Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems

Nikola Kasabov and Brendon Woodford
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
(E-mail : nkasabov@otago.ac.nz, bjwoodford@infoscience.otago.ac.nz)

Abstract

The paper discusses the concept of intelligent expert systems and suggests tools for building adaptable, in an on-line or in an off-line mode, rule base during the system operation in a changing environment. It applies evolving fuzzy neural networks EFuNNs as associative memories for the purpose of dynamic storing and modifying a rule base. Algorithms for rule extraction and rule insertion from EFuNNs are explained and applied to a case study using gas furnace data and the iris data set.

Keywords: intelligent expert systems; evolving neuro-fuzzy systems; rule insertion; rule extraction; neural networks.

1.Introduction: On-line Adaptive Intelligent Expert Systems

The traditional expert systems, based on a fixed set of rules, have significantly contributed to the development of AI and intelligent engineering systems in the past two years. Despite their success, more flexible tools for dynamic rule adaptation, rule extraction from data, and rule insertion in a rule base are particulary needed when expert systems have to operate in an adaptive, on-line mode, and in a dynamically changing environment. The complexity and the dynamics of many real-world problems, especially in engineering and manufacturing, require sophisticated methods and tools for building adaptive intelligent expert systems (IES). Such systems should be able to grow as they work, to build-up their knowledge and refine the model through interaction with the environment. Seven major requirements of the next generation intelligent systems are discussed in [8, 9, 12, 13]. Here they are presented in the context of the new generation of intelligent expert systems (IES).

Knowledge is the essence of what an IES has learned. Knoweldge-Based Neural Networks (KBNN) are neural networks pre-structured to allow for data and knowledge manipulation, including learning from data, rule insertion, rule extraction, adaptation and reasoning. KBNN have been developed either as a combination of symbolic AI systems and NN [2, 5], or as a combination of fuzzy logic systems and NN [1, 3-7, 14, 16, 18, 19] Rule insertion and rule extraction operations are examples of how a KBNN can accommodate existing knowledge along with data, and how it can explain what it has learned. There are different methods for rule extraction well experimented and broadly applied so far [2-7]. Unfortunately, the methods proposed so far are not appropriate when the system is working in an adaptive, on-line mode in a changing environment.

The paper suggests a methodology of using evolving fuzzy neural networks EFuNNs for the purpose of building IES and illustrates it on two case study benchmark data sets.

2.Fuzzy Neural Networks FuNNs

2.1.The FuNN architecture and its functionality

Fuzzy neural networks are neural networks that realise a set of fuzzy rules and a fuzzy inference machine in a connectionist way [3-7, 16, 18]. FuNN is a fuzzy neural network introduced in [6, 7, 14] and developed as part of a comprehensive environment FuzzyCOPE/3 for building intelligent systems (available free on the WWW http://kel.otago.ac.nz/). It is a connectionist feed-forward architecture with five layers of neurons and four layers of connections. The first layer of neurons receives the input information. The second layer calculates the fuzzy membership degrees to which the input values belong to predefined fuzzy membership functions, e.g. small, medium, large. The third layer of neurons represents associations between the input and the output variables, fuzzy rules. The fourth layer calculates the degrees to which output membership functions are matched by the input data, and the fifth layer calculates exact values for the output variables' defuzzification. A FuNN has features of both a neural network and a fuzzy inference machine. A simple FuNN structure is shown in Fig.1. Through growth or shrinkage, the number of neurons in each layer can potentially change during operation. The number of connections can also be modified through learning with forgetting, zeroing, pruning and other operations [6, 14].

The membership functions (MF) used in FuNN to represent fuzzy values, are of triangular type, the centres of the triangles being attached as weights to the corresponding connections. The MF can be modified through learning, altering the centres and the widths of the triangles.

Several training algorithms have been developed for FuNN [6, 14]. Several algorithms for rule extraction from FuNNs have also been developed and applied [6, 17].

