16 Fault Detection and Process Control

learning objectives:

- complex processes: priorities and problems involved in identifying malfunctions, and in controlling conditions to achieve constant results
- comparison of classical methods and neural networks
- one danger of careless choice of data representation: introducing spurious correlations into the dataset; one way to avoid this
- special requirements of time-dependent data, and one technique (the "moving window") for dealing with them
- how the counter-propagation network behaves as a self-organizing lookup table
- example of fault detection in a catalytic conversion reactor
- comparison of results from back-propagation and counter-propagation networks
- relationship between **forward** model (input $\rightarrow$ results) and **inverse** model (results $\rightarrow$ input)

16.1 The Problems

Two basic problems have to be faced in running chemical processes; first, the recognition and detection of faults, and second, control of the process itself. Fault detection is regarded mainly as qualitative, since it is necessary only to identify the defective part(s) in the system. On the other hand, controlling the process is quantitative, since the values of process variables must be continuously monitored and maintained within prescribed limits. This information is, of course, a prerequisite for detecting (and identifying) system faults.

Immediate recognition of faults prevents loss of time, of production quotas, or even of the equipment. Therefore, a permanent on-line diagnosis based on the process variables is needed in addition to equipment testing and preventive maintenance, operator training, etc.

Control of the process requires feedback calculated on the basis of a model believed to govern the entire process. The better the model, the more reliable are the calculated responses and the smoother is the process. The model should be set up either on the basis of the known chemical and technological subprocesses, or learned on the basis of past monitoring.

There may be many sources of difficulty in achieving fault prediction and/or process control based on continuous monitoring of the process variables:

- reliable models that yield accurate behavior of the process under all circumstances are often not known
- the dependence of the behavior of the process on the controlling variables is in most cases nonlinear
- the data monitored as a basis for the feedback can be noisy or uncertain
- there exists no one-to-one correspondence between the set of observed symptoms and the correct diagnosis.

Since it is important to know as much as possible about the behavior of the process, especially under unusual conditions, there has been considerable research into the theory of process control. Different approaches to the problem have been tried:

- statistical analyses
- linear models using classical techniques
- pattern recognition
- knowledge (rule) based expert systems
- neural networks

The quality and accuracy of predictions based on rule-based expert systems strongly depend on the extent and quality of the rule database, which must be painstakingly collected from human experts, a time-consuming and expensive process. Therefore, neural network techniques in general and the back-propagation algorithm in particular have stirred enormous interest among chemical engineers.

Some of the more important advantages of neural networks over classical approaches are believed to be that:

- neural networks can learn from examples, making analytical models unnecessary;
- the handling of nonlinear models is easy for neural networks, because of their inherently nonlinear response;
- neural networks can perform association, i.e., they can handle incomplete or slightly corrupted data, which is important for good modeling outside the trained regions,
- neural networks can handle continuous variables as well as discrete ones (in rule-based expert systems, discrete variables are usually used;
- neural networks can model inverse functions, an important feature for generating feedback control systems.

Hoskins and Himmelblau (Reference 16-1) have discussed the applications of neural networks in various areas of chemical process engineering, such as fault detection, diagnosis, process control, process design, and process simulation. Sometimes it is difficult to clearly separate one task from another, but we will try to summarize them in this chapter.

16.2 The Data

A chemical process, whether on plant or pilot scale, is monitored by measuring a number of variables, x_i, at different points of the process, and supervised by feedback signals to various controllers. The task of a *controller* is to keep the measured variables in a given state (on/off), or at a defined level of a given variable (the *set-point*), or within a defined interval. These variables may include flow-rate, temperature, concentration, pressure, liquid level, etc. The controllers supervise heaters, pumps, valves, stirring devices, etc.

The assembly of all measured variables and states at a given time t, x_{it}, is called the process vector $\boldsymbol{P}_t$:

$$\boldsymbol{P}_t = (x_{1t}, x_{2t}, \dots, x_{it}, \dots, x_{mt}) \tag{16.1}$$

When teaching the neural network to recognize and to predict faults in a chemical process, the training input vectors $\boldsymbol{X}$ are sets of measured variables known to result in a "smooth" or "no-fault" process, along with sets that describe faulty conditions. For each of the latter, a vector describing the faulty state(s) must be given as a target vector $\boldsymbol{Y}$ that pinpoints the faults caused by that particular $\boldsymbol{X}$.

Quite often, the causes are binary or discrete states, such as presence or absence of fluid flow (pump working/not working), etc. The cause of a malfunction can be, of course, a combination of faulty states.

The best way to handle multiple discrete states of a variable is to transform it into a *distributed representation*, i.e., to transform it into as many binary variables as there are different states for the original discrete variable.

For example, suppose that a variable x_v called "state of the valve" can have one of three possible states: "left pipeline", "shut", and "right pipeline" (Figure 16-1). The process vector $\boldsymbol{X}$ should have **three** new sub-variables to describe these three states as 0 or 1:

$$\boldsymbol{X} = \left(x^{left}, x^{shut}, x^{right}, \dots\right) \tag{16.2}$$

Then for given values of the other variables, **X** may be:

$$\begin{aligned}
\boldsymbol{X}_i &= (1, 0, 0, \dots) && \text{valve open to the left pipeline} \\
\boldsymbol{X}_j &= (0, 1, 0, \dots) && \text{valve shut} \\
\boldsymbol{X}_k &= (0, 0, 1, \dots) && \text{valve open to the right pipeline}
\end{aligned} \tag{16.3}$$

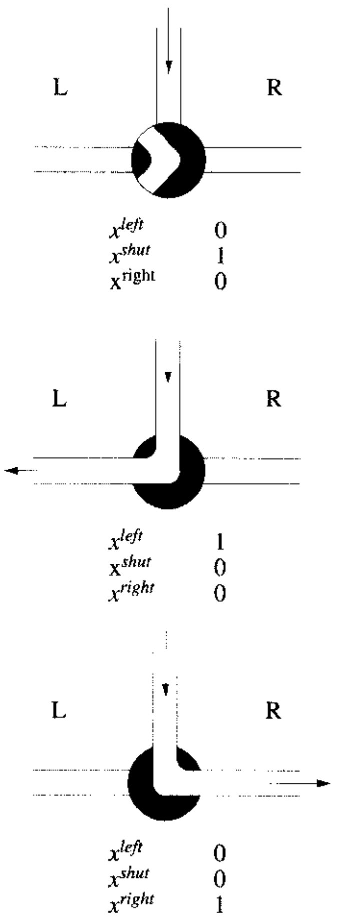

Figure 16-1: Description of three states of a valve and the corresponding value of the process vector.

Obviously, any other combination, such as $\mathbf{X} = (1, 1, 0, \dots)$ or $(0, 1, 1, \dots)$ corresponds to impossible states. It might seem a waste to use three variables and then throw away six of their nine possible states; perhaps a new variable having three values, +1, 0 and −1, would be more efficient.

