J Syst Eng (1995)5:207–222
© 1995 Springer-Verlag London Limited

**Journal of Systems Engineering**

# Fuzzy Adaptive Recurrent Counterpropagation Neural Networks: A Tool for Efficient Implementation of Qualitative Models of Dynamic Processes

François E. Cellier and YaDung Pan

Department of Electrical and Computer Engineering, The University of Arizona, USA

*In this paper, a new method for efficient implementation of qualitative and mixed quantitative/qualitative models of time-varying (dynamic) processes is shown. It involves a special brand of recurrent neural network termed fuzzy adaptive recurrent counterpropagation neural network (FARCNN). This is the first paper on FARCNNs ever written. It explains the methodology in detail, and ends with an illustrative example of their use.*

**Keywords:** Modelling; Simulation; Time-series analysis; Forecasting; Mixed quantitative and qualitative models; Neural networks; Fuzzy systems; Learning systems; Artificial intelligence

## 1. Introduction

Qualitative models of dynamic processes are used for many purposes. A common application of such models arises in the context of fault monitoring and diagnosis of technical processes [1, 2]. Other applications include descriptions of systems or subsystems for which no analytical (i.e., quantitative) models are available, such as in the case of many biomedical [3, 4] and economical [5] applications.

Qualitative models rely on discrete or discretised variables for their description [6]. While some qualitative models are simulated using discrete event simulation [7], others are simulated by means of knowledge inferencing from a finite state machine representation [8].

Monitoring and diagnostic systems are not of much use if they cannot be implemented in real time. For example, the Anaesthetic Expert Advisor RESAC [9] became so complex that it could no longer render its intended purpose on the machine upon which it was implemented. Therefore, the same research group created an alternative system, ANNAD [10] that was based on a neural network architecture. This system acts similarly to RESAC, but provides its advice much faster. Neural networks are naturally parallel, and therefore can be implemented very efficiently for use in real-time environments.

Unfortunately, most neural networks are structured very differently from the rule-based systems that were used previously. Therefore, the software system has to be designed from scratch. Also, neural networks are difficult to train, and it is not easy to anticipate in any given situation, how long it will take to design and train a neural network, or whether the approach will work at all.

It seems therefore an interesting task to study whether a neural network architecture can be designed that resembles much more closely the rule-based systems of the past, and that therefore can be implemented much more rapidly than the conventional neural network architectures if a rule-based design already exists; a network, the convergence of which can be guaranteed from the outset.

In the remainder of this paper, such a neural network architecture is presented.

*Received 15 July 1994*

*Correspondence and offprint requests to*: F. E. Cellier, Department of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona 85721, USA.

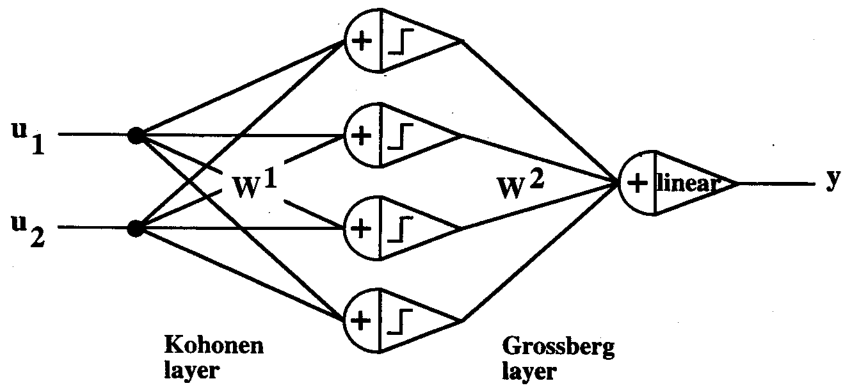

**Fig. 1.** Counterpropagation network for XOR function.

## 2. Counterpropagation: Back to the Basics

The basic properties of counterpropagation are best introduced by means of a simple example: the infamous XOR problem. It is desired to design a neural network that can reproduce the behaviour of an XOR gate: the counterpropagation network (CNN) for this problem is shown in Fig. 1. The first layer, the so-called *Kohonen layer*, consists of four simple perceptrons with a threshold of 1.0. Its weight matrix, $\mathbf{W}^1$, is similar to the truth table of the input patterns in Table 1:

$$\mathbf{W}^1 = \begin{pmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{pmatrix} \tag{1}$$

except that the representation of *false* is now $-1$ rather than 0, while the representation for *true* is still $+1$. In this paper, superscripts always point to the layer of the neural network, whereas subscripts denote elements of vectors or matrices.

If a particular input pattern, e.g. $\mathbf{u}^1 = (-1\ +1)^T$, is shown to the neural network, the matrix multiplication, $\mathbf{W}^1 \cdot \mathbf{u}^1$ produces the vector

$$\mathbf{x}^1 = (0\ +2\ -2\ 0)^T$$

**Table 1.** Truth table of XOR gate.

| $u_1$ | $u_2$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

and the output activation functions of the four neurons produce the vector

$$\mathbf{y}^1 = (0\ 1\ 0\ 0)^T$$

accordingly. Thus, the Kohonen layer acts as a *powerset generator*. It contains as many perceptrons as there exist different input patterns. If the CNN has $k$ different binary inputs, a complete Kohonen layer will consist of $2^k$ perceptrons, the thresholds of which can be set to $d_i^1 = k - 1$. There is no competitive learning as in the case of Hecht-Nielson's CNNs [11]. Both the weight matrix, $\mathbf{W}^1$, and the threshold vector of the perceptrons, $\mathbf{d}^1$, can be fixed from the onset, and the perceptrons do not need to cooperate with each other at all.

The second layer, or *Grossberg layer*, consists of as many linear neurons as there are output variables; in the case of the XOR example, a single linear neuron. The weight matrix, $\mathbf{W}^2$, is similar to the transpose of the output patterns of the truth table, in the case of the XOR problem thus:

$$\mathbf{W}^2 = (-1\ +1\ +1\ -1) \tag{2}$$

The Grossberg layer acts as a *selector*. The product $\mathbf{W}^2 \cdot \mathbf{y}^1$ picks out the output pattern that corresponds to the input pattern that was shown to the CNN. As in the case of the Kohonen layer, no learning is necessary. The weight matrix, $\mathbf{W}^2$, can be predetermined just like the weight matrix $\mathbf{W}^1$.

In the remainder of this paper, it is this type of neural network that will be referred to as a counterpropagation neural network (CNN). A CNN is a *binary neural network* in that all its inputs and outputs are binary, and the *false* value is represented by $-1$, while the *true* value is represented by $+1$.

The output layer of the CNN does not necessarily have to be binary. The $\mathbf{W}^2$ matrix could assume any values. In such a case, the network will be called *Generalised counterpropagation neural network*, (GCNN).

## 3. CNN: a Tool for Rapid Implementation of Finite State Machines

A finite state machine (FSM) is similar to a logic truth table, except that its variables may employ multi-valued logic. Often, the number of output variables equals the number of input variables, and the semantic meaning of the FSM is a transition from the input state to the output state, i.e., the current values of the set of input variables represent the current state of the system, whereas the values of the set of output variables represent the next state of the system. This is where the name 'finite state machine' comes from. However, the concept of an FSM can be generalised by defining it to mean an arbitrary truth table with $i$ input variables and $o$ output variables, each of which can assume a finite set of values that can be represented either by integers, such as '1', '2', and '3', or by symbols, such as 'small', 'medium', and 'large'.

The purpose of the CNN is that of a table-lookup function. Whenever one of the input patterns is shown to it, it reacts by presenting the corresponding output pattern at its output.

The realisation of this neural network is quite trivial. It is shown in Fig. 2. It consists of a set of finite state to binary (FS/B) converters, followed by a regular CNN, followed by a set of binary to finite state (B/FS) converters. The example shown in Fig. 2 contains three input signals, $u_1$ to $u_3$, and two output signals, $y_1$ and $y_2$. Each of the five signals is a multi-valued logic signal. The FS/B converter converts one multi-valued logic signal into multiple binary signals. For example, if $u_1$ has eight levels, it can be converted into three separate binary signals, $u_{1a}$ to $u_{1c}$, etc. Each variable can be converted separately, thereby providing the means also to parallelise this operation.

The B/FS converters can themselves be realised as GCNNs. Since cascaded CNNs can always be amalgamated into one, the B/FS converters could be combined with the CNN to their left, forming a single GCNN.