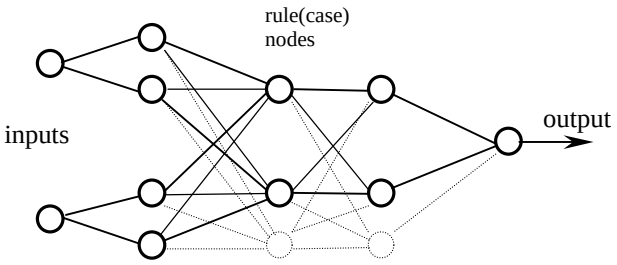

Figure 1: A FuNN structure of 2 inputs (input variables), 2 fuzzy linguistic terms for each variable (2 membership functions). The number of the rule (case) nodes can vary. Two output membership functions are used for the output variable.

One of them represents each rule node of a trained FuNN as an *IF-THEN* fuzzy rule.

3.Evolving Fuzzy Neural Networks EFuNNs

3.1.A general description

EFuNNs are FuNN structures that evolve according to the ECOS principles [8]. EFuNNs adopt some known techniques from [6, 15, 16] and from other known NN techniques, but here all nodes in an EFuNN are created during (possibly one-pass) learning. The nodes representing MF (fuzzy label neurons) can be modified during learning. As in FuNN, each input variable is represented here by a group of spatially arranged neurons to represent a fuzzy quantisation of this variable. For example, two neurons can be used to represent "small" and "large" fuzzy values of a variable. Different membership functions (MF) can be attached to these neurons (triangular, Gaussian, etc.). New neurons can evolve in this layer if, for a given input vector, the corresponding variable value does not belong to any of the existing MF to a degree greater than a membership threshold. A new fuzzy input neuron, or an input neuron, can be created during the adaptation phase of an EFuNN.

The EFuNN algorithm, for evolving EFuNNs, has been first presented in [9]. A new rule node rn is created and its input and output connection weights are set as follows: *W1(rn)=EX*; *W2(rn) = TE*, where *TE* is the fuzzy output vector for the current fuzzy input vector *EX*. In "one-of-n" EFuNNs, the maximum activation of a rule node is propagated to the next level. Saturated linear functions are used as activation functions of the fuzzy output neurons. In case of "many-of-n" mode, all the activation values of rule (case) nodes, that are above an activation threshold of *Athr*, are propagated further in the connectionist structure.

3.2.The EFuNN learning algorithm

The EFuNN evolving algorithm is given as a procedure of consecutive steps [8, 9, 10, 11, 13]:

1. Initialise an EFuNN structure with a maximum number of neurons and zero-value connections. Initial connections may be set through inserting fuzzy rules in a FuNN structure. FuNNs allow for insertion of fuzzy rules as an initialization procedure thus allowing for existing information to be used prior to the evolving process (the rule insertion procedure for FuNNs can be applied [6, 14]). If initially there are no rule (case) nodes connected to the fuzzy input and fuzzy output neurons with non-zero connections, then *connect* the first node *rn=1* to represent the first example *EX=x1* and set its input *W1(rn)* and output *W2(rn)* connection weights as follows: <Connect a new rule node *rn* to represent an example *EX* >: *W1(rn)=EX*;*W2(rn) = TE*, where *TE* is the fuzzy output vector for the (fuzzy) example it EX.

2. WHILE <*there are examples*> DO
   Enter the current example x_i , *EX* being the fuzzy input vector (the vector of the degrees to which the input values belong to the input membership functions). If there are new variables that appear in this example and have not been used in previous examples, create new input and/or output nodes with their corresponding membership functions as explained in [13]

3. Find the normalized fuzzy similarity between the new example *EX* (fuzzy input vector) and the already stored patterns in the case nodes *j=1,2..rn*: *Dj = sum (abs (EX - W1(j))/ 2) / sum (W1(j))*.

4. Find the activation of the rule (case) nodes *j, j=1..rn*. Here radial basis activation function, or a saturated linear one, can be used on the *Dj* input values i.e. *A1(j) = radbas(Dj)*, or *A1(j) = satlin(1 - Dj)*. Previous activation at the same layer can be taken into account [13]

5. Update the local parameters defined for the rule nodes, e.g. Short-Term Memory (STM), age, average activation as pre-defined.

