Volume 4, Issue 3, March 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Comparing Performance of Different Neural Network Algorithms Identifying Fault Proneness in Software Modules

Jasleen Kaur*
Computer Science Engineering Department,
KCET Amritsar, India

R.S Virk
Computer Science Engineering Department,
GNDU Amritsar, India

Abstract— Software quality and reliability is as important as delivering it within scheduled budget and time. Software quality models to identify high risk program modules are used. Identifying faults early in the software lifecycle can help to predict the need for quality checking, monitoring, and amount of testing required. Fault-proneness metric is used to calculate the chances of faults in the modules of the software. In software engineering fault proneness is a famous metric measurement of faults and failure. In the study of Neural Network the real-time defect data sets are taken from the online data repository. The comparisons are made on the basis of the more accuracy and least value of MAE and RMSE error values. Accuracy value of the prediction model is the major criteria used for comparison. The mean absolute error is chosen as the standard error. The technique having lower value of the mean absolute error is chosen as the best fault prediction technique.

Keywords— Neurons, Artificial Neural Network, layers, Metrics, Variable Learning Rate, and Back Propagation.

1. INTRODUCTION

A software bug is the common term used to describe an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result. A program that contains a large number of faults that seriously interfere with its functionality, is said to be fault prone. The desired software quality and reliability to be met for a project is as important as delivering it within scheduled budget and time. To achieve desired software quality, software quality models to identify high risk program modules are used. Metrics available in the early lifecycle data can be used to verify the need for increased quality monitoring during the development. To predict the fault in software, data a variety of techniques have been proposed which includes statistical method, machine learning methods, neural network and clustering techniques.

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way a biological nervous system in human brain works. Large number of neurons present in the human brain form the key element of the neural network paradigm and act as elementary processing elements. These neurons are highly interconnected and work in union to solve complex problems. An artificial neuron is a small processing unit and performs a simple computation that is fundamental to the operation of a neural network. The model of a neuron contains the basic elements like inputs, synaptic weights and bias, summing junction and activation function.

Inputs to the input neurons are given directly. The output of these neurons is fed to the neurons of the next layer of the network by multiplying its amplitude with interconnection synaptic weight value. Summing junction termed as an adder sums up weighted input signal to any neuron. Net bias has the effect of addition or subtraction to the net input of the activation function, depending on whether net bias is positive or negative. An activation function is used for limiting the amplitude of the output of a neuron.

In a broader prospective, artificial neural networks can be divided into two major categories based on their connection topology: Feed forward and Feed backward neural networks.
Feed-forward neural networks allow the signal to flow in the forward direction only. The signals from any neuron do not flow to any other neuron in the preceding layer. In Feedback neural networks the signal from a neuron in a layer can flow to any other neuron whether it be preceded or succeeding layers. However, the drawback with these networks is that these are complex and are difficult to implement.

In ANNs, neurons are arranged into groups called layers. Input layer: It consists of a set of neurons that receive inputs form the external environment. The number of neurons in this layer is equal to the number of input variables. Hidden Layer: In any Multilayer Perceptron neural network, there can be a number of hidden layers between the input and output layers. In this thesis, Multilayer Perceptron having a single hidden layer is used for training and implementation of the ANN model of self-excited induction generator for its performance analysis. Output Layer: The output layer consists of neurons that communicate the output of the system to the user or the external environment. When the input layer receives input, its neurons produce output that becomes input to the neurons of the next layer (hidden layer). The process continues until the output layer is invoked and its neurons fire their output to the external environment. A neural network learns about its environment through a set of input-output training samples and is an interactive process of adjustment applied to its synaptic weights and bias levels. Learning in neural networks is alternatively called training and is classified into two types: Supervised learning i.e adjusts the weights of interconnection according to the difference between the desired and actual network outputs corresponding to a given input. Unsupervised learning, i.e. the network adapts the interconnection weights automatically in order to cluster the input patterns into groups with similar features.

2. OBJECTIVES

Analyze Neural Network based approaches for prediction of fault Proneness for software systems using the combined structural and requirement metrics data. To find the structural code and design attributes of software systems. To find the Requirement phase attributes of software systems.

3. SCOPE

The scope of the study is the inclusion of more modules for fault prediction, impact of attributes on fault prediction, to classify faults into categories.

