APPLICATION OF A MODIFIED NEURAL FUZZY NETWORK AND AN IMPROVED GENETIC ALGORITHM TO SPEECH RECOGNITION

[1]K.F. Leung, [1]F.H.F. Leung, [2]H.K. Lam and [1]S.H. Ling
[1] *Centre for Multimedia Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[2] *Division of Engineering, King's College London, Strand, London, United Kingdom*

This paper presents the recognition of speech commands using a modified neural-fuzzy network (NFN). By introducing the associative memory (the tuner NFN) into the classification process (the classifier NFN), the network parameters could be made adaptive to changing input data. Then, the search space of the classification network could be enlarged by a single network. To train the parameters of the modified neural-fuzzy network, an improved genetic algorithm is proposed. As an application example, the proposed speech recognition approach is implemented in an eBook experimentally to illustrate the design and its merits.

Keywords: Neural network, genetic algorithm, fuzzy logic, speech recognition and pattern recognition.

I. Introduction

Speech is a natural tool for communications with others. However, when we want to communicate with a machine, it is quite difficult for the machine to recognize each of our spoken words. In general, the solution to a speech recognition problem involves two steps: feature extraction and classification.

Feature extraction is a preprocessing step for speech recognition. It is used to extract the specific voice features from the speech signals. In a noise free environment, each word or phoneme has its corresponding formant frequencies. However, when the environment is noisy, the speech signals are impure, and it is difficult to identify the corresponding features of speeches. The problem becomes more complicated when the speeches to be recognized have similar phonemes. Thus, researchers worked on developing some distinctive feature extraction techniques. The most commonly used approaches are filter-bank modeling and linear predictive coding (LPC) analysis [1]. Filter-bank modeling involves a bank of band-pass filters, which are used to model the characteristics of human ears. Since, human ears are sensitive to lower-frequency signals, the frequency scale of the band-pass filters are usually distributed in mel-scale or Bark-scale. LPC analysis [1] approximates the current sampled speech as a linear combination of its past samples. The time-domain speech signals are first windowed into frames, and the

autocorrelation coefficients between frames are obtained. This approach mimics the human vocal tract.

Classification is the next step to identify input speeches based on the feature parameters. Speech signals classification can be done in either a pattern recognition approach or a statistical approach. Neural networks (NNs) and hidden Markov model (HMM) are commonly employed in the pattern recognition approach and statistical approach respectively [1, 8]. NN is distinct in discrimination, and the classification can be done by measuring the closeness of the testing data to the training templates. However, a large number of mathematical operations will be required if the number of speech samples is large and the duration of the speech is long. HMMs are good at statistically modeling continuous speech signals. The states in the HMM characterize the phonemes. The speech is formulated into a sequence of states. HMM is usually used for continuous speech recognition.

Recognizing Cantonese-digit speeches is a challenge task. Cantonese is a nine-tonal and syllabic language [2]. Some Cantonese digits are difficult to discriminate when they are spoken in Cantonese, such as the digits '1' and '7'. Other human factors will introduce additional difficulties to achieving a good performance in Cantonese speech recognition. Good algorithms for the speech feature extraction and classification are important.

Notebook Computers and Personal Digital Assistants (PDAs) are changing our reading habit. Electronic Books (eBooks) have been winning their popularity as a kind of media that can offer rich contents and features within a small handheld device. An eBook Reader should have no keyboard or mouse. The main input device is a touch screen. As many functions are implemented in a single eBook Reader, it is not convenient to access these functions through menus and hot keys alone. By using a small microphone, a one-step commanding process using speeches is proposed for eBooks, so that when a user says a predefined word, the eBook can respond to the command represented by that word. To realize speech recognition for commanding, a variable-parameter neural-fuzzy network (NFN) trained by an improved GA is proposed in this paper. The proposed NFN consists of two NFNs such that one NFN is responsible for providing the parameters of another NFN. In this way, the trained NFN will have variable-parameters. Effectively, the associative memory for each recognized pattern will change according to the pattern itself. On applying the proposed NFN, the performance of speech recognition is improved. The proposed GA with improved genetic operations performs better as

compared with conventional GA in terms of the fitness value and convergence rate. In this paper, the variable-parameter NFN is used to recognize ten Cantonese-digit speeches, and implemented in an eBook Reader practically.

This paper is organized as follows. The variable-parameter neural-fuzzy network is presented in section II. The improved genetic algorithm is presented in section III. In order to test its searching performance, the improved genetic algorithm is applied in some benchmark test functions. The speech feature extraction and the classification procedures will be presented in section IV. The implementation and results of the recognizer will be reported in sections V and VI. A conclusion will be drawn in section VII.

II. Variable-parameter Neural-fuzzy Network

A variable-parameter neural-fuzzy network is proposed to recognize speeches. Referring to Fig. 1, the proposed NFN consists of two NFNs, namely a tuner NFN (associative memory) and a classifier NFN (processor). In general, the parameters of traditional NFNs are fixed after the training. In the proposed NFN, some parameters of the classifier NFN are adjusted by the tuner NFN (which have fixed parameters after training) to cope with the changing environment during the operation.

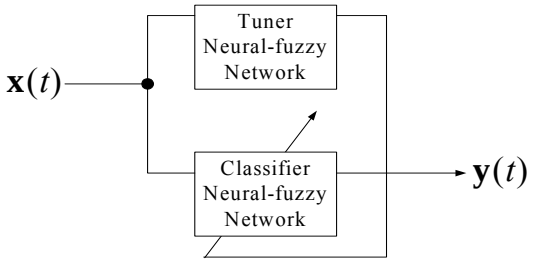


Fig. 1. Block diagram of the variable-parameter neural fuzzy network.

We use a fuzzy associative memory (FAM) [3-4, 18] type of rule base for both the tuner and classifier NFNs. An FAM is formed by partitioning the universe of discourse of each fuzzy variable according to the level of fuzzy resolution chosen for the antecedents, thereby generating a grid of FAM elements. The entry at each grid element in the FAM corresponds to a fuzzy premise. An FAM is thus interpreted as a geometric or tabular representation of a fuzzy logic rule base. The tuner or classifier NFN is shown in Fig. 2.

