Rule-Based Fuzzy Polynomial Neural Networks in Modeling Software Process Data

Byoung-Jun Park, Dong-Yoon Lee and Sung-Kwun Oh

Abstract: Experimental software datasets describing software projects in terms of their complexity and development time have been the subject of intensive modeling. A number of various modeling methodologies and modeling designs have been proposed including such approaches as neural networks, fuzzy, and fuzzy neural network models. In this study, we introduce the concept of the Rule-based fuzzy polynomial neural networks (RFPNN) as a hybrid modeling architecture and discuss its comprehensive design methodology. The development of the RFPNN dwells on the technologies of Computational Intelligence (CI), namely fuzzy sets, neural networks, and genetic algorithms. The architecture of the RFPNN results from a synergistic usage of RFNN and PNN. RFNN contribute to the formation of the premise part of the rule-based structure of the RFPNN. The consequence part of the RFPNN is designed using PNN. We discuss two kinds of RFPNN architectures and propose a comprehensive learning algorithm. In particular, it is shown that this network exhibits a dynamic structure. The experimental results include well-known software data such as the NASA dataset concerning software cost estimation and the one describing software modules of the Medical Imaging System (MIS).

Keywords: Rule-based fuzzy polynomial neural networks, rule-based fuzzy neural networks, polynomial neural networks, computational intelligence, genetic algorithms, design methodology, software data.

1. INTRODUCTION

Empirical studies in software engineering employ experimental data to gain insight into the software development process and assess its quality. Data concerning software products and software processes are crucial to their better understanding and, in the sequel, developing effective ways of producing high quality software. However, data have no meaning per se and their semantics arise in the context of a conceptual model of the phenomenon under study [1]. The analysis of software data is challenging in many different ways.

- Software engineering models are not governed by any law of *physics* meaning that we cannot rely on and exploit assumptions that are pertinent to well-known models such as linear regression. The assumption of continuity of software processes and quality of software products is also at stake. It is evident that software is not continuous, namely small changes in one variable may cause significant changes in the performance of the system or cause the system to totally collapse;
- The amount of experimental data in software engineering is usually quite limited. This calls for a thorough analysis and prudent validation of the models especially when it comes to their approximation capabilities;
- The origin of software data comes with their inherent variability. This variability calls for models with a high level of flexibility. Ideally, we would anticipate models containing solid mechanisms of adaptivity (learning).

Yet, the mechanism describing the software development process is either insufficiently understood or becomes too complicated to allow an exact model to be postulated from the theory. Bearing these in mind, we are vitally interested in the development of adaptive and highly nonlinear models that are capable of handling efficacies of software processes. This calls for models that are anchored in the framework of fuzzy sets and neural networks. In particular, we are concerned with the hybridization of these two technologies giving rise to so-called fuzzy neural network systems.

Moreover, efficient modeling techniques should allow for the selection of pertinent variables and the

Manuscript received May 14, 2002; accepted March 15, 2003. This work was supported by the Korea Science and Engineering Foundation (KOSEF) under Grant R05-2000-000-00284-0.

Byoung-Jun Park and Sung-Kwun Oh is with the Department of Electrical Electronic and Information Engineering, Wonkwang University, Korea (e-mail: {lcap, ohsk} @ wonkwang.ac.kr).

Dong-Yoon Lee is with the Department of Information Engineering, Joongbu University, Korea (e-mail: dylee@mail.joongbu.ac.kr).

formation of highly representative datasets. The models should be able to take advantage of the existing domain knowledge (such as a prior experience of human observers or operators) and augment (calibrate) it by available numeric data to form a coherent data-knowledge modeling entity. Most lately the omnipresent modeling tendency is the one that exploits techniques of Computational Intelligence (CI) by embracing fuzzy modeling, neurocomputing, and genetic optimization [2-6].

In this study, we develop a hybrid modeling architecture, called the Rule-based Fuzzy Polynomial Neural Networks (RFPNN). In a nutshell, RFPNN is composed of two main substructures, namely a rule-based fuzzy neural networks (RFNN) and a polynomial neural networks (PNN). From the standpoint of rule-based architectures, one can regard the RFNN as an implementation of the antecedent part of the rules while the consequents (conclusion parts) are realized with the aid of PNN. The role of the RFNN based on fuzzy inference and the Back-Propagation (BP) algorithm is to interact with input data and granulate the corresponding input spaces (viz. converting the numeric data into representations at the level of fuzzy sets). The role of the PNN is to carry out nonlinear transformation at the level of the fuzzy sets (and corresponding membership grades) formed at the level of RFNN. The PNN, which has a flexible and versatile structure [6], is constructed on the basis of a Group Method of Data Handling (GMDH [8]). In this network, the number of layers and the number of nodes in each layer are not predetermined (unlike in most neural-networks) but can be generated dynamically through some growth process. The number of the input variables used in a partial description (PD) is extended and the order of regression polynomial is also made higher to represent other types of nonlinearities. In particular, the number of nodes in each layer of the PNN architecture can be modified and new modes can be added, if required. To assess the performance of the proposed model, we exploit the well-known NASA dataset [13] and Medical Imaging System (MIS) [16,17] widely used in software engineering.

2. THE ARCHITECTURE AND DEVELOPMENT OF THE RFPNN

In this section, we elaborate on the architecture and design process of the RFPNN. These networks emerge as a synergy between two other general constructs such as RFNN and PNN [6]. First, we briefly discuss these two classes of models by underlining their profound features and afterwards demonstrate how a synergy develops between them.

2.1. Rule-based fuzzy neural networks

The premise part of the RFPNN is constructed with the aid of RFNN. We use fuzzy space partitioning in terms of all variables based on the fuzzy relation based approach. Let us consider an extension of the network by considering the fuzzy partition realized in terms of fuzzy relations. The fuzzy partitions formed for the all variables lead us to the topology visualized in Fig. 1. The RFNN structure shows one possible connection point with the rest of the model for combination with PNN. The location of this point implies the character of the network (both in terms of its flexibility and learning capabilities). Note that the connection point allows for perception of each linguistic manifestation of the original variables (viz. these variables are transformed by fuzzy sets and normalized). Fig. 1 illustrates the architecture of such RFNN in the case of two inputs and a single output, where each input assumes three membership functions. The "circles" denote units of the RFNN and the neuron denoted by $\prod$ indicates a Cartesian product. The outputs of these neurons are taken as a product of all the incoming signals. The "N" identifies a normalization procedure applied to the outputs taken as a product of membership grades. The "$\sum$" neuron is described by a linear sum.

