RECURRENT NEURAL NETWORK MODELS OF MULTIVARIABLE DYNAMIC SYSTEMS: NUMERICAL METHODS AND EXPERIMENTAL EVALUATION

By

Enrique D. Ferreira

Advisor: Prof. Bruce H. Krogh

September 5, 1995

Report

submitted to the department of electrical and computer engineering

and the committee on graduate studies

of Carnegie Mellon University

in partial fulfillment of the requirements

for the degree of

Master in Electrical Engineering

Contents

List of Tables

List of Figures

Abstract

This report covers the development and evaluation of Neural Network models of complex dynamic systems. It describes the experimental and numerical procedures to identify a particular system, a Plasma Enhanced Chemical Vapor Deposition (PECVD) system, located in Carnegie Mellon, using recurrent neural net models for a multi-input multi-output control system. Neural Networks have an advantage over standard techniques for complex non-linear systems when it is very difficult to introduce a priori knowledge in the modeling. The complexity of the PECVD process makes it necessary to develop empirical models of the dynamics. Several possibilities were investigated and improvements were made over existing techniques. Also, some comparisons with standard system identification results were analyzed.

The report describes the work accomplished during the period between September 1994 and June 1995.

1 Introduction

The use of neural networks in system identification and control has been growing in recent years. Narendra [19] [20], and Nerrand et al. [21] have shown many architectures and learning algorithms to deal with different types of system models. Many applications have been developed taking advantage of the properties of these models, e.g. in robotics [27], forecasting [33] and communications [24]. Neural networks are useful for systems that have a very complex nonlinear dynamic behavior and, when incorporated into appropriate control algorithms, offer the possibility to improve the performance over methods based on standard linear identification techniques.

The principal objective of the research reported here was to investigate and evaluate techniques for obtaining more sophisticated models for dynamic systems using neural networks. Starting from the previous works mentioned above we developed a new technique to find a good neural network that models a multivariable dynamic system when the data that can be obtained from the system is corrupted by outliers and non-Gaussian noise.

Such a job is not completed if we do not apply our technique to a problem that has these characteristics. A Plasma Enhanced Chemical Vapor Deposition (PECVD) process can be classified within this group of complex nonlinear systems. The PECVD is a critical process in the manufacture of integrated circuits. In current practice, the PECVD process is controlled by independent regulators for macroscopic variables such as temperature, pressure and gas flow rates. Setpoint values for these variables are determined by off-line studies and applied without adjustment during the processing of a given batch of wafers. Although setpoint tuning may be performed from batch-to-batch using techniques such as statistical process control [26], the use of feedback to adjust the process setpoints in real time has not been possible until recently because of the absence of in situ measurements of process variables more directly related to the actual process performance variables.

Other authors have already applied neural network techniques to model a similar system [5] [7] [10] with successful results. They have focused, however, on the surface response model identification, i.e. a mapping between the setpoint values of the macroscopic variables and performance variables. On the other hand, we are interested in modeling the dynamic behavior of the PECVD system. This is a more difficult task. The work by Engell [3] on a PH-regulator problem and a two-tank system is more related to our work but we have used recurrent neural nets to improve model matching.

The first section of the report presents a brief explanation of neural networks in general. Following, there is a more detailed covering of the application of neural networks to system identification. The usual methods applied and some comments on the advantages and disadvantages of each method are explained. Next we introduce the variations on the algorithms we developed to improve some of the drawbacks with current methods. The next section is dedicated to the experimental application of the algorithms. There is a description of the experimental system and the steps taken to model the

system with this method. In the following section the results of the modeling procedure are shown and compared with other techniques for the same system. In the concluding section the results of this project are summarized and some future lines to investigate on the subject are proposed.

2 Concepts of Neural Networks

A neural network is a set of simple elements, called neurons or units, interconnected and with some inputs and outputs available from the outside. Each interconnection has a specific "weight" or importance which defines the relationship between the inputs and the outputs of the whole net. Thinking in this way, there comes the idea of finding a method to set these weights in order to have the network compute a desired mapping. Looking at this definition of neural network it is clear that, in order to completely define such a structure we have to specify:

1. The units that make up the network.
2. The way in which these units are interconnected.
3. A way to compute the weights, i.e. a learning algorithm.

Now, we introduce these items.

2.1 Concept of a Neuron

From a functional point of view, a unit is simply an active element with some number of inputs and only one output. The relationship between the inputs and the output is given by an equation:

$$V = f(\sum_{i=0}^{n} w_i x_i + u) \tag{1}$$

where x_i are the inputs to the unit, w_i are the weights of each connection between the input and the unit and u stands for the threshold. The function f, or the characteristic function of the unit, can have different shapes. The most usual functions used are:

1. Sign function.
2. Linear function with saturation.

3. Sigmoidal function.
$$f(s) = \frac{1}{1 + e^{-\beta s}} \tag{2}$$

4. Hyperbolic Arctangent function.
$$f(s) = \frac{1 - e^{-\beta s}}{1 + e^{-\beta s}} \tag{3}$$

It is usual to insert the threshold as another input with constant value and varying weight to simplify the equation (1). All these possibilities as well as others that we will see later are used to classify the neural network.

2.2 History: The Biological Neuron

The use of the term neural network is motivated by the study of structures begun by biologists trying to discover how the human brain works. They built these models of the neurons that resembled the real ones with a cellular body, with the dendrites as the inputs and the axon as its output. The dendrites are in general small extensions and are usually connected to other neurons receiving a signal from them. On the other hand, the axon can be very large and it is the output of the neuron, i.e. the element of the neuron that carries the signal that it generates. The biological neuron analyzes external impulses that come through the dendrites and by some "computation" it decides whether or not to fire an impulse through the axon. The way by which this decision is made is very sophisticated and not completely understood at this time. It involves complex chemical mechanisms such as synapses and action potentials which are far from this study. The engineering science took these simplified models trying to understand their properties and searching for new methods to solve problems that cannot be solved by traditional tools. We do not restrict research to the models that biologists use; we create from them other models that have the same philosophy but serve our purposes better. That is where the name *Artificial Neural Network* comes from.

2.3 Network Interconnection

The next point to define is the way by which the units are connected.

1. **Full Interconnection.** Each unit of the network has connections with all the other ones and even with itself.

2. **Layer Interconnection.** The units are arranged in sets called layers and the units can have connections only with units belonging to other layers.

The layer interconnections have some variation depending on the possibility of each layer to have what is called lateral inhibition, i.e. allowing the units belonging to the same layer to be connected,

but these connections are inhibitory. This means that if some unit has a positive output it causes the other neurons in the same layer to lower their outputs. Competition networks acts in this way [6].

2.4 Concept of Learning

Learning means to acquire knowledge of something by experience or study. Applying this concept to the structure we are defining we say that learning means the ability of neural network structures to classify or distinguish, using some measure, the inputs to them. This is possible through the appropriate selection of the interconnection weights mainly or being more generally through the appropriate selection of the structure of the network. The outputs of the network give us the classification. This classification can be static or time varying. This suggests using neural networks in pattern classification problems, associative memory as well as in the representation of dynamic systems.

The process of selection of the weights is what is called the **learning algorithm** and it is one of the main topics of most papers on neural networks. Each problem can be solved quickly or more efficiently using the correct algorithm, and sometimes there is only one way to do it right.

We can classify the learning methods as follows:

1. **Supervised Learning.** This means that the classes we would like the network to learn are pre-specified and we "tell" the network about them. In other words, we know the classification and we use it in the learning algorithm to train the network. Multilayer networks are usually in this class.

2. **Unsupervised Learning.** This is of course the opposite of the supervised learning. We let the network to do the classification. Usually, this classification is made on the basis of some measure of the input space and the statistical distribution of the input vectors presented to the network. A Kohonen network is a very good example of this type of networks [13].

All these characteristics allow us to make a general classification of any type of neural network. The most known types of neural networks are the following:

2.4.1 Hopfield Networks

This is a fully interconnected neural network. The basic structure uses the sign function in the units and in general all the signals used are digital (or 1 and -1). All the units are accessible from the outside and are at the same time inputs and outputs. More than this, they are the state of the whole network. A learning algorithm usually used is the Hebb's rule which can be described as follows.

Given the set of patterns x_{ij}, i.e. the desired values of all the unit outputs j for each pattern i the values of the weights are computed as.

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{N} x_{ki} x_{kj} \tag{4}$$

The main use of this kind of network is in information retrieval. The usual procedure is the following. We set the activation of each neuron to some arbitrary value and let the network evolves using the equation (1) to calculate the activation values of the units in the next time step. It is desirable that the neural network reaches a stable state where the activation values of the network are identical to the ones in a pattern. Several papers by Hopfield and others try to evaluate the performance of the neural network. Some bounds on the number of patterns to learn with respect to the size of the network were found [9]. However, this network has the disadvantage of the creation of spurious or false patterns affecting the recognition process. Also, not any set of patterns can be learned by the Hopfield network using the Hebb's rule. A detailed study of these problems, some counterexamples and ways to avoid them can be found for example in [6] and [18]. Hopfield proposed some applications in the engineering field such as A/D convertors, optimization problems and pattern recognition [8].

This is also the simplest version of the Hopfield Network. There are many variations to this structure and algorithms improving its performance.