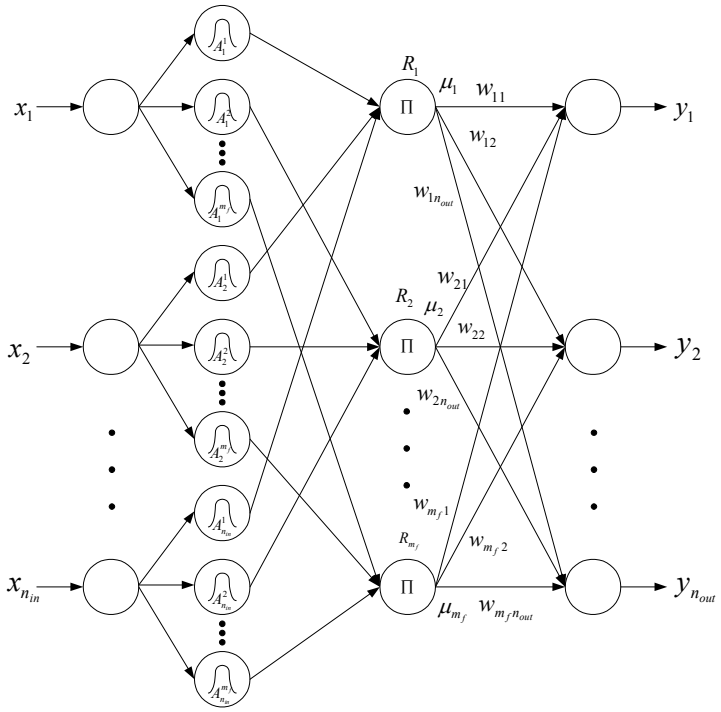


Fig. 2. Three-layer neural fuzzy network.

We define the input and output variables as x_i and y_j respectively; where i = 1, 2, …, n_{in}; n_{in} is the number of input variables; j = 1, 2, …, n_{out}; n_{out} is the number of output variables. The behavior of y_j of the NFN is governed by m_f fuzzy rules of the following format:

R_g: IF $x_1(t)$ is $A_1^g\ (x_1(t))$ AND $x_2(t)$ is $A_2^g\ (x_2(t))$ AND … AND $x_{n_{in}}(t)$ is $A_{n_{in}}^g\ (x_{n_{in}}(t))$

THEN $y_j(t)$ is w_{gj}, g = 1, 2, …, m_f; t = 1, 2, …, u (1)

where u denotes the number of input-output data pairs; m_f is the number of rules; $A_i^g\ (x_i(t)) = e^{\frac{-(x_i(t)-\bar{x}_{ig})^2}{2\sigma_{ig}^2}}$ is a bell-shaped membership function; $\bar{x}_{ig}$ and σ_{ig} are the mean value and the standard deviation of the membership function respectively; w_{gj}, j = 1, 2, …, n_{out}, is the output singleton of the rule g. The grade of the membership of each rule is defined as,

$$\mu_g(t) \in [0 \quad 1] = A_1^g\ (x_1(t)) \times A_2^g\ (x_2(t)) \times \ldots \times A_{n_{in}}^g\ (x_{n_{in}}(t)),\ g = 1, 2, \ldots, m_f \quad (2)$$

The j-th output of the tuner NFN, $y_j(t)$, is defined as,

$$y_j(t) = \frac{\sum_{g=1}^{m_f} \mu_g(t) w_{gj}}{\sum_{g=1}^{m_f} \mu_g(t)} \tag{3}$$

It should be noted that for this partially connected neural-fuzzy network, the number of rules m_f is equal to number of membership functions used for each input variables. The structures of both the tuner and classifier NFNs are the same, and the outputs are governed by (3). The main difference is that the outputs of the tuner NFN are the values of the output singleton, λ_{gj}, of the classifier NFN as shown in Fig. 3. Hence, the task of the tuner NFN is to bring the information corresponding to the input patterns into the associative memory to help the classification process. Referring to Fig. 3, the outputs of the turner NFN offer the values to the connection weights λ_{gj} of the classifier NFN according to the present input information. The connection weights λ_{gj} are thus varying during the operation of the classification. In practice, each input pattern will generate its own set of parameters for the operation of the classifier NFN. The output of the classifier NFN is shown as follows:

$$y_j(t) = \frac{\sum_{g=1}^{m_c} \mu_g^2(t) \lambda_{gj}}{\sum_{g=1}^{m_c} \mu_g^2(t)}, j = 1, 2, \ldots, n_{out} \tag{4}$$

where m_c is the number of rules of the classifier NFN.

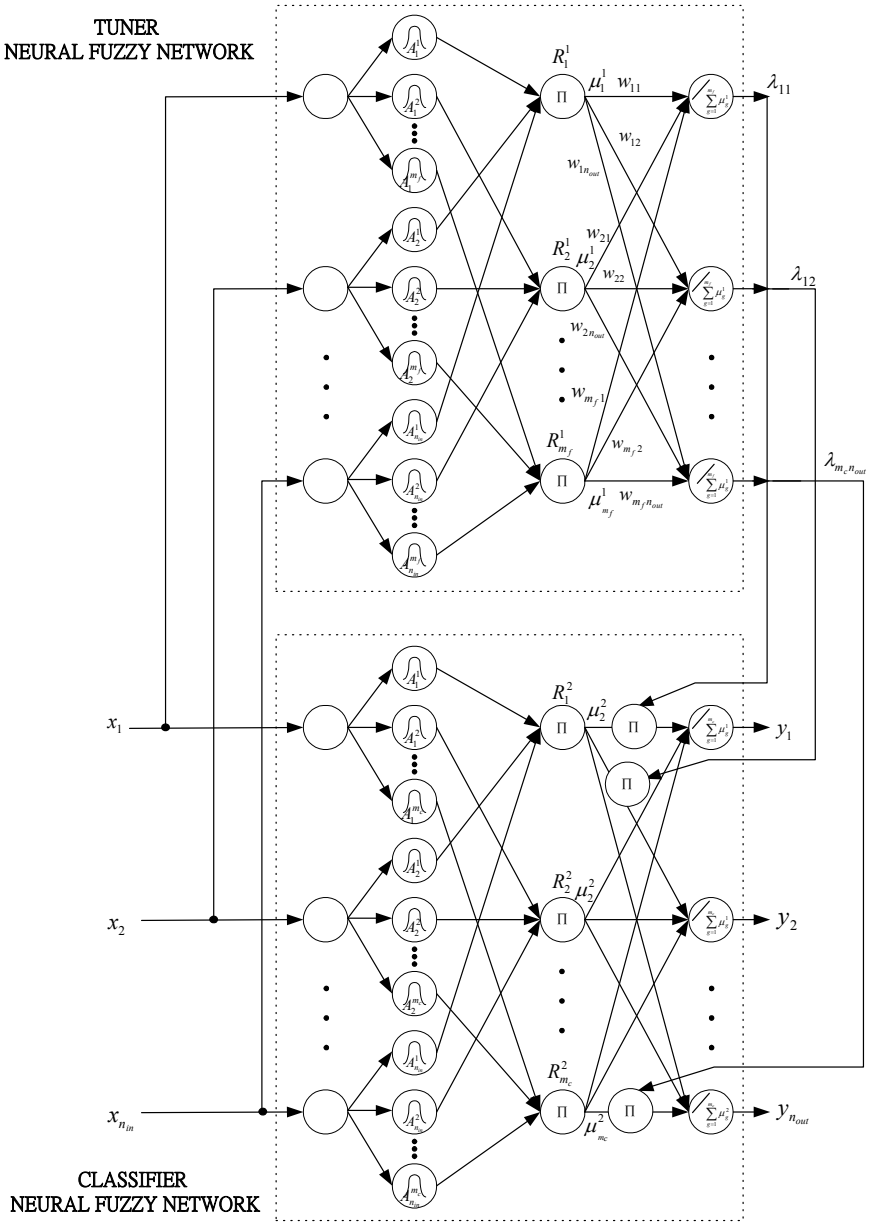


Fig. 3. The variable-parameter neural-fuzzy network

III. Improved Genetic Algorithm

Genetic algorithm (GA) is a directed random search technique [6] that is widely applied in optimization problems [6-8, 10]. This is especially useful for complex optimization problems where the number of parameters is large and the analytical global solutions are difficult to obtain.

