S/3-67
/61813

N93-29529

A Neuro–Fuzzy Architecture for Real–Time Applications

p. 10

P.A. Ramamoorthy and Song Huang

Department of Electrical & Computer Engineering,
University of Cincinnati, M.L. #30
Cincinnati, Ohio 45221-0030
Tel: 513-556-4757, FAX: 513-556-7326
Email: pramamoo@nest.ece.uc.edu

Abstract

Neural networks and fuzzy expert systems perform the same task of functional mapping using entirely different approaches. Each approach has certain unique features. The ability to learn specific input–output mappings from large input/output data possibly corrupted by noise and the ability to adapt or continue learning are some important features of neural networks. Fuzzy expert systems are known for their ability to deal with fuzzy information and incomplete/imprecise data in a structured, logical way. Since both of these techniques implement the same task (that of functional mapping and we regard "inferencing" as one specific category under this class), a fusion of the two concepts that retains their unique features while overcoming their individual drawbacks will have excellent applications in the real world. In this paper, we arrive at a new architecture by fusing the two concepts. The architecture has the trainability/adaptibility (based on input/output observations) property of the neural networks and the architectural features that are unique to fuzzy expert systems. It also does not require specific information such as fuzzy rules, defuzzification procedure used etc., though any such information can be integrated into the architecture. We show that this architecture can provide a performance better than is possible from a single two or three layer feedforward neural network. Further, we show that this new architecture can be used as an efficient vehicle for hardware implementation of complex fuzzy expert systems for real–time applications. A numerical example is provided to show the potential of this approach.

1 Introduction

In general, fuzzy logic uses linguistic variables which are not crisply defined and logical relations between these variables to define the relationship between system inputs and outputs. On the other hand, neural networks use simple linear and nonlinear building blocks, interconnections among these blocks and training or learning procedures to obtain the system input–to–output mapping from large input/output samples. Thus, even though the research on neural networks and fuzzy logic have progressed for all practical purposes on two independent paths, it can be seen that both the architectures serve as models for arbitrary nonlinear mapping (f: $x \rightarrow y$, where x represents the input vector, y the output vector and f, the nonlinear transformation). Hence, it is important to understand the similarities and differences between these two approaches and the strengths/drawbacks of each. More importantly, it would be desirable to arrive at a hybrid approach/architecture that inherits the unique strengths of each without their shortcomings. In this paper, we show how such an architecture can be arrived and what are its important features. First, we provide a background to the problem in section 2 and present the new architecture in section 3. In section 4, we present results of simulation using the new architecture.

2 Background

2.1 Multi–layer Feedforward Networks

A neural network can be considered as a system that maps an input u, a vector of size N, into an output y, a vector of size M, by the function $f : u \rightarrow y$ [1]. The mapping is performed in the network or system by weighting each and every input, summing the results, subtracting a bias value and passing the result through a nonlinear function which may produce a binary or bipolar or continuous value (between -1 to 1) for each output. Thus, it can be noticed that a neural network is nothing but a non-linear network. The mapping function f is assumed to be unknown and is estimated from several numerical I/O samples (u_i, y_i) through the training procedure.

Above, we described a one-layer feed-forward model of a neural network. It is widely assumed that the Kolmogrov's theorem on functional approximation is a proof that a two-layer neural network is sufficient for approximating arbitrary non-linear systems given sufficient number of hidden nodes [2]. But, it is only an existence proof and does not tell us how to arrive at the network. In fact, there surfaced questions as to whether this theorem itself is applicable to the problem at hand [3], but we are not concerned about that issue here. From our perspective, a neural network is a non-linear system with interconnected neurons, which maps inputs into the outputs via the non-linear function f, and the function f is not given or known but estimated from a set of numerical I/O samples.

