FuNN - A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition

N. K. Kasabov, J. S. Kim, A. R. Gray, M. J. Watts

Department of Information Science

University of Otago, P.O.Box 56, Dunedin, New Zealand

Phone: +64 3 479 8319, Fax: +64 3 479 8311, email: nkasabov@otago.ac.nz

Abstract

Fuzzy neural networks have several features that make them well suited to a wide range of knowledge engineering applications. These strengths include fast and accurate learning, good generalisation capabilities, excellent explanation facilities in the form of semantically meaningful fuzzy rules, and the ability to accommodate both data and existing expert knowledge about the problem under consideration. This paper investigates adaptive learning, rule extraction and insertion, and neural/fuzzy reasoning for a particular model of a fuzzy neural network called FuNN. As well as providing for representing a fuzzy system with an adaptable neural architecture, FuNN also incorporates genetic algorithms as one of its adaptation strategies.

1. Introduction

Fuzzy neural networks (FNN) have been proposed as a knowledge engineering technique and used for various application domains by several authors including Yamakawa and Uchino [1], Uchikawa and Furuhashi [2], and others [3, 4, 5, 6, 9]. The FNN proposed by these authors have been successfully used for learning and tuning fuzzy rules as well as solving classification,

prediction and control problems. Some recent publications suggest methods for training FNNs in order to adjust to new or dynamically changing data and situations [4, 5].

This paper introduces a new architecture for FNN, called FuNN, which stands for Fuzzy Neural Network, and investigates some learning and adaptation strategies associated with it. In addition, the use of rule insertion and rule extraction algorithms are investigated. One of the unique aspects of the FuNN architecture is that it combines neural network, fuzzy logic, and genetic algorithms into a single system. This multi-paradigm approach has produced good results for a wide range of problems.

2. The Architecture of FuNN

The FuNN model is designed to be used in a distributed, and eventually agent-based, environment. The architecture facilitates learning from data and approximate reasoning, as well as fuzzy rule extraction and insertion. It allows for the combination of both data and rules into one system, thus producing the synergistic benefits associated with the two sources. In addition, it allows for several methods of adaptation (adaptive learning in a dynamically changing environment). Considerable experimentation can be carried out using this system to evaluate different adaptation strategies.

FuNN uses a multi-layer perceptron (MLP) network and a modified backpropagation training algorithm. The general FuNN architecture consists of 5 layers of neurons with partial feedforward connections as shown in figure 1. It is an adaptable FNN where the membership functions of the fuzzy predicates, as well as the fuzzy rules inserted before training or adaptation, may adapt and change according to new data. Below a brief description of the components of the FuNN architecture and the philosophy behind this architecture are given.

2.1 Input Layers

The *input layer* of neurons represents the input variables as crisp values. These values are fed to the *condition element layer* which performs fuzzification. This is implemented using three-point triangular membership functions with centers represented as the weights into this condition element layer. The triangles are completed with the minimum and maximum points attached to adjacent centers, or shouldered in the case of the first and last membership functions.

The triangular membership functions are allowed to be non-symmetrical and any input value will belong to a maximum of two membership functions with degrees differing from zero (it will always involve two unless the input value falls exactly on a membership function center in which case the single membership will be activated, but this equality is unlikely given floating point variables). These membership degrees for any given input will always sum up to one, ensuring that some rules will be given the opportunity to fire for all points in the input space.

This center-based membership approach taken by FuNN avoids the problems of uncovered regions in the input space that can exist with more flexible membership representation strategies. These do not always limit centers and widths in such a way as to ensure complete coverage. While algorithms could be formulated and used in such cases to force the memberships to cover the input space, the simple center-based approach taken by FuNN seems both more efficient and more natural, with fewer arbitrary restrictions. It should be noted that there are no “bias” connections necessary for this representation in FuNN.

The weights from the input to condition element layers of neurons can take values in [0,1] only since the data are assumed to be normalised to this range. This normalisation is normally carried out as part of the FuNN pre-processing operations, and can be performed and reverse transparently in applications.

Initially the membership functions are spaced equally over the weight space, although if any expert knowledge is available this can be used for initialisation. In order to maintain the semantic meaningfulness of the memberships contained in this layer of connections some restrictions are placed on adaptation. When adaptation is taking place the centers are limited remain within equally sized partitions of the weight space. The avoids problems with violating the semantic ordering of membership function labels. Therefore, under the FuNN architecture labels can be attached to weights when the network is constructed and these will remain valid for the lifetime of the network. For example, a membership function weight representing low will always have a center less than medium, which will always be less than high.

The condition element layer of neurons is potentially expandable during the adaptation phase with more nodes representing more membership functions for the input variables. Simple activation functions are used in the condition element nodes to perform fuzzification (see appendix A for the full algorithm for the FuNN).

An important aspect of this layer is that different inputs can have differing numbers of membership functions. The same principle applies to the output membership functions. This allows for very different types of inputs to be used together. As a simple example, temperature may be divided into seven different membership functions representing the range from cold to hot, while holiday (which is a binary variable to indicate whether it is a public holiday or not) can be represented using just two for yes and no.