GA has been applied in different areas such as fuzzy control [11-13, 17], path planning [14], greenhouse climate control [15], modeling and classification [16], etc.

Much effort has been spent to improve the performance of GA. Different selection schemes and genetic operators have been proposed. Selection schemes such as rank-based selection, steady-state election and tournament selection have been reported [21]. There are two kinds of genetic operations, namely crossover and mutation. Apart from random mutation and one-point crossover, other crossover and mutation algorithms have been proposed. For crossover, two-point crossover, multipoint crossover, arithmetic crossover and heuristic crossover have been reported [6, 20-22]. For mutation, boundary mutation, uniform mutation and non-uniform mutation can be found [6, 20-22].

An improved GA is presented here for training the network parameters of the proposed network. The conventional GA [6-7, 10] is modified by replacing the conventional crossover, mutation and reproduction operations with a set of operations that better suit the training of network parameters of the proposed network. The improved GA process is shown in Fig. 4 and its details will be given as follows.

```
Procedure of the improved GA
begin
        τ→0                 // τ: number of iteration
        initialize P(τ)     //P(τ): population for iteration τ
        evaluate f(P(τ))    // f(P(τ)):fitness function
while (not termination condition) do
        begin
        τ→τ+1
        select 2 parents p1 and p2 from P(τ–1)
        perform crossover operation with p_c according to equations (10) - (16)
        perform mutation operation with p_m according to equations (17) - (22)
         to generate the offspring os
        // reproduce a new P(τ)
                if random number < p_a   // p_a: probability of acceptance
                    os replaces the chromosome with the smallest fitness value in the
                    population
                else if f(os) > smallest fitness value in the P(τ–1)
                    os replaces the chromosome with the smallest fitness value
                end
        evaluate f(P(τ))
        end
end
```

Fig. 4. Procedure of the improved GA.

A. Initial Population

The initial population is a potential solution set *P*. The first set of population is usually generated randomly.

$$P = \left\{ \mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{pop\_size} \right\} \tag{5}$$

$$\mathbf{p}_i = \left[p_{i_1} \quad p_{i_2} \quad \cdots \quad p_{i_j} \quad \cdots \quad p_{i_{no\_vars}} \right],$$

$$i = 1, 2, \ldots, pop\_size; j = 1, 2, \ldots, no\_vars \tag{6}$$

$$para^j_{\min} \le p_{i_j} \le para^j_{\max} \tag{7}$$

where *pop_size* denotes the population size; *no_vars* denotes the number of variables to be tuned; p_{i_j}, $i = 1, 2, \ldots, pop\_size$; $j = 1, 2, \ldots, no\_vars$, are the parameters to be tuned; $para^j_{\min}$ and $para^j_{\max}$ are the minimum and maximum values of the parameter p_{i_j} respectively for all *i*. It can be seen from (5) to (7) that the potential solution set *P* contains some candidate solutions $\mathbf{p}_i$ (chromosomes). The chromosome $\mathbf{p}_i$ contains some variables p_{i_j} (genes).

B. Evaluation

Each chromosome in the population will be evaluated by a defined fitness function. The better chromosomes will return higher values in this process. The fitness function to evaluate a chromosome in the population can be written as,

$$fitness = f(\mathbf{p}_i) \tag{8}$$

The form of the fitness function depends on the application.

C. Selection

The selection of a new population is done with respect to a probability distribution based on fitness values. The selection method of spinning the roulette wheel [6] is used. It is believed that high potential parents will produce better offspring (survival of the best ones). The chromosome having a higher fitness value should therefore have a higher chance to be selected. The selection can be done by assigning a probability q_i to the chromosome $\mathbf{p}_i$:

$$q_i = \frac{f(\mathbf{p}_i)}{\sum_{k=1}^{pop\_size} f(\mathbf{p}_k)}, \; i = 1, 2, \ldots, pop\_size \tag{9}$$

The cumulative probability $\hat{q}_i$ for the chromosome $\mathbf{p}_i$ is defined as,

$$\hat{q}_i = \sum_{k=1}^{i} q_k \text{ , } i = 1, 2, \ldots, pop\_size \tag{10}$$

The selection process starts by randomly generating a nonzero floating-point number, $d \in [0 \quad 1]$. Then, the chromosome $\mathbf{p}_i$ is chosen if $\hat{q}_{i-1} < d \le \hat{q}_i$ ($\hat{q}_0 = 0$). It can be observed from this selection process that a chromosome having a larger $f(\mathbf{p}_i)$ will have a higher chance to be selected. Consequently, the best chromosomes will get more offspring, the average will stay and the worst will die off.

D. Genetic Operations

The genetic operations are to generate some new chromosomes (offspring) from their parents after the selection process. They include the crossover and the mutation operations.

Crossover

The crossover operation is mainly for exchanging information between two parents ($\mathbf{p}_1$ and $\mathbf{p}_2$) that are obtained by the selection operation. In the crossover operation, a probability of crossover p_c will be adopted, which gives the expected number $p_c \times pop\_size$ of chromosomes that undergo the crossover operation in a generation. First, four chromosomes will be generated according to the following equations,

$$\mathbf{os}_c^1 = \left[os_1^1 \quad os_2^1 \quad \cdots \quad os_{no\_vars}^1\right] = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} \tag{11}$$

$$\mathbf{os}_c^2 = \left[os_1^2 \quad os_2^2 \quad \cdots \quad os_{no\_vars}^2\right] = \mathbf{p}_{\max}(1-w) + \max(\mathbf{p}_1, \mathbf{p}_2)w \tag{12}$$

$$\mathbf{os}_c^3 = \left[os_1^3 \quad os_2^3 \quad \cdots \quad os_{no\_vars}^3\right] = \mathbf{p}_{\min}(1-w) + \min(\mathbf{p}_1, \mathbf{p}_2)w \tag{13}$$

$$\mathbf{os}_c^4 = \left[os_1^4 \quad os_2^4 \quad \cdots \quad os_{no\_vars}^4\right] = \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1-w) + (\mathbf{p}_1 + \mathbf{p}_2)w}{2} \tag{14}$$

$$\mathbf{p}_{\max} = \left[para_{\max}^1 \quad para_{\max}^2 \quad \cdots \quad para_{\max}^{no\_vars}\right] \tag{15}$$

