Software Size Estimation Using Fuzzy Backpropagation Network Method

B V A N S S PRABHAKAR RAO[1] & P SEETHA RAMAIAH[2]
[1] Research Scholar, Department of Computer Science and Engineering, JNTU Kakinada, Kakinada, Andhra Pradesh 533 003, India

[2] Professor, Dept. of CS & SE, College of Engineering (A), Andhra University Visakhapatnam, Andhra Pradesh 530 003, India

Abstract

This paper presents the problems of effort estimation for software development projects. Most of the research has focused on the construction of formal software effort estimation models. All most all the models were typically based on the regression analysis or mathematically derived from theories from other domains. This paper focuses on Soft Computing Hybrid Systems in general and Fuzzy Back-propagation Network method in particular. Though Hybrid Systems have a tremendous potential to solve problems, an inappropriate use of the technology can backfire. Hybrid systems are those which employ integrated technologies to effectively solve the problem. Software development efforts estimation is the process of predicting the most realistic use of effort required to develop or maintain software product in an optimized way for the benefit of all the stakeholders. Effort estimates may be used as input to the plans, budgets, investment, pricing processes and bidding rounds. In this context we are suggesting the Fuzzy based Backpropagation Network to solve the problem and to produce the better results.

Keywords**:** *Artificial Neural Network, Backpropagation, Neuro-Fuzzy, Software Estimation, Fuzzy Logic, LOC, Sizing.*

1. Introduction

Software Project planning encompasses five major activities – estimation, scheduling, risk analysis, quality management planning, and change management planning. Estimation includes your attempt to determine how much money, effort, resources, and time it will take to build a specific software-based product. Software project managers using information solicited from project stakeholders and software metrics data collected from past projects [1, 6]. Most of the software researchers and practitioners have been addressing the problems of effort estimation for software development projects since at least the 1960s; Published surveys on estimation practice suggest that expert estimation is the dominant strategy when estimating software development effort. Typically, effort estimates are over-optimistic and there is a strong over-confidence in their accuracy. The mean effort overrun seems to be about 30% and not decreasing over time. Currently the term "effort estimate" is used to denote as different concepts as most likely use of effort (modal value), the effort that corresponds to a probability of 50% of not exceeding (median), the planned effort, the budgeted effort or the effort used to propose a bid or price to the client. This is believed to be unfortunate, because communication problems may occur and because the concepts serve different goals [2] [3].

Estimation begins with a description of the scope of the problem. The problem is then decomposed into a set of smaller problems, and each of these is estimated using historical data and experience as guides. Problem complexity and risk are considered before a final estimate is made.

2. Challenges

The advantage of buying software is automation, better analyzing of business, thus making life easier. Software Development is intangible, Customer can see the benefits of buying only when he uses the software for quiet a period of time and then says ' its worth of it'. We do not quote too less, programmers work for overnight that leads to lose the project or end doing social service, or loss. Do not quote too high that lose the project. So, be fair to ourselves and our customers. Hence, there is need to use of a repeatable, clearly defined and well understood software development process that has to be the most effective method.

- It's easy to estimate what you know.
- It's hard to estimate what you know you don't know.
- It's very hard to estimate things that you don't know you

don't know.

3. Estimation methods

The uncertainty of an effort estimate can be described through a prediction interval (PI). An effort PI is based on a stated certainty level and contains a minimum and a maximum effort value.

3.1 Selection of Estimation approach

The use of artificial neural networks is a cross-disciplinary field that integrates neuroscience, information science and computer sciences. Among many neural network models, the back-propagation (BP) neural network displays a strong learning ability using nonlinear models with a high fault tolerance. The evidence on differences in estimation accuracy of different estimation approaches and models suggest that there is no "best approach". There are many ways of categorizing estimation approaches, see for example [4][5]. The top level categories are the following:

3.2 Expert estimation: The quantification step based on judgmental processes.

3.3 Formal estimation model: The quantification step is based on mechanical processes. Formal estimation models not tailored to a particular organization's own context, may be very inaccurate. Use of own historical data is consequently crucial if one cannot be sure that the estimation model's core relationships (e.g., formula parameters) are based on similar project contexts [6, 7].

3.4 Combination-based estimation: The quantification step is based on a judgmental or mechanical combination of estimates from different sources.

