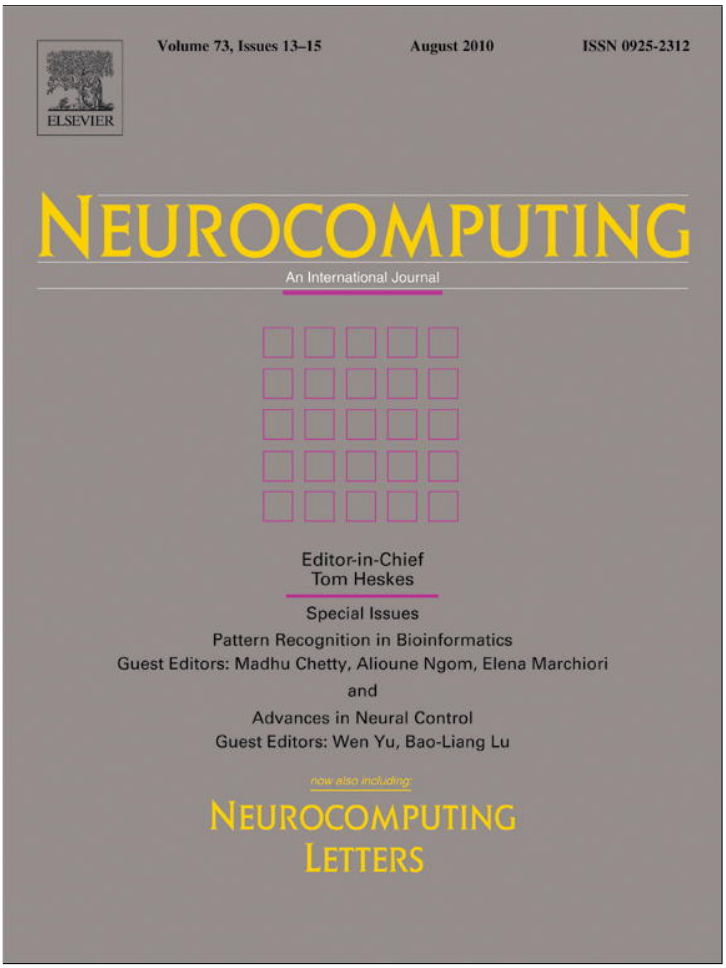
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Neurocomputing 73 (2010) 2624–2631

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Constructive training of recurrent neural networks using hybrid optimization

Niranjan Subrahmanya, Yung C. Shin*

School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907, USA

ARTICLE INFO

Article history:
Received 22 February 2009
Received in revised form
11 January 2010
Accepted 12 May 2010
Communicated by G. Palm
Available online 3 July 2010

Keywords:
Recurrent neural networks
Hybrid optimization
Constructive method
Structure and parameter learning

ABSTRACT

Training of recurrent neural networks (RNNs) is known to be a very difficult task. This work proposes a novel constructive method for simultaneous structure and parameter training of Elman-type RNNs using a combination of particle swarm optimization (PSO) and covariance matrix adaptation based evolutionary strategy (CMA-ES). The proposed method allows the imposition of certain stability conditions, which can be maintained throughout the constructive process. The examples reported show a monotonic decrease in training error throughout the constructive process and also demonstrate the efficiency of the proposed method for structure and parameter training of RNNs.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Recurrent neural networks (RNNs) are artificial neural networks with feedback connections and have the capability to model spatial as well as temporal dependencies between input and output sequences. RNNs with appropriate structure can approximate any dynamical system with arbitrary precision [19] and this makes them suitable for a wide range of applications including time series prediction, dynamic process modeling, process control, dynamic optimization and creation of associative memory banks. However, the presence of feedback connections makes the training of RNNs much more difficult as compared to static neural networks. Hence, the theoretical guarantees about the capabilities of RNNs do not always ensure good results in practice and a good, automated algorithm for determining appropriate architecture, structure and parameters from input/output observations would be an important tool in alleviating this problem.

The architecture of a network defines the important properties of an RNN such as the nature of feedback connections considered, the number of time lags considered and the division of network nodes into input, output and hidden nodes. Based on the type of connections, RNN architectures may be divided into fully connected and partially connected nets [15]. Partially connected nets allow the description of RNNs as a generalization of multilayer feed-forward networks with distinct input, output and hidden nodes, whereas fully connected networks do not distinguish between various nodes or layers. Two simple and popular partially connected architectures, which add feedback into a feed-forward multilayer neural network, are the Elman network [4] and the Jordan network [11]. The Elman network introduces feedback from the hidden layer to a set of context nodes, whose output is then used as an input to the hidden layer in the next iteration. Context nodes are simply used for storage of the hidden node outputs at the previous iteration. Jordan networks use feedback from the output layer to a set of context nodes. While a number of other architectures with feedback delays greater than one [20] have been considered, this work considers only the Elman network architecture as it is shown to be sufficient to approximate the characteristics of any dynamic system that can be represented in the state space form.

For a given architecture, the structure of the RNN defines the number of inputs, the number of hidden layers, the number of nodes in each hidden layer and the number of outputs. Finally, the parameters of an RNN are the weights associated with the connections and the nodes. Generally, for a given application, the architecture of an RNN is either chosen randomly or based on user's experience. A number of algorithms have been developed over the years for parameter learning assuming a fixed structure. These include various gradient based methods [16,18,22,23] and filter-based methods [9,17]. The use of these algorithms involves the user trying a large number of structures for the selected architecture and then learning the parameters for each of the chosen structures using one of the above methods. This is obviously an extremely tedious procedure and involves significant user expertise and involvement. Moreover, many of these algorithms involve the use of gradient based methods, which for some architectures are unable to learn long term temporal

* Corresponding author. Tel.: +1 765 494 9775; fax: +1 765 494 0539.
E-mail address: shin@purdue.edu (Y.C. Shin).

0925-2312/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.05.012

Author's personal copy

dependencies as the errors flowing back in time either decay exponentially or blow up [10].

