Phoneme-based speech recognition via fuzzy neural networks modeling and learning

N.K. Kasabov *, R. Kozma, M.J. Watts

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

Received 28 May 1996; accepted 10 October 1997

Abstract

Fuzzy neural networks (FNN) 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. The paper presents one particular architecture called FuNN and discusses two alternative ways to optimise its structure, namely a genetic algorithm and a method of learning-with-forgetting. The optimised structure has much less connections and can easily be interpreted in terms of fuzzy rules. Such a structure can be effectively used for on-line adaptation which is demonstrated on a phoneme-based speech recognition problem. © 1998 Elsevier Science Inc. All rights reserved.

Keywords: Fuzzy neural network; Speech recognition; Structural learning

1. Introduction

Different architectures of fuzzy neural networks (FNN) have been proposed as a knowledge engineering technique and used for various applications [1–8]. FNN systems 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

*Corresponding author. Tel.: +64 3 479 8319; fax: +64 3 479 8311; e-mail: nkasabov@otago.ac.nz.
dynamically changing data and situations [9,10]. In several publications [3,4] the
general architecture of a FNN called FuNN, which stands for Fuzzy Neural
Network, was introduced along with algorithms for learning, adaptation and
rules extraction. A further development of the FuNN principles and one partic-
ular implementation called FuNN/2 are presented in [5]. In spite of the obvious
advantages of FNN, real applications where large FNNs are used require much
fewer connections in order to achieve fast on-line training and adaptation. Pre-
liminary research shows that a large percentage of the connection weights in a
neural network and in a FNN in particular, are redundant. The task is how to
obtain the optimal structure of a FNN suitable for a fast adaptation without
sacrificing the accuracy of its performance. This is the main issue of this paper.

The paper consists of six sections. Section 2 presents the FuNN architecture
and its FuNN/2 implementation. Section 3 introduces the speech recognition
case study. Sections 4 and 5 introduce a genetic algorithm (GA) approach
and a learning-with-forgetting approach respectively, to optimise a FuNN
structure. Section 6 suggests how the above two approaches can be used in a
concert for effective adaptation of a FuNN and section seven gives conclusions
and directions for further research.

2. The architecture of FuNN

The FuNN model is designed to be used in a distributed, and eventually
agent-based, environment [3,4]. The architecture facilitates learning from data
and approximate reasoning, as well as fuzzy rules extraction and insertion. It
allows for the combination of both data and rules into one system, thus pro-
ducing the synergistic benefits associated with the two sources. In addition, it
allows for several methods of adaptation (adaptive learning in a dynamically
changing environment). FuNN uses a multi-layer perceptron (MLP) network
and a modified backpropagation training algorithm [3–5]. The general FuNN
architecture consists of five layers of neurons with partial feedforward connec-
tions as shown in Fig. 1. It is an adaptable FNN where the membership func-
tions (MFs) of the fuzzy predicates, as well as the fuzzy rules inserted before
training or adaptation, may adapt and change according to new data. A signif-
ificant feature of a FuNN is that each of the connections and nodes in it has a
semantic meaning. The meanings of the connections and nodes vary from layer
to layer. The various meanings attached to the FuNN elements make its func-
tioning interpretable. 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 which may take ei-
ther crisp or fuzzy values. These values are fed to the condition element layer
which performs fuzzification. This is implemented in FuNN/2 using three-point triangular MFs 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 MFs.

The triangular MFs are allowed to be non-symmetrical and any input value will belong to a maximum of two MFs with degrees differing from zero (it will always involve two unless the input value falls exactly on a MF 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. Using triangular MFs makes the fuzzification and the defuzzification procedures in FuNN fast without compromising the
accuracy of the solution. It also allows for using a modified gradient descent algorithm for adapting the MFs.

Initially the MFs 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 meaning of the memberships contained in this layer of connections some restrictions are placed on adaptation. Under the FuNN/2 architecture labels can be attached to weights when the network is constructed. When MF adaptation is taking place the centers are spatially constrained according to some constraining rules such as the membership function weight representing "low" will always have a center less than "medium", which will always be less than "high".