2.2 Rule Layer

In the *rule layer* each node represents a single fuzzy rule. The layer is also expandable in that nodes can be added to represent more rules as the network adapts. The activation function is

the sigmoidal logistic function with a variable gain coefficient g (a default value of 1 is used giving the standard sigmoidal activation function). For the gain coefficient large values will make it close to the hard limited thresholding function. A value of 2.19722 would ensure that a rule node would provide activation values from 0.1 to 0.9 when the net input values are between -1 and +1. These values may be desirable as part of the architecture's "fuzziness."

The semantic meaning of the activation of a node is that it represents the degree to which input data matches the antecedent component of the associated fuzzy rule. However the synergistic nature of rules in a fuzzy-neural architecture must be remembered when interpreting such rules. The connection weights from the condition element layer (also called the membership functions layer) to the rule layer represent semantically the degrees of importance of the corresponding condition elements for the activation of this node.

The values of the connection weights to and from the rule layer can be limited during training to be within a certain interval, say [-1,1], thus introducing non-linearity into the synaptic weights. This option mimics a biologically plausible phenomenon [10] but should be implemented in accordance with an appropriate gain factor for the activation function. For example, if the interval is [-1,1] a suitable value for the gain factor may be 2.19722 as described above. The weight limitation would ensure that inputs into the rules remain within [-1,1] (since the input membership functions are all between 0 and 1) and the gain factor would only allow the rules to output values in [0.1,0.9]. This further enhances the meaningfulness of the rules and weight saturation will not occur. As an example of the problems of rule interpretation with unrestricted weights, it is difficult to interpret a rule that has input weights that are very high values, say 16, without some form of normalisation. With this weight limiting option, the necessity for such normalisation is removed.

2.3 Output Layers

In the *action element layer* a node represents a fuzzy label from the fuzzy quantisation space of an output variable, for example small, medium, or large for the output variable "required change in the velocity." The activation of the node represents the degree to which this membership function is supported by the current data used for recall the FuNN. So this is the level to which the membership function for this fuzzy linguistic label is "cut" according to the current facts. The connections from the rule layer to the action element layer represent conceptually the confidence factors of the corresponding rules when inferring fuzzy output values. They are subject to constraints that require them to remain in specified intervals as for the previous layer with the same advantages of semantic interpretability. The activation function for the nodes of this layer is the sigmoidal logistic function with the same (variable) gain factor as in the previous layer. Again, this gain factor should be adjusted appropriately given the size of the weight boundary.

The *output layer* performs a modified center of gravity defuzzification. Singletons, representing centers of triangular membership functions, as it was the case of the input variables, are attached to the connections from the action to the output layer. Linear activation functions are used here.

As an example, small, medium and large can be represented as connection weights having values of 0, 0.5 and 1.0 respectively from the output range of [0,1] if normalised outputs are considered. Adapting the output membership functions would mean moving the centers, but the requirement that the membership degrees to which a particular output value belongs to the various fuzzy labels must always sum up to one, is always satisfied. For each centre, there is a constraining band (partition) where this value can move to. This is used in the same way as the input membership function centers' restrictions are. More than one output variable can be used

in a FuNN structure and the different output variables can have different numbers of membership functions.

