Convergence Study in Extended Kalman Filter-based Training of Recurrent Neural Networks

Xiaoyu Wang and Yong Huang, *Member, IEEE*

***Abstract*— Recurrent neural network (RNN) has emerged as a promising tool in modeling nonlinear dynamical systems, but the training convergence is still of concern. This paper aims to develop an effective extended Kalman filter-based RNN training approach with a controllable training convergence. The training convergence problem during extended Kalman filter-based RNN training has been proposed and studied by adapting two artificial training noise parameters: the covariance of measurement noise (R) and the covariance of process noise (Q) of Kalman filter. The R and Q adaption laws have been developed using the Lyapunov method and the maximum likelihood method, respectively. The effectiveness of the proposed adaption laws has been tested using a nonlinear dynamical benchmark system and further applied in cutting tool wear modeling. The results show that the R adaption law can effectively avoid the divergence problem and ensure the training convergence, whereas the Q adaption law helps improve the training convergence speed.**

***Index Terms*— Adaptive training algorithm, extended Kalman filter, recurrent neural network, training convergence.**

I. Introduction

NEURAL network (NN) is a computational model emulating the information processing capacity of biological NNs. It usually consists of an interconnected group of artificial neurons, propagating and processing information through weighted connections of networked neurons. Based on its architecture organization, NN can be classified into two main classes: single-layer or multilayer feedforward NN (FFNN) and recurrent NN (RNN) [1], which has at least one internal and/or external feedback loop. The presence of feedback loops has a profound impact on the network learning capacity and its performance in modeling nonlinear dynamical phenomena [1].

NN can be viewed as a multi-input and multioutput nonlinear system having a layered structure, and its weight learning/training process can be regarded as a parameter estimation process for such a nonlinear system [2]. Two issues are of great importance in NN training: how to avoid training divergence and how to converge fast. Network training convergence is still a challenge in modeling input-output mapping relationships using NN, especially RNN. RNN training is still an open topic because adjustments of network parameters such as the network weight can affect the entire NN state variables during the network evolution due to the inherent feedback and distributive parallel structure [3] and training is usually complex and might be divergent [4]. Numerous network parameter sensitivity and optimization studies have been conducted for improved network performance [5], [6]. It should be pointed out that training convergence is different from the state or output convergence, which is usually of concern in applying the trained RNN for associative memory applications [7].

Backpropagation through time, real-time recurrent learning, and extended Kalman filter (EKF) are the most popular RNN training algorithms [1]. Various approaches have been proposed to improve training convergence of these algorithms. For example, a modified backpropagation algorithm is found to be better than the standard backpropagation algorithm in terms of convergence and generalization performance [8]. However, among these training algorithms, EKF has often been favored in terms of its training efficiency and accuracy [9]–[11] and has been continuously improved such as by the use of a stable bounding ellipsoid algorithm [12], [13]. Unfortunately, training convergence of EKF-based RNN is still not well studied [14]. Up to the present, only a few studies have been conducted on convergence of EKF-based NN training [14], [15] including RNN training [14], unfortunately, the authors have introduced many assumptions to make these pioneering studies less generic and less efficient. For effective implementation of EKF-based RNN training, some theoretical studies on training convergence must be performed and tested with some applications.

In previous studies, EKF has been used to train the fully forward connected neural network (FFCNN) [16] and RNN [17] with applications in cutting tool wear modeling. Specifically, EKF-based optimized RNN was developed to model tool wear in hard turning without any consideration of possible training divergence [17]. The objective of this paper is to develop an effective EKF-based RNN training approach with a controllable training convergence, which can be viewed as an extension of [17] to develop a robust and convergent training algorithm for various process modeling applications. This paper is organized as follows. First, the convergence study for general dynamical system and the divergence challenges in EKF-based parameter estimation are introduced. Then, a general RNN architecture used in this paper is illustrated and the adaptive training algorithm is proposed based on the Lyapunov method and the maximum likelihood method. The

Manuscript received March 4, 2010; revised January 22, 2011; accepted January 23, 2011. Date of publication March 10, 2011; date of current version April 6, 2011. This work was supported in part by the South Carolina Space Grant Consortium and the National Aeronautics and Space Administration Ames Research Center, California.

X. Wang was with Clemson University, Clemson, SC 29630 USA. He is now with the Emerging Pathogens Institute, University of Florida, Gainesville, FL 32611 USA (e-mail: xywang@ufl.edu).

Y. Huang is with Clemson University, Clemson, SC 29630 USA (e-mail: yongh@clemson.edu).

Digital Object Identifier 10.1109/TNN.2011.2109737

proposed algorithm is further tested in training RNN to model a nonlinear dynamical benchmark system and a cutting tool wear progression. Finally, some conclusions and future work are drawn based on this paper. While this paper focuses on the EKF-training method, it can be further extended for the unscented Kalman filter-based training algorithm [18], which is attracting more and more attention in RNN training [19].

II. Preliminary

A. Convergence Study in EKF Training

While EKF has been proved to be very useful in a wide variety of estimation or prediction applications, its effectiveness can be nullified by its divergence [20], which can be classified as follows [21]: 1) apparent divergence, in which the associated errors do not approach infinity but the errors are too large to allow the estimates to be useful; and 2) true divergence, in which the mean square errors of estimation can actually approach infinity as training goes on, and this true divergence is of interest in this paper.

Several approaches have been proposed to deal with the divergence problem in EKF [22]: 1) increase the arithmetic precision; 2) artificially add white noise to the (noiseless) process equation; 3) use square root Kalman filters; 4) make the state estimation error covariance matrix P symmetric; 5) use a fading-memory Kalman filter; and 6) adapt filter parameters. Evaluation of the effectiveness of the aforementioned approaches is often difficult and case-dependent. The first approach is suitable for applications dealing with hardware implementation in which the high precision in hardware is often prohibitive. The second to fourth approaches aim to make the covariance matrix P nonnegative definite and/or symmetric. During the filtering process, the covariance matrix P may fail to meet the nonnegative definite and/or symmetric requirements, resulting in the divergence problem. This can be alleviated by artificially adding a process noise (the second approach), and the magnitude of the additive noise is chosen to be large enough to ensure that the P matrix is nonnegative [23]. The square-root Kalman filter is a more refined method to solve this divergence problem, and the covariance matrix is propagated in a square-root form by using the Cholesky factorization. However, the square-root algorithm is computationally intensive, which makes it less attractive in engineering applications [24]. The fading-memory filter is another way of forcing the filter to forget measurements in the distant past and place more emphasis on recent measurements, however, it may result in the loss of optimality of the Kalman filter [22]. The sixth approach is of interest in this paper by adapting the covariance of measurement noise (R) and the covariance of process noise (Q) of Kalman filter. It is recognized that the poor statistics about R and Q may cause the divergence problem in estimation using the Kalman filter [25], so this paper will investigate the EKF training algorithm stability in RNN training by adaptively adjusting the two noise covariances R and Q.

The stability of a dynamical system is usually evaluated using the Lyapunov theorems, which give a precise characterization of valid energy functions in the vicinity of equilibrium points [26]. Lyapunov stability is concerned mainly with stability of equilibrium points, and a Lyapunov stable system is a system for which the states remain bounded for all time [27]. In RNN training, the equilibrium points can be viewed as the optimal weight solutions that minimize the mean square error of the outputs of the NN. The training process aims to find the optimal weights as the system's equilibrium points, and the Lyapunov indirect method is used here to study the convergence of training process by adapting R.

While the training convergence is first guaranteed by adapting R, the process noise parameter Q is to be estimated to accelerate the training process, which needs the simultaneous estimation of the noise statistics and the update of the Kalman filter gain. The noise covariance matrixes can be estimated through the Bayesian estimation [28], the correlation method [29], the covariance matching method [29], and the maximum likelihood method [30]. The maximum likelihood method is favored here, because it is more efficient, consistent, and suitable for online applications. It should be pointed out that this method may generate biased estimates for small sample sizes. However, because the maximum likelihood estimates tend to have the true value of the estimated variable close to the center of their distributions, the bias is often negligible [30].

B. RNN Structure

The RNN network structure of interest is introduced here. FFCNN proposed by Werbos [31] is adopted as the backbone of the proposed RNN. The proposed RNN is evolved from FFCNN by introducing the intra-neuron internal recurrency inside its hidden section. As shown in Fig. 1, the RNN is comprised of m neurons in its input section, h neurons in its hidden section, and n neurons in its output section. In addition to the forward connections as in FFCNN, each neuron in its hidden section of RNN takes feedback connections from neurons right to it (the dashed lines in Fig. 1). This internal recurrency makes RNN fundamentally different from FFCNN since it not only operates on the input space but also on the internal state space, which makes it more suitable for modeling of nonstationary and dynamical systems [1].

