Creating efficient nonlinear neural network process models that allow model interpretation

Gary M. Scott* and W. Harmon Ray†

Department of Chemical Engineering, 1415 Johnson Drive, University of Wisconsin, Madison, WI 53706, USA

Received 4 May 1993; revised 9 August 1993

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 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 modelling a non-isothermal CSTR 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 accuracy. By structuring the neural network according to the approximate linear model, the model can be readily interpreted.

Keywords: neural networks; modelling; chemical reactor

Research into the design of neural networks for process modelling has largely ignored existing knowledge about the task at hand. One form this knowledge (often called the 'domain theory') can take is embodied in traditional modelling paradigms. The recently developed KBANN approach addresses this issue for tasks for which a domain theory is available. Existing process models have also been incorporated into the structure or combined with neural networks. The basis of our 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.

This paper describes the MANNIDENT algorithm, a method of using a traditional modelling 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 neural networks. 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. Finally, the resulting topology is much easier to interpret than the topology resulting from the standard network modelling approach.

The next sections introduce the concept of artificial neural networks and describes the MANNIDENT algorithm. The task to which this paradigm is being applied is modelling the temperature and concentration in a non-isothermal continuously stirred tank reactor (CSTR) that is highly nonlinear. The concluding sections describe some related work in the area of neural networks in modelling, an application of this model in model-based control and some extensions of this algorithm.

Recurrent neural networks

Artificial neural networks (ANNs) – also known as connectionist models, parallel distributed processing models and neuromorphic models – are a rapidly evolving facet of artificial intelligence (AI). ANNs are massively paral-
Efficient nonlinear neural network process models: G. M. Scott and W. H. Ray

lel, interconnected networks of simple (usually adaptive) elements which interact with the objects of the real world in the same way as biological nervous systems do. Because of the biological basis for artificial neural networks, it is not surprising that many of the terms used in their study are borrowed from neurophysiology. The processing units are also known as neurons, nodes or processors in the literature, while the connections between the units are also known as interconnects, synapses or weights. The pattern of the connections between the units determines the architecture of the network which, in the extremes, can be fully interconnected (recurrent networks) or connected in one direction only (feed forward networks). The architecture, of course, depends largely on the desired task that the network is to perform. These two components, the units and the connections, are discussed extensively in the literature, along with the third component of ANNs, the learning algorithms.

Backpropagation, or the generalized delta rule, is an extension of the delta rule that can train multilayer ANNs. Backpropagation, with its variations, is currently the mainstay of artificial neural network learning. Several researchers developed this learning paradigm independently. Many others discuss it in general. Others have compared backpropagation to other forms of learning in the field of AI and with more traditional modeling approaches. Furthermore, many general articles contain at least an introduction to backpropagation.

The three-layer artificial neural network represents a feed-forward structure that is useful for many applications. In fact, most modelling in the literature uses this architecture or a variation thereof. In the three-layer ANN, the units are arranged into three layers: An input layer, which receives input data from outside the network; a hidden layer, which receives signals from the input layer; and an output layer, which receives signals from the hidden layer and represents the final output of the network. In the multi-layer, ANN, the calculation of the network activation proceeds layer by layer from the input layer through the hidden layers to the output layer.

A type of artificial neural network often used in dynamic modeling is the recurrent feed-forward network. This type of network addresses the issue that in typical feed-forward networks, the state of the network is not allowed to depend on the previous states. In a recurrent network, the inputs to the network are the current \( u(n) \) and past \( u(n-i) \) inputs to the system, as well as the past outputs \( y(n-i) \). The network then predicts the current output of the system. In Figure 1, the boxes labelled \( z^{-i} \) are time delay elements which take as an input the current value of a variable, and output an augmented vector which also includes the variable at points in the past. This type of modeling creates an ARMA-like (auto-regressive moving average) model using ANN[2,5,9,21,22]. Subsequently, an ANN model of this type will be referred to as an ANN-ARMA model.

An alternative to using the outputs of the network as inputs is to use the past values of the hidden layer as inputs to the network (Figure 2), creating what has been termed in the literature to be an Elman network[24,25]. In this type of network architecture, the network can adjust the weights from the past state values to the hidden layer \( \psi(W) \) in order to encode information about context. The task of time series analysis, which is highly dependent on context or prior states, is represented well by this architecture. This architecture is applied to process modeling in what follows where the hidden units are interpreted as the states of the process.

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 3.

MANNIDENT (multivariable artificial neural network identification) is a construct that uses knowledge-based neural networks for the task of process modeling. In this case, the knowledge used to configure and initialize the network are approximate linear models that can

---

Figure 1 A recurrent feed-forward network in which the past inputs and past outputs are also inputs to the network.

Input Layer Hidden Layer Output Layer
\[ u \rightarrow u_{out} \rightarrow \psi(W^{in}) \rightarrow f \rightarrow \tilde{y}_{out} \rightarrow \psi(W^{out}) \rightarrow \tilde{y} \]

Figure 2 A recurrent feed-forward network in which the past hidden unit values and past outputs are the inputs to the network ('Elman'-style network).

Input Layer Hidden Layer Output Layer
\[ \tilde{u} \rightarrow \tilde{u}_{out} \rightarrow \psi(W^{in}) \rightarrow f \rightarrow \tilde{y}_{out} \rightarrow \psi(W^{out}) \rightarrow \tilde{y} \]
Figure 3 Overview of the MANNIDENT approach to process modelling showing the three phases of the process. I = knowledge insertion; II = network training; III = network interpretation. The terms representing program names (e.g. WTADI, LAPLACE, etc.) are explained later in the text.

be easily identified with traditional identification techniques. This section gives an overview of this approach. Note that there are essentially three phases to the process of creating a model: knowledge insertion, ANN training and ANN interpretation. The three phases are described in detail below.

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 layers and hidden units. Second, the initialization of some of the weights to non-zero values gives the network a ‘good’ starting point 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.

The following sections discuss two methods of initializing the weights of an ANN to be used for process modelling. The first technique depends on a finite difference approximation to the time domain realization of the Laplace transfer function matrix in which each element is represented as a first-order process with time delay. The second method depends on a pulse transfer function where each element can be of arbitrary order, thus leading to arbitrarily complex networks.

Using continuous transfer functions for network initialization. Consider a process which has \( N_{m} \) inputs, namely \( u_{1}, \ldots, u_{N_{m}} \), and \( N_{out} \) outputs, namely \( y_{1}, \ldots, y_{N_{out}} \), 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 CONSYSYNA package, or some other linear identification technique, a transfer function matrix

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

can be written where each element of this matrix is of the form:

\[
g_{ij}(s) = \frac{K_{ij}e^{-t_{ij}p}}{r_{ij} + 1}
\]

where \( K_{ij}, r_{ij} \) and \( t_{ij} \) are the steady-state gain, time constant and time delay of each element, respectively.

