# Detection of Error-Prone Software Modules Using Neural Network

[1]U.Ankaiah, [2]M.R.Narasinga Rao, [3]V.Ramakrishna
[1,2,3]Dept. of CSE, KL University, Vaddeswaram, Guntur, AP, India

**Abstract**
Software complexity metrics of a software module represent a measure of the functional complexity of the module. Identifying software modules based on their software complexity metrics into different error prone categories is a difficult problem in software engineering. This research investigates the applicability of neural network classifiers for identifying fault-prone software modules using a dataset from a software system. A prototype multi layer perceptron neural network classifier using a modified back propagation algorithm is constructed for this purpose. Our preliminary results suggest that a multi layer perceptron network can be used as a tool for identifying fault-prone software modules. Other issue such as representation of software metrics is also discussed.

**Keywords**
Artificial neural network, Software reliability engineering, Error-prone software modules, Back propagation algorithm

## I. Introduction

Artificial neural networks are a computational symbol stimulated by studies of the brain and nervous systems in biological organisms. They are highly idealized mathematical models of the essence of our present understanding of how simple nervous systems work [1]. Neural networks operate on the principle of learning from examples. Neural networks are liken to nonparametric model in the statistical literature. Recent development in neural networks has shown that they can be applied in a variety of problem domains [1]. For example, neural networks are used to solve complex nonlinear function approximation problems, difficult linearly inseparable pattern classification problems, speech recognition and control problems, and complex time-series modeling problems. Though the neural network technology has been applied in various fields including medical domain [2], its utility in software engineering has not been completely explored [1]. The purpose of this paper is to introduce how this newly emerging technology can be used in software reliability engineering applications. In particular, we demonstrate the utility of neural networks models for solving problems in the area of software reliability engineering. We illustrate how a neural network can be used as a classifier to identify fault-prone or error-prone software modules from their static complexity metrics [1]. We demonstrate the applicability of this approach using metrics data from different software's. Our results suggest that the neural network classifier may provide an frame over simple classifiers in certain categories.

## II. Methodology

### A. Data and the Detection of Error-Prone Software Modules

The metrics used in the detection of software error prone modules were obtained from different software projects data. The suitable metrics like product requirement metrics and product module metrics out of these data sets are considered [3]. For example we can considered failure software project which consists of approximately 50 modules and for each module 8 software complexity metrics were extracted from each module. These metrics include [1]:

1. Total number of lines
2. Number of executable lines
3. Total number of characters
4. Number of comments
5. Number of comment characters
6. Number of code characters
7. Total number of operators
8. Total number of unique operators

For the purpose of our classification study, these metrics represent 8 input(both real and integer) variables of classifier. We consider a software module as a low-error-prone module if there is 0 to 5 change and as a medium-error-prone module if there is 6 to 9 changes else as a high-error-prone module if there are 10 or more changes.

Here we demonstrate how neural network can be used as a pattern classifier to detect error-prone software modules early during the development cycle. There are two reasons for this demonstration:

- To show that neural network classifiers can be developed as a complement to existing statistical classifiers and [1].
- To demonstrate that classifiers can be developed without the usual assumptions about the input metrics [1].

Here, we expand and evaluate the neural network approach using a multi layer perceptron network. We also address other issues such as selection of proper training samples and representation of software metrics.

### B. Multilayer Perceptron Classifier

A perceptron with a hard-limiting threshold can be considered as a realization or a simple nonparametric linear distinguish classifier. If we use a sigmoid unit, then the continuous valued output of the perceptron can be interpreted as a likelihood or probability with which inputs are assigned to different classes. To train a perceptron network we can use back-propagation or quick-propagation procedures, or a simple optimization procedure.

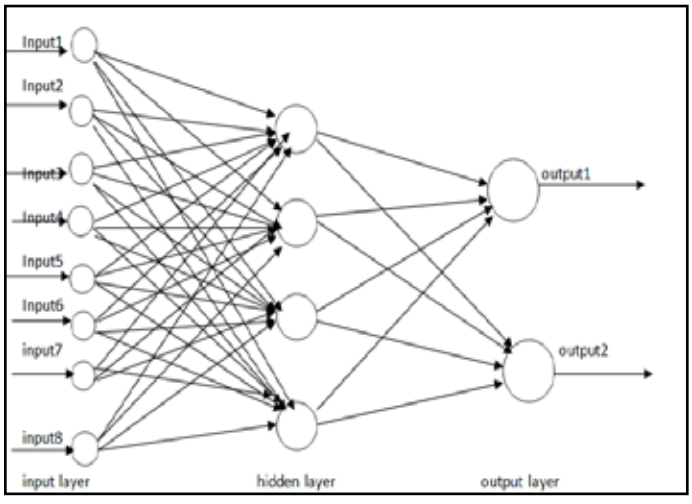



Fig. 1: Neural Network Architecture

### 1. Sigmoid Function

A sigmoid function is a mathematical function. Often, sigmoid function refers to the special case of the logistic function shown on the right and defined by the formula.