For each neuron i, its output at the time step k, $x_i(k)$, is determined by the neuron's activation function $f_i(\cdot)$ and net input $net_i(k)$

$$x_i(k) = f_i(net_i(k)) \qquad 1 \le i \le m+h+n. \tag{1}$$

The net input for neuron i in the input and output sections is defined as

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij} x_j(k) \qquad i \notin [m+1, m+h]. \tag{2}$$

The net input for neuron i in the hidden section is defined as

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij} x_j(k) + \sum_{j=i}^{m+h} w_{ij} x_j(k-1) \qquad m < i \le m+h \tag{3}$$

where W_{ij} is the connection weight from neuron j to neuron i and the weight matrix W is formed by W_{ij}. Since the feedback connections are introduced in the hidden neuron section, the

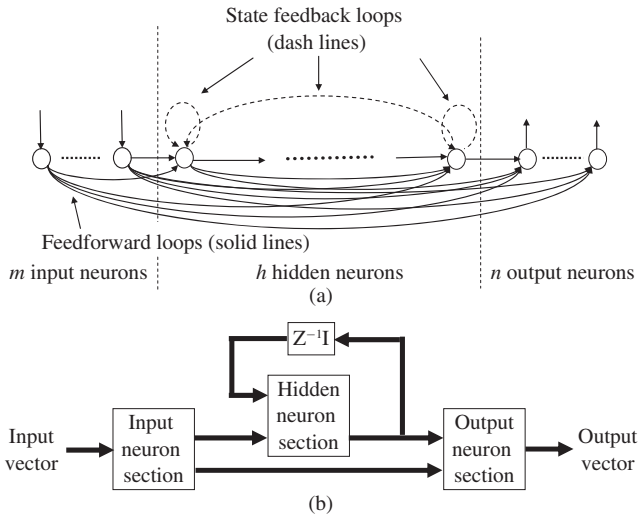

Fig. 1. Proposed RNN. (a) Architecture. (b) State-space representation.

net input of each neuron in this section is comprised of two parts: the weighted summation of outputs at the current step from the neurons left to it and the weighted summation of outputs at the previous step of hidden neurons right to it for this one-step internal recurrency network. Instead of a fully connected RNN [31], the partially connected recurrent network structure is adopted to mimic a state-feedback system in this paper. It is of interest here to develop a convergent EKF-based training algorithm, and the developed algorithm can be further extended for training of fully connected recurrent networks.

C. EKF-Based Training Algorithm

Generally, the EKF-based training algorithms can be divided into two classes: parallel EKF and parameter-based EKF [32]. For parallel EKF, both the network states (the neuron outputs) and weights are treated as states to be estimated by EKF [33], for parameter-based EKF, only the weights are viewed as states to be estimated [32]. As favored in [31] and [32], the parameter-based EKF is favored in this paper which treats the training process as a filtering process with the following formulation in estimating the optimal weight(s):

$$\vec{x}_r(k+1) = f(\vec{x}_r(k), \vec{u}(k), \vec{w}_r(k)) \tag{4}$$

$$\vec{w}_r(k+1) = \vec{w}_r(k) + \vec{\eta}(k) \tag{5}$$

$$\vec{y}^*(k) = h(\vec{x}_r(k-1), \vec{w}_r(k), \vec{u}(k)) + \vec{v}(k) \tag{6}$$

where $\vec{x}_r(k-1)$ is the column vector composed of neuron output x_i $(1 \le i \le m+h+n)$ at the previous time step, $\vec{u}(k)$ is the input vector, $\vec{y}^*(k)$ is the measurement vector, $\vec{\eta}(k)$ and $\vec{v}(k)$ are the process and measurement noise, respectively, and $\vec{w}_r(k)$ is the optimal weight vector to be estimated. A weight vector is a column vector transformed from the corresponding weight matrix by cascading the rows of the weight matrix. It is essentially a filtering problem to estimate the unknown $\vec{w}^*$ using EKF. Ideally, $\vec{w}_r(k)$ should be a constant, and (5) can be written as $\vec{w}_r(k+1) = \vec{w}_r(k)$. However, the artificial process noise $\vec{\eta}(k)$ is often added in (5) to provide more flexibility in tuning the filter.

Remark 1: The neuron output $\vec{x}_r(k)$ is not to be estimated under this framework; instead, it is determined using (4) by assuming a known initial state $\vec{x}_r(0)$. Using such a representation, only the weight vector $\vec{w}_r(k)$ is to be estimated by EKF. Here, (4)–(6) are used to describe the network dynamics without including $\vec{x}_r(k)$ estimation during the training process because of the following reasons.

1) The initial state of neuron outputs can be assumed known based on physical conditions (usually zero initial conditions) for some applications [31], [34].
2) While $\vec{x}_r(0)$ can be estimated with given data in training, it cannot be estimated in testing [31].
3) The training convergence instead of the network representation is of interest here, and the developed algorithm can be further integrated with the parallel EKF training algorithm.

The EKF algorithm was first introduced to train neural networks by Singhal and Wu [35]. Compared to the most widely adopted backpropagation (BP) algorithm, the EKF training algorithm has the following advantages. 1) The EKF algorithm helps to reach the training steady state much faster for non-stationary processes [36]. 2) The EKF algorithm excels the BP algorithm when the training data is limited [32]. Hence the EKF learning algorithm [1] is favored here, and the EKF-based network weight updating at the time step k is introduced as follows:

$$\begin{cases} \vec{w}(k+1) = \vec{w}(k) - K(k)\left(\vec{y}(k) - \vec{y}^*(k)\right) \\ \vec{y}(k) = h\left(\vec{x}(k-1), \vec{w}(k), \vec{u}(k)\right) \end{cases} \tag{7}$$

$$K(k) = P(k)H(k)\left[R(k) + H(k)^T P(k) H(k)\right]^{-1} \tag{8}$$

$$P(k+1) = Q(k) + \left[I - K(k)H(k)^T\right] P(k) \tag{9}$$

where $\vec{w}(k)$ is the estimate of the optimal weight vector $\vec{w}^*$ at time step k, $\vec{w}(k+1)$ is the estimate of weight vector at the next time step, K is the Kalman gain matrix, $\vec{y}(k)$ is the output of RNN which is composed of the outputs of neurons in the output section x_i $(m+h < i \le m+h+n)$, $\vec{y}^*(k)$ is the target value for $\vec{y}(k)$, $\vec{x}(k-1)$ is the vector composed of neuron output x_i $(1 \le i \le m+h+n)$ at the previous time step $(k-1)$ based on $\vec{w}(k-1)$, $h(\cdot)$ defines network output as a function of $\vec{x}(k-1)$, the weight vector $\vec{w}(k)$, and the input $\vec{u}(k)$, H is the orderly derivative matrix $(\partial^+ \vec{y}/\partial \vec{w})(k)$ of the network outputs with respect to the trainable network weights which is computed using the orderly chain rule considering all the possible connections contributing to the network outputs as shown in the Appendix, R is the covariance matrix of measurement noise, Q is the covariance matrix of process noise, P is the covariance matrix of the estimation error, I is the identity matrix, and k and $k+1$ represent the kth and $(k+1)$th (one step after) time steps, respectively.

Remark 2: The process noise $\vec{\eta}$ and measurement noise $\vec{v}$ are artificial noises, and the corresponding covariance matrices Q and R are assumed as diagonal matrices as $\eta(k) \sim N(0, R(k))$ and $v(k) \sim N(0, Q(k))$.

Remark 3: Covariance of measurement noises for outputs is assumed as uncorrelated but with the same variance $r(k)$ as $R(k) = r(k)I$.

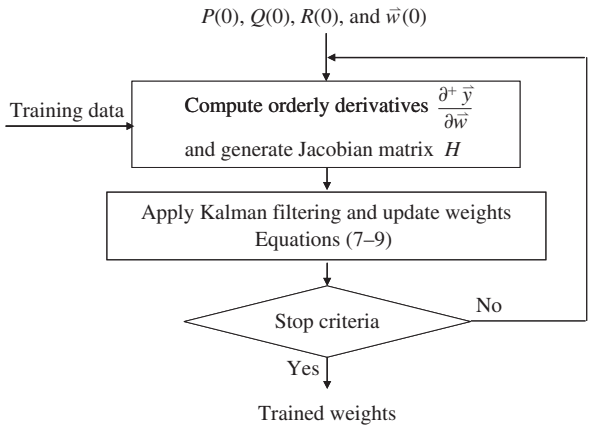

Fig. 2. EKF-based RNN training flowchart.

The RNN training process is illustrated in Fig. 2. The parameters of EKF, P, Q, and $\vec{w}$ are first initialized, and then the orderly derivatives of the network's outputs with respect to the weight vectors ($\partial^+\vec{y}/\partial\vec{\mathbf{w}}$) are calculated using the chain rule [16], forming a Jacobian matrix H. Finally, the Kalman filter (7)–(9) are applied to train the network weights until certain stop criteria are met.

III. Math Adaptive Training Algorithm Development

A. R Adaption Law

The covariance of measurement noise R describes the statistics of network modeling error, and this information is generally not available for a RNN training process. As seen from (8), a small R value might lead to a large Kalman gain, which may make training divergent. For this training divergence problem, a dead-zone Kalman filter was developed to train a state space RNN to avoid divergence in training [14]. This paper further extends the modified EKF work in [14] to train general NN architectures without the prior knowledge of the upper bound of the first-order output approximation residue. Note that (5) and (6) correspond to (9) in [14]. The proposed method to adapt R using Lyapunov's indirect method is illustrated as follows.

The n-dimensional network output $\vec{y}(k)$ is first expanded at the optimal weight vector $\vec{w}^*$ based on (7)

$$\begin{aligned}\vec{y}(k) &= h\left(\vec{w}^*\right) + \frac{\partial h}{\partial w}\left(\vec{w}(k) - \vec{w}^*\right) + \vec{\xi}(k)\\ &= \vec{y}^*(k) + H(k)^T\vec{w}_e(k) + \vec{\xi}(k)\end{aligned} \tag{10}$$

where $\vec{w}_e(k) = \vec{w}(k) - \vec{w}^*$, $\vec{w}^*$ is the optimal weight vector, and $\vec{\xi}(k)$ is the first-order approximation residue. Then the output estimation error $\vec{e}(k)$ is further represented as follows based on (10):

$$\vec{e}(k) = \vec{y}(k) - \vec{y}^*(k) = H(k)^T\vec{w}_e(k) + \vec{\xi}(k). \tag{11}$$

Notation 1: $B(k) \equiv R(k) + H(k)^T P(k) H(k)$ is the covariance matrix of network output.