More recent works on RNN training have focused on simultaneous structure and parameter learning using derivative free direct optimization methods in an attempt to automate the training process to a larger extent. These methods also have a higher probability of finding the global optimum of the error function and can overcome to a certain degree the problem of error decay/blow up in RNNs since they do not depend on the derivatives. Different methods such as evolutionary algorithms [1,5], particle swarm optimization (PSO) and their hybrids [2] have been used to solve the optimization problem. The objective function to be minimized could just be the training error or it could be a combination of the training error and a measure of the complexity of the network for simultaneous structure and parameter learning. It is also possible to make use of multi-objective optimization achieve tradeoffs between network complexity and accuracy [3]. Although the results from these works indicate that direct optimization using derivative free methods is superior to gradient based methods for training RNNs, the computational cost of these algorithms can prove to be prohibitive for large problems. This is because the increase in the dimension of the optimization problem is at least proportional to the square of the number of hidden nodes and this not only increases the computational cost of the direct optimization based methods but could also result in decreasing their robustness to network initialization.

Constructive methods, especially those based on least squares methods, have enjoyed a lot of success in determining the structure and parameters of feed-forward networks in a computationally efficient manner. After some initial attempts to extend this for recurrent neural networks [6,7,14], this approach has not been gaining a lot of attention as of late, with the focus having shifted to the use of direct optimization of the entire network. The main reason for this is that unlike static neural networks, the addition of a node to the hidden layer of an RNN alters the outputs of all the previously added nodes and hence the optimization of the parameters of the additional node still requires the consideration of its effect on the entire network. However, the number of parameters to optimize during the constructive procedure increases only linearly with the number of hidden nodes and hence this approach has considerable promise for finding fast and efficient solutions to the problem of automated RNN training from data. In this paper, a novel constructive procedure is proposed for simultaneous structure and parameter training of RNNs using a combination of particle swarm optimization (PSO) [12] for optimization during the constructive phase with intermittent training of the entire network parameters using covariance matrix adaptation based evolutionary strategies (CMA-ES) [8]. Moreover, a specialized parameterization of the network parameters, which makes use of the singular value decomposition (SVD), is used to ensure certain stability conditions throughout the training procedure.

Section 2 presents the network architecture, structure and parameterization considered in this work. Section 3 presents the hybrid optimization based structure and parameter learning procedure. Results from the application of the proposed algorithm to problems in time series prediction and dynamic system modeling are presented in Section 4, while the conclusions and scope for future work are given in Section 5.

2. RNN architecture and paramterization

The first issue to be addressed is the choice of a suitable architecture for the constructive procedure. As mentioned earlier, this involves the choice of the nature of connections allowed and the type of nodes used. In this section, a suitable architecture is derived based on its sufficiency to represent any dynamic system that can be represented in the state space form. The RNN can be represented in the state space form with all the measured variables (including the measured states of the original system) as outputs and the hidden node activations as the states. Assume that the discrete state space equation representing the system is

$$\begin{aligned} x_{k+1} &= f(x_k,u_k) \\ y_k &= h(x_k) \end{aligned} \tag{1}$$

where $x_k \in R^n$ is the state vector, $y_k \in R^p$ is the output vector, $u_k \in R^m$ is the input vector and $f(x_k,u_k)$ is the non-linear plant model and $h(x_k)$ is the non-linear observation equation. Using the capability of feed forward networks with a single hidden layer with sigmoidal hidden units to represent any Lipschitz continuous function to an arbitrary degree of accuracy, the functions f and h may be replaced by equivalent feed forward networks given in (2) where σ denotes the sigmoid function.

$$\begin{aligned} f(x_k,u_k) &= \mathbf{V}_f \sigma(\mathbf{W}_f x_k + \mathbf{B}_f u_k + \boldsymbol{\theta}_f) \\ h(x_k) &= \mathbf{V}_h \sigma(\mathbf{W}_h x_k + \boldsymbol{\theta}_h) \end{aligned} \tag{2}$$

In order to relate (2) to the structure of an RNN, use the following substitution:

$$\begin{aligned} \eta_k^x &= \mathbf{W}_f x_k + \mathbf{B}_f u_k + \boldsymbol{\theta}_f \\ \eta_k^y &= \mathbf{W}_h x_k + \boldsymbol{\theta}_h \end{aligned} \tag{3}$$

Combining (2) and (3) the expressions given in (4) and (5) may be obtained for the evolution of the newly defined states.

$$\begin{aligned} \eta_{k+1}^x &= \mathbf{W}_f x_{k+1} + \mathbf{B}_f u_{k+1} + \boldsymbol{\theta}_f \\ &= \mathbf{W}_f f(x_k,u_k) + \mathbf{B}_f u_{k+1} + \boldsymbol{\theta}_f \\ &= \mathbf{W}_f \{\mathbf{V}_f \sigma(\mathbf{W}_f x_k + \mathbf{B}_f u_k + \boldsymbol{\theta}_f)\} + \mathbf{B}_f u_{k+1} + \boldsymbol{\theta}_f \\ &= \mathbf{W}_f \mathbf{V}_f \sigma(\eta_k^x) + \mathbf{B}_f u_{k+1} + \boldsymbol{\theta}_f \end{aligned} \tag{4}$$

Similarly,

$$\begin{aligned} \eta_{k+1}^y &= \mathbf{W}_h x_{k+1} + \boldsymbol{\theta}_h \\ &= \mathbf{W}_h f(x_k,u_k) + \mathbf{B}_h u_{k+1} + \boldsymbol{\theta}_h \\ &= \mathbf{W}_h \{\mathbf{V}_h \sigma(\mathbf{W}_h x_k + \mathbf{B}_h u_k + \boldsymbol{\theta}_h)\} + \mathbf{B}_h u_{k+1} + \boldsymbol{\theta}_h \\ &= \mathbf{W}_h \mathbf{V}_h \sigma(\eta_k^x) + \mathbf{B}_h u_{k+1} + \boldsymbol{\theta}_h \end{aligned} \tag{5}$$

Using (2)–(5) the following augmented state space system may be obtained and shown to be equivalent to the state space system (1):

$$\begin{aligned} \eta_{k+1}^x &= \mathbf{W}_f \mathbf{V}_f \sigma(\eta_k^x) + \mathbf{B}_f u_{k+1} + \boldsymbol{\theta}_f \\ \eta_{k+1}^y &= \mathbf{W}_h \mathbf{V}_f \sigma(\eta_k^x) + \mathbf{B}_h u_{k+1} + \boldsymbol{\theta}_h \\ y_{k+1} &= \mathbf{V}_h \sigma(\eta_{k+1}^y) \end{aligned} \tag{6}$$