$$\mathbf{p}_{\min} = \left[para_{\min}^1 \quad para_{\min}^2 \quad \cdots \quad para_{\min}^{no\_vars}\right] \tag{16}$$

where $w \in [0 \quad 1]$ denotes the weight to be determined by users, $\max(\mathbf{p}_1, \mathbf{p}_2)$ denotes the vector with each element obtained by taking the maximum among the corresponding element of $\mathbf{p}_1$ and $\mathbf{p}_2$. For instance, $\max([1 \quad -2 \quad 3], [2 \quad 3 \quad 1]) = [2 \quad 3 \quad 3]$. Similarly, $\min(\mathbf{p}_1, \mathbf{p}_2)$ gives a vector

by taking the minimum value. For instance, $\min([1 \quad -2 \quad 3],[2 \quad 3 \quad 1]) = [1 \quad -2 \quad 1]$. Among $\mathbf{os}_c^1$ to $\mathbf{os}_c^4$, the one with the largest fitness value is used as the offspring of the crossover operation. The offspring is defined as,

$$\mathbf{os} \equiv \begin{bmatrix} os_1 & os_2 & \cdots & os_{no\_vars} \end{bmatrix} = \mathbf{os}_c^{i_{os}} \tag{17}$$

i_{os} denotes the index i which gives a maximum value of $f\left(\mathbf{os}_c^i\right)$, i = 1, 2, ,3 ,4.

If the crossover operation can provide a good offspring, a higher fitness value can be reached in less iteration. In general, two-point crossover, multipoint crossover, arithmetic crossover or heuristic crossover can be employed to realize the crossover operation [6, 20-22]. The offspring generated by these methods, however, may not be better than that from our approach. As seen from (11) to (14), the potential offspring after the crossover operation spreads over the domain. While (1`) and (14) result in searching around the centre region of the domain (a value of w near to 1 in (14) can move $\mathbf{os}_c^4$ to be near $\frac{\mathbf{p}_1 + \mathbf{p}_2}{2}$), (12) and (13) move the potential offspring to be near the domain boundary (a small value of w in (12) and (13) can move $\mathbf{os}_c^2$ and $\mathbf{os}_c^3$ to be near $\mathbf{p}_{\max}$ and $\mathbf{p}_{\min}$ respectively).

Mutation

The offspring (17) will then undergo the mutation operation. The mutation operation is to change the genes of the chromosomes. Consequently, the features of the chromosomes inherited from their parents can be changed. In general, various methods like boundary mutation, uniform mutation or non-uniform mutation [6, 20-22] can be employed to realize the mutation operation. In this paper, a different process of mutation is proposed. The details are as follows. Every gene of the offspring **os** of (17) will have a chance to mutate governed by a probability of mutation, $p_m \in [0 \quad 1]$, which is defined by the user. This probability gives an expected number ($p_m \times no\_vars$) of genes that undergo the mutation. For each gene, a random number between 0 and 1 will be generated such that if it is less than or equal to p_m, the operation of mutation will take place on that gene and updated instantly. The gene of the offspring of (17) is then mutated by:

$$\hat{os}_k = \begin{cases} os_k + \Delta o_{s_k}^U \text{ if } f(\mathbf{os} + \Delta \mathbf{o}_{\mathbf{s}_k}^U) \geq f(\mathbf{os} - \Delta \mathbf{o}_{\mathbf{s}_k}^L) \\ os_k - \Delta o_{s_k}^L \text{ if } f(\mathbf{os} + \Delta \mathbf{o}_{\mathbf{s}_k}^U) < f(\mathbf{os} - \Delta \mathbf{o}_{\mathbf{s}_k}^L) \end{cases}, k = 1, 2, \ldots, no\_vars \quad (18)$$

where

$$\Delta o_{s_k}^U = w_{m_k} r\left(para_{\max}^k - os_k\right) \quad (19)$$

$$\Delta o_{s_k}^L = w_{m_k} r\left(os_k - para_{\min}^k\right) \quad (20)$$

$$\Delta \mathbf{o}_{\mathbf{s}_k}^U = \begin{bmatrix} 0 & 0 & \cdots & \Delta o_{s_k}^U & \cdots & 0 \end{bmatrix} \quad (21)$$

$$\Delta \mathbf{o}_{\mathbf{s}_k}^L = \begin{bmatrix} 0 & 0 & \cdots & \Delta o_{s_k}^L & \cdots & 0 \end{bmatrix} \quad (22)$$

$r \in [0 \quad 1]$ is a randomly generated number; $w_{m_k} \in (0 \quad 1]$ is a weight governing the magnitudes of $\Delta o_{s_k}^U$ and $\Delta o_{s_k}^L$. The value of weight w_{m_k} is varied by the value of $\frac{\tau}{T}$, where τ is the iteration number and T is the total number of iteration. In order to perform a local search, the value of weight w_{m_k} should become small as $\frac{\tau}{T}$ increases in order to reduce the significance of the mutation. Based on this idea, a monotonic decreasing function governing w_{m_k} is proposed as follows,

$$w_{m_k} = w_f \left(1 - \frac{\tau}{T}\right)^{\frac{1}{w_\tau}} \geq 0 \quad (23)$$

where $w_f \in [0 \quad 1]$ and $w_\tau > 0$ determine the initial value and the decay rate respectively. Their values are chosen by the user. For a large value of w_f, it can be seen from (19) and (20) that $\Delta o_{s_k}^U \approx r\left(para_{\max}^k - os_k\right)$ and $\Delta o_{s_k}^L \approx r\left(os_k - para_{\min}^k\right)$ initially as $\left(1 - \frac{\tau}{T}\right)^{\frac{1}{w_\tau}} \approx 1$ which ensure a large search space. When the value of $\left(1 - \frac{\tau}{T}\right)^{\frac{1}{w_\tau}} \approx 0$, the values of $\Delta o_{s_k}^U$ and $\Delta o_{s_k}^L$ are small to ensure a small search space for fine tuning.

E. Reproduction

The new offspring will be evaluated using the fitness function of (8). This new offspring will replace the chromosome with the smallest fitness value among the population if a randomly generated number between 0 and 1 is smaller than $p_a \in [0 \quad 1]$, which is the probability of

acceptance defined by the user. Otherwise, the new offspring will replace the chromosome with the smallest fitness value if the fitness value of the offspring is greater than the fitness value of that chromosome in the population. p_a is effectively the probability of accepting a bad offspring in order to reduce the chance of converging to a local optimum. Hence, the chance of reaching the global optimum is kept.

After the operation of selection, crossover, mutation and reproduction, a new population is generated. This new population will repeat the same process. Such an iterative process can be terminated when it meets a defined condition, e.g. a defined number of iteration has been reached.

F. Benchmark Test Functions

Some benchmark test functions [8-9, 19] are used to examine the applicability and efficiency of the improved GA. Six test functions, $f_i(\mathbf{x})$, i = 1, 2, 3, 4, 5, 6 will be used, where $\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^{\mathrm{T}}$; n is an integer denoting the dimension of the vector $\mathbf{x}$. The six test functions are defined as follows.