4. Artificial Neural Network

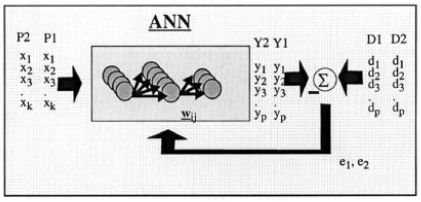
An artificial neural network is a system based on the operation of biological neural networks, in other words, is an emulation of biological neural system. Why would be necessary the implementation of artificial neural networks? Although computing these days is truly advanced, there are certain tasks that a program made for a common microprocessor is unable to perform; even so a software implementation of a neural network can be made with their advantages and disadvantages.

Advantages:

- A neural network can perform tasks that a linear program cannot.
- When an element of the neural network fails, it can continue without any problem by their parallel nature.
- A neural network learns and does not need to be reprogrammed.
- It can be implemented in any application.
- It can be implemented without any problem.

Of course the neural network needs training to operate.

In the world of engineering, neural networks have two main functions: Pattern classifiers and as non linear adaptive filters. As its biological predecessor, an artificial neural network is an adaptive system. By adaptive, it means that each parameter is changed during its operation and it is deployed for solving the problem in matter. This is called the training phase. An artificial neural network is developed with a systematic step-by-step procedure which optimizes a criterion commonly known as the learning rule. The input/output training data is fundamental for these networks as it conveys the information which is necessary to discover the optimal operating point.



The style of neural computation.

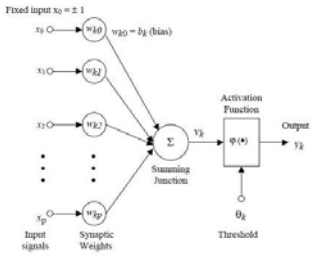
Basically, an artificial neural network is a system. A system is a structure that receives an input, process the data, and provides an output. Commonly, the input consists in a data array which can be anything such as data from an image file, a WAVE sound or any kind of data that can be represented in an array. Once an input is presented to the neural network, and a corresponding desired or target response is set at the output, an error is composed from the difference of the desired response and the real system output. The error information is fed back to the system which makes all adjustments to their parameters in a systematic fashion (commonly known as the learning rule). This process is repeated until the desired output is acceptable.

4.1 The Mathematical Model

Once modeling an artificial functional model from the biological neuron, we must take into account three basic components. First off, the synapses of the biological neuron are modeled as weights. Let's remember that the synapse of the biological neuron is the one which

interconnects the neural network and gives the strength of the connection. For an artificial neuron, the weight is a number, and represents the synapse. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections. The following components of the model represent the actual activity of the neuron cell. All inputs are summed altogether and modified by the weights. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematically, this process is described in the figure



From this model the interval activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

The output of the neuron, yk, would therefore be the outcome of some activation function on the value of vk.

4.2 Activation functions

As mentioned previously, the activation function acts as a squashing function, such that the output of a neuron in a neural network is between certain values (usually 0 and 1, or -1 and 1). In general, there are three types of activation functions, denoted by Φ(.) . First, there is the Threshold Function which takes on a value of 0 if the summed input is less than a certain threshold value (v), and the value 1 if the summed input is greater than or equal to the threshold value.

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Secondly, there is the Piecewise-Linear function. This function again can take on the values of 0 or 1, but can also take on values between that depending on the amplification factor in a certain region of linear operation.

$$\varphi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} > v > \frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases}$$

Thirdly, there is the sigmoid function. This function can range between 0 and 1, but it is also sometimes useful to use the -1 to 1 range. An example of the sigmoid function is the hyperbolic tangent function.

$$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1-\exp(-v)}{1+\exp(-v)}$$

4.3 A framework for distributed representation

An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections. A set of major aspects of a parallel distributed model can be distinguished:

- a set of processing units ('neurons,' 'cells');
- a state of activation yk for every unit, which equivalent to the output of the unit;
- connections between the units. Generally each connection is defined by a weight wjk which determines the effect which the signal of unit j has on unit k;
- a propagation rule, which determines the effective input sk of a unit from its external inputs;
- an activation function Fk, which determines the new level of activation based on the efective input sk(t) and the current activation yk(t) (i.e., the update);
- an external input (aka bias, offset) øk for each unit;
- a method for information gathering (the learning rule);
- an environment within which the system must operate, providing input signals and if necessary error signals.

4.4 Neural Network topologies

In the previous section we discussed the properties of the basic processing unit in an artificial neural network. This section focuses on the pattern of connections between the units and the propagation of data. As for this pattern of connections, the main distinction we can make is between:

- Feed-forward neural networks
- Recurrent neural networks

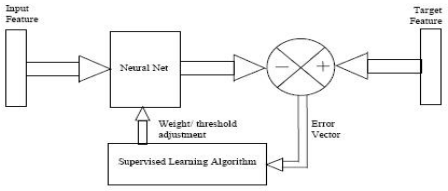
Classical examples of feed-forward neural networks are the Perceptron and Adaline.