Now, let $\eta_k = \begin{bmatrix} \eta_k^x \\ \eta_k^y \end{bmatrix}$, $\mathbf{W} = \begin{bmatrix} \mathbf{W}_f \mathbf{V}_f & \mathbf{0} \\ \mathbf{W}_h \mathbf{V}_f & \mathbf{0} \end{bmatrix}$, $\mathbf{V} = \begin{bmatrix} \mathbf{0} & \mathbf{V}_h \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_h \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_f \\ \boldsymbol{\theta}_h \end{bmatrix}$. Then equations in (6) may be represented compactly as shown in (7). Although it is possible to reduce the number of parameters during optimization by considering the exact structure of **W** and **V**, for the sake of simplicity, it is assumed that **W** and **V** are full matrices. Note that this means that we are considering a more flexible structure than the minimal

Author's personal copy

requirement and any resulting additional computational cost may be offset by this flexibility. Anyway this does not change the scaling of the algorithm as the number of parameters to optimize with each additional node would remain linear in terms of the number of hidden nodes in both the cases. A graphical representation of the resulting RNN architecture is shown in Fig. 1.

$$\eta_{k+1} = \mathbf{W}\sigma(\eta_k) + \mathbf{B}u_{k+1} + \boldsymbol{\theta}$$
$$y_{k+1} = \mathbf{V}\sigma(\eta_{k+1}) \tag{7}$$

Assuming that $\sigma(x)$ is given by $\tanh(x)$, which has a maximum slope of 1, it may be shown using the contraction mapping theorem [21] that the discrete dynamic system given by (7) is globally asymptotically stable if $\|\mathbf{W}\| < 1$ [21]. Here, $\|\mathbf{W}\|$ denotes the induced matrix norm or the spectral norm. For further analysis, only the dynamic part of (7), given by $\eta_{k+1} = \mathbf{W}\sigma(\eta_k)$, is considered as the inputs do not interact with the states η_k and hence do not affect the stability of the system but can only change the equilibrium point.

The contraction mapping theorem states that a mapping, $g(\eta) : B_a^n \to B_a^n$, where B_a^n is a n-dimensional ball of radius a centered about the origin (i.e., $\eta \in B_a^n \Rightarrow \|\eta\| \le a$) has a unique fixed point in B_a^n if the function g satisfies the following two conditions:

- There is a constant $G < 1$, such that $\|g(\eta_1) - g(\eta_2)\| \le G\|\eta_1 - \eta_2\|$ for $\eta_1 \in B_a^n, \eta_2 \in B_a^n$.
- $\|g(0)\| \le (1-G)a$

Further, the sequence $\eta_{k+1} = g(\eta_k)$ can be shown to converge to this fixed point. For the case of the RNN, $g(\eta) = \mathbf{W}\sigma(\eta)$. It is easy to see that if $\sigma(x)$ is assumed to be $\tanh(x)$, then $g(0) = 0$, thus satisfying condition 2 for any $G < 1$ and all a. Moreover, by definition, this is a fixed point of the mapping $g(\eta) : B_a^n \to B_a^n$. Now, if the assumption $\|\mathbf{W}\| < 1$ holds (the spectral norm is used here), then it can be shown that the first condition also holds with $G = \|\mathbf{W}\|$ as given below:

$$\begin{aligned}\|g(\eta_1) - g(\eta_2)\| &= \|\mathbf{W}\sigma(\eta_1) - \mathbf{W}\sigma(\eta_2)\| \\ &\le \|\mathbf{W}\|\,\|\sigma(\eta_1) - \sigma(\eta_2)\| \\ &\le \|\mathbf{W}\|\,\|\eta_1 - \eta_2\| \\ &= G\|\eta_1 - \eta_2\|\end{aligned}$$

The above inequality makes use of the fact that $\|\sigma(\eta_1) - \sigma(\eta_2)\| \le \|\eta_1 - \eta_2\|$, which is based on the assumption that the slope of the sigmoid function is less than one and hence the inequality holds element-wise and also for the entire vector. Moreover, the above two conditions are met in any ball of radius a and hence they hold globally. Therefore, the RNN is globally asymptotically stable with fixed point at $\eta = 0$ (assuming zero input) if $\|\mathbf{W}\| < 1$.

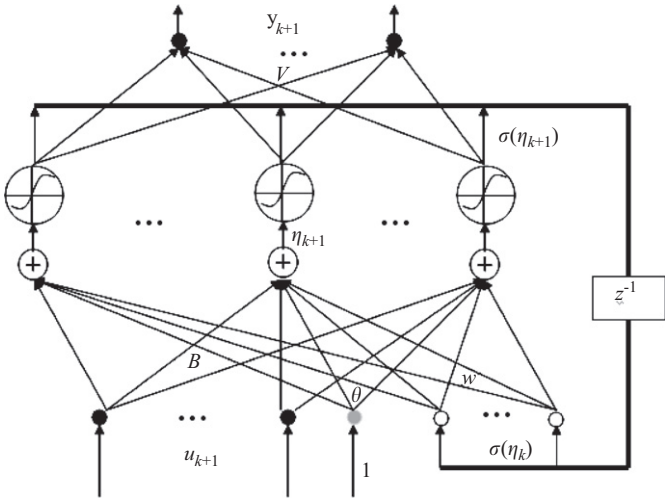


Fig. 1. Proposed structure for the recurrent neural network.

3. Constructive structure and parameter learning

In this section, a constructive algorithm for learning the structure (number of hidden nodes) and parameters of an RNN given by (7) is presented. A novel parameterization of the network weight parameters, especially matrix $\mathbf{W}$, is proposed in this section. This parameterization allows the imposition of bounds on the singular values of $\mathbf{W}$ and thus ensures stability of the network throughout the constructive training process.