A realization of this transfer function uses \( N_{out}/N_{m} \) states, namely \( x_{1}, x_{2}, \ldots, x_{N_{out}} \) where one state is used for each element of the matrix. This would lead to the equivalent model in the time domain of:

\[
\begin{align*}
\frac{dx_{i}(t)}{dt} &= Ax(t) + Bu(t - t_{d}) \\
y(t) &= Cx(t)
\end{align*}
\]

\[a_{ik} = -\frac{1}{r_{ij}} \text{ where } k = (j - 1)N_{m} + i \]

\[b_{ik} = K_{ij}/r_{ij} \text{ where } k = (j - 1)N_{m} + i \]

\[c_{ik} = 1 \text{ where } k = (j - 1)N_{m} + i \]

for all \( i = 1 \ldots N_{m} \) and \( j = 1 \ldots N_{out} \)

An individual element from Equation (1) can be written as

\[
\frac{dx_{i}(t)}{dt} = a_{ik}x(t) + b_{ik}u(t - t_{d})
\]

where \( k = (j - 1)N_{m} + i \). Taking the finite difference approximation, the model becomes:
Efficient nonlinear neural network process models: G. M. Scott and W. H. Ray

Figure 4 Recurrent feed-forward network with topology based on a continuous transfer function (ANN-CT model)

\[
\frac{\Delta x_i(n)}{\Delta t} = a_{ik}x_k(n) + b_{ki}u(n - t_{ik}/\Delta t)
\]

\[x_i(n) = x_i(n - 1) + \Delta x_i(n)\]

where \( n \) now indexes the discrete time steps. Note that this approximation is most appropriate when \( \Delta t \ll \tau_p \).

This results in:

\[x_i(n) = (1 + a_{ik}\Delta t)x_i(n - 1) + b_{ki}\Delta tu(n - t_{ik}/\Delta t)\]  \(2\)

\[y(n) = \sum_{i} c_{ik}x_i(n)\]  \(3\)

Now consider using Equations (2) and (3) to configure and initialize the network shown in Figure 4, which depicts the network created for a two-input, two-output system. Note that an Elman style of network results. 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). These additional weights allow the network to learn the interactions that are not accounted for in the linear model of the process. 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 \(u_{n-k}^{(m)}\) layer is initialized in order to account for this delay. In the case that the delay is not an exact multiple of \(\Delta t\), 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.

Using discrete transfer functions for network initialization. Although artificial neural networks constructed using continuous transfer functions are less of a 'black box' than the ARMA models found in the literature, they still exhibit several drawbacks. First, the insertion of initial information into the network required that a finite difference approximation be made, which may be inaccurate if \(\Delta t\) is chosen too large. Second, the algorithm, as described, required that the transfer functions be written in a first-order form, although this could be easily extended to allow each element to be written as the sum of such terms. Finally, there are difficulties in recovering information from the trained network in the same form in which it was installed. The following describes a method that again utilizes a traditional model identification technique to determine the architecture and initial weights of an ANN model. In this case, however, a discrete (pulse transfer function) model is used.

Consider a process which has \(N_{in}\) inputs, namely \(u_1, \ldots, u_{N_{in}}\), and \(N_{out}\) outputs, namely \(y_1, \ldots, y_{N_{out}}\), with a nonlinear, dynamic relationship between them. Again, an identification program, such as MIDENT \(^{23}\) is used to determine an approximate linear, discrete model of the relationship between the inputs and the outputs. This results in a transfer function of the form:

\[G(z) = \left[ \begin{array}{cccc} g_{11}(z) & g_{12}(z) & \cdots & g_{1N_{out}}(z) \\
g_{21}(z) & g_{22}(z) & \cdots & g_{2N_{out}}(z) \\
\vdots & \vdots & \ddots & \vdots \\
g_{N_{in}1}(z) & g_{N_{in}2}(z) & \cdots & g_{N_{in}N_{out}}(z) \end{array} \right] \]

where each element of this matrix is of the form:

\[g_{ij}(z) = \sum_{k=0}^{M_{ij}} B_{jk}z^{-k} + \sum_{k=0}^{N_{ij}} A_{jk}z^{-k} \]

where \( M_{ij} \) and \( N_{ij} \) are the degree of the numerator and denominator of the \( z \)-transfer function, respectively, and \( n_{d_{ij}} \) is the number of time delay steps. \( A_{jk} \) and \( B_{jk} \) are the coefficients of the polynomial.

A realization of this transfer function uses \(N_{out}/N_{in}\) states, namely \(x_{11}, x_{12}, \ldots, x_{N_{out}N_{in}}\), where one state is used for each element of the matrix. This would lead to the equivalent model in the time domain of:

\[x_{ji}(n) = K_{ji} \sum_{k=1}^{M_{ji}} B_{jk}u_{n-k}(x_{j,k-1} - x_{j,k-2} - \cdots - x_{j,k-N_{out}}) + \sum_{k=1}^{N_{in}} a_{jk}x(n - k) \]

\[y_{j}(n) = \sum_{j=1}^{N_{out}} \sum_{i=1}^{N_{in}} x_{ji}(n) \]

Now consider using Equations (4) to configure and initialize the network shown in Figure 5, which depicts the network created for a two-input, two-output system. In the figure, solid lines indicate connections initialized...
Table 1  Sizes and interpretations of the units in the ANN-CT network

<table>
<thead>
<tr>
<th>Layer of units</th>
<th>Size</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{u}_{\text{ct}}(n) )</td>
<td>( N_u = \max(t_u/\Delta t) + 1 )</td>
<td>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_u ) units in the layer contain the current values, the next ( N_u ) contain the values from one time step past, etc.</td>
</tr>
<tr>
<td>( x_{\text{ct}}(n-1) )</td>
<td>( N_u N_n )</td>
<td>The past states of the model</td>
</tr>
<tr>
<td>( \hat{x}(n) )</td>
<td>( N_u N_n )</td>
<td>The new states of the model calculated by the model</td>
</tr>
<tr>
<td>( \hat{x}(n) )</td>
<td>( N_u N_n )</td>
<td>The new outputs of the model</td>
</tr>
</tbody>
</table>

Table 2  Initial values of the weights in the ANN-CT network

<table>
<thead>
<tr>
<th>Weight</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>To</td>
</tr>
<tr>
<td>( \hat{u}_{\text{ct}}(n) )</td>
<td>( \hat{x}(n) )</td>
</tr>
<tr>
<td>( x_{\text{ct}}(n-1) )</td>
<td>( x(n) )</td>
</tr>
<tr>
<td>( \hat{x}(n) )</td>
<td>( \hat{x}(n) )</td>
</tr>
</tbody>
</table>

Figure 5  Recurrent feed-forward network with topology based on a pulse transfer function (ANN-DT model)

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). Again, these weights allow the network to learn unmodelled interactions between the variables. Table 3 explains the various layers of units in this network, their sizes and interpretations. Table 4 summarizes how the weights of the network are initialized. The network model created in this manner is referred to as an ANN-DT model since it was based on a discrete time model. Note that an important difference between the ANN-CT and ANN-DT models is that the latter can have as inputs to the network a larger number of past states because the ANN-DT model is based on an impulse response model with arbitrary sample time, \( \Delta t \), while the ANN-CT model must have \( \Delta t \) relatively small. Thus the ANN-DT model allows the network output to depend on network states further into the past. However, it must be remembered that both types of network predict the process outputs at discrete time intervals into the future.