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \,, \; -5.12 \le x_i \le 5.12 \tag{24}$$

where n = 3 and the minimum point is at $f_1(0, 0, 0) = 0$.

$$f_2(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100\left(x_{i+1} - x_i^{\,2}\right)^2 + \left(x_i - 1\right)^2 \right), \; -2.048 \le x_i \le 2.048 \tag{25}$$

where n = 2 and the minimum point is at $f_2(0, 0) = 0$.

$$f_3(\mathbf{x}) = 6n + \sum_{i=1}^{n} floor(x_i) \,, \; -5.12 \le x_i \le 5.12 \tag{26}$$

where n = 5 and the minimum point is at $f_3([5.12, 5], \ldots, [5.12, 5]) = 0$. The floor function, $floor(\cdot)$, is to round down the argument to the nearest integer.

$$f_4(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + Gauss(0,1) \,, \; -1.28 \le x_i \le 1.28 \tag{27}$$

where n = 3 and the minimum point is at $f_4(0, 0, 0) = 0$. $Gauss(0, 1)$ is a function to generate uniformly a floating-point number between 0 and 1 inclusively.

$$f_5(\mathbf{x}) = \frac{1}{k} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6}, \quad -65.356 \le x_i \le 65.356 \tag{28}$$

where

$$\mathbf{a} = \{a_{ij}\} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ 32 & 32 & 32 & 32 & 32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & & & & 16 & 16 & 16 & 16 & 16 \\ & & & & & & & & & & 32 & 32 & 32 & 32 & 32 \end{bmatrix},$$

$k = 500$ and the maximum point is at $f_5(32, 32) \approx 1$.

$$f_6(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^{\,2} - 10\cos(2\pi x_i) + 10 \right], \; -5.12 \le x_i \le 5.12 \tag{29}$$

where $n = 3$ and the minimum point is at $f_6(0, 0, 0) = 0$.

It should be noted that the minimum values of all functions in the defined domain are zero except for $f_5(\mathbf{x})$. The fitness functions for f_1 to f_4 and f_6 are defined as,

$$fitness = \frac{1}{1 + f_i(\mathbf{x})}, \; i = 1, 2, 3, 4, 6. \tag{30}$$

and the fitness function for f_5 is defined as,

$$fitness = f_5(\mathbf{x}) \tag{31}$$

The proposed GA goes through these 6 test functions. The results are compared with those obtained by the standard GA with arithmetic crossover and non-uniform mutation [6, 20-22]. The control parameters of the proposed GA and the standard GA are tabulated in Table I. These control parameters are selected by trial and error through experiments for good performance. The population size is 50. The initial values of **x** in the population for a test function are set to be the same for both the proposed and the standard GAs. The number of iteration for each test function is listed in Table II. For test functions 1 to 6, the initial values are $[1 \;\; 1 \;\; 1]$, $[0.5 \;\; 0.5]$, $[1 \;\; \cdots \;\; 1]$, $[0.5 \;\; \cdots \;\; 0.5]$, $[10 \;\; \cdots \;\; 10]$ and $[1 \;\; 1 \;\; 1]$ respectively. The results of the average fitness values over 50 times of simulations based on the proposed and standard GAs are shown in Fig. 5 and tabulated in Table II. Generally, it can be seen that the performance of the proposed GA is better than that of the standard GA in terms of the convergences rate and the fitness value.

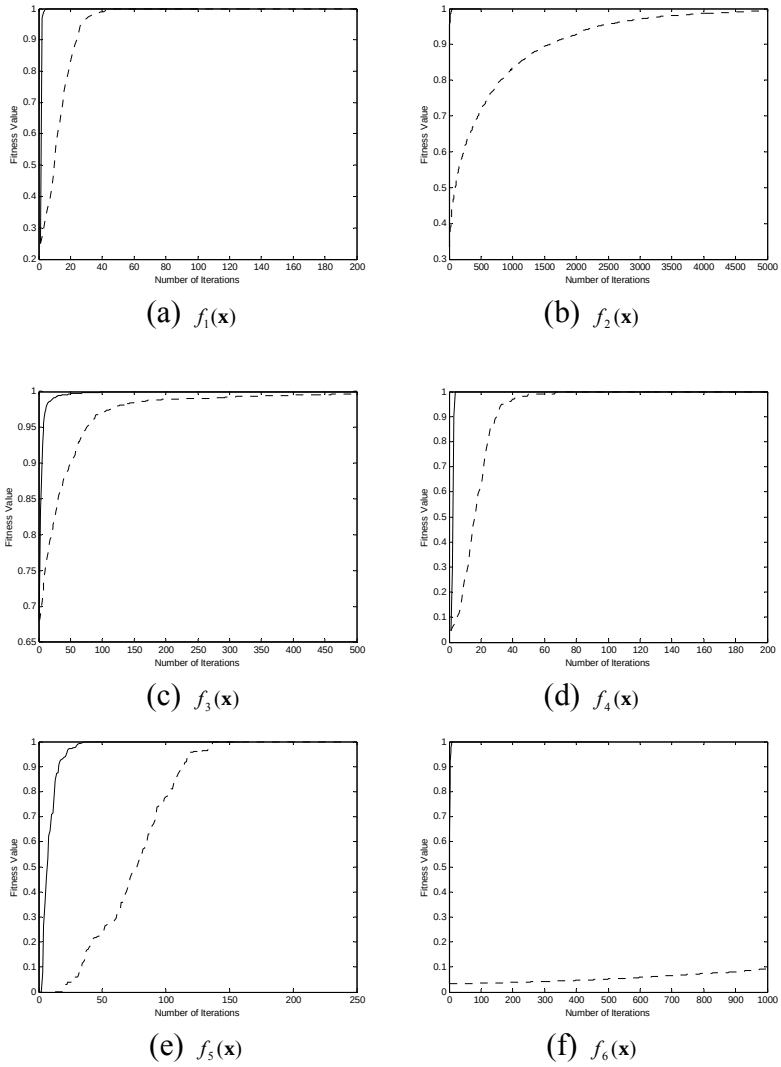


Fig. 5. The average fitness value using the improved GA (solid line) and the standard GA (dotted line) for different test functions.

IV. Speech Command Recognition in an eBook

The purpose of speech recognition in an eBook reader is to provide a voice command environment for the users. A block diagram of the speech recognition system is shown in Fig. 6.