2.2 Cerebellar Model Articulation Controller Neural Networks

Another example of a feedforward neural network is that of the Cerebellar Model Articulation Controller (CMAC). An example of this network is shown in Figure 1 for a simple two input and one output system. This neural network was introduced by Albus [4, 5, 6] and seems to be getting renewed attention through the work of Miller, et al., [7, 8], Ersu, et al., [9] and Moody [10]. The nonlinear mapping is achieved in CMAC through nonlinear building blocks such as input sensors (a simple range detector — that is, each sensor produces an output of 1 if the input value falls in a certain range and 0 otherwise), AND gates (state space detectors) and OR gates (multiple field detectors) and linear weighting and summing blocks. It is claimed that CMAC can be an alternative for backpropagation networks [1] to achieve better performance [8]. Since backpropagation is basically a gradient descent technique applied to a multilayer nonlinear network, it needs a large computation time, converges slowly for large systems, and has an error surface which may contain local minima. The CMAC network contains a single linear feedforward network (that has to be trained) and hence does not require error propagation etc. and therefore can learn the mapping rather quickly. Miller, et al., recently modified the original CMAC architecture [11] where it is suggested that: 1). The input sensors implement local receptive fields with tapered sensitivity functions (that is the sensor output is 1.0 if the input is in the center of the receptive field, and the output decreases linearly towards 0.0 for inputs near the edges of the fields). 2). The state-space detectors can be considered as analog units (multiplication rather than logic AND gates) with the property that the unit output is 1.0 if all inputs are 1.0, while the unit output

[1] Backpropagation refers to an approach used to train multilayer networks and can be applied to any network. Hence it is not correct to call the multilayer perceptron network as a backpropagation network. We use it here as it has become a common practice.

decreases to 0.0 if any input decreases to 0.0. And 3). The multiple field detectors can be considered as simple summing units (rather than logic OR gates). The network output is then the sum of products of a certain number of non-zero multiple field detector outputs and the corresponding weights. It is indicated that the modified CMAC architecture has better properties than the original CMAC because the modified version provides continuous instead of piece-wise function approximations.

2.3 Fuzzy Logic and Fuzzy Expert Systems

The fuzzy systems can be also considered as implementing a mapping function f: $u \rightarrow y$ [12 - 16]. The mapping is effected via:

1. Splitting the input(s) and output(s) total–range of values into a number of subsets or ranges which can possibly overlap.
2. Assigning membership functions corresponding to these sets for all range of values of the variables. Together these sets can be considered as fuzzy sets where the membership functions denote the degree of belongings of a particular input or output value to the various fuzzy sets of those inputs or outputs.
3. Defining Boolean relationships among the input fuzzy sets and output fuzzy sets. These Boolean expressions identify the output fuzzy sets under which the expected outputs might fall when the inputs fall under certain input fuzzy sets.
4. Procedure (also known as defuzzification) to find the final or crisp output(s) from the output fuzzy sets (that are selected or activated) and the various membership functions.

The steps involved in implementing a fuzzy expert system is shown in Figure 2. From the figure, it can be noted that the functional mapping is achieved in a fuzzy expert system through three well defined sub–blocks. We will look into this architecture in the next section.

3 New Neuro–Fuzzy Architecture

Let us examine more closely the steps that would be involved in hardware implemention of a fuzzy expert system. Let M, N be the number of inputs to and outputs of the system, m_i ($i = 1$ to M), the number of fuzzy sets for the input i and n_j ($j = 1$ to N), the number of fuzzy sets for the output j. We will assume that the inputs and outputs are represented in a fixed-point weighted binary representation with B bits for each variable. The inputs can then be converted into an unweighted binary representation (with $Q = 2^{**}B$ binary lines for each input and only one bit "on or 1" for any input at any given time) using M number of B to $2^{**}B$ line decoders as shown in Figure 3. Now, given the exact input values, the first step in the implementation of a fuzzy expert system is to identify the fuzzy sets under which these input values fall. In a hardware implementation, this can be achieved by assigning one bit for each fuzzy set such that a particular bit gets turned on if and only if the input values fall under the range of that particular fuzzy set. We can call these bits as input fuzzy set pointers (IFSP) and there will be $m = \sum m_i$ IFSPs. The logic for conversion from the input values to (unweighted binary representation) IFSPs is then simply a set of m_i OR gates for the ith input as shown in the figure.

Having identified the input fuzzy sets to which the given values of the inputs belong, the next task is the identification of the corresponding output fuzzy sets and this is accomplished through the fuzzy rules or fuzzy associative memories (FAMs). This step can be implemented in hardware by assigning bits to identify the various output fuzzy sets as we did for the case of input fuzzy sets. Thus, we will have $n = \sum n_i$ output fuzzy set pointers (OFSPs) and the values of these binary variables will depend upon the values of the IFSPs. Since any binary variable in general can be represented in a two-level (or three level if we consider logical "negative" as one level [2]) sum-of-product expression involving the input binary variables, the fuzzy inferencing can be implemented in a two level logic as shown in the figure.