2.4.2 Multilayer Network

This is by far the most used network in the engineering world. It is a neural network structure composed of several ordered layers of neurons connected in sequence without lateral inhibition. The first layer is accessible from the outside through its inputs and we can observe the outputs of the last layer. The information flows then in only one direction, from the first to the last layer. It is called a *feedforward neural network* because of that. The neurons in each layer have the same behavior, but they can be different from the ones in other layers. It is quite common to use linear functions in the input and output layers and sigmoidal or hyperbolic tangent functions in the inner or hidden layers. There exists many learning methods to train the weights of this type of network. The usual method employed is the *Backpropagation* method, attributed to Rumelhart, Hinton and Williams [25], although it is not the most efficient. It is a supervised learning method which assumes the knowledge of the patterns to learn in the form of pairs of vectors representing the values of the inputs x_{ij} and its corresponding output $\hat{y}_{ik}$, where the index i is the pattern number, j the unit number in the input layer and k the unit number in the output layer. The algorithm minimizes the sum for all the patterns of the square difference between the desired output and the actual output y_{ik},

$$E[\mathbf{w}] = \frac{1}{2} \sum_{ik} (\hat{y}_{ik} - y_{ik})^2 \tag{5}$$

This error is a function of the weights of each unit in the network. The backpropagation algorithm minimizes this sum using the ordinary steepest descent technique. To do that it needs to calculate the partial derivatives of the expression (5) with respect to each weight of the network. The backpropagation algorithm can be summarized by these expressions. At each step the weights of the networks are updated using

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij} \tag{6}$$

where Δw_{ij} is evaluated as,

$$\Delta w_{ij}^{m} = \eta \delta_i^m y_j^{m-1} \tag{7}$$

with

$$\delta_i^{m-1} = f'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \tag{8}$$

$$\delta_i^M = f'(h_i^M)(\hat{y}_i - y_i) \tag{9}$$

and

$$y_i^m = f(h_i^m) = f(\sum_j w_{ij}^m y_j^{m-1}) \tag{10}$$

In these expressions w_{ij}^m refers to the weight that links the output of the j unit in the $m-1$ layer to the i unit of the m layer, M is the total number of layers in the network and η is a constant usually called the learning coefficient. It can be noted that while the output of each layer is computed from the previous ones through equation (10), the update values for the weights must be computed the other way around. Equations (8) and (9) allow us to compute delta values from the output layer M back to the input layer, hence the name backpropagation. This algorithm has been studied for several years and many extensions and modifications have been considered. The basic algorithm given above is slow and may fail to converge in a multilayer network. The modifications suggested tend to improve speed of convergence, avoid local minima and improve generalization. The most common improvement is the introduction of a momentum term in the equation (6) to take into account not only the current gradient direction but also the previous one. This variation improves speed notably. Later, an algorithm usually called *quickprop* was developed which gives a significantly improvement in speed with respect to the original algorithm [4]. The main reason why this type of network is used nowadays is because there are mathematical results which states that a three-layer neural network can learn any function [2]. A hidden layer gives us the ability to reproduce a function with the desired accuracy. However, it has to be understood that the theorem does not state the number of hidden units required to achieve this goal. Other works have found some bounds on this important issue. Also very important is the fact that even though one knows that any function can be approximated by a three-layer neural network, it is not sure that backpropagation can always do the job. The starting point to reach the global minimum or using refined mathematical techniques to optimize expression (5) is needed.

2.4.3 Kohonen Self-Organizing Map

The neurons in this network are arranged in a bi-dimensional structure. They are all accessible form the outside. There are not connections between the units, only with the inputs. As it was pointed out before, the learning procedure is unsupervised here. This means that the weights of the connections between the inputs and each unit in the lattice are set up in such a way that the network will associate specific regions of the map to similar vector inputs in some measure space. This is performed in the following way:

1. For each pattern the activations of all the units are evaluated.
2. The neuron which has the maximum value is selected.
3. The weights of the connection for the neurons in a neighborhood of the neuron selected are updated using the equation:

$$w_{ij}^{k+1} = w_{ij}^{k} + \eta\lambda(j, j^*)(x_i - w_{ij}^{k}) \tag{11}$$

where w_{ij}^k stands for the connection between the i input to the j unit at time step k and j^* is the winner neuron in the measure sense. The function $\lambda(j, j^*)$ is the neighborhood function [13].

2.4.4 Recurrent Neural Networks

This type of neural network is similar in structure to Hopfield networks since it allows connections between any pair of units but keeps the concept of input units and output units inherent to multilayer networks. Because of the possibility of having connections from each unit to any other unit independently of their positions, these networks have feedback within themselves. The concept of mapping or function approximation that is used in multilayer networks is therefore not applicable easily. Since we have dynamics, in order to talk of mapping it is necessary that the network reach stable states. The reason why scientists started to look at these highly nonlinear dynamic structures is because they were looking for powerful structures to have better performance in supervised learning than usual networks. The drawback is that they did not have a good algorithm to train the networks. Some years after the backpropagation algorithm was published they discovered a similar algorithm for recurrent neural networks. It was called *recurrent backpropagation*. This method maintains the concept of optimizing the error function given in the same way as in multilayer networks, i.e. by equation (5). Here, the outputs of the networks are the stable states reached by the structure with the specified input values. The weights are updated in the same way as in the backpropagation method but the delta values can not be computed in the same way. They are evaluated as a result of the stable state of another neural structure similar to the original but in the reverse order. This

means that the output units turn out to be the input units and the input values are the error of each output unit in the original network [6]. It was shown also that the previous backpropagation algorithm is a special case of this method. The approach of finding the update values using another network has been extended and incorporated in the backpropagation method since then. We describe recurrent neural networks more extensively in the next section.

As a summary, we have reviewed some basic concepts of neural networks, their history and some of their most known structures. In the next section we focus on the application of neural networks to system identification.

3 System Identification and Neural Networks

We focus on non-linear, black box system identification using recurrent neural networks. This type of neural network has proven to be better in system identification tasks than feedforward neural networks because they include internal dynamics. The name black box comes from the fact that we are modeling an input-output relationship in which we do not assume a priori any specific knowledge that can simplify our model, e.g. from first principles. When a system has a very complex dynamics and it is very difficult to apply rigorously any laws of physics, a black box approach is justifiable.

Many neural network architectures are possible for identification procedures [28]. Using the work of Nerrand et. al. [21] who showed that any feedback network can be represented by a canonical form using a static feedforward network, we focus our research on these three black box models [22]: the *output error* model, the *NARX* model (Nonlinear AutoRegressive with eXogenous inputs), and the *NARMAX* model (Nonlinear AutoRegressive Moving Average with eXogenous inputs). The models can be expressed analytically by the following equations:

Output error model:

$$\begin{aligned} x(k) &= f(X(k-1), U(k-1)), \\ y(k) &= x(k) + w(k) \end{aligned} \tag{12}$$

NARX model:

$$\begin{aligned} x(k) &= f(X(k-1), U(k-1)) + w(k), \\ y(k) &= x(k) \end{aligned} \tag{13}$$

NARMAX model:

$$\begin{aligned} x(k) &= f(X(k), U(k), W(k)) + w(k), \\ y(k) &= x(k) \end{aligned} \tag{14}$$

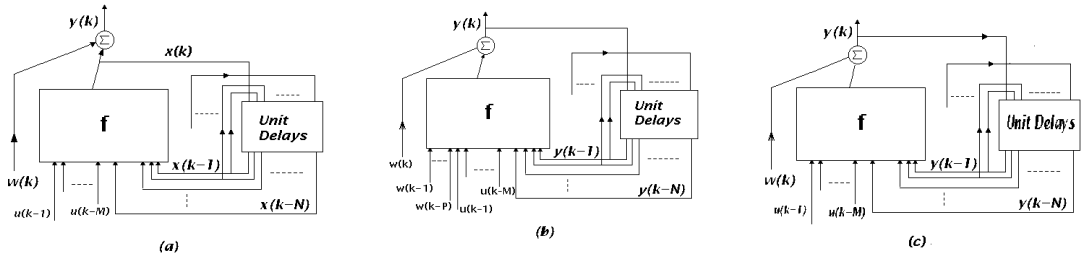

Figure 1: Diagrams for the Neural Network Models. (a) Output error model. (b) NARMAX model. (c) NARX model.

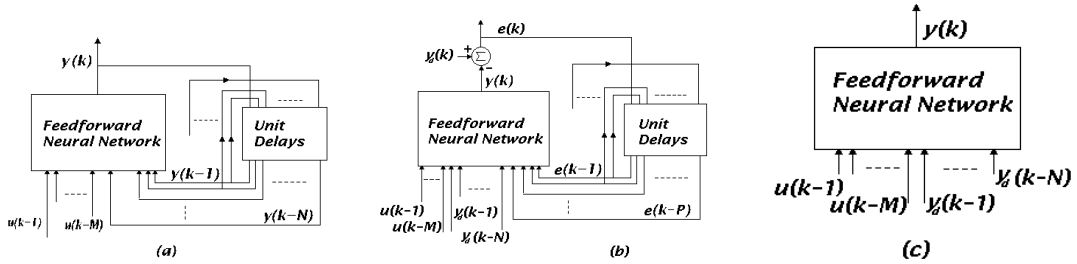

Figure 2: Neural Prediction Schemes. (a) Output error model. (b) NARMAX model. (c) NARX model.

with:

$$\begin{aligned} X(k-1) &= [x(k-1), x(k-2), \ldots, x(k-N)], \\ U(k-1) &= [u(k-1), u(k-2), \ldots, u(k-M)], \\ W(k-1) &= [w(k-1), w(k-2), \ldots, w(k-P)], \end{aligned}$$

where the vectors $X(k)$, $U(k)$ and $W(k)$ stand for the states and inputs of the neural network and noise, respectively, at the time step k, and f is a nonlinear function of its arguments. Diagrams for the equations above are shown in the figure 1.

Figure 2 shows the prediction scheme for each model type. For example, in figure 2.a the neural predictor for an output error model with canonical recurrent networks is shown. The feedforward part of the network is fed with the external inputs of the system in the M previous time steps and the N previous outputs of the whole structure. That part is represented for the loops around the time delay box in the figure. Figure 2.b shows the neural predictor scheme for a NARMAX model. In this case the feedback signal is the difference between the real system output and the output from the model. But this is not the only difference, the input to the feedforward network is composed of the previous inputs to the system, the already mentioned feedback error signal in P previous time steps and N previous values of the real system. In the last picture, representing the neural predictor for the $NARX$ system, feedback does not appear. This type of models is naturally represented by a

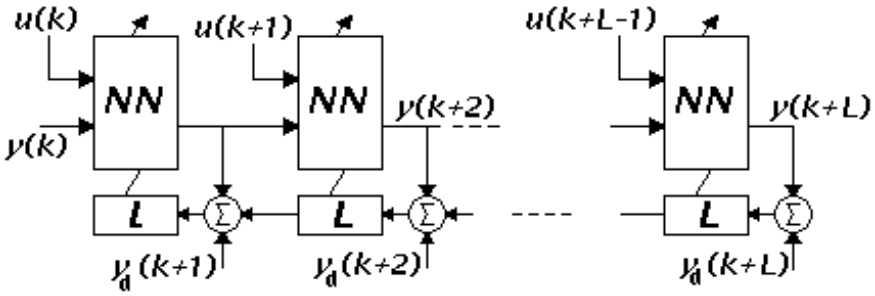

Figure 3: Window prediction and learning scheme.

feedforward network. The figures show that the information feedback only appears from the output to the input of the neural net as mentioned before. The fact that the $NARX$ model is a special case of the $NARMAX$ does not exclude a priori its use. The differences in the ways the models are determined open the possibility to have a better $NARX$ model than its corresponding $NARMAX$ model if the system behaves the way a $NARX$ model assumes.