Notation 2: $h(k) \equiv Trace\left(H(k)^T P(k) H(k)\right)$, is the trace of $H(k)^T P(k) H(k)$.

Theorem 1: For multiple-input multiple-output (MIMO) systems, the EKF training algorithm (7)–(9) is convergent if

$$r(k) = \frac{1}{2}\left(\frac{3h(k)}{n} + \frac{\left\|\vec{e}(k)\right\|^2}{64n}\right), \quad \text{when } \left\|\vec{e}(k)\right\| > \sqrt{192h(k)}. \tag{12}$$

Proof: a Lyapunov function is selected based on a previous work [14] as follows:

$$V(k) = \vec{w}_e(k)^T P(k)^{-1}\vec{w}_e(k). \tag{13}$$

Plug (8) and (11) into (7), and the following equation is derived:

$$\begin{aligned}\vec{w}_e(k+1) &= \vec{w}_e(k) - P(k)H(k)B(k)^{-1}e(k)\\ &= \vec{w}_e(k) - P(k)H(k)B(k)^{-1}H(k)^T\vec{w}_e(k)\\ &\quad -P(k)H(k)B(k)^{-1}\vec{\xi}(k)\\ &= \left(I - P(k)H(k)B(k)^{-1}H(k)^T\right)\vec{w}_e(k)\\ &\quad -P(k)H(k)B(k)^{-1}\vec{\xi}(k).\end{aligned} \tag{14}$$

Rewrite (9) as follows:

$$\left(P(k+1) - Q(k)\right) = \left(I - P(k)H(k)B(k)^{-1}H(k)^T\right)P(k). \tag{15}$$

Apply the matrix inversion lemma on $\left(P(k)^{-1} + H(k)R(k)^{-1}H(k)^T\right)^{-1}$, according to the matrix inversion lemma

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B\left(DA^{-1}B + C^{-1}\right)^{-1}DA^{-1} \tag{16}$$

which leads to

$$\begin{aligned}&\left(P(k)^{-1} + H(k)R(k)^{-1}H(k)^T\right)^{-1}\\ &\quad = \left(I - P(k)H(k)B(k)^{-1}H(k)^T\right)P(k).\end{aligned} \tag{17}$$

Compare (15) and (17), and we find

$$\left(P(k+1) - Q(k)\right)^{-1} = \left(P(k)^{-1} + H(k)R(k)^{-1}H(k)^T\right) > 0. \tag{18}$$

From (14), (17), and (18), it can be derived that

$$\begin{aligned}&\left(P(k+1) - Q(k)\right)^{-1}\vec{w}_e(k+1)\\ &= P(k)^{-1}\vec{w}_e(k) - \left(P(k+1)\right.\\ &\quad \left. -Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k).\end{aligned} \tag{19}$$

Based on the properties of positive definite matrix, we find

$$\begin{aligned}&\vec{w}_e(k+1)^T P(k+1)^{-1}\vec{w}_e(k+1)\\ &< \vec{w}_e(k+1)^T\left(P(k+1) - Q(k)\right)^{-1}\vec{w}_e(k+1).\end{aligned} \tag{20}$$

Consider (19) and (20), which leads to

$$\begin{aligned}\Delta V(k) &= \vec{w}_e(k+1)^T P(k+1)^{-1}\vec{w}_e(k+1)\\ &\quad -\vec{w}_e(k)^T P(k)^{-1}\vec{w}_e(k)\\ &< \vec{w}_e(k+1)^T\left(P(k+1) - Q(k)\right)^{-1}\vec{w}_e(k+1)\\ &\quad -\vec{w}_e(k)^T P(k)^{-1}\vec{w}_e(k)\\ &= \left[\vec{w}_e(k+1) - \vec{w}_e(k)\right]^T P(k)^{-1}\vec{w}_e(k)\\ &\quad -\vec{w}_e(k+1)^T\left(P(k+1) - Q(k)\right)^{-1}\\ &\quad \times P(k)H(k)B(k)^{-1}\vec{\xi}(k).\end{aligned} \tag{21}$$

Plug (14) into (21) and use the fact that $P(k)$ and $B(k)$ are symmetric matrices, then we get

$$\begin{aligned}\Delta V(k) < & -\vec{w}_e(k)^T H(k)B(k)^{-1}H(k)^T\vec{w}_e(k)\\ & -\vec{\xi}(k)^T B(k)^{-1}H(k)^T\vec{w}_e(k)\\ & -\vec{w}_e(k+1)^T\left(P(k+1)-Q(k)\right)^{-1}\\ & \times P(k)H(k)B(k)^{-1}\vec{\xi}(k). \end{aligned} \tag{22}$$

The third term on the right side of (22) is a scalar term, which is simplified by considering (19)

$$\begin{aligned}&\vec{w}_e(k+1)^T\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\\ &=\vec{\xi}(k)^T B(k)^{-1}H(k)^T P(k)\left(P(k+1)\right.\\ &\quad\left.-Q(k)\right)^{-1}\vec{w}_e(k+1)\\ &=\vec{\xi}(k)^T B(k)^{-1}H(k)^T P(k)\left[P(k)^{-1}\vec{w}_e(k)\right.\\ &\quad\left.-\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\right]\\ &=\vec{\xi}(k)^T B(k)^{-1}H(k)^T\vec{w}_e(k)\\ &\quad-\vec{\xi}(k)^T B(k)^{-1}H(k)^T P(k)\left(P(k+1)-Q(k)\right)^{-1}\\ &\quad\times P(k)H(k)B(k)^{-1}\vec{\xi}(k). \end{aligned} \tag{23}$$

Plug (23) into (22) and consider the following relationships which are mainly derived based on properties of positive definite matrix.

1) $R(k)+H(k)^T P(k)H(k)=B(k)$.
2) $R(k)^{-1}>B(k)^{-1}$.
3) $H(k)^T P(k)H(k)B(k)^{-1}<I$.

Then (22) becomes (24) at the bottom of the page.

Based on the matrix property, we know

$$\begin{aligned}&\lambda_m\left(B(k)^{-1}\right)\left\|\vec{e}(k)\right\|^2\le\vec{e}(k)^T B(k)^{-1}\vec{e}(k)\\ &\quad\le\lambda_M\left(B(k)^{-1}\right)\left\|\vec{e}(k)\right\|^2 \end{aligned} \tag{25}$$

$$\begin{aligned}&\lambda_m\left(R(k)^{-1}\right)\left\|\vec{\xi}(k)\right\|^2\le\vec{\xi}(k)^T R(k)^{-1}\vec{\xi}(k)\\ &\quad\le\lambda_M\left(R(k)^{-1}\right)\left\|\vec{\xi}(k)\right\|^2 \end{aligned} \tag{26}$$

where $\lambda_m\left(B(k)^{-1}\right)$ and $\lambda_M\left(B(k)^{-1}\right)$ are the minimum and maximum eigenvalues of matrix $B(k)^{-1}$, respectively, and $\left\|\vec{e}(k)\right\|$ is the Euclidean norm of $\vec{e}(k)$ as $\sqrt{\vec{e}(k)^T\vec{e}(k)}$.

Plug (25) and (26) into (24), we can see

$$\Delta V(k)<-\lambda_m\left(B(k)^{-1}\right)\left\|\vec{e}(k)\right\|^2+3\lambda_M\left(R(k)^{-1}\right)\left\|\vec{\xi}(k)\right\|^2. \tag{27}$$

Since there is no prior information about the measurement noise, this paper simplifies the measurement noises for each output as uncorrelated but with the same variance r as follows (Remark 3):

$$R(k)=r(k)I \tag{28}$$

where r is a positive real number to be determined, I is the $n\times n$ identity matrix, and n is the dimension of output vector. Then

$$\lambda_M\left(R(k)^{-1}\right)=\frac{1}{r(k)}. \tag{29}$$

The minimum eigenvalue of $B(k)^{-1}$ is the inverse of maximum eigenvalue of $B(k)$

$$\lambda_m\left(B(k)^{-1}\right)=\frac{1}{\lambda_M\left(B(k)\right)}. \tag{30}$$

The maximum eigenvalue of $B(k)$ should follow the relationship

$$\frac{1}{n}\sum_i\lambda_i\left(B(k)\right)\le\lambda_M\left(B(k)\right)\le\sum_i\lambda_i\left(B(k)\right) \tag{31}$$

where $\lambda_i\left(B(k)\right)$ is the ith eigenvalue of $B(k)$. The summation of eigenvalues of a matrix equals the trace of the matrix, and it is known that the covariance matrix $B(k)$ is positive definite, which leads to

$$\begin{aligned}\sum_i\lambda_i\left(B(k)\right) &= Trace\left(B(k)\right)\\ &= Trace\left(H(k)^T P(k)H(k)\right)\\ &\quad+nr(k)\equiv h(k)+nr(k)\end{aligned} \tag{32}$$

where $Trace\left(H(k)^T P(k)H(k)\right)$ is the trace of $H(k)^T P(k)H(k)$ and is denoted by $h(k)$ (Notation 2).