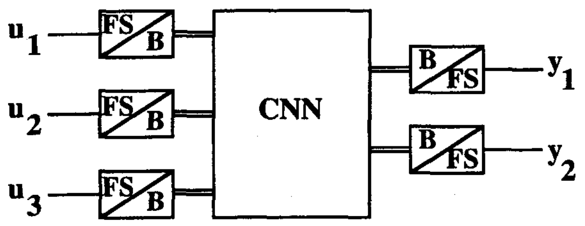

**Fig. 2.** Counterpropagation network for a finite state machine.

## 4. Bringing Time Into the Picture

So far, the neural networks presented and the functions they implemented were all static in nature. Time did not fit into the scheme at all. However, since the goal of this paper is to present a neural network capable of identifying dynamic systems and imitating their behaviour, the concept needs to be enhanced.

Many dynamic systems can be approximated by difference equations of the form

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{u}(t),\mathbf{y}(t-\Delta t),\mathbf{u}(t-\Delta t), \mathbf{y}(t-2\Delta t),\mathbf{u}(t-2\Delta t),\ldots) \quad (3)$$

The output vector, $\mathbf{y}$, at time $t$ is a function of the input vector, $\mathbf{u}$, at the same time, the output vector, $\mathbf{y}$, one sampling interval back, the input vector one sampling interval back, the output vector two sampling intervals back, etc.

If the vector function $\mathbf{f}$ is a tabular vector function rather than an analytical vector function, and if all variables are of the enumerated type, the situation is exactly the same as in the previous section of this paper, except that now, the variables may be sampled at different time instances.

Previous research in time-series analysis has provided answers as to how often the variables need to be sampled and how many past values need to be used in the model [12, 5].

Assuming that a history of two sampling intervals suffices to capture the dynamics of the system, the outputs, once produced, can be delayed and wrapped around to form the past output values needed as additional inputs to the GCNN. This is illustrated in Fig. 3. The illustrated example represents a single-input triple-output system. The boxes denoted as $z^{-1}$ represent delays. All FS/B converters have been amalgamated for simplicity into a single multi-input FS/B box, and similarly for the B/FS converters. Assuming that each of the multi-valued logic signals, $u$, $y_1$, $y_2$, and $y_3$, has four levels, the CNN will have $2 \times (1 + 1 + 1 + 3 + 3) = 18$ binary inputs and $2 \times 3 = 6$ binary outputs.

The feedback provides the memory necessary to capture the system dynamics. The neural network has become recurrent, and it will therefore be called *recurrent counterpropagation neural network* (RCNN).

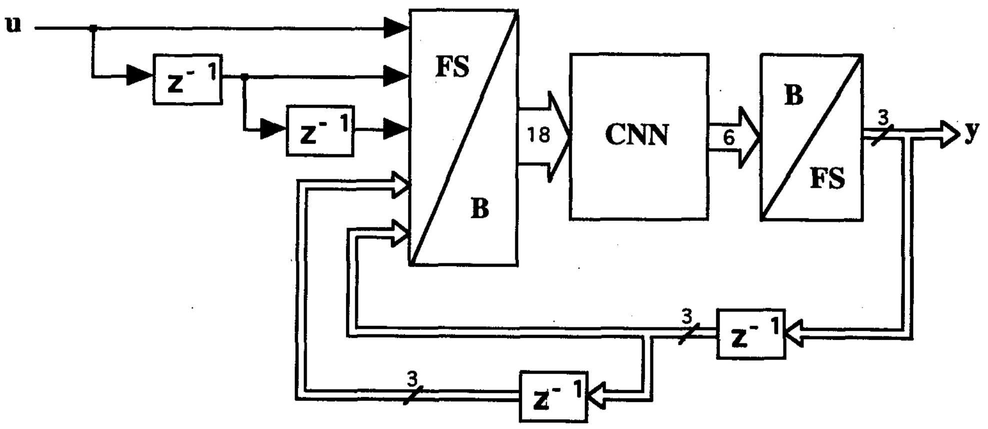

**Fig. 3.** Recurrent counterpropagation network.

## 5. Processing Continuous Variables in an RCNN

The next question to be raised concerns what to do about continuously changing phenomena. Difference equations may still provide a valid approximation, but the variables to be processed by them are no longer of the enumerated type.

The first solution that comes to mind is to preprocess continuous input and output variables using standard analogue to digital (A/D) converters. This solution is shown in Fig. 4. A continuous-time system with three inputs and two outputs is used to illustrate the approach. The real system is first used as a source of data for training the neural network. The system is excited as well as possible at all frequencies by preferably applying *binary random noise* (BRN) to all inputs [13, 8], and these inputs and the reaction of the system to the excitation are recorded once every $\Delta t$ time units.

Although the physical plant to be modelled is a continuous-time system that should be properly described by sets of differential equations, it has been shown that difference equations, if chosen carefully, are able to capture the dynamics of such plants [8, 12, 5]. It is important though to select a proper value of $\Delta t$ [12], and to choose a sufficiently large number of past values [5]. In the example shown in Fig. 4, it is assumed that two sets of past values suffice to characterise the plant dynamics adequately.

It is furthermore assumed that 12-bit A/D converters are used, converting the $3 + 3 + 3 + 2 + 2 = 13$ analogue inputs to $12 \times 13 = 156$ digital inputs, and the two analogue outputs to $12 \times 2 = 24$ digital outputs. A standard CNN can now be devised that relates the 156 binary inputs to the 24 binary outputs.

After identification of the RCNN, the A/D converters at the output can be replaced by digital to analogue (D/A) converters. Continuous input signals shown to the network are first converted to binary input signals. These are then processed by the CNN. The resulting binary outputs are finally converted back to continuous output signals to be delayed and fed back.

There are two problems with this approach. Since the discrete representation of the system is only an approximation, it is not evident that the input/output relationship is still fully deterministic, i.e., it is conceivable that, for the same set of input values, different output values can be observed. This should not be a big problem. If the representation is chosen carefully, the input/output relationship should be fairly deterministic, or at least, if different digital output values result, they should correspond to very similar analogue output values. Consequently, any of them may be chosen during forecasting, and it is usually quite acceptable always to select the most frequently occurring input/output combination. Thus, the CNN can still be made fully deterministic.

The more serious problem with this approach is the sheer size of the CNN. Since the CNN has 156 binary inputs, there exist conceptually $2^{156}$ legal combinations or input states. Thus, a complete CNN should have a hidden layer of length $2^{156}$, which is, of course, quite unreasonable. Although not all of these combinations may occur in practice, the required number of training data records for the CNN would still be unmanageably large.

*Fuzzy reasoning* has taught us a better way [13]. Rather than converting analogue signals to digital signals by means of A/D converters, these signals can be *fuzzified*. In our own dialect of the fuzzy reasoning methodology [13], quantitative