Structure learning in the proposed constructive method is straight-forward and involves the successive addition of neurons or hidden nodes until the training error drops below a pre-specified limit or the maximum specified number of nodes have been added. It is assumed that after $(l-1)$ neurons have been added to the hidden layer using the constructive procedure, the parameters $\mathbf{W}_{l-1}, \mathbf{B}_{l-1}, \boldsymbol{\theta}_{l-1}$ and $\mathbf{V}_{l-1}$ are available. It should be noted that from now on, only the structure in (7), where the matrices $\mathbf{W}$, $\mathbf{B}$, $\boldsymbol{\theta}$ and $\mathbf{V}$ have distinct roles, is used and the subscript l denotes the number of hidden nodes and should not be confused with the subscripts f and h used earlier. In order to add the lth neuron to the network, it is further assumed that $\mathbf{W}_l = \begin{bmatrix} \mathbf{W}_{l-1} & \mathbf{w}_2 \\ \mathbf{w}_1^T & \mathbf{w}_3 \end{bmatrix}$, $\mathbf{B}_l = \begin{bmatrix} \mathbf{B}_{l-1} \\ \mathbf{b} \end{bmatrix}$, $\boldsymbol{\theta}_l = \begin{bmatrix} \boldsymbol{\theta}_{l-1} \\ t \end{bmatrix}$, i.e., the sub-matrices of $\mathbf{W}_l$, $\mathbf{B}_l$ and θ_l corresponding to the previously added hidden nodes are assumed to be fixed. The parameters $\mathbf{w}_1 \in R^{l-1}$, $\mathbf{w}_2 \in R^{l-1}$, $\mathbf{w}_3 \in R$, $\mathbf{b} \in R^m$ and $t \in R$ are then learnt by using particle swarm optimization (PSO) with the network prediction error as the objective function to be minimized.

Particle swarm optimization (PSO) belongs to the class of population based, stochastic, iterative, optimization techniques and was inspired by social behavior of bird flocking or fish schooling as opposed to evolutionary strategies, which were inspired by Darwin's theory of evolution. In PSO, the system is initialized with a population of random solutions, called particles and the algorithm searches for optima by flying these particles through the problem space by making use of the current knowledge of optimum locations. In its simplest form, the position of each particle is iteratively updated by following two "best" values. The first one is the best solution the particle has achieved so far and the second one is the best value, obtained so far by any particle in the population. A randomly weighted combination of the vectors pointing towards these "best" values is then used to update the particle's location in a manner that incorporates both exploration and optimum seeking behavior [12].

In order to calculate the prediction error of the network, the matrix $\mathbf{V}_l$ is learnt using a least squares technique. First, the dynamics of the hidden node can be simulated for the entire time for which the training input data are available using the first equation of (7) with $\mathbf{W}_l$, $\mathbf{B}_l$ and θ_l as the network parameters. Let $\mathbf{Z}_l$ be a matrix with the so-obtained values of $\sigma(\eta_k)$ arranged along the kth row, where k denotes the time index. Further let $\mathbf{Y}$ denote the matrix obtained by arranging the y_k values along the kth row. Then, $\mathbf{V}_l$ is determined by the least squares solution to the problem $\mathbf{Y}_l = \mathbf{Z}_l\mathbf{V}_l^T$. This is easily done using the pseudo-inverse of $\mathbf{Z}_l$ and is given by

$$\mathbf{V}_l = \left[(\mathbf{Z}_l^T\mathbf{Z}_l)^{-1}\mathbf{Z}_l^T\mathbf{Y}_l\right]^T \tag{8}$$

Therefore, for given $\mathbf{W}_l$, $\mathbf{B}_l$ and θ_l it is straight-forward to obtain $\mathbf{V}_l$. And once $\mathbf{V}_l$ is estimated, it is possible to obtain the predicted

Author's personal copy

network outputs using this estimate in the second part of (7). Therefore, given the network parameters with $l-1$ nodes and a candidate solution for the parameters $\mathbf{w}_1 \in R^{l-1}$, $\mathbf{w}_2 \in R^{l-1}$, $\mathbf{w}_3 \in R$, $\mathbf{b} \in R^m$ and $t \in R$, it is possible to predict the output of a network with l nodes. Therefore, PSO is used to optimize a $(2l+m-1)$ dimensional problem to determine $(\mathbf{w}_1,\mathbf{w}_2,\mathbf{w}_3,\mathbf{b},t)$. Unlike in the case of static neural networks, where the number of parameters to be optimized remains constant, the number of parameters to be optimized as each hidden node is added increase linearly for an RNN. This is in accordance with the assumption that more complex dynamics are added to the system for each additional hidden neuron. The objective function is calculated by normalizing the prediction error of the network by comparing it to a dummy model, which estimates the output at the previous time instant to be the output at the next time instant as shown in (9). In (9), the subscript o is used to represent the outputs, p is the total number of outputs and V_l^o corresponds to the o^{th} row of $\mathbf{V}_l$.

$$\mathbf{E} = \frac{\sum_{k=1}^{N}\sum_{o=1}^{p}\left(y_k^o - V_l^o\sigma(\eta_k)\right)^2}{\sum_{k=1}^{N}\sum_{o=1}^{p}\left(y_k^o - y_{k-1}^o\right)^2} \tag{9}$$

