Communicated by Robert Jacobs

Neural Network Process Models Based on Linear Model Structures

Gary M. Scott*
W. Harmon Ray
Department of Chemical Engineering, 1415 Johnson Drive, University of Wisconsin, Madison, WI 53706 USA

The KBANN (Knowledge-Based Artificial Neural Networks) approach uses neural networks to refine knowledge that can be written in the form of simple propositional rules. This idea is extended by presenting the MANNIDENT (Multivariable Artificial Neural Network Identification) algorithm by which the mathematical equations of linear dynamic process models determine the topology and initial weights of a network, which is further trained using backpropagation. This method is applied to the task of modeling a nonisothermal chemical reactor in which a first-order exothermic reaction is occurring. This method produces statistically significant gains in accuracy over both a standard neural network approach and a linear model. Furthermore, using the approximate linear model to initialize the weights of the network produces statistically less variation in model fidelity. By structuring the neural network according to the approximate linear model, the model can be readily interpreted.

1 Introduction

Research into the design of neural networks for process modeling has often ignored existing knowledge about the task at hand. One form this knowledge (often called the "domain theory") can take is embodied in traditional modeling paradigms. The recently developed KBANN approach (Towell *et al.* 1990) addresses this issue for tasks for which a domain theory (written using simple, nonrecursive propositional rules) is available. The basis of this approach is to use the existing knowledge to determine an appropriate network topology and initial weights, such that the network begins its learning process at a "good" starting point. One extension of this approach uses the mathematical form of a PID controller to determine the structure and initial weights of a neural network controller (Scott *et al.* 1992).

*Current address: Fiber Processes and Products Group, Forest Products Laboratory–USDA, One Gifford Pinchot Drive, Madison, WI 53705.

This paper describes the MANNIDENT algorithm, a method of using a traditional modeling paradigm to determine the topology and initial weights of a network. The use of linear models in this way eliminates network-design problems such as the choice of network topology (i.e., the number of hidden units) and reduces the sensitivity of the network to the initial values of the weights. Furthermore, the initial configuration of the network is closer to its final state than it would normally be in a randomly configured network. Thus, the MANNIDENT networks perform better and more consistently than the standard, randomly initialized three-layer approach.

The task we examine here is learning to model a nonlinear Multiple-Input, Multiple-Output (MIMO) system. There are a number of reasons to investigate this task using the MANNIDENT neural network modeling approach. First, many processes involve nonlinear input–output relationships, which can be handled by the nonlinear nature of neural networks. Second, there have been a number of earlier successful applications of neural networks to this task (Bhat and McAvoy 1990; Bhat *et al.* 1990; Donat *et al.* 1990; Haesloop and Holt 1990; Jordan and Jacobs 1990; Narenda and Parthasarathy 1990). Finally, the resulting MANNIDENT topology is much easier to interpret than the topology resulting from the standard network modeling approach.

In what follows, we introduce the principles of the MANNIDENT algorithm and then present an application of this technique to modeling of the temperature and concentration in a nonisothermal continuous-stirred, tank reactor (CSTR) that is highly nonlinear. The concluding sections describe some related work and some extensions and applications of the algorithm.

2 Design of Modeling Networks

In the modeling methodology presented here, we wish to create a neural network structure and initialize the weights using as much prior knowledge as is available. In addition, we would like the resulting neural network model to be capable of interpretation in terms of more traditional model forms. In this way, we can create efficient neural network models which can be readily understood in physical terms. To accomplish these goals, we describe in this section the MANNIDENT system, outlined in Figure 1.

MANNIDENT (**M**ultivariable **A**rtificial **N**eural **N**etwork **Ident**ification) is a construct that uses knowledge-based neural networks for the task of process modeling. There are several forms that the initial knowledge can take, depending on the type of mathematical model structure chosen for the system. In our case, the knowledge used to configure and initialize the network is embodied in approximate linear models that can be easily identified with traditional identification techniques. This technique is

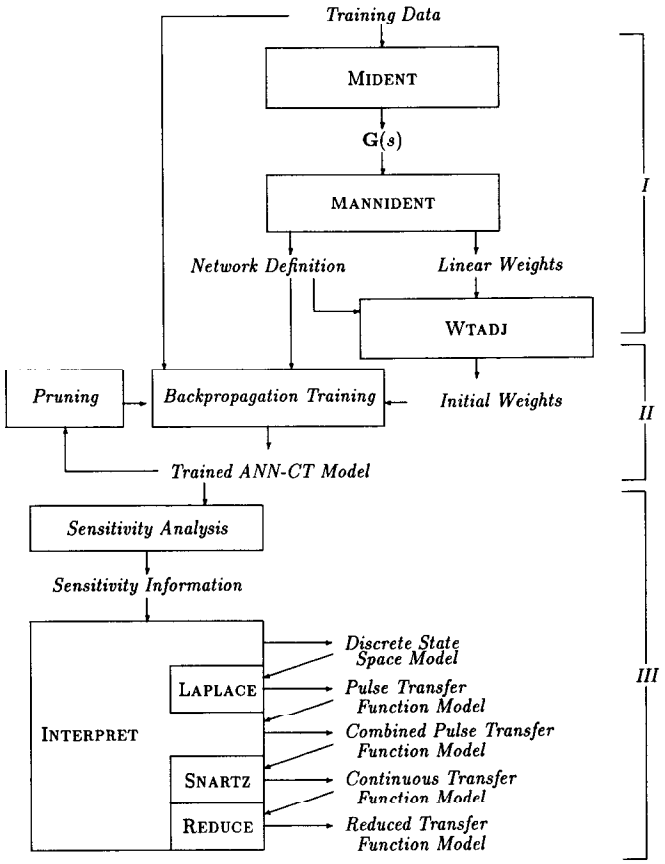

Figure 1: Overview of the MANNIDENT approach to process modeling showing the three phases of the process. *I* = Knowledge Insertion; *II* = Network Training; *III* = Network Interpretation. Many of the programs named in the figure are part of the CONSYD package (Holt *et al.* 1987).

easily extended to other more general model forms. This section gives an overview of this approach. Note that there are essentially three phases to the process of creating a model: knowledge insertion, network training, and network interpretation. The three phases are described in detail below.