Making use of the language of the rule-based systems, the structure translates into the following collection of rules:

$$\begin{aligned}
&R^1 : \textit{If}\ \ x_1\ \ is\ \ A_{11}\ \ and\ \cdots\ x_k\ \ is\ \ A_{1k}\ \ then\ \ y_1 = w_1\\
&\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots\\
&R^i : \textit{If}\ \ x_1\ \ is\ \ A_{i1}\ \ and\ \cdots\ x_k\ \ is\ \ A_{ik}\ \ then\ \ y_i = w_i\\
&\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots\\
&R^n : \textit{If}\ \ x_1\ \ is\ \ A_{n1}\ \ and\ \cdots\ x_k\ \ is\ \ A_{nk}\ \ then\ \ y_n = w_n
\end{aligned} \tag{1}$$

The fuzzy rules in (1) constitute an overall network of the RFNN as shown in Fig. 1. The output f_i of each node generates a final output $\hat{y}$ of the form.

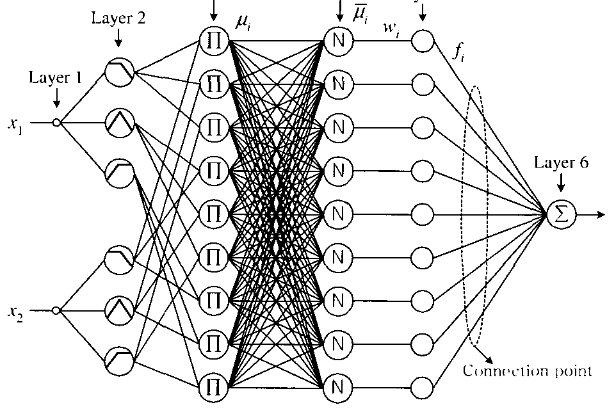


Fig. 1. RFNN structure in the type of fuzzy partition realized by fuzzy relations.

$$\hat{y}=\sum_{i=1}^{n} f_i=\sum_{i=1}^{n}\bar{\mu}_i\cdot w_i=\sum_{i=1}^{n}\frac{\mu_i\cdot w_i}{\sum_{i=1}^{n}\mu_i}. \tag{2}$$

The learning algorithm in RFNN is realized by adjusting connection weights w_i of the neurons and as such it follows a standard Back-Propagation (BP) algorithm. In this study, we use two measures (performance indexes).

• The use of the Euclidean error as a performance measure.

$$E_p=(y_p-\hat{y}_p)^2, \tag{3}$$

where E_p is an error for the *p-th* data, y_p is the *p-th* target output data and $\hat{y}_p$ stands for the *p-th* actual output of the model for this specific data point. For *N* input-output data pairs, an overall (global) performance index comes as a sum of the errors.

$$E=\frac{1}{N}\sum_{p=1}^{N}(y_p-\hat{y}_p)^2. \tag{4}$$

• The mean magnitude of relative error (MMRE) is defined as:

$$E_p=\frac{|y_p-\hat{y}_p|}{y_p}, \tag{5}$$

$$E=\frac{1}{N}\sum_{p=1}^{N}\frac{|y_p-\hat{y}_p|}{y_p}. \tag{6}$$

As far as learning is concerned, the connections change as follows:

$$w(new)=w(old)+\Delta w, \tag{7}$$

where the updated formula follows the gradient descent method

$$\Delta w_i=\eta\cdot\left(-\frac{\partial E_p}{\partial w_i}\right)=-\eta\cdot\frac{\partial E_p}{\partial\hat{y}_p}\cdot\frac{\partial\hat{y}_p}{\partial f_i}\cdot\frac{\partial f_i}{\partial w_i}. \tag{8}$$

Each part of right side in (8) is expressed in the form

i) $y_p \geq \hat{y}_p$

$$-\frac{\partial E_p}{\partial\hat{y}_p}=-\frac{\partial}{\partial\hat{y}_p}\left(\frac{y_p-\hat{y}_p}{y_p}\right)=\frac{1}{y_p},\quad \frac{\partial\hat{y}_p}{\partial f_i}=1,$$
$$\frac{\partial f_i}{\partial w_i}=\bar{\mu}_i. \tag{9}$$

ii) $y_p \langle \hat{y}_p$

$$-\frac{\partial E_p}{\partial\hat{y}_p}=-\frac{\partial}{\partial\hat{y}_p}\left(\frac{\hat{y}_p-y_p}{y_p}\right)=-\frac{1}{y_p},\quad \frac{\partial\hat{y}_p}{\partial f_i}=1,$$
$$\frac{\partial f_i}{\partial w_i}=\bar{\mu}_i. \tag{10}$$

Therefore, Δw_i summarizes as follows:

$$\begin{cases}\Delta w_i=\ \ \eta\cdot\dfrac{\bar{\mu}_i}{y_p} & \mathit{for}\ y_p\geq\hat{y}_p\\ \Delta w_i=-\eta\cdot\dfrac{\bar{\mu}_i}{y_p} & \mathit{for}\ y_p<\hat{y}_p,\end{cases} \tag{11}$$

with η being a positive learning rate.

Quite commonly to accelerate convergence, a momentum term is added to the learning expression. By combining (11) and a momentum term, we have

$$\begin{cases}\Delta w_i=\ \ \eta\cdot\dfrac{\bar{\mu}_i}{y_p}+\alpha(w_i(t)-w_i(t-1)) & \mathit{for}\ y_p\geq\hat{y}_p\\ \Delta w_i=-\eta\cdot\dfrac{\bar{\mu}_i}{y_p}+\alpha(w_i(t)-w_i(t-1)) & \mathit{for}\ y_p\langle\hat{y}_p\end{cases} \tag{12}$$

Here the momentum coefficient, α, is constrained to the unit interval.

