Parametric neural network with fuzzy inputs configured by genetic algorithm on benchmark

Xuemei Xiong*, Claudio Moraga†
*Department of Electrical Engineering, Nanjing Agricultural University
Dianjiangtailu 40, 210031, Nanjing, China
email:xiong@ls1.cs.uni-dortmund.de
†Department of Computer Science I, University of Dortmund
Otto-Hahn-str. 16, 44221, Dortmund, Germany
email:moraga@ls1.cs.uni-dortmund.de

***Abstract*— This paper proposed a parametric feedforward neural network with fuzzy inputs configured by genetic algorithm (PFNN_FG). Variant sigmoid function was used at different hidden layers. Fuzzy set theory was employed at the input layer to make the processing of linguistic variables possible. The parameters of variant sigmoid function and the fuzzy parameters were configured by genetic algorithm. A comparative analysis between PFNN_FG and other models on benchmark problems shows that PFNN_FG is comparable with the other methods in terms of accuracy of the obtained results, but it is much faster.**

I. Introduction

The complexity of plant disease problems requires sophisticated methods and tools for building intelligent systems. Such systems should be able to (1) *learn fast* from a mound of data; (2) deal with *knowledge* (e.g. rules) as well as with data; (3) have *optimal* structure. Several methods and systems have been developed so far that meet some of the criteria above. These are methods and systems for: fast learning [1]; optimal structure [2]; dealing with knowledge [3] [4]. The parametric feedforward neural network with fuzzy inputs configured by genetic algorithm (PFNN_FG) presented in this paper has elements from all the works above, but differs from the above in many aspects.

II. Parametric FNN

In order to overcome the problem of attenuation and dilution of error signals which contribute to the slow learning of Backpropagation(Bp), one way is to enlarge the error signals in different hidden layers respectively. It is simple to change the activation functions of different hidden layers and output layer in order to enlarge the error signals respectively. Semi-linear functions are bounded differentiable real functions that are defined for all real input values and that have positive derivatives everywhere. They show a sufficient degree of smoothness and are also a suitable extension of the soft limiting nonlinearities. The logistic, hyperbolic Tangent function (tanh) and the Elliott function from the semi-linear family are chosen for the hidden layers and output layer to test the behavior. Fig. 1 shows that the ranges and slopes of the derivatives by these functions are enlarged correspondingly.

The error signals in different layers can also be changed by tuning the parameters of the variant sigmoid function [5] :

$$f(net_j(t), b_1, b_2, b_3) = \frac{b_1}{1 + \exp\left(-b_2\ net_j(t)\right)} - b_3 \quad (1)$$

where
$net_j(t)$: net input in unit j in step t;
b_1: parameter for the dynamic range of the function;
b_2 : parameter for the slope of the function;
b_3 : parameter for the symmetry (or bias) of the function.

By regulating these three parameters in different layers the problem of attenuation and dilution of the error signal can be reduced. Using different parameter configurations of variant sigmoid functions in different layers, a parametric network (PFNN) [6] is therefore formed which is the essential ingredient of the proposed PFNN_FG described in the following sections.

For example: apply variant sigmoid function to Soybean problem (see section V), the structure is $81-16^{N-1}-8^N-19$ (4-layer neural network with 81 inputs and 19 outputs, 2 intermediate layers each with 16 and 8 neurons). The activation functions for hidden and output layers are given as:

$$f(h^{N-1}) = \frac{2}{1 + \exp\left(-6net_j(t)\right)} - 1 \quad (2)$$

$$f(h^N) = \frac{1}{1 + \exp\left(-1.6net_j(t)\right)} \quad (3)$$

$$f(o) = \frac{1}{1 + \exp\left(-0.6net_j(t)\right)} \quad (4)$$

The parameters of variant sigmoid function (for the $(N-1)$th layer : $b_1 = 2, b_2 = 6, b_3 = 1$; for the (N)th layer : $b_1 = 1, b_2 = 1.6, b_3 = 0$; for the output layer: $b_1 = 1, b_2 = 0.6, b_3 = 0$) was optimized by employing a genetic algorithm. The fine tuning of the parameters of the $(N-1)$th hidden layer results in a hyperbolic Tangent function. Fig. 2 shows that the ranges and slopes of the derivatives by these functions are much more enlarged correspondingly.