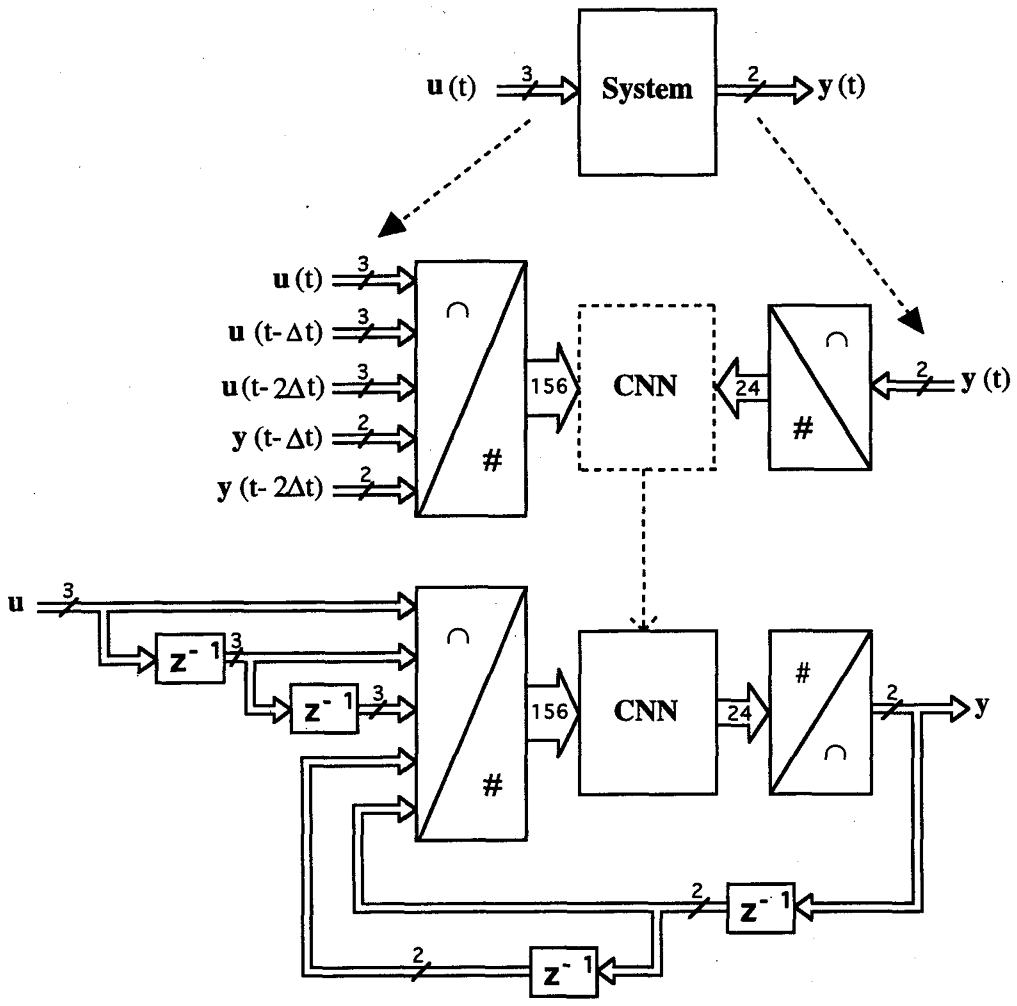

**Fig. 4.** Recurrent counterpropagation network for continuous signals.

(analogue) signals are converted into *qualitative triples* consisting of a *class value* of the enumerated type, a *side value* of the binary type, and a real-valued *fuzzy membership function*. Figure 5 illustrates the approach. A quantitative variable, the systolic blood pressure of a human being is converted to its qualitative counterpart. A systolic blood pressure of 115.0 (quantitative value) is thereby considered ‘normal’ (class value) with a fuzzy membership value of 0.8, and a side value of ‘left’ (the point is to the left of the maximum of the fuzzy membership function that represents the

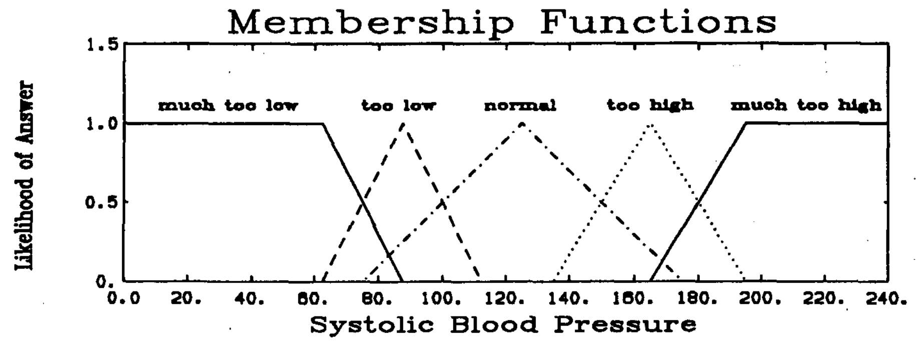

**Fig. 5.** Fuzzification of systolic blood pressure.

selected class). Notice that a quantitative value of 150.0 can be fuzzified either into class 'normal' with a side value of 'right' and a fuzzy membership value of 0.5, or it can be fuzzified into class 'too high' with side value 'left' and a fuzzy membership value of 0.5. However, it does not matter which fuzzification is chosen, since the defuzzified value will in both cases be the same, namely 150.0. In the current implementation, the side function assumes the values $-1$ corresponding to 'left', and $+1$ corresponding to 'right'.

If the fuzzy membership functions are known, it is possible to regenerate the quantitative values from the qualitative triples without any loss of information; thus, the process of fuzzy recoding is reversible.

The class values are of the enumerated type, and the side value is even binary. Thus, it is easy to generate an RCNN that translates the class and side values of observed inputs into the class value and side value of an observed output. Since the discretisation is now much more coarse than before, the input/output relationship will be even more deterministic, and the number of discrete inputs to the CNN is drastically reduced. Experience has shown [8] that one usually gets away with three to eight classes. Thus, fuzzy two-bit converters will often suffice. Together with the binary side value, an analogue signal is thereby discretised into three binary signals and a real-valued fuzzy membership function. In the presence of abundant training data, fuzzy three-bit converters may yield slightly more accurate results. A fuzzy three-bit converter maps an analogue signal into four binary signals and a real-valued fuzzy membership function. The approach is illustrated in Fig. 6 using the same example as in Fig. 4.

The *fuzzifier* replaces the A/D converter. It converts 13 analogue inputs into 39 binary inputs plus 13 fuzzy membership values. The *defuzzifier* replaces the former D/A converter. It converts six binary outputs and two fuzzy membership values back into two analogue outputs.

There are still two problems with this approach. The first objection deals with the size of the CNN. Although the number of inputs of the CNN has been reduced from 156 to 39, there are nevertheless $2^{39}$ legal states left in the system; still a frighteningly large number. The paper will deal with this objection in due course.

A second – and possibly more serious – objection can be raised. Originally, a qualitative model was sought that would translate 13 analogue inputs into two analogue outputs. The advocated solution reduces this problem to the identification of a CNN, and the identification of another box that still is supposed to translate 13 analogue inputs into two analogue outputs. This box has been marked with a '?' in Fig. 6. Has anything been achieved at all? The answer is yes. Even if the box with the '?' is removed from the system, the network will work, although the prediction would be rather crude since the neural network will only predict the class value. The purpose of the box with the '?' is to interpolate smoothly between neighbouring discrete predictions, i.e., this box corrects for an error in the prediction that is of second order small.

However, there is yet a more important distinction between the original modelling problem and the problem to be solved by the box with the '?'. The original qualitative modelling problem can be solved (and has often been solved in the past [5]) by a *Backpropagation Neural Network* (BNN). Of course, the modified problem can also be solved with a BNN. However, whereas a new BNN has to be identified for each new application when solving the original modelling problem, the '?'-box problem can – with a small modification of the architecture – be solved once-and-for-all by a standard BNN that will work for any and all applications. This statement will be proven in due course.

Most fuzzy inferencing and defuzzification techniques, such as the *mean of maxima* (MoM) technique and the *centre of area* (CoA) technique, use the information stored in the tails of the overlapping fuzzy membership functions of the inputs to generate multiple discrete output values and smoothly interpolate between them using the relative importance of the input classes, i.e., the membership value of the output is inferred by looking at the input space alone [14]. The advantage of this approach is that no training data need to be stored. If training data are used at all, they only serve to help with the shaping of the fuzzy membership functions.

It was shown in [15] that – at least in a data-rich environment – it may be more beneficial to store away the available training data set, to look at the input space for computing a distance function between the new vector of inputs and the vector of inputs in the training data set, use this information to determine the relative importance of the inputs stored in the training data set, pick out the five nearest neighbours, and finally look at the output space to interpolate smoothly between the output values of these five nearest neighbours from the training data set. This approach looks similar in nature to the '?'-box approach advocated in this paper.

The outlined approach can be viewed in the

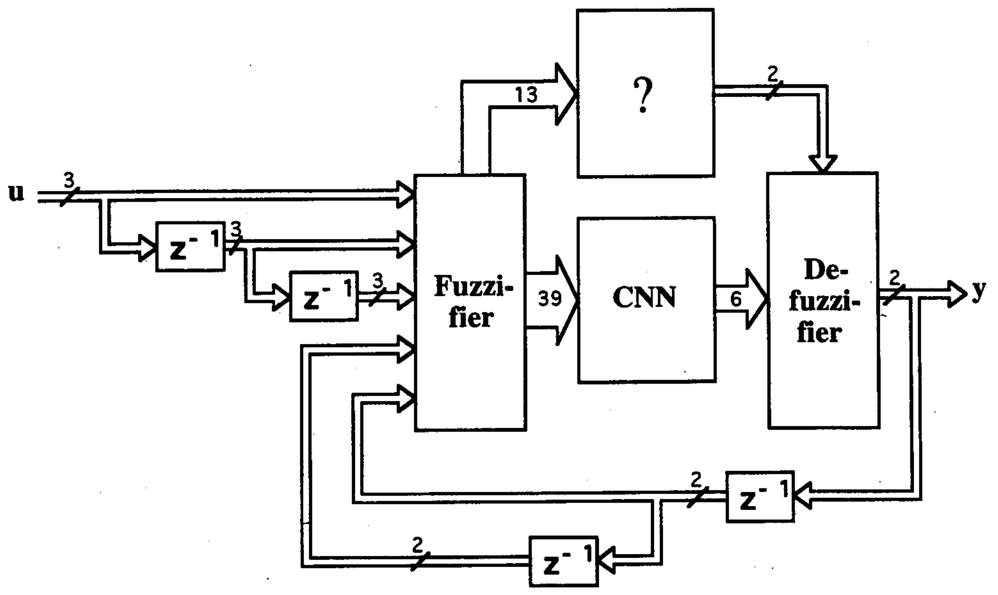

**Fig. 6.** Fuzzy inferencing for counterpropagation.

context of fuzzy measures. It was demonstrated in [15] that the membership value of the output can be written as a linear combination (weighted sum) of the membership values of the output of previous occurrences of the same input/output pattern. The weights themselves are deterministic functions of the membership and side values of the inputs. The functions are independent of the shape of the fuzzy membership function and of the landmark values that separate neighbouring classes. This fuzzy inferencing technique has been termed the *five-nearest-neighbour* (5NN) rule.