$$\begin{aligned}\Delta V(k) < & -\vec{w}_e(k)^T H(k)B(k)^{-1}H(k)^T\vec{w}_e(k)-\vec{\xi}(k)^T B(k)^{-1}H(k)^T\vec{w}_e(k)\\ & -\vec{\xi}(k)^T B(k)^{-1}H(k)^T\vec{w}_e(k)\\ & +\vec{\xi}(k)^T B(k)^{-1}H(k)^T P(k)\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\\ =& -\left[\vec{w}_e(k)^T H(k)B(k)^{-1}H(k)^T\vec{w}_e(k)+2\vec{\xi}(k)^T B(k)^{-1}H(k)^T\vec{w}_e(k)\right.\\ & \left.+\vec{\xi}(k)^T B(k)^{-1}\vec{\xi}(k)\right]+\vec{\xi}(k)^T\left[B(k)^{-1}\right.\\ & \left.+B(k)^{-1}H(k)^T P(k)\left(P(k)^{-1}+H(k)R(k)^{-1}H(k)^T\right)P(k)H(k)B(k)^{-1}\right]\vec{\xi}(k)\\ =& -\left[H(k)^T\vec{w}_e(k)+\vec{\xi}(k)\right]^T B(k)^{-1}\left[H(k)^T\vec{w}_e(k)+\vec{\xi}(k)\right]\\ & +\vec{\xi}(k)^T[B(k)^{-1}\left(I+H(k)^T P(k)H(k)B(k)^{-1}\right)\\ & +B(k)^{-1}H(k)^T P(k)H(k)R(k)^{-1}H(k)^T P(k)H(k)B(k)^{-1}]\vec{\xi}(k)\\ \le & -e(k)^T B(k)^{-1}e(k)+\vec{\xi}(k)^T\left[B(k)^{-1}(I+I)+R(k)^{-1}\right]\vec{\xi}(k)\\ \le & -e(k)^T B(k)^{-1}e(k)+3\vec{\xi}(k)^T R(k)^{-1}\vec{\xi}(k). \end{aligned} \tag{24}$$

Plug (32) into the left side of (31), and we get

$$\frac{1}{n}\,h(k)+r(k)\le\lambda_M\left(B(k)\right). \tag{33}$$

Plug (33) into (30), and we get

$$\lambda_m\left(B(k)^{-1}\right)\le\frac{1}{\frac{1}{n}\,h(k)+r(k)}=\frac{n}{h(k)+nr(k)}. \tag{34}$$

Hence, the inequality (27) will be satisfied if

$$\Delta V(k)<-\frac{1}{\frac{1}{n}\,h(k)+r(k)}\left\|\vec{e}(k)\right\|^2+3\lambda_M\left(R(k)^{-1}\right)\left\|\vec{\xi}(k)\right\|^2. \tag{35}$$

Plug (29) into (35), and we get

$$\Delta V(k)<-\frac{1}{\frac{1}{n}\,h(k)+r(k)}\left\|\vec{e}(k)\right\|^2+\frac{3}{r(k)}\left\|\vec{\xi}(k)\right\|^2. \tag{36}$$

The training process is convergent if $\Delta V(k)<0$, which leads to the following sufficient condition based on (35):

$$r(k)>\frac{\frac{3h(k)}{n}\cdot\left\|\vec{\xi}(k)\right\|^2}{\left\|\vec{e}(k)\right\|^2-3\left\|\vec{\xi}(k)\right\|^2}. \tag{37}$$

Suppose $\left\|\vec{\xi}(k)\right\|$ is bounded by $\bar{\xi}(k)$ $\left(\bar{\xi}(k)\ge\left\|\vec{\xi}(k)\right\|\right)$, consider the following two cases:

Case 1: The output estimation error $\left\|\vec{e}(k)\right\|^2$ is greater than $4\bar{\xi}(k)^2$

$$\left\|\vec{e}(k)\right\|^2>4\bar{\xi}(k)^2. \tag{38}$$

Then plug (38) in (37)

$$r(k)>\frac{\frac{3h(k)}{n}\cdot\left\|\vec{\xi}(k)\right\|^2}{4\left\|\vec{\xi}(k)\right\|^2-3\left\|\vec{\xi}(k)\right\|^2}=\frac{3h(k)}{n}. \tag{39}$$

It can be seen that if inequality (39) holds true, then (37) holds true, and then $\Delta V(k)<0$ as guided in (36), and the training process is convergent. $\vec{\xi}(k)$, the upper bound of the norm of $\vec{\xi}(k)$ in (38), should be known as a prior condition. As $\vec{\xi}(k)$ is the residual of liner model of $\vec{e}(k)$ as shown in (11), it follows the normal distribution $\vec{\xi}(k)\sim N([\vec{0}]_{n\times1},r(k)I_{n\times n})$ based on the EKF algorithm. Each element of $\vec{\xi}(k)$, i.e., $\xi_i(k)$, follows the normal distribution as $\xi_i(k)\sim N(0,r(k))$. Then the upper bound of $\xi_i(k)$ can be approximated as $\eta\sqrt{r(k)}$. If η is selected as 4, then at least 99.99% $\xi_i(k)$ values are bounded by $4\sqrt{r(k)}$. As an approximation, $\bar{\xi}(k)^2$ is taken as $(4\sqrt{r(k)})^2n$, which is $16r(k)n$.

Thus, (38) becomes

$$\left\|\vec{e}(k)\right\|^2>4\bar{\xi}(k)^2=64\,r(k)n. \tag{40}$$

Combine inequalities (39) and (40), and we get

$$\frac{3h(k)}{n}<r(k)<\frac{\left\|\vec{e}(k)\right\|^2}{64n}. \tag{41}$$

When (41) holds, both (38) and (40) hold, which leads to $\Delta V(k)<0$, and the training should be convergent as guaranteed by the Lyapunov method.

To implement (41), the training error information at each step, i.e., $\vec{e}(k)$, should be considered first. When $(3h(k)/n)<\left(\left\|\vec{e}(k)\right\|^2/64n\right)$, or $\left\|\vec{e}(k)\right\|>\sqrt{192h(k)}$, (41) is satisfied if $r(k)$ is set as the average of $(3h(k)/n)$ and $\left(\left\|\vec{e}(k)\right\|^2/64n\right)$ as follows:

$$r(k)=\frac{1}{2}\left(\frac{3h(k)}{n}+\frac{\left\|\vec{e}(k)\right\|^2}{64n}\right)\text{ when }\left\|\vec{e}(k)\right\|>\sqrt{192h(k)}. \tag{42}$$

Under this circumstance, $\Delta V(k)<0$, which means a convergent training process.

Case 2: The output estimation error $\left\|\vec{e}(k)\right\|^2$ is less than $4\bar{\xi}(k)^2$.

The training error is bounded and no adaption needs to be implemented at training step k. This concludes the proof. ■

Corollary 1: For multiple-input and single-output (MISO) systems, Theorem 1 can be simplified as follows:

$$r(k)=\frac{1}{2}\left(3H(k)^TP(k)H(k)+\frac{e(k)^2}{64}\right)\text{ when }|e(k)|>\sqrt{192\;H(k)^TP(k)H(k)}. \tag{43}$$

The above condition [(42) or (43)] is the sufficient condition for a convergent training process instead of as a necessary condition.

It should be noted that (13)–(22) here are similar to (24)–(30) in [14]. The proposed study has the following two advantages when comparing with [14].

The proposed approach significantly expands the application scope of [14]. The EKF model (7)–(9) here applies to any network types, and the proof process is valid for any multiple-output networks, while in [14] the EKF model is only for specific state-space recurrent networks, and the proof process is only valid for single-output networks (scalar case).

The governing conditions proposed in this paper are more practical. Some governing conditions have been proposed to guarantee training convergence in [14], however, one of the parameters, i.e., the upper bound of uncertainty $\bar{\xi}$, is difficult to determine, which makes it less practical. In this paper, the variables ($H(k)$,$P(k)$, and $\vec{e}(k)$) in condition (12) can be retrieved directly from the training process.

B. Q Adaption Law

While the training process can be guaranteed convergent using the R adaption law, the training process can be further accelerated by adapting the EKF process noise Q using the maximum likelihood method, and the Q adaption law is introduced as follows:

Remark 4: $Q(k)$ is assumed constant over the period from step $k-N+1$ to k during training, where N is the window size selected to maximize the training convergence speed by trial and error.

Notation 3: $\Delta\vec{w}(j)=\vec{w}(j)-\vec{w}(j-1)$ is the difference of estimated weights at j and $j-1$ time steps.

Theorem 2: The maximum likelihood estimator of $Q(k)$ is

$$\hat{Q}(k) = \frac{1}{N}\left\{\sum_{j=k-N+1}^{k}\left\{\Delta\vec{w}(j)\Delta\vec{w}(j)^T\right\} + P(k) - P(k-N)\right\}. \tag{44}$$

Proof: Take the likelihood function as $L = \ln\left(f_{\vec{\psi}(k),Y_N(k)|Y(k-N),\vec{q}}\right)$ [30], where $\vec{\psi}(k)$ is the random weight vector at training time step k, $Y(k)$ is the measurement history $\left\{\vec{y}^*(1), \vec{y}^*(2), \ldots, \vec{y}^*(k)\right\}$ composed of all the measurement vectors up to time step k, $\vec{q}$ is the vector composed of diagonal elements of matrix Q, and $Y_N(k) = \left\{\vec{y}^*(k-N+1), \vec{y}^*(k-N+2), \ldots, \vec{y}^*(k-1), \vec{y}^*(k)\right\}$ is the measurement history composed of N most recent measurement vectors, then the conditional density function can be written as

$$\begin{aligned} f_{\vec{\psi}(k),Y_N(k)|Y(k-N),\vec{q}} &= f_{\vec{\psi}(k)|Y_N(k),Y(k-N),\vec{q}}\, f_{Y_N(k)|Y(k-N),\vec{q}} \\ &= f_{\vec{\psi}(k)|Y_N(k),Y(k-N),\vec{q}}\, \frac{f_{Y_N(k),Y(k-N)|\vec{q}}}{f_{Y(k-N)|\vec{q}}} \\ &= f_{\vec{\psi}(k)|Y(k),\vec{q}}\, \frac{f_{\vec{y}(k),Y(k-1)|\vec{q}}}{f_{Y(k-N)|\vec{q}}} \\ &= f_{\vec{\psi}(k)|Y(k),\vec{q}}\, \frac{f_{\vec{y}(k)|Y(k-1),\vec{q}}\, f_{Y(k-1)|\vec{q}}}{f_{Y(k-N)|\vec{q}}} \\ &= f_{\vec{\psi}(k)|Y(k),\vec{q}} \prod_{j=i-n_w+1}^{i} f_{\vec{y}(j)|Y(j-1),\vec{q}}. \end{aligned} \tag{45}$$