2.2. A genetic optimization of RFNN

Genetic algorithms (GAs [9]) has proven to be useful in the optimization of such problems because of their ability to efficiently use historical information to obtain new solutions with enhanced performance and the global nature of search supported there. GAs is also theoretically and empirically proven to support robust searches in complex search spaces. Moreover, they do not become trapped in local minima as opposed to gradient decent techniques being quite susceptible to this shortcoming. GAs is a stochastic search technique based on the principles of evolution, natural selection, and genetic recombination by simulating "survival of the fittest" in a population of potential solutions (individuals) to the problem at hand. GAs are capable of globally exploring a solution space, pursuing potentially fruitful paths while also examining random points to reduce the likelihood of setting for a local optimum. The main features of genetic algorithms concern individuals viewed as strings, population-based optimization (search through the genotype space) and stochastic search mechanism (such as selection and crossover). A fitness function (or fitness, for short) used in genetic optimization is a vehicle to evaluate the performance of a given individual (string). The search of the solution space is completed with the aid of several genetic operators. There are three basic genetic operators used in any GAs - supported searches that are reproduction, crossover, and mutation. Reproduction is a process in which the mating pool for the next generation is chosen. Individual strings are copied into the mating pool according to their fitness function values. Crossover usually proceeds in two steps. First, members from the mating pool are mated at random. Second, each pair of strings undergoes crossover as follows: a position l along the string is selected uniformly at random from the interval $[1, l\text{-}1]$, where l is the length of the string. Two new strings are created by swapping all characters between positions k and l. Mutation is a random alteration of the value of a string position. In a binary coding, mutation means changing a zero to a one or vice versa. Mutation occurs with small probability. Those operators, combined with the proper definition of the fitness function, constitute the main body of the genetic computation.

In order to enhance the learning of the RFPNN and augment its performance, we use genetic algorithms to adjust learning rate, momentum coefficient and the parameters of the membership functions of the antecedents of the rules. Here, GAs use the binary type serial method, roulette-wheel as the selection operator, one-point crossover, and an invert operation in the mutation operator.

2.3 Polynomial neural networks

We use PNN in the consequence structure of the RFPNN. Each neuron of the network realizes a polynomial type of partial description (PD) of the mapping between input and output variables. The structure of the PNN is not fixed in advance but becomes dynamically organized during the growth process. In this sense, PNN is a self-organizing network. The PNN algorithm based on the GMDH method can produce an optimal nonlinear system by selecting significant input variables and forming various types of polynomials. The GMDH is used in selecting the best ones in PDs according to a discrimination criterion. Successive layers of the RFPNN are generated until we reach a structure of the best performance. The input-output relation formed by the PNN algorithm can be described in the following way:

$$y = f(x_1, x_2, \cdots, x_n). \tag{13}$$

The estimated output $\hat{y}$ of the actual output y is

$$\begin{aligned}\hat{y} &= \hat{f}(x_1, x_2, \cdots, x_n) \\ &= c_0 + \sum_{k1} c_{k1}x_{k1} + \sum_{k1k2} c_{k1k2}x_{k1}x_{k2} \\ &+ \sum_{k1k2k3} c_{k1k2k3}x_{k1}x_{k2}x_{k3} + \cdots,\end{aligned} \tag{14}$$

where c_ks are the coefficients of the model to be optimized.

Table 1. Types of regression polynomial.

Order of the polynomial \ No. of Inputs	2	3	4
1 (Type 1)	Bilinear	Trilinear	Tetralinear
2 (Type 2)	Biquad-ratic-1	Triquad-ratic-1	Tetraquad-ratic-1
2 (Type 3)	Biquad-ratic-2	Triquad-ratic-2	Tetraquad-ratic-2

The following types of polynomials are used

- Bilinear $= c_0 + c_1x_1 + c_2x_2$
- Biquadratic-1 = Bilinear+ $c_3x_1^2 + c_4x_2^2 + c_5x_1x_2$
- Biquadratic-2 = Bilinear$+c_3x_1x_2$

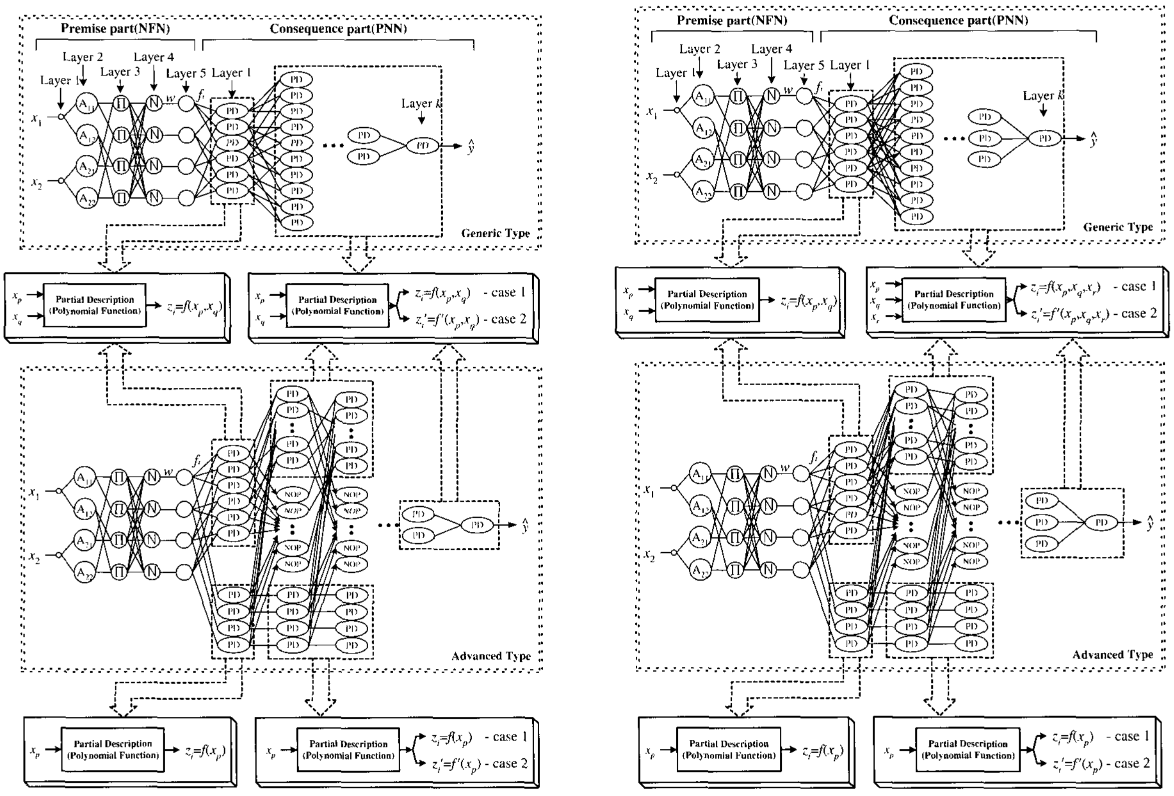
To obtain the estimate $\hat{y}$, we construct a PD for each pair of independent variables occurring in the problem. PDs use regression polynomials, refer to Table 1. Next, we determine the coefficients of the PD through the standard least squares error (LSE) method. The optimal structure of the model is determined stepwise: we form layers of PDs operating on pairs of variables and then select only the best ones. Once the final layer of the structure has been chosen, the node characterized by the best performance is selected as the output node and all other nodes in this layer are removed. Furthermore, all the nodes at the previous layers that do not affect this output node are also eliminated. The removal is carried out by tracing the dataflow back to the previous layers.