4.5 Training of artificial neural networks

A **neural network** has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to **'train' the neural network**

by feeding it teaching patterns and letting it change its weights according to some learning rule.
We can categorise the learning situations in two distinct sorts. These are:

- **Supervised learning** or Associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the neural network (self-supervised).



- **Unsupervised learning** or Self-organisation in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

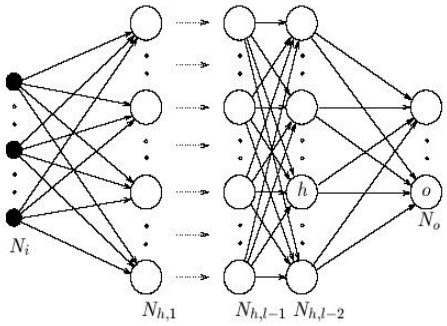
- **Reinforcement Learning** This type of learning may be considered as an intermediate form of the above two types of learning. Here the learning machine does some action on the environment and gets a feedback response from the environment. The learning system grades its action good (rewarding) or bad (punishable) based on the environmental response and accordingly adjusts its parameters. Generally, parameter adjustment is continued until an equilibrium state occurs, following which there will be no more changes in its parameters. The self organizing neural learning may be categorized under this type of learning.

4.6. Multi-layer feed-forward networks

A feed-forward network has a layered structure. Each layer consists of units which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit. There are no connections within a layer. The Ni inputs are fed into the first layer of Nh;1 hidden units. The input units are merely 'fan-out' units; no processing takes place in these units. The activation of a hidden unit is a function Fi of the weighted inputs plus a bias, as given in in eq

$$y_k(t+1) = \mathcal{F}_k(s_k(t)) = \mathcal{F}_k\left(\sum_j w_{jk}(t)\, y_j(t) + \theta_k(t)\right),$$

The output of the hidden units is distributed over the next layer of Nh;2 hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of No output units .



Although backpropagation can be applied to networks with any number of layers, just as for networks with binary units it has been shown (Hornik, Stinchcombe, & White, 1989; Funahashi, 1989; Cybenko, 1989; Hartman, Keeler, & Kowalski, 1990) that only one layer of hidden units success to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear (the universal approximation theorem). In most applications a feed-forward network with a single layer of hidden units is used with a sigmoid activation function for the units.

5. Understanding Backpropagation

The equations derived in the previous section may be mathematically correct, but what do they actually mean? Is there a way of understanding back-propagation other than reciting the necessary equations? The answer is, of course, yes. In fact, the whole back-propagation process is intuitively very clear. What happens in the above equations is the following. When a learning pattern is clamped, the activation values are propagated to the output units, and the actual network output is compared with the desired output values, we usually end up with an error in each of the output units. Let's call this error eo for a particular output unit o. We have to bring eo to zero The simplest method to do this is the greedy method: we strive to change the connections in the neural network in such a way that, next time around, the error eo will be zero for this particular pattern. We know from the delta rule that, in order to reduce an error, we have to adapt its incoming weights according to. $\Delta w_{ho} = (d_o - y_o)y_h$.

That's step one. But it alone is not enough: when we only apply this rule, the weights from input to hidden units are never changed, and we do not have the full representational power of the feed-forward network as promised by the universal approximation theorem. In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for δ for the hidden units. This is

solved by the chain rule which does the following: distribute the error of an output unit o to all the hidden units that is it connected to, weighted by this connection. Differently put, a hidden unit h receives a delta from each output unit o equal to the delta of that output unit weighted with (= multiplied by) the weight of the connection between those units.

5.1 Working with back-propagation

The application of the generalized delta rule thus involves two phases: During the first phase the input x is presented and propagated forward through the network to compute the output values yp o for each output unit. This output is compared with its desired value do, resulting in an error signal δp o for each output unit. The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and appropriate weight changes are calculated.