2.1 Knowledge Insertion. The first step of the MANNIDENT process is the knowledge insertion phase. This step is important for several reasons. First, it defines the architecture of the network, thus eliminating the need for trial-and-error techniques to determine the proper number of hidden units. Second, the initialization of some of the weights to non-near-zero values gives the network a "good" starting place from which to continue learning, thus decreasing the learning time. Also, the use of knowledge-based networks tends to decrease the size of the network necessary to learn a given mapping. The initialization also tends to reduce the variability between runs since the effect of the initial weight randomization is reduced. Finally, since the network structure and weights correspond to parameters in a traditional linearized model, the physical interpretation of the trained ANN model is possible.

Consider a process which has N_{in} inputs, namely $u_1, \ldots, u_{N_{\text{in}}}$, and N_{out} outputs, namely $y_1, \ldots, y_{N_{\text{out}}}$, with a nonlinear, dynamic relationship between them. Given a time series of the inputs and the resulting outputs, a first-order approximation to the relationship can be determined. Using, for example, the MIDENT program of the CONSYD CAD package (Holt *et al.* 1987), or other linear identification technique, a transfer function matrix

$$\mathbf{G}(s) = \begin{bmatrix} g_{11}(s) & g_{12}(s) & \cdots & g_{1N_{\text{in}}}(s) \\ g_{21}(s) & g_{22}(s) & \cdots & g_{2N_{\text{in}}}(s) \\ \vdots & \vdots & \ddots & \vdots \\ g_{N_{\text{out}}1}(s) & g_{N_{\text{out}}2}(s) & \cdots & g_{N_{\text{out}}N_{\text{in}}}(s) \end{bmatrix}$$

can be written where each element of this matrix is in the form of a Laplace transform

$$g_{ji}(s) = \frac{K_{ji}e^{-t_{d,ji}s}}{\tau_{ji}s + 1}$$

where K_{ji}, τ_{ji}, and $t_{d,ji}$ are the steady-state gain, time constant, and time delay of each element, respectively. This represents a common form often used in empirical modeling (Stephanopoulos 1984). Greater dynamic flexibility of the ANN model can be achieved by representing the elements of this matrix by a sum of such Laplace transforms, which would result in additional hidden units in the network.

A realization of this transfer function uses $N_{\text{out}}N_{\text{in}}$ states, namely $x_1, x_2, \ldots, x_{N_{\text{out}}N_{\text{in}}}$, where one state is used for each element of the matrix. This would lead to the equivalent model in the time domain of

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t - t_{d,ji}) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) \end{aligned} \tag{2.1}$$

$$
\begin{array}{ll}
a_{kk} = -\frac{1}{\tau_{ji}} & \text{where } k = (j-1)N_{\text{in}} + i \\
a_{lk} = 0 & \text{otherwise} \\
b_{ki} = \frac{K_{ji}}{\tau_{ji}} & \text{where } k = (j-1)N_{\text{in}} + i \\
b_{ki} = 0 & \text{otherwise} \\
c_{jk} = 1 & \text{where } k = (j-1)N_{\text{in}} + i \\
c_{jk} = 0 & \text{otherwise}
\end{array}
$$

$$\text{for all } i = 1 \ldots N_{\text{in}} \text{ and } j = 1 \ldots N_{\text{out}}$$

An individual element from equation 2.1 can be written as

$$\frac{dx_k(t)}{dt} = a_{kk}x_k(t) + b_{ki}u_i(t - t_{d,ji})$$

where $k - (j-1)N_{\text{in}} + i$. Taking the finite difference approximation, the model becomes

$$
\begin{array}{rcl}
\frac{\Delta x_k(n)}{\Delta t} & = & a_{kk}x_k(n) + b_{ki}u_i\left(n - \frac{t_{d,ji}}{\Delta t}\right) \\
x_k(n) & = & x_k(n-1) + \Delta x_k(n)
\end{array}
$$

where n now indexes the discrete time steps. Note that this approximation is most appropriate when $\Delta t \ll \tau_{ji}$. This results in

$$x_k(n) = (1 + a_{kk}\Delta t)x_k(n-1) + b_{ki}\Delta t u_i\left(n - \frac{t_{d,ji}}{\Delta t}\right) \tag{2.2}$$

$$y_j(n) = \sum_k c_{jk}x_k(n) \tag{2.3}$$

Now consider using equations 2.2 and 2.3 to configure and initialize the network shown in Figure 2, which depicts the network created for a two-input, two-output system. Note that an "Elman" style of network results (Elman 1990). The important distinction of this network is that the values of the hidden layer are recurrent rather than the values of the output layer. In the figure, solid lines indicate connections initialized with process information that involve the nonlinearity of the hidden layer. All other weights connecting a layer of units to all subsequent layers are initialized to small random numbers (shown as dotted lines). Table 1 explains the various layers of units in this network, their sizes, and interpretations. Table 2 summarizes how the weights of the network are initialized. Note that in the case of time delays, the weight from the appropriate element in the $\hat{\mathbf{u}}_{\text{aug}}(n)$ layer is initialized in order to account for this delay. In the case that the delay is not an exact multiple of Δt, the weights corresponding to the units with the two nearest integer delays are proportionally initialized. The ANN resulting from this process is referred to as an ANN-CT model because it is initially based on a continuous time transfer function model.

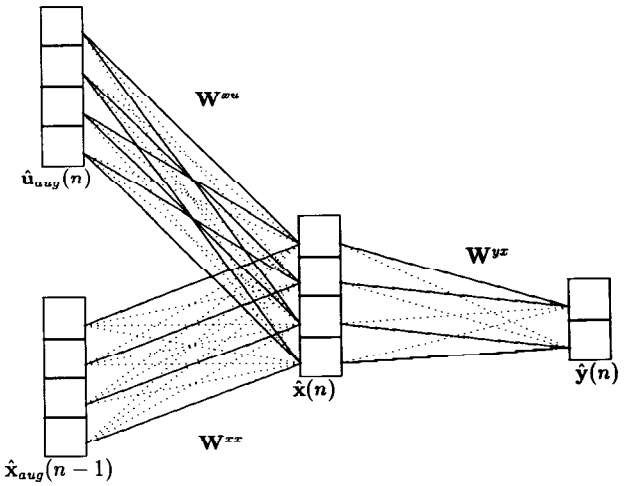

Figure 2: Recurrent feedforward network with topology based on a continuous transfer function (ANN-CT model).