Depending on the number of the input variables as shown below, two types of generic PNN architectures are discussed. The structure of PNN is selected on the basis of the number of input variables and the order of PD in each layer. We distinguish between two categories of the PNN structures referring to them as basic and modified PNNs. In the sequel, each category comes with two cases, see also Fig. 2.

The basic and modified PNN architectures are shown in Fig. 2, where z_i'(Case 2) in the 2nd layer or higher denotes that the polynomial order of the PD of each node has a different or modified type in comparison with z_i of the 1st layer.

In the advanced type shown in Fig. 2, the "NOP" node means the Ath node of the current layer that is the same as the node of the corresponding previous layer (NOP denotes no operation). An arrow to the NOP node is used to show that the corresponding identical node moves from the previous layer to the current layer.

2.4. RFPNN topologies: Combination of RFNN and PNN.



(a) Basic RFPNN architecture. (b) Modified RFPNN architecture.

Fig. 2. Configuration of RFPNN architecture insert a line.

The RFPNN is an architecture combined with the RFNN and PNN as shown in Fig. 2. These networks result as a synergy between two other general constructs such as RFNN and PNN. The RFPNN distinguish between two kinds of architectures, namely basic and modified architectures. Moreover, for each architecture of the RFPNN we identify two cases.

(a) Basic architecture – the number of input variables of PDs of PNN is the same in every layer.

(b) Modified architecture – the number of input variables of PDs of PNN differs across the layers.

Case 1: The polynomial order of PDs of PNN is the same in every layer.

Case 2: The polynomial order of PDs in the 2nd layer or higher of PNN has a different or modified type in comparison with one of the PDs existing in the 1st layer.

As mentioned above, the topologies of the RFPNN depend on those of the PNN used for the consequence part of RFPNN.

The RFPNN architecture is combined with the RFNN and PNN as shown in Fig. 2. Let us recall that the RFNN is constructed with the aid of the space partitioning realized by fuzzy relations. We also identify two types as the following:

- Generic type of RFPNN; Combination of the RFNN and the generic PNN; and,
- Advanced type of RFPNN; Combination of the RFNN and the advanced PNN.

2.5. The algorithmic framework of RFPNN

The design procedure for each layer in the premise and the consequence of RFPNN is as follows. We discuss the architecture in detail by considering the functionality of the individual layers (refer to Figs. 1 and 2).

The premise of RFPNN : RFNN

Layer 1: Distributing the signals to the nodes in the next layer as an input layer.

Layer 2: Computing activation degrees of linguistic labels (fuzzy sets; small, large, etc.).

Layer 3: Computing fitness of premise rule: Every node in this layer is a fixed node labeled $\prod$, whose output is the product of all the incoming signals.

$$\mu_i = \mu_A(x_1)\times\mu_B(x_k), \quad A, B = \text{small, large, etc.} \tag{15}$$

A node in this layer represents one fuzzy rule and each node output represents the firing strength of a rule.

Layer 4: Normalization of a degree of activation (firing) of the rule.

$$\bar{\mu}_i = \frac{\mu_i}{\sum_{i=1}^{n}\mu_i}, \tag{16}$$

where n is number of rules.

Layer 5: Multiplying a normalized activation degree of the rule by connection weight.

$$f_i = \bar{\mu}_i \times w_i = \frac{\mu_i \cdot w_i}{\sum_{i=1}^{n} \mu_i}, \tag{17}$$

where f_i is given as the input variable of the PNN, which is the consequence structure of RFPNN.

Layer 6: Computing output of RFNN: The output in the 6th layer of the RFNN is inferred by the center of gravity method.

$$\hat{y} = \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} \bar{\mu}_i \cdot w_i = \sum_{i=1}^{n} \frac{\mu_i \cdot w_i}{\sum_{i=1}^{n} \mu_i} . \tag{18}$$

<u>*The consequence of RFPNN : PNN*</u>

Step 1: Configuration of input variables: $x_1=f_1$, $x_2=f_2, \cdots, x_n=f_i\,(n=i,\ i$: rule number).

Step 2: Formation of a PNN structure: We select input variables and establish the order of PDs.

Step 3: Estimation of the coefficients of the PDs: The vector of coefficients of the PD's(C_j) in each layer is produced by the standard least-squares method and expressed in the form.

$$C_j = (X_j^T\ X_j)^{-1} X_j^T Y, \tag{19}$$

where j denotes node number. This procedure is implemented repeatedly for all nodes of the layer and also for all layers in the consequence part of the RFPNN.

Step 4: Choose PDs in case that the training and testing dataset is taken into consideration: Each PD is constructed and evaluated using the training and testing dataset, respectively. Then we compare the values of the performance index and select PDs using an aggregate performance index with a sound balance between approximation and prediction capabilities. We may use i) the predetermined number W of the PDs (width of the layer) or ii) all of them whose performance index is lower than a certain prespecified value. Method (ii) in particular uses the threshold criterion θ to select the node with the best performance in each layer.

$$\theta = E_{\min} + \delta, \tag{20}$$

where θ is a new value of the criterion, δ is a positive constant (increment) and E_{min} denotes the performance index with the smallest value obtained in each layer.