The defuzzification process makes use of the input values, corresponding input membership function values and the OFSPs, and the membership functions of the selected output fuzzy sets to produce the final outputs (see the third block of Figure 3). This block is more complex. However, it can be sub-divided into a number of sub-blocks or smaller sub-networks as shown in Figure 3B [3]. Here, we have n sub-networks with one output fuzzy set pointer acting as enable/disable signal for each network. They generate intermediate outputs which are combined (simple addition, for example) to produce the final outputs as shown in Figure 3B.

From the above discussions, we find that a hardware representation of a fuzzy expert system involves three separate blocks where each block has a unique function. The first two blocks and the sub-networks of the third block can in turn be 2 or 3 layer feedforward networks [4]. Thus, a fuzzy expert system can be considered as consisting of a number of multilayer feedforward networks with structured interconnections between them. Therefore, it is quite conceivable that fuzzy expert systems can provide a superior performance for functional mapping as compared to a single 2 or 3 layer network [5]. Kong and Kosko [17] illustrated this point through an example. Similar arguments can be made while comparing CMAC neural networks with fuzzy expert systems [6].

The superior performance of fuzzy expert systems can be attributed to the use of additional information as compared to that for multilayer neural networks. In the case of neural networks, we use the input/output samples, a fixed architecture (or a time evolving architecture as in [19]) and a training procedure. In the case of fuzzy expert systems, additional information such as fuzzy sets, fuzzy rulebase etc. are used to obtain the mapping. Some of the information such as the number and the ranges of fuzzy sets, membership functions can be obtained from the problem at hand [7]. Thus, we can use such information, the derived architecture (shown in Figure 2) and training procedures to implement any functional mapping. This trainable architecture can be called a "Neuro-Fuzzy Architecture". The advantages of such an architecture are:

A structure consisting of smaller networks that can be trained easily and faster;

[2]The fuzzy rules do not use negation (input not falling under the range of a fuzzy set) and perhaps is a limitation of the classical fuzzy expert systems. This has to be researched further.

[3]There are many different possibilities and we discuss only one.

[4]This is due to the fact that any mapping can be achieved by multilayer feedforward networks.

[5]Cybenko has showed mathematically that a 3-layer network is sufficient for any functional mapping. But the paper dose not address the limits on the error of approximation.

[6]In another paper, we show that a CMAC network can be considered as a special case of a fuzzy expert system [18].

[7]The initial choice may not be optimal. However, it can be argued that the incorporation of some known information is better than incorporating none.

Incorporation of information about the problem [8];

Adaptibility;

Identification of the fuzzy rules: Since there is a one-to-one correspondence between the blocks and the tasks in a fuzzy expert system implementation, we can train the second block if the fuzzy set regions are given and use that block to identify the fuzzy rulebase;

As an vehicle for hardware implementation: That is, rather than using a language based processing (with its associated MFLOPS or mega fuzzy logic operations per second ratings), we can use the new architecture as a digital or hard-wired implementation of a fuzzy expert system. Such an implementation will have a tremendous edge in real-time applications since the number of rules increases exponentially with a linear increase in the number of inputs. For example, if we assume that there are 5 fuzzy sets per input, the number of rules for a five input system and a ten input system will be 3125 and 9,765,625 respectively. Perhaps due to this problem, fuzzy expert systems considered in the literature are mostly two input systems or separability is assumed when there are more than two inputs;

Modeling of complex systems. By modeling the block corresponding to the rulebase by a 3 layer feedforward neural network and the defuzzification block by a number of neural networks, we will be able to model complex systems than is possible based on the presently used approaches.