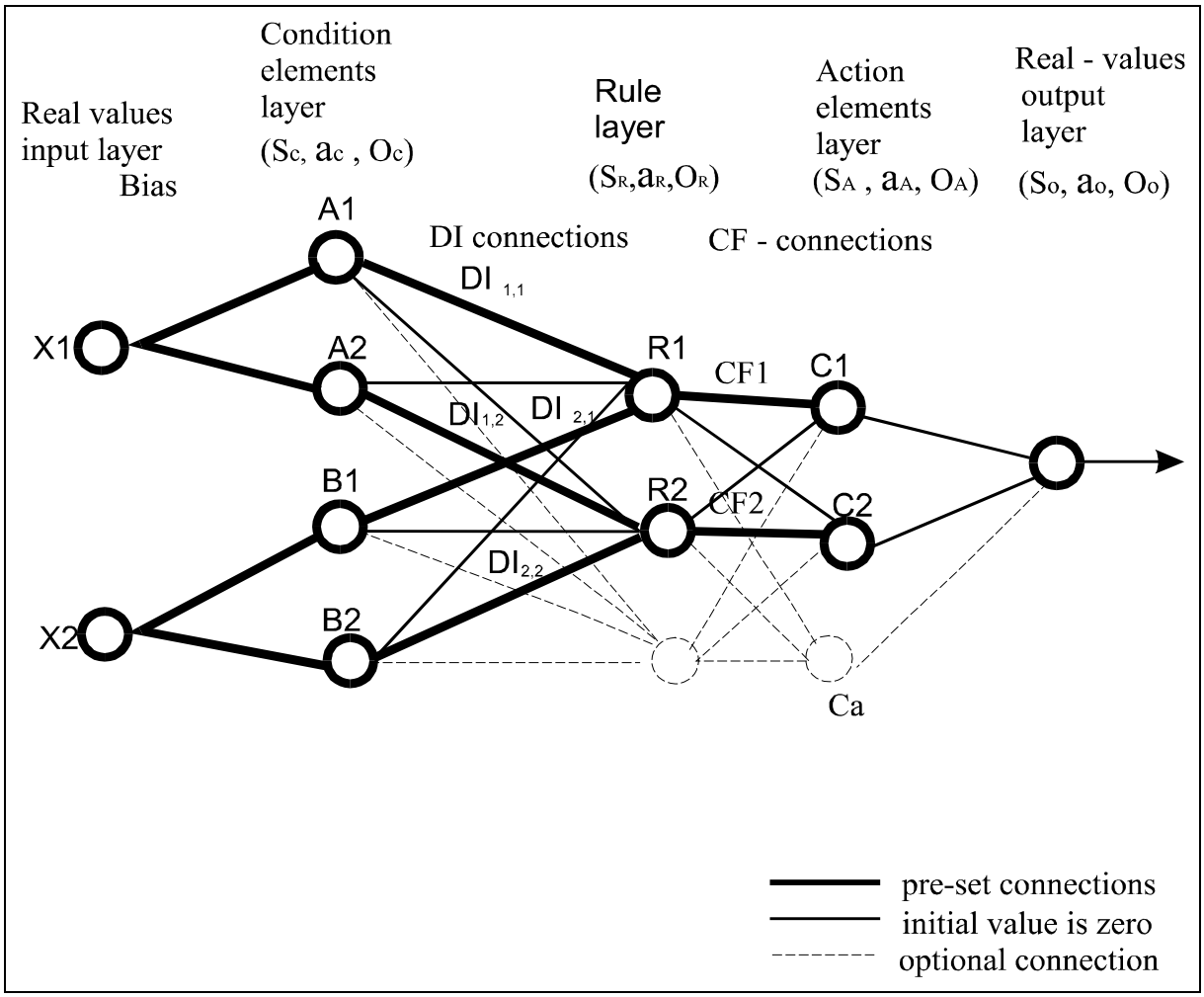

Figure 1 A FuNN structure for two initial fuzzy rules: R_1: IF x_1 is A_1 ($DI_{1,1}$) and x_2 is B_1 ($DI_{2,1}$) THEN y is C_1 (CF_1); R_2: IF x_1 is A_2 ($DI_{1,2}$) and x2 is B_2 ($DI_{2,2}$) THEN y is C_2 (CF_2), where DIs are degrees of importance attached to the condition elements and CFs are confidence factors attached to the consequent parts of the rules (from [4])

One of the advantages of the FuNN architecture is that it manages to provide a fuzzy logic system without having to unnecessarily extend the traditional MLP. Since standard transfer functions, linear and sigmoidal, are used along with a slightly modified back-propagation algorithm, the main departure being partitions, much of the large body of theory regarding such

networks is still applicable. For those results not immediately applicable for FuNN the modifications are made much simpler given FuNN's natural structure and algorithm.

3. Adaptation of FuNN

There are three versions of weight updating in the FuNN according to the mode of training and adaptation. These are not mutually exclusive versions but are all provided within the same environment and the versions can be switched between as needed. These methods of adaptation are:

(a) a partially adaptive version where the membership functions (MF) of the input and the output variables do not change during training and a modified backpropagation algorithm is used for the purpose of rule adaptation. This adaptation mode can be suitable for systems where the membership functions to be used are known in advance or where the implementation is constrained by the problem in some way.

(b) a fully adaptive version with an extended backpropagation algorithm, as explained in section 3 and appendix A. This version allows changes to be made to both rules and membership functions, subject to constraints necessary for retaining semantic meaningfulness.

(c) a partially adaptive version with the use of a genetic algorithm for adapting the membership functions. This mode does not alter the rules. The partition limitations are only partially imposed on the version, with the [0,1] limit used, but not the intermediate limits. The algorithm used is described in section 5.

These modes are not alternatives as such to be chosen between, but can be used together in whatever combination is most appropriate for the given problem at a certain time. It may be

useful to use several different modes in an iterative manner, with each version of the adaptation algorithm best suited to some part of the adaptation task.

4. Extended Backpropagation Algorithm for Adaptive Training of FuNN

Here the fuzzification layer and the defuzzification layer change their input connections based on simple and intuitive formulas. These changes reflect the concepts represented by the layers and must satisfy the restrictions (partitioning) imposed on the membership functions (the movements of their centres cannot take them out of the membership's partition).

The same principles apply for the two layers, but different formulas are used to calculate the change of the weights (see appendix A). Figure 2 shows the initial membership functions of a variable x (either input or output) and the membership functions after adaptation. The amount of change has been exaggerated in order to demonstrate the concept involved. In the normal course of training changes to membership functions are limited to small, gradual movements, with the majority of weight changes occurring with the weights into and out of the rules.