Let us turn now to the other objection. The *fuzzy inductive reasoning* (FIR) approach [13, 8] has taught us that not all of the possible inputs are needed to predict the output correctly. A selector function picks a subset of the available inputs. This subset is problem-dependent. The FIR methodology teaches us how to find the selector function. In FIR, this is called the *optimal mask*. A different optimal mask (selector function) is needed for each of the output variables. Each optimal mask requires, on the average, somewhere between three to five inputs. Figure 7 illustrates the proposed architecture for the same example as that shown in Fig. 6.

In this example, it is assumed that the optimal mask to compute $y_1$ has four inputs, whereas the optimal mask to compute $y_2$ has only three inputs. Four classes (fuzzy two-bit converters) are used for all analogue variables. The top CNNs that is used to compute $y_1$ has now 12 binary inputs corresponding to $2^{12} = 4096$ states, while the bottom CNN, used to compute $y_2$, has nine binary inputs corresponding to $2^9 = 512$ different states. These numbers are now quite reasonable. The CNNs predict the class and side values of the output from the class and side values of the inputs. The BNN predicts the membership value of the output from the membership and side values of the inputs. Notice that the two BNNs are identical (they use the same weights), although the number of input variables is different. The BNN was trained such that unused membership inputs can be fixed at 0.0.

The network architecture depicted in Fig. 7 is called a *Fuzzy Recurrent Counterpropagation Neural Network* (FRCNN).

Any type of inductive modelling is somehow related to an optimisation problem. Saying that a neural network needs to be 'trained' is just another way of saying that an optimisation problem needs to be solved. Backpropagation training is slow, since the search space is a high-dimensional continuous space. Consequently, the solution of the associated optimisation problem is very expensive. Fuzzy recoding has alleviated this problem to some extent. The continuous search space is discretised (the search extends only over the space of discrete class values), and the grain size is even quite coarse. The discretisation is in fact so coarse that the resulting outputs are not smooth enough. The fuzzy membership functions are used to interpolate between neighbouring grid points. Fuzzy inferencing provides a means for finding a smooth interpolation function. The better the fuzzy inferencing is done, the smoother will be the interpolated output.

The FRCNN architecture provides us with a good

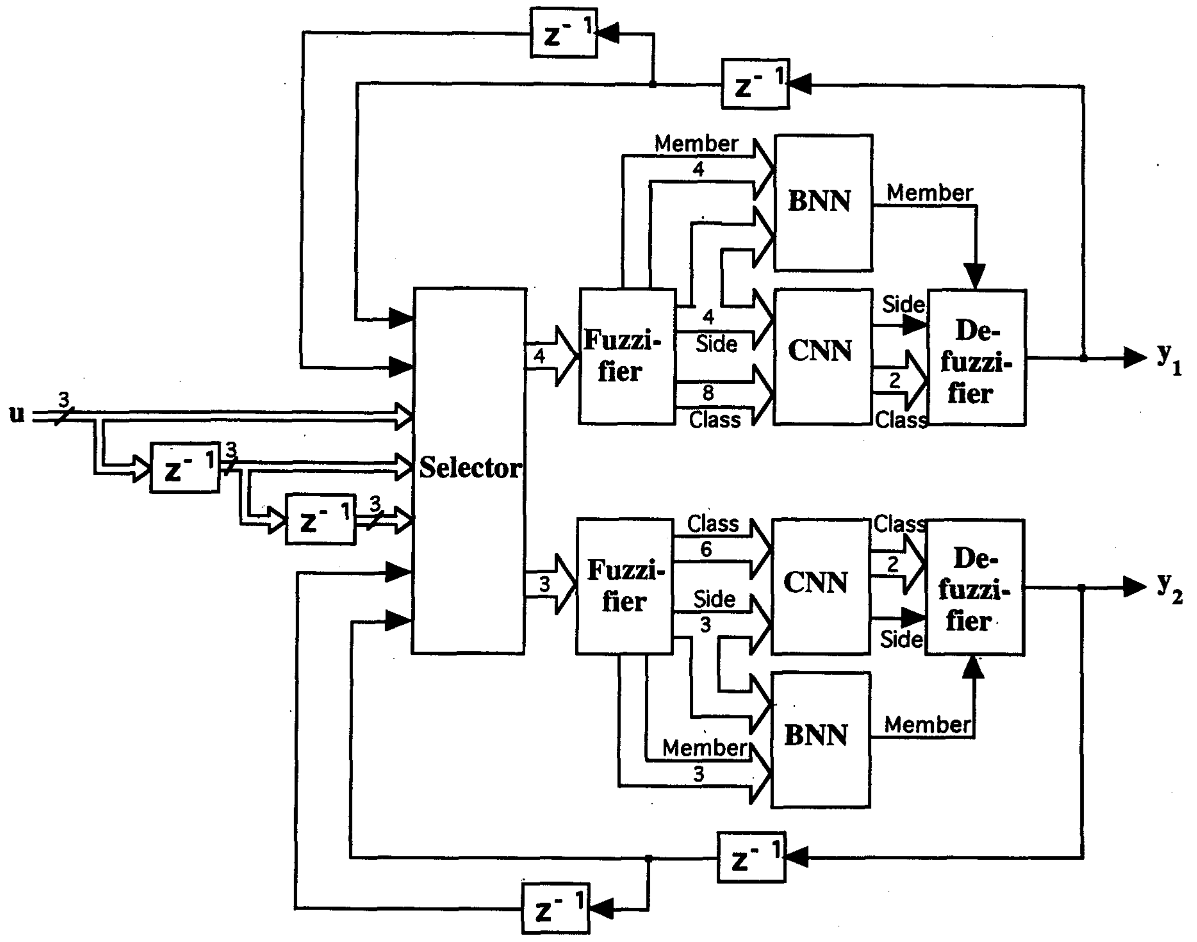

**Fig. 7.** Basic FRCNN architecture.

compromise between smooth approximation and fast training. The training of the BNN is slow, but can be done off-line and once-and-for-all. Training of the CNN is extremely fast. As soon as a particular pattern has been seen once, it will be recognised again for all future. FRCNNs are really nothing new. They are simply a highly efficient way of implementing the FIR methodology [13, 8] for real-time applications.

This concludes the basic description of the FRCNN architecture. The fuzzy converters are quite harmless. The fact that they operate on each variable separately and in the same manner, offers an easy means for parallelisation. Fuzzy converters can be integrated onto fast and inexpensive chips just as regular converters can.

## 6. Practical Implementation of the FRCNN Architecture

In this section, the practical implementation of the FRCNN architecture shall be demonstrated by means of an example. In order to be able to compare our approach with other previously published techniques, an example was chosen from the literature.

It is desired to generate a qualitative model of a non-linear static function with three inputs and one output [16–19]:

$$y = (1.0 + u_1^{0.5} + u_2^{-1} + u_3^{1.5})^2 \qquad (4)$$

where $u_1$, $u_2$ and $u_3$ are three analogue inputs, and $y$ is the analogue output.

Figure 8 illustrates our approach. The three analogue inputs are fuzzified using three separate fuzzy two-bit A/D conveters (FF). $c_1$ is the low bit and $c_2$ is the high bit of the converted signal. $s$ denotes the side value and $M$ stands for the fuzzy memberhsip value.

The CNN is fed by the class and side values from the three fuzzifiers (FFs). Consequently, the CNN has nine binary inputs. Its three binary outputs are the two bits of the class value of the output $y$, as well as its side value. Due to the coarse discretisation, the input/output map should be fairly deterministic.