$$S(t) = \frac{1}{1 + e^{-t}}.$$

There are wide variety of sigmoid functions have been used as the activation function of artificial neural networks (neurons), including the logistic and hyperbolic tangent functions. Sigmoid curves are also common in statistics as cumulative distribution functions, such as the integrals of the logistic distribution.

### 2. Back Propagation Algorithm

Back propagation, an abbreviation for backward propagation of errors, is a common method of training artificial neural networks [5]. From a desired output, the network learns from many inputs. For better understanding, the back propagation learning algorithm can be divided into two stages: propagation and weight update

### Stage 1: Propagation

Every propagation involves the following steps:

- Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
- Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas of all output and hidden neurons.

### Stage 2: Weight update

For every weight-synapse follow the following steps:

- Multiply its output delta and input activation to get the gradient of the weight.
- Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

This ratio influences the speed and quality of learning; it is called the learning rate. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

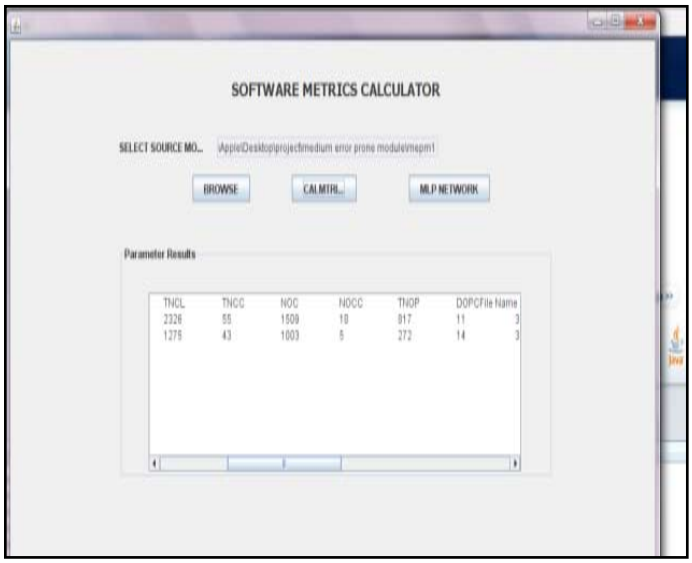
Repeat stage 1 and 2 until the performance of the network is satisfactory.