While the above parameterization seems the most straightforward, it offers no control over the stability of the RNN as the constructive procedure continues. This can pose a real problem to the constructive procedure, as during the initial stages there is a tendency for the network to use large values of parameters that make the system unstable but still provides a better approximation to finite length trajectories in the training data. With the simple parameterization of $\mathbf{W}_l$ in terms of $\mathbf{W}_{l-1}$, proposed previously, once $\mathbf{W}_{l-1}$ becomes unstable (i.e., its maximum singular value becomes greater than 1), there is no solution for $(\mathbf{w}_1,\mathbf{w}_2,\mathbf{w}_3)$ that can stabilize $\mathbf{W}_l$. This can lead to very poor results for the constructive procedure with the error stagnating at a very high value since the system cannot be stabilized. Therefore, instead of this simple parameterization, another parameterization of $\mathbf{W}_l$ is proposed here. Let $\mathbf{W}_{l-1} = \mathbf{U}_{l-1}^1\mathbf{S}_{l-1}(\mathbf{U}_{l-1}^2)^T$ be the singular value decomposition of $\mathbf{W}_{l-1}$, where $\mathbf{U}_{l-1}^1$ is an orthogonal matrix of left eigen-vectors, $\mathbf{U}_{l-1}^2$ is an orthogonal matrix of right eigen-vectors and $\mathbf{S}_{l-1}$ is a diagonal matrix of singular values. The matrices $\mathbf{U}_l^1$, $\mathbf{S}_l$ and $\mathbf{U}_l^2$ are derived from $\mathbf{U}_{l-1}^1$, $\mathbf{S}_{l-1}$ and $\mathbf{U}_{l-1}^2$ using a $(2l-2)$ dimensional parameter vector given by $(\mathbf{u}^1,s,\mathbf{u}^2)$, where $\mathbf{u}^1 \in R^{l-1}$, $s \in R$ and $\mathbf{u}^2 \in R^{l-1}$. The matrix $\mathbf{U}_l^1$ is then derived as the orthogonal matrix obtained by using a QR decomposition of the augmented matrix $\begin{bmatrix}\mathbf{U}_{l-1}^1 \\ \mathbf{u}^1\end{bmatrix}$. $\mathbf{U}_l^2$ is obtained from $\mathbf{U}_{l-1}^2$ and $\mathbf{u}^2$ in a similar fashion, $\mathbf{S}_l$ is obtained from $\mathbf{S}_{l-1}$ by adding s as the last diagonal term and finally $\mathbf{W}_l$ may be obtained using $\mathbf{W}_l = \mathbf{U}_l^1\mathbf{S}_l(\mathbf{U}_l^2)^T$. In this case, it may be noted that although the dimension of the parameter space being searched remains the same $(2l+m-1)$, it is possible that all the entries of $\mathbf{W}_l$ are different compared to $\mathbf{W}_{l-1}$. The main advantage of this parameterization is that it is easy to ensure stability of the RNN by using a simple box constraint on s during each stage to restrict it to a range between zero and one. PSO is still used to optimize this problem and once $\mathbf{W}_l$ is computed, the rest of the procedure to calculate the objective function remains the same as before. PSO is used for this problem as it is easy to implement and has relatively few parameters to tune. Moreover, it has been observed to converge to the vicinity of the global solution quickly.

Finally, although the constructive method provides a computational advantage by exploiting the trained parameters of smaller networks and adding the best possible node at each stage in a sequential manner, it is necessary to realize that this greedy approach can lead convergence to a "local minimum" and thus lead to stagnation in training error as more nodes are added. Therefore, it is important to allow optimization of the entire network parameters during such stagnations in the learning process to allow the algorithm to escape from such local minima. During the constructive procedure, if the drop in error after the addition of two nodes is less than 10%, the last two nodes are deleted from the network and the parameters of the entire network are then optimized using CMA-ES starting from the parameters determined by the constructive procedure.

CMA-ES also belongs to the class of population-based, stochastic, iterative optimization methods and is typically applicable to unconstrained or bounded constraint non-linear optimization problems in the continuous domain. The covariance matrix adaptation (CMA) is a method to update the covariance matrix of a multivariate normal mutation distribution in the evolution strategy. New candidate solutions are sampled according to the mutation distribution and the covariance matrix describes the pairwise dependencies between the variables in the distribution. Adaptation of the covariance matrix amounts to learning a second order model of the underlying objective function similar to the approximation of the inverse Hessian matrix in the quasi-Newton method in classical optimization. Hence it is expected to be faster than other population based techniques when the search is conducted close to the optimal solution. The CMA-ES also has several desirable invariance properties (i.e., its performance is unchanged) such as invariance to strictly monotonic transformations of the objective function value and invariance to affine transformations of the search space. The CMA-ES is used for this procedure, because unlike many other direct optimization methods, it is known to be computationally efficient and comparable to second order gradient search techniques for local search [8]. Moreover, it makes good use of the initialization to perform a local search around the initial point if the step size is chosen sufficiently small. It is expected that the constructive procedure gives a good starting point and the optimization of the entire network from this point using CMA-ES should enable the algorithm to find a near optimal solution. The final constructive algorithm is as given below:

Algorithm 1.

Required Inputs: (1) Measured inputs and outputs of the system 2) Target training error or maximum number of nodes.

Initialization: Set the number of nodes (l) to zero. $\mathbf{W}_0=[]$,$\mathbf{B}_0=[]$, $\theta_0=0$ and $\mathbf{V}_0=[]$

Do

$l=l+1$

$\mathbf{P}=[\mathbf{u}^1 s\ \mathbf{u}^2\ \mathbf{b}\ t]$ //Non-linear parameter set to be optimized

Find the optimal $\mathbf{P}$ using PSO which optimizes (9) with $\mathbf{W}_l,\mathbf{B}_l,\boldsymbol{\theta}_l$ obtained from $\mathbf{W}_{l-1},\mathbf{B}_{l-1},\boldsymbol{\theta}_{l-1}$ and $\mathbf{P}$.

Find $\mathbf{V}_l$ using least squares method for the optimal $\mathbf{W}_{l-1},\mathbf{B}_{l-1},\boldsymbol{\theta}_{l-1}$.

Find the prediction error for this model.

If (drop in prediction error over the last two nodes $< 10\%$)

Roll back the network to its state two nodes ago

$l=l-2$

If the entire network has not been optimized with l nodes

Optimize entire network parameters using CMA-ES

Else

return;

End If

End If

If ($l >$ Max. no. of nodes or prediction error $<$ allowed error)

return;

End Do

Both PSO and CMA-ES optimize the same problem-type and hence the decision to use both of them is further clarified here. The PSO was chosen for the constructive phase because of its robustness to parameter initialization and fast convergence

Author's personal copy

properties. This is consistent with the expectation that no good guess is available for the parameters of a newly added node and that a quick estimation of the value of node addition is required during the constructive phase. CMA-ES was chosen for the global optimization phase because of its ability to perform good local search for badly conditioned objective functions (the error surface for recurrent neural networks is expected to have deep and narrow ravines) and also because it is able to exploit information about the good starting point provided by the constructive training process. Thus the two algorithms are expected to complement each others' strengths and weaknesses to make the overall learning algorithm more robust and efficient.