6. Find all case nodes *j* with an activation value *A1(j)* above a sensitivity threshold *Sthr*.

7. If there is no such case node, then <*Connect a new rule node*> using the procedure from step 1.
   ELSE
   Find the rule node *inda1* that has the maximum activation value. (*maxa1*).

   (a) in case of "one-of-n" EFuNNs, propagate the activation *maxa1* of the rule node inda1 to the fuzzy output neurons. Saturated linear functions are used as activation functions of the fuzzy output neurons: *A2 = satlin (A1(inda1) * W2)*.

   (b) in case of "many-of-n mode", only the activation values of case nodes that are above an activation threshold of $Athr$ are propagated to the next neuronal layer.

8. Find the winning fuzzy output neuron *inda2* and its activation *maxa2*.
9. Find the desired winning fuzzy output neuron *indt2* and its value *maxt2*.
10. Calculate the fuzzy output error vector: *Err = A2 - TE*.
11. IF (*inda2* is different from *indt2*) or *(abs(Err (inda2)) > Errthr*) then *<Connect/create a rule node>*
12. Update: (a) the input, and (b) the output connections of rule node *k=inda1* as follows:
    (a) *Dist = EX-W1(k)*; *W1(k) = W1(k) + lr1*Dist*, where *lr1* is the learning rate for the first layer;
    (b) *W2(k) = W2 (k) + lr2*Err*maxa1*, where *lr2* is the learning rate for the second layer. If STM is used update the feedback connections in the rule layer.
13. Prune rule nodes *j* and their connections that satisfy the following fuzzy pruning rule to a pre-defined level: *IF (node (j) is OLD) and (average activation A1av(j) is LOW) and (the density of the neighbouring area of neurons is HIGH or MODERATE) and (the sum of the incoming or outgoing connection weights is LOW) and (the neuron is NOT associated with the corresponding "yes" class output nodes (for classification tasks only) THEN the probability of pruning node (j) is HIGH.*

    The above pruning rule is fuzzy and it requires that all fuzzy concepts such as OLD, HIGH, etc., are defined in advance. As a partial case, a fixed value can be used, e.g. a node is old if it has existed during the evolving of a FuNN from more than 60 examples.
14. Aggregate rule nodes into layer clusters (prototype) nodes (see [15])
15. END of the WHILE loop and the algorithm
16. Repeat steps 2-15 for a second presentation of the same input data or for ECO training if needed.

3.3.Learning strategies for ECOS

EFuNNs allow for different learning strategies to be applied, depending on the type of data available and on the requirements to the learning system. Several of them are introduced and illustrated in [8, 9, 12, 13]:

- Incremental, one-pass learning
- Using positive examples only
- Cascade eco-learning.
- Sleep eco-training.
- Unsupervised and reinforcement learning

4.Fuzzy rule insertion, on-line rule adaptation, and rule extraction from EFuNNs

EFuNNs store fuzzy data exemplars in their connections. Fuzzy exemplars cover patches in the problem space. These patches can be represented as fuzzy rules.

4.1.Rule insertion and rule adaptation

A FuNN and an EFuNN in particular can be adequately represented by a set of fuzzy rules through rule extraction techniques [6, 7, 10, 13]. The fuzzy inference is embodied in the connectionist structure. In this respect an EFuNN can be considered as an evolving fuzzy system. The rules that represent the rule nodes need to be aggregated in clusters of rules. The degree of aggregation can vary depending on the level of granularity needed. Sometimes, for explanation purposes, the number of rules needed, could be even less than the number of the fuzzy output values.

At any time (phase) of the evolving (learning) process, fuzzy, or exact rules can be inserted and extracted. Insertion of fuzzy rules is achieved through setting a new rule node for each new rule, such as the connection weights W1 and W2 of the rule node represent the fuzzy or the exact rule [13].

Example1: The fuzzy rule IF x1 is Small and x2 is Small THEN y is Small, can be inserted into an EFuNN structure by setting the connection weights of a new rule node to the fuzzy condition nodes x1- Small and x2- Small to 0.5 each, and the connection weights to the output fuzzy node y-Small to a value of 1. The rest of the connections are set to zero.