2.2. Rule layer

In the rule layer each node represents a single fuzzy rule. The layer is also potentially expandable in that nodes can be added to represent more rules as the network adapts or potentially shrinkable. The activation function is the sigmoidal logistic function with a variable gain coefficient $g$ (a default value of one is used giving the standard sigmoidal activation function). The semantic meaning of the activation of a node is that it represents the degree to which input data matches the antecedent component of an 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 MFs 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.

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 MF is supported by the current data used for recall. The activation function for the nodes of this layer is the sigmoidal logistic function with a 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 MFs, 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. 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 the constraining rules, much of the large body of theory regarding such networks is still applicable. For those results not immediately applicable to FuNN/2 the modifications are made much simpler given FuNN/2s natural structure and algorithm.

There are five versions of weight updating in the FuNN according to the mode of training and adaptation [3–5]. 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 training and adaptation are:

- (a) A partially adaptive version where the MFs 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 MFs to be used are known in advance or where the implementation is constrained by the problem in some way.
- (b) A partially adaptive version as in (a) but a forgetting factor is introduced as in [20].
- (c) A fully adaptive version with an extended backpropagation algorithm, as explained in [5]. This version allows for changes to be made to both rules and MFs, subject to constraints necessary for retaining semantic meaning.
- (d) A partially adaptive version with the use of a genetic algorithm for adapting the MFs. This mode does not alter the rules. The algorithm used is described in Section 5.
- (e) A fully adaptive training on the whole FuNN structure with the use of a generic algorithm (GA).

These modes can either be used as alternatives or they can be used together in whatever combination is most appropriate for the given problem at a certain time as discussed in Section 6. 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. A MS Windows version of FuNN/2, which is part of an integrated hybrid development tool called FuzzyCOPE/2 [4,9] is made available free from the WWW site:

\[ \text{http://divcom.otago.ac.nz:800/COM/INFOSCI/KEL/fuzzycop.htm} \]

FuNN/2 allows for different training and adaptation strategies to be tested before the most suitable is selected for a certain application. According to the type of connectivity and the number of inputs \( N_i \), number of connections \( N_c \) and number of nodes \( N_m \), there are at least two types of networks. Namely, fully connected networks (as the example on Fig. 1(a)) and partially connected ones. Reducing the connections and nodes to (near) optimum values according to the data and the task is discussed in Section 3.
3. The case study of phoneme-based speech recognition

Building adaptive speech recognition systems is an extremely difficult task to achieve [11]. A general architecture of a phoneme-based speech recognition system, where modular neural networks are used for recognising the English phonemes and fuzzy systems are used for modelling language linguistic rules, is shown in [12,13]. Here another approach is used as shown in Fig. 2. The system consists of several blocks as follows: signal processing; elementary sound (phoneme) recognition; language modelling; lookup table (partial match after a search in a dictionary); interface to the user (answer formation); adaptation (that is how the system adapts to new speakers).

Multi-modal, adaptive structure of FuNNs is used for building the adaptive phoneme recognition module. A separate FuNN specialises in recognising one phoneme or other elementary speech unit. This FuNN can accommodate linguistic knowledge in the form of fuzzy IF-THEN rules and can be trained on existing speech data (phoneme data). The input vectors in the currently experimented system are three 26-element mel-scale coefficient vectors (MSC) obtained after transforming three consecutive time frames of the signal, each of them of 12 msec duration, with 50% of overlap. A major problem is how to design optimal FuNNs having enough connections and MF to be trained on existing data and to adapt to new speakers, but not too many redundant connections which will make them (1) slow to adapt in real time, and (2) prone to local minima and overfitting [4]. In short, the task is to optimise a FuNN structure and make it suitable for adaption. In the experiments, 3 MFs are used which denote "low", "medium" and "high" values of each of the MSC. 3 MFs are also used for the output variable ("the uttered sound is unlikely to be this particular phoneme", "it may be", "the sound is likely to be the phoneme"). The whole system can adapt to a new speaker, whom the system did not recognise at the beginning, through discovering which are the particular sounds the system did not recognise correctly and which caused the problem, and then adapt the corresponding FuNNs. Adaptation of a FuNN can be done by applying different adaptation algorithms, such as genetic algorithm or additional training of the FuNN with the new data (taken from the new speaker). So, the FuNN structure should have the potential to adapt in a real time.