Step 5: Termination condition: We take into consideration a stopping condition ($E_{min} \geq E_{min*}$) for better performance and the number of iterations (size of the network) predetermined by the designer. Where E_{min} is a minimal identification error at the current layer while E_{min*} denotes a minimal identification error in the previous layer.

Step 6: Determination of new input variables for the next layer: The outputs of the preserved PDs serve as new inputs to the next layer. In other words, we set $x_{1i}=z_{1i}$, $x_{2i}=z_{2i}$, $\cdots$, $x_{Wi}=z_{Wi}$. The consequence part of RFPNN is repeated in steps 3-6.

2.6. Model selection

Our model selection procedure is based on seeking an acceptable compromise between training and generalization errors. The main performance measure that we use in this paper is the mean magnitude of relative error (MMRE) of (4). For evaluation of generalization ability, many estimates have been proposed in the available literature; the most popular ones being the holdout estimate and the k-fold cross-validation estimate [10]. The holdout estimate is obtained by partitioning the dataset into two mutually exclusive subsets called training and testing sets. The error estimate on the testing set is used to assess generalization ability. On the other hand, the k-fold cross-validation estimate is obtained by a sample reuse technique. The dataset is divided into k mutually exclusive subsets of almost equal size, k-1 subsets are used for training, and the kth is used for prediction. This process is repeated k times, each employing a different subset for prediction.

When k is equal to data size, it is called leave-one-out cross-validation (LOOCV) estimate. In this study, we employ the LOOCV estimate of generalization error because of two reasons. First, it possesses sound mathematical properties [11]. Second, it seems to be particularly suited for software engineering applications where the best available data are relatively small sets [12]. Thus, our model selection is based on the analysis of LOOCV estimate of generalization error for RFPNN models.

3. EXPERINENTAL STUDIES

In this section, we illustrate the development of the RFPNN and show its performance for well-known and widely used datasets in software engineering. The first one is the NASA dataset [13]. The second one is the Medical Imaging System (MIS) [16].

3.1. NASA software data

The experimental studies are concerned with a well-known software effort dataset from NASA [13]. The dataset consists of two independent variables, viz., Developed Lines of Code (DL) and Methodology (ME), and one dependent variable, viz., Effort (Y). DL is in KLOC and Y is in man-months. Here, ME is a composite measure of methodologies employed in this NASA software environment. The dataset is shown in Fig. 3.

In the following, we develop software effort estimation models for two collections of independent variables, i.e., DL and (DL, ME).

3.1.1 Results of RFNN modeling

According to RFNN structures, the identification errors of PI and E_PI are shown below in Table 2. Where PI is a performance index for the training dataset and E_PI concerns performance index for the testing dataset using LOOCV. GAs help optimize learning rate, momentum coefficient, and the parameters of the membership functions. GAs was run for 100 generations with a population of 60 individuals. Each string was 10 bits long. The crossover rate was set to 0.6 and probability of mutation was equal to 0.1.

In the RFNN structure of Table 2, two membership functions for each input variable are used. The fuzzy rules with 1 input are conveyed by (21). (22) represents 4 fuzzy rules for RFNN with 2 inputs.

$$\begin{aligned} R^1 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{11}\ \textit{then}\ y_{11} = w_{11} \\ R^2 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{12}\ \textit{then}\ y_{12} = w_{12} \end{aligned} \tag{21}$$

$$\begin{aligned} R^1 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{11}\ \textit{and}\ \text{ME}\ \textit{is}\ A_{21}\ \textit{then}\ y_1 = w_1 \\ R^2 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{11}\ \textit{and}\ \text{ME}\ \textit{is}\ A_{22}\ \textit{then}\ y_2 = w_2 \\ R^3 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{12}\ \textit{and}\ \text{ME}\ \textit{is}\ A_{21}\ \textit{then}\ y_3 = w_3 \\ R^4 &: \textit{If}\ \text{DL}\ \textit{is}\ A_{12}\ \textit{and}\ \text{ME}\ \textit{is}\ A_{22}\ \textit{then}\ y_4 = w_4 \end{aligned} \tag{22}$$

The tuned parameters related to the two membership functions of each variable are showed in Table 3 and Fig. 4.

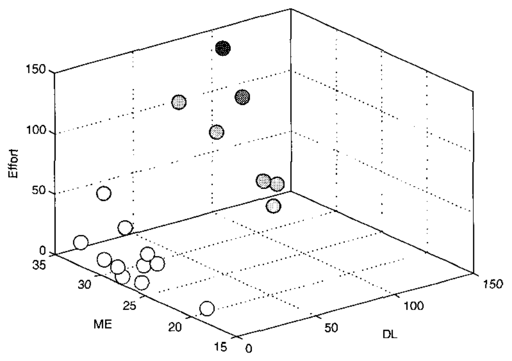


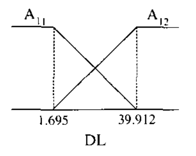
Fig. 3. The dataset of the NASA software project.

Table 2. Performance index of the RFNN: a system with both a single input and dual input.

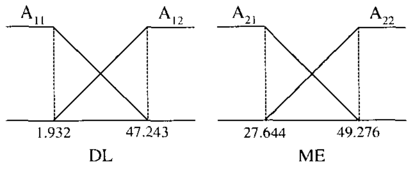
	1 system input(DL)		2 system inputs (DL, ME)	
	PI	E_PI	PI	E_PI
RFNN	0.2870	0.2990	0.2115	0.2415

Table 3. Connection weights of RFNN.

No. of inputs	Connection weights of RFNN	
1 input	w_1	4.2744
	w_2	54.893
2 inputs	w_1	4.9874
	w_2	-5.9834
	w_3	68.757
	w_4	7.9251



(a) 1 input



(b) 2 inputs

Fig. 4. Tuned parameters of membership functions of each input variable.

3.1.2 Results of RFPNN modeling

We now present the details of the RFPNN modeling using the methodology described in the previous section II.