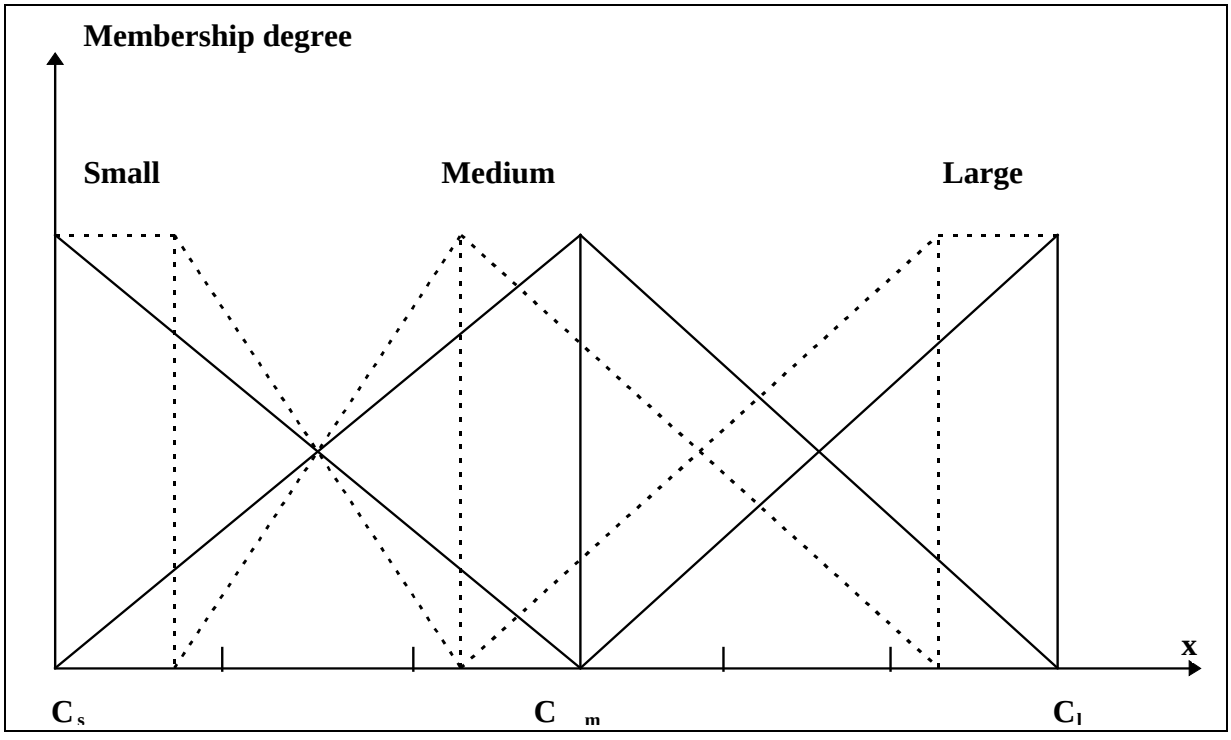

Figure 2. Initial membership functions (solid lines) of a variable x (either input, or output) represented in a FuNN and the membership functions after adaptation (dotted lines). The boundaries, to which each centre can move but not cross, are also indicated.

5. Using Genetic Algorithm for Adaptation of MF in FuNN

Genetic algorithms (GA's) are highly effective search algorithms that are based on the principles of natural genetics and natural selection [7]. By iteratively building on the success of previous attempts at a problem solution, GA's are able to quickly investigate large search spaces to find approximate solutions to difficult combinatorial problems.

Given that the optimisation of fuzzy membership functions may involve many changes to many different functions, and that a change to one function may effect others, the large possible solution space for this problem is a natural candidate for a GA based approach. This has already been investigated in [8], and has been shown to be more effective than manual alteration. The application of these principles to a FuNN can be seen as a logical extension of this previous work.

The work carried out in [8] focused on the use of small changes to the width and centre positions of the membership functions of the system. These delta values may be easily encoded within a genetic algorithm chromosome structure. They also provide a convenient means of restricting the amount of movement allowed by a single application of the GA system. By measuring the performance of the system resulting from these changes, the fitness of the chromosome may be calculated. A similar approach has been taken in the GA module of FuNN. In this case only centers need to be represented by the chromosome strings speeding up the adaptation process and possibly reducing spurious local minima over the center and width approach used by [8].

The GA module for adapting FuNN is designed as a stand alone system for optimising fuzzy neural networks that have rules inserted or have rules that have been adapted. The system will optimise the membership functions of both the condition and action layers. Although some parameters such as the length of the chromosomes are determined automatically, the rest of the parameters, such as the size of the population, the mutation rate and the use of fitness normalisation are user configurable. The GA used in the system is, in essence, the same as Goldberg's Simple Genetic Algorithm [7], with the important exception that the chromosomes are represented as strings of floating point numbers, rather than strings of bits. Also, mutation of a gene is implemented as a reinitialisation of the gene, rather than an alteration of the existing allele.

The operating process for the GA system will be briefly described here. After the user has specified the FuNN to be optimised, the system calculates the length of the chromosome necessary to represent the changes to be applied to the input and output membership functions. A copy is made of the original FuNN (with all zero deltas) to ensure, along with elitism, that the error function is monotonically decreasing for GA training.

After the random initialisation of the population, in which the current FuNN is present, each individual is evaluated. To evaluate an individual, the delta values encoded within the individual's chromosome are applied to the copy of the original FuNN. A user defined test data file is then submitted to the modified FuNN, and the Root Mean Square (RMS) error calculated. The fitness of the modified FuNN is then calculated as the inverse of the RMS error, since those individuals with the lower RMS errors are more fit than those with higher errors.

After optionally normalising the fitness values and with optional elitism (which ensures that the most fit network is retained without modification in the nest population), a breeding

population is selected using either tournament or roulette wheel selection. Finally, after an optional mutation phase, a new population is created. One point crossover is used for the reproduction of chromosomes.