5.2 Weight adjustments with sigmoid activation function.

- The weight of a connection is adjusted by an amount proportional to the product of an error signal δ, on the unit k receiving the input and the output of the unit j sending this signal along the connection: $\Delta_p w_{jk} = \gamma \delta_k^p y_j^p.$
- If the unit is an output unit, the error signal is given by $\delta_o^p = (d_o^p - y_o^p)\, \mathcal{F}'(s_o^p).$ Take as the activation function F the 'sigmoid' function as defined $y^p = \mathcal{F}(s^p) = \frac{1}{1+e^{-s^p}}.$ In this case the derivative is equal to

$$\begin{aligned}\mathcal{F}'(s^p) &= \frac{\partial}{\partial s^p}\frac{1}{1+e^{-s^p}} \\ &= \frac{1}{(1+e^{-s^p})^2}(-e^{-s^p}) \\ &= \frac{1}{(1+e^{-s^p})}\frac{e^{-s^p}}{(1+e^{-s^p})} \\ &= y^p(1-y^p).\end{aligned}$$

such that the error signal for an output unit can be written as: $\delta_o^p = (d_o^p - y_o^p)\, y_o^p(1-y_o^p).$

- The error signal for a hidden unit is determined recursively in terms of error signals of the units to which it directly connects and the weights of those connections. For the sigmoid activation function:

$$\delta_h^p = \mathcal{F}'(s_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho} = y_h^p(1-y_h^p)\sum_{o=1}^{N_o}\delta_o^p w_{ho}.$$

5.3 Learning rate and momentum

The learning procedure requires that the change in weight is proportional to $\partial E^p/\partial w$. True gradient descent requires that infinitesimal steps are taken. The constant of proportionality is the learning rate . For practical purposes we choose a learning rate that is as large as possible without leading to oscillation. One way to avoid oscillation at large , is to make the change in weight dependent of the past weight change by adding a momentum term:
$\Delta w_{jk}(t+1) = \gamma \delta_k^p y_j^p + \alpha \Delta w_{jk}(t),$ where t indexes the presentation number and F is a constant which determines the effect of the previous weight change.

6. Design of the Fuz zy Backpropagation Network

This study aimed at building and evaluating an Artificial Intelligence System in general and neuro-fuzzy model in particular to estimate software size. Neural network techniques are based on the principle of learning from historical data, whereas fuzzy logic is a method used to make rational decisions in an environment of uncertainty and vagueness. However, fuzzy logic alone does not enable learning from the historical database of software projects. Once the concept of fuzzy logic is incorporated into neural network, the result is a neuro-fuzzy system that combines the advantages of both techniques.

Let $x_j \in [0,1] \subset \Re$, $j = 1, \ldots, n_0$, and all network weights $w_{ij}^{(l)} \in [0,1] \subset \Re$, where $l = 1, 2$. Let $P(J)$ be the power set of the set J of the network inputs (indices), where J_{ij} is the jth input of the ith neuron. Further on for simplicity, network inputs with their indices are identified. Let $g : P(J) \to [0,1]$ be a function defined as follows:

$$g(\emptyset) = 0,$$

$$\vdots$$

$$g(\{J_{ij}\}) = w_{ij}^{(1)}, \quad 1 \leqslant j \leqslant n_0,$$

$$\vdots$$

$$g(\{J_{ij_1}, J_{ij_2}, \ldots, J_{ij_r}\}) = w_{ij_1}^{(1)} \vee w_{ij_2}^{(1)} \vee \cdots \vee w_{ij_r}^{(1)},$$
where $\{j_1, j_2, \ldots, j_r\} \subset \{1, 2, \ldots, n_0\}$,

$$\vdots$$

$$g(\{J_{i1}, J_{i2}, \ldots J_{ij}, \ldots, J_{in_0}\}) = 1,$$

6.1 Neuro-Fuzzy ystem

The hybridization of neural networks and fuzzy logic is the basic idea behind the neuro-fuzzy system. Neuro-fuzzy hybridization is done in two ways: fuzzy neural networks - is a neural network equipped with the capability of handling fuzzy information and neuro-fuzzy systems - is a fuzzy system augmented by neural networks to enhance some characteristics like flexibility and adaptability.

The fuzzy neural network was integrated with an artificial neural network in this study. Fuzzy logic can express the logical meanings commonly used by humans in a more natural and direct way. Logical decision-making was performed according to the language rules proposed by experts, which can solve nonlinear questions that cannot be addressed using rigorous modeling methods. The integration of fuzzy logic with a neural network can facilitate self-adaptation through a learning function and automatically acquire an algorithm for the information expressed as fuzzy or precise data. This method can overcome the difficulties in expressing time-varying knowledge and processes, a feature of fuzzy logic. The integration of these two components can compensate for the insufficiency of a neural network in fuzzy data processing and the deficiencies of pure fuzzy logic in learning functions. To construct a fuzzy neural network structure, this system adds a fuzzification layer before the input layer of the neural network and a defuzzification layer after the output layer, which results in the fuzzification of the input information and defuzzification of the output information, respectively. The premises and the confidence level of the conclusions in fuzzy logic, provided by experts in the field, were used as the inputs and expected outputs of the learning samples for the neural network. With the aid of the strong learning and associative memory capabilities of the artificial neural network, the neural network is trained for its learning function to automatically acquire fuzzy rules that are stored in the network in the forms of weights and thresholds. Thus, our fuzzy neural network obtains the capabilities of analyzing fuzzy questions and making diagnoses and achieves an effective combination of fuzzy logic with a neural network.