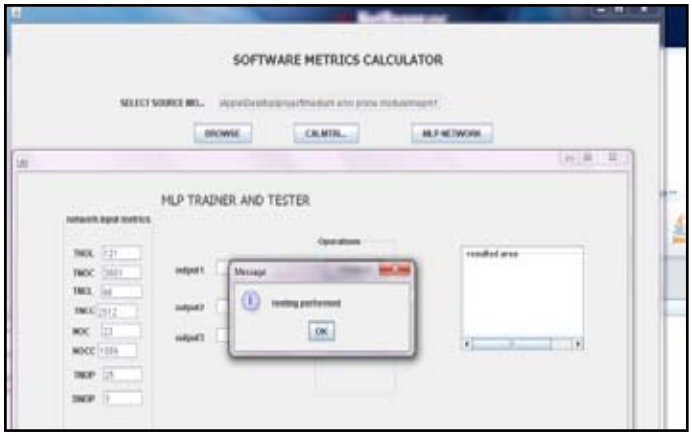
## III. Experimental Approach

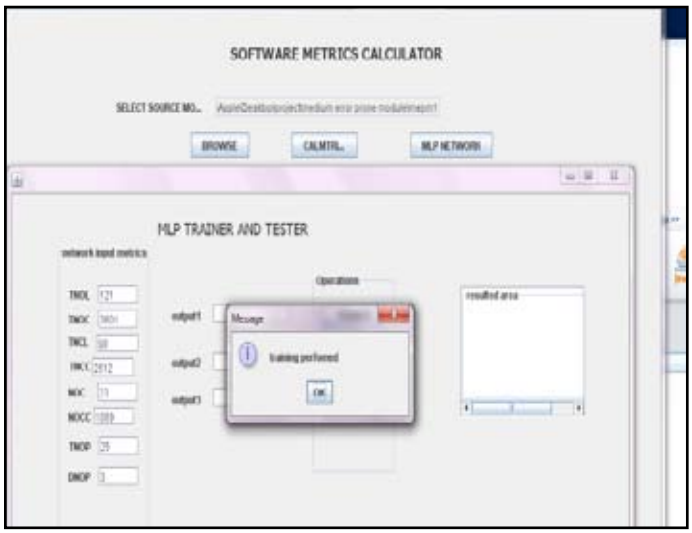
In our experiments, we used the data set with perceptron network architectures. We used eight primitive metrics as input variables and one output variable that are classify the program modules [4]. The perceptron classifier represents the simplest model of the neural network family, while the feed-forward network is a typical realization of a complex nonlinear classifier [1]. Since we are interested in assigning modules into three extreme categories, we can use three output units in our neural nets corresponding to these categories. Thus, a given arbitrary vector x is assigned to low-error-prone module if the output unit 1 is greater than remaining two output values, medium-error-prone module if the output unit 2 is greater than remaining two output values, and to high-error-prone module if the output unit3 is greater than remaining two output values[1]. We can observe the example table1, it tell that take can as 8 input values and three output values based on that we can train the network into three extreme categories. And example table2 represents the testing values of the neural network, at training stage we take the input value and desired output values based on that values we can train the network, but at testing stage we can considered only input values network will be generate output value based on that value we are classify the taken module will be low, medium or high error-prone module.

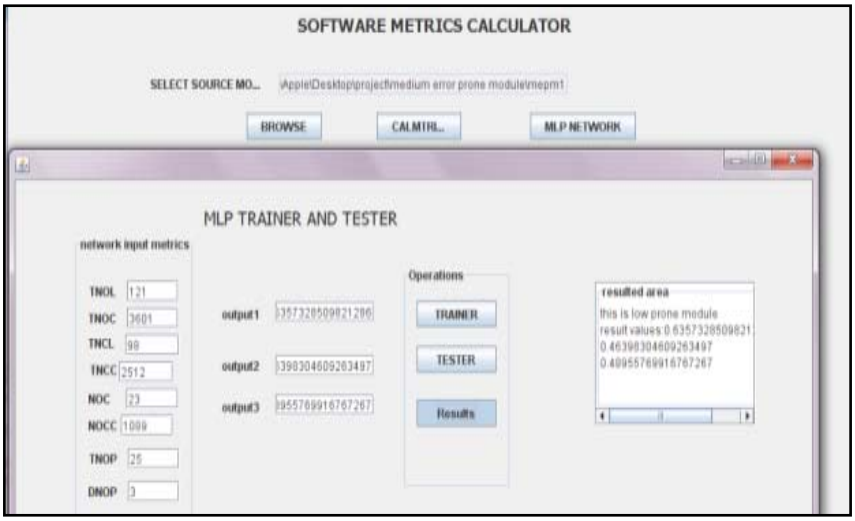
## IV. Result

### A. Output Screens






Example Tables:

Table 1: Network Training Samples

| S.no | Network input values | | | | | | | | Network training outputs | | | Type of module |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TNOL | TNOC | TNCL | TNCC | NOCL | NOCC | TNOP | DNOP | OUTPUT1 | OUTPUT2 | OUTPUT3 | |
| 1 | 282 | 8453 | 266 | 8249 | 16 | 204 | 167 | 4 | 0.4567 | 0.4756 | 0.5248 | High error-prone |
| 2 | 72 | 3614 | 64 | 3156 | 8 | 458 | 5 | 2 | 0.3457 | 0.4536 | 0.5127 | High error-prone |
| 3 | 95 | 2886 | 83 | 2880 | 12 | 6 | 34 | 4 | 0.5347 | 0.4563 | 0.4235 | Low error-prone |
| 4 | 108 | 3718 | 95 | 3667 | 13 | 51 | 56 | 4 | 0.4791 | 0.4967 | 0.5674 | High error-prone |
| 5 | 448 | 15073 | 388 | 14039 | 60 | 1034 | 231 | 4 | 0.4892 | 0.5134 | 0.5467 | High error-prone |