4 Example

Here, we consider the problem of designing a controller to successfully back up a truck to a loading dock from any reasonable initial location. This problem was solved earlier by Nguyen and Widrow [20] using a two-layer neural network architecture with 26 nodes and later by Kong and Kosko [17] using a fuzzy expert system. The details of the problem are shown in Figure 4. For this example, we assume that the fuzzy sets (of inputs and outputs), the corresponding membership functions and the fuzzy rules are known (shown in Figures 5 and 6) but the defuzzification procedure is unknown. Thus, there is no need to train the first two blocks of Figure 3A, but only the third block (and the corresponding sub-networks) needs to be trained. The desired outputs corresponding to a set of inputs are obtained using the centroid defuzzification method (see [17] for details) and are used in the training. Two different approaches are used in the training: 1) Input x, ϕ values and the IFSPs as the inputs to the networks and θ as the desired outputs and 2) Inputs x, ϕ, and the corresponding membership function values as the inputs and θ as the desired outputs. It should be noted here that both approaches do not use the output membership function values. Since more accurate results are needed when the truck is in the center area or near center area, we selected more samples for x-position around 50, and less samples to the extremes. The training samples of ϕ are chosen in the same fashion. This led to 34 x-positions and 72 ϕ angles. Thus 2448 samples are used to train the controller. The y-positions are not used in training, thus simplifying the training process. There are 7 sub-networks corresponding to the seven output fuzzy sets. The whole set of training samples are divided into 7 smaller

[8]Concepts such as representing/designing a larger system by a number of smaller subsystems are not new in engineering.

groups according to their belongings to the output fuzzy sets. The largest group contained 826 training samples and the smallest one has 271 samples. Some samples are used in more than one sub-network due to the overlapping of the output fuzzy sets. This brought the total training samples for all the sub-networks to 3624.

The training samples are normalized to the range of -0.5 to 0.5. We selected 10 second-layer nodes for every sub-network. The backpropagation algorithm is used for the training. The number of iterations for training varied from few hundreds (for smaller sample groups) to few thousands (for larger groups). The training converged in both the cases with the average squared errors from 0.0005 (for the samples chosen near the center) to 0.0015 (for the extreme sets). A truck trajectory produced using the trained neural network corresponding to case 1) is shown in Figure 7A, and Figure 7B shows one trajectory corresponding to the case 2). It can be noted both methods produce smooth trajectories as compared to the one generated by a two layer neural network (as shown in Kong and Kosko's paper [17] and shown in Figure 8A) for this particular initial condition. Further, the trajectories by these networks are very similar to the ones produced by the original fuzzy expert system (the teacher) as can be seen comparing Figures 7A and 7B with Figure 8B.

5 Conclusions

Using functional mapping as a common framework, we showed how neural networks and fuzzy expert systems can be merged to arrive at a new Neuro-Fuzzy architecture. The architecture has the trainability/adaptibility (based on input/output observations) property of the neural networks and the architectural features that are unique to fuzzy expert systems. It also does not require specific information such as fuzzy rules, defuzzification procedure used etc., though any such information can easily be integrated into the architecture. We showed that this architecture can provide a better performance than is possible from a single two or three layer feedforward neural network, and can be used as an efficient vehicle for hardware implementation of complex fuzzy expert systems for real-time applications. A numerical example is also provided to show the potential of this approach. Many variations of the architecture seem to be possible and further work needs to be done to exploit the potentials offered by the new architecture.

6 References

1. R. P. Lippmann, "An introduction to computing with neural nets", *IEEE ASSP Magazine,* pp. 4-22, April 1987.
2. G. Cybenko, "Approximation by superpositions of a sigmoidal function", Technical Report, University of Illinois, 1988.
3. F. Girosi and T. Poggio, "Representation Properties of Networks: Kolmogrov's Theorem," *Neural Computation,* Vol. 1, pp. 465-469, 1889.
4. J.S. Albus, "A theory of cerebellar functions", *Mathematical Biosciences,* Vol. 10, pp. 25-61, 1971.
5. J.S. Albus, "Theoretical and Experimental Aspects of a Cerebellar Model", PhD thesis, Univ. of Maryland, 1972.
6. J.S. Albus, "Data storage in the cerebellar model articulation controller", *J. of Dynamic Systems, Measurement and Control,* pp. 228-233, Sept. 1975.