Software estimation accuracy is among the greatest challenges for software developers. Software cost estimation is the process of predicting the effort required to develop a software system. Many estimation models have been proposed. This paper provides a general overview of software cost estimation methods. As a number of these models rely on a software size estimate as input, we first provide an overview of common size metrics then we are using Fuzzy Backpropagation Network for estimating the size.

6.2 Software sizing

The software size is the most important factor that affects the software cost. The lines of code and function point are the most popular software size metrics used in practice.

Lines of Code (LOC): This is the number of lines of the *delivered source code* of the software, excluding comments and blank lines and is commonly known as *LOC*. Although LOC is programming language dependent, it is the most widely used software size metric. Most models relate this measurement to the software cost. However, exact LOC can only be obtained after the project has completed. Estimating the code size of a program before it is actually built is almost as hard as estimating the cost of the program.
A typical method for estimating the code size is to use experts' judgment together with a technique called Project Evaluation and Review Technique (*PERT)*.

It involves experts' judgment of three possible code-sizes:
l, the lowest possible size;
h the highest possible size;
and m, the most likely size.
Estimation of the code size, S = (l+h+4m)/6.

PERT can also be used for individual components to obtain an estimate of the software system by summing up the estimates of all the components. It is generally popular among the users. Here, we are suggesting the Fuzzy Backpropagation Network (instead of PERT) for producing better results.

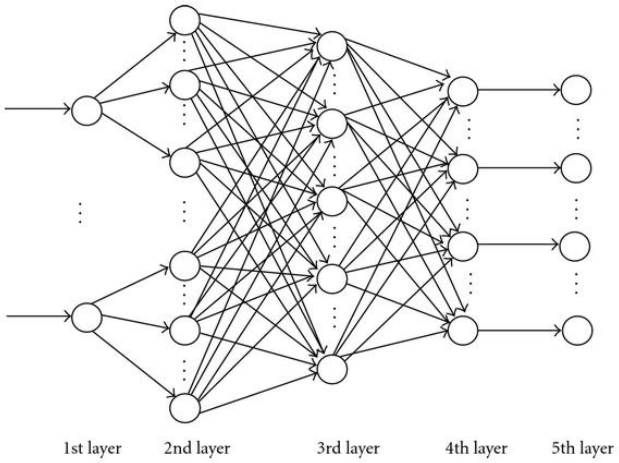
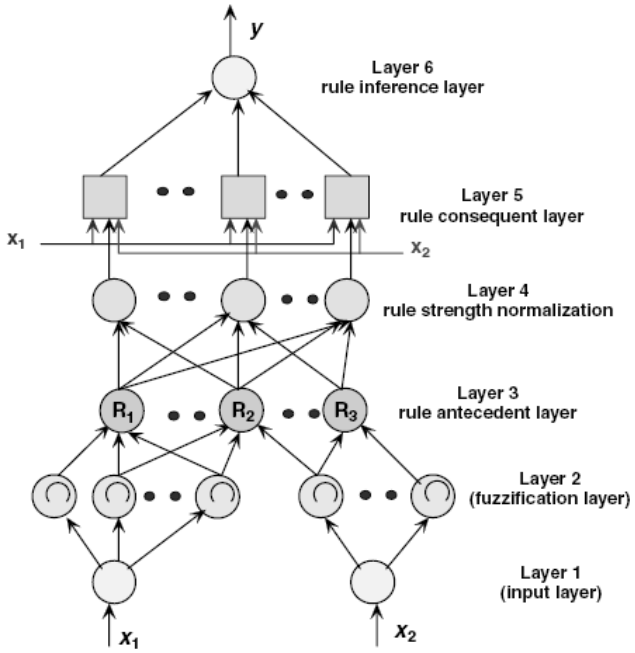


Figure: The structure of the series of fuzzy BP neural network consisting of five layers.

The entire network is composed of two parts; fuzzy processing and a conventional BP network, with the network input the same as a conventional BP network

input. Fuzzy processing was performed on the network input through the membership function in the fuzzy processing part, and the processed data were submitted directly to the BP network for further processing. The output data were compared with the expected output and reversely adjusted based on the mean square error to specify the network connection weights.

The structure of a series of fuzzy BP neural networks is shown below.