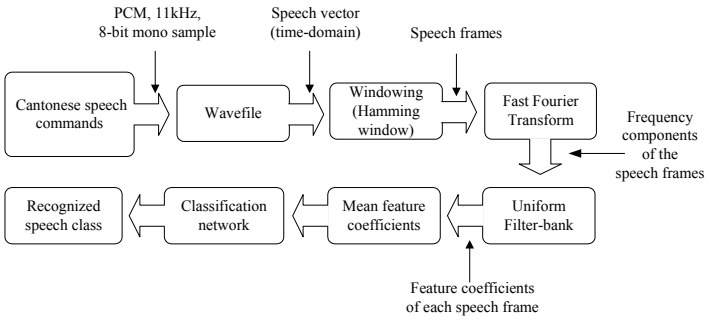


Fig. 6. Block diagram of the Cantonese-digit speech recognition system.

To command the eBook, speech signals are recorded from the microphone of the eBook. They are in mono, 8-bit PCM format sampled at 11kHz. Then the speech signal in time-domain $s(\tau)$ is windowed by 128-sample Hamming windows $wh(\cdot)$ with 50% overlap to form speech frames; where

$$wh(\tau) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi\tau}{127}\right), & 0 \le \tau \le 127 \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

The windowed speech signals will be transformed into frequency spectrums using fast Fourier transform (FFT). The real part of the frequency components at the n-th speech frame is defined as follows.

$$\mathbf{Sf}_n = \begin{bmatrix} Sf_n(0) & Sf_n(1) & \cdots & Sf_n(127) \end{bmatrix} = Re\{\text{FFT}([wh(\tau)s_n(\tau)])\}, \quad 0 \le \tau \le 127 \tag{33}$$

where $Re\{\cdot\}$ denotes the real part of the argument vector, and $s_n(\tau)$ is the τ-th element of the n-th speech frame.

A uniform filter-bank is then applied to process the speech frames in order to retrieve the feature coefficients (one filter gives one coefficient.) The process done by the uniform filter-bank can be described as follows.

$$c_n^{\beta} = \frac{20\log_{10}\sum_{l=1}^{\alpha} Sf_n(\rho_\beta + l - 1)}{\alpha}, \quad \beta = 1, 2, \ldots, no\_filter \tag{34}$$

$$\alpha = floor\left(\frac{128}{no\_filter}\right) \tag{35}$$

$$\rho_\beta = \alpha(\beta - 1), \ \beta = 1, 2, \ldots, no\_filter \tag{36}$$

$$\mathbf{D}_n = \left[c_n^2 - c_n^1 \quad c_n^3 - c_n^2 \quad \cdots \quad c_n^{no\_filter} - c_n^{no\_filter-1}\right] \tag{37}$$

where c_n^β denotes the mean power of a speech frame generated by the n-th band-pass filter for the n-th frame, *no_filter* denotes the number of band-pass filter, α denotes the number of the frequency components entering the band-pass filter, *floor*(·) denotes the floor function which is used to round-up a floating point number; $\mathbf{D}_n$ is a vector formed by magnitude differences between two consecutive band-pass filter outputs of the n-th speech frame. The mean feature coefficient of all the speech frames will be calculated and they will be normalized as the neural network inputs, which perform the template matching classification process. The training of the proposed NFN is to maximize the following fitness value:

$$fitness = \frac{1}{1 + err} \tag{38}$$

$$err = \frac{\sum_{t=1}^{num\_pat} \frac{\left\|\mathbf{y}_d(t) - \mathbf{y}(t)\right\|^2}{num}}{num\_pat} \tag{39}$$

where $\mathbf{y}_d(t)$ denotes the desired output vector; $\mathbf{y}(t)$ denotes the network output vector, *num* denotes the dimension of the output vector (the number of classes to be recognized), *num_pat* denotes the number of training patterns. The desired output vector of the network is defined as follows.

$$\mathbf{y}_d = [a_1, a_2, a_3, \ldots, a_{num}] \tag{40}$$

where a_i, i = 1, 2, …, *num*, describes the target class of the system. Only the value of a_i for a particular class i is equal to 1, and the rest elements of $\mathbf{y}_d$ are all zero. The improved GA will be employed to train the modified NFN. During the recognition, the position of the element of $\mathbf{y}(t)$ that has the largest value indicates the possible class of the input pattern.

V. Application and Results

The speech recognition approach as shown in Fig. 6 is implemented in a practical eBook reader. The Cantonese speech signals are sampled at 11kHz, 8-bit mono. The speech samples are recorded from a male speaker and the amplitude of each sample is normalized to lie between

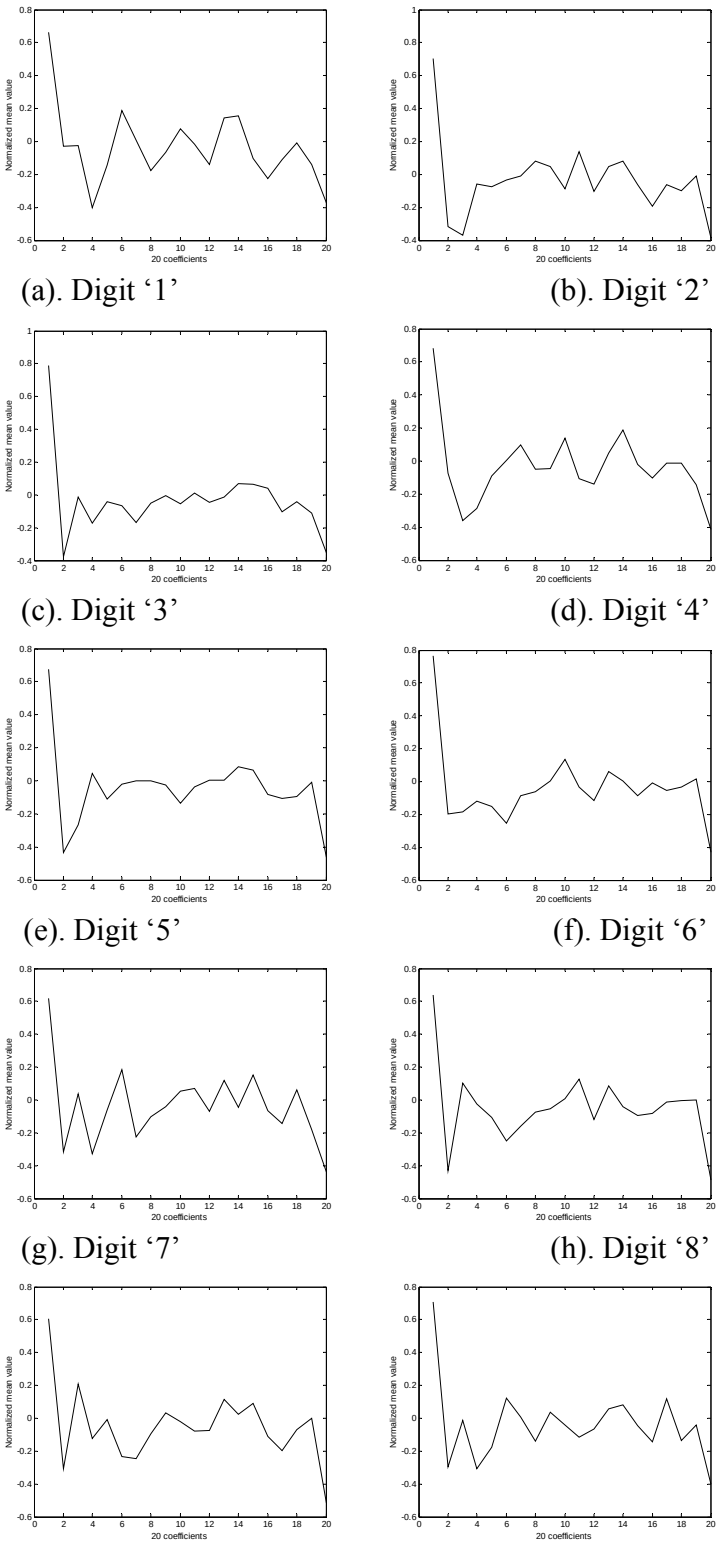
–1 and 1. 50 training patterns and 20 testing patterns for each digit are collected. The Cantonese-digit speeches are '0' to '9' (10 classes). We use the method of the last section to obtain 20 feature parameters representing each digit. Examples of the feature parameters of the digits '0' to '9' are shown in Fig. 7.