This representation can of course be used, but it has some unintended side effects. The sequence of values (+1, 0, −1) implies that the states are **not equivalent**: the state "shut" (0) is closer to both "open" states than these two states are to each other. Such *unjustified* *correlation* among the equivalent states, if included in the learning procedure, may have an undesired influence on the results. The more equivalent discrete states a variable has, the worse is the situation when the condensed distribution is used. A very similar case of unwanted correlation among the states of one variable is discussed in Chapter 17, where coding of proteins by amino acids is described. The situation is solved exactly as described here: each multistate variable is replaced by as many binary inputs as the variable has different states.

In order to predict how the process variable x_i will vary with time, **at least two of its values** in each input vector of the training set must correspond to **consecutive times**, e.g., x_{it} and x_{it+1}. (In terms of the moving window technique described in Section 9.5, the past horizon of the training vector must be at least two events long.) The same variable x_i should also appear in the target vector $\boldsymbol{Y}$ with the value corresponding to the time $t + 2$. As usual, the future horizon of the training vector, or target, can be only one event long.

However, the future horizon during **prediction** may be longer than one, if predictions are to be made several time units ahead. However, it is advisable to keep the horizons only a few time intervals long, because there is always noise in the measured data. **Usually, the past horizon is at least two times longer than the future horizon.**

In order to give you a feeling for how the training vectors are composed, a sequence of training vectors with a past horizon of length 3 and a future horizon of length 1 is given in Table 16-1. Each process vector $\boldsymbol{P}_t = (x_{1t}, x_{2t}, ..., x_{it}, ..., x_{mt})$ (Equation (16.1)) has the same form. All inputs from $t = 0$ to $t = r - 4$ form **one epoch**.

In many cases, especially if the model will be used for quantitative control, it is not necessary that **all** process vectors $\boldsymbol{P}_t$ in the training vector contain **all** process variables $x_1, x_2, ..., x_i, ..., x_m$; only those variables that must be **controlled** and those that can be **manipulated** by the operators are mandatory. Although it may be desirable to include other variables that influence the process, one must be careful not to inject too much redundancy into the input data.

If, for example, the controlled variable is xc and the variable that can be manipulated by the operator is xm, the training vector should look something like this (Figure 16.2):

$$\boldsymbol{X}_t = (..., xc_t, xm_t, ..., xc_{t+1}, xm_{t+1}, ..., xc_{t+2}, xm_{t+2}, ...) \quad (16.4)$$

time t	input X_t	target Y_t
0	(P_0, P_1, P_2)	(P_3)
1	(P_1, P_2, P_3)	(P_4)
2	(P_2, P_3, P_4)	(P_5)
	....	
$t-2$	(P_{t-2}, P_{t-1}, P_t)	(P_{t+1})
$t-1$	(P_{t-1}, P_t, P_{t+1})	(P_{t+2})
t	(P_t, P_{t+1}, P_{t+2})	(P_{t+3})
$t+1$	$(P_{t+1}, P_{t+2}, P_{t+3})$	(P_{t+4})
	....	
$r-5$	$(P_{r-5}, P_{r-4}, P_{r-3})$	(P_{r-2})
$r-4$	$(P_{r-4}, P_{r-3}, P_{r-2})$	(P_{r-1})
	— new epoch —	
$r-3$	(P_0, P_1, P_2)	(P_3)
$r-2$	(P_1, P_2, P_3)	(P_4)
$r-1$	(P_2, P_3, P_4)	(P_5)
r	(P_3, P_4, P_5)	(P_6)
$r+1$	(P_4, P_5, P_6)	(P_7)
	....	

Table 16-1: Data sequence in a moving window learning scheme; the historical database contains r process vectors, P_0 to P_{r-1}.

while as target, only the manipulated variable *xm* can be given:

$$Y_t = (xm_{t+3}) \tag{16.5}$$

The "future" values of the variables should be obtained from either a historical database, or calculated by a theoretical model.

16.3 The Methods

Until now, all applications of neural networks in chemical engineering have used the back-propagation algorithm, because fault detection, modeling of processes, and feedback control applications require supervised learning.

The size of a network is mainly influenced by the numbers of input and output variables. All those back-propagation neural networks described in the chemical engineering literature are small compared to those used in some other applications, such as spectrum-structure correlations, or the determination of the secondary structures of

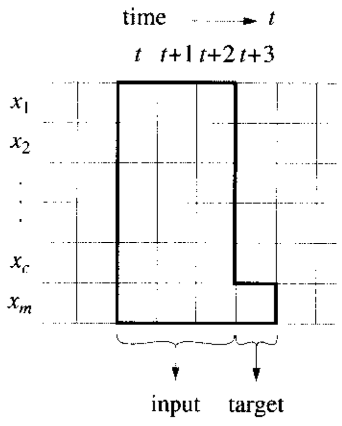

Figure 16-2: The shape of the moving window can have various forms; here, a window is shown with only one variable from the future horizon.

proteins. The number of input and output parameters in chemical processes are a few tens at most, so there is no need for more neurons in the hidden and output layers.

However, the learning times can be quite long in spite of the small sizes of the networks, especially when models are sought, because a large number of different input vectors must cover the problem space quite meticulously if the model is to be reliable.

We believe that the counter-propagation architecture (see Chapter 7) offers attractive opportunities in process control applications of neural networks. Two layers of neurons are combined in a counter-propagation network: a Kohonen layer influenced by the input vectors, and an output layer influenced by the targets. The counter-propagation network acts like a self-organizing lookup table (Section 7.2): all required answers, i.e., the responses for different variables, are pre-calculated and stored in as many lookup tables as there are output variables.

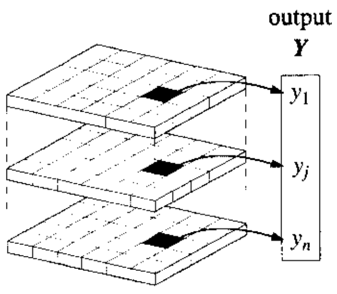

Figure 16-3: All components of the output vector, i.e., the multivariable responses, are stored in all n lookup tables at corresponding locations.

These lookup tables are of the same size and aligned one upon the other; hence, although the complete n-variable output vector $\boldsymbol{Y}$ (y_1, y_2, ..., y_n) is stored in n different lookup tables, all answers y_1, y_2, ..., y_n are stored in a hypercolumn that links **all** n lookup tables at the corresponding row and column (Figure 16-3).

16.4 Predictions of Faults

This example follows the work by K. Watanabe, and coworkers (Reference 16-3), who investigated the catalytic conversion of *heptane* into *toluene* occurring in a reactor:

$$C_7H_{16} \longrightarrow C_7H_8 + 4\,H_2 \tag{16.6}$$

Three variables were measured: the outlet concentration of the product *toluene*, c_p, the heater outlet temperature T_h, and the output signal s_h of the controller that regulates the temperature T_h in the reactor; five different faults can be deduced from these three measured variables. The simplified process scheme is shown in Figure 16-4.