Adjustment for nonlinearity. Initializing the weights of the network in the manner given above assumes that the threshold functions of the units in the network are linear, that is:

\[
\hat{y}_i = f_{\text{linear}}(\sigma^e) = \sigma^e_i
\]

but the strength of neural networks lie in their having nonlinear (typically sigmoidal) threshold functions. For this reason, the MANNIDENT system initially adjusts the weights described above so that the linear response is approximated by a sigmoidal function at the origin. This means that the two conditions:

\[
\begin{align*}
  f'_{\text{linear}}(0) &= f'(0) \\
  f_{\text{linear}}(0) &= f(0)
\end{align*}
\]

must be satisfied at the origin. Since from Equation (5):

\[
\begin{align*}
  f'_{\text{linear}}(0) &= 1 \\
  f'(0) &= \frac{1}{2}
\end{align*}
\]

for units in the \( x \) layer having the threshold function given by:

\[
\hat{y}_i = f(\sigma^e_i) = \frac{1 - \exp(-\sigma^e_i)}{1 + \exp(-\sigma^e_i)}
\]

this is accomplished by multiplying each of the initial weights in Tables 2 and 4 by a factor of 2. No adjustments were made to the weights feeding into the output layer, \( \hat{y} \), because they had a small effect on the training performance.

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.

**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 criterion was met after that point.

**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.

**Illustrative 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 were generated by a first principles model of a continuous stirred tank reactor (CSTR) with an irreversible reaction. The system consists of a well-stirred reactor in which an exothermic, first-order reaction is taking place. While this system is one of the simplest reactors that can be described, it exhibits several nonlinear characteristics such as multiple steady states, oscillations and extreme parametric sensitivity.

First, 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 5 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:

\[
\begin{align*}
\frac{dx_1}{dt} &= \frac{1}{\tau} \left[ - (u_1 + 1)(x_1 - u_1) + Da(1 - x_1) \right] \\
&\quad \exp \left( \frac{\gamma x_2}{x_2 + 1} \right) \\
&\quad \frac{\gamma x_2}{x_2 + 1} \right] \\
\end{align*}
\]\n
The data consisted of a training set of 1000 points representing the input and the output of the CSTR sampled at one time unit interval. The inputs to the CSTR were a linear random distribution in the range of the scaled values. Furthermore, the time between changes in the inputs was exponentially and independently distributed. A sample of this data set is shown in Figure 6. In this figure, as well as in all subsequent figures showing process inputs, the first input, \(u_1\), is indicated by the solid line, and the second input \(u_2\) is indicated by the dotted line. A testing set (labelled as testing set 1), distinct from
Table 1  Sizes and interpretations of the units in the ANN-CT network

<table>
<thead>
<tr>
<th>Layer of units</th>
<th>Size</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_{\text{out}}(t)$</td>
<td>$N_{\text{in}} \max\left(n_{\text{out}}/\Delta t\right) + 1$</td>
<td>The current input to the model and a number of past inputs depending on the maximum time delay in the transfer function. The first $N_{\text{in}}$ units in the layer contain the current values, the next $N_{\text{out}}$ contain the values from one time step past, etc.</td>
</tr>
<tr>
<td>$\hat{x}_{\text{out}}(t - 1)$</td>
<td>$N_{\text{in}} N_{\text{out}}$</td>
<td>The past states of the model</td>
</tr>
<tr>
<td>$\hat{y}(n)$</td>
<td>$N_{\text{out}}$</td>
<td>The new states of the model calculated by the model</td>
</tr>
<tr>
<td>$\hat{y}(n)$</td>
<td>$N_{\text{out}}$</td>
<td>The new outputs of the model</td>
</tr>
</tbody>
</table>

Table 2  Initial values of the weights in the ANN-CT network

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_{\text{out}}(n)$</td>
<td>$\hat{x}_{\text{out}}(n - 1)$</td>
<td>$\theta_{\text{out}} \Delta t$</td>
</tr>
<tr>
<td>$\hat{x}_{\text{out}}(n)$</td>
<td>$\hat{x}(n)$</td>
<td>$1 - \exp(-\alpha_{\text{out}} \Delta t)$</td>
</tr>
<tr>
<td>$\hat{x}(n)$</td>
<td>$\hat{y}(n)$</td>
<td>$\alpha_{\text{out}}$</td>
</tr>
</tbody>
</table>

Figure 5  Recurrent feed-forward network with topology based on a pulse transfer function (ANN-DT model)

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). Again, these weights allow the network to learn unmodelled interactions between the variables. Table 3 explains the various layers of units in this network, their sizes and interpretations. Table 4 summarizes how the weights of the network are initialized. The network model created in this manner is referred to as an ANN-DT model since it was based on a discrete time model. Note that an important difference between the ANN-CT and ANN-DT models is that the latter can have as inputs to the network a larger number of past states because the ANN-DT model is based on an impulse response model with arbitrary sample time, $\Delta t$, while the ANN-CT model must have $\Delta t$ relatively small. Thus the ANN-DT model allows the network output to depend on network states further into the past. However, it must be remembered that both types of network predict the process outputs at discrete time intervals into the future.

Adjustment for nonlinearity. Initializing the weights of the network in the manner given above assumes that the threshold functions of the units in the network are linear, that is:

$$\hat{y}_j = f_{\text{linear}}(\sigma_j') = \sigma_j'$$

but the strength of neural networks lie in their having nonlinear (typically sigmoidal) threshold functions. For this reason, the MANNIDENT system initially adjusts the weights described above so that the linear response is approximated by a sigmoidal function at the origin. This means that the two conditions:

$$f_{\text{linear}}(0) = f'(0)$$
$$f_{\text{linear}}(0) = f(0)$$

must be satisfied at the origin. Since from Equation (5):

$$f_{\text{linear}}(0) = 1$$

and

$$f'(0) = \frac{1}{2}$$

for units in the $x$ layer having the threshold function given by:

$$\hat{y}_j = f(\sigma_j') = \frac{1 - \exp(-\sigma_j')}{1 + \exp(-\sigma_j')}$$

this is accomplished by multiplying each of the initial weights in Tables 2 and 4 by a factor of 2. No adjustments were made to the weights feeding into the output layer, $\hat{y}$, because they had a small effect on the training performance.