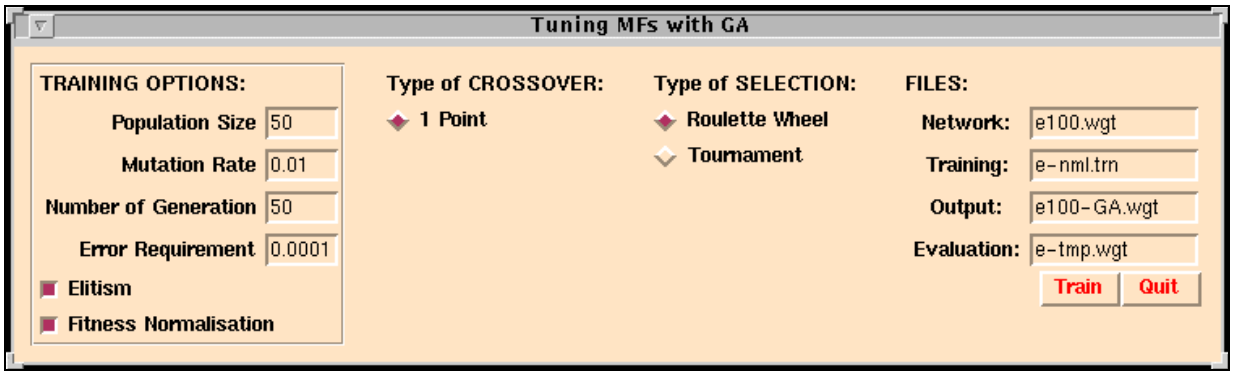

Figure 3 The interface of the GA module

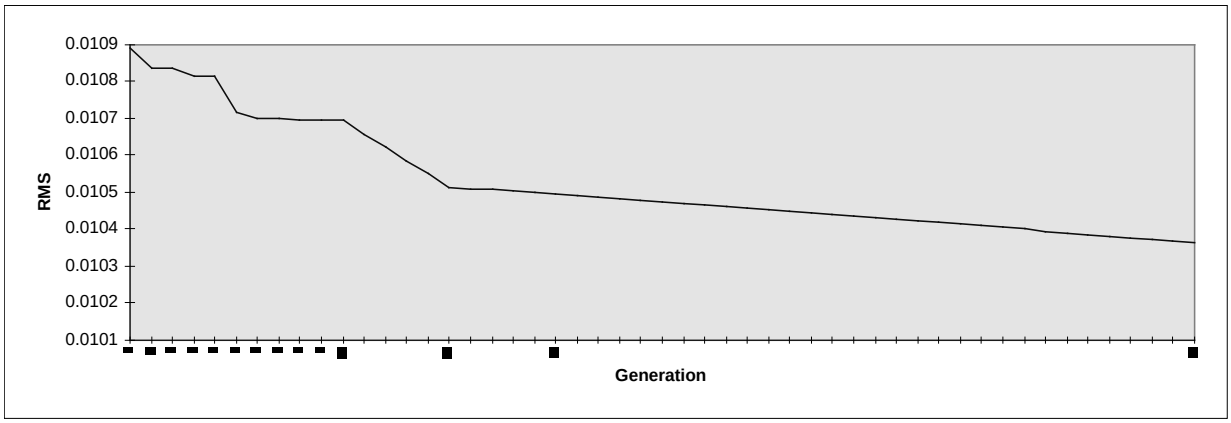

Figure 4 The decrease of RMS with the increase of generations for phoneme /e/ after 100 epochs of training with FuNN in full-adaptation mode

6. Experimenting with Different Training and Adaptation Strategies for FuNN

FuNN (figure1) has a flexible architecture which allows for different training and adaptation strategies to be tested before the most suitable is selected for a certain application. Some of the issues involved in this adaptation are discussed below.

- Initialisation: uniformly distributed triangular MF can be used as initial values for the input variables, and uniformly distributed singletons can be used as initial values for the output variables. These are the defaults that are used in the absence of other information.

- Membership function insertion. If some expert knowledge is available then this can be used to initialise the memberships, or at least initialise those for which knowledge exists with the remained being initialised using the default method.

- Rule insertion. If initial set of rules is available, it is used for initialisation of the FuNN structure through rules insertion mode. The rules are represented as weights so as well as inserting the existence of a rule, the relative importance of that rule and its sensitivity to input variables can be provided.

- Training the FuNN can be accomplished either for the inner two rule weight layers, in which case the system adapts its fuzzy rules but does not adapt the membership functions, or for the four weight layers, in which case the system adapts both the rules and the membership functions. The only difference between these two options is that the connections in the fuzzification and defuzzification layers are "frozen" in the former case and they are subject to change in the latter case.

- Adaptation through a GA algorithm. With this option only the membership functions change. Since the rules will need some small adjustments for the new membership functions the best network can then be subject to a short period of training by the restricted membership version of the FuNN backpropagation algorithm.

When training the FuNN the opportunity arises for selecting several training and adaptation parameters. For the back-propagation training option these include gain factors for each layer,

learning rates and momentum rates for each layer, and boundaries for the two sets of rule connection weights as explained in the previous sections. For the genetic algorithm training the parameters are population size, number of generations, mutation rate, optional fitness normalisation, optional elitism, and the type of selection mechanism used.

Two main areas of applications arise for using a FuNN system. These are either for data approximation (as it is the case with the standard, feed-forward neural networks [4]), or for fuzzy rule interpretation and adaptation.