Table 1: Sizes and Interpretations of the Units in the ANN CT Network.

Layer of units	Size N_{in} = number of inputs N_{out} = number of outputs	Interpretation
$\hat{\mathbf{u}}_{\text{aug}}(n)$	$N_{\text{in}}[\max(\frac{t_{d,ji}}{\Delta t}) + 1]$	The current input to the model and a number of past inputs depending on the maximum time-delay in the transfer function. The first N_{in} units in the layer contain the current values, the next N_{in} contain the values from one time step past, etc.
$\hat{\mathbf{x}}_{\text{aug}}(n-1)$	$N_{\text{out}}N_{\text{in}}$	The past states of the model.
$\hat{\mathbf{x}}(n)$	$N_{\text{out}}N_{\text{in}}$	The new states of the model calculated by the model.
$\hat{\mathbf{y}}(n)$	N_{out}	The new outputs of the model.

Table 2: Initial Values of the Weights in the ANN-CT Network.

Weight		
From	To	Initial value
$\hat{\mathbf{u}}_{\text{aug}}(n)$	$\hat{\mathbf{x}}(n)$	$b_{ki}\Delta t$
$\hat{\mathbf{x}}_{\text{aug}}(n-1)$	$\hat{\mathbf{x}}(n)$	$(1+a_{kk}\Delta t)$
$\hat{\mathbf{x}}(n)$	$\hat{\mathbf{y}}(n)$	c_{jk}

2.2 Network Training. The second major step in the process is the training of the configured network. This step contains two tasks: The first is network learning using the backpropagation algorithm. The second optional task is pruning, which removes connections (and units) that do not significantly contribute to the performance of the network. These two tasks are discussed below.

2.2.1 Backpropagation Learning. Although more sophisticated learning algorithms are available, backpropagation was used because of its simplicity and ease of implementation. Good values for the learning parameters were found empirically and used for all trials with no further modification. Training was terminated when the difference between the training error and the testing error began to diverge. This was done to prevent "memorization" of the training data causing a degradation of the ability of the network to generalize. The network was then pruned (see below) and retrained. Retraining was allowed to proceed for at least twice as long as the initial training in order to allow enough time for the network to reorganize and stopped when again the termination criteria was met after that point.

2.2.2 Weight Pruning. A relatively simple pruning technique was used here to remove unnecessary connections from the network: Since weight decay was used in training, the assumption is made that "low" valued weights are such because they had decayed to this value and are thus insignificant to the network's performance. For this reason, weights smaller than a certain fraction (usually 0–10%) of the average weight in that group of connections were set to zero and clamped to remain there. In this way, weights that contribute minimally were removed from the network.

3 Modeling Example

Here we provide an example illustrating three methods of creating a model for a process using a time series of input–output data. The time

series data was generated by a first principles model of a Continuous Stirred-Tank Reactor (CSTR) with an irreversible reaction (Uppal *et al.* 1976). The system consists of a well-stirred reactor in which an exothermic, first-order reaction is taking place. The parameters of this system were chosen such that it exhibited strong parametric sensitivity in the range to be modeled.

In order to simplify the differential equations resulting from material and energy balances of the system, several dimensionless quantities are defined and the inputs, disturbances, and outputs to the system are also made to be dimensionless. Also, since the threshold function of the units had a range of $(-1, 1)$, the inputs and outputs to the network were scaled to be within this range. These scaled values also represent deviation variables from steady state values. Table 3 summarizes these quantities and the constants of the system as well as the scaling used. The symbols used for this model are defined in the notation section.

The resulting differential equations (after substituting for the dimensionless quantities) are given below.

$$\frac{dx_1}{dt} = \frac{1}{\tau}\left[-(u_2+1)(x_1-d_2) + Da(1-x_1)\exp\left(\frac{\gamma x_2}{x_2+1}\right)\right] \tag{3.1}$$

$$\frac{dx_2}{dt} = \frac{1}{\tau}\left[(u_2+1)(d_1-x_2) + \frac{BDa}{\gamma}(1-x_1)\exp\left(\frac{\gamma x_2}{x_2+1}\right) - \beta(x_2-u_1)\right] \tag{3.2}$$

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 \end{aligned}$$

The data consisted of a training set of 1000 points representing the input and the output of the CSTR sampled at one time unit intervals. The inputs to the CSTR were a linear random distribution in the ranges of the scaled values. Furthermore, the time between changes in the inputs was exponentially and independently distributed. A testing set (labeled as testing set 1), distinct from the training set but similarly determined, also consisted of 1000 data points. A second testing set (labeled as testing set 2), consisting of 450 data points, consisting of individual step changes in each of the two inputs. Three models were created based on these data sets: A continuous linear model with first-order elements, a nonlinear ANN-ARMA model of the type found in the literature (Bhat and McAvoy 1990; Bhat *et al.* 1990; Donat *et al.* 1990; Haesloop and Holt 1990; Jones *et al.* 1989; Jordan and Jacobs 1990; Narenda and Parthasarathy 1990; Pineda 1989; Waibel 1989), and a nonlinear ANN-CT model of the architecture described above. For each of the network models, results are averaged over 10 runs each. Reported in the following section are the mean quadratic error for each of the data sets, as well as the 95% confidence intervals based on the multiple runs.

Table 3: Summary of Quantities in CSTR Model.