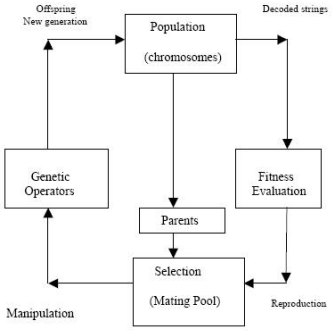
This network consists of five layers. The processing procedure of each layer is as follows.

(1) The first layer is the input layer. The nodes in this layer receive input from the outside and send it to the next layer. The connection weight constant between the first and second layer is 1.
(2) The second layer is the fuzzification layer. It performs fuzzy processing on the input and calculates the membership function value for each input component.
(3) The third layer is connected to the output of fuzzy processing through the weights. This layer is equivalent to the hidden layer of a three-layered BP network.
(4) The fourth layer is the defuzzification layer. It performs defuzzification processing on the output of the BP network.
(5) The fifth layer is the output layer.

7. Optimization of the Netw ork Structure Using the Genetic Algorithm Approach

Professor John Holland in 1975 proposed an attractive class of computational models, called Genetic Algorithms (GA), that mimic the biological evolution process for solving problems in a wide domain. A Genetic Algorithms operates through a simple cycle of stages:
i) Creation of a “;population” of strings,
ii) Evaluation of each string,
iii) Selection of best strings and
iv) Genetic manipulation to create new population of strings.

The cycle of a Genetic Algorithms is presented below



Each cycle in Genetic Algorithms produces a new generation of possible solutions for a given problem. In the first phase, an initial population, describing representatives of the potential solution, is created to initiate the search process. The elements of the population are encoded into bit-strings, called chromosomes.
The crossover points of any two chromosomes are selected randomly. The second step in the genetic manipulation process is termed mutation, where the bits at one or more randomly selected positions of the chromosomes are altered. The mutation process helps to overcome trapping at local maxima. The offsprings produced by the genetic manipulation process are the next population to be evaluated.

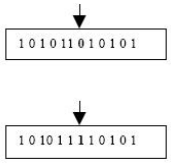


Fig.: Mutation of a chromosome at the 5th bit position.

The Genetic Algorithms cycle is illustrated in this example for maximizing a function f(x) = x2 in the interval 0 = x = 31. In this example the fitness function is f (x) itself. The larger is the functional value, the better is the fitness of the string. In this example, we start with 4 initial strings. The fitness value of the strings and the percentage fitness of the total are estimated in Table A. Since fitness of the second string is large, we select 2 copies of the second string and one each for the first and fourth string in the mating pool. The selection of the partners in the mating pool is also done randomly. Here in table B, we selected

partner of string 1 to be the 2-nd string and partner of 4-th string to be the 2nd string. The crossover points for the first-second and second-fourth strings have been selected after o-th and 2-nd bit positions respectively in table B. The second generation of the population without mutation in the first generation is presented in table C.

Table A:

Initial population and their fitness values

string no.	initial population	x	f(x)	strength fitness	(% of total)
1	01101	13	169	14.4	
2	11000	24	576	49.2	
3	01000	08	64	5.5	
4	10011	19	361	30.9	
Sum-fitness			1170	100.00	

Table B:

Mating pool strings and crossover

String No.	Mating Pool	Mates string	Swapping	New population
1	01101	2	0110[1]	01100
2	11000	1	1100[0]	11001
2	11000	4	11[000]	11011
4	10011	2	10[011]	10000

Table C:

Fitness value in second generation

Initial population	x	f(x) (fitness)	strength (% of total)
01100	12	144	8.2
11001	25	625	35.6
11011	27	729	41.5
10000	16	256	14.7
Sum-fitness =		1754	100.00

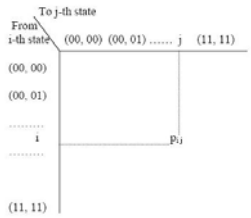
A Schema (or schemata in plural form) / hyperplane or similarity template is a genetic pattern with fixed values of 1 or 0 at some designated bit positions.

7.1 . The Markov Model for Convergence Analysis

For the sake of understanding, let us now consider the population size = 3 and the chromosomes are 2-bit patterns, as presumed earlier. The set S now takes the following form.

S = {(00, 00, 00), (00, 00, 01), (00, 00, 10), (00, 00, 11), (00, 01, 00), (00, 01, 01), (00, 01, 10), (00, 01, 11), ... (11, 11, 00), (11, 11, 01), (11, 11, 10), (11, 11, 11) }

It may be noted that the number of elements of the last set S is 64. In general, if the chromosomes have the word length of m bits and the number of chromosomes selected in each Genetic Algorithm cycle is n, then the cardinality of the set S is 2 ^ mn. The Markov transition probability matrix P for 2-bit strings of population size 2, thus, will have a dimension of (16 x 16), where the element pij of the matrix denotes the probability of transition from i-th to j-th state. A clear idea about the states and their transitions can be formed from fig.