Example 2: The exact rule IF x1 is 3.4 and x2 is 6.7 THEN y is 9.5 can be inserted in the same way as in example 1, but here the membership degrees to which the input values x1=3.4 and x2=6.7 belong to the corresponding fuzzy values are calculated and attached to the connection weights instead of values of 1.0. The same procedure is applied for the fuzzy output connection weights.

Changing a MF during operation may be needed for a refined performance after a certain time of the system operation. For example, instead of three MF the system can change to five MF. In traditional fuzzy neural networks this change is not possible, but in EFuNNs it is possible, because an EFuNN stores in its W1 and W2 connections fuzzy exemplars. These exemplars, if necessary, can be defuzzifyied at any time of the operation of the whole system, then used to evolve a new EFuNN structure [12, 13].

4.2.Rule extraction

The following are the steps that comprise an algorithm for rule extraction from a trained EFuNN, where *W1* represents the weight matrix of the connections between the fuzzy inputs layer and the rule layer, and *W2* represents the weight matrix of the connections between the rule layer and the fuzzy output layer [13]:

1. A EFuNN is evolved on incoming data.

2. The values *W1(i,j)* are thresholded:
   if *W1(i,j))>Thr1* then *W1t(i,j)=W1(i,j)*, otherwise: *W1t(i,j)=0*.

3. The values *W2(j,k)* are thresholded in a similar way with the use of a threshold *Thr2*.

4. A rule *Rj* that represents a node *j (j=1,2... Nrn)* is formed as follows:
   Rj: IF x1 is I1 [W1t1 (i1,j)] AND x2 is I2 [W1t2(i2,j)] AND ...
   AND xn is In [W1t n (in,j)]
   THEN y1 is L1 [W2 1 (j, l1)] AND y2 is L2 [W2 2 (j, l2)] AND...
   AND ym is Lm [W2m (j,lm)], where *I1,I2,...*,In are the fuzzy values (labels) of the input variables *x1,x2,...,xn*, correspondingly, with the highest connection weights to the rule node *j* that are above the threshold *Thr1*. *L1, L2,...,Lm* are the fuzzy values of the output variables *y1,y2,...,ym* correspondingly, that are supported by the rule node *j* by connection weights above the threshold *Thr2*. The values *[W1t (i,j)]* are interpreted as fuzzy co-ordinates at the clusters represented in the rules *Rj* and the values *[W2(j,l)]* are interpreted as certainty factors that this cluster belongs to the fuzzy output class.

5. The rules that have the same condition and action parts, but differ in fuzzy co-ordinate value and certainty factors, are aggregated by taking the average values across the fuzzy co-ordinates and the maximum value for the certainty degrees. Taking an average value for the fuzzy co-ordinates is equivalent to finding the geometrical centre at the cluster that aggregates several rule nodes into one. The rule insertion and rule extraction methods are illustrated in the next section on two benchmark case studies. The methods are included in the general architecture of IES presented in section 6.

5.Case studies of adaptive learning, rule insertion and rule extraction

The first case study is a benchmark problem of adaptive time-series prediction (approximation) illustrated on the gas-furnace data. The second case study is of a classification problem illustrated on the benchmark Iris data.

5.1.Gas-Furnace data set

The gas-furnace data has been used by many researchers in the area of neuro-fuzzy engineering [4, 15]. The data set consists of 292 consecutive values of methane at a time moment (t-4), and the carbon dioxide CO_2 produced in a furnace at a time moment (t-1) as input variables, with the produced CO_2 at the moment (t) as an output variable. The following steps were taken: (1) Evolve an EFuNN on half the dataset for one pass and then test on the whole dataset. (2) Retrain the EFuNN on the entire dataset initially for one pass, then repeat another pass of training, and then test on the entire dataset. (3) Extract rules from the EFuNN and then insert into a brand new EFuNN structure and then re-test on the entire dataset.