The speech feature coefficients of the ten Cantonese-digit speeches are processed by the variable-parameter NFN with 20 inputs and 10 outputs. The improved GA is employed to train the modified NFN based on the training patterns in order to maximize the following fitness function.

$$fitness = \frac{1}{1+err} \tag{41}$$

$$err = \frac{\sum_{t=1}^{500} \frac{\left\|\mathbf{y}_d(t) - \mathbf{y}(t)\right\|^2}{10}}{500} \tag{42}$$

The number of training patterns for the modified NFN is 500. The first 50 training patterns are the feature parameters of the Cantonese digit '1'. Thus, the first 50 vectors of $\mathbf{y}_d$ are [1 0 0 0 0 0 0 0 0 0]. Similarly, the next 50 training patterns are the feature parameters of the Cantonese digit '2', and the corresponding 50 vectors of $\mathbf{y}_d$ are [0 1 0 0 0 0 0 0 0 0], and so on. The number of membership functions used for the tuner and classifier NFN are (3,5), (4,5), (5,5), (5,4) and (5,3) where the first number in the bracket is the number of membership functions used by the tuner NFN, and the second number in the bracket is the number of membership functions used by the classifier NFN. The learning process is carried out by a personal computer with a P4 1.4GHz CPU and 256 MB RAM. The number of iteration for training is 50000; $w = 0.5$, $p_m = 0.01$; $w_f = 0.5$, $w_\tau = 5$, $p_a = 0.1$ for all cases. After training, 200 testing patterns (10 Cantonese-digits × 20) are used to test the performance of the proposed NFN. The fitness values and the errors are the measures of the network performance. The training fitness values are tabulated in Table III. The testing fitness values and the testing errors for each spoken digit are tabulated in Tables IV and V respectively. From Table V, it can be seen that the best result of 98.5% accuracy is obtained when the numbers of membership functions in the tuner NFN and the classifier NFN are 5 and 5 respectively.



(a). Digit ‘1’ (b). Digit ‘2’

(c). Digit ‘3’ (d). Digit ‘4’

(e). Digit ‘5’ (f). Digit ‘6’

(g). Digit ‘7’ (h). Digit ‘8’

(i). Digit ‘9’ (j). Digit ‘10’

Fig. 7. Examples of the first speech feature pattern (20 feature parameters) for the digits ‘0’-‘9’.

VI. Comparison

For comparison, a traditional NFN with different number of membership functions trained by the genetic algorithm with arithmetic crossover and non-uniform mutation [10] is also used to do the recognition. The number of iteration is 50000. The shape parameter, probability of crossover and probability of mutation of the GA [10] are chosen to be 1, 0.8 and 0.05 respectively. The results are summarized in Tables VI to VIII. It can be seen that the performance of the proposed NFN is generally better than that of the traditional NFN. When the same number of parameters (650) is used, the proposed NFN provides a better result. The accuracy of recognition for the testing patterns is 98.5% for the proposed method and 97% for the traditional method. It is interesting to see that the recognition error of the proposed NFN increases when the number of membership functions in the tuner NFN is bigger than that of the classifier NFN. Referring to Table IV and Table V, the recognition accuracy drops from 89% of the network working with a (3, 5) membership function combination to 79% of the network working with a (5, 3) membership function combination. It indicates that the number of membership functions in the classifier NFN should be large in the classification process. It is because a larger classifier NFN requires a larger number of associative memory outputs. In this example, the number of parameters supplied by the tuner NFN for a (3, 5) membership function combination is 50, while only 30 parameters can be supplied by the tuner NFN for a (5, 3) membership functions combination. In other words, if the classifier NFN is larger, more information can be acquired from the tuner NFN after the network has been trained.

Test function	w	p_c	p_m	w_f	w_τ	p_a
$f_1(\mathbf{x})$	0.1	0.8	0.3	0.001	0.001	0.1
$f_2(\mathbf{x})$	0.5	0.8	0.5	0.01	10	0.1
$f_3(\mathbf{x})$	0.1	0.8	0.8	1	1000	0.1
$f_4(\mathbf{x})$	0.5	0.8	0.35	0.001	10	0.1
$f_5(\mathbf{x})$	0.5	0.8	0.8	0.1	0.1	0.1
$f_6(\mathbf{x})$	0.01	0.8	0.1	0.01	0.01	0.1

(a)

Test function	b, (Shape parameter of mutation)	p_c (Probability of crossover)	p_m (Probability of mutation)
$f_1(\mathbf{x})$	5	0.7	0.9
$f_2(\mathbf{x})$	1	0.8	0.8
$f_3(\mathbf{x})$	0.1	0.7	0.6
$f_4(\mathbf{x})$	1	0.8	0.35
$f_5(\mathbf{x})$	0.1	0.8	0.5
$f_6(\mathbf{x})$	0.9	0.7	0.4

(b)

Table I. Control parameters of GAs for the benchmark test functions: (a) improved GA, (b) standard GA with arithmetic crossover and non-uniform mutation.

Test function	Average fitness value from proposed GA	Average fitness value from standard GA	Number of iteration
$f_1(\mathbf{x})$	1.0000	1.0000	200
$f_2(\mathbf{x})$	1.0000	0.9997	5000
$f_3(\mathbf{x})$	0.9996	0.9924	500
$f_4(\mathbf{x})$	1.0000	1.0000	200
$f_5(\mathbf{x})$	1.0000	1.0000	250
$f_6(\mathbf{x})$	1.0000	0.0986	1000

Table II. Average fitness values obtained from the proposed GA and the traditional GA for the benchmark test functions.

Membership functions combination	*(3,5)*	*(4,5)*	*(5,5)*	*(5,4)*	*(5,3)*
Fitness value	0.9968	0.9963	0.9995	0.9969	0.9940
Number of parameters	470	560	650	560	470

Table III. Fitness values under different combinations of numbers of membership functions in the variable-parameter NFN (50 training patterns for each digit).