It needs mention that since from a given i-th state, there could be a transition to any 16 j-th states, therefore the row sum of P matrix must be 1. Formally,

$$\sum_{\forall j} P_{ij} = 1,$$

for a given i.

Now, let us assume a row vector πt, whose k-th element denotes the probability of occurrence of the k-th state at a given genetic iteration (cycle) t; then πt + 1 can be evaluated by

$$\pi_{t+1} = \pi_t . P$$

Thus starting with a given initial row vector π0 , one can evaluate the state probability vector after n-th iteration πn by

$$\pi_n = \pi_0 . P^n$$

The first part of the present study focused on improving the optimization of the momentum terms and structure of the BP network, to eliminate the disadvantages of BP network algorithms such as their liability to fall into a local minimum, difficulties in determining the number of hidden layer nodes, slow convergence rate in algorithm learning, poor generalization of the network, and excessive sensitivity to initial values.

The optimization was focused on the number of hidden layers and the number of nodes in each layer of the network.

(1) Number of hidden layers: an increase in the number of hidden layers can form more complex decision-making domains, which can enhance the ability of the network to solve nonlinear problems. An appropriate number of hidden layers can also minimize the system error of the network. Based on the results of a large number of experiments, a three-layered network structure can solve most complex problems.

(2) Number of nodes in the hidden layers: the choice of neuron numbers in the hidden layers is a very complex issue, which is dependent on the experience of the designer and the results from multiple experiments. No optimal analytical expression exists. The number of hidden layer units is directly related to the requirements of the problem and the number of input/output units. If the

number is too small, the information obtained through the network will be insufficient to solve the problem. If the number is too large, it will lead to increased training time, longer learning time, a nonoptimal error rate, poor fault tolerance, failure to recognize samples that were not involved in the previous training set, and the possibility of the so-called "transitional agreement" issue. Therefore, the selection of an optimal number of hidden layer units is crucial.

The optimal boundary numbers of the hidden layer units, min and max, were first determined through the incorporated use of formulas. The network training then started from the minimum unit number min, followed by a gradual increase in unit number until the maximum unit number max was validated. For each number of hidden layer units, the network convergence speeds were compared after network convergence was achieved using the same training samples. Finally, the optimal number of hidden layer units was determined based on the training and testing errors of the training results. This approach can effectively reduce the verification time and provide the fastest way to identify the optimal number of hidden layer units.

Conclusion

Today, almost no model can estimate the cost of software with a high degree of accuracy. For the time being we suggest this approach for better understanding. Software size estimation is the key of entire software program project, and the accurate estimation immediately affects the success of project. An improved function point analysis (FPA) method was proposed for analyzing the software size. The method combined the advantage of the fuzzy rules and back propagation (BP) network. Firstly, fuzzy inference system based on the complexity weight matrix of function component was established. Then the adjusted complexity weight was used for modifying the software function point. The adjusted data as samples were transferred to BP network. By the advantage of BP network function approaching, the relationship between software components and software size was established. Finally, BP network was used for estimating software size. The experiment results show that the method could eliminate discontinuity among the different complexity grades, and could make the best of history data, which enhances the accuracy of function point estimation. Further research work with sufficient amount of data should be conducted to increase the accuracy of this method.

References

[1] Roger S Pressman, Seventh Edition, Software Engineering, A Practitioner.s Approach; McGraw Hill International Edition.

[2] Fu Limin (1994), Neural Networks in Computer Intelligence, McGraw Hill Inc.

[3] SRajasekharan, G A Vijayalakshmi Pai (2009), Neural Networks, Fuzzy Logic and Genetic Algorithms – Synthesis and Applications, PHI Learning Private Limited.

[4] B Yegnanarayana (2010), Artificial Neural Netwoks, PHI Learning Private Limited.

[5] C H Chen, Fuzzy Logic and Neural Netwokk Handbook, New York: McGraw-Hill Inc.,1996.

[6] F L Chung and T Lee, "Fuzzy competitive learning", Neural Networks, Vol 7, No 3, pp.539-552, 1994.

[7] M.T. Su, T.C.Ling, K.K.Phang, C.S.Liew, P.Y.Man, "Enhanced Software Development Effort and Cost Estimation Using Fuzzy Logic Model", Malaysian Journal of Computer Science, Vol. 20, No. 2, 2007, pp. 199-207.