For the above task an EFuNN was set up with seven membership functions and parameters of sensitivity threshold *Sthr*=0.85; error threshold *Errthr*=0.1; learning rate for both the first and second layer lr=0.5. In part (1) the number of rule nodes generated was 112. The result in Fig.2 shows that the EFuNN has accurately learnt the first half of the dataset and generalised well on the other half of the dataset just after one pass of learning.

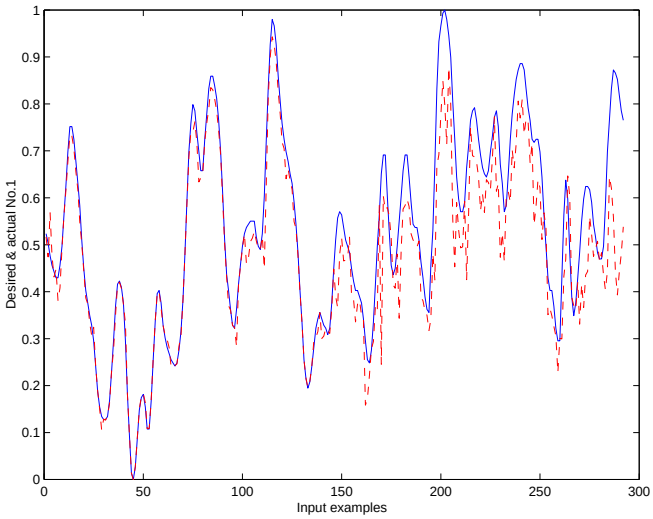

Figure 2: Initial testing of trained EFuNN when first half of the gas-furnace dataset is used.

In part (2) 201 rule nodes were generated. Fig.3 shows the results of testing the entire dataset on the retrained EFuNN after one pass and after the second pass of training. In this case the EFuNN has retained the memory of the first half of the dataset whilst achieving a better function approximation of the entire dataset. The EFuNN improved after the second pass of training.

In the final part of the experiment a set of rules were extracted from the EFuNN from Fig.3. Where there was a case in which rules that had the same condition and action part existed, the average of the condition values and the maximum of the action values were taken. This resulted in 35 rules extracted using the algorithm for *Thr1*=0.1 and *Thr2*=0.8. The new EFuNN was tested on the entire gas-furnace dataset and the results shown in Fig.4. The results show that only 35 clusters (new rule nodes) were created to aggregate the previous 201.

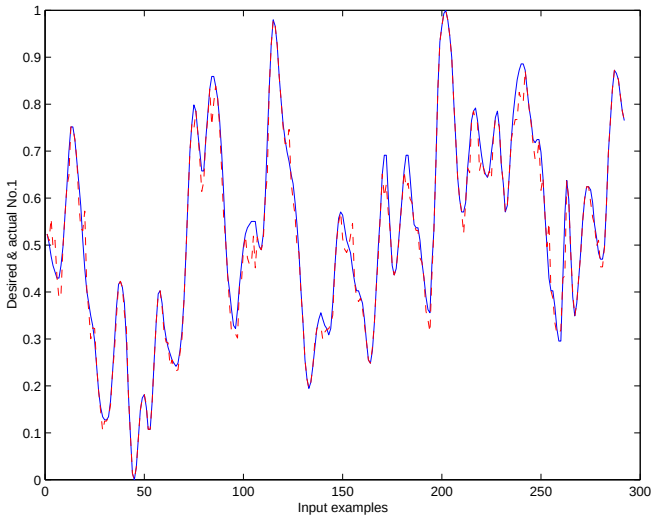

Figure 3: Testing of trained EFuNN when the entire gas-furnace dataset is used after one pass of training and after a second pass of training on the entire dataset.