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
Table 3: Sizes and interpretations of the units in the ANN-DT network

<table>
<thead>
<tr>
<th>Layer of units</th>
<th>Size</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{u}_n(n) )</td>
<td>( N_u \max(n_u + M_u) + 1 )</td>
<td>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_u ) units in the layer contain the current values, the next ( N_u ) contain the values from one time step past, etc.</td>
</tr>
<tr>
<td>( \hat{x}_{n-1} )</td>
<td>( N_u N_v \max(N_d) )</td>
<td>The current states of the model and a number of past states depending on the order of the model</td>
</tr>
<tr>
<td>( \hat{y}(n) )</td>
<td>( N_v )</td>
<td>The new states of the model calculated by the model</td>
</tr>
<tr>
<td>( \hat{y}(n) )</td>
<td>( N_\text{out} )</td>
<td>The new outputs of the model</td>
</tr>
</tbody>
</table>

Table 4: Initial values of the weights in the ANN-DT network

<table>
<thead>
<tr>
<th>Weight</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>To</td>
</tr>
<tr>
<td>( \hat{u}_n(n) )</td>
<td>( \hat{x}(n) )</td>
</tr>
<tr>
<td>( \hat{x}_{n-1} )</td>
<td>( \hat{x}(n) )</td>
</tr>
<tr>
<td>( \hat{y}(n) )</td>
<td>( \hat{y}(n) )</td>
</tr>
</tbody>
</table>

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.

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 stop when again the termination criterion was met after that point.

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.

Illustrative 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 were generated by a first principles model of a continuous stirred-tank reactor (CSTR) with an irreversible reaction. The system consists of a well-stirred reactor in which an exothermic, first-order reaction is taking place. While this system is one of the simplest reactors that can be described, it exhibits several nonlinear characteristics such as multiple steady states, oscillations and extreme parametric sensitivity.

First, 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 5 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:

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

(6)

\[
\frac{dx_2}{dt} = \frac{1}{\tau} \left[ (u_2 + 1)(d_1 - x_2) + \frac{BD_\text{in}}{\gamma} (1 - x_1) \right] \exp \left( \frac{-\gamma x_2}{x_2 + 1} \right) - \beta(x_2 - u_1) \]

(7)

\[y_1 = x_1 \]

\[y_2 = x_2 \]

The data consisted of a training set of 1000 points representing the input and the output of the CSTR sampled at one time unit interval. 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 sample of this data set is shown in Figure 6. In this figure, as well as in all subsequent figures showing process inputs, the first input, \( \hat{u}_1 \), is indicated by the solid line, and the second input, \( \hat{u}_2 \), is indicated by the dotted line. A testing set (labelled as testing set 1), distinct from
Table 5 Summary of quantities in the CSTR model

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value or range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau = V_1/F_0 )</td>
<td>1.0</td>
<td>Nominal spacetime of reactor</td>
</tr>
<tr>
<td>( D_0 = k_g V_1/F_0 )</td>
<td>0.11</td>
<td>Damköhler number</td>
</tr>
<tr>
<td>( \gamma = E/R T_0 )</td>
<td>20</td>
<td>Activation energy</td>
</tr>
<tr>
<td>( R = \exp (-\Delta H_{\text{CT}}/RT_0) )</td>
<td>7.0</td>
<td>Heat of reaction</td>
</tr>
<tr>
<td>( \Phi = \frac{b A_0}{F_0 \theta C_f} )</td>
<td>0.5</td>
<td>Heat transfer coefficient</td>
</tr>
<tr>
<td>( F_v )</td>
<td>1.0</td>
<td>Nominal feed flow</td>
</tr>
<tr>
<td>( T_v )</td>
<td>300</td>
<td>Nominal feed temperature</td>
</tr>
<tr>
<td>( c_{A0} )</td>
<td>1.0</td>
<td>Nominal feed concentration</td>
</tr>
</tbody>
</table>

Inputs (control variables)

\( u_1 = \frac{T_v - T_0}{T_v} \) : \( T_v \in [250, 350] \) Coolant temperature  
\( u_2 = \frac{F - F_v}{F} \) : \( F \in [0.5, 1.5] \) Input feed rate

Scaled inputs

\( \bar{u}_1 = (u_1 - 500)/200 \) : \( \bar{u}_1 \in [-1.1] \) Scaled coolant temperature  
\( \bar{u}_2 = (u_2 - 1)/0.5 \) : \( \bar{u}_2 \in [-1.1] \) Scaled input feed rate

Outputs (measured variables)

\( y_1 = \frac{c_{A0} - c_A}{c_{A0}} \) : Outflow concentration  
\( y_2 = \frac{T_v - T_0}{T_v} \) : Outflow temperature

Scaled outputs

\( \bar{y}_1 = \frac{c_{A0} - 0.755}{0.65} \) : \( \bar{y}_1 \in [-1.1] \) Scaled outflow concentration  
\( \bar{y}_2 = \frac{T_v - 317.1}{65} \) : \( \bar{y}_2 \in [-1.1] \) Scaled outflow temperature

Disturbances (noise)

\( d_1 = \frac{T_v - T_0}{T_v} \) : \( T_v \in [295, 305] \) Feed temperature  
\( \bar{d}_2 = \frac{c_{A0} - c_A}{c_{A0}} \) : \( \bar{d}_2 \in [0.9, 1.1] \) Feed concentration

Figure 6 This graph shows a sample of the input and output data used for training of the ANN models. Note that the inputs are randomly distributed both in magnitude and duration. The samples were taken at one time step intervals. In this graph, as well as in all subsequent ones, \( \bar{u}_i \) is the solid line and \( \bar{u}_i \) is the dotted line for the inputs

the training set but similarly determined, also consisted of 1000 data points. A second testing set (labelled as testing set 2), consisting of 450 data points, consisted of individual step changes in each of the two inputs. Four models were created based on these data sets: a contin-

uous linear model with first-order elements, a nonlinear ANR-ARMA model of the type found in the literature, and shown in Figure 1, a nonlinear ANN-CT model of the architecture described above, and a nonlinear ANN-DT model having the structure discussed in the previous section. 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.