III. Fuzzy set in PFNN_FG

From our point of view, 'neuro-fuzzy' means the employment of learning strategies derived from the domain of fuzzy

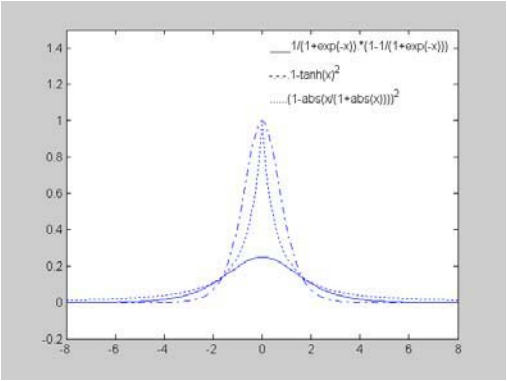

Fig. 1. Derivatives of the logistic, tanh and Elliott functions

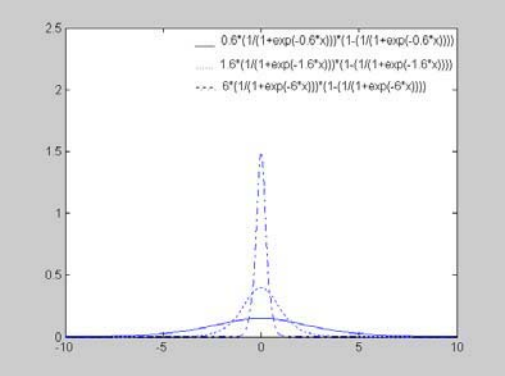

Fig. 2. Derivatives of the variant sigmoid functions

theory to support the development of neural networks. Fuzzy set-theoretic concepts are incorporated at the input layer, enabling to handle both linguistic and numeric inputs.

A. Input vector

Consider a multi-layered PFNN (Fig. 3) with an $(n + m)$-dimensional pattern (n-dimensional linguistic inputs and m-dimensional numeric inputs) $I_i = [I_{i1}, \cdots, I_{in}, I_{i(n+1)}, \cdots, I_{i(n+m)}]$ is represented as a $(3n + m)$-dimensional vector [4].

$$\begin{aligned} I_i &= [\mu_{L(I_{i1})}, \cdots, \mu_{H(I_{i(3n)})}, X_{i(3n+1)}, \cdots, X_{i(3n+m)}] \\ &= [X_1^{(0)}, \cdots, X_{3n}^{(0)}, X_{(3n+1)}^{(0)}, \cdots, X_{(3n+m)}^{(0)}] \end{aligned} \quad (5)$$

where the μ values indicate the membership functions of the corresponding linguistic π-sets low (L), medium (M), and high (H) along each feature axis and the $X^{(0)}$ values refer to the activations of the $(3n + m)$ neurons in the fuzzy input layer. When the input feature I_j is numeric,

$$\pi(I_j, c, \lambda) = \begin{cases} 2(1 - \frac{\|I_j - c\|}{\lambda})^2, & \frac{\lambda}{2} \le \|I_j - c\| \le \lambda \\ 1 - 2(\frac{\|I_j - c\|}{\lambda})^2, & 0 \le \|I_j - c\| \le \frac{\lambda}{2} \\ 0, & otherwise \end{cases} \quad (6)$$