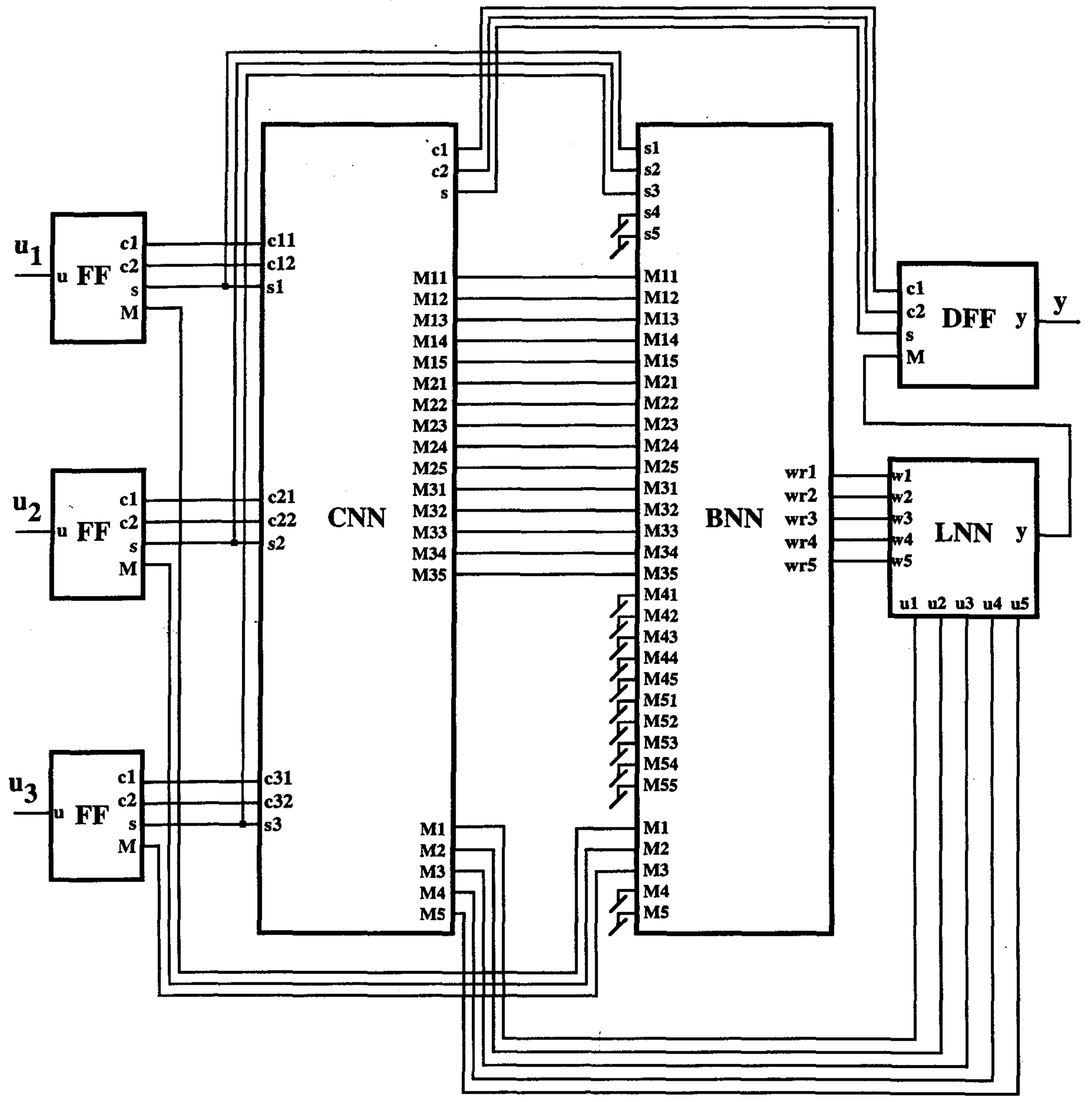

**Fig. 8.** Implementational issues of FRCNN architecture.

Only in the vicinity of a landmark between two classes (the membership value is somewhere around 0.5), will there be non-determinism.

Since the CNN has nine binary inputs, the hidden layer should have $2^9 = 512$ perceptrons. However, in order to be compatible with [16], only 216 training data points were used. Thus, the CNN will not be exhaustive. The training data are uniformly distributed over the input range $[1.0, 6.0] \times [1.0, 6.0] \times [1.0, 6.0]$.

However, the CNN is in reality a GCNN, since its Grossberg layer (the output layer) also computes 20 analogue outputs besides the three binary ones. These are the old (training) values of the fuzzy membership functions of the three inputs and the output. $M_{ij}$ stands for the $j$th membership value of the $i$th input, and $M_j$ stands for the $j$th membership value of the output.

In order to understand how the BNN works, it is necessary to recall some of the results from [15]. In particular, it has yet to be proven that the BNN is indeed application-independent.

In [15], the following algorithm is proposed to infer the fuzzy membership value of the output. For each input, position functions, $p_{ij}$ and $p_i$, are computed as follows:

$$p_{ij} = c_{ij} + s_{ij}(1.0 - M_{ij}) \qquad (5)$$

$$p_i = c_i + s_i(1.0 - M_i) \tag{6}$$

where $M_{ij}$ denotes the $j$th old membership value of the $i$th input from the training data, and $M_i$ stands for the new membership value of the $i$th input from the $i$th fuzzifier. Corresponding conventions hold for the class values, $c_{ij}$ and $c_i$, and for the side values, $s_{ij}$ and $s_i$.

The position values represent normalised versions of defuzzified signals. They are normalised since the lowest class ($c_i = 0$) corresponds to a position variable range from 0.0 to 0.5, the next higher class ($c_i = 1$) corresponds to the range [0.5, 1.5], and so on. Each class, except for the lowest and highest, is exactly equally wide. The slope of all membership functions is identical relative to the normalised position variables.

$\mathbf{p}_j$ is the vector of $p_{ij}$ elements, and $\mathbf{p}$ is the vector of $p_i$ elements:

$$\mathbf{p}_j = (p_{1j}, p_{2j}, p_{3j})^{\mathrm{T}} \tag{7}$$

$$\mathbf{p} = (p_1, p_2, p_3)^{\mathrm{T}} \tag{8}$$

Distance functions, $d_j$, are now computed as follows:

$$d_j = |\mathbf{p} - \mathbf{p}_j| \tag{9}$$

The largest of these distance functions is called $d_{\max}$.

Absolute weights are then computed using the equation

$$w_{\mathrm{abs}_j} = \frac{d_{\max} - d_j}{d_{\max}} \tag{10}$$

The sum of all absolute weights is

$$s_w = \sum_{\forall j} w_{\mathrm{abs}_j} \tag{11}$$

From this, it is possible to compute relative weights as follows:

$$w_{\mathrm{rel}_j} = \frac{w_{\mathrm{abs}_j}}{s_{\mathrm{w}}} \tag{12}$$

The relative weights are numbers in the range [0.0, 1.0]. Their sum is 1.0.

As can be seen, this algorithm depends only on current and past values of the fuzzy membership, class and side functions of the *input* variables. The output variable does not appear anywhere in these equations. Consequently, the BNN that emulates this algorithm cannot be application-dependent, i.e., does not contain any information about the input/output relationship of the system to be modelled.

According to [15], each discrete pattern should ideally be observed five times during the training of the CNN. Thus, the training data should ideally consist of 5 × 512 = 2560 records. In reality, there are not even 10% of these data available. Thus, many of the patterns will in fact be observed only once or twice during the training period, and some will not be observed at all. It is necessary to discuss what the CNN should do if it does not contain sufficient data records.

In [15], it was proposed simply to search the input space for the five nearest neighbours, irrespective of whether they belong to the same class or not. If a particular class and side pattern for the combinations of inputs has not been observed at least five times, the training data set is searched for input patterns that vary only in their side values, and these records are then used as neighbours. If there are even not enough of those patterns available, neighbouring cells of the input space are searched.

Since Eq. (5) depends not only on the membership values of the previously observed neighbour inputs, but also on their class and side values, all these quantities would have to be passed from the CNN to the BNN to enable the BNN to implement Eq. (5). In order to avoid this, the old membership values are modified as follows:

$$\bar{M}_{ij} = s_i((c_i - c_{ij}) + (s_i - s_{ij}) + s_{ij} M_{ij}) \tag{13}$$

where $c_{ij}$ stands for the current class value of the qualitative triple $\langle c_{ij}, s_{ij}, M_{ij} \rangle$, and $c_i$ stands for the class into which the qualitative triple is to be mapped. The corrected qualitative triple has a class value of $c_i$, a side value of $s_i$, and an extended membership value of $\bar{M}_{ij}$. Equation (13) is not valid for $s_i = 0$, but this does not matter, since, while individual data points may have accidentally a value of $M_{ij} = 1$ together with $s_{ij} = 0$, the 'boxes' inside the CNN into which the five neighbours must be mapped never assume a value of $s_i = 0$.