4. METHODOLOGY

Fault-proneness predicts the faults and failure early in the design phase. It helps in better quality and cost management. The statistical and machine learning modelling techniques like statistical method, machine learning methods, parametric models and mixed algorithms can easily estimate software quality using fault proneness data. Therefore, there is a need to find the best prediction techniques for a given prediction problem. Complexity, density, LOC product metrics are used. The methodology consists of the following process. Firstly, to find the structural code and design attributes of software systems i.e. software metrics. The real-time defect data sets are taken from an online data repository like planet-source-code.com and promise software engineering public data sets. Then the suitable metrics like product module metrics out of these data sets are considered. The term product is used to refer to module level data. The term metrics data apply to any finite numeric values, which describe measured qualities and characteristics of a product. The term product refers to anything to which defect data and metrics data can be associated.

The product module metrics are as follows:

1. BRANCH_COUNT - Number of branches for each module.
2. CALL_PAIRS - This metric describes the number of calls to other functions in a module.
3. CYCLOMATIC_COMPLEXITY - Number of linearly independent paths. V (G) = E – N + 2.
4. CYCLOMATIC_DENSITY - Ratio of the module's cyclomatic complexity to its length.
5. LOC_BLANK - This metric describes the number of blank lines in a module.
6. LOC_CODE_AND_COMMENT - Number of lines which contain both code & comment in a module.
7. LOC_COMMENTS - This metric describes the number of lines of comments in a module.
8. DECISION_COUNT - It describes the number of decision points in a given module.
9. DECISION_DENSITY - Average cyclomatic density of the code lines within the procedures of your project, CC / LLOC, where CC is cyclomatic complexity and LLOC, logical LOC.
10. DESIGN_COMPLEXITY - Measure of a module's decision structure.
11. DESIGN_DENSITY – It describes the design density of a module and is calculated as design complexity divided by cyclomatic complexity.
12. ERROR_DENSITY - Number of defects per 1000 lines of code for a module. It is given by, ERROR_DENSITY = 1000* (ERROR_COUNT/LOC_TOTAL).
13. ERROR_COUNT - This metric describes the number of defects associated with a module.
14. PARAMETER_COUNT – It describes the number of parameters to a given module.
15. GLOBAL_DATA_COMPLEXITY - Global data complexity measures the complexity of the global and parameter data with a module.
16. GLOBAL_DATA_DENSITY – The Global Data density is calculated as: gdv (G) / v (G), i.e. dividing global data complexity by cyclomatic complexity.
17. HALSTEAD_CONTENT - Describes the Halstead length content of a module. The Halstead measures are based on four scalar numbers derived directly from a program's source code.
    n1 is the number of distinct operators, n2 is the number of distinct operands,
    N1 is the total number of operators and N2 is the total number of operands.
    Therefore, Halstead length content, n = n1 + n2.
18. HALSTEAD_DIFFICULTY (D) - D is proportional to the ratio between the total number of operands and the number of unique operands. It is calculated as, D = (n1 / 2) * (N2 / n2).
19. . HALSTEAD_EFFORT - The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program. It is given by, E= D * V, where V stands for volume.
    1. HALSTEAD_ERROR_EST - This metric describes the halstead error estimate metric of a module. It is an estimate for the number of errors in the implementation.
    2. HALSTEAD_LENGTH - Describes the halstead length metric of a module which includes the total number of operator occurrences and total number of operand occurrences.
    3. HALSTEAD_LEVEL - Level at which the program can be understood. The program level (L) is the inverse of the error proneness of the program, L = 1 / D.
20. HALSTEAD_PROG_TIME - This metric describes the halstead programming time metric of a module. It is given by, T = E / 18.
21. HALSTEAD_VOLUME - Describes the minimum number of bits required for coding the program. The program volume (V) of vocabulary size (n), V = N * log2 (n)

22. MAINTENANCE_SEVERITY - The Maintenance Severity is calculated as: ev (G)/v (G) i.e. essential complexity divided by cyclomatic complexity.
23. NODE_COUNT - This metric describes the number of nodes found in a given module. Nodes are a base metric, used to calculate many of the more involved complexity metrics.
24. NORMALIZED_CYLOMATIC_COMPLEXITY - Normalized complexity is simply a module's cyclomatic complexity divided by the number of LOC in the module.
25. NUM_OPERANDS - This metric describes the number of operands contained in a module.
26. NUM_OPERATORS - This metric describes the number of operators contained in a module.
27. NUM_UNIQUE_OPERANDS - Number of unique operands contained in a module.
28. NUM_UNIQUE_OPERATORS -Number of unique operators contained in a module.
29. NUMBER_OF_LINES - Number of lines in a module.
30. PATHOLOGICAL_COMPLEXITY - Degree to which a module contains extremely unstructured constructs.
31. PERCENT_COMMENTS - Number of fully commented LOC divided by the total number of LOC and is used as an indicator of readability.