Membership functions combination	*(3,5)*	*(4,5)*	*(5,5)*	*(5,4)*	*(5,3)*
Fitness value	0.9818	0.9855	0.9952	0.9854	0.9675

Table IV. Fitness values under different combinations of numbers of membership functions in the variable-parameter NFN (20 testing patterns for each digit).

	Membership functions combination				
Digit #	*(3,5)*	*(4,5)*	*(5,5)*	*(5,4)*	*(5,3)*
1	5	5	0	13	4
2	1	1	1	1	3
3	5	3	0	1	1
4	2	1	1	0	3
5	0	0	0	0	0
6	2	1	0	0	1
7	0	2	0	3	15
8	5	3	0	0	13
9	1	1	1	0	1
0	1	0	0	1	1

Table V. Number of recognition errors for the Cantonese digits '0'-'9' using the proposed NFN (20 testing patterns for each digit).

Number of membership functions	*9*	*10*	*11*	*12*	*13*
Fitness value	0.9891	0.9930	0.9655	0.9957	0.9962
No. of parameters	450	500	550	600	650

Table VI. Fitness values of the traditional neural-fuzzy network trained by the genetic algorithm with arithmetic crossover and non-uniform mutation (50 training patterns for each digit).

Number of membership functions	*9*	*10*	*11*	*12*	*13*
Fitness value	0.9626	0.9764	0.9895	0.9890	0.9932

Table VII. Fitness values of the traditional neural-fuzzy network trained by the genetic algorithm with arithmetic crossover and non-uniform mutation (20 testing patterns for each digit).

	Number of membership functions				
Digit #	*9*	*10*	*11*	*12*	*13*
1	9	5	10	1	1
2	2	2	2	1	1
3	3	2	2	1	1
4	1	3	1	1	0
5	0	0	2	1	0
6	2	1	0	0	2
7	13	4	2	4	0
8	14	4	0	0	0
9	1	0	1	1	1
10	2	1	2	0	0

Table VIII. Number of recognition errors for the Cantonese-digit speeches '0'-'9' using the traditional neural-fuzzy network trained by the genetic algorithm with arithmetic crossover and non-uniform mutation (20 testing patterns for each digit).

VII. Conclusion

A variable-parameter neural-fuzzy network has been proposed. The associative memory technique has been successfully implemented in the variable-parameter neural-fuzzy network, making its parameters to change according to the changing input data. The variable-parameter neural-fuzzy network has been applied to recognize Cantonese-command speeches. It is found that the performance in terms of recognition accuracy has been improved from 97% to 98.5% as compared with the traditional neural-fuzzy network that uses the same number of parameters. The search space for the neural-fuzzy network has been widened thanks to the specific structure of the proposed NFN. An improved genetic algorithm has been proposed to train the parameters of the proposed neural-fuzzy network. Six benchmark tests have been introduced to show the merits of the improved GA. The variable-parameter neural-fuzzy network has been implemented in an eBook reader practically.

Acknowledgement

The work described in this paper was fully supported by a grant from the Centre for Multimedia Signal Processing, The Hong Kong Polytechnic University (Project No. A432.)

References

[1] L. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition.* Englewood Cliffs, New Jersey, Prentice Hall, 1993.

[2] T. Lee, W.K. Lo, P.C. Ching, and H. Meng, "Spoken language resources for Cantonese speech processing," *Speech Communication*, vol. 36, issues 3-4, pp. 327-342, March 2002.

[3] B. Kosko, *Neural Networks and Fuzzy System: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, New Jersey, Prentice Hall, 1991.

[4] S. Haykin, *Neural Networks: A Comprehensive Foundation,* 2nd Ed. Prentice Hall, 1999.

[5] G.P. Zhang, "Neural network for classification: a survey," *IEE Trans. Syst., Man, Cybern. C,* vol. 30, no. 4, Nov. 2000.

[6] J.H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[7] D.T. Pham and D. Karaboga, *Intelligent Optimization Techniques, Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer, 2000.

[8] Y. Hanaki, T. Hashiyama, and S. Okuma, "Accelerated evolutionary computation using fitness estimation," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, 1999, vol. 1, pp. 643-648.

[9] K.A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* (Ph.D. Thesis). Ann Arbor, MI: University of Michigan, 1975.

[10] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution Programs,* 2nd extended ed. Springer-Verlag, 1994.

[11] F.H.F. Leung, H.K. Lam, S.H. Ling, and P.K.S. Tam, "Optimal and stable fuzzy controllers for uncertain nonlinear systems based on an improved genetic algorithm," *IEEE Trans. Ind. Electron.*, vol. 51, no. 1, pp. 172-182, Feb. 2004.

[12] Y.S. Zhou and L.Y. Lai "Optimal design for fuzzy controllers by genetic algorithms," *IEEE Trans. Ind. Applications*, vol. 36, no. 1, pp. 93-97, Jan.-Feb. 2000.

[13] C.F. Juang, J.Y. Lin, and C.T. Lin, "Genetic reinforcement learning through symbiotic evolution for fuzzy controller design," *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 30, no. 2, pp. 290–302, April, 2000.

[14] H. Juidette and H. Youlal, "Fuzzy dynamic path planning using genetic algorithms," *Electronics Letters*, vol. 36, no. 4, pp. 374-376, Feb. 2000.

[15] R. Caponetto, L. Fortuna, G. Nunnari, L. Occhipinti, and M.G. Xibilia, "Soft computing for greenhouse climate control," *IEEE Trans. Fuzzy Systems*, vol. 8, no. 6, pp. 753-760, Dec. 2000.

[16] M. Setnes and H. Roubos, "GA-fuzzy modeling and classification: complexity and performance," *IEEE. Trans, Fuzzy Systems*, vol. 8, no. 5, pp. 509-522, Oct. 2000.

[17] K. Belarbi and F. Titel, "Genetic algorithm for the design of a class of fuzzy controllers: an alternative approach," *IEEE Trans. Fuzzy Systems*, vol. 8, no. 4, pp. 398-405, Aug. 2000.

[18] M. Brown and C. Harris, *Neuralfuzzy Adaptive Modeling and Control*. Prentice Hall, 1994.

[19] S. Amin and J.L. Fernandez-Villacanas, "Dynamic local search," in *Proc. 2nd Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1997, pp. 129-132.

[20] X. Wang and M. Elbuluk, "Neural network control of induction machines using genetic algorithm training," in *Conf. Record 31st IAS Annual Meeting*, vol. 3, 1996, pp. 1733-1740.

[21] L. Davis, *Handbook of Genetic Algorithms*. NY: Van Nostrand Reinhold, 1991.

[22] M. Srinivas and L.M. Patnaik, "Genetic algorithms: a survey," *IEEE Computer*, vol. 27, issue 6, pp. 17-26, June 1994.