Modelling results

Linear continuous models. Using the MIDENT program of the CONS YD package, a continuous transfer 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 (four steady-state gains, four time constants and four time delays).

\[
G(s) = \frac{-1.104s^{-0.0018s}}{8.24s+1} + \frac{0.265s^{-0.018s}}{10.07s+1} + \frac{0.986s^{-0.0038s}}{8.24s+1} - \frac{0.136s^{-0.056s}}{0.91s+1}
\]  

The performance of this model is shown in Figure 7 and represents the performance with which the MANIDENT network discussed below was initialized. As can be seen, this type of model was unable to account correctly 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 it 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 probably the result of the
Efficient nonlinear neural network process models: G. M. Scott and W. H. Ray

Figure 8  Learning curve of the three-layer (nine units in hidden layer) ANN-ARMA network used for dynamic modelling of the CSTR. Curve 1 is the error on the first testing set and curve 2 is the error on the second testing set.

Figure 9  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, $\tilde{u}_1$ is represented by the solid line and $\tilde{u}_2$ is represented by the dotted line.

Identification software attempting to find the best fit linear model to a nonlinear process.

ANN-ARMA model. For a nonlinear ARMA network model, a three-layer recurrent feed-forward network as shown in Figure 1 was configured. The initial weights were chosen as small random numbers. For comparison purposes, networks containing three to 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 ANN-CT and ANN-DT MANNIDENT models. Only the results from the network with nine hidden units are shown in Figures 8 and 9, but the results from all the networks are summarized in Table 6.

Figure 8 shows the learning curve of the ANN-ARMA network. The learning curve shows the mean square error on the training set and the testing sets plotted against the number of epochs of training, where one epoch is one complete presentation of the training set. In this graph, the solid line represents the performance on the training set, while the dotted lines represent the performance on the testing sets. The first testing set (labelled 1 in the graph) has similar distribution characteristics to the training set, while the second testing set (labelled 2 in the graph) represents regular step changes in the inputs. As can be seen, the curve shows that learning was initially quite rapid, and slowed as training progressed. Also note that as training progressed past 30 epochs, the performance on the training set continued to improve, while the testing performance on the first set did not. In the case of the second testing set, the error also began to increase. This is an indication that the network was beginning to 'memorize' the training data at the expense of being able to generalize. Since both the training set and the first testing set had similar characteristics (i.e. they were generated in the same manner), it is expected that if the network is generalizing properly these two curves should be parallel. Increasing the number of hidden units beyond nine improved the performance on the training set, but actually degraded the testing set performance, thus again indicating memorization of the training data because of too many parameters for the network to learn.

Figure 9 shows the performance of the model on the second testing set of data for the network with nine hidden units. Note that this model was able to represent the different gains resulting from positive and negative steps in the input. However, the model did seem to show a slight but consistent offset for all values.

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 transfer function of the process as given in Equation (8). This model was then used to create and initialize a network with the structure depicted in Figure 4. Figure 10 shows the learning curve for this network. In contrast to the learning curve for the ANN-ARMA model (Figure 8), the learning curve for this model showed both the errors on the training set and the testing set decreasing at approximately the same rate. This indicates that, unlike the ANN-ARMA model, this model continued to improve its generalization ability especially after the network was pruned after 12 epochs. Figure 11 shows its performance on the test set. Again, this network model, like the ANN-ARMA model, was able to represent the changing gains for positive and negative steps. Also, statistical analyses using a t-test show that this model had significantly less error than the ANN-ARMA model on both test sets (99.99% confidence). The improvement on the training set was not as significant (90% confidence).

The use of additional hidden units (increasing the size of the $e(n-1)$ and $x(n)$ layers) did not improve the performance of the network, but did in fact slow down the learning process. These additional 'state' units were not initialized with any model information, but instead
Table 6  Comparison of final error values of ANN models and traditional models. 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

<table>
<thead>
<tr>
<th>Model</th>
<th>Training set error</th>
<th>Testing Set I error</th>
<th>Testing Set II error</th>
<th>Model size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous transfer function models (linear)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>0.05387</td>
<td>0.05790</td>
<td>0.01426</td>
<td>12</td>
</tr>
<tr>
<td>1st/2nd</td>
<td>0.02764</td>
<td>0.03165</td>
<td>0.01747</td>
<td>20</td>
</tr>
<tr>
<td>ANN-ARMA models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td>0.00178 ± 0.000075</td>
<td>0.00356 ± 0.000141</td>
<td>0.00222 ± 0.000242</td>
<td>35</td>
</tr>
<tr>
<td>(4)</td>
<td>0.00175 ± 0.000081</td>
<td>0.00330 ± 0.000157</td>
<td>0.00188 ± 0.000161</td>
<td>46</td>
</tr>
<tr>
<td>(5)</td>
<td>0.00175 ± 0.000059</td>
<td>0.00316 ± 0.000114</td>
<td>0.00186 ± 0.000132</td>
<td>57</td>
</tr>
<tr>
<td>(7)</td>
<td>0.00177 ± 0.000086</td>
<td>0.00303 ± 0.000119</td>
<td>0.00139 ± 0.000152</td>
<td>79</td>
</tr>
<tr>
<td>(9)</td>
<td>0.00182 ± 0.000066</td>
<td>0.00287 ± 0.000102</td>
<td>0.00111 ± 0.000137</td>
<td>101</td>
</tr>
<tr>
<td>ANN-CT models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpruned</td>
<td>0.00222 ± 0.000001</td>
<td>0.00212 ± 0.000144</td>
<td>0.00021 ± 0.000002</td>
<td>40</td>
</tr>
<tr>
<td>Pruned</td>
<td>0.00168 ± 0.000004</td>
<td>0.00218 ± 0.000014</td>
<td>0.00032 ± 0.000003</td>
<td>40</td>
</tr>
<tr>
<td>ANN-DT models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpruned</td>
<td>0.00222 ± 0.000003</td>
<td>0.00276 ± 0.000003</td>
<td>0.00072 ± 0.000005</td>
<td>46</td>
</tr>
<tr>
<td>Pruned</td>
<td>0.00167 ± 0.000011</td>
<td>0.00214 ± 0.000008</td>
<td>0.00031 ± 0.000005</td>
<td>40</td>
</tr>
</tbody>
</table>

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 was appropriate to this task.

The initialization of all the weights in the ANN-CT network to small random numbers substantially degraded the performance of the network with respect to both the training times and final error values. Furthermore, the variation in the results was greater, as can be expected since there is a greater dependence on the random initialization. Thus, both the architecture and the weight initialization are important to the performance of the ANN-CT model.