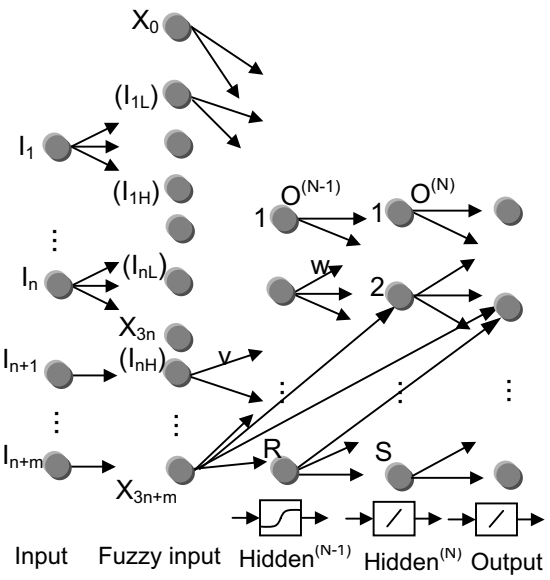

Fig. 3. Neural network with fuzzy inputs

where λ ($\lambda > 0$) is the radius of the π-function with c as the central point. When the input feature I_j is linguistic, its membership values for the π-sets: low(L_j) , medium(M_j), and high(H_j) are quantified as:

$$L_j = \{0.95, \pi(I_j(0.95, c_{jm}, \lambda_{jm}), \pi(I_j(0.95), c_{jh}, \lambda_{jh})\} \quad (7)$$

$$M_j = \{\pi(I_j(0.95), c_{jl}, \lambda_{jl}), 0.95, \pi(I_j(0.95), c_{jh}, \lambda_{jh})\} \quad (8)$$

$$H_j = \{\pi(I_j(0.95), c_{jl}, \lambda_{jl}), \pi(I_j(0.95), c_{jm}, \lambda_{jm}), 0.95\} \quad (9)$$

where $c_{jl}, \lambda_{jl}, c_{jm}, \lambda_{jm}, c_{jh}, \lambda_{jh}$ indicate the centers and radii of the three linguistic properties along the jth axis. In (7) 0.95 means a membership of 0.95 for L, and $I_j(0.95)$ denotes the corresponding feature value I_j at which the linguistic property L attains membership values of 0.95. $\pi(I_j(0.95), c_{jm}, \lambda_{jm})$ refers to the membership attained by the π-set M for that I_j which caused π-set L to have a membership value of 0.95. Similarly, $\pi(I_j(0.95), c_{jh}, \lambda_{jh})$ refers to the membership attained by the π-set H for that I_j which caused π-set L to have a membership value of 0.95. Let a sample pattern has for example 3 linguistic componets I_1, I_2, I_3. This is mapped to the nine-dimension according to the equations (7)(8)(9).

Note that we could have used more linguistic variables at the expense of increased computational complexity.

B. Rule extraction

The rule extraction method derived from compensating system [3] can extract rules from any trained feedforward neural networks. Consider a PFNN with a fuzzy input layer and two hidden layers ($(N-1)$ and (N)) such that all nodes have a variant sigmoid function f (an S-activation function [3]). The network has n numeric inputs and m linguistic inputs and an output layer with linear function.

The output of the jth hidden node ($j = 1, \cdots, R$) of the $(N-1)$th hidden layer is given by:

$$O_j^{(N-1)} = f(v_{j0}X_o + \cdots + v_{j(3n+m)}X_{(3n+m)}) \quad (10)$$

Bias θ_j is included as a weight associated to an extra input I_0 with the constant value 1. The output of the kth hidden unit ($k = 1, \cdots, S$) at the (N)th hidden layer is:

$$O_k^{(N)} = f(w_{k1}O_1^{(N-1)} + \cdots + w_{kR}O_R^{(N-1)}) \tag{11}$$

According to definition in [3], the output of the node may be expressed as:

$$O_k^N = f(w_{k1}O_1^{(N-1)}) \otimes f(w_{k2}O_2^{(N-1)}) \otimes \cdots \otimes f(w_{kR}O_R^{(N-1)}) \tag{12}$$

where $\otimes$ denotes the symmetric sum [7]. The weights may be associated as a new parameter of the S-activation functions, modifying their slope. Accordingly,

$$O_k^N = f^{(w_{k1})}(O_1^{(N-1)}) \otimes f^{(w_{k2})}(O_2^{(N-1)}) \otimes \cdots \otimes f^{(w_{kR})}(O_R^{(N-1)}) \tag{13}$$