4. Using genetic algorithms for optimisation of MF in FuNN

Genetic algorithms (GAs) are highly effective search algorithms that are based on the principles of natural genetics and Darwinian selection [14]. By iteratively building on the success of previous attempts at a problem solution, GAs are able to rapidly explore large search spaces to find approximate solutions to difficult combinatorial problems. Three ways of using a GA to opti-
Fig. 2. A block diagram of HySpeech/2, a phoneme-based speech recognition system.
mise a FuNN structure have been explored here: (1) optimising MF of a FuNN structure; (2) optimising the whole structure; (3) optimising the input (feature) space. In this section, the method of optimising MFs through GA will be explored.

Given that the optimisation of fuzzy MFs in a FuNN structure may involve many changes to many different MF, and that a change to one MF may affect others, the large solution space possible for this problem suggests that a GA based approach may be effective. This has been previously investigated in [15] and has been shown to be more effective than manual alteration. Applying these principles to a FuNN structure can be seen as a logical extension of this previous work. The work carried out in [15] focused on the application of small changes to the width and centre positions of the MF of the system. These delta values may be easily encoded within the chromosome structure of a genetic algorithm. By measuring the performance of the modified system resulting from these changes, the effectiveness of the system and hence the fitness of the chromosome may be calculated.

A similar approach has been chosen here for optimising the FuNN structure. As FuNN automatically calculates the width of the MF, only changes to the centres need be represented by the chromosome strings.

The GA module optimises the MF of both the condition and action layers. Although some parameters such as the length of the chromosome are calculated automatically, the remaining parameters, such as population size, rate of mutation and the use of fitness normalisation are all user configurable. The GA used here is based upon Goldberg's simple genetic algorithm [14], with a few important exceptions. Firstly, chromosomes are represented as strings of floating point numbers, as opposed to the binary encoding schema used by Goldberg, with each floating point number (or gene) representing one delta value. Secondly, rather than altering the existing value of a gene during mutation, an entirely new value is assigned to the gene.

The algorithm for the GA system is conceptually very simple, and will be briefly described here. After the user has specified the FuNN to be optimised, the system automatically calculates the length of the chromosome necessary to represent the deltas to be applied to each MF attached to the input and output layer. As each MF is represented by a single connection weight, the length of the chromosome is equal to the sum of the number of nodes in the condition elements layer and the action elements layer. Although the values assigned to each gene of most individuals in the population are determined randomly, one individual of the population has deltas of all zero, which represents the original FuNN. This, in addition to elitism, ensures that the training error can never increase, only remain static or decrease.

After the population has been initialised, each individual is evaluated. To evaluate an individual, the delta values encoded within the individual's chromosome are applied to a 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, as those individuals with the lower RMS errors are more fit than those with higher errors.

After optionally normalising the fitness values a breeding population is selected using either tournament or roulette wheel selection. A new population is then created from the breeding population using one point crossover of the chromosomes. After an optional mutation phase, the old population is replaced by the new, with optional elitism preserving unchanged the most fit individual of the old population.

Fig. 3 shows the test classification accuracy of a FuNN classifier before and after a GA-based MF adaptation. The FuNN had the following structure: $3 \times 26$ MSC, 3 MFs for each MSC; 10 rule nodes; 3 output MFs; 1 output node. The FuNN has been trained on speech data representing the New Zealand English phoneme /e/. The FuNN was initially trained with the fixed mode backpropagation algorithm (where only the rules are allowed to change). Fig. 3 shows that the accuracy of the classification increases after GA adaptation of the MFs. Fig. 4 depicts histograms of the changing MFs after 100 iterations of GA training. It can be seen that the majority of the MFs did not

![Diagram](image)

**Fig. 3.** The test accuracy of a FuNN classifier for phoneme /e/ before and after a GA adaptation of the MF. The solid line represents the desired output values; dotted line indicates the result obtained from the FuNN.
change which may suggest that they are not important and therefore they would not affect the performance of the FuNN. The figure also shows that some of the MFs hit the 5% constraint for the MF change, which means that for a better result this limit may need to be extended to probably 10%. It is interesting to observe the dynamics of the MF change for all MFs (their number is $26 \times 3 \times 3 = 234$ for the input variables and 3 for the output variable). Three major types of temporal patterns can be distinguished:

1. Arbitrary oscillation of the weights between the limits prescribed by the adaptation constraint (in the present case $\pm 5\%$).
2. Certain delta-weights approach, after some initial changes, zero value and remain at that value during the further iterations.