LOC_TOTAL - Total number of lines for a given module.

4.1 To Explore different Neural Network Techniques

It is very important to find the suitable algorithm for modeling of software components into different levels of fault severity in software systems. Three Neural Network algorithms are studied: Variable Learning Rate, Variable Learning Rate training with momentum, Resilient Back propagation.

4.1.1 Variable Learning Rate:

With the standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm can oscillate and become unstable. If the learning rate is too small, the algorithm takes too long to converge. It is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface.

Variable Learning Rate without momentum is a network training function that updates weight and bias values according to gradient descent with adaptive learning rate. Here, Back propagation is used to calculate derivatives of performance DPERF with respect to the weight and bias variables X. Each variable is adjusted according to gradient descent:

$$\partial X = l_r \times \frac{\partial PERF}{\partial X} \quad \ldots\ldots.(1)$$

Each of the epoch, if performance decreases toward the goal, then the learning rate is increased by the factor lr_inc. If performance increases by more than the factor max_perf_inc, the learning rate is adjusted by the factor lr_dec and the change, which increased the performance, is not made. Training stops when any of these conditions occurs: 1.The maximum number of Epochs (repetitions) is reached, 2. The maximum amount of Time to train has been exceeded, 3. Performance has been minimized to the Performance Goal, 4. The performance gradient falls below the Minimum Performance Gradient value, 5. Validation performance has increased more than maximum number of validation Failures values.

Gradient descent w/momentum & adaptive lr back propagation or Variable Learning Rate Training with momentum is a network training function that updates weight and bias values according to gradient descent momentum and an adaptive learning rate. In this algorithm Back propagation is used to calculate derivatives of performance PERF with respect to the weight and bias variables X. Each variable is adjusted according to the gradient descent with momentum:

$$\partial X = m_c \times \partial X_{PERF} + l_r \times m_c \times \frac{\partial PERF}{\partial X} \quad \ldots\ldots(2)$$

Where ∂X_{PERF} he previous change to the weight or bias, l_r is the learning rate and m_c is the momentum constant. For each epoch, if performance decreases toward the goal, then the learning rate is increased by the factor lr_inc. If performance increases by more than the factor max_perf_inc, the learning rate is adjusted by the factor lr_dec and the change, which increased the performance, is not made. Training stops when any of the conditions mentioned above occurs.

4.1.2 Resilient Back propagation:

Multilayer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called "squashing" functions, because they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slopes must approach zero as the input gets large. This causes a problem when you use steepest descent to train a multilayer network with sigmoid functions, because the gradient can have a very small magnitude and, therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

The purpose of the resilient back propagation training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by some factor whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by some factor whenever the derivative with respect to that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating, the weight change is reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change increases.

Resilient Back propagation can train any network as long as its weight, net input, and transfer functions have derivative functions. In this algorithm Back propagation is used to calculate derivatives of performance PERF with respect to the weight and bias variables X. Each variable is adjusted according to the following

$$\text{eq}\ \partial X = \Delta X \times sign(g_X) \quad \ldots\ldots(3)$$

Where the elements of *ΔX* are all initialized to 0 and g_X is the gradient. At each iteration the elements of ΔX are modified. If an element of g_X changes sign from one iteration to the next, then the corresponding element of deltaX is decreased by delta_dec. If an element of gX maintains the same sign from one iteration to the next, then the corresponding element of delta X is increased by delta_inc [26]. Thereafter the trained network is tested by testing data in the testing phase.

The results of the different algorithms are expressed in terms of MAE, RMSE and Accuracy values. The details of the different criteria used are in the next step. The following steps will be followed to train a Neural Network: 1. Load the data, 2. Divide the data into Training, Validation and Test data, 3. Set number of hidden neurons, 4. Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs, 5. If there is a difference between the actual and target outputs, the weights are adjusted to produce a set of outputs closer to the target values, 6. Network weights are determined by adding an error correction value to the old weight, 7. The amount of correction is determined ,8. This Training procedure is repeated until the network performance no longer improves, 9. If the network is successfully trained, it can then be given new sets of input and generally produce correct results on its own.