[8] V Cherkassky and N Vassilas, "Performance of Backporpagation networks for associative database retrieval", in Proceedings of International Joint Conference on Neural Networks, Washington D C, Vol 1, pp. 77-84, 1989.

[9] Edwards, J.S. Moores, T.T. (1994), "A conflict between the use of estimating and planning tools in the management of information systems.". European Journal of Information Systems 3(2): 139-147.

[10] Briand, L. C. and I. Wieczorek (2002). Resource estimation in software engineering. Encyclopedia of software engineering. J. J. Marcinak. New York, John Wiley & Sons: 1160-1196.

[11] P D Wasserman, Neural Computing, Theory and Practice, New York; Van Nostrand Reinhold, 1989.

[12] S.N. Sivanandam, S. Sumathi, S.N. Deepa, "Introduction to fuzzy logic using MATLAB", Springer, 2007.

[13] P D Wasserman, Advanced Methods in Neural Computing, New York; Van Nostrand Reinhold, 1993.

[14] R A Jacobs, "Increaed rates of convergence through learning rate adaptation", Neural Networks, vol 1, No 4, pp. 295-307, 1988.

[15] Adeli H. and Wu M. (1998) "Regularization neural network for construction cost estimation", Journal of Construction Engineering and Management, ASCE 124(1), pp 18-24.

[16] J Von Neumann, The Computer and the Brain, New Haven, CT:Yale University Press, 1958.

[17] Hill Peter (ISBSG) - Estimation Workbook 2 - published by International Software Benchmarking Standards Group ISBSG - Estimation and Benchmarking Resource Centre

[18] A. J. Albrecht, "Measuring Application Development Productivity," Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.

[19] Flyvbjerg B. H., Mette S., and Buhl S. (2002), "Underestimating costs in public works projects, error or lie", Journal of the American Planning Association, Vol. 68, No.3, pp 279-292.

[20] Jorgensen, M.. "A Review of Studies on Expert Estimation of Software DevelopmentEffort". .

[21] Morris Pam - Overview of Function Point Analysis Total Metrics - Function Point Resource Centre.

[22] A.A. Moataz, O.S.Moshood, A.Jarallah, "Adaptive fuzzy-logic-based framework for software development effort prediction", Information and Software Technology, Vol. 47, Issue 1, 2005, pp. 31-48.

[23] Jørgensen, M. Shepperd, M.. "A Systematic Review of Software Development Cost Estimation Studies".
[24] Goodwin, P. (1998). Enhancing judgmental sales forecasting: The role of laboratory research. Forecasting with judgment. G. Wright and P. Goodwin. New York, John Wiley & Sons: 91-112.
[25] N K Bose and P Liang, Neural Netwrok Fundamentals with Graphs, Algorithms and Applications, McGraw-Hill, Inc., Editions, 1996.
[26] S. Mitra, Y.Hayashi, "Neuro-Fuzzy Rule Generation: Survey in Soft Computing Framework", IEEE Transactions on Neural Networks, Vol.11, No.3, 2000, pp. 748-768.
[27] D. Nauck, F. Klawonn, R. Kruse, "Foundations of Neuro-Fuzzy Systems", Wiley, Chichester, 1997.

Biography of Author(s)

B V A N S S Prabhakar Rao is obtained his Bachelor's Degree in Electronics, Master of Computer Applications & M.Tech (CST) from Andhra University and also pursuing his Ph.D through JNTUK, Kakinada. Previously he worked with Govt. Degree College, Tekkali & Dr. V S Krishna Govt. Degree College, Visakhapatnam as Lecturer in Computer Applications and Science. And with GITAM University as Assistnat Professor. Presently he working as Associate Professor with Department of Computer Science Engineering, Miracle Educational Society Group of Institutions, Vizianagaram. He is Life Member in ISTE, ISCA, IACSIT, CSTA and IAENG.

Dr. P. Seetha Ramaiah is presently working as Professor in the Department of Computer Science and Systems Engineering, College of Engineering (A), Andhra University. He received his Ph.D in Computer Science from Andhra University in 1990. He is the Principal Investigator for several Defense R&D projects and Department of Science and Technology (DST) projects of the Government of India in the areas of Embedded Systems and Robotics. He has published several Journal papers, and presented around Fifteen International Conference papers in addition to twenty one papers at National Conferences in India. His areas of research include Safety-Critical Computing- Software Safety, Computer Networks, VLSI and Embedded Systems, Real-Time Systems, Microprocessor-based System Design, Robot Hand-Eye Coordination, Signal Processing Algorithms on fixed-point DSP processors, and Bio-Electronics Systems.