Fig. 4. Histograms of the change in the MFs for the /æ/ phoneme-FuNN after 100 iterations of a GA. Each of the 78 inputs and one output of the FuNN have three MFs: 'small', 'medium', and 'large'.
3. An initially stable weight (which has zero delta value) starts to change from a given iteration step.

Using GA for full training and feature selection. Optimising all the connection weights in a connectionist structure can be done by using GA as an alternative training procedure. GAs can be used also for selecting the test set of features (inputs). Fig. 5(a)–(c) show test results from one experiment performed using GA for those two tasks. Another approach to optimising a neural network structure, and FuNN structure in particular, is discussed in the next section.

5. Using learning with forgetting for structural optimisation of fuzzy neural networks

5.1. Structural learning methods

In the past few years, significant efforts have been devoted to elaborate algorithms which find optimal neural network architectures. Two major approaches can be distinguished: either growing an increasingly elaborate network starting from a simple architecture, or reducing the size and complexity of an initially very complex neural networks; see [16,17]. The latter approach is called network pruning. In the present study, a special type of network pruning is applied, which is a modified backpropagation learning algorithm with forgetting the connection weights [18]. Modified backpropagation with forgetting belongs to the class of structural learning algorithms. By applying learning with forgetting, the weights decrease continuously, unless they are reinforced by the backpropagation rule. At the end of the training, only the essential weights deviate significantly from zero. By pruning the weights which are close to zero, a skeleton network is obtained.

Based on the skeleton structure, knowledge can be obtained about the analyzed patterns. A neural network which is able to create and process abstract knowledge is called an artificially intelligent network, compared to computational nets which simply process numerical data. The structural learning method applied in this work has the potential to generate artificially intelligent neural networks in the above sense. NNs trained by forgetting algorithm are expected to have better generalisation properties than the ones trained by standard BP, because the skeleton structure obtained after training with forgetting is usually more suitable for the given problem than a predefined architecture used in standard BP. By decreasing the effective number of weights during training with forgetting, the danger of overtraining can be reduced as well. A disadvantage of BP with forgetting is the increased computational time. This problem, however, can be compensated for by removing the unnecessary connections.
Fig. 5a,b.
5.2. Implementation of structural learning in the FuNN environment

In this section, the main features of the backpropagation learning algorithm with forgetting are summarised. The basic idea is to update the connection weights as follows [18]

\[ \Delta w_{ij} = \Delta w'_{ij} - \varepsilon \text{sgn}(w_{ij}). \]  

Here \( \Delta w'_{ij} \) is the change of the \( ij \)th weight using standard backpropagation algorithm, \( \varepsilon \) is the so-called forgetting rate. \( \text{sgn}(x) \) denotes the sign function. The second term on the right-hand side of Eq. (1) describes a decreasing tendency for the connection weights. Indeed, the weights decrease continuously, unless they are reinforced by the backpropagation rule. The corresponding cost function is given by

\[ J = \sum_i (y_i - y_i^*)^2 + \varepsilon \sum_{ij} |w_{ij}|. \]
Here \( y_i \) and \( y_i^* \) are the actual and the target values of the network outputs, respectively. \( e^t = \lambda e \), where \( \lambda \) is the learning rate and \( e \) is the forgetting rate. The first term is the usual sum of squared errors (SSE) between the actual and target values of the output of the NN. The second term is the sum of the absolute values of weights (SW) with an appropriate proportionality constant.