Symbol	Value or range	Description
	Model Parameters	
$\tau = \frac{V}{F_0}$	1.0	Nominal spacetime of reactor
$Da = \frac{k_0 e^{-\gamma} V}{F_0}$	0.11	Damköhler number
$\gamma = \frac{E}{RT_{f0}}$	20	Activation energy
$B = \frac{(-\Delta H)c_{A0}}{\rho C_p T_{f0}}[\frac{E}{RT_{f0}}]$	7.0	Heat of reaction
$\beta = \frac{hA}{F_0 \rho C_p}$	0.5	Heat transfer coefficient
F_0	1.0	Nominal feed flow
T_{f0}	300	Nominal feed temperature
c_{A0}	1.0	Nominal feed concentration
	Inputs (Control variables)	
$u_1 = \frac{T_c - T_{f0}}{T_{f0}}$	$T_c \in [250, 350]$	Coolant temperature
$u_2 = \frac{F - F_0}{F_0}$	$F \in [0.5, 1.5]$	Input feed rate
	Scaled Inputs	
$\bar{u}_1 = \frac{T_c - 300}{50}$	$\bar{u}_1 \in [-1, 1]$	Scaled coolant temperature
$\bar{u}_2 = \frac{F - 1}{0.5}$	$\bar{u}_2 \in [-1, 1]$	Scaled input feed rate
	Outputs (Measured variables)	
$y_1 = \frac{c_{A0} - c_A}{c_{A0}}$		Outflow concentration
$y_2 = \frac{T - T_{f0}}{T_{f0}}$		Outflow temperature
	Scaled Outputs	
$\bar{y}_1 = \frac{c_A - 0.755}{0.65}$	$\bar{y}_1 \in [-1, 1]$	Scaled outflow concentration
$\bar{y}_2 = \frac{T - 317.1}{65}$	$\bar{y}_2 \in [-1, 1]$	Scaled outflow temperature
	Disturbances (Noise)	
$d_1 = \frac{T_f - T_{f0}}{T_{f0}}$	$T_f \in [295, 305]$	Feed temperature
$d_2 = \frac{c_{A0} - c_{Af}}{c_{A0}}$	$c_{Af} \in [0.9, 1.1]$	Feed concentration

4 Modeling Results

4.1 Linear Continuous Model. Using the MIDENT program of the CONSYD package, a continuous transfer function model was identified from the data set. The structure of each element of the model was assumed to be first-order with time delay, which resulted in a model with 12 parameters (4 steady-state gains, 4 time constants, and 4 time delays).

$$\mathbf{G}(s) = \begin{bmatrix} \frac{-1.104e^{-0.0018s}}{8.24s+1} & \frac{0.265e^{-0.0184s}}{10.07s+1} \\ \frac{0.986e^{-0.0038s}}{8.46s+1} & \frac{-0.136e^{-0.0560s}}{9.91s+1} \end{bmatrix} \tag{4.1}$$

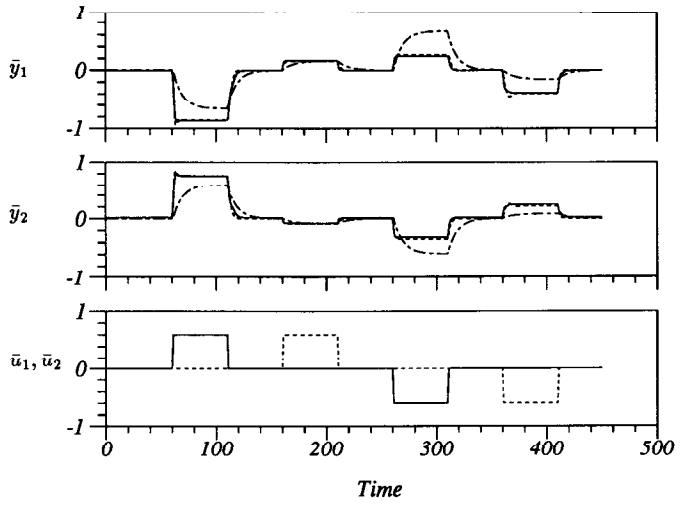

Figure 3: Concentration and temperature predictions of a continuous transfer function model with first-order elements and the ANN-CT model for testing set 1. The solid line is the ANN-CT model output, the dotted–dashed line is the linear model output, and the dotted line is the actual process output. For the input graph, $\bar{u}_1$ is represented by the solid line and $\bar{u}_2$ is represented by the dotted line. Note that the ANN-CT response and the actual process response are overlapping.

The performance of this model on testing set 1 is shown in Figure 3 (shown by the dotted–dashed lines) and represents the performance with which the MANNIDENT network discussed below was initialized. As can be seen, this type of model was unable to correctly account for the differing steady-state gains for positive and negative step changes in the inputs. Rather the model seemed to average the two effects, thus undershooting the correct value in one direction while overshooting the correct value in the other direction. Also, the identified time constants seemed to be too high; that is, the identified model outputs seemed to react too sluggishly to the shown changes in the inputs. These effects are the result of the identification software attempting to find the best fit linear model to a nonlinear process.

4.2 ANN-CT Model. A nonlinear network model using the MANNIDENT method developed here was created. First, MIDENT, using the network training data, identified a continuous first-order model of the pro-

cess as given in equation 4.1. This model was then used to create and initialize a network with the structure depicted in Figure 2. Figure 3 shows its performance on testing set 1, where the solid line represents this model's output which follows the desired output (dotted line) to a high degree of accuracy. Note that unlike the linear model, the ANN-CT model was able to represent the changing gains for positive and negative steps in the inputs.

The use of additional hidden units [increasing the size of the $\hat{\mathbf{x}}_{\text{aug}}(n-1)$ and $\hat{\mathbf{x}}(n)$ layers] did not improve the performance of the network, but did in fact slow down the learning process. These additional "state" neurons were not initialized with any model information, but instead had all of their associated weights assigned small random values. This failure to improve performance with the addition of more units is an indication that the chosen architecture and network size were appropriate to this task.

4.3 ANN-ARMA Model. For comparison purposes, ANN-ARMA models were also created for the example system. For a nonlinear ARMA network model, a standard three-layer recurrent feedforward that included past outputs of the network in the input layer of the network was configured. The initial weights were chosen as small random numbers. For comparison purposes, networks containing three through nine units in the hidden layer were created; this requires 35 to 101 weight parameters which covers the range of weights in the networks for the ANN-CT MANNIDENT models. Figure 4 shows the performance of the model on the testing set for the network with nine hidden units. Note that this model was also able to show the different gains resulting from positive and negative steps in the input. However, the model did show a noticeably greater offset from the actual process than the ANN-CT model.