7. W.T. Miller, "Non-linear learning controller for robotic manipulators", *Proc. SPIE, Intelligent Robots and Computer Vision,* Vol. 726, pp. 416–423, Oct. 1986.
8. W.T. Miller, F.H. Glanz, and L.G. Kraft, "CMAC: An associative neural network alternative to backpropagation", *Proc. IEEE,* pp. 1561-1567, Oct. 1990.
9. E.Ersu and H.Tolle, "Hierarchical learning control— an approach with neuron-like associative memories", *Proc. IEEE Conf. on Neural Infor. Proc. Systems,* Nov. 1988.
10. J.Moody, "Fast learning in multi-resolution hierarchies", in *Advances in Neural Information Processing,* edited by D.Touretzky, Morgan Kaufmann, 1989.
11. W.T. Miller, E.An, and F.H. Glanz, "The design of cmac neural networks for control", *Proc. 6th Yale Workshop on Adaptive and Learning Systems,* Aug. 1990.
12. L.A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes", *IEEE Trans. Systems, Man and Cybernetics,* Vol. SMC-3, pp. 28–44, 1973.
13. L.A. Zadeh, "Commonsense Knowledge Representation Based on Fuzzy Logic," *IEEE Computer,* Vol. 10, pp. 61–65, 1983.
14. L.A. Zadeh, *Selected Paper by L.A. Zadeh,* edited by R.R. Yager et al., John Wiley and Sons, 1987.
15. L.A. Zadeh, "Fuzzy logic", *IEEE Computer,* pp. 83–93, April 1988.
16. B. Kosko, "Fuzzy Entropy and Conditioning," *Information Sciences,* Vol. 40, pp. 165–174, 1986.
17. S.Kong and B.Kosko, "Comparison of fuzzy and neural truck backer-upper control systems", *Proc. of IJCNN 1990,* Vol. III, pp. 349-358, June 1990.
18. P.A. Ramamoorthy and Song Huang, "Cerebellar Model Articulation Controller Neural Network — A Simple Fuzzy Expert System in Disguise?", to appear in the Conf. Proc. of Artificial Neural Networks in Engineering, Nov. 1991.
19. G. Govind and P.A. Ramamoorthy, "A New Neural Network Structure and Algorithm for Nonlinear System Identification", to be submitted to *IEEE Transactions on Systems, Man, and Cybernetics,* 1991.
20. D. Nguyen and B. Widrow, "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," *Proceedings of IJCNN 1989,* Vol. 2, pp. 357-363, June 1989.

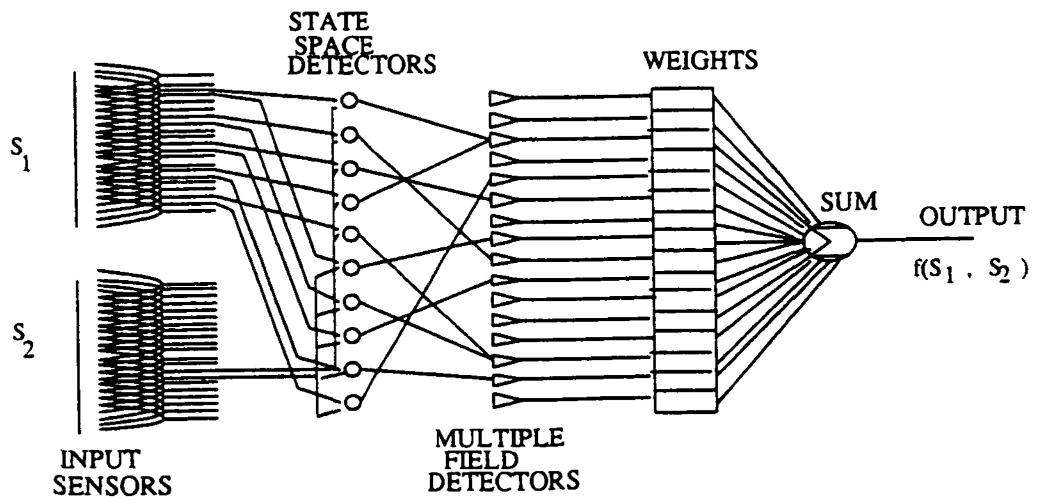


Figure 1. A simple CMAC system with two inputs and one output.