Structural learning with forgetting can be viewed as a pruning algorithm, in which a significantly reduced network is obtained after training. The second term in Eq. (2) represents the sum-of-weights penalty condition. A large variety of penalty terms are applied in the literature, including exponential weight decay, optimal brain damage, enthalpy pruning, lateral inhibition, etc. The results of comparative studies show that structural learning with forgetting scores very well compared with other methods as far as the discovery of regularities and generalisation capabilities are concerned [18–20]. For example, consider the widely used sum-of-square weights penalty term. This cost function results in weight changes which are proportional to the weights. Therefore, the change is slow in the case of small weights and fast for large weights. As a result, large weights easily decay and weights with smaller magnitude survive for a long time. This makes it difficult to infer information from the network structure and also causes degradation of the generalisation properties. The forgetting algorithm given by Eqs. (1) and (2) does not suffer from these difficulties.

At the beginning of the training the quadratic error function plays a dominant role in the cost function in Eq. (2). SSE drops quickly while the sum of the weights changes only slightly. As the learning advances, SSE changes less intensively, because cost function \( J \) can be reduced via decreasing the absolute values of the weighs as well. This can lead to a situation when the network output error decreases very slowly. The SSE can even increase and a moderate hill climbing can take place if the sum of weights is comparable with the SSE. The decreased convergence is a disadvantage of BP with forgetting. Nevertheless, this shortcoming can be compensated for by reducing the system size as the forgetting progresses and the number of active nodes decreases. Exactly this is done during Zeroing in the FuNN system.

The proper choice of the forgetting rate is crucial to the success of the modified BP with forgetting. If the forgetting rate is large, the weights decrease quickly and the network structure becomes rigid. In this case, the convergence of the training can be very poor, which causes unsatisfactory testing performance. If the forgetting rate is too small, the effect of forgetting appears slowly and very long training is needed to achieve proper performance of the network. Therefore, the optimum selection of the forgetting rate is crucial to the success of the structural learning. If the forgetting rate is properly selected, a better generalisation can be achieved by the forgetting algorithm than by the standard BP method and overtraining can be avoided.
5.3. Structural learning in phoneme recognition

Structural learning has been applied to the rule layers of FuNN, while the input and output layers remained unchanged, i.e., the MFs have not been optimised by the modified gradient descent algorithm. This task can be performed by a GA algorithm. The question of sharing the optimisation task between the GA and forgetting methods will be discussed in Section 6.

Various values of the forgetting constant $\varepsilon$ have been applied in the training experiments using data for the phoneme /el/. It has been found that $\varepsilon = 10^{-5}$ is an appropriate choice allowing good convergence and also facilitating proper pruning in order to extract rules from the trained network structure. Note that forgetting has been applied at each step of the pattern-by-pattern learning. Therefore, forgetting has been applied 444 times during one epoch which includes 444 training examples of the phoneme. The pattern-by-pattern learning strategy corresponds to the need of on-line phoneme identification in the framework of our present speech recognition projects. There are several methods for the optimisation of the forgetting mechanism, e.g., delayed forgetting, batch mode, and adaptive forgetting rate. These questions, however, go beyond the topic of the present study. In this paper we concentrate on the possibility of rule extraction from the phoneme data by applying standard, uniform forgetting.

Training is started with FuNN with 20 nodes in the rule layer. The training converges within a few 100 epochs with the SSE saturating at a level close to the value obtained by standard BP algorithm. In order to have proper forgetting level, training is conducted for 1000 epochs. At the end of the training, about 6 nodes remains active in the rule layer. This is illustrated in Fig. 6 where the weights connecting the rule layer and action layer are depicted. It is seen that the weight magnitudes of nodes #1, #2, #5, #7, #12, and #16 are much larger than those of the other nodes. Node #19 represents an intermediate state, while the other nodes have weight magnitudes well below two. Weights connecting the condition layer to the rule layer are shown in Fig. 7. The effect of forgetting is much less pronounced for these weights, nevertheless, a clear correlation can be observed between the weight distribution in Figs. 6 and 7, respectively. As rule node #2 is not supported by significant connections, it would not be used in the final, optimised FuNN structure. As a result of structural learning and consequent pruning of the FuNN classifier for the phoneme /el/, the initial FuNN structure ($N_i = 78$, $N_c = 4977$, $N_n = 336$) is significantly reduced and optimised. The corresponding values of the final structure being: $N_i = 21$, $N_c = 183$, $N_n = 66$.