Because of the different structures, each model has a different learning procedure. As we said earlier, learning means to find a set of weights for the interconnections between neurons and with the inputs and outputs that best matches the system input-output relationship. It is then a supervised learning process. Usually, that procedure develops in an algorithm where the weights are changing slowly through many repetitions of the same set of input-output patterns from the original system. Each set of weight updates through a single pass by all the patterns is often called an *epoch*. The algorithms are based on an optimization criteria.

Sometimes it is easier to represent the learning process through a diagram showing the way in which the neural network is used to predict the outputs for the model and the signal used to train the network. But, we have to specify the kind of algorithm used and the criteria utilized for each algorithm.

In the following subsections we explain the algorithms we worked with in this project: *backpropagation through time*, its variation using the BFGS optimization procedure and the Extended Kalman Filter approach. These methods are the most common methods used for these kinds of networks or some variation of them. However, when applied to system identification, the standard methods do not perform very well and we added some tools to enhance them.

3.1 Learning Using Backpropagation Through Time.

In this well known method [31] we minimize the sum of the squared errors in the output of the neural predictor over an arbitrary horizon, i.e. over a subsequent number of steps in the future. This sum is given by:

$$E(W) = \sum_{k=1}^{k=L} (y_d(k) - y(k))^2 \tag{15}$$

where $y_d(k)$ is the desired output in the time step k, $y(k)$ is the output of the neural network predictor and L is the number of steps in the future or prediction horizon used. The notation $E(W)$ states explicitly the dependency of the error function on the whole set of weights of the network. At one time step we evaluate the predictions of the neural network model over this horizon as if the weight connections do not change for the different steps within the horizon. The method allows us, using what is called order derivatives, to find the gradient of the error function with respect to the weights of the network in a backward manner.

During a learning epoch, the error function, given by equation (15), is evaluated for each subset of L predictions of future values taken from data of the real system. The gradient of the error function is found and the weights are updated according to the following formula:

$$\begin{aligned} \Delta W_{k+1} &= W_{k+1} - W_k \\ &= \alpha\ \Delta W_k + (1-\alpha)\ \eta\ \frac{\partial E}{\partial W} \end{aligned} \tag{16}$$

where α is the momentum coefficient and η is the learning coefficient. Because we are using a window of dimension L sliding over the system data, the algorithm is recursive. We apply one variation called the *semidirected method* by which, in order to calculate the error function and its gradient, it is assumed that at the beginning of the current window applied the real system and the neural networks are in the same initial condition. In other words, we assume that previous to the time when the current window begins the neural network was in the same state as the system. Figure 3 shows a scheme of the prediction horizon and weight updates for the output error model approach.

This is a typical gradient descent algorithm with all the problems that it implies. Thus, η has to be small enough to avoid oscillation and large enough to escape from bad local minima. The momentum term helps a lot in this respect. This method is also very sensitive to bad data from the real system. In a later subsection I am going to explain how we work out this problem.

3.2 Learning Using BFGS Optimization procedure.

In this method we chose the same error function as in the BPTT algorithm, equation (15), and we also use backpropagation of the error function through the different blocks to evaluate its gradient.

The BFGS method [17], developed independently in the '70s by Broyden, Fletcher, Goldfarb and Shanno, estimates the Hessian of the error function using the formula:

$$H_{k+1} = H_k + \left[1 + \frac{\gamma_k^T H_k \gamma_k}{\delta_k^T \gamma_k}\right] \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{\delta_k \gamma_k^T H_k + H_k \gamma_k \delta_k^T}{\delta_k^T \gamma_k} \tag{17}$$

where, H_k is the Hessian approximation at the step k and,

$$\delta_k = W_{k+1} - W_k \tag{18}$$

$$\gamma_k = \nabla E(W_{k+1}) - \nabla E(W_k) \tag{19}$$

In this method the direction of the step we take each time is found with the expression:

$$d_k = -H_k \, \nabla E(W_k) \tag{20}$$

and the next point W_{k+1} in the weight space is found solving a one dimensional minimization problem:

$$\begin{array}{ll} Min & E(W_k + \theta d_k) \\ \theta \geq 0 & \end{array} \tag{21}$$

One can also use the economical approach of Goldstein or the one by Wolfe and Powell. Both require far less evaluations of the function than doing a complete minimization in one dimension. For example, if we define the unidimensional function g as:

$$g(\theta) = E(W_k + \theta d_k), \tag{22}$$

Goldstein's rule selects the next point W_{k+1} by choosing θ such that

$$\begin{aligned} g(\theta) &\leq g(0) + m_1 \theta g'(0) \\ g(\theta) &\geq g(0) + m_2 \theta g'(0) \\ m_1 &\in [\, 0,\ 1\,] \\ m_2 &\in (\, m_1,\ 1\,) \end{aligned}$$

The BFGS method as well as other similar algorithms show better convergence rates than steepest descent methods like the one presented in the previous section but at the expense of memory requirements and time consumption per step of the order of N^2, where N is the dimension of the weight space in which the minimization process takes place. It also requires the approximation of the Hessian to be reset at the order of N iterations in order to assure convergence.

3.3 Learning Using Kalman Filtering Approach

In the 60's Kalman and Bucy developed a recursive algorithm to predict the behavior of a linear system corrupted by Gaussian noise. It is called from that time the Kalman-Bucy Filter. Later, it

was extended to non-linear systems by taking the gradient and applying the Kalman-Bucy Linear Filter to the linearized version of the non-linear system. This was called the Extended Kalman Filter (EKF) [14]. There is a way to use this framework in the neural network field. It is based on the EKF method. The use of EKF capabilities to train neural networks have already been explored for example in [23] [32] and [1].

The key to applying EKF to neural networks is based on the fact the we can express the dynamics of a recurrent neural network with the structure we were analyzing by the following equations:

$$y_{k+1} = f(y_k, W_k, u_k) \tag{23}$$

$$W_{k+1} = W_k \tag{24}$$

$$z_k = h_k(y_k, W_k) \tag{25}$$

where the function f is mapping the inputs of the neural network to its future activations parameterized in the weights W_k, y_k is the vector of activations values for all the units in the network and z_k is the output of the neural network. If we choose as a representation of the system the vector

$$x = \begin{bmatrix} y \\ W \end{bmatrix} \tag{26}$$

we have a formulation in state representation form of the non-linear dynamics of the neural network and we can apply the EKF method. For that, we are going to need the gradient of the function f. From the previous equations it immediately follows that :

$$\frac{\partial x_{k+1}}{\partial x_k} = \begin{bmatrix} G_1 & G_2 \\ 0 & I \end{bmatrix} \tag{27}$$

where G_1 and G_2 contain the partial derivatives of the nonlinear function f with respect to the nodes and weights respectively. To calculate G_1 and G_2 we follow a similar strategy as the one applied in the two previous neural network methods.

Now, using the usual notation in Kalman Filtering approach we have,

$$F(k-1) = \left.\frac{\partial f}{\partial x}\right|_{x=\hat{x}(k-1|k-1)} \tag{28}$$

$$\hat{x}(k|k-1) = f(\hat{x}(k-1|k-1), u_{k-1}) \tag{29}$$

$$P(k|k-1) = F(k-1)P(k-1|k-1)F(k-1)^T + Q(k-1) \tag{30}$$

for the predictor, where $\hat{x}(k|k-1)$ stands for the prediction value of the state of the network given the value at the previous time step, $P(k|k-1)$ is the error correlation matrix and Q(k-1) is the covariance of the noise in the dynamic equation.

For the corrector filtered values we have the following equations:

$$H(k) = \left.\frac{\partial h_k}{\partial x}\right|_{x=\hat{x}(k|k-1)} \tag{31}$$

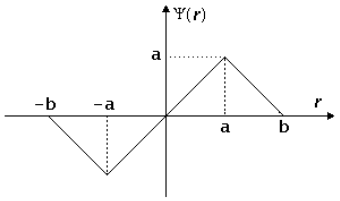

Figure 4: Hampel function

$$K(k) = P(k|k-1)H(k)^T \left[H(k)P(k|k-1)H(k)^T + R(k)\right]^{-1} \quad (32)$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k)(z_k - h_k(\hat{x}(k|k-1))) \quad (33)$$

$$P(k|k) = P(k|k-1) - K(k)H(k)P(k|k-1) \quad (34)$$

where $K(k)$ is the Kalman gain matrix, R(k) is the covariance of the additive error in the output of the system and $\hat{x}(k|k)$ is now the correction value, i.e. the corrected value of the states of the network once we have measured the outputs at the same time step.

This approach is well suited for real time implementations because we do not require a horizon as in the previous cases and adaptation can be performed on-line without any special processing. We pay in memory requirements and also in cpu time per adaptation step which in this case is proportional to N^4 in both cases, N being the number of units in the network [32]. It has the advantage over the previous methods presented in that it is introducing disturbances in the form of Gaussian noise to the neural network, making the algorithm more robust to noise. It is no good however, when the noise has a distribution very different to a Gaussian distribution. An example is the existence of outliers. As a disadvantage with respect to the other methods is the linearization procedure. It is very important to have a good starting point to assure a good behavior. In the following we explain a modification to improve the performance of the modeling technique to such cases.