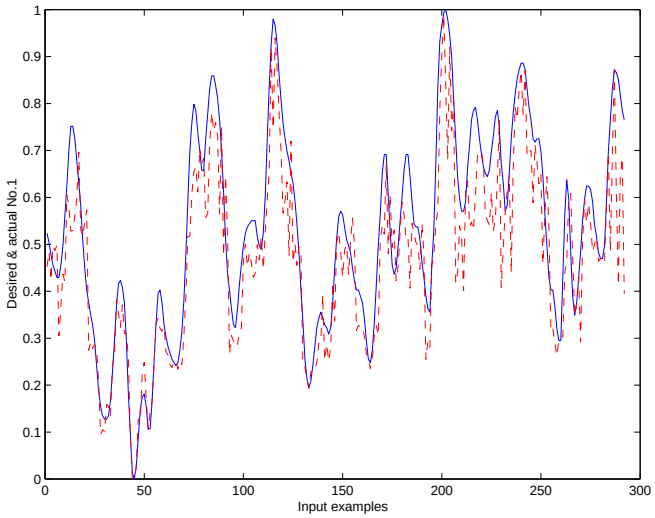

Figure 4: Testing of EFuNN with 35 rules inserted into its structure (no further training).

5.2.Iris Dataset

The realisation of rule extraction and insertion was used on the benchmark Iris dataset. (150 instances; 3 classes -setosa, versicolour and virginica; four attributes - X1-sepal length, X2-sepal width, X3-petal length, X4-petal width). The following steps were taken: (1) Insert three initial rules into an EFuNN structure and test on the entire data set. The rules are: *If X3 is Small and X4 is Small then Setosa. If X3 is Medium and X4 is Medium then Versicolour. If X3 is Large and X4 is Large then Virginica*. (2) Train the EFuNN on the dataset and test it. (3) Extract rules from the EFuNN and then insert into a brand new EFuNN structure and then re-test on the entire dataset.

For the above task a EFuNN was set up with three membership functions and parameters of sensitivity threshold *Sthr*=0.85; error threshold *Errthr*=0.1; learning rate for both the first and second layer lr=0.5. Overall classification rate after part (1) was Setosa - 50/50 (100 %); Versicolour - 50/50 (100%) Virginica 30/50 (60%). Once part (2) had been completed the classification rate was Setosa - 50/50 (100%); Versicolour - 50/50 (100%) Virginica 50/50 (100%). Using the rule extraction algorithm 17 rules were extracted. When part (3) was completed the classification rate was Setosa - 50/50 (100%); Versicolour - 48/50 (96%) Virginica 45/50 (90%). The results were based on 17 rules.

6.An Architecture of Intelligent Expert System that Utilizes Rule Insertion and Rule Extraction from EFuNNs

Traditionally the architecture of conventional expert systems had a fixed structure of modules and a fixed rule base. Although they were successful in very specific areas, this particular architecture resulted in little or no flexibility for the expert system to adapt to the changes required by the user or the environment in which the expert system operated.

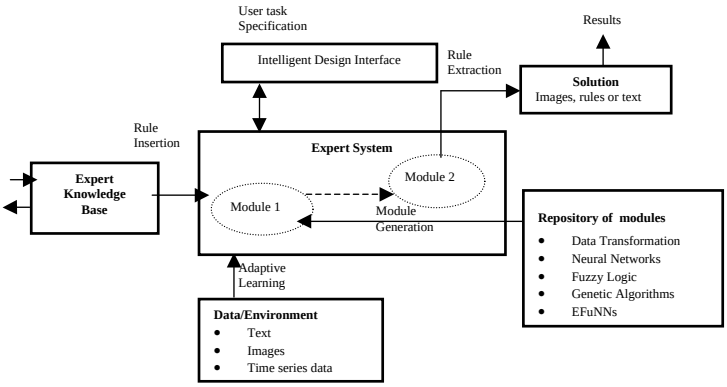

Figure 5: A block diagram of an on-line adaptive expert system.

To address these deficiencies, we propose an Intelligent Expert System (IES) that consists of a series of modules which are agent based and generated “on the fly” as they are needed. Fig.5 shows a general architecture of an IES that is under development. The *User* specifies the initial problem parameters and task to be solved. Intelligent agents then create *Modules* that may initially have no rules in them or may be set up with rules from the Expert Knowledge Base. The Modules then combine the rules with the Data from the Environment to form the Expert System. The *Modules* are then trained with the *Data* from the *Environment*. The rules may be extracted for later analysis, or aggregated and re-inserted into new *Modules* for re-training or testing. Once the *Modules* have been trained, they are tested with new *Data* from the *Environment* and the results extracted. These then form the solution to the problem and may be further interpreted by another set of *Modules*. The *Modules* will dynamically adapt their ruleset as the environment changes since the number of rules is dependent on the data that is being presented to the

Module. *Modules*(agents) are dynamically created, updated and connected. They will be destroyed if necessary at a later stage of the operation of the IES.

