in the PE array for each batch of data. Data set up at Front End contains the number of network layers, number of neurons in each layer, and the number of batches of training data, and are then transmitted to ACU. The initial values for weight vectors and error covariance matrices are used only for the network training at DPU and generated in the ACU and PE array respectively. The initial weight vectors are small random values, and the matrices $\mathbf{P}_i(0)$ are initialized as a diagonal matrix on PE array.

The network training follows the following procedure: generation of $\mathbf{H}_i$, computation of $\mathbf{A}^{-1}$, matrix inversion to get $\mathbf{A}$, computation of Kalman gain $\mathbf{K}_i$, updates of error covariance matrices $\mathbf{P}_i$ and weight $\mathbf{w}_i$ for each 32 patterns. After the updates for the first 32 patterns, another 32 patterns will be picked up to repeat the same computations. This process continues until all the patterns are used up. Then, a parallel randomization for all patterns in the PE array will be executed and the network will be retrained with the randomized patterns if the stopping criterion is not reached.

VI. Training the Neural Networks with Multiple Affecting Factors and Data Set Covering Different Seasons

The wind turbine power production depends mainly on the energy contained in the wind. The basic unit of measurement for this resource is wind power density [2], or power per unit of area normal to the wind azimuth, calculated as Eq. (7), where P_W is wind power density (W/m^2), ρ is air density (kg/m^3), and V is horizontal component of the mean free-stream wind velocity (m/s).

$$P_W=0.5\rho V^3 \qquad (7)$$

According to Eq. (7), the performance of a wind energy conversion system is first affected by wind speed because wind power is proportional to cube of wind speed.

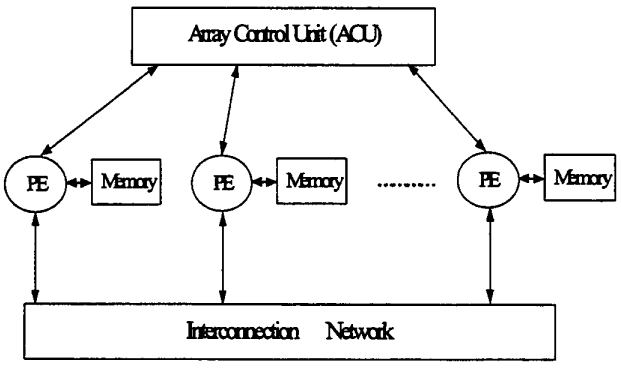

Figure 2 The MasPar SIMD supercomputer system architecture

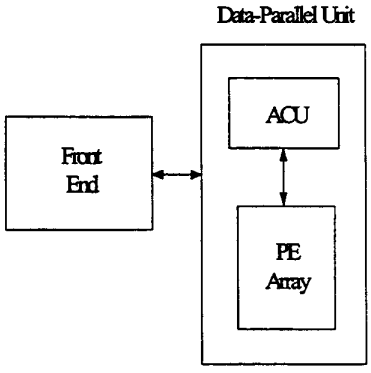

Figure 3 The MasPar programming model

However, actual wind speed is different from turbine to turbine because of the terrain. Wind speeds also vary with elevation above the ground. The mean horizontal wind speed is zero at the earth's surface and increases with altitude in the atmospheric boundary layer (Figure 4). This variation of wind speed with elevation is referred to as the vertical profile of the wind speed or the wind shear, and is supposed to have important influence on turbine power production.

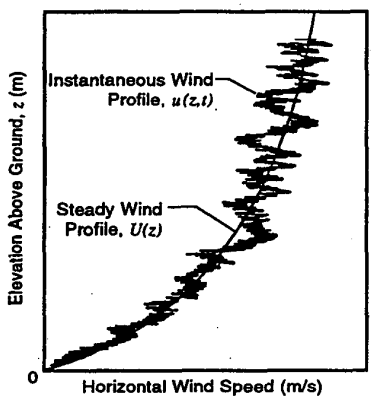

Figure 4 Typical vertical profiles of the wind speed

The air density ρ is another factor that can influence the energy contained in the wind. According to [2], the ρ is related to temperature and barometric pressure as Eq. 8 in which T is temperature in °C, and P is atmospheric pressure in mb.

$$\rho = \frac{0.0012754}{1+0.00366T} \times P \qquad (8)$$

From Eq. (7), it can be seen that wind power is linearly proportional to air density. However, it is difficult to see this linear relationship directly from measured data when wind speeds as well as other factors can cause influence on turbine power production simultaneously.