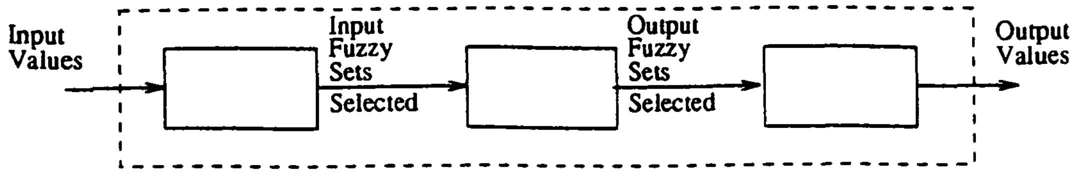
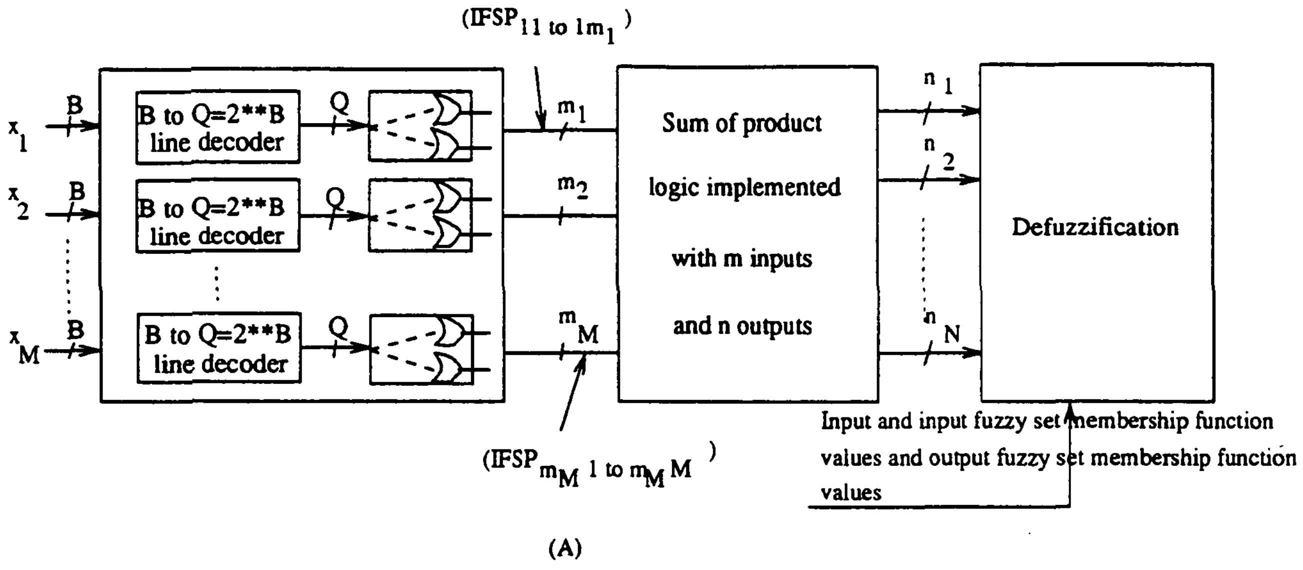


Figure 2. Block diagram of a fuzzy expert system.



(IFSP 11 to 1m1)
x1
x2
xM
B
B to Q=2**B line decoder
Q
m1
m2
mM
Sum of product logic implemented with m inputs and n outputs
n1
n2
nN
Defuzzification
(IFSP mM 1 to mM M)
Input and input fuzzy set membership function values and output fuzzy set membership function values
(A)