3.4 Improving Robustness

To the standard procedure mentioned above we added a variation to make the estimation robust with respect to outliers in the original data and non-Gaussian observation noise. Starting from the original works of Masreliez [16] and Martin [15] and the more recent paper by Connor [1] who developed this technique for a time series prediction we modified the algorithm to extend the approach to multivariable systems with external inputs.

In his work, Masreliez established an algorithm of the same type as the famous Kalman-Bucy filtering to estimate the states of the linear system when the observation noise is not Gaussian.

Connor et al. took that work and in order to use the results in non-linear systems they applied the EKF algorithm using the following system:

$$x_t = f(x_{t-1}) + \epsilon_t \quad (35)$$

$$y_t = h^T x_t + v_t \quad (36)$$

where ϵ_t is Gaussian and v_t represents the outliers. The output y_t is also a scalar magnitude. The robust one step prediction and filter estimates are given by,

$$\hat{x}_t^{t-1} = f(\hat{x}_{t-1}) \quad (37)$$

$$\hat{x}_t = f(\hat{x}_{t-1}) + \frac{M_t h s_t}{s_t^2}\psi(\frac{y_t - \hat{y}_t^{t-1}}{s_t}) \quad (38)$$

where ψ is the Hampel function shown in figure 4 and M_t is the prediction covariance evaluated as in the EKF method,

$$M_{t+1} = \nabla f(\hat{x}_t) P_t \nabla f^T(\hat{x}_t) + Q \quad (39)$$

$$P_t = M_t - M_t h w(\frac{y_t - \hat{y}_t^{t-1}}{s_t}) \quad (40)$$

where $w(r) = \psi(r)/r$ and $s_t^2 = (M_t)_{11}$. They also showed that minimizing the squared error function with inputs filtered of the outliers is equivalent to solving the Maximum Likelihood estimate for the weights of the network.

The algorithm filters the feedback inputs based on an estimate of the error covariance, and trains the network using a modification of EKF with this filtered data.

We extended the algorithms to cover a more general system identification problem, a multivariable system with external inputs. The introduction of the external inputs does not cause a problem if we assume no initial statistical correlation between the inputs and the noise and states of the system, which is reasonable in almost all cases.

In the univariable case they constrained the filtering adaptation to be in the region free of outliers by modifying the residual by the Hampel function. In the multivariable case we have to analyze the residuals in the different directions. Then, we analyze the errors in the singular value decomposition of the error covariance matrix $S(k)$. In order to normalize the error we split the covariance matrix. We know that $S^{\frac{1}{2}}$ exists because it is at least positive semidefinite. The predictor equations are not modified with respect to the standard EKF approach but the corrector ones are modified in this way:

$$K(k) = P(k|k-1)H(k)^T \left[H(k)P(k|k-1)H(k)^T + R(k)\right]^{-1} \quad (41)$$

$$S(k) = \left(H(k)P(k|k-1)H(k)^T + R(k)\right)^{-1} \quad (42)$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) + K(k)S(k)^{-1/2}\psi\left(S(k)^{1/2}(z_k - h_k(\hat{x}(k|k-1)))\right) \quad (43)$$

$$P(k|k) = P(k|k-1) + P(k|k-1)H^T(k)\Phi(k)H(k)P(k|k-1) \quad (44)$$

where $\psi(x)$ is the Hampel function shown in Figure 4 and $Phi(k)$ is given by the following:

$$\Phi(k) = \frac{\partial \left(S(k)^{-1/2}\psi\left(S(k)^{1/2}(z_k - h_k(\hat{x}(k|k-1)))\right)\right)}{\partial z_k} \tag{45}$$

The idea behind the Hampel function is the same as the Maximum Likelihood estimation procedures from which it comes. When the data is in the current outlier region, set up by the current error covariance estimate and the choice of the parameters a and b, the corrector value stays in the prediction one. For values in the "safety" region the modification algorithm reduces to the usual EKF and there is a transition region that weights the data on the basis of its distance in the error subspace.

As mentioned above, the procedure also filters the feedback inputs. The filter comes from an estimation of the prediction residuals r_k,

$$r_k = z_k - h_k(x(k|k-1)) \tag{46}$$

where care must be taken in using the raw data, i.e. without filtering, in the output prediction. The prediction residuals usually used is the MADM, (median absolute deviation about the mean) $\hat{\sigma}_\epsilon$ given by,

$$\hat{\sigma}_\epsilon = 1.483 < |r_k - < r_k > | > \tag{47}$$

where the $< . >$ stands for the mean value of the expression inside. We estimate the mean value in the time domain for every epoch.

To start this procedure we need a "good" first point and a reasonable estimate for the error. Then it is advisable to start first with one of the previous methods and train the network for a sufficient number of epochs and then use this procedure.

After that we run the algorithm for an arbitrary number of epochs or until convergence and update the estimation for the error. With the new estimation, the data is filtered again and we repeat the process.

This modification of the procedure makes it a little more difficult to use in on line adaptation. In that case we should implement the estimation of the residuals recursively at each step.

3.5 Preprocessing of the inputs

In all three cases described above some processing of the inputs was made before feeding the neural network. This filtering improves the learning and prevents overtraining too. In the case of the Extended Kalman Filter approach, the filter comes with the procedure to have a maximum likelihood estimation of the weights of the network as was explained in section 3.4. In the other two cases something similar can be applied. If we know the existence of outliers, the MADM residual estimation

gives a good value to keep them away. For both cases this estimation and corresponding filter was applied. Some practical considerations can be stated here too. An appropriate scaled of the experimental data is advisable to prevent saturation or getting stuck in bad local minima. This is done by defining minimum and maximum values for the different inputs. These values come naturally from the restrictions of the problem. A filter was also added between the output and the feedback inputs. This provides more robustness to bad data and overtraining. The filter was designed using an estimation of the mean square error for each horizon. We use the same estimation as the one shown in the Kalman Filter algorithm.

4 Experimental Procedure

This section describes the application of the method developed in the previous section to a real system, a plasma enhanced chemical vapor deposition (PECVD) system currently located in the clean room of Hammershlag Hall. First, we give a brief description of the experimental equipment in order to have the minimal background to develop the system identification ideas used in the research. A more detailed description of the experimental apparatus and preliminary control results are described in a previous thesis [30], giving the first steps of the whole project and, in its more recent configuration, in [11] and [12]. After that, the steps taken to have useful data from the system and application of the modeling procedure are explained.

4.1 PECVD system. Physical Concepts

Plasma deposition is a process in which a chemical layer is deposit over a substrate by a highly reactive ionized gas. In our case we deposit silicon nitride layers over a silicon substrate. The gases are usually ionized using a microwave power supply in order to keep the reacting surfaces at lower temperatures. The whole process takes place in a highly non-equilibrium thermodynamic state. This is one of the reasons why it is so difficult to model. Besides nonlinearities, we have to deal with significant drift too. This is believed to be due to changes in composition and temperatures of the different surfaces within the process chamber.

From the point of view of the engineering process, we would like to control the deposition rate, film density and uniformity, the silicon/nitrogen ratio and of course the electrical properties of the film. For these purposes, reacting gas flows, operating pressure, RF power and temperature are used as manipulated variables. Plasma processes, once started, are stable in nature, at least during the period of time in which deposition occurs. However, some different problems arise: the start, the operating region and the regulation of the process. In general it is very difficult to establish an effective control because the most important variables to take care of cannot be measured until the process is done and the film has been made. New measurement devices are being developed to gather more information during the process in the hope that these new measurements will have a stronger correlation with the ultimate performance of the semiconductor product.

4.2 Experimental Setup

The PECVD reactor is composed of the main chamber having a pair of parallel, circular conducting electrodes, custom made. The one where the wafers are located is heated. The other one is used to introduce the reactant gases into the chamber. The reactant gases are SiH_4 and NH_3 with N_2 as a diluent gas to control the reaction. There is also an additional exhaust valve to control the overall

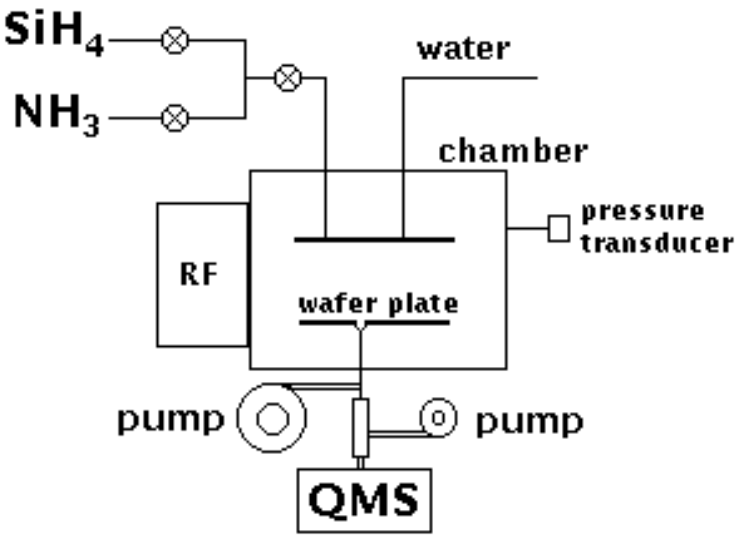

Figure 5: PECVD system layout

chamber pressure.

The in situ sensor is a differentially pumped quadrupole mass spectrometer (QMS) that essentially is a selective counter of molecules, i.e. it allows us to measure particle flows with a specific, selective, mass-charge ratio. It was introduced in the system through a sampling orifice in the wafer plate that leads to a secondary chamber and the QMS system. We are then taking a sample from the flow coming to the wafer and, using the QMS, we can discriminate what kind of particles are incident on the wafer and in what amounts. It also can be translated to the partial pressure of the different species in the main chamber using statistical considerations.

The whole set of valves, pumps, heaters and RF power are computer controlled using a Gateway 486/66 MHz running LabWindows/CVI. Figure 5 is a simplified schematic of the PECVD system showing the parts of the installation relevant to this project.

As a final comment we have to point out that the physical system is continously being improved, e.g. with the addition of an electron multiplier to have a better signal to noise ratio in the measurements from the QMS, and the results showed in this report reflects the configuration for the PECVD system at the time the experiments were performed.