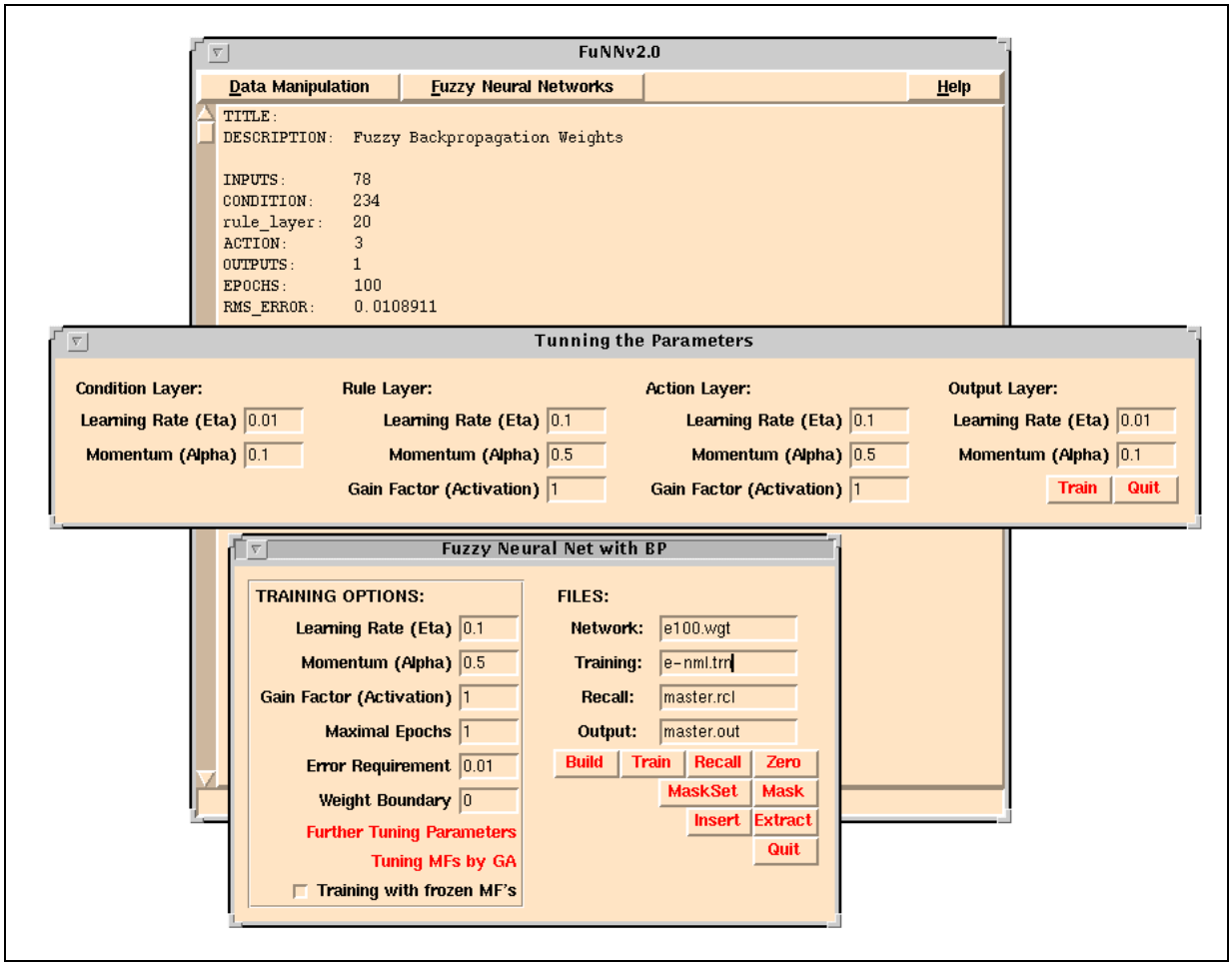

Figure 5 The FuNN interface

Since FuNN implements a fuzzy system, the initial insertion of rules and membership functions should allow the network to learn both faster and more accurately. The first point is not

regarded as too important for reasons for be discussed below, but the second is an important aspect of the FuNN system. This greater accuracy of learning, through avoiding some local minima by placing the initial weights closer to the global minima, can also be an important feature when dealing with small, not necessarily fully-representative, data sets.

The greater number of weights used in a FuNN compared to a corresponding MLP will slow training to some degree. This is not regarded as problematic since the learning rates and momentum terms are usually set to lower values for FuNN than for traditional MLPs. Still, some investigations into optimal training periods and speeds to maximise the network's generalisation capabilities are necessary.

When new data becomes available decisions must be made regarding how to adapt to, potentially, new circumstances and relationships expressed in this data. Depending on the size of this new data segment two general learning strategies can be distinguished and used [3]:

- *aggressive* - a section of new data is used for further training and adaptation without using any of the old, previously used, data.

- *conservative* - new data is added to all of the old data and training is performed on the entire set.

Obviously, these concepts of aggressiveness and conservatism in training are fuzzy since some proportion of old data can be combined with new data to produce a retraining data set. In fact, some compromise of using the new data with a percentage of old data tends to be most efficient. The amount of old data retained depends on operating requirements (since for on-line adaptation using a large data set may not be feasible), the stationarity of the relationships, and the length of time that changes tend to persist for until returning, if at all, to the original relationships. An example here is that of the stock market. This tends to exhibit long term

trends with occasional departures from that trend. However except in unusual circumstances, such as a large financial market crash, the long-term pattern will eventually be restored..