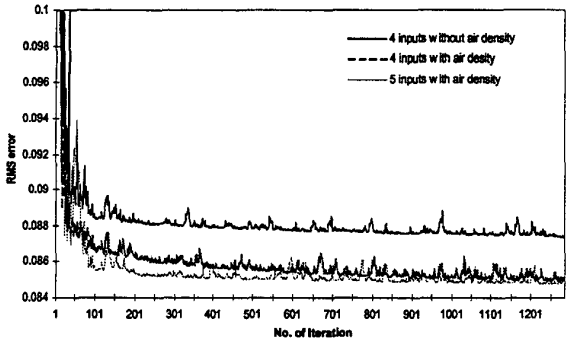

Figure 5 Comparison of RMS errors among three different network input schemes (Turbine 6)

We first trained the neural network with air density as an additional consideration. In all the following training schemes, the 40m wind speeds and wind directions are always taken as the network inputs because wind speed and terrain are two most important factors to affect wind power. The network is trained by two schemes. In the first scheme, we use five network inputs, i.e., two 40m wind speeds, two 40m wind directions, and air density, and the desired output is normalized measured turbine power. In the second scheme, the network is trained with four inputs (two 40m wind speeds and two 40m wind directions) but the desired network output is the preprocessed turbine power which is obtained by dividing measured turbine power by air density.

The network learning process (Figure 5) shows that the RMS error is statistically very close for the two schemes. It is found if there is certain relationship between network inputs and outputs, the network will try to learn it.

The information about vertical wind profile can be presented to the network in several ways. Here we use three schemes to train the network. The first is to include two wind shear exponents Ks computed from wind speeds measured at three elevations [9] from the two meteorological towers as additional two network inputs. The second is to introduce two 10m measured wind speeds from the two meteorological towers as additional network inputs. The third is to involve time of a day as an additional network input because wind shear also exhibits a diurnal cycle, which can be explained in qualitative terms by following a typical diurnal cycle of heating and cooling [2].

Among the three schemes, the first one has the highest information regarding to wind shear and the third has the lowest. The network training statistically shows (Figure 6) the more the information presented through network inputs to the network, the smaller the RMS error, and the better the trained network performance. It is difficult to

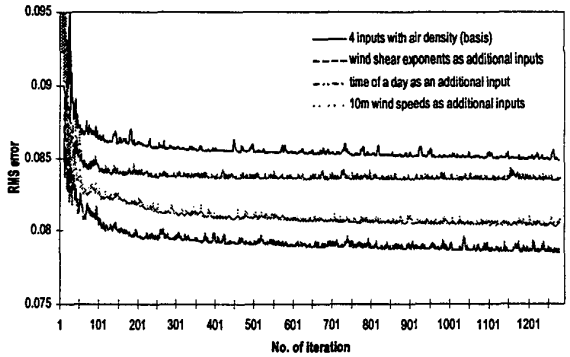

Figure 6 Comparison of RMS errors for three network input schemes regarding to wind shear (Turbine 6)