Table 2: Network Testing Sample

| S.no | Network input values | | | | | | | | Network testing outputs | | | Type of module |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TNOL | TNOC | TNCL | TNCC | NOCL | NOCC | TNOP | DNOP | OUTPUT1 | OUTPUT2 | OUTPUT3 | |
| 1 | 282 | 8453 | 266 | 8249 | 16 | 204 | 167 | 4 | 0.4560 | 0.5568 | 0.6256 | High error-prone |
| 2 | 72 | 3614 | 64 | 3156 | 8 | 458 | 5 | 2 | 0.5173 | 0.3559 | 0.6003 | High error-prone |
| 3 | 95 | 2886 | 83 | 2880 | 12 | 6 | 34 | 4 | 0.6218 | 0.4863 | 0.4163 | Low error-prone |
| 4 | 108 | 3718 | 95 | 3667 | 13 | 51 | 56 | 4 | 0.4800 | 0.5287 | 0.5983 | High error-prone |
| 5 | 448 | 15073 | 388 | 14039 | 60 | 1034 | 231 | 4 | 0.4939 | 0.4167 | 0.6840 | High error-prone |
| 6 | 435 | 13360 | 381 | 12295 | 54 | 1065 | 260 | 4 | 0.5720 | 0.6638 | 0.4437 | Medium error-prone |
| 7 | 509 | 15310 | 440 | 14224 | 69 | 1086 | 221 | 5 | 0.4432 | 0.6116 | 0.6287 | High error-prone |
| 8 | 68 | 2304 | 55 | 1820 | 13 | 484 | 37 | 4 | 0.6508 | 0.6089 | 0.4148 | Low error-prone |
| 9 | 89 | 3041 | 66 | 2314 | 23 | 727 | 41 | 4 | 0.6489 | 0.5392 | 0.3950 | Low error-prone |
| 10 | 43 | 1480 | 39 | 1409 | 4 | 71 | 32 | 3 | 0.4741 | 0.6511 | 0.3943 | Medium error-prone |

## V. Future Work

In future we can classify the software modules more effectively by using four more software complexity metrics values those are total number of operands, total number of unique operands, count the number of paths through the code [1] and computes the average nesting level of instructions [1].

## VI. Conclusion

In this paper we are using the neural networks for modeling to detect the error-prone software modules. The neural network offers an alternative to conventional analytic models that are obtained using experiential methods or developed using some a priori assumptions.

Though we have demonstrated the application of neural nets in the context detection of error-prone software modules related to software reliability engineering, their applicability need not be restricted to only these problems in software reliability engineering. With the availability of public domain and commercial neural network software, the neural network approach may be used as a tool in other software engineering problems such as identification of software reuse components, test-case generation, document understanding, and partitioning of test cases.

## References

[1] Michael R.Lyu,"Handbook of software reliability engineering", IEEE computer society press and McGraw Hill Book company, April 1996.

[2] Kornel Papik, Bela Molnar, Rainer Schaefer, Zalan Dombovari, Zsolt Tulassay, Janos Feher,"Application of neural networks in medicine: A review", Medical Science Monthly 1998, 4, 538-46.

[3] Rachna Ratra, Navneet Singh Randhawa, Parneet Kaur, Dr. Gurdev Singh,"Early Prediction of Fault Prone Modules using clusteringbased vs. Neural network approach in software systems", IJECT, Vol. 2, Issue 4, Oct-Dec. 2011.

[4] Nirvikar Katiyar, Raghuraj Singh,"Prediction of Software Development Faults Using Neural Network", VSRD-IJCSIT, Vol. 1 (8), 2011, pp. 556-566.

[5] XinghuoYu,"A general back propagation algorithm for feed forward neural networks learning", IEEE, Vol. 13, Issue 1, Jan 2002.

[6] K. K. Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra", Application of Artificial Neural Network for Predicting Maintainability using Object Oriented Metrics", World Academy of science, Engineering and Technology 22 2008.

[7] Nirvikar Katiyar, Dr. Raghuraj Singh,"Prediction of Number of Faults And Time To Remove Errors", International Journal Of Computational Engineering Research, Vol. 03, Issue, 4.

U.Ankaiah is pursuing M. Tech (CSE) in K L University. He received under graduate degree from JNTUA, Ananthapur, India. He has published two papers in international journal. His research interests Include Neural Networks and Software Engineering.

Dr.Manda.R.Narasinga Rao is a PhD in Computer Science and Systems Engineering from Andhra University Visakhapatnam, India. He holds a M.Tech in Computer Science from Birla Institute of Technology, Ranchi, India. He is currently working as Professor in the department of CSE at KL University. He has a rich experience in Teaching and in software industry. His research interests include Neural Networks, Information Retrieval, Finite Automata and Pattern Recognition. He is a senior member of Computer Society of India.

Dr. V.Rama Krishna is a PhD in statistics from Nagarjuna University, Guntur, India. He holds a M.Tech in Computer Science from Nagarjuna University in India. He is currently working as Associate Professor in the department of CSE at KL University. He has a rich experience in Teaching and in software industry. His research interests include Software Engineering, Neural Networks and statistics. He is a senior member of Computer Society of India.