7. Rule Insertion to Initialise FuNN and Rule Extraction from Trained FuNN

Several different methods for rules extraction are applicable on FuNN. Two of them have been implemented and investigated on test cases. The first one, called REFuNN [3, 4], is based on the simple idea of thresholding connection weights according to a pre-set limit. The weights, which are above the threshold are represented as condition, or action elements in the extracted fuzzy rules with their associated weights representing degrees of importance and confidence factors. One rule node in the FuNN architecture is represented by several fuzzy rules each of them representing a combination of the input condition elements which would activate that node. A simplified version of the rules without degrees of importance can also be extracted and interpreted later in a classical fuzzy inference engine [4].

A second algorithm for interpreting a FuNN structure in terms of aggregated fuzzy rules is also implemented. Each rule node is represented as one fuzzy rule. The strongest connection from a condition element node for an input variable to the rule node, along with the neighbouring condition element nodes, are represented in the corresponding rule. The connection weights of these connections are interpreted as degrees of importance attached to the corresponding condition elements. In order to keep in a FuNN structure these connection weights only a *masking* procedure has been implemented. One rule has in general as many consequent elements as the number of the nodes in the action element layer, each action (class) inferred with a certainty degree (confidence factor) defined by the connection weights from this rule node to that class node. An example of a rule set extracted from a FuNN is given in figure 6 below.

RULES

if <Input1 is A 2.88> and <Input1 is not B 5.69> and <Input2 is not A 1.57> and <Input2 is B 1.66> <Input2 is not C 1.06> **then** <Output1 is not A 2.21> and <is not B 3.98> and <is C 2.08>

else

if <Input1 is not A 6.27909> and <Input1 is B 2.75163> and <Input1 is not C 01.23> and <Input2 is not B 2.85492> and <Input2 is C 4.99544> and <Input2 is not D 7.41543> **then** <Output1 is A 2.59163> and <is B 1.22> and <is not C 4.84>and <is not D 4.04>

Figure 6 Two exemplar fuzzy rules extracted from a *masked* FuNN

The extracted rules from a FuNN can be *inserted* in other FuNN modules through the rules insertion mode. This enables the use of the round-trip engineering methodology as described in [9].

8. Conclusions

Here a fuzzy neural network architecture is introduced. The adaptive learning algorithms used along with the rule extraction and rule insertion techniques show that this is a promising approach to building adaptive intelligent information processing systems which suits many applications. These application areas include signal processing (particularly speech recognition), time-series modelling and prediction, adaptive control, data mining and knowledge acquisition, and image processing.

Acknowledgements

This research is partially supported by a research grant UOO 606 funded by the Public Good Science Fund of the Foundation of Research Science and Technology (FRST) in New Zealand.

The following colleagues take part in the many discussions before the release of the current version of FuNN: Dr Martin Purvis, Dr Feng Zhang, Brendan Hallett, Richard Kilgour, and Steve Israel.

References

[1] Yamakawa, T., Kusanagi, H., Uchino, E. and Miki, T., "A new Effective Algorithm for Neo Fuzzy Neuron Model", in:*Proceedings of Fifth IFSA World Congress*, (1993) 1017-1020.

[2] Hashiyama, T., Furuhashi, T., Uchikawa, Y., "A Decision Making Model Using a Fuzzy Neural Network", in: *Proceedings of the 2nd International Conference on Fuzzy Logic & Neural Networks, Iizuka, Japan*,(1992) 1057-1060.

[3] Kasabov, N., "Learning fuzzy rules and approximate reasoning in Neuro-Fuzzy hybrid systems", *Fuzzy Sets and Systems*, 82 (2), 1996, pp.1-19

[4] Kasabov, N., *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*, The MIT Press, CA, MA, 1996

[5] Kasabov, N., "Investigating the adaptation and forgetting in fuzzy neural networks by using the method of training and zeroing", in: *Proceedings of the International Conference on Neural Networks ICNN'96*, IEEE Press, Washington DC, June 3-6, 1996

[6] Hauptmann, W., Heesche, K., A Neural Net Topology for Bidirectional Fuzzy-Neuro Transformation, in: *Proceedings of the FUZZ-IEEE/IFES* , Yokohama, Japan, (IEEE, 1995), 1511-1518.

[7] Goldberg, D., *Genetic Algorithms is Search, Optimization and Machine Learning* , Addison Wesley, 1989

[8] Mang, G. Lan, H., Zhang, L., "A Genetic-based method of Generating Fuzzy Rules and Membership Functions by Learning from Examples", in: *Proceedings of International Conference on Neural Information Processing (ICONIP '95)* Volume One, pp 335 - 338. 1995.

[9] Gray, A. R., Kasabov, N., "Round-Trip System Engineering in Neuro-Fuzzy Hybrid Systems", *Journal of Intelligent and Fuzzy Systems*, 1996, to appear.

[10] Carpenter, G., "Distributed Learning, Recognition, and Prediction by ART and ARTMAP Neural Networks", in: *Proceedings of ICNN'96* , IEEE Press, Volume "Plenary Panel and Special Sessions", pp 244-249.

Appendix A. Adaptive learning algorithm for adjusting membership functions in a FuNN architecture

This section explains the algorithm used for the FuNN architecture, both the feed-forward phase and the backpropagation of errors. The algorithm is discussed in terms of layers of neurons since they are the focus of a fuzzy architecture, rather than the weights as such. Where layers of weights are intended this should be evident from the context.