(a) In case of 1 system input (DL)

The goal is to seek a parsimonious model that provides a suitable fit to the DL and Y data of NASA software project data while exhibiting solid generalization capability. Table 4 summarizes the results of the preferred architectures according to advanced type and architecture (basic or modified).

Table 4. Performance index of RFPNN (System input: DL).

RFPNN			Premise part; RFNN		Consequence part; PNN		PI	E_PI
			No. of MFs	Connection	No. of inputs & type	Layer		
Advanced Type	Basic	Case 1	2	1	2 inputs Type 2	1	0.2486	1.2356
						2	0.1801	0.2775
		Case 2	2	1	2 inputs Type 3→2	1	0.2479	0.3166
						2	0.1748	0.1850
	Modified	Case 1	2	1	2→3 inputs Type 2	1	0.2486	1.2356
						2	0.1797	0.4720
		Case 2	2	1	2→3 inputs Type 3→2	1	0.2479	0.3166
						2	0.2285	0.2866
		Case 2	2	2	1→2 inputs Type 1→2	1	0.4719	0.5344
						2	0.2479	0.5344

In light of the values reported there, the basic RFPNN in Case 2 is the preferred architecture of the network in the advanced type and its detailed topology is visualized in Fig. 5. The values of the performance index of the RFNN optimized by GAs are PI=0.2870 and E_PI=0.2990. When considering both PI and E_PI, the minimal value of the performance index, PI=0.1748 and E_PI=0.1850 are obtained by using Type 3 in the 1st layer and Type 2 in the 2nd layer or higher (Type 3→2) with 2 node inputs.

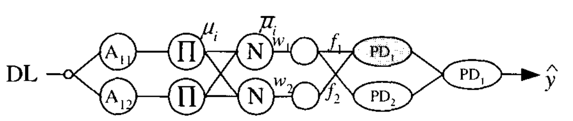
The form of each Type is shown in Table 1. Fig. 5 shows an optimal architecture in the advanced type of the RFPNN that is composed of RFNN and PNN with 2 inputs-Type 3→2 topology. The way in which learning has been realized is shown in Fig. 6 where training errors (performance index) are illustrated. In Fig. 5, $\textcircled{\scriptsize PD}$ is the Ath node of each corresponding layer used for the generation of the output $\hat{y}$, $\textcircled{\scriptsize PD_A}$ is the Ath node of each corresponding layer used for the generation of the output and indicates the optimal node in each layer. The parameters and connection weights of the premise part of RFPNN are shown in Fig. 4 and Table 3 of Section 3.1.1.

(b) In case of 2 system inputs (DL, ME)

Now we develop an effort estimation model based on two independent variables DL and ME. The performance index of RFPNN is shown in Table 5.

Table 5 summarizes the values of the performance index for the RFPNN with 2 system inputs. In Table 5, the modified RFPNN in Case 2 is the preferred architecture of the network in the generic or the advanced type, respectively. An optimal architecture of the RFPNN in the advanced type is visualized in Fig. 7. In Fig. 7, the RFNN part of the network uses two membership functions for each input variable, so the architecture has 4 rules. The values of the performance index of the RFNN optimized by GAs are 0.2115 for the training data and 0.2415 for the testing data. The PNN part of the networks is constructed by using Type 2 with 3 node inputs in the 1st layer and Type 3 with 4 node inputs in the 2nd layer (3→4 inputs and Type 2→3). The final results of the RFPNN topology are PI = 0.0231 and E_PI = 0.0252. Fig. 8 illustrates the performance of the obtained networks.

Table 6 contains a comparative analysis including the previous model introduced in the literature [13]. The comparative analysis reveals that the RFPNN comes with high accuracy and improved prediction (generalization) capabilities.



$\hat{y} = \mathrm{PNN}(f_1,\ f_2)$, where $f_i = \bar{\mu}_i \cdot w_i \quad (i = 1,2)$

Fig. 5. The optimal topology of the advanced and basic RFPNN in Case 2 (2 node inputs and Type 3→2).

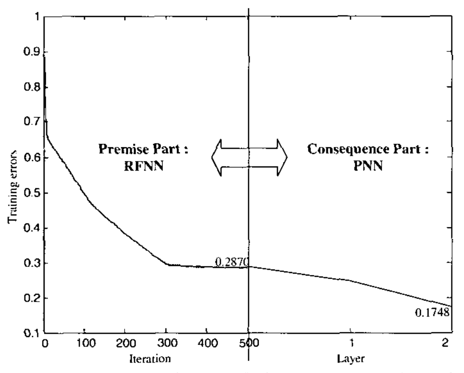
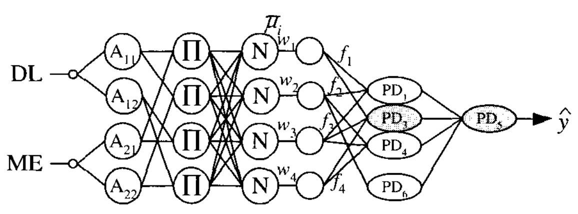


Fig. 6. Learning procedure of the advanced and basic RFPNN in Case 2 (2 node inputs and Type 3→2).

Table 5. Performance index of RFPNN (System inputs: DL, ME).

RFPNN			Premise; RFNN	Consequence; PNN		PI	E_PI
			No. of MFs	No. of inputs & type	Layer		
Generic Type	Basic	Case 1	2×2	2 inputs	1	0.1369	0.1130
				Type 3	2	0.1085	0.0600
		Case 2	2×2	2 inputs	1	0.1369	0.1130
				Type 3→2	2	0.1048	0.0621
	Modified	Case 1	2×2	2→3 inputs	1	0.1369	0.1130
				Type 3	2	0.0893	0.0527
		Case 2	2×2	2→3 inputs	1	0.1369	0.1130
				Type 3→2	2	0.0493	0.0565
dvanced Type	Basic	Case 1	2×2	3 inputs	1	0.0695	0.1890
				Type 2	2	0.0420	0.0417
		Case 2	2×2	3 inputs	1	0.0695	0.1890
				Type 2→3	2	0.0390	0.0429
	Modified	Case 1	2×2	3→4 inputs	1	0.0695	0.1890
				Type 2	2	0.0189	0.0397
		Case 2	2×2	3→4 inputs	1	0.0695	0.1890
				Type 2→3	2	0.0231	0.0252



$\hat{y} = \mathrm{PNN}(f_1,\ f_2,\ f_3,\ f_4)$, where
$f_i = \bar{\mu}_i \cdot w_i \quad (i = 1,2,3,4)$

Fig. 7. The optimal topology of the advanced and modified RFPNN in Case 2 (3→4 node inputs and Type 2→3).