4.2 Comparison of Algorithms

The comparisons are made on the basis of the more accuracy and least value of MAE and RMSE error values. Accuracy value of the prediction model is the major criteria used for comparison. The mean absolute error is chosen as the standard error. The technique having lower value of the mean absolute error is chosen as the best fault prediction technique.

- **Mean absolute error**

Mean absolute error, MAE is the average of the difference between the predicted and actual value in all test cases; it is the average prediction error [13]. The formula for calculating MAE is given in equation 4.

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad \ldots\ldots(4)$$

- **Root mean-squared error**

RMSE is frequently used measure of the differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated. [13]. It is just the square root of the mean square error as shown in equation 5.

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}} \quad \ldots\ldots(5)$$

5. ANALYSIS

The real-time defect data set to be discussed here contains 178 modules of C Programming language with different values of software fault severity labeled present as 1, 2 and 3. Assume the level 1 represents the highest severity, level 2 represents the medium and level 3 represents the minor fault which can be ignored. Details of the type of faults existing in different number of modules of the Dataset are shown the in the bar chart of figure 1and numeric values are tabulated in table 1.

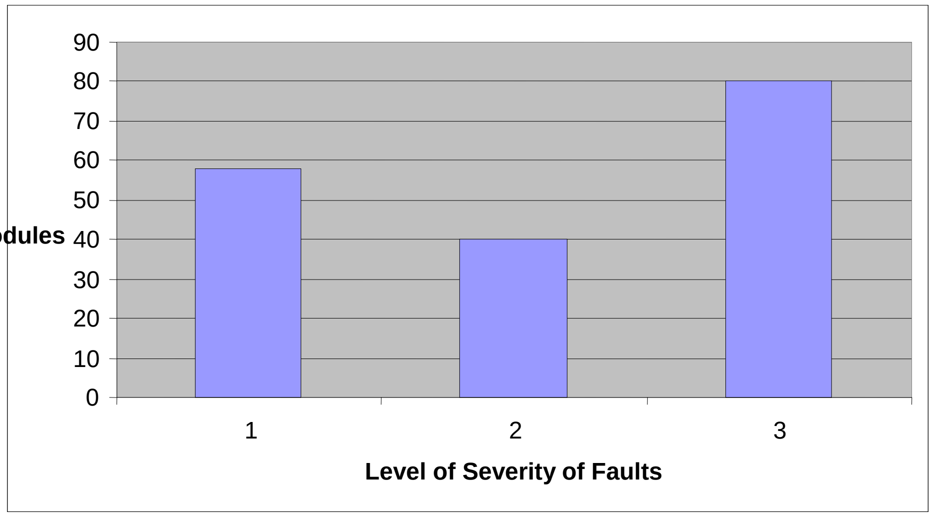

Fig. 1 Graphical Representation of different Severity of Faults in Modules

Table 1: Details of the different Severity of Faults in Modules

Severity Level	Number of Modules
1	58
2	40
3	80

The algorithms are evaluated on the basis of the following criteria:
The mean absolute error, root mean squared error, relative absolute error and root relative squared error are to be used for comparison. However, the most commonly reported errors are the mean absolute error and root mean squared error. Studies have shown that the root mean squared error is more sensitive to outliers in the data than the mean absolute error. In order to minimize the effect of outliers, mean absolute error is chosen as the standard error. The prediction technique having least value of mean absolute error is chosen as the best prediction technique. Mean absolute error, MAE is the average of the difference the between predicted and actual value in all test cases. The root mean-squared error, i.e. RMSE is simply the square root of the mean-squared-error. The root mean-squared error gives the error value as the same dimensionality as the actual and predicted values. The mean-squared error is one of the most commonly used measures of success for numeric prediction. This value is computed by taking the average of the squared differences between each computed value and its corresponding correct value. The MAE and the RMSE can be used together to diagnose the variation in the errors in a set of forecasts. The RMSE will always be larger or equal to the MAE.