Remark 5: Each of the density functions in (45) is assumed as a Gaussian density function as in [30] as follows:

$$\begin{aligned} &f_{\vec{\psi}(k)|Y(k),\vec{q}}(\vec{\varphi}(k)|\delta(k),\vec{\rho}) \\ &= \frac{1}{(2\pi)^{m/2}\,|P(k)|^{1/2}} \\ &\quad \times \exp\left\{-\frac{1}{2}\left[\vec{\varphi}(k) - \vec{w}(k)\right]^T P(k)^{-1}\left[\vec{\varphi}(k) - \vec{w}(k)\right]\right\} \end{aligned} \tag{46}$$

$$\begin{aligned} &f_{\vec{y}(j)|Y(j-1),\vec{q}}(\vec{\gamma}(j)|\delta(j-1),\vec{\rho}) \\ &= \frac{1}{(2\pi)^{n/2}\,|B(j)|^{1/2}} \\ &\quad \times \exp\left\{\begin{array}{l} -\frac{1}{2}\left[\vec{\gamma}(j) - H(j)\vec{w}(j-1)\right]^T \\ \times B(j)^{-1}\left[\vec{\gamma}(j) - H(j)\vec{w}(j-1)\right] \end{array}\right\}. \end{aligned} \tag{47}$$

Plug (46) and (47) into the likelihood function. Then take derivative of the likelihood function with respect to $\vec{q}$ and make it zero, which will provide the maximum likelihood equation as follows:

$$\begin{aligned} &Trace\left\{P(k)^{-1}\frac{\partial P(k)}{\partial q_k}\right\} \\ &\quad -2\sum_{j=i-N+1}^{i}\frac{\partial\vec{w}(j-1)}{\partial q_k}H(j)^T B(j)^{-1}\vec{r}(j) \\ &\quad +\sum_{j=i-N+1}^{i} tr\left\{\left[A(j)^{-1} - A(j)^{-1}\vec{r}(j)\right.\right. \\ &\quad \left.\left.\times\,\vec{r}(j)^T\,B(j)^{-1}\right]\right\} \end{aligned}\Bigg|_{\vec{q}=\vec{q}^*(k)} = 0 \tag{48}$$

where q_k is the kth component of $\vec{q}$, and $\vec{r}_j = \vec{y}_j - H(t_j)\vec{w}(j-1)$.

To enhance online applicability, some less significant terms in (48) are neglected [30] and thus follows an approximated maximum likelihood equation:

$$\sum_{j=k-N+1}^{k}\left[P(j-1) + Q(j-1) - P(j) - \Delta\vec{w}(j)\Delta\vec{w}(j)^T\right] = 0. \tag{49}$$

The $Q(k)$ matrix can be estimated by solving (49)

$$\hat{Q}(k) = \frac{1}{N}\left\{\sum_{j=k-N+1}^{k}\left\{\Delta\vec{w}(j)\Delta\vec{w}(j)^T\right\} + P(k) - P(k-N)\right\}. \tag{50}$$

This concludes the proof. ■

The developed Q estimator actually is an approximate maximum likelihood estimator because of the simplification from (48) and (49). The R and Q adaption-based training algorithm is summarized in Fig. 3. Training parameters such as the window size N, the noise parameters $Q(0)$ and $R(0)$, and the error covariance $P(0)$ should be specified first. The adapted $R(k)$ and $Q(k)$ at each training step are fed into the EKF training algorithm to update the network weights. The training process iterates until the stop criteria are met, and the trained weights are obtained.

It should be noted that the R adaption law is a sufficient condition, so the convergence trend still holds with the Q adaption law. It can also be seen from the EKF model (7)–(9): $Q(k)$ is directly related to $P(k+1)$ which is used to update the Kalman gain $K(k+1)$ and further update the weight $\vec{w}(k+1)$ to $\vec{w}(k+2)$. During the update cycle, $R(k+1)$ is adapted by the proposed R adaption law to prevent divergence. As so, the impact of $Q(k)$ is always absorbed by $R(k+1)$, and the R adaption law actually takes place after the Q adaption law. Hence the Q adaption law will not blow the system.

IV. Methodology Validation in Modeling a Nonlinear Dynamical System

A. Training Setup

To validate the proposed adaption laws, a nonlinear dynamical benchmark system [16], which represents a single-input single-output (SISO) nonlinear plant, has been modeled using RNN. Such a benchmark system (51), as shown in Fig. 4, has been selected due to its generality as well as analytical

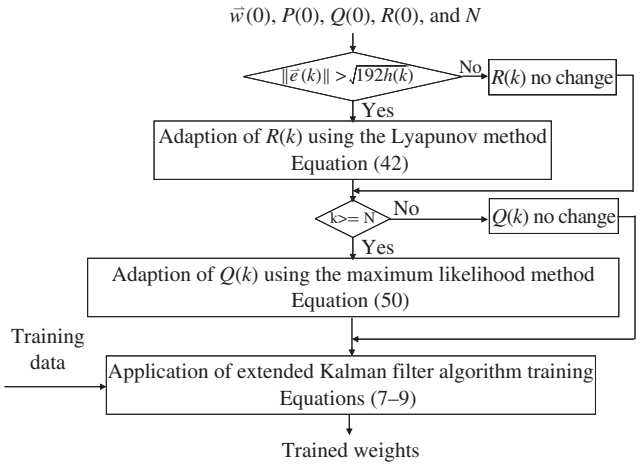

Fig. 3. Proposed adaptive RNN training algorithm.

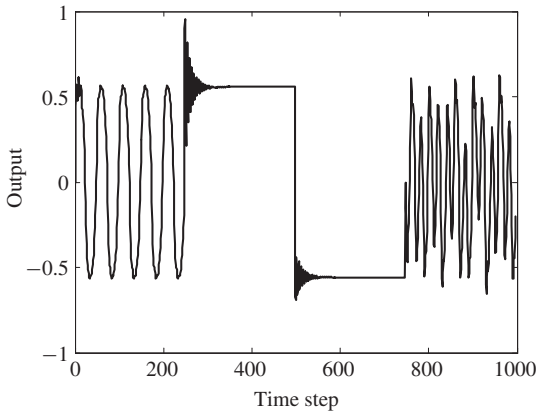

Fig. 4. Nonlinear dynamical benchmark system.

tractability

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{51}$$

where $f(x_1, x_2, x_3, x_4, x_5) = (x_1x_2x_3x_5(x_3-1)+x_4/1+x_2^2+x_3^2)$ and

$$u(k) = \begin{cases} \sin(\frac{\pi k}{25}), & 0 \le k < 250 \\ 1.0, & 250 \le k < 500 \\ -1.0, & 500 \le k < 750 \\ 0.3\sin(\frac{\pi k}{25}) + 0.1\sin(\frac{\pi k}{32}) & \\ +0.6\sin(\frac{\pi k}{10}), & 750 \le k < 1000. \end{cases}$$

In modeling this benchmark system using RNN, six network inputs are selected as follows: three outputs at the previous steps, i.e., $y_p(k)$, $y_p(k-1)$, and $y_p(k-2)y_p(k-2)$; two control actions, i.e., $u(k)$ and $u(k-1)u(k-1)$; and a constant bias 1. The network output is $y_p(k+1)$. The training pairs or patterns are generated using (51); and the zero initial condition is applied to generate data $y_p(k)$, where $k = 1, 2, \ldots, 1000$. The network weights are first randomly initialized in the region of $[-1, 1]$. Without any specific note, in this paper the error covariance matrix P, the process noise covariance matrix Q, and the measurement noise covariance matrix R are all diagonal matrices, and all diagonal elements of P, Q, and R are initialized as 100, 0.01, and 100, respectively as in [32].

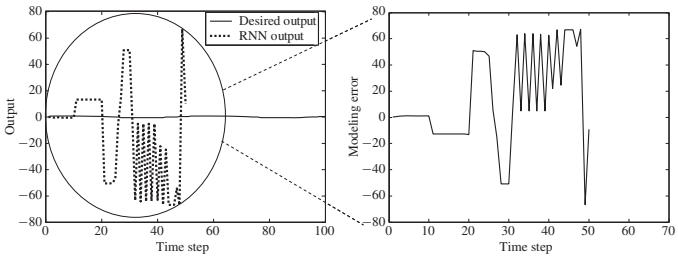

Fig. 5. Training result and modeling error without the R adaption law applied.

The number of hidden neurons is determined by trial and error to achieve the best modeling performance for aforementioned P, Q, and R, and the final network topology is 6-9-1 with nine hidden neurons.

Either of the following two stop criteria is adopted here. 1) The number of training epochs (also called the training step, which is a complete training process covering all available 1000 training patterns) should be not greater than 100. 2) The training error is less than a predetermined case-dependent value (here is 3%) and the difference between the current training error and that of 20 epochs before is less than another predetermined case-dependent value (here is 0.03%). Training error accounts for the overall modeling error of all available training patterns used in training, and the training error for a training epoch j is represented using a normalized sum of square error (SSE) as follows:

$$e(j) = \sqrt{\frac{\left(\vec{y}_o(j) - \vec{y}^*\right)^T \left(\vec{y}_o(j) - \vec{y}^*\right)}{\vec{y}^{*T}\vec{y}^*}} \times 100\% \tag{52}$$

where $\vec{y}^*$ is the target data at an epoch and $\vec{y}_o(j)$ is the output of RNN at epoch j.