Similary,

$$O_j^{N-1} = f^{(v_{j0})}(X_0) \otimes f^{(v_{j1})}(X_1) \otimes \cdots \otimes f^{(v_{j(3n+m)})}(X_{(3n+m)}) \tag{14}$$

So

$$\begin{aligned} O_k^N &= \otimes_{j=1}^{R}(f^{(w_{kj})})(O_j^{(N-1)}) \\ &= \otimes_{j=1}^{R}(f^{(w_{kj})})(\otimes_{i=0}^{(3n+m)} f^{(v_{ji})}(X_i)) \end{aligned} \tag{15}$$

Since $f^{(w_{ji})}(X_i)$ may be given the interpretation of a right (left) open fuzzy set on X_i (depending on whether v_{ji} is positive (negative)), then O_j represents the generalized symmetric sum of the degree of satisfaction of the corresponding fuzzy sets.

This equation may be given the following interpretation: "IF I_0 is bounded by T_0 and $X_1(i.e.I_{1L})$ is bounded by T_1 $\cdots$ and $X_{(3n)}(i.e.I_{nH})$ $\cdots$ $X_{3n+m}(i.e.I_{n+m})$ is bounded by $T_{(3n+m)}$ THEN O is given by the (generalized) symmetric sum of their degrees of satisfaction"

where "bounded" denotes "at least" if the corresponding weight v is positive or "at most" if v is negative, and T_i is the linguistic label associated to the fuzzy set represented by $f^{(v_{ji})}$. The value of the argument with membership 0.9 will be used as reference to apply the conditions "at least" or "at most", respectively. The output rule is a linear combination of the rules generated at the hidden nodes.

IV. Genetic algorithm in PFNN_FG

It would be a time consuming and uncertain task to find out good initial parameter configurations by hand. The configuration of parameters can be solved by genetic algorithm (GA) since it can be converted to an optimization problem. The application of a GA can be divided into three parts of work: optimizing of the initial weigh, fuzzy parameters and the parameters of variant sigmoid functions.

A. Chromosomal representation

During learning a feedforward neural network (FNN) searches for the set of connection weights that corresponds to some local minima. In other words, it searches for that set of weights that minimizes the difference between the target vector and the actual output (obtained by the FNN). Note that there may be a large number of such minimum values corresponding to various good solutions. If we initially set these weights so as to be near one such good solution, the searching space may be reduced and learning thereby becomes faster. Therefore sufficient reduction in training time can be obtained, because a very small number of training cycles are required when the network is already very close to the solution.

GAs represent the parameters for the given problem by the chromosome which may contain one or more substring(s). Each chromosome, therefore, contains a possible solution to the problem. The problem variables consist of the weight values, the input fuzzification parameters and the parameters of variant sigmoid functions.

The links are initialized to small random values, which forms part of the initial population of the GA. The algorithm then searches through the space of network topologies. Each of the weights is encoded into a binary word of 16 bit length, where $[000\cdots0]$ decodes to -128 and $[111\cdots1]$ decodes to 128. An additional bit is assigned to each weight to indicate the presence or absence of the link. If the GA results in the absence of a link from a hidden unit, this unit will be eliminated. So the number of hidden units is also be optimized by GA. This results in an optimal structure of NN which benefits the rule extraction of NN because the more complicated the topology of a feedforward neural network though, the heavier the burden on the interpretability of the possibly chained extracted rules.

The fuzzification parameters tuned are the centers(c) and radius(λ) for each of the linguistic attributes low, medium and high of each feature. These are also coded as 16 bit strings in the range[0, 2].

Another part of the phenotype of each solution (individual) consists of the three parameters of the variant sigmoild functions for the hidden layer(s) respectively. Every parameter is represented by 10 bits. For parameter b_1 and b_2 the input range is $[0.01, 5]$, for b_3 it is $[-8, 8]$.