This modification allows us to rewrite Eq. (5) as

$$p_{ij} = c_i + s_i (1.0 - \bar{M}_{ij}) \tag{14}$$

thus, only the old membership values are still needed. Moreover, the class value, $c_i$, can be dropped from Eqs (6) and (14), since this is only an additive constant that will drop out again when computing Eq. (9).

In this way, the CNN can be filled up until it is fully exhaustive. The Grossberg layer of *all* (class/side) combinations refers to exactly five old (corrected) membership values to be used by the BNN. Thus, in the neural network implementation of fuzzy inductive reasoning, fuzzy inferencing is done with a predefined set of five neighbours (5N) for each (class/side) combination, rather than searching the input space for the nearest five neighbours (5NN) as was proposed in [15]. The advantage of this modification is that no search is needed. The FRCNN architecture will reproduce the five required $\tilde{M}_{ij}$ values instantaneously.

The BNN shown in Fig. 8 is a general-purpose BNN that can be used for any system with up to five input variables and up to five data records per input variable in the training data.

The BNN is a cascaded feedforward neural network. Each subnetwork implements one of the fuzzy inferencing equations. It consists of an input layer, two hidden layers, and an output layer, trained by means of backpropagation. The overall BNN contains 32 layers with altogether 958 neurons.

According to the 5NN (or 5N) algorithm, the membership function of the output can now be computed as follows:

$$M = \sum_{\forall j} w_{\text{rel}_j} M_j \tag{15}$$

where $w_{\text{rel}_j}$ are the outputs of the BNN, and $M_j$ are the – until now unused – membership functions of the outputs of the five neighbours used in the fuzzy inferencing process.

This equation can be realised by a neural network with a single linear neuron and five inputs. The network will be called *linear neural network* (LNN). The old membership values of the output variable from the training data set, $M_j$, are fed into this LNN as inputs, whereas the relative weights are multiplied into the LNN as weights.

It is possible that the five neighbours predict *different* (class/side) combinations. In this case, the most likely (class/side) combination is chosen, and the membership values accompanying other predictions are corrected using Eq. (13). If the output of the LNN happens to be outside the range [0.5, 1.0], the assumption was incorrect, and needs to be revised. This can, for example, be accomplished by the defuzzifier (DFF) that generates an approximation of the quantitative output, $y$.

After the FRCNN has been built, testing data can be generated to validate the approach. In accordance with [16], 125 testing data were selected that are uniformly distributed over the range [1.5, 5.5] × [1.5, 5.5] × [1.5, 5.5]. The testing data set is completely distinct from the training data set.

The same performance index, the average percentage error (APE), was used that had been applied in [16]:

$$\text{APE} = \frac{1}{k}\left(\sum_{\ell=1}^{k} \frac{|y(\ell) - \hat{y}(\ell)|}{|y(\ell)|}\right) \times 100\% \tag{16}$$

where $k$ is the number of testing data pairs, $y(\ell)$ is the $\ell$th true output value, and $\hat{y}(\ell)$ is the $\ell$th output value as computed by the FRCNN.

The FRCNN approach leads to a value of APE of 11.27%. The APE value can be improved to 6.18% if the five nearest neighbours (5NN) are used instead of just any five neighbours (5N). However, no neural network implementation of the 5NN approach has been found yet, thus the larger APE value is what should be used to get a fair estimate of the method. This value is considerably larger than the APE values reported in the literature for this problem [16–19]. The 5NN value is about the same as the value reported for the GMDH method [17]. The best APE value reported so far is 1.066% [16].

Yet, the results presented here may still be attractive. Most of the techniques reported earlier require very long training periods. In contrast, the FRCNN architecture does not require any training at all (beside from feeding it once with the training data set). Thus, the FRCNN architecture is well suited for real-time applications.

Moreover, the application presented here is notorious for its lack of available training data. If the FRCNN architecture is presented with a decent amount of training data, its performance will be excellent. The example *is* meaningful though as stated, since it tests the ability of a scheme for generalising knowledge. As mentioned in [5], the XOR problem is utterly uninteresting – as uninteresting as memorising the Manhattan phonebook and then reciting from it. The basic CNN has no knowledge generalisation abilities whatsoever. It is just a smart memorising scheme. The knowledge generalisation capabilities of the FRCNN architecture are caused by the fuzzification exclusively, but good knowledge generalisation schemes usually imply some sort of learning, and the FRCNN architecture was, on purpose, designed as a synthesis tool and *not* as a learning tool.

## 7. Dealing With Variable-Structure Systems

Many technical applications are slowly time-varying. This means that their behavioural patterns change slowly over time. The neural network mimicking the behaviour of such a system needs to be trainable 'on the job' in order to update its behaviour on the basis of observed modifications in the system's behaviour. This means, the network should be able to *adapt*.

Minor behaviour adaptations are reflected by a modification in the fuzzy inferencing, while the predicted classes remain the same. This effect can be implemented in the FRCNN very easily. Whenever a new test pattern has been processed, its membership values are simply copied into the past data vector of the Grossberg layer of the CNN, replacing one of the older data records. A round-robin strategy can be used to ensure that always the oldest data point is replaced by the newest. Of course, membership values outside the range [0.5, 1.0] (data borrowed from other class/side combinations) are replaced with highest priority.

An FRCNN that implements this adaptation mechanism will henceforth be called a *fuzzy adaptive recurrent counterpropagation neural network* (FARCNN).

Major behavioural changes are reflected by a modification in the selector stage. Other variables are now needed to predict the system's behaviour correctly. This is much more difficult to realise in the network, since a modification of the selector stage calls for an entirely new set of training data, which means that the network cannot be used again at once. For this reason, it may be better to include all variables that might be potentially useful in the selector stage from the beginning. However, this means that the number of input variables of the CNN will grow, and with it also the number of legal input states. It can therefore no longer be assumed that sufficient training data will be available to exhaust all legal input states.

The following adaptation mechanism is proposed to cure this problem. Once a new pattern has been processed, a search determines whether this pattern has ever been seen before. If this is not the case, an additional perceptron is added to the Kohonen layer of the CNN, and the new input/output pattern is added as an additional row of its $\mathbf{W}^1$ matrix and as an additional column of its $\mathbf{W}^2$ matrix.

## 8. Mixed Qualitative/Quantitative Modelling

In this section, a complete example of a qualitative model of a continuous-time non-linear dynamic process shall be presented. In order to be able to compare the results obtained with the state-of-the-art, again an example from literature was chosen [8].

Figure 9 shows the position control of a hydraulic motor with a four-way servovalve. The detailed equations of this differential equation model are of no immediate concern here. They can be found in [8]. The controller is a P-controller with limiter; the mechanics of the servovalve contain two fast mechanical time constants (velocity and position of the piston) and another non-linearity (limiter); the hydraulics contain two hydraulic time constants (for the pressures in the two chambers) and the non-linear relationship between pressure and flow rate in the valve; the mechanics of the hydromotor contain two slow mechanical time constants (angular velocity and position of the rotor); and the measurement dynamics contain one fast electrical time constant. Thus, the overall system is of seventh order and highly non-linear.

The quantitative model of this system will be used as a gauge to check the accuracy of a mixed quantitative/qualitative model in which the mechanics of the servovalve and the hydraulics are replaced by a qualitative model (a FARCNN), whereas the controller, the mechanics of the hydromotor, and the measurement dynamics are kept as differential equation models.

It is important to realise that the qualitative model, which effectively replaces a differential equation model by a difference equation model, contains implicitly a sample-and-hold circuit. Thus, the Shannon sampling theorem must be satisfied. This dictates a sampling rate of $\Delta t = 0.0025$ s. However, in order to capture the dynamics of the hydraulic time constants, a much larger sampling rate would be indicated. In fact, the FIR methodology [8] suggests the following qualitative model:

$$T_m(t) = f(u(t), \omega_m(t - 0.05), T_m(t - 0.05)) \tag{17}$$

Thus, both the angular velocity of the motor, $\omega_m$, and the motor torque $T_m$, need to be delayed by $20\Delta t$. Equation (17) also indicates, which selector function should be used. The qualitative model will operate on three analogue inputs only, the control input, $u$, at the current time, the angular motor velocity, $\omega_m$, delayed by $20\Delta t$, and the motor torque, $T_m$, also delayed by $20\Delta t$.