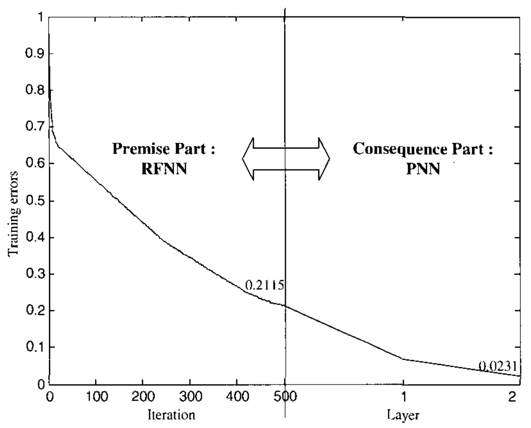


Fig. 8. Learning procedure of the advanced and modified RFPNN in Case 2 (3→4 node inputs, and Type 2→3).

Table 6. Comparison of identification error with previous modeling methods.

Model		System input	Training (PI)	Generalization (E_PI)
Shin and Goel's RBF model [13]		DL	0.1579	0.1870
		DL, ME	0.0870	0.1907
Our model (RFPNN)	Advanced, Basic, Case 2	DL	0.1748	0.1850
	Generic, Modified, Case 2	DL, ME	0.0493	0.0565
	Advanced, Modified, Case 2	DL, ME	0.0231	0.0252

3.2. Medical imaging system

We consider a Medical Imaging System (MIS [16]) subset of 390 modules written in Pascal and FORTRAN for modeling. These modules consist of approximately 40,000 lines of code. To design an optimal model from the MIS, we utilize 11 system input variables such as, LOC, CL, TChar, TComm, MChar, DChar, N, $\hat{N}$, N_F, V(G), and BW. And as an output variable, "Changes" is used. Also, we use 4 system inputs [TComm, MChar, DChar, N] from among the 11 system inputs. Here 4 system inputs are selected from structure identification by the GMDH method.

When applying any modeling technique, an assessment of predictive quality is important. Data splitting is a modeling technique that is often applied to test predictive quality. Applying this technique, one randomly partitions the dataset to produce two datasets. The first 60% dataset is used for fitting the models. The remaining 40% dataset, the testing dataset, provides for quantifying the predictive quality of the fitted models.

Using the MIS dataset, the regression equation is obtained as follows.

$$\begin{aligned}\text{Changes} = &-1.8371+0.14097\cdot \text{LOC}-0.14115\cdot \text{CL}\\ &-0.0026487\cdot \text{TChar}+0.10312\cdot \text{TComm}\\ &+0.0019233\cdot \text{MChar}+0.0086053\cdot \text{DChar} \qquad (23)\\ &-0.024508\cdot \text{N}+0.16458\cdot \hat{\text{N}}-0.20678\cdot \text{N}_F\\ &+0.16353\cdot \text{V(G)}-0.95354\cdot \text{BW}\end{aligned}$$

This simple model comes with the value of PI=40.056 and E_PI=36.322 for 11 system inputs. We will be using this as a reference point when discussing the RFPNN proposed in this paper.

For the RFNN structures, the identification errors of PI and E_PI are shown below in Table 7. GAs help optimize learning rate, momentum coefficient, and the parameters of the membership functions.

In the RFNN structure, two membership functions for each input variable are used. As such, the RFNN structure is represented by 16 fuzzy rules for the 4 system inputs.

Table 7. Performance index of the RFNN.

Model	No. of system inputs	PI	E_PI
Regression	11	40.056	36.322
RFNN	4	50.920	39.215

Table 8. Performance index of RFPNN with 4 system inputs.

RFPNN			Premise part; RFNN	Consequence part; PNN		PI	E_PI
			No. of MFs	No. of inputs & type	Layer		
Generic Type	Basic	Case 1	2x2x2x2	3 inputs Type 2	1 2	40.033 38.972	25.5 20.26
			2x2x2x2	3 inputs Type 3	1 2	50.671 40.558	25.005 21.223
			2x2x2x2	4 inputs Type 2	1 2	39.179 29.360	23.864 19.299
			2x2x2x2	4 inputs Type 3	1 2	38.668 35.205	23.041 16.611

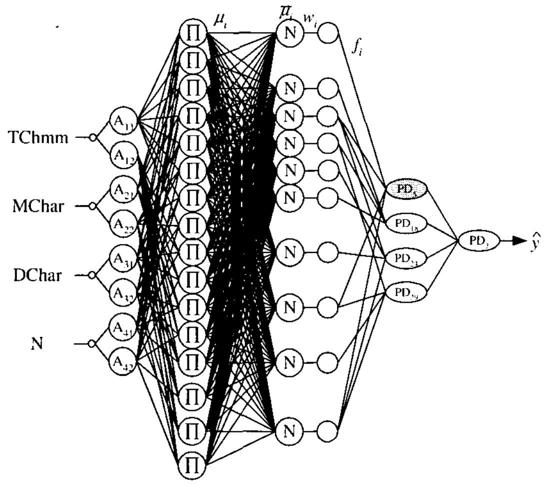


Fig. 9. The topology of the generic and basic RFPNN in Case 1 (4 node inputs and Type 2).

Table 9. Comparison of identification error with previous modeling methods.

Model		No. of inputs	PI	E_PI
Regression model		11	40.056	36.322
Our model (RFPNN)	Generic, Modified, Case 1	4	29.36	19.299

Table 8 summarizes the results of the preferred architectures according to the RFPNN. Here we select the basic RFPNN in Case 1 for the 4 inputs-Type 2, and its detailed topology is visualized in Fig. 9.

Table 9 contains a comparative analysis including the previous model. Regression models are constructed by way of a linear equation. The comparative analysis reveals that the RFPNN comes with high accuracy and improved prediction (generalization) capabilities.