B. Implementation of R Adaption Law

The R adaption law is implemented first to appreciate its importance during training. As mentioned before, RNN training divergence may occur due to the poor selection of training noise parameters. Fig. 5 shows the result of a modeling scenario as r is first initialized as 5 and it then decreases linearly to 0.5 at -4.5×10^{-5} per training pattern during first 100 000 training patterns. The corresponding modeling error is also shown in Fig. 5, which diverges right after 50 patterns even before finishing a training epoch. At that time, r has been linearly reduced from 5 to 4.9978. The modeling error for a certain training pattern is defined as the difference between its desired output and the RNN output.

For this divergent case, RNN is further trained by applying the proposed R adaption law. As shown in (43), the R adaption law takes effect when the modeling error is beyond a certain boundary $\left(|e(k)| > \sqrt{192H(k)^T P(k) H(k)}\right)$. As in the aforementioned divergent case, the r is first initialized as 5 and gradually reduced whereas the R adaption law is concurrently implemented. As shown in Fig. 6, the training process becomes convergent using the R adaption law with a training error of 3.6%. It is also found that the evolution of r under the R

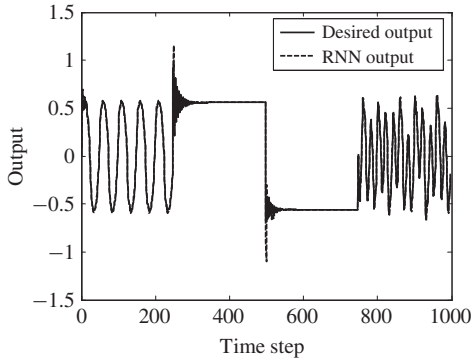

Fig. 6. RNN modeling performance with the R adaption law applied.

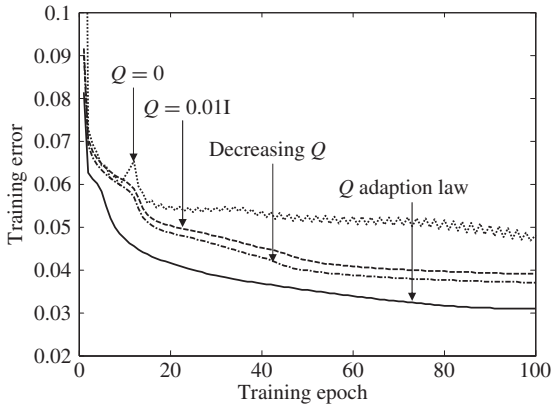

Fig. 7. Comparison of RNN training with different Q settings.

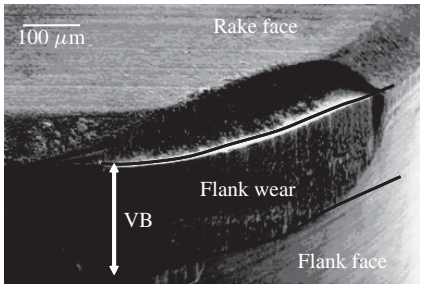

Fig. 8. Typical tool wear picture in CBN hard turning.

adaption law is not linearly decreasing during the training process. When the precondition is met, the R adaption law takes effect and the r values are adapted to make the training process convergent.

C. Implementation of Q Adaption Law

While the training process is convergent, the Q adaption law (50) is further implemented to speed up the training convergence process. Four simulation scenarios are studied to appreciate the importance of the Q adaption law. 1) The Q matrix is set as a null matrix, which means a zero covariance matrix of process noise. 2) The RNN is trained using constant Q ($Q = 0.01I$). 3) The RNN is trained as follows: each diagonal element of Q is initialized as 0.01 and this value decreases linearly during 100 000 training patterns until it reaches a limit of 0.000 001. 4) The proposed Q adaption law is implemented during the training process. The window size N in (50) is set as 20 to have the best training performance. In all the cases, r is initialized as 100 and it is reduced linearly until r reaches a limit of 2 as in a previous study [32]. It should be pointed out that the diagonal elements of Q under the third scenario are always the same, whereas they might be different under the fourth scenario.

For the above four scenarios, the training process is found always stable. Fig. 7 illustrates the effectiveness of introducing the Q adaption law during training. As seen from Fig. 7, the Q adaption law has helped to achieve the best convergence performance with a minimum final training error (3.2%) and fastest training speed, followed by the decreasing Q setting (Scenario 3 with a 3.7% error), the constant Q setting (Scenario 2 with a 3.9% error), and the zero Q setting (Scenario 1 with a 4.8% error).

V. Case Study in Cutting Tool Wear Modeling

A. Tool Wear Modeling

The proposed adaption laws are further applied in modeling of tool wear in hard turning. As shown in [16, Fig. 8], tool flank wear length or wearland (VB) is generally regarded as the tool life criterion or an important index to evaluate the tool performance in hard turning. Hard turning with cubic boron nitride (CBN) tools has been proved to be more effective and efficient in turning hardened steels than traditional grinding operations. However, rapid tool wear is always a problem, which is a hurdle in the wide implementation of hard turning in industry. Accurate modeling of CBN tool wear progression helps to optimize cutting conditions and tool geometry to reduce tool wear.

The wear mechanisms in hard turning are complicated, which makes the data-driven modeling approach, such as NN, a good candidate for tool wear modeling. Among the NN family, RNN is favored here because of its better modeling capability for nonlinear dynamical and nonstationary systems.

B. Experimental Setup and Network Structure

Hardened AISI 52100 bearing steel with a hardness 62 HRc was machined on a horizontal Hardinge lathe using a low CBN content tool insert (Kennametal KD050) with a $-20°$ and 0.1-mm wide edge chamfer and 0.8 mm nose radius. The DCLNR-164D (ISO DCLNR-164D) tool holder was used. No cutting fluid was applied. Flank wear length was measured using an optical microscope (Zygo NewView 200). The experiment was stopped when a sudden force jump was observed, signaling a chipping or broken tool condition.

Cutting tests were performed based on a standard central composite design test matrix with an alpha value of 1.414. The center point (0, 0) was determined based on the tool manufacturer's recommendation [37]. A typical depth of cut was suggested as 0.203 mm, which was used in the test matrix. It is noted that, due to the experimental parameter selection, the dynamic machining process does not have much time-varying features so that the FFCNN network can satisfactorily model the tool wear progression process. However, the RNN

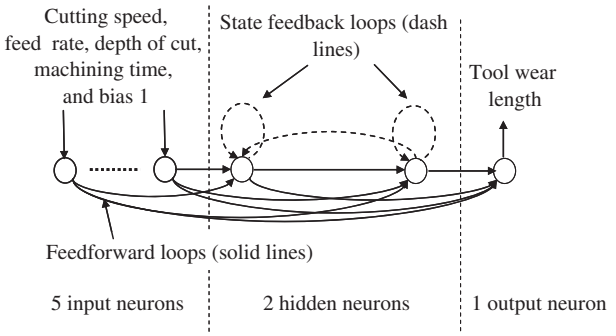

Fig. 9. RNN structure for tool wear modeling.

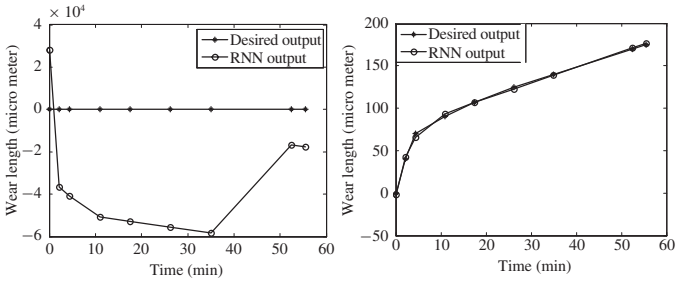

Fig. 10. Comparison of training results with and without the R adaption law applied.

is applied here because it is found to better model the system in terms of accuracy and robustness than FFCNN, and it is applied to test the adaption laws developed in this paper.

The five NN inputs include cutting speed, feed rate, depth of cut, machining time, and bias 1. The single network output is the coded tool flank wear length, and there are two hidden neurons as shown in Fig. 9. The details of the experimental setup, the training data preparation, and the network structure selection process can be referred from a previous study [17], and in the following the effect of R and Q adaption laws in training convergence is presented.

C. Implementation of R and Q Adaption Laws

As discussed, without the implementation of the R adaption law, training may diverge due to the poor selection of training noise parameters. Fig. 10(a) shows the result of a representative divergent training scenario (speed = 1.52 m/s, feed = 0.076 mm/revolution, and depth of cut = 0.102 mm) as r is first initialized as 0.45 and it then decreased linearly to 0.1 at -3.5×10^{-6} per training pattern during 100 000 training patterns. The training process diverges right after 922 patterns have been used.

RNN is also trained using the proposed R adaption law. The r value is initialized as in the aforementioned divergent case, and the R adaption law takes effect when the modeling error is beyond the certain threshold as specified by (43). As shown in Fig. 10(b), it is clear that the training process is stabilized under the R adaption law, and the final training error is 4.1%.

The effect of the Q adaption law is also tested in tool wear modeling under the same representative cutting conditions. As in the benchmark validation section, the four simulation scenarios are studied to appreciate the importance of the Q adaption law. As seen from Fig. 11, the Q adaption law has helped to achieve the best convergence performance with the minimum final training error (3.8%) and the fastest training speed, and the zero Q setting (Scenario 1 with a 4.8% error) results in the worst training performance.