An FARCNN is identified using the techniques that were outlined earlier in this paper. The data used for the qualitative model are records of the three variables $u$, $\omega_m$ and $T_m$, stored once every 0.0025 s. To this end, binary random noise (BRN) was applied to the quantitative model in closed-loop: $\theta_{set}$ = BRN. The clock for the noise generator was 0.1 s, i.e., the value of $\theta_{set}$ was randomly assigned a new value of either 1.0 rad or $-1.0$ rad once every 0.1 s. Figure 10 shows the FARCNN used in this case. Both the BNN and the LNN are *exactly* the same as had previously been used in the non-linear function example.

The quantitative model was simulated across 2.5 s of simulated time corresponding to 1001 data records. The first 2.25 s, or 901 data records, were

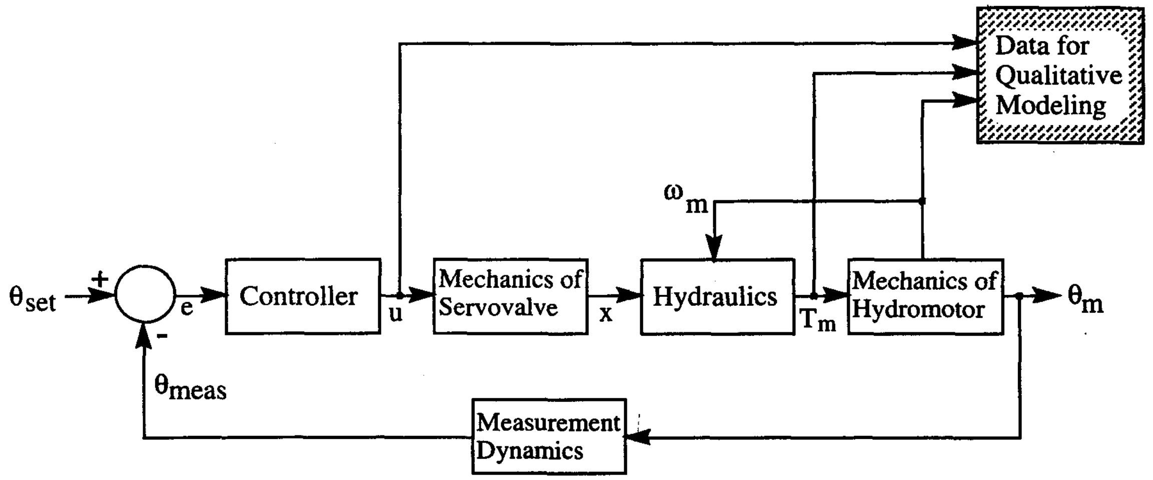

**Fig. 9.** Position control of a hydraulic motor.

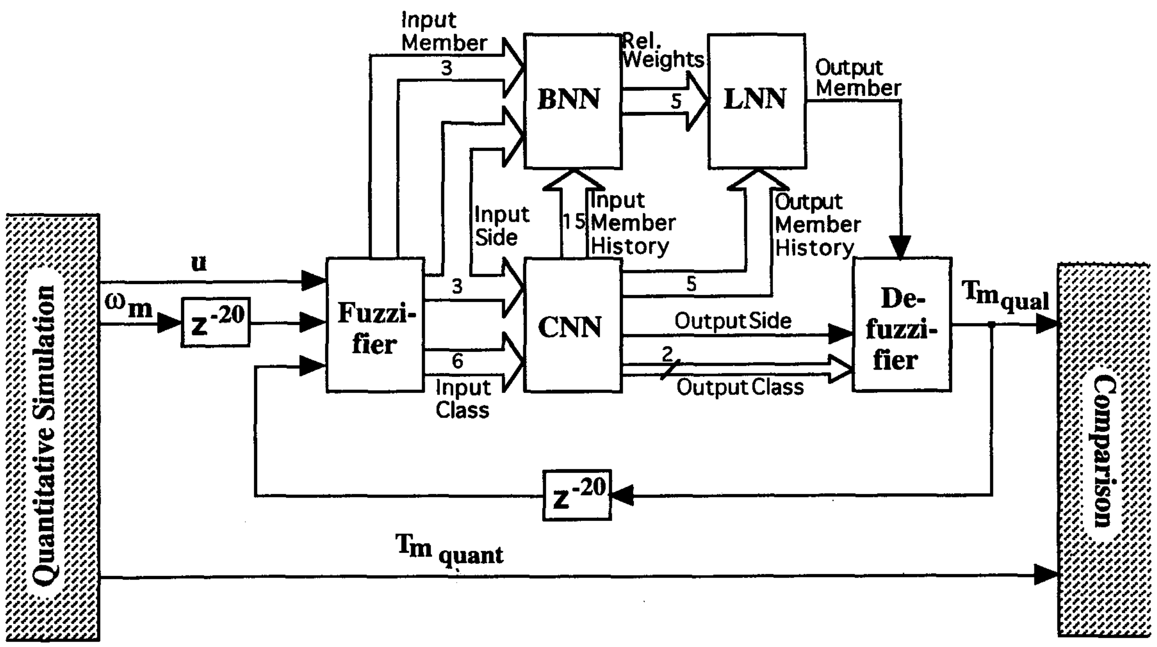

**Fig. 10.** FARCNN for hydraulic motor.

used as training data, and the final 100 data records were used for model validation. Figure 11 shows a comparison of the last 100 values of the torque, $T_m$, stored during the quantitative simulation (solid line) with the values, $\hat{T}_m$, that were predicted by the FARCNN when used in open-loop (dashed line). In this experiment, $u$ and $\omega_m$ were driven by the data previously stored during the quantitative simulation, and only $T_m$ was fed back into the FARCNN.

During the next experiment, the hydraulic motor control loop was closed around the FARCNN. Here, $u$ and $\omega_m$ were generated on-line by the quantitative model, and $T_m$ was fed back from the FARCNN to the quantitative model, thereby influencing future values of $u$ and $\omega_m$. Figure 12 shows the closed-loop configuration. The differential equation models describing the electrical and mechanical components of the overall control system had now to be simulated simultaneously with the FARCNN. Figure 13 shows the results of this experiment. The solid line shows the results from the purely quantitative simulation, whereas the dashed line shows the results from the mixed quantitative and qualitative simulation run.

As was to be expected, the results are slightly

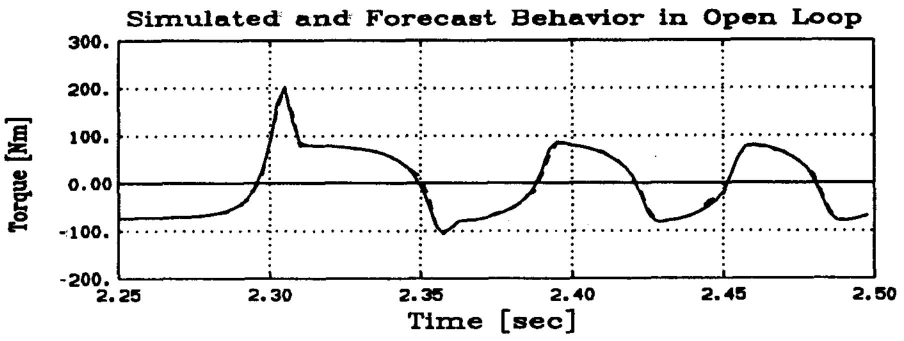

**Fig. 11.** Qualitative simulation in open loop.

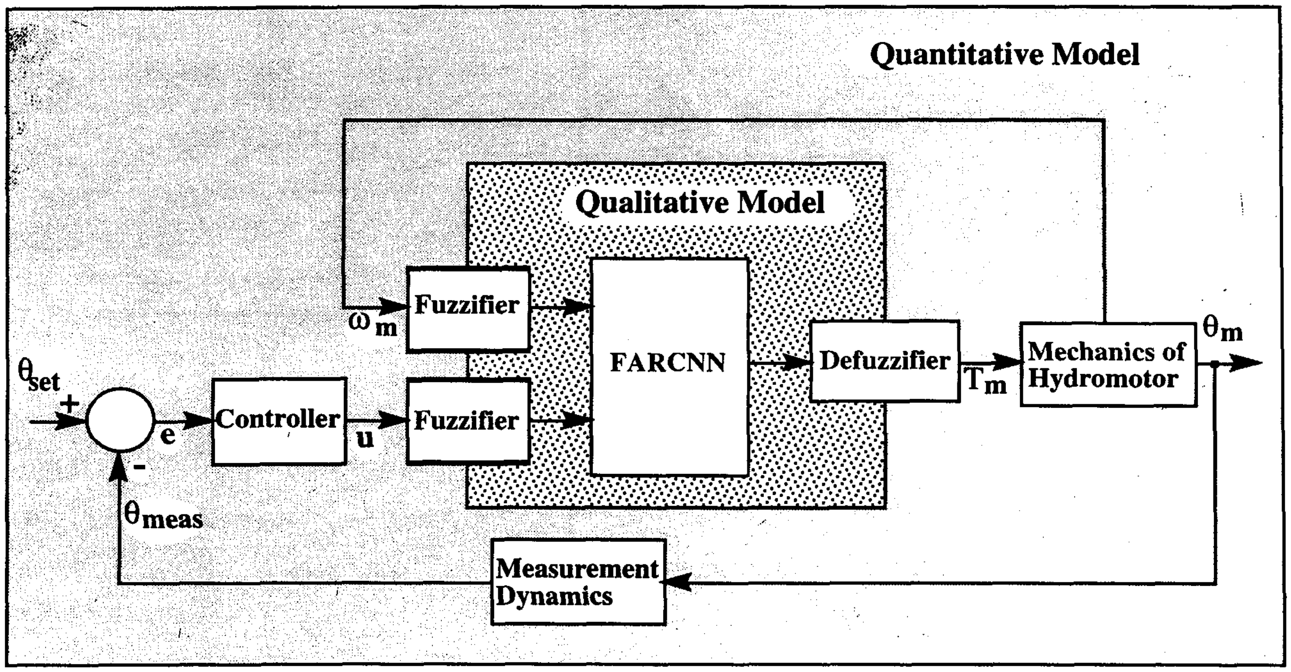

**Fig. 12.** Mixed quantitative/qualitative system configuration.

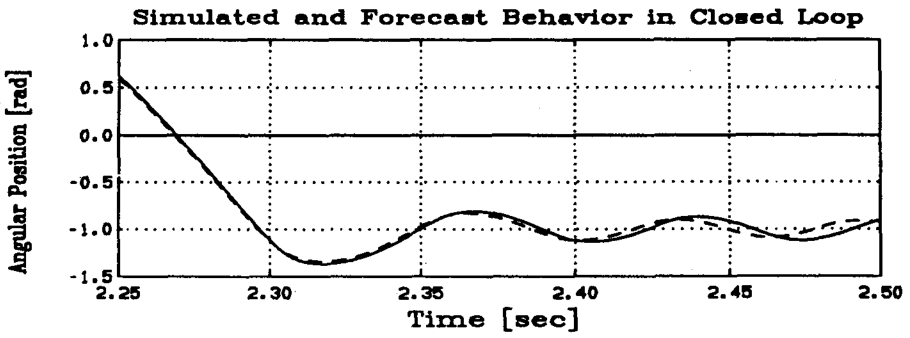

**Fig. 13.** Mixed quantitative/qualitative simulation in closed loop.

better for the open-loop case, but even the closed-loop results are excellent. The results are almost identical to those shown in [8]. This is not further surprising, since FARCNNs are simply a means for fast implementation of the FIR qualitative modelling methodology for real-time applications. The difference between the 5N and 5NN fuzzy inferencing schemes is not critical here, since sufficient training data were made available to the FARCNN architecture. To compensate for the loss in precision, the number of classes in the fuzzification was increased from five to eight, which is still realizable with fuzzy three-bit converters.

## 9. Summary and Conclusions

A new neural network architecture was presented that allows us to implement rapidly a large variety of different static and dynamic functional relationships with minimal learning needs.

The basic CNN architecture provides a parallel implementation mechanism for binary truth-table lookup. As a typical application of this technology, an arbitrary combinatorial digital circuit can be realised quickly and efficiently. A single pass through the Kohonen and Grossberg layers will suffice to produce the desired outputs to an arbitrary set of inputs. The gate delays will thereby be minimised.

The GCNN generalises the CNN architecture to deal with multi-valued logic. This enhanced architecture can for example be used for efficient implementation of finite state machines (FSMs). Each pass through the network corresponds to one clock of the FSM. This allows for real-time implementation of Petri net simulators.

Another application of this technology may be the real-time implementation of forward chaining expert systems. The Kohonen layer can implement the conditions for rules to be fired, while the Grossberg layer implements the consequences of such firings.

The RCNN enhancement allows to incorporate memory into the network. It enables the transition from dealing with purely static relationships only to being able to handle fully dynamic functional relationships. It enables us to implement electronic circuits containing flip-flops efficiently, it allows us to realise second generation time-dependent expert systems for real-time applications, and it provides us with the means to implement crisp inductive reasoning models.

The processing of continuously changing information causes considerable difficulties to the inherently digital network architecture. However, a solution to this dilemma was proposed involving newly designed fuzzy A/D and fuzzy D/A converters and a standardised backpropagation neural network (BNN). The problem of propagating fuzzy membership values across the network was addressed, and a network architecture for accomplishing the fuzzy signal propagation was presented.

## References

1. de Albornoz A, Cellier FE. Variable selection and sensor fusion in automated hierarchical fault monitoring of large-scale systems. In: Piera Carreté N, Singh MG (eds). Proc QUARDET'93, IMACS workshop on qualitative reasoning and decision technologies, Barcelona, Spain, 16–18 June, 1993, pp 722–734
2. de Kleer J, Williams B. Diagnosing multiple faults. Artif Intell 1987; 32(1): 97–130
3. Nebot A, Cellier FE, Linkens DA. Controlling an anaesthetic agent by means of fuzzy inductive reasoning. In: Piera Carreté N, Singh MG (eds). Proc QUARDET'93 IMACS workshop on qualitative reasoning and decision technologies, Barcelona, Spain, 16–18 June 1993, pp 345–356
4. Uckun S, Dawant BM. Qualitative modeling as a paradigm for diagnosis and prediction in critical care environments. Artif Intell Med 1992; 4(2): 127–144
5. Weigend AS, Gershenfeld NA. Time series prediction: forecasting the future and understanding the past. Addison-Wesley, Reading, MA, 1994
6. Cellier FE. Qualitative modelling and simulation – promise or illusion. In: Nelson BL, Kelton WD, Clark GM (eds). Proc winter simulation conference, Phoenix, AZ, 8–11 December, 1991, pp 1086–1090.
7. Zeigler BP, Chi SD. Symbolic discrete-event system specification. IEEE Trans Syst Man Cybern 1992; 22(6): 28–43
8. Cellier FE, Nebot A, Mugica F, de Albornoz A. Combined qualitative/quantitative simulation models of continuous-time processes using fuzzy inductive reasoning techniques. Int J Gen Syst 1995; 25(1)
9. Linkens DA, Greenhow DA, Asbury AJ. An expert system for the control of depth of anaesthesia. Biomed Meas Inform Control 1986; 1(4): 223–228
10. Linkens DA, Rehman HU. Nonlinear control for anaesthetic depth using neural networks and regression. In: Proc ISIC'92, IEEE int symp on intelligent control, Glasgow, Scotland, UK, 11–13 August, 1992, pp 410–415
11. Hecht-Nielsen R. Neurocomputers. Addison-Wesley, Reading, MA, 1990
12. Nebot A, Medina S, Cellier FE. The causality horizon: limitations to predictability of behaviour using fuzzy inductive reasoning. In: Guasch A, Huber R (eds). Proc ESM'94, SCS European simulation multiconference, Barcelona, Spain, 1–3 June, 1994, pp 492–496
13. Cellier FE. Continuous system modeling. Springer, New York, 1991
14. Filev DP, Yager RR. A generalized defuzzification method via BAD distributions. Int J Intell Syst 1991; 6: 687–697
15. Mugica F, Cellier FE. A new fuzzy inferencing

method for inductive reasoning. In: Proc int symp on artificial intelligence, Monterrey, Mexico, 20–24 September 1993, pp 372–379
16. Jang JS. ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 1993; 23(3): 665–685
17. Kondo T. Revised GMDH algorithm estimating degree of the complete polynomial. Trans Soc Instrum Control Engrs 1986; 22(9): 928–934
18. Sugeno M, Kang GT. 'Structure identification of fuzzy models'. Fuzzy Sets Syst 1988; 28: 15–33.
19. Takagi T, Hayashi I. NN-driven fuzzy reasoning. Int J Approx Reason 1991; 5(3): 191–212