The greater difference between them, the greater the variance in the individual errors in the sample. If root mean squared error is equal to mean absolute error, then all the errors are of the same magnitude. Both root mean squared error and mean absolute error can range from 0 to ∞.

MAE and RMSE are negatively-oriented scores and lower values are better. So, an algorithm with least value of the mean absolute error is considered as the best algorithm. The training phase performance of the Variable Learning Rate without momentum, Variable Learning Rate with momentum and Resilient Back propagation is shown in figure.

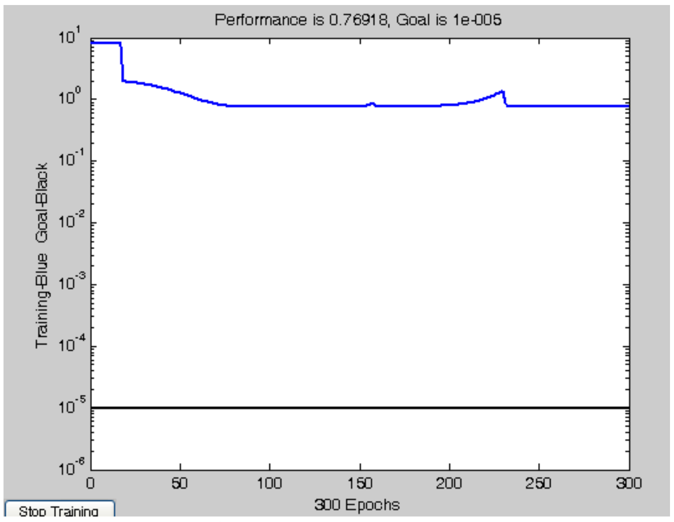

Fig. 2 Training Performance of Variable Learning Rate without momentum algorithm

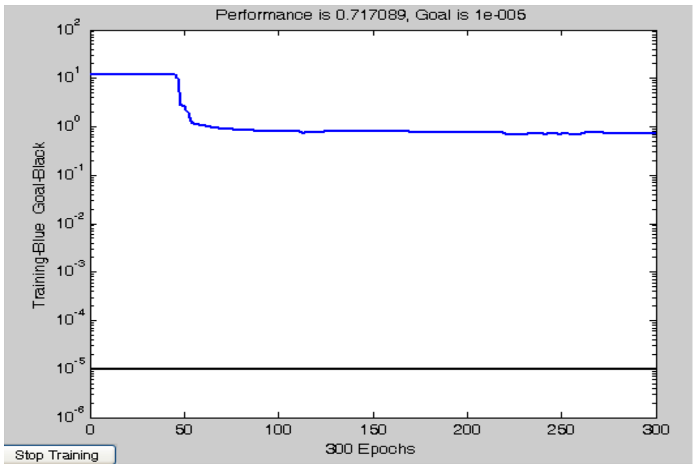

Fig. 3 Training Performance of Variable Learning Rate with momentum algorithm

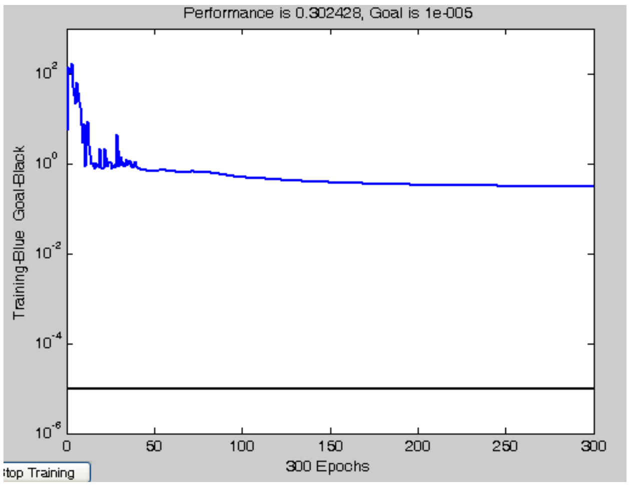

Fig. 4 Training Performance of Resilient Back propagation In this analysis the three Neural Network based algorithms experimented in Matlab are discussed.