4.3 Problem Definition and Scope

Our objective is to apply our method to perform a multi-input multi-output black-box, non-linear system identification of the PECVD system developed at Carnegie Mellon. Considering the quality

Silane	Ammonia	RF Power	Temperature	Chamber pressure
(sccm)	(sccm)	(W)	(C)	(Torr.)
3.0	60	20	250	0.400

Table 1: PECVD Operating point chosen.

of the measurements obtained for the different variables we took a 3x3 system composed of:

Inputs:	RF Power	Silane flow, SiH_4	Ammonia flow, NH_3
Outputs:	DC Bias	$pp(SiH_2)$	$pp(NH_2)$

The DC bias voltage is directly related to the deposition mechanism of the plasma, and is attractive for control purposes in that it tends to have a fast ($\ll 1$ s) response to input changes. The other outputs are QMS measurements, partial pressures corresponding to species with masses 16 (NH_2) and 30 (SiH_2). Mass 16 is the radical NH_2^+, which is indicative of the amount of ammonia which has been consumed in the reaction with silane or silane derivatives. Mass 30 is the radical SiH_2^+ and is indicative of the amount of silane remaining in the chamber. Sometimes we are going to mention a species by its mass number.

We will describe the experiments carried out to have enough data to perform a system identification procedure. We are going to investigate which model of the ones given by the equations (12)-(14) is best for this application and also the differences between the learning methods used.

We are also going to try to address the problem of the drift in the measurements and to be able to find a way of predicting it.

4.4 Design of Experiments

Since the project started two years ago, data have been collected and stored giving us experience about how these kinds of processes behave. All of this gave us a framework in which to introduce neural networks.

A safety region where the desired operating point should lie was defined. The silane flow was limited to the region 0.5–5.5 SCCM (Standard Cubic Centimeters per Minute), the lower bound due to slow deposition rate and the upper bound because one of the goals of PECVD is to deposit a nitrogen rich film and higher flows of silane would produce a nitrogen poor film. The ammonia flow was limited to the region 20–72 SCCM. The lower bound of 20 was used, based on initial experimentation showing that the pressure controller had significant trouble keeping a constant pressure at lower flows. This is not a real problem because we want to produce nitrogen rich films. The upper bound of 72 was chosen because the ammonia mass flow controller has a maximum setting of 73 SCCM, and if it is pegged at 73, flow control would not be possible. A lower bound of 5 W was set for the RF power, due to resolution of the RF matching network, tuning and plasma extinction.

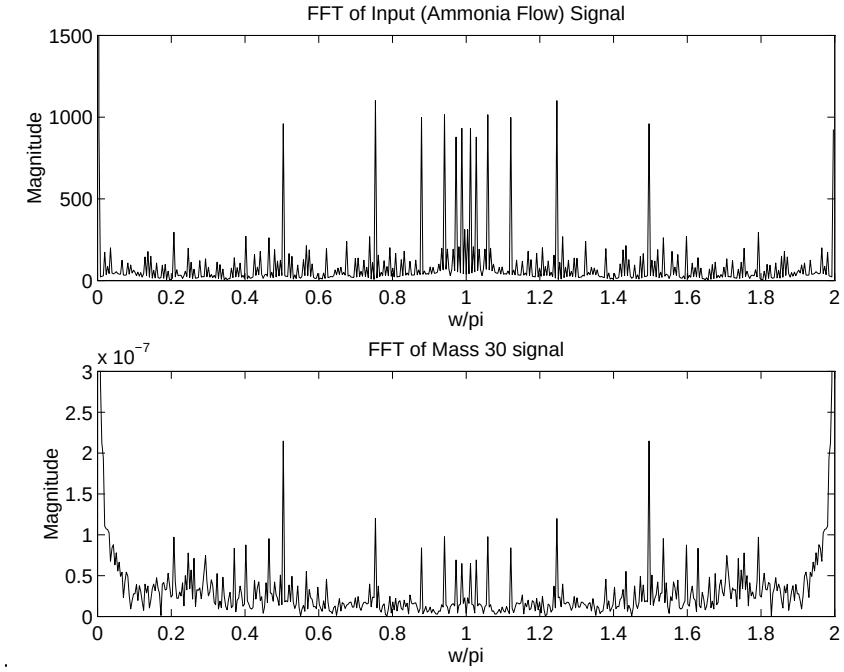

Figure 6: Input/Output signals (FFT) used in determining the frequency response of the system.

We need a minimum amount of energy to maintain the plasma in the chamber. The limit for starting the plasma is different – larger – though. An upper bound of 35 W was set for RF power, because powdering becomes a problem at higher powers.

Using the work of Smith [29] as a starting point, factorial experiments were run to determine the best values for the steady state operation of the system. Analysis of the resulting films led to the choice of the setpoint values given in Table 1.

The data we used from the system were step responses and control experiments. Through FFT analysis we were able to characterize a frequency response for the system in a first approximation. Then, we created an input signal with high frequency content in the regions of interest. To determine the basic form of the frequency responses for the different inputs, a waveform was generated whose energy spectrum was non-zero at points logarithmically spaced between discrete frequencies $\theta = 0$ and $\theta = \pi$. Since we expected the response to have a cutoff frequency on the order of 1 Hz or less we used a sample time smaller than 1 second. Using this waveform, Single Input Multiple Output (SIMO) experiments were run, in which a scaled version of the (discretized) waveform was applied to one control input, while the other inputs were held at their nominal operating points. The SIMO silane experiment consisted of flows ranging from 2.5–3.5 SCCM, the SIMO ammonia experiment consisted of flows ranging from 50–70 SCCM and the SIMO RF power experiment consisted of powers ranging from 15–25 W. The magnitude of the frequency spectra of a typical input and output are

shown in Figure 6. The data from these SIMO experiments were used to estimate the frequency response at those frequencies contained in the input signal. From the data shown in Figure 6 and similar for the other inputs and outputs we observe some frequency ranges where the response was important. A low frequency and high frequency regime can be selected. We selected the frequency of $\frac{\pi}{8}$ as a cutoff between the high frequency regime (hereafter referred to as the fast regime), and the low frequency regime. We focused on the fast regime which will be the primary regime for real-time multivariable control action. The phase response did not add more information to select the cutoff frequency.

A set of experiments was run in which this waveform was applied to a single input while the others were held constant. The amplitudes of the waveform were evenly spaced over the operating region, and the order of the experiments was randomized. Three complete sets of experiments were used for identification purposes, and a fourth complete set was used for validation. The duration of the waveform was 10 minutes, which is approximately the time it takes to complete a deposition process. The preparation of the system for the run takes a while longer. In this case the system was warmed up and etched for one hour before actually begin the experiment. This is required to have a standard initialization process and have reliable measures from the QMS. The data for each data set consisted of roughly 600 data points since we took measurements at a rate of 1 Hz.

As final comments we would like to say that the limit of 1 Hz in the sampling, imposed by the QMS status and communication protocol, was a real problem for a fast response study and would have to be fixed in future work. I was involved in the design of the experiments and signal generation while the operating point and safety regions were evaluated by other members of the research group.

4.5 Methodology

The steps followed to identify the PECVD system were:

- From the training data given by the experiments described in section 4.4 train the neural networks of the types chosen to predict the behavior of the system in the next time step.
- Optimize the characteristics of the network model, e.g. the number and distribution of inputs and hidden nodes and the rest of the parameters defining each learning procedure to get an acceptable performance.
- To prevent over-training, observe the performance in the validation sets of data during training, stopping the process when the performance begins to degrade.
- Due to the usual local minima problem we repeat the experiments several times from different starting conditions for the weights.

To perform the computations we made use of the Stuttgart Neural Network Simulator, SNNS, version 3.3, and the commercial package *Matlab* with its own routines and also with some routines of our own to meet our needs. We implemented in Matlab the routines for the three methods. The NARX model was trained and tested using the SNNS software. The other two models were training with custom made *Matlab* routines. There are several reasons for this choice. One is that the SNNS package is more oriented to pattern recognition tasks and its capabilities are not well oriented to recurrent nets and time prediction problems of the type we deal with in control engineering. *Matlab* has a well oriented interface for the user and for control (*Simulink*) that allows easier development and future use. As a drawback *Matlab* is several times slower than SNNS, but this is not considered a problem at this stage of the research.

The structure of the feedforward part of the neural networks was that of a 3-layer neural network, having the hidden units a *tanh* activation function and linear input and output units.

The exogenous inputs were composed of M previous time inputs to the system, N previous output measurements (in the case of the $NARMAX$ model) and the time from the beginning of the experiment. The feedback inputs are N previous output values for the *output error* model or P previous errors for the $NARMAX$ model.

To optimize the number of hidden neurons a possibility suggested is the use of Genetic Algorithms [3], i.e. having a pool of nets with different values for the number of neuron and/or the desired characteristics to optimize and combining them, taking into account the ones that have a better fit with the experimental data, to obtain new candidate networks. This method in general is computationally expensive although is good to avoid local minima problems. We prefer to initiate the model with a large number of units and after a specified number of iterations of the learning algorithm eliminate those hidden units with small correlation with the output of the network and train again.

Prop.	Output Error	NARX	NARMAX
no. exogenous Inputs	10	10	19
no. feedback Inputs	9	9	9
no. Hidden units	22	29	25
no. Output units	3	3	3
Total epochs	30000	30000	30000
Learning coef.(η)	.0001	.0001	.0001
Momentum term (α)	0.9	0.9	0.9
MSE training set			
DCbias	0.84	0.95	0.52
Mass 16 ($\times 10^{19}$)	1.39	1.25	0.66
Mass 30 ($\times 10^{21}$)	3.01	3.57	1.52
MSE validation set			
DCbias	2.33	3.47	1.56
Mass 16 ($\times 10^{19}$)	6.03	6.42	3.85
Mass 30 ($\times 10^{21}$)	8.43	7.18	5.12

Table 2: Neural Network Model using BPTT learning.

5 Results

In this section we present the results of the neural modeling and compare them to linear models for the same system.