7.Conclusions

The paper suggests a methodology for building intelligent expert systems (IES), expert systems that can change their knowledge base dynamically as the system works. They can adapt in an on-line mode, possibly in real time, to a changing environment. The methodology uses rule insertion, rule extraction and fast adaptation methods for evolving fuzzy neural networks. The methodology is applied to practical problems in prediction and classification.

Acknowledgements

This research is part of a research programme funded by the New Zealand Foundation for Research Science and Technology, UOO808.

References

[1] S. Amari and N. Kasabov. *Brain-like computing and intelligent information systems*. Springer Verlag, first edition, 1997.

[2] R. Andrews, J. Diederich, and A. B. Tickle. A Surbey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. *Knowledge-Based Systems*, 8:373–389, 1995.

[3] T. Hashiyama, T. Furuhashi, and Y. Uchikawa. A Decision making Model Using a Fuzzy Neural Network. In *Proceedings of the 2nd International Conference on Fuzzy Logic & Neural Networks, Iizuka, Japan*, pages 1057–1060, 1992.

[4] R. Jang. ANFIS: Adaptive Network-Based Fuzzy Inference System. *IEEE Trans. on Syst., Man, Cybernetics*, 23(3):665–685, 1993.

[5] N. Kasabov. Adaptable connectionist production systems. *Neurocomputing*, 13(2-4):5–117, 1996.

[6] N. Kasabov. *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*. MIT Press, Cambridge: MA, first edition, 1996.

[7] N. Kasabov. Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. *Fuzzy Sets and Systems*, 82(2):2–20, 1996.

[8] N. Kasabov. ECOS: A Framework For Evolving Connectionist Systems And The eco Learning Paradigm. In *Proc. of ICONIP'98.,Kitakyushu, Oct 1998*, pages 1232–1236, 1998.

[9] N. Kasabov. Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation. In *Proc. of Iizuka'98, Iizuka, Japan*, pages 271–274, 1998.

[10] N. Kasabov. Fuzzy Neural Networks, Rule Extraction and Fuzzy Synergistic Reasoning Systems. *Research and Information Systems*, 8:45–59, 1998.

[11] N. Kasabov. *Evolving Connectionist And Fuzzy Connectionist System For On-Line Decision Making And Control*, volume Soft Computing in Engineering Design and Manufacturing. Springer-Verlag, 1999.

[12] N. Kasabov. *Evolving Connectionist and Fuzzy-Connectionist Systems: Theory and Applications for Adaptive, On-line Intelligent Systems*, volume Neuro-fuzzy Tools and Techniques for Intelligent Systems, N. Kasabov and R. Kozma (eds). Springer-Verlag, first edition, 1999.

[13] N. Kasabov. Learning, reasoning, and rule extraction in Evolving Fuzzy Neural Networks. *Submitted to Neurocomputing*, 1999.

[14] N. Kasabov, J. S. Kim, M. Watts, and A. Gray. FuNN/2- A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition. *Information Sciences - Applications*, 101(3-4):155–175, 1996.

[15] T. Kohonen. The Self-Organizing Map. *Proceedings of the IEEE*, 78(9):1464–1497, 1990.

[16] C. T. Lin and C. S. Lee. *Neuro Fuzzy Systems*. Prentice Hall, 1996.

[17] G. Towel, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge based neural networks. In M. Kaufmann, editor, *Proc. of the 8th national Conf. on Artificial Intelligence AAAI'90*, pages 861–866, 1990.

[18] T. Yamakawa, E. Uchino, and T. Miki. A new Effective Algorithm for Neo Fuzzy Neuron Model. In M. Kaufmann, editor, *Proc. of the Fifth IFSA World Congress*, pages 1017–1020, 1993.

[19] L. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.