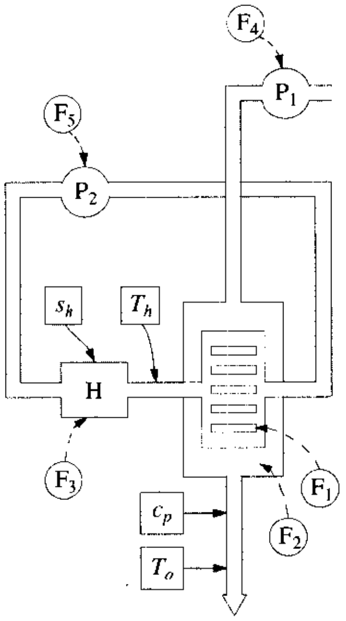

Figure 16-4: The catalytic reactor for the conversion of *heptane* into *toluene*.

A constant flow of *heptane* is maintained by pump no. 1. The temperature in the reactor is sustained by a heater operated by no-fault controller and the pump (no. 2) that cycles steam between the reactor and the heater. *Toluene* leaves the reactor with an outlet concentration of c_p and with the temperature T_o; depending on T_o, the controller adjusts the signal s_h that goes to the heater.

The input value of each of the three variables is normalized to its value in the steady state condition. Therefore, all input values are within an interval between 0.75 and 1.33, with 1.00 corresponding to the : "no-fault" condition.

The five possible faults to be determined from the above three variables are:

1: deterioration of the catalyst

2: fouling of the heat exchanger in the reactor

3: fouling of the heat exchanger in the heater

4: malfunction of pump no. 1

5: malfunction of pump no. 2

Faults 4 and 5 also include the clogging of the pipes.

These faults correspond to the output values y_1 to y_5. They are represented in the learning procedure in two ways; first, they are used as discrete binary variables (yes/no); second, when a fault is confirmed, its severity is represented as a discrete variable having five different values: 0.5, 1, 2, 3, and 4, which requires five output (yes/no) neurons.

To handle this fault diagnosis scheme, they set up six different networks (Figure 16-5): one at the top level, determining which faults have occurred (Figure 16-6), and five networks one level below, each of which indicates the severity of each of the five faults. Each of the five outputs describes a higher degree of deterioration; the first level (0.5) means close to normal.

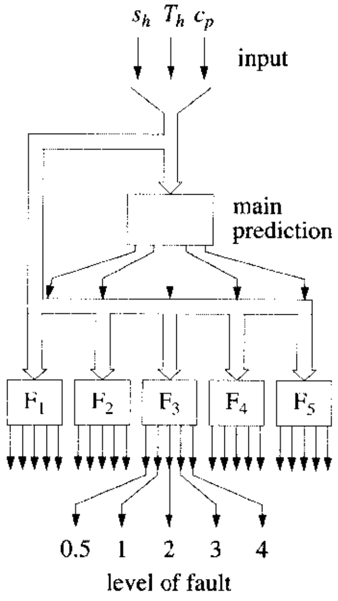

Figure 16-5: Six single-hidden-layer neural networks for the prediction of five four-level faults from three input variables.

To make matters simple, the architectures of all six neural networks are identical: they all have three input nodes, four neurons on the hidden layer, and five output neurons. In the training procedure, the learning rate η is set equal to 0.1, and the momentum μ to 0.9 (Equation (8.1) given in Sections 8.3 and 8.7). The input and output vectors used in training for fault recognition (top level network) are shown in Table 16-2.

The results of fault detection in the catalytic process were rather encouraging: the system of six networks is able to detect the five faults correctly. However, as Table 16-3 shows, the determination of the **exact level** of fault is not as clear-cut. Nevertheless, if we take the largest value of the output as the degree of fault, the correct answer is always obtained.

fault	input X			output Y				
	x_1 s_h [mV]	x_2 T_h [K]	x_3 c_p [gmol/m^3]	y_1	y_2	y_3	y_4	y_5
1	219	885	498	1	0	0	0	0
2	240	906	524	0	1	0	0	0
3	248	889	524	0	0	1	0	0
4	212	878	550	0	0	0	1	0
5	201	889	524	0	0	0	0	1
normal	223	889	524	0	0	0	0	0

Table 16-2: Data for training the first level recognition of faults: yes/no (Watanabe et al., AIChE Journal 1989).

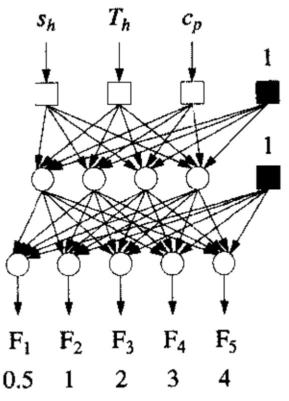

Figure 16-6: One-hidden layer (3 x 4 x 5) neural network for the prediction of faults. The same (3 x 4 x 5) architecture is used for five second-level networks, each predicting the severity of the corresponding fault.

Where totals are less than 100%, it is because the **first** network misidentified the fault. The results are obtained from 1000 test vectors. (Each row of Table 16-3 represents a separate test input. When a fault of type 1 at level 0.5 was input, the system identified it correctly 84% of the time; 4% of the time, it identified the fault as type 1, level 1; and 12% (100–88) of the time, it misclassified the fault type altogether.)

16.5 Modeling and Controlling a Continuously Stirred Tank Reactor (CSTR)

As described earlier, two things are required in order to control a given process reliably. First, we need a model M able to predict the critical parameters of the process for a few time intervals in the future. Second, it must be possible to determine adjustments to the correction variable(s), i.e., the variable(s) that can be manipulated, from the predicted data accurately enough so that the system will return to normal if these adjustments are applied. In other words, *a model* M^{-1} *inverse to the initial model* M *must be obtained.*

This section discusses the control of a nonisothermal continuously stirred tank reactor (CSTR). We will first follow it as it was originally described by a group from the Department of Chemical Engineering at the University of Pennsylvania (see Reference 16-11).

test vector		predicted level of fault in %					
fault	level	0.5	1	2	3	4	total
1	0.5	84	4	0	0	0	88
	1	9	63	13	0	0	85
	2	0	12	75	13	0	100
	3	0	0	3	73	3	79
	4	0	0	0	2	85	87
2	0.5	83	3	0	0	0	86
	1	8	63	17	0	0	88
	2	0	22	63	10	1	96
	3	0	0	3	49	18	70
	4	0	0	0	16	70	86
3	0.5	98	0	0	0	0	98
	1	2	77	18	0	0	97
	2	0	22	67	11	0	100
	3	0	0	1	59	15	75
	4	0	0	0	13	70	83
4	0.5	95	0	0	0	0	95
	1	2	74	8	0	0	84
	2	0	10	81	3	0	94
	3	0	0	3	86	0	89
	4	0	0	0	2	92	92
5	0.5	91	0	0	0	0	91
	1	0	98	0	0	0	98
	2	0	1	80	7	0	88
	3	0	0	2	82	0	84
	4	0	0	0	0	100	100

Table 16-3: Ability of the system of five networks to predict the type and level of fault.