**ANN-DT model.** An ANN-DT model of the CSTR was also created. In this case, the network was based on a first-order discrete transfer function model of the process. This resulted in a model that was identical in structure to the ANN-CT model discussed above. Furthermore, the initial weights were also very similar and upon training essentially the same results were obtained\[9\]. Hence these results will not be presented here.

**Discussion**

Table 6 summarizes the performance of all of 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 and ANN-DT models, the random initialization only refers to those weights not initialized with model information. Also given for each of the mean errors are the 95% confidence intervals that 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 wide range of conditions means that a single ANN 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/DT models performed significantly better than both the linear model with first-order element as well as the linear model with first-order-second-order elements (99.99% confidence). A single linear model was not sufficient to adequately model the process over this range. The major drawback of ANN models as compared to linear models is their interpretation (their 'black box' nature). This issue is addressed by our use of approximate linear models for network design and weight initialization, which facilitates their interpretation as discussed below.

Comparisons can also be made between the ANN-ARMA models and the ANN-CT/DT (pruned) models. While the improvement in training errors for the ANN-CT/DT model is not as highly significant (90% confidence for the ANN-ARMA model with seven hidden nodes, 99.5% confidence for the ANN-ARMA model with nine 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/DT 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/DT models showed significantly less variation between runs as compared to the ANN-ARMA models (99.99% confidence). The implication of this is that with the ANN-CT/DT 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. Thus, the ANN-CT/DT models show significant performance improvements over the ANN-ARMA model due to the lesser variability between runs and the ease of architecture determination, so that either the ANN-CT or ANN-NT formulation would be a good model structure for nonlinear modelling.

**Knowledge extraction**

The final step in the modelling process is the interpretation of the trained network. This is important since the ANN is essentially a 'black box' and being able to interpret these results is necessary in order to have confidence in the results. Also, being able to extract 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.

**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:

\[
x(n) = \Phi x(n - 1) + \beta u(n) \\
y(n) = C x(n)
\]

where \( \Phi, \beta \) and \( C \) are constant matrices. These matrices can be seen to take the following values resulting from the sensitivity analysis:

\[
\Phi = \frac{\partial x(n)}{\partial x_{\text{out}}(n - 1)} = f(\sigma) W^{xx} \\
\beta = \frac{\partial x(n)}{\partial u(n)} = f(\sigma) W^{xu} \\
C = \frac{\partial y(n)}{\partial x(n)} = f(\sigma) W^{yx}
\]

which are easily calculated from the weights of the network \( W^{xx}, W^{xu} \) and \( W^{yx} \) and the current activation of the units \( (\sigma^x, \sigma^u) \). It must be noted, however, that in the discrete state space formulation that both \( x_{\text{out}}(n - 1) \) and \( x(n) \) must be of the same size, which is not necessarily the case in the ANN-CT model (as in Figure 3). To manage this, the \( x(n) \) array of the state space model is extended to be the same size as the \( x_{\text{out}}(n - 1) \) array, with the \( \Phi, \beta \) and \( C \) matrices also being suitably resized. The additional elements of the \( \beta \) and \( C \) matrices are given the value of 0, while the additional elements in the \( \Phi \) matrix are given the values of either 0 or 1 to reflect the relationship of the past states represented in the matrix with the current states. The resulting model is in the form of a discrete state space model (a DSC file of the CONS-YD CAD package).

**Approximate linear models**

The above analysis results in a discrete state space model of the process at a particular operating point. The CONS-YD package has several utilities that allow this model to be converted into other forms also useful for modelling. These utilities and their use are described below in general terms. For the theory behind these transformations, Reference 26 is recommended.

*The pulse transfer function model.* The CONS-YD program LAPLACE takes state space models and produces transfer function models. In this case, since the input
model is discrete, a pulse transfer function is produced
(CONSYD file type DTC). This step yields a model of the form:

\[ y(z) = G(z)u(z) \]  

(9)

where \( G(z) \) is the transfer function matrix. Each element of this matrix has the form:

\[ g_{ij}(z) = \frac{P(z)}{Q(z)} \]

where \( P(z) \) and \( Q(z) \) are polynomials in \( z \). The degree of these polynomials depends on both the number of inputs, outputs and states (hidden units) of the model as well as the sparsity of the \( \Phi, \beta \) and \( C \) matrices.

The combined pulse transfer function. Looking at Figure 4, it can be seen that the input to the network, \( \hat{u}(n) \), actually consists of the current input (the first two elements for a two-input system) as well as some past inputs (the remaining elements). For this reason, the pulse transfer function given in the previous paragraph contains a larger number of inputs than the process being modelled. The elements of \( G(z) \) that correspond to the same physical inputs (although some are delayed) can be combined by making use of the relationship that:

\[ u(n-1) = u(n)z^{-1} \]

This allows the various rows of the matrix to be 'folded' together by adding the elements of the rows after multiplying the delayed inputs by an appropriate power of \( z^{-1} \). The resulting model has the same form as given in Equation (9), except that the number of inputs in the model is now the same as the number of inputs to the physical system. The resulting polynomials are, of course, of higher degree.

The continuous transfer function model. The CONS YD program SNARTZ takes pulse transfer function models and produces continuous transfer function models (CONSYD file type CTC). This step yields a model of the form:

\[ y(s) = G(s)u(s) \]  

(10)

where \( G(s) \) is the transfer function matrix. Each element of this matrix has the form:

\[ g_{ij}(s) = \frac{P(s)e^{-i\delta}}{Q(s)} \]

where \( P(s) \) and \( Q(s) \) are polynomials in \( s \) and \( i_\delta \) is a time delay. Again, the degree of the polynomials depends on the size and sparsity of the original discrete state space model.

The reduced transfer function model. The resulting model is often of quite high order. The CONS YD program REDUCE allows the model to be approximated by a model of appropriate lower order. This allows the extracted model to be compared with linearized models of the process or reduced to an order that is more useful for control system design or physical understanding.