4.4 Discussion. Table 4 summarizes the performance of all the models described above. For each of the models, the table gives the mean training error, the mean testing error on both test sets, and the number of adjustable parameters in the model (weights and biases for the network models). For the network models, these averages were taken over 10 runs for each configuration, with each run only differing in the random initialization of the weights. In the case of the ANN-CT the random initialization refers only to those weights not initialized with model information. Also given for each of the mean errors are the 95% confidence intervals for which this mean is the true value. These data are used as a measure of the variance in the models between runs.

Note that while the ANN models used anywhere from three to six times the number of parameters as the transfer function models, their applicability over a wider range of conditions means that a single ANN

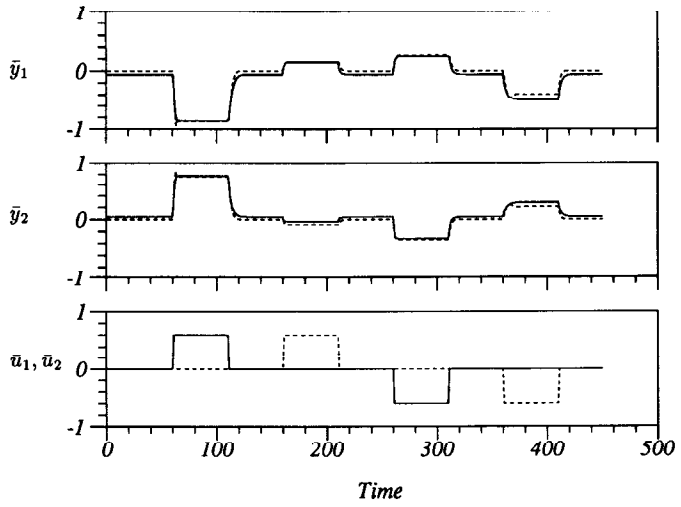

Figure 4: Concentration and temperature predictions of the three-layer ANN-ARMA network model for the second testing set. The solid line is the network output and the dotted line is the process output. For the inputs, $\bar{u}_1$ is represented by the solid line and $\bar{u}_2$ is represented by the dotted line.

model would be sufficient where several linear models (at different operating points) would be needed. Over the range studied in the previous examples, both the ANN-ARMA models and the ANN-CT models performed significantly better than either the linear model with first-order elements or the linear model with first-over–second-order elements (99.99% confidence). A single linear model was not sufficient to adequately model the process over this range.

Comparisons can also be made between the ANN-ARMA models and the ANN-CT (pruned) models. While the improvement in training errors for the ANN-CT model is not as highly significant (90% confidence for the ANN-ARMA model with 7 hidden nodes, 99.5% confidence for the ANN-ARMA model with 9 hidden nodes), the difference in the testing errors is significant (99.99% confidence for all ANN-ARMA models). This is an indication again that the ANN-CT models are better able to generalize to unseen testing data. However, of more importance is the statistical analysis using an F-test of the amount of variation between different runs of the same model. In all cases, the ANN-CT models showed significantly less variation between runs as compared to the ANN-ARMA models (99.99%

Table 4: Comparison of Final Integral Square Error Values of ANN Models and Traditional Models.[a]

Model	Training set error	Testing set I error	Testing set II error	Model size	Time (min)
		Continuous Transfer Function Models (Linear)			
First	0.05387	0.05790	0.01426	12	
First/Second	0.02764	0.03165	0.01242	20	
		ANN-ARMA Models			
(3)	0.00178 ± 0.000075	0.00356 ± 0.000141	0.00222 ± 0.000242	35	2.02
(4)	0.00175 ± 0.000081	0.00330 ± 0.000157	0.00188 ± 0.000161	46	2.16
(5)	0.00175 ± 0.000050	0.00318 ± 0.000114	0.00158 ± 0.000132	57	2.33
(7)	0.00177 ± 0.000086	0.00303 ± 0.000119	0.00139 ± 0.000152	79	2.91
(9)	0.00182 ± 0.000066	0.00287 ± 0.000102	0.00111 ± 0.000137	101	3.39
		ANN-CT Models			
Unpruned	0.00222 ± 0.000001	0.00277 ± 0.000002	0.00072 ± 0.000002	46	0.45
Pruned	0.00168 ± 0.000004	0.00218 ± 0.000014	0.00032 ± 0.000003	40	3.38

[a]The model size refers to the number of parameters in the model. The number in parentheses for the ANN-ARMA model is the size of the hidden layer.

confidence). The implication of this is that with the ANN-CT models, there is greater confidence that the model developed is close to the "true" model. From a computational point of view, fewer trials are needed in order to have confidence in the model that is produced. Also, the ANN-CT model, on subsequent runs, converged to near the same local minimum, allowing a comparison of individual weight values, which is not possible in the case of the randomly-initialized ANN-ARMA model. Thus, the ANN-CT models show significant performance improvements over the ANN-ARMA model, so that the ANN-CT formulation would be a good model structure for nonlinear modeling.

The training times between the various ANN models can also be compared. As can be seen from Table 4, the unpruned ANN-CT model trains approximately five times faster than all of the ANN-ARMA models. Also, this network is better able to generalize to the testing sets as shown by its better performance on those sets. The pruning of the ANN-CT model results in a network that takes approximately the same time to develop as the best ANN-ARMA model, but which has significantly better performance.

5 Knowledge Extraction

The final step in the modeling process is the interpretation of the trained network. This is important since the ANN is essentially a "black box" and the ability to interpret the model greatly enhances confidence in the results. Also, the capability of extracting traditional models from the ANN model allows traditional controller design techniques to be used based on the extracted model. A sensitivity analysis of the network at a particular operating point is the basis for the interpretation of the network. From this analysis, a discrete state space model of the process is created which is further manipulated to create a model of the desired form. The details of each step are given below.