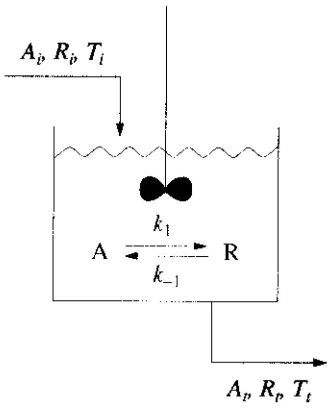

Figure 16-7: Nonisothermal continuously stirred reactor. A_i and R_i are the inlet concentrations of the reactants, and T_i is the corresponding temperature. A_t, R_t, and T_t are the same variables at time t.

These authors used the back-propagation algorithm; we will show that the **counter-propagation** approach can yield comparably good results, and some additional information as well.

A first-order reversible reaction:

$$\mathrm{A} \rightleftarrows \mathrm{R}$$

occurs in the reactor, which is shown schematically in Figure 16-7.

A long, consecutive history of process data must be available in real-world applications to set up a model M and its inverse model M^{-1}. In the present case, a long sequence of data triplets measured at equal time intervals t is required: the concentration of the starting material, A_t, the concentration of the product, R_t, and the reaction temperature, T_t. These measurements should be influenced by as many different circumstances as can possibly occur during the process, for example, sudden fluctuations in the concentrations A or R, and/or changes of temperature T.

For simple processes, a reliable theoretical model can be found. In this example, the chemical process can be described by the following equations:

$$\frac{dA_t}{dt} = \frac{A_i - A_t}{\tau} - k_1 A_t + k_{-1} R_t$$

$$\frac{dR_t}{dt} = \frac{R_i - R_t}{\tau} - k_1 A_t + k_{-1} R_t$$

$$\frac{dT_t}{dt} = \frac{T_i - T_t}{\tau} - H(k_1 A_t + k_{-1} R_t) \qquad (16.7)$$

$$k_1 = Be^{-\frac{C}{T}}$$

$$k_{-1} = De^{-\frac{E}{T}}$$

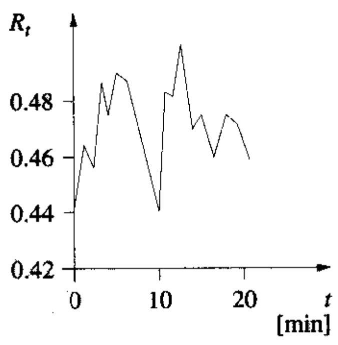

Figure 16-8: Changes of R_t if the input temperature T^m randomly varies by about 3%.

The inlet values A_i and R_i, and the parameters in the model Equations (16.7) have the values and units given in Table 16-4.

A_i	=	1 mol/l	R_i	=	0 mol/l
B	=	5×10^3 sec^{-1}	C	=	5033 K
D	=	1×10^6 sec^{-1}	E	=	7550 K
τ	=	60 sec	H	=	5 lK/mol
T_i	=	410 K			

Table 16-4: Values and units of the parameters used in the system of Equations (16.7).

The database of state variables A_t, R_t and T_t can be calculated at any time t by inserting the above parameters into Equations (16.7) and integrating them.

The process can be manipulated mainly by the temperature at which the reaction is run. Therefore, a historical database covering many different cases can be obtained from Equations (16.7) by taking any given triplet of variables A_t, R_t and T_t and randomly changing the temperature for the calculation of the new triplet A_{t+1}, R_{t+1} and T_{t+1}.

This randomly changed temperature is considered as an additional variable T^m (*manipulated temperature*); T^m specifies the temperature at which the controller is holding the system, while T_t is the temperature of the system if it is left to itself. (Of course, very few reaction systems can be "left to themselves"!)

In the present case, the goal of the process is to maintain the yield R_t constant. As can be seen (and calculated from Equations (16.7)) any perturbation of the system variables will change R_t; uncontrolled change of the variables can lead to serious problems (Figure 16-8 shows the fluctuation of R_t if the temperature T^m randomly varies by about ±3%).

Neural networks will be used twice in the construction of a controlling device for the CSTR: first, to develop a model M that will quantitatively predict the correct value of the yield R_{t+1} for any triplet of data A_t, R_t and T_t and the manipulated temperature T^m; and second, to develop the inverse model M^{-1} capable of predicting the needed adjustment of T^m from four variables – the triplet A_t, R_t and T_t **and** the yield R_{t+1}. That is, we adjust T^m to compensate **for all** changes away from optimum conditions.

Because we are concerned with neural networks and not process control as such, we will be satisfied with the scheme shown in Figure 16-9, in spite of its rather schematic form. The scope of this book does not permit too many details; if you are interested in this area, see the literature cited in Section 16.6 (Bhat, Ungar, Himmelblau, Bulsari, etc.).

As shown in Figure 16-9, there are four general steps in this algorithm, as follows. The state variables A_t, R_t and T_t of the process P are measured at regular time intervals dt (a). The model M checks the outcomes of the process P by regular forward prediction (b). If the controlled variable R_t changes for some reason, corrective action must be taken.

Therefore, the inverse model M^{-1} calculates the corrections T^m needed (c). This correction is input to the process P (d) and the actual consequences of the correction are monitored by comparison of the process data with the set-point data and/or with those predicted by the model.

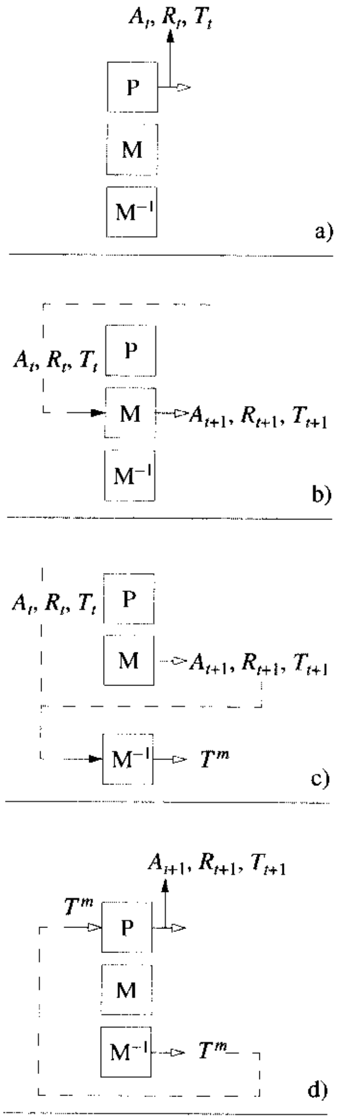

Figure 16-9: Control of the process P with the model M and the controller C (C = M^{-1}, the inverse model) shown in four phases.