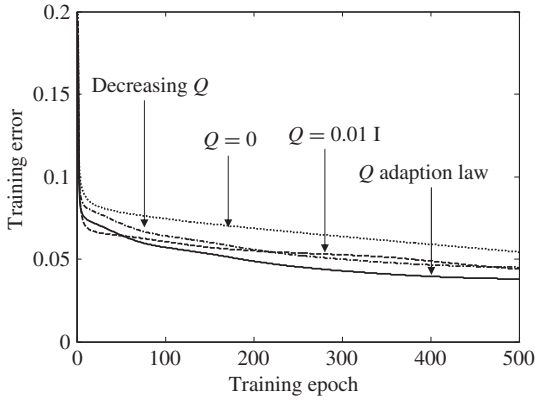

Fig. 11. Comparison of RNN training with different Q settings.

Through this case study, the proposed Q and R adaption laws are further verified in training RNN for tool wear modeling applications, and the proposed adaption laws have demonstrated their effectiveness in stabilizing the training process and speeding up the converge speed.

VI. Discussion and Conclusion

The weight update progress during EKF-based training is affected by the measurement and process noises. Proper selection and control of these two noise parameters helps to stabilize the RNN training process, to improve the training accuracy, and to accelerate the training process. The training convergence problem during EKF-based RNN training has been proposed and studied by adapting two artificial training noises: the covariance of measurement noise (R) and the covariance of process noise (Q) of Kalman filter. The R and Q adaption laws have been developed using the Lyapunov method and the maximum likelihood method, respectively.

The R adaption law provides a sufficient condition to stabilize the training process when the modeling error during training is larger than a specific threshold. Generally, small r values give more confidence to the measurements to make the network outputs match the measurement data fast, resulting in a fast training process. However, using too small r values may make the training process divergent. It is critical to apply the R adaption law to guarantee the training convergence.

The Q adaption law helps to adapt Q during training based on the training performance and accelerate the training convergence. It is further found that, when the modeling error is relatively large, the Q values are enlarged to drive the training process more efficiently; when the modeling error is very small, the Q values are adjusted to be small, which means there is no need to change the weights too much. Note that Q here is a scalar for the above studies because the outputs of both two cases are scalar.

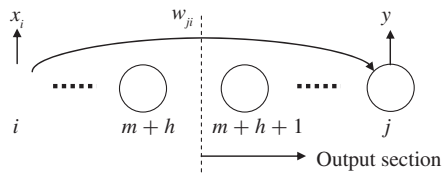

Fig. 12. Representative illustration for Case I.

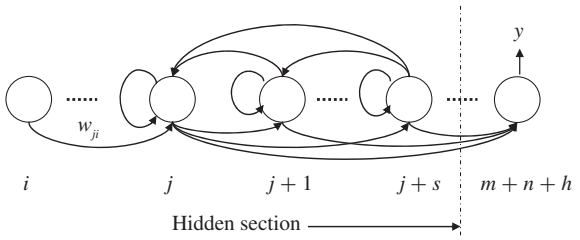

Fig. 13. Representative illustration for Case II.

The effectiveness of the proposed R and Q adaption laws has been tested using a nonlinear dynamical benchmark system [16] and further applied in cutting tool wear modeling. The results show that the R adaption law can effectively avoid the divergence problem and ensure the training convergence, whereas the Q adaption law helps to improve the training convergence speed. While this paper focuses on the EKF-training method, it can be further extended for the unscented Kalman filter-based training algorithm [18], which is attracting more and more attention in RNN training [19].

It should be pointed out that the R adaption law may also affect the training speed. To further optimize the training convergence speed, R needs to be adapted to guarantee convergence as well as to accelerate the training speed. Future work should mathematically understand the coupling effects of the two noise parameters on EKF training and further improve the training convergence performance.

ACKNOWLEDGMENT

The authors would like to acknowledge K. Krishna Kumar and N. Nguyen of the NASA Ames Research Center, CA, for constructive discussions.

APPENDIX

Generally, an element of the orderly derivative matrix H can be written as

$$\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial^+ y}{\partial x_j}\frac{\partial x_j}{\partial net_j}\frac{\partial^+ net_j}{\partial w_{ji}} \tag{A1}$$

where y is the output of an output neuron, w_{ji} is the connection weight in matrix form which represents a weight from neuron i to neuron j, x_j is the output of neuron j, and net_j is the net input of neuron j. Calculations here and in the following are based on the current time step k unless specified.

The derivatives take different forms depending on the specific weight connections, and the derivation process is illustrated in the following.

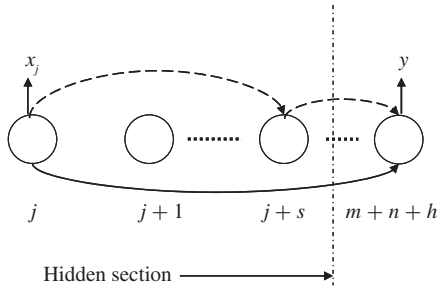

Fig. 14. Illustration of $(\partial^+ y/\partial x_j)$ calculation in case II.

Case I: w_{ji} is with a feedforward connection ($i<j$), neuron j is in the output section, and neuron i is in the input or hidden section.

Case I is the simplest case that only forward connections need to be considered to calculate the orderly derivative. For a weight w_{ji}, only direct impacts act on y through the weight so the ordinary derivative is considered here. Using a representative illustration (Fig. 12), the orderly derivative for Case I can be written as follows:

$$\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial y}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial y}{\partial net_j}x_i. \tag{A2}$$

Case II: w_{ji} is with a feedforward connection ($i < j$) and neuron j is in the hidden section.

Case II deals with the situation that the weight w_{ji} is on a forward connection ($i < j$) and neuron j is in the hidden section. As shown in Fig. 13, both forward loops and feedback loops need to be considered to derive the orderly derivative $(\partial^+ y/\partial w_{ji})$. Fig. 14 shows how to calculate the orderly derivative $(\partial^+ y/\partial x_j)$ in (A1). Here x_j, the output of neuron j, contributes to y in two ways: directly through the weight $w_{n_n j}$ [the weight on the solid line in Fig. 14 $n_n \equiv m+n+h(n_n \equiv m+n+h)$] and indirectly through other forward loops to the right of neuron j (the dashed lines in Fig. 14)

$$\begin{aligned}\frac{\partial^+ y}{\partial x_j} &= \sum_{s=1}^{m+h+n-j}\frac{\partial^+ y}{\partial x_{j+s}}\frac{\partial x_{j+s}}{\partial x_j} \\ &= \sum_{s=1}^{m+h+n-j}\frac{\partial^+ y}{\partial y x_{j+s}}\frac{\partial x_{j+s}}{\partial net_{j+s}}\frac{\partial net_{j+s}}{\partial x_j} \\ &= \sum_{s=1}^{m+h+n-j}\frac{\partial^+ y}{\partial x_{j+s}}\frac{\partial x_{j+s}}{\partial net_{j+s}}w_{(j+s)j}.\end{aligned} \tag{A3}$$

Fig. 15 shows the effect of w_{ji} on net_j through two signal paths, a forward path at current time step k and a feedback path through $x_{j+s}(k-1)$ to $net_j(k)$.

Then the orderly derivative $(\partial^+ net_j/\partial w_{ji})$ can be computed as follows. Consider the effect of the forward path

$$\frac{\partial^+ net_j(k)}{\partial w_{ji}} = x_i\,(k)\,. \tag{A4}$$

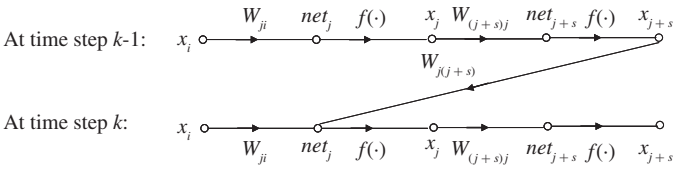

Fig. 15. Signal flow graph to compute $(\partial^+ net_j / \partial w_{ji})$.

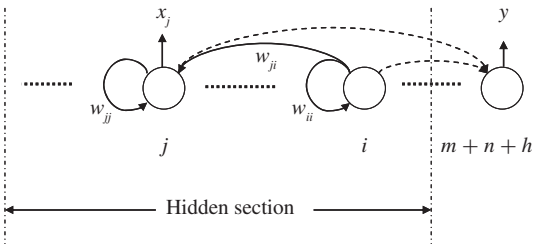

Fig. 16. Representative illustration for Case III.

Consider the effect of the feedback paths

$$\begin{aligned}
&\frac{\partial^+ net_j(k)}{\partial w_{ji}} \\
&= \frac{\partial^+ net_j(k)}{\partial x_j(k-1)} \frac{\partial x_j(k-1)}{\partial w_{ji}} \\
&= \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial x_{j+s}(k-1)} \frac{\partial^+ x_{j+s}(k-1)}{\partial x_j(k-1)} \right) \frac{\partial x_j(k-1)}{\partial w_{ji}} \\
&= \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial x_{j+s}(k-1)} \frac{\partial^+ x_{j+s}(k-1)}{\partial x_j(k-1)} \right) \\
&\quad \times \frac{\partial x_j(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \qquad \text{(A5)}
\end{aligned}$$

$$\frac{\partial^+ x_{j+s}(k-1)}{\partial x_j(k-1)} = \frac{\partial x_{j+s}(k-1)}{\partial net_{j+s}(k-1)} \sum_{p=1}^{s} \frac{\partial net_{j+s}(k-1)}{x_{j+s-p}(k-1)} \times \frac{\partial x_{j+s-p}(k-1)}{\partial x_j(k-1)}. \qquad \text{(A6)}$$

Combine (A4) and (A5), the orderly derivative is

$$\begin{aligned}
\frac{\partial^+ net_j(k)}{\partial w_{ji}} &= x_i(k) \\
&+ \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial x_{j+s}(k-1)} \frac{\partial^+ x_{j+s}(k-1)}{\partial x_j(k-1)} \right) \\
&\times \frac{\partial x_j(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}}. \qquad \text{(A7)}
\end{aligned}$$

Consider (A3) and (A7), and (A1) can be written as

$$\begin{aligned}
&\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial^+ y}{\partial x_j} \frac{\partial x_j}{\partial net_j} \frac{\partial^+ net_j}{\partial w_{ji}} \\
&= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial x_{j+s}} \frac{\partial x_{j+s}}{\partial net_{j+s}} w_{(j+s)j} \frac{\partial x_j}{\partial net_j} \frac{\partial^+ net_j}{\partial w_{ji}} \\
&= \sum_{s=1}^{m+h+n-j} \frac{\partial^+ y}{\partial x_{j+s}} \frac{\partial x_{j+s}}{\partial net_{j+s}} w_{(j+s)j} \frac{\partial x_j}{\partial net_j} \\
&\quad \times \left[x_i(k) + \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial x_{j+s}(k-1)} \frac{\partial x_{j+s}(k-1)}{\partial x_j(k-1)} \right) \right. \\
&\quad \left. \times \frac{\partial x_j(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \right]. \qquad \text{(A8)}
\end{aligned}$$

Case III: Weight w_{ji} is with a feedback connection $(j < i)$.