In order to obtain rules from the data set, weights of magnitude below two are set to zero. FuNN has an automatic rule inferencing engine based on such zeroing technique, which gives six rules in the present case. Some of the rules are quite simple and have a support of just a few nodes in the input layer, while
other rules are more complicated and involve a wide range of the Mel-Scale Cepstrum. For illustration, two rules are given in Fig. 8. The first rule is affirmative with respect to the presence of phoneme /e/, i.e., it has a consequent part including C (== Yes). The second rule in Fig. 8 corresponds to the absence of /e/ via the presence of A (== No). It is remarkable that the rules shown in Fig. 8 are related to a narrow region of the Mel-Scale coefficients, namely, to the 1st, 4th and 5th, and 10th input nodes. Some of the other rules obtained for phoneme /e/ do have a support exceeding the above narrow bands. Nevertheless, the observed regularities open the way to the development of a self-learning, adaptive system for speech recognition.

6. Alternative and combined usage of the GA and the learning-with-forgetting methods for structural optimisation and on-line adaptation

As it can be seen from the material above, either of the above two methods, GA and learning-with-forgetting, can be used to optimise a FuNN structure.
GA has its strengths in fine tuning through random search. But the GA is computationally expensive, so it would be more efficient to use it on an already roughly optimised structure which has much less connections and nodes than the initial FuNN. The latter approach is a 'logical' way, rather than random, to keep the significant connections in a connectionist structure and the impor-

if <Input4 is C 2.23209> and <Input11 is C 2.0374> and <Input31 is A 2.22804> and <Input57 is A 2.84964> then <Output1 is not A 4.58342> and <Output1 is C 4.81485> else if <Input1 is not A 2.34047> and <Input4 is B 2.19175> and <Input27 is not A 2.57478> and <Input31 is not A 2.19999> and <Input36 is B 2.06656> and <Input36 is not C 2.11849> and <Input53 is not A 3.10538> and <Input57 is not A 2.69553> then <Output1 is A 3.64425> and <Output1 is not C 3.71175>

Fig. 8. Two simple rules obtained by FuNN after 1000 iterations and a final pruning at the level of ±2.
tant nodes (rules in the fuzzy rules interpretation of a FuNN). This method facili-
titates a better interpretation of a FuNN which results in a simpler set of rules extracted from it.

It is possible to design a FuNN optimisation strategy, in which learning-
with-forgetting and GA are used in a serial way. If the gradient-based meth-
od reaches saturation due to the local minimum problem, the GA algorithm can help to improve the FuNN performance due to its random search na-
ture. The combined use of the two methods can be realized iteratively by applying rough and fine tuning during the training process. It is a promising way of optimising a FuNN structure for a better on-line adaptation where GA can be used on already pruned FuNN with new data for the recall pro-
cess.

7. Conclusions and directions for further research

The paper presents one particular architecture called FuNN and discusses two alternative ways to optimise its structure, namely a genetic algorithm and a method of learning-with-forgetting. The optimised structure has much less connections and can easily be interpreted in terms of fuzzy rules. Such a structure can be effectively used for on-line adaptation which is demonstrated on a phoneme-based speech recognition problem. It is emphasised that learn-
ing with forgetting and GA are two different approaches to the optimisation of the FuNN structure. It is suggested that learning-with-forgetting is used for simplifying the network structure and for obtaining the first, sometimes rough rules. At a later stage, GA can be used for fine-tuning the architecture and the extracted rules.

GAs and learning with forgetting will be further experimented and used for phoneme recognition. These results will be incorporated into a compre-
hensive speech recognition system (see Fig. 2) and they will serve as a core module for a speaker-independent, adaptive system. The results of this speech recognition project will be reported elsewhere. Applications of the methodology presented here, are anticipated also in the area of adaptive control and data mining.

Acknowledgements

This research is supported by a research grant UOO 606 funded by the Pub-
lic Good Science Fund of the Foundation of Research Science and Technology (FRST) in New Zealand. We would like to thank Jaesoo Kim for conducting some of the experiments included in this paper.
References