5.1 Neural Network Models

Following the steps expressed in the previous section we obtain models for all the structures and learning algorithms. Tables 2 3 and 4 show the results obtained for each type of network and each learning algorithm. The upper part of each table lists the parameters for each class of neural network and learning algorithm. In table 2 we list the significant parameters in the case where we are using back-propagation through time as the learning algorithm. In the first column we display the optimum values found when we use the *output error* model. The exogenous inputs are composed of the three previous values of the three inputs to the system and the time from the beginning of the deposition. The feedback inputs correspond to the three previous outputs of the neural network. For the *NARMAX* model the exogenous inputs are composed of the time from the beginning of the deposition, the three previous values of the inputs and the three previous measured output values. In table 3 we list the values of the number of exogenous inputs, the feedback inputs to the network, the number of hidden units found in the procedure, the output units, the total number of epochs trained and the values of the parameters m_1 and m_2 of the Goldstein's rule used. The lower part of the tables show the mean squared errors (MSE) for each output, for the training set and also for a validation set of data.

Prop.	Output Error	NARX	NARMAX
no. exogenous Inputs	10	10	16
no. feedback Inputs	6	6	6
no. Hidden units	20	25	20
no. Output units	3	3	3
Total epochs	5000	5000	5000
m_1	0.2	0.2	0.2
m_2	0.7	0.7	0.7
MSE training set			
DCbias	0.74	0.93	0.41
Mass 16 ($\times 10^{19}$)	1.12	1.28	0.55
Mass 30 ($\times 10^{21}$)	2.37	2.45	1.28
MSE validation set			
DCbias	2.29	3.25	1.42
Mass 16 ($\times 10^{19}$)	5.84	6.59	3.73
Mass 30 ($\times 10^{21}$)	6.22	5.31	4.11

Table 3: Neural Network Models using BFGS optimization procedure.

Prop.	Output Error	NARX	NARMAX
no. exogenous Inputs	10	10	19
no. feedback Inputs	9	9	9
no. Hidden units	22	23	19
no. Output units	3	3	3
a(Hampel)	1.5	1.5	1.5
b(Hampel)	3.0	3.0	3.0
MSE training set			
DCbias	0.70	1.05	0.33
Mass 16 ($\times 10^{19}$)	1.05	1.25	0.39
Mass 30 ($\times 10^{21}$)	2.12	2.51	1.03
MSE validation set			
DCbias	2.20	3.49	1.32
Mass 16 ($\times 10^{19}$)	5.61	6.55	3.56
Mass 30 ($\times 10^{21}$)	6.06	5.45	3.91

Table 4: Neural Network Models using Kalman Filter Approach

Learning	Input(Step)	Output Error	NARX	NARMAX
BPTT	RF	0.90	1.04	0.60
	NH_3	1.21	2.61	1.09
	SiH_4	2.82	5.13	1.24
BFGS	RF	0.83	1.06	0.62
	NH_3	1.15	2.51	0.98
	SiH_4	2.51	5.13	1.18
EKF	RF	0.81	1.06	0.55
	NH_3	1.03	2.45	0.85
	SiH_4	2.27	5.01	1.09

Table 5: Mean Square Error for DC Bias Voltage.

Learning	Input(Step)	Output Error	NARX	NARMAX
BPTT	RF	1.65	2.91	1.30
	NH_3	1.90	3.21	1.64
	SiH_4	2.33	4.02	1.85
BFGS	RF	1.62	2.72	1.25
	NH_3	1.82	3.20	1.53
	SiH_4	2.33	3.80	1.64
EKF	RF	1.54	2.72	1.24
	NH_3	1.55	3.15	1.33
	SiH_4	2.12	3.54	1.42

Table 6: Mean Square Error for Mass 16 ($\times 10^{19}$)

Learning	Input(Step)	Output Error	NARX	NARMAX
BPTT	RF	8.45	9.25	7.92
	NH_3	8.12	8.11	7.84
	SiH_4	8.87	8.62	8.29
BFGS	RF	8.23	9.08	7.89
	NH_3	7.94	7.90	7.75
	SiH_4	8.56	8.42	8.21
EKF	RF	8.15	9.12	7.78
	NH_3	7.94	7.86	7.61
	SiH_4	8.42	8.45	8.13

Table 7: Mean Square Error for Mass 30 ($\times 10^{21}$)

There was also available experimental data of the system from step change experiments. This data is useful in other activities of the project as in equipment testing, operation search and linear system identification and control. Tables 5, 6 and 7 show the performance of the three models for step changes in each input for DCBias, mass-16 and mass-30 respectively. In table 5 the first three rows display the mean squared errors in the DC Bias of the three models in step responses to RF power, NH_3 and SiH_4 separately. The other two tables list the same information for the other two outputs.

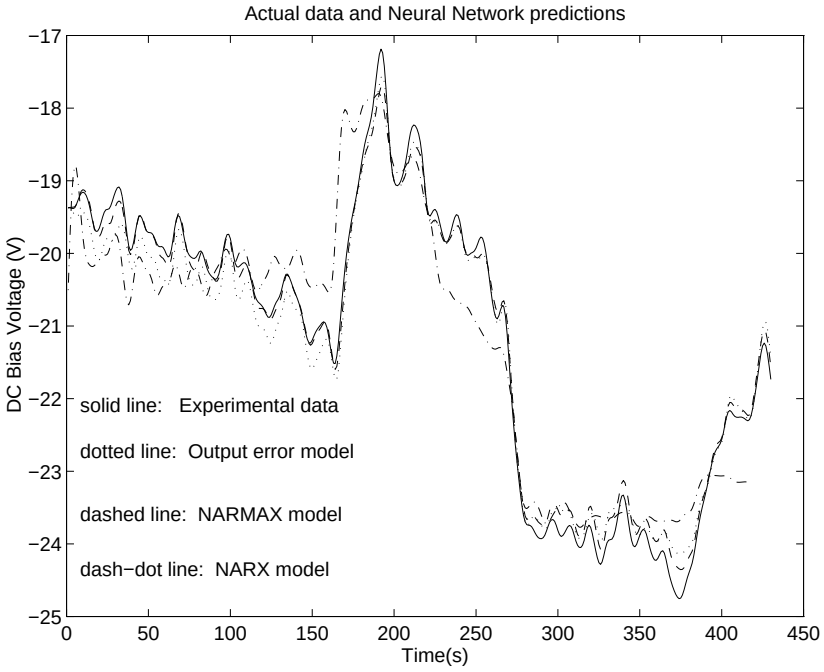

Figure 7: Comparison of Neural Network predictions of DC Bias for a step response.

The step responses are more suitable to display graphs as the ones shown in Figures 7,8 and 9. These graphs show the response of the three models to step changes in the inputs and the error.

Looking into the tables 2 3 and 4 we observe that the mean squared error in both, the training data and the validation data, for the column corresponding to the *NARMAX* model is the better than the MSE corresponding to the other ones. This is verified for all the outputs and all the learning algorithm used. That results enforce the idea that the *NARMAX* model is the right choice of the three to model the PECVD process. If we look in the outputs separately we observe a better fit for the DCBias, follow by *mass-16* and *mass-30* in this order. This is directly related to the exactitude of the measurements and it is a first estimation of the noise presents in them. We observe also, by comparing training and validation results, a larger degradation of the performance for the QMS measurements which is directly related to the drift problem in the system. In Figure 7 the effects of the drift can be seen clearly. Even without any variation of an input signal the outputs vary consistently. One of the reasons for that behavior is the deposition of material all over the chamber. Since this is a phenomenon strongly related with the time from the beginning of deposition we tried to use this variable as another input to the neural model. The introduction of time from the beginning of the experiment as an external input to the neural network was very useful in order to compensate for the drift mentioned above.

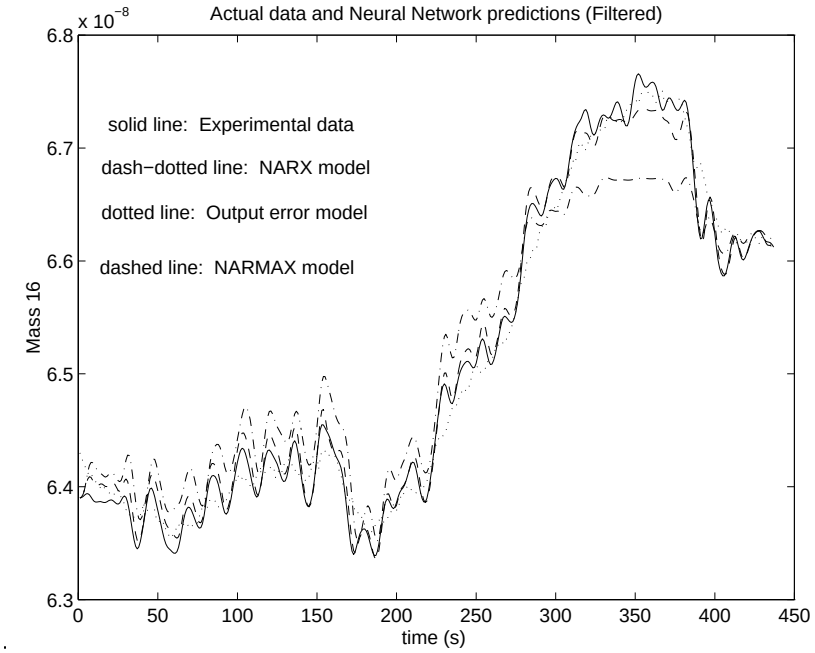

Figure 8: Comparison of Neural Network predictions of Mass 16 flow for a step response.