The chromosome is obtained by concatenating all the above strings. An initial population size (P_s) of 64 was considered.

B. Implementation

1) Crossover: It is obvious that due to the large string length, single point crossover would have little effectiveness. Multiple point crossover is adopted, with the distance between two crossover points being a random variable between 8 and 24 bits. This is done to ensure a high probability for only one crossover point occurring within a word encoding a single weight. The crossover probability (P_c) is fixed at 0.7.

2) Mutation: The search string being very large, the influence of mutation is more on the search compared to crossover. The maximum value of P_m is chosen to be 0.4 and the minimum value as 0.01. The mutation probabilities vary along the encoded string, the bits corresponding to initial links and sigmoid function parameters being assigned a probability P_m (i.e., the value of P_m at that iteration) and fuzzy parameters assigned a probability $P_m/10$. This is done to ensure least alterations in the structure of the individual modules already evolved.

3) Choice of fitness function: An objective function is chosen as

$$F = \frac{No.\,of\,Correctly\,Classified\,Sample}{Total\,No.\,of\,Samples} \tag{16}$$

Note that we optimize the network connectivity, weights, input fuzzification parameters and variant sigmoid function parameters simultaneously.

Selection is done by the roulette wheel method. Elitism is incorporated in the selection process to prevent oscillation of the fitness function with generation. The fitness of the best individual of a new generation is compared with that of the current generation. If the latter has a higher value the corresponding individual replaces a randomly selected individual in the new population.

The incorporation of neural network training into GAs can be summarized as follows. Initial configurations were set in the appropriate units of all layers. Then the neural network is trained with a given problem data. In every cycle all patterns of the problem data are trained once and the deviations (network output minus desired output) are accumulated to a mean squared error giving the whole learning error of the cycle. The GA so tends towards individuals resulting in more correctly classified examples in training set for a problem.

- Step1: Set the values of the parameters for GA: maximum number of generation (G_n), population size (P_s), crossover rate (P_c), and the mutation rate (P_m).
- Step2: Construct initial population $P(0)$.
- Step3: Decode the chromosome of each individual in the population and give a fitness value to each individual in the population by (16).
- Step4: Evolve the new population $P(i+1)$ by reproduction, crossover and mutation.
- Step5: Increase the generation number by replacing old generation with new generation $P(i) = P(i+1)$. During the replacement, preserve an individual which has the maximum fitness value by the elitist reproduction.
- Step6: Repeat Steps3-5 until the satisfactory population appears or the generation number is reached to G_n, or the fitness function is not increased for the prescribed number of generations.

V. Results of benchmark datasets

A. Discription of datasets

The research reported here is based on data from the Proben1 benchmark set [8]. These benchmarks are the Diabetes, Cancer, Glass, Soybean and Thyroid problems. The data sets representing all these problems are real-world data. In this work, all data sets representing the problems are divided into three sets: the training set, the validation set and the test set. The numbers of examples in the training set, validation set and test set are given as in [8] in order to make comparison with this work possible.

1) The Diabetes problem: Diabetes data are diagnose diabetes of Pima Indians based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.), try to decide whether a Pima Indian individual is diabetes positive or not. It has 768 examples with 8 inputs and 2 outputs. Due to a relatively small data set and high noise level, the diabetes problem is one of the most challenging problems in machine learning.

2) The Cancer problem: Breast cancer dataset classifying a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination has input attributes for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei. It has 699 examples with 9 inputs and 2 outputs.

3) The Glass data: 214 Glass data with 9 inputs and 6 outputs classify glass types. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, container, tableware, or head lamps.

4) The Thyroid data: Since all the above-mentioned problems were small classification problems, we chose Thyroid data with 7200 examples and a 19-class Soybean problem as two large problems for comparative purpose. Thyroid data try to decide whether the patient's thyroid has overfunction, normal function or underfunction based on patient query data and patient examination data. Each of examples consisted of 21 inputs and 3 outputs.