Finally we provide some discussion about the use of such a constructive procedure to approximate the dynamics of unstable systems. Since the constructive procedure forces the RNN to be stable at all times, trying to approximate an unstable system with such a strategy may not give very good results. On the other hand, while RNNs are capable of modeling the dynamics of any system, one of the major problems encountered when trying to model unstable systems using unstable RNNs is the fact that the simulated dynamics would be dependent on the assumed initial conditions of the RNN (which are chosen randomly). Therefore, during training the parameters of the RNN have a very high tendency to overfit the training data using the randomly chosen initial conditions. A good method to overcome this problem is to augment the input vector with the outputs at the previous time instant, i.e., $u_k^{\text{mod}} = \begin{bmatrix} u_k \\ y_{k-1} \end{bmatrix}$. In this case, we could consider the structure of the RNN to approximate a non-linear observer for the dynamic system, and this can be assumed to be stable even though the original system itself is not.

4. Results

Both the examples considered in this section are MIMO non-linear processes represented in the state space form. The experiments were repeated ten times each and the final results (variation of training and testing RMSE values with the number of nodes) were almost identical for each run of the proposed method indicating a certain degree of robustness of the learning algorithm. Hence only the results from a single typical run are presented for each example in this section. The performance of the constructive RNN training procedure is compared against the use of constructive procedure for static systems with delayed inputs and outputs (non-linear ARMA models represented by radial basis function networks or RBFNs designed using OLSGA) [13]. While the OLSGA has the capability to automatically determine the optimal number of hidden nodes to use in an RBFN, it is not capable of selecting the best inputs or delay terms. In the two synthetic examples given below, since the exact model structure is known, the optimal number of input and output delay terms, as determined from the structure of the state space equations, were used for modeling purposes to obtain the best possible results for OLSGA. It should however be noted that, in practice this information is not available and could result in degraded performance for the OLSGA. At the very least it would add significantly to the amount of manual intervention required in model building. The results seem to indicate that the RNN model can achieve comparable accuracy. Although the prediction accuracies are comparable, the RNN model has the following advantages:

(1) It does not require tuning or selection of the delay terms for the inputs and outputs.

(2) It actually learns the dynamics of the system, which makes it useful for long term prediction. This is useful in applications like model predictive control and prognostics.

For more details about the experimental settings (such as the optimization parameter settings and so on) and results on additional datasets, the reader is referred to the supplementary material presented with this paper.

Example 1. The dynamics of the system considered in this example is given by

$$x_1(k+1) = 0.5\left[\frac{x_1(k)}{(1+x_2^2(k))} + u_1(k)\right]$$
$$x_2(k+1) = 0.5\left[\frac{x_1(k)x_2(k)}{(1+x_2^2(k))} + u_2(k)\right]$$
$$y(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(k) \tag{10}$$

The training data were generated by using swept sine waves as inputs to the system. 600 data points were used for training. The test data included data generated using swept sine waves with frequencies outside the range used for training as well as step inputs. 2000 data points were used for testing. The data for RNN training was split into 3 batches to make the RNN robust to initial conditions. The optimal delays were used for training the RBFN, i.e., $y(k)$ and $u(k)$ were used as inputs to predict $y(k+1)$. A comparison of the actual and predicted outputs using RNN and RBFN are given in Figs. 2 and 3, respectively. The variation in training and testing errors versus the number of nodes added to the network is given in Fig. 4 for the two cases. During training, it was observed that global optimization was performed for the first time after the addition of the 9th node. It can be seen from Fig. 4 that although this reduced the training error, it resulted in an increase in the testing error indicating overfitting. Thus, for this example, it appears that the constructive procedure itself (say with 8 nodes) was sufficient to do an excellent job of modeling the process.

The fact that the test RMSE for both the methods is so close, inspite of the fact that the RNN makes use of only the inputs at the current time step to predict the next output value indicates that the RNN has learnt the dynamics of the system effectively and is a good model for model predictive control and prognostics. In both the cases, it can be seen that the model tends to overfit the

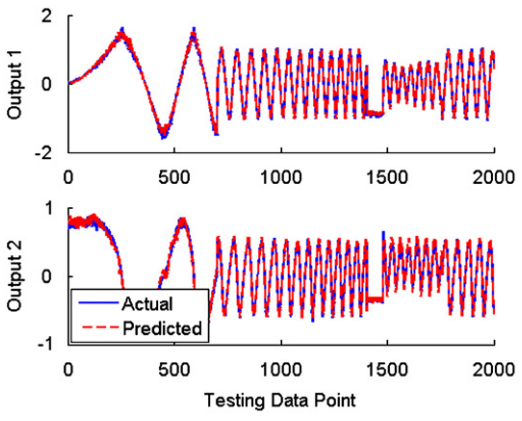


Fig. 2. Comparison of actual and predicted output values for Example 1 during testing using RNN (Output 1 RMSE: 0.0661, Output 2 RMSE: 0.0640).

Author's personal copy

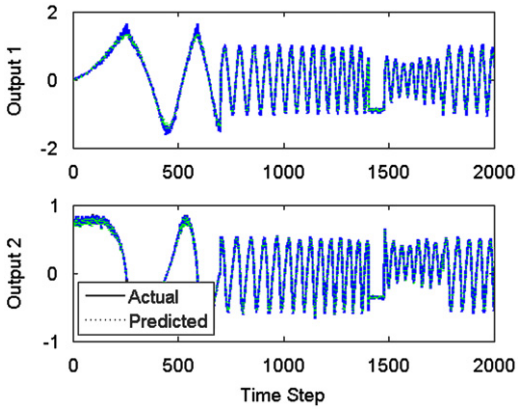


Fig. 3. Comparison of actual and predicted output values for Example 1 during testing using RBFN (Output 1 RMSE: 0.0624, Output 2 RMSE: 0.0540).

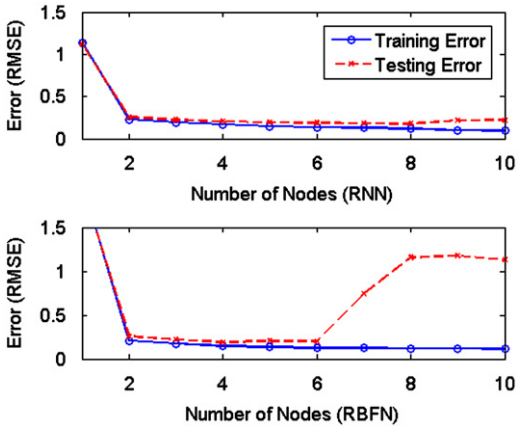


Fig. 4. Training and testing error versus number of nodes for RNN and RBFN.

training data as more nodes are added to the network, but the effect of overfitting seems to be significantly mitigated for the RNN.