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 centre operating point. The method described above resulted in an approximate linear model of the following form around the steady state in Table 5.

\[
G(s) = \begin{bmatrix}
-1.300e^{-0.28s} & (0.030s + 0.451)e^{-0.131s}
0.690s^2 + 2.19s + 1 & 0.033s^2 + 0.727s + 1
(0.049s + 1.226)e^{-0.005s} & (0.021s - 0.257)e^{-0.043s}
0.337s^2 + 1.94s + 1 & 0.012s^2 + 1.42s + 1
\end{bmatrix}
\]  

(11)

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:

\[
G_{\text{exact}}(s) = \begin{bmatrix}
-1.24 & (0.416s + 0.411)
2.21s^2 + 2.85s + 1 & 2.21s^2 + 2.85s + 1
(0.849s + 1.124) & -(0.291s + 0.244)
2.21s^2 + 2.85s + 1 & 2.21s^2 + 2.85s + 1
\end{bmatrix}
\]  

(12)

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 small amplitude step changes in the input is shown in Figure 12. From the figure, it is apparent that...
the response of the extracted model was very similar to
that of the exact linearized model. This is an indication
that the extracted model was a very good 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 ele-
ments consisting of first-order responses with time delay.
The extracted model and reduced linearized model are
shown in Equations (13) and (14), respectively.

\[
G(s) = \begin{bmatrix}
-1.300e^{-0.604s} & 0.451e^{-0.114s} \\
1.59s + 1 & 0.68s + 1 \\
1.22e^{-0.134s} & -0.257e^{-0.02s} \\
1.78s + 1 & 1.45s + 1 \\
\end{bmatrix}
\]  

(13)

\[
G_{\text{excx}}(s) = \begin{bmatrix}
-1.240e^{-0.745s} & 0.411e^{-0.083s} \\
2.20s + 1 & 1.80s + 1 \\
1.124e^{-0.167s} & -0.244e^{-0.030s} \\
1.99s + 1 & 1.64s + 1 \\
\end{bmatrix}
\]  

(14)

A comparison of these two transfer functions shows that
they are in very close agreement, with the single excep-
tion 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. Both were in
fairly good agreement with a model identified using
MIDENT from data involving only local step changes in
the inputs of ±0.1. This model is given as:

\[
G_{\text{local}}(s) = \begin{bmatrix}
-1.590e^{-0.291s} & 0.413e^{-0.150s} \\
2.60s + 1 & 1.68s + 1 \\
1.370e^{-0.005s} & -0.245e^{-0.000s} \\
2.35s + 1 & 1.54s + 1 \\
\end{bmatrix}
\]  

(15)

The comparisons above show that the network model
is a good linear approximation to the process. To explore
the fidelity of the network model over a range of input
amplitudes, plots of the steady state gain as the inputs
are changed are shown in Figures 13 and 14. As can be
seen, the agreement was fairly good over the entire range
of the inputs to the model. Thus the ANN-CT model
captures the nonlinear gain behaviour for the process
and represents the response to dynamic inputs.

More complex nonlinear behaviour of the ANN-CT
model

The ANN-CT structure is also able to model successfully
more complex dynamic behaviour such as multiple
steady states and limit cycle oscillations. This is shown
below in two examples in which nonlinear ANN-CT
models were created and trained on data sets that
showed this complex behaviour. For each example, the
first principles CSTR model was used to generate training
and testing data with which to train the network.

These data sets were again scaled within a range of (−1,
1) to be deviations from a steady state. The parameters
used in the first principles model for each example are
given in Table 7. We used the same procedures to create
the following models as were used for the earlier ANN-
CT/DT models.

Multiple steady state behaviour. The MANNIDENT
approach, starting with a linear model of the CSTR, was
used to construct and train a nonlinear ANN-CT model
of the process. Figure 15 shows the performance of the
model on the second testing set as input changes were
made. Note that this model was able to represent the multiple steady state behaviour of the process much better than a linear model. This can be most easily seen in looking at the ỹ j curve from time 0 to time 160, where the first input was stepped from 0.0 to 0.6 and back to 0.0. Note that concentration does not return to its original value after the step down in the input.

Figure 16 plots the steady state value of the first output, ỹ 1 , against the first input, u i , for the linear model (dotted-dashed line), the ANN-CT model (solid line) and the exact first principles model (dotted line). The network began with the linear model represented by the dotted-dashed line, and through backpropagation training, changed its representation to that given by the solid line. Thus, the trained ANN-CT model is a much closer representation of the process than the initial linear model. This is also shown by the steady state gain curves depicted in Figures 17 and 18 where the multiple steady states are indicated by the multiplicity of gains at some points. In this representation of the trained model, the agreement between the ANN-CT model and the exact first principles model is not exact; however, the network predicts the correct qualitative behaviour, which the linear model did not show at all. It appears that the training set did not have enough input amplitudes in the range (-0.4, 0.4) in order to learn the fine details of multiplicity. With further training with a wider range of control input amplitudes, the steady-state agreement could be made closer.

Limit cycle oscillations. As before, the MANNIDENT method was used to create a nonlinear ANN-CT model using a time series of input-output data and a first-order continuous linear model. Figure 19 shows the performance of the linear model over the series of input step changes in the second testing set. This type of model was
Application to control

The nonlinear models obtained from the ANN-CT structure were put to the task of controlling the CSTR. Two controlled schedules were used for testing the controllers. The first involved two setpoint changes at three different places in the operating range of the CSTR, which were chosen because of their varying characteristics of the process. In Figure 13, it can be seen that the gain of the process passed through a maximum at one point, and also the sign of the $(2, 2)$-element changed. These were two places that are used for the control task. The third was the ‘nominal’ location: That is, the centre of the range. At each of these places, steps were made in each of the two setpoints. The second schedule involved disturbances. The controllers were required to maintain the process at each of the three setpoints described above when the two disturbances were independently stepped. In all the examples in this section, the reference trajectory was a first-order delay with a time constant of 2 time units. It was from this trajectory that all error values are measured.

For comparison purposes, two extremes of controller design were also investigated and are reported here. First, for the exact inverse controller, the differential equations describing the process were exactly solved for the control action. This would represent the ‘best’ controller since it is based on the first principles differential equations of the system. First it must be noted that the reference trajectories (desired response to setpoint changes) are given by the differential equations:

$$\theta_1 \frac{dy_1}{dt} + y_1 = y_{d1}$$
$$\theta_2 \frac{dy_2}{dt} + y_2 = y_{d2}$$

where $y_{d1}$ and $y_{d2}$ are the desired setpoints and $\theta_1$ and $\theta_2$ are the time constants of the desired response to the setpoints. Rearranging leads to the following equations for the derivatives:

$$\frac{dx_1}{dt} = \frac{dx_1}{dt} - \frac{1}{\theta_1} (y_{d1} - y_1) = \frac{1}{\theta_1} \epsilon_i$$
$$\frac{dx_2}{dt} = \frac{dx_2}{dt} - \frac{1}{\theta_2} (y_{d2} - y_2) = \frac{1}{\theta_2} \epsilon_2$$

where $\epsilon_i$ and $\epsilon_2$ are the error signals. Integrating these two expressions leads to the following expressions for $x_1$ and $x_2$:

$$x_1 = x_{i0} + \frac{1}{\theta_1} \int_0^t \epsilon_1 dt = I_1$$
$$x_2 = x_{i0} + \frac{1}{\theta_2} \int_0^t \epsilon_2 dt = I_2$$

Substituting the above equations into Equations (6) and
(7) and solving for the process inputs (control actions) yields the nonlinear direct synthesis controller equations:

\[ u_2 = -\frac{\frac{1}{\beta} c_1}{L} \frac{Q}{I_1} - 1 \]  

(16)

\[ u_1 = \frac{\frac{1}{\beta} c_1 + (u_2 + 1) I_2 - \frac{B}{\beta}}{L} + I_2 \]  

(17)

where \( I_1 \) and \( I_2 \) are the integrals given above and:

\[ Q = \frac{\alpha(1 - I_1)}{I_1} \exp\left(\frac{\gamma I_2}{I_2 + 1}\right) \]

The second controller design tested was the case when the control inputs are simply stepped to the new steady state, so that it is in some sense a ‘worst case’ controller. Also investigated for comparison was the performance of a fixed PI controller and linear direct synthesis and IMC controllers \(^{28}\). Table 8 summarizes the integral square error measurements for each of these controllers.