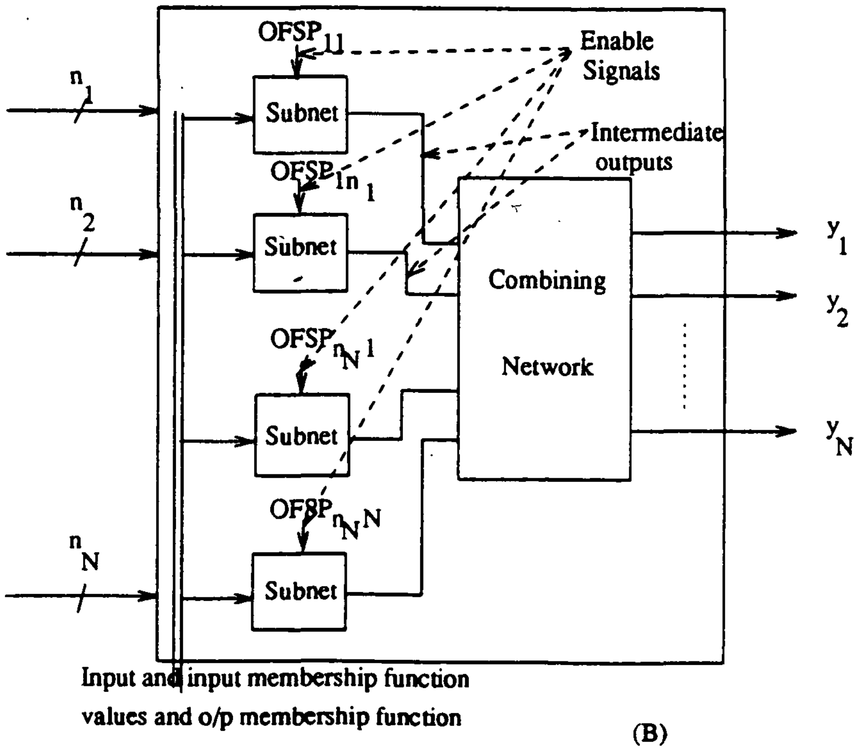
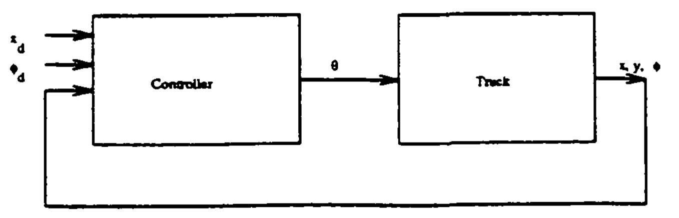


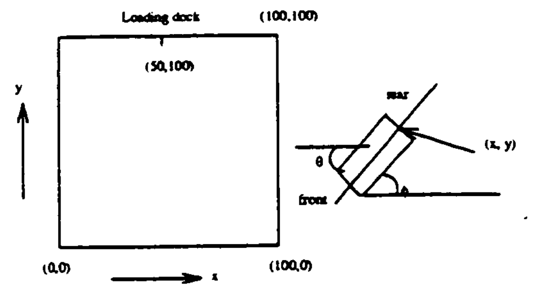
Figure 3. (A) Block diagram of the Neuro--Fuzzy system
(B) Detailed representation of the third block in Figure (A).



$$(x, y)_{new} = (x, y)_{old} + (dx, dy)$$

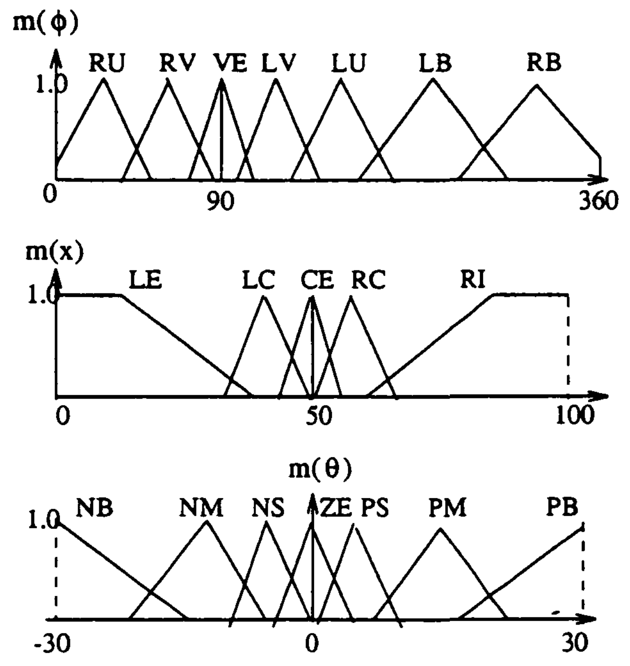
$$dx = r\cos(\phi + \theta)$$
$$dy = r\sin(\phi + \theta)$$

(A)



(B)

Figure 4. (A) Block diagram of a fuzzy expert system based controller to back up a truck. (B) Details of the loading zone and the truck.



Fuzzy Sets of TBU Problem

Angle ϕ	X-Position	Steering Signal θ
RB: Right Below	LE: Left	NB: Negative Big
RU: Right Upper	LC: Left Center	NM: Negative Medium
RV: Right Vertical	CE: Center	NS: Negative Small
VE: Vertical	RC: Right Center	ZE: Zero
LV: Left Vertical	RI: Rifgt	PS: Positive Small
LU: Left Upper		PM: Positive Medium
LB: Left Below		PB: Positive Big

Figure 5. The fuzzy sets and the membership functions of inputs and outputs of the fuzzy controller.