5) The Soybean problem: The Soybean is well known in the machine learning and neural networks communities. It is a classification problem with 683 examples. It has 35 inputs and 19 outputs. The discrimination is done based on a description of the bean (e.g. whether its size and color are normal) and the plant (e.g. the size of spots on the leafs, whether these spots have a halo, whether plant growth is normal, whether roots are rotted) plus information about the history of the plant's life (e.g. whether changes in crop occurred in the last year or last two years, whether seeds were treated, how the environment temperature is). This is the problem with the highest number of classes in the benchmark set. The 26th input attribute "int-discolor" is originally encoded as 4-bit binary string. Since "no discolor", "black", "brown" are fuzzy terms, here they are mapped to 12 fuzzy inputs according to equations (7), (8) and (9).

B. Results

A comparative study on the performance of PFNN_FG versus FNN, Radial Basis Function (RBF) network, network using (Pruned) Cascade Correlation (CC/PCC) algorithms is

TABLE I

RESULTS OF CANCER DATASET

Algo.	Hidd. units	Param.[a]	error (%)	accuracy (%)	epochs
PCC	8	93	2.87	96.55	
PFNN	4+2	S[b]:100	1.15	95.98	110
		NS[c]: 56	2.30	97.70	140
RBF_G[d]	10	122	1.72	97.13	45
PFNN_FG	4	45[e]	2.30	97.70	55

[a]links+bias
[b]shortcut connections
[c]without shortcut connections
[d]with Gaussian function
[e]instead of 50(NS) or 68(S)

TABLE II

RESULTS OF GLASS DATASET

Algo.	Hidd. units	Param.	error (%)	accur. (%)	epochs
PCC	19	237	15.09	56.60	
PFNN	8	S:188	16.98	49.06	90
		NS:134	16.98	49.06	120
RBF_S[a]	10	166	9.43	37.74	100
PFNN_FG	8	122[b]	15.01	60.60	120

[a]with thin plate spines function
[b]instead of 134(NS) or 188(S)

TABLE III

RESULTS OF DIABETS DATASET

Algo.	Hidd. units	Param.	error (%)	accur. (%)	epochs
CC	33	381	28.65	68.23	
PFNN	2+2	S:66	9.43	71.70	90
RBF_S	90	992	19.79	71.88	150
PFNN_FG	5	61[a]	10.50	78.00	120

[a]instead of 57(NS) or 73(S)

TABLE IV

RESULTS OF SOYBEAN DATASET

Algo.	Hidd. units	Param.	error (%)	accur. (%)	epochs
CC	2	1781	2.94	80.59	
	2+5	2297	4.71	76.47	
PFNN	16+8	S:4153	5.29	85.88	780
		NS:1635	5.29	88.24	700
	2	S:1781	4.17	85.29	40
		NS:223	12.94	43.53	5000
RBF_G	20	2059	3.53	79.41	780
RBG_M[a]	20	2059	4.12	80.00	780
PFNN_FG	12	1605[b]	5.23	88.21	700

[a]with multi quadratic function
[b]instead of 1243(NS) or 2801(S)

TABLE V

RESULTS OF THYROID DATASET

Algo.	Hidd. units	Param.	error (%)	accur. (%)	epochs
PCC	26+3	428	1.22	97.0	
PFNN	16+8	S:794	0.94	97.17	100
RBF_S	10	253	7.28	92.72	
PFNN_FG	9	163[a]	1.15	97.17	120

[a]instead of 228(NS) or 291(S)

done, and illustrated in tables I through V. In these tables the colomn labeled "error" gives the percentage of wrong classified samples, meanwhile the colomn labeled "accuracy" gives the percentage of correct classified samples. The difference to 100% corresponds to rejected samples. Classification is given with Analyzing function of SNNS [9] with the following criterion: a pattern is classified correctly if: the output of exactly one output unit is $\geq h$; the "teaching output" of this unit is the maximum teaching output(> 0) of the pattern; and the output of all other output units is $\leq l$. A pattern is classified incorrectly if: the output of exactly one output unit is $\geq h$; the "teaching output" of this unit is NOT the maximum "teaching output" of the pattern or there is no "teaching output" > 0; and the output of all other units is $\leq l$. A pattern is unclassified in all other cases. Default values are: $l = 0.4$ $h = 0.6$.