Table 2: Performance Results of Different Neural Network Algorithms

Sr. No.	**Algorithm**	***MAE***	***RMSE***	***Accuracy%***
1	Variable Learning Rate without momentum	0.6146	0.7084	40
2	Variable Learning Rate training with momentum	0.6820	0.8564	53.3333
3	Resilient Back propagation	0.3980	0.5385	80

The overall testing performance of the different algorithms is shown in table 2. The results reveal that the Resilient Back propagation algorithm have outperformed all other algorithm under study with 0.3980, 0.5385 and 80% as MAE, RMSE and Accuracy values respectively.
In case of the resilient back propagation network the actual severity values and the estimated severity values of the tested modules is illustrated in table 3.

Table 3: Actual and Severity Value estimated by Resilient Back propagation Neural Network

Module Sr. No.	**Actual Severity Value**	**Estimated Severity Value**
1	3	3
2	2	2
3	3	3
4	2	2
5	2	2
6	2	2
7	1	1
8	3	3
9	2	3
10	3	4
11	3	3
12	3	3
13	3	4
14	2	2
15	1	1

The analysis returns three values: 0.552, 0.975 and 0.674 as Slope of the linear regression, Y intercept of the linear regression and Regression R-value respectively. So, correlation coefficient (R-value) between the outputs and targets is 0.674. It is a measure of how well the variation in the output is explained by the targets. If this number is equal to 1, then there is a perfect correlation between targets and outputs. In results, the number is towards close to 1, which indicates a fair fit. The figure 5 illustrates the graphical output provided by analysis. The network outputs are plotted versus the targets as open circles. The best linear fit is indicated by a dashed line. The perfect fit (output equal to targets) is indicated by the solid line.

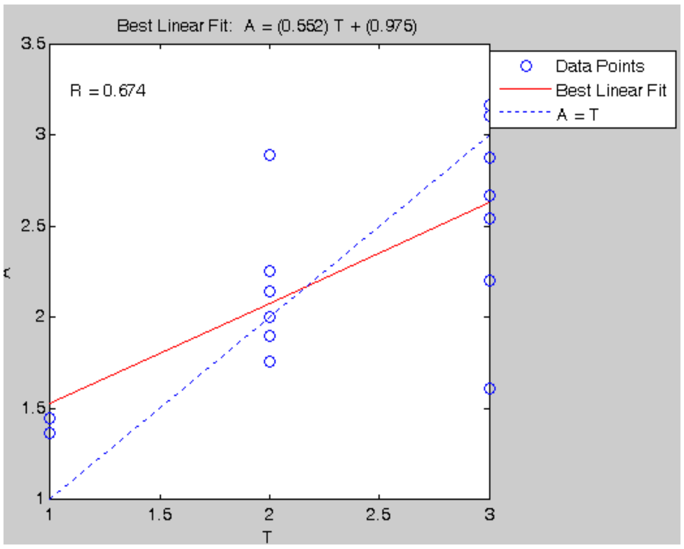

Fig. 5 Resilient Back propagation fitting

6. CONCLUSION

Fault prediction is used to improve software process control and achieve high software reliability. It is concluded the model is implemented and the best algorithm for modeling of the function based software modules into different level of severity of impact of the fault is found to be resilient back propagation based Neural Network technique. Hence, the proposed algorithm can be used to identify modules that have major faults and require immediate attention. MAE and RMSE values are calculated by applying different algorithms on two types of datasets Results of different Neural Network Based algorithms for Prediction of Fault Proneness in the function based software modules.