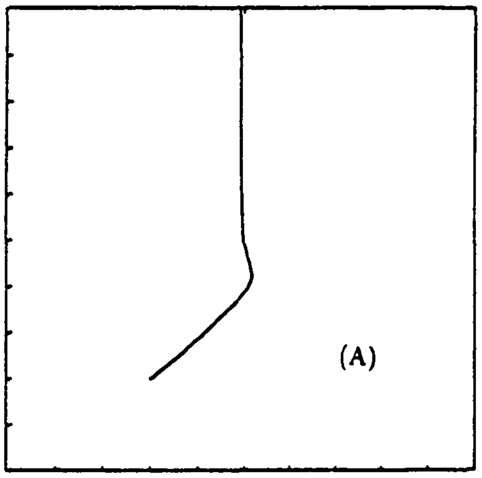
x - position

angle (ϕ)

PB	PB	PM	NB	NB
ZE	PS	PM	PB	PB
NB	NM	PS	PM	PB
NB	NM	ZE	PM	PB
NB	NM	NS	PM	PB
NB	NB	NM	NS	ZE
PB	PB	NM	NB	NB

Figure 6. The rule base for the fuzzy controller.

Truck Backer–Upper Controller (FES–NN)
$x = 30,\ y = 20,\ \phi = 30$
Input Fuzzy Set Pointers used for training



Truck Backer–Upper Controller (FES–NN)
$x = 30,\ y = 20,\ \phi = 30$
Membership function values used for training.

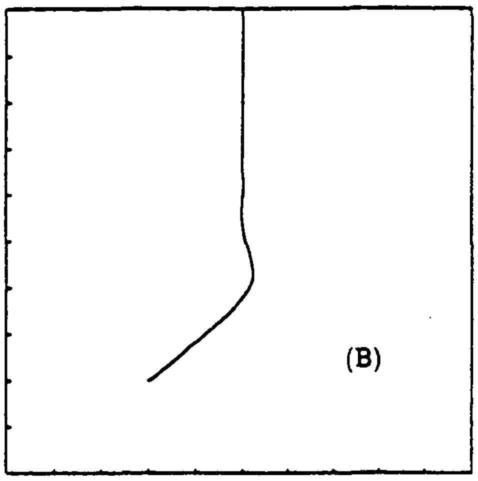
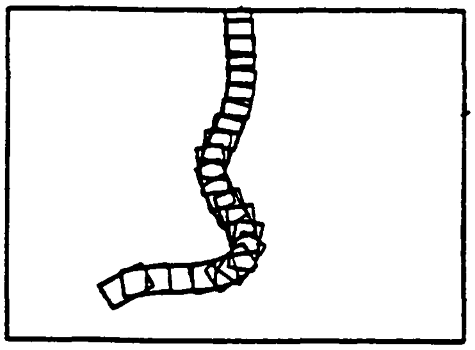


Figure 7 The trajectories of the truck using the Neuro-Fuzzy controller,

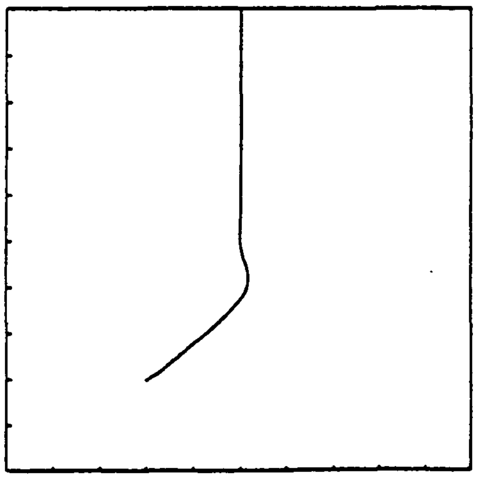
(A) Using Input Fuzzy Set Pointers in the training, and

(B) Using Membership function values.



(A)

Truck Backer–Upper Controller (FES)
$x = 30,\ y = 20,\ \phi = 30$



(B)

Figure 8 (A) A trajectory of the truck using a neural network controller.

(B) A trajectory of the truck using a fuzzy controller.