Forward pass

This phase computes the activation values of all the nodes in the network from the first to fifth layers. In this section a superscript indicates the layer and a subscript describes connection weights between layers.

◊ **Layer 1 (Input Layer)**. The nodes in this layer only transmit input values (crisp values) to the next layer directly without modification.

◊ **Layer 2 (Condition Layer)** . The output function of this node is the degree that the input belongs to the given membership function. The input weight represents the center for that particular membership function, with the minimum and maximum determined using the adjacent membership's centers. In the case of the first and last membership function for a particular variable a shoulder is used instead. Hence, this layer acts as the fuzzifier. Each membership function is triangular and an input signal activates only two neighbouring membership functions simultaneously, the sum of the grades of these two neighbouring membership functions for any given input is always equal to 1. For a triangle-shaped membership function as in FuNN, the activation functions for a node i are:

if $a_i < x < a_{i+1}$, then $Act_i^c = 1 - \frac{x - a_i}{a_{i+1} - a_i}$

if $a_{i-1} < x < a_i$, then $Act_i^c = 1 - \frac{a_i - x}{a_i - a_{i-1}}$

if $x = a_i$, then $Act_i^c = 1$

where a is the centre of the triangular membership function

◊ **Layer 3 (Rule Layer)** : The connections from the condition layer to this rule layer are used to perform pre-condition matching of fuzzy rules. The connection weights in this layer may be set either randomly and then trained or according to a set of rules (rules insertion). The net inputs and activations are respectively:

$$Net^r = \sum w_{rc} \times Act^c$$

$$Act^r = \frac{1}{1 + e^{(-g \times Net^r)}}$$

where g is a gain factor.

◊ **Layer 4 (Action Layer)** : The nodes in this layer and the connection weights function as those in layer 3 for net input and activation:

$$Net^a = \sum w_{ar} \times Act^r$$

$$Act^a = \frac{1}{1 + e^{(-g \times Net^a)}}$$

◊ **Layer 5 (Output Layer)** : This layer performs defuzzification to produce a crisp output value. Among the commonly used defuzzification strategies, the *Center of Gravity* (COG) method yielded the best result. In this layer, linear output activation function is used:

$$Net^o = \sum w_{oa} \times Act^a$$

$$Act^o = \frac{Net^o}{\sum Act^a}$$

Backward pass

The goal for this phase is to minimise the error function:

$$Error = \frac{1}{2}\sum (y^d - y^a)^2$$

where y^d is the desired output, and y^a is the current output.

Hence the general learning rule (gradient descent) used is

$$\Delta w \approx -\frac{\partial E}{\partial w}$$

$$w_{t+1} = w_t + \eta(-\frac{\partial E}{\partial w}) + \alpha(\Delta w_t)$$

where η is the learning rate and α is the momentum coefficient, and

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Net} \times \frac{\partial Net}{\partial w} = -\delta \times Act$$

Hence the weight update rule is:

$$\Delta w_{t+1} = \eta\delta \times Act + \alpha\Delta w_t$$

◊ **Layer 5 (Output Layer):**

$$\delta^o = -\frac{\partial E}{\partial Net^o} = -\frac{\partial E}{\partial Act^o} \times \frac{\partial Act^o}{\partial Net^o} = y^d - y^a$$

Then the weights are adapted a partitioning is taken into account which imposes restrictions to the change of the centres of the membership functions. Any weight that moves outside of its partition is then moved back to the appropriate limit.

If $w_t + w_{t+1} < \text{Partition}_k$, then $w_{t+1} = \text{Partition}_k$, else

If $w_t + w_{t+1} > \text{Partition}_{k+1}$, then $w_{t+1} = \text{Partition}_{k+1}$, else

$w_{t+1} = w_t + w_{t+1}$

◊ **Layer 4 (Action Layer):** The error for each node in the action layer is calculated individually based on the output error and on the activation of this node having in mind the type of the membership functions triangular) used in(the defuzzification layer as well as the type of defuzzification:

If $a_i < y < a_{i+1}$, then $d^a = 1 - \frac{y - a_i}{a_{i+1} - a_i}$

If $a_{i-1} < y < a_i$, then $d^a = 1 - \frac{a_i - y}{a_i - a_{i-1}}$

If $y = a_i$, then $d^a = 1$

Hence,

$$\delta^a = f(Net^a) \times E^a = Act^a(1 - Act^a) \times (d^a - Act^a)$$

◊ **Layer 3 (Rule Layer):**

$$\delta^r = Act^r(1 - Act^r) \times \sum (w_{ar} \times \delta^a)$$

◊ **Layer 2 (Condition Layer):** The weight w_{ic} is assigned as follows. If x^i lies in the fuzzy segment, then the corresponding weight should be increased directly proportional to the propagated error from the previous layer, because the error is caused by the weight. This proposition can be represented by the following equation:

$$\delta^c = Act^c \times \sum (w_{rc} \times \delta^r)$$

Thus the weight-updating rule of this layer is:

$$w_{ic_{t+1}} = \eta \delta^c + \alpha \Delta w_{ic_t}$$

The new centres of the input triangular membership functions are also adjusted according to a partition range as for the output layer.