Back-propagation approach. Here, we will follow the work of Watanabe et al. (see Reference 16-3). In back-propagation, two networks must be used: first for learning the model, M, and second, for learning its inverse model, M^{-1}. In the Watanabe approach, the first network has an architecture of (4 x 8 x 3) and the second one (4 x 8 x 1). Both networks are shown in Figures 16-10 and 16-11. Setting up the model M is called *forward learning*, while setting up the inverse model M^{-1} is called *inverse learning*.

The most important requirement of the model M is that, at time t, it predicts the future yield R_{t+1} (at time $t + 1$) as precisely as possible. Forward learning thus requires input of A_t, R_t, T_t and T^m at time t, and targets of A_{t+1}, R_{t+1}, and T_{t+1} at time $t + 1$.

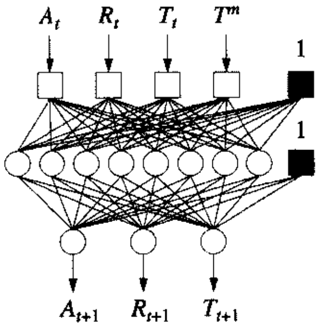

Figure 16-10: The neural network for learning the model M.

The inverse learning, however, requires as input the state variables A_t, R_t and T_t **and** the variables of the future process vector A_{t+1}, R_{t+1}, T_{t+1}, while the target is the manipulated temperature T^m.

Of course, an input for the inverse model composed of two complete consecutive triplets of state variables, (A_t, R_t and T_t) and (A_{t+1}, R_{t+1}, T_{t+1}) requires a larger network with more weights to adjust, and consequently longer learning times. To keep the network for M^{-1} as small as possible, only the most important variable, i.e., the controlled variable, R_{t+1} from the future horizon, is retained for learning M^{-1}.

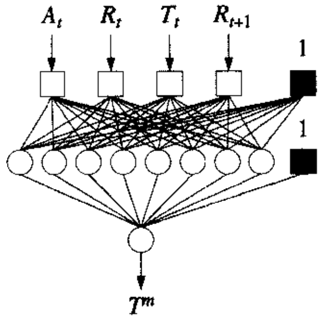

Figure 16-11: The neural network for learning the inverse prediction model, M^{-1}.

Recall the discussion of the moving window concept (Sections 9.5 and 16.2). The calculation of the model M and its inverse M^{-1} is a typical application of the moving window learning approach, although the past and future horizons are only one time step long.

A database of about 400 triplets taken at 30 second intervals was calculated using Equations (16.7) and the parameter values given in Table 16-4. By varying the manipulated temperature T^m randomly within 12 degrees around 410 K, a comparatively good distribution of all possibilities is obtained. The first 200 triplets were taken as the learning set and the rest as the test set.

Both networks were trained by standard back-propagation equations (Section 8.7), with all inputs and outputs scaled between 0 and 1. As shown in Figure 16-12, the neural network predicts the yield R_{t+1} within a few percent of the calculated values.

Theoretically, we should be able to perform a consistency check: the inverse model M^{-1} should yield for **any** input X an output Y, which, if input to the forward model M, will produce an output Y' **exactly equal** to the input X originally supplied to the inverse model

M^{-1}. Unfortunately, because **two** networks have to be trained to accomplish the entire controlling scheme from **two different** sets of data, a certain discrepancy is **always** present between them (see the inputs and outputs of Figures 16-11 and 16-12).

Psichogios and Ungar (References 16-11) reported a discrepancy of about 5% between the results obtained from M and M^{-1}. In a nonlinear controlling scheme, especially if the control needs to keep the system very near the point of maximum yield, even a small error has sufficiently large consequences to invalidate the above controlling scheme.

Therefore, they suggested a slightly altered (but more sophisticated) controlling technique employing a delay device on the output of the model M. With this, they were able to use the forward and inverse models developed by the neural networks without any problems. For more details, consult Reference 16-11.

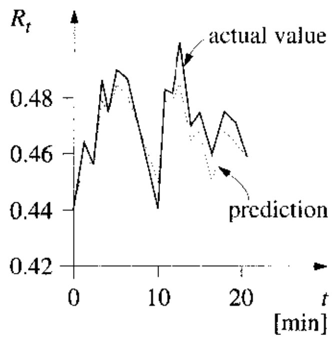

Figure 16-12: Differences between the calculated yield R_t and the one predicted by the neural network model M.

Counter-propagation approach. Here, we will try to solve the CSTR problem using a counter-propagation neural network (see Chapter 7). This example will show that counter-propagation can be as good at modeling as the back-propagation model.

The training and test data sets used in this example are obtained by the same set of Equations (16.7), using the same initial conditions. In addition, the number and selection of input and output variables and their scaling are the same as in the training and test sets employed by Psichogios and Ungar in the back-propagation model.

The only difference between the data (training and test) used in the two methods is that the datasets used in counter-propagation are more than twice as large.

Counter-propagation learning can be regarded as a *nonlinear smoothing* procedure; therefore, larger amounts of input data are necessary for a given accuracy. Fortunately, in counter-propagation learning, the total training time does not rise rapidly as a function of dataset size. Since the number of epochs needed for training is similar to those needed in Kohonen learning, the number of epochs needed for stabilizing the counter-propagation network is orders of magnitude smaller than for back-propagation learning.

The main goal of this example is to show that a counter-propagation network trained as the forward model M can be used as the inverse model, M^{-1}, too.

The counter-propagation network selected for this example is made up of 5000 neurons spread out in two (50 x 50) layers. The size of the network (2500 neurons) offers enough room for 1000 training vectors, (A_t, R_t, T_t, T^m), without too many conflicts. (Because some of the training vectors are quite similar to each other, conflicts cannot be completely avoided.)

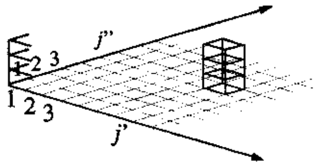

Figure 16-13: A three-dimensional lookup table. By selecting two coordinates, *j'* and *j"*, in the network's map, a "box" containing values of three different variables is always obtained.

Remember, the trained counter-propagation network is a lookup table in which all the **multiresponse** answers are already calculated, stored in boxes, and waiting to be retrieved.

Hence, a (50 x 50) network can handle 2500 **multivariate** answers. The real problem actually is, to select the proper output neuron for the needed answer. Figure 16-13 shows a three-dimensional lookup table.

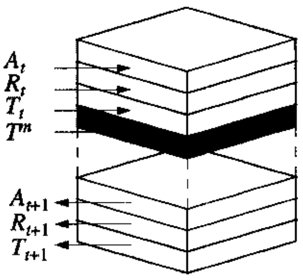

Figure 16-14: The architecture of the counter-propagation network used for learning the forward model M.

The neurons in the first (Kohonen) layer have four weights that are linked to the input signals, A_t, R_t, T_t , T^m. The neurons in the second (output) layer have three weights from which the output values, A_{t+1}, R_{t+1}, and T_{t+1}, are recalled. The described counter-propagation network, which has (50 x 50 x 4) + (50 x 50 x 3) = 35,000 weights, is schematically shown in Figure 16-14.