4. CONCLUSIONS

In this study, we have introduced a class of RFPNN regarded as a modeling vehicle for nonlinear and complex systems. We have studied its properties, come up with a detailed design procedure and used these networks to model a well-known NASA dataset and MIS dataset that are experimental data widely used in software engineering. The RFPNN is constructed by combining RFNN with PNN. In this sense, we have constructed a coherent platform in which all components of the CI are fully utilized. The model is inherently dynamic - the use of the PNN, which comes with a highly versatile architecture, is essential to the process of "growing" the network by expanding its depth. A comprehensive design procedure was developed.

REFERENCES

[1] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experimenters*, John Wiley & Sons, 1978.

[2] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning," *Int. J. of Approximate Reasoning*, vol. 5, no. 3, pp. 191-212, 1991.

[3] S. Horikawa, T. Furuhashi, and Y. Uchigawa, "On fuzzy modeling using fuzzy neural networks with the back propagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 801-806, 1992.

[4] N. Imasaki, J. Kiji, and T. Endo, "A fuzzy rule structured neural networks," *Journal of Japan Society for Fuzzy Theory and Systems*, vol. 4, no. 5, pp. 985-995, 1992 (in Japanese).

[5] H. Nomura and Wakami, "A self-tuning method of fuzzy control by descent methods," *4th IFSA World Conference*, pp. 155-159, 1991.

[6] S.-K. Oh, D.-W. Kim, and B.-J. Park, "A study on the optimal design of polynomial neural networks structure," *The Trans. of the Korean Institute of Electrical Engineers*, vol. 49d, no. 3, pp. 145-156, 2000 (in Korean).

[7] T. Yamakawa, "A new effective learning algorithm for a neo fuzzy neuron model," *5th IFSA World Conference*, pp. 1017-1020, 1993.

[8] A. G. Ivahnenko, "The group method of data handling: a rival of method of stochastic approximation," *Soviet Automatic Control*, vol. 13, no. 3, pp. 43-55, 1968.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison - Wesley, 1989.

[10] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford Univ. Press, 1995.

[11] M. Kearns and D. Ron, "Algorithmic stability and sanity-check bounds for leave-one-out cross-validation," *Proc. 10th Ann. Conf. Computational Learning Theory*, pp. 152-162, 1997.

[12] C. F. Kemerer, "An empirical validation of software cost estimation models," *Comm. ACM*, vol. 30, no. 5, pp. 416-429, May 1987.

[13] M. Shin and A. L. Goel, "Empirical data modeling in software engineering using radial basis functions," *IEEE Trans. on Software Engineering*, vol. 26, no. 6, June, 2000.

[14] B.-J. Park, S.-K. Oh, and W. Pedrycz, "The hybrid multi-layer inference architecture and algorithm of FPNN based on FNN and PNN," *Joint 9th IFSA World Congress*, pp. 1361-1366, 2001.

[15] S.-K. Oh, T.-C. Ahn , and W. Pedrycz, "A study on the self-organizing polynomial neural net-

works," *Joint 9th IFSA World Congress*, pp. 1690-1695, 2001.

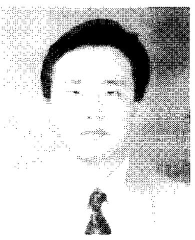
[16] M. R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, pp. 510-514, 1995.

[17] S.-K. Oh, W. Pedrycz, and H.-S. Park, "Self-organizing networks in modeling experimental data in software engineering," *IEE Proc. Computers and Digital Techniques*, vol. 149, Issue 03, pp. 61-78, 2002.

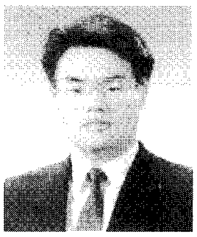
[18] B.-J. Park, W. Pedrycz, and S.-K. Oh "Fuzzy polynomial neural networks: hybrid architectures of fuzzy modeling," *IEEE Trans. on Fuzzy Systems*, vol. 10, no. 5, pp. 607-621, October 2002.

[19] S.-K. Oh, *Fuzzy Model & Control System by C-Programming*, Naeha press, 2002.

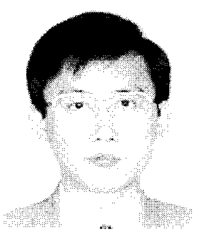
[20] S.-K. Oh, *Computational Intelligence by Programming focused on Fuzzy, Neural Networks, and Genetic Algorithms*, Naeha press, 2002.



Byoung-Jun Park received his B.S. and M.S. degrees in Control and Instrumentation Engineering from Wonkwang University, Korea in 1998 and 2000, respectively. He is currently pursuing a Ph.D. degree from the School of Electrical, Electronic and Information Engineering, Wonkwang University. His research interests encompass fuzzy, neurofuzzy systems, genetic algorithms, Computational Intelligence, hybrid systems, and intelligent control.



Dong-Yoon Lee received his B.S. degree from the Department of Electrical Engineering, Wonkwang University, in 1987. He received his M.S. and Ph.D. degrees from the Department of Electrical Engineering, Yonsei University, in 1990 and 2001, respectively. He was employed as a research engineer at Intellect, J-tek and KIST from 1990 to 2000. From March 2001 to February 2002, he was a B.K. Professor in the School of Electrical, Electronic and Information Engineering, Wonkwang University. He is currently a full-time professor in the school of Information Engineering, Joongbu University.



Sung-Kwun Oh received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from Yonsei University, Seoul, Korea, in 1981, d1983 and 1993 respectively. From 1983-1989, he worked as the Senior Researcher at the R&D Lab. of Lucky-Goldstar Industrial Systems Co., Ltd. He was employed as a Postdoctoral fellow in the Department of Electrical and Computer Engineering at the University of Manitoba, Canada, from 1996 to 1997. He is currently an Associate Professor at the School of Electrical, Electronic and Information Engineering, Wonkwang University, Korea. His research interests include the fields of fuzzy system, fuzzy-neural networks, automation systems, advanced Computational Intelligence, and intelligent control. He is a member of IEEE. He currently serves as an Associate Editor for the Korean Institute of Electrical Engineers (KIEE) and the Institute of Control, Automation & Systems Engineers (ICASE), Korea.