REFERENCES

[1] Bellini, P. (2005), “Comparing Fault-Proneness Estimation Models”, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), vol. 0, 2005, pp. 205-214.
[2] Caligula, Bastani B. and Yen (2006). “A Unified framework for Defect Data Analysis using the MBR Technique”. Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), Washington, pp. 39-46.
[3] Ceylan E., Kutlubay F.O. and Bener A.B. (2006), “Software Defect Identification Using Machine Learning Techniques” Proceeding of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), Turkey, pp. 240-247.
[4] Deodhar Manasi (2002), “Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems”, MS Thesis, Michigan Tech. University, Dec. 2002.
[5] Eman, K., S. Benlarbi, N. Goel and S. Rai, (2001), “Comparing case-based reasoning classifiers for predicting high risk software components”, Journal of Systems Software, 55(3): 301 – 310.
[6] Fenton N.E. and Pfleeger S.L. (1997), “Software Metrics: A Rigorous and Practical Approach”. PWS publishing Company: ITP, Boston, MA, 2nd edition, pp.132-145.
[7] Fenton, N. E. and Neil, M. (1999), “A Critique of Software Defect Prediction Models”, Bellini, I. Bruno, P. Nesi, D. Rogai, University of Florence, IEEE Trans. Softw. Engineering, vol. 25, Issue no. 5, pp. 675-689.
[8] Giovanni Denaro (2000), ”Estimating Software Fault-Proneness for Tuning Testing Activities” Proceedings of the 22nd International Conference on Software Engineering (ICSE2000), Limerick, Ireland, June 2000.
[9] Hudepohl, J. P., S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. E. Mayrand, (1996), “Software Metrics and Models on the Desktop”, IEEE Software, 13(5): 56-60.
[10] Khoshgoftaar, T. M., E. B. Allen, K. S. Kalaichelvan, and N. Goel, (1996), “Early quality prediction: a case study in telecommunications”, IEEE Software (1996), 13(1): 65-71.
[11] Khoshgoftaar, T.M., K. Gao and R. M. Szabo (2001), “An Application of Zero-Inflated Poisson Regression for Software Fault Prediction. Software Reliability Engineering”, ISSRE 2001. Proceedings of 12th International Symposium on, 27-30 Nov. (2001), pp: 66 -73.
[12] Khoshgoftaar, T. M. and N. Seliya, (2002), “Tree-based software quality estimation models for fault prediction”, METRICS 2002, the Eighth IIIE Symposium on Software Metrics, pp: 203-214.

[13] Lanubile F., Lonigro A., and Visaggio G. (1995) "Comparing Models for Identifying Fault- Prone Software Components", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, June 1995, pp. 12-19.

[14] Munson, J. and T. Khoshgoftaar, (1990) "Regression Modeling of Software Quality: An Empirical Investigation", Information and Software Technology, 32(2): 106 - 114.

[15] Munson, J. C. and T. M. Khoshgoftaar, (1992), "The detection of fault-prone programs", IEEE Transactions on Software Engineering, 18(5): 423-433.

[16] Menzies, T., K. Ammar, A. Nikora, and S. Stefano, (2003), "How Simple is Software Defect Prediction?" Journal of Empirical Software Engineering, October (2003).

[17] Ma Y. and Guo L. (2006), "A Statistical Framework for the Prediction of Faults Proneness", product Focused Process Improvement, Edition: First, Publisher: Springer Berlin/Heidelberg, pp. 204-214.

[18] Pigoski M. and Nelson E. (1994), "Software Maintenance Metrics: A Case Study", Proceedings of IEEE Conference on Software Maintenance, Canada, pp. 392-401.

[19] Reidmiller, Proceedings of the IEEE Int. Conf. on NN (ICNN) San Francisco, 1993, pp. 586- 591

[20] Runeson, Wohlin C. and Ohlsson M.C. (2001), "A Proposal for Comparison of Models for Identification of Fault-Proneness", Journal of System and Software, Volume: 56, Issue: 3, pp. 301–320.

[21] Stark E. (1996), "Measurements for Managing Software Maintenance", International Conference on Software Maintenance, USA, pp. 152-161.

[22] Seliya N., T. M. Khoshgoftaar, S. Zhong, (2005), "Analyzing software quality with limited fault-proneness defect data", Ninth IEEE international Symposium, Oct 12-14, (2005).

[23] Yue Jiang, Bojan Cukic, Tim Menzies, Nick Bartlow (2008),"Comparing Design and Code Metrics for Software Quality Prediction" The Lane Department of Computer Science and Electrical Engineering West Virginia University.

[24] A Abraham (2005), "*Artificial neural networks* (*ANN*) John Wiley & Sons, Ltd. ISBN: 0-470-02143-8

[25] Parvinder Singh, Sunil Kumar (2007)" Intelligence System for Software Maintenance Severity Prediction" Journal of Computer Science 3 (5): 281-288, 2007 ISSN 1549-3636, Science Publications

[26] Sonia Manhas, Rajeev Vashisht, Parvinder S. Sandhu and Nirvair Neeru,"Reusability Evaluation Model for Procedure Based Software Systems" International Journal of Computer and Electrical Engineering, Vol.2, No.6, December, 2010 1793-8163

[27] Constantinescu "Analyzing the effect of permanent, intermittent and transient faults on a gracefully degrading microcomputer Microelectronics Reliability", Volume 32, Issue 6, Pages 861-866