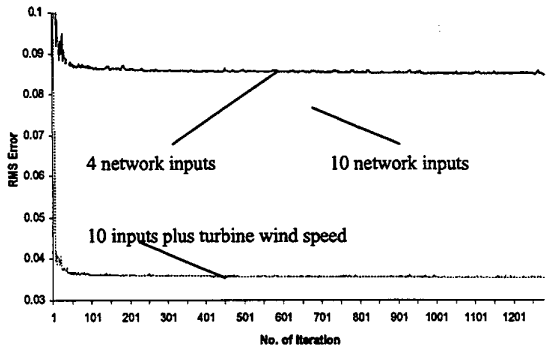

Figure 7 RMS error after intruding turbine wind speed

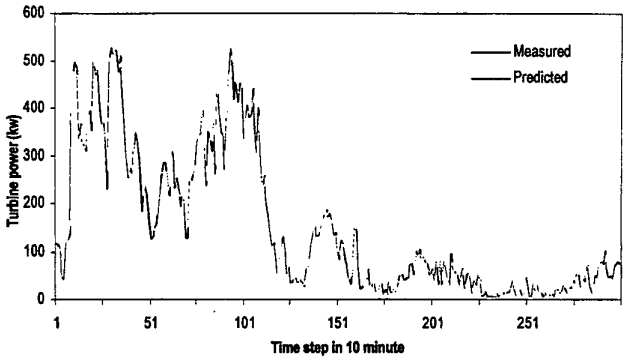

Figure 8 Comparison between turbine power generation and prediction (No. 6)

understand this by just looking at the training data set and from curve fitting point of view.

The network is finally trained with the consideration of various influence factors and training data set covering equally all the seasons of a year and different levels of wind speed. Even though we have improvement in the RMS error (Figure 7, 10 network inputs) and the trained network performance, the RMS error cannot be reduced as small as we expected. By looking at training data set, it seems patterns are different from each other and ideally we should have a training algorithm to allow the RMS error as low as we hope. But this kind of training algorithm actually does not exist. It is not the training algorithm but the limited information presented to the network through network inputs and data that prevent us to get a well trained network. It is found the farther the turbine away from the two meteorological towers, the larger the RMS, and the worse the trained network performance. However, with any additional wind information close to a wind turbine, the RMS error can reduce greatly (Figures 7) and the measurements and network predictions match much better (Figure 8).

VII. Conclusions

The extended Kalman filter training algorithm has strong learning capability and superior convergence properties. The parallel DEKF algorithm can provide very fast training speed and makes it more realistic to train a neural network with many influence factors and a big data set.

We have used neural networks to estimate wind turbine power generation for a wind farm with complicated terrain. The most important factors to affect wind turbine power production are wind speed and terrain. The farther away a wind turbine is from the meteorological towers, the weaker the terrain information, and the worse the trained network performance. However, for a wind farm, the number of meteorological towers needed depends on the complexity of the terrain, the size of the wind farm, and how the meteorological towers are arranged.

VIII. References

[1] S. Li, D. Wunsch, E. O'Hair and M. Giesselmann, "Neural network for wind power with compressing function", *IEEE International Conference on Neural Network*, Houston, June 1997, pp. 115-120.

[2] W. Frost and C. Aspliden, "Characterics of the wind", *in Wind Turbine Technology*, David A. Spera, AMSE Press, chap. 8, 1995, pp. 371-445.

[3] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended Kalman filter algorithm", *in Advances in Neural Information Processing Systems*, San Mateo, CA: Morgan Kaufmann, 1989, pp. 133-140.

[4] F. L. Lewis, *Optimal Estimation: with an Introduction to Stochastic Control Theory*, John Wiley & Sons, 1986.

[5] G. V. Puskorius and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks", *in International Joint Conference on Neural Networks*, Seattle 1991, pp. 771-777.

[6] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks", *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, March 1994, pp. 279-297.

[7] L. A. Feldkamp and G. V. Puskorius, "Training Controller for robustness: multi-stream DEKF", *IEEE International Conference on Neural Network*, June 1994, pp. 2377-2382.

[8] Blank, T., "The MasPar MP-1 Architecture", Proceedings of IEEE CompCon, February 1990.

[9] Faruk, Wasim, "Initial evaluation of a utility connected wind farm", Master's Thesis, Department of Electrical Engineering, Texas Tech University, 1996.