To compare with these standard benchmark controllers, an ANN-C1 network was used as the process model for the design of a nonlinear direct synthesis controller. Figure 21 shows the performance of this controller for a schedule of both setpoint tracking and disturbance rejection tasks. Table 8 reports the error for the ANN-DSC controller as well as for the benchmark cases. Note that the nonlinear ANN-DSC controller is considerably better than the linear controller and almost as good as the ‘exact’ DSC controller. More extensive control studies with ANN models and the details of ANN model inversion will be the topic of a companion paper.

Conclusions

The MANNIDENT algorithm for network design and initial weight specification significantly improved the performance of the nonlinear models in the following ways:

- Improved accuracy
- Improved generalization to testing data
- Less variability between runs
- Reduced training time

The MANNIDENT algorithm quickly determines a relevant network architecture without resorting the trial-and-error method, and through initialization of the weights with prior information, gives the learning algorithm an appropriate direction in which to continue learning. Finally, since the units and some of the weights initially had understandable meanings, the MANNIDENT networks were easier to interpret after training, allowing classical models to be extracted from the network.

For the nonlinear CSTR example, the ANN-CT network was able to model multiple steady state and limit cycle oscillations when the training data set contained these phenomena.

Finally, a nonlinear direct synthesis controller, which used the ANN-CT model, showed excellent performance in the face of setpoint changes and process disturbances.

References

Efficient nonlinear neural network process models: G. M. Scott and W. H. Ray

12 Caudill, M. AI Expert, 1988, 3(6): 53
16 Werbos, P. J. in ‘International Conference on Neural Networks’, Vol. 1, IEEE, San Diego, CA, 1988, 343

Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN notation</td>
<td></td>
</tr>
<tr>
<td>$A_{in}, B_{in}$</td>
<td>$z$-transfer function coefficients</td>
</tr>
<tr>
<td>$A, B, C$</td>
<td>time domain model coefficients</td>
</tr>
<tr>
<td>$f(\cdot)$</td>
<td>network threshold function</td>
</tr>
<tr>
<td>$i, j, k$</td>
<td>subscript indices</td>
</tr>
<tr>
<td>$G(\cdot), I_{ij}$</td>
<td>transfer function models</td>
</tr>
<tr>
<td>$K_m, K_p, K_i$</td>
<td>linear model parameters</td>
</tr>
<tr>
<td>$M_d, N_d$</td>
<td>degree of $z$ polynomials</td>
</tr>
<tr>
<td>$m_{in}, n_{in}$</td>
<td>discrete time delay</td>
</tr>
<tr>
<td>$N_{in}, N_{in}$</td>
<td>number of process inputs and outputs</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>sizes of network layers</td>
</tr>
<tr>
<td>$u$</td>
<td>discretization time step of model</td>
</tr>
<tr>
<td>$\hat{u}_{out}$</td>
<td>linear model inputs</td>
</tr>
<tr>
<td>$\mathbf{W}^{\sigma}$</td>
<td>network inputs</td>
</tr>
<tr>
<td>$x_{\sigma}$</td>
<td>scaled deviation inputs</td>
</tr>
<tr>
<td>$\mathbf{x}$</td>
<td>weight matrix from $\mathbf{x}$ to $\mathbf{y}$</td>
</tr>
<tr>
<td>$\mathbf{X}, \mathbf{X}_{in}$</td>
<td>linear model states</td>
</tr>
<tr>
<td>$\mathbf{y}$</td>
<td>network states (hidden units)</td>
</tr>
<tr>
<td>$\mathbf{y}_{out}$</td>
<td>linear model outputs</td>
</tr>
<tr>
<td>$\mathbf{y}$</td>
<td>network outputs</td>
</tr>
<tr>
<td>$\delta$</td>
<td>scaled deviation outputs</td>
</tr>
<tr>
<td>$y^*$</td>
<td>network output targets</td>
</tr>
<tr>
<td>$\phi, \beta, C$</td>
<td>controller error</td>
</tr>
<tr>
<td>$\mathbf{e}$</td>
<td>discrete state space parameters</td>
</tr>
<tr>
<td>$\psi(\cdot)$</td>
<td>unit summation function</td>
</tr>
<tr>
<td>$\sigma_i$</td>
<td>total input to unit</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>unit bias value</td>
</tr>
</tbody>
</table>

CSTR model nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>heat transfer area</td>
</tr>
<tr>
<td>$c_{in}$</td>
<td>outlet concentration</td>
</tr>
<tr>
<td>$c_0$</td>
<td>feed concentration</td>
</tr>
<tr>
<td>$c_0$</td>
<td>nominal feed concentration</td>
</tr>
<tr>
<td>$C_0$</td>
<td>heat capacity</td>
</tr>
<tr>
<td>$E$</td>
<td>reaction activation energy</td>
</tr>
<tr>
<td>$F$</td>
<td>volumetric feed rate</td>
</tr>
<tr>
<td>$p_{in}$</td>
<td>nominal feed rate</td>
</tr>
<tr>
<td>$h$</td>
<td>heat transfer coefficient</td>
</tr>
<tr>
<td>$H$</td>
<td>heat of reaction</td>
</tr>
<tr>
<td>$k_0$</td>
<td>kinetic pre-exponential factor</td>
</tr>
<tr>
<td>$\rho$</td>
<td>gas constant</td>
</tr>
<tr>
<td>$T$</td>
<td>time</td>
</tr>
<tr>
<td>$T_1$</td>
<td>feed temperature</td>
</tr>
<tr>
<td>$T_2$</td>
<td>coolant temperature</td>
</tr>
<tr>
<td>$T_0$</td>
<td>nominal feed temperature</td>
</tr>
<tr>
<td>$V$</td>
<td>reactor volume</td>
</tr>
<tr>
<td>$\rho$</td>
<td>density</td>
</tr>
</tbody>
</table>