Recall that in the counter-propagation network, the output is obtained differently from other networks (see Chapter 7 for general and Section 7.4 for particular explanations about output in this type of network). The difference is that the output neurons in the counter-propagation network **do not** calculate the answer from the weights using the equations given in Chapter 2 ((2.9) and (2.39)); instead, the adapted weights **are** the answers already. The output weights in the counter-propagation network, w_{ji}^{out}, are labeled as c_{ji} (Section 7.7), in order to distinguish them from the weights w_{ji} in the Kohonen layer.

A counter-propagation network uses supervised learning, which is essentially a competitive or Kohonen learning with an additional adaptation of weights, c_{ji}, in the output layer to make them close to the targets Y_s. As explained above, the targets in our example are the three process variables, A_{t+1}, R_{t+1}, and T_{t+1} ($t + 1$ is the next time interval).

Learning begins in the Kohonen layer and consists of the self-organization of the multivariate inputs exactly in the same way as described in Chapter 6 for Kohonen learning.

Simultaneously with the self-organization going on in the Kohonen layer, a similar self-organization is being carried out among the corresponding **targets** which are **input** into the lower (output) layer. The method has obtained the name "counter-propagation" precisely because of these two "inputs" coming from two opposite directions: the input vectors into the Kohonen layer and the targets into the output layer.

The only difference between the two self-organization procedures is that the winning neuron is determined only once – in the Kohonen layer. Once the position (j', j'') of the winning neuron c ("central") in the Kohonen layer is determined:

$$out_c \leftarrow \min\left\{\sum_{i=1}^{m}(x_i - w_{ji})^2\right\} \tag{16.8}$$

the neuron from the output layer at the **same position (j', j'')** is selected. The correction of the weights w_{ji} in the Kohonen layer uses the input vector $\boldsymbol{X}$ according to the Kohonen strategy (Equation (16.9); see also Equation (7.6)). The correction of weights c_{ji} in the output layer using the target $\boldsymbol{Y}$ is made according to Equation (16.10), (7.7). For a more detailed explanation, see Chapter 7.

$$w_{ji}^{(new)} = w_{ji}^{(old)} + \eta(t)\,\mathrm{a}(d_c - d_j)\left(x_j - w_{ji}^{(old)}\right) \tag{16.9}$$

$$c_{ji}^{(new)} = c_{ji}^{(old)} + \eta(t)\,\mathrm{a}(d_c - d_j)\left(y_j - w_{ji}^{(old)}\right) \tag{16.10}$$

The learning rate $\eta(t)$, which plays a more important role in Kohonen learning than in back-propagation learning, is calculated using Equation (6.5) (Chapter 6):

$$\eta(t) = (a_{max} - a_{min})\frac{t_{max} - t}{t_{max} - 1} + a_{min} \tag{16.11}$$

The parameters a_{max} and a_{min} determine the maximum and minimum corrections of the central neuron (a $(d_c - d_j) = 0$) at the beginning of training ($t = 1$, $\eta(t) = a_{max}$), and at the end of training ($t = t_{max}$, $\eta(t) = a_{min}$). The values of a_{max} and a_{min} must be defined within the interval 1.0 and 0.0.

The fact that the counter-propagation network is "merely" a lookup table, able to give only a limited number of different answers, may appear at first glance to be a serious drawback. Real models giving continuous answers to different sets of input variables might seem to be much better. However, more important than the **number** of possible answers is the **size of the error** of these answers. If the manipulated variable can take any value within a 5% interval around the reference point, let us say ±20 K around 400 K, this means that the 2500 answers can cover this interval in temperature steps as small as 0.016 K.

But in real applications, one temperature value can be associated with different combinations of values of other variables, which means that several boxes in the lookup table will have the same value of the temperature – which means that the temperature steps could **not** be as small as 0.016 K. Nevertheless, if the number of answers or the precision associated with this approach is not sufficient for a given application, the network can easily be enlarged.

In any case, the problem of evaluating the correct answer is reduced to finding the box with the most appropriate answer in the lookup table.

Because all the answers must be calculated in advance, rigorous experimental design with a uniform distribution of all possible cases is extremely important. This requirement was the basis for the initial selection of the network size. Figure 16-15 shows the distribution on the 50 x 50 map of the process vectors from the training set.

In the present example, 1000 input vectors $X = (A_t, R_t, T_t, T^m)$ with 1000 targets $Y = (A_{t+1}, R_{t+1}, T_{t+1})$ were used for the forward model. Altogether 20 epochs (i.e. 20,000 inputs) were needed for the network to stabilize when the value of the correction function $\eta(t)$ was:

$$\eta(t) = (0.5 - 0.01)\frac{20000 - t}{20000 - 1} + 0.05 \qquad (16.12)$$

Learning for 20 epochs means that in the total training period:

- process vectors are input 20,000 times;
- the central neuron, $c = (j', j'')$, is found each time:
  - all neurons in the corresponding neighborhood are corrected
  - the target process vector is input into the location (j', j'') in the output layer
  - the neurons in the corresponing neighborhood around the (j', j'') location in the output layer are corrected.

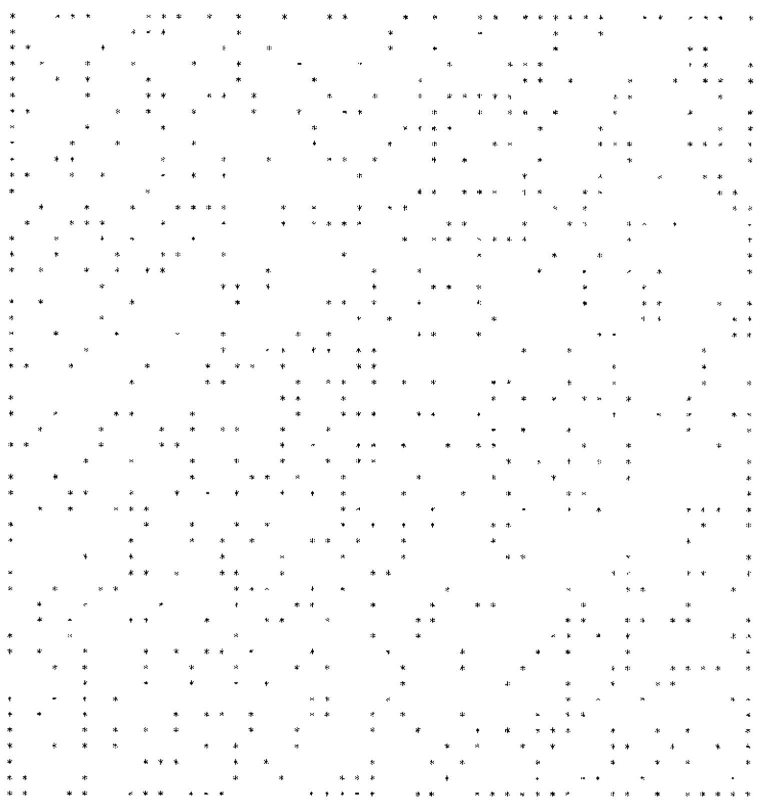

Figure 16-15: Distribution on a (50 x 50) map of the process vectors used in the training set. Because toroidal mapping was **not** used, slightly more objects (points) appear at the border lines of the map.

Once the counter-propagation network is trained, retrieval is straightforward: the neuron in the Kohonen network having weights most similar to the input variables is located; the weights of the neuron at the same position (*j*', *j*") in the output layer contain the answer.

The trained counter-propagation network was tested with the thousand process vectors not used in the training; it has excellent prediction ability for all three process variables (Table 16-5).

variable	recall		prediction	
	σ	$\sigma/\sqrt{1000}$	σ	$\sigma/\sqrt{1000}$
A [%]	0.004	1.2 x10^{-4}	0.005	1.6 x10^{-4}
R [%]	0.004	1.2 x10^{-4}	0.005	1.6 x10^{-4}
T [K]	0.86	0.027	1.03	0.032

Table 16-5: Recall and prediction ability of the (50 x 50 x 4) + (50 x 50 x 3) counter-propagation network.

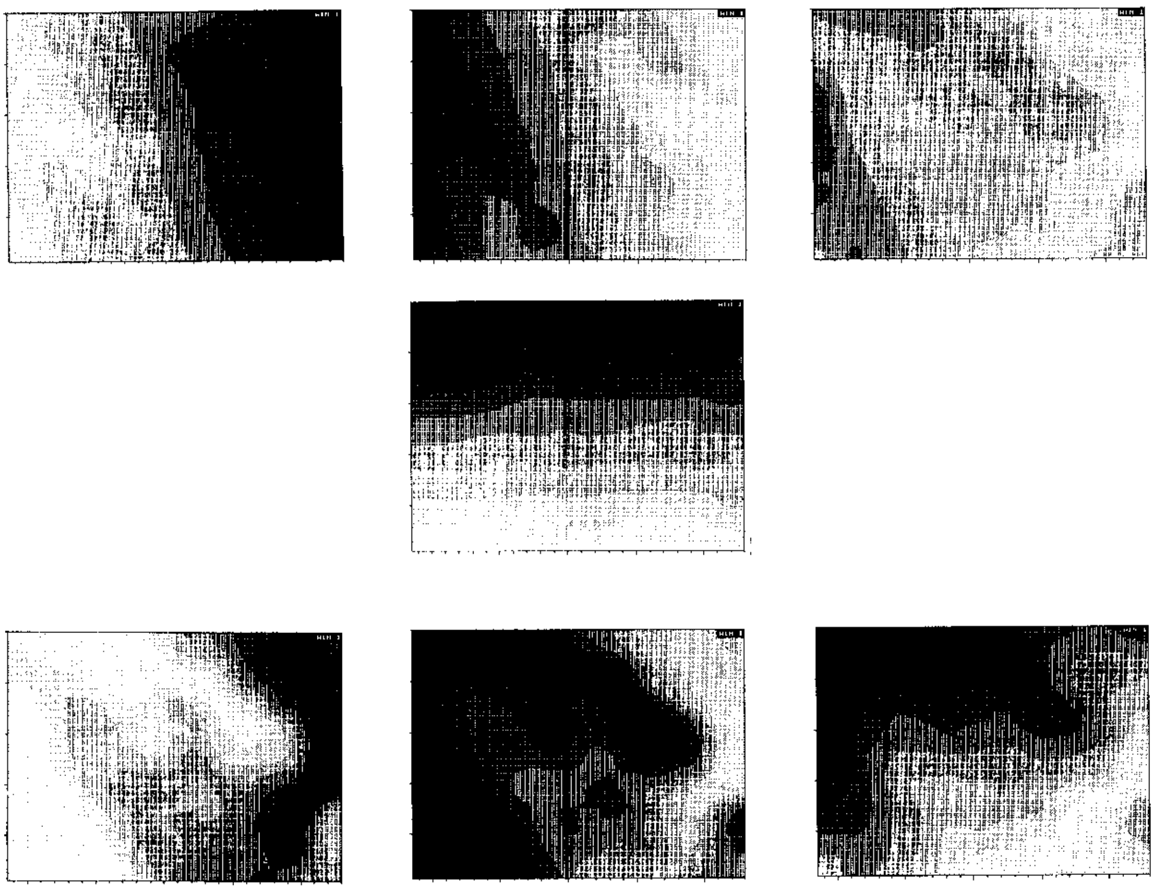

Figure 16-16: The seven process variable maps (four input and three output) obtained by the counter-propagation neural network. The first three (input A_t, R_t, T_t), and the fourth (input manipulated temperature, T^m) were obtained in the Kohonen layer, while the three output maps (A_{t+1}, R_{t+1}, T_{t+1}) were obtained from the weights in the output layer. The darker parts of the map shows higher values.

The standard error σ of 0.86 in the temperature prediction is equivalent to an error of less than 1 degree Kelvin. Errors of the means were calculated for 1000 tests; they have no particular significance.

Each level of weights, whether in the Kohonen or in the output layer, can be represented as a map (see Figures 7-16 and 7-17 of the tennis example in Section 7.5), and "contoured" by drawing lines (*iso-value lines*) connecting neurons having the same weight. The resulting seven maps (four input and three output maps) of weights trained to give the forward model M are shown in Figure 16-16.

These two-dimensional maps are very informative. For example, if we overlay the fourth input map (T^m), upon any of the other three input maps (A_t, R_t and T_t), it can be seen that the iso-value lines of T^m

cut the iso-value lines of T_t approximately perpendicularly. This confirms the assumption that our randomly selected values of T^m cover the entire variable space adequately: the selection of T^m does not depend on the choice of either of the three other variables.

In addition, the iso-value lines of the predicted temperature T_{t+1} (seventh map, third output map) almost exactly follow the iso-lines of the input T^m (manipulated temperature), though slightly shifted. This shows that the control is executed exactly.

Of course, the most important single features of **any** of the maps are the peaks in the R_{t+1} map, because the entire goal is to keep the system running with the highest possible yield of R.

The set of seven maps shown in Figure 16-16 offers a very attractive idea, which was briefly addressed at the end of Chapter 7. All seven maps are of exactly the same size, with apparently no "natural" order (except that the upper four maps belong to the input variables, while the lower three belong to the output). If, as shown in Figure 16-17, the maps are **rearranged** in a different order, a new network is obtained.

The new arrangement of maps is not just random; the maps are divided into two groups: the first one consists of six maps representing two consecutive process vectors, $\boldsymbol{P}_t$ and $\boldsymbol{P}_{t+1}$, i.e., (A_t, R_t, T_t, A_{t+1}, R_{t+1}, T_{t+1}); the T^m map stands alone.

This rearranged stack of the weight maps can be regarded as corresponding to a new network having a Kohonen layer capable of accepting the six variables A_t, R_t , T_t, A_{t+1}, R_{t+1}, T_{t+1} as input, and yielding an output vector $\boldsymbol{Y}$ composed of the single variable T^m. Thus:

A counter-propagation network for the inverse model M^{-1} can be obtained by **rearranging** the previous forward model network, M.

That is, we have just designed a new (50 x 50 x 6) + (50 x 50 x 1) counter-propagation network by rearranging the weight maps of another one – without the need for any training.

The rearranged network, M^{-1}, was tested twice: first, by predicting 1000 T^m values from the dataset on which the original network was based; second, by predicting 1000 T^m values from a new set of data obtained from a set of random process vectors. Both results are given in Table 16-6. Once again, remember that this network is a lookup table: the input variables are used to determine the neuron (j', j''), i.e.,

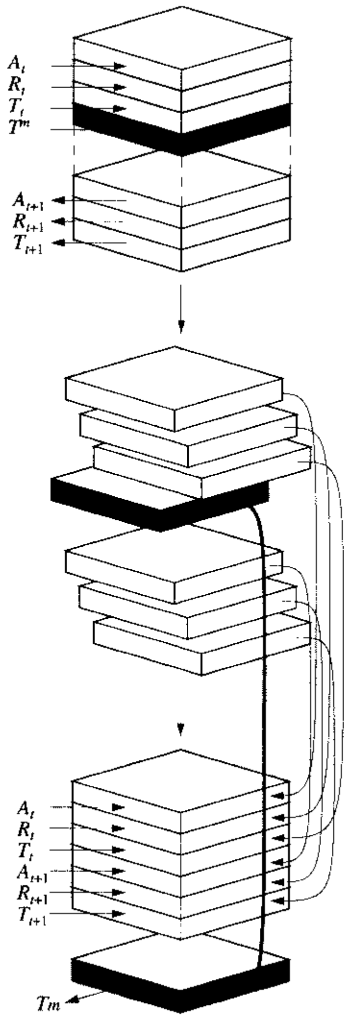

Figure 16-17: Rearrangement of seven weight maps of the forward

to determine the address of a neuron in the output layer, whose weights are the desired answer.

variable	1st set		2nd set	
	σ	$\sigma/\sqrt{1000}$	σ	$\sigma/\sqrt{1000}$
T [K]	3.4	0.11	3.8	0.12

Table 16-6: Predictive ability of the inverse model obtained by rearranging the forward one.

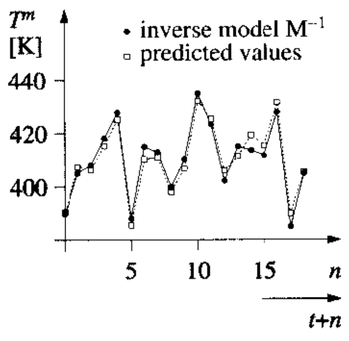

Figure 16-18: Comparison of test and predicted values obtained by the inverse counter-propagation model.

The prediction ability of the inverse model is worse than that of the forward model; however, the standard errors are still below 4 K, which means errors less than 1%. Figure 16-18 shows a few predictions made by this inverse model.

It is surprising that such a simple lookup table, containing only 2500 answers and obtained as a byproduct of learning the forward model, can produce results that good. According to some authors (see Reference 16-15), even much more sophisticated and rigorous methods fail to produce satisfactory inverse models, because they rely on higher order derivatives, and are therefore highly sensitive to noise and numerical errors. Hence, this achievement is at least comparable to if not better than most of the other contemporary methods.

Of course, there are still a number of questions to be answered before this method of obtaining the inverse model can be generally accepted. For example: Is the obtained answer always the best possible one? Is it possible that there is an even better answer in another box that happens to show slightly worse agreement with the input variables? If this is the case, how can such local minima be found? How robust is this method?

16.6 References and Suggested Readings

16-1. J. C. Hoskins and D. M. Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering", *Comput. Chem. Eng.* **12** (1988) 881 – 890.

16-2. V. Venkatasubramanian, R. Vaidyanathan and Y. Yamamoto, "Process Fault Detection and Diagnosis Using Neural Networks I. Steady State Processes", *Comput. Chem. Eng.* **14** (1990) 699 – 712.

16-3. K. Watanabe, I. Matsuura, M. Abe, M. Kubota and D. M. Himmelblau, "Incipient Fault Diagnosis of Chemical Process via Artificial Neural Networks", *AIChE Journal* **35** (1989) 1803 – 1812.

16-4. P. Bhagat, "An Introduction to Neural Nets", *Chem. Eng. Prog.* **86** (1990) 55 – 60.

16-5. V. Venkatsubramanian and K. Chan, "A Neural Network Methodology for Process Fault Diagnosis", *AIChE Journal* **35** (1989) 1993 – 2002.

16-6. L. H. Ungar, B. A. Powel and S. N. Kamens, Adaptive Networks for Fault Diagnosis and Process Control", *Comput. Chem. Eng.* **14** (1990) 561 – 572.

16-7. N. Bhat and T. J. McAvoy, "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems", *Comput. Chem. Eng.* **14** (1990) 573 – 583.

16-8. N. Bhat, P. A. Minderman, Jr., T. J. McAvoy and N. S. Wang, "Modeling Chemical Process Systems via Neural Computation", *IEEE Control Systems Magazine* (April 1990) 573 – 582.

16-9. J. Leonard and M. A. Kramer, "Improvement of the Back-Propagation Algorithm for Training Neural Networks", *Comput. Chem. Eng.* **14** (1990) 337 – 41.

16-10. A. Bulsari and H. Saxen, "Applicability of an Artificial Neural Network as a Simulator for a Chemical Process", *Proc. 5-th Intern. Symposium on Computer and Information Sciences*, Nevsehir, Turkey, (October 1990) 143 – 151.

16-11. D. C. Psichogios and L. H. Ungar, "Direct and Indirect Model Based Control Using Artificial Neural Networks", *Ind. Eng. Chem. Res.* **30** (1991) 2564 – 2573.

16-12. J. R. Lang, H. T. Mayfield, M. V. Henly and P. R. Kromann, "Pattern Recognition of Jet Fuel Chromatographic Data by Artificial Neural Networks with Back-Propagation of Error", *Anal. Chem.* **63** (1991) 1256 – 1261.

16-13. R. Hecht-Nielsen, "Counterpropagation Networks", *Appl. Optics* **26** (1987) 4979 – 84.

16-14. D. G. Stork, "Counterpropagation Networks: Adaptive Hierarchical Networks for Near Optimal Mappings", *Synapse Connection* **1** (1988) 9 – 17.

16-15. C. E. Economu and M. Morari, "Internal Model control. 5. Extension to Nonlinear Systems", *Ind. Eng. Chem. Process Des. Dev.* **25** (1986) 403 – 411.

16-16. J. Zupan and M. Novic, "Counterpropagation Learning Strategy in Neural Networks and Its Application in Chemistry", in *Further Advances in Chemical Information,* Ed.: H. Collier, Roy. Soc. of Chem., Cambridge, UK, 1994, pp. 92 – 108.