Networks optimized by PFNN_FG performance significantly better than other algorithms (networks). Further, that superior performance is achieved with smaller nets. Redundant topology of the network is eliminated, without any loss of classification performance.

As to Glass dataset in table II, two of this dataset have hardly any correlation with the result. Since the number of examples is quite small, the problem is sensitive to algorithms.

The problem of Thyroid requires a very good classification, because 92% of the patterns belong to one class. So a useful network must classify much better than 92%. A further challenge is the large number of training patterns. CC algorithm is superior to training algorithms using fixed topologies. Nevertheless the network performance could not be further improved. The CC algorithm was able to train networks with 100% recognition rate with respect to the training set, if sufficient hidden units are used. Nevertheless generalization ability with respect to the testing set could not be improved [10]. The network obtained by PFNN_FG had the same performance as PFNN, but only 20% of the weights.

The weights distribution of trained networks with PFNN_FG shows that these networks have most of their weights values zero while majority of their non-zero weights have a high value. These networks with strong links are suitable for rule extraction.

In the experiments made above, RBF networks and FNNs play very similar roles in that they both provide techniques for approximating. The success of both networks is highly dependent on training sets.

VI. Conclusion

New degrees of freedom of feedforward neural networks by using variant sigmoid activation function leads to much faster learning capabilities than with conventional neural networks. A suitable initial setting of the parameters can be found by the application of genetic algorithms. The fuzzy methods are used in the input layer of network to enable user to input linguistic variables. Rules are extracted. Number of rules is the number of hidden units at the Nth hidden layer. After training the PFNN_FG with the same training patterns and testing the network with the same testing patterns, the results indicate that the hybrid approach can generalize better than other networks in all five test cases.

References

[1] M. Riedmiller, H. Braun: *A direct adaptive method for faster backpropagation learning: the RPROP algorithm*. Proc IEEE Int Neural Networks, San Francisco, CA **1**(1993), 586-591.

[2] S. E. Fahlman, C. Lebiere: *The cascade-correlation learning architecture*. Technical Reprot CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, August 1991.

[3] C. Moraga: *Neuro-fuzzy modeling of compensating systems*. Quo vadis Copputational Intelligence? (P. Sincak, J. Vascak eds.), Springer Verlag, Heidelberg, 2000.

[4] M. S. Mitra, S. K. Pal: *Rough fuzzy MLP: knowledge encoding and classification*. IEEE Trans. on Neural Networks, **9** (1998),(6), 1203-1216

[5] J. Han, C. Moraga: *The influence of the Sigmoid function parameters on the speed of Backpropagation learning*. in Mira, Sandoval (Eds.): From natural to artificial neural computation. Springer Verlag, (1995)195-201.

[6] C. Moraga: *Properties of Parametric Feedforward Networks*. Proceedings XXIII Conferencia Latinoamericana de Informática (1997), 861-870, Valparaíso, Chile.

[7] W. Silvert: *Symmetric summation: a class of operations on fuzzy sets*. IEEE Trans. on Systems, Man and Cybernetics, **9** (1979), 659-667

[8] L. Prechelt: *PROBEN-1 - A set of benchmarks and benchmarking rules for neural network training algorithms*. Technical Report 21/24, Faculty of Computer Science, University of Karlsruhe, Germany. ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.gz

[9] Zell A, Mamier G, Vogt M et al. *SNNS Stuttgart Neural Network Simulator User Manual, Version 4.2*. Unversity of Stuttgart, 1992.

[10] W.Schiffmann, M. Joost, R. Werner: *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons*. Technical Report, Institute of Physics, University of Koblenz, Germany. http://www.cs.bham.ac.uk/ pxt/NC/schiffmann.bp.pdf