Case III deals with the situation when the weight w_{ji} is on a feedback connection $(j < i)$. In this case, both neurons$i$ and j are in the hidden section. It should be pointed out that the weight w_{ji} does not have any impact on $x_i(k-1)$ as only one-step recurrency is of interest.

As seen from Fig. 16, if weights are in the feedback loop $(j < i)$, the orderly derivatives are computed as follows:

$$\begin{aligned}
\frac{\partial^+ y}{\partial w_{ji}} &= \frac{\partial^+ y}{\partial x_j(k-1)} \frac{\partial x_j(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \\
&= \frac{\partial^+ y}{\partial x_j(k-1)} \frac{\partial x_j(k-1)}{\partial net_j(k-1)} x_i(k-1). \qquad \text{(A9)}
\end{aligned}$$

REFERENCES

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.

[2] Y. Liguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter," *IEEE Trans. Signal Process.*, vol. 40, no. 4, pp. 959–966, Apr. 1992.

[3] Q. Song, Y. Wu, and Y. C. Soh, "Robust adaptive gradient-descent training algorithm for recurrent neural networks in discrete time domain," *IEEE Trans. Neural Netw.*, vol. 19, no. 11, pp. 1841–1853, Nov. 2008.

[4] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: Unifying the algorithms and accelerating convergence," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 697–709, May 2000.

[5] S.-S. Yang, C.-L. Ho, and S. Siu, "Computing and analyzing the sensitivity of MLP due to the errors of the i.i.d. inputs and weights based on CLT," *IEEE Trans. Neural New.*, vol. 21, no. 12, pp. 1882–1891, Dec. 2010.

[6] Q. Song, X. Yang, Y. C. Soh, and Z. M. Wang, "An information-theoretic fuzzy C-spherical shells clustering algorithm," *Fuzzy Sets Syst.*, vol. 161, no. 13, pp. 1755–1773, Jul. 2010.

[7] H. Tang, K. C. Tan, and Z. Yi, *Neural Networks: Computational Models and Applications*. New York: Springer-Verlag, 2007.

[8] Q. Song, "On the weight convergence of Elman networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 3, pp. 463–480, Mar. 2010.

[9] M. B. Matthews, "Neural network nonlinear adaptive filtering using the extended Kalman filter algorithm," in *Proc. Int. Neural Netw. Conf.*, vol. 1. Paris, France, 1990, pp. 115–119.

[10] C.-S. Leung and L.-W. Chan, "Dual extended Kalman filtering in recurrent neural networks," *Neural Netw.*, vol. 16, no. 2, pp. 223–239, Mar. 2003.

[11] W. Liu, L.-L. Yang, and L. Hanzo, "Recurrent neural network based narrowband channel prediction," in *Proc. IEEE 63rd Veh. Technol. Conf.*, vol. 5. Melbourne, Australia, May 2006, pp. 2173–2177.

[12] W. Yu and J. De J. Rubio, "Recurrent neural networks training with stable bounding ellipsoid algorithm," *IEEE Trans. Neural Netw.*, vol. 20, no. 6, pp. 983–991, Jun. 2009.

[13] J. de J. Rubio, W. Yu, and A. Ferreyra, "Neural network training with optimal bounded ellipsoid algorithm," *Neural Comput. Appl.*, vol. 18, no. 6, pp. 623–631, Sep. 2009.

[14] J. de J. Rubio and W. Yu, "Nonlinear system identification with recurrent neural networks and dead-zone Kalman filter algorithm," *Neurocomputing*, vol. 70, nos. 13–15, pp. 2460–2466, Aug. 2007.

[15] A. Alessandri, M. Cuneo, S. Pagnan, and M. Sanguineti, "On the convergence of EKF-based parameters optimization for neural networks," in *Proc. 42nd IEEE Conf. Decis. Control*, vol. 6. Dec. 2003, pp. 6181–6186.
[16] X. Wang, W. Wang, Y. Huang, N. Nguyen, and K. K. Kumar, "Design of neural network-based estimator for tool wear modeling in hard turning," *J. Intell. Manuf.*, vol. 19, no. 4, pp. 383–396, 2008.
[17] X. Wang and Y. Huang, "Optimized recurrent neural network-based tool wear modeling in hard turning," *Trans. NAMRI/SME*, vol. 37, pp. 213–220, Jun. 2009.
[18] E. A. Wan and R. van der Merwe, "The unscented kalman filter," in *Kalman Filtering and Neural Networks*, S. Haykin, Ed. New York: Wiley, 2001, ch. 7. pp. 1–50.
[19] J. Choi, T. H. Yeap, and M. T. Bouchard, "Online state-space modeling using recurrent multilayer perceptrons with unscented Kalman filter," *Neural Process. Lett.*, vol. 22, no. 1, pp. 69–84, 2005.
[20] R. Fitzgerald, "Divergence of the Kalman filter," *IEEE Trans. Autom. Control*, vol. 16, no. 6, pp. 736–747, Dec. 1971.
[21] F. H. Schlee, C. J. Standish, and N. F. Toda, "Divergence in the Kalman filter," *AIAA J.*, vol. 5, no. 6, pp. 1114–1120, 1967.
[22] D. Simon, *Optimal State Estimation*. New York: Wiley, 2006.
[23] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering: A Practical Approach*. Washington D.C.: AIAA, 2001.
[24] A. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge, U.K.: Cambridge Univ. Press, 1989.
[25] D.-J. Jwo and T.-S. Cho, "A practical note on evaluating Kalman filter performance optimality and degradation," *Appl. Math. Comput.*, vol. 193, no. 2, pp. 482–505, Nov. 2007.
[26] S. Sastry, *Nonlinear Systems: Analysis, Stability, and Control*. New York: Springer-Verlag, 1999.
[27] H. K. Khalil, *Nonlinear Systems*. Upper Saddle River, NJ: Prentice-Hall, 2002.
[28] D. Alspach, "A parallel filtering algorithm for linear systems with unknown time varying noise statistics," *IEEE Trans. Autom. Control*, vol. 19, no. 5, pp. 552–556, Oct. 1974.
[29] R. Mehra, "Approaches to adaptive filtering," *IEEE Trans. Autom. Control*, vol. 17, no. 5, pp. 693–698, Oct. 1972.
[30] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, vol. 2. New York: Academic, 1982.
[31] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[32] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 279–297, Mar. 1994.
[33] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 4. Baltimore, MD, Jun. 1992, pp. 241–246.
[34] J. Henriques and A. Dourado, "Multivariable adaptive control using an observer based on a recurrent neural network," *Int. J. Adapt. Control Signal Process.*, vol. 13, no. 4, pp. 241–259, Jun. 1999.
[35] S. Singhal and L. Wu, "Training feed-forward networks with the extended Kalman algorithm," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, vol. 2. Glasgow, U.K., May 1989, pp. 1187–1190.
[36] L. Zhang and P. B. Luh, "Neural network-based market clearing price prediction and confidence interval estimation with an improved extended Kalman filter method," *IEEE Trans. Power Syst.*, vol. 20, no. 1, pp. 59–66, Feb. 2005.
[37] Y. Huang and S. Y. Liang, "Modeling of CBN tool flank wear progression in finish hard turning," *J. Manuf. Sci. Eng.*, vol. 126, no. 1, pp. 98–106, 2004.

Xiaoyu Wang received the M.S. degree in mechanical engineering from Florida International University, North Miami, and the Ph.D. degree in mechanical engineering from Clemson University, Clemson, SC, in 2010.

He is currently a Post-Doctoral Researcher at the University of Florida, Gainesville. His current research interests include encompass machine learning, statistic process modeling, and nonlinear control.

Yong Huang (M'98) received the B.S. degree in mechatronics engineering from Xidian University, Xi'an, China, in 1993, the M.S. degree in mechanical engineering from Zhejiang University, Hangzhou, China, in 1996, another M.S. degree in electrical and computer engineering from the University of Alabama, Huntsville, in 1999, and the Ph.D. degree in mechanical engineering from the Georgia Institute of Technology, Atlanta, in 2002.

He has been a Professor of mechanical engineering at Clemson University, Clemson, SC, since 2003. His research addresses various challenges in advanced manufacturing. His current research interests include additive manufacturing for biological and energy applications, precision engineering, and manufacturing process monitoring.

Prof. Huang was the recipient of the American Society of Mechanical Engineers Blackall Machine Tool and Gage Award in 2005, the Society of Manufacturing Engineers Outstanding Young Manufacturing Engineer Award in 2006, and the National Science Foundation CAREER Award in 2008.