Example 2. The dynamics of the system considered in this example is given by

$$x_1(k+1) = 0.5\tanh(x_1(k)x_3(k)) + \left[2 + \frac{1.5x_1(k)u_1(k)}{1+x_1^2(k)u_1^2(k)}\right]u_1(k) + x_4(k)$$

$$x_2(k+1) = [3+\tanh(2x_1(k))]u_2(k)$$

$$x_3(k+1) = x_2(k)[1+\tanh(4x_2(k))] + \left[\frac{x_1(k)}{1+x_1^2(k)}\right]$$

$$x_4(k+1) = \left[0.1x_1(k) + \frac{2x_1(k)}{1+x_1^2(k)}\right]u_2(k)$$

$$y(k) = \begin{bmatrix} x_1(k) & x_2(k) & x_3(k) \end{bmatrix} \quad (11)$$

Almost the same training and testing conditions were used as in Example 1. The training data were generated by using swept sine waves as inputs to the system. 600 data points were used for training. The test data included data generated using swept sine waves with frequencies outside the range used for training as well as step inputs. 2000 data points were used for testing. The data for RNN training were split into 3 batches to make the RNN robust to initial conditions. Considering the structure of the state space

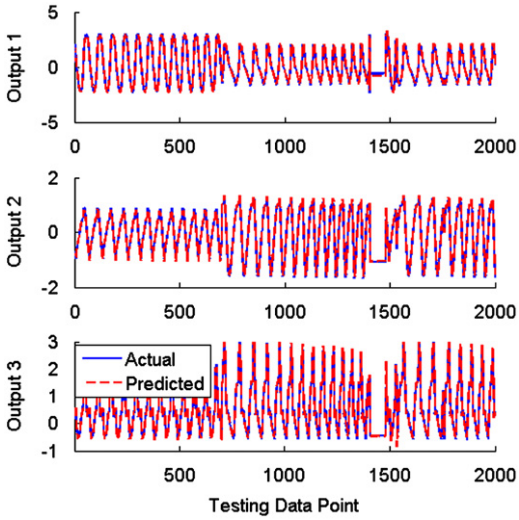


Fig. 5. Comparison of actual and predicted values for Example 2 during testing using RNN (Output 1 RMSE: 0.1508, Output 2 RMSE: 0.1308, Output 3 RMSE: 0.0840).

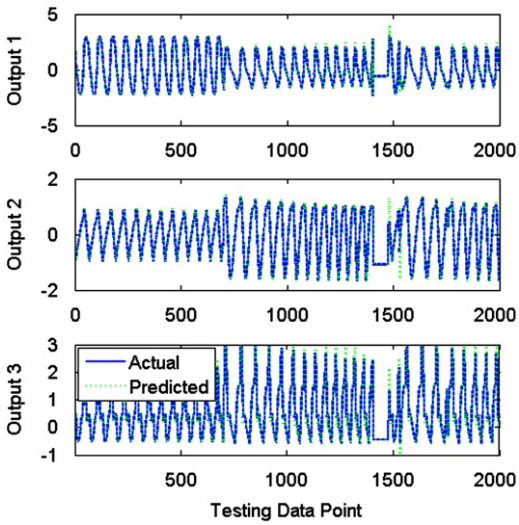


Fig. 6. Comparison of actual and predicted values for Example 2 during testing using RBFN (Output 1 RMSE: 0.1073, Output 2 RMSE: 0.0812, Output 3 RMSE: 0.1548).

model, $y(k)$, $y(k-1)$, $u(k)$ and $u(k-1)$ were used for training the RBFN. A comparison of the actual and predicted outputs using RNN and RBFN is given in Figs. 5 and 6, respectively. The variation in training and testing errors versus the number of nodes added to the network is given in Fig. 7 for the two cases. Again, it can be seen from Fig. 7, by the additional drop in training error for the RNN that global optimization was performed after the addition of the 8th node. Unlike the previous example, here the global optimization helps overcome the stagnation of the training process and results in a decrease in both training and testing error thus indicating the learning of a better model. Thus, this example demonstrates the advantage of using global optimization when the constructive procedure stagnates.

The exact same conclusions regarding the modeling capabilities as well as the robustness to overfitting can be drawn

Author's personal copy

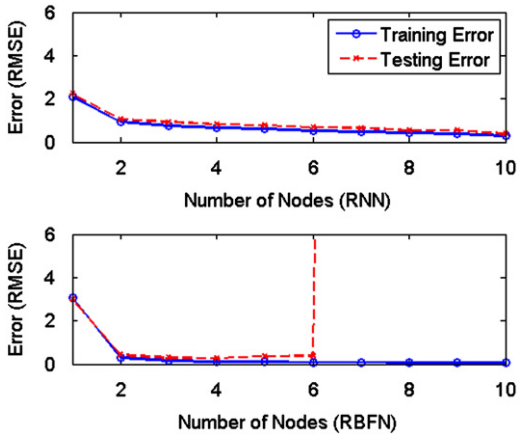


Fig. 7. Training and testing error versus number of nodes for RNN and RBFN.

regarding the RNN model. Therefore, the architecture and parameterization proposed for recurrent neural networks are sufficient to capture the dynamics of the MIMO processes considered in the examples. The specialized parameterization of the matrix **W** allows the algorithm to ensure stability of the resultant RNN throughout the constructive process and this in turn helps ensure that the resulting training error decreases monotonically throughout the training process. The results presented highlight the capabilities of the proposed algorithm to automatically determine the structure (number of hidden nodes) as well as the optimal parameters of the RNN.