5.1 Sensitivity Analysis. Since the interpretation of the network's connections produces an approximate linear model, the point of linearization must be chosen. After the network has achieved steady state at this point, a sensitivity analysis can be performed. Once this analysis is complete, the information can be interpreted to develop a discrete state space model of the process at the chosen point. The model takes the form of

$$\begin{aligned} \mathbf{x}(n) &= \Phi\mathbf{x}(n-1) + \beta\mathbf{u}(n) \\ \mathbf{y}(n) &= \mathbf{C}\mathbf{x}(n) \end{aligned}$$

where Φ, β, and $\mathbf{C}$ are constant matrices. These matrices can be seen to

take the following values resulting from the sensitivity analysis:

$$\begin{aligned}\Psi &= \frac{\partial \hat{\mathbf{x}}(n)}{\partial \hat{\mathbf{x}}_{\text{aug}}(n-1)} = \mathbf{f}'(\boldsymbol{\sigma}^x)\mathbf{W}^{xx} \\ \boldsymbol{\beta} &= \frac{\partial \hat{\mathbf{x}}(n)}{\partial \hat{\mathbf{u}}_{\text{aug}}(n)} = \mathbf{f}'(\boldsymbol{\sigma}^x)\mathbf{W}^{xu} \\ \mathbf{C} &= \frac{\partial \hat{\mathbf{y}}(n)}{\partial \hat{\mathbf{x}}(n)} = \mathbf{f}'(\boldsymbol{\sigma}^y)\mathbf{W}^{yx}\end{aligned}$$

which are easily calculated from the weights of the network ($\mathbf{W}^{xx}$, $\mathbf{W}^{xu}$, and $\mathbf{W}^{yx}$) and the current activation of the units ($\boldsymbol{\sigma}^x$ and $\boldsymbol{\sigma}^y$). The resulting model is in the form of a discrete state space model.

5.2 Approximate Linear Models. The result of the above analysis results in a discrete state space model of the process at a particular operating point. The model can then be transformed into transfer function models (either continuous or discrete) of the desired order. The details of the steps of the transformation are given in Scott (1993). The CONSYD package has several utilities that allow this model to be converted into other forms also useful for modeling (Holt *et al.* 1987).

5.3 Interpretation of a Trained ANN-CT Model. The trained ANN-CT model from the previous section was used for the extraction of a linear model around the center operating point. The method described above resulted in an approximate linear model of the following form around the steady state in Table 3

$$\mathbf{G}(s) = \begin{bmatrix} \frac{-1.300e^{-0.288s}}{0.690s^2+2.19s+1} & \frac{(0.030s+0.451)e^{-0.131s}}{0.033s^2+0.727s+1} \\ \frac{(0.049s+1.226)e^{-0.005s}}{0.337s^2+1.94s+1} & \frac{(0.021s-0.257)e^{-0.043s}}{0.012s^2+1.42s+1} \end{bmatrix} \tag{5.1}$$

where the degree of each element was determined to match the degree of the elements of the linear model that results from an exact local linearization of the original differential equations that describe the CSTR. This exact local linearization model is

$$\mathbf{G}_{\text{exact}}(s) = \begin{bmatrix} \frac{-1.24}{2.21s^2+2.85s+1} & \frac{(0.416s+0.411)}{2.21s^2+2.85s+1} \\ \frac{(0.849s+1.124)}{2.21s^2+2.85s+1} & \frac{-(0.291s+0.244)}{2.21s^2+2.85s+1} \end{bmatrix} \tag{5.2}$$

As can be seen, the model extracted from the network agreed very well with the local linear model in terms of the steady state gain. However, the denominator of each element shows some differences. The response of each of these models to step changes in the input is shown in Figure 5. From the figure, it is apparent that the response of the extracted model was very similar to that of the linearized model. This is an indication

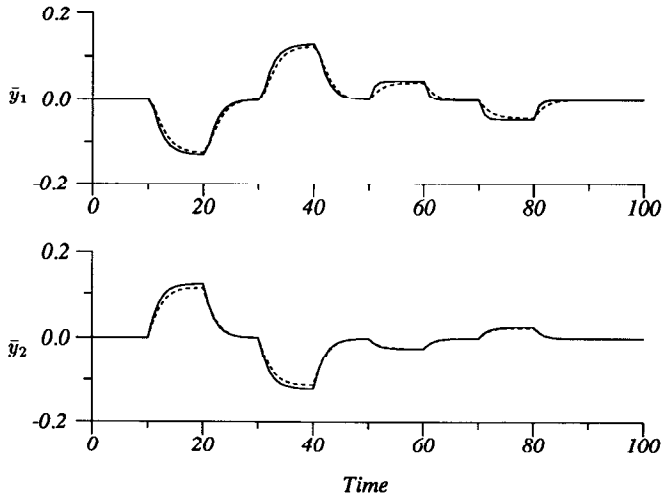

Figure 5: Comparison of response of linear model extracted from ANN-CT model and a linearized model of the actual process. The solid line is the model extracted from the ANN-CT model and the dotted line is the exact linearized model.

that the extracted model is an adequate representation of the process at this operating point for small deviations from steady state.

To demonstrate this further, the order of both the extracted model and the linearized model were reduced (using the CONSYD program REDUCE) to have elements consisting of first-order responses with time delay. The extracted model and reduced linearized model are shown in equations 5.3 and 5.4, respectively.

$$\mathbf{G}(s) = \begin{bmatrix} \frac{-1.300e^{-0.604s}}{1.89s+1} & \frac{0.451e^{-0.114s}}{0.68s+1} \\ \frac{1.226e^{-0.138s}}{1.78s+1} & \frac{-0.257e^{-0.092s}}{1.45s+1} \end{bmatrix} \tag{5.3}$$

$$\mathbf{G}_{\text{exact}}(s) = \begin{bmatrix} \frac{-1.240e^{-0.743s}}{2.20s+1} & \frac{0.411e^{-0.085s}}{1.80s+1} \\ \frac{1.124e^{-0.167s}}{1.99s+1} & \frac{-0.244e^{-0.050s}}{1.64s+1} \end{bmatrix} \tag{5.4}$$

A comparison of these two transfer functions shows that they are in good agreement, with the single exception of the time constant in the (1,2)-element. This again supports the idea that the extracted model was a good representation of the process at this point.