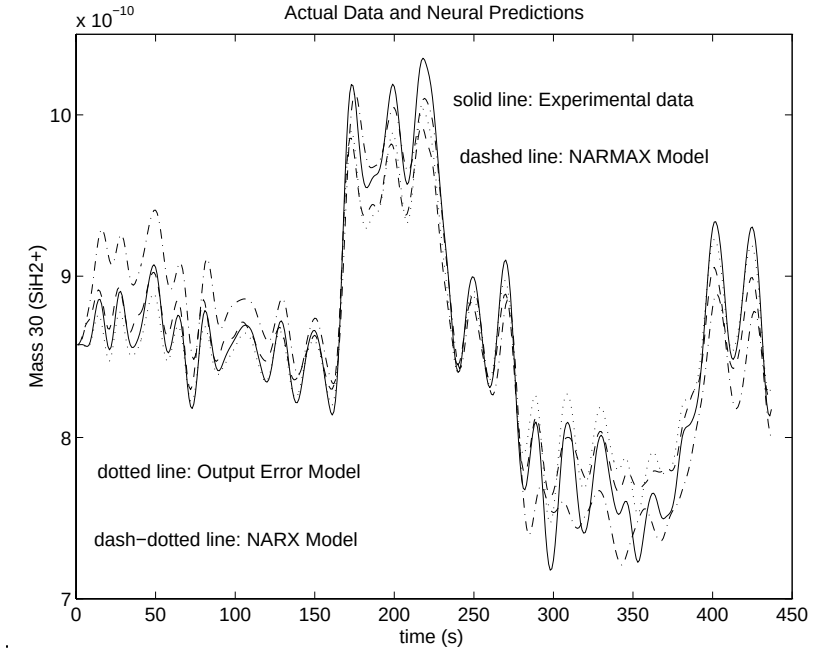

Figure 9: Comparison of Neural Network predictions of Mass 30 flow for a step response.

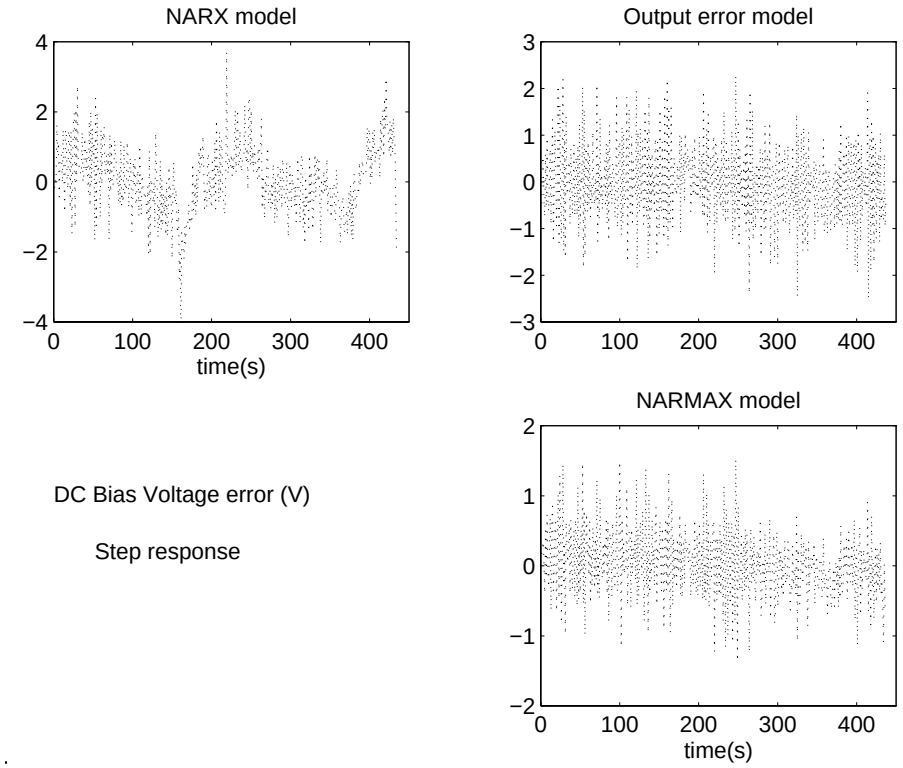

Figure 10: Error in the DC bias voltage for a step input in RF power.

5.2 Other types of models

It would be very useful to have a way to compare the results we obtained with results from other system identification procedures. For the development of the PECVD project it was necessary to have a standard model to use in Model Predictive Control algorithms. Therefore, linear, first and higher order models were developed for the PECVD system by other members of the group, based on the same data we used for the neural networks models.

Here we briefly describe these other models and present the results of these identification methods.

5.2.1 Description of the Linear Models

The models are Multiple Input Multiple Output (MIMO) transfer function matrices. Two classes were considered:

- First Order ARX (AutoRegressive with eXternal input) [14] models with time delay. These models are computationally simple and are widely used in industry. These are also the models upon which initial control designs have been based [12].
- The second class of models developed were Box-Jenkins ARMAX (AutoRegressive Moving Average with eXternal input) models [14] of the form:

$$y_i(t) = \sum_{j=1}^{3} \frac{B_{ij}(q)}{F_{ij}(q)} u_j(t) + \frac{C_i(q)}{D_i(q)} e_i(t), \tag{48}$$

  where the inputs u_j, are the inputs to the system and y_i the outputs. This model structure was chosen because it allows characterization of the error independent of the inputs to the system, and because for suitable orders of B, C, D and F it reduces to most other commonly used model structures, including the ARX structure.

5.2.2 Results

The transfer function for the models found are:

First Order Model (FO):

$$\begin{bmatrix} DCBias(t) \\ Mass16(t) \\ Mass30(t) \end{bmatrix} = \begin{bmatrix} \frac{0.0203}{1-0.675q^{-1}} & \frac{0.734q^{-2}}{1-0.805q^{-1}} & \frac{-0.448}{1-0.785q^{-1}} \\ \frac{0.504}{1-0.750q^{-1}} & \frac{-11.3q^{-2}}{1-0.467q^{-1}} & \frac{-1.12}{1-0.692q^{-1}} \\ \frac{-0.0339q^{-5}}{1+0.375q^{-1}} & \frac{36.8q^{-1}}{1-0.233q^{-1}} & \frac{-0.197}{1+0.346q^{-1}} \end{bmatrix} \begin{bmatrix} NH_3(t) \\ SiH_4(t) \\ RFPower(t) \end{bmatrix}$$

Higher Order Model (HO):

$$\begin{aligned} y_{DCBias}(t) &= \frac{0.0117 + 0.0660q^{-1}}{1 - 0.482q^{-1} + 0.544q^{-2}} u_{NH_3}(t) + \frac{0 + 1.38q^{-1} - 1.15q^{-2}}{1 - 1.35q^{-1} + 0.434q^{-2} - 0.0729q^{-3}} u_{Silane}(t) \\ &+ \frac{-1.82 + 0.140q^{-1}}{1 - 0.225q^{-1}} u_{RF}(t) + \frac{1 + 0.0233q^{-1} + 0.206q^{-2} - 0.624q^{-3}}{1 - 0.272q^{-1} - 0.0361q^{-2} - 0.692q^{-3}} e_{DCBias}(t) \end{aligned}$$

$$\begin{aligned} y_{Mass30}(t) &= \frac{-0.128}{1} u_{NH_3}(t) + \frac{11.7 + 17.3q^{-1}}{1 - 0.158q^{-1}} u_{Silane}(t) \\ &+ \frac{-1.84}{1} u_{RF}(t) + \frac{1 - 0.212q^{-1} - 0.512q^{-2} - 0.120q^{-3}}{1 - 0.210q^{-1} - 0.7811q^{-2}} e_{Mass30}(t) \end{aligned}$$

$$\begin{aligned} y_{Mass16}(t) &= \frac{0.220 + 1.85q^{-1} + 0.202q^{-2}}{1 - 0.291q^{-1} + 0.304q^{-2}} u_{NH_3}(t) + \frac{0 - 2.74q^{-1} - 2.42q^{-2}}{1 - 0.822q^{-1} + 0.155q^{-2}} u_{Silane}(t) \\ &+ \frac{-0.368 - 1.61q^{-1}}{1 - 0.494q^{-1}} u_{RF}(t) + \frac{1 - 0.411q^{-1} - 0.440q^{-2}}{1 - 0.996q^{-1}} e_{Mass16}(t) \end{aligned}$$

Figure 11 shows the response of the Linear models in comparison to the real data and neural network model predictions.

Tables 8 and 9 show the mean square errors (MSE) obtained for the linear models for the same validation data and step changes we used to validate the Neural Networks Models.

Output	**First Order**	**Higher Order**	**Scaling**
DC Bias	16.7	7.63	1
Mass 16	26.2	5.71	10^{-19}
Mass 30	51.1	14.2	10^{-21}

Table 8: Mean Square Error in the Linear Models for Validation Data

Step Change Input	Outputs					
	DC Bias		Mass 16 $\times 10^{19}$		Mass 30 $\times 10^{21}$	
	FO	**HO**	**FO**	**HO**	**FO**	**HO**
RF	1.80	0.557	10.8	1.27	15.9	7.94
NH_3	2.15	0.749	3.46	1.25	8.50	7.63
SiH_4	4.56	0.623	2.54	1.65	10.2	7.84

Table 9: Mean Square Error in the Linear Models for Step Changes

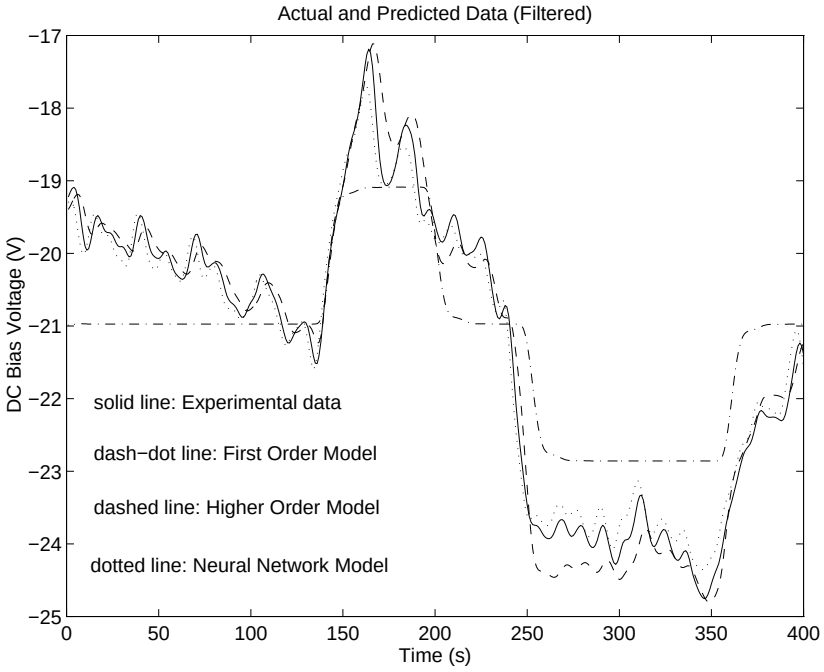

Figure 11: Actual and predicted system response for a Silane step change.

5.3 Comparison

In this section we compare the performances of the neural networks models obtained and against the performances of the linear models presented in the previous section. The following list summarizes the comparisons between the neural network models:

- Observing the MSE for the training and validation sets in tables 2 to 7 we conclude that the *NARMAX* model is the best choice to represent the system by a neural network model. This result came independently of the learning method selected.
- The error graphs in figure 10 clearly show that the *NARX* model is a bad choice and the *NARMAX* is the one with a more uniform and smaller error of the three methods.
- Within the *NARMAX* structure, looking in the third column of the tables 2 to 4 and the fourth column of tables 5 to 7, we observe that the best results came using the Extended Kalman Filter approach.
- The BFGS method listed in table 3 represents the fastest method of the three, having less number of iterations with respect of the BPTT method and also being faster per iteration with respect to the EKF method.

- The EKF method listed in table 4 combined with the elimination of nodes with small correlation gave us the smaller number of hidden nodes of all the methods.

Looking now at the results of the neural network approach together with the linear model approach we may conclude:

- Comparing the neural network tables 5 to 7 with the linear model table 9 for the step changes we may conclude that the neural network approach does not seem to represent an advantage to model the system against a higher order linear model in a small region around the operating point. It does constitute an advantage with respect to the first order model with time delay though.
- The table 8 shows the results of the linear models for the validation data. Comparing them with the results of the neural nets models shown in tables 2 to 4 we may say that the best neural network model performs considerably better when we examined the whole range of observations, like in the case of the validation data.
- These observations suggest that a neural network model can be more useful in the beginning of the process, when the system is in a very different state from the setting point designed, and performs as well as the higher order linear model in the operating region.
- However, this is not conclusive from both points of view since it has to be studied the possibility of having several linear systems for each region and a switching mechanism to connect them as well as having several neural networks models.

6 Discussion

In this section we summarize the work done in this project and present some future lines of research.

First, we present a basic introduction to neural networks. After that, the application of recurrent neural networks to model a dynamic system is explained in detail. We examined several architectures and algorithms for black-box system identification, including the *back-propagation* and Extended Kalman Filter approach. Some improvements to deal with multivariable systems, external inputs and non Gaussian noise are shown. In the second part of the report we present the experimental application of the algorithms developed. First, the physical system, a plasma enhanced chemical vapor deposition (PECVD) system is explained. The development of models based on experimental data for the PECVD system is necessary because the process is too complex to consider getting a good model by applying first principles. Neural networks were used in the model architecture to try to model the nonlinearities of the process and to overcome the drift. A detailed description of

the steps taken to perform the modeling, i.e. the data generation and the application of the neural network structures and algorithms, is given and the results are displayed. To have a better picture of the procedure and performance we compare these results with others obtained from linear model system identification methods, for the same system and using the same data.

From the development of the techniques and the comparisons obtained some ideas for future research can be pointed out.

- Because of the difficulties posed by the PECVD system to apply rigorously any first principle a black-box identification was performed. However, some basic knowledge of the process could be inserted in the model to reduce the number of parameters to estimate. The way in which we could introduce this a priori knowledge in a neural network would be a very important improvement in the future of neural network modeling.

- From the neural network point of view the approach made could deal with the drift problem introducing the time from the beginning of the experiments, but there is the possibility of making a robust approach directly related to the drift and not only to outliers. The introduction of the time suggests that the drift dynamics was not well represented to the neural network by the previous inputs, or their filter values. The study of an optimum representation of the data to feed the network would be critical to improve the performance of the model.

- The idea pointed out in the previous paragraph can be extensive to the outputs of the neural network. There can be a better representation for the system that its outputs or physical states itself. A research conducting to find the best representation of the system to be learned by a neural network is another possible line of investigation.

- Taking into account the main goal of the PECVD project there is also another main subject to study and this is how to introduce the neural network models effectively into the control architecture. There are already some ways to do it using mainly feedforward networks in an approach not totally rigorous, so this is still an open problem too.

Acknowledgments

I would like to thank to all the members of the PECVD group for helping with smart questions and suggestions to successfully conclude this project. This PECVD research group is supported by the National Science Foundation, and MKS, Inc. This particular research was supported by the Uruguayan Government (CONICYT) and the Interamerican Bank of Development (IBD).

References

[1] J.T. Connor, R.D. Martin, and L.E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Networks*, 5(2):240–254, March 1994.

[2] G. Cybenko. Approximation by superposition of sigmoidal funstions. *Mathematics of Controls, Signals and Systems*, (2):303–314, 1989.

[3] A. Draeger and S. Engell. Nonlinear model predictive control using neural net plant models. Technical report, Dept. of Chem. Engineering, Univ. of Dortmund, 1994.

[4] S.E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In T.J. Sejnowsky, G.E. Hinton, and D.S. Touretzky, editors, *1988 Connectionist Models Summer School*. Morgan Kaufmann, San Mateo, CA, 1988.

[5] S.S. Han, M. Ceiler, S.A. Bidstrup, P. Khol, and G. May. Modeling the properties of pecvd silicon dioxide films using optimized back-propagation neural networks. *IEEE Trans. Comp. Pack. and Man. Tech.*, 17:174–182, June, 1994.

[6] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*, volume I. Addison-Wesley, 1990. Lectures Notes.

[7] C.D. Himmel and G.S. May. Advantages of plasma etch modeling using neural networks over statistical techniques. *IEEE Trans. Semiconductor Manufactoring*, 6:103–111, May, 1993.

[8] J.J. Hopfield and D.W. Tank. "neural" computation of decision in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[9] J.J.Hopfield. Neural networks and physical systems with emergent computational abilities. In *Proc. National Academy of Sciences*, volume 79, pages 2554–2558, USA, 1982.

[10] B. Kim and G.S. May. An optimal neural network process model for plasma etching. *IEEE Trans. Semiconductor Manufactoring*, 7:12–21, February, 1994.

[11] T.J. Knight, X. Cheng, D.W. Greve, B.H. Krogh, and M.A. Gibson. Multivariable control of plasma enhanced chemical vapor deposition using real-time mass spectroscopy. In *32nd Allerton Conference on Communication, Control and Computing.*, 1994.

[12] T.J. Knight, X. Cheng, B.H. Krogh, D.W. Greve, and M.A. Gibson. Application of sampling quadrupole mass spectrometry to multivariable process control. In *Proc. Symp. on Porcess Control, Diagnostics, and Modeling in Semiconductor Manufacturing (I)*, Reno, Nevada, May 1995. Electrochemical Society Spring Meeting. to appear.

[13] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, 3rd. ed. edition, 1989.

[14] L. Ljung. *System Identification: Theory for the user.* Prentice Hall, 1987.

[15] R.D. Martin and D.J. Thomson. Robust-resistant spectrum estimation. *Proceedings IEEE*, 70(9):1097–1115, 1982.

[16] C.J. Masreliez. Approximate non gaussian filtering with linear state and observation relations. *IEEE Trans. AC*, 64(2):107–110, 1975.

[17] Michel Minoux. *Mathematical Programming. Theory and Algorithms.* John Wiley & Sons Ltd., 1986.

[18] B.L. Montgomery and B.V.K. Vijaya Kumar. Evaluation of the use of the hopfield neural network model as a nearest-neighbor algorithm. *Applied Optics*, 25(20):3759–3766, October 1986.

[19] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, pages 4–27, 1990.

[20] K.S. Narendra and K. Parthasarathy. Gradient methods for the optimization of dynamical systems containing neural networks. *IEEE Trans. Neural Networks*, pages 252–262, March, 1991.

[21] O. Nerrand, P. Rousell-Ragot, L. Personnaz, and G. Dreyfus. Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms. *Neural Computation*, pages 165–199, 1993.

[22] O. Nerrand, P. Rousell-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus. Training recurrent neural networks: Why and how? an illustration in dynamical process modeling. *IEEE Trans. Neural Networks*, 5:178–184, 1994.

[23] G.V. Puskorius and L.A. Feldkamp. Decoupled extended kalman filter training of feedforward layered networks. In *Proceedings IJCNN*, volume I, pages 771–777, 1991.

[24] T. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Trans. Neural Networks*, 5:298–305, 1994.

[25] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, (323):533–536, 1986.

[26] E. Sachs, A. Hu, and A. Ingolfsson. Run by run process control: Combining spc and feedback control. *IEEE Tran. on Semiconductor Manufacturing*, 8(1):26–43, Feb 1995.

[27] Arthur C. Sanderson. Application of neural networks in robotics and automation for manufacturing. In III W.T.Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control*, chapter 15, pages 365–386. MIT Press, second edition, 1991.

[28] Jonas Sjöberg. *Non-Linear System Identification with Neural Networks.* PhD thesis, Dept. of Electrical Engineering, Linköping University, 1995.

[29] D.L. Smith, A.S. Alimonda, D.C. Chen, S.E. Ready, and B. Wacker. Mechanism of sihxhx deposition from nh3/sih4 plasma. *J. Electrochemical Soc.*, 137, 1990.

[30] Takao Tanabe. Real-time control of plasma enhanced chemical vapor deposition (pecvd). Master's thesis, ECE Department, Carnegie Mellon University, 1994.

[31] P. Werbos. Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78(10):1550–1560, 1990.

[32] R.J. Williams. Training recurrent networks using the extended kalman filter. In *Proceedings IJCNN*, volume IV, pages 241–246, 1992.

[33] B.E. Ydstie. Forecasting and control using adaptive connectionist networks. *Computers Chem. Engng.*, 14:583–599, 1990.