5. Conclusions

This work presented a sufficient architecture and parameterization for recurrent neural networks to capture the dynamics of any process that can be represented in the state space form. A novel constructive method for RNNs with this architecture and parameterization was presented using a hybrid optimization strategy, which makes use of PSO for the constructive phase and CMA-ES for overall tuning of the entire network. The results presented highlight the capabilities of the proposed algorithm to automatically determine the structure (number of hidden nodes) as well as the optimal parameters of the RNN. Although, the algorithm worked well in the experiments presented, it is possible that the constructive method could over-estimate the number of hidden nodes required. Hence, future work will concentrate on developing a suitable pruning method, which can be used in conjunction with the algorithm presented here for determining an optimal structure for RNNs.

The constructive method proposed in this paper achieves a few important objectives, which make it a viable candidate for automated structure and parameter learning in Elman-type recurrent neural networks. It achieves a good tradeoff between computational complexity, difficulty of optimization (combinatorial optimization versus continuous optimization) and model performance. The method reduces computational complexity by optimizing for only a few parameters at a time and performing parameter optimization for the entire model only when necessary. Moreover, the effective use of least-squares technique to learn the parameters of the output layer further reduces the number of parameters optimized by PSO/CMA-ES. It avoids combinatorial optimization by using constructive learning for structure identification. The experimental results seem to indicate that the performance of the models trained using this method is excellent. Specialized parameterization of the matrix **W** allows the algorithm to ensure stability of the resulting RNN throughout the constructive process and this in turn helps ensure that the resulting training error decreases monotonically throughout the training process. This also allows the user to specify either the exact value or a range of the singular values of the matrix **W**, which determines the dynamics of the system. Moreover, multiple runs of the method during the experiments gave very similar results during training and testing indicating that the method is robust to initializations and thus avoids the need to train multiple times. The method also seems to be robust to overfitting as indicated by the fact that the testing error does not increase significantly with the addition of redundant nodes to the network. Overall, the proposed method is particularly suited for small to medium scale problems, especially for modeling discrete-time continuous state dynamic systems.

Appendix A. Supplementary materials

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.neucom.2010.05.012.

References

[1] P. Angeline, G. Saunders, J. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1) (1994) 54–65.
[2] J. Chia-Feng, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, IEEE Transactions on Systems, Man, and Cybernetics, Part B 34 (2) (2004) 997–1006.
[3] M. Delgado, M.C. Pegalajar, A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference, Pattern Recognition 38 (9) (2005) 1444–1456.
[4] J.L. Elman, Finding structure in time, Cognitive Science 14 (1990) 179–211.
[5] A. Esparcia-Alcazar, K. Sharman, Evolving recurrent neural network architectures by genetic programming, in: Genetic Programming 1997: Proceedings of the Second Annual Conference, 1997, pp. 89–94.
[6] S. Fahlman, The recurrent cascade-correlation architecture, Technical Report CMU-CS-91- 100, 1991.
[7] C. Giles, D. Chen, G. Sun, H. Chen, Y. Lee, M. Goudreau, Constructive learning of recurrent neural networks: limitations of recurrent cascade correlation and a simple solution, IEEE Transactions on Neural Networks 6 (4) (1995) 829–836.
[8] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2) (2001) 159–195.
[9] S. Haykin, in: Kalman Filtering and Neural Networks, Wiley and Sons, New York, 2001.
[10] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, in: J.F. Kolen, S. Kremer (Eds.), A Field Guide to Dynamical Recurrent Networks, IEEE Press, New York2001, pp. 237–243.
[11] M. Jordan, Generic constraints on underspecified target trajectories, in: Proceedings of the International Joint Conference on Neural Networks, vol. 1, 1989, pp. 217–225.
[12] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, 1995, pp. 1942–1948.
[13] C.W. Lee, Y.C. Shin, Growing radial basis function networks using genetic algorithm and orthogonalization, International Journal of Innovative Computing, Information and Control, 5 (11A) (2009) 3933–3948.
[14] M. Lehtokangas, Constructive backpropagation for recurrent networks, Neural Processing Letters 9 (3) (1999) 271–278.
[15] L.R. Medsker, L.C. Jain, in: Recurrent Neural Networks—Design and Applications, CRC Press, New York, 2000.
[16] B. Pearlmutter, Gradient calculations for dynamic recurrent neural networks: a survey, IEEE Transactions on Neural Networks 6 (1995) 1212–1232.
[17] G.V. Puskorius, L.A. Feldkamp, Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent neural networks, IEEE Transactions on Neural Networks 5 (2) (1994) 279–297.
[18] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge, 1986, p. 45.
[19] H.T. Siegelmann, E.D. Sontag, Turing computability with neural nets, Applied Mathematics Letters 4 (6) (1991) 77–80.
[20] A.C. Tsoi, A.D. Back, Discrete-time recurrent neural network architectures: a unifying review, Neurocomputing 15 (3) (1997) 183–223.
[21] M. Vidyasagar, in: Nonlinear Systems Analysis, Prentice-Hall International Editions, 1993.

Author's personal copy

[22] P. Werbos, in: The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting, Wiley, New York, 1993.
[23] R. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural Computation 1 (1989) 270–280.

Niranjan Subrahmanya received his Ph.D. degree from Purdue University, Indiana, USA, in May 2009. His research interests include the development of machine learning and artificial intelligence techniques with applications in data-based systems modeling, monitoring and diagnostics. He is also interested in non-linear system identification and control.

Yung C. Shin received his Ph.D. in mechanical engineering from University of Wisconsin in Madison in 1984, and currently is a professor of mechanical engineering at Purdue University. He worked as a senior project engineer at the GM Technical Center from 1984 to 1988 and faculty at the Pennsylvania State University from 1988 to 1990, prior to joining Purdue University in 1990.

His research areas include intelligent and adaptive control, process monitoring and diagnostics, laser processing of materials, high speed machining, process modeling and simulation. He has published over 200 papers in archived journals and refereed conference proceedings and has authored chapters in several engineering handbooks and co-edited two books. He also is a coauthor of *Intelligent Systems: Modeling, Optimization and Control* (CRC Press, 2008). He has organized or coorganized many conferences and symposia in his areas of research including the first and second "Artificial Neural Networks in Engineering Conference", and "Symposium on Neural Networks in Manufacturing and Robotics" and "Symposium on Intelligent Design and Manufacturing" at the ASME IMECE.