The comparisons above show that the network model is a adequate linear approximation to the process. Since large amplitude input changes

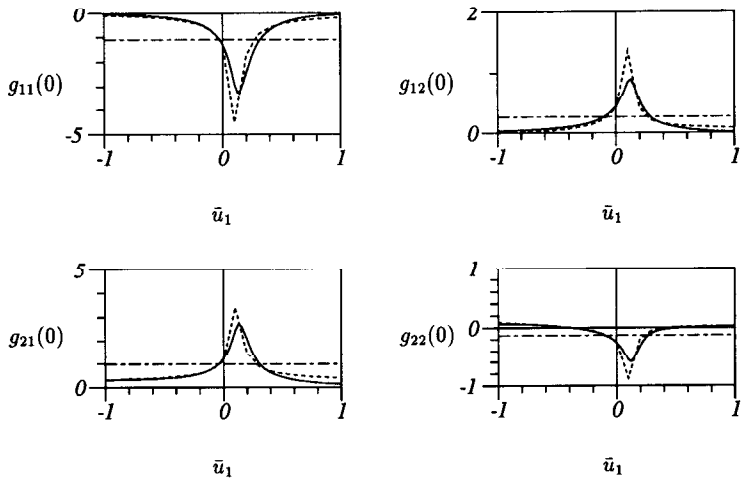

Figure 6: Extracted steady state gains from ANN-CT model of the CSTR as the first input is varied and the second input is held constant at zero. The solid line shows the steady state gain extracted from the network model; the dotted line shows the actual gain of the process at that point; and the dotted–dashed line shows the linear gain identified by MIDENT.

were used for training the network, the linearization reflects this. This approximatation could be made better if more training set data were chosen closer to the the desired steady state. To demonstrate the fidelity of the network model over a larger range of input amplitudes, plots of the steady state gain as the first input is changed are shown in Figure 6. In this figure, the horizontal dotted–dashed line is the constant gain from the linear model that was used to initialize the network. Through training, the steady-state gain of the network (the solid line) much more closely approximates the actual gains of the process (the dotted line). Thus the ANN-CT model captures the nonlinear gain behavior for the process and represents the response to dynamic inputs.

6 Related Work

Although many ANN applications in the literature do not consider prior knowledge, there have been other techniques developed for this purpose. As previously mentioned, the KBANN approach is one such method in which information in the form of propositional rules was used as the

basis for the structuring and initialization of the network (Towell *et al.* 1990). It was this idea that formed the basis of the MANNIDENT system described here.

The method of Haesloop and Holt uses linear information in the form of direct links from the input layer to the output layer, in addition to the standard pathways for activation through the hidden layers (Haesloop and Holt 1990). In this case, the output of the network would be calculated as the sum of the linear model and the nonlinear network, which is trained to account for only the residual nonlinear behavior. Foster *et al.* (1990) use an ANN to combine the predictions of traditional forecasting schemes to produce a combined forecast. Psichogios and Ungar (1992a,b) use an ANN to estimate unmeasured process parameters of a first principles model, creating a hybrid model with an ANN. Note that this typically requires the backpropagation of the error signals through the first principles model to the network.

The multivariate statistical methods such a Principal Components Analysis (PCA) and Projection to Latent Structures (PLS) can easily be used in conjunction with artificial neural networks. First, PCA and PLS can be used to preprocess the data in order to form a more meaningful set of inputs and outputs from which the ANN can learn. Removing highly correlated variables in this manner should potentially reduce the size of the network as well as the time necessary to train it. The method described here has the effect of replacing the regression step in PLS algorithm, with an ANN in order to introduce nonlinearities into the model (MacGregor *et al.* 1991; Qin 1991; Qin and McAvoy 1991). Second, PCA and PLS can be used to assign initial weights to an ANN, which a learning algorithm such a backpropagation can further refine (Piovoso and Owens 1991). Using this method, each of the units in the hidden layer represent one of the principal components of the input data.

7 Conclusions

The MANNIDENT algorithm for network design and initial weight specification significantly improves the performance of the nonlinear network in several ways. The algorithm quickly determines a relevant network architecture without resorting to trial-and-error methods, and through initialization of the weights with prior information, gives the learning algorithm an appropriate direction in which to continue learning, thus decreasing the training time and less variability between runs. Also, since the units and some of the weights initially have physical meaning, the MANNIDENT networks are easier to interpret after training, allowing classical models to be extracted from the network. Finally, since the network structure is based on a linear model of a type often used in modeling, the MANNIDENT algorithm is able to build on what is familiar.

Further enhancements using the MANNIDENT method are possible and are discussed more fully in Scott (1993) and Scott and Ray (1993a). Exploring different operating regions of the nonlinear CSTR, the ANN-CT network was able to model multiple steady state and limit cycle oscillations when the training data set contained these phenomena. Further enhancements expand the generality of the ANN-CT model. For example, one may use higher order continuous models or a discrete linear model of the process as the basis for the network. Since these models can be of arbitrary order, more complex network structures result. An example is given in Scott (1993) and Scott and Ray (1993a). Finally, the ANN-CT models can be incorporated into nonlinear model-based controllers, which show excellent performance in the face of setpoint changes and process disturbances and have excellent robustness properties (Scott and Ray 1993b).

8 Notation

ANN notation

$\mathbf{A}, \mathbf{B}, \mathbf{C}$	=	time domain model coefficients
$f(\cdot)$	=	network threshold function
i, j, k	=	subscript indices
$\mathbf{G}(s), \mathbf{G}(z)$	=	transfer function models
$K_{ji}, \tau_{ji}, t_{d,ji}$	=	linear model parameters
$N_{\text{in}}, N_{\text{out}}$	=	number of process inputs and outputs
Δt	=	discretization time step of model
$\mathbf{u}$	=	linear model inputs
$\hat{\mathbf{u}}, \hat{\mathbf{u}}_{\text{aug}}$	=	network inputs
$\bar{\mathbf{u}}$	=	scaled deviation inputs
$\mathbf{W}^{yx}$	=	weight matrix from $\hat{\mathbf{x}}$ to $\hat{\mathbf{y}}$
$\mathbf{x}$	=	linear model states
$\hat{\mathbf{x}}, \hat{\mathbf{x}}_{\text{aug}}$	=	network states (hidden units)
$\mathbf{y}$	=	linear model outputs
$\hat{\mathbf{y}}, \hat{\mathbf{y}}_{\text{aug}}$	=	network outputs
$\bar{\mathbf{y}}$	=	scaled deviation outputs
$\Phi, \beta, \mathbf{C}$	=	discrete state space parameters
σ_j^y	=	total input to unit

CSTR model nomenclature

A	=	heat transfer area
c_A	=	outlet concentration
c_{Af}	=	feed concentration
c_{A0}	=	nominal feed concentration
C_p	=	heat capacity
E	=	reaction activation energy
F	=	volumetric feed rate

F_0 = nominal feed rate
h = heat transfer coefficient
ΔH = heat of reaction
k_0 = kinetic preexponential factor
R = gas constant
t = time
T = feed temperature
T_c = coolant temperature
T_f = feed temperature
T_{f0} = nominal feed temperature
V = reactor volume
ρ = density

References

Bhat, N., and McAvoy, T. J. 1990. Use of neural nets for dynamic modeling and control of chemical process systems. *Comput. Chem. Eng.* **14**, 573–583.

Bhat, N. V., Minderman, P. A., McAvoy, T., and Wang, N. S. 1990. Modeling chemical process systems via neural computation. *IEEE Control Syst. Mag.* **10**, 24–29.

Donat, J. S., Bhat, N., and McAvoy, T. J. 1990. Optimizing neural net based predictive control. In *American Control Conference*, Vol. 3, pp. 2466–2471. IEEE, San Diego, CA.

Elman, J. L. 1990. Finding structure in time. *Cog. Sci.* **14**, 179–211.

Foster, B., Collopy, F., and Ungar, L. 1990. Neural network forecasting of short, noisy time series. Tech. Rep., University of Pennsylvania.

Haesloop, D., and Holt, B. R. 1990. A neural network structure for system identification. In *American Control Conference*, Vol. 3, pp. 2460–2465. IEEE, San Diego, CA.

Holt, B. et al. 1987. CONSYD: Integrated software for computer aided control system design and analysis. *Comput. Chem. Eng.* **11**(2), 187–203.

Jones, R. D., Lee, Y. C., Barnes, C. W., Flake, G. W., Lee, K., Lewis, P. S., and Qian, S. 1989. Function approximation and time series predication with neural networks. Tech. Rep. LA-UR 90-21, Los Alamos National Laboratory.

Jordan, M. I., and Jacobs, R. A. 1990. Learning to control an unstable system with forward modeling. In *Advances in Neural Information Processing Systems*, Vol. 2, pp. 325–331. Morgan Kaufmann, San Mateo, CA.

MacGregor, J. F., Marlin, T. E., Kresta, J. V., and Skagerberg, B. 1991. Multivariate statistical methods in process analysis and control. In *Fourth Interational Conference on Chemical Process Control*, Y. Arkun and W. H. Ray, eds., pp. 79–99. CACHE, AIChE, Padre Island, TX.

Narendra, K. S., and Parthasarathy, D. 1990. Identification and control of dynamical systems using neural networks. *IEEE Transact. Neural Networks* **1**(1), 4–27.

Pineda, F. J. 1989. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Comp.* **1**, 161–172.

Piovoso, M. J., and Owens, A. J. 1991. Sensor data analysis using artifical neural networks. In *Fourth Interational Coference on Chemical Process Control*, Y. Arkun and W. H. Ray, eds., pp. 101–118. CACHE, AIChE, Padre Island, TX.

Psichogios, D. C., and Ungar, L. H. 1992a. A hybrid neural network-first principles approach to process modeling. *AIChE J.* **38**(10), 1499–1511.

Psichogios, D. C., and Ungar, L. H. 1992b. Process modeling using structured neural networks. In *Proceedings of the 1992 American Control Conference* pp. 1917–1921. IEEE, Piscataway, NJ.

Qin, S. J. 1991. Neural net PLS approach to dynamic modeling: Method and application. Tech. Rep., Department of Chemical Engineering, University of Maryland.

Qin, S. J., and McAvoy, T. J. 1991. Nonlinear PLS modeling using neural networks. Tech. Rep., Department of Chemical Engineering, University of Maryland.

Scott, G. M. 1993. Knowledge-based artificial neural networks for process modelling and control. Ph.D. thesis, University of Wisconsin, Madison, WI.

Scott, G. M., and Ray, W. H. 1993a. Creating efficient nonlinear neural network process models that allow model interpretation. *J. Process Control* **3**(3), 163–178.

Scott, G. M., and Ray, W. H. 1993b. Experiences with model-based controllers based on neural network process models. *J. Process Control* **3**(3), 179–196.

Scott, G. M., Shavlik, J. W., and Ray, W. H. 1992. Refining PID controllers using neural networks. *Neural Comp.* **4**(5), 746–757.

Stephanopoulos, G. 1984. *Chemical Process Control: An Introduction to Theory and Practice.* Prentice Hall, Englewood Cliffs, NJ.

Towell, G., Shavlik, J., and Noordewier, M. 1990. Refinement of approximate domain theories by knowledge-base neural networks. In *Eighth National Conference on Artificial Intelligence*, pp. 861–866. AAAI Press, Menlo Park, CA.

Uppal, A., Ray, W. H., and Poore, A. B. 1976. The classification of the dynamic behavior of continuous stirred tank reactors—influence of reactor residence time. *Chem. Eng. Sci.* **31**, 205–214.

Waibel, A. 1989. Modular construction of time-delay neural networks for speech recognition. *Neural Comp.* **1**, 39–46.

Received May 